SIVO: Semantically Informed Visual Odometry and Mapping

by

Pranav Ganti

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2018

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Accurate localization is a requirement for any autonomous mobile robot. In recent years, cameras have proven to be a reliable, cheap, and effective sensor to achieve this goal. Visual simultaneous localization and mapping (SLAM) algorithms determine camera motion by tracking the motion of reference points from the scene. However, these references must be static, as well as viewpoint, scale, and rotation invariant in order to ensure accurate localization. This is especially paramount for long-term robot operation, where we require our references to be stable over long durations and also require careful point selection to maintain the runtime and storage complexity of the algorithm while the robot navigates through its environment.

In this thesis, we present SIVO (Semantically Informed Visual Odometry and Mapping), a novel feature selection method for visual SLAM which incorporates machine learning and neural network uncertainty into an information-theoretic approach to feature selection. The emergence of deep learning techniques has resulted in remarkable advances in scene understanding, and our method supplements traditional visual SLAM with this contextual knowledge. Our algorithm selects points which provide significant information to reduce the uncertainty of the state estimate while ensuring that the feature is detected to be a static object repeatedly, with a high confidence. This is done by evaluating the reduction in Shannon entropy between the current state entropy, and the joint entropy of the state given the addition of the new feature with the classification entropy of the feature from a Bayesian neural network.

Our method is evaluated against ORB_SLAM2 and the ground truth of the KITTI odometry dataset. Overall, SIVO performs comparably to ORB_SLAM2 (average of 0.17% translation error difference, 6.2×10^{-5}deg/m rotation error difference) while removing 69% of the map points on average. As the reference points selected are from static objects (building, traffic signs, etc.), the map generated using our algorithm is suitable for long-term localization.

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Steven L. Waslander, for his guidance, encouragement, and support throughout the development of this work. Thank you for giving me an opportunity to develop skills in a field I knew little about, and for allowing me to explore areas I was passionate about as I gained experience in this amazing field. My experience at the WAVELab has helped me grow immensely both personally and professionally.

Thanks to all of my colleagues at the WAVELab for making the last two years fun, even through the stress swirling around us. I would like to thank my awesome roommates, Ben, Leo, Jung, and Sean, for making sure we could never separate our work life from our home life. Thank you to Mike, Chris, and Nima for the fun conversations and advice over the countless journeys to get lunch and coffee. Thank you to Stan for convincing me to join the WAVELab and for the advice and guidance that followed. Lastly, I would like to thank Arun and Jason for not only sharing their SLAM secrets, but also for always being willing to take a break so that we could watch the Raptors lose.

I would like to thank Jordan and Jonathan for their help preparing the training data and testing various configurations of the neural network used in this project.

Thank you Rebecca, for your love and support from across the country. I could not have achieved this without your unconditional support and encouragement.

Lastly, I would like to thank my parents for, well, everything. Thank you for your immeasurable support, patience, guidance, dedication, love, encouragement, and advice. Thank you for all of the sacrifices you have made and for giving me the opportunity to pursue my goals and dreams.

Dedication

This thesis is dedicated to my parents.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Localization is a key problem in the field of autonomous mobile robotics. Accurate knowledge of a robot's location facilitates a variety of lower-level tasks, such as vehicle control, as well as higher level tasks, such as motion planning or object tracking. Accurate positioning information is also a matter of safety for autonomous cars, as localization accuracy has to be known on the order of centimetres in order to prevent collisions or maintain lane positioning. Although sensors such as a Global Positioning System (GPS) can provide localization information to the desired accuracy, there are numerous situations where this is not possible for autonomous driving. These GPS denied (or weakened) scenarios can occur in numerous different conditions, such as passing through a tunnel or under a bridge, or even in dense urban environments where skyscrapers and similar structures cause reflections of the GPS signal, resulting in multipath effects [1].

In recent years, visual odometry (VO) [2] has emerged as a reliable technique for vehicle localization through the use of cameras. By observing the apparent motion of distinct reference points, or features, in the scene, we can determine the motion of a camera through the environment. This method has also been extended to simultaneously estimate and store the 3D position of these tracked features in addition to the camera motion. This process is referred to as Visual Simultaneous Localization and Mapping (SLAM). The map generated by the SLAM algorithm can be used for long-term localization, providing the vehicle with known reference points if it returns to a pre-mapped area.

1.1 Motivation

In order to accurately track camera motion by observing features in the scene, it is imperative that "good" reference points are selected. This task is performed by the *front end* of the SLAM algorithm, which works to extract useful references and correspond them in multiple scenes to provide an initial estimate of camera motion. This information is then passed on to the *back end* of the SLAM algorithm, which is responsible for fusing together this information with additional sensor measurements such as wheel odometry, acceleration information from an Inertial Measurement Unit (IMU), or GPS pose measurements. The back end consists of an optimization pipeline and provides an estimated value for the camera pose. Although there are numerous accurate approaches for visual SLAM, this is still an open area of research; the emergence of machine learning has opened new avenues for SLAM, ranging from methods which add context and scene understanding to methods which use machine learning in an end-to-end approach, using only input images to generate a pose output.

Feature and keyframe selection for visual SLAM is an area of research that is still being explored. When selecting reference points for visual SLAM, these points should meet a number of criteria. Ideally, the selected features should be:

1. Viewpoint invariant
2. Scale invariant
3. Rotation invariant
4. Illumination invariant
5. Season invariant
6. Static

Traditional feature detectors and descriptors, such as SIFT [3], SURF [4], FAST [5], or ORB [6] aim to tackle the first 3 criteria, while appearance based methods such as FAB-Map [7] or SeqSLAM [8] aim to tackle criteria 4 and 5. Typically, visual SLAM algorithms depend on outlier rejection schemes such as RANSAC [9] to characterize an object as dynamic. In this case, the motion of the dynamic reference point would be an outlier compared to the motion of static objects, which *should* comprise the majority of the scene.

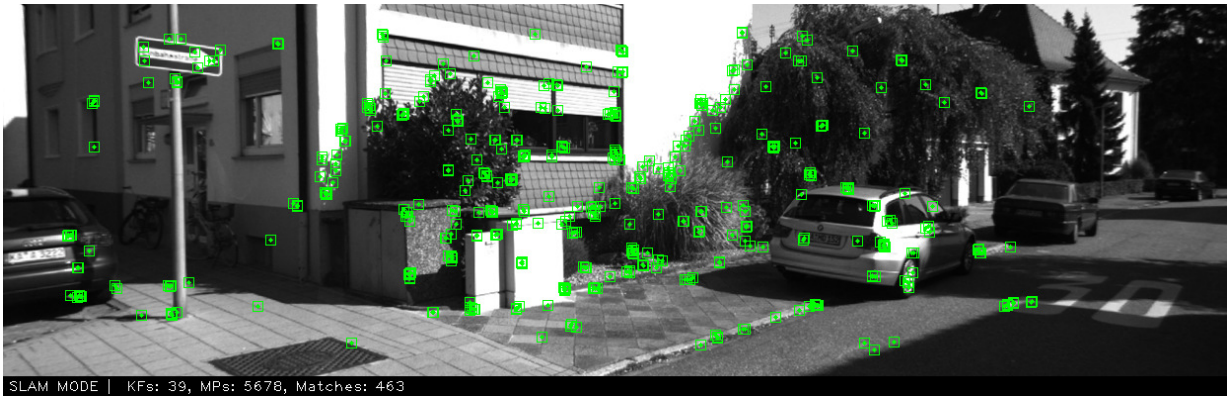


Figure 1.1: Sample visual features extracted from ORB_SLAM2 [10], on KITTI [11] dataset trajectory 00

Figure 1.1 illustrates typical features used by a visual SLAM algorithm. From the criteria, we see that there are some excellent points selected, but also some poor ones. The best reference points are most likely on the street sign or the corners of the building; these are extremely stable reference points that will be useful long-term references, and would only be modified in the event of major construction or vandalism. In contrast, the reference points on the car may be gone within the hour, and the features on foliage will no longer be present as the seasons change. Due to the emergence of deep learning in recent years, advances in scene understanding have paved the way for *context* to be incorporated into a visual SLAM algorithm to address our final criterion. This would allow us to dictate which references are more likely to be stable from our understanding of typical static and dynamic object behaviour in the scene.

This work mainly focuses on criteria 1, 2, 3, and 6, by supplementing traditional feature detectors with deep-learning based scene understanding. The incorporation of semantic information is not necessarily novel (as discussed in Section 1.2.2), however, the majority of methods to date do not emphasize the importance of network *uncertainty*, primarily because this concept is still an open area of research [12]. What could be the potential consequences of an incorrect answer, and how can we mitigate this effect? Sünderhauf et al. [13] recently discussed the limits and potential of deep learning in robotics applications, as robotics is a field that revolves around uncertainty. Sünderhauf states, "Robots have to perceive, decide, plan, and execute actions - all based on incomplete and uncertain knowledge" [13]. The probabilistic and uncertain nature of the SLAM problem allows it to be described as "...simply tracking a normal distribution through a large state space..." [14]. As we continue to develop our machine learning based methods, it is imperative

to understand how much *trust* we can place in the network, and use this uncertainty in our SLAM algorithm to facilitate robot decisions. This work proposes a novel front end algorithm which blends together aspects of machine learning, neural network uncertainty, and information theory in order to select better features for visual SLAM.

1.2 Related Works

While there is a wide catalogue of work concerned with feature selection, we will examine related approaches which use information-theoretic criteria to reduce the complexity of the algorithm, as well as approaches which incorporate semantic information to supplement a SLAM method.

1.2.1 Information-Theoretic Approaches to SLAM

Map point maintenance is crucial in order to maintain the runtime and storage complexity of a visual SLAM algorithm, and the use of information-theoretic methods have been prominent in achieving this goal. The main concept behind these techniques is to only select features, keyframes, or poses which maximize the *information gain* (see Section 2.7), therefore reducing the number of landmarks or poses in the optimization without appreciably compromising the accuracy of the SLAM solution. In essence, a variable which has low information is redundant, and is wasting valuable computational resources.

Information-Theoretic Feature Selection

One of the earliest information-theoretic approaches was proposed by Dissanayake et al. [15] who presented a method to reduce the computational burden of maintaining a large map by removing landmarks without affecting the statistical consistency of the estimation process. The runtime and map storage complexities of the SLAM problem scale to $O(N^3)$ and $O(N^2)$ respectively with respect to the number of landmarks. The authors propose a map management strategy which deletes the majority of landmarks from the map within a predetermined distance segment travelled by the robot. For each segment, the authors first identify the set of landmarks that changed states from invisible to visible, and then propose that only one landmark should be kept in the map: the landmark which has the maximum information content for the set. The information content of a landmark was determined by calculating the reciprocal of the trace of the covariance matrix for each

landmark variable. This method reduced the total number of landmarks by 8-fold, without significantly compromising estimation accuracy.

Hochdorfer and Schlegel [16] present a method to manage the continuously growing number of landmarks by building on the method proposed by Dissanayake [15]. In addition to quantifying the quality of a landmark, the authors propose that it is also relevant to take spatial position into account, and state that a landmark's benefit should be determined in terms of observability regions. The observability is introduced as the arithmetic mean of the observation positions for each measurement of the landmark. From here, landmarks within the same observation region are clustered with k-means clustering, and the landmarks' information content for each cluster is computed using the simple method proposed by Dissanayake [15]. The landmark with the lowest information content from the cluster with the largest spread of information is then removed in order to have the smallest degradation of localization quality. The authors argue that the cluster which has the largest difference of information content between its landmarks should cope best with the removal of a landmark.

Davison [17] proposes a method which uses Shannon information theory to evaluate the *quality* of a visual measurement with the aims of highlighting the location within an image to focus processing resources. This approach uses *mutual information* as its evaluation criteria (refer to Section 2.7.5). The variables of interest are the current pose and a particular landmark measurement; the landmark with the highest mutual information between the pose and itself will reduce the pose uncertainty the most. Once a landmark is selected, the pose estimate and covariance is updated and the landmark selection process is then repeated. As landmarks are selected, each new landmark will provide less information for the state estimate. Therefore, Davison selects features until the mutual information for the best feature drops below a certain threshold.

Zhang et al. [18] propose an entropy-based approach to select the best visual features in a scene. This approach formulates feature selection as an optimization problem which maximizes the amount of information acquired by minimizing the entropy. The authors first incorporate each new measurement into the *a posteriori* probability density function (PDF) of the system state, and then calculate the entropy of the distribution based on the updated covariance matrix. This step is repeated for all new landmarks in a scene. Once the updated entropy is calculated, the entropy difference for each feature in the map is calculated, and features are only selected if they lead to a sufficient information gain. Overall, this system showed considerable speedup over the naive solution which used all available features when tested both in simulation and experimentally.

Kaess and Dellaert [19] investigate efficient recovery of marginal covariance matrices for

data association and landmark selection. The method presented in their work recovers the pertinent parts of the covariance matrix efficiently by exploiting the sparsity and structure of the square root information matrix. While the majority of the paper focuses on data association techniques, the authors show that selecting informative measurements also requires extraction of the marginal covariance, as one method to make this selection is through investigation of the mutual information between the original state estimate and the state estimate with the addition of a single landmark, similar to the work of Davison [17]. The authors analyze the added information of each measurement using the "predicted" estimate, as they wish to know which landmarks provide the most information prior to actually taking the estimate. This method takes care of redundant features, and the authors use a threshold of 2 bits to deem a landmark as not informative enough.

Choudhary et al. [20] present two methods to reduce the number of landmarks: an active minimization approach, and an incremental approach which can be used online. This is done by minimizing a tradeoff between the memory requirement and estimation error. The subset of landmarks and poses is obtained by actively removing the least informative landmark until the desired objective function is minimized; any poses which do not see a landmark are then marginalized. The active minimization approach is effectively a batch method, which removes the least informative landmark during each iteration. The incremental method performs this active minimization every N poses, which should allow this to run in real-time. However, the incremental approach marginalizes out the remaining poses. Therefore, if the robot returns to an area where there were informative landmarks in the past, these can never be added back, leading to a slightly higher error than the active minimization approach.

Information-Theoretic Keyframe Selection and Pose Reduction

While this work looks to reduce the number of landmarks, it would be remiss to not discuss information-theoretic SLAM approaches beyond feature selection. The main goal of using information theory is to reduce the total computational complexity, and there are methods which do this by reducing the number of poses in the optimization.

Das and Waslander [21] propose an entropy-based approach to keyframe selection, as traditional keyframe selection is mostly heuristic driven. The authors propose to add keyframes when the current variance of any of the pose values $(x, y, z, \text{roll}(\phi), \text{pitch}(\psi), \text{yaw}(\theta))$ is above a user specified threshold. The entropy reduction method, Cumulative Point Entropy Reduction (CPER), evaluates the predicted covariance of the state estimate after adding each keypoint in the scene. The predicted covariance can be very quickly calculated, and the authors evaluate the uncertainty reduction for each keyframe by comparing

the difference in entropy between the prior and new predicted covariance. The multi-frame (keyframes at an instance in time for each camera in a multi-camera application) which maximizes this difference is the one selected as the next multi-frame. Overall, this method added significantly fewer keyframes in comparison to both a distance-based or point-overlap-based keyframe selection method, and resulted in an increased localization accuracy.

Ila et al. [22] present an information-theoretic approach to pose SLAM. As opposed to full SLAM, pose SLAM only estimates the robot trajectory instead of both the trajectory and landmark positions. The authors look to reduce complexity in their approach by only incorporating non-redundant and informative poses, which shifts the computational bottleneck from state recovery to data association. The authors consider a pose redundant if it is too close to another pose already included in the optimization, or the addition of this pose does not provide sufficient information. The authors also evaluate the mutual information, which measures the amount of uncertainty removed from the state when the link between poses is used. If this information gain is above a specific threshold and the pose is non-redundant, then it is added to the optimization. Their approach was tested on various trajectories, where it significantly reduced the number of poses used in the optimization and the number of loop closure links, therefore reducing the execution time. The authors found that this did not appreciably affect the consistency of the solution, as 99% of the sampled Monte Carlo trajectory simulations ended up inside the 95% confidence interval of the last pose covariance.

Information-theoretic approaches have proven to be quite effective in reducing the complexity of the SLAM problem. This is a promising direction which can be further supplemented through the use of semantic information.

1.2.2 Semantic SLAM

The idea to incorporate semantic information into the visual SLAM formulation is not novel. The emergence of deep learning based approaches has resulted in increasingly accurate methods for extracting semantic information from images, and are similarly becoming easy to integrate into a SLAM algorithm. The following works discussed are not exhaustive, but will provide an idea of some of the approaches to date.

An early approach to incorporate semantic information was SLAM++, proposed by Salas-Moreno et al. [23]. In contrast to most SLAM work, which uses low-level primitive features such as points, lines, or patches for localization, SLAM++ observes features at the level of objects by accounting for domain knowledge of the SLAM environment. The

authors first generate a 3D object database by scanning various common objects in their environment such as tables, chairs, and desks using KinectFusion [24]. The SLAM formulation is a graph-based approach, where each node represents the historical pose of the camera or the pose of an object in the scene. Pose measurements of visible objects are calculated using a 6 degree of freedom (DoF) object recognition algorithm, and are stored as binary factors in the graph linking together one camera pose and one object pose. Camera to camera constraints are computed using Iterative Closest Point (ICP) [25], and are also added as constraints to the graph. SLAM++ also includes a relocalization mode and loop closure detection, and this algorithm is also suitable for large scale mapping and detecting if a pre-mapped object has since moved.

Bowman et al. [26] formulate their SLAM problem to include inertial, geometric, and semantic constraints into a joint optimization framework. In most SLAM literature, a "hard" data association decision is made to indicate that a feature was observed at a certain pose. Correspondence using advanced feature descriptors such as SIFT [3] or ORB [6] will generally provide accurate answers to the data association problem, however an incorrect or ambiguous data association could yield significant issues depending on the optimization robustness. To bypass this issue, the authors consider the entire density of all possible data associations when estimating the state and landmark values and use expectation maximization (EM) to determine the optimal data association. This density acts as a weighting factor, effectively "averaging" over all of these associations. The semantic factor proposed by the authors has three separate components: the class, the confidence for the class detection, and the bounding box generated by a deformable parts model (DPM) object detector. The total measurement likelihood for the semantic factor is decomposed as the product of the probabilities for these three separate components, with the probability of the class corresponding to the confusion matrix of the object detector, and the probability of the bounding box assumed to be normally distributed with mean as the object centroid and covariance proportional to the bounding box dimensions. The EM algorithm now iteratively solves for the data association in the *expectation* step, while the sensor states and landmark positions are solved for in the *maximization* step. The authors conduct their own indoor tests using an indoor positioning system (IPS), and also test their algorithm on KITTI [11] odometry sequences 5 and 6. This work achieves better results than ORB_SLAM mono [27] or VISO2 [28], but not ORB_SLAM2 [10] stereo.

An et al. [29] propose a VO pipeline which incorporates aspects of both indirect and direct SLAM methods as well as semantic information to reduce the effect of dynamic objects in the scene on the SLAM solution. While methods such as RANSAC [9] or preemptive RANSAC [30] are often used to remove outliers, this fails to exploit prior knowledge of class stability and depends on the quality of the RANSAC. This method proposes to weigh

features with different contributions depending on their semantic category. The authors first semantically segment the scene using SegNet [31], and then weigh the contribution of a particular category by calculating the total reprojection error for the pixels of the category between images using the essential matrix. This probability is incorporated into the matching step as a weighting factor for RANSAC selection. Feature classes with a lower reprojection error are more likely to be selected; this leads to a higher weighting of static objects, as dynamic objects will most likely have a higher reprojection error. The total optimization cost has contributions from a direct and indirect pipeline, blended together via a tunable parameter. The selected points from the RANSAC algorithm comprise the reprojection cost, while the direct framework looks to minimize a photometric error for pixel patches. For a pixel to be selected for the direct pipeline, the authors select points which meet 3 criteria: the points have to belong to a planar surface, be motionless, and the depth from these patches should be easily estimated. As a result, only pixel patches from the road marking, road, and pavement categories were selected, although the authors note that road pixels were not very good, as most feature information for this category comes from shadows. The pixel patches for this section were selected from a bird's eye view (BEV) projected image, and then matched between consecutive frames. Overall, this method provided lower estimation errors than VISO2 [28], DSO [32], and ORB_SLAM [10].

Murali et al. [33] extend a custom map builder and GPS-denied localization method [34], as well as the localization functionality of ORB_SLAM2 to use semantic scene information. The semantic segmentation is done using a low-rank version of SegNet [31] trained on the CamVid [35, 36] dataset. Each detected landmark is assigned a class from one of 12 labels (sky, building, pole, road marking, road, pavement, tree, sign symbol, fence, vehicle, pedestrian, and bike). Based on traditional feature descriptor matching, the labels for all feature correspondences are summed, and a condition variable is set using the final counts. This condition variable defines whether the landmark is a "valid" semantic class for localization. A class is deemed to be invalid if the class is a temporal object (car, bike, pedestrian), or too far away (sky, road). This condition variable is then incorporated into the factor graph formulation; factors connecting landmarks and poses are gated by the variable, and landmarks are only selected if this condition variable is true. This feature selection scheme is only used for mapping, while all features are incorporated for visual odometry. Final results illustrate an improvement in 3D root-mean-squared error (RMSE) by approximately 20% over both their custom method and ORB_SLAM when they incorporate this semantic information.

Li and Belaroussi [37] present a real-time method for 3D mapping, which incorporates semantic labels into LSD-SLAM [38]. The main contributions are the 2D-3D transfer of semantic information via correspondence between connected keyframes, as well as map

regularization in the semi-dense framework. The authors use DeepLab [39] as their semantic segmentation network, and only run the network on selected keyframes in order to save computational resources. The main claim is that 2D semantic segmentation may not provide the same result of the same patch of pixels over multiple keyframes. In order to manoeuvre around this issue, the authors propose a method to incrementally fuse together the semantic label information in a Bayesian manner, which then allows them to label a 3D point based on the information from all previous keyframes. The second contribution is semi-dense map regularisation, which uses a dense Conditional Random Field (CRF) [40] to smooth the labelling between keyframes, incorporating contextual information. The CRF minimizes the Gibbs energy, which uses a negative logarithm of label probability for the unary potential, and uses a semantic score-related kernel for the pairwise potential.

Semantic information can be incorporated into a SLAM algorithm in a variety of different ways. While the previously discussed work looked to use semantics for object or pixel classification, Schönberger et al. [41] propose a method which blends together semantics with an *appearance* based method for relocalization and place recognition. Through the use of a variational autoencoder (VAE) [42], the method generates a novel descriptor for each scene which combines geometric and semantic information. The input to the VAE is an incomplete 3D segmented volume of the scene, extracted using stereo images. The encoder section of the VAE reduces the input into a descriptor representation, and the decoder then attempts to convert the descriptor into a *completed* 3D segmented volume. In essence, the descriptor is learning an encoding function that jointly encodes the scene semantics and geometry. The output segmented volume is then compared against the ground truth segmentation during network training. As the network trains, the descriptor will encode more information such that the encoder can reconstruct obscured parts of the subvolume. Once the network is trained, the descriptor generated for new segmented images can then be used for relocalization or place recognition.

There are several other efforts to incorporate semantic information into a SLAM algorithm. Stenborg et al. [43] use only the 3D location of a feature and its semantic label as a descriptor, and use a particle filter in order to bypass the use of traditional feature detectors. Radwan et al. [44] propose a novel architecture to jointly estimate odometry, camera global pose, and semantics using multitask learning. Reddy et al. [45] emphasize the importance of separating the static and aspects of a scene; through semantic segmentation and optical flow, the authors propose a method to 3D reconstruct the static and dynamic aspects of the scene separately, and fuse this information together for localization. As our knowledge of deep learning techniques continues to grow, there undoubtedly will be more methods proposed which looks to use this information to supplement visual SLAM.

1.3 Statement of Contributions

The works presented in Section 1.2 considered various difficulties involved with long-term visual localization. Information-theoretic approaches manage the number of variables added to the optimization pipeline, in order to maintain real-time operation of the SLAM algorithm as the robot continues to navigate through its environment. However, these methods do not incorporate semantic information into the formulation, which in turn does not guarantee long-term stability of the created map. While the rapid advancement of deep learning has allowed for the development of semantic SLAM algorithms, the work to date treats network output as deterministic, and does not consider network uncertainty in the formulation. Uncertainty is a key component of SLAM, and as we work to incorporate deep learning methods into our SLAM algorithm, it is important to consider the amount of trust we can place in the network.

The first contribution of this thesis is SIVO, a novel feature selection algorithm for visual simultaneous localization and mapping, specifically for an autonomous driving application. Our algorithm incorporates context into the visual SLAM formulation by semantically segmenting the scene using a Bayesian neural network and fusing together the network uncertainty into an information-theoretic approach to feature selection. This methodology creates a sparse map for long-term visual SLAM, by selecting features which provide significant information to reduce the uncertainty of the state estimate and ensuring the feature is repeatedly classified as a static object with a high confidence. This is done by evaluating the reduction in Shannon entropy between the current state entropy and the joint entropy of the state given the addition of the new feature with the classification entropy of the feature.

The second contribution is our implementation on Github[1]. The algorithm builds off of the ORB_SLAM2 [10] repository, and also contains a custom C++ implementation of Bayesian SegNet [46] built using Caffe's [47] C++ API.

Lastly, we compare the localization performance of our algorithm to the results of ORB_SLAM2 on the KITTI [11] odometry dataset, and compare the total number of keyframes and map points used, as well as the translation and rotation error over the course of the trajectory. Overall, SIVO performed comparably to the state-of-the art (average of 0.17% translation error difference, 6.2×10^{-5}deg/m rotation error difference) while removing 69% of the map points on average. The map generated by the algorithm can facilitate long-term localization of a robot, due to its sparsity and selection of static reference points.

[1] https://www.github.com/navganti/SIVO

1.4 Thesis Organization

The remainder of this thesis will be organized as follows. Chapter 2 will contain the background knowledge required to formulate the SIVO feature selection algorithm, including: notation, Gaussian processes, computer vision, semantic segmentation using deep networks, uncertainty in neural networks, information theory, and a summary of the ORB_SLAM2 algorithm. Chapter 3 will outline the novel feature selection policy, and is the main contribution of this thesis. Chapter 4 will show the results of SIVO in comparison to ORB_SLAM2 [10] on a selection of the KITTI [11] odometry trajectories, and also discuss the notable results. Lastly, Chapter 5 will conclude the thesis and examine potential improvements to the algorithm as future work.

Chapter 2

Background

This section will discuss the background material for SIVO.

2.1 Notation

2.1.1 Coordinate Frames

The main goal of our SLAM algorithm is to track the pose of a camera through time. Therefore, let us begin by denoting our conventions for coordinate frames following the notation proposed by Furgale [48]. We define a coordinate frame with origin at the point A, as $\mathcal{F}_A$. This is a Cartesian coordinate frame which follows the right-hand convention.

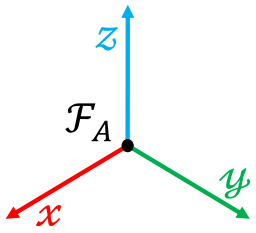


Figure 2.1: Coordinate frame notation example. Image adapted from [48].

Vectors express relations between two coordinate frames, such as a displacement, velocity, or angular velocity of one frame with respect to another frame. These quantities need three decorators in order to be fully defined. In this work, all vectors and matrices will be

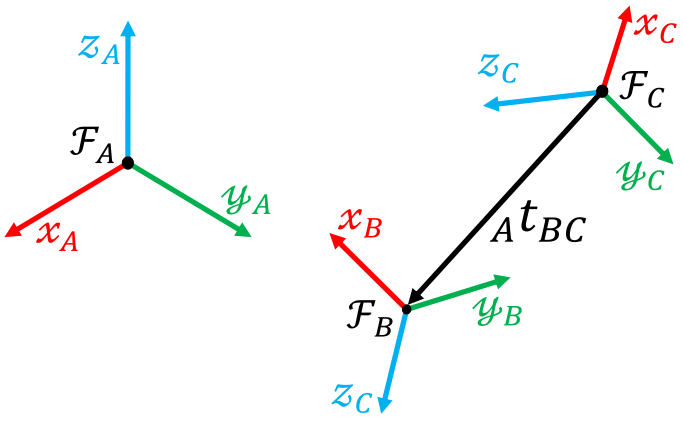


Figure 2.2: Vector ${}_A\mathbf{t}_{BC}$ illustrating the *displacement* of frame $\mathcal{F}_C$, with respect to frame $\mathcal{F}_B$, expressed in frame $\mathcal{F}_A$. Image adapted from [48].

bold. We require three decorators in order to describe which frame the vector quantity affects, the frame with which this quantity is with *respect* to, and the frame in which this quantity is *expressed*. Therefore, ${}_A\mathbf{t}_{BC} \in \mathbb{R}^3$ (Figure 2.2) illustrates the displacement (or translation) of frame $\mathcal{F}_C$ with respect to frame $\mathcal{F}_B$, expressed in coordinate frame $\mathcal{F}_A$. In contrast, *rotations* or *transformations* only require two decorators, as they express relative operations between frames. Rotation matrices are elements of the Special Orthogonal Lie group in 3 dimensions, $\mathbb{SO}(3)$ [49]. The variable $\mathbf{R}_{AB} \in \mathbb{SO}(3)$ can be described in two

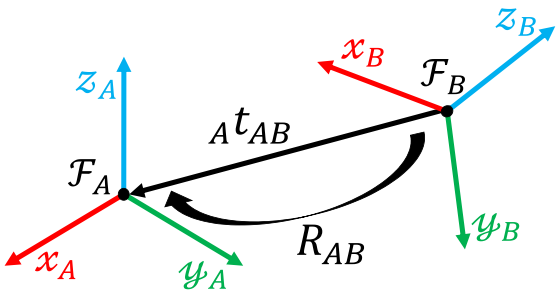


Figure 2.3: Notation and decorators for frame rotation. Image adapted from [48].

ways. It indicates the *rotation* of $\mathcal{F}_B$ with respect to $\mathcal{F}_A$, but can also be described to "rotate vectors *from* $\mathcal{F}_B$ *into* $\mathcal{F}_A$" [48]. The use of these decorators allows us to easily express the rotation of a vector into another frame, while verifying that our calculations are correctly propagated through the kinematic chain.

$$ {}_A\mathbf{t}_{BC} = \mathbf{R}_{AB}({}_B\mathbf{t}_{BC}) \tag{2.1}$$

In Figure 2.3, we notice that there is both a translation and rotational component in order to express the position of $\mathcal{F}_B$ with respect to $\mathcal{F}_A$. This can be expressed via a *transformation* matrix, which is an element of the Special Euclidean group in 3 dimensions, $\mathbb{SE}(3)$. The transformation matrix is a 4×4 matrix which contains the rotation and

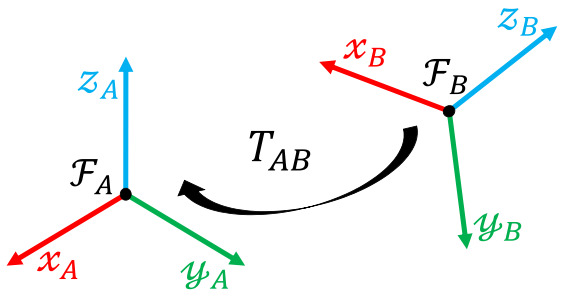


Figure 2.4: Notation and decorators for frame transformation. Image adapted from [48].

translation aspects as such:

$$\mathbf{T}_{AB} = \left[\begin{array}{c|c} \mathbf{R}_{AB} & {}_A\mathbf{t}_{AB} \\ \hline \mathbf{0}^{1\times 3} & 1 \end{array}\right] \in \mathbb{SE}(3) \tag{2.2}$$

In a similar manner to rotations, transformations can describe the *pose* of $\mathcal{F}_B$ with respect to $\mathcal{F}_A$, or express an operation which "transforms a point from $\mathcal{F}_B$ into $\mathcal{F}_A$" [48]. The transformation chain would be expressed as follows:

$${}_A\mathbf{t}_{AC} = \mathbf{T}_{AB}({}_B\mathbf{t}_{BC}) = \mathbf{R}_{AB}\mathbf{t}_{BC} + {}_A\mathbf{t}_{AB} \tag{2.3}$$

Note that the transformation operation actually modifies two decorators: the prescript, and the first subscript. This is due to the effect of rotation and translation on the kinematic chain.

