Visual Localization for UAVs Using Satellite Images

by

Mollie Bianchi

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science

Graduate Department of the Institute for Aerospace Studies
University of Toronto

© Copyright 2021 by Mollie Bianchi

Visual Localization for UAVs Using Satellite Images

Mollie Bianchi
Master of Applied Science
Graduate Department of the Institute for Aerospace Studies
University of Toronto
2021

Abstract

The primary objective of this thesis is to develop a method for the offline generation of a map containing only Google Earth (GE) rendered images against which an Unmanned Aerial Vehicle (UAV) is able to localize using vision online. This eliminates the need for a manual mapping flight and would allow the UAV to fly in new unseen areas autonomously. The first part of this thesis presents a novel method for localizing live UAV images against a collection of rendered and encoded GE images that is fast and has low storage requirements. The second part of this thesis implements this new measurement into the existing Visual Teach & Repeat (VT&R) system. This includes writing a script to generate a compatible map entirely offline, computing the measurement in the VT&R pipeline, and using the measurement in the localization optimization. We demonstrate the performance of using GE image-based measurements on a dataset of real UAV collected images at six different times of day.

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Unmanned Aerial Vehicles (UAVs) are being used more and more in a wide variety of outdoor applications, such as surveillance and reconnaissance, agriculture monitoring, package delivery, and emergency response, yet they remain primarily dependent on GPS for localization. The disadvantage of a GPS-based localization system is that it is susceptible to dropout, interference, and intentional jamming. Due to safety concerns regarding GPS failure, UAVs are limited to line-of-sight flights so that the operator is able to take over in case of failure.

Vision is often used as an additional sensor, or as the primary sensor in GPS-denied environments, due to its low weight and fast computation; however, current vision-based methods are limited as well. Visual Odometry (VO) is commonly employed, but still requires corrections to prevent drift. Visual Simultaneous Localization and Mapping (SLAM) is a common solution to the issue of drift, but its use on UAVs has been primarily demonstrated in indoor environments or small areas [8], [31], [37]. With a focus on providing capabilities for longer outdoor flights, there is still much to be improved upon with UAV vision systems.

One system providing outdoor autonomous flight capabilities is Vi-

sual Teach and Repeat (VT&R) [12]. This method generates a locally consistent visual map on an outbound pass under manual or GPS control, and the UAV is then able to return autonomously along the path without GPS using a stereo camera. The requirement for a mapping pass is the largest downside to this approach. GPS or manual operation is required to generate the map limiting where the UAV is able to fly. As well, because VT&R relies primarily on point-feature matching (e.g., Speeded-Up Robust Features (SURF) [6]), the return pass must be completed shortly after the outbound pass so that the lighting conditions along the path have not changed significantly. It does not allow for the repeated traversal of the path using a map generated much earlier.

A unique opportunity for outdoor aerial vehicles is the existence of easily available databases of satellite images covering the entire globe. Google Earth (GE) is one of the most accessible. In many areas, GE has used the satellite images to generate a detailed 3D reconstruction of the area. This reconstruction takes into account the 3D structures of buildings, trees, and other objects, making it possible to render a realistic image at any desired pose. In [27], the idea to replace the manual outbound mapping pass in VT&R with a virtual pass in GE was proposed.

The biggest challenge with this idea is finding a way to accurately and robustly localize real live images captured from a UAV with the artificial, reconstructed images rendered from GE in real time. Since the satellite images used for the reconstruction were captured years ago, there are differences with the live images in terms of lighting, small object movement (e.g. vehicles, trailers), large structural changes (e.g. building additions/demolitions), and unusual object reconstruction, particularly for non-rectangular objects like trees. This makes it

difficult for feature-based methods to obtain accurate and robust results in many cases.

Figure 1.1: Between the real images and the GE images there are differences in colour and lighting, the movement, addition, or removal of smaller objects like cars and trailers, and renovations to some of the buildings.

Proof of concept for the ability to match live images and GE images was demonstrated by Patel et al. [27] who developed a mutual-information(MI)-based matching scheme. In their work, GE images were rendered all around the desired flight path. MI was used to select the best initial GE match to the live image and a finer alignment optimization was also conducted. This approach was computationally expensive and would require storing thousands of full-sized images on board the UAV. It was never fully integrated with the existing VT&R

system and is not capable of running in real time.

In this thesis, we develop a new method to localize live UAV images to pre-rendered images from GE that is fast and storage efficient. As in [27], images are rendered around the desired flight path in GE. With this novel method, an autoencoder is then trained on these path-specific images to compress them to a much smaller vector representation. The same autoencoder is used to compress the live image as well. The compressed live image vector is compared to all nearby compressed GE image vectors through an inner-product kernel. This results in weights associated with each of the corresponding GE image poses. From these weights, a localization for the longitude, latitude, and heading with accompanying covariance is computed.

This method is described in detail in Chapters 2 and 3. Image registration using this method has been demonstrated on a real UAV dataset of images collected along a 1.1km path at six different times of day covering several lighting conditions. In comparison with [27], we are able to achieve the same accuracy performance on image registration and run in 1% of the computation time with approximately half the storage requirements.

The second half of this thesis integrates this new measurement into the existing VT&R system. This includes generating a VT&R compatible teach pass using only monocular images and pose data from GE automatically and entirely offline. The computation of the new GE image-based measurement mentioned above is added into the C++ pipeline and used as an additional factor in the localization optimization. This work is described in Chapter 4 and is demonstrated by running the modified VT&R system using the offline generated map on the real, multiple time of day dataset.

1.2 Related Work

1.2.1 Aerial Image Registration

Aerial image registration typically takes a live downward-facing image from a UAV and estimates the relative pose change to align it with a georeferenced satellite image. Early works in this area looked at using edge detection followed by image warping to find an optimal alignment [11], or a combination of classical techniques with learned semantic segmentation [24]. Both of these appraoches require the presence of buildings or roads in the images and perform better at high-altitude flights where more structure is present in the images. They suffer in areas comprising mainly grass and trees.

Feature-based localization is the current standard for visual state estimation. Traditional point features (such as SIFT [18], SURF[6], and BRIEF[9]) have led to significant achievements in localization and mapping systems. For example, features have been used to match street view images to images captured from a ground robot [3] and a UAV [19, 20]. This is achieved by first performing place recognition using a visual bag-of-words technique and is followed by image registration using Scale Invariant Feature Transform (SIFT) keypoints. However, a disadvantage of low-level point-based features is that they often look drastically different under different lighting conditions, seasonal change, or between image modalities.

As an alternative to point-based features, some works have introduced bespoke detectors that identify larger image structures. Instead of applying the same generic descriptor to images from all locations, Linegar et al. [17] and McManus et al. [22] curate a large bank of spatially indexed detectors for distinctive visual elements. They achieve this by collecting multiple images with a variety of appearances at a lo-

cation, partitioning the images into tiles, and training SVM detectors on each of the tiles. An iterative classification process removes non-distinctive tiles that are consistently mislabelled and uses a geometric consistency check to select distinctive patches in each of the images.

Learning-based methods for visual pose estimation have also been increasing in popularity. When provided with enough training data, a network is capable of learning to predict the global pose given a single image within the training area as demonstrated with PoseNet [16]. Networks are also capable of computing the relative pose change between two images [23], [14]. The benefit of learning-based approaches is that given enough training data, they should be able to learn to be robust to changes in lighting and season; however, these approaches are limited by the available training data. Real data is expensive to collect and label, and synthetic data does not typically generalize directly to the real world. An alternative is the use of unsupervised methods [41], [40], [29], which have become more popular with self-driving and the large datasets available.

Registering images of different modalities, such as GE images and real images, poses an additional problem. It is difficult to match features across modalities and training a network requires large quantities of both types of images or a sim-to-real domain transfer. Mutual Information (MI) is a measure of the mutual dependence between two variables. In the context of images, it is maximized when the images are correctly aligned and was initially used in the medical field to register images of different modalities [35]. It has since been used in robotic localization systems such as [32][25] to register live camera images to images generated from 3D meshes. Most relevant to this thesis is the work done by Patel et al. [27] where the optimal warping parameters to maximize the MI between a georeferenced satellite image and a live

UAV image were found. This method was able to achieve less than 3m and 3° root mean square error (RMSE) on real datasets collected at various times of day; however, it is largely limited by the costly computation and storage requirements. This method will be used for comparison with the results in Chapter 3 and a more detailed background can be found in Section 3.2.1.

1.2.2 Visual Teach & Repeat

The Visual Teach and Repeat (VT&R) method [12] enables the long distance autonomous, outdoor, aerial navigation in GPS-denied environments. VT&R is a system by which a robot is capable of long-range autonomy using a stereo camera as its sole sensor. For the teach pass, the robot is manually driven over a desired path while constructing a local topologically connected map. The robot is then able to autonomously repeat this path by localizing to the pre-built map. By repeating the path over gradual changes in experience (e.g., lighting conditions, seasonal variations, etc.) the map can be extended to allow for more robust localization performance in varying conditions [28].

The VT&R system has been successfully adapted for use onboard a multirotor UAV to allow for the completion of autonomous emergency returns in the case of GPS failure [36]. The UAV flies out under GPS or manual control and is able to return along the same path if GPS fails using the onboard camera. This approach is limited as a GPS or manual flight is still required for the outbound pass, and in the case of emergency, the UAV must return along the same path it flew out on even if a shorter return route exists. The ability to localize to existing map images, which is the main focus of this thesis, would allow the UAV to autonomously fly routes that have never been flown before.

A more detailed explanation of the VT&R system and how it has

been modified for use on a UAV can be found in Section 4.2.

1.3 Contributions

The main contributions of this thesis are:

- An autoencoder trained purely on GE images for a specified path that is capable of encoding never before seen real images along the same path into a low-dimensional vector. This allows for the efficient storage and comparison between real images and GE images. The development and performance of this network is detailed in Chapter 2.

- A method using this trained autoencoder to generate a codebook of georeferenced satellite images and match a live image to this codebook through inner kernel computations to obtain a localization measurement with covariance. The methodology to compute this measurement and its performance on real data is detailed in Chapter 3. Along with contribution #1, this work resulted in the following journal paper:

  Mollie Bianchi and Timothy D. Barfoot. UAV Localization Using Autoencoded Satellite Images. *IEEE Robotics and Automation Letters (RAL)*, 6(2):1761–1768, 2021. doi: 10.1109/LRA.2021.3060397.

- A script to automatically generate offline a VT&R compatible map provided only with a set of path coordinates, GE rendered images at the specified coordinates, and the encoded GE reference images. The development and explanation of steps performed by this script are detailed in Chapter 4.

- Modifications to the VT&R pipeline adding the computation of the GE image-based measurement to the localization pipeline and

adding this measurement as an additional factor in the localization optimization. The development and the evaluation of the modified VT&R system on a dataset of real UAV images at six different times of day is detailed in Chapter 4.

Chapter 2

Autoencoding Satellite Images

2.1 Introduction

One of the unique opportunities of working with aerial vehicles is that large databases of high-resolution satellite images exist for much of the world and are easily accessible. One of these databases is Google Earth (GE). In addition to planar satellite images, GE has used the satellite images to create 3D reconstructions of many areas. Images can be rendered at any location, altitude, and orientation in one of these 3D reconstructed areas and the result is an image manufactured from multiple satellite images that takes into account the geometry of the scene. This is ideal for lower-altitude flights where planer assumptions break down since the height of buildings is significant compared to the UAV altitude and causes a parallax effect as the UAV flies.

Due to computation and connectivity constraints onboard the UAV, it is not feasible to render images from the GE reconstruction as the UAV flies. Therefore a map of these images must be created offline and loaded onto the UAV. One of the limiting factors of [27] was the large storage requirements and lengthy computation time required to process thousands of full-size images. A method was needed that could compress the images to a much smaller representation but still retain

enough information that the compressed representations could be used to match between the live and GE images. Autoencoders are neural networks commonly used for the compression of images and were selected for this task.

Exploiting the idea of place-specific excellence, we focused on training a specific autoencoder offline for each flight path. In order to do so, images are rendered offline in GE all around the desired flight path and are then used as training data for the autoencoder. After training, the autoencoder is able to generalize and successfully compress never-before-seen live images. The next step after generating a codebook of compressed GE reference images is to develop a means of matching the encoded live image to these reference images, which is the focus of Chapter 3.

This chapter is organized as follows. First, we provide some background on autoencoders and how they are typically used. Then, we discuss the procedure for rendering the images from GE, the network structure selected, and the training regime. We conclude with the performance and some areas for future work.

2.2 Background and Related Work

An autoencoder typically refers to a neural network containing encoder and decoder sections. The input is fed into the encoder part of the network which compresses it down to a lower-dimensional vector. This encoded vector is provided as input to the decoder network. The decoder then upsamples from this vector to result in a final output that is the same size as the original input. Typically images are used as the input. By training to minimize some loss function between the original image and the reconstructed image, the network can learn to retain

the key features in the bottleneck vector. If desired, an additional loss function can be implemented on the compressed vectors to encourage them into following a selected distribution.

Many works use an encoder to compress the original images and then perform computations with the vector representations. For example, Hou et al. [15] use a deep feature consistent loss function during training to learn to encode facial images. They then linearly combine the vectors and decode to generate images with new or mixed facial features. There has been a lot of work on autoencoders, including trying new loss functions [30], adversarial autoencoders [21], and combining autoencoders with neural autoregressive models [10].

Similar to our work, Sundermeyer et al. [33] use an autoencoder to generate a codebook of images rendered in simulation at various 6DoF poses around an object. They then encode a real live image of the object and match it against the codebook to get an estimate of the object's pose. They use a generalized version of a denoising autoencoder [34]. A denoising autoencoder adds random artificial noise to the input image, but the reconstruction target is kept noise free. They show that this training regime enforces invariance against noise and also different input regimes, which aids in the transfer from sim to real data.

2.3 Rendering Google Earth Images

Given a desired path, images are rendered from GE [1] at the intended orientation every 0.5m along the path. Additional images are rendered at 0.5m lateral offsets out to 5m on both sides of the path. This requires 42 images for each meter of the path. This coverage could be modified based on expected performance of the UAV. For example, if the UAV is expected to operate in windy conditions more images could be rendered

further from the path. Regardless, this leads to a high number of images for non-trivial path lengths. Storing and comparing these images in full size would be infeasible. The next step and key aspect of this method is to use an autoencoder to compress the images to a low-dimensional representation while maintaining the key features of each image such that comparisons with an encoded live image yield sensible results.

2.4 Network Structure

The autoencoder architecture is based on [15] as implemented in [2]. The input is a 320×160 greyscale GE image. The encoder is composed of six layers. Each of the first five layers performs a 2D convolution with a kernel size of four and a stride of two followed by a batch normalization layer. The number of channels doubles as indicated in Figure 2.1. Finally, a linear layer maps the output of the final convolution layer to the bottleneck vector. Different sizes were experimented with for the dimension of the bottleneck, but a bottleneck of dimension 1000 was decided upon as it was the smallest size that could still achieve the desired accuracy.

The decoder part of the network behaves opposite the encoder. A linear layer first maps the bottleneck variable to 1024 channels. This is then passed to the first of five layers, each of which performs upsampling by a factor of two, followed by convolution with a kernel size of three and stride of one, and batch normalization. The number of channels is halved in each layer until an output greyscale image with the same dimensions as the input image is generated. To obtain the compressed image vector, the output after only the encoder part of the network is used.

