Particle Filter for Skid-Steered Robot

Rodolfo Corona[1] Chenyun Wu[2] Samer Nashed[3] Joydeep Biswas[4]

Abstract

The ability to determine the location of a robot in its environment is a key requirement of autonomous robotics tasks. The particle filter algorithm, which may be used with various different combinations of sensors, is a common approach to this problem. Through the development of effective motion models for a skid-steered, outdoor robot, we have implemented a particle filter which may be used to test the effectiveness of a variety of sensors. Having tested the algorithm with a combination of odometry and GPS, we will explore methods from computer vision, such as convolutional neural networks (CNNs), for effectively incorporating video cameras as sensors. To this end, we have collected a data set of images mapped to locations which was used to run preliminary experiments of our methods.

I. INTRODUCTION

For an autonomous robot to operate effectively in its environment, it is necessary that it be able to track its location over time. More formally, **localization** may be defined as the task of determining a robot's **pose**, which is of the form $(x\ y\ \theta)^T$ and denotes a robots location and orientation within a given map M. Although simple to define, this task is made difficult by the inherently noisy nature of sensors. Even if a particular sensor's error is small, the dependence of each reading on the previous ones can quickly give rise to large systematic errors.

In order to combat this problem, localization algorithms often require and are greatly aided by the use of multiple sensors. By getting location estimates from a variety of sources, a tighter probability distribution for a robot's pose may be computed, allowing the error and uncertainty in pose estimates to be mitigated.

In this paper we detail the implementation of a **particle filter**, a popular localization algorithm, for use with the Jackal, a skid-steered, outdoor robot developed by ClearPath Robotics[1]. Furthermore, we derive a motion model for the robot and employ the use of GPS sensors as well as its internal odometry for our implementation.

This work also explores how cameras may be used as sensors for localization. In order to map images to pose estimates, we use convolutional neural networks (CNNs) to extract high-level feature vectors from images, which are then experimented with using a variety of different classification algorithms.

II. BACKGROUND

A. Localization

The task of any localization algorithm is to determine a likely estimate for a robot's current pose $(x\ y\ \theta)^T$. In order to accomplish this, the algorithm must work effectively with readings from its ensemble of sensors [13].

[1]Rodolfo Corona is an undergraduate student at UT Austin who worked on this project as an REU student at UMass Amherst, he is responsible for the localization components of this project.

[2]Chenyun Wu is a graduate student at UMass Amherst working under Dr. Subhransu Maji and conducted the vision related work for this project.

[3]Samer Nashed is a graduate student at UMass Amherst working under Dr. Joydeep Biswas and served as Rodolfo Corona's graduate mentor for the project.

[4]Joydeep Biswas is an Assistant Professor in the College of Information and Computer Sciences at UMass Amherst and served as Rodolfo Corona's faculty advisor for this project.

*This work was supported in part by the NSF.

[1]https://www.clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/

Fig. 1. The ClearPath Jackal equipped with two fish-eye camera lenses.

At each time step t, a robot receives a **control** u_t, and a **sensor reading** z_t. The control is an estimate of the robot's translation and rotation since the last time step as given by its internal sensors, such as odometry and IMU (inertial measurement unit), and is of the form $(\Delta x\ \Delta y\ \Delta\theta)^T$. Similarly, the sensor readings come from sensors external to the robot, such as GPS or laser range finders, and give a pose estimate of the form $(x_z\ y_z\ \theta_z)^T$.

Using these readings, the robot's most likely pose at time step t, $\mathbf{x}_t = (x_t\ y_t\ \theta_t)^T$, may be found using the following formula:

$$\mathbf{x}_t = argmax_{\mathbf{x}}\ P(\mathbf{x}|\mathbf{x}_{0:t-1}, u_{0:t}, z_{0:t})$$

The fact that the space of possible poses is continuous, however, makes the task of exhaustively searching through it intractable. Further, the cost of computing such a function may grow quickly over time because of the dependence of the estimate on all previous estimates and readings.

In order to combat the growth of the cost, the **Markov assumption** may be used, allowing one to compute an estimate using only the current time step's readings and the previous time step's pose estimate by assuming independence from all prior data points:

$$\mathbf{x}_t = argmax_{\mathbf{x}}\ P(\mathbf{x}|\mathbf{x}_{t-1}, u_t, z_t)$$

For working tractably in a continuous space, localization algorithms use a variety of methods to approximate this function. The most common strategy is to assume that the control and sensor readings are independent from each other and split the function into two statistical models, the **motion model** and the **perceptual model**. The motion model approximates the probability distribution $P(\mathbf{x}_t|\mathbf{x}_{t-1}, u_t)$ using the control reading and noise generated from experimentally derived distributions, such as Gaussians. The perceptual model approximates the probability distribution $P(\mathbf{x}_t|z_t)$ using experimentally derived distributions to model sensor uncertainty. In general, the motion model is used to estimate the trajectory of the robot over time while the perceptual model is used to bias the trajectory's uncertainty. More concretely, the dependence of each motion model estimate on the previous time step's estimate makes the motion model highly prone to systematic errors. The perceptual model is therefore used to bias this trajectory estimate and keep the error in check.

B. Particle Filters

The approach of the particle filter algorithm to the problem of tractability is to use a discrete set of N random variables known as **particles** [12]. At each time step, particles are assigned different pose estimates based on the robot's statistical models. This distribution of particles is then used in order to approximate the true probability distribution of the robot's pose.

To accomplish this, the algorithm iterates through three main procedures at each time step. Firstly, each particle's pose estimate is computed using its previous pose and the current control readings in the *elapse time* step. Following this, the *weighing* step uses the current sensor reading in order to assign weights to each particle's estimate. Finally, in the *resampling* step, a probability distribution of possible poses is computed based on the particle poses and weights, which is then used to re-sample the particles' poses in order to bias the distribution towards more likely estimates.

1) Elapse Time: At each time step t the robot's motion model is used in conjunction with the current control reading u_t in order to derive a new pose for each particle:

$$\mathbf{x}_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + x_u + \varepsilon_x \\ y_{t-1} + y_u + \varepsilon_y \\ \theta_{t-1} + \theta_u + \varepsilon_\theta \end{bmatrix}$$

Where ε represents an error reading sampled from an experimentally approximated distribution for each of the three dimensions.