As per Furgale's notation, we can simplify our decorators when discussing *points*. Let us imagine we have a point $\mathbf{p}$, expressed in frame $\mathcal{F}_A$. While we can express this point using the translation ${}_A\mathbf{t}_{A\mathbf{p}}$ as shown in Figure 2.5, the decorators are redundant and not very informative. Therefore, Furgale proposes the following notation

$${}_A\mathbf{t}_{A\mathbf{p}} = {}_A\mathbf{p} \tag{2.4}$$

which illustrates the frame the point is expressed in as the prescript. This also holds for indicating rotations or transformations,

$${}_A\mathbf{p} = \mathbf{T}_{AB}{}_B\mathbf{p} \tag{2.5}$$

which illustrates the transformation of the point from frame $\mathcal{F}_B$ into $\mathcal{F}_A$.

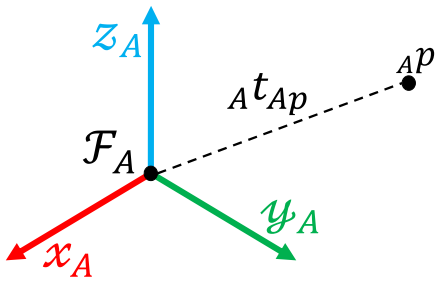


Figure 2.5: Notation and decorators for a point expressed in a frame. Image adapted from [48].

2.1.2 Homogeneous Coordinates

In theory, the point ${}_A\mathbf{p} \in \mathbb{R}^3$. However, in order to use this with transformation matrices $\mathbf{T} \in \mathbb{SE}(3)$, we require ${}_A\mathbf{p} \in \mathbb{R}^4$. This is done through the use of *homogeneous coordinates*. Homogeneous coordinates $\in \mathbb{R}^N$ are expressed as a vector $\in \mathbb{R}^{N+1}$ by placing a 1 as the last element. For example, let us define a point as $\mathbf{p} = (x, y)^T \in \mathbb{R}^2$. In homogeneous coordinates, we denote this point as $\mathbf{p} = (x, y, 1)^T \in \mathbb{R}^3$, and say that this represents the same point. The main benefit of using homogeneous coordinates in computer vision is that we can easily define points at infinity by $\mathbf{p} = (x, y, 0)^T$. Throughout this work, we will use the same notation ${}_A\mathbf{p}$ to describe homogeneous coordinates and inhomogeneous coordinates, depending on the equation.

2.2 Gaussian Distributions and Gaussian Processes

SLAM is effectively the process of tracking a Gaussian distribution through time. This section will introduce our notation for Gaussian variables and Gaussian processes. For further background on probability, please see Appendix A.

2.2.1 Gaussian Random Variables

A Gaussian, or normal distribution, describes a variable whose pdf describes a bell curve. This is expressed by Equation 2.6 in the univariate case.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.6}$$

In Equation 2.6, $\mu \in \mathbb{R}$ is the *mean* of the distribution, and $\sigma^2 \in \mathbb{R}$ represents the variance. For the multivariate case, the pdf is expressed by Equation 2.7

$$p(\mathbf{x}) = \frac{1}{\sqrt{2\pi\boldsymbol{\Sigma}}} \exp^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})} \tag{2.7}$$

In the multivariate case, $\boldsymbol{\mu} \in \mathbb{R}^N$ represents the multivariate mean, while $\boldsymbol{\Sigma} \in \mathbb{R}^{N\times N}$ represents the covariance matrix. An example 2D multivariate normal distribution can be seen in Figure 2.6; the center of the ellipse represents the mean, while the covariance matrix describes the parameters of the ellipse itself. The sides of the image are bell curves illustrating the univariate Gaussian distributions for each variable.

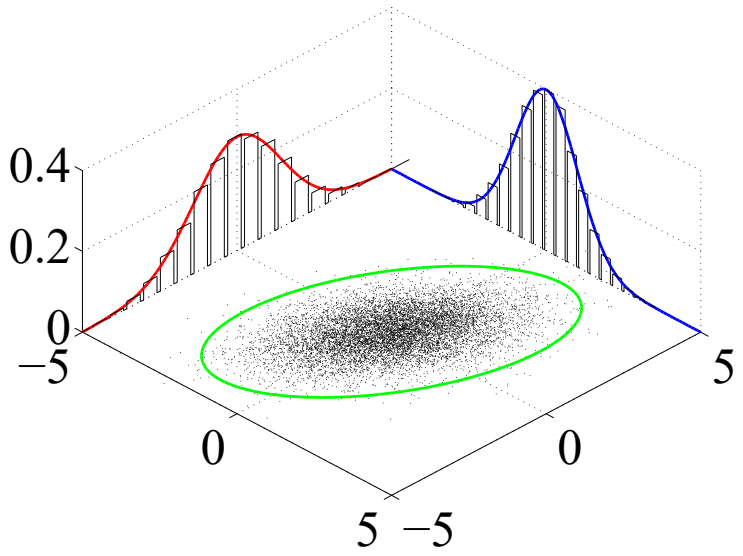


Figure 2.6: Illustration of a 2D multivariate Gaussian distribution.[1]

Gaussian variables play a key part in this work. For SLAM, we typically make the assumption that all measurements are normally distributed with a mean measurement value and noise covariance to illustrate the uncertainty in the measurement. Throughout this work, we will denote Gaussian variables as

$$\begin{aligned} x &\sim \mathcal{N}(\mu, \sigma^2) \\ \mathbf{x} &\sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \end{aligned} \tag{2.8}$$

[1]Image retrieved from: https://upload.wikimedia.org/wikipedia/commons/9/95/Multivariate_normal_sample.svg on 2018/07/27.

Note on Marginalization

An important concept in probability which briefly appears in this work is the idea of *marginalization*. The marginal likelihood of a multivariate Gaussian variable represents the distribution of a *subset* of the variables. This is illustrated in Figure 2.6; as discussed above the green ellipse and point distribution represents the multivariate distribution, however we see the individual bell curves of each variable illustrated on the side in blue and red. These univariate distributions represent the *marginal distribution* of the individual variables which comprise the 2D multivariate Gaussian distribution.

2.2.2 Gaussian Processes

While we have introduced the idea of a Gaussian variable, this work requires some background knowledge about the Gaussian Process (GP). A GP can be succinctly described as a distribution over functions [50]. While a discrete Gaussian variable can be characterized by its mean and covariance, we require a mean and covariance *function* to characterize a GP [51]. Let us denote the mean function as $\boldsymbol{\mu}(t)$, and the covariance function as $\boldsymbol{\Sigma}(t, t')$. We denote the entire GP as

$$\mathbf{x} \sim \mathcal{GP}(\boldsymbol{\mu}(t), \boldsymbol{\Sigma}(t, t')) \tag{2.9}$$

The covariance function controls function *smoothness*, and indicates the correlation between any two instances in time, t and t' [51]. Figure 2.7 illustrates a visualization of a 2D GP. At any instance in time, a variable of the GP will just be a simple Gaussian random

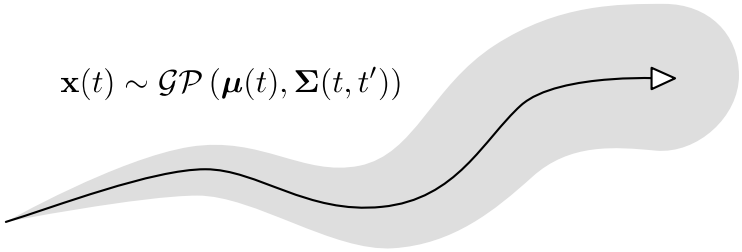


Figure 2.7: Illustration of a 2D Gaussian process. Image retrieved from Barfoot [51].

variable. For example, at time τ

$$\begin{aligned} \mathbf{x}(t) &\sim \mathcal{GP}(\boldsymbol{\mu}(t), \boldsymbol{\Sigma}(t, t')) \\ \mathbf{x}(\tau) &\sim \mathcal{N}(\boldsymbol{\mu}(\tau), \boldsymbol{\Sigma}(\tau, \tau)) \end{aligned} \tag{2.10}$$

the mean function will yield a mean value, and the covariance function becomes a simple covariance matrix, as seen before. In essence, we have marginalized out all other instances of time, and are observing the distribution at a single point [51].

2.3 Computer Vision

This work uses a camera as its major exteroceptive sensor. This section will discuss some fundamentals and techniques used in computer vision.

2.3.1 Camera Projection

A camera transforms points from 3D to 2D. Through a camera, our physical world is represented as blue, green, and red (BGR) pixel values in a 2D image. Figure 2.8 illustrates

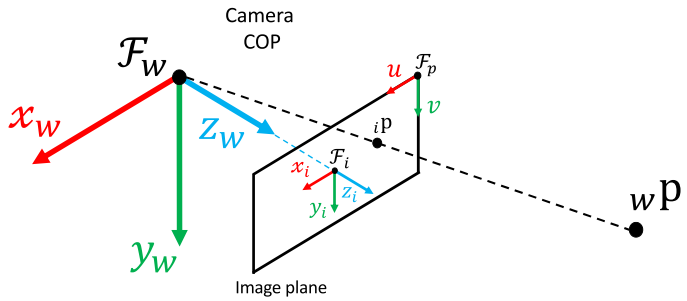


Figure 2.8: Illustration of a camera projecting a 3D point in the world frame, ${}_\mathbf{w}\mathbf{p}$, to the image frame, ${}_\mathbf{i}\mathbf{p}$. Image adapted from Hartley and Zisserman [52].

this process, which is called *central projection* [52]. In this example, the *world* frame, $\mathcal{F}_w$, is expressed at the camera's *center of projection* (COP), and the image frame, $\mathcal{F}_i$, is located at the image plane. A ray from a 3D point passes through the camera COP, intersecting the image plane. As the image plane consists of the pixels, the intersection of the ray with the image plane is the 2D representation of the 3D point that we can use for our algorithms. In the top right corner of the image plane is another coordinate frame which is the standard for expressing pixels, and is pixel $(0, 0)$. **Note:** in reality, the image plane actually lies *behind* the COP, and the image is inverted. For the sake of convenience, it is common to place the image plane in front of the COP.

Figure 2.9 shows a top-down view of the camera projection process. We see that the location of the projected point can be easily determined through similar triangles. Here, f represents the camera's *focal length*. If we represent the point in the world frame with

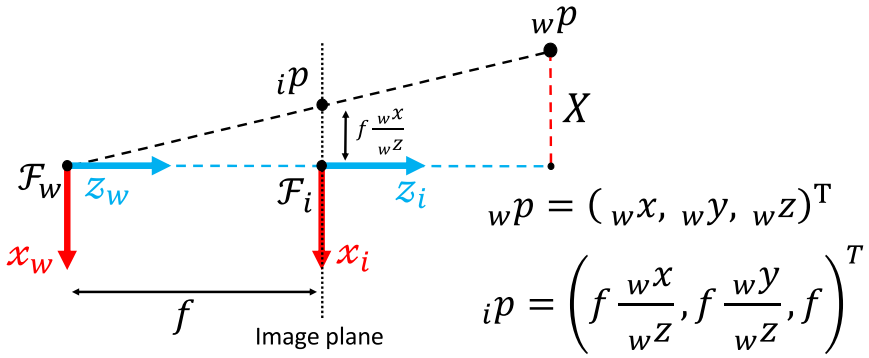


Figure 2.9: Top view of the camera projection process from Figure 2.8. Image adapted from Hartley and Zisserman [52].

$$ {}_w\mathbf{p} = ({}_wx, {}_wy, {}_wz) \tag{2.11} $$

it follows through similar triangles that the x and y coordinates of the projected point are scaled by the ratio $\frac{f}{Z}$.

$$ {}_i\mathbf{p} = \left(f\frac{{}_wx}{{}_wz}, f\frac{{}_wy}{{}_wz}, f \right) \tag{2.12} $$

As we are only interested in the x and y coordinate on the image plane, we can drop the final coordinate, yielding the mapping

$$ ({}_wx, {}_wy, {}_wz)^T \mapsto \left(f\frac{{}_wx}{{}_wz}, f\frac{{}_wy}{{}_wz} \right)^T \tag{2.13} $$

This mapping can be easily defined using the projection function, π

$$ {}_i\mathbf{p} = \pi({}_w\mathbf{p}) = \pi \begin{pmatrix} {}_wx \\ {}_wy \\ {}_wz \end{pmatrix} = \begin{pmatrix} f_x \frac{{}_wx}{{}_wz} \\ f_y \frac{{}_wy}{{}_wz} \end{pmatrix} \tag{2.14} $$

The above mapping illustrates the scenario where we have placed our world coordinate frame at the COP. However, typically we express our pixel values as per the coordinate

frame $\mathcal{F}_p$ in Figure 2.8, which is in discrete pixel values. This means we need to shift the values to the new origin and convert the focal length from metres to pixels. Ideally, the horizontal (x) and vertical (y) focal lengths would be the same value when converted to pixels, but this is not typically the case due to imperfections in manufacturing the camera sensors. Therefore, we have a focal length in both the x and y direction, f_x and f_y, and the center of the image is denoted by c_x, c_y, with both sets of values measured in pixels. As opposed to projecting the point in the *world* frame, we will now be projecting the point in the *camera* frame, ${}_c\mathbf{p} = [{}_cx, {}_cy, {}_cz]^T$. This new mapping is expressed by [52]

$$({}_cx, {}_cy, {}_cz)^T \mapsto \left(f_x \frac{{}_cx}{{}_cz} + c_x, f_y \frac{{}_wy}{{}_wz} + c_y \right)^T \tag{2.15}$$

This is especially true in a SLAM application, where we have the world frame as a starting point, and want to track the moving camera with reference to that origin. This is illustrated

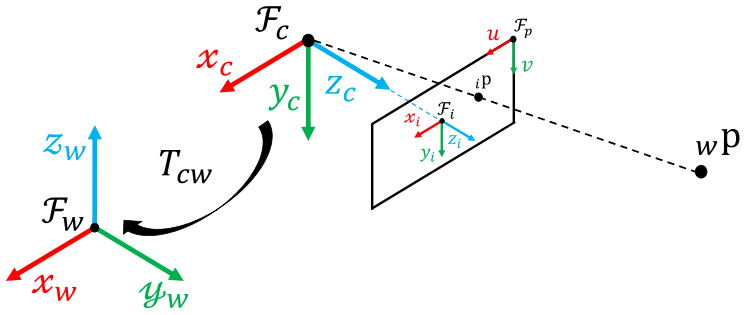


Figure 2.10: Illustration of a camera projecting a 3D point from the world frame, ${}_\mathbf{w}\mathbf{p}$, to the image frame, ${}_\mathbf{i}\mathbf{p}$, with the camera frame $\mathcal{F}_c$ and world frame $\mathcal{F}_w$ separated. Image adapted from Hartley and Zisserman [52].

in Figure 2.10, where we see that there are now 4 main frames of interest: the world frame ($\mathcal{F}_w$), camera frame ($\mathcal{F}_c$), image frame ($\mathcal{F}_i$), and the pixel frame ($\mathcal{F}_p$). The transformation $\mathbf{T}_{cw}$ represents the pose of $\mathcal{F}_w$ with respect to $\mathcal{F}_c$. In order to use the projection discussed above, we must first convert the points from the world frame to the camera frame. This is defined by

$${}_c\mathbf{p} = \mathbf{T}_{cw\,w}\mathbf{p} = \mathbf{R}_{cw\,w}\mathbf{t}_{w\mathbf{p}} + {}_c\mathbf{t}_{cw} = {}_c\mathbf{t}_{c\mathbf{p}} \tag{2.16}$$

where ${}_c\mathbf{t}_{cw}$ represents the position of $\mathcal{F}_w$ with respect to $\mathcal{F}_c$, expressed in $\mathcal{F}_c$.

We therefore define the projection function, π, as

$$ {}_p\mathbf{p} = \pi({}_c\mathbf{p}) = \pi \begin{pmatrix} {}_cx \\ {}_cy \\ {}_cz \end{pmatrix} = \begin{pmatrix} f_x \frac{{}_cx}{{}_cz} + c_x \\ f_y \frac{{}_cy}{{}_cz} + c_y \end{pmatrix} \tag{2.17} $$

where π represents the projection function, ${}_cx, {}_cy, {}_cz$ represent the x, y, and z-coordinates of the point in the camera frame ${}_c\mathbf{p}$, and f_x, f_y, c_x, c_y represent the camera intrinsic matrix parameters.

2.3.2 Stereo Projection Measurement Model

Our measurement model is the rectified stereo projection model, where we assume that the transformation between the right and left cameras is defined by a horizontal translation equivalent to the baseline. This is defined by the function π_s

$$ \pi_s({}_c\mathbf{p}) = \pi_s \begin{pmatrix} {}_cx \\ {}_cy \\ {}_cz \end{pmatrix} = \begin{pmatrix} f_x \frac{{}_cx}{{}_cz} + c_x \\ f_y \frac{{}_cy}{{}_cz} + c_y \\ f_x \frac{({}_cx - b)}{{}_cz} + c_x \end{pmatrix} \in \mathbb{R}^3 \tag{2.18} $$

where ${}_cx, {}_cy, {}_cz$ represent the x, y, and z-coordinates of the point in the camera frame ${}_c\mathbf{p}$, f_x, f_y, c_x, c_y represent the camera intrinsic matrix parameters, and b is the baseline between stereo cameras.

2.4 ORB_SLAM2

ORB_SLAM is an open source SLAM algorithm developed by Mur-Artal and Tardós with both monocular [27] and stereo/RGBD [10] functionality. This algorithm has gained immense popularity due to its well-documented, usable source code, as well as its excellent speed and accuracy. This is one of the common benchmarks when comparing SLAM algorithms, and we therefore chose to build upon this codebase when developing SIVO. ORB_SLAM has 3 main threads that operate in parallel, as seen in Figure 2.11.

The tracking thread comprises the majority of the front end, and is where we will make our modifications for SIVO. This thread extracts stereo ORB features from the environment, and creates a Bag of Words representation in order to describe the features [53].

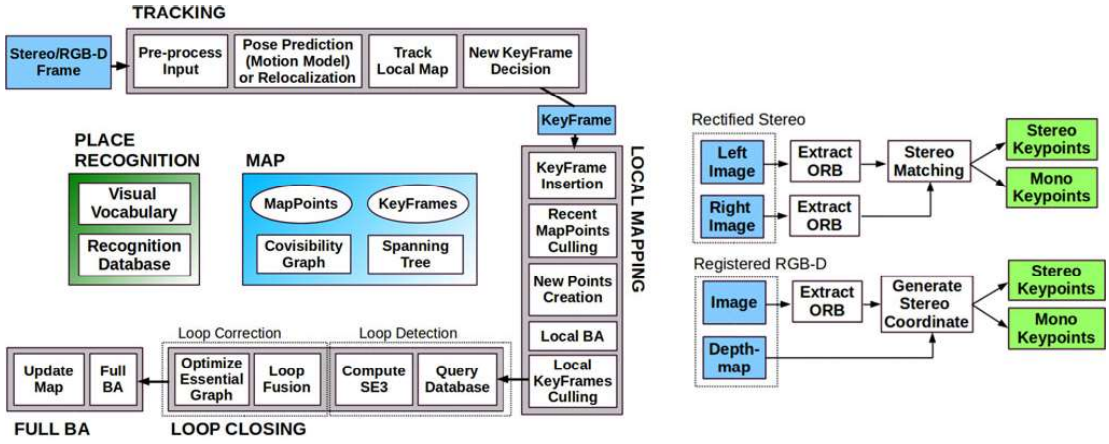


Figure 2.11: ORB_SLAM2 architecture diagram. Image retrieved from Mur-Artal and Tardós [10].

Any features detected in an image are matched to the *local map*, and matched measurements are added to a "motion-only" bundle-adjustment, which only optimizes the camera rotation and translation.

The second thread is the local mapping thread, which comprises part of the back end. The local map is defined through a *covsibility* graph, which links keyframes observing common points [10]; all points observed are part of the local map. New keyframes are sent from the tracking thread to the local mapping thread, which then adds the newly observe/tracked points to the local mapping optimization. This thread then only optimizes the location of the 3D world points within the bundle adjustment equation.

The final thread is the loop closing thread, which works to detect loops and perform "full" bundle adjustment. Using the bag of words representation for ORB features, this thread detects when the camera is visiting a pre-mapped area. Once the loop is detected, it spawns a fourth thread to perform a full pose-and-point bundle adjustment, optimizing over the entire trajectory. We use all of the functionality of ORB_SLAM2, however we modify the tracking thread to include our novel feature selection method.

2.5 Semantic Segmentation

Semantic segmentation is the task of labelling each pixel of an image with a distinct class, such as a car, pedestrian, or traffic sign. In recent years, deep neural networks have proven

Figure 2.12: Semantic segmentation example using Bayesian SegNet [46] trained on the CityScapes dataset [55].

to be extremely capable at this task with architectures such as SegNet [31], PSPNet [54], and DeepLab [39]. Figure 2.12 shows a segmented scene from the CityScapes dataset [55]. This was performed using Bayesian SegNet [46], which will be discussed in Section 2.6.8. Having class knowledge of each pixel allows us to add *context* to our SLAM algorithm. The typical corner features used by a SLAM algorithm (see Section A.3.1) can now be supplemented by this contextual understanding.

2.5.1 Convolutional Neural Networks (CNNs)

Most semantic segmentation neural network architectures are based on the Convolutional Neural Network (CNN), which rose to prominence from the work of Krizhevsky [56]. A CNN is a feed-forward NN which uses a combination of convolution kernels, non-linear *activation functions* (such as a Rectified Linear Unit (ReLU), or sigmoid function), and pooling in its architecture. These networks have proven to be extremely effective in learning tasks ranging from object detection to natural language processing. When using a CNN for object detection, *fully connected* (FC) layers are typically employed to represent the detection in vector form. Long et al. [57] "convolutionalized" the FC layers in a CNN (see Figure 2.13), which resulted in an output of a heatmap, as opposed to the typical feature vector. This heatmap can then be upsampled back to the original image size in order to determine a true pixel-wise segmentation.

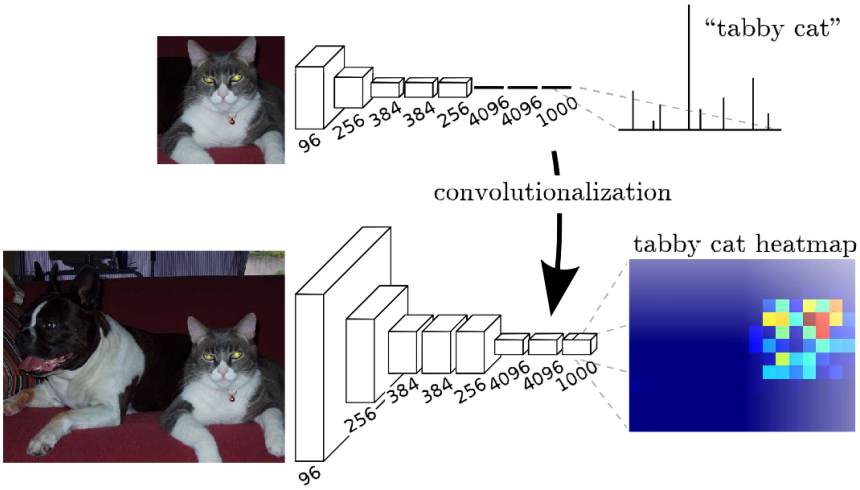


Figure 2.13: Convolutionalization of the fully connected (FC) layers in a convolutional neural network (CNN). Image retrieved from Long et al. [57].

2.5.2 SegNet

SegNet [31] was one of the first architectures to build on the fully-convolutional pipeline. The authors proposed an encoder-decoder architecture, which is now very common in most semantic segmentation networks. The goal of an encoder-decoder architecture is to reduce the original image into a reduced-dimension representation, which should encode all of the information in the scene. This can then be deconvolved through the decoder and passed through a softmax function (Equation 2.21) to obtain the classification.

Figure 2.14 illustrates the SegNet architecture. The main contribution of this network is to save the max-pooling indices from the encoder, and use these to upsample the inputs in the decoder. In contrast to *learning* the upsampling, this allowed for the entire network to be trained at once at a reasonable learning rate, which at the time was not common for such a large network. SegNet is the base of the segmentation network used in this work.

2.6 Uncertainty in Machine Learning

While machine learning development has exploded in recent years, the idea of network *uncertainty* is still an open research area [12]. How much trust can we place in a network's output ? What if the network's prediction is wrong, and what are the consequences? This

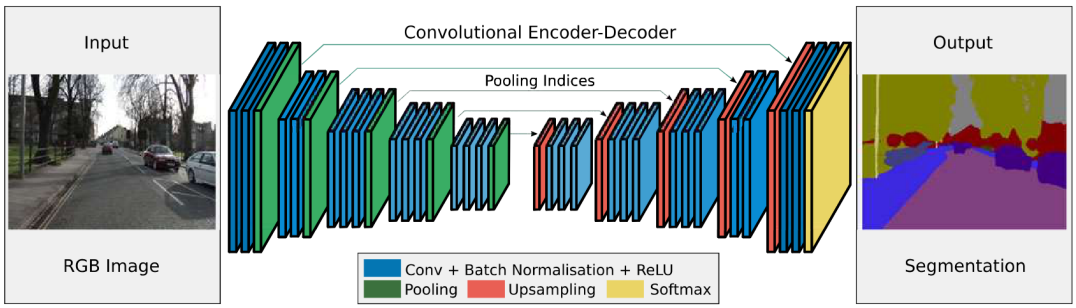


Figure 2.14: The SegNet architecture for semantic segmentation. Image retrieved from Badrinarayanan et al. [31].

is especially important in a robotics application, where decisions have to be made in spite of uncertainty [13]. While this is an open research area, we will be using the methodology presented by Gal [58, 59, 60, 61].

2.6.1 Gaussian Processes and Bayesian Neural Networks

In Section 2.2.2 we discussed the concept of a Gaussian Process, which can be summarized as a distribution over functions. It has been shown that a neural network with one layer, an infinite number of weights, and a Gaussian distribution placed over each of its weights, converges to a GP [62]. We can intuitively see that this is the case; neural networks have long been considered as "function approximators", and placing a Gaussian distribution over the weights would then result in a distribution over the function.

An infinitely-wide neural network is obviously impossible to construct, however, finite NNs with distributions over its weights have been studied as *Bayesian Neural Networks* [62]. The claim made by Gal [60] is that by incorporating dropout [63] before every weight layer in a neural network with arbitrary depth and nonlinearities is a mathematically equivalent approximation to the Bayesian Neural Network, which in turn is an approximation to the deep GP.

2.6.2 Dropout

Dropout was first presented by Srivastava et al. [63] as a method to prevent overfitting and inhibit the co-adaptation of features at each layer. Starting with a single layer NN

with K weights, let us define $\mathbf{W_1}, \mathbf{W_2}$ as the weight matrices for each layer, $\sigma(\cdot)$ as the element-wise nonlinearity (e.g. ReLU), $\mathbf{b}$ as the bias term, $\mathbf{x}$ as the input, and $\mathbf{y}$ as the output. Here, $\mathbf{x} \in \mathbb{R}^Q$, $\mathbf{b} \in \mathbb{R}^K$ and $\mathbf{y} \in \mathbb{R}^D$. This leads to $\mathbf{W_1} \in \mathbb{R}^{K \times Q}$ and $\mathbf{W_2} \in \mathbb{R}^{D \times K}$. For a standard NN, the network output is determined by

$$\mathbf{y} = \mathbf{W_2} \cdot \sigma(\mathbf{W_1}\mathbf{x} + \mathbf{b}) \tag{2.19}$$

Let us now define two binary vectors $\mathbf{z_1} \in \mathbb{R}^Q, \mathbf{z_2} \in \mathbb{R}^K$. Each element of these vectors is drawn from a *Bernoulli* distribution, i.e.

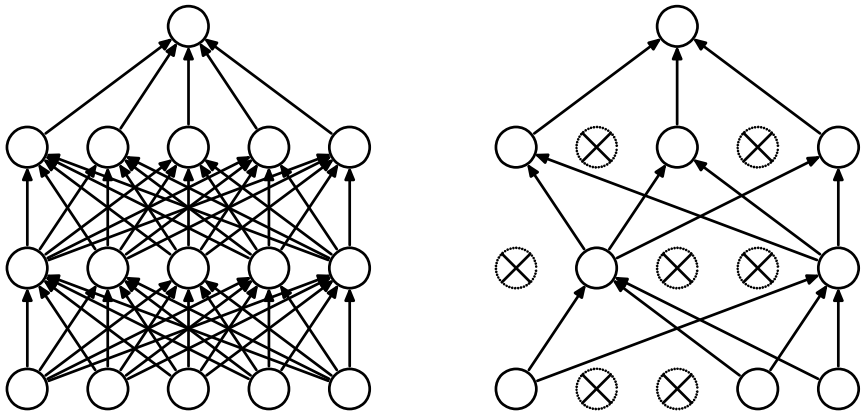
$$p_i \in [0, 1] \qquad \forall i = 1, 2$$

$$\mathbf{z_{1,q}} \sim Bernoulli(p_1) \quad \mathbf{z_{2,k}} \sim Bernoulli(p_2)$$

These binary vectors can now be multiplied element-wise by the input and hidden layer vectors

$$\mathbf{y} = \mathbf{W_2}(\mathbf{z_2}^T \sigma(\mathbf{W_1}(\mathbf{z_1}^T \mathbf{x}) + \mathbf{b})) \tag{2.20}$$

Randomly setting elements to 0 prevents the network from overfitting by sampling from "thinned out" neural networks, and preventing co-adaptation between features present in the training data during each mini-batch [63]. This is illustrated in Figure 2.15.



Standard Neural Net After applying dropout

Figure 2.15: Illustration of dropout randomly setting input values to zero, and creating a "thinned-out" neural network. Image retrieved from Srivastava et al. [63].

