Autonomous Robotic Navigation Using Self-Organized Maps

Becky Tang and Alex Robey

December 18, 2017

Abstract

In this paper we propose a novel algorithm that allows an agent to navigate from a starting location to a goal location. Just as humans use a combination of senses to locate themselves in their surroundings, our algorithm uses a variety of sensors to allow an agent to localize itself in its world. Furthermore, this localization method facilitates the training of a self-organizing map; this map conforms to the paths that avoid obstacles and walls in a simple world and creates routes for an agent to follow. With this approach, we show that an agent can successfully navigate in a simple world in a semi-autonomous way. Additionally, we provide an analysis of the activations of the layers in the neural networks used in our algorithms and we show that deeper layers find more distinctive features and representations of input data.

1 Introduction

Navigation from one place to another requires an understanding of the layout of an environment and of the obstacles contained within it. For humans, navigation is relatively trivial. After taking a 360° scan of a room, one can almost always locate the door and navigate towards it. However, for robots, navigation is not so simple. Even if a robotic agent knows the position of a goal location, it has no *a priori* way of determining how to navigate toward the target without encountering obstacles or walls. One approach to solving this problem is to allow an agent to construct a map of the layout of its world; this map in turn can be used to determine a sequence of actions that will move the agent closer to its target.

In this paper, we demonstrate an algorithm that enables a robotic agent to locate itself in its world and to navigate from one location to another in a semi-autonomous manner. In particular, our approach has two distinct steps. First, we show that a robot can successfully and accurately find its position in its world. To do so, we used two kinds of artificial neural networks (ANNs). In particular, we used feed-forward and convolutional neural networks (CNNs) to predict the robots (x, y) location based on sensor and image inputs. Because ANNs transform input data into more abstract representations, we also explore the ways in which different layers of nodes in the ANNs we used for the localization task provide different kinds of information for correctly predicting location. In the second step, we use these predicted locations as input into a self-organizing map (SOM), which is another kind of ANN. We used the SOM to develop a path for navigation from one location to another; this demonstrates that an agent can learn from data to navigate in a simplistic world.

2 Background

Robotic navigation is a notoriously difficult and convoluted task. Our approach has been influenced by a number of important past results; in this section, we will enumerate on some of these results.

One of the most well-known and widely-used algorithms for localizing a robot in its environment is known as Simultaneous Localization and Mapping, or SLAM. SLAM is used to simultaneously map an unknown environment and to keep track of the position of an agent in that environment. By taking sensor readings as input, SLAM algorithms often seek to detect landmarks and estimate the current state and position of an agent. For example, Cheein et al. showed that a SLAM algorithm controlled by can electromyographic signals can be used to successfully navigate in an unknown environment [2]. In this work, the authors demonstrate that an autonomous motorized wheelchair agent can learn to navigate in an unknown environment using a kinematic controller. While we did not explicitly use SLAM in this project, our algorithm draws inspiration from the idea of constructing a map of an unknown world based on sensor readings. Although SLAM is often used to reconstruct a 3D representation of a given world, in this paper we will only discuss the simpler case of 2D reconstruction.

We were also motivated by the work of Ran et al., who formulated the robot navigation task as a series of classification tasks. They experimented with using a spherical, 360° fisheye panorama camera to capture images [5]. They proposed an end-to-end CNN framework, where inputs into the network were raw, uncalibrated spherical images, and the outputs were steering direction and angle. However, our method is more practical as we do not require the use of a panoramic camera. On the contrary, the images we use to navigate are relatively low resolution.

We chose to incorporate a SOM after studying Blank et. al's work on developing a path-planning neural network for the purpose of goal-seeking [1]. Their work demonstrates the successful implementation of using camera images and sensors as input to create abstract representations of both the current state and the goal state. In one of the their experiments, the input data is transformed into two SOMs: one for images and one for sonar values. These abstract representations were then used to determine an action towards achievement of a navigation goal.

One of the primary aims of this project was to reproduce and improve upon the work of Vlassis et al. Their paper "Robot Map Building by Kohonen's Self-Organizing Neural Networks" details a method for allowing a robot to navigate in a simple environment by reconstructing the world with a self-organizing map. Their results show that SOMs can be effectively trained to create a map of the world that avoids obstacles and walls. However, Vlassis et al. only trained their SOM on (x, y) pairs of input data to generate their graph [3]. Our paper presents an end-to-end system that first predicts the location of a robot based on camera and sonar data, and then creates the SOM using these predictions.