2) Weigh: Immediately after receiving a pose estimate, each particle is assigned a weight based on the current sensor model reading z_t and the filter's perceptual model. This is done by using a probability distribution which is derived by taking into account the level of uncertainty of the sensor.

3) Resample: Using each of the N particles' weights, a discrete probability density function of pose estimates is built such that the probability of any particle's pose estimate is proportional to its weight:

$$\forall_{1 \le i \le N} \; P(\mathbf{x}^i) = \frac{w_i}{\sum_{i=1}^{N} w_i}$$

Where $\mathbf{x}^i$ and w_i denote the i-th particle's pose estimate and weight. Each particle's pose estimate is then re-sampled using this distribution, which results in the particle distribution being concentrated around the most likely robot locations.

C. Convolutional Neural Networks

A neural network (NN) is a model used for machine learning which represents a function $f : X \to Y$ mapping data points to labels. Generally, these models are used in order to classify data in tasks such as object recognition in images.

In order to do this, a network takes a vector or tensor representation of data $\mathbf{x}$ as input (such as a $W \times H \times 3$ matrix for an image of dimensions $W \times H$ and the 3 RGB color channels) and performs a set of operations on it to generate its output. Each **layer** of a network consists of a set of **neurons**, with neurons at different layers having input/output connections to each other. Every neuron is assigned a weight vector, the weights of which must be learned using machine learning techniques, with each weight corresponding to each of its inputs from the previous layer.

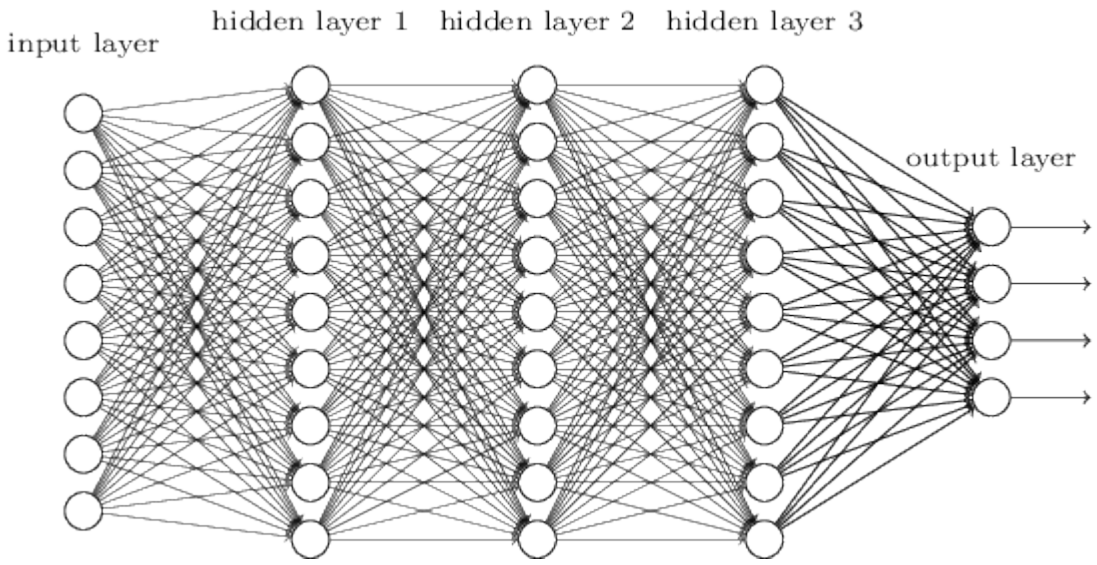

Fig. 2. A fully connected, feed-forward neural network. Neurons are connected to each neuron in the previous and next layers. Image source: http://neuralnetworksanddeeplearning.com/chap6.html

At each layer, a neuron will compute a dot product between its input vector and its weight vector, the result of which is then fed to the neurons at the next layer to which it is connected to. In more concrete terms, the model performs a matrix multiplication at each layer in order to derive its output. This basic neural network model is known as a **feed forward** architecture and is also generally implemented as a **fully connected** model in which the neurons in each layer will be connected to all of the neurons in the next.

It may be observed that as the size of the input (e.g. images) grows, the number of weights that must be learned in a network can quickly become intractable. Due to the growth in the number of learned parameters, the amount of training data required may also become infeasible. Furthermore, in many cases, including images, much of the inputted information is redundant and need not all be processed for effective classification. Convolutional neural networks alleviate this problem by employing the use of **convolutional** and **pooling** layers in their architecture.

Instead of taking as input the entirety of an image, neurons in a convolutional layer will instead receive small patches, such as 3×3 pixel patches, and convolve the values in this patch using what is known as a **kernel function**. A kernel function used within a neuron is also known as a **filter** because each kernel function is constructed to detect the presence of different types of

features in its input, such as edges, corners, or colors. In other words, neurons in a convolutional layer will output higher values, or become ”activated”, when they detect the presence of certain features.

Neurons in a pooling layer will take input from neurons in convolutional layers and perform some form of downsampling on their input, which reduces the dimensionality of its output and results in a lower number of weights going forward to deeper layers. Similarly to the convolutional layers, neurons in this layer will be activated by the presence of certain features in the input, but are invariant to the ordering of the input with respects to factors such as orientation and translation. More precisely, whereas convolutional layer neurons seeks features in specific patches of the input, pooling layer neurons simply seek the presence of the features in the set of patches. This dynamic allows CNNs to detect objects in images independently of where they are located within them.

Finally, the fully connected layers of a CNN work identically to those of a fully connected network, and are generally used as the last layers in order to compute a final classification for the given input.