2.6.3 Classification Loss

For classification, the output of the neural network is passed through an element-wise softmax mapping, indicating the probability of detecting class c_i out of C potential classes.

$$p_{c_i} = \frac{\exp(y_{c_i})}{\sum_{c_i \in C} \exp(y_{c_i})} \tag{2.21}$$

The ground truth is defined as C-dimensional one-hot encodings for each pixel. A cross-entropy loss over the number of pixels (N) is used for classification, which becomes the Softmax loss due to the use of one-hot encodings. For each pixel, only the probability output for the desired class, $p_{c_i,n}$ is considered.

$$E = -\frac{1}{N} \sum_{n=1}^{N} \log(p_{c_i,n}) \tag{2.22}$$

In addition to this loss term, regularization terms are typically added for the weight matrices and bias vectors. This results in the total cost,

$$\mathcal{L}_{dropout} := E + \lambda_1 \|\mathbf{W_1}\|_2^2 + \lambda_2 \|\mathbf{W_2}\|_2^2 + \lambda_3 \|\mathbf{b}\|_2^2 \tag{2.23}$$

which uses L_2 regularization weighted by a weight decay term, λ_i. For a deeper network, this regularization terms would include all weight matrices and bias vectors.

$$\mathcal{L}_{dropout} := E + \lambda \sum_{i=1}^{L} (\|\mathbf{W}_i\|_2^2 + \|\mathbf{b}_i\|_2^2) \tag{2.24}$$

2.6.4 Gaussian Processes for Machine Learning

Let us define:

Inputs: $\mathbf{X} \in \mathbb{R}^{N \times Q} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$
Outputs: $\mathbf{Y} \in \mathbb{R}^{N \times D} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$
Functions: $\mathbf{F} \in \mathbb{R}^N = \{\mathbf{f}_1, \ldots, \mathbf{f}_N\}$
Covariance Function: $\mathbf{K}(\mathbf{X}, \mathbf{X}')$

We place a prior over the space of functions to define a posterior distribution which describes the probability that a particular function generated our input and output data. This is defined as [61]

$$p(\mathbf{f}|\mathbf{X}, \mathbf{Y}) \propto p(\mathbf{Y}|\mathbf{X}, \mathbf{f})p(\mathbf{f})$$

For classification tasks, we can now estimate the posterior. Passing the model outputs through the softmax function defined in 2.21, we can sample from a *categorical* distribution.

$$\mathbf{F}|\mathbf{X} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X}')) \tag{2.25}$$

$$\mathbf{Y}|\mathbf{F} \sim \mathcal{N}(\mathbf{F}, \mathbf{0}) \tag{2.26}$$

$$c_n|\mathbf{y}_n \sim \text{Categorical}\left(\frac{\exp(y_{c_i,n})}{\sum_{c_i \in C} \exp(y_{c_i,n})}\right) \tag{2.27}$$

We see that $\mathbf{Y} = \mathbf{F}$. This notation was selected by Gal in order to define the same framework for classification and regression tasks.

2.6.5 Bayesian Neural Networks

As discussed above, a Bayesian Neural Network is a finite NN with a distribution over its weights, and can be thought of as an approximation to the deep GP. Typically, we place standard Gaussian priors over each weight matrix, $p(\mathbf{W}_i)$, with a point estimate on the bias vectors.

$$\mathbf{W}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

The output of a neural network, $\mathbf{y}$ with Gaussian distributed weights can be expressed as

$$\mathbf{y} = \mathbf{f}(\mathbf{x}, (\mathbf{W}_i)_{i=1}^L) \tag{2.28}$$

which is a random variable dependent on the input, $\mathbf{x}$, and the weight matrices $\mathbf{W}_i$. The Bayesian NN effectively approximates the function $\mathbf{f}$. In classification tasks such as semantic segmentation, we will then pass this output through the softmax function.

$$p(c_i|\mathbf{x}, (\mathbf{W}_i)_{i=1}^L) = \text{Categorical}\left(\frac{\exp(y_{c_i})}{\sum_{c_i \in C} \exp(y_{c_i})}\right) \tag{2.29}$$

We therefore can obtain a probability for each class from our categorical distribution. In this work, we have 15 possible outputs.

2.6.6 Variational Inference

For any new point $\mathbf{x}^*$, the predicted output can be determined by integrating over all possible functions.

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})d\mathbf{f} \tag{2.30}$$

This integral is typically intractable. Therefore, Gal proposes to use *variational inference* to approximate the posterior [60]. This process first conditions the model on a set of finite variables, $\boldsymbol{\omega}$, and then defines a *varational distribution*, $q(\boldsymbol{\omega})$, which is easy to evaluate. The key assumption is that these finite variables are *sufficient statistics* for our model. Minimizing the Kullback-Leibler (KL) divergence (Equation 2.44) between this variational distribution and the true distribution should result in an accurate approximation. Incorporating these finite variables yields the new predictive distribution.

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{x}^*, \boldsymbol{\omega})p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})d\mathbf{f}d\boldsymbol{\omega}$$

Typically the distribution $p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$ is also intractable, therefore we define $q(\boldsymbol{\omega})$ as described above. Including the variational distribution results in the following predictive distribution.

$$q(\mathbf{y}^*|\mathbf{x}^*) = \int p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{x}^*, \boldsymbol{\omega})q(\boldsymbol{\omega})d\mathbf{f}d\boldsymbol{\omega} \tag{2.31}$$

Minimizing the KL divergence is equivalent to maximizing the *log evidence lower bound* [59], which is expressed by

$$\mathcal{L}_{VI} := \int q(\boldsymbol{\omega})p(\mathbf{F}|\mathbf{X}, \boldsymbol{\omega}) \log(p(\mathbf{Y}|\mathbf{F}))d\mathbf{F}d\boldsymbol{\omega} - \mathrm{KL}(q(\boldsymbol{\omega})||p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})) \tag{2.32}$$

2.6.7 Approximating the Bayesian Neural Network using Variational Inference

The main goal of variational inference is to approximate the posterior distribution of the sufficient statistics, $\boldsymbol{\omega}$, given our input and output data, $\mathbf{X}$ and $\mathbf{Y}$. This is expressed as

$$p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})$$

For a Bayesian neural network, the sufficient statistics are the weights.

$$\boldsymbol{\omega} = (\mathbf{W}_i)_{i=1}^{L}$$

Gal and Ghahramani [59] define the variational distribution for each layer as

$$\mathbf{W}_i = \mathbf{M}_i \cdot \text{diag}([\mathbf{z}_{i,j}]_{j=1}^{K_i}) \tag{2.33}$$

$$\mathbf{z}_{i,j} \sim \text{Bernoulli}(p_i) \quad \text{for} \quad i = 1, \ldots, L,\ j = 1, \ldots, K_{i-1} \tag{2.34}$$

In the above equation, $\mathbf{z}_{i,j}$ are Bernoulli distributed random variables with probability p_i, and $\mathbf{M}_i$ are the variational parameters to optimize. Equation 2.32 cannot be evaluated analytically, and so it must be approximated. Gal and Ghahramani use Monte Carlo integration to approximate the integral, resulting in

$$\hat{\mathcal{L}}_{VI} := \sum_{i=1}^{N} E(\mathbf{y}_i, \mathbf{f}(\mathbf{x}_i, \hat{\boldsymbol{\omega}})) - \text{KL}(q(\boldsymbol{\omega})||p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y})) \tag{2.35}$$

where $E(\cdot)$ is the softmax loss, illustrated in Equation 2.22. Gal [61] illustrates in detail the steps required to approximate the KL divergence, and show that this is equivalent to minimizing the dropout cost from Equation 2.24.

Predicting new outputs using this model use Equation 2.31, but the integration can similarly be approximated using Monte Carlo integration. In contrast to averaging the weights as described in [63], the outputs from the "thinned" networks are averaged as the final output. This is referred to as *MC dropout* [59].

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) \approx \int p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega}) q(\boldsymbol{\omega}) d\boldsymbol{\omega} \approx \frac{1}{T} \sum_{t=1}^{T} p(\mathbf{y}^*|\mathbf{x}^*, \hat{\boldsymbol{\omega}}_t) \tag{2.36}$$

The values of $\hat{\boldsymbol{\omega}}_i$ are realizations from $q(\boldsymbol{\omega})$. This method is effectively averaging samples obtained from the posterior distribution over models, or in other words, is sampling from different *function* realizations from the GP by using dropout to create "thinned out" neural networks for each Monte Carlo sample.

2.6.8 Bayesian SegNet

Kendall et al. [46] applied the Bayesian Neural Network approximation to SegNet [31], by adding dropout layers to both the encoder and decoder sections of the architecture.

The authors implemented various dropout schemes in order to determine the most effective architecture. Although dropout should be added before *every* layer as per Gal [60], this acted as too strong of a regularizer and prevented the network from training at a

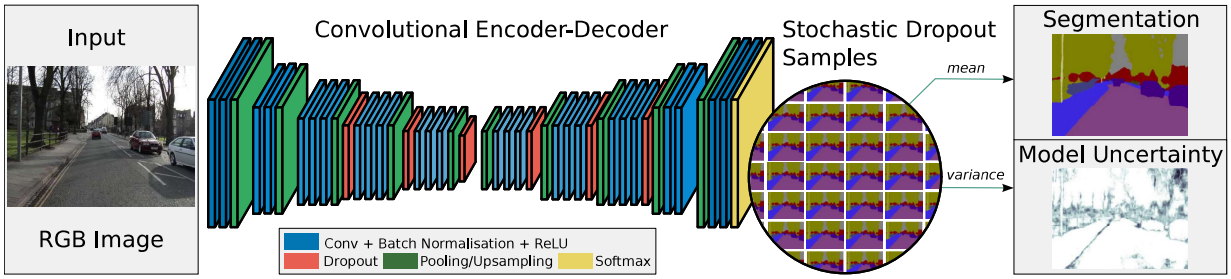


Figure 2.16: Bayesian SegNet architecture for semantic segmentation. Image retrieved from Kendall et al. [46]

reasonable rate. In addition to this, placing dropout before every layer did not perform as well as a network with dropout only used after these central layers. The posterior distribution over the models, $p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y})$, is approximated by using Monte Carlo sampling with dropout at test time, and the results are then passed through the softmax classifier (2.21) to obtain the posterior distribution of softmax class probabilities. As per Kendall [46], it is important to note the distinction between the softmax "probabilities" and the obtained distribution from Monte Carlo sampling; the softmax mapping describes relative probabilities between class detections, but is not an absolute measure of certainty.

2.7 Information Theory

Information theory is a field with its basis in communication, and is concerned with data compression and transmission. We will be approaching information theory in the context of encoding the *information content* of a stochastic variable.

2.7.1 Entropy

Entropy quantifies the average uncertainty in a random variable [64], or can be considered as a metric for the expected information content of a variable [65]. For a stochastic variable $X = \{x_0, x_1, \ldots, x_n\}$ with pmf $p(x)$, the entropy is defined by

$$H(x) = -\sum_{x \in X} p(x) \log p(x) \tag{2.37}$$

The units of entropy are defined by the base of the logarithm from Equation 2.37. The use of the natural logarithm (ln) indicates a measurement unit of *nats*, while calculations

using the base-2 logarithm (log) are measured in *bits*.

2.7.2 Joint Entropy

The definition of entropy discussed above was only concerned with one random variable, however this can be extended to multiple variables. The joint entropy between a pair of discrete random variables $X = \{x_0, x_1, \ldots, x_n\}$ and $Y = \{y_0, y_1, \ldots, y_m\}$ can be expressed by [64]

$$H(X,Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log p(x,y) \tag{2.38}$$

In the event that X and Y are independent, the joint entropy $H(X,Y)$ can be expressed as a sum of the individual entropies.

$$H(X,Y) = H(X) + H(Y) \tag{2.39}$$

2.7.3 Conditional Entropy

The conditional entropy is a metric to express the information content of one variable when we have knowledge of another *correlated* variable. The conditional entropy is defined by [64]

$$\begin{aligned} H(Y|X) &= -\sum_{x \in X} p(x) H(Y|X = x) \\ &= -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log p(y|x) \end{aligned} \tag{2.40}$$

Looking at equations 2.37, 2.38, and 2.40, we see that they are all intuitively related [64].

$$H(X,Y) = H(X) + H(Y|X) \tag{2.41}$$

2.7.4 Entropy of a Gaussian Random Variable

The entropy for a univariate or multivariate Gaussian are well defined. The univariate case is defined by

$$H(x) = \frac{1}{2} \log\left(\sigma^2 2\pi e\right) \tag{2.42}$$

where σ represents the standard deviation of the distribution. This can easily be extended for a multivariate Gaussian

$$H(x) = \frac{1}{2}\log((2\pi e)^n \det(\Sigma)) \tag{2.43}$$

where Σ represents the covariance matrix of the random variable, and n is the number of variables comprising the random variable. We see that the metric of entropy for a mutivariate Gaussian is directly related to the *volume* of the covariance hyperellipsoid. A variable with a larger entropy would have a larger covariance hyperellipsoid, which follows our intuitive understanding that entropy is a measurement of uncertainty.

2.7.5 Relative Entropy and Mutual Information

The relative entropy, or *Kullback-Leibler* (KL) divergence is a metric to express the difference between two probability distributions [64].

$$D(p||q) = \sum_{x \in X} p(x) \log\left(\frac{p(x)}{q(x)}\right) \tag{2.44}$$

The KL divergence can also be used to express the *inefficiency* of using distribution q to approximate p [64]. This metric is always non-negative, and is only 0 if $p = q$.

The concept of relative entropy is directly related to that of Mutual Information; the mutual information is defined as the relative entropy between the joint distribution and the product of individual distributions. This can also be considered as the amount of information shared between two variables [65].

$$I(X;Y) = \sum_{x \in X}\sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \tag{2.45}$$

2.7.6 Relationship Between Entropy and Mutual Information

By rearranging the definition of mutual information, we can extract a relationship between this metric and entropy [64].

$$\begin{aligned} I(X;Y) &= \sum_{x\in X}\sum_{y\in Y} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} \\ &= \sum_{x\in X}\sum_{y\in Y} p(x,y)\log\frac{p(x|y)}{p(x)} \\ &= -\sum_{x\in X}\sum_{y\in Y} p(x,y)\log p(x) + \sum_{x\in X}\sum_{y\in Y} p(x,y)\log p(x|y) \\ &= -\sum_{x\in X} p(x)\log p(x) - \left(-\sum_{x\in X}\sum_{y\in Y} p(x,y)\log p(x|y)\right) \\ &= H(X) - H(X|Y) \end{aligned} \tag{2.46}$$

Therefore, the mutual information describes the *reduction* of uncertainty in variable X, due to gaining knowledge of variable Y. One last interesting point as per [64] is that the mutual information of a random variable with *itself* is the entropy of the random variable. Hence, entropy is sometimes referred to as self-information. Figure 2.17 clearly illustrates the relationship between entropy, joint entropy, conditional entropy, and mutual information.

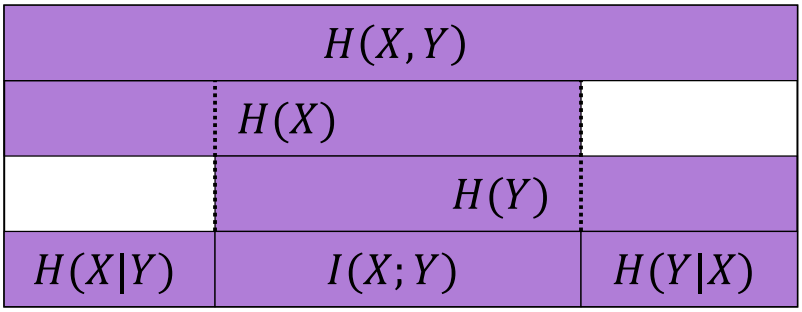


Figure 2.17: Relationship between entropy, joint entropy, conditional entropy, and mutual information. Adapted from MacKay [66].

2.7.7 Mutual Information of a Multivariate Gaussian

The mutual information of a multivariate Gaussian has been well explored in literature. Let us denote a multivariate Gaussian variable by

$$\mathbf{x} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}, \quad \mathbf{\Sigma} = \begin{bmatrix} \mathbf{\Sigma}_{aa} & \mathbf{\Sigma}_{ab} \\ \mathbf{\Sigma}_{ba} & \mathbf{\Sigma}_{bb} \end{bmatrix} \tag{2.47}$$

where $\mathbf{\Sigma}$ represents the covariance matrix for the variable. Chli [65] showed that the mutual information between the two partitions of $\mathbf{x}$ can be written as

$$I(\mathbf{a}; \mathbf{b}) = \frac{1}{2} \log \left(\frac{\det(\mathbf{\Sigma}_{aa}) \det(\mathbf{\Sigma}_{bb})}{\det(\mathbf{\Sigma})} \right) \tag{2.48}$$

2.7.8 Uncertainty in Classification Results for a Bayesian Neural Network

Gal [58] outlines various metrics for uncertainty in classification, including entropy. In this setting, entropy will capture the information in the predictive distribution. Let us denote $\mathbf{y}$ as our network output, $\boldsymbol{\mathcal{I}}$ as our input image data, $\boldsymbol{\mathcal{D}}$ as our training data, and c_i as a particular class output with $c_i \in C$ potential classes. The probability of these class values is obtained by calculating the Softmax of the network output. Gal defines the entropy as

$$H(c_i|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) := -\sum_{c_i \in C} p(c_i|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) \log p(c_i|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) \tag{2.49}$$

We see that Equation 2.49 clearly derives from 2.37, with our variable of interest being the class output given our input and training data. Equation 2.49 reaches a maximum value when all of the class outputs are equiprobable, and a minimum value of 0 when one class is predicted with a probability of 1.

The probability values are the confidence values of each class *after* averaging the stochastic passes through the network. Although the confidence values individually do not necessarily have any meaning of uncertainty, the entropy calculation will observe the spread in the confidence value for each class output of a pixel. Therefore, we set

$$p(c_i|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) = \frac{1}{T} \sum_{t}^{T} p(c_i|\boldsymbol{\mathcal{I}}, \hat{\boldsymbol{\omega}}_{\boldsymbol{t}}) \tag{2.50}$$

where $\hat{\boldsymbol{\omega}}_{\boldsymbol{t}}$ represents our model parameters which optimize our variational distribution, $q_\theta^*(\boldsymbol{\omega})$. Each probability value is expressed using the Softmax mapping from Equation 2.21.

$$[p(c_i = 1|\boldsymbol{\mathcal{I}}, \hat{\boldsymbol{\omega}}_t), \ldots, p(c_i = C|\boldsymbol{\mathcal{I}}, \hat{\boldsymbol{\omega}}_t)] := \text{Softmax}(\mathbf{y}) = \frac{\exp(y_{c_i})}{\sum\limits_{c_i \in C} \exp(y_{c_i})} \tag{2.51}$$

Therefore, we can write an expression for the approximate entropy for the confidence output

$$H(c_i|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) = -\sum_{c_i \in C} \left(\frac{1}{T} \sum_t^T p(c_i|\boldsymbol{\mathcal{I}}, \hat{\boldsymbol{\omega}}_t) \right) \log \left(\frac{1}{T} \sum_t^T p(c_i|\boldsymbol{\mathcal{I}}, \hat{\boldsymbol{\omega}}_t) \right) \tag{2.52}$$

which expresses the uncertainty of our semantic segmentation in *bits*.

The entropy in classification can be visualized as the spread of points on a *probability simplex* [67]; in our application of Bayesian SegNet, this would be a 15-simplex due to our 15 output classes.

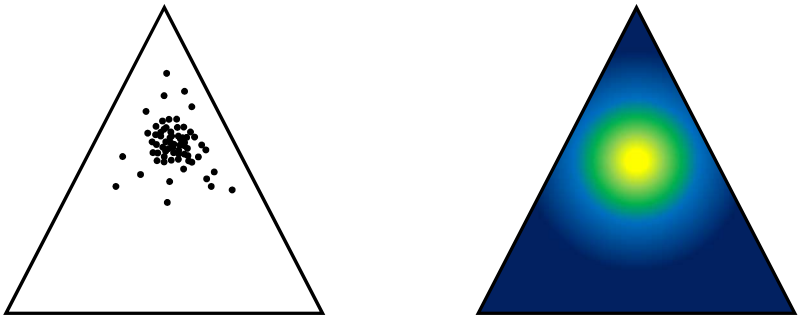


Figure 2.18: Ensemble and distribution of a categorical distribution over a 3-simplex. Adapted from Malinin and Gales [67].

The representation shown in Figure 2.18 is only true for one image input, $\boldsymbol{\mathcal{I}}^*$, passed through the network repeatedly. Each point in the ensemble is a categorical distribution, representing an implicit conditional distribution over the simplex. Assuming we have taken enough samples, the entropy of the ensemble should then represent the uncertainty in the *distribution* [67].

Chapter 3

SIVO Feature Selection Criteria

This section outlines the main contribution of this work: the feature selection methodology for long-term visual SLAM. As discussed earlier in this thesis, we blend together aspects of semantic segmentation (Section 2.5), neural network uncertainty (Section 2.6), and information theory (Section 2.7) in order to select the most stable and informative reference points.

Our method builds upon and combines the methods proposed by Davison [17] as well as Kaess and Dellaert [19]. We will first outline those methods in detail, and then present the SIVO feature selection strategy.

3.1 Information-Theoretic Feature Selection Methods

Davison [17] proposes a method to determine the quality of a measurement in order to determine the best location within an image to focus processing resources. Let us define our 6DOF robot pose parameterization at some time t as

$$\mathbf{x}_t = \begin{bmatrix} x & y & z & \phi & \psi & \theta \end{bmatrix}^T \in \mathbb{R}^6 \tag{3.1}$$

The pose has an associated covariance matrix, denoted as $\mathbf{\Sigma}_t \in \mathbb{R}^{6\times 6}$. Measurements are defined through a non-linear measurement model, $h(\mathbf{x}_t)$, as

$$\mathbf{z}_i = h_i(\mathbf{x}_t) + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_i) \tag{3.2}$$

which represents the rectified stereo projection model illustrated in Equation 2.18. This therefore implies that $\mathbf{z}_i \in \mathbb{R}^3$. Any new feature measurement will also have random noise, defined as a zero mean Gaussian with covariance $\mathbf{Q}_i \in \mathbb{R}^{3\times 3}$.

Assume that at time t, we have n available features distributed throughout the scene that we can select for our optimization pipeline. We can stack the current pose with the candidate measurements into a vector as such

$$\hat{\mathbf{x}} = \begin{bmatrix} \mathbf{x}_t \\ \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_n \end{bmatrix} = \begin{bmatrix} \mathbf{x}_t \\ h_1(\mathbf{x}) \\ h_2(\mathbf{x}) \\ \vdots \\ h_n(\mathbf{x}) \end{bmatrix} \tag{3.3}$$

As each of these random variables are described by multivariate Gaussian distributions, it follows that the stacked vector, $\hat{\mathbf{x}}$, also can be described by a multivariate Gaussian distribution. This stacked variable will also have a covariance matrix, which is defined through the pose covariance, and propagation of this pose covariance through the measurement model. This is defined by

$$\hat{\mathbf{\Sigma}} = \begin{bmatrix} \mathbf{\Sigma}_t & \mathbf{\Sigma}_t \frac{\partial h_1}{\partial \mathbf{x}}^T & \mathbf{\Sigma}_t \frac{\partial h_2}{\partial \mathbf{x}}^T & \cdots & \mathbf{\Sigma}_t \frac{\partial h_n}{\partial \mathbf{x}}^T \\ \frac{\partial h_1}{\partial \mathbf{x}} \mathbf{\Sigma}_t & \frac{\partial h_1}{\partial \mathbf{x}} \mathbf{\Sigma}_t \frac{\partial h_1}{\partial \mathbf{x}}^T + \mathbf{Q}_1 & \frac{\partial h_1}{\partial \mathbf{x}} \mathbf{\Sigma}_t \frac{\partial h_2}{\partial \mathbf{x}}^T & \cdots & \frac{\partial h_1}{\partial \mathbf{x}} \mathbf{\Sigma}_t \frac{\partial h_n}{\partial \mathbf{x}}^T \\ \frac{\partial h_2}{\partial \mathbf{x}} \mathbf{\Sigma}_t & \frac{\partial h_2}{\partial \mathbf{x}}^T \mathbf{\Sigma}_t \frac{\partial h_1}{\partial \mathbf{x}} & \frac{\partial h_2}{\partial \mathbf{x}} \mathbf{\Sigma}_t \frac{\partial h_2}{\partial \mathbf{x}}^T + \mathbf{Q}_2 & \cdots & \frac{\partial h_2}{\partial \mathbf{x}} \mathbf{\Sigma}_t \frac{\partial h_n}{\partial \mathbf{x}}^T \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial h_n}{\partial \mathbf{x}} \mathbf{\Sigma}_t & \frac{\partial h_n}{\partial \mathbf{x}} \mathbf{\Sigma}_t \frac{\partial h_1}{\partial \mathbf{x}}^T & \frac{\partial h_n}{\partial \mathbf{x}} \mathbf{\Sigma}_t \frac{\partial h_2}{\partial \mathbf{x}}^T & \cdots & \frac{\partial h_n}{\partial \mathbf{x}} \mathbf{\Sigma}_t \frac{\partial h_n}{\partial \mathbf{x}}^T + \mathbf{Q}_n \end{bmatrix} \tag{3.4}$$

This information-theoretic feature selection strategy revolves around the idea of selecting the measurement which best reduces the uncertainty of our pose through calculation of the mutual information. Recall from Section 2.7.5 that provided two random variables ($\mathbf{X}$, $\mathbf{Y}$) are dependent, the mutual information describes the reduction of uncertainty in variable $\mathbf{X}$, due to gaining knowledge of variable $\mathbf{Y}$. Here, we look to select the feature measurement $\mathbf{z}_i$ which best reduces the uncertainty in $\mathbf{x}$, or

$$\underset{\mathbf{z}_i}{\operatorname{argmax}} \, I(\mathbf{x}; \mathbf{z}_i)$$

This can be easily calculated using Equation 2.48, however we first must extract the *marginal* covariance for each feature $\mathbf{z}_i$ (refer to Section 2.2.1). Luckily, the marginalized covariance is just a selection of the relevant variables. For example, the marginal covariance of the state $\mathbf{x}$ and measurement $\mathbf{z}_i$ from the full covariance matrix in Equation 3.4 is

$$\hat{\mathbf{\Sigma}}_i = \begin{bmatrix} \mathbf{\Sigma}_{\mathbf{xx}} & \mathbf{\Sigma}_{\mathbf{x}\mathbf{z}_i} \\ \mathbf{\Sigma}_{\mathbf{z}_i\mathbf{x}} & \mathbf{\Sigma}_{\mathbf{z}_i\mathbf{z}_i} \end{bmatrix} = \begin{bmatrix} \mathbf{\Sigma}_t & \mathbf{\Sigma}_t \frac{\partial h_i}{\partial \mathbf{x}}^T \\ \frac{\partial h_i}{\partial \mathbf{x}} \mathbf{\Sigma}_t & \frac{\partial h_i}{\partial \mathbf{x}} \mathbf{\Sigma}_t \frac{\partial h_i}{\partial \mathbf{x}}^T + \mathbf{Q}_i \end{bmatrix} \tag{3.5}$$

We can then calculate the mutual information $I(\mathbf{x}; \mathbf{z}_i)$ as

$$I(\mathbf{x}; \mathbf{z}_i) = \frac{1}{2} \log \left(\frac{\det(\boldsymbol{\Sigma}_{\mathbf{xx}}) \det(\boldsymbol{\Sigma}_{\mathbf{z}_i \mathbf{z}_i})}{\det(\boldsymbol{\Sigma}_i)} \right) \tag{3.6}$$

This quantity should then describe, in bits, the expected information gain about the state by making this measurement. The feature which provides the maximum mutual information is then selected for the optimization pipeline, and we calculate a new state estimate. Once the state has been updated, this process is repeated until the maximum information provided by a new measurement falls below a user-defined threshold value, which we will denote as ΔH.

Kaess and Dellaert [19] build upon this method and evaluate features in a similar manner in their work. However, they argue that selecting individual features and then updating the state estimate is not practical. In most SLAM solutions, updating the state and extracting the marginal pose covariance values can be quite expensive depending on the algorithm used. Another option could be to calculate the mutual information between the state and all possible combinations of measurements, in order to determine the combination which provides the best mutual information. This however, scales with a 2^N complexity, which is unfeasiblet. Therefore, Kaess and Dellaert propose to select the best measurement *without* updating the state space, and then repeat the process with the remaining measurements until the best feature available provides an information gain less than ΔH. Although this is will not guarantee that the optimal landmark is selected for each iteration, it is far cheaper than the alternatives.

3.2 SIVO Feature Selection

We now enhance this information-theoretic approach to incorporate semantic information. As discussed in Section 2.3, we know that each measurement is a 3D point projected to a 2D pixel on the image frame. Using semantic segmentation, we can then determine a discrete class value for each pixel measurement as well. Therefore, we can determine the points which best reduce our uncertainty while incorporating contextual knowledge to determine whether they will also be stable references.

For example, say we have 4 measurements, $\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{z}_4$. Using the mutual information

calculation from Equation 3.6, the *value* of each measurement is

$$\begin{aligned} I(\mathbf{x};\mathbf{z}_1) &= 7.0 \text{ bits} \\ I(\mathbf{x};\mathbf{z}_2) &= 5.0 \text{ bits} \\ I(\mathbf{x};\mathbf{z}_3) &= 4.5 \text{ bits} \\ I(\mathbf{x};\mathbf{z}_4) &= 5.0 \text{ bits} \end{aligned} \tag{3.7}$$

We see that measurement $\mathbf{z}_1$ clearly provides the most information gain about the state, followed by $\mathbf{z}_2$ and $\mathbf{z}_4$, with $\mathbf{z}_3$ providing the least information. Now, let us take semantic information into account, by denoting the detected class for a measurement as c_i. The detected classes for these 4 features are

$$\begin{aligned} c_1 &= \text{Bicycle} \\ c_2 &= \text{Building} \\ c_3 &= \text{Traffic Sign} \\ c_4 &= \text{Tree} \end{aligned} \tag{3.8}$$

This is an interesting, albeit predictable development for this example; the most informative point happens to be part of a bicycle, which we know from experience is most likely not a stable reference. While the bicycle could be static and parked for several hours, there is a higher probability that it is currently moving. Therefore, tracking the motion of this feature will not provide an accurate representation of vehicle motion. Therefore, it seems that we should select one of the other measurements for long-term mapping. However, c_2 and c_4 provide the same amount of information for our state estimate. We now look to determine which of the two is a better feature by introducing network uncertainty.

We asked earlier: what happens if the NN provides us with the wrong answer? In our example, this could manifest as the network incorrectly segmenting the image and misclassifying a vehicle as a building. Although the network should be trained to accurately segment images in the test environment, an out-of-domain example could result in a poorly segmented image, such as Figure 3.1. While part of the van is being properly classified (blue) in Figure 3.1, the logo on the van has confused the network, causing it to be labelled as a traffic sign. In this work, we segment images using Bayesian SegNet [46], and therefore can incorporate network uncertainty into our feature selection methodology by using MC dropout.

Recall from Equation 2.46 the relationship between entropy and mutual information. For the purely information-theoretic approach with our robot state and new feature measurements, we can rewrite the mutual information criterion for feature selection in terms

Figure 3.1: Example of low quality image segmentation. In this scenario, the logo on the right van (blue) is being misclassified as a traffic sign (yellow), while the left van is being misclassified as a building (grey).

of entropy,

$$I(\mathbf{x}; \mathbf{z}_i) = \Delta H = H(\mathbf{x}|\mathbf{Z}) - H(\mathbf{x}|\mathbf{z}_i, \mathbf{Z}) \tag{3.9}$$

where $\mathbf{Z}$ represents all previous measurements made in order to obtain our current state estimate. This value, ΔH, is the same threshold for feature selection denoted above; if this value is greater than a pre-defined threshold for a measurement $\mathbf{z}_i$, it is selected as a reference for the SLAM algorithm in both Davison's [17] and Kaess and Dellaert's [19] works.

