Belief-Based Localization using Deep Neural Networks

Timmy Hussain and Alexandros Tzikas
Stanford University

BIOGRAPHY

Timmy Hussain is an M.S Candidate in the Department of Aeronautics and Astronautics at Stanford University. He received his B.S in Aerospace Engineering from Massachusetts Institute of Technology. His research interests include the control and navigation of autonomous systems.

Alexandros Tzikas is a Ph.D. Candidate in the Department of Aeronautics and Astronautics at Stanford University. He received his B.S. in Electrical and Computer Engineering from the Aristotle University of Thessaloniki. His research interests are in the field of navigation of autonomous systems.

I. INTRODUCTION

Autonomous robots are currently an exciting technological area, because of their multi-faceted applications that include autonomous driving, exploration, wildfire surveillance and search and rescue, among others. One of the important challenges in the field of autonomous robots is the problem of localization. Localization is required by autonomous agents, since it allows for the completion of various downstream tasks, such as planning and target navigation. Due to its ability to greatly enhance the performance of a robot, but also due to its importance in the safety of the robot system and of other agents in the environment [1], localization is considered as one of the fundamental problems in mobile robotics [2]. Furthermore, for a robot system, there is noise and uncertainty in sensing and motion. Left unaccounted for, the noise can result in a degradation of the robot's localization ability over time, as errors accumulate.

1. Related Work

It is typical to consider localization works along the axis of global and local localization. Local localization requires a known initial position for the agent. The agent's position is then tracked as it moves. The local localization algorithm introduces small corrections to the estimate to account for the change in position due to the robot's motion. The Kalman filter, geometry-based visual odometry and learning-based visual odometry are some of the methods that can perform local localization. A major limitation of local localization algorithms however is the number of restrictive assumptions about the robot's position. For example, the Kalman filter assumes that the uncertainty follows a Gaussian distribution. The restrictive assumptions lead to an inability to handle significant localization failures. In addition, after losing track, it is impossible for the algorithm to recover [3]. Consequently, local localization algorithms require human monitoring [4]. On the contrary, global localization assumes that the initial position of the robot is unknown. Therefore, global localization is a far more difficult problem than local localization [4] and in fact, it is still an open problem. In global localization, however, we infer the position of the robot in a global frame of reference. Due to the more relaxed assumptions, global localization methods can cope with global uncertainty of position and are thus suitable to tackle the kidnapped robot problem and handle scenarios with serious positioning errors [3], providing added robustness. For the aforementioned reasons, global localization techniques are often preferable.

We may also consider localization along the axis of the utilized framework; model-based or learning-based. Model-based approaches are typically probabilistic implementations that require some sensing and motion models to be provided for the robot. One such method is Markov localization [3] which maintains a probability distribution over the possible robot positions, updating it if new measurements arise. For model-based methods however, inaccuracies in the models may seriously degrade performance. It is nonetheless important to mention that model-based methods are able to account for the underlying uncertainty when employing a probabilistic framework. They can also perform sensor fusion, which essentially bridges the gap between different sensor modalities, although explicit modelling, which might be challenging, is required. In recent years, there has been a shift towards more end-to-end and learning-based approaches [5]. End-to-end approaches have two benefits: they preclude the need for explicit sensor modelling or principled fusion techniques and they allow the localization problem to be solved alongside a secondary objective, such as navigation. While this is the case, and in contrast to the model-based approach which isolates the localization problem, it is not uncommon to have learning-based localization solutions embedded within an end-to-end network. Learning-based frameworks are able to generalize well to unseen scenarios and environments (given that they learn

patterns and features from data) and are thus more robust to unpredicted errors [6]. In addition, learning-based architectures open up the possibility of combining multiple information sources without needing to explicitly model the relationship between the sources. However, while these approaches do not require principled models of the underlying information sources, the use of neural networks still elicits the need for a principled model architecture, training and evaluation process.

Finally, there are various options for the sensor modality used as the input to the localization algorithm. Sensor modalities such as GNSS receivers are able to provide global localization, but are strongly dependent on the environment (line of sight is required between receiver and GNSS satellites), while environment-agnostic sensors such as IMU/odometry encoders work well for local localization, but can easily fall victim to error accumulation over time and could lead to large deviations between the robot's belief of its position and its actual position. LiDAR data is an attractive input, because it can be efficiently simulated with high-fidelity, it is easily transferred between simulation and the real world and it also has a relative small size [7]. Use of LiDAR within learning has been explored, although the formulations are often quite different. In [8], the LiDAR measurements are passed through a network to produce an intensity map, which is compared to a similar embedding obtained from the ground truth map and the comparison is done via convolutional matching. While this approach extends well to any LiDAR model, it requires training a model to generate this embedding given a measurement. Images can also be used as the input to the localization method. However, images are computationally expensive to simulate and have a large size [7]. In addition, if images are used, the localization framework must be robust to various factors, including illumination changes.

