Hybrid-SLAM: A Comparison between ORB-SLAM and LIFT-SLAM

Kelvin Kang
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Gerald D'Ascoli
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Jonathan Lord-Fonda
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

***Abstract*—Simultaneous Localization and Mapping (SLAM) allows robots to build a consistent map of the surrounding environment and to localize themselves within that newly-created map. Many robots accomplish this by using visual SLAM, or VSLAM, by identifying unique features in an environment and tracking those features between image frames. These features are often based on SIFT, or Scale Invariant Feature Transform, because SIFT features track well from different scales and angles. A powerful monocular SLAM method came out in 2015 that uses ORB features, or Oriented multi-scale FAST (Features from Accelerated Segment Test) and Rotated BRIEF (Binary Robust Independent Elementary Features), that tested very successfully in application. Recently, there has been exploration into learned features generated from deep neural networks that could adapt to the local environment and generate features better tailored to the robot's surroundings. One such method is LIFT, or Learned Invariant Feature Transform. This paper seeks to produce a hybrid implementation by using LIFT features with a traditional ORB-SLAM backend. Experiments were conducted on the KITTI dataset to evaluate the performance of these two feature bases as separate implementations of SLAM.**

***Index Terms*—SLAM, Visual SLAM, VSLAM, LIFT, Learned Features, ORB**

I. Introduction

Simultaneous Localization and Mapping (SLAM) systems are integral to today's self-driving car research. They allow not only the collection of data for producing highly-detailed maps of cities, but also allow autonomous vehicles to generalize to never-before-seen locations. The most flexible SLAM algorithms require only monocular input data to create maps and find loop closures. These SLAM methods can be roughly split into two categories, those with hand-engineered feature detection and loop-closure back-end, and end-to-end deep learned algorithms that utilize the flexibility of neural networks to train SLAM algorithms from large data sets. Both of these methods have their advantages, the hand-engineered systems tend to have better performance and loop closure probabilities, while the learned systems are more flexible and can be applied to a variety of data modalities with ease. However, there is a third category of SLAM algorithms, the hybrid approach that seeks to combine both methods in order to reap the accumulated benefits. The hybrid approach being presented in this work is a combination of deep learned LIFT features with a traditional SLAM back-end, specifically derived from ORB-SLAM. It is the hope of these authors that deep-learned features will provide significant advantages over ORB features, and that the tried and true SLAM back-end will be able to handle the complex statistical analysis of SLAM and loop closures. Successful integration of these two concepts may lead to a SLAM system both accurate and flexible enough to be successfully deployed in a variety of scenarios and sensor modalities. To this end a LIFT-SLAM hybrid method was deployed on the monocular, black-and-white KITTI self-driving dataset and the outcome was compared to traditional ORB-SLAM.

II. Related Work

A. ORB-SLAM

ORB-SLAM is a seminal work that builds upon PTAM to solve the SLAM problem. At each time step an input image is sent to the system by cameras and this image is scanned for ORB features. These features are recorded and saved for future use, and also plotted in estimated 3-d poses in the newly-mapped world. The estimated pose of the subject, and the corresponding ORB features, are saved as a "keyframe," in a pose graph and used throughout the process for loop-closure. If the ORB features in a given keyframe are sufficiently redundant, then the keyframe is removed from the system, so as to reduce the amount of memory required to build and maintain the map of the world, resulting in diminishing memory increases throughout the life of a run. Additionally, if an input image's ORB features match a keyframe at some other point in the dataset at least as well as a selection of recent frames, then the system searches its keyframe database for possible matches that it can perform loop closure over. Any keyframes found to be at least as good as the worst recent frame are considered possible candidates. Once at least n possible loop-closure candidates are found in consecutive frames, then the system identifies it as a loop closure and aligns the two sections of the covisibility graph. [4] Subsequent bundle adjustment will then reduce the l2-norm error across all of the sighted points, thereby readjusting the map to account for the new loop closure and fix the map's alignment. One of the primary achievements of ORB-SLAM is its use of ORB features for all tasks, mapping, tracking, loop closure, and relocalization. [4] ORB features were used in the original paper for two reasons. First, their rotational invariance allows them to recognize similar locations from a variety of

viewpoints, which is important in SLAM applications as the pose of the robot can never be guaranteed to replicate previous positions. Additionally, ORB features are fast to compute [4], allowing the algorithm to spend more of its computational power on advanced forms of bundle adjustment, leading to more accurate map reconstruction and loop closure.