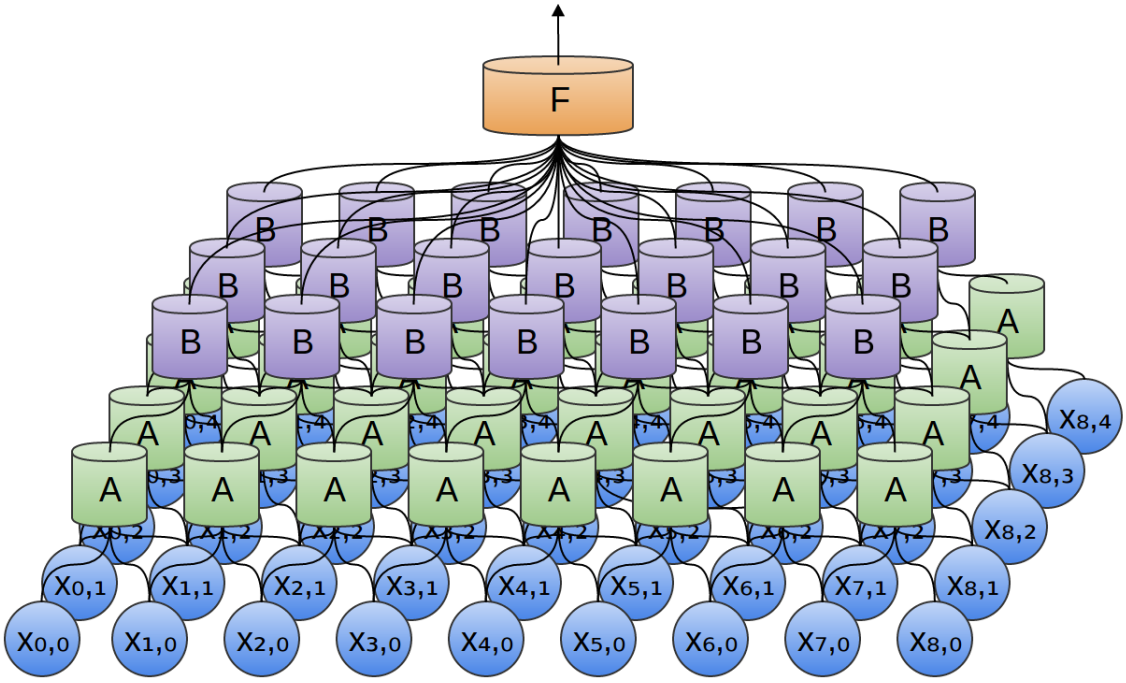

Fig. 3. A convolutional neural network. The number of learned parameters is greatly reduced compared to fully connected architectures through the use of convolutional and pooling layers. Image source: http://colah.github.io/posts/2014-07-Conv-Nets-Modular/

At a high level, the network may be thought of as finding certain features in specific patches in the convolutional layers. The pooling layers may be thought of as seeking out only the presence of these features among groups of patches, indifferent to which patch. As the input travels through the network, the types of features which are computed also become more abstract, taking the form of fuller patterns or even representing partial objects themselves.

Although a lower level description of CNNs is beyond the scope of this work, a more in depth treatment of the subject matter may be found in [4], [9].

III. Related Work

There has been much work done on indoor robot localization, in which the environments remain mostly static and do not vary in factors such as weather, lighting, and obstacle locations. The presence of walls also makes it much easier to use sensors such as laser-range finders [11].

Outdoor localization, however, is complicated by many factors which can vary greatly based on time of day and season, amongst other things. Irie et al. propose a solution to this problem which uses stereo cameras in order to generate 2D point clouds of the environment [7]. This method is used during data collection to create a global grid map of the robot's environment. During localization, point clouds are generated at each time step and are used in two ways. Firstly, the motion of the robot is estimated using the changes in point clouds, something which serves as the motion model for their particle filter implementation. As a perceptual model, they compare the local point cloud to the global map that was generated during data collection in order to form a probability distribution for particle weighing. They claim that their methods are more resilient to changes in lighting than other approaches.

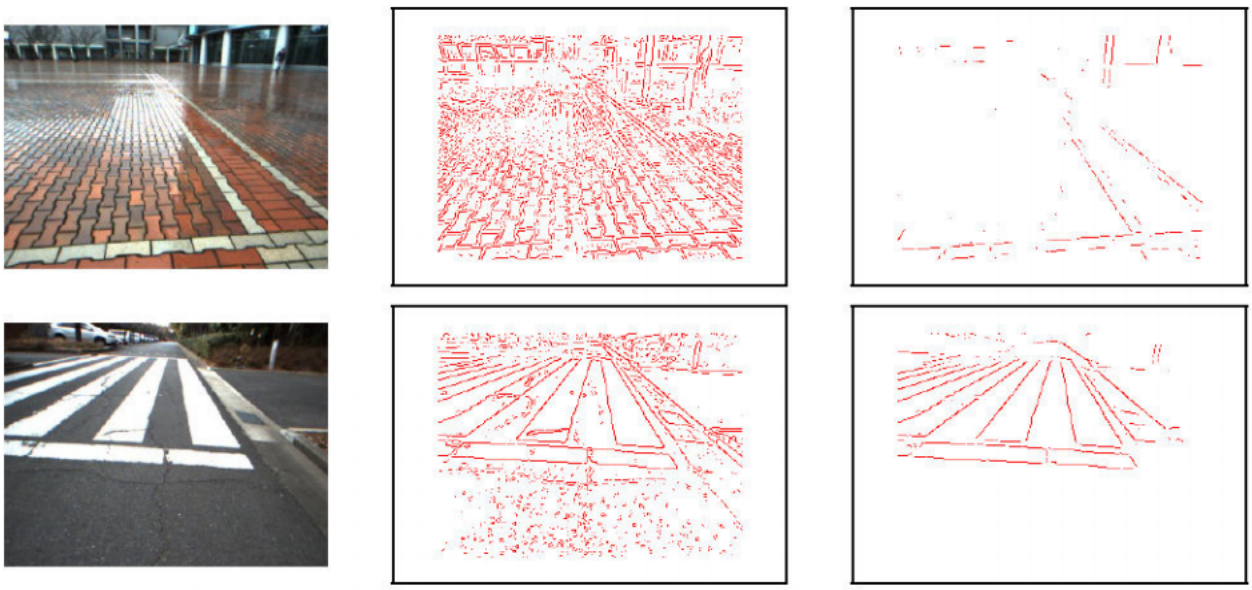

Fig. 4. Point clouds generated by Irie et al. on their data set for localization using stereo cameras. Image source: *http://www.furo.org/irie/stereo_localization_ros10.pdf*

Similarly to Irie et al., Agrawal et al. use the Harris corner detection method [6] along with stereo cameras in order to calculate movement between time steps to serve as a motion model [1]. Instead of using vision as a perceptual model, however, they use GPS sensors. Additionally, they implement a Kalman Filter for localization [8].