2. Contributions

In this paper, we design a neural network framework for global localization given a known map. Our method solves for a belief over a given map with no prior knowledge of the robot's position (also known as the kidnapped robot problem). It is therefore able to cope with larger amounts of noise and can be used in scenarios with serious positioning errors, being able to even handle the kidnapped robot problem. Our learning-based framework allows for more robustness to unexpected errors and enhanced generalization ability, with minimal modeling required, in contrast to model-based techniques. We train the network with LiDAR measurements, but avoid learning unnecessarily complex representations, such as LiDAR to images or intensity representations. We compare our approach against a model-based baseline and find that our algorithm performs equally well with a standard model-based method, while being faster.

3. Organization

The remainder of the paper is organized as follows. In Section II we describe the problem formulation. Section III discusses the baseline approach we compare our method against while Section IV provides details of our proposed approach. Section V explains the data collection process using AirSim and finally, in Section VI we present our simulation results. Section VII concludes the paper, Section VIII discusses potential future extensions and Section IX briefly mentions the contributions of the team members to the project.

II. PROBLEM STATEMENT

We assume a known map of the environment in which the agent of the problem is located. The map consists of an occupancy grid M of dimensions $d_x \times d_y$, where each cell in the grid is either empty, occupied by an obstacle or occupied by the agent. A cell occupied by an obstacle is represented by a 1 in the occupancy map, while all other cells are represented as a 0. The dimensions of the cells, dx and dy, are assumed to be known in advance. An example map is shown in Figure 1.

We assume that the agent can occupy a cell in the map and the coordinates of the cell indicate the robot's state-position. The position of the agent $l^* = (x^*, y^*)$ consists of the x-coordinate x^* and the y-coordinate y^*. It is assumed unknown but constant. x^* can take values $\{1, ..., d_x\}$, while y^* can take values $\{1, ..., d_y\}$. Given that the robot's state is unknown, we assume a random variable L that can take values $l = (x, y)$, where $x \in \{1, ..., d_x\}$, $y \in \{1, ..., d_y\}$. The probability of the agent being at l at time t is called the belief at l at time t:

$$Bel(L_t = l) = Pr(L_t = l). \tag{1}$$

The belief is actually a discrete probability distribution over $d_x \times d_y$ possible state tuples $l = (x, y)$.

The problem we aim to solve is that of state estimation for the agent from incoming measurements. We account for the uncertainty in the agent's state, using the belief formulation. The belief framework also accounts for the sensing and motion noise of the robot. We update the belief using incoming measurements in order for it to converge to a distribution concentrated around the agent's true state.

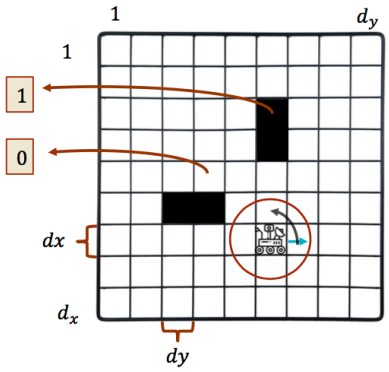


Figure 1: Example Environment for the Problem.

III. BASELINE

The baseline approach is based on [3] and [9] and is termed Markov localization. Markov localization is a method that globally estimates the state of a robot in the environment. It is based on a probabilistic framework, since it uses a belief to represent the confidence about the state of the agent. Markov localization is able to re-localize the robot in the case of localization failures, while also providing accurate position estimates [3]. The method discussed in [3] and [9] uses a fine-grained discretization of the state-space. This type of representation has several advantages over previous ones, such as using Gaussian or coarse-grained topological representations, because it provides more accurate estimates and can incorporate raw sensor inputs.

Section III.1, below, discusses the general framework of the method that is widely applicable. Further details of the baseline implementation are discussed in the Appendix in Section IX.2. This includes the necessary modeling such that the baseline approach is compatible with the AirSim collected LiDAR data in our project.

1. General Baseline Algorithm

For the Markov localization baseline, we assume a map, as shown in Figure 1. However, for this algorithm the state of the agent is a tuple $l = (x, y, \theta)$. $x \in \{1, ..., d_x\}$ and $y \in \{1, ..., d_y\}$ denote the x-coordinate and y-coordinate of the agent in the grid map. θ represents the orientation of the robot within the cell it occupies. We assume $\theta \in \{1, ..., 8\}$, given that there are 8 adjacent cells for a cell. The 8 values correspond to orientations towards the cells located to the N, NE, E, SE, S, SW, W and NW of the agent.

