CamDroneLoc: Camera Vision-based Drone Localization

Vinay Lanka
University of Maryland
Email: vlanka@umd.edu

Vikram Setty
University of Maryland
Email: vikrams@umd.edu

Mayank Deshpande
University of Maryland
Email: msdeshp4@umd.edu

***Abstract*—This report gives insights into a novel camera vision-based drone localization method CamDroneLoc that provides visual odometry in known environments using a particle filter pipeline. Compared to traditional vision-based localization techniques like template matching and even deep-learning approaches, CamDroneLoc makes use of the drone's motion model and runs at a higher computational efficiency. This is done by using an image encoding pipeline to ensure a quick measurement update by the particle filter leading to fast and accurate convergence of the odometry prediction measurements. This project has been developed and is being submitted as a part of the final project of the course CMSC733: Computer Processing of Pictorial Information offered by the University of Maryland.**

I. Introduction

Localization is an important aspect of any robotic system. The ability to figure out exactly where an autonomous/manually driven system is in a known/unknown environment is extremely important to carry out desired tasks. This property is even more important considering aerial systems, especially drones. While surveying, monitoring, or actively participating in mapped/unmapped environments, it is crucial for dromes to know where they are. Lapses in localization accuracy and precision can prove to be fatal based on the application. However, drone localization is tricky. Though hardware like RTK GPS systems that localize with extreme accuracy exist, they are often of no use when drones fly in GPS-denied environments, which is often the case in military applications. To circumvent these issues, drones must rely on other traditional localization methods that make sure of hardware like lidar, radar, and cameras. Radars though effective for surrounding objects' velocity estimation, if not necessarily the best input source for localization considering the amount of noise that comes in the readings. Lidar sensors and cameras make better choices, though each with their own advantages and disadvantages. Lidar sensors can map the 3d world to a decent level of precision and can operate with a full field of view but they are active sensors and can end up interacting with other lidar sensors or reflective surfaces. Cameras on the other hand have a limited field of view but can capture information like color unique to many features in an environment. However, camera images may not prove to be useful in case of bad weather conditions. A calibrated lidar-camera pair overcomes takes the advantages of each of these sensors but can methods that use these sensors together can get computationally expensive and physically heavy for the drone depending on the hardware availability.

In this project, we present a vision-based method that uses camera image input for localization and visual odometry in a mapped environment by using an encoder to encode camera input which is then used as the measurement for a particle filter localization system. This method, though suspect to the problems of traditional vision-based localization methods, provides an extra layer of robustness by making use of feature properties of the environment. We assume a setup where a drone with constant zero pitch and roll is moving around in a mapped environment with a monocular camera facing down at all times (with a shutter speed high enough to prevent image imperfections while moving at high speeds). We also assume known control inputs (within the limits of reasonable error) to drive the drone in a particular desired motion, a property that becomes a basis for the drone's motion model used in the particle filter implementation, explained in further sections. A basic depiction of the drone's setup in a realistic Gazebo PX4 SITL world with the drone camera's view projected onto its image plane is shown in Figure 4.

Fig. 1. Setup of the Drone in the Gazebo PX4 SITL World

The following sections go over the overall pipeline, including both theoretical aspects and implementation details. Starting with exploring all of the image encoding methods tried, tested, and compared to then integrating the particle filter system to produce localization predictions in a PX4 Autopilot-based Gazebo SITL world, this report provides a comprehensive analysis of CamDroneLoc.

II. Image Encoding

A big bottleneck in drone localization is often latency. Though complete and effective methods exist, they are rarely efficient. The only place where the image is used in the particle filter is the measurement model, where the similarity between the observed input and expected input is compared to update probability distribution beliefs (explained in more detail in further sections). Depending on how we use and compare the image data to calculate this information, the latency of the system may vary greatly. Naive traditional methods like template matching using image correlation, or end-to-end black-box deep-learning methods are don't appropriately make use of the assumption that the environment is mapped/known and thus are less efficient in both performance and computation. We formulate a method of using image encodings/vectors instead of the images themselves to use as an input to the particle filter's measurement model. With an intuition of how to encode the images to preserve the necessary information of the image to be used specifically to calculate similarity with other images, this method can reduce the time-complexity of image similarity calculation from $O(max(M,N)^2)$ to $O(max(M,N))$, where M, N are the height and width of the images respectively. Through extensive literature review and experimentation, we have formulated three image encoding methods best suited to this application. In the following subsections, we provide a theoretical background, implementation intuition, and performance analysis for each major method significantly explored.