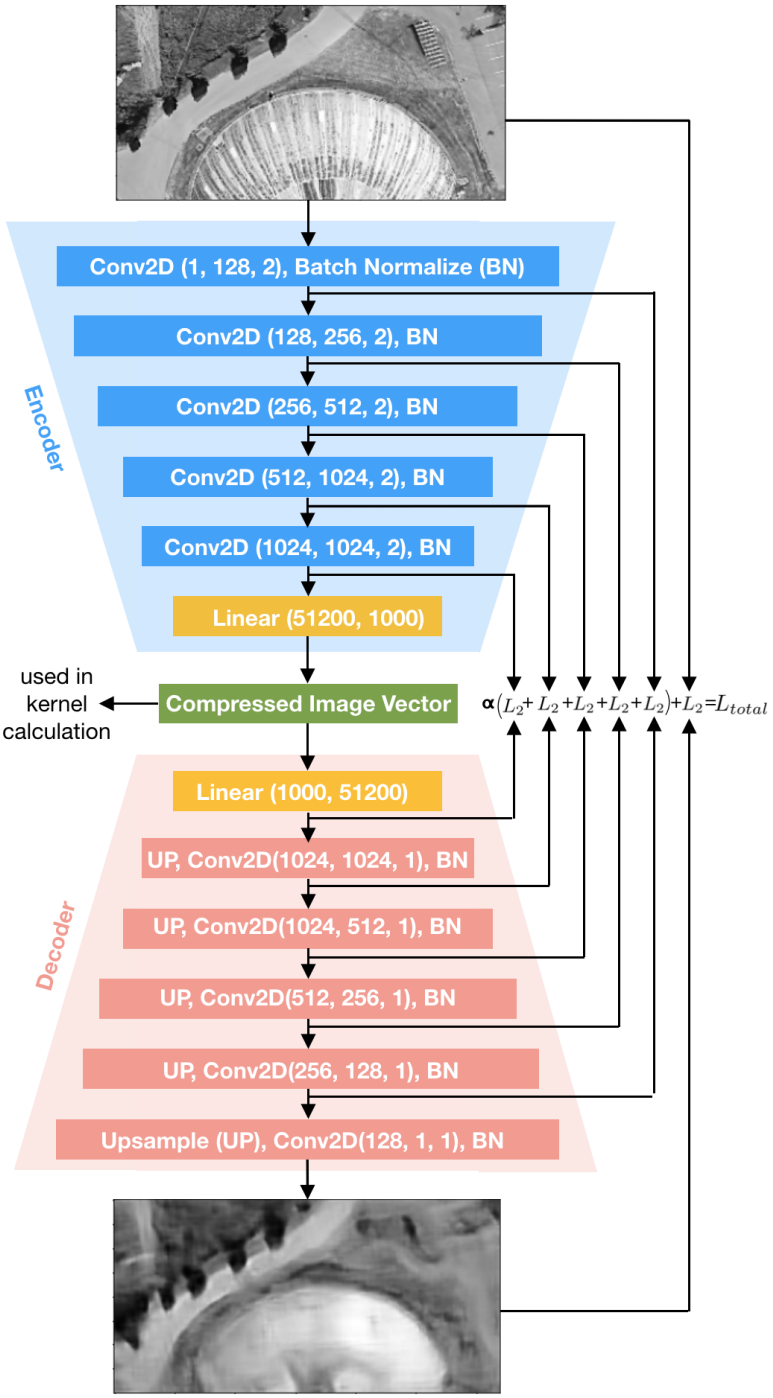


Figure 2.1: The encoder network has five convolutional layers each followed by a batch normalization layer. The number of input channels, output channels, and stride is indicated in parentheses. The final layer of the encoder is a linear layer producing a 1000 dimensional vector. The first layer of the decoder network is a linear layer mapping the bottleneck vector to 1024 channels. There are five more layers, each of which performs upsampling by a factor of two, followed by convolution and batch normalization. An L2 loss function is used between the reconstructed image and the original image as well as between outputs of corresponding layers in the encoder and decoder.

Figure 2.2: The training and validation errors, orange and blue respectively, decrease when training with or without skip losses. However, the errors on samples from two real datasets, red and green, decrease more when training with skip losses.

2.5 Training

The autoencoder is trained solely using the precollected images from GE for a specific desired path. A new autoencoder would need to be trained for each path. Since only the GE images are used for training, this step can be completed entirely offline before ever flying the path.

The loss function used to train the network is a combination of photometric L2 loss between the input and output images, i.e., $L = (I_{\text{input}} - I_{\text{output}})^2$, and L2 loss between the outputs of corresponding layers (referred to as skip losses). These skip losses help each layer of the decoder to learn to perform the opposite of the corresponding encoder layer. These additional skip losses are weighted with a value of 0.01 and were found to improve performance on the real image validation sets.

For the path used in the experiments, the network was trained with approximately 48,000 images. The network was trained for 20 epochs with a learning rate of 1e-4. On an Nvidia DGX Station using a single Tesla V100 GPU training took around 20 hours to complete.

2.6 Performance

The purpose of the autoencoder is to generate lower-dimensional representations of the images that still retain enough of the key structural information to be used for matching. Using the trained autoencoder, 320×160 images are able to be compressed to a 1000 dimensional vector. This is a significant reduction from the 51,200 pixels in the original image and an even larger reduction from the 176,400 pixels used in the larger sized images in [27].

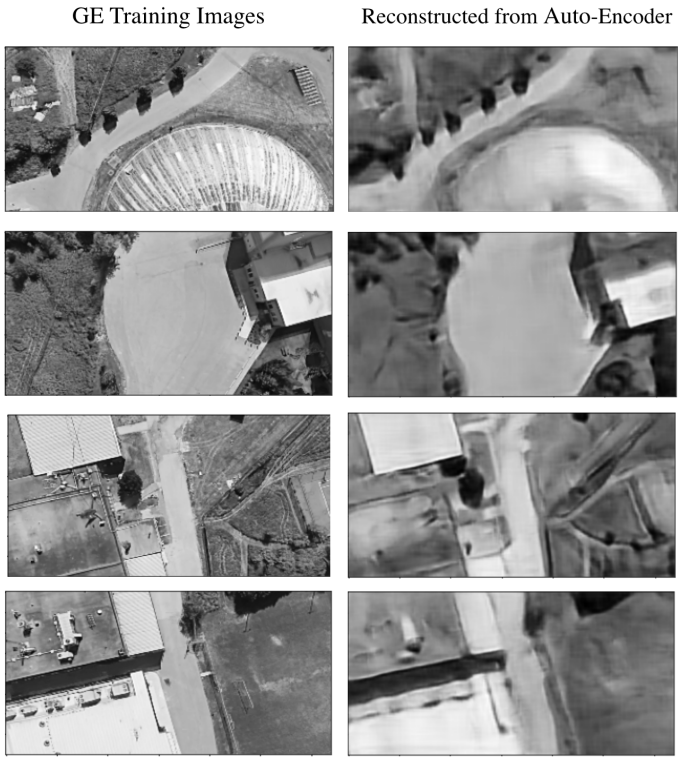


Figure 2.3: GE images used for training the autoencoder are shown on the left and the corresponding reconstructions are on the right.

To get a sense of how much key structural information is retained in the encoded vectors, we visualize the reconstructed images. Figure 2.3 shows some example GE training images from various parts along the path in the left column and their corresponding reconstructions in the right column. The reconstructions retain a lot of the core information

from the original images. From just a 1000-dimensional vector, the position of the roads, buildings, and trees can be recovered.

Figure 2.4 shows real images collected from the UAV in the left column and their corresponding reconstructions on the right. These real images were not used in any part of the training. In comparison with the training image reconstructions, these reconstructions are not as sharp or crisp but the main structural elements are present. The buildings, paved parking lots, roads and grassy areas can be seen in the reconstructions.

Figure 2.4: The original real UAV images are shown on the left and the corresponding reconstructions are on the right. The real UAV images were not used during training, but the network is still able to generate a reasonable encoding from which the core scene elements can be extracted.

In Chapter 3, the encoded vectors are used in kernel computations to compute a measure of similarity between the GE images and a live image. In order for this similarity measure to be accurate, the network

must able to accurately encode both GE and real images. A more detailed, quantitative evaluation of the comparisons between the encoded vectors can be seen in Chapter 3.

2.7 Summary

In this chapter, we detailed the training procedure for an autoencoder to learn to compress GE and real images along the path. The key aspect of this procedure is that the encoder is trained entirely on images rendered from GE, yet it can still generalize fairly well to live images. This offline-only aspect is towards the goal of generating a map entirely offline, which would allow a UAV to fly autonomously in new areas without having to complete an online mapping pass.

By reducing the size of images required, from 320×160 or 51,200 pixels, to just 1000 dimensions, we have significantly reduced the storage requirements required with a map of thousands of images. Since our encoded image sizes are so small, we can actually render more images at a finer spacing than [27] and still use less storage. This reduction also has a positive impact on the computation time. Using fewer dimensions to represent the same image will speed up any calculations. The next chapter details how we use the encoded vectors to get a localization measurement and present quantitative results for the storage and runtime requirements.

Chapter 3

Localization Using Kernels

3.1 Introduction

Using the autoencoded reference images as described in the previous chapter, we are able to generate a codebook to which we will match the live image. This proposed approach can be divided into several steps as depicted in Figure 1. The first step is offline preprocessing. Images are rendered from GE around the desired flight path and an autoencoder is trained for this specific path as described in Chapter 2. The encoded reference images and the trained encoder weights are then saved. While the offline processing is significant, it only needs to be completed once per path and would eliminate the need for manual mapping flights before each autonomous flight as is currently done in [36].

In the online component of the pipeline, the live image is passed through the trained encoder and compared to a subset of nearby encoded reference images using an inner product kernel computation. This results in a set of weights for each reference image, which are used alongside the known reference image coordinates to compute a weighted average localization measurement for the latitude, longitude, and yaw of the live image. The weights are also used to compute a

covariance estimate which is used for outlier rejection.

We evaluate this new method on a real world dataset collected at six different times of day. The performance is comparable to [27] achieving an RMSE of less than 3m, but our new method runs in 1% of the computation time and with approximately half the storage requirements.

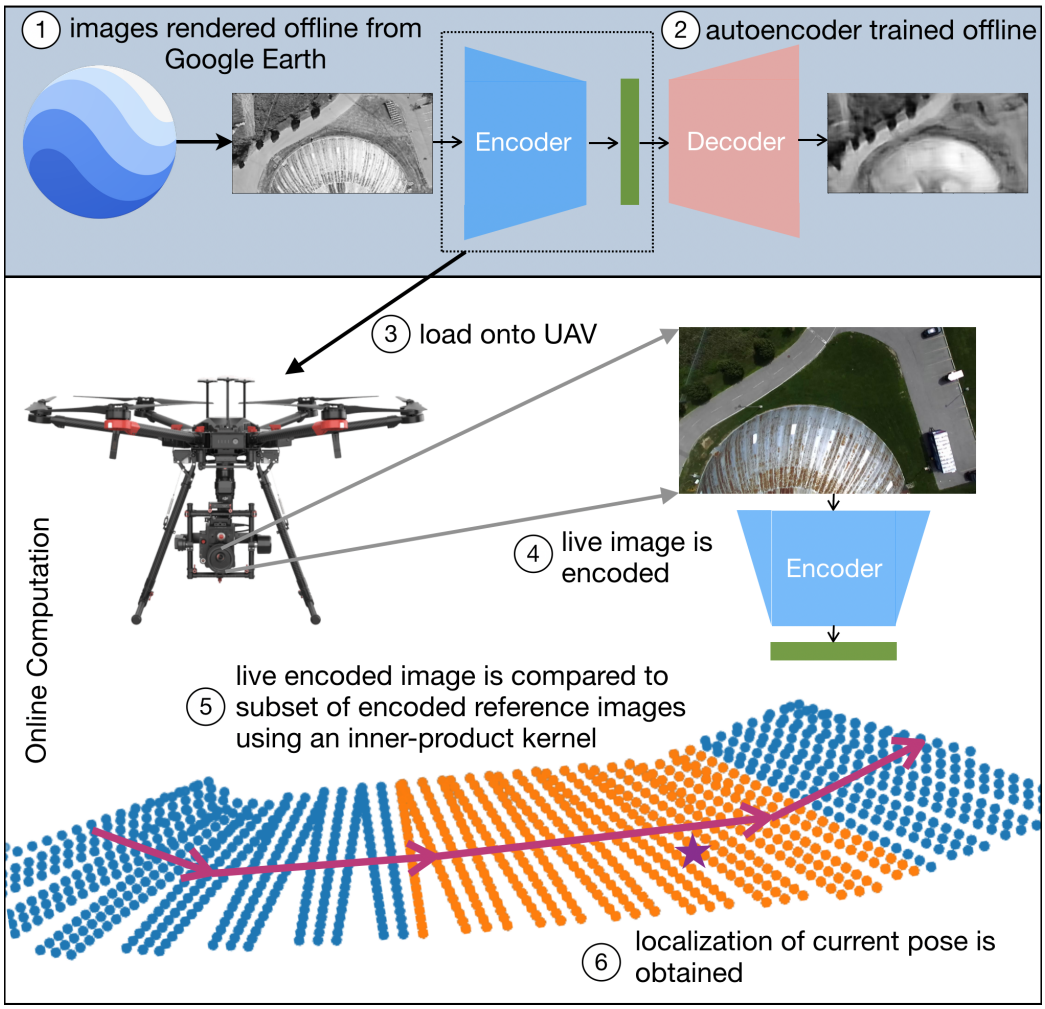


Figure 3.1: 1. Offline before flight, images are rendered in a grid pattern around the desired path using Google Earth [1]. An example of this grid pattern is shown at the bottom of the figure. 2. These images are used to train an autoencoder using photometric loss and skip losses. 3. All the encoded training images and the encoder are transferred onto the UAV. 4. The live image captured by the UAV is passed through the trained encoder. 5. The encoded live image is compared with a subset of the encoded reference images using an inner-product kernel outputting a weight for each reference image. 6. These weights are used to compute the localization and covariance.

This chapter is organized as follows. First, we provide a background on related work in three relevant areas. Then we go into the details of how the localization measurement is obtained from the codebook. Finally, we provide some experimental results and comparisons with [27] before concluding with a summary.

3.2 Related Work

3.2.1 Mutual Information Based Matching

The most relevant method for localization using satellite images, and the one that will be used for comparison, is the Mutual Information(MI) based approach presented by Patel et al. [27]. This was largely inspired by [25, 26] in which MI had been used to localize monocular camera images within a textured 3D model of the environment. Adapting this idea to a UAV, Patel et al. [27] were able to achieve less than 3m and 3° RMSE on low-altitude flights at six different times of day. In their work, images were rendered from GE beforehand every 3m along the path and around the path at intervals of 6m. The Normalized Information Distance (NID), which is a MI based metric, was computed between the live image and all images within 4m of the prior pose estimate (e.g., from filtering) to select the best-matching image. The alignment between this geo-referenced image and the live image was then computed by a series of coarse and refined optimizations of the warping parameters. Each step of the optimization required numerically computing the Jacobian of the NID with respect to the warping parameters. This process was quite slow making this method incapable of running in real time. As well, the images were stored in their full 560×315 resolution resulting in large storage costs for longer paths. We build on the idea of prerendering images around a desired path in

GE, but improve upon [27] by eliminating the costly optimization step, improving runtime, and decreasing storage requirements.

3.2.2 Kernels

Kernels, such as the inner-product or exponential kernel, are often used for matching patches between images [13], [39]. They provide a measure of the similarity between the two patches and are fast to compute, but they are not commonly used for comparisons of whole images due to the high number of pixels involved. In this new method, we use kernels on the autoencoded representations of whole images. Since these autoencoded images are small enough to quickly compute a kernel between them, it eliminates the need for extracting and matching patches from an image.

3.2.3 Learning a Codebook

Sundermeyer et al. [33] use a similar method to what is proposed in this work to perform 6D Object Detection. Instead of explicitly learning from 3D pose annotations during training, they implicitly learn representations from rendered 3D model views. Using a denoising autoencoder, they generate a codebook containing the encoded representations of tens of thousands of rendered images from simulation of the desired object at uniformly distributed poses. The same autoencoder is used to encode a live image and a cosine similarity metric is used to match the live image with the closest poses from the codebook.

