Inferring Polygon Structures from Point Clouds

Iris Seaman, Cat McQueen
Computer Science Department
Brigham Young University
Provo, UT 84602 USA
irisrseaman@gmail.com, catherineamcqueen@gmail.com

Abstract

One of the main problems with autonomous agents inferring their environment, is that they commonly rely on partially observable and noisy data. If we assume an autonomous agent holds a full map consisting of noisy observed data points, could it model the map as a polygon representation instead? This paper discusses different approaches using Bayesian statistics to infer polygon structures, specifically lines in polygon structures, from 2-D points on a point cloud map. This polygon representation of a map would be highly useful and necessary to do complex computations when running search algorithms on the environment. Assuming the map is described as a 2 dimensional point cloud, we discuss using deep learning to determine the number of polygon structures. Then, we use that number to run a clustering algorithm to infer which points belong to each polygon. From there, we experiment with four different algorithms to determine linear relationships that would describe line segments in a polygon. Specifically, we show how using a Expectation Maximization can be used to infer lines in a polygon using multivariate Gaussian parameters of means and covariances, as well as tube-like Gaussian parameters of slopes and y-intercepts for each line cluster.

Introduction

Autonomous decision making is a current and popular research topic in the field of machine learning. Specifically when it comes to making decisions based on an environment. One of the main problems when making autonomous decisions, such as path planning, or search, is having a partially observable environment. In many cases, agents use built-in sensors that retrieve information about their soundings. The agent must then use the observed noisy data to infer a model of the environment. Algorithms such as RANSAC (RANdom Sample Consensus), a linear regression model, is a common algorithm used to infer walls in an environment, especially as it travels down hallways or streets (Se, Lowe, and Little 2005). Another algorithm used to infer lines from observations is the Hough-Transform. However, the question we attempt to answer in this paper is, given all of the partially observed data, such as a 2-dimensional point cloud, can an autonomous agent infer the structures in the point cloud as polygon representations in the map?

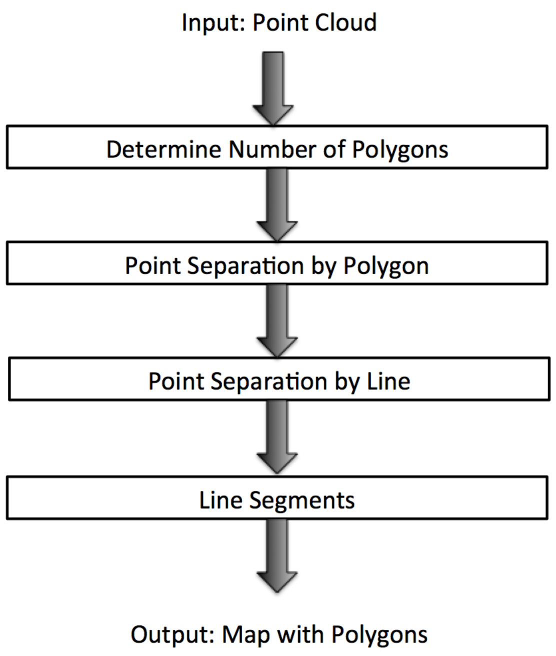

Input: Point Cloud
Determine Number of Polygons
Point Separation by Polygon
Point Separation by Line
Line Segments
Output: Map with Polygons

Figure 1: An outline of the goals and steps of our project.

RANSAC In many problems where agents are required to infer the environment using observations gathered through sensors or other methods, RANSAC is commonly used to detect walls the agent may be moving along side of.

RANSAC is an improved linear regression model that favors inliers as opposed to outliers (Derpanis 2010). In general the slope of a RANSAC regression would be a more accurate representation of the daa than traditional linear regression. Figure 2 shows an example of a sample 2-D point cloud and a RANSAC regression. As seen, RANSAC chooses the best linear regression based on the observed data. Alone, RANSAC could not determine all the co-linear points on the map. If multiple RANSAC algorithms were run on the same data, the same RANSAC regression would appear multiple times as a regression to the total point cloud, not revealing a distributed number of lines on the data points.

Hough-Transform One of the experiments we ran was using Hough-Transform to determine if lines could be inferred from points. In Figure 3, we show the outputs for a set of points that form a pentagon. a) shows the edges detected in the image of points. From the edges, the Hough-Transform algorithm can begin to try finding the best fitted lines for the edges it detected (Duda and Hart 1972). Unfortunately, the sparsity of the data does not allow the edge detection algorithm to form coherent lines. Instead, edges are formed around each individual point as opposed to forming along co-linear points. This makes it difficult for the Hough-Transform algorithms to work well with simple parameters. b) shows the Hough-Transform using a more efficient algorithm called the Probabilistic Hough-Transform. We show that lines formed, but again lines were formed on the edges for each point. This produces small short lines, which basically did not do anything different to the data. c) shows the traditional Hough-Transform algorithm's results. Here, lines are formed in almost every point as well. However, one can argue that further line detection can be done if similar slopes were averaged. The parameter of what slopes are considered 'similar' would have to be adjusted.