3 Theory

Self-organizing maps (SOMs) are a kind of ANN that were first conceived by Teuvo Kohonen in the early 1980s [4]. SOMs are trained using unsupervised learning. This stands in stark contrast to more classical ANNs, which "learn" from data in a supervised manner by means of labeled training data. SOMs are generally used for projecting high-dimensional data down to a lower dimension and are thus useful for determining a low-dimensional representation of an input space. This representation is sometimes referred to as a *map*.

SOMs are initialized by arranging a set of nodes I into a lattice. This is shown in Figure 1(a). The high-level idea is that, after initializing and adjusting the weights of the nodes in the SOM, the graph will form a representation of a given a set of n-dimensional training data. To this end, each node $i \in I$ in the lattice is initialized with a weight $\vec{w_i} = \{w_{i1}, w_{i2}, \ldots, w_{in}\}$ that corresponds to its starting location in the lattice. For simplicity, the weights of the nodes are generally normalized between 0 and 1.

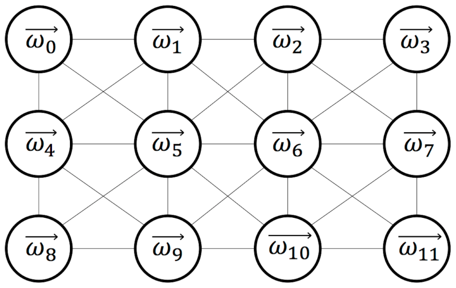
After initializing the nodes, a random sample $\vec{x}$ is drawn from the set of training data. For each input vector, the goal is to find the *best matching unit*, or the node k with weight vector $\vec{w_k}$ that is most similar to the input vector $\vec{x}$. In particular, the goal is to find the unique node $k \in I$ with a corresponding weight vector $\vec{w_k}$ satisfying

$$\arg\min_{i \in I} ||\vec{x} - \vec{w_i}|| = \vec{w_k}.$$

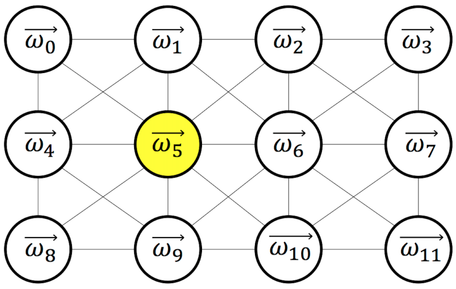
The SOM "learns" from data by iteratively updating the weight of the best matching unit for each input data point $\vec{x}$. In this project, we employed a simple weight updating scheme. In particular, the weight $\vec{w_k}$ of the best matching unit is updated by

$$\vec{w_k} = \vec{w_k} + \alpha(t)\,(\vec{x} - \vec{w_k})$$

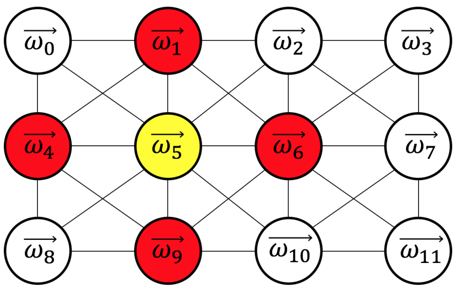
where $\alpha(t)$ is a time-decaying learning rate. One such best matching unit is shown in yellow in Figure 1(b). The weights of the nodes neighboring the best matching unit are also updated, as shown in Figure 1(c). A time-decaying radius $r(t)$ is commonly defined to determine which



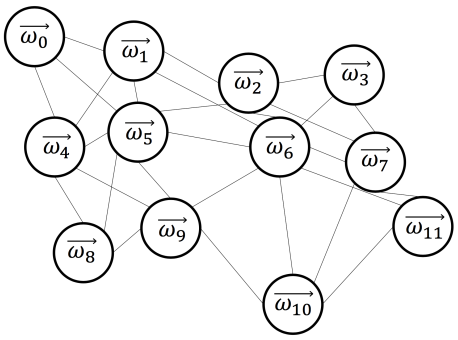
(a) The nodes in a SOM are initially placed in a lattice. Each node is initialized with a unique weight that corresponding to its location in the lattice.



(b) During each training step, we look for the best matching unit, or the cell whose weight is closest to the input vector.