3.3 Computing the Localization Measurement

3.3.1 Latitude and Longitude

To minimize time spent loading the autoencoded GE reference images, all image vectors are stacked and loaded into a $1000 \times N$ dimensional matrix denoted $\mathbf{Y}_{\text{ge}}$, which is loaded onto the GPU. Then, based on a prediction of the current live image pose, $\mathbf{Y}_{\text{ge}}$ is indexed to include only the reference images that are 4m ahead and behind along the path, which is the same search area used in [27]. The live image is passed through the trained autoencoder and the resulting compressed 1000×1 vector is denoted as $\mathbf{y}$. The weights, $\mathbf{w}$, are computed for the subset of autoencoded GE reference images using a basic inner-product kernel:

$$\mathbf{w} = \mathbf{Y}_{\text{ge}}^{T}\mathbf{y}. \tag{3.1}$$

The weights represent a similarity measurement between the live image and each of the reference images. Since there are 336 images in our nearby area and thus being used for comparison, many of these images have low, but non-zero, weights. These weights tend to pull the mean value towards the centre of the area covered by the reference images. To prevent this and get a result closer to the images with the highest weights, a new set of weights, $\mathbf{w}_{\text{th}}$, is created by setting all values of the weights less than one standard deviation of the max weight to zero. The thresholded weights are then normalized:

$$\bar{\mathbf{w}}_{\text{th}} = \frac{\mathbf{w}_{\text{th}}}{\sum_i w_{\text{th},i}}. \tag{3.2}$$

The longitude and latitude coordinates of each reference image are stacked in a $2 \times N$ matrix $\mathbf{X}_{\text{ge}}$. The thresholded weights are used

to compute the localization for the longitude and latitude according to

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} = \mathbf{X}_{\text{ge}} \bar{\mathbf{w}}_{\text{th}}. \tag{3.3}$$

3.3.2 Yaw

Instead of rendering reference images at multiple headings and including them in the previous computation, we use a similar tactic to Wolcott and Eustice [38]. The heading computation is performed after the above step by rotating the live image. First, the reference image with the largest weight, $\mathbf{y}^{\star}_{\text{ge}}$, is selected for comparison. The uncompressed live image is then rotated in 1° increments between -5° and 5°. All these rotated images are encoded and stacked into an 1000×11 matrix, $\mathbf{Y}_{\theta}$. The kernel computation from (3.1) is repeated:

$$\mathbf{w}_{\theta} = \mathbf{Y}_{\theta}^{T} \mathbf{y}^{\star}_{\text{ge}}. \tag{3.4}$$

These weights are normalized, $\bar{\mathbf{w}}_{\theta}$, and then used to compute a heading measurement following the same procedure as in (3.3) using a stacked vector of the rotation values, $\mathbf{x}_{\theta}$:

$$\hat{\theta} = \mathbf{x}_{\theta}^{T} \bar{\mathbf{w}}_{\theta}. \tag{3.5}$$

This mean heading value, $\hat{\theta}$, by which the live image has been rotated is then subtracted from the heading of the selected reference image, β, to get a global heading, $\beta - \hat{\theta}$. While not included here, a similar process could be used to get an estimate for altitude without having to render

additional reference images. The full localization is then

$$\hat{\mathbf{p}} = \begin{bmatrix} \hat{x} \\ \hat{y} \\ \beta - \hat{\theta} \end{bmatrix}. \tag{3.6}$$

3.4 Computing Covariance

In the above computations, we generate a set of weights. These weights can indicate a sense of how confident we are in the localization. If the weights are very sharply peaked, as they are in the example on the left side of Figure 3.2, we are more confident in our estimate. If instead the weights are more evenly spread out, as on the right side of Figure 3.2, or there are two peaks, we are less confident in our measurement. We can use the following equation to generate a covariance estimate based on the original weight values:

$$\mathbf{P} = \sum_i w_i (\mathbf{x}_{\text{ge},i} - \hat{\mathbf{x}})(\mathbf{x}_{\text{ge},i} - \hat{\mathbf{x}})^T. \tag{3.7}$$

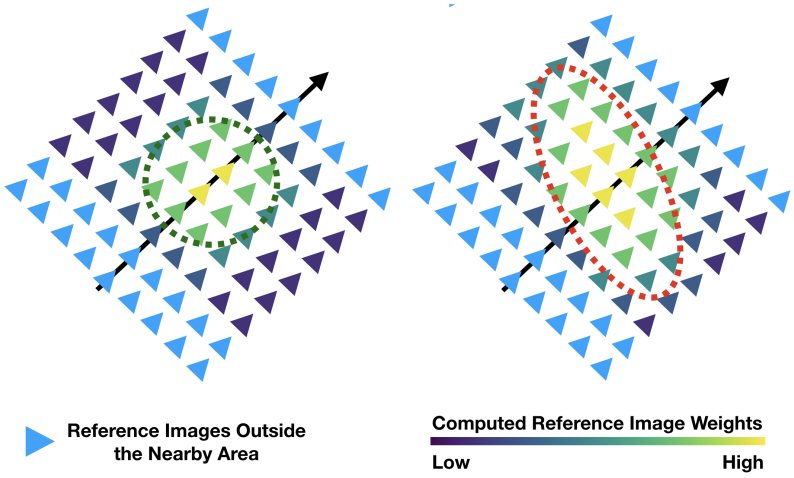


Figure 3.2: On the left, we show an example of a highly peaked weight distribution, which results in a smaller covariance. On the right, we show an example of a more spread out distribution, which yields a larger covariance.

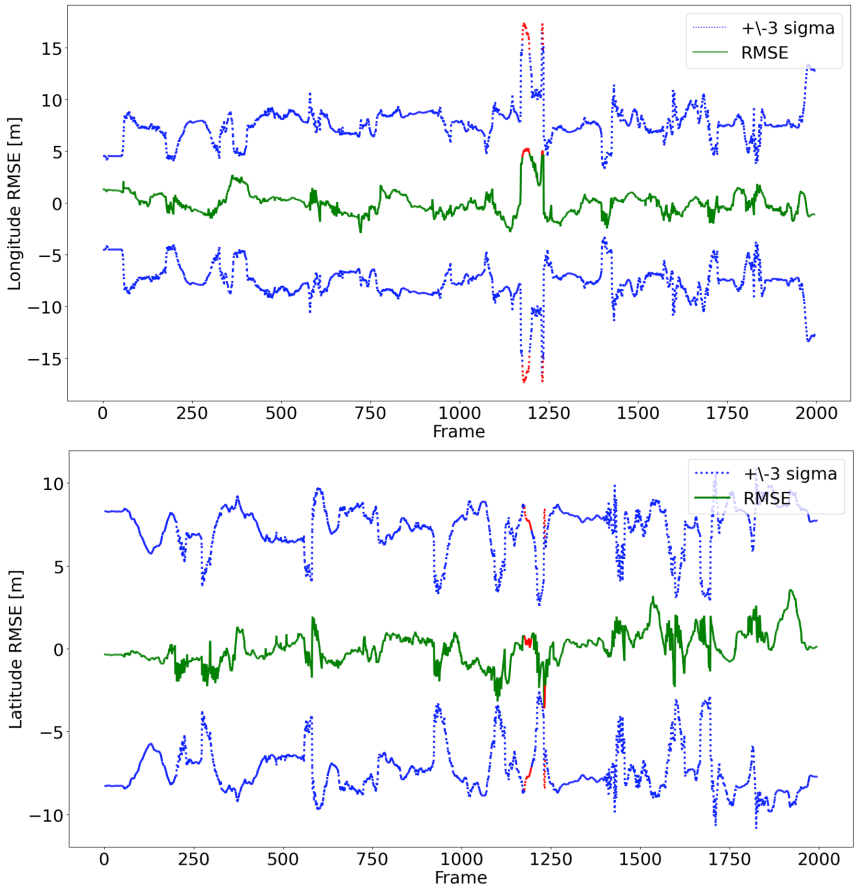


Figure 3.3: Plots of the RMSE for the longitude and latitude coordinates of the sunrise test are shown here with accompanying 3σ uncertainty envelope. The registrations that were rejected due to either σ_{long} or $\sigma_{\text{lat}} > 5$ are shown in red.

3.4.1 Outlier Rejection

Since we compute a covariance with our localization based on the weights, we can also use this to reject outliers. When the weights have a single narrow peak away from the edges of the area covered by the reference images, the σ^2_{long} and σ^2_{lat} values are small. When the weights are more spread out with a less-well-defined peak, when there are multiple peaks, or when the peak occurs very close to the edge of the reference area, this results in larger values for σ^2_{long} and/or σ^2_{lat}. We reject localizations that have either σ_{long} or σ_{lat} greater than 5. Figure 3.3 plots the RMSE for the longitude and latitude coordinates for one

of the flights in the experiments section with accompanying covariance. Rejected registrations are indicated in red.

3.5 Experimental Results

3.5.1 Experimental Setup and Dataset

Image registration to obtain the longitude, latitude, and heading was performed on the same dataset as in [27]. We did not focus on estimating the roll, pitch, or altitude of the vehicle as those can be measured by complementary sensors to vision. In this dataset, the data was collected at UTIAS using a DJI Matrice 600 Pro multirotor UAV with a 3-axis DJI Ronin-MX gimbal (see Figure 3.4). A StereoLabs ZED camera provides stereo images at 10FPS. The RTK-GPS system and IMU provide the vehicle pose for ground truth.

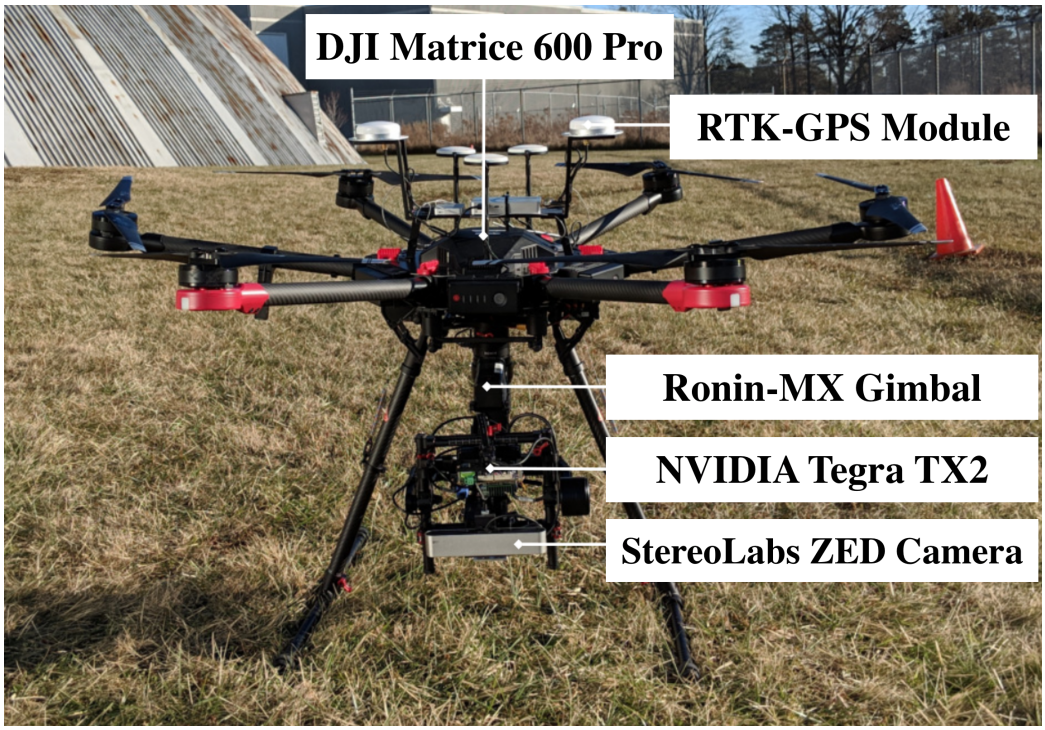
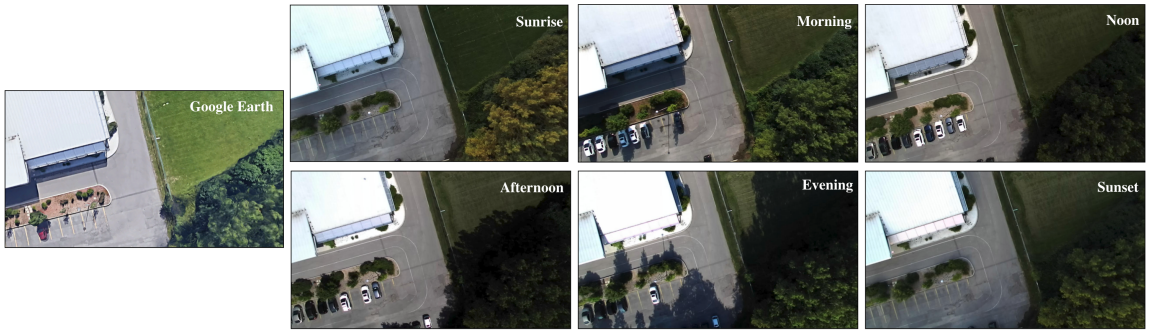


Figure 3.4: The dataset was collected by a 3-axis gimballed stereo camera on a multirotor UAV also equipped with an RTK-GPS module to collect the ground truth pose.

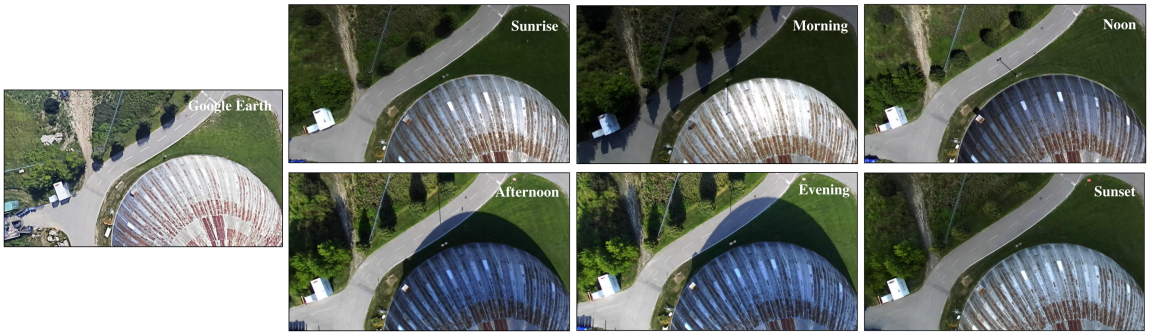
The dataset consists of six traversals of a 1132m path over built-up areas with roads and buildings as well as large areas of grass and trees as seen in Figure 3.5. Each run captures the distinctive lighting condition at different times of day: sunrise, morning, noon, afternoon, evening, and sunset. These lighting conditions are shown at three locations along the path in Figure 3.6 alongside the accompanying GE rendered image. The UAV flies at an altitude of 40m with a constant heading and the camera pointed in the nadir direction.

Figure 3.5: The 1.1km path covers a variety of terrain around UTIAS including large areas of vegetation and more built-up areas with roads and buildings. The path starts in the center of the image and proceeds in a clockwise direction.

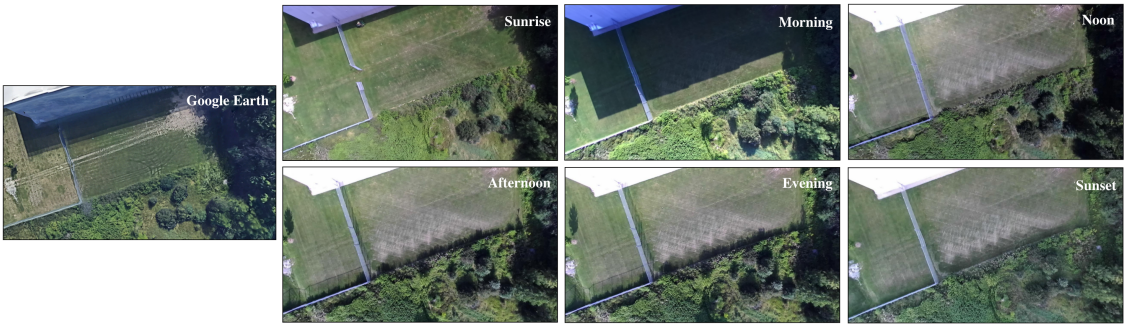
There is an unknown offset between the RTK-GPS frame and the GE frame so 10% of the successful image registrations are used to align the frames. These registrations are then omitted from all error calculations [27].



(a) Substantial differences in colour can be seen between the GE image and the real images. The presence of shadows also varies throughout the times of day with the real images.