A. Approach 1: CNN Encoder

The core concept of this encoder lies in encoding images into a vector so that assessing the visual similarity between the particles' synthetic images and the actual image captured by the sensor (the "main image") would be the basis of the measurement model of the particle filter (explained in the next section). To achieve this, pre-trained Convolutional Neural Networks i.e. CNNs (ResNet18 and VGG18) are employed to extract feature vectors from both image sets by extracting them from the average pooling layer followed by normalization. These feature vectors encapsulate high-level semantic information about the content of the images, providing essential inputs for our particle filter's decision-making mechanism. A high cosine similarity score indicates that the synthetic image generated from a particle's state closely resembles the actual scene, suggesting a higher likelihood that the particle represents the true location.

B. Approach 2: VecKM Encoder

In their work, Yuan et. al. [1] present a linear in time and space geometry encoder via a Vectorized Kernel Mixture (VecKM). Using this approach, they were able to encode 3D point clouds into single-dimension vectors while preserving local geometry features in $O(nd + np)$ time compared to traditional encoders that take in the order of $O(n^2 + nKd)$ (where n is the number of points in the point cloud, d is the encoding dimension, K is the neighborhood size, and p is the marginal factor). The formulation for encoding the point cloud presented in the paper is shown in the equation below.

$$E_A\left(\mathcal{N}(x_0)\right) = \frac{1}{n} exp\left(i(x_k - x_0)A_{3\times d}\right)$$

Here, x_0 represents the point around which the encoding is being generated, E represents the encoding itself, $\mathcal{N}$ represents the neighborhood of that point while x_k represents the set of all points in that neighborhood, n represents the total number of such points in that neighborhood, and A is a $3 \times d$ dimensional matrix following the normal distribution i.e. $N(0, \alpha^2)$ i.e. having zero mean and a standard deviation of α, a tunable parameter.

Though initially designed for point-cloud encoding, the property of preserving local geometry in encodings in linear time stands out as a desirable trait, especially in our problem statement of encoding images. However, we'd need to encode a discrete number of points and not the whole image itself to use the VecKM approach. To enable these, there are several possible techniques that can be used. 3D points in point clouds can be substituted by feature key points, feature descriptor vectors, or a combination of feature key points and descriptors. The descriptor vector could be added as the original or processed feature descriptor. To obtain these key points and descriptors, any standard image feature extractor like SIFT, SURF, ORB, or BRISK could be used. The choice and performance of each of these point representations depends on the application. In our case, we require a representation that preserves the local geometry of the image which is analogous to the the relative arrangement of features invariant to scaling, orientation, and illumination. After extensive experimentation, we observed the best performance (high encoding similarity) between related images (having an overlapping field of view) when using a VecKM Encoder using SIFT feature key points at a low value of α. Using this formulation, the VecKM-based encoding for the image at time t i.e. $\mathcal{I}_t$ is shown below.

$$E_A\left(\mathcal{I}_t\right) = \frac{1}{n} exp\left(i(x_{\mathcal{I}_t,j})A_{3\times d}\right)$$

Analogous the original VecKM formulation, n is the total number of SIFT features in the image $\mathcal{I}_t$, and $x_{\mathcal{I}_t,j}$ where $j \in [1, n]$ represents each two-dimensional SIFT feature keypoint.

C. Approach 3: Histogram of Features Encoder

Another approach explored for encoding the captured images from the drone to fixed-size vectors using feature information includes using feature histograms. The intuition behind this method is to initially build a clustering model based on all the features in the mapped world environment that maps any SIFT feature descriptor to a specific bin in the clustering model. Then, using this clustering model, the descriptors for each SIFT feature in an image can be parsed to build a histogram. Then, two images can be compared by calculating the correlation or intersection between their respective histograms. The advantage of this encoding framework is that it captures the type of features (through respective

histogram bins) present in an image without encoding any positional/local geometry information allowing for images with an overlapping/rotated field of view to have a histogram similarity based on overlapping features.

III. Particle Filter-based Localization

The main system that the drone uses for localization in this approach is a particle filter model. In a particle filter, multiple particles (or beliefs) of the system's (drone's) state (position in space and time) are initialized, and based on the system's predicted movement (from local odometers/other hardware/software) and temporal measurements (in this case the encodings of the images of the world the drone captures), the model updates its particles and re-weights/resamples them from time to time to let the collective average of the particle's state belief converge to the actual state of the system. The three major steps of a particle filter per iteration include the motion model/prediction step (where each particle's state is updated based on the predicted motion of the system), the measurement model (where the importance/weight of a particle is assessed/assigned/updated based on the similarity of its predicted state and the system's measurement), and the update step (where the particles are re-weighted and resampled based on their importance calculated in the measurement model step). Each of these three steps along with their mathematical formulation and usage in this approach (CamDroneLoc) is explained in detail in the following subsections.