(c) The cell associated with the weight that is most similar to a given input vector (shown in yellow) is updated during the training process. Furthermore, the weights of each of the bordering nodes (shown in red) are also updated.



(d) After training, each node moves to a location corresponding to the updated weight of each node in the lattice. This "map" preserves the topological properties of the input space.

Figure 1: The training process for SOMs.

neighboring nodes are updated; if the Euclidean distance between the weight of the best matching unit and a neighboring nodes is less than the radius $r(t)$, the weight of that neighboring node is also updated. To be precise, the weights $\vec{w_j}$ of the nodes neighboring node k are updated by

$$w_j = w_j + \alpha(t)\beta(t)\,(\vec{x} - \vec{w_j}) \qquad \text{where} \qquad \beta(t) = \exp\left(-\frac{||\vec{w_k} - \vec{w_j}||}{[r(t)]^2}\right)$$

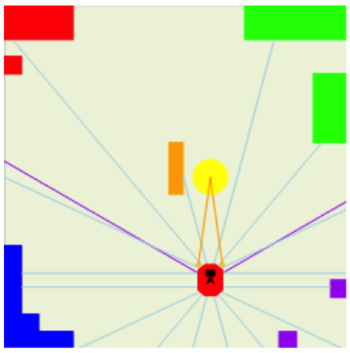
and where node j neighbors node k if $||w_j - w_k|| \leq r(t)$. In this expression, $\beta(t)$ is a time-decaying influence rate that ensures that the weights of the neighboring nodes are updated by a lesser amount than the weight of the best matching unit. Since the radius decays over time, fewer neighboring nodes are updated as training progresses until only the best matching unit is updated in a given time step.

The process of comparing a randomly drawn sample to the weights of the nodes in the lattice and updating the weights accordingly is repeated for a set number of iterations. After training completes, the resulting map no longer resembles the uniformly arranged lattice of Figure 1(a). Instead, the map resembles a graph that conforms to the properties of the training data, as in Figure 1(d). For the sake of visualization, each cell in the lattice is generally connected to its four or six nearest neighbors. Although these connections do not impact the training process, they are useful for understanding how weights change during training.

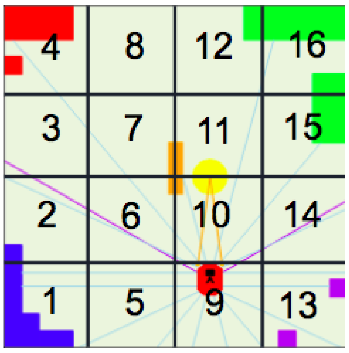
4 Experiments

4.1 Prediction Task

In the first part of our experiment, we attempted to solve the following prediction task: can an agent predict its (x, y) coordinates based on camera images and sensor data? To determine whether this task would be feasible, we first attempted to solve a simpler classification task. In particular, we sought to train an agent to predict which part of a partitioned world it was located in based



(a) One of the simple worlds we created using the Jyro simulator.



(b) An example partitioning of a world into 16 sections.

Figure 2: We created simplistic worlds using the Jyro simulator. The data collected in these worlds was used to solved the prediction task of localizing the robot in its world.