B. LIFT

LIFT stands for Learned Invariant Feature Transform and is a deep-learning method used to find features in an image that correspond to unique locations. [2] These features can then be used for a variety of applications, to perform object detection, classification, or in this case, matching specific structures and locations to other instances or images of themselves. The LIFT feature detection pipeline involves three distinct steps. First, unique or 'interesting' locations in the image are detected. [2] Finding unique points of interest is important as they are what allow programs to draw connections only between the same or very similar objects. They need to be rare; a patch of flat, blue sky without any variation will match almost every other patch of sky, thereby making it a poor choice for a feature. Previous methods of feature point detection, such as SURF, SIFT, BRIEF, or even ORB, depend largely on hand-engineered methods to locate these features. [2] By contrast, LIFT identifies scale and orientation robust features by learning a network that can detect them, deciding which points are worth logging and which are not. After a feature point is detected, its orientation has to be established. There is far less research on this area so most algorithms rely on SIFT or ORB. [2] LIFT, again, takes a different approach and uses neural networks to estimate the orientation of the given feature points. [2]. Finally, the feature point, along with its orientation, is converted into a feature descriptor that can be used to recognize the same (or similar) points, even from different angles and illumination circumstances. Again, previous methods use hand-engineered features, such as convolutions of gradients in different directions, to produce these descriptions, but LIFT develops superior feature descriptors by learning them from a large dataset. [2] LIFT feature descriptors are trained by passing in two images of the same feature, viewed from different angles, a negative example that should demonstrate a difference from the features, and a "blank" with no features that serves as a negative example. [2] These end-to-end deep learning methods are connected by a series of differentiable functions to allow learning to be performed from the result all the way back to the input. [2] Being able to learn these feature methods allows far greater flexibility in matching features, such as matching the same location during two different seasons. [2] This would be nigh-impossible with hand-engineered features because gradients can't account for snow or trees losing their leaves, but LIFT can handle this by passing in correlated images specifically taken at different times throughout the year. It is the expectation of the authors of this paper that the greater, flexibility and versatility of deep-learned features will allow SLAM to perform even under new or difficult matching conditions.

C. LIFT-SLAM

LIFT-SLAM is a hybrid SLAM approach that ties together the two methods previously discussed in this paper (ORB-SLAM and LIFT features). [1] The implementation begins by training LIFT features on a black and white self-driving dataset. The original LIFT network was trained on a photo-tourism dataset which contained a variety of images taken from different cameras at varying angles, which is rather different from the types of data seen in self-driving applications. [1] For these applications, the camera is constant and merely shifts from one location to another, typically only rotating around the vertical axis. However, retraining LIFT on the same dataset used for validation of the LIFT-SLAM system may cause it to be over-fit and not able to generalize to new regions or climates. After training LIFT on their dataset, the feature matching capability of their network began to surpass that of ORB, but it should also be noted that they ran feature detection and mapping sequentially, instead of aiming for the online-capable, parallelized ORB-SLAM method. [1] Beyond the new LIFT features, the implementation is largely the same as that presented in the ORB-SLAM method section. Keyframes are saved, as our 3-d positions of LIFT features, and continual checking and culling of these keyframes minimizes the memory requirements of the system. For this particular implementation, they created a specific bag of words out of LIFT features for the loop closure method. [1]

In the original paper, the authors created multiple varieties of their system to explore the possibilities of what could be achieved. Their first method was called "Fine-tuned LIFT-SLAM" and was essentially LIFT-SLAM, except that the LIFT half was trained specifically on runs from the same dataset that the entire system was validated on. Their second method, "Adaptive LIFT-SLAM" adjusted the feature-matching threshold based on the current ratio of map points and outlier points. [1] The goal here was that they wanted to be as strict as possible in their feature matching, so as to prevent incorrect correspondences, while still ensuring that they didn't lose their current location. Adjusting the threshold accomplishes this by always ensuring that they have correspondences, but only as many as they need. The authors of this paper tested their system on both the KITTI and Euroc datasets and showed success on both of them. It is this paper that is being re-implemented here, along with certain changes and adjustments. [1]

III. Methods

We built up both SLAM implementations using a similar structure based on the ORB-SLAM implementation. In order to compare the LIFT vs ORB feature methods, we had to use the same robust SLAM base otherwise to ensure validity of juxtaposition. We pulled the tuned hyperparameters, bag of words vocabulary, and other basic SLAM functions from this basis. To properly represent the LIFT features [2], we pulled the pre-trained model from the paper to generate the feature space. The ORB features were pulled from the ORB-SLAM

implementation [4] to ensure accurate representation of the ORB-SLAM implementation in comparison.