We propose to modify ΔH, and evaluate the entropy difference between the current state and the joint entropy of the state given the new feature measurement, and the semantic segmentation classification, using Equation 2.52. This can be expressed as

$$\Delta H = H(\mathbf{x}|\mathbf{Z}) - H(\mathbf{x}, c_i|\mathbf{z}_i, \mathbf{Z}, \boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) \tag{3.10}$$

where $\boldsymbol{\mathcal{I}}$ represents the current image and $\boldsymbol{\mathcal{D}}$ represents the dataset used to train the neural network. We assume that the classification entropy and state entropy are conditionally independent. Therefore, using Equation 2.39, we can rewrite the last term as

$$\begin{aligned} H(\mathbf{x}, c_i|\mathbf{z}_i, \mathbf{Z}, \boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) &= H(\mathbf{x}|\mathbf{z}_i, \mathbf{Z}, \boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) + H(c_i|\mathbf{z}_i, \mathbf{Z}, \boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) \\ &= H(\mathbf{x}|\mathbf{z}_i, \mathbf{Z}) + H(c_i|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) \end{aligned} \tag{3.11}$$

The state is not dependent on the actual image or dataset, thus we can remove these conditionally dependent terms from the individual entropy terms. Similarly, the classification

detection is not dependent on any of the feature measurements. Therefore, we can now rewrite Equation 3.10 as

$$\Delta H = H(\mathbf{x}|\mathbf{Z}) - H(\mathbf{x}|\mathbf{z}_i, \mathbf{Z}) - H(c_i|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) \tag{3.12}$$

The first two terms are exactly the mutual information criteria introduced in Section 3.1. Therefore, we can further simplify the equation to yield

$$\boxed{\Delta H = I(\mathbf{x}; \mathbf{z}_i) - H(c_i|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}})} \tag{3.13}$$

Equation 3.13 highlights the main claim of this work. We argue that the best reference points should not only provide the most information to reduce the uncertainty of the state, but they should be static reference points which have been detected as such with a very high certainty. This feature selection criteria provides us with a weighting between the value of a feature for the state estimate and the certainty of the feature's classification. Recall from Section 2.7.8 that the minimum value of $H(c_i|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}})$ is 0 when the network predicts one class with a confidence of 100%, and all other classes as 0%. Therefore, in an ideal world where we can perfectly identify the class of each pixel in every image, we will be selecting the features which best reduce the uncertainty of the state as per the original information-theoretic criterion outlined in Section 3.1.

Let us continue with our example. Imagine we run our current image frame through Bayesian SegNet, averaging the result of an arbitrarily selected 12 stochastic passes. After averaging these values, we extract the following maximum confidence outputs from the softmax layer

$$\begin{aligned} p_2 &= 20\% \\ p_3 &= 95\% \\ p_4 &= 60\% \end{aligned} \tag{3.14}$$

We can calculate the classification entropy using Equation 2.49 and convert the confidence values into bits. Recall that the entropy value for classification also requires the confidence values of all the *other* class outputs, c_i, even if they are not the maximum value. For this example, we assume the remaining confidence values are evenly distributed amongst the 14 other classes, as this is closest to our real-world scenario discussed in Chapter 4. The

entropy values are calculated to be

$$
\begin{aligned}
H(c_2|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) &= -0.2\log(0.2) + 14 \times \left(-\frac{0.8}{14}\log\left(\frac{0.8}{14}\right)\right) = 1.134 \text{ bits} \\
H(c_3|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) &= -0.95\log(0.95) + 14 \times \left(-\frac{0.05}{14}\log\left(\frac{0.05}{14}\right)\right) = 0.144 \text{ bits} \\
H(c_4|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) &= -0.6\log(0.6) + 14 \times \left(-\frac{0.4}{14}\log\left(\frac{0.4}{14}\right)\right) = 0.751 \text{ bits}
\end{aligned} \tag{3.15}
$$

As expected, we see that the lower the averaged network confidence, the higher the entropy. Fusing together the values from Equation 3.7 and 3.15 using Equation 3.13 yields

$$
\begin{aligned}
\Delta H_2 &= I(\mathbf{x}; \mathbf{z}_2) - H(c_2|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) = 5.0 - 1.134 = 3.866 \text{ bits} \\
\Delta H_3 &= I(\mathbf{x}; \mathbf{z}_3) - H(c_3|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) = 4.5 - 0.144 = \boxed{4.356 \text{ bits}} \\
\Delta H_4 &= I(\mathbf{x}; \mathbf{z}_4) - H(c_4|\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{D}}) = 5.0 - 0.751 = 4.249 \text{ bits}
\end{aligned} \tag{3.16}
$$

As per our evaluation criteria, point 3 should be the point we select for our visual SLAM pipeline. The entropy reduction it provides, ΔH_3, is the highest for all features. This point has been detected as a building with 95% confidence, and also significantly reduces the uncertainty of the state estimate. We follow the methodology of Kaess and Dellaert [19], and select all points that have an entropy reduction above a defined threshold. We modify this threshold value during our experiments, which is discussed further in Section 4. By selecting these points, our aim is to simultaneously reduce the number of map points stored while improving the quality of the selected points to aid long-term robot autonomy.

Note on Keyframe Selection

The keyframe selection process for our algorithm involves tweaking the criteria used by ORB_SLAM2. Mur-Artal [10] proposes a keyframe selection scheme based on the number of close and far keypoints detected in the scene. If the algorithm detects less than 100 close points in the current frame, but 70 new close points can be added, then the current frame is inserted as a keyframe. As SIVO discards points from cars, we found that we were using significantly fewer close features, causing keyframes to be added very frequently. Therefore, we modified these parameters to add a keyframe if less than 35 close points are detected in the current frame, but 70 new close points can be added.

Note on Outlier Rejection

The incorporation of semantic information also aids our outlier rejection method in the matching step. In addition to matching the descriptors over multiple scenes, we also require that the semantic classification must match for each detection of the feature. This is an added layer of robustness which ensures that feature tracks belong to the same static object, and the dual criteria is more reliable than purely relying on descriptor matching.

Chapter 4

Results

This section will discuss the implementation details and results of the algorithm presented in Chapter 3.

4.1 Implementation Details

Our implementation of SIVO is publicly available on Github[1], and the repository contains instructions and all dependencies required to run the code. Our training setup is adapted from the original SegNet repository, and can also be found online[2].

4.1.1 Training

We train Bayesian SegNet using the Cityscapes dataset [55], and fine-tune the network using the KITTI [11] dataset. We only use the finely annotated images from the dataset, which contains 2975 training and 500 validation images. An example of the annotated Cityscapes data can be seen in Figure 4.1. Cityscapes images are labelled with 33 distinct classes, however, we modify the labels using the authors' "Cityscapes scripts"[3] in order to generate training data with only 15 classes. The 15 classes selected are: road, sidewalk, building, wall/fence, pole, traffic light, traffic sign, vegetation, terrain, sky, person/rider, car, truck/bus, motorcycle/bicycle, and *void*.

[1] https://www.github.com/navganti/SIVO
[2] https://www.github.com/navganti/SegNet
[3] https://www.github.com/navganti/cityscapesScripts

Figure 4.1: Ground truth annotations for Cityscapes training data.[4]

In order to verify performance using the KITTI odometry dataset, we first prepare the Cityscapes data in a similar format to the KITTI test sequence images. Cityscapes images have a resolution of $1,024 \times 2,048$, while KITTI images are $376 \times 1,241$, $375 \times 1,242$, or $370 \times 1,226$. Bayesian SegNet requires both the height and width of the image must be divisible by 16, and in order to maintain most of the image data while also preserving some GPU memory, we select a resolution of $352 \times 1,024$. Therefore, the KITTI images are center cropped, while the Cityscapes images are cropped from the bottom to remove the ego-vehicle and downsampled to maintain the aspect ratio.

Due to the large image size, we use the Bayesian SegNet *Basic* network in order to preserve GPU memory and speed up inference time. This network contains fewer layers in both the encoder and decoder sections of the architecture in comparison to the original Bayesian SegNet. The network is trained for $215,000$ *iterations* with a mini-batch size of 8 images, corresponding to $\sim$ 578 *epochs*. The learning rate, η, is set to 0.001 and the momentum vector, β, is set to 0.9. An example of segmentation on Cityscapes validation data using the trained Bayesian SegNet Basic network can be seen in Figure 4.2. Although the trained network performs below the standard of top segmentation results on the Cityscapes benchmark,it has sufficiently learned the details it needs to effectively fine-tune performance using the KITTI semantic dataset.

To fine-tune the network, we use the trained network weights (which generated Figure 4.2) as initial values, and then provide new training examples. As the network has already

[4]Image retrieved from: https://www.cityscapes-dataset.com/wordpress/wp-content/uploads/2015/07/zuerich00.png

Figure 4.2: Semantic segmentation on Cityscapes [55] validation data using Bayesian SegNet Basic [46].

been trained for a particular task, it quickly adapts to the new data. The new training examples come from the KITTI [11] semantic dataset, which is comprised of 200 training and test images. The training data is split into 160 training images and 40 validation images. The learning rate and momentum values are kept as 0.001 and 0.9 respectively, and the network is trained for 2500 iterations, or 125 epochs. Results of Bayesian SegNet Basic on a validation image can be found in Figure 4.3. The segmentation quality is

Figure 4.3: Semantic segmentation on KITTI [11] semantic validation data using Bayesian SegNet Basic [46].

representative for SegNet, given the limited training resources and small dataset.

4.1.2 Network Inference and Hardware

We implement network inference using Caffe's [47] C++ interface in order to integrate the results from Bayesian SegNet with ORB_SLAM2. Any relevant information stored in the Caffe "Blobs," such as the network's pixel-wise confidence output, are converted to Eigen[5] tensors and matrices. The rest of the SIVO algorithm is embedded in the ORB_SLAM2 source code, specifically in the Tracking class. We do not disable any of ORB_SLAM2's loop closure or relocalization functionality. We have also created a custom SIVO library which uses hand-calculated Jacobian matrices (see Appendix B) to quickly and efficiently construct the marginal covariance matrix required to calculate the mutual information.

The algorithm is deployed on an Nvidia Titan Xp GPU and an Intel i7-6700 CPU @ 3.40 GhZ. For an image of resolution $352 \times 1,024$, inference takes approximately 108ms for 2 dropout samples, 310ms for 6 dropout samples, and 625ms for 12 samples. This performance is partially due to the size of the images being used, but there are also some further optimizations (such as the use of CUDA) which could improve inference time.

4.2 Experiments and Notation

We validate performance of our algorithm on the KITTI odometry dataset. The tunable parameters are the feature selection entropy threshold (H_{th}), and the number of samples for MC Dropout (T). The experiments are referred to as follows:

Bayesian **S**egNet T **E**ntropy H_{th}

So, for example, an experiment where the number of samples $T = 6$, and the entropy threshold is set to 4 bits is deemed **BS6E4**. For all trajectories, we evaluated the following configurations:

- BS2E4
- BS6E2
- BS6E3
- BS6E4

[5] http://eigen.tuxfamily.org/index.php?title=Main_Page

- BS12E4

These values are selected because $T = 6$ was determined to be the point where MC dropout outperformed the *weight averaging* technique [63] typically used with dropout at test time [46]. Through experimentation, we deemed that 5 bits was too strict of a threshold, and the ORB_SLAM2 feature matching pipeline did not have enough features to localize the robot. These configurations of SIVO are compared to ORB_SLAM2 as well as the KITTI ground truth values.

4.3 Results on KITTI Trajectories

The SIVO pose estimation results are compared to both the ground truth of the KITTI odometry set, as well as the performance of ORB_SLAM2. The trajectories are evaluated using the same metrics as the benchmark[6]. First, both rotation and translational errors for all *subsequences* from lengths 100m to 800m are evaluated. These values are then averaged over the subsequences to provide a final translation error (%) and rotation error (deg/m) for each trajectory. The results for each trajectory also include the total number of map points and keyframes used for each trajectory. We provide a selection of the results in this section, and direct the reader to Appendix C for detailed results on all trajectories.

4.3.1 Summary

Table 4.1 contains a summary of the SIVO experiments for each of the KITTI odometry sequences. For each sequence, the table details the SIVO configuration which performed best, the camera translation estimation error (%) for ORB_SLAM2 and SIVO, the camera rotation estimation error (deg/m) for ORB_SLAM2 and SIVO, and the number of map points used by the two algorithms. This table is ordered by translation error performance compared to ORB_SLAM2.

SIVO outperformed ORB_SLAM2 on KITTI sequence 09 (Section 4.3.2), performed comparably (average of 0.17% translation error difference, 6.2×10^{-5}deg/m rotation error difference) on sequences 00, 02, 05, 07, 08, and 10, and performed worse on sequences 01, 03, 04, and 06 (sequences separated by the midline) while removing, on average, 69% of the map points. The comparable results on 7 out of the 11 odometry sequences indicates that

[6] http://www.cvlibs.net/datasets/kitti/eval_odometry.php

Table 4.1: Translation error, rotation error, and map reduction for ORB_SLAM2 and SIVO on the KITTI dataset.

KITTI Seq.	**ORB Trans. Err. (%)**	**ORB Rot. Err. (deg/m)**	**SIVO Trans. Err. (%)**	**SIVO Rot. Err. (deg/m)**	**ORB Map Points**	**SIVO Map Points**	**SIVO Config.**
09	1.20	4.63E−5	**1.18**	**3.60E−5**	64,442	**18,893**	BS6E2
10	**0.95**	**4.85E−5**	0.97	7.34E−5	33,181	**9,369**	BS2E4
08	**1.15**	**4.89E−5**	1.29	4.98E−5	127,810	**40,574**	BS6E3
05	**0.59**	**2.70E−5**	0.76	2.93E−5	73,463	**22,237**	BS6E3
07	**0.58**	**4.43E−5**	0.80	5.08E−5	29,632	**9,684**	BS6E3
00	**1.18**	**3.88E−5**	1.44	4.68E−5	138,153	**45,875**	BS6E4
02	**1.37**	**3.95E−5**	1.70	4.86E−5	202,293	**58,894**	BS12E4
04	**0.67**	2.20E−5	1.50	**1.97E−5**	21,056	**6,328**	BS12E4
03	**2.72**	**3.75E−5**	4.65	1.29E−4	27,209	**8,449**	BS12E4
01	**1.01**	**3.10E−5**	3.17	9.31E−5	101,378	**37,233**	BS2E4
06	**0.67**	**3.91E−5**	7.10	3.27E−4	47,461	**11,396**	BS6E3

the points removed by SIVO are redundant and the remaining points should be excellent long-term reference points for visual SLAM, although it is not possible to verify feature persistence with the KITTI dataset.

ORB_SLAM2 generally outperformed SIVO in both translation and rotation estimation error. However, the rotation errors for both algorithms are on the magnitude of 10^{-5} deg/m. Over an 800m sequence this corresponds to a total rotation error of 0.008deg, which is negligible. We therefore focus our analysis on the translation estimation performance of the algorithms. We now further examine the results on sequences 09, 00, 08, and 01.

4.3.2 Trajectory 09

The first trajectory selected for further analysis is sequence 09. This route is selected as it is the trajectory where SIVO performs better than ORB_SLAM2 for both the translation and rotation estimates. It is important to note that although this sequence forms a loop, there is no loop closure due to the early termination of the sequence; both ORB_SLAM2 and SIVO require a few more images along the visited region to detect the loop closure. Therefore, the errors include all drift accumulated over the trajectory.

Figure 4.4 shows an overlaid trajectory including the ground truth, the 5 different SIVO configurations, as well as the ORB_SLAM2 localization result for KITTI sequence 09.

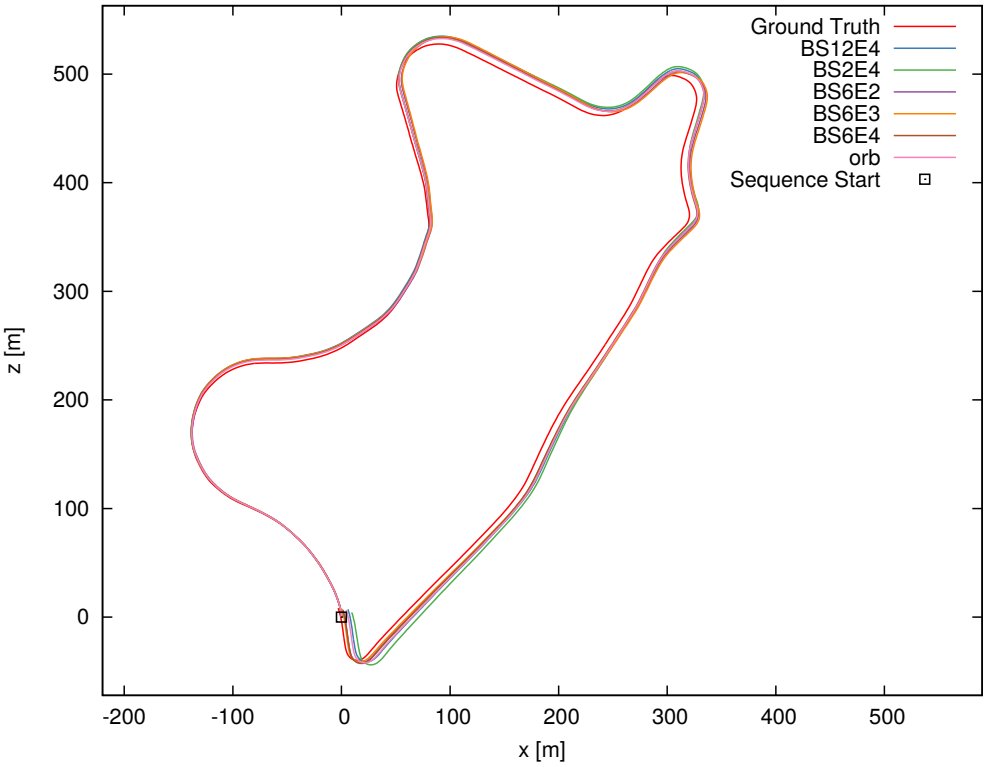


Figure 4.4: Overlaid trajectory results for sequence 09 from the KITTI odometry set comparing various SIVO configurations, ORB_SLAM2, as well as the ground truth.

Table 4.2 compares the keyframes and map points used by the various SIVO configurations on KITTI sequence 09 as well as the translation and rotation errors. SIVO uses 30% fewer keyframes and 70% fewer map points on average. For trajectory 09, we see

that ORB_SLAM2 uses the most map points at 64,442, while the BS2E4 configuration of SIVO uses the least at 17,090. This achieves part of our goal; one argument for our feature selection strategy was to reduce the number of map points in order to maintain the runtime and storage complexity of the algorithm as the robot navigates through its environment.

Algorithm	**Keyframes**	**Map Points**	**Trans. Error** (%)	**Rot. Error** (deg/m)
ORB_SLAM2	608	64,442	1.20	4.63×10^{-5}
BS2E4	448	**17,090**	1.43	4.44×10^{-5}
BS6E2	**342**	18,893	**1.18**	$\mathbf{3.60 \times 10^{-5}}$
BS6E3	375	18,053	1.39	4.83×10^{-5}
BS6E4	487	17,353	1.21	3.90×10^{-5}
BS12E4	473	17,513	1.34	5.31×10^{-5}

Table 4.2: Keyframe and map point comparisons between ORB_SLAM2 and various SIVO configurations on KITTI sequence 09.

SIVO configuration BS6E2 has the best rotation and translation error at 3.60×10^{-5}deg/m and 1.18% respectively, compared to ORB_SLAM2 with rotation and translation errors of 4.63×10^{-5}deg/m and 1.20% respectively. The rotation error discrepancy is negligible, while the translation error difference corresponds to a 16cm difference in localization accuracy over an 800m subsequence.

Figure 4.6 illustrates the translational errors for subsequences of length 100m to 800m on KITTI sequence 09. ORB_SLAM2 has the lowest translation errors over subsequences from 100m to 400m, however SIVO configurations BS6E2, BS6E3, and BS6E4 outperform ORB_SLAM2 at various points over subsequences of length 500m to 800m.

Figure 4.5: Example image from KITTI sequence 09.

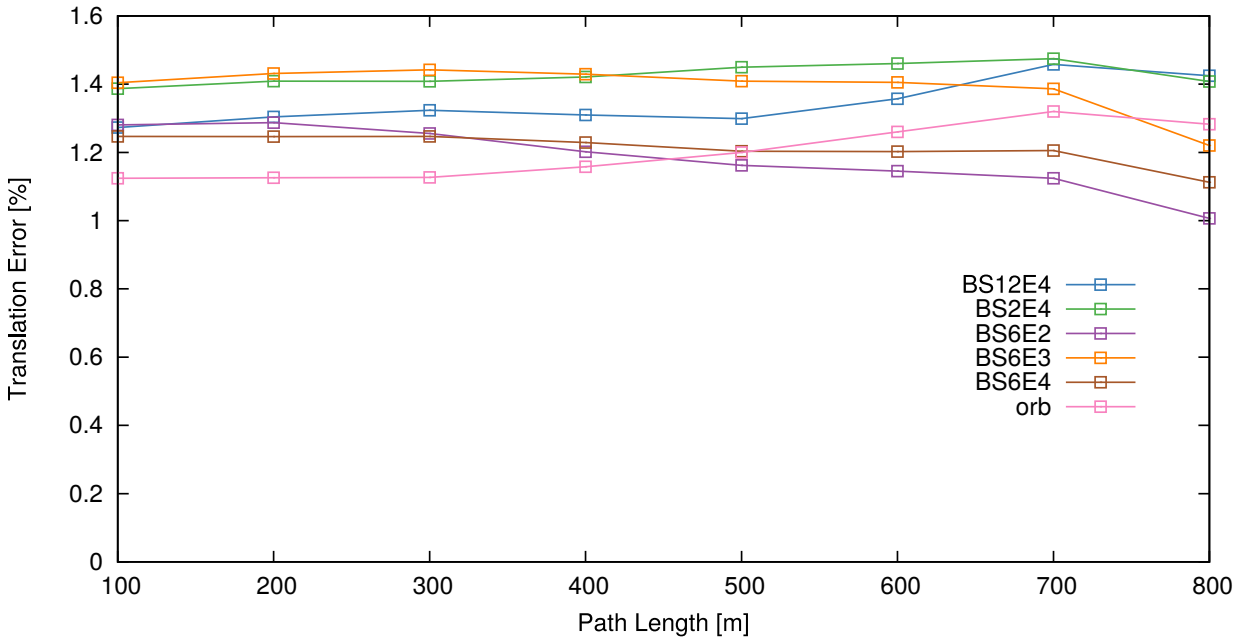


Figure 4.6: Translational errors for subsequences of length 100m to 800m on KITTI sequence 09 comparing various SIVO configurations and ORB_SLAM2 to the ground truth. SIVO BS6E2 outperforms ORB_SLAM2 for this trajectory.

Figure 4.5 illustrates a sample image from the KITTI 09 sequence. Like most of the KITTI odometry sequences, the sequence consists of mostly static objects. However, this sequence contains significantly fewer parked cars. As discussed in Section 4.3.6, removing points on parked cars eliminates close features for the algorithm to use, which results in

worse translation estimates. Sequence 09 has significant stretches with no parked cars in the scene, which results in better feature distribution for SIVO and results in the comparable performance to ORB_SLAM2.

4.3.3 Trajectory 00

The second trajectory selected for further analysis is trajectory 00. This sequence is one of the longest in the KITTI odometry dataset, and it contains multiple loop closures which correct for some of the accumulated drift in the algorithm. Figure 4.7 shows an overlaid trajectory including the ground truth, the 5 different SIVO configurations, as well as the ORB_SLAM2 localization result. We see that there is a significant difference between *all* localization methods and the ground truth, however the various localization methods perform similarly.

Figure 4.7: Overlaid trajectory results for sequence 00 from the KITTI odometry set comparing various SIVO configurations, ORB_SLAM2, as well as the ground truth.

Table 4.3 compares the keyframes and map points used by the various algorithms on KITTI sequence 00 as well as the translation and rotation errors. All algorithms use a similar number of keyframes, but ORB_SLAM2 uses approximately 65% more map points. For trajectory 00, we see that ORB_SLAM2 uses the most map points at 138,153, while the BS12E4 configuration of SIVO uses the least at 45,786.

Algorithm	**Keyframes**	**Map Points**	**Trans. Error** (%)	**Rot. Error** (deg/m)
ORB_SLAM2	1,452	138,153	**1.18**	$\mathbf{3.88 \times 10^{-5}}$
BS2E4	1,304	46,783	1.48	4.31×10^{-5}
BS6E2	**900**	52,550	1.60	4.81×10^{-5}
BS6E3	985	48,307	1.56	4.50×10^{-5}
BS6E4	1,352	45,875	1.44	4.68×10^{-5}
BS12E4	1,356	**45,786**	1.45	4.55×10^{-5}

Table 4.3: Keyframe and map point comparisons between ORB_SLAM2 and various SIVO configurations on KITTI sequence 00. All algorithms use a similar number of keyframes, and SIVO uses approximately 65% more map points on average without drastically compromising localization accuracy.

Removing these map points did have a minor, but adverse effect on localization performance for this trajectory. ORB_SLAM2 has the best rotation and translation error at 3.88×10^{-5}deg/m and 1.18% respectively, while the best SIVO configuration, BS6E4, has a rotation and translation error of 4.68×10^{-5}deg/m and 1.44% respectively. The rotational error discrepancy is negligible, while this translational error difference over an 800m subsequence converts to a further localization error of 2.08m. This is an insignificant decrease in accuracy for a 65% reduction in map size.

Figure 4.8 illustrates the translational errors for subsequences of length 100m to 800m on KITTI sequence 00. ORB_SLAM2 consistently has the lowest error for all subsequences over this trajectory.

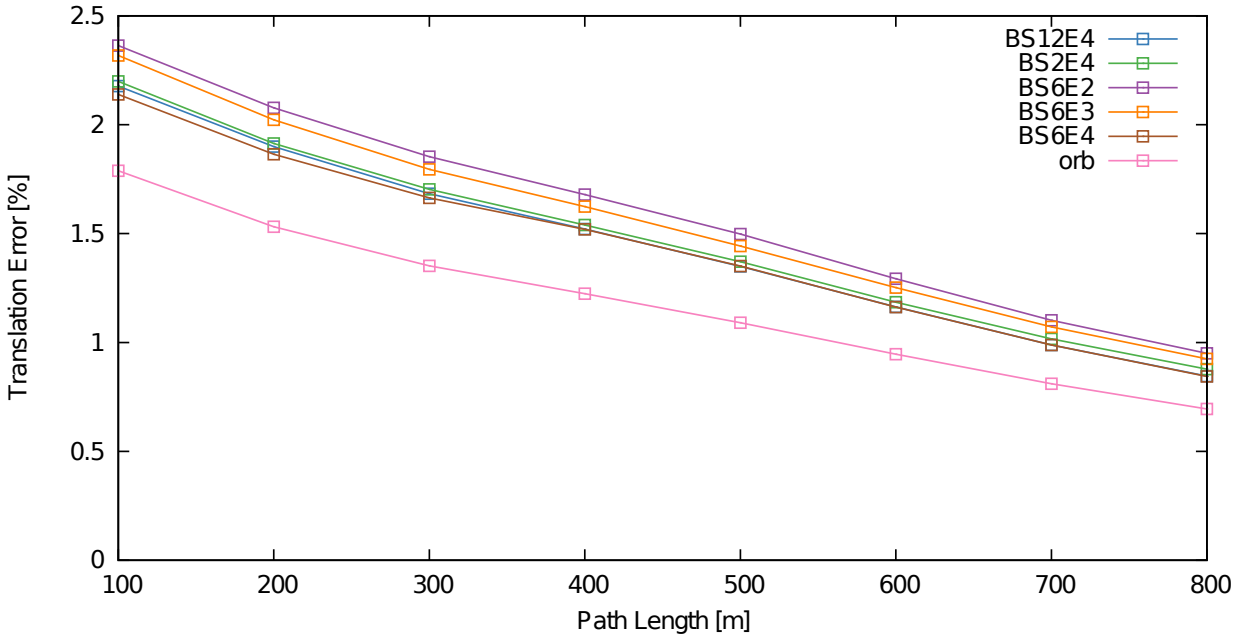


Figure 4.8: Translational errors for subsequences of length 100m to 800m on KITTI sequence 00 comparing various SIVO configurations and ORB_SLAM2 to the ground truth. ORB_SLAM2 has the lowest translation error over all subsequences.

4.3.4 Trajectory 08

The next trajectory selected for further analysis is sequence 08. Similar to sequence 00, this is one of the longer sequences from the odometry dataset. However, this sequence does not have any loop closures, and the results contain all accumulated drift over the duration of the sequence. Figure 4.9 shows an overlaid trajectory including the ground truth, the 5 different SIVO configurations, as well as the ORB_SLAM2 localization result for KITTI sequence 08.

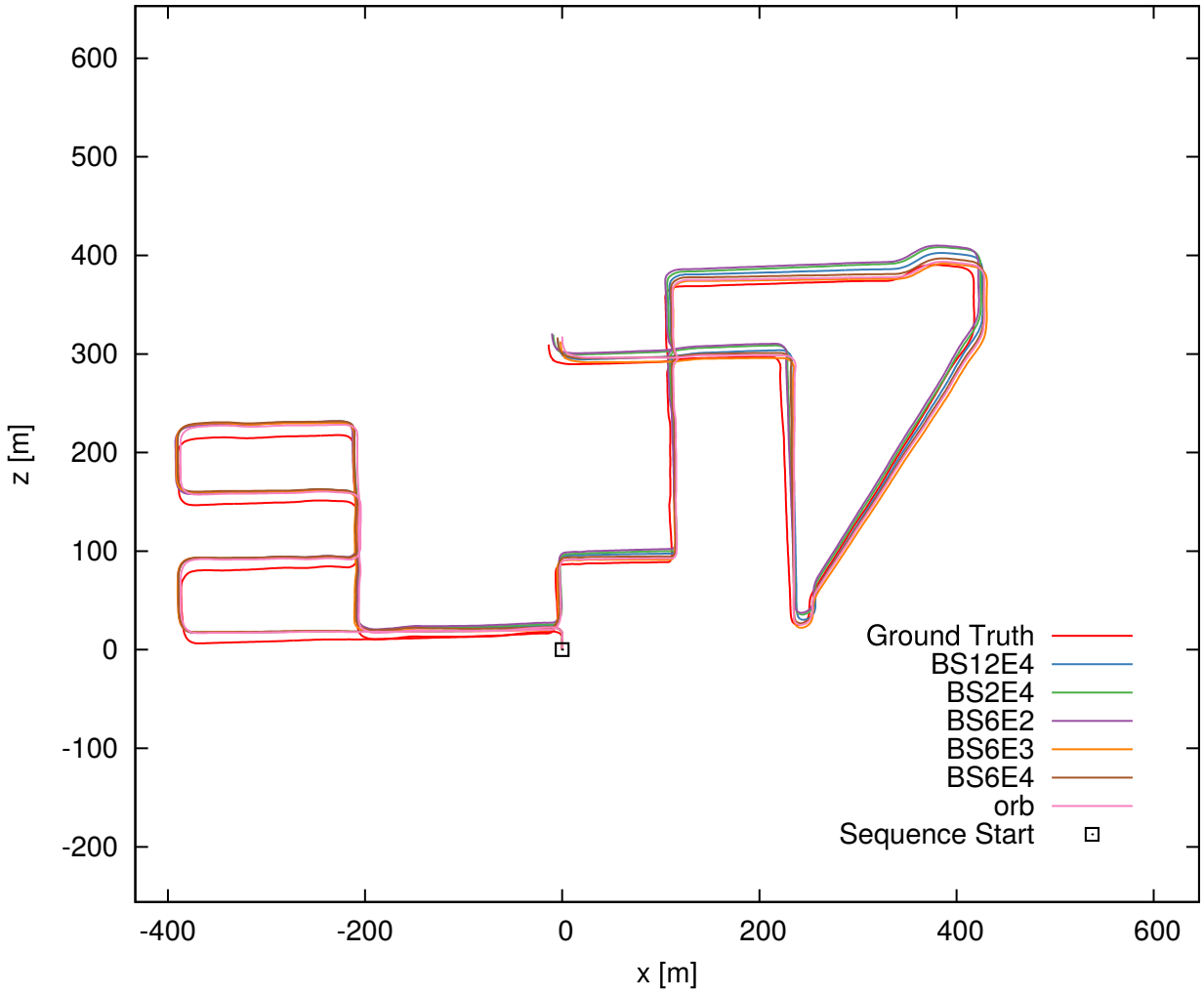


Figure 4.9: Overlaid trajectory results for sequence 08 from the KITTI odometry set comparing various SIVO configurations, ORB_SLAM2, as well as the ground truth.