Valgren et al. instead propose to use SIFT and SURF features for image matching [14]. In their work, they generate SIFT [10] and SURF [3] feature vectors for the images collected by their robot and use nearest neighbor approaches to match them to images in their training data.

In our work, we propose to use neural networks to serve the role of a perceptual model. With enough training data, we believe that neural networks would be able to learn the invariant features of a map, which would allow them to be robust to different lighting and weather conditions like point cloud methods as well as robust to variance in the set of objects in the environment like SIFT and SURF feature vectors. Furthermore, CNNs do not require one to perform any feature

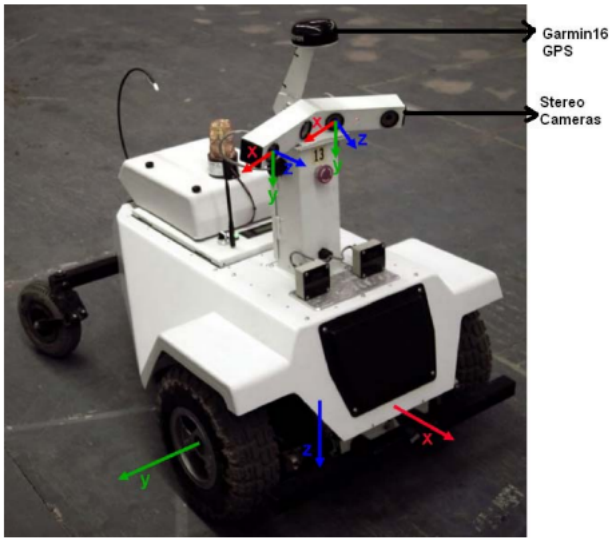

Fig. 5. A picture of the robot used in the work of Agrawal et al., which employs stereo cameras with GPS for localization. Image source: http://www.ai.sri.com/ agrawal/icpr06.pdf

engineering on the data, which not only cuts down development time, but may also make our methods robust to different environments which may have very different types of differentiating features.

IV. Methodology

A. Data Set

A roughly 280 meter path was chosen for use as a map for testing our methods. This path was traversed a total of 19 times throughout one week at different times of day in order to capture as much variability in daylight and weather as possible. Through each run, data from all robot sensors was collected (e.g. GPS, cameras, wheel odometry, etc.), with data log files averaging roughly 2GB in size.

B. Particle Filter Algorithm and Visualizer

For our particle filter implementation, internal wheel odometry sensors were used to approximate trajectory in our motion model. Additionally, a GPS perceptual model was integrated in order to test the efficacy of the algorithm while the visual perceptual model was being developed.

In order to effectively use GPS for localization, the latitudinal and longitudinal reading variances, σ^2_{lon} and σ^2_{lat}, for the Jackal's GPS were approximated experimentally by collecting readings from a set of three marked places five different times. Using these values, the following loss function was constructed:

$$f(x_e, y_e, x_g, y_g) = \frac{1}{\sigma^2_{long} + e^{|x_e - x_g|}} \cdot \frac{1}{\sigma^2_{lat} + e^{|y_e - y_g|}}$$

Where x_e and y_e are position estimates and g_x and g_y are GPS position readings. This function was then used in order to assign a weight to each particle in the weighing step of the algorithm, where a greater distance from the GPS reading results in a lower weight being assigned.

For generating a pose estimate at each time step, the mean pose was taken from the set of particles in the filter.