Gibbs Sampling / Statistical Generative Models Statistical generative models are a natural way to describe difficult models such as inferring polygons from point clouds (Gilks, Richardson, and Spiegelhalter 1995). Unfortunately, one must be careful how their generative model is designed so inference may be done to answer the appropriate questions. To infer polygon representations from data points, ideally we would create a generative model that shows the dependencies of how points fit into lines, lines fit to polygons, which points belong to which polygon, and which lines belong to which polygon. This model would be complex, and out of the scope of the project at this time. However, in our paper, we tried to create a simple generative model inspired by the LDA model. We naively tested a very similar model on our data, and later updated our model to consider weights for a line using RANSAC. This is described later in the paper.

Expectation Maximization for Clustering Points into Polygons Assuming we have a 2-D point cloud map of N number of polygons, we would like to figure out which points most likely belong to a particular polygon. Expectation Maximization (EM) is one approach to clustering data points (Moon 1996). The issue with EM on clustering data is knowing how many clusters are needed to best describe the groupings in the data. Assuming that this number of clusters is given to us by the neural network described later in the paper, we can run EM to infer which points most likely belong to particular polygons. This information is useful if we desire to focus on one cluster, or polygon at a time to infer further information about the polygon of points.

Expectation Maximization for Clustering Points into Lines Since EM proves to be useful to describe points belonging to particular clusters, or polygons on a sample of a 2-D point cloud map, we experimented using a couple variations of EM to describe lines using points. In the first approach, given the number of lines in a polygon, we ran EM to infer cluster parameters of means and covariances of multivariate Gaussians. To compare against this EM model, we created another variation of EM attempting to infer the parameters describing a line: a slope and y-intercept. In our experiments we demonstrate the weaknesses and strengths of both approaches and go into detail in the implementation and experiments later in the paper.

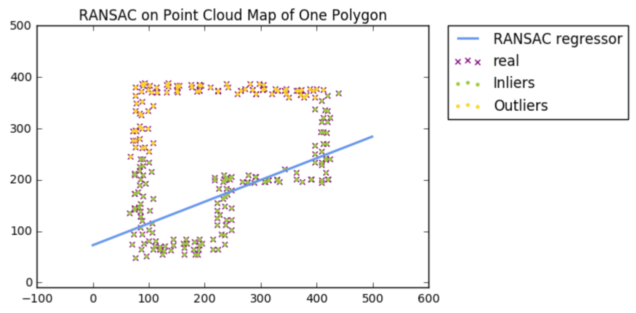


Figure 2: An example of the RANSAC regression on a sample 2-D point cloud map.

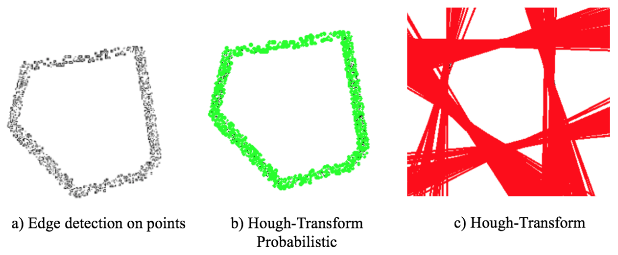


Figure 3: Describe the hough example..and why its doesn't work for our problem

Neural Networks to count clusters on 2-D Point cloud As discussed earlier, one of the main problems with any clustering algorithm is knowing how many clusters are appropriate to run on a given set of data. Since our data is visually coherent, a person could simply count the number of polygons that could be described on a 2-D point cloud map. However, we desire an agent to do this on its own. To attempt providing a solution, we turned to Deep Neural Networks (DNNs). We created a DNN that attempted to classify a 2-D point cloud map by its number of point clustering, or polygons. For experiments we trained the classifier to count from 1 to 5.

In this paper we discuss the different steps required to infer polygon structures from simple 2-D point cloud maps. We

- discuss the process in which we determine how many clusters we must use in our sample map,
- use that number of clusters to run a clustering EM on the data to infer which points belong to which polygon,
- use that information to isolate points belonging to a sim-

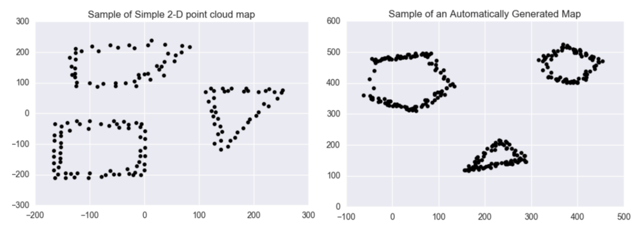


Figure 4: (left)A sample of a manually created simple 2-D point cloud map of points describing polygon structures using the python library Pygame to draw points on a window (right) A sample of an automatically generated map using parameters to describe the dimensions of the map, the number of polygons, and the noise to add to the sample points.

ple polygon and run different statistical machine learning algorithms to infer lines from points.

We tested four different statistical machine learning algorithms to infer lines from the point cloud given. The four different approaches we implemented are:

1. full Gibbs sampling on a simple LDA-inspired generative model
2. full Gibbs sampling on a slightly altered generative model using RANSAC for weights in a Dirichlet distribution
3. clustering EM using multivariate Gaussian parameters means and covariances to describe clusters
4. clustering EM using Gaussian to describe line parameters such as slope and y-intercept

A diagram of the full pipeline of what we tried to accomplish in this project is shown in Figure 1.