Table 4.4 compares the keyframes and map points used by the various algorithms on KITTI sequence 08 as well as the translation and rotation errors. SIVO uses 30% fewer keyframes and 70% fewer map points on average. For trajectory 08, we see that ORB_SLAM2 uses the most map points at 127,810, while the BS6E3 configuration of SIVO uses the least at 40,574.

Algorithm	**Keyframes**	**Map Points**	**Trans. Error** (%)	**Rot. Error** (deg/m)
ORB_SLAM2	1,266	127,810	**1.15**	$\mathbf{4.89 \times 10^{-5}}$
BS2E4	953	41,047	1.29	5.01×10^{-5}
BS6E2	**737**	40,833	1.29	5.00×10^{-5}
BS6E3	784	**40,574**	1.29	4.98×10^{-5}
BS6E4	965	40,817	1.30	5.73×10^{-5}
BS12E4	1,001	41,496	1.30	5.20×10^{-5}

Table 4.4: Keyframe and map point comparisons between ORB_SLAM2 and various SIVO configurations on KITTI sequence 08.

For sequence 08, ORB_SLAM2 has the best rotation and translation error at 4.89×10^{-5}deg/m and 1.15% respectively, while the best SIVO configuration, BS6E3, has a rotation and translation error of 4.98×10^{-5}deg/m and 1.29% respectively. The rotation error discrepancy is negligible, while the translation error difference corresponds to a 1.12m discrepancy over an 800m subsequence. Interestingly, this additional translation error is less than the additional error for sequence 00, even without loop closures. Once again, these results are comparable considering the 70% reduction in map size.

Figure 4.10 illustrates the translational errors for subsequences of length 100m to 800m on KITTI sequence 08. ORB_SLAM2 has the lowest translation errors over all subsequences, however all SIVO configurations are very similar.

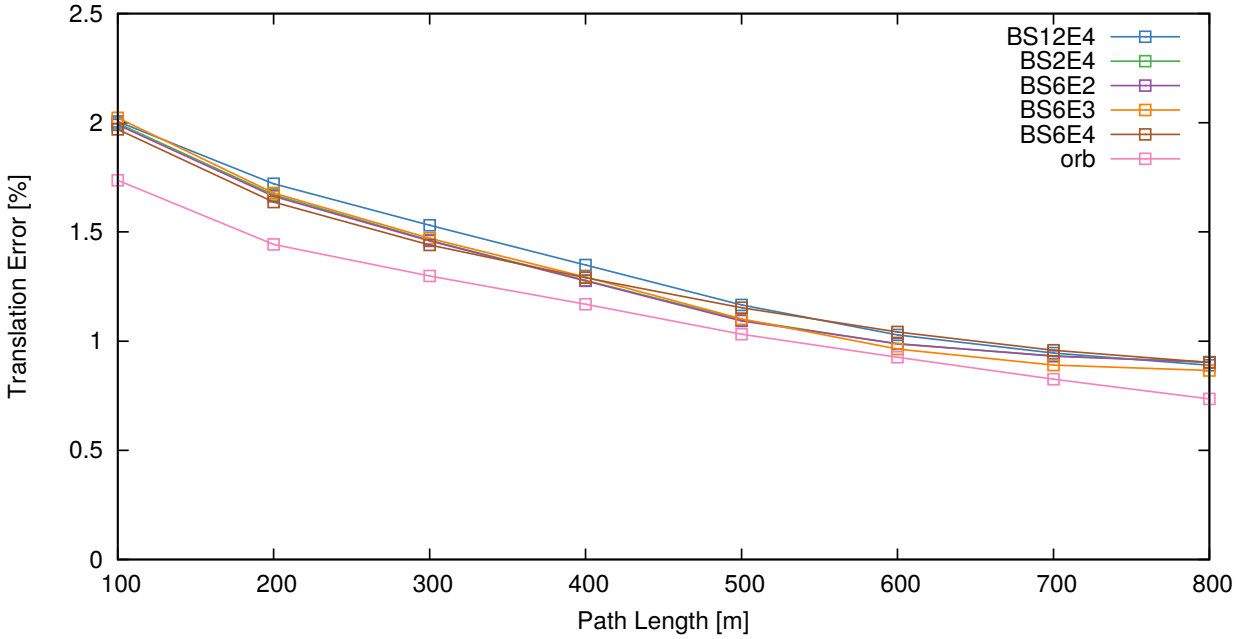


Figure 4.10: Translational errors for subsequences of length 100m to 800m on KITTI sequence 08 comparing various SIVO configurations and ORB_SLAM2 to the ground truth. ORB_SLAM2 has the lowest translation error over all subsequences.

4.3.5 Trajectory 01

The last trajectory selected for further analysis is sequence 01. SIVO performed significantly worse than ORB_SLAM2 on this sequence. This sequence also does not contain any loop closures, implying the final estimation contains all accumulated drift from the sequence. Figure 4.11 shows an overlaid trajectory including the ground truth, the 5 different SIVO configurations, as well as the ORB_SLAM2 localization result for KITTI sequence 01.

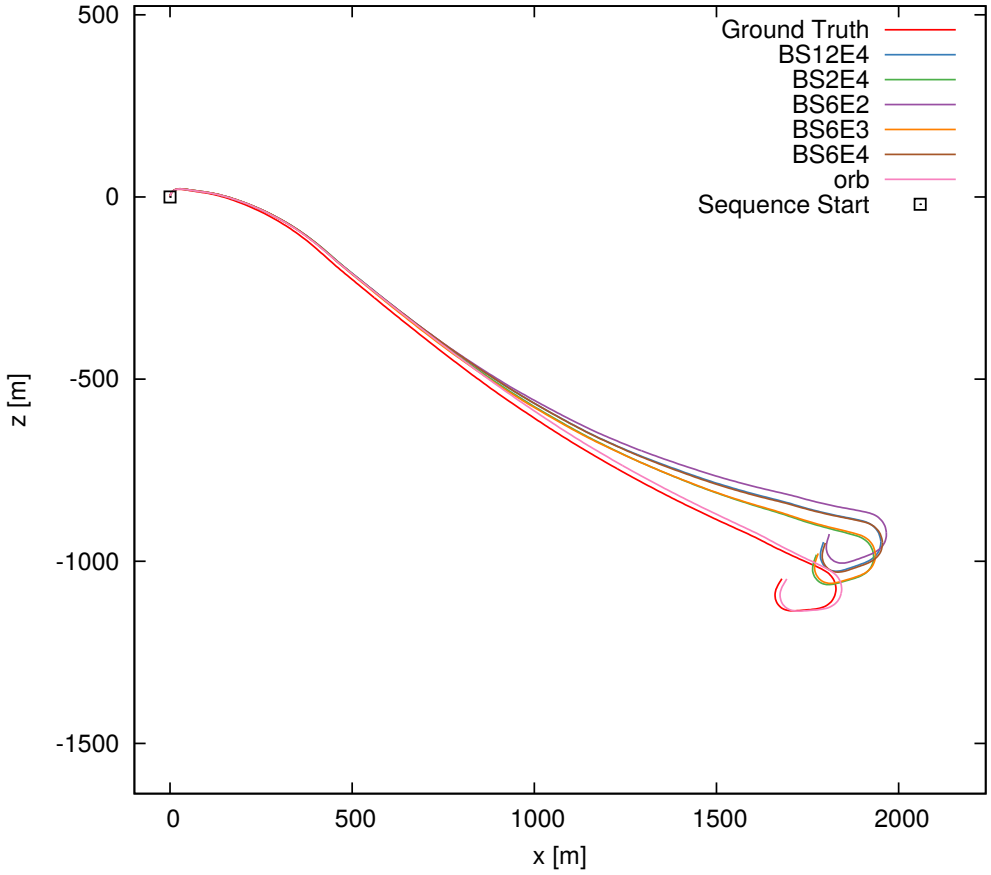


Figure 4.11: Overlaid trajectory results for sequence 01 from the KITTI odometry set comparing various SIVO configurations, ORB_SLAM2, as well as the ground truth.

Table 4.5 compares the keyframes and map points used by the various algorithms on KITTI sequence 01 as well as the translation and rotation errors. SIVO uses approximately

40% fewer keyframes and 65% fewer map points than ORB_SLAM2. For trajectory 01, we see that ORB_SLAM2 uses the most map points at 101,378, while the BS12E4 configuration of SIVO uses the least at 34,015.

Algorithm	**Keyframes**	**Map Points**	**Trans. Error** (%)	**Rot. Error** (deg/m)
ORB_SLAM2	1,038	101,378	**1.01**	$\mathbf{3.10 \times 10^{-5}}$
BS2E4	606	37,233	3.17	9.31×10^{-5}
BS6E2	**593**	35,317	3.85	1.22×10^{-4}
BS6E3	598	35,366	3.35	1.03×10^{-4}
BS6E4	597	34,331	3.72	1.19×10^{-4}
BS12E4	604	**34,015**	3.56	1.17×10^{-4}

Table 4.5: Keyframe and map point comparisons between ORB_SLAM2 and various SIVO configurations on KITTI sequence 01. SIVO uses approximately 40% fewer keyframes and 65% fewer map points than ORB_SLAM2, but performs signficantly worse.

For sequence 01, ORB_SLAM2 has the best rotation and translation error at 3.10×10^{-5}deg/m and 1.01% respectively, while the best SIVO configuration, BS2E4, has a rotation and translation error of 9.31×10^{-5}deg/m and 3.17% respectively. The rotational error discrepancy is again negligible, however the translational error difference over an 800m subsequence converts to a further localization error of 17.28m. Figure 4.12 illustrates the translational errors for subsequences of length 100m to 800m on KITTI sequence 01. Unlike the other trajectories discussed thus far, the translation errors increase for each SIVO configuration as the subsequence length is increased. This matches the results seen in Figure 4.11, which shows significant increasing drift as the vehicle makes its way through the sequence.

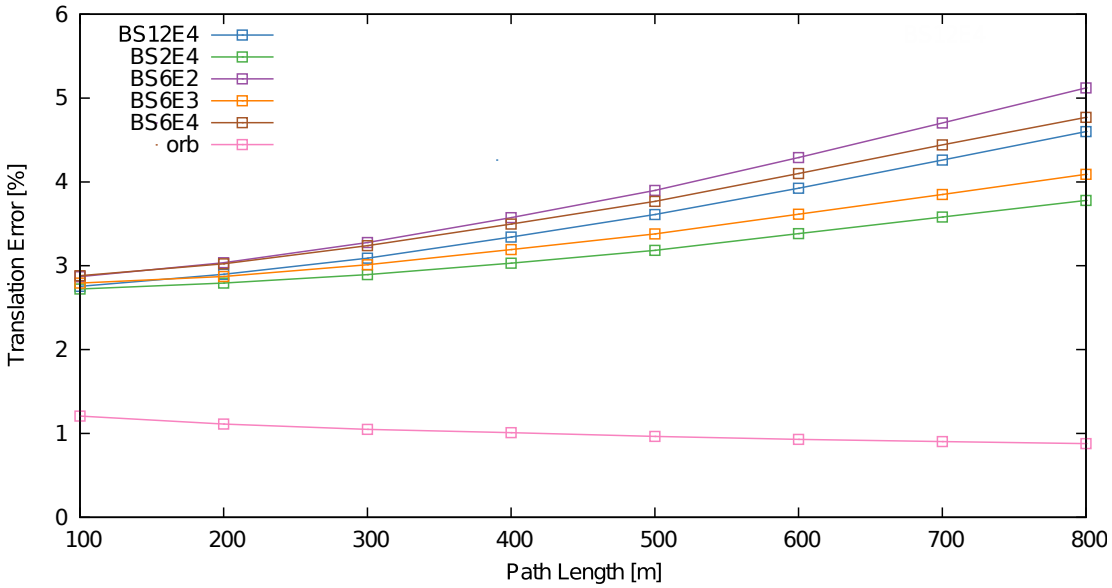


Figure 4.12: Translational errors for subsequences of length 100m to 800m on KITTI sequence 01 comparing various SIVO configurations and ORB_SLAM2 to the ground truth. ORB_SLAM2 has the lowest translation error over all subsequences.

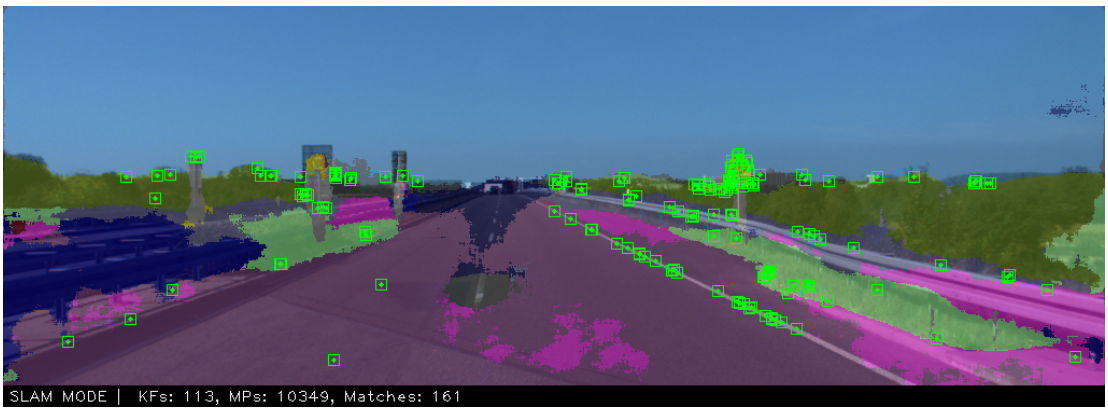


Figure 4.13: Low quality segmentation results on KITTI sequence 01. The highway sequence is not well represented in the semantic dataset, and the network has not generalized for this scenario. Close features such as the highway divider are misclassified as a car and ignored. Therefore, SIVO relies on further features for this trajectory and translation performance suffers.

One reason for the possible poor performance of SIVO for this sequence is poor segmentation; KITTI 01 is the "highway" sequence, which is not well represented in the semantic dataset. For example as seen in Figure 4.13, part of the highway divider as well as the bridge in the distance are misclassified as a car, which is immediately ignored by the algorithm. As these features make up most of the close features for this sequence, SIVO does not receive a well-distributed set of features, causing translation performance to suffer.

4.3.6 Discussion

Overall, our feature selection scheme is a good first approach to incorporating neural network uncertainty into a visual SLAM formulation. In most cases the results between SIVO and ORB_SLAM2 are comparable, with a translation error of 0.17% and a rotation error difference of 6.2×10^{-5}deg/m even when removing, on average, 69% of the map points used by the optimization pipeline. SIVO successfully removed points from the environment which are uninformative and/or dynamic.

In some cases, however, removing these map points did have an adverse effect on localization performance. Some trajectories have significantly worse performance, which can be mostly atrributed to the removal of short range feature points. As discussed in Chapter 3, SIVO immediately removes a point if it has been designated as a dynamic class. However, the KITTI odometry set has been curated to contain mostly static scenes, and the majority of the sequences contain numerous parked cars. These cars make up the majority of close features in the scene. This is illustrated in Figures 4.14a and 4.14b. We know that to accurately estimate camera pose, we require an even distribution of points throughout the image as well as in 3D space. Far points will help with rotation estimation but are poor translation estimates, and close points can help with both. We see the effect of this reflected in the results. For all trajectories, ORB_SLAM2 and SIVO have very comparable, very accurate rotation estimation, but SIVO generally performed worse in estimating translation.

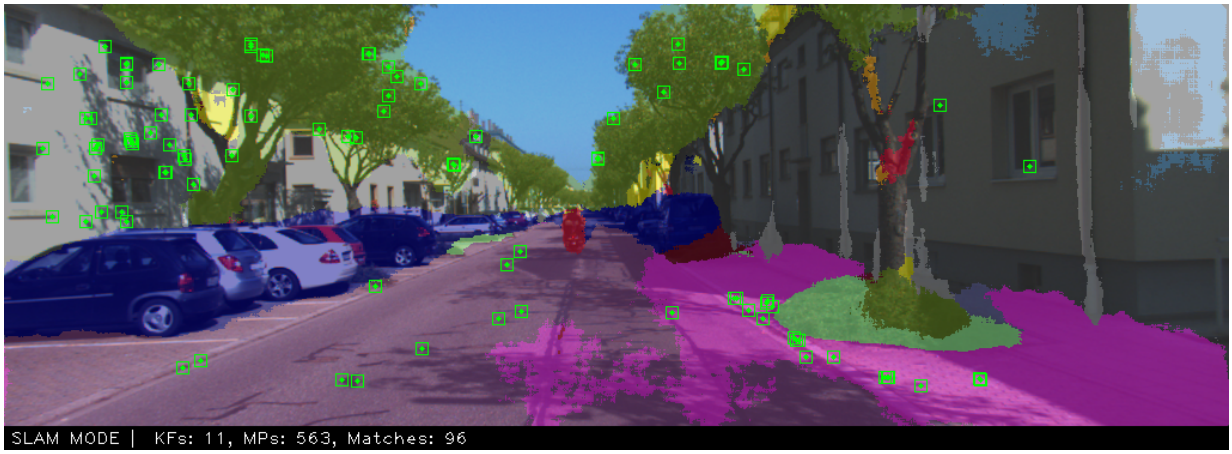
This issue is especially prevalent for trajectories 01, 03, 04, and 06, as they are "straight-line" trajectories. For these sequences, the apparent motion of the features is quite small. The selected features are far away from the vehicle, as is common with the other sequences, however most of the features now lie directly ahead of the vehicle. As the vehicle navigates through the sequence, the features selected remain directly ahead of the vehicle, and their projected location remains fairly consistent. This lack of apparent motion leads to poor rotation and translation estimates.

One interesting observation from Figure 4.14b is the location of the detected features on

various objects in the scene. As discussed, the features we extract from the environment are generally corners, which will typically lie at the boundaries between objects. However, we notice that the neural network struggles the most at distinguishing these edges. This makes sense; the border between objects and the different combinations of objects which can border each other will vary drastically between examples. It is extremely difficult for the neural network to generalize for these cases, and this is clearly illustrated in Figures 4.15a and 4.15b, which illustrate the segmentation and variance results from our trained Bayesian SegNet model on the KITTI semantic validation set. These images clearly demonstrate the observation; the *borders* of objects typically have the highest variance, and it is extremely difficult for the network to correctly determine which pixels belong to which object at these boundaries. This has a significant effect on our feature selection method. These points which lie on the boundaries will be the ones typically extracted by our feature detector, but they will also have a higher entropy and will be discarded by our algorithm. Although our evaluation metric accounts for the value a point provides through evaluation of the mutual information, this could eliminate the few candidates we have, especially in a feature-poor scene.

For a SLAM algorithm, there have been two main methods for outlier rejection which have their pros and cons. The first is to use robust pose estimation, which discards outliers in a multi-stage process. Typically RANSAC [9] will be used as a first outlier rejection step, followed by the use of a robust cost function in the optimization pipeline. The use of a robust cost reduces the impact of outliers in the estimation process. In contrast, the outlier rejection scheme used by SIVO works to carefully select inlier features. In addition to the matching step, we require semantic matches of the feature for over multiple frames. Both of these methods have negative impacts on the estimation. The first method will hit an accuracy ceiling; while the robust cost will mitigate the effect of outliers, the inclusion of incorrect information will still degrade the quality of the optimization. However, our method lowers the feature count and does not have the benefit of averaging all of the feature information. This could eventually lead to the reliance on the same set of points, even if the first method uses more map points.

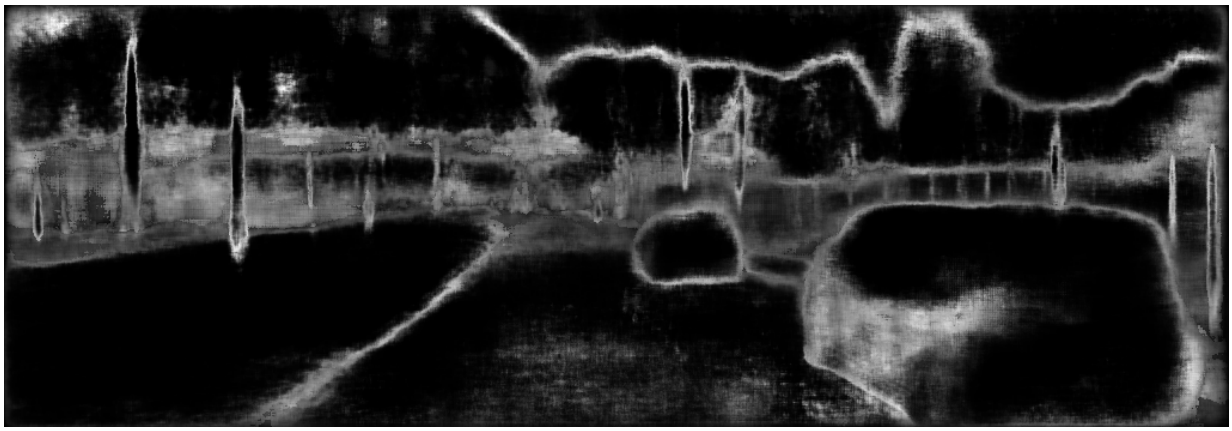
(a) ORB_SLAM2 extracting features on KITTI sequence 00. There are a number of stationary cars which ORB_SLAM2 uses for localization. These points are excellent for short-term localization, but are poor for long-term map construction.



(b) SIVO extracting features on KITTI sequence 00. SIVO ignores the points on the cars as they are deemed as dynamic objects. While this is good for long-term map construction, this inhibits short-term localization.

Figure 4.14: Feature extraction comparison between SIVO and ORB_SLAM2 on KITTI sequence 00.

(a) Semantic segmentation results from the KITTI semantic validation set. This result is the average of 40 stochastic passes of MC dropout.



(b) Variance image results from the KITTI semantic validation set. This result is the average of 40 stochastic passes of MC dropout. White indicates a high variance, while black indicates higher certainty.

Figure 4.15: Semantic segmentation and variance image from the KITTI semantic validation set. This result is the average of 40 stochastic passes of MC dropout. The variance values are determined through the spread of the confidence values output through the softmax layer. White indicates a high variance, while black indicates higher certainty.

Chapter 5

Conclusion

Accurate localization is a requirement for any autonomous mobile robot. Cameras have proven to be a reliable, cheap, and effective sensor to achieve this goal through visual odometry (VO) or visual simultaneous localization and mapping (SLAM) algorithms. In order to accurately determine camera motion, it is important to extract static features from the scene which are viewpoint, scale, and rotation invariant. Due to the emergence of deep learning techniques in recent years, we can easily supplement traditional visual localization methods with context through the use of semantic segmentation.

In this thesis, we present SIVO (Semantically Informed Visual Odometry and Mapping), a novel feature selection algorithm for visual SLAM which fuses together neural network uncertainty with an information-theoretic approach to visual SLAM. While incorporating semantic information into the SLAM algorithm is not novel, to the best of our knowledge there is no algorithm which directly incorporates neural network uncertainty into the probablistic SLAM formulation. Our method selects points which provide significant information to reduce the uncertainty of our state estimate, while ensuring that the feature is detected to be a static object repeatedly, with a high confidence. This is done by evaluating the reduction in Shannon entropy between the current state entropy and the joint entropy of the state given the addition of the new feature with the classification entropy of the feature from the Bayesian neural network.

We evaluated our algorithm against ORB_SLAM2 as well as the GPS ground truth from the KITTI odometry dataset. Our method outperformed ORB_SLAM2 on KITTI sequence 09, and performed comparably well (average of 0.17% translation error difference, 6.2×10^{-5}deg/m rotation error difference) on 6 of the 10 remaining trajectories while removing 69% of the map points used on average. The 4 trajectories with worse performance can

be attributed to a lack of apparent motion in the features due to the straight-line motion of the vehicle, and poor segmentation performance due to dataset limitations discussed in Section 5.1.

Overall, this feature selection scheme is a good first approach to incorporate neural network uncertainty into a information-theoretic visual SLAM formulation. This approach creates sparse maps made up entirely of static objects, which will provide the most benefit in long-term localization. Our method maintains the runtime and storage complexities of the SLAM algorithm by selecting the most informative points in the scene while ensuring they are excellent static long-term references.

5.1 Future Work

While this algorithm has potential, there is significant room for improvement. As discussed in Section 4.3.6, SIVO immediately removes a point if it has been designated as a dynamic class. This poses an issue in evaluating odometry using the KITTI dataset. Although the sequences are mostly static, there are a significant amount of parked cars on the roads. Features from these objects are ignored by the algorithm, however we require an even distribution of points throughout the scene. Points which are close to the camera assist in translation estimation as well as rotation, and the exclusion of these points inhibits the quality of localization. One way to mitigate this effect would be to introduce further context, and determine whether or not a car is static or dynamic. This would allow us to use the features detected on the static cars in a visual odometry solution for local pose estimation, while still ignoring these points in our long-term map. We leave this as future work. One avenue of exploration would be to incorporate these objects into the algorithm for short-term localization, but ignore them for long-term map construction. This would maintain our goal to select static long-term references, while resulting in better localization estimates.

Another area for future work lies in further parsing the network output. Most of the features extracted from the environment are corners, which will typically lie on the edges of an object. This is the area that a neural network will struggle to classify the most. Borders are difficult to generalize, and uncertainty has been shown to be higher in these regions (see Section 4.3.6). Our algorithm will penalize these points due to their high classification entropy, but removing these points can be quite detrimental in feature poor areas. One method of improving this could be to determine the class of both objects along the border, and penalize this point less if the border is between two static objects. For example, if we

have the border between a traffic sign and a building, either one of the points would be an excellent candidate, and the point should still be selected.

Although our map should mostly consist of points that are valid long-term references, this is difficult to verify due to the limitations of the datasets available to us for autonomous driving. We are unable to test the performance of relocalization using our long-term map, as the dataset scenarios are unchanging. At the University of Waterloo, we have collected significant amounts of data using our autonomous driving research platform, Autonomoose[1]. However, this data does not have semantic ground truth, and the Cityscapes/KITTI-trained neural network did not generalize well to this data. To verify long-term localization with SIVO, we require a "long-term dataset" which drives the same route at different times in the day (or different days) with accurate ground truth as well as detailed segmentation annotation. To the best of our knowledge, there is currently no dataset which meets this requirement. Our use of the Cityscapes and KITTI datasets was an attempt to achieve this by amalgamating one dataset with detailed segmentation information, and another with limited segmentation and detailed odometry information. Poor segmentation definitely affected the results for some of the trajectories, especially sequences 01 (see Figure 4.13) and 06. With the demand for data and the rapid development of deep learning approaches in visual SLAM, this will not be a limitation in the near future.

[1] https://www.autonomoose.net/

References

[1] T. Kos, I. Markezic, and J. Pokrajcic, "Effects of multipath reception on gps positioning performance," in *International Symposium ELMAR*. IEEE, 2010, pp. 399–402.

[2] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1. IEEE, 2004, pp. I–I.

[3] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision (IJCV)*, vol. 60, no. 2, pp. 91–110, 2004.

[4] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European Conference on Computer Vision (ECCV)*, 2006, pp. 404–417.

[5] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European Conference on Computer Vision (ECCV)*, 2006, pp. 430–443.

[6] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 2564–2571.

[7] M. Cummins and P. Newman, "Fab-map: Probabilistic localization and mapping in the space of appearance," *The International Journal of Robotics Research (IJRR)*, vol. 27, no. 6, pp. 647–665, 2008.

[8] M. J. Milford and G. F. Wyeth, "Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2012, pp. 1643–1649.

[9] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[10] R. Mur-Artal and J. D. Tardós, "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.

[11] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012, pp. 3354–3361.

[12] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?" in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 5580–5590.

[13] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, *et al.*, "The limits and potentials of deep learning for robotics," *The International Journal of Robotics Research (IJRR)*, vol. 37, no. 4-5, pp. 405–420, 2018.

[14] G. Sibley, "A sliding window filter for slam," *University of Southern California, Technical Report*, 2006.

[15] G. Dissanayake, H. Durrant-Whyte, and T. Bailey, "A computationally efficient solution to the simultaneous localisation and map building (slam) problem," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2. IEEE, 2000, pp. 1009–1014.

[16] S. Hochdorfer and C. Schlegel, "Landmark rating and selection according to localization coverage: Addressing the challenge of lifelong operation of slam in service robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2009, pp. 382–387.

[17] A. J. Davison, "Active search for real-time vision," in *IEEE International Conference on Computer Vision (ICCV)*, vol. 1. IEEE, 2005, pp. 66–73.

[18] S. Zhang, L. Xie, and M. D. Adams, "Entropy based feature selection scheme for real time simultaneous localization and map building," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2005, pp. 1175–1180.

[19] M. Kaess and F. Dellaert, "Covariance recovery from a square root information matrix for data association," *Robotics and Autonomous Systems*, vol. 57, no. 12, pp. 1198–1210, 2009.

[20] S. Choudhary, V. Indelman, H. I. Christensen, and F. Dellaert, "Information-based reduced landmark slam," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 4620–4627.

[21] A. Das and S. L. Waslander, "Entropy based keyframe selection for multi-camera visual slam," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 3676–3681.

[22] V. Ila, J. M. Porta, and J. Andrade-Cetto, "Information-based compact pose slam," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 78–93, 2010.

[23] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison, "Slam++: Simultaneous localisation and mapping at the level of objects," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2013, pp. 1352–1359.

[24] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE, 2011, pp. 127–136.

[25] P. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 2, pp. 239–256, 1992.

[26] S. L. Bowman, N. Atanasov, K. Daniilidis, and G. J. Pappas, "Probabilistic data association for semantic slam," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1722–1729.

[27] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[28] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *IEEE Intelligent Vehicles Symposium*, Baden-Baden, Germany, June 2011.

[29] L. An, X. Zhang, H. Gao, and Y. Liu, "Semantic segmentation–aided visual odometry for urban autonomous driving," *International Journal of Advanced Robotic Systems (IJARS)*, vol. 14, no. 5, p. 1729881417735667, 2017.

[30] D. Nistér, "Preemptive ransac for live structure and motion estimation," *Machine Vision and Applications*, vol. 16, no. 5, pp. 321–329, 2005.

[31] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 39, no. 12, pp. 2481–2495, 2017.

[32] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 4, 2017.

[33] V. Murali, H. P. Chiu, S. Samarasekera, and R. T. Kumar, "Utilizing semantic visual landmarks for precise vehicle navigation," *arXiv preprint arXiv:1801.00858*, 2018.

[34] H. P. Chiu, M. Sizintsev, X. S. Zhou, P. Miller, S. Samarasekera, and R. Kumar, "Sub-meter vehicle navigation using efficient pre-mapped visual landmarks," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2016, pp. 505–512.

[35] G. J. Brostow, J. Shotton, J. Fauqueur, and R. Cipolla, "Segmentation and recognition using structure from motion point clouds," in *European Conference on Computer Vision (ECCV)*, 2008, pp. 44–57.

[36] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, 2009.