A. LIFT Implementation

LIFT was implemented on Python with Theano and Lasagne framework library. The grayscale input image is first resized to the appropriate input size for LIFT, and then an image pyramid is created by processing the input images at different scales to capture features from multiple scales. This multi-scale pyramid is then passed into the LIFT network and the feature point detector extracts all of the points that the network deems potentially interesting. Non-max suppression is then applied to this output to find points that are both interesting and also spaced out throughout the given image. Without non-max suppression points of interest would collect all on the same landmark and it would be far more difficult for the SLAM system to perform estimation and loop closure, because only a few landmarks would collect points of interest. After this, the keypoints data is sent through a network to determine the orientation and then is encoded into a feature vector that can be matched to other, similar features. For the implementation in this paper, all three sections were run together.

After LIFT was implemented, the primary parameter that needed to be tuned was the feature distance threshold. The feature distance threshold represents how ”close” features need to be in order to be accepted as mapped features in the SLAM system. Initial attempts had a threshold that were far too low and matched random points in shadows and blue sky. Restricting the threshold improved the LIFT performance significantly. Additionally, SIFT features were briefly tried but produced poor results and so were abandoned and are not discussed further in this paper.

B. SLAM Implementation

The implementation of the SLAM backend of the hybrid system had few adjustments from the original ORB-SLAM method. On the practical side, the SLAM algorithm was adjusted to read features from the disk. This was for two reasons; first, the SLAM implementation was in C++ while the LIFT features were coded in Python. Secondly, the LIFT features took too long to compute and run in real time, so the features were pre-computed and stored on disk, from which the SLAM backend read them. This is similar to the original paper's implementation of LIFT-SLAM which also did not run online, but instead computed sections of the algorithm sequentially. [1]

C. Combined LIFT-SLAM Implementation

In order to integrate the LIFT features into the SLAM backend, we first transformed the LIFT feature vector to re-balance the distribution of values. This is necessary because to achieve real-time performance in the SLAM backend, descriptor vectors between two different feature points are computed using bits comparison instead of using floating point arithmetic. Therefore, feature descriptor vectors are compressed from its original 64-bit floating point data type to 8-bit unsigned integer, and LIFT features has a heavily skewed distribution as seen in Figure 1. This skew will result in sub-optimal feature matching as most values once it is compressed to 8-bit unsigned integer will be in the range of 251 to 255. To counter this we applied a deterministic transform to re-balance the distribution. The values are first linearly shifted and inverted its sign, then a logarithmic function is applied to it, and lastly the values are compressed to 8-bit unsigned integer. The resulting distribution has two Gaussian-like peaks as seen in Figure 2, this should not be an issue as the feature matching is agnostic to it. Additionally, LIFT