(b) Large shadows are present on the east side of the dome in the afternoon and evening images, but not in the GE image.



(c) Shadows on the west side of the building are present in the GE image and the morning run of the data.

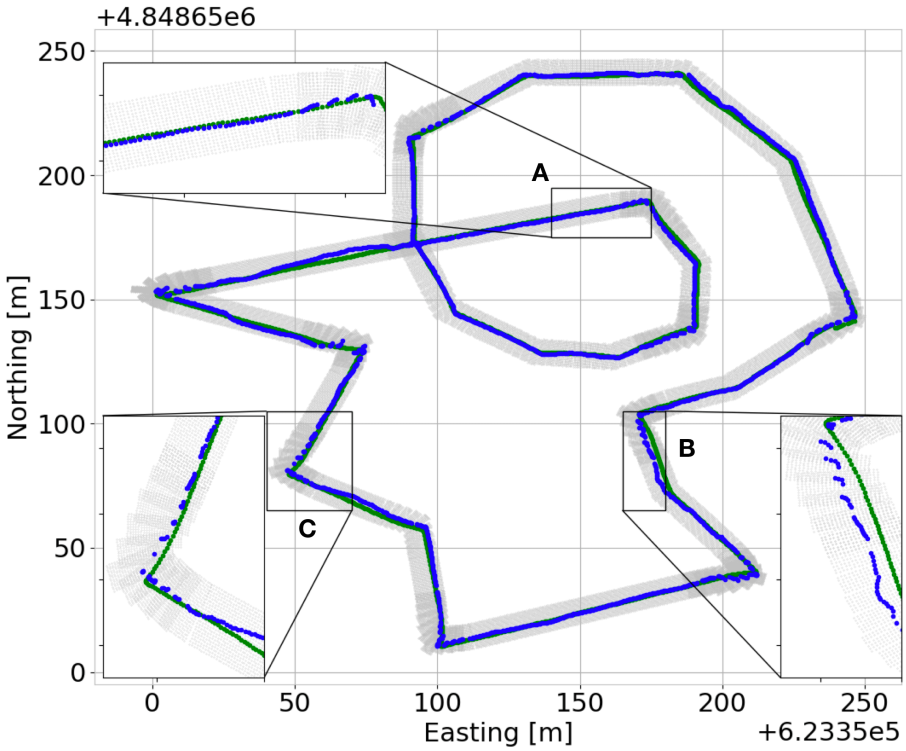
Figure 3.6: The dataset consists of six runs at different times of day: sunrise, morning, noon, afternoon, evening, and sunset. Images from each of these runs and the corresponding GE image are shown at three locations along the path.

3.5.2 Performance Under Various Lighting Conditions

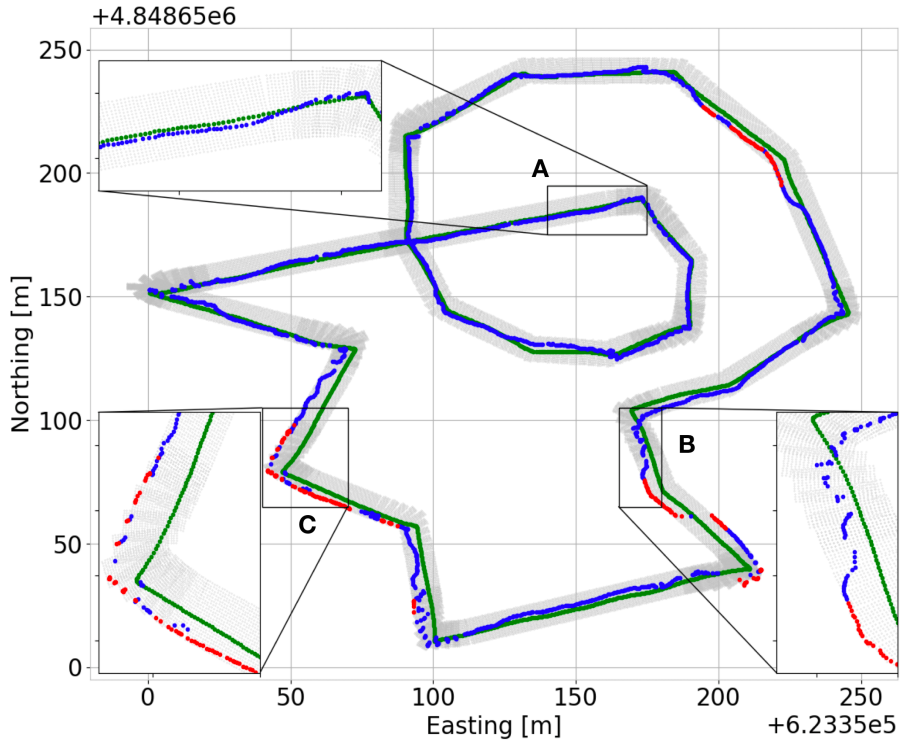
Image registration was performed on each of the six lighting conditions included in the dataset. The best performance was achieved on the morning lighting condition in terms of registration success rate and RMSE. As seen in Figure 3.6, the morning lighting condition has shadows on the west side of buildings and trees. In the GE images, the shadows also appear on the west side of the objects. The registration results for the morning lighting condition can be seen in Figure 3.7a. The grey dots indicate the reference image positions and the green dots indicate the ground truth. The blue dots are the localization results. It can be seen that the localization results follow the ground truth quite closely in most areas.

The evening dataset has large shadows appearing on the east side of objects which is opposite to how the shadows appear in the GE images. The localization performance on this evening dataset was the worst out of the six lighting conditions in terms of registration success rate and RMSE. The results can be seen in Figure 3.7b, where blue dots indicate successful localization results and red dots indicate localization results that were rejected by our outlier rejection method. While there are some areas along the path where the registrations that are deemed successful stray from the ground truth, most localizations with large errors are identified and rejected.

It should be noted that these are pure image registration results and they have not been passed through any sort of filtering. Adding in a prior estimate from VO and a motion model should help smooth out the path and propagate the estimate forward in areas where the localization was not successful. This is the focus of Chapter 4 where we integrate the localization results with the VT&R system.



(a) MORNING



(b) EVENING

Figure 3.7: Registration results showing our best (morning) and worst (evening) localization results. Grey dots indicate the reference image positions. Green dots indicate the ground truth live image positions. Blue dots indicate accepted localizations and red dots indicate rejected localizations. Shadows on the opposite sides of objects as compared to the GE reference images cause higher errors and more rejected registrations in the evening run.

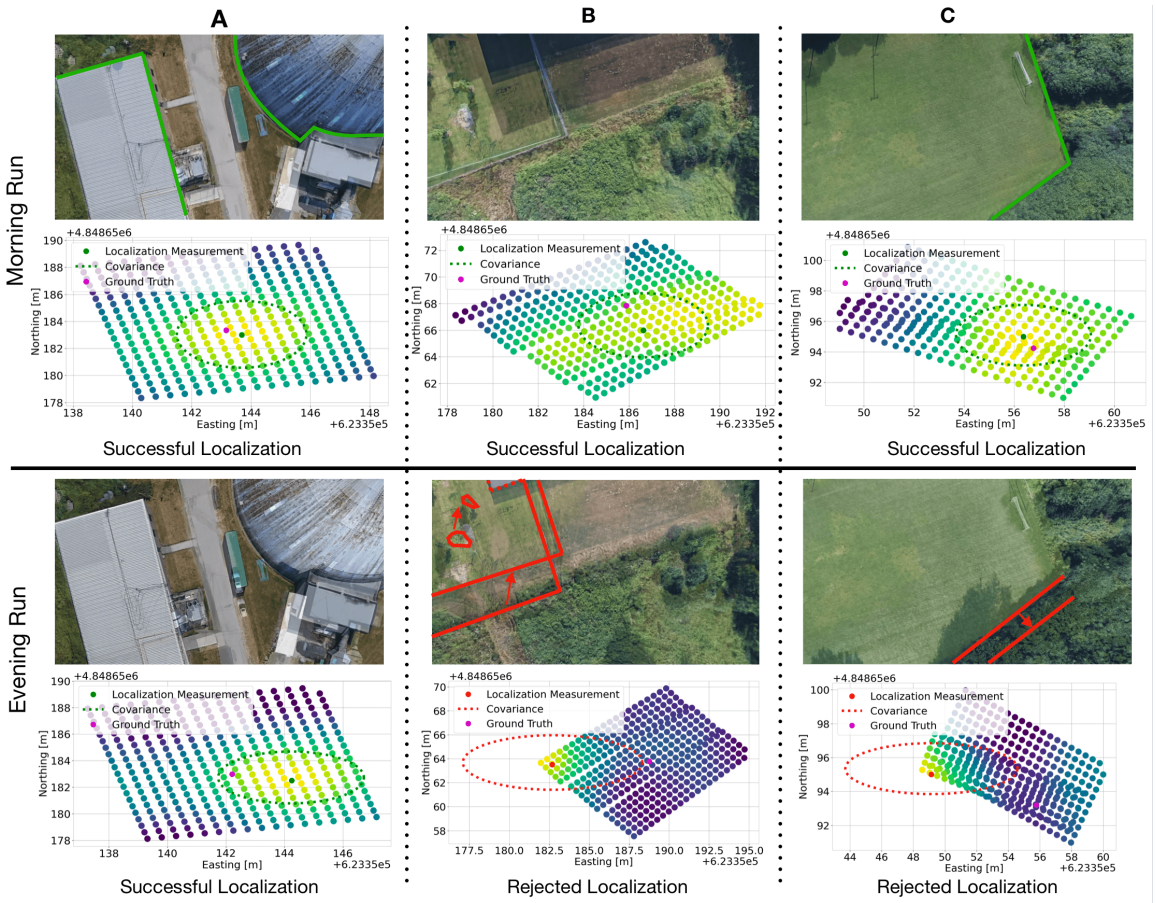


Figure 3.8: An example frame from each of the three areas indicated in Figure 3.7 for the morning lighting condition is shown at the top of this figure and for the evening lighting condition on the bottom. For each lighting condition, the top row shows the overlay between the live image and the GE reference image closest to the localization. The heat maps plotted in the bottom row show the value of the weights for each of the nearby GE reference images with yellow indicating a higher weight. The resulting localization and covariance is shown in green for successful registrations and in red for rejected registrations. The ground truth is shown in magenta.

Figure 3.8 shows some examples from three different areas along the path (indicated as A, B, and C in Figure 3.7) for the morning and evening light condition in the upper half and lower half, respectively. The top row for each lighting condition shows an overlay of a live image at that location and the GE reference image closest to the localization. The bottom row shows a heat map of the value of the weights of the nearby GE reference images computed from an inner-product with the

live image. Here a yellow colour indicates a higher weighting. The resulting localization from the weighted average computation and the covariance estimate are plotted as well. For the successful registrations (Morning A, B, C, and Evening A), the covariance envelope is smaller and the resulting localization is plotted in green. For the rejected localizations (Evening B and C), shadows cause the highest weights to occur on misaligned images at the very edge of the reference images. In these cases, the covariance that results from the weights is larger than in the successful registrations, very elongated, and does not pass our threshold for outlier rejection. In these unsuccessful cases, the rejected localization result and covariance are plotted in red.

3.5.3 Comparison with State of the Art

All Registrations

Most vision-based localization methods rely on features. Previously, Patel et al. [27] evaluated the ability of SURF features to match between GE rendered images and live images on the same path used here under various lighting conditions using the VT&R framework in [36]. Using the GE images for the teach pass and the live images for the repeat, features were only capable of producing less than 7% successes per repeat if a successful registration is defined as having greater than 30 Maximum Likelihood Estimation Sample Consensus (MLESAC) inliers. We are not currently aware of any works localizing GE images to real UAV images at a similar altitude and orientation as our flight path other than [27].

For performance evaluation, we compare our new method with the MI-based approach from [27] and were able to achieve comparable results on the same dataset. In Table 3.1, we present our RMSE for the longitude, latitude, and heading for all registrations on each of the six

runs. We achieve lower errors as compared to the results presented in [27]. We use the same search area as in [27] to select our subset of reference images, which correspond to a maximum RMSE of 6.4m.

Table 3.1: Comparison of Errors for All Registrations

Lighting Condition	**All Registrations RMSE**					
	longitude [m]		latitude [m]		yaw [degree]	
	Ours	MI [27]	Ours	MI [27]	Ours	MI [27]
Sunrise	**1.18**	1.87	**0.98**	1.47	**0.35**	2.80
Morning	**0.90**	2.24	**0.95**	1.39	**0.31**	2.97
Noon	**1.04**	1.26	**0.87**	1.02	**0.36**	2.70
Afternoon	**1.84**	2.14	**0.90**	1.57	**0.35**	2.63
Evening	**2.53**	4.09	**1.19**	3.63	**0.42**	5.25
Sunset	**1.64**	3.03	**0.97**	1.95	**0.37**	3.06

Successful Registrations

In Table 3.2, the error results from only successful registrations are compared against the errors from successful registrations in [27]. We also compare two different methods for determining whether a localization is successful with our method. For "Ours A", we use the outlier rejection scheme described in Section 3.4.1 to reject registrations with a high covariance estimate. For "Ours B", we use the outlier rejection from [27] along with our localization method. In [27], registrations are deemed unsuccessful if the localization is too far from the previous estimate. There is minimal difference in the performance between "Ours A" and "Ours B". The benefit of our outlier rejection scheme is that it is based purely on the covariance of the localization result and does not require a prior estimate.

Table 3.2: Comparison of Errors for Successful Registrations

			Lighting Condition					
			Sunrise	Morning	Noon	Afternoon	Evening	Sunset
Registration Success		Ours A	98.8	**100**	**100**	**98.0**	90.0	97.4
		Ours B	**99.9**	**100**	**100**	**98.6**	**96.0**	**98.7**
		MI [27]	94.7	95.1	97.8	96.0	81.3	87.5
Successful Registration RMSE	Longitude [m]	Ours A	**1.05**	**0.90**	1.04	**1.64**	**2.16**	**1.37**
		Ours B	1.17	**0.90**	1.04	1.67	2.17	1.48
		MI [27]	1.10	1.02	**0.78**	1.69	3.03	1.95
	Latitude [m]	Ours A	0.97	0.95	0.87	**0.88**	1.18	0.96
		Ours B	0.97	0.95	0.87	**0.87**	**1.14**	**0.94**
		MI [27]	**0.71**	**0.58**	**0.61**	0.91	1.32	1.12
	Yaw [degree]	Ours A	**0.35**	**0.31**	**0.36**	**0.34**	**0.42**	**0.36**
		Ours B	**0.35**	**0.31**	**0.36**	**0.34**	**0.41**	**0.36**
		MI [27]	2.28	2.57	1.82	1.17	2.49	2.64

In comparison with [27], for all but the noon lighting condition, we achieve lower RMSE error on the longitude coordinate. For the latitude coordinate, we have lower RMSE error for three of the runs and for the other three runs we are an average of 0.3m higher. Our success rate of registrations is higher for all the lighting conditions. Particularly in the evening and sunset runs, we see an increase of approximately 10% in the success rate accompanied by a decrease in RMSE as compared to [27].

Storage and Runtime Comparison

The most significant advantage of our method over [27] is the substantial reduction in runtime. For the purpose of comparison, both methods were run on a Lenovo P52 laptop with an Intel i7 8th generation core, a Nvidia Quadro P2000 GPU, and 32 GB of RAM. With the MI based approach of [27], the average runtime to complete the 1.1km path used in the experiments is approximately 18000 seconds, or 5 hours. With our new method, the average runtime is only 221 seconds, or just under four minutes. This substantial reduction is largely due to the fact that a single MI calculation takes on average 109 milliseconds but a single kernel computation only takes 0.26 milliseconds. The full MI based registration can take between 5 and 35 seconds per frame due to the costly optimization component which requires warping the image and recomputing the MI up to 150 times per registration. Instead, the full kernel-based registration per frame takes between 0.09 seconds and 0.15 seconds.