A. Motion Model (Prediction Step)

The motion model is a crucial component of the particle filter, responsible for predicting the new positions of the particles based on the drone's motion. Essentially, each particle represents a potential location of the drone, and by adding the drone's velocity to these positions, we simulate the movement of the drone across the environment.

The PX4 Autopilot uses an Extended Kalman Filter to estimate the position of the drone. It uses the EKF2 estiomation system that works on a 24 states (Attitude, Velocity, Position, Gyro bias offsets, Gyro scale factors, Z accel bias, Earth magnetic field, Body magnetic field, Wind Velocity). The EKF runs on a delayed 'fusion time horizon' to allow for different time delays on each measurement relative to the IMU.

$$v_{new} = \begin{bmatrix} v_n & v_e & v_d \end{bmatrix} + \begin{bmatrix} 0 & 0 & g \end{bmatrix} dt + T_{bn} dV_{truth}$$

$$dV_{truth} = dV_{meas} - dV_{bias} - \begin{bmatrix} dv_x Noise & dv_y Noise & dv_z Noise \end{bmatrix}$$

Where:

- v_{new} is the updated velocity vector.
- v_n, v_e, and v_d are the current velocity components in the North, East, and Down directions.
- T_{bn} is the direction cosine matrix that rotates vectors from the body frame to the navigation frame.
- g is the acceleration due to gravity.
- dV_{meas} is the measured delta velocity vector from the IMU.
- dV_{bias} is the delta velocity bias vector.
- $dv_x Noise$, $dv_y Noise$, and $dv_z Noise$ represent the noise in the IMU delta velocity measurements.

We take the velocity measurements as a ROS topic in our mavros node, and update our state with

$$x = x_{previous} + v_x \delta t$$
$$y = y_{previous} + v_y \delta t$$

Imagine each particle as a tiny drone. If the actual drone moves north by 5 meters, every particle also moves north by 5 meters. This prediction step is essential because it allows the particles to move according to the expected movement of the drone, based on its velocity. By doing this, we prepare the particles for the next step, where we will compare their predicted positions against the actual observed data. This process helps refine the estimate of the drone's true location by ensuring the particles are distributed around where the drone is likely to be after moving.

This prediction step is also known as the propagation step, where we propagate the particles forward in time. By continuously updating the particle positions based on the velocity, we maintain a dynamic set of possible locations for the drone. This dynamic update is crucial for accurately tracking the drone as it moves through the environment, especially in scenarios where GPS data is unavailable.

B. Measurement Model

The measurement model is the part of the particle filter that compares the current view from the drone's camera with what each particle "sees" from its predicted position. In this project, this comparison is achieved using image encoding and similarity measurement. First, the current image captured by the drone's camera is encoded using the desired encoding method.

For each particle, a predicted image is generated based on its position. This involves using the particle's coordinates to extract a portion of the map image (the known environment) that corresponds to what the drone would see if it were at that particle's location. By encoding this predicted image in the same way as the current image, we obtain a comparable feature vector. The similarity between the encoded current image and the encoded predicted image is then calculated. This similarity score indicates how close the particle's predicted view is to the actual drone's view.

Particles with higher similarity scores are considered more likely to represent the drone's true position. These particles receive higher weights, which means they have a greater influence on the final position estimate. This step ensures that particles aligning better with the observed data are given more importance, thus refining the estimate of the drone's location. By continuously updating the particle weights based on the similarity scores, the particle filter adjusts its belief about where the drone is, making the estimation more accurate over time.