We also assume that we start with an initial belief $Bel(L_0)$ that is uniform over all possible states. The belief is updated for two reasons: either a LiDAR range measurement arrives or the robot moves within the environment. The updates at time t are as follows, assuming that we have the probability distribution $Bel(L_{t-1} = l)$ at time $t - 1$:

- **Perception Update**: If at time t a LiDAR range measurement r_t appears, then the belief at every state $l = (x, y, \theta)$ is updated as follows:

$$Bel(L_t = l) \propto p(r_t|l)Bel(L_{t-1} = l), \tag{2}$$

  where $p(r_t|l)$ is the environment perception model. The likelihood that we observe the range measurement r_t, given that we are in state l, can be assumed dependent only on the expected measurement from state l, $\hat{r}_l$, as in [9]. Therefore:

$$p(r_t|l) = p(r_t|\hat{r}_l). \tag{3}$$

- **Motion Update**: If at time t an odometry measurement a_t arrives, then the belief at every state l is updated as follows:

$$Bel(L_t = l) = \int p(l|a_t, l')Bel(L_{t-1} = l')dl', \tag{4}$$

  where $p(l|a_t, l')$ is the motion model and represents the probability of transitioning to state l, given that the robot is at state l' and we have measured odometry data a_t at time t. The motion update is based on the law of total probability.

IV. PROPOSED APPROACH

We consider the localization task as a multiclass classification problem, where each class is a cell in the belief map. The network inputs are the LiDAR data and the known map. Using this information, we would like to make a prediction as to which cell(s) the robot is most likely to be in.

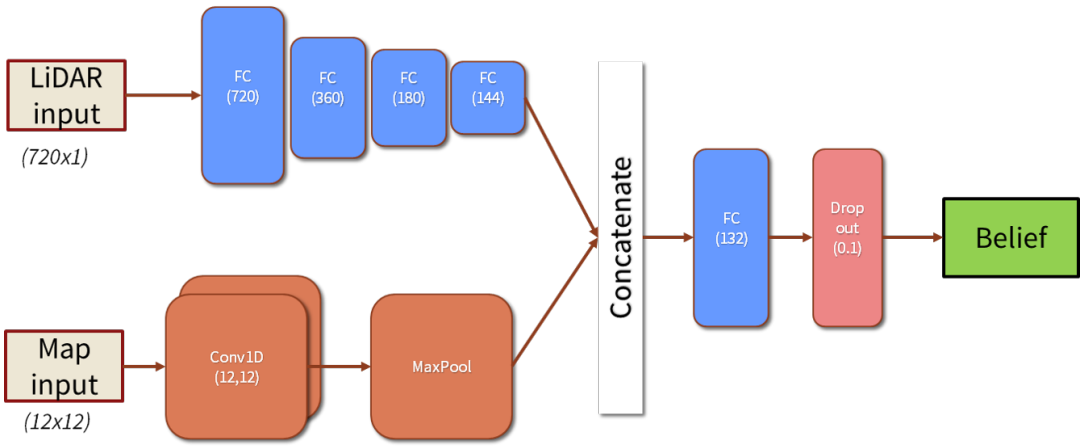


Figure 2: Proposed Neural Network architecture

1. Network Architecture

The overall architecture of the proposed model is shown in Figure 1. It consists of two main components which are the map feature extraction layers and the LiDAR processing layers. The outputs of these layers are then concatenated and propagated through more processing layers before being passed through a dropout layer for regularization. The softmax activation function is applied in the final layer of the network, resulting in a multi-class probability distribution over the map. We will discuss below some design decisions made along the way.

2. Loss Function

The model is trained on a Mean Squared Error (MSE) loss function which is defined as follows:

$$L(y_{\text{true}}, y_{\text{pred}}) = \sum_{i=1}^{\text{output size}} \left(y^i_{\text{pred}} - y^i_{\text{true}}\right)^2.$$

Another loss function considered was the Categorical Cross-Entropy loss (Softmax Loss) which is defined as follows:

$$L(y_{\text{true}}, y_{\text{pred}}) = \sum_{i=1}^{\text{output size}} y^i_{\text{pred}} \log y^i_{\text{true}}.$$

However, when trained for 1000 epochs, the MSE model outperformed the softmax model.

3. Data Processing and Representation

A characteristic of most neural networks is the requirement of fixed and defined input and output data sizes and types. The use of LiDAR measurements conflicts with this requirement due to the environment-and-sensor-dependent number of points detected by the sensor at any given time-step. We developed a data representation approach to avoid this problem by using fixed size vectors of size 720, which consists of the mean ranges of LiDAR points at fixed angle increments of 0.5^o. This is a transformation of the following form:

$$\mathbf{O_1} = \begin{bmatrix} x_1, y_1 \\ \vdots \\ x_n, y_n \end{bmatrix} \longrightarrow \mathbf{O'_1} = \begin{bmatrix} r_{\theta_1} \\ \vdots \\ r_{\theta_n} \end{bmatrix}$$

where r_θ is the mean range of points at orientation θ and x, y represent the x and y position of the points relative to the LiDAR laser scanner.