Table 3.3: Runtime and Storage Requirements Comparison

	Comparison Method	
	Kernel (Ours)	MI [27]
Average Runtime for 1.1 km Path	**221 s**	18422 s
Average Time per Frame	**0.11 s**	9.23 s
Average Time per Comparison Computation	**0.26 ms**	109 ms
Storage Cost per Image	**4.2 kB**	88 kB
Storage Cost per m of Path	**0.241 MB**	0.722 MB
Fixed Storage Cost	158 MB	**0 MB**
Total Cost for 1.1km Path	**423 MB**	794 MB

Both approaches are still limited by the storage available on the UAV. By encoding the reference images, we only need 1000 numbers to represent each image. Recording these numbers as half-precision floats only requires 4.2 kB per image. In [27], the reference images are stored as 560×315 4-bit greyscale images requiring approximately 88 kB of storage which is 22 times as large. As a result of this reduction, we are able to render more images per meter of the path while still having a lower per meter storage cost, 0.241 MB compared to 0.722 MB. Encoding the images also makes the base comparison computation faster. An inner-product computation between two 1000 dim vectors takes on average 0.26ms, whereas a MI computation between two 176,400 pixel images takes on average 109ms. The disadvantage of our approach is that in addition to the autoencoded reference images, we must also store the weights for the trained neural networks in order to encode the live image on board. However, this is a fixed cost that does not increase with the length of the path. So for the 1.1km path in the dataset used, our total storage is still less than what is required by [27] for the same path. Computation and storage comparisons are summarized in Table 3.3.

3.6 Summary

In this chapter, we presented a means for localizing a real UAV image using a collection of encoded reference images rendered from GE. The kernel-based computation is fast and accurate. We evaluated our method on a dataset of images collected by a real UAV at six different times of day. The performance was comparable to the next best method [27] achieving RMSE of less than 3m on a 1.1km path. The largest benefit of our new method is that it runs in 1% of the compu-

tation time and has only half the storage requirements as compared to [27].

The ability to localize a live image to images from GE is one step further towards being able to operate a UAV autonomously in new, never before seen areas. Given a map rendered offline entirely comprised of GE images, we have now developed a means to localize the UAV against it. The next chapter looks at generating this map and integrating the new GE image-based measurement into the existing VT&R system onboard the UAV.

Chapter 4

Integration with VT&R

4.1 Introduction

The ultimate goal of this thesis topic is to develop a method to enable a UAV to fly autonomously in new, never-flown-before areas. Currently, the VT&R system onboard the UAV requires an outbound mapping pass flown under manual or GPS control. As well, because of the significant changes to scene appearance due to lighting changes under an aerial perspective and the brittle nature of features upon which the VT&R system is based, this mapping pass needs to be repeated before every autonomous flight. We seek to eliminate the need for this repeated mapping pass and replace it with an offline map creation system using GE images that can be used for repeated flights at various times of day.

The purpose of this chapter is to integrate the localization measurement obtained from the GE codebook of images as described in Chapter 3 with the VT&R system. In order to do so, we developed a map generation technique that can be performed entirely offline before any flight data is collected using only GE images and data that is available offline. The resulting map is fully compatible with the VT&R system.

We also developed an additional module to run inside the VT&R

localization pipeline that computes the localization measurement as described in Chapter 3. This measurement is then used as an additional factor in the optimization to find the transform from the map vertex to the live image. This allows it to be used with or without features. Adding the measurement to the VT&R system means we are able to take advantage of the existing infrastructure including easy integration with the UAV controller.

This chapter is organized as follows. First, we provide some background and details on the current VT&R system. Then, we go over the offline map generation procedure and how the new localization measurement is incorporated into the existing factor graph. Finally, we show some experimental results running the full VT&R system with these additions on real UAV data with the offline generated map.

4.2 Overview of VT&R

This section provides an overview of the Visual Teach and Repeat (VT&R) system and specifically how it has been implemented for use on a multirotor UAV.

VT&R consists of two stages: the teach pass where the map is created, and the repeat pass where the vehicle localizes to the generated map and operates autonomously. With the UAV, the focus is on completing emergency returns purely using vision after GPS fails [36]. For this application, the UAV completes an outbound teach pass under GPS or manual control. Then a GPS failure is simulated by turning off GPS, and the UAV returns autonomously along the path by travelling in the reverse direction purely using vision.

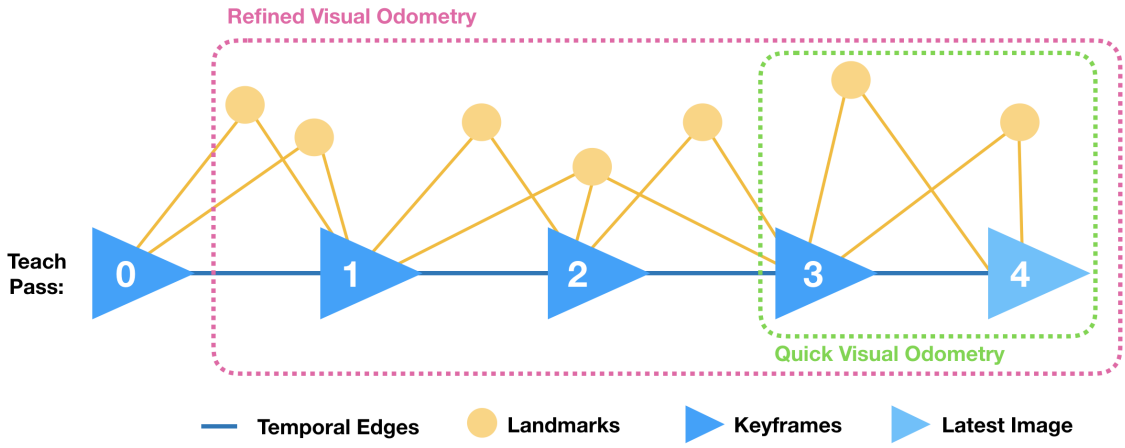


Figure 4.1: The teach pass is constructed by running VO on an outbound flight pass. Quick VO estimates the transformation from the previous keyframe to the current frame. Once a threshold for translation or rotation has been exceeded, a new keyframe is dropped and added to the graph. Refined VO runs bundle adjustment on the last five keyframes and updates the position of the landmark measurements.

On the outbound teach pass, a map is generated by saving keyframes as vertices in a Spatio-Temporal Pose Graph (STPG) based on the results of feature-based visual odometry (VO). Beginning with an initial keyframe, a stereo camera is used to extract features and Quick VO (QVO) is performed to estimate the transform from the latest keyframe to the current image. QVO optimizes this transform based on minimizing the cost function associated with the motion prior, the landmark measurements, and the IMU measurements. In Figure 4.1, QVO is solving for $\mathbf{T}_{43}$. Once the requirement to drop a keyframe has been reached, e.g., the threshold for rotation or translation has been exceeded, a vertex is added to the graph. The edge from the previous vertex to the newly dropped vertex stores the transformation between the keyframes. This type of edge, i.e., an edge between vertices on the same run, is referred to as a temporal edge. The 3D positions of all the landmarks observed at that keyframe are also saved in the vertex. Once a new keyframe is dropped, Refined VO (RVO) runs bundle adjustment on the last five keyframes using a similar cost function as in QVO. In

addition to optimizing the transformations between keyframes, RVO also adjusts the landmark positions.

In the original implementation of VT&R on ground vehicles, there is nothing to ensure the map is globally consistent. When running VT&R onboard the UAV, it has been noted that the results from VO are consistently under scaled in comparison with the ground truth from GPS. This is one of the difficulties of using a stereo camera with a small baseline at higher altitudes. Since the current application for UAVs is emergency returns after GPS failure, this means that the GPS is available during the teach pass. GPS can then be used as a cost term during the teach pass to help improve this scale difference.

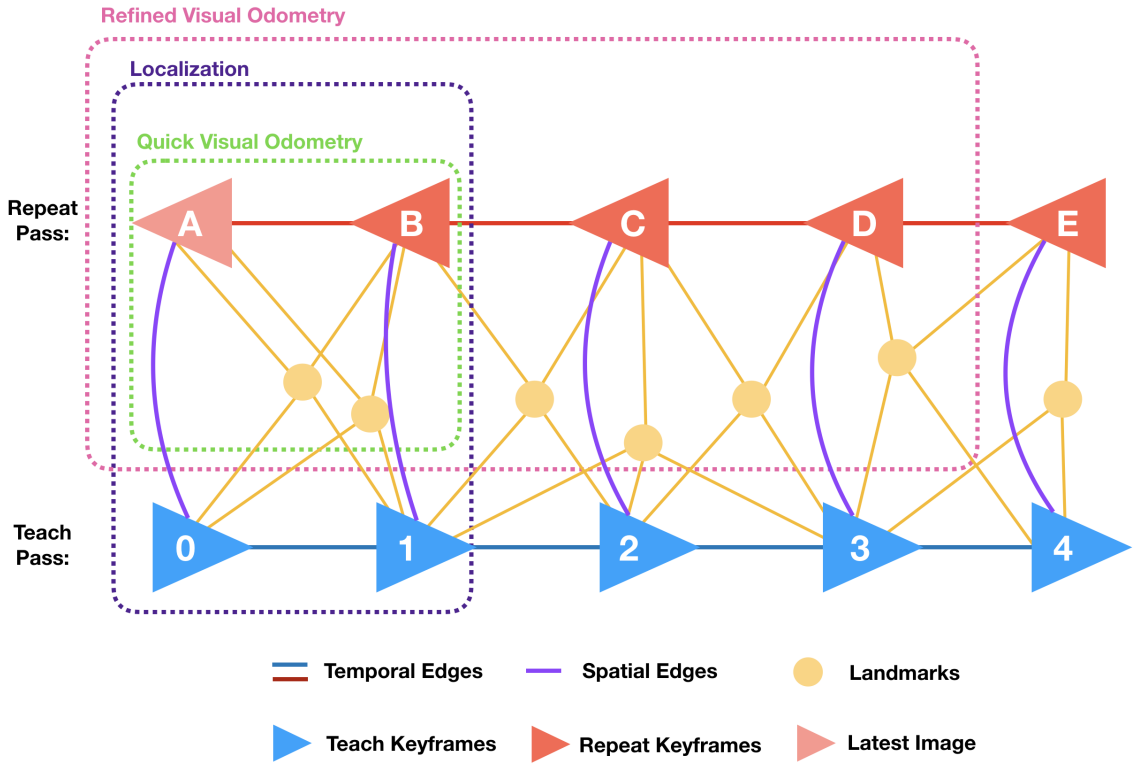


Figure 4.2: The repeat pass of VT&R on the UAV runs in the opposite direction of the teach pass. QVO and RVO run as in the teach pass but when a keyframe is dropped it is also localized to the map. Additional spatial edges between vertices in the repeat pass and the teach pass store the transformation from the map keyframe to the repeat keyframe.

On the repeat pass, QVO and RVO run as described above, but once a new keyframe is dropped it must also be localized back to the teach pass in order to correct the drift associated with VO. VT&R keeps track of the spatially closest vertex in the teach pass as the estimate for the UAV pose evolves. The localization problem is trying to find the transformation from this closest teach vertex to the new keyframe dropped during the repeat pass. Looking at Figure 4.2, this is solving for $\mathbf{T}_{\mathrm{A0}}$. The cost function consists of terms from the landmark measurements observed at both keyframes and a prior estimate of the transformation. The prior estimate comes from compounding the transforms already stored in the map from VO and the previous localization, i.e. $\mathbf{T}_{\mathrm{AB}}\mathbf{T}_{\mathrm{B1}}\mathbf{T}_{10}$. Adding this prior term smooths out the localization results.

4.3 Offline Map Creation

In order to create a map offline, we need to replicate the graph generation that occurs during the teach pass of VT&R. As described above, VT&R runs QVO and RVO to generate a graph consisting of vertices at each keyframe that store the observed 3D landmark positions and edges connecting these vertices containing the transformations between keyframes. The graph is saved using the Robochunk library. Robochunk is a fast serialization library based on Google's protocol buffers.

The first step is to decide on a desired path and record the world coordinates every 1m along this path. These will be the coordinates of the teach vertices. Next, a monocular GE image needs to be rendered at each vertex pose. A C++ script has been written to complete the next steps automatically when provided with a list of path coordinates

and the folder location of the corresponding images.

Using the Robochunk tools, a new graph is created. The first vertex that gets added to the graph will act as the origin. To insert the next vertex, the relative transform from the previous camera frame needs to be computed from the global camera frame coordinates. With the known global coordinates we can construct a transform, $\mathbf{T}_{iw}$, from the world frame, $\mathcal{F}_w$, to the camera frame, $\mathcal{F}_i$. Since we have $\mathbf{T}_{iw}$ for all the images we can use them to find the transform from the previous camera frame to the next camera frame: $\mathbf{T}_{i,i-1} = \mathbf{T}_{iw}\mathbf{T}_{i-1,w}^{-1}$. $\mathbf{T}_{i,i-1}$ is then stored in the edge connecting vertex i and $i-1$. This edge is designated as a temporal edge.

The next step is to save the 3D positions of the visible landmarks at each vertex. Since we are using monocular images we do not have a pair of stereo images separated by the camera's baseline. Instead, we use adjacent images to act as a pseudo-stereo pair where we use the distance between them as the baseline. We use the existing feature detection and matching modules from VT&R to extract and match features between adjacent image pairs.

In order to reject outlier matches, we check that the match satisfies an epipolar constraint. Since the image pairs are not separated by a translation purely along the x-axis (as would be the case with most stereo pairs), the epipolar line will run on a diagonal through the image. We can find this line using [5]

$$\mathbf{q}_i^T \mathbf{F}_{i,i-1} \mathbf{q}_{i-1} = 0, \tag{4.1}$$

where

$$\mathbf{F}_{i,i-1} = \mathbf{K}^{-\mathrm{T}} \mathbf{E}_{i,i-1} \mathbf{K}^{-1}, \tag{4.2}$$

and $\mathbf{K}$ is the known camera calibration matrix. We construct the essential matrix, $\mathbf{E}_{i,i-1}$, using the known rotation, $\mathbf{C}_{i,i-1}$, and translation, $\mathbf{r}_{i,i-1}^{i-1}$, between the camera poses:

$$\mathbf{E}_{i,i-1} = \mathbf{C}_{i,i-1}\mathbf{r}_{i-1}^{i,i-1\wedge}. \tag{4.3}$$

Then with the coordinates of the feature in image i, $\mathbf{q}_i = [u_i\, v_i\, 1]^T$, and image $i-1$, $\mathbf{q}_{i-1} = [u_{i-1}\, v_{i-1}\, 1]^T$ our constraint is

$$\mathbf{q}_i^T \mathbf{F}_{i,i-1}\mathbf{q}_{i-1} < \delta, \tag{4.4}$$

where we choose $\delta = 0.01$.

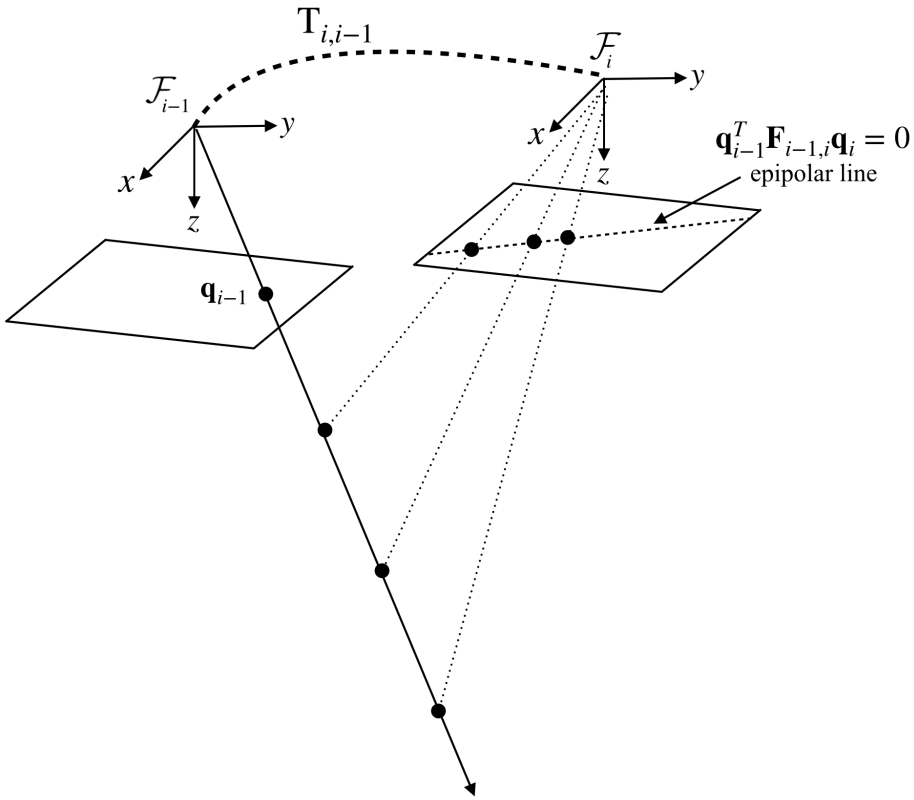


Figure 4.3: A feature at image coordinates $\mathbf{q}_{i-1}$ can lie anywhere along a line extending from the camera's pinhole through that point on the image plane depending on its depth away from the camera. This depth also affects where that feature will lie on a second image plane separated from the first image plane by a translation, but the feature must lie on an epipolar line that satisfies $\mathbf{q}_i^T \mathbf{F}_{i,i-1}\mathbf{q}_{i-1} = 0$.