For the CNN-based encoding method described in the previous section, the measurement model is the cosine similarity between each encoding. It can be expressed mathematically as for two encodings X_1 and X_2 as shown below shown below.

$$S(X_1, X_2) = \frac{X_1 X_2}{||X_1||||X_2||}$$

If using a VecKM encoder for encoding the drone's images, the similarity can be computed using the formulation below. The Hadamard operator signifies element-wise multiplication and $\bar{X_2}$ signifies the conjugate of that encoding/vector.

$$S(X_1, X_2) = \sum X_1 \odot \bar{X_2}$$

If using a histogram of features encoder with a correlation similarity, the similarity between the encodings can be computed as follows. This method calculates the correlation (over the probability distribution) of the two histograms/encodings.

$$S(X_1, X_2) = \frac{\sum_I (X_1(I) - \bar{X_1})(X_2(I) - \bar{X_2})}{\sqrt{(\sum_I (X_1(I) - \bar{X_1})^2)(\sum_I (X_2(I) - \bar{X_2})^2)}}$$

If using a histogram of features encoder with intersection similarity, the similarity between the encodings can be computed as follows. This approach basically counts the bins where the histograms/encodings overlap.

$$S(X_1, X_2) = \sum_I min(X_1(I), X_2(I))$$

C. Update and Resample

After getting our measurements, we now use the similarity in the update step, basically altering our probability distribution modeled by the particles.

which is an implementation of the equation

$$x = \|\mathcal{L}\bar{x}\|$$

which is a realization of Bayes theorem:

$$\begin{aligned} P(x \mid z) &= \frac{P(z \mid x)\, P(x)}{P(z)} \\ &= \frac{\texttt{likelihood} \times \texttt{prior}}{\texttt{normalization}} \end{aligned}$$

After updating the particle weights based on the measurement model, the next step is to normalize these weights and resample the particles. Normalization ensures that the sum of all particle weights equals one, converting them into valid probabilities. This is necessary because the weights initially may not sum up to one due to the continuous updates during the measurement step. Normalizing the weights allows us to use them directly in the resampling process.

Resampling is the process of generating a new set of particles based on their normalized weights. Particles with higher weights, which represent more likely positions, have a higher probability of being selected multiple times. Conversely, particles with lower weights, which are less likely to represent the drone's true position, may be discarded.

We don't resample at every iteration as this leads to sample impoverishment. We've faced this as our methods tend to converge very quickly due to high similarities. There could be false positives which lead to wrong estimations. One way to combat this is to increase the number of particles and brute force the estimation process. We can determine when to resample by using something called the *effective N*, which approximately measures the number of particles that meaningfully contribute to the probability distribution. The equation for this is

$$\hat{N}_{\text{eff}} = \frac{1}{\sum w^2}$$

If $\hat{N}_{\text{eff}}$ falls below some threshold it is time to resample. We've found that $N/2$ works as a good point to resample.

By concentrating the particles around the most probable locations, the resampling step enhances the accuracy of the position estimate. It effectively filters out unlikely particles and focuses computational resources on the areas where the drone is most likely to be. This iterative process of prediction, measurement update, and resampling allows the particle filter to continuously refine its estimate of the drone's position, ensuring robustness even in the absence of GPS data. Over successive iterations, the particle filter hones in on the true position of the drone, providing reliable localization based on the visual data from the camera.

D. General Distribution and Resampling methods

We initially faced the Filter Degeneracy From Inadequate Samples problem, as we're bound by hardware to cover such a large area. To solve this, initially, we sample either a Gaussian Distribution about a state or a Uniform Distribution across an area, depending on whether we know where the last location of the drone is. This choice is crucial as we've found experimentally that using a Gaussian distribution around the last location of the drone makes the particles converge faster and more accurately.

Choosing a resampling method was tricky due to the nature of the problem we're trying to solve. We cannot resample too often, as we lose out on potential positives, but due to the lack of particles, we would need to resample to cover ground. We've checked out many resampling methods and their performance is given below.

Multinomial Resampling computes the cumulative sum of the normalized weights. This gives you an array of increasing values from 0 to 1. This method works well to generate some variance in the distribution but also sometimes resamples smaller weights more frequently. Stratified Resampling aims to make selections relatively uniformly across the particles. It works by dividing the cumulative sum into N equal sections and then selects one particle randomly from each section. This guarantees that each sample is between 0 and $\frac{2}{N}$ apart. Systematic resampling is similar to stratified resampling where

the space is divided into N divisions. We then choose a random offset to use for all of the divisions, ensuring that each sample is exactly $\frac{1}{N}$ apart.

E. Estimation

To give a final likelihood of the drone agent, we need to extract meaningful location data from the particles. This is done by extracting a weighted mean of the state of the particles.

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x^i w^i$$

IV. Simulation and Results

A. Simulation Environment

The simulation environment used for development and testing is on the Gazebo Classic simulator with the PX4 Software In the Loop (SITL) simulation to best mimic the real-life conditions of a UAV flight. The PX4 SITL provides a realistic drone trajectory with realistic imbalance while hovering at a point.