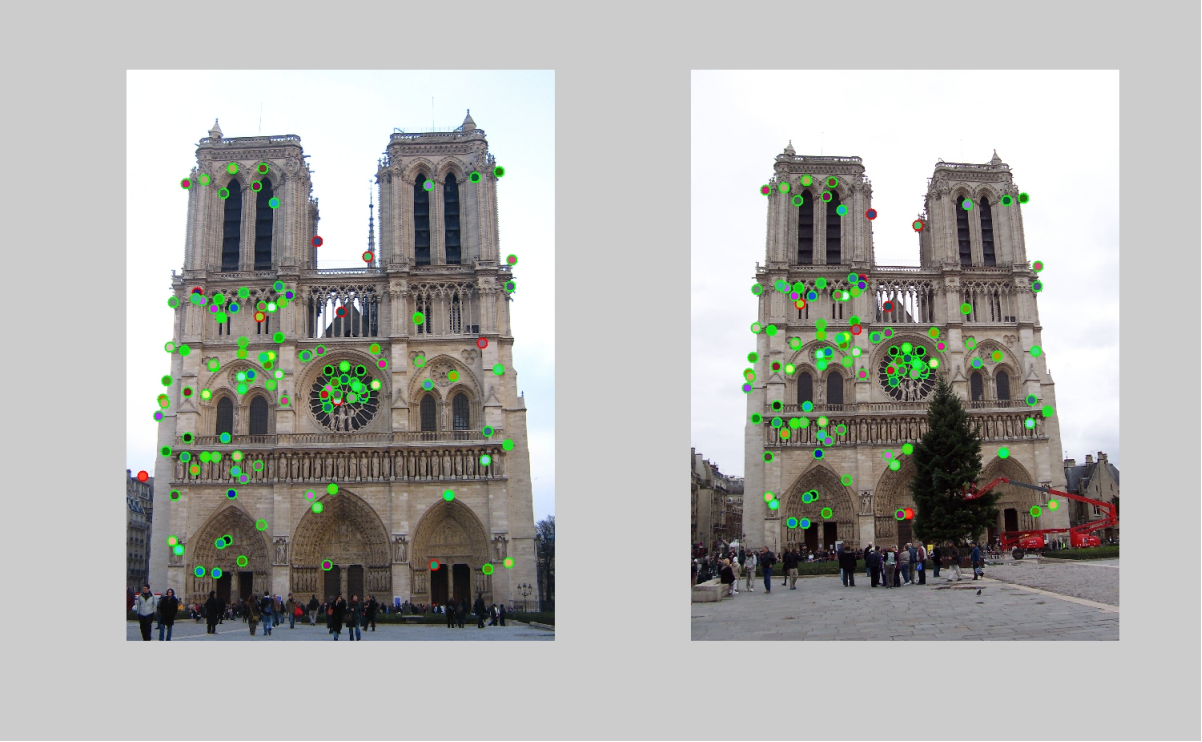

Fig. 6. An example of SIFT features generated for an image. Both SIFT and SURF generate feature vectors for images by finding key points such as edges, corners, or other areas where certain image gradients are higher. Image source: http://cs.brown.edu/courses/cs143/results/proj2/valayshah/

In order to be able to qualitatively evaluate the particle filter implementation, the CMU Cobot [15] localization visualizer was integrated into the system. Using the Cobot visualization library, the particle filter visualizer may take a log of saved sensor data from our data set and recreate it visually, effectively allowing the user to replay a run along the map.

C. Neural Perceptual Model

In order to effectively use cameras as sensors, a method that can work directly with raw RGB information with as little preprocessing as possible is desirable. CNNs are a well known tool that has been previously used with raw image data for a variety of machine learning tasks, particularly in object recognition.

Through a series of convolution operations on the image data, CNNs are able to extract higher level features from images such as edges and partial objects, features which we believe will prove effective in differentiating pictures taken at different locations from each other.

To this end, a pre-trained model of the AlexNet CNN architecture [9], which won the 2012 ImageNet Large Scale Visual Recognition Competition (ILSVRC), was used in order to extract higher level features from the images in our data set. Rather than use the entire network for our classification task, these feature vectors were extracted from different non-output layers and used with the K-Nearest Neighbor classification algorithm [5] in order to run preliminary experiments that would allow us to test the efficacy of our hypothesis.

Fig. 7. A satellite image of the map used in our data set. The red set of points shows the path traversed by the robot during data collection. Image source: https://www.google.com/maps

Fig. 8. An image collected for our data set using fish-eye camera lenses.

V. Results

A. Particle Filter

After completion, the particle filter implementation was evaluated qualitatively in a variety of settings in order to ensure that it behaved as would be expected under different conditions.

As a sanity check, the filter was first tested on synthetic odometry data which gave readings denoting only forward movement on the x-axis at a consistent rate of 2 meters per second. One such run with only odometry readings may be seen in figure 11. As would be expected, the error models developed for the Jackal result in some uncertainty in trajectory at each time step, which is reflected by the distribution of particles. Without any correction from other senors, the

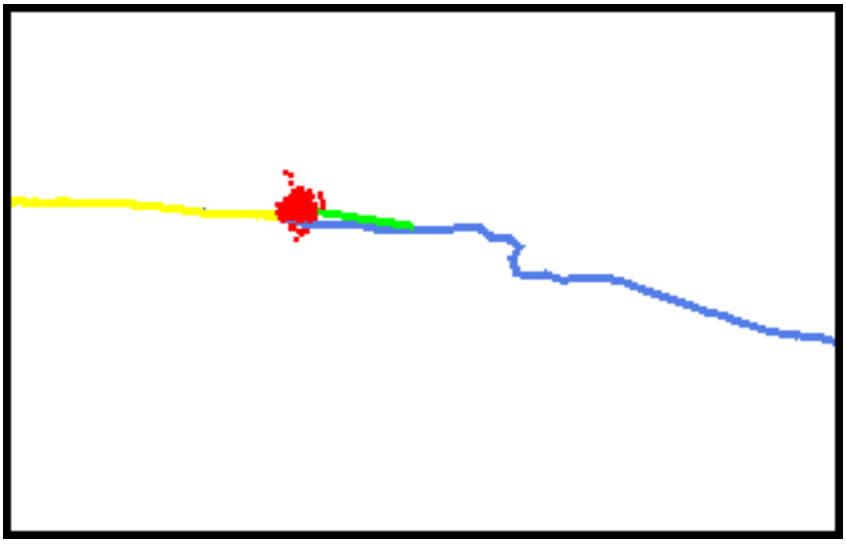

Fig. 9. A visualization of the particle filter generated using the Cobot visualizer. The blue path denotes the set of all GPS readings while the yellow path denotes the readings already given to the filter from a run data log file. The red dots represent the particle distribution and the green line, stemming from the particle cloud, represents the robot's pose estimate heading.

distribution can be seen to quickly diverge, denoting an ever growing level of uncertainty of the robot's true pose.

Following this, synthetic GPS data was added to this evaluation by giving a GPS reading at each time step with coordinates matching the 2 meter per second rate of movement. As may be seen in figure 12, the uncertainty of the run is greatly mitigated, with the distribution of particles clearly remaining within the GPS sensor's roughly 10 meter radius of uncertainty.

B. Visual Classification

Because classification algorithms operate on a discrete set of possible values, it was necessary to discretize the continuous space of GPS readings for our task. In order to do this, K-Means clustering [2] was used on our training data in order to generate 100 clusters along our map, which served as the set of possible classes which images could be labeled with.

For a preliminary evaluation of our methods, a K-Nearest Neighbor classifier was trained using 18 runs from our data set, with the 19th being used for testing. For evaluation, feature vectors were extracted for the images in the test set from a variety of layers of the AlexNet CNN, which were then run through the classifier. The classification accuracy of this experiment performed for each layer may be seen in figure 14. From these results, it was determined that the fifth pooling layer should be used for feature extraction.

C. Error Analysis

Given the low accuracy of our classification results, error analysis was conducted in order to determine the possible sources of error.

Figure 15 shows the results from the error analysis which was conducted. In general, it was found that the majority of misclassified images had been labeled with a class that was within 10 meters from the correct class, which suggests that the model was able to learn general locations relatively effectively. It may be seen that there were only certain areas with much higher error.