[37] X. Li and R. Belaroussi, "Semi-dense 3d semantic mapping from monocular slam," *arXiv preprint arXiv:1611.04144*, 2016.

[38] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European Conference on Computer Vision (ECCV)*, 2014, pp. 834–849.

[39] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 40, no. 4, pp. 834–848, 2018.

[40] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[41] J. Schönberger, M. Pollefeys, A. Geiger, and T. Sattler, "Semantic visual localization," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[42] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[43] E. Stenborg, C. Toft, and L. Hammarstrand, "Long-term visual localization using semantically segmented images," *arXiv preprint arXiv:1801.05269*, 2018.

[44] N. Radwan, A. Valada, and W. Burgard, "Vlocnet++: Deep multitask learning for semantic visual localization and odometry," *arXiv preprint arXiv:1804.08366*, 2018.

[45] N. D. Reddy, P. Singhal, V. Chari, and K. M. Krishna, "Dynamic body vslam with semantic constraints," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 1897–1904.

[46] A. Kendall, V. Badrinarayanan, and R. Cipolla, "Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding," *arXiv preprint arXiv:1511.02680*, 2015.

[47] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *ACM International Conference on Multimedia*. ACM, 2014, pp. 675–678.

[48] P. Furgale, "Representing robot pose: The good, the bad, and the ugly," Workshop: Lessons Learned from Building Complex Systems, IEEE International Conference on Robotics and Automation (ICRA), 2014, accessed: 2018-07-25. [Online]. Available: http://paulfurgale.info/s/Workshop-Rotations_v102key.pdf

[49] M. Bloesch, H. Sommer, T. Laidlow, M. Burri, G. Nuetzi, P. Fankhauser, D. Bellicoso, C. Gehring, S. Leutenegger, M. Hutter, *et al.*, "A primer on the differential calculus of 3d orientations," *arXiv preprint arXiv:1606.05285*, 2016.

[50] C. E. Rasmussen, "Gaussian processes in machine learning," in *Advanced lectures on machine learning*, 2004, pp. 63–71.

[51] T. D. Barfoot, *State Estimation for Robotics*. Cambridge University Press, 2017.

[52] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge University Press, 2003.

[53] D. Gálvez-López and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, 2012.

[54] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2881–2890.

[55] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 3213–3223.

[56] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[57] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3431–3440.

[58] Y. Gal, "Uncertainty in deep learning," Ph.D. dissertation, University of Cambridge, 2016.

[59] Y. Gal and Z. Ghahramani, "Bayesian convolutional neural networks with bernoulli approximate variational inference," *arXiv preprint arXiv:1506.02158*, 2015.

[60] ——, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *International Conference on Machine Learning (ICML)*, 2016, pp. 1050–1059.

[61] ——, "Dropout as a bayesian approximation: Appendix," *arXiv preprint arXiv:1506.02157*, 2015.

[62] R. M. Neal, "Bayesian learning for neural networks," Ph.D. dissertation, University of Toronto, 1995.

[63] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research (JMLR)*, vol. 15, no. 1, pp. 1929–1958, 2014.

[64] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[65] M. Chli, "Applying information theory to efficient slam," Ph.D. dissertation, Department of Computing, Imperial College London, 2010.

[66] D. MacKay, *Information theory, inference and learning algorithms*. Cambridge University Press, 2003.

[67] A. Malinin and M. Gales, "Predictive uncertainty estimation via prior networks," *arXiv preprint arXiv:1802.10501*, 2018.

[68] E. Eade, "Lie groups for 2d and 3d transformations," 2013, accessed: 2018-08-02. [Online]. Available: http://ethaneade.com/lie.pdf

[69] J.-L. Blanco, "A tutorial on se(3) transformation parameterizations and on-manifold optimization," *University of Malaga, Tech. Rep*, vol. 3, 2010.

[70] C. Harris and M. Stephens, "A combined corner and edge detector." in *Alvey vision conference*, vol. 15, no. 50. Citeseer, 1988, pp. 10–5244.

[71] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*, 2010, pp. 778–792.

APPENDICES

Appendix A

Fundamentals

This section will explain in detail some fundamental background material, including Lie theory, computer vision basics, as well as probability basics.

A.1 Lie Groups and Lie Algebra

Throughout this work, we mentioned that rotation matrices are members of the Special Orthogonal Lie group in 3D, $\mathbb{SO}(3)$, and transformation matrices are members of the Special Euclidean Lie group in 3D, $\mathbb{SE}(3)$. This section will quickly introduce some background on these groups.

A *Lie group* is a smooth, differentiable manifold which is also a topological group [68]. Each Lie group has an associated *Lie algebra*, which represents the *tangent space* to the manifold at the identity element of the Lie group. Similar to the tangent of any function, the tangent space is an approximation of the manifold about a particular point. A tangent space can be generated at any element of the Lie group, however it is only the tangent space at the *identity* element which is the Lie algebra. All tangent spaces are *isomorphic* to the Lie algebra, which allows us to perform calculations on the Lie algebra and use the *adjoint* related identity to transform the results to our tangent space of interest. Lie group and Lie algebra theory allows for convenient composition, inversion, differentiation, and interpolation of 3D transformations, all of which are crucial for SLAM [68].

A.1.1 $\mathbb{SO}(3)$: Special Orthogonal Lie Group in 3D

The Lie group $\mathbb{SO}(3)$ represents all rotation matrices in 3D. It is defined as

$$\mathbb{SO}(3) = \{\mathbf{R} \in \mathbb{R}^{3\times 3} : \mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I}^{3\times 3}, \det(\mathbf{R}) = 1\} \tag{A.1}$$

Its associated Lie algebra is denoted as $\mathfrak{so}(3)$.

$$\mathfrak{so}(3) = \{\mathbf{A} \in \mathbb{R}^{3\times 3} : \mathbf{A} = -\mathbf{A}^T\} \tag{A.2}$$

We see that this is the set of all 3×3 skew-symmetric matrices. In a similar manner to $\mathbb{R}^3$, where we have 3 basis vectors to define the axes, the Lie algebra $\mathfrak{so}(3)$ has 3 basis *generators* [68].

$$\mathbf{G}_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{G}_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_3 = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{A.3}$$

As we only require 3 values to define a skew-symmetric matrix, we follow the notation of Barfoot [51] and introduce the $(\cdot)^\wedge$ operator to map elements of $\mathbb{R}^3$ to skew-symmetric matrices. The inverse of this operator is $(\cdot)^\vee$. This provides us with a vector space in which we can easily perform calculations.

$$\boldsymbol{\omega}^\wedge = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}^\wedge = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{A.4}$$

Linear combinations of the generators can create elements of $\mathfrak{so}(3)$. This representation has a physical equivalent that we are familiar with; each of the generators represents one of the degrees of rotation: roll (ϕ), pitch (ψ), and yaw (θ). These are *Euler angles*, which we will use throughout this work to parameterize rotation matrices. As the Lie algebra represents the tangent space of $\mathbb{SO}(3)$, it follows that the 3 elements of $\boldsymbol{\omega}$ represent an angular velocity for roll, pitch, and yaw.

Exponential Map and Log Map

The exponential map and log map are the operators which relate the Lie group to its associated Lie algebra. We denote the exponential map by [49]

$$\exp : \mathfrak{so}(3) \mapsto \mathbb{SO}(3) \tag{A.5}$$

and it is equal to the exponential of the skew-symmetric matrix. Through use of Taylor expansion and the Rodrigues formula, this results in [51, 68].

$$\begin{aligned}\mathbf{R} = \exp(\boldsymbol{\omega}^\wedge) &= \sum_{n=0}^{\infty} \frac{1}{n!}(\boldsymbol{\omega}^\wedge)^n \\ &= \mathbf{I}^{3\times 3} + \left(\frac{\sin(\theta)}{\theta}\right)\boldsymbol{\omega}^\wedge + \left(\frac{1-\cos(\theta)}{\theta^2}\right)(\boldsymbol{\omega}^\wedge)^2 \\ &\approx \mathbf{I}^{3\times 3} + \boldsymbol{\omega}^\wedge\end{aligned} \tag{A.6}$$

where $\theta^2 = \boldsymbol{\omega}^T\boldsymbol{\omega}$. The log map is denoted by

$$\log : \mathbb{SO}(3) \mapsto \mathfrak{so}(3) \tag{A.7}$$

and is the inverse of the exponential map. This is similarly defined by the Rodrigues rotation formula

$$\log(\mathbf{R}) = \frac{\theta}{2\sin(\theta)}(\mathbf{R} - \mathbf{R}^T) \tag{A.8}$$

where $\theta = \cos^{-1}\left(\frac{\mathrm{tr}(\mathbf{R})-1}{2}\right)$.

Group Operations

Unlike a vector space such as $\mathbb{R}^N$, Lie groups have no addition or subtraction operators. In fact, the only operation that is defined is the *group* operation

$$\circ : \mathbb{SO}(3) \times \mathbb{SO}(3) \mapsto \mathbb{SO}(3) \tag{A.9}$$

which is just a simple matrix multiplication (composition) between two elements of $\mathbb{SO}(3)$ [49]. We therefore define the $\boxplus$ operator to facilitate addition.

$$\begin{aligned}\boxplus : \mathbb{SO}(3) \times \mathfrak{so}(3) &\to \mathbb{SO}(3) \\ \mathbf{R}, \boldsymbol{\omega}^\wedge &\mapsto \exp\left(\boldsymbol{\omega}^\wedge\right) \circ \mathbf{R}\end{aligned} \tag{A.10}$$

Similarly, we can define the $\boxminus$ operator to define subtraction.

$$\begin{aligned}\boxminus : \mathbb{SO}(3) \times \mathbb{SO}(3) &\to \mathfrak{so}(3) \\ \mathbf{R}_1, \mathbf{R}_2 &\mapsto \log\left(\mathbf{R}_1 \circ \mathbf{R}_2^{-1}\right)\end{aligned} \tag{A.11}$$

These tools allow us to blend together orientations (elements of $\mathbb{SO}(3)$) with differences of orientations (tangent elements on $\mathfrak{so}(3)$) with ease.

Adjoint

As stated above, all tangent spaces are *isomorphic* to the Lie algebra. This allows us to perform calculations on the Lie algebra, and then transform the results to our point of interest. This transformation is done via the adjoint-related identity. As defined by Eade, the adjoint "linearly and exactly transforms tangent vectors from one tangent space to another" [68]. For $\mathbf{R} \in \mathbb{SO}(3)$ and $\boldsymbol{\omega} \in \mathbb{R}^3$, this is defined by

$$\mathbf{R} \circ \exp(\boldsymbol{\omega}^\wedge) = \exp(\mathrm{Adj}_{\mathbf{R}} \cdot \boldsymbol{\omega}^\wedge) \cdot \mathbf{R} \tag{A.12}$$

The above equation indicates that

$$\exp(\mathrm{Adj}_{\mathbf{R}} \cdot \boldsymbol{\omega}^\wedge) = \mathbf{R} \cdot \exp(\boldsymbol{\omega}^\wedge) \cdot \mathbf{R}^{-1} \tag{A.13}$$

By replacing $\exp(\boldsymbol{\omega}^\wedge)$ with the generators of $\mathfrak{so}(3)$, Eade [68] shows that the adjoint for $\mathbb{SO}(3)$ is just

$$\mathrm{Adj}_{\mathbf{R}} = \mathbf{R} \tag{A.14}$$

This means that for $\mathbb{SO}(3)$, the adjoint transformation of an element is defined by the rotation matrix of the element itself. Rotating a tangent vector will result in the rotated tangent at the element of interest on the manifold.

A.1.2 $\mathbb{SE}(3)$: Special Euclidean Lie Group in 3D

The Lie group $\mathbb{SE}(3)$ represents all rigid body transformations in 3D. It is defined as

$$\mathbb{SE}(3) = \{\mathbf{T} \in \mathbb{R}^{4\times4} : \left[\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0}^{1\times3} & 1 \end{array}\right] : \mathbf{R} \in \mathbb{SO}(3), \mathbf{t} \in \mathbb{R}^3\} \tag{A.15}$$

Its associated Lie algebra is denoted as $\mathfrak{se}(3)$.

$$\mathfrak{se}(3) = \{\mathbf{B} \in \mathbb{R}^{4\times4} : \left[\begin{array}{c|c} \boldsymbol{\omega}^\wedge & \mathbf{u} \\ \hline \mathbf{0}^{1\times3} & 0 \end{array}\right] : \mathbf{u} \in \mathbb{R}^3, \boldsymbol{\omega}^\wedge \in \mathbb{R}^{3\times3}, \boldsymbol{\omega}^\wedge = -(\boldsymbol{\omega}^\wedge)^T\} \tag{A.16}$$

There are 6 basis generators for $\mathfrak{se}(3)$, which represent the 6 degrees of freedom for a rigid body.

$$\begin{aligned}
\mathbf{G}_1 &= \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_2 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \\
\mathbf{G}_4 &= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_5 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{G}_6 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}
\end{aligned} \tag{A.17}$$

Elements of $\mathfrak{se}(3)$ are created through linear combinations of the 6 generators. Our element of $\mathfrak{se}(3)$, a matrix $\in \mathbb{R}^{4\times4}$, can be parameterized by 6 values: 3 for the skew-symmetric matrix $\boldsymbol{\omega}^\wedge$, and 3 for the vector $\mathbf{u}$. We now denote a *twist* vector, $\boldsymbol{\xi}$,

$$\boldsymbol{\xi} = \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\omega} \end{bmatrix} \tag{A.18}$$

where $\mathbf{u}$ represents a velocity along the 3 axes of $\mathbb{R}^3$, (x, y, z), and as before, $\boldsymbol{\omega}$ represents an angular velocity in roll, pitch, and yaw. We denote the $(\cdot)^\wedge$ operator for $\mathfrak{se}(3)$ to map elements of $\mathbb{R}^6$ to an element of $\mathfrak{se}(3)$. The inverse of this operator is $(\cdot)^\vee$

$$\boldsymbol{\xi}^\wedge = \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\omega} \end{bmatrix}^\wedge = \left[\begin{array}{c|c} \boldsymbol{\omega}^\wedge & \mathbf{u} \\ \hline \mathbf{0}^{1\times3} & 0 \end{array}\right] \tag{A.19}$$

Exponential Map and Log Map

The exponential map for this Lie group is defined as

$$\exp : \mathfrak{se}(3) \mapsto \mathbb{SE}(3) \tag{A.20}$$

and it is equal to the exponent of a linear combination of the generators [68]. This can be expressed by [51, 68, 69]

$$\begin{aligned}
\mathbf{T} = \exp(\boldsymbol{\xi}^\wedge) &= \sum_{n=0}^{\infty} \frac{1}{n!} (\boldsymbol{\xi}^\wedge)^n \\
&= \begin{bmatrix} \exp(\boldsymbol{\omega}^\wedge) & \mathbf{V}\mathbf{u} \\ \mathbf{0}^{1\times3} & 1 \end{bmatrix} \\
&\approx \mathbf{I}^{4\times4} + \boldsymbol{\xi}^\wedge
\end{aligned} \tag{A.21}$$

where $\exp(\boldsymbol{\omega}^\wedge)$ is as seen in Equation A.6, $\mathbf{V}$ is defined by

$$\mathbf{V} = \mathbf{I}^{3\times 3} + \frac{1-\cos(\theta)}{\theta^2}\boldsymbol{\omega}^\wedge + \frac{\theta - \sin(\theta)}{\theta^3}(\boldsymbol{\omega}^\wedge)^2 \tag{A.22}$$

and $\theta = \boldsymbol{\omega}^T\boldsymbol{\omega}$, again as seen in Equation A.6.

The log map is defined by

$$\log : \mathbb{SE}(3) \mapsto \mathfrak{se}(3) \tag{A.23}$$

and is the inverse of the exponential map. The log map for $\mathbb{SE}(3)$ is defined in two sections; one for the rotation aspect, and a second for the translation. Recall that $\boldsymbol{\xi} = \begin{bmatrix}\mathbf{u}\\ \boldsymbol{\omega}\end{bmatrix}$, and that $\mathbf{T} = \left[\begin{array}{c|c}\mathbf{R} & \mathbf{t}\\ \hline \mathbf{0}^{1\times 3} & 1\end{array}\right] \in \mathbb{SE}(3)$. The two sections are defined by [68, 69]

$$\begin{aligned}\boldsymbol{\omega} &= (\log(\mathbf{R}))^\vee\\ \mathbf{u} &= \mathbf{V}^{-1}\mathbf{t}\end{aligned} \tag{A.24}$$

where $\mathbf{V}$ is defined in Equation A.22.

Group Operations

The $\mathbb{SE}(3)$ Lie group also does not have an addition or subtraction operator, and the group operation is defined by

$$\circ : \mathbb{SE}(3) \times \mathbb{SE}(3) \mapsto \mathbb{SE}(3) \tag{A.25}$$

We define the $\boxplus$ operator for addition,

$$\begin{aligned}\boxplus : \mathbb{SE}(3) \times \mathfrak{se}(3) &\to \mathbb{SE}(3)\\ \mathbf{T}, \boldsymbol{\xi}^\wedge &\mapsto \exp(\boldsymbol{\xi}^\wedge) \circ \mathbf{T}\end{aligned} \tag{A.26}$$

and the $\boxminus$ operator for subtraction.

$$\begin{aligned}\boxminus : \mathbb{SE}(3) \times \mathbb{SE}(3) &\to \mathfrak{se}(3)\\ \mathbf{T}_1, \mathbf{T}_2 &\mapsto \log\left(\mathbf{T}_1 \circ \mathbf{T}_2^{-1}\right)\end{aligned} \tag{A.27}$$

Adjoint

For $\mathbf{T} \in \mathbb{SE}(3)$ and $\boldsymbol{\xi} \in \mathbf{R}^6$, the adjoint is similarly calculated from the generators just like $\mathbb{SO}(3)$ [68]

$$\mathbf{T} \cdot \exp(\boldsymbol{\xi}^\wedge) = \exp(\mathrm{Adj}_{\mathbf{T}} \cdot \boldsymbol{\xi}^\wedge) \cdot \mathbf{T} \tag{A.28}$$

which indicates that

$$\exp(\mathrm{Adj}_{\mathbf{T}} \cdot \boldsymbol{\xi}^\wedge) = \mathbf{T} \cdot \exp(\boldsymbol{\xi}^\wedge) \cdot \mathbf{T}^{-1} \tag{A.29}$$

Replacing $\exp(\boldsymbol{\xi}^\wedge)$ with a linear combination of the 6 generators yields the following result for $\mathrm{Adj}_{\mathbf{T}}$ [68].

$$\mathrm{Adj}_{\mathbf{T}} = \left[\begin{array}{c|c} \mathbf{R} & \mathbf{t}^\wedge\mathbf{R} \\ \hline \mathbf{0}^{3\times3} & \mathbf{R} \end{array}\right] \in \mathbb{R}^{6\times6} \tag{A.30}$$

A.2 Probability

SLAM is effectively the process of tracking a probability distribution through time. This section will cover some basics of probability which are useful for this work.

A.2.1 Notation and Random Variables

Let us denote two random variables, $X, Y \in \mathbb{R}$. We will denote the probability of a random variable by $p(\cdot)$. This work will examine the probabilities of both discrete and continuous variables.

Discrete Variables

In the event that the elements of X are *discrete*, then $p(X)$ is described by a *probability mass function* (pmf). Let us imagine that X can take on N values. This would be represented by

$$X \in \{x_1, x_2, \ldots, x_N\} \tag{A.31}$$

The notation $p(X = x_i)$ or $p(x_i)$ represents the probability of X taking on the value of x_i.

Continuous Variables

In the event that the elements of Y are *continuous*, then $p(Y)$ is described by a *probability density function* (pdf), such as Equation 2.6. We denote the probability that Y takes on a certain value as $p(Y = y)$, or $p(y)$.

A.2.2 Mean, Variance, and Covariance

The *mean* of a random variable is the expected value. This is denoted by

$$\mu = E[X] \tag{A.32}$$

For a discrete random variable, this is equal to

$$\mu = \sum_{i=1}^{n} x_i p(x_i) \tag{A.33}$$

and for a continuous random variable, is expressed by

$$\mu = \int x p(x) dx \tag{A.34}$$

The variance is a measure of the dispersion of a random variable from its mean. This can be expressed by

$$\sigma^2 = E[(X - \mu)^2] \tag{A.35}$$

which, in the discrete case results in

$$\sigma^2 = \sum_{i=1}^{n} (x_i - \mu)^2 p(x_i) \tag{A.36}$$

and for the continuous case, is expressed by

$$\sigma^2 = \int (x - \mu)^2 p(x) dx \tag{A.37}$$

The variance is the square of the standard deviation, which similarly illustrates the dispersion of a random variable. While the variance illustrates the dispersion for a variable $\in \mathbb{R}$,

we require a *covariance* matrix to describe the dispersion of a variable $\in \mathbb{R}^N$. Covariance a measure of variability for 2 random variables, and is defined by

$$cov(x, y) = E[(x - \mu_x)(y - \mu_y)] \tag{A.38}$$

For a multidimensional variable, we use a covariance matrix to describe the variability between each pair of variables. In the 2D example, this is written as

$$\Sigma = \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 \end{bmatrix} \tag{A.39}$$

where $\sigma_{xx}^2, \sigma_{yy}^2$ are the variance of each of the individual variables, and the off-diagonal elements represent the *cross-correlation*. If X and Y are independent, the off-diagonal elements will be zero.

A.2.3 Joint Probability

The joint probability is the probability of multiple variables. From our original case, this would be written as

$$p(X, Y) \tag{A.40}$$

and can be extended to any number of variables. If the two random variables are independent, then

$$p(X, Y) = p(X)p(Y) \tag{A.41}$$

A.2.4 Conditional Probability

The conditional probability describes the probability of a random variable given knowledge of another, usually *dependent* variable.

$$p(X = x|Y = y) = p(x|y) \tag{A.42}$$

If X and Y are independent, then

$$p(x|y) = p(x) \tag{A.43}$$

as any new knowledge of variable Y does not affect variable X. Lastly, conditional probability relates to joint probability by the following

$$p(x|y) = \frac{p(x, y)}{p(y)} \tag{A.44}$$

A.2.5 Likelihood and Posterior

One last concept to discuss in this section is the idea of a *likelihood* and *posterior* distribution. The posterior distribution is the probability distribution of a random quantity, *conditional* on evidence. That is, we must have some *prior* knowledge about the parameters for the distribution describing our random variable, and we take that information into account. This is described by

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} \tag{A.45}$$

where θ represents the parameters of our distribution (for example μ, σ in the Gaussian case), and x represents the evidence, or *realizations*, of our random variable. In contrast, the likelihood function represents expected evidence from a distribution, based on the value of the parameters. This is represented by the first numerator term in Equation A.45, or

$$\mathcal{L}(\theta; x) = p(x|\theta) \tag{A.46}$$

For multiple measurements of the same variable, this is represented by

$$\mathcal{L}(\theta|x_1, \ldots, x_n) = p(x_1, \ldots, x_n) = p(x_1|\theta) \cdot p(x_2|\theta) \cdots p(x_n|\theta) = \prod_{i=1}^{n} p(x_i|\theta) \tag{A.47}$$

which is crucial to formulating the graph-based implementation of a SLAM equation. The last term of interest in Equation A.45 is the second term of the numerator, which represents any *prior* knowledge of the parameters; this is aptly referred to as the prior. For the sake of completeness, we note that the denominator is a normalization term.

MAP and MLE

For SLAM, we are trying to determine the best estimate of our robot pose and map point positions through knowledge of uncertain measurement information. Depending on the information we have available, we will try to determine the *maximum a posteriori* (MAP) or *maximum likelihood* (ML) estimate.

The MAP estimate indicates the most probable value for the parameters of the posterior distribution, based off of all the information we have [51]. This is equal to the *mode* of the posterior distribution, which for a Gaussian distribution, is equal to the mean. The ML estimate is very closely related to the MAP estimate, however there is no prior information

involved; the ML estimate is a special case of the MAP estimate where we can assume a *uniform* prior is applied to the parameters. The MAP estimate is denoted by

$$\hat{\theta}_{MAP}(x) = \underset{\theta}{\operatorname{argmax}}\, p(x|\theta) \tag{A.48}$$

and the ML estimate is denoted by

$$\hat{\theta}_{ML}(x) = \underset{\theta}{\operatorname{argmax}}\, \mathcal{L}(\theta; x) \tag{A.49}$$

As we are assuming our camera poses are Gaussian in nature, the MAP or ML estimate is effectively calculating the most probable value for our poses (the parameter mean), given all of our measurements (realizations of our pose).

A.3 Computer Vision

This section will outline some fundamentals for computer vision which this work builds upon.

A.3.1 Oriented Fast and Rotated BRIEF (ORB) Features

In a visual SLAM application, camera motion is determined through the apparent motion of points in the scene. These points in 3D pass through the projection function above, and their 2D representations are used as *feature measurements*. As this work focuses on selecting better features from the environment, we will first discuss how our baseline points of interest are selected.

The front end for most visual SLAM algorithms consists of a feature *detector*, and a *descriptor*. The goal of the detector is to determine which aspects of the image are the most important and distinctive, while the goal of the descriptor is to efficiently encode this information into a representation which can be easily compared against in multiple frames. The most common features selected for visual SLAM are *corners* from various points in the scene. Harris [70] was one of the first to use corners as salient image features, and this has inspired a wide variety of work including SIFT [3], SURF [4], FAST [5], and ORB [6]. Although we are referring to these detectors as "corner" detectors, in reality they are selecting all image areas with a sufficient *gradient* in multiple directions [3]. The Oriented Fast and Rotated BRIEF, or ORB, feature detector and descriptor was created in order to improve on performance of SIFT and SURF, primarily with regards to speed [6].

The ORB feature detector is based off of the FAST corner detector proposed by Rosten and Drummond [5] (Figure A.1). A point is deemed a corner if there are n contiguous points in the surrounding circle which are all brighter or darker than the candidate point. While Figure A.1 illustrates a scenario where $n = 12$, ORB uses a variant which only

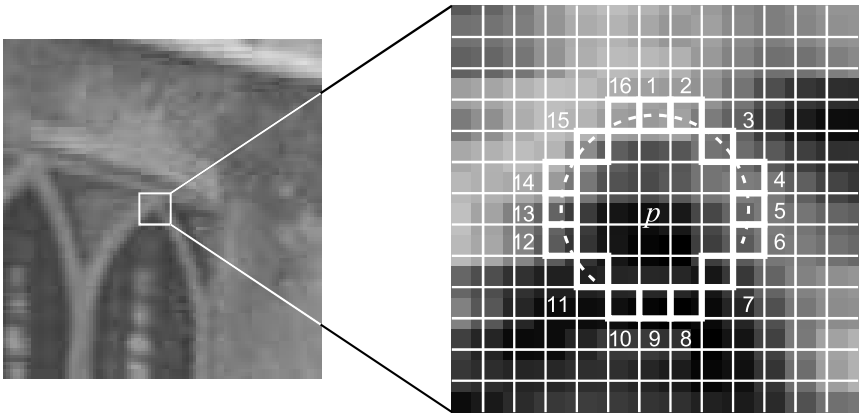


Figure A.1: Corner detection using the FAST corner detector with 12 contiguous points required to deem the point a corner. Image retrieved from Rosten and Drummond [5].

requires 9 contiguous points. The authors further supplement this method to include multi-scale features and calculate an *intensity centroid* to ensure the selected points are rotation-invariant.

The ORB descriptor builds on the work of Calonder et al. [71], Binary Robust Independent Elementary Features (BRIEF), which is a 256 bit *binary* vector constructed from intensity comparisons between two points. This was selected as it is extremely efficient to compute and construct, and the ORB descriptor improves on this work by ensuring it is rotation invariant.

Overall, this detection and description scheme drastically outperformed the state of the art in terms of speed without compromising quality. It has since been used as the primary feature detector for several Visual SLAM algorithms, such as ORB_SLAM [10, 27].

Appendix B

SIVO Jacobians

This section will outline the Jacobian matrices required to determine the covariance of a measurement. Recall from Chapter 3 that in order to calculate the mutual information between the state and measurement, we need to construct the marginal covariance matrix between the two quantities. This matrix, $\hat{\mathbf{\Sigma}}_i$, is defined by

$$\hat{\mathbf{\Sigma}}_i = \begin{bmatrix} \mathbf{\Sigma}_{\mathbf{xx}} & \mathbf{\Sigma}_{\mathbf{x}\mathbf{z}_i} \\ \mathbf{\Sigma}_{\mathbf{z}_i\mathbf{x}} & \mathbf{\Sigma}_{\mathbf{z}_i\mathbf{z}_i} \end{bmatrix} = \begin{bmatrix} \mathbf{\Sigma}_t & \mathbf{\Sigma}_t \frac{\partial h_i}{\partial \mathbf{x}}^T \\ \frac{\partial h_i}{\partial \mathbf{x}} \mathbf{\Sigma}_t & \frac{\partial h_i}{\partial \mathbf{x}} \mathbf{\Sigma}_t \frac{\partial h_i}{\partial \mathbf{x}}^T + \mathbf{Q}_i \end{bmatrix}$$

The Jacobian of interest is the value $\frac{\partial h_i}{\partial \mathbf{x}}$.

B.1 Jacobian of Rectified Stereo Projection

This section introduces the Jacobian of the rectified stereo projection function with respect to the point in the camera frame. Our measurement model is the rectified stereo projection model, where we assume that the transformation between the right and left cameras is defined by a horizontal translation equivalent to the baseline. This is defined by the function π_s

$$h_i = \pi_s({}_c\mathbf{p}) = \pi_s \begin{pmatrix} {}_cx \\ {}_cy \\ {}_cz \end{pmatrix} = \begin{pmatrix} f_x \frac{{}_cx}{{}_cz} + c_x \\ f_y \frac{{}_cy}{{}_cz} + c_y \\ f_x \frac{({}_cx - b)}{{}_cz} + c_x \end{pmatrix} \tag{B.1}$$

where x, y, z represent the x, y, and z-coordinates of the point in the camera frame ${}_c\mathbf{p}$, f_x, f_y, c_x, c_y represent the camera intrinsic matrix parameters, and b is the baseline between stereo cameras.