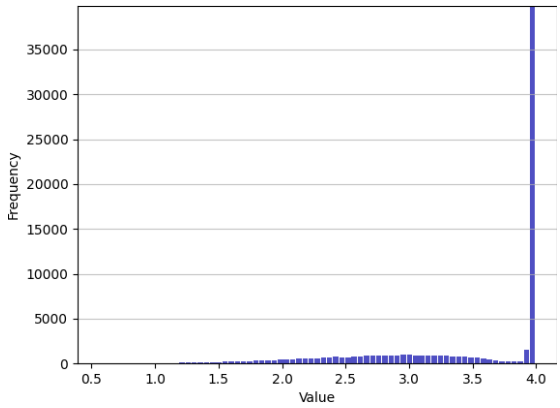

Fig. 1. Original LIFT feature vector distribution

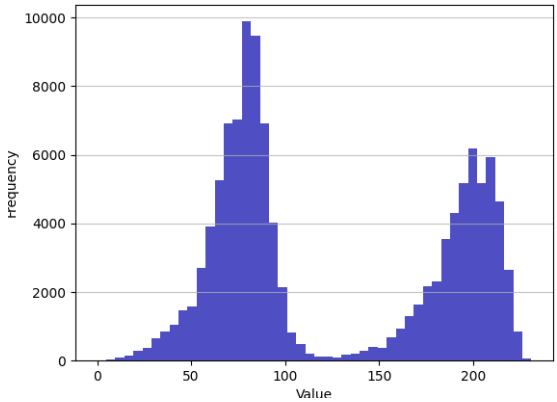

Fig. 2. Transformed LIFT feature vector distribution

feature vector is of length 128, while ORB has a feature vector of length 32. Therefore, we needed to customize the feature-matching function to take into account the difference in length, especially during the bits comparison.

D. Evaluation Metrics

To evaluate these SLAM results, we compared the euclidean distance between the ground truth poses and the poses generated by each SLAM implementation. We used both L1 distance and L2 distance in order to have more depth to these error metrics.

We evaluated the poses of both the results and ground truth by building the poses into the homogeneous transform format.

The ground truth poses for the KITTI data set are structured in a N x 12 table, where N is the number of frames of the sequence and each row represents the pose of the left camera coordinate system (i.e., z pointing forwards) via a 3x4 transformation matrix. To build this into a homogeneous transform matrix, each line was structured as follows:

$$\begin{bmatrix} gt[0] & gt[1] & gt[2] & gt[3] \\ gt[4] & gt[5] & gt[6] & gt[7] \\ gt[8] & gt[9] & gt[10] & gt[11] \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The results come in the form of Nx8, where N is the number of frames of the sequence and each row represents the pose of the left camera, similar to the ground truth, but then each row contains the timestamp (at the first index) and the pose in the form of the x, y, z position as the the first three pose indexes and the rotation in quaternion form in the last 4 pose indexes (q_w, q_x, q_y, q_z). Each row's was converted into the frame's pose homogeneous transform matrix as follows:

$$\begin{bmatrix} 2(q_w^2+q_x^2)-1 & 2(q_xq_y-q_wq_z) & 2(q_xq_z+q_wq_y) & x \\ 2(q_xq_y+q_wq_z) & 2(q_w^2+q_y^2)-1 & 2(q_yq_z-q_wq_x) & y \\ 2(q_xq_z-q_wq_y) & 2(q_yq_z+q_wq_x) & 2(q_w^2+q_z^2)-1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To compare each frame to ground truth, we first had to match all of the result key frames to the ground truth key frames based on which ground truth frame had the nearest timestamp to the timestamp on the result frame. Because of delay in initialization in both ORB-SLAM and LIFT-SLAM, the result frames are all offset since the first key frame is initialized to the identity pose so each key frame after that can be a pose transform from there. In order to properly compare each result key frame pose with the ground truth pose, a generalized homogeneous transform had to be calculated to transform the result key frames into the ground truth frame. This turned out to be equivalent to the ground truth transform matrix at the initialized timestamp because the initial result frame is just an identity transform. Therefore, all result matrices are transformed as follows:

$$pose_{result,transformed} = T_{match} * pose_{result}$$
$$\text{where}$$
$$T_{match} = pose_{groundtruth} @ \text{ initialization time}$$

One major drawback of monocular SLAM is a lack of scaling in distance. Because our metrics are based on euclidean distance between frames, this scaling issue significant impacts the evaluation error metrics. To adjust the distances in the result frames, we calculated the scaling factor by minimizing a linear equation (Ax=b) with the A matrix comprised of the stacked position (x,y,z) from the results frames and the b matrix comprised of the stacked position (x,y,z) from the paired ground truth frames, therefore solving for the scalar "x" as the scaling factor. We solved the Ax=b linear equation minimization using least-squares.