Data

Our problem focuses on one type of data, a simple 2-D point cloud map. A point cloud consists of multiple points in some space. In a 2-dimensional space, points are described with x and y coordinates. Point clouds may take on any shape or form. Many of the current point-clouds available are in 3-D space and consist of millions of points, such as the 3-D point cloud of the city of Bremen, Germany (Bre). For our experiments, we work with simplified point-clouds. Figure 4 shows two examples of maps we generated. These sample maps consist of (x,y) data points, and are meant to describe 3 clusters of polygons that represent building structures on a 500x500 dimensional map. These maps were generated using unique methods we will describe later.

Generation of Maps

For experiments and evaluations, we created two python scripts for different approaches to create data.

The first script allowed us to manually generate a map. We used a python drawing library, Pygame, to open a window and manually create points by clicking in it. Once we finished clicking/generating new points on a map, we closed

Algorithm 1 Expectation Maximization (EM)

```
given:
covs: list of random covariance matrixes
mus: list of random means
mws: random mixing weights for each cluster
for $n = 1$ to $N$ do
    $\text{cluster}_{\text{weights}} = \text{Expectation}(\text{covs}, \text{mws})$
    $\text{mws}, \text{mus}, \text{covs} = \text{Maximization}(\text{cluster}_{\text{weights}})$
end for
```

the window and pickled the data into a .dat format. Figure 4 (left) is an example of a manually created map.

The second script allowed us to automatically generate sample maps. The parameters required to generate maps are: the dimensions of the map, number of polygons, and the range of vertices to randomly generate for each polygon. The script begins by generating regular polygons on the map using a list of line segments to describe a polygon. The program adds a buffer of distance between polygons as it places them on the map to ensure no overlaps of polygons occur. The buffer can also be used to describe how close or far the polygon structures may placed. Next, we sample points from each of the line segments in the polygon and add noise to each point. Figure 4 (right) is an example of an automatically created map using 500x500 as its map dimension, 3 as its number of polygons, 3-6 as its range of vertices for each polygon.

Approach

Our goal is to have an autonomous agent infer a polygon representation of a map from a point cloud. We begin with a simple approach to solving this problem. In this section, we discuss the different steps with their respective approaches needed to provide a solution.

Step 1: Using Deep Learning Classify the Number of Polygons on a Map

The first question many ask in a clustering problem is: how many clusters are appropriate to represent the grouping of data most correctly? Dealing a point cloud map, we first need to determine which points are most likely belong to a simple cluster representation of a coarse polygon. In order for an agent to determine a polygon (list of line segments) based representation of its environment, it must autonomously decide how many clusters of polygons are most likely on the map. There are several techniques used to tackle this problem; however, we turned to deep learning to see if a Deep Neural Network (DNN) could determine the number of clusters on a 2-D point cloud.

Training and Test Data As discussed in the previous section, we created a generative model that sampled 2-D point cloud maps based on the number of polygons given, and some additional parameters. We used the automatic map generator to create training data for the neural network. We created 5 classifications: 1 to represent 1 polygon cluster, 2 for 2 polygons, 3 for 3 polygons, etc. Each classification

Algorithm 2 Expectation

```
given:
data: points from sample map
covs: list of covariance matrixes
mus: list of means
mws: mixing weights for each cluster
clusters = []
for $n = 1$ to $N$ do
    $\mathrm{cluster_n} = \mathrm{multivariateNormal(data, mus[n], covs[n]}$
    $\mathrm{cluster_n} = \mathrm{cluster_n} * \mathrm{mws[n]}$
    $\mathrm{clusters.append(cluster_n)}$
end for
$\mathrm{clusters} = \mathrm{Normalize(clusters)}$
return clusters
```

Algorithm 3 Maximization

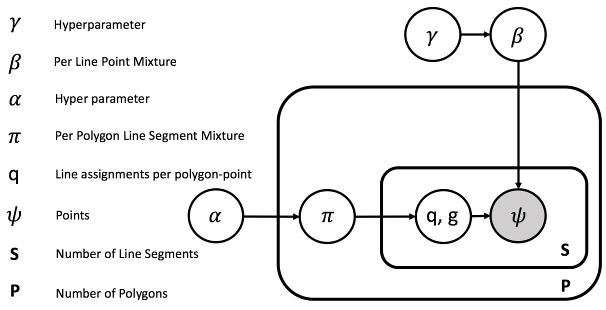
```
given:
$\mathrm{cluster_w}$ : list of weights for each point for each cluster
$\mathrm{mws_{new}} = []$
$\mathrm{mus_{new}} = []$
$\mathrm{covs_{new}} = []$
for $n = 1$ to $N$ do
    $\mathrm{mws_{new}.append(Normalize(cluster_w[n])}$
    $\mathrm{diff} = \mathrm{data} - \mathrm{cluster_w[n]}$
    $\mathrm{covs_{new}.append(Normalize(diff \cdot diff))}$
    $\mathrm{mus_{new}.append(Normalize(cluster_w[n] \cdot data))}$

end for
return $\mathrm{mws_{new}, mus_{new}, covs_{new}}$
```

held a total of 5,000 randomly sampled point cloud maps. A total of 25,000 maps were generated and we randomly sampled 20,000 from the data to train the DNN, and used the remaining 5,000 for testing. Before training, the data was preprocessed into gray-scale 25x25 images (originally 1000x1000). This was done for computational purposes, since we trained and tested the DNN on our local machines (with only CPUs).