If a matched pair passes the outlier rejection, we then solve for the depth and 3D landmark position. We can write an equation relating the image coordinates of the feature and the 3D position of the landmark in that camera frame using the known calibration quantities:

$$\begin{bmatrix} u_k \\ v_k \end{bmatrix} = \frac{1}{z_k} \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \end{bmatrix} \begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix}, \quad k = i, i-1. \tag{4.5}$$

For the desired path used in the experiments, the camera frames are all at a constant altitude and heading, see Figure 4.4. Therefore, the transformation matrix is simply

$$\mathbf{T}_{i,i-1} = \begin{bmatrix} 1 & 0 & 0 & \Delta x_{i-1}^{i,i-1} \\ 0 & 1 & 0 & \Delta y_{i-1}^{i,i-1} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.6}$$

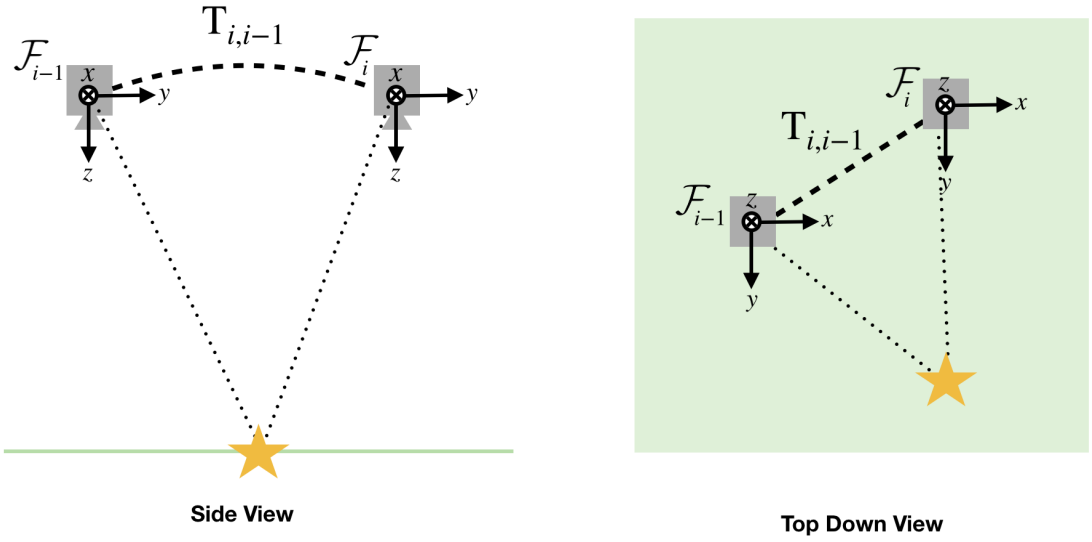


Figure 4.4: All the GE images are rendered at a constant altitude and heading with the cameras oriented in the nadir direction. Between the camera frames, there is only translation along the x and y axes.

We can then relate the 3D landmark positions as observed in the two camera frames:

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \mathbf{T}_{i,i-1} \begin{bmatrix} x_{i-1} \\ y_{i-1} \\ z_{i-1} \end{bmatrix}, \tag{4.7}$$

which gives three equations:

$$x_i = x_{i-1} + \Delta x_{i-1}^{i,i-1}, \tag{4.8}$$

$$y_i = y_{i-1} + \Delta y_{i-1}^{i,i-1}, \tag{4.9}$$

$$z_i = z_{i-1}. \tag{4.10}$$

Since the flight path is at a constant altitude, we can see that the depth from each camera to the landmark is the same, i.e., $z_i = z_{i-1}$, which we will simply denote as z. Substituting (4.8) and (4.9) into (4.5), we arrive at

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \frac{1}{z} \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \end{bmatrix} \begin{bmatrix} x_{i-1} + \Delta x_{i-1}^{i,i-1} \\ y_{i-1} + \Delta y_{i-1}^{i,i-1} \\ z \end{bmatrix}. \tag{4.11}$$

Using (4.11) and (4.5) with $k = i - 1$, we can write two equations for z:

$$z = \frac{f \Delta x_{i-1}^{i,i-1}}{u_i - u_{i-1}}, \tag{4.12}$$

$$z = \frac{f \Delta y_{i-1}^{i,i-1}}{v_i - v_{i-1}}. \tag{4.13}$$

In a perfect scenario, these two equations would give the same result for

z. In actuality, due to noise in the feature detection they give slightly different answers, so we then average the two results for z to get a depth measurement.

Once we have z, we can plug this back into (4.5) with $k = i$ rearranged to solve for x_i and y_i:

$$x_i = \frac{z}{f}(u_i - c_x), \tag{4.14}$$

$$y_i = \frac{z}{f}(v_i - c_y). \tag{4.15}$$

Now that we have the 3D landmark positions, $[x_i\, y_i\, z]^T$, they can be saved in the ith graph vertex.

There are some additional quantities that also need to be saved in the graph in order for VT&R to run properly. This includes storing the camera calibration in the vertex and the transform from vehicle to sensor, $\mathbf{T}_{sv}$. For $\mathbf{T}_{sv}$, we save the stationary transform, i.e., the rotation component is fixed in the nadir direction and the translation component is fixed at [0.039m, 0.125m, 0.168m] at each vertex.

At this point, we have generated a graph that contains the same information as if it had been generated using the VT&R teach system except we constructed the edge transforms using global coordinates for each vertex, which makes our map globally accurate instead of just locally consistent.

4.3.1 Adding Encoded Reference Images into the Map

The next step is to add the encoded reference images into the generated map. In order to do so, all the images must be rendered ahead of time from GE and encoded as described in detail in Chapter 2. Using the coordinates for each reference image, they are assigned to the closest graph vertex. Each vertex will have around 42, 1000-dimensional en-

coded vectors. Since Robochunk, which is the library used to save the VT&R graph, stores all data as a one-dimensional array, the 42 vectors are flattened. Because of corner cases where there are slightly more or less than 42 vectors, the exact number of vectors is also saved in the vertex to be used when loading from the graph and reshaping back into a 2D array.

We transform the global coordinates for each encoded vector to the frame of the first map vertex, which acts as the origin for the whole map. We then save these transformed coordinates in the same vertex with their corresponding encoded vector.

With the addition of these encoded vectors and coordinates, the offline generated map now contains all the information necessary to run VT&R and localize using the GE reference images.

4.4 Modifications to the Visual Teach and Repeat System

With the offline, generated map, we have eliminated the need for an outbound mapping pass. This means that the UAV can fly out autonomously along the path instead of only being able to return backwards along the outbound path. With this change, the repeat pass is now flown in the same direction as the teach pass, see Figure 4.5 as compared to Figure 4.2.

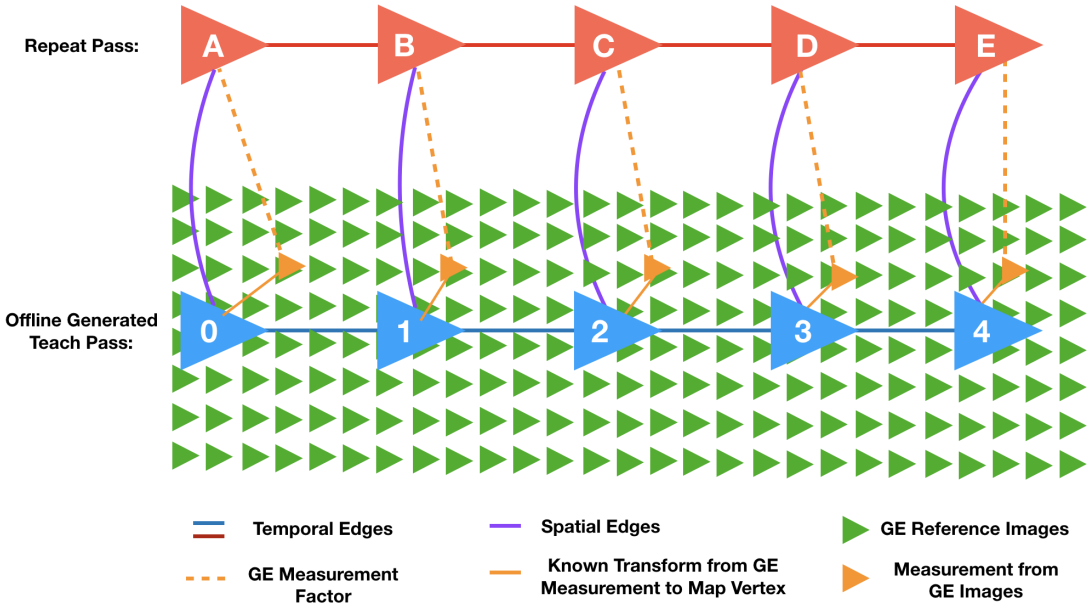


Figure 4.5: Using the offline generated map, the UAV now flies in the same direction as the map during the repeat pass. In the generated teach pass we are storing the encoded reference images, which will be used to compute an additional localization measurement shown by the orange triangles. The dotted orange lines indicate the additional factor that we are trying to minimize in the localization optimization.

In addition to the vertices and landmarks stored in the map in the original VT&R, we are now also storing a codebook of multiple GE reference images at each vertex. From this codebook, we will be able to compute an additional localization measurement to be used during the localization optimization.

4.4.1 Running the Pytorch Network in C++

The original encoder network was written in Python 3 using PyTorch 1.2. PyTorch includes a C++ API that can be used after installing the binary distribution. To use the C++ library in VT&R, the `Torch` package and folder location need to be added to the `CMakeLists` file in VT&R `asrl__navigation`. The `torch/torch.h` file can then be included in the relevant C++ files.

PyTorch also has the built-in capability to export a model, including the network structure and weights, to a serialized representation that can be loaded and executed purely from C++. This is enabled by Torch Script. In Python, running the command:

```
torch.jit.script(TrainedEnocderModel)
```

creates an instance of `ScriptModule` that is ready for serialization. In order to serialize it, simply call save on the instance and pass it a `.pt` filename and location. In the C++ file, after including `torch/torch.h`, the serialized model representation can be loaded using:

```
torch::jit::script::Module encoder =
torch::jit::load("path/to/.pt/file") .
```

4.4.2 Google Earth Localizer Module

A new module named `ge_localizer` was created and added to the localization assembly. This module uses the method described in Chapter 3 to obtain a localization measurement and then saves it in the cache to later be accessed by the localization optimization. The new module runs after the `ransac` module and just before the `steam` module as shown in Figure 4.6.

In the `ge_localizer` module, the first step is to retrieve the live image, resize it to 320×160, and normalize all the pixel values so they are between 0 and 1. The `torch::from_blob` function is used to convert the image from a `opencv::Mat` variable type to a `torch::Tensor` type. As a result of the opencv conventions, the third dimension of the converted tensor needs to be moved to the second position. This tensor can then be added to an array of `torch::jitt::IValue`s which can be used as the input to the encoder model by running

`encoder.forward(inputs.toTensor())`. The output will be a tensor containing the encoded vector representation of the live image.

The next step is to retrieve all the nearby encoded reference images stored in the map. VT&R keeps track of the closest map vertex, so we take advantage of this existing capability to select the nearby reference images. Starting from two vertices prior to the closest map vertex and ending at ten vertices after, all the encoded references images stored in these vertices are loaded. This covers an area along the path from approximately 2m behind and 10m ahead of the current image. Along with the encoded images, the corresponding coordinates are loaded from the vertices as well. Stacking all the loaded reference images, the weights are computed according to Equation 3.1. Following the rest of the procedure in Chapter 3, we also compute the standard deviation of the weights and threshold all weights less than half a standard deviation from the max value to zero and normalize. All the loaded coordinates are stacked and we use (3.3) to obtain our measurement.

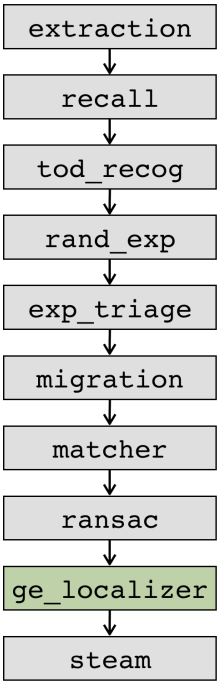


Figure 4.6: The original localizer assembly modules are shown in grey and the newly added module, `ge_localizer` is shown in green.

The resulting measurement will be the position of the live camera frame relative to the origin of the map/the first vertex, $\mathbf{p}_0^{q0} = [x_0^{q0}\, y_0^{q0}]^T$ because all of the coordinates are saved in the origin frame, $\mathcal{F}_0$. In the `steam` module, the optimization is solving for the transform from the closest map vertex to the live camera frame, so we convert this measurement into the closest map vertex frame, $\mathbf{p}_m^{qm} = [x_m^{qm}\, y_m^{qm}]^T$, and save it in the map cache.

4.4.3 Scaling the Visual Odometry

The resulting transforms from both QVO and RVO have been observed to have a translation component that is significantly smaller than the ground truth value, see Figure 4.8. The transform from QVO, which is based on feature matches between the live image and the last keyframe image and a motion model, is used to generate a prior estimate for the localization optimization. This prior estimate helps to smooth out the localization measurements and propagates the pose estimate forward in areas where the GE measurements are classified as outliers. If the scale of the VO is significantly off, it is difficult to rely on the VO measurements for a longer period of time without getting off the path and being unable to recover.

The VO scale being much smaller than the true scale was less noticeable when the map pass was generated online using VO because the map was also much smaller. When generating the map with global coordinates this issue becomes much more apparent and so we need to estimate the scale online. Similar to the approach taken in [27], we estimate a scale factor for the VO as the UAV travels along the path based on a sliding window of the most recent GE measurements.

We do this by maintaining a rolling average, $\bar{d}_{\text{VO}}$ of the last 30 QVO distances between keyframes. Since QVO estimates the transform from the previous keyframe to the current image, we compute the magnitude of the translation component of this transform every time a new keyframe is dropped. This is then used to update $\bar{d}_{\text{VO}}$.

Finding the distance between adjacent GE measurements requires recording the last GE measurement in the origin frame, $\mathbf{p}_0^{i-1,0}$. Then when we obtain the next GE measurement expressed in the origin frame, we can compute $\mathbf{p}_0^{i,i-1} = \mathbf{p}_0^{i,0} - \mathbf{p}_0^{i-1,0}$. We use the magnitude of $\mathbf{p}_0^{i,i-1}$ to compute the distance between adjacent GE measurements

and use it to maintain our rolling average of the last 30 distances, $\bar{d}_{\text{GE}}$. Since the GE measurements can include outliers, if the current distance between GE measurements is greater than twice or less than half of $\bar{d}_{\text{GE}}$, it is not used to update the average.