We then calculated the L1 norm and L2 norm for the distance between the results and the ground truth as representation of the error in the SLAM pose. The L1 norm was calculated using the A & b matrices from scaling factor calculation using the following formula:

$$\text{L1 norm} = mean(|Ax - b|)$$

The L2 norm was calculated using the pose from the match transformed result homogeneous matrix and the paired ground truth pose. After pulling out the positional terms (p_i for results and $\hat{p}_i$ for ground truth), the difference was normalized along the first axis to get the difference between each positional measurement, then was squared, then was averaged across the whole error vector (all frames in $I_{results}$), then was square rooted to give the L2 norm. This is represented in the following equation:

$$\text{L2 norm} = \sqrt{\frac{1}{|I_{results}|}\sum_{i \in I_{results}} \left\| T_{match} p_i - \hat{p}_i \right\|^2}$$

IV. Experiments

We developed our experiments such that the baseline performance of each SLAM implementation can be evaluated by isolated metrics in positional drift, rotational drift, initialization time, and correspondence matching as well as more binary success criteria such as loop closure. We use our objective evaluation metrics and more subjective visual analysis of the SLAM performance to compare the two implementations.

A. Dataset

Our experiments were conducted on the KITTI (Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago) dataset. We used the training data of the "Visual Odometry/SLAM Evaluation" benchmark dataset for training, testing, and evaluation. We specifically used the left camera images of the grayscale odometry dataset and compared it to the provided ground truth poses. Because the ground truth poses were only available for the training sequences (sequences 0 to 10), we were limited to using those sets as we needed ground truth poses to generate the euclidean errors in order to compare ORB-SLAM vs LIFT-SLAM. We highlighted two sequences specifically that show baseline performances of SLAM and could suitably represent the accuracy and success of SLAM in comparison.

The first selected was KITTI odometry sequence 4 because it was a straight line drive with some suburban features around the road mixed in with less feature heavy nature surroundings. This sequence best highlighted the euclidean error because it kepted the x and y terms of the position relatively constant to more easier identify the positional drift in one dimension.

The second selected was KITTI odometry sequence 6 because it was the simplest complete closed loop in the training dataset. This sequence was key to proving the actual functionality of each SLAM implementation in that it tested the loop closure. Because this sequence was a loop, it also highlighted positional drift, but also helped understand errors in rotation changes as well as testing overall success of the

implementation based on whether or not the implementation achieved proper loop closure.

B. Experiment #1: Straight Line Driving

The first experiment is to test the tracking capability of the LIFT and ORB in a simple scenario without large frame-to-frame changes and without much reuse of mapped key frames. Therefore, we can use this to isolate LIFT and ORB and not have to worry about other confounding factors from relocalization, loop-closure, or previously mapped key frames. After the additional work was done to improve our LIFT-SLAM implementation, we were able to get an overall lower L1 and L2 error rate as compared to ORB-SLAM. This is in line with the the original LIFT-SLAM paper achieved, and shows the LIFT performs better than ORB as feature detector and descriptor in the nominal case.

C. Experiment #2: Loop Closure

The loop closure test was run on a specific example from the KITTI dataset in which a car drives down a long, straight stretch of road, turns onto a parallel road facing back the other way, drives past the starting point, and then finally turns back onto the original road and passes its starting point, thereby forming at least one full loop, with some overlap. The purpose of training on this section of the dataset was to determine whether the LIFT-SLAM system could handle loop-closure scenarios. This is particularly difficult with monocular SLAM because there is no sense of depth scaling, so it is easy for the algorithm to drift significantly between the two halves of the loop. During testing on this experiment, LIFT-SLAM could perform loop closure, but ORB-SLAM was more consistent and closed the loop significantly faster. While ORB-SLAM had almost immediate loop closure, LIFT-SLAM was lost for many seconds before it was finally able to establish overlap and perform appropriate loop closure.

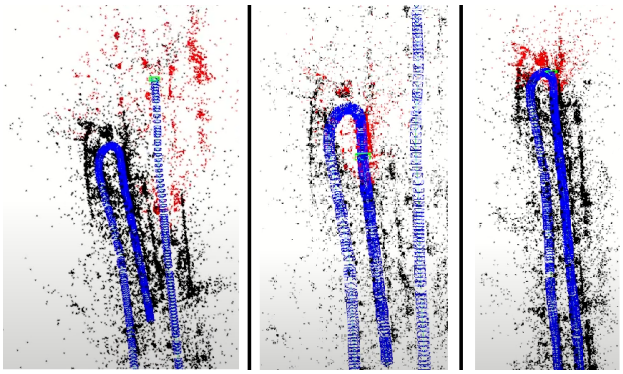

Fig. 3. Example of a Successful LIFT-SLAM Loop Closure

D. Overall Comparison

Based on our experiments, standard ORB-SLAM and LIFT-SLAM each win in certain categories. LIFT-SLAM seems better in our evaluation metrics, but ORB-SLAM showed better frame-to-frame correspondences, better loop closure, and could be run in real time.