DNN Model The DNN was created as a Convolutional Neural Network (CNN). During construction and experimentation with the DNN, we trained the CNN with several different learning rates. This was done to determine which rate was most appropriate to avoid over-fitting (a network that memorizes the training data well, but performs poorly on unseen testing data). We discuss results of different learning rates in the experiments section.

The CNN was designed as simple classification net and was constructed on TensorFlow as follows:

1. Convolutional layer with 32 filters with a Leaky Rectified Linear Unit (Leaky ReLU) as an activation function.
2. Max pooling layer that down samples by 2X.
3. Convolutional layer with 64 filters followed again with a Leaky ReLU activation
4. Max pooling again
5. Fully connected layer followed by a Leaky ReLU



γ Hyperparameter
β Per Line Point Mixture
α Hyper parameter
π Per Polygon Line Segment Mixture
q Line assignments per polygon-point
ψ Points
S Number of Line Segments
P Number of Polygons
γ
β
α
π
q, g
ψ
S
P

Figure 5: Simple Generative model “highly” inspired by Latent Dirichlet Allocation model

6. 0.5 Drop-Out as regularization
7. And end with fully connected layer.

The model used TensorFlow’s softmax cross-entropy with logits as a loss function and TensorFlow’s Adam Optimizer for gradient descent optimization.

Step 2: Inferring Polygon Clusters from Points

Once the number of clusters, or polygons were determined, we could then run a clustering algorithm to determine which points best cluster into a polygon.

Expectation Maximization Expectation Maximization (EM) is an algorithm that works well when clustering data due to its properties of balancing itself out while reassigning cluster weights to each data point based on parameters and reassigning parameters based on weights.

Since EM can be used for clustering, it became an ideal algorithm for finding unique polygons on the maps. Algorithm 1 shows the Expectation Maximization algorithm. The algorithm begins with random parameters. In this case, we use mu to describe the mean, and $covs$ to describe our covariances of a cluster. Then, for an N number of times, we alternately run the **Expectation** function, Algorithm 2, and **Maximization**, Algorithm 3. We assume the N will be a sufficient number of iterations that will allow the algorithm to converge.

Algorithm 2 shows our implementation of the Expectation in EM for our clustering algorithm. Given a set of parameters, means and covariances, we go through each cluster and draw new responsibilities for each point. Then, we multiply the responsibilities with the mixing weights for the cluster. After all points have responsibilities for each cluster, we normalize their responsibilities across clusters.

Algorithm 3 shows our implementation of Maximization. Given a list of weights for each point in each cluster, we now re-evaluate our parameters, mus and $covs$, and our mixing weights for the clusters mws.

With this algorithm, we are able to form coherent clusters around points in polygons. Given the inferred number of clusters given by the DNN mentioned above, we ran EM using multivariate Gaussian means and covariances as parameters to describe a cluster. Specifically, we use EM to

Algorithm 4 Simple Generative Model to Infer Lines from Points (LDA inspired)

```
given:
qs: list of lists of points belonging to a polygon for every polygon
lines: Per lines point Dirichlet distribution
for $n = 1$ to $N$ do
    $\mathrm{p} = \mathrm{ComputeDataLikelihood(qs, lines)}$
    $\mathrm{qs} = \mathrm{ResampleQS()}$
    for $y = 1$ to Y do
        $\mathrm{pplm[y]} =$ Dirichlet$(\alpha + \mathrm{c_{yl}[y]})$
    end for
    for $l = 1$ to L do
        $\mathrm{lines[l]} =$ Dirichlet$(\gamma + \mathrm{c_{pl}[l]})$
    end for
end for
```

Algorithm 5 Revised Resampling for Per-Line Point Mixtures Using Simple Heuristic

```
for $l = 1$ to L do
    $\mathrm{p_l} = \mathrm{PointsAssignedToLine(l)}$
    $\mathrm{LineReg_l} = \mathrm{RANSAC(p_l)}$
    for $p = 1$ to P do
        $\mathrm{Weights_l[l][p]} = \frac{1}{\mathrm{EuclideanDistance(p, LineReg_l)}}$
        $\mathrm{lines[l]} = \mathrm{Dirichlet}(\gamma + \mathrm{Weights_l[l]})$
    end for
end for
```

find polygons in a 2-D point cloud map. Once the algorithm converges with formed clusters, we convert the clusters into lists of points for each polygon and pickle the data. This data collection is done to move on to the next step of the full implementation.

Step 3 - Inferring Lines From Points

Approach A: Generative Statistical Model to infer lines in point cloud The next task is to use the data provided by the previous method, lists of points belonging to particular polygon clusters, to find linear relationships. By having points separated into their respective polygons, we are then able to focus on a single cluster of points at a time. This became useful when implementing some of our inference methods.