In order to get a clearer idea of which parts of the map the model failed in, a confusion matrix was constructed which visualized the correlation between the assigned and true labels

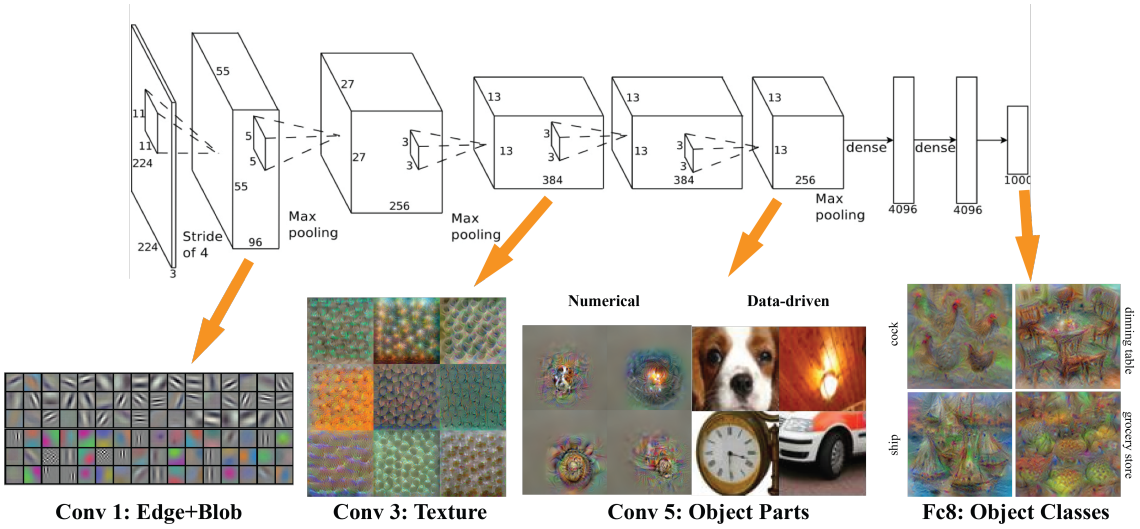

Fig. 10. The AlexNet CNN architecture. Features are extracted at each layer, starting from lower level features such as edges and colors, growing in abstraction to feature types such as partial objects. Image source: http://www.cc.gatech.edu/ hays/compvision/proj6/

for the images. Looking at the result of this analysis in figure 16, it may be seen that the model performed well throughout most of the map, only faltering at either end, with the error being particularly pronounced at the bottom right of the matrix. By once more looking back at the clusters in figure 13, it may be observed that the clusters are very consistent throughout the middle of the track, and diverge much more at the ends, particularly at the bottom right end of the map.

VI. Discussion and Future Work

Because the same path was followed each time data was collected on our map, it is clear that the divergence in clustering classes is due to GPS error. This variance in classes in parts of our map proved to cause too adverse an effect on our results. Because of this inconsistency found within the GPS, it was determined that a different method of collecting ground-truth locations for images is necessary.

Another factor that proved too problematic in the methodology was the data collection step. Because neural networks need large amounts of data to train, images were collected from many runs, a process which took approximately a week for a 280 meter path. Because a long-term goal of the project is to be able to create a map of the entire campus, this method of collecting data would prove to be too time consuming, necessitating different methods.

For the reasons delineated above, it was determined that a first alternative to explore would be the use of satellite imagery. We believe that an efficient method of mapping ground images to their respective patch of a satellite image would prove fruitful for our goals, since such a method might not require as large an amount of training data to derive a sufficiently accurate location probability distribution.

VII. Conclusion

A particle filter was implemented for use with the Jackal robot's GPS and internal wheel odometry sensors. Furthermore, the filter was integrated with visualization software which allowed for a qualitative analysis of the algorithm. Preliminary methods for developing a visual

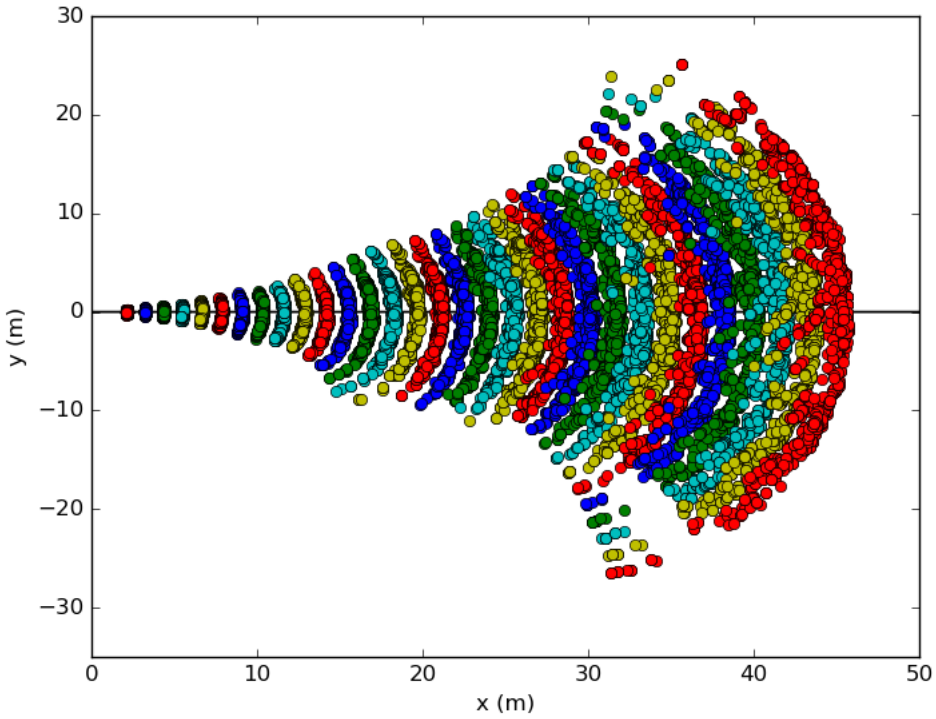

Fig. 11. A visualization of the particle filter using only wheel odometry from synthetic data denoting forward movement. As may be seen, without any correction from a perceptual model, the pose estimate uncertainty quickly grows.

perceptual model using CNNs were explored and it was determined that the methods were too inaccurate and time consuming for the long-term goals of the project. For future work, we plan on exploring methods of alleviating these problems, such as using satellite imagery, in order to derive a robust visual perceptual model which will allow for outdoor localization around campus and other outdoor environments.

