Particle Filter for Skid-Steered Robot

Rodolfo Corona and Joydeep Biswas

***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 on a variety of maps which vary in size and path complexity, something which will allow us to test both the efficacy and robustness of the methods we will explore.**

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]. To this end, we derive a motion model for the robot and employ the use of GPS sensors as well as its internal odometry for our implementation.

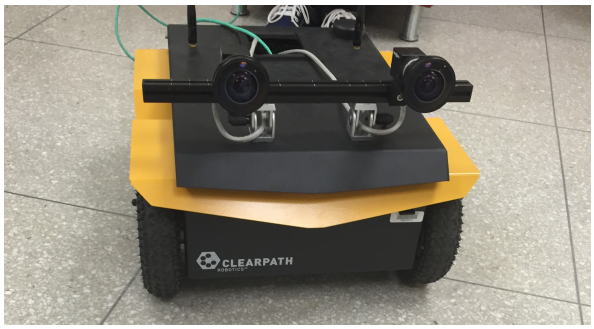

Fig. 1. The ClearPath Jackal equipped with two fish-eye camera lenses.

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.

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,

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

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 $\mathbf{x}_t = (x_t\ y_t\ \theta_t)^T$ may be found using the following formula:

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

Given that the space of possible poses is continuous, however, makes the task of searching through it intractable. Further, the cost of computing such a function can quickly grow too large 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 us to compute an estimate using only the current time step's readings and the previous time step's pose estimate:

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

For working tractably in a continuous space, localization algorithms use a variety of methods to approach 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 in order to estimate the trajectory of the robot over time. Given the dependence of each motion model estimate on the previous time step's estimate, the motion model is highly prone to systematic errors. The perceptual model is therefore used in order 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**. At each time step, particles are assigned different pose estimates based on the robot's 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 used to re-sample the particles' poses.

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:

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

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. In order to effectively use GPS for localization, the latitudinal and longitudinal 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 is then used in order to assign a weight to each particle, where a greater distance from the GPS reading results in a lower weight being assigned.

3) Resample: Using each of the N particles' weight, 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:

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

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.

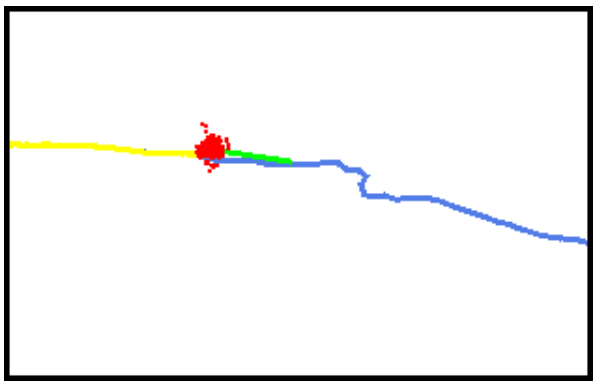

Fig. 2. A visualization of the particle filter using GPS. The blue path denotes the set of all GPS readings while the yellow path denotes the readings already given to the robot. The red dots represent the particle distribution and the green line, stemming from the particle cloud, represents the robot's pose.

III. Methodology

A. Data Set

A roughly 280 meter path was chosen for use as a map for testing our particle filter. This path was traversed a total of 19 times at different times of day in order to capture as much variability in daylight as possible. Through each run, data from all robot sensors was collected (e.g. GPS, cameras, etc.).

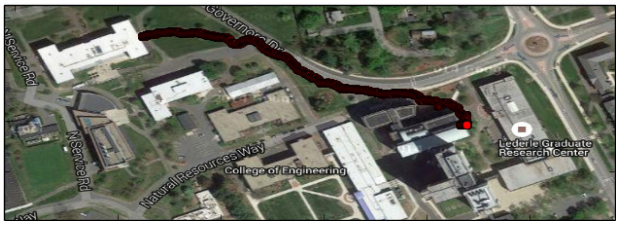

Fig. 3. A satellite image of the map used in our data set. The red map shows the path traversed by the robot during data collection.

B. 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, which won the 2012 ImageNet Large Scale Visual Recognition Competition (ILSVRC), is 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 are used with a variety of different classification algorithms (e.g K-Nearest Neighbors, SVNs, etc.) in order to determine an effective model for localization.

IV. Results

A. Particle Filter

B. Visual Classification

Because classification algorithms operate on a discrete set of possible values, it is necessary to discretize the continuous space of GPS readings for our task. In order to do this, K-Means clustering 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, and the 19th was used for testing. For evaluation, feature vectors were extracted for our 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 5. From these results, it was determined that the fifth pooling should be used.

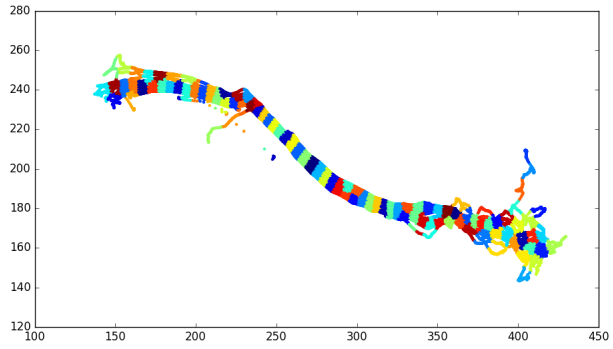

Fig. 4. A visualization of the classes generated on our dataset through K-Means clustering using cartesian coordinates in meters. Each colored patch denotes a different class, the average distance of GPS points in our data set to the nearest cluster center was 5.53 meters.

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

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 6 shows the results from some error analysis that 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 for the images. Looking at the result of this analysis in figure 7, 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 4, 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.

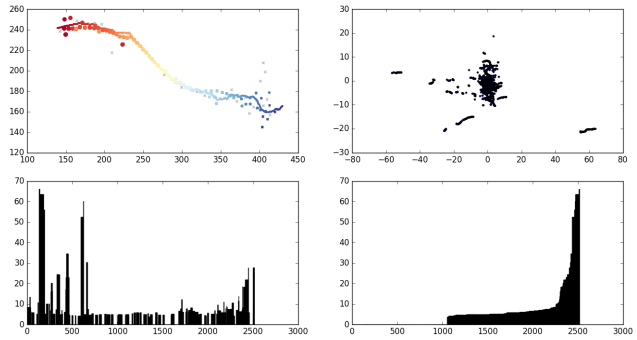

Fig. 6. Visualizations of error from the fifth pooling layer. *Top Left) Top Right) Bottom Left) Bottom Right)*

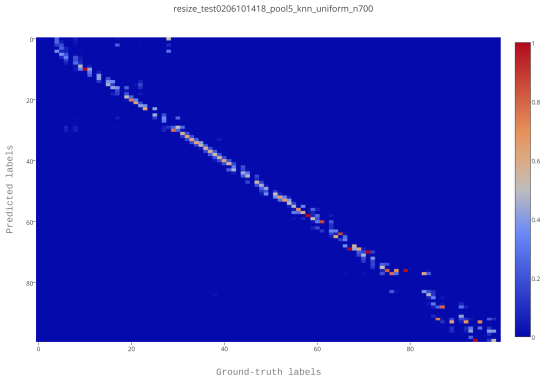

Fig. 7. Visualizations of error from the fifth pooling layer. *Top Left) Top Right) Bottom Left) Bottom Right)*

V. 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 and 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.

VI. Conclusion