The Gazebo world is a custom-built replica of the College Park environment, with the ground plane having a satellite image of the University of Maryland campus. This is a realistic replica of what a drone would see, typically hovering at 200-300m above ground level. The ground plane is an image and we assume flat ground due to the height at which typical drones operate.

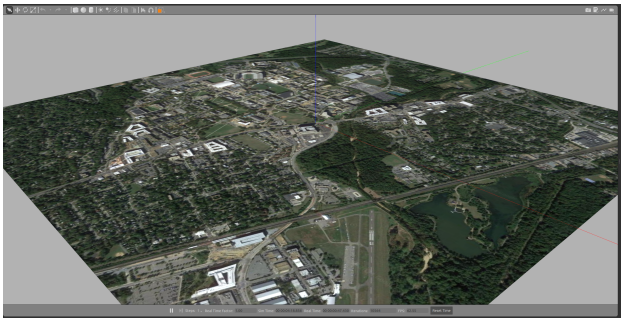

Fig. 2. College Park Gazebo World

The waypoints for the mission are set by a Ground Control Station (GCS). The GCS chosen for this project is QGroundControl, as it's powerful and has a good visualization window to view the current state, trajectory, and movements of the drone in real-time.

Our Particle filter is visualized as points in a *Matplotlib* window. This is for ease of use while debugging and developing the application and is useful for viewing the position of particles with an accurate grid representation.

B. Encoder Analysis

To compare the performance of the different encoders introduced earlier, we compared the similarity scores captured by the encoders on different pairs of satellite images extracted from Google Maps [2]. These images include the following.

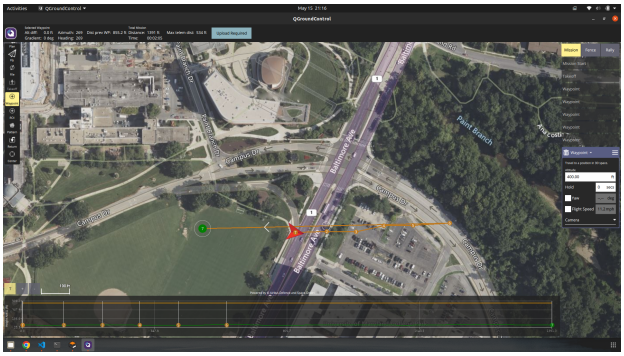

Fig. 3. QGroundControl Ground Control Station

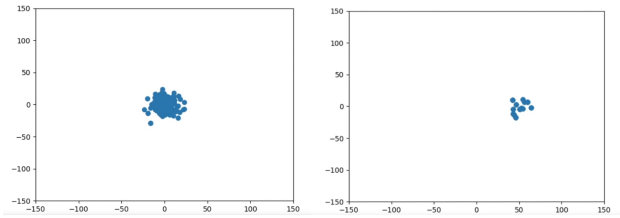

Fig. 4. CamDroneLoc Filter over time

- College Park 1
- College Park 1 - Translated
- College Park 2
- San Francisco 1
- San Francisco 1 - Rotated
- San Francisco 2

All of these images can be visualized in Figure 5. Using these images, we used each of our encoders to encode the images as vectors and we calculated the similarity scores between the following sets of image encodings.

- Pair 1: College Park 1 & College Park 1 - Translated
- Pair 2: College Park 1 & College Park 2
- Pair 3: College Park 1 - Translated & College Park 2
- Pair 4: San Francisco 1 & San Francisco 1 - Rotated
- Pair 5: San Francisco 1 & San Francisco 2
- Pair 6: San Francisco 1 - Rotated & San Francisco 2
- Pair 7: College Park 1 & San Francisco 1

The similarity scores for each of these image pairs are encoded using the four encoding methods introduced previously (CNN-based, VecKM-based, Histogram of Features with Correlation Similarity, Histogram of Features with Intersection Similarity) are shown in Table I. Each encoder and similarity score computation method uses a different scale/distribution for expressing similarity as can be seen. Along with the encoder similarities, a SIFT feature IoU between each encoded image pair is also shown. This IoU is computed by taking the intersection over union of the features common between images over the total unique features in both images combined. the Feature IoU observed for all pairs overall in Table I is low as a high FLANN Feature Matching threshold ratio was used to prevent outliers from being classified as matches.