Angle orientations at which there are no points in the measurement, due to the distance being greater than the maximum measurable LiDAR distance, are represented by a value of -1. An alternative approach is to represent such ranges with the maximum range of the LiDAR sensor. The practical meaning of these two representations differs slightly, however; with the latter suggesting that there are points at LiDAR maximum range distance. The -1 representation was chosen to encode the idea of "no points" as opposed to using the maximum range for this purpose.

The map representation, as discussed in Section II, is that of an occupancy grid. This results in a well defined data structure of fixed size $\mathbb{R}^{(d_x, d_y)}$ with values constrained between 0 and 1. This is suitable for use in a neural network architecture. It is also appropriate to consider the map as a single-channel heatmap image. True position data is represented in a similar way and is similarly bounded in the range $[0, 1]$, as a belief. However, the 2D representation is not preserved and the data structure is instead flattened to a 1D array of size $\mathbb{R}^{d_x \times d_y, 1}$. This is for ease in computing the loss between the network output and the true position.

4. Layer Width

The width of the input and output layers are required to match the size of the input data and training position data. The hidden layers, however, are not so well defined. The layer widths used in the network are shown in Figure 2. The general idea here is to try and extract features from the input data and ultimately discard extra information, resulting in diminishing layer widths as the depth of the network increases. The map feature extraction follows the same idea with the use of a max pool layer to shrink the size of the input volume and help extract relevant features from the map.

5. Layer Types

Considering the map as a single channel image is what drove the decision behind using convolutional layers as part of the map feature extraction component. Initially, convolutional layers were used for the LiDAR processing as well (with the 720×1 array reshaped into a 30×24 array), however, this resulted in poor performance. This is likely related to the LiDAR input data being much less well-defined than the map input and the convolutional layers working better with very well-defined and bounded inputs. The use of negative numbers (-1) in the LiDAR data was also not helpful. The use of convolutional processing layers with the alternative LiDAR representation (for distances greater than the threshold use the threshold), normalized by the maximum range (thereby bounding all values between 0 and 1) might yield better results although preliminary testing did not confirm this. Ultimately, dense fully connected layers were used for the LiDAR processing. A dropout layer was used for regularization and to stop the model from over-fitting and becoming overconfident in its belief predictions, although the dropout rate of 0.1 was arbitrarily chosen.

6. Activation Functions

One approach when training neural networks is to use ReLU activation functions first as a default and make changes if necessary. This is what we did for this project. We trained one model using ReLU activation functions (ReLU model) and another using $tanh$ activation functions ($tanh$ model). Although the ReLU model looked promising with shallow training of 1000 epochs, when fully trained, we observed that the model became "washed out" and made the same predictions irrespective of the input data. One possible cause of this is the vanishing gradient problem. As Figure 3 shows, the ReLU activation function is constant for all negative inputs and therefore results in 0 gradient for such inputs. This could cause or exacerbate the vanishing gradient problem and result in the homogeneous property exhibited by the ReLU model. The $tanh$ function, however, preserves the gradients of negative values. It also performed better upon testing. The leaky ReLU might also provide an improvement on the ReLU function for hidden layers. For the output layer, a softmax activation function was used which is the multi-class extension of the sigmoid function, also shown in Figure 3.

7. Number of Epochs

Figure 4 shows the training loss for models trained for 10000 epochs and 5000 epochs. Both models exhibit the trend of decreasing and then increasing loss suggesting that the model begins to overfit to the data. The network would benefit from early stopping, and the optimal number of epochs looks to be around 2000.

V. DATA COLLECTION

Data for training and for the simulations were collected using the AirSim simulation environment. Within AirSim, we create the grid map for our problem and collect the simulated LiDAR measurements based on the robot's position. Data was collected by initializing a vehicle with an on-board laser scanner to a random unoccupied position in the map with a random orientation