To infer lines from the given set of points is a difficult task. The first approach we attempted was a statistical generative model "highly" inspired by the Latent Dirichlet Allocation model, a topic modeling algorithm (Blei, Ng, and Jordan 2003). We constructed a model that followed very similar dependencies and focused on frequencies of points rather than linear relationship of points. We hoped to get a baseline from this attempt. Figure 5 shows our first generative model. Just as the LDA model contains a set of documents with words, and tries to find clusters of topics (word relationships), we hoped to replace the concept of documents with *polygons* and words with *points* to try to infer clusters of lines. Algorithm 4 shows the psuedo code of how we ran Full-Gibbs Sampling on the model. As seen, new distribu-

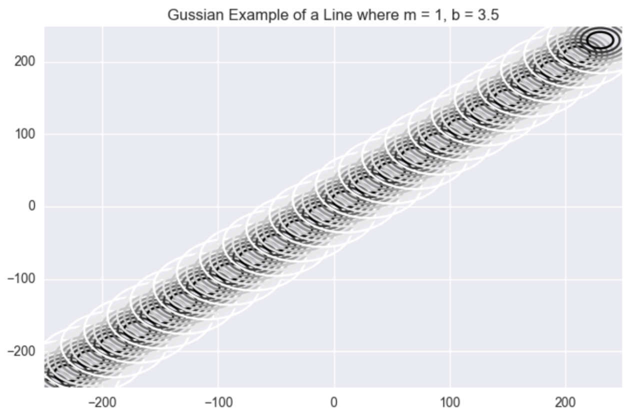


Figure 6: Describe the parameters of a line

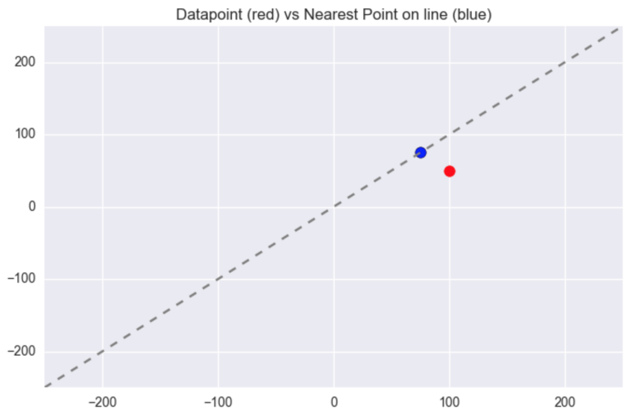


Figure 7: Describe for each point, we find the nearest point and create a Gaussian on it

tions are drawn for the per polygon line mixtures $pplm$ and per line point mixture $lines$ are both drawn from a Dirichlet distribution influenced by frequency counts. This proved to be unsuccessful, with results in the experiments section.

In Algorithm 5, we provide a revised snippet of code for the simple generative model discussed previously. Instead of resampling per line point mixtures using frequencies, we utilize the fact that points appear together in the data. We attempt to add a weighted linear relationship to the generative model by using a simple heuristic. Before assigning new weights for $lines$, we use qs to tell us which points are currently assigned to each line. For each line, we use the points assigned to each line and RANSAC to create a linear regression among those points. With the line predictions, we use the predicted line to find the distance of every point in the data to that line. Weights for a point belonging to a line are determined with a Dirichlet distribution using 1 divided by the Euclidean distance of the data point from the line.

Approach B: Expectation Maximization to infer lines in polygon cluster The next approach we attempted was to create relationships between points and lines was to use the

Algorithm 6 Expectation (REVISED FOR LINES)

```
given:
data: points from sample map
mws: mixing weights for each cluster
clusters = []
for $n = 1$ to $N$ do
  for $p = 1$ to $P$ do
    $\mathrm{nearestPoint_l} = \mathrm{GetNearestPtOnLine(p, slopes[n], yInter[n])}$
    $\mathrm{distance} = \mathrm{EuclideanDist(p, nearestPoint_l)}$
    $\mathrm{cluster_{np}} = \mathrm{N(distance, 0, COV)}$
  end for
  $\mathrm{cluster_n} = \mathrm{cluster_n} * \mathrm{mws[n]}$
  $\mathrm{clusters.append(cluster_n)}$
end for
$\mathrm{clusters} = \mathrm{Normalize(clusters)}$
return $\mathrm{clusters}$
```

Algorithm 7 Maximization (REVISED FOR LINES)

```
given:
$\mathrm{cluster_w}$ : list of weights for each point for each cluster

$\mathrm{mws_{new}} = []$
$\mathrm{slope_{new}} = []$
$\mathrm{yinters_{new}} = []$
for $n = 1$ to $N$ do
  $\mathrm{mws_{new}.append(Normalize(cluster_w[n])}$
  $\mathrm{weightedPts_n} = \mathrm{GetPointsAsWeights(data, cluster_w[n])}$
  $\mathrm{lineReg_l} = \mathrm{RANSAC(weightsPts_n)}$
  $\mathrm{slope_{new}.append(Slope(linReg_l)}$
  $\mathrm{yinters_{new}.append(YInter(linReg_l)}$
end for
return $\mathrm{mws_{new}, slope_{new}, yinters_{new}}$
```

same clustering EM algorithm we to cluster whole polygons on a 2-D point cloud map, but for each individual line. The only difference we made from the previous model was the number of clusters it tried on the data. We chose the number of clusters to represent the estimated number of point lines there were on the map.