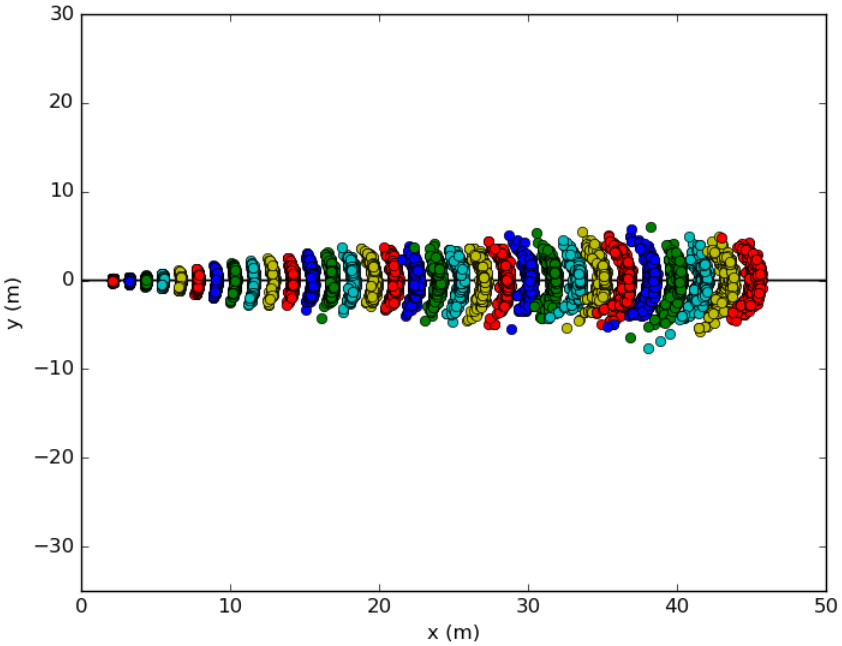

Fig. 12. A visualization of the particle filter using wheel odometry and GPS with synthetic data denoting forward movement. It may be observed that the distribution of particles is maintained within the GPS radius of uncertainty.

REFERENCES

[1] Motilal Agrawal and Kurt Konolige. Real-time localization in outdoor environments using stereo vision and inexpensive gps. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 3, pages 1063–1068. IEEE, 2006.

[2] Khaled Alsabti, Sanjay Ranka, and Vineet Singh. An efficient k-means clustering algorithm. 1997.

[3] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[4] Ian Goodfellow Yoshua Bengio and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[5] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[6] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[7] Kiyoshi Irie, Tomoaki Yoshida, and Masahiro Tomono. Mobile robot localization using stereo vision in outdoor environments under various illumination conditions. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 5175–5181. IEEE, 2010.

[8] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[10] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[11] Viet Nguyen, Agostino Martinelli, Nicola Tomatis, and Roland Siegwart. A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1929–1934. IEEE, 2005.

[12] Sebastian Thrun. Particle filters in robotics. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 511–518. Morgan Kaufmann Publishers Inc., 2002.

[13] Sebastian Thrun et al. Robotic mapping: A survey. *Exploring artificial intelligence in the new millennium*, 1:1–35, 2002.

[14] Christoffer Valgren and Achim J Lilienthal. Sift, surf and seasons: Long-term outdoor localization using local features. In *EMCR*, 2007.

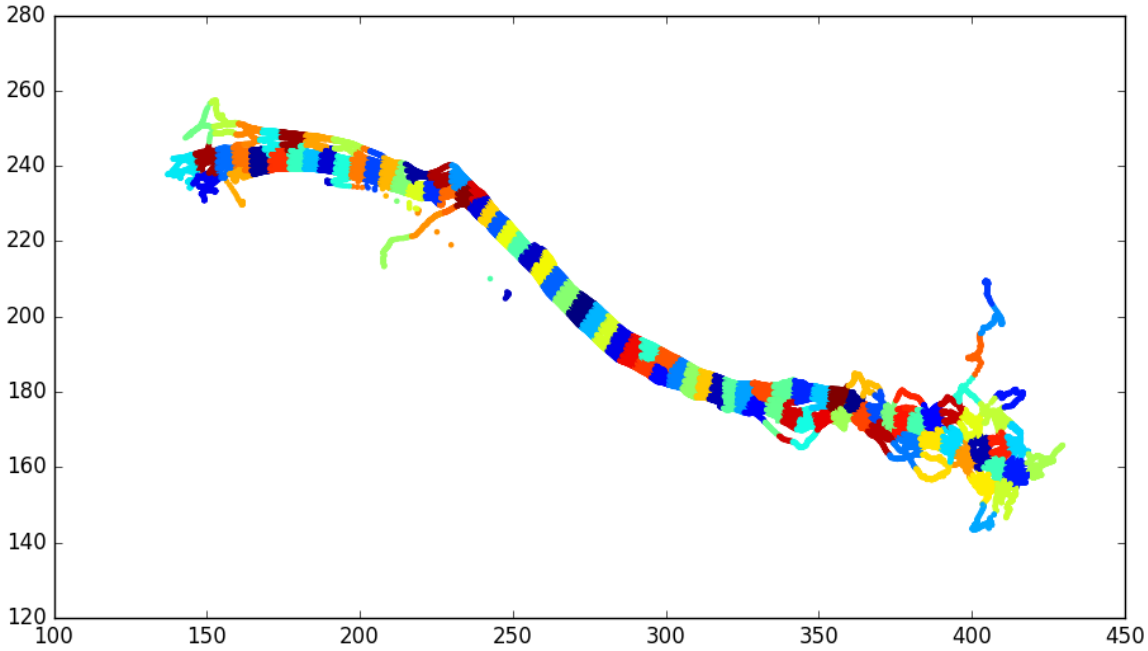

Fig. 13. A visualization of the classes generated on our dataset through K-Means clustering using Cartesian coordinates in meters. Each colored patch denotes a different class, the average distance of GPS points in our data set to the nearest cluster center was 5.53 meters.