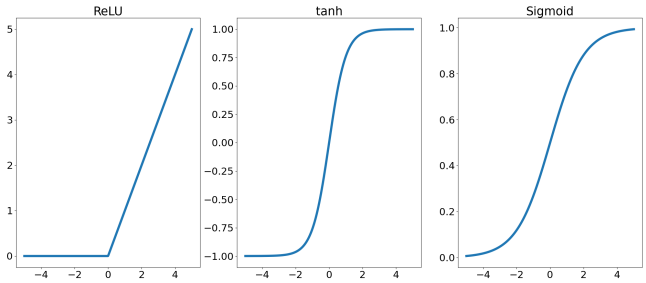
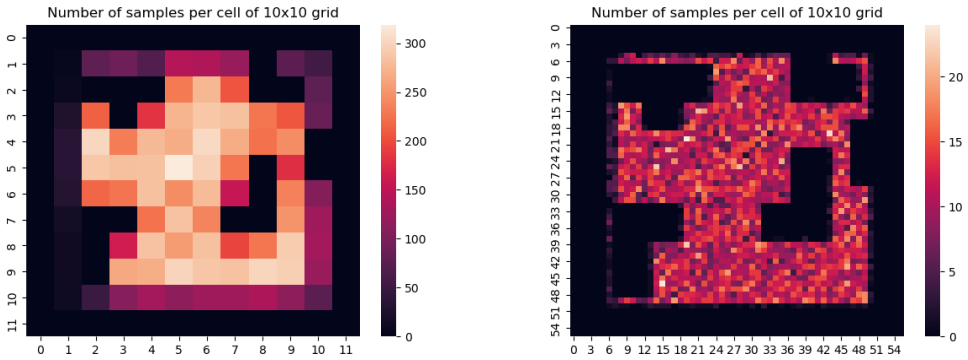


Figure 3: Activation Functions

Figure 4: Training loss for different number of epochs



(a) 10x10 map **(b)** 50x50 map

Figure 5: Distribution over the map for data collection

in the range $[-\pi, \pi]$. Once the vehicle spawned in that location, the position and orientation of the vehicle as well as a LiDAR pointcloud measurement was captured and saved. In the event that the laser scanner was unable to detect any nearby points, an empty list was stored. This was repeated $15,300$ times. The heatmaps shown in Figure 5 show the distribution of locations around a 10×10 and 50×50 map from which data was collected.

VI. SIMULATION RESULTS

In this section, we compare our proposed neural network approach with the Markov localization baseline. We create a custom environment in AirSim that consists of a 10×10 grid map, with obstacles, as shown in Figure 6. The designed grid map contains areas that are feature rich (close to corners) and feature deficit (in open space). The cell dimensions are $10m \times 10m$ and the maximum LiDAR detected distance is $10m$.

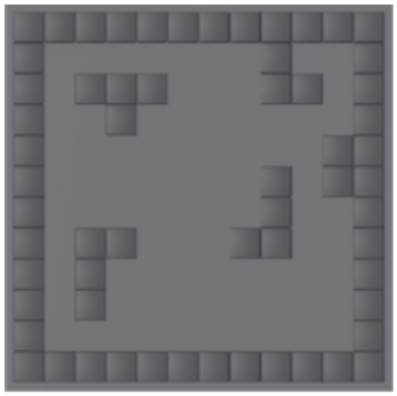


Figure 6: Simulation Environment Map.

In order to analyze the performance of our scheme, we will use two distinct collections of results on the same environment. Firstly, we will show the output belief of the neural network and Markov localization methods for different robot positions in the map. Secondly, we will compare the two methods in terms of error from true position for a given trajectory. In this case, for each position of the robot in a given trajectory, the algorithms calculate the belief without taking into consideration the history of the robot's motion. As both algorithms only consider the LiDAR measurements at that time step, we refer to them as one-shot.

1. Output Belief for Different Robot Positions

In this section, we demonstrate the algorithms' belief output for different robot positions in the map. Four situations have been chosen that correspond to positions with varying localization difficulty. Localization difficulty depends on the information the neighborhood of the true position contains (how feature rich the robot's position is). This collection of results corresponds to a qualitative analysis of the performance of the algorithms.

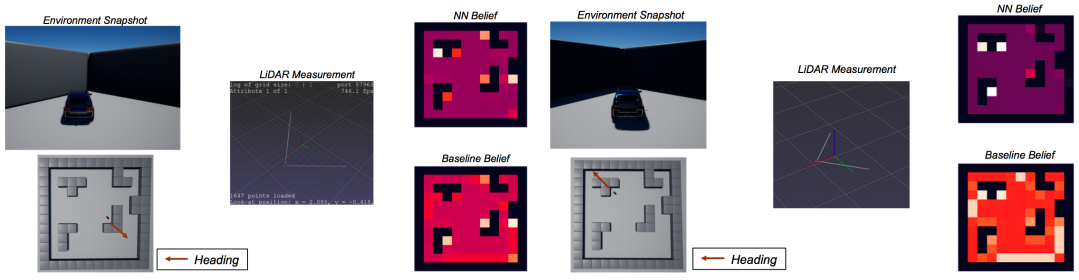


Figure 7: Belief Outputs for Different Robot Positions.