From there, we decided to try a more sophisticated model of EM. The previous EM method used the parameters of means and covariance to describe multivariate Gaussian clusters on the map. In our final attempt to try to find lines in a polygon, we used EM again, but with parameters that described a line: a slope, and y-intercept as opposed to the covariance and mu of the cluster. We then created an infinite tube like Gaussian around every cluster, i.e line. An example of this tube-like Gaussian on a line is shown in Figure 6. This tube Gaussian allowed us to create appropriate responsibilities for each point for each cluster. We hoped that this approach would encourage linear clustering as opposed to rounded multivariate Gaussian clustering. To find responsibilities for every point for every line, we used the point and the line to find the *nearest point* on the line to the data point. This algorithm is shown in Algorithm 6 and Algorithm 7. Figure 7 shows an example the nearest point on

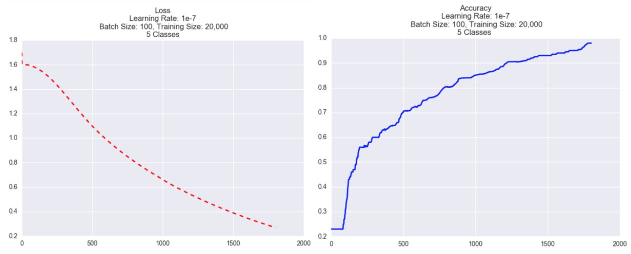


Figure 8: Loss and Accuracy of DNN. The network is trained on 1 classes (1 building, 2 buildings, 3 buildings, etc) with 4900 in each class. 20,000 random training samples, 5,000 testing samples. Accuracy reaches .98. Test Accuracy reaches .70

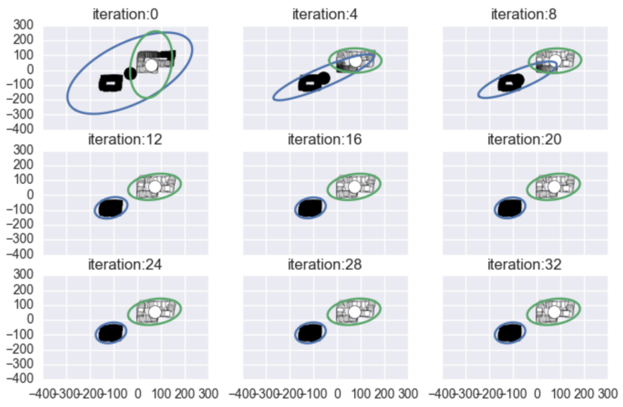


Figure 9: EM Clustering Algorithm to infer what points best belong to a polygon. Simple example of 2 polygons on a map.

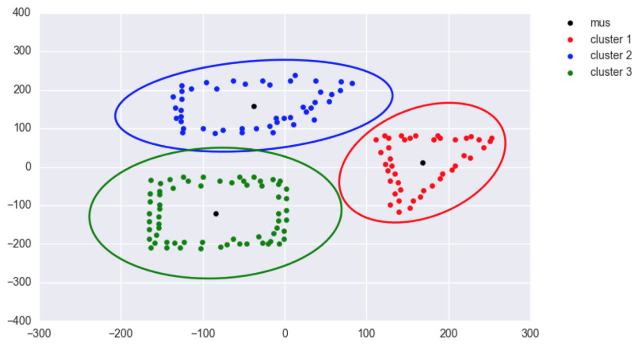


Figure 10: EM Clustering Algorithm to infer what points best belong to a polygon. Simple example of 3 polygons on a map.

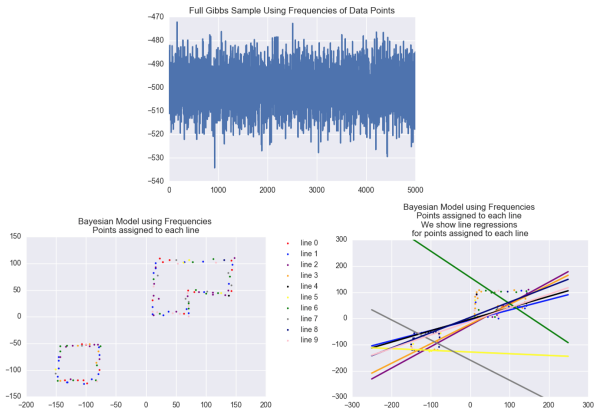


Figure 11: The results of what points belong to what line using full Gibbs with frequencies of points. Inspired by LDA's generative statistical model

the line (shown in blue) to a data point (shown in red). With the nearest point on the line, and the distance between the nearest point and the data point, we created a 1-Dimensional Gaussian centered on 0, and sampled the weight from the Gaussian with the distance between the two points as a parameter and a set a fixed covariance to describe a line. The closer the data point was to the nearest point, the more the weight became to one, and the further the data point was to the line, the closer it became to zero.