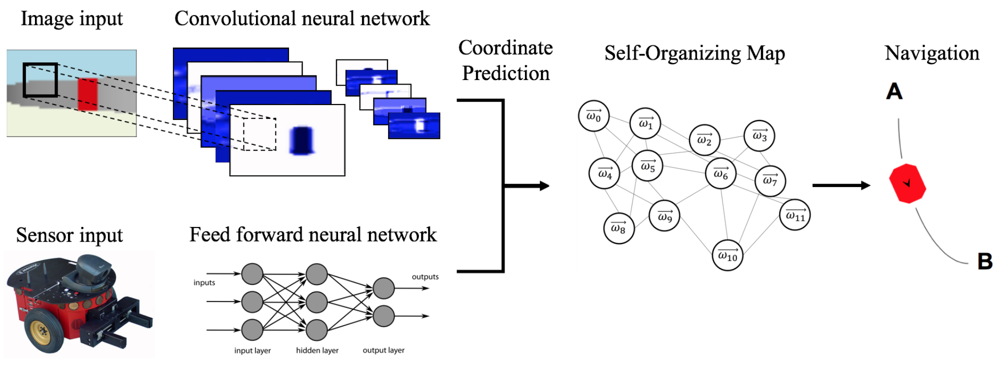


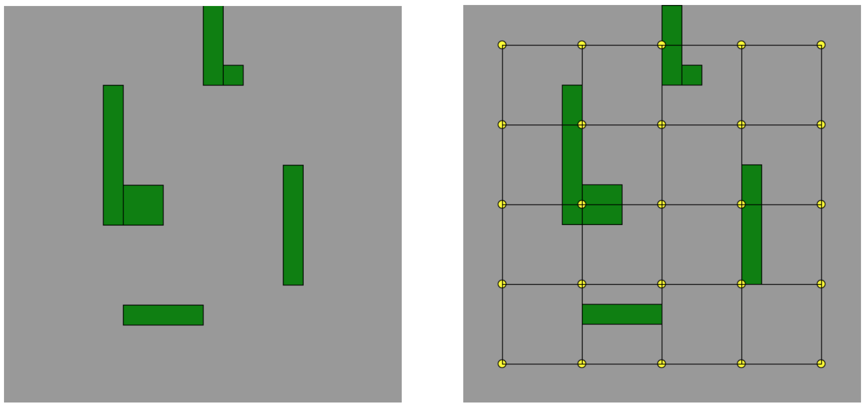
Figure 3: End-to-end pipeline used to allow an agent to navigate from a starting location to a goal location in a simple world.

on its sensors and camera images. A simple world and its partition into sixteen different sections are shown in Figures 2(a) and 2(b) respectively. We attempted to solve this classification task by using three different neural network architectures:

1. A CNN which uses 40×60 pixel images as input.
2. A feed-forward neural network which uses the values of two light sensors and eight frontward-facing sonar sensors as input.
3. A bifurcated neural network with two branches: a CNN branch which uses images as input and a feed-forward branch which uses light sensors and sonars as input. These branches were merged so that the network predicts location based on both sensors and images.

We trained and tested each of these network architectures on the same world. This world is shown in Figure 2(a). In this world, each of the corners are visually distinctive due to differences in color and physically distinctive due to different block sizes and shapes; we hoped that this would differentiate the sections of the partition and make the classification task easier for the agent. The agent collects training data, which consists of sensor readings and the ground-truth location of the robot, with a simple obstacle avoidance algorithm. At each time step, the agent records a 40×60 pixel image from the front-facing camera and the values of the two light sensors and eight sonars.

The architectures of the three networks used in this experiment were fairly simple. The feed-forward network consisted of two hidden layers, the second of which fed into the classification output layer. In general, we found that the most successful networks required 30 nodes in the first hidden layer and 15 nodes in the second hidden layer. The convolutional neural network consisted of two convolutional layers, each of which was followed by a max-pooling layer. The second max-pooling layer was followed by a flattening layer and a hidden layer, which in turn fed into an output layer. The number of nodes in the output layers for both the feed-forward and


(a) The simple world used to train the self-organizing map.

(b) The initialization of nodes in the SOM arranged in a lattice.

Figure 4: The world used to train the self-organizing map has more central obstacles, which poses a more difficult navigation challenge.

convolutional neural networks was determined by the partition of the world into smaller subregions. For an $n \times n$ partition, we used n^2 output nodes - one for each section of the partition. Thus for the partition shown in Figure 2(b), there were 16 nodes in the output layer.

The merged network was a combination of the feed-forward and convolutional neural networks described above. The feed-forward and convolutional neural network both feed into a hidden layer. This hidden layer is connected to a smaller hidden layer, which in turn feeds into the output layer.

We also attempted to use a similar network to predict an agent's (x, y) location based on sensor and camera data. To do so, we simply changed the categorical cross-entropy loss function used to perform the classification to a mean-squared error loss function. The results from the partition classification and coordinate prediction neural networks are presented in the Results section of this paper.

4.2 Navigation Task

In the second part of our experiment, we designed an algorithm to determine a path from a starting location to a goal location in a simple world. To do so, we used the coordinate prediction merged network described in the previous section to predict an agent's (x, y) location based on sensor and camera data. These predictions were then used to train a self-organizing map. In this way, we use SOMs to create a map that conforms to the topological properties of the space of predicted (x, y) locations. The end-to-end pipeline for this experiment is shown in Figure 3.

To test this algorithm, we created a new world similar to the world described in Vlassis et al. We chose to use this world rather than the world in 2(a) because it poses a more interesting navigation challenge; because there are more barriers and obstacles in the center of the world, it is more difficult for the agent to navigate from one place to another. The world we used and the SOM initialization are shown in Figure 4(a) and 4(b) respectively.