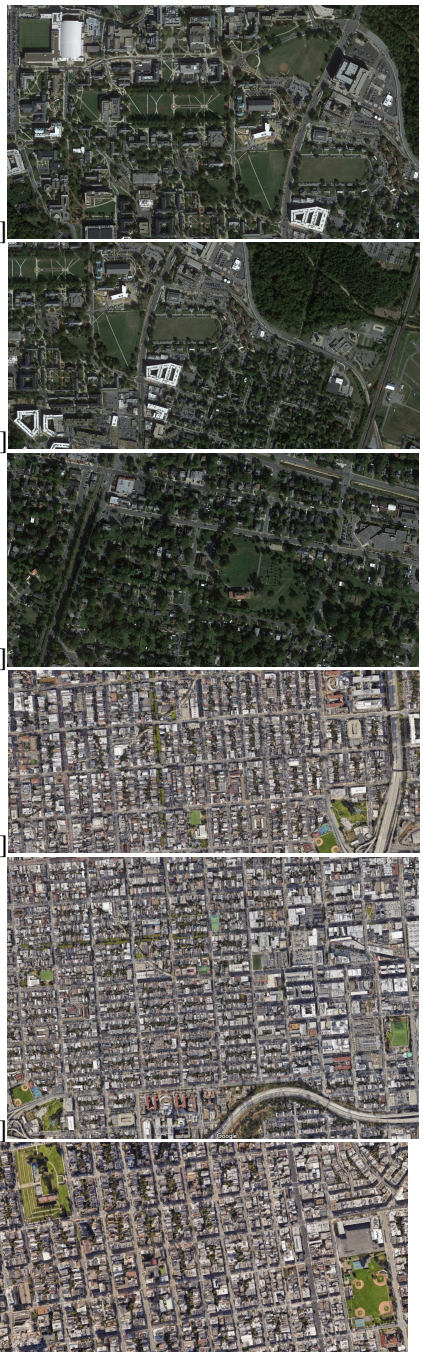

Fig. 5. The Images used for Encoder Performance Analysis: (a)CP1, (b)CP1-Translated, (c)CP2, (d)SFO1, (e)SFO1-Rotated, (f)SFO2

Overall, we expect relatively higher similarity scores for Pair 1 and Pair 4. compared to the other pairs as they represent an overlapped area between them (through translation and rotation respectively). As expected, they are the only image pairs with a non-zero Feature Matching IoU.

TABLE I
SIMILAIRTY SCORES (ON DIFFERENT SCALES BASED ON THE ENCODER) FOR DIFFERENT PAIRS OF IMAGES

Image Pair	CNN	VecKM	Hist. Corr.	Hist. Int.	Feature IoU
1	0.89	0.045	0.63	85	0.053
2	0.72	0.024	0.40	58	0
3	0.68	0.019	0.55	64	0
4	0.83	0.063	0.39	82	0.028
5	0.83	0.039	0.12	71	0
6	0.91	0.026	0.28	76	0
7	0.73	0.015	0.01	61	0

Overall, the following inferences can be drawn from the results in Table I.

- The CNN-based encoder captures overall similarity between images well but fails when there are drastic changes in scale and rotation making it not the most suitable for this approach.
- The VecKM encoder provides decent similarity scores as per expectations and also runs computationally efficiently, without needed prior information of the environment. However, it is also important to note that the VecKM model works only at very low values of α (the value of α used in Table I was 0.1), to capture local geometry-like properties in the image.
- The Histogram of Features encoder (using both similarity metrics) also produces desirable results. However, the clustering model first needs to be trained/developed using the features of the entire environment before use. Further, it is slightly slower in execution in comparison to the VecKM encoding approach.
- Overall, the VecKM-based encoding approach is the most desirable to use followed closely by the Histogram of Features Encoder.

C. Performance Analysis

Over the course of developing CamDroneLoc, we've found that a good performance benchmark can be assessed by taking the image similarity performance as a quantifiable metric to estimate the Bayes probability in recursive state estimation. The image similarity of various methods we've tested are shown below.

D. Inferences and Discussions

V. CONCLUSION

We've proposed, developed and demonstrated CamDroneLoc which provides visual odometry in known environments using a particle filter pipeline that is based on an efficient image encoding pipeline to ensure a quick measurement update.

The 2 proposed ideas of using similarity between feature vectors as a metric for the measurement vector to update weights and the encoding for such a metric are novel methods both of which work successfully. Both are based on intuitive ideas where the local geometry is preserved during encoding and does not change too much in similar frames, and the similarity of such a vector can be used to update beliefs in recursive bayes state estimation.

REFERENCES

[1] Veckm encoder. https://arxiv.org/abs/2404.01568.
[2] Google maps. https://www.google.com/maps.