Experiments

Results of Training the CNN The CNN was trained on several learning rates: 1e-2, 1e-4, 1e-7, and 1e-8. As the learning rates decreased, the number of epochs needed to reach high accuracies (near 1) increased drastically. 1e-2 required 20 epochs and took minutes to run while 1e-8 took 14,000 epochs with a total of 3 days to run. As we decreased the learning rates for the CNN, the accuracies for the test data increased. 1e-2 produces test accuracies of .06, 1e-4 produces .15, 1e-7 produced .70, and lastly 1e-8 produced .70 as well. Although we were not able to produces high cutting-edge test accuracies, we proved a net is capable of learning to count clusters of data, specifically clusters of points on a map. Test accuracies further improved when we increased the number of sample maps for each classification (1000 maps per classification to 5000 maps). This means that we can improve our results by deepening our CNN and increasing the amount of training data. Figure 8 shows the training behaviors of the loss function and the accuracies for the learning rate 1e-7. Training the CNN with a learning of 1e-7 took 2 days and only required 1,800 epochs to reach .98 accuracy. The loss function decreased in a smooth manner and the accuracy increase and converged at .98 training accuracy.

Results of EM Clustering to Determine Polygons EM Clustering proved effective as a method of determining polygons out of the point cloud. Using the number of polygons from the CNN, we were able to separate the points into

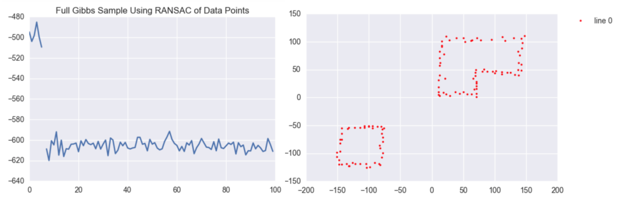


Figure 12: The results of what points belong to what line using full Gibbs with RANSAC as weights for the Dirichlet distribution.

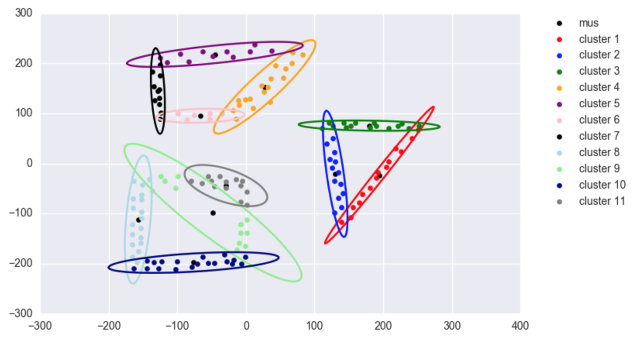


Figure 13: The final results from the EM clustering algorithm using the number of lines as the count for the number of Gaussians to put in.

their respective polygon clusters with high accuracy. However, this accuracy depended on the initial covariance, which was determined by the given number of clusters in the point cloud as shown in Figures 9 and 10.

Results on Full Gibbs based on Frequencies (Baseline) Full Gibbs sampling using frequencies of points to determine clusters did not perform significantly better than random line generation. Though the points tended to cluster slightly to a single polygon, the algorithm could not determine which points belonged to which line. This model could not converge and never produced accurate clusters of line representations. This was expected as clusters were formed using frequencies of points appearing near each other in each polygon. The method of determining the strength of the correlation of each point to a line was composed of a simple heuristic: the inverse of the distance. We believe that this simple heuristic contributed to lack of convergence. See results of the Full Gibbs sampling in Figure 11.

Results on Full Gibbs based on Linear Relationships As an experiment, we updated the manner in which we sampled weights for the per-line point mixture. We used a heuristic of 1 divided by the Euclidian distance between data points and lines as a parameter for our Dirichlet distribution. Figure 12 shows the results of this experiment. Unfortunately, our simple heuristic was not sufficient to produce more than one cluster. We know now that our model

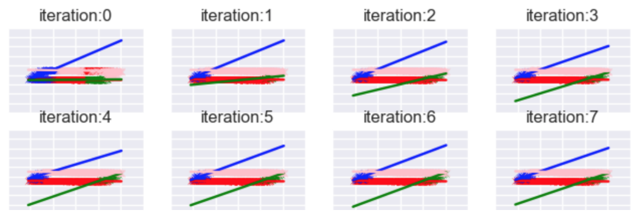


Figure 14: EM Line Clustering Algorithm to infer what points best belong to a line. Simple example of a single rectangle.

would have to have a similar heuristic used in our clustering EM algorithm with tube-like Gaussian to produce more accurate results. Perhaps using parameters such as the end points of a line segment would work better. Further experiments must be done.

Results of Expectation Maximization Clustering for Each Line The next approach we used to find linear relationships in polygon point clusters was using Expectation Maximization. In this experiment we focused on multivariate Gaussian parameters to describe clusters. This model is exactly the same as the EM clustering algorithm to determine polygons. We simply updated the number of clusters to form to represent the number of lines on the map. We did this to test if linear relationships could be determined. This model, with results shown in Figure 13 worked well for most polygons, and fit most of the lines correctly. We were pleasantly surprised. This model was very accurate when trying to cluster lines on just one polygon, and worked as well when trying to cluster lines on a map with multiple polygons. However, this model struggled fitting clusters to more complicated polygons, and we figured that this approach did well due to the simple map and cleanliness of the data. The downfall of this current method is that it requires a method of solving for how many lines in the map to look for. Since we have not created a DNN for this method, we have inserted the line count manually specific to each map.