After training the SOM, we ran Dijkstra's algorithm on the nodes of the reorganized graph. Because the SOM conforms around the obstacles in the world, its connected nodes create routes that an agent can use to move from a start location to a goal location. In particular, the agent need only follow a finite number of straight line segments along the reorganized connections between nodes in the graph. The resulting map and the paths connecting different locations in the world are shown in the Results section of this paper.

5 Results

Before testing our networks, we predicted that the merged neural network architecture would outperform the convolutional and feed-forward neural networks. To test this hypothesis, we compared the performance of the partition classification task on several partitions of the same world.

# of Partitions	Network	# of Epochs	Training Accuracy	Validation Accuracy
4	CNN	20	97.2%	97.8%
	Feed-Forward	20	96.2%	90.2%
	Merge	20	97.9%	98.2%
9	CNN	50	97.7%	96.5%
	Feed-Forward	50	87.8%	85.0%
	Merge	50	97.3%	96.6%

Table 1: Comparison of the CNN, feed-forward, and merged network on predicting partition in a world divided into four and nine sections.

# of Partitions	# of Epochs	Training Accuracy	Validation Accuracy
16	60	91%	87%
36	50	88%	84%
100	150	85%	82%
400	300	84%	70%

Table 2: Performance of the merged network on various partitionings of the world.

5.1 Prediction Task

As shown in Table 1, each of the three networks performed comparably on the world partitioned into four sections; all networks achieved training and validation accuracy in the high 90% range. However, the feed-forward network performed noticeably worse on the 3×3 world, which agrees with our hypothesis.

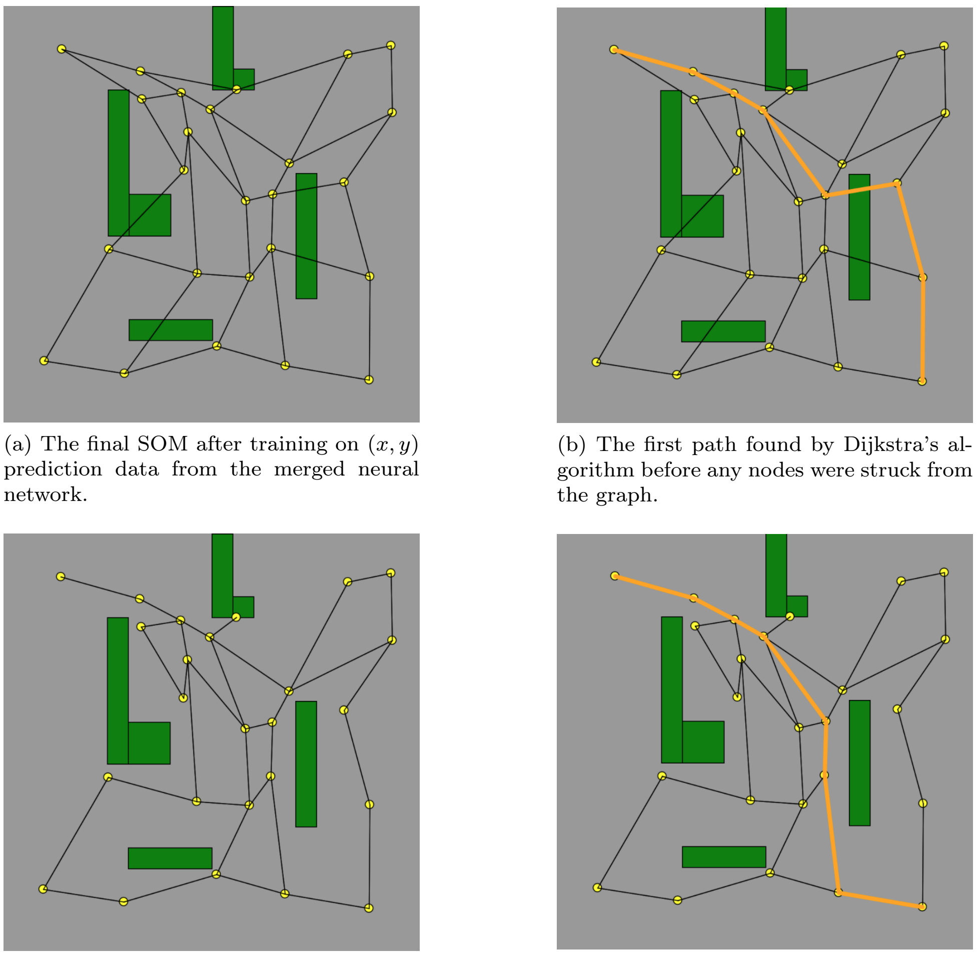
We also tested the robustness of the merged network on more finely-partitioned worlds. Table 2 shows that as the number of partitions increased, the performance of the network seemed to drop. This is not entirely unexpected, because there were many sections of the fine partitions that the robot never encountered during training; for example, parts of the 20×20 partition were right in the corners of the world in Figure 2(a) and thus were inaccessible to the agent. However, perhaps the most encouraging result from these localization tests was the accuracy of the coordinate prediction architecture. After 2000 epochs, this network leveled off at a training error of 0.17 and a validation error of 0.44. Because this network was a part of our end-to-end navigation system, these low errors were crucial to the results of the self-organizing map.