The Jacobian of the projection function with respect to the point is defined by

$$\frac{\partial \pi_s}{\partial {}_c\mathbf{p}} = \begin{bmatrix} \frac{f_x}{{}_cz} & 0 & -f_x\frac{{}_cx}{{}_cz^2} \\ 0 & \frac{f_y}{{}_cz} & -f_y\frac{{}_cy}{{}_cz^2} \\ \frac{f_x}{{}_cz} & 0 & -f_x\frac{{}_cx-b}{{}_cz^2} \end{bmatrix} \tag{B.2}$$

B.2 Jacobian of the Transformed Point

This section introduces the Jacobian matrix of the transformed point (in the camera frame) with respect to the state. Recall that our state at any given point in time, $\mathbf{x}$, can also be represented as the pose of the camera frame with respect to the world frame, or $\mathbf{T}_{cw}$. This is represented by

$$\frac{\partial \mathbf{T}_{cw}{}_w\mathbf{p}}{\partial \mathbf{T}_{cw}} \tag{B.3}$$

with $\mathbf{T}_{cw} \in \mathbb{SE}(3)$, and ${}_w\mathbf{p} \in \mathbb{R}^3$. As our state is an element of $\mathbb{SE}(3)$, it is common practice to first linearize about this quantity using *perturbation theory* [51], and then take the derivative with respect to the linearized point. Therefore, let us define a twist element $\boldsymbol{\xi} \in \mathbb{R}^6$, with its skew symmetric form $\boldsymbol{\xi}^\wedge \in \mathfrak{se}(3)$. We can therefore rewrite the Jacobian as

$$\frac{\partial (\mathbf{T}_{cw} \boxplus \boldsymbol{\xi}^\wedge){}_w\mathbf{p}}{\partial \boldsymbol{\xi}} = \frac{\partial \exp(\boldsymbol{\xi}^\wedge)\mathbf{T}_{cw}{}_w\mathbf{p}}{\partial \boldsymbol{\xi}} \tag{B.4}$$

Blanco [69] shows that this Jacobian is equal to

$$\frac{\partial (\mathbf{T}_{cw} \boxplus \boldsymbol{\xi}^\wedge){}_w\mathbf{p}}{\partial \boldsymbol{\xi}} = \left(\mathbf{I}^{\mathbf{3}\times\mathbf{3}} | -{}_c\mathbf{p}^\wedge\right) = \begin{bmatrix} 1 & 0 & 0 & 0 & {}_cz & -{}_cy \\ 0 & 1 & 0 & -{}_cz & 0 & {}_cx \\ 0 & 0 & 1 & {}_cy & -{}_cx & 0 \end{bmatrix} \tag{B.5}$$

B.3 Full Projection Jacobian

We can now compose the above results together to obtain the full Jacobian of the projection function with respect to the state. This is equal to

$$\frac{\partial h_i}{\partial \mathbf{x}} = \frac{\partial \pi_s({}_c\mathbf{p})}{\partial \mathbf{T}_{cw}} = \frac{\partial \pi_s}{\partial {}_c\mathbf{p}} \cdot \frac{\partial \exp(\boldsymbol{\xi}^\wedge)\mathbf{T}_{cw}{}_w\mathbf{p}}{\partial \boldsymbol{\xi}} \tag{B.6}$$

Combining Equations B.2 and B.5 yields

$$\frac{\partial h_i}{\partial \mathbf{x}} = \frac{\partial \pi_s}{\partial {}_c\mathbf{p}} \cdot \frac{\partial \exp(\boldsymbol{\xi}^\wedge)\mathbf{T}_{cw}\mathbf{p}}{\partial \boldsymbol{\xi}} = \begin{bmatrix} \frac{f_x}{{}_cz} & 0 & -f_x\frac{{}_cx}{{}_cz^2} \\ 0 & \frac{f_y}{{}_cz} & -f_y\frac{{}_cy}{{}_cz^2} \\ \frac{f_x}{{}_cz} & 0 & -f_x\frac{{}_cx-b}{{}_cz^2} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & {}_cz & -{}_cy \\ 0 & 1 & 0 & -{}_cz & 0 & {}_cx \\ 0 & 0 & 1 & {}_cy & -{}_cx & 0 \end{bmatrix}$$

$$\frac{\partial h_i}{\partial \mathbf{x}} = \boxed{\begin{bmatrix} \frac{f_x}{{}_cz} & 0 & -f_x\frac{{}_cx}{{}_cz^2} & -f_x\frac{{}_cx{}_cy}{{}_cz^2} & f_x\left(1+\frac{{}_cx^2}{{}_cz^2}\right) & -f_x\frac{{}_cy}{{}_cz} \\ 0 & \frac{f_y}{{}_cz} & -f_y\frac{{}_cy}{{}_cz^2} & -f_y\left(1+\frac{{}_cy^2}{{}_cz^2}\right) & f_y\frac{{}_cx{}_cy}{{}_cz^2} & f_y\frac{{}_cx}{{}_cz} \\ \frac{f_x}{{}_cz} & 0 & -f_x\frac{{}_cx-b}{{}_cz^2} & -f_x\frac{({}_cx-b){}_cy}{{}_cz^2} & f_x\left(1+\frac{{}_cx({}_cx-b)}{{}_cz^2}\right) & -f_x\frac{{}_cy}{{}_cz} \end{bmatrix}} \tag{B.7}$$

Using this Jacobian, we can now construct the marginal covariance matrix required to evaluate the mutual information between the state and any feature measurement.

B.4 Motion Model Propagation

Once we have optimized for our state and map estimates, we can extract the state covariance. To initialize the state and covariance for the next timestep, we must propagate our state through the motion model. Both SIVO and ORB_SLAM2 use a constant-velocity motion model for state propagation. In order to determine the transform to the next timestep, we look at the *previous* transformation, and apply this to the current state. Let us imagine we have 3 states: $\mathbf{T}_{cw}^{t-1}$, $\mathbf{T}_{cw}^{t}$, and $\mathbf{T}_{cw}^{t+1}$, where t indicates the current timestep. We are looking to apply our motion model to provide a prediction for $\mathbf{T}_{cw}^{t+1}$. We will now slightly abuse notation, and denote a twist vector between our two previous states, $\boldsymbol{\xi} \in \mathfrak{se}(3)$ as

$$\boldsymbol{\xi} = \mathbf{T}_{cw}^{t} \boxminus \mathbf{T}_{cw}^{t-1} = \log\left(\mathbf{T}_{cw}^{t}(\mathbf{T}_{cw}^{t-1})^{-1}\right) \tag{B.8}$$

Note that in contrast to the example above, $\boldsymbol{\xi}$ is an element of $\mathfrak{se}(3)$, instead of $\mathbb{R}^6$. The constant-velocity motion model then applies this transformation to the *current* state to determine the next state. That is,

$$\mathbf{T}_{cw}^{t+1} = \mathbf{T}_{cw}^{t} \boxplus \boldsymbol{\xi} \tag{B.9}$$

Substituting Equation B.8 into B.9 yields

$$\begin{aligned} \mathbf{T}_{cw}^{t+1} &= \mathbf{T}_{cw}^{t} \boxplus \boldsymbol{\xi} \\ &= \exp(\boldsymbol{\xi})\mathbf{T}_{cw}^{t} \\ &= \exp\big(\log\big(\mathbf{T}_{cw}^{t}(\mathbf{T}_{cw}^{t-1})^{-1}\big)\big)\mathbf{T}_{cw}^{t} \\ &= \mathbf{T}_{cw}^{t}(\mathbf{T}_{cw}^{t-1})^{-1}\mathbf{T}_{cw}^{t} \\ \therefore \mathbf{T}_{cw}^{t+1} &= f(\mathbf{T}_{cw}^{t}) = f(\mathbf{x}) \end{aligned} \tag{B.10}$$

As discussed previously, our current state, $\mathbf{x}$, is equivalent to the pose of the camera frame with respect to the world frame, $\mathbf{T}_{cw}^{t}$. We define our motion model as $f(\mathbf{x})$.

We know that our current state also has a covariance matrix associated with it, $\boldsymbol{\Sigma}_t$. It is well known that the propagation of the covariance matrix through the motion model is defined by

$$\boldsymbol{\Sigma}_{t+1} = \mathbf{F}\boldsymbol{\Sigma}_t\mathbf{F}^T + \mathbf{R}_t \tag{B.11}$$

where $\mathbf{F}$ represents the Jacobian of the motion model with respect to the current state, and $\mathbf{R}_t$ represents the random noise associated with the motion model.

B.4.1 Jacobian of Motion Model Propagation

We will now determine the Jacobian of our motion model propagation with respect to the current state, $\mathbf{F} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial f(\mathbf{T}_{cw}^{t})}{\partial \mathbf{T}_{cw}^{t}}$. We will apply first principles and define this derivative using the limit [49].

$$\begin{aligned} \frac{\partial \mathbf{T}_{cw}^{t+1}}{\partial \mathbf{T}_{cw}^{t}}\frac{\partial f(\mathbf{T}_{cw}^{t})}{\partial \mathbf{T}_{cw}^{t}} = \mathbf{F} &= \lim_{\boldsymbol{\xi}\to\mathbf{0}} \frac{(\mathbf{T}_{cw}^{t} \boxplus \boldsymbol{\xi})(\mathbf{T}_{cw}^{t-1})^{-1}(\mathbf{T}_{cw}^{t} \boxplus \boldsymbol{\xi}) \boxminus \mathbf{T}_{cw}^{t}(\mathbf{T}_{cw}^{t-1})^{-1}\mathbf{T}_{cw}^{t}}{\boldsymbol{\xi}} \\ &= \lim_{\boldsymbol{\xi}\to\mathbf{0}} \frac{\exp(\boldsymbol{\xi})\mathbf{T}_{cw}^{t}(\mathbf{T}_{cw}^{t-1})^{-1}\exp(\boldsymbol{\xi})\mathbf{T}_{cw}^{t} \boxminus \mathbf{T}_{cw}^{t}(\mathbf{T}_{cw}^{t-1})^{-1}\mathbf{T}_{cw}^{t}}{\boldsymbol{\xi}} \end{aligned} \tag{B.12}$$

To simplify the equation, let us define $\mathbf{A} = \mathbf{T}_{cw}^{t}(\mathbf{T}_{cw}^{t-1})^{-1}\mathbf{T}_{cw}^{t}$, and $\mathbf{B} = \mathbf{T}_{cw}^{t}(\mathbf{T}_{cw}^{t-1})^{-1}$. We can now rewrite Equation B.12 as

$$\mathbf{F} = \lim_{\boldsymbol{\xi}\to\mathbf{0}} \frac{\exp(\boldsymbol{\xi})\mathbf{B}\exp(\boldsymbol{\xi})\mathbf{T}_{cw}^{t} \boxminus \mathbf{A}}{\boldsymbol{\xi}} \tag{B.13}$$

We can apply the adjoint related identity (see Section A.1.2) in order to move the twist vector to another tangent space. This is applied as follows

$$\mathbf{B}\exp(\boldsymbol{\xi}) = \exp(\mathrm{Adj}_{\mathbf{B}}\xi)\mathbf{B} \tag{B.14}$$

Substituting Equation B.14 into B.13 yields

$$\mathbf{F} = \lim_{\boldsymbol{\xi} \to \mathbf{0}} \frac{\exp(\boldsymbol{\xi}) \exp(\mathrm{Adj}_{\mathbf{B}} \boldsymbol{\xi}) \mathbf{B} \mathbf{T}_{cw}^{t} \boxminus \mathbf{A}}{\boldsymbol{\xi}} \tag{B.15}$$

Recall that $\mathbf{B} = \mathbf{T}_{cw}^{t} (\mathbf{T}_{cw}^{t-1})^{-1}$, therefore $\mathbf{B}\mathbf{T}_{cw}^{t} = \mathbf{T}_{cw}^{t} (\mathbf{T}_{cw}^{t-1})^{-1} \mathbf{T}_{cw}^{t} = \mathbf{A}$. We substitute this into Equation B.15 and apply the $\boxminus$ operator.

$$\begin{aligned} \mathbf{F} &= \lim_{\boldsymbol{\xi} \to \mathbf{0}} \frac{\log(\exp(\boldsymbol{\xi}) \exp(\mathrm{Adj}_{\mathbf{B}} \boldsymbol{\xi}) \mathbf{A}\mathbf{A}^{-1})}{\boldsymbol{\xi}} \\ &= \lim_{\boldsymbol{\xi} \to \mathbf{0}} \frac{\log(\exp(\boldsymbol{\xi}) \exp(\mathrm{Adj}_{\mathbf{B}} \boldsymbol{\xi}))}{\boldsymbol{\xi}} \end{aligned} \tag{B.16}$$

We apply a 0^{th} order Baker-Campbell-Hausdorff (BCH) approximation [51], which states that $\log(\exp(\mathbf{A}) \exp(\mathbf{B})) = \mathbf{A}+\mathbf{B}$. Applying this to Equation B.16 yields our final Jacobian

$$\begin{aligned} \mathbf{F} &= \lim_{\boldsymbol{\xi} \to \mathbf{0}} \frac{\boldsymbol{\xi} + \mathrm{Adj}_{\mathbf{B}} \boldsymbol{\xi}}{\boldsymbol{\xi}} \\ \therefore \boxed{\mathbf{F} &= \mathbf{I}^{\mathbf{6} \times \mathbf{6}} + \mathrm{Adj}_{\mathbf{B}}} \end{aligned} \tag{B.17}$$

which is used to propagate the current state covariance, $\boldsymbol{\Sigma}_t$ to the next timestep, $\boldsymbol{\Sigma}_{t+1}$.

Appendix C

Detailed Results on the KITTI Dataset

This section will present detailed results on the KITTI trajectories. Results on sequences 00, 01, 08, and 09 can be found in Chapter 4, while this section contains the results for sequences 02, 03, 04, 05, 06, 07, and 10.

C.1 Trajectory 02

Figure C.1 shows an overlaid trajectory including the ground truth, the 5 different SIVO configurations, as well as the ORB_SLAM2 localization result for KITTI sequence 02. Similarly to trajectory 00, ORB_SLAM2 outperforms SIVO, but not by a significant margin.

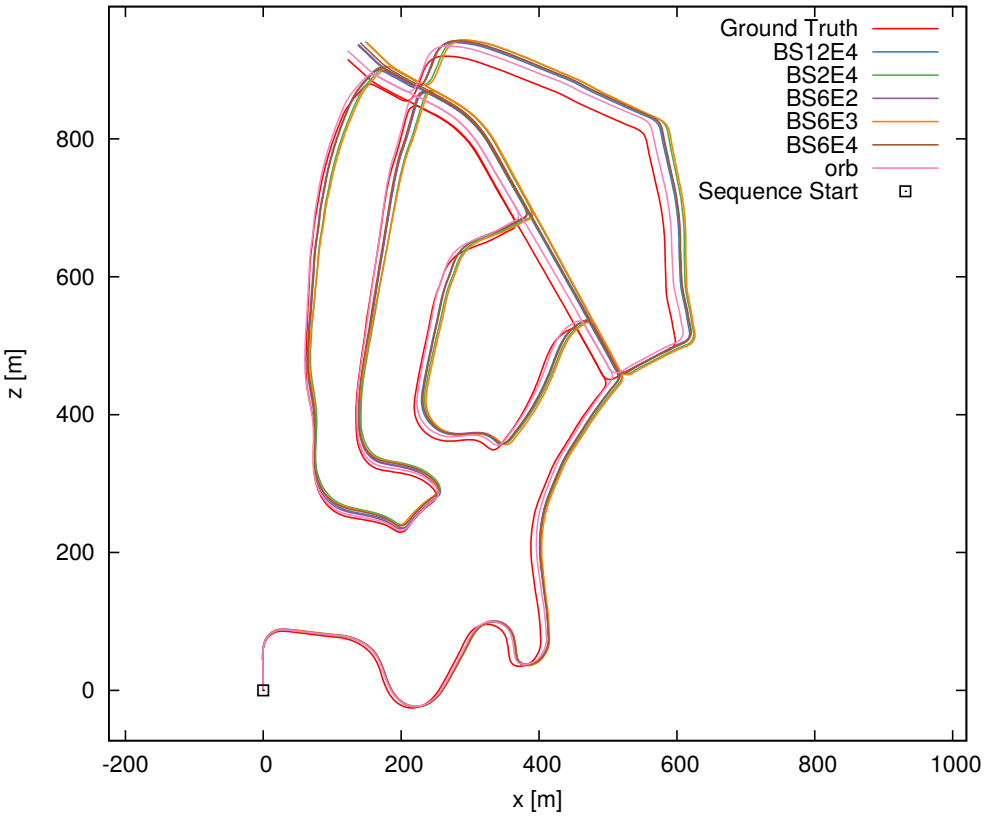


Figure C.1: Overlaid trajectory results for sequence 02 from the KITTI odometry set comparing various SIVO configurations, ORB_SLAM2, as well as the ground truth.

Table C.1 compares the keyframes and map points used by the various algorithms on KITTI sequence 02 as well as the translation and rotation errors. SIVO uses 30% fewer keyframes and 70% fewer map points than ORB_SLAM2 on average. For trajectory 02, we see that ORB_SLAM2 uses the most map points at 202,293, while the BS12E4 configuration of SIVO uses the least at 58,894.

Algorithm	**Keyframes**	**Map Points**	**Trans. Error** (%)	**Rot. Error** (deg/m)
ORB_SLAM2	1,836	202,293	**1.37**	$\mathbf{3.95 \times 10^{-5}}$
BS2E4	1,364	59,039	1.86	5.83×10^{-5}
BS6E2	**995**	63,316	1.92	5.05×10^{-5}
BS6E3	1,075	61,172	1.95	5.54×10^{-5}
BS6E4	1,442	60,704	1.80	5.61×10^{-5}
BS12E4	1,422	**58,894**	1.70	4.86×10^{-5}

Table C.1: Keyframe and map point comparisons between ORB_SLAM2 and various SIVO configurations on KITTI sequence 02. All algorithms used a similar number of keyframes, and SIVO used 70% fewer map points.

For sequence 02, ORB_SLAM2 has the best rotation and translation error at 3.95×10^{-5}deg/m and 1.37% respectively, while the best SIVO configuration, BS12E4, has a rotation and translation error of 4.86×10^{-5}deg/m and 1.70% respectively. The rotation error discrepancy is negligible, while the translation error difference corresponds to a 2.64m difference over an 800m subsequence. This performance is comparable considering the removal of over 140,000 map points.

Figure C.2 illustrates the translational errors for subsequences of length 100m to 800m on KITTI sequence 02. ORB_SLAM2 had the lowest translation error over all subsequences.

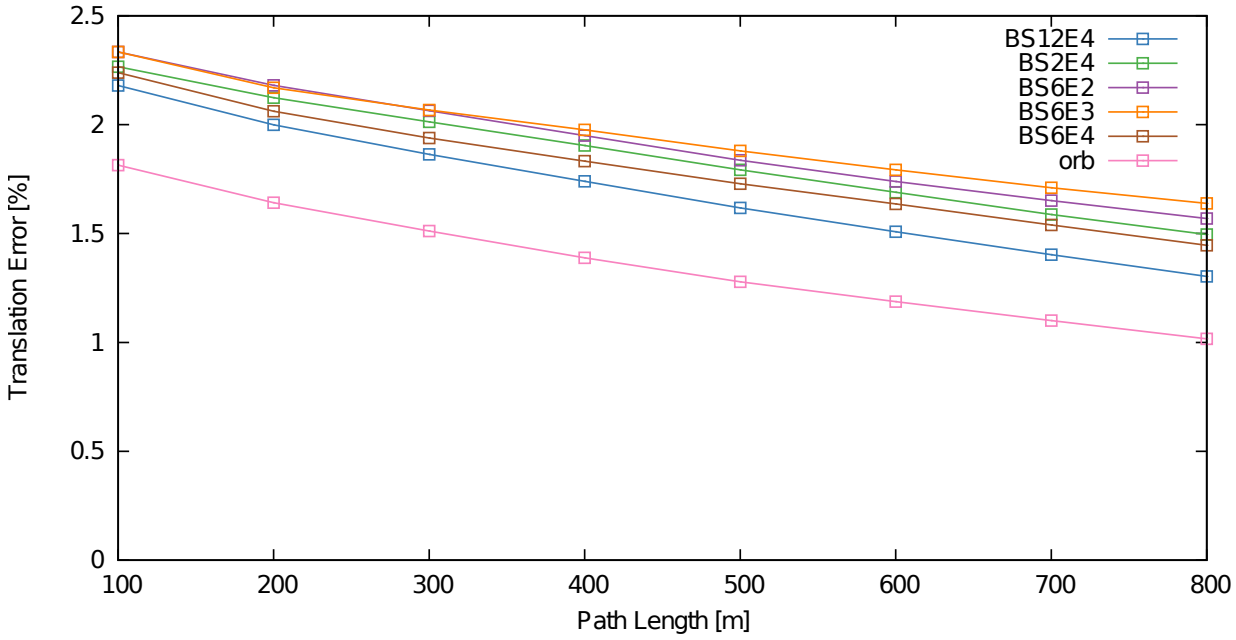


Figure C.2: Translational errors for subsequences of length 100m to 800m on KITTI sequence 02 comparing various SIVO configurations and ORB_SLAM2 to the ground truth. ORB_SLAM2 has the lowest translation error over all subsequences.

C.2 Trajectory 03

Figure C.3 shows an overlaid trajectory including the ground truth, the 5 different SIVO configurations, as well as the ORB_SLAM2 localization result for KITTI sequence 03. The results for this trajectory are similar to those obtained from sequence 01, where ORB_SLAM2 significantly outperforms SIVO. This is not immediately apparent from Figure C.3.

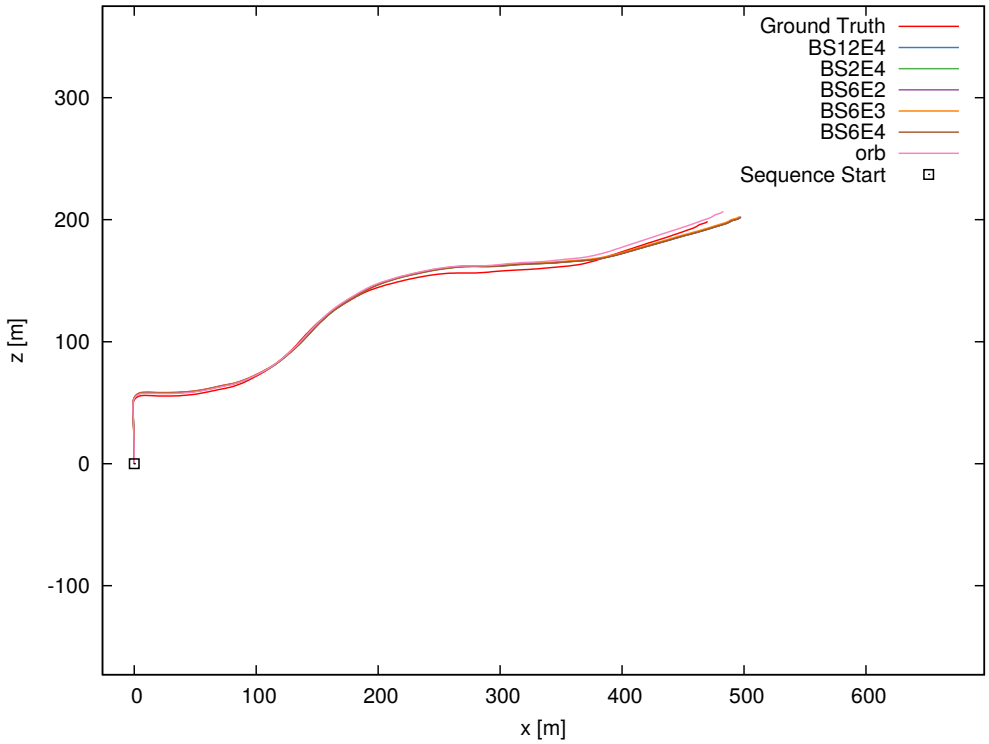


Figure C.3: Overlaid trajectory results for sequence 03 from the KITTI odometry set comparing various SIVO configurations, ORB_SLAM2, as well as the ground truth.

Table C.2 compares the keyframes and map points used by the various algorithms on KITTI sequence 03 as well as the translation and rotation errors. ORB_SLAM2 uses approximately twice the amount of keyframes and 70% more map points on average. For trajectory 03, we see that ORB_SLAM2 uses the most map points at 27,209, while our BS12E4 configuration of SIVO uses the least at 8,449.

Algorithm	**Keyframes**	**Map Points**	**Trans. Error** (%)	**Rot. Error** (deg/m)
ORB_SLAM2	228	27,209	**2.72**	$\mathbf{3.75 \times 10^{-5}}$
BS2E4	112	8,534	4.79	1.28×10^{-4}
BS6E2	**102**	8,941	4.91	1.45×10^{-4}
BS6E3	105	8,956	4.73	1.26×10^{-4}
BS6E4	110	8,467	4.72	1.29×10^{-4}
BS12E4	115	**8,449**	4.65	1.29×10^{-4}

Table C.2: Keyframe and map point comparisons between ORB_SLAM2 and various SIVO configurations on KITTI sequence 03. SIVO used approximately half of the keyframes and 68% fewer map points compared to ORB_SLAM2.

For sequence 03, ORB_SLAM2 has the best rotation and translation error at 3.75×10^{-5}deg/m and 2.72% respectively, while the best SIVO configuration, BS12E4, has a rotation and translation error of 1.29×10^{-5}deg/m and 4.65% respectively. The rotation error discrepancy is negligible, while the translation error difference corresponds to a 9.65m difference over an 500m subsequence.

Figure C.4 illustrates the translational errors for subsequences of length 100m to 500m on KITTI sequence 03. ORB_SLAM2 consistently has the lowest translation errors over all subsequences.

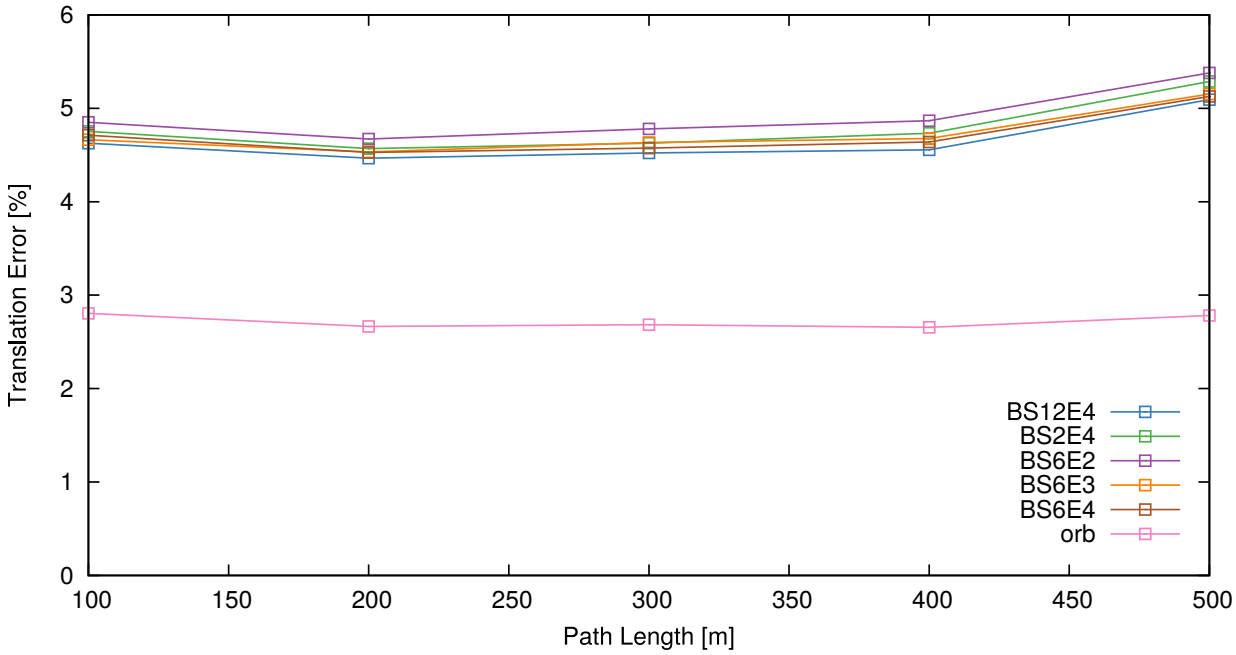


Figure C.4: Translational errors for subsequences of length 100m to 500m on KITTI sequence 03 comparing various SIVO configurations and ORB_SLAM2 to the ground truth. ORB_SLAM2 has the lowest translation error over all subsequences.

C.3 Trajectory 04

Figure C.5 shows an overlaid trajectory including the ground truth, the 5 different SIVO configurations, as well as the ORB_SLAM2 localization result for KITTI sequence 04.

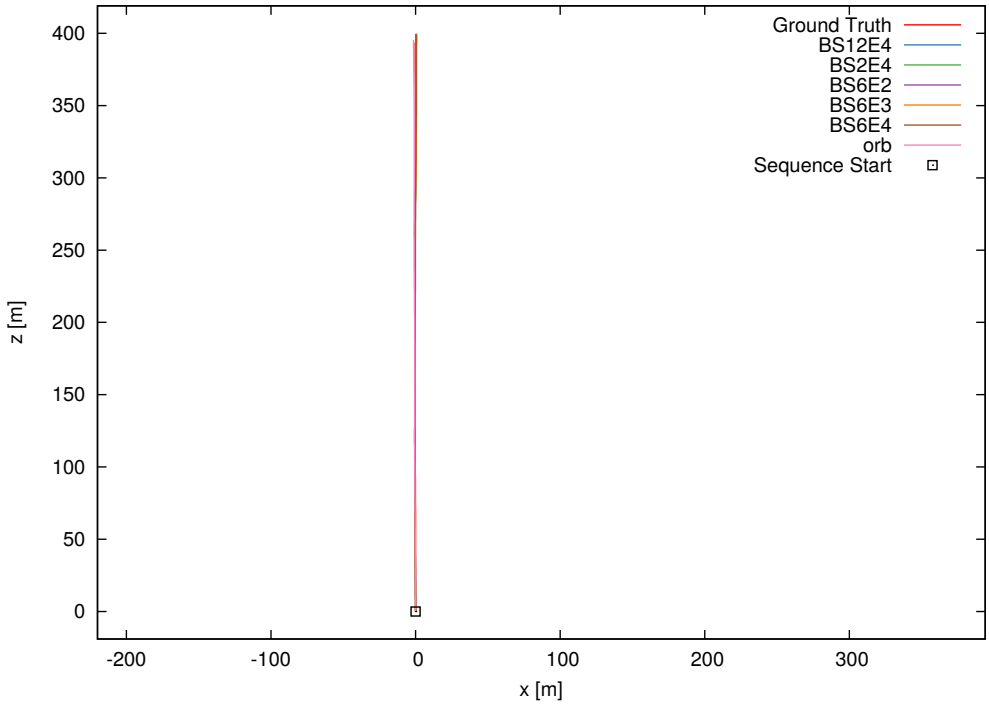


Figure C.5: Overlaid trajectory results for sequence 04 from the KITTI odometry set comparing various SIVO configurations, ORB_SLAM2, as well as the ground truth.

Table C.3 compares the keyframes and map points used by the various algorithms on KITTI sequence 04 as well as the translation and rotation errors. ORB_SLAM2 uses the same amount of keyframes and 70% more map points on average. For trajectory 04, we see that ORB_SLAM2 uses the most map points at 21,056, while our BS6E4 configuration of SIVO uses the least at 6,152.

Algorithm	**Keyframes**	**Map Points**	**Trans. Error** (%)	**Rot. Error** (deg/m)
ORB_SLAM2	173	21,056	**0.67**	2.20×10^{-5}
BS2E4	166	6602	1.52	$\mathbf{1.97 \times 10^{-5}}$
BS6E2	**81**	7128	1.62	2.73×10^{-5}
BS6E3	103	6880	1.71	2.93×10^{-5}
BS6E4	110	**6152**	1.55	3.73×10^{-5}
BS12E4	184	6328	1.50	4.73×10^{-5}

Table C.3: Keyframe and map point comparisons between ORB_SLAM2 and various SIVO configurations on KITTI sequence 04.

For sequence 04, ORB_SLAM2 has the best translation error at 2.72%, but SIVO configuration BS2E4 had the best rotation error at 1.97×10^{-5}deg/m. ORB_SLAM2 still had the best translation error at 0.67%, while the BS12E4 SIVO configuration had a translation error of 1.50%. The rotation error discrepancy is negligible, while the translation error difference corresponds to a 2.49m difference over a 300m subsequence.

Figure C.6 illustrates the translational errors for subsequences of length 100m to 300m on KITTI sequence 04. ORB_SLAM2 consistently has the lowest translation errors over all subsequences.