Layer	Accuracy
Fully Connected 7	0.31
Fully Connected 6	0.38
Pooling 5	0.42
Convolutional 4	0.41
Convolutional 3	0.35
Pooling 2	0.41
Pooling 1	0.37

Fig. 14. The accuracy of the K-Nearest Neighbor classification on feature vectors pulled from a variety of layers of the AlexNet CNN.

[15] Manuela Veloso, Joydeep Biswas, Brian Coltin, Stephanie Rosenthal, Tom Kollar, Cetin Mericli, Mehdi Samadi, Susana Brandao, and Rodrigo Ventura. Cobots: Collaborative robots servicing multi-floor buildings. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5446–5447. IEEE, 2012.

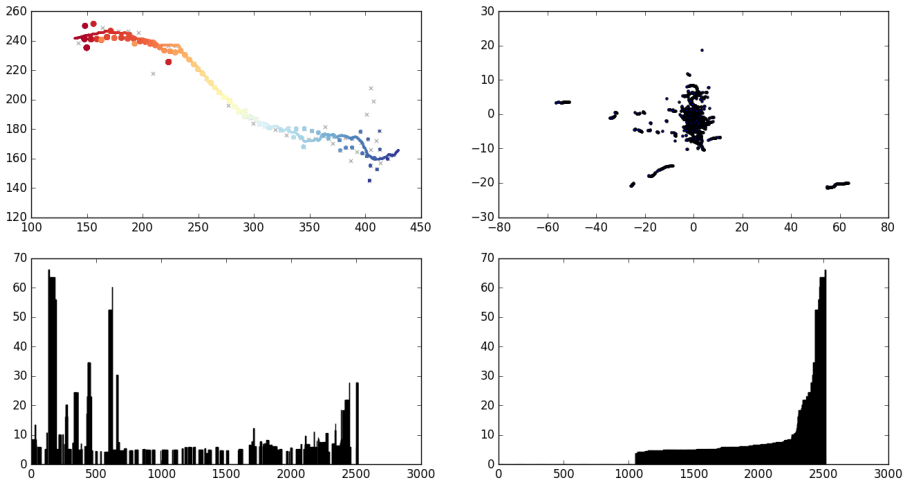

Fig. 15. Visualizations of error from the fifth pooling layer. *Top Left)* The continuous multi-colored curve represents the ground truth image labels of the path taken in the test run. The locations of colored dots represent the location cluster labels images were given, while the color corresponds to the true label that should have been assigned. *Top Right)* The distance between labels assigned to images and the location of their true label. This graphic shows that the majority of error was kept within a roughly 10 meter radius. *Bottom Left)* Another measure of the distance between assigned and true labels in sequential order. From this it may be inferred that the greatest error occurred towards the beginning of the run and at the end. *Bottom Right)* The same measure as the bottom left, sorted by error magnitude instead. Like the top right, this graphic implies that the majority of error was kept within roughly 10 meters.

APPENDIX

A. Robot Operating System

The Robot Operating System[2] (ROS) was used to write the majority of the software used in this project. ROS allows for robotics software to be written modularly, which is expressed in the form of different ”nodes”, which may each be executed to perform a particular function. Each node may also publish data through messages for named topics, which allows for the transfer of information between nodes.

For this project, three separate nodes were written, a particle filter node, a GPS reading conversion node, and a particle filter visualizer node.

The GPS conversion node uses a latitude and longitude value to set an origin in a coordinate plane and then converts readings from the robot's GPS sensor to meters on the x and y axis based on the reading's distance from the origin. The particle filter node subscribes to topics published by the robot's internal odometry and as well as the GPS conversion node, runs the information through our particle filter implementation, and publishes a pose estimate for the robot. Finally, the particle filter visualizer subscribes to all of the above topics in order to visualize the process.

B. Cameras and Image Preprocessing

For image data collection, two Point Grey[3] fish-eye cameras with fish-eye lenses were used due to the fish-eye lenses' wide field of view.

[2]http://www.ros.org/

[3]https://www.ptgrey.com/

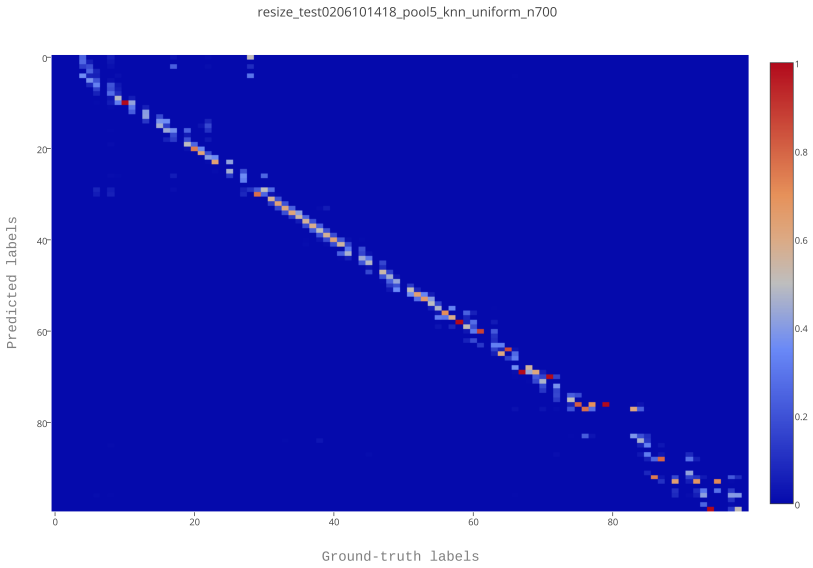

Fig. 16. A confusion matrix showing the correlation between the assigned and true labels in our data set using K-Nearest Neighbors classification.

Because the AlexNet CNN architecture expects square images of size 224×224, the OpenCV[4] library was used in order to undistort the images and resize them to the correct resolution.

[4]http://opencv.org/