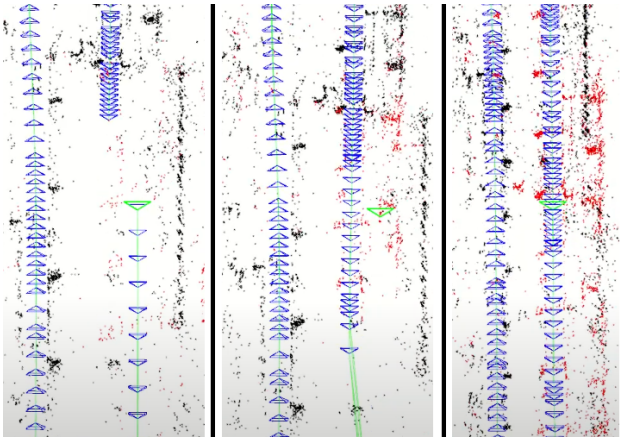

Fig. 4. Example of a Successful ORB-SLAM Loop Closure

The frame-by-frame matching with LIFT showed poorer matching quality with more erroneous matches and more unmatched points. A juxtaposition of each implementation's feature correspondences is shown in figure 5 and 6. These figures show that ORB has fewer noisy features (fewer dot detections) and better correspondences (more horizontal lines) compared to LIFT which has many unmatched features and many more incorrect feature matches. These feature issues could stem from problems in the training of LIFT features, as discussed in the Methods section, including being trained on a dataset of images aimed with the camera upwards and with no/slow camera movement connecting images as well as feature re-scaling corrupting the features themselves.

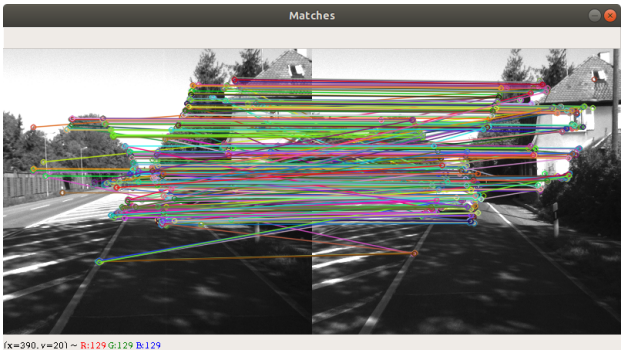

Fig. 5. ORB Feature Correspondences

This poor frame-to-frame tracking in LIFT causes the motion model to perform poorly, particularly due to errors in the angular estimates when the vehicle turned because of the poor tracking. This error manifested in overshooting the rotations and distance travelled.

Based on our objective evaluated metrics, LIFT actually outperformed ORB in both L1 norm and L2 norm euclidean distance error. Shown in figure 7, LIFT-SLAM had slightly less error than ORB-SLAM for sequence 4, with 0.49244 meters versus 0.85253 meters respectively for the L1 norm and 1.3311 meters versus 2.1171 meters respectively for the L2 norm,

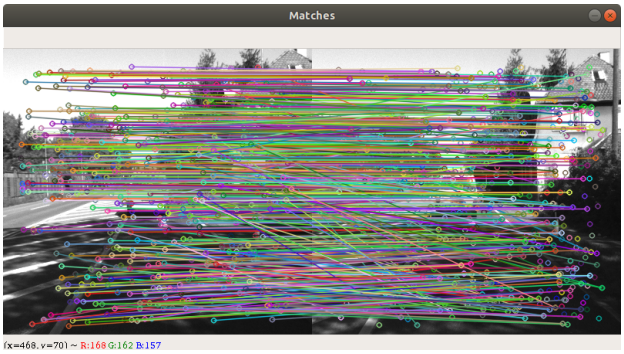

Fig. 6. LIFT Feature Correspondences

and for sequence 6, with 5.8965 meters versus 6.8919 meters respectively for the L1 norm and 20.325 meters versus 22.324 meters respectively for the L2 norm. LIFT-SLAM beat out