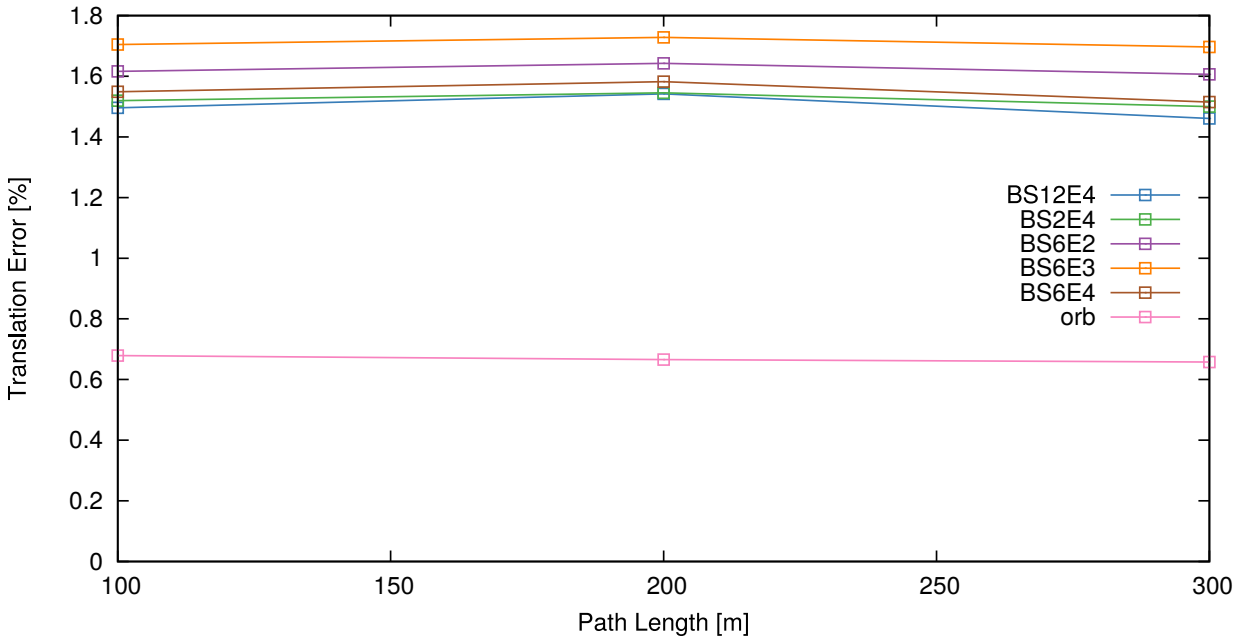


Figure C.6: Translational errors for subsequences of length 100m to 300m on KITTI sequence 04 comparing various SIVO configurations and ORB_SLAM2 to the ground truth. ORB_SLAM2 has the lowest translation error over all subsequences.

C.4 Trajectory 05

Figure C.7 shows an overlaid trajectory including the ground truth, the 5 different SIVO configurations, as well as the ORB_SLAM2 localization result for KITTI sequence 05. This sequence has several loop closures which help correct for drift.

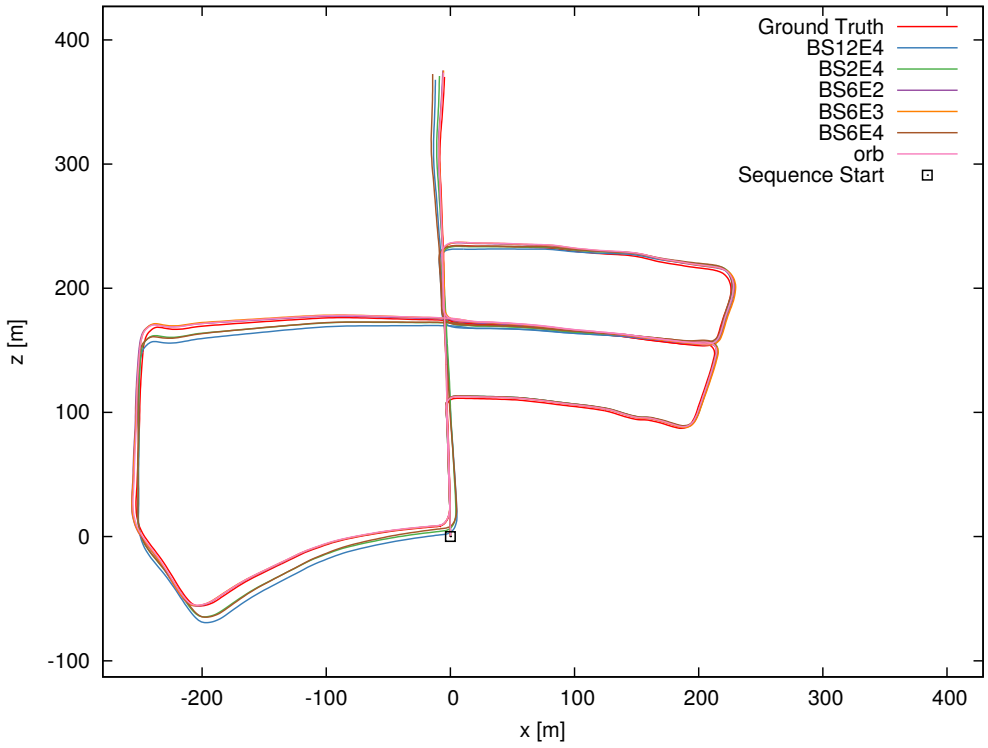


Figure C.7: Overlaid trajectory results for sequence 05 from the KITTI odometry set comparing various SIVO configurations, ORB_SLAM2, as well as the ground truth.

Table C.4 compares the keyframes and map points used by the various algorithms on KITTI sequence 05 as well as the translation and rotation errors. ORB_SLAM2 uses the same amount of keyframes and 70% more map points on average. For trajectory 04, we see that ORB_SLAM2 uses the most map points at 73,463, while our BS12E4 configuration of SIVO uses the least at 21,590.

Algorithm	**Keyframes**	**Map Points**	**Trans. Error** (%)	**Rot. Error** (deg/m)
ORB_SLAM2	753	73,463	**0.59**	$\mathbf{2.70 \times 10^{-5}}$
BS2E4	676	22,313	0.81	3.54×10^{-5}
BS6E2	**477**	22,662	0.76	3.24×10^{-5}
BS6E3	512	22,237	0.76	2.93×10^{-5}
BS6E4	718	21,929	0.81	4.00×10^{-5}
BS12E4	709	**21,590**	0.88	4.78×10^{-5}

Table C.4: Keyframe and map point comparisons between ORB_SLAM2 and various SIVO configurations on KITTI sequence 05.

For sequence 05, ORB_SLAM2 has the best rotation and translation error at 2.70×10^{-5}deg/m and 0.59% respectively, while the best SIVO configuration, BS6E3, has a rotation and translation error of 2.93×10^{-5}deg/m and 0.76% respectively. The rotation error discrepancy is again negligible, while the translation error difference corresponds to a 1.36m discrepancy over an 800m subsequence. This is an insignificant difference considering the 70% reduction in map size.

Figure C.8 illustrates the translational errors for subsequences of length 100m to 800m on KITTI sequence 05. ORB_SLAM2 has the lowest translation errors over all subsequences.

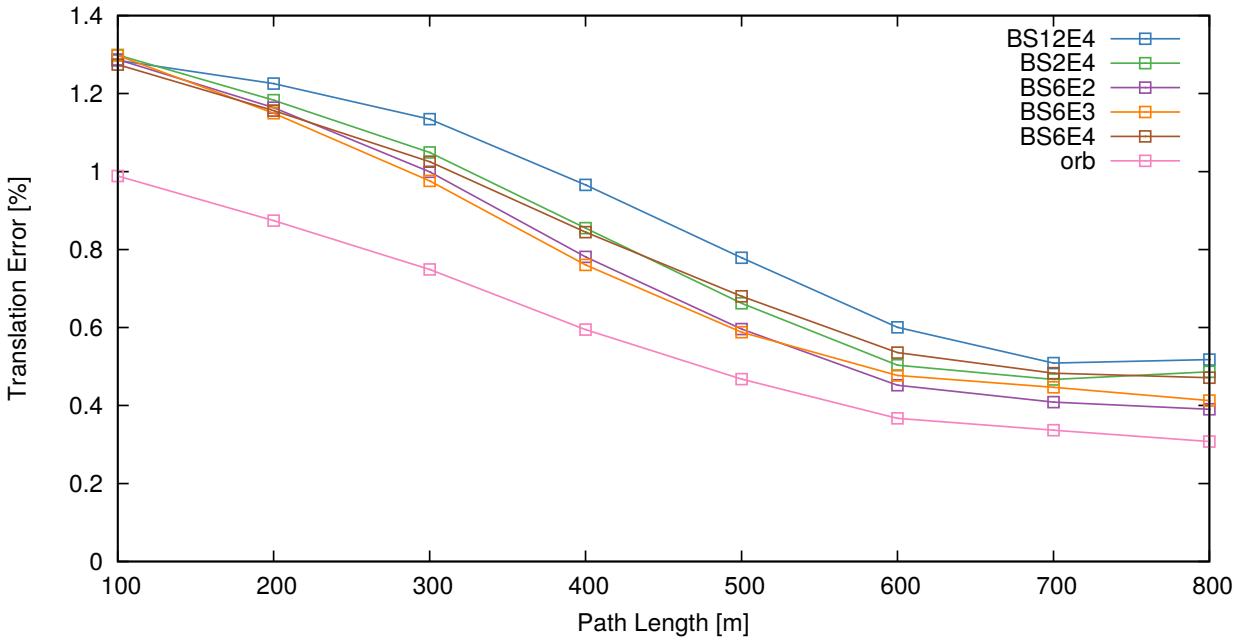


Figure C.8: Translational errors for subsequences of length 100m to 800m on KITTI sequence 05 comparing various SIVO configurations and ORB_SLAM2 to the ground truth. ORB_SLAM2 has the lowest translation error over all subsequences.

C.5 Trajectory 06

Unfortunately, we were not able to extract a full set of results for this trajectory, as SIVO lost tracking. The threshold of 4 bits used in other trajectories was too high for this sequence, and so we we only extracted results for BS6E2 and BS6E3. Similar to sequence 01, this trajectory had significant out-of-domain data for the network which resulted in poor segmentation quality. Figure C.9 shows an overlaid trajectory including the ground truth, the 2 SIVO configurations mentioned above, as well as the ORB_SLAM2 localization result for KITTI sequence 06.

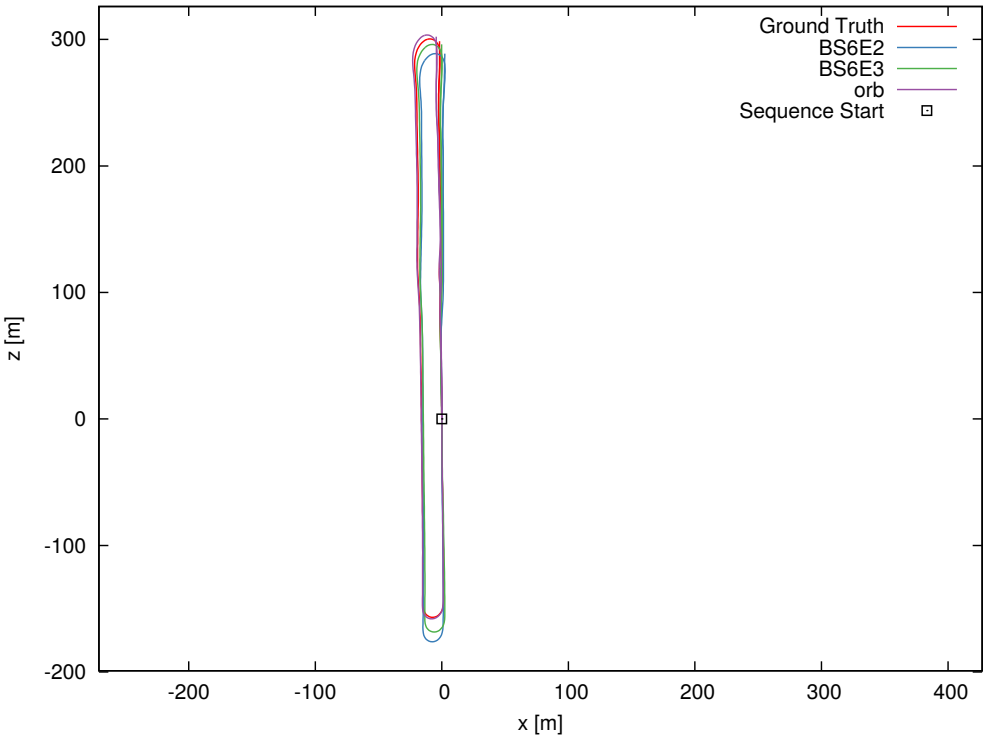


Figure C.9: Overlaid trajectory results for sequence 06 from the KITTI odometry set comparing various SIVO configurations, ORB_SLAM2, as well as the ground truth.

Table C.5 compares the keyframes and map points used by the various algorithms on KITTI sequence 06. ORB_SLAM2 uses significantly more map points, however the translation and rotation error are significantly better than SIVO.

Algorithm	**Keyframes**	**Map Points**	**Trans. Error** (%)	**Rot. Error** (deg/m)
ORB_SLAM2	490	47,461	**0.67**	$\mathbf{3.91 \times 10^{-5}}$
BS6E2	**284**	12,607	12.06	5.40×10^{-4}
BS6E3	338	**11,396**	7.10	3.27×10^{-4}

Table C.5: Keyframe and map point comparisons between ORB_SLAM2 and various SIVO configurations on KITTI sequence 06.

Figure C.10 illustrates the translational errors for subsequences of length 100m to 800m on KITTI sequence 06. ORB_SLAM2 consistently has the lowest translation errors over all subsequences.

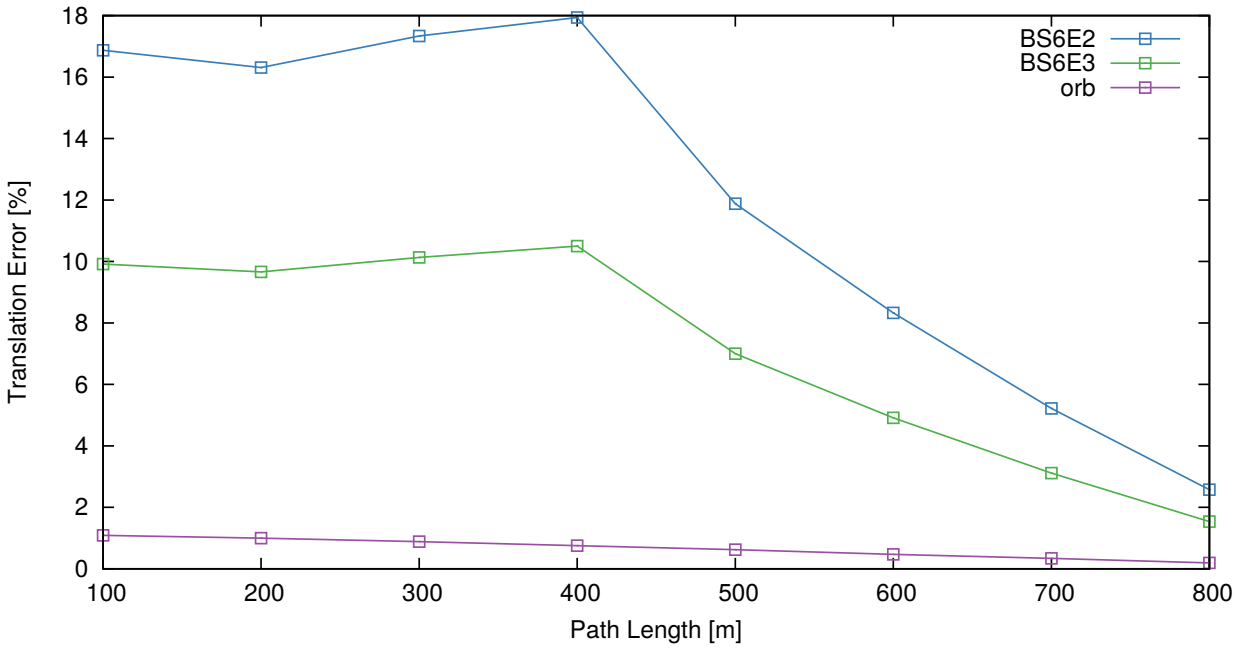


Figure C.10: Translational errors for subsequences of length 100m to 800m on KITTI sequence 06 comparing various SIVO configurations and ORB_SLAM2 to the ground truth. ORB_SLAM2 has the lowest translation error over all subsequences.

C.6 Trajectory 07

Figure C.11 shows an overlaid trajectory including the ground truth, the 5 different SIVO configurations, as well as the ORB_SLAM2 localization result for KITTI sequence 07. Although this sequence does form a loop, the termination of the sequence prevents the algorithm from actually performing the loop closure optimization. The results for this sequence contain all accumulated drift.

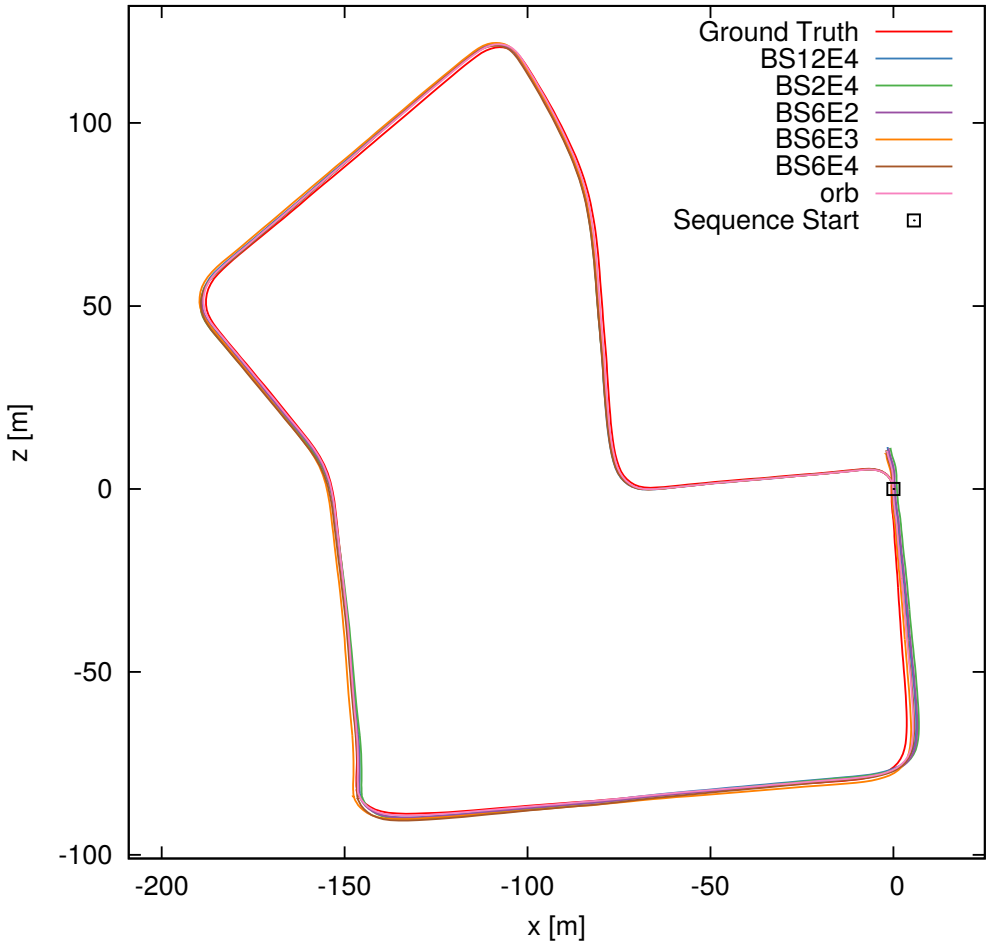


Figure C.11: Overlaid trajectory results for sequence 07 from the KITTI odometry set comparing various SIVO configurations, ORB_SLAM2, as well as the ground truth.

Table C.6 compares the keyframes and map points used by the various algorithms on

KITTI sequence 07 as well as the translation and rotation errors. SIVO uses a similar amount of keyframes to ORB_SLAM2 but 70% fewer map points on average. For trajectory 07, we see that ORB_SLAM2 uses the most map points at 29,632, while the BS2E4 configuration of SIVO uses the least at 9,029.

Algorithm	**Keyframes**	**Map Points**	**Trans. Error** (%)	**Rot. Error** (deg/m)
ORB_SLAM2	266	29,632	**0.58**	$\mathbf{4.43 \times 10^{-5}}$
BS2E4	197	**9,029**	0.93	6.15×10^{-5}
BS6E2	193	9,139	0.86	6.65×10^{-5}
BS6E3	180	9,684	0.80	5.08×10^{-5}
BS6E4	**171**	10,151	0.81	5.93×10^{-5}
BS12E4	187	9,088	1.06	8.82×10^{-5}

Table C.6: Keyframe and map point comparisons between ORB_SLAM2 and various SIVO configurations on KITTI sequence 07.

For sequence 07, ORB_SLAM2 has the best rotation and translation error at 4.43×10^{-5}deg/m and 0.58% respectively, while the best SIVO configuration, BS6E3, has a rotation and translation error of 5.08×10^{-5}deg/m and 0.80% respectively. The rotation error discrepancy is negligible, while the translation error difference corresponds to a 1.32m discrepancy over a 600m subsequence. This is an insignificant difference considering the removal of 70% of the map points.

Figure C.12 illustrates the translational errors for subsequences of length 100m to 600m on KITTI sequence 07. ORB_SLAM2 has the lowest translation errors over all subsequences, however we see for the 500m and 600m subsequences, BS6E3 effectively has the same translation error.

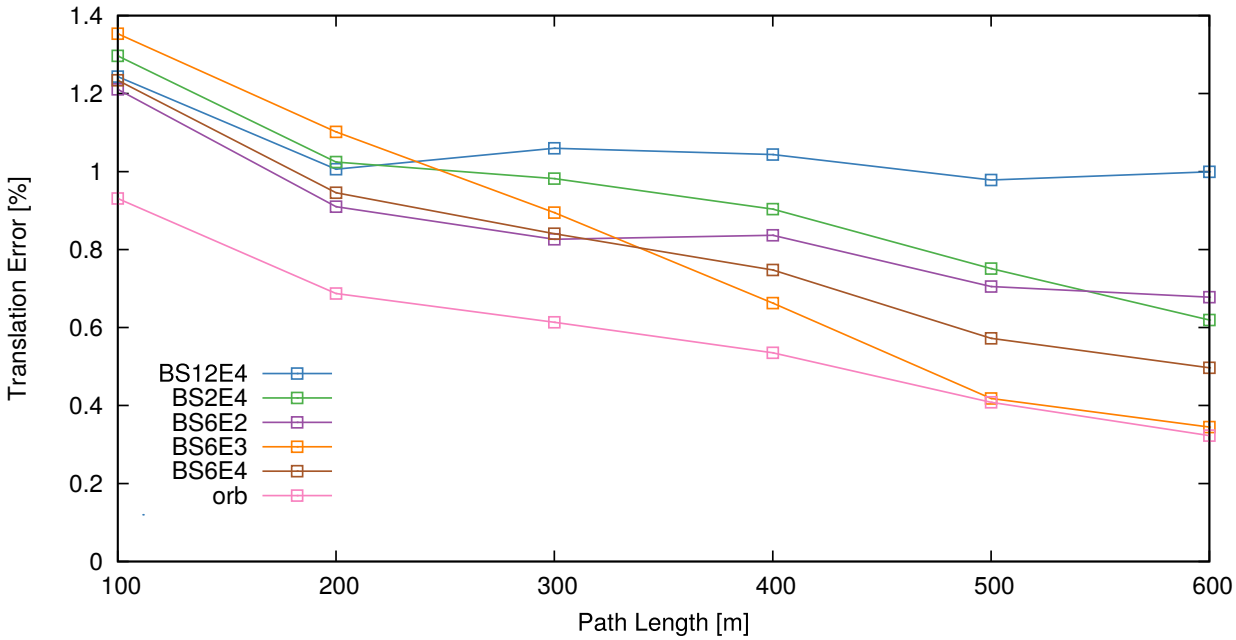


Figure C.12: Translational errors for subsequences of length 100m to 600m on KITTI sequence 07 comparing various SIVO configurations and ORB_SLAM2 to the ground truth. ORB_SLAM2 has the lowest translation error over all subsequences.

C.7 Trajectory 10

Figure C.13 shows an overlaid trajectory including the ground truth, the 5 different SIVO configurations, as well as the ORB_SLAM2 localization result for KITTI sequence 10.

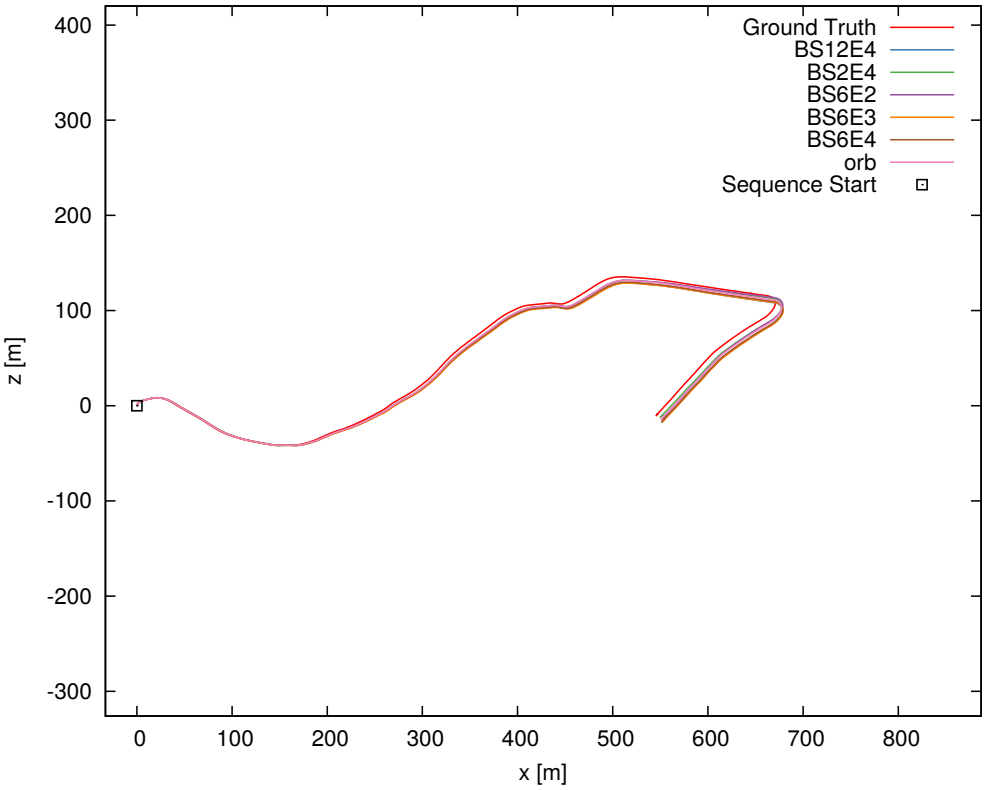


Figure C.13: Overlaid trajectory results for sequence 10 from the KITTI odometry set comparing various SIVO configurations, ORB_SLAM2, as well as the ground truth.

Table C.7 compares the keyframes and map points used by the various algorithms on KITTI sequence 10 as well as the translation and rotation errors. SIVO uses 40% fewer keyframes and 70% fewer map points on average. For trajectory 10, we see that ORB_SLAM2 uses the most map points at 33,181, while the BS12E4 configuration of SIVO uses the least at 9,136.

Algorithm	**Keyframes**	**Map Points**	**Trans. Error** (%)	**Rot. Error** (deg/m)
ORB_SLAM2	335	33,181	**0.95**	$\mathbf{4.85 \times 10^{-5}}$
BS2E4	**198**	9,369	0.97	7.34×10^{-5}
BS6E2	202	9,335	1.14	8.63×10^{-5}
BS6E3	208	9,769	1.26	9.00×10^{-5}
BS6E4	206	9,248	1.26	1.02×10^{-4}
BS12E4	204	**9,136**	1.14	7.06×10^{-5}

Table C.7: Keyframe and map point comparisons between ORB_SLAM2 and various SIVO configurations on KITTI sequence 10.

For sequence 10, ORB_SLAM2 has the best rotation and translation error at 4.85×10^{-5}deg/m and 0.95% respectively, while SIVO configuration BS2E4 has a rotation and translation error of 7.34×10^{-5} deg/m and 0.97% respectively. The rotation error discrepancy is negligible, while the translation error difference corresponds to a 16cm discrepancy over an 800m subsequence. This is an insignificant difference considering the removal of 70% of the map points.

Figure C.14 illustrates the translational errors for subsequences of length 100m to 800m on KITTI sequence 10. ORB_SLAM2 has the lowest translation errors over subsequences of length 100m to 700m, with BS2E4 having a lower error for the 800m subsequences.

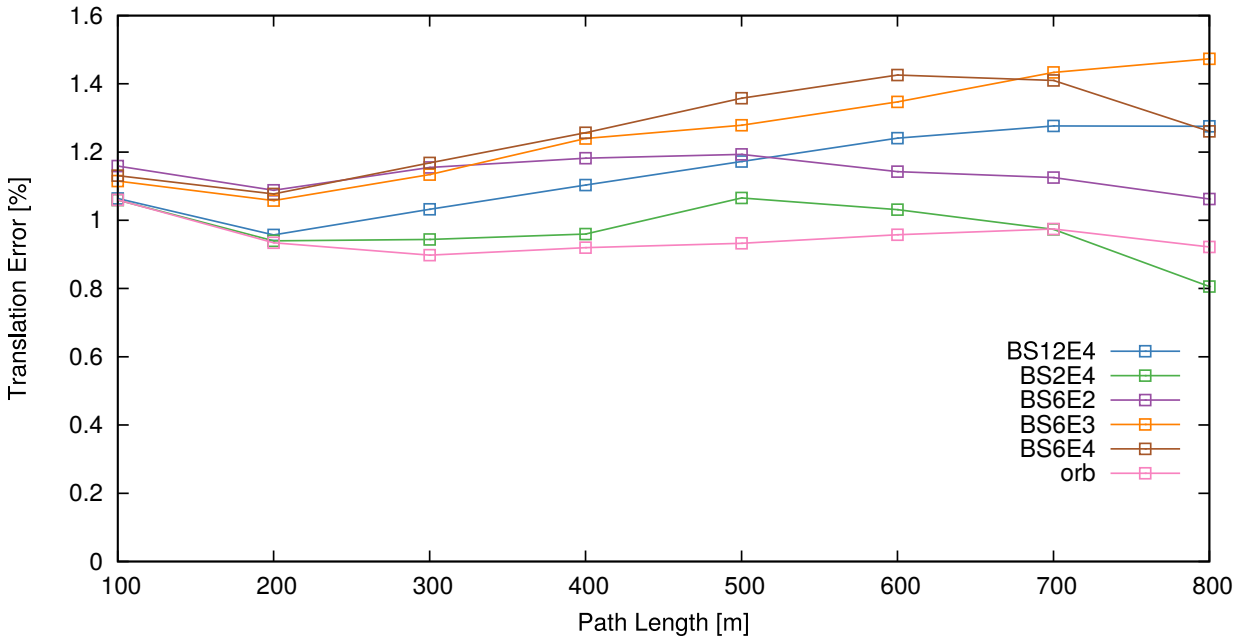


Figure C.14: Translational errors for subsequences of length 100m to 800m on KITTI sequence 10 comparing various SIVO configurations and ORB_SLAM2 to the ground truth. Overall, ORB_SLAM2 outperforms SIVO, but for 800m subsequences SIVO BS2E4 outperforms the state of the art.