In the scenario presented in Figure 7a, the vehicle is placed in a corner. Its LiDAR sensor detects the corner, but due to large distances, does not detect anything in the rear side of the vehicle. In general, corners are feature rich areas, because they have walls in two sides. This information is useful for the localization algorithms, which are actually able to detect that the agent is in a corner, as shown in the belief outputs of Figure 7a. In the belief outputs, a lighter color indicates a higher belief-probability

for that cell. In this case, we observe that the baseline assigns a higher probability to the true position than the proposed neural network, which nonetheless includes the true position in the probable states. In addition, both approaches assign high probabilities to corners, showing that they understand the features present in the LiDAR measurement.

Figure 7b shows the results for another corner. In this case, the baseline belief contains more uncertainty, although it includes the true position in the most probable states. The neural network is able to provide a far more specific solution. The varying performance of the baseline between the 2 corner cases examined up until now can be attributed to the fact that in the AirSim environment the agent may be anywhere within each cell, while the baseline algorithm assumes the agent to be in the middle of a cell always. The neural network is able to better adapt to this uncertainty. In addition, in this case, we observe that it is certain that the true position is at a corner, indicating learning of useful features.

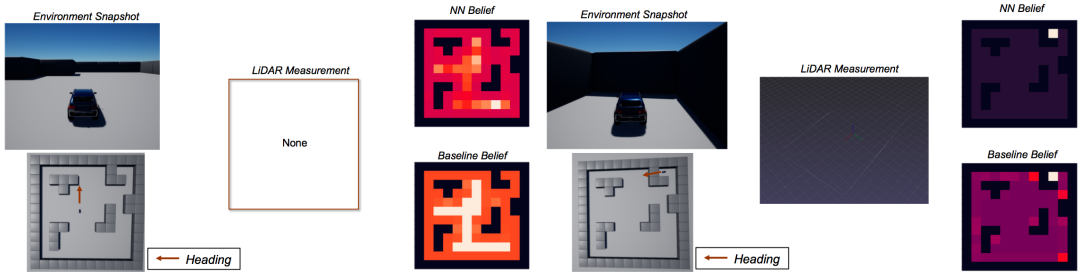


Figure 8: Belief Outputs for Feature-Sparse and Feature-Rich Robot Positions.

In Figure 8a, we have an example position that is far away from any structure. This results in all the distances being greater than the maximum LiDAR threshold. Therefore, the LiDAR scan vector only contains values -1. The baseline is able to represent this case extremely well, assigning an equal and high probability to all cells with a distance greater than the LiDAR threshold to any surface. Our neural network follows the same trend, but does not assign equal probabilities.

In Figure 8b, we place the vehicle in the most feature rich cell of the map; the corner which is closed in 3 out of its 4 sides. In this case, our proposed approach is able to output a correct and accurate solution, while the baseline also attributes the highest probability to the true position, assigning nonzero probability to some other corner cells too however. This can be attributed the fact that in the AirSim environment the agent may be anywhere within each cell, while the baseline algorithm assumes the agent to be in the middle of a cell always.

The reasonable belief output of our neural network in all 4 cases indicates that it is in fact learning features from the data.

2. Error Analysis for Specific Vehicle Trajectory

In this section, we will test the performance of both our proposed approach and of the Markov localization baseline for a vehicle that follows a specific trajectory within the map. In each time-step, the agent uses the current LiDAR scan vector and calculates a localization solution that is independent of the localization solution at all other time-steps. Although available information arising from the time-dependency of the localization solutions at successive time-steps is ignored, this cannot be avoided due to the one-shot structure of the algorithms.

At each time-step we calculate the output belief of the two methods and, using the probabilities indicated by these beliefs, we calculate the expected position of the robot for each of the two methods:

$$(\bar{x}, \bar{y}) = \sum_{x \in \{1,\ \ldots,\ d_x\}, y \in \{1,\ \ldots,\ d_y\}} (x, y) Bel\left(l = (x, y)\right) \tag{5}$$

and find the cell it belongs to, since our algorithms only have cell-level accuracy.

We then calculate the number of cells between this expected position and the true position. Given that the agent in the AirSim environment can be anywhere within a cell, but the algorithms can only localize with a cell-accuracy, a distance metric is not adequate. Therefore, we use the number of cells distance, which captures the number of cells difference between the expected position and the true position. This metric is just the L2 distance between the indices of the cells containing the true and expected position. For example, in the case where the true position is within cell (x, y) and the expected is within $(x, y + 1)$, the cell distance is 1.

For a specific trajectory (as shown in slide 23 found here), the results are included in Figure 9. The mean error is shown in

Table 1. Overall, the two methods perform equally well. The baseline achieves a localization solution with smaller variance and smaller mean error, mostly because it maintains a level of uncertainty in the localization solution in cases it is confident, in contrast to the neural network approach that does not follow this logic, as shown in Figure 8b. The error trend is however the same for the two methods, owing to the use of the expected position of the belief output. In addition to the comparable accuracy, the neural network approach is much faster, as shown by the larger number of points in the neural network error curve. This allows for more localization solutions compared to the baseline approach in the same time interval.