5.2 Navigation Task

As shown in Figure 5(a), the SOM architecture successfully learned a representation or "map" of the input space. This figure shows that the nodes conform around the boundaries and obstacles of the simple world of Figure 4. However, the map in Figure 5(a) is imperfect, as it contains connections that travel through obstacles in the world. This is a problem for the agent, because as shown in Figure 5(b), Dijkstra's algorithm can find paths that pass over obstacles. We correct for this by allowing the agent to follow the path found by Dijkstra's algorithm until it reaches an obstacle. If its sonars detect an obstacle, the agent strikes the bad connection from the graph and returns to the start location. Dijkstra's algorithm is run once more, and the agent begins to follow a new path. An updated SOM found by the agent after detecting the bad connections is shown in Figure 5(c). Finally, the agent can use Dijkstra's algorithm to navigate from the top left node to the bottom right node of the world, as shown in Figure 5(d).

6 Discussion

As expected, the network that used only sensors and sonars as input did not perform as well as the other networks. We believe that the convolutional neural network was the most effective tool in enabling the agent to create a rich and informative representation of the world. Therefore it is not entirely surprising that the networks that used convolutional layers outperformed those that did not. While we were initially surprised by how similarly the CNN and merged network performed,



(a) The final SOM after training on (x, y) prediction data from the merged neural network.

(b) The first path found by Dijkstra's algorithm before any nodes were struck from the graph.

(c) The SOM after the robot has followed dead end paths and struck bad connections from the graph.

(d) The path found by Dijkstra's algorithm after bad connections are struck from the graph.

Figure 5: The final SOM and the paths created before and after eliminating the edges that pass over known obstacles.

this is most likely attributed to the fact that the sensors and sonars are simply not as informative as we initially expected. In other words, the CNN and merged network most likely performed similarly because images are more informative for this task than sonars or light sensors.

The performance of the merged network does decrease as the world is more finely partitioned (Table 2). However, provided that these networks are trained for a large enough number of epochs, it is possible to achieve high training and validation accuracies. As we have shown, after the agent trains 2000 training epochs on the coordinate prediction data, the agent effectively locates itself using images, light sensors and frontward-facing sonar values as input.

In addition, the corners of the world in which we trained and tested the agent were both physically and visually unique. We believe that the uniqueness of color leads to higher performance in prediction of partition for the CNN. In this way, if we performed a similar task where every corner was the same color, we would expect the merged network to outperform the CNN. Just as humans usually rely on multiple senses to orient themselves and navigate, best performance will likely occur when the network has access to many or all possible input data sources. The robustness of the merged network suggests that, like humans, the network requires access to all of its available senses to accurately predict its location in the world.

The agent's performance on the SOM navigation task was highly encouraging. The results suggest that the agent was able to create a map of the world that conformed around the boundaries and obstacles in the world. This created routes for the robot to follow from one location to another. Furthermore, this method was made even more robust by identifying and stiking graph connections that passed over obstacles. These results build upon the work done by Vlassis et al. because they