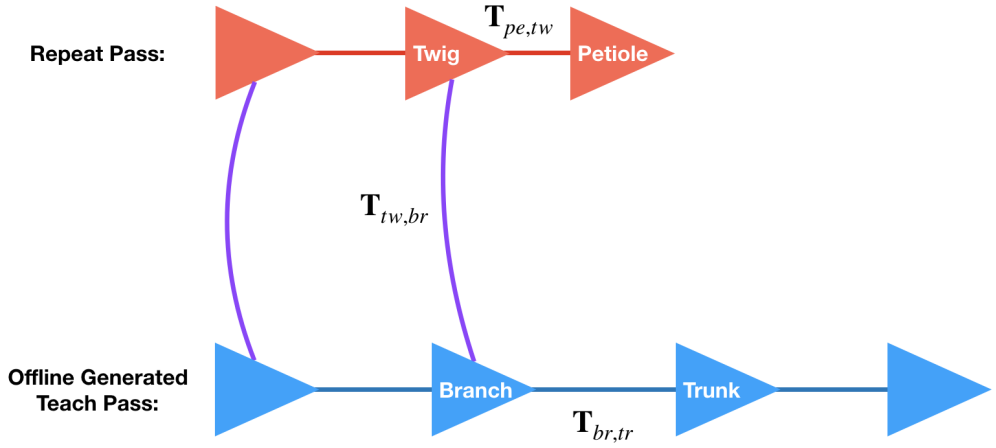
To understand how the computed scale is used to adjust the QVO result, we need to explain what happens in the localization chain. The localization chain keeps track of the following vertices (see Figure 4.7a):

- **Twig Vertex**: the last keyframe in the repeat pass that has been localized
- **Branch Vertex**: the vertex to which the twig vertex has been localized
- **Petiole Vertex**: the newly dropped keyframe that is going to be localized next
- **Trunk Vertex**: the closest map vertex to the petiole vertex, and the vertex against which the petiole vertex will be localized (can be the same as the Branch Vertex)

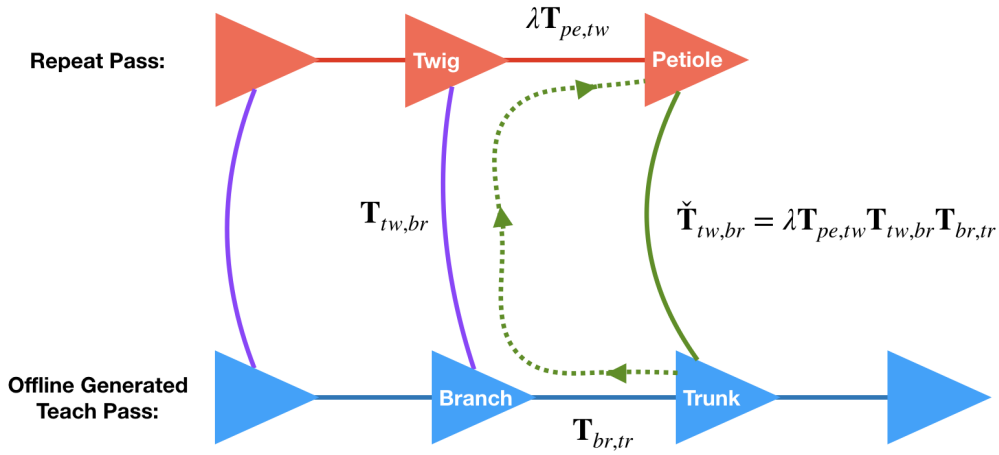
Once a new keyframe is dropped, but just before it is localized, the localization chain resets the twig and branch vertices. To do this it compounds $\mathbf{T}_{pe,tw}$, $\mathbf{T}_{tw,br}$, and $\mathbf{T}_{br,tr}$ and saves this as a new prior estimate for $\check{\mathbf{T}}_{tw,br}$. Instead, we now multiply the translation component of $\mathbf{T}_{tw,br}$ by the scale factor, λ, which we compute as

$$\lambda = \frac{\bar{d}_{\text{GE}}}{\bar{d}_{\text{VO}}}, \tag{4.16}$$

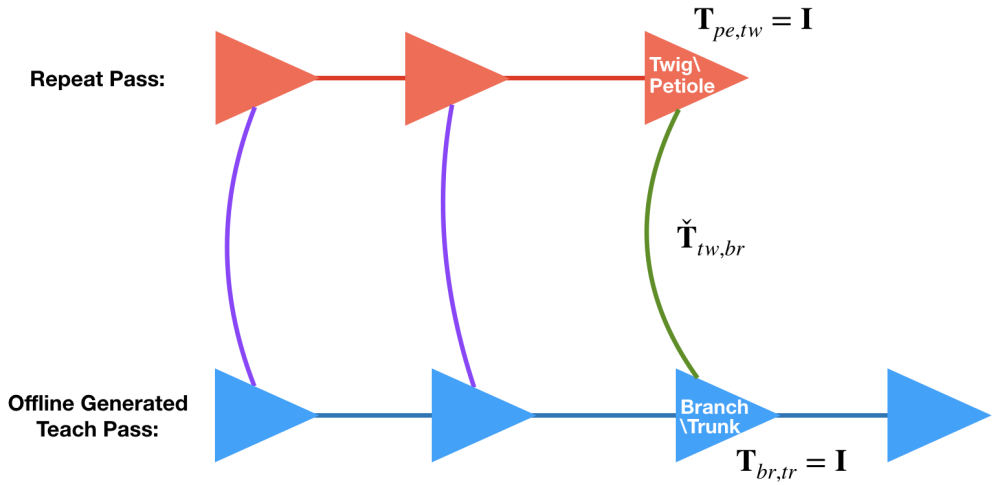
before computing the new $\check{\mathbf{T}}_{tw,br}$ value. This is shown in Figure 4.7b. Finally, the twig vertex is updated to be the same as the petiole vertex and the branch is updated to be the same as the trunk vertex. See Figure 4.7c.



(a) Before resetting the twig and branch vertices.



(b) Compounding the transforms, including the scaled $\mathbf{T}_{pe,tw}$, to get a new $\check{\mathbf{T}}_{tw,br}$.



(c) Updating the twig and branch vertices.

Figure 4.7: The three stages of resetting the twig and branch vertices.

The $\check{\mathbf{T}}_{tw,br}$ that is stored in the localization chain is accessed by the `steam` module to act as a prior in the optimization problem. After the optimization runs, $\check{\mathbf{T}}_{tw,br}$ is updated with the optimized value, $\mathbf{T}_{tw,br}$.

4.4.4 Adding GE Measurement to Localization Optimization

The localization optimization runs in the `steam` module shown in Figure 4.6. It uses the STEAM library to find the optimal transform, $\mathbf{T}_{qm}$, from the closest map vertex (i.e. the trunk vertex) to the live image based on a collection of cost terms. Prior to any modifications, these cost terms consisted of the landmarks observed by both the map vertex and the live vertex and a prior transform that comes from VO as explained in the previous section. Between the map vertex images from GE and the live images, there are very few or no landmark matches. The GE measurement term is added to the optimization so that even if there are no landmark matches we still have a cost term aside from just the prior.

From the `ge_localizer` module, we have a measurement $\mathbf{p}_m^{qm} = [x_m^{qm}\, y_m^{qm}]^T$ from the trunk vertex frame to the live image. For the path used in the experiments, the UAV flies at a constant altitude and heading and uses a gimbal to stabilize the camera. So we create a pseudo-measurement for the other degrees of freedom holding them fixed. We combine this with the GE measurement to get a measurement of map to query transform:

$$\mathbf{T}_{qm}^{\mathrm{GE}} = \begin{bmatrix} 1 & 0 & 0 & x_m^{qm} \\ 0 & 1 & 0 & y_m^{qm} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.17}$$

Since our measurement is in the form of a transform and the

state we are optimizing for is $\mathbf{T}_{qm}$, we can use the existing `steam::TransformErrorEval` error evaluator, which evaluates the following error:

$$\mathbf{e}_{\text{GE}} = \ln(\mathbf{T}_{qm}^{\text{GE}}\mathbf{T}_{qm}^{-1})^{\vee}. \tag{4.18}$$

To minimize the impact of outlier measurements, we use a robust cost function. Specifically, we use the Steam library `steam::DcsLossFunc(3)` initialized with a threshold of 3 [4]. This implements the following loss function:

$$L(e) = \begin{cases} \frac{1}{2}e^2 & e^2 \leq k^2 \\ \frac{2k^2e^2}{(k^2+e^2)} - \frac{1}{2}k^2 & e^2 > k^2, \end{cases} \tag{4.19}$$

where e is the whitened error norm and we have selected $k = 3$. This cost function downweights the GE measurement when the cost is higher as would be the case with outliers. We use a fixed covariance of $\text{diag}(10, 10, 0.1, 0.1, 0.1, 0.1)$ for all the GE measurements.

In the original VT&R implementation, the initial guess for $\mathbf{T}_{qm}$ comes from the output of the `ransac` module running on the feature matches. Since there are very few if any feature matches, we instead use $\check{\mathbf{T}}_{tw,br}$ which as described in the previous section comes from QVO and the previous keyframe localization. $\check{\mathbf{T}}_{tw,br}$ is also used in a prior cost term in the optimization problem. The prior cost term uses a standard L2 Loss function and the uncertainty is fixed at $\text{diag}(2, 2, 2, 2, 2, 2)$.

4.5 Experimental Results

We evaluate the modified VT&R system on the same dataset used for evaluation in Chapter 2. That is a 1.1km path flown around UTIAS at six different times of day by a DJI Matrix 600 Pro multirotor UAV equipped with a 3-axis DJI Ronin-MX gimbal and a StereoLabs ZED camera. RTK-GPS was used for ground truth. See Section 3.5.1 for a

more detailed description.

In the experiments, we generate the map offline using the script developed as explained in Section 4.3. The repeat is then conducted using the VT&R offline tools providing the location of the generated map and the real data as input.

4.5.1 Scaling the VO

We first evaluate the performance of our online scaling method. To plot the VO results, we compound all of the transformations. Figure 4.8 shows the original results from QVO after running VT&R on the morning dataset in magenta. The ground truth of this run is plotted in green so the original QVO is clearly undersized, but the shape of the path looks accurate. The light blue line shows the scaled QVO results, which are closer to the ground truth in terms of magnitude but the drift is more apparent.

Figure 4.9 shows the computed scale factor versus keyframe for all six runs. We wait until ten measurements have occurred before we start to estimate the scale, which accounts for the first 10 scale factors being equal to 1. Throughout the path, the scale varies typically staying between 2.0 and 2.5. Between the six runs, the scale seems to follow the same trend. The large increase in the evening and afternoon runs around keyframe 250 is likely caused by outlier measurements as this is one of the hardest areas along the path to localize due to the presence of shadows.

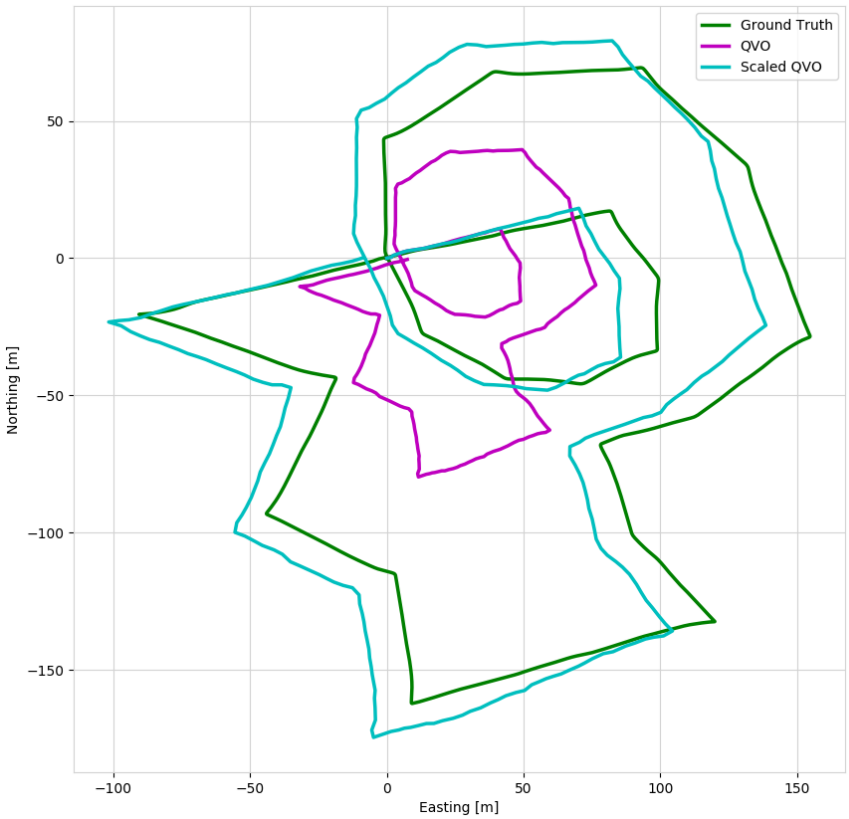


Figure 4.8: The results from QVO for the morning dataset are compounded and plotted. The original, unscaled QVO results are shown in magenta. The scaled QVO results are shown in light blue. The path starts at (0,0) and proceeds clockwise.

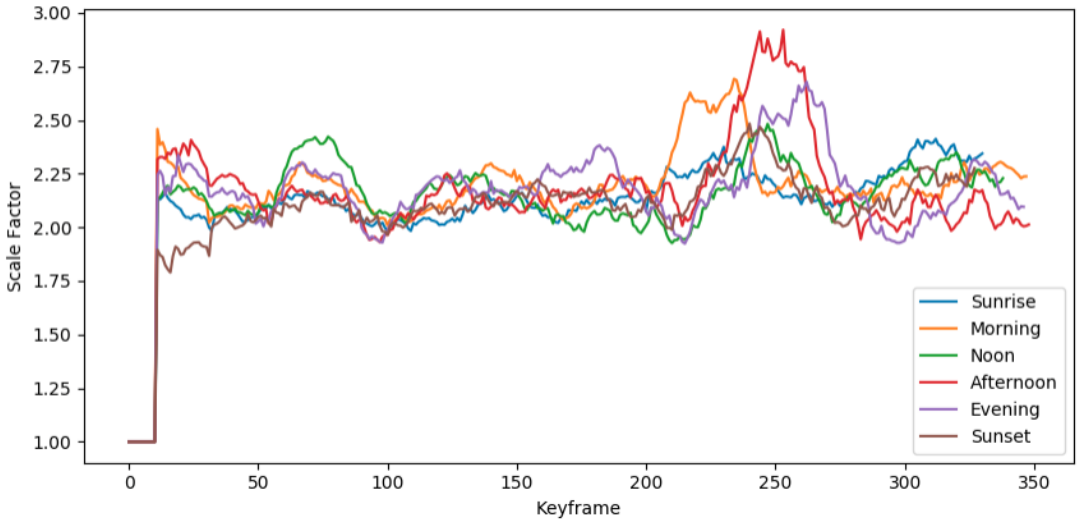


Figure 4.9: The estimated scale factor versus keyframes for each of the six dataset runs.

Table 4.1 shows the ground truth path lengths in meters for each of the runs. The length of the path from compounding all the unscaled QVO transforms is just under 50% of the ground truth length for all lighting conditions. When using the scaled QVO, we can see an improvement in terms of the magnitude of the path length. In all cases, it is either very close to the ground truth length or slightly larger.

Table 4.1: Comparison of Path Lengths

Lighting Condition	**Length of Path**					
	Ground Truth		Unscaled QVO		Scaled QVO	
	m	% of GT	m	% of GT	m	% of GT
Sunrise	1137.17	100	537.89	47	1145.46	100
Morning	1134.94	100	534.19	47	1180.25	104
Noon	1134.79	100	533.40	47	1153.75	102
Afternoon	1137.55	100	538.22	47	1184.03	104
Evening	1131.29	100	528.52	47	1149.33	102
Sunset	1131.82	100	540.87	48	1144.35	101

4.5.2 Localization Performance

We were able to successfully complete runs on all six datasets. By combining the scaled QVO and the GE measurements in our localization optimization, we were able to obtain an estimate that is globally accurate and smooth. In areas where the GE measurements were significantly far from the path, our robust cost function downweighted their contribution. The scaled QVO was able to propagate the estimate forward until more reasonable GE measurements were able to be obtained. This reliance on scaled QVO often occurred in the lower right portion of the paths as seen in Figures 4.13, 4.14, and 4.15.