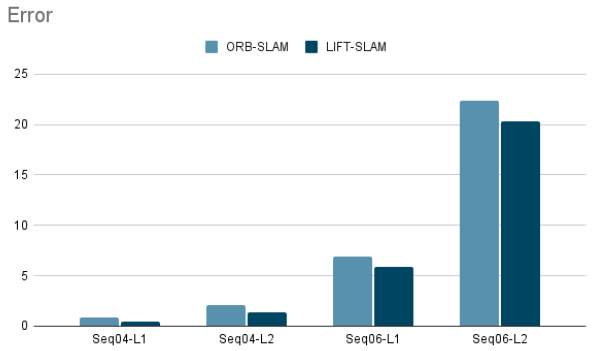

Fig. 7. L1 & L2 Norm Errors of ORB and LIFT SLAM

ORB-SLAM in these metrics despite having poorer frame-to-frame matching because the LIFT features gave opportunity to detect more unique features learned from the environment while ORB was stuck with merely geometrically interesting features, similar to that of corner detection. As seen in Figure 6, even though LIFT has more erroneous correspondences, it is able to obtain features from a wider variety of image patches which is impossible with classical methods, such as the features on the road surface or inside the trees that do not have a strong enough gradient to be used by ORB.

As discussed in experiment #2, ORB-SLAM performs loop closure better by converging back faster. LIFT-SLAM takes a hit on this because the angular overshoot on turns causes the position to be further off-course, making it harder to recognize the closed loop. This difference is significant because the longer it takes to close the loop, the longer it is lost on the map, which could prove dangerous for self-navigating robots.

An additional point to note is that LIFT could not be performed in real time while ORB could. LIFT features required pre-computation for the full sequence of images so that the SLAM method could run in real time because the feature detections themselves were too slow, even on a relatively powerful desktop computer. ORB feature detection could easily run in real time on all sequences, making it more realistically applicable to a SLAM application.

V. Conclusion

LIFT-SLAM proved to be better than ORB-SLAM in our objective monocular SLAM distance error evaluation metrics, but ORB-SLAM is still a superior SLAM implementation based on the other significant factors. ORB-SLAM has better loop closure, better frame-to-frame correspondences, and is faster to compute, making it a better fit for real-time SLAM applications. LIFT offers more unique features learned from the environment outside of the traditional geometrically interesting methods used for ORB, but this also gives way to more unmatched features and more erroneous matches. In order to improve the LIFT feature's performance, we could refactor it to train on full RGB images instead of grayscale to allow more depth to the recognized features. Major improvement could come from re-training the LIFT feature network on self-driving datasets, such as KITTI, so that it could learn features more common to the driving setting. However, we avoided this in our implementation because it seems to contradict one of the main purposes of SLAM, that it should be relatively environment invariant with similarly successful functionality. Having a system that requires pre-training on its environment as well as pre-computing the map features for the trajectory defeats the purpose of SLAM because it requires a quasi-map to be provided, essentially leaving only a need to localize. While there are still clear beneficial applications to deep-learning-based localization, LIFT features currently can't beat ORB in monocular SLAM for unknown, untrained environments.

For the benefit of the reader, we link the GitHub repository containing our code base: LIFT SLAM

Acknowledgment

This work was done for CMU's Robot Localization and Mapping course, class code 16-833, Spring 2022. This was accomplished with the support of Professor Michael Kaess and teaching assistants Dan McGann, Mosam Dabhi, Seungchan Kim, Rouyang Xu, and Arjun Teh.

References

[1] H. M. S. Bruno and E. L. Colombini, "LIFT-SLAM: a deep-learning feature-based monocular visual SLAM method," arxiv, 02-Jun-2021. [Online]. Available: https://arxiv.org/pdf/2104.00099.pdf. [Accessed: 20-Feb-2022].

[2] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua, "LIFT: Learned Invariant Feature Transform," EPFL, 29-Jul-2016. [Online]. Available: https://www.epfl.ch/labs/cvlab/research/descriptors-and-keypoints/lift/. [Accessed: 20-Feb-2022].

[3] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. The International Journal of Robotics Research, 32(11):1231–1237, 2013.

[4] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras," IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255–1262, 2017.