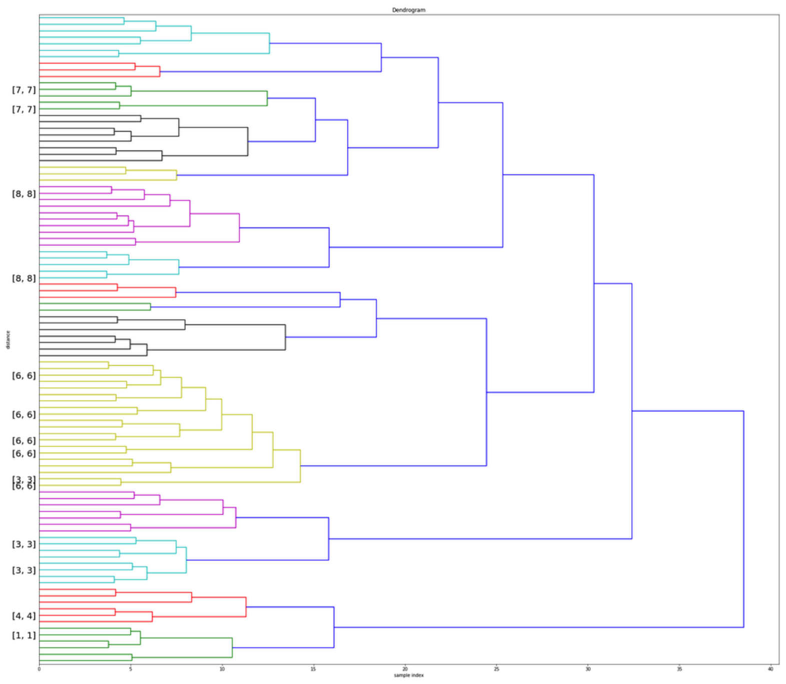


Figure 6: Dendrogram visualizing the clustering from nodes of the final hidden layer of the merged network for a world divided into 9 partitions. The first number in the labels denotes the predicted partition, and the second number denotes the actual partition.

show that agents can use their sensors to create an effective self-organizing representation of a simple world.

6.1 Visualizations

As mentioned in the introduction, we were also interested in visualizing how the merged network uses the input data to accurately classify an agent's position. We first performed cluster analysis with a dendrogram and visualized the activations of the ten nodes from the last hidden layer in the merged neural network for a world partitioned into nine sections (Figure 6). The dendrogram shows hierarchical clustering, so from left to right, the data goes from most similar to least similar. The colors in the dendrogram denote clusters, determined by closest Euclidean distance. The labels along the y-axis show the predicted and ground-truth partition. We see that data that correspond to partition six are grouped together in yellow. Similarly, the data corresponding to the third partition are clustered in bright blue as are the data corresponding to partition seven in green. However, the clustering is somewhat imperfect, which is to be expected because the network is not 100% accurate. Nevertheless, because points from the same partitions were clustered together, we can confirm that the network did a reasonable job of classifying the partitions.

However, the dendrogram cannot tell us *which* features are the most important for performing the classification task. To try and answer this question, we performed Principal Component Analysis (PCA) to better understand which data were most useful for the classification task. The main idea of PCA is to project high-dimensional data down to a lower dimension for the sake of visualization and analysis. PCA takes a set of possibly correlated data and transforms it into a set of uncorrelated variables called principle components. Each of these principal components (PCs) explains some amount of variation of the data, and because the PCs are uncorrelated, the amount of explained variation is solely representative of that PC. Therefore, we are interested in the PCs that explain the most variation in the data.

For the world partitioned into four and nine sections, we plotted the data projected onto the first two PCs for three layers of the network: the input layer, which took image and sensor input, the thirty nodes from the first hidden layer after the merge, and the fifteen nodes from the second hidden layer after the merge. Because the images have 40×60 pixels, and each pixel is three-dimensional (RGB), the images combined with the ten sensor valued created 7210 dimensions. As the plots in Figure 7 show, merging the networks and having two hidden layers leads to clearer,