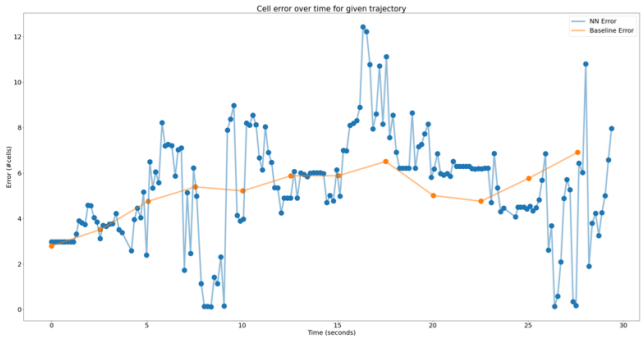


Figure 9: Error for Simulated Trajectory.

Table 1: Error Comparison of Methods

Metric	Baseline Approach	Proposed Approach
Mean Error (number of cells over time)	5.20	5.37

VII. CONCLUSION

In this paper, we propose a simple learning-based method for global localization. Our implementation is based on a simple neural network structure that takes as inputs the LiDAR measurement and the environment grid map and outputs a belief over that map. Our proposed approach is able to circumvent the LiDAR-Image approach and associated computation overhead. It learns useful features and patterns from the data, performing comparably well with respect to error as the baseline Markov localization approach. It even outperforms the baseline approach with respect to speed, allowing for more belief updates in the same time interval, providing a highly certain and correct estimate, in contrast to the baseline, in several scenarios. Overall, the results are encouraging and indicate the potential of the belief-based neural network.

VIII. FUTURE DIRECTIONS

There are various extensions to the work presented in this paper. One possible route is to add sensor modalities, such as camera images, and improve the network's architecture. In addition, we can extend the framework to also decide on future actions that increase localization accuracy, transitioning from passive to active localization. Finally, it would be interesting to extend the framework for the collaborative multi-robot active global localization scenario, where an agent's belief is influenced by other agents' shared information.

IX. CONTRIBUTIONS

Timmy set up the AirSim simulation environment and collected the LiDAR measurements. He pre-processed the data and implemented the neural network for the proposed approach. Alex implemented the baseline approach, tested it in simple cases using the AirSim map and ensured it was compatible with the collected AirSim data. Both Timmy and Alex reviewed the literature and contributed in brainstorming sessions throughout the completion of the project. Finally, both Timmy and Alex worked on preparing the required presentations and reports for the project.

APPENDIX

1. Code

The code for the project can be found at

- `https://github.com/alextzik/multi-robot_active_localization`
- `https://github.com/timmyhussain/NN-Perception-Model`

2. Baseline Algorithm Compatible with AirSim LiDAR Data

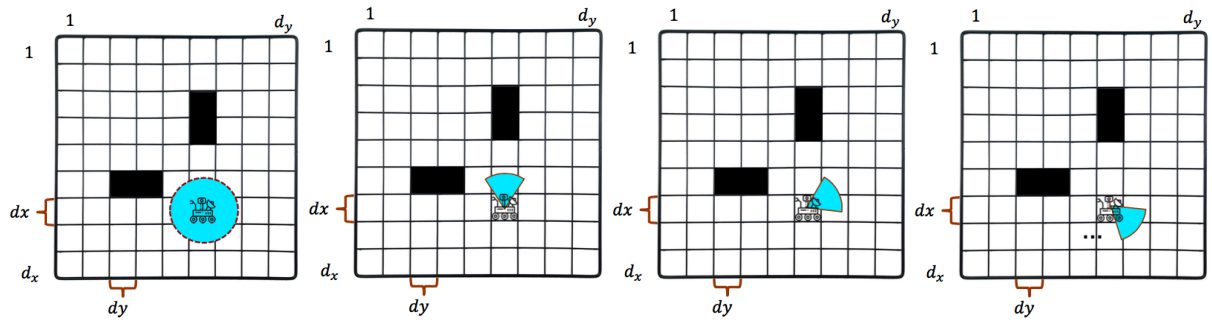
There is an important observation to make regarding the data format of the LiDAR measurement extracted from the AirSim environment: the data is a collection of LiDAR range measurements that the robot gathers from all possible directions (360^o scan) around its position. In other words, we are not given a single range measurement, but a collection of range measurements whose directions of arrival differ. This is in contrast to the Markov localization framework discussed before, where we only have an update (the perception update) for a single range measurement.