Results of Expectation Maximization using Lines In this experiment we used Gibbs Sampling with the updated sampling for the per-line point mixtures. We used tube-like Gaussians to determine weights for each point for each cluster. Figures 15 shows an example result of how well the algorithm performed on a single polygon sample of three sides. 14 shows another example where this method worked well on a quadrilateral. For these simple polygons, convergence of the algorithm occurred within few iterations. However, this method proved unsuccessful when attempting to cluster lines on more complicated polygons. This approach was very dependent on the initialization of the line slopes and y-intercepts. Several times, lines would end up clustering in the same location which led to a single line being formed by multiple lines in the data. Each cluster would simply capture different sections of noisy data.

To further test if this algorithm could perform well for maps with several polygons, we ran the algorithm on a map consisting of three polygons. Figure 16 shows the results of

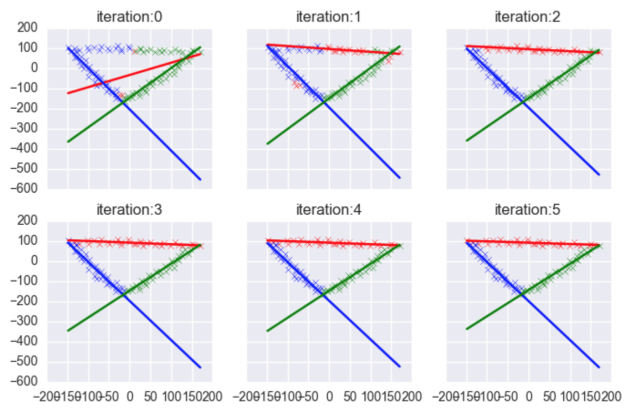


Figure 15: EM Line Clustering Algorithm to infer what points best belong to a line. Simple example of a single triangle.

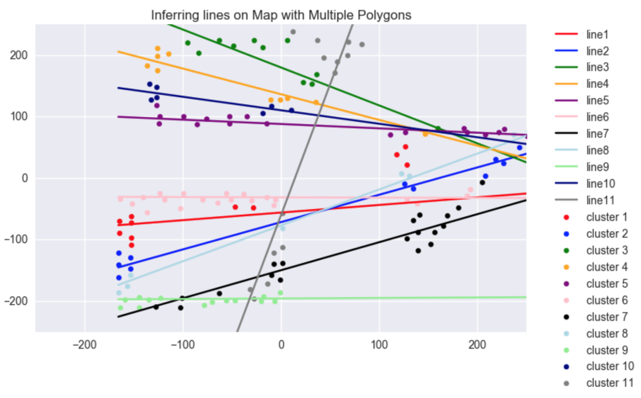


Figure 16: EM Line Clustering Algorithm to infer what points best belong to a line. Simple example of 3 polygons on a map.

this experiment. There, we can see that lines were shared across polygons. This is when we realized that perhaps this method could be augmented with parameters that described line segments instead of lines that go across the entire map. This method proved inaccurate for multiple polygon point clusters and would require previous polygon sorting as mentioned earlier to achieve accuracy.

Conclusion

We found the EM clustering method to be the most effective for finding the points belonging to each line. Of the experiments, it was the most consistently correct in clustering point collections as line structures, and provided sufficient information to determine coarse line segment representations for each cluster. The consistently accurate point collections for each line could quickly be used to find linear regressions using RANSAC or other traditional methods. Additionally, this method works for any number of polygons in a simple 2-D point cloud map. However, experiments demonstrate it is not completely reliable with complicated polygons, which have lines that are not always correctly found using this method. Additionally a new DNN would need to be created to determine the appropriate number of lines in a 2-D point cloud map of polygons. In conclusion, this method proved to be the most consistent and provides the most information to infer a semi-coherent map model of polygons from 2-D point cloud maps.

Future Work

We are really motivated to continue this research problem using more sophisticated models. All these experiments allowed to learn more about the issues of this problem and the behaviors of certain algorithms applied to the data. We are really interested in the possibility of using parameters that describe line segments instead of full lines across the map, and perhaps trying new heuristics using statistical generative models. Another interesting augmentation we could try is using these algorithms on 3-D point clouds.

References

Blei, D. M.; Ng, A. Y.; and Jordan, M. I. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3(Jan):993–1022.

Dataset generated by dorit borrman and andreas nchter of jacobs university bremen. http://kos.informatik.uni-osnabrueck.de/3Dscans/. Accessed: 2017.

Derpanis, K. G. 2010. Overview of the ransac algorithm. *Image Rochester NY* 4(1):2–3.

Duda, R. O., and Hart, P. E. 1972. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM* 15(1):11–15.

Gilks, W. R.; Richardson, S.; and Spiegelhalter, D. 1995. *Markov chain Monte Carlo in practice*. CRC press.

Moon, T. K. 1996. The expectation-maximization algorithm. *IEEE Signal processing magazine* 13(6):47–60.

Se, S.; Lowe, D. G.; and Little, J. J. 2005. Vision-based global localization and mapping for mobile robots. *IEEE Transactions on robotics* 21(3):364–375.