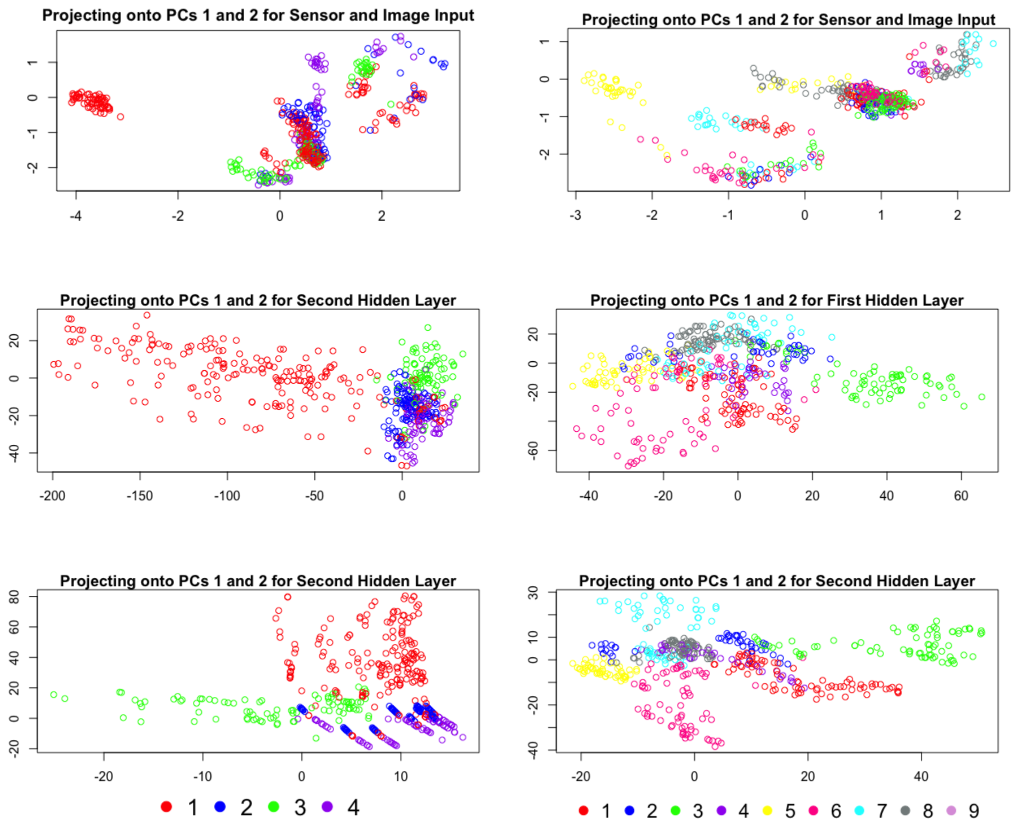


Figure 7: Projections of predicted partitions onto principal components after 1) the sensor and image input layer (top); 2) the first hidden layer after the merge (middle); and 3) the second hidden layer after the merge (bottom). The plots for the world with a partition of four sections are shown on the left and the plots for the world with a partition of nine sections are shown on the right.

more distinctive classification of partitions. Each color represents a partition. The first plot shows how the input data itself appears plotted on the first two PCs, and it is quite apparent that the data do not lead to distinctive clusters. However, the partitions for the first hidden layer after merge are much more separated. Similarly, the plots corresponding to the second hidden layer after the merge show even tighter clusters with less overlap. Thus, these plots show us that having a second hidden layer after the merge does in fact aid in partition prediction.

7 Conclusions and Future Work

In this paper, a convolutional neural network and a feed-forward network were merged together to create a robot localization framework. The task of navigation began with accurate prediction of the robot's location in the world, using inputs of images and sensor values. Then the output of predicated coordinate position was fed into a self-organizing map to learn a representation of the world and to generate a path from a start location to a goal location.

We were pleased that the agent was able to create a self-organizing map of its world using sensor inputs. However, we would like to extend this work to more challenging worlds. We believe that this architecture can be improved upon by incorporating other sensors and by tuning the hyperparameters of the self-organizing map algorithm. We are also interested in making the visual and physical features of a world less distinct, which would make the localization task much more difficult. Finally, we would like to perform PCA and create dendrograms for deeper, more complex neural networks to determine how tightly clustered data are at even deeper layers.

References

[1] D. Blank, D. Kumar, L. Meeden, and J. Marshall. Bringing up robot: Fundamental mechanisms for creating a self-motivated, self-organizing architecture. *Cybernetics and Systems*, 2005.

[2] Cheein et al. SLAM Algorithm Applied to Robotics Assistance for Nnavigation in Unknown Environments. *Journal of NeuroEngineering and Rehibilitation*, 2010.

[3] Vlassis et al. Robot Map Building by Kohonen's Self-Organizing Neural Networks. *Proc. 1st Mobinet Symposium*, 1997.

[4] Teuvo Kohonen. Self-Organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 1982.

[5] L. Ran, Y. Zhang, Q. Zhang, and T. Yang. Convolutional neural network-based robot † navigation using uncalibrated spherical images. *Sensors*, 2017.