In order for Markov Localization to be compatible with our data format, we follow the steps below:

1. We collect the 360^o LiDAR scan vector at time t, as shown in Figure 10a. This will be used to update the belief.
2. We divide the points in the LiDAR scan vector into 8 sections (to be compatible with our framework that has 8 possible orientations θ). We assume that the range points of each section come from the same orientation-direction of arrival (in our framework there are 8 possibilities).
3. We start with one section arbitrarily, as in Figure 10b:
   - We update the belief by performing a perception update for each LiDAR range point in this section at every state l. For our discrete world, we assume that $p(r_t|\hat{r}_l)$ is a binned Gaussian distribution. The standard deviation is left as a hyper-parameter and depends on the accuracy of the sensor model and the environment model. In addition, if $\hat{r}_l$ is greater than the maximum measurable distance of the LiDAR sensor, we assign $\hat{r}_l = -1$, because AirSim assigns a value of -1 to a distance measurement greater than this threshold.
   - Before moving on clockwise to the next section of range points, we must account for the change of direction of arrival of the range points of the next section, compared to the current section. To account for the change in incoming direction, we perform a motion update. This is necessary, because currently we have a belief for all tuples $l = (x, y, \theta)$. However, in order for the robot to sense the new section it must rotate clockwise by one orientation value. Therefore, our current belief for (any) orientation θ, must become our belief for orientation $(\theta + 1)\ mod\ 8$, where $(\theta + 1)\ mod\ 8$ denotes the next (after θ) orientation clockwise. We must perform this translation for all orientations in the current belief, as shown in Figure 11. Mathematically, this is denoted as:

$$Pr(l|a_t, l') = \delta(x' - x)\delta(y' - y)\delta(\theta' - (\theta - 1)\ mod\ 8), \tag{6}$$

     where $\delta(\cdot)$ denotes the Dirac delta function.
4. We move to the next LiDAR vector section clockwise and repeat the process, as shown in Figure 10c:
   - We perform perception updates using the section's range points.
   - We perform a motion update to account for the change in direction of the upcoming section.
5. We stop after all sections are accounted for, as shown in Figure 10d.



(a) Incoming LiDAR Measurement at time t. **(b)** Belief Update using First Section at time t. **(c)** Belief Update using Second Section at time t. **(d)** Belief Update Next Steps at time t.

Figure 10: Belief Update using AirSim LiDAR Measurements.

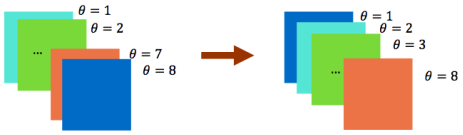


Figure 11: Motion Update in the Belief Update.

The major limitation of the baseline is the need for a perception and a motion model. Inaccuracies in the models may seriously degrade performance. Specifically, in our implementation the standard deviation for the binned Gaussian in the perception model greatly affects performance and a suitable value must be used. In addition, the baseline approach cannot take into account the 360^o LiDAR scan at once, but requires a more complex methodology, as discussed previously.

REFERENCES

[1] Everett, M., Chen, Y. F., and How, J. P., "Collision avoidance in pedestrian-rich environments with deep reinforcement learning," *IEEE Access*, vol. 9, pp. 10 357–10 377, 2021.

[2] Cox, I. J. and Wilfong, G. T., Eds., *Autonomous Robot Vehicles*. Berlin, Heidelberg: Springer-Verlag, 1990.

[3] Fox, D., Burgard, W., and Thrun, S., "Markov Localization for Mobile Robots in Dynamic Environments," *Journal of Artificial Intelligence Research*, vol. 11, no. 1, pp. 391–427, 1999.

[4] Chaplot, D. S., Parisotto, E., and Salakhutdinov, R., "Active neural localization," *arXiv preprint arXiv:1801.08214*, 2018.

[5] Lin, J., Yang, X., Zheng, P., and Cheng, H., "End-to-end Decentralized Multi-robot Navigation in Unknown Complex Environments via Deep Reinforcement Learning," *Proceedings of 2019 IEEE International Conference on Mechatronics and Automation, ICMA 2019*, pp. 2493–2500, 2019.

[6] Chaplot, D. S., Gandhi, D., Gupta, S., Gupta, A., and Salakhutdinov, R., "Learning to explore using active neural slam," *arXiv preprint arXiv:2004.05155*, 2020.

[7] Fan, T., Long, P., Liu, W., and Pan, J., "Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 856–892, 2020.

[8] Barsan, I. A., Wang, S., Pokrovsky, A., and Urtasun, R., "Learning to localize using a lidar intensity map," *arXiv preprint arXiv:2012.10902*, 2020.

[9] Fox, D., Burgard, W., Kruppa, H., and Thrun, S., "A probabilistic approach to collaborative multi-robot localization," *Autonomous Robots*, vol. 8, no. 3, pp. 325–344, 2000.