There is a trade-off between smoothness and accuracy that comes with trusting the scaled QVO more than the GE measurements. But

as can be seen in the following figures our combined localization results perform better than either the scaled QVO or the GE measurements on their own.

There are some areas where the GE measurements are consistently offset from the path and this results in our localization being offset from the ground truth path as well. We can see this on the left side of Figure 1.7. There are significant shadows in this area which causes the GE measurement to align the tree line incorrectly. However, the localization result is able to stay close to the path and recover once it leaves that area.

Table 4.2 shows our RMSE on the latitude, longitude, and altitude coordinates for each of the six lighting conditions included in the dataset. As with the pure registration results, our best runs occurred on the sunrise and morning datasets where the shadow conditions most closely resemble those in GE. The worst runs occurred on the afternoon and evening dataset where we see shadows falling on the opposite sides of objects as compared to GE.

Table 4.2: Comparison of Errors

Lighting Condition	**RMSE for Localization Result**		
	longitude [m]	latitude [m]	altitude [m]
Sunrise	1.54	1.91	0.05
Morning	1.93	1.87	0.08
Noon	2.25	1.60	0.05
Afternoon	3.94	2.09	0.21
Evening	3.91	2.46	0.08
Sunset	2.71	1.70	0.07

Figure 4.16 shows the cumulative distribution of the latitude errors for all six datasets. The best performing runs, sunrise and morning, have approximately 80% of all latitude errors less than 2m. The worst

performing runs, afternoon and evening, have 60% of latitude errors less than 3m. Figure 4.17 shows the cumulative distribution of the longitude errors. All runs except for the evening run have 80% of longitude errors less than 2m. For the evening run, 75% are less than 3m.

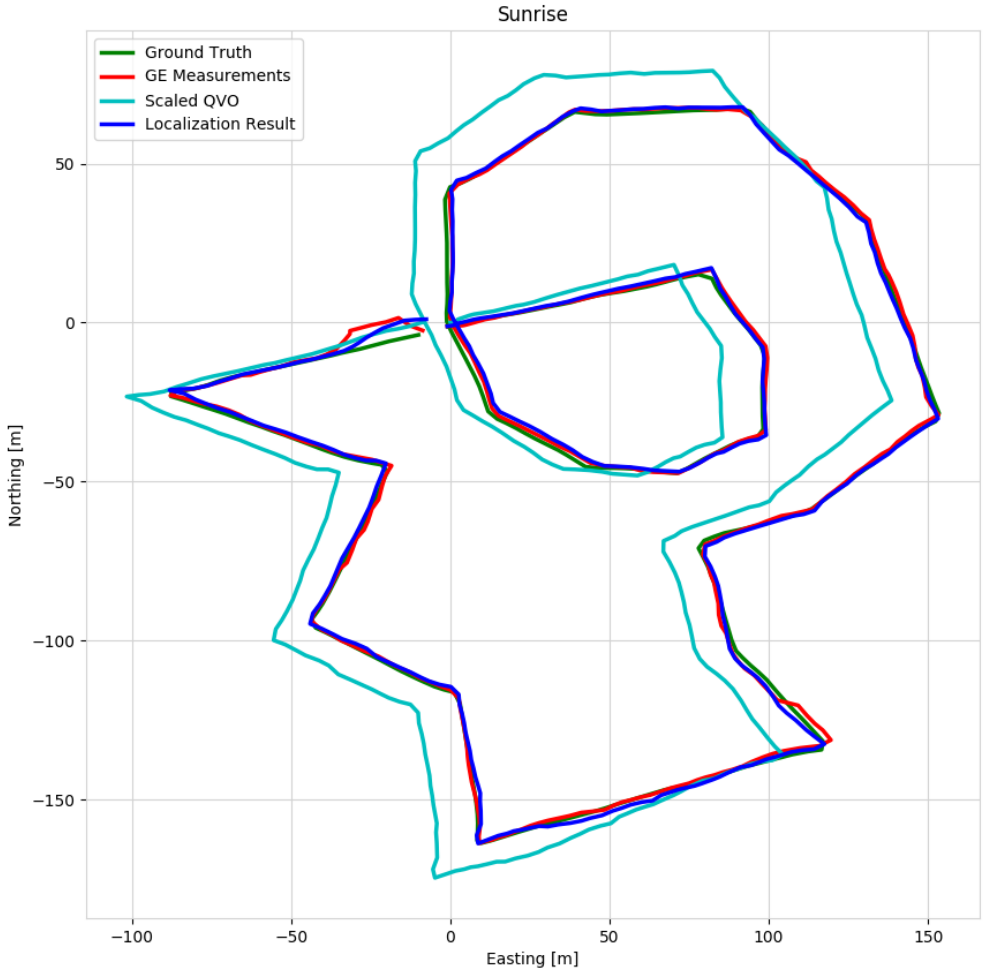


Figure 4.10: Position estimates for the 1132km sunrise path. The ground truth path is shown in dark green. The raw measurements from the GE localizer are shown in red. The compounded, scaled QVO estimates are shown in light blue. The optimized localization results that combine these two sources are shown in dark blue. For the sunrise path, the GE measurements are quite close to the ground truth and adding the scaled QVO factor helps smooth out the overall path.

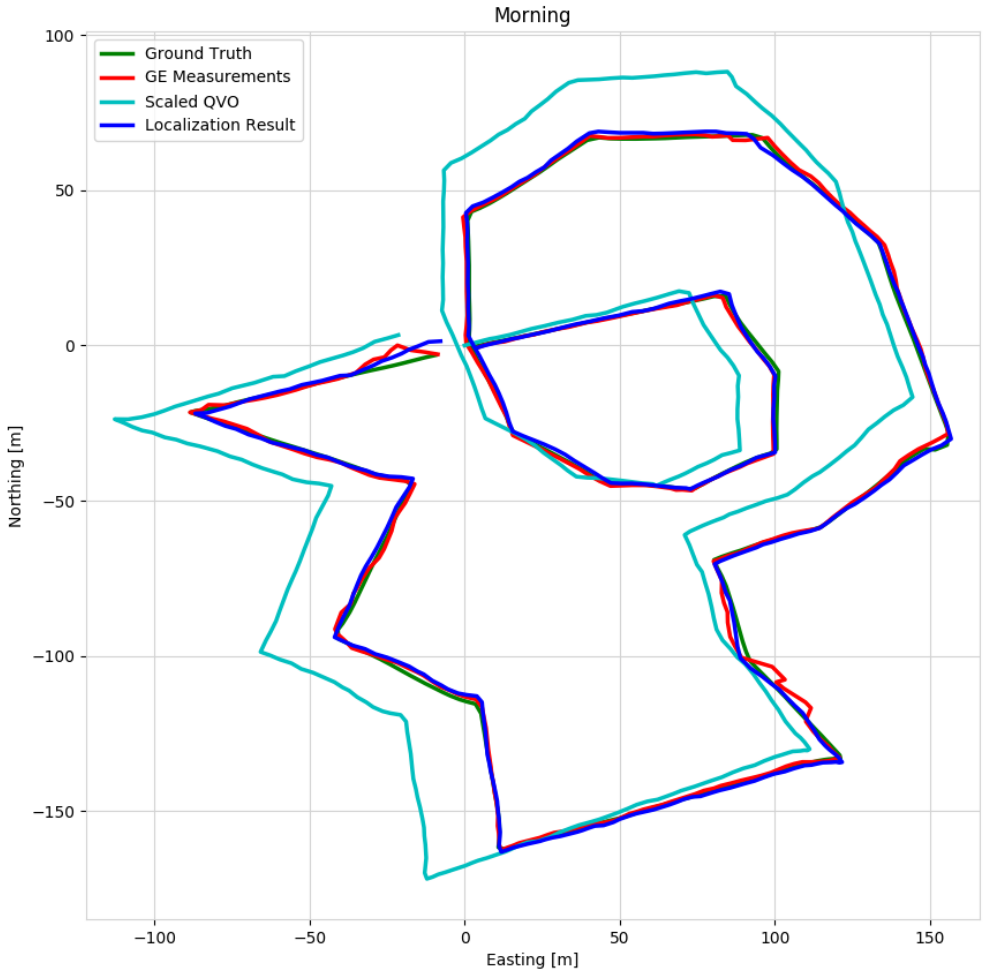


Figure 4.11: Position estimates for the 1132km morning path. The ground truth path is shown in dark green. The raw measurements from the GE localizer are shown in red. The compounded, scaled QVO estimates are shown in light blue. The optimized localization results that combine these two sources are shown in dark blue. For the morning path, most of the GE measurements are quite close to the ground truth except for a small part in the lower right part of the plot. Here we can see the raw GE measurements in red are off the path, but our estimate in blue is able to ignore these and propagate forward with scaled QVO.

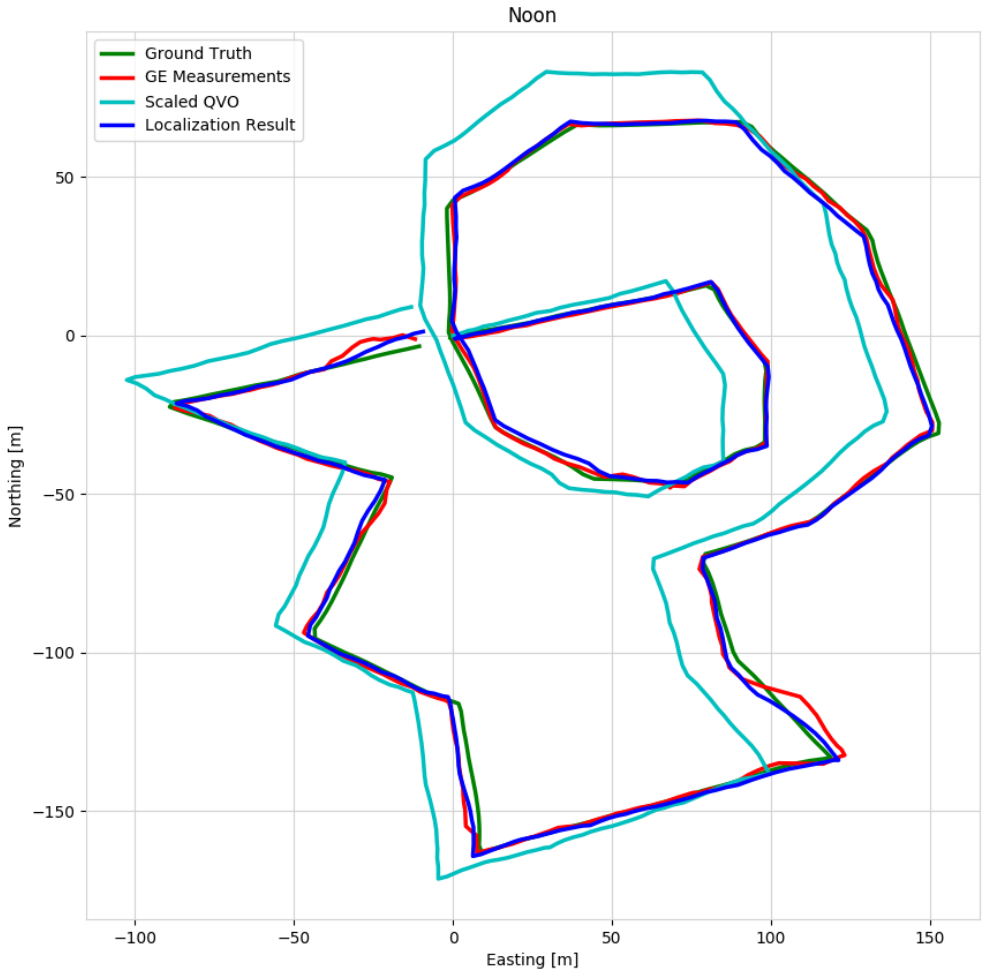


Figure 4.12: Position estimates for the 1132km noon path. The ground truth path is shown in dark green. The raw measurements from the GE localizer are shown in red. The compounded, scaled QVO estimates are shown in light blue. The optimized localization results that combine these two sources are shown in dark blue. For the noon path, the GE measurements are less smooth compared to the sunrise and morning runs but we can see how combining with the scaled QVO smooths out the result.

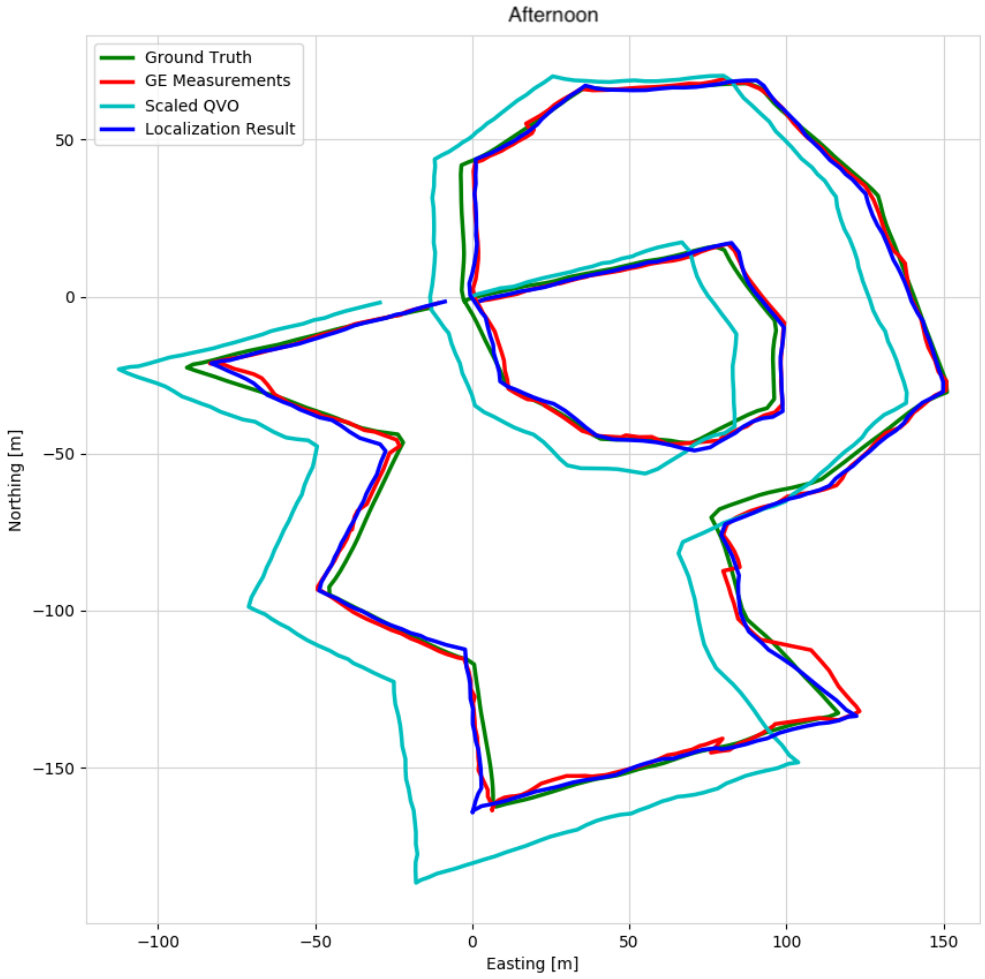


Figure 4.13: Position estimates for the 1132km afternoon path. The ground truth path is shown in dark green. The raw measurements from the GE localizer are shown in red. The compounded, scaled QVO estimates are shown in light blue. The optimized localization results that combine these two sources are shown in dark blue. For the afternoon path, we can see that the GE measurements are now starting to struggle. Along the bottom of the path, which corresponds to the wooded area, the measurements are quite jerky. We also see some areas on the lefthand side where the GE measurements are consistently off the path. This is an area where shadows cast by the treeline seem to cause a constant offset in the measurements. The combined localization result is still quite smooth and does a reasonable job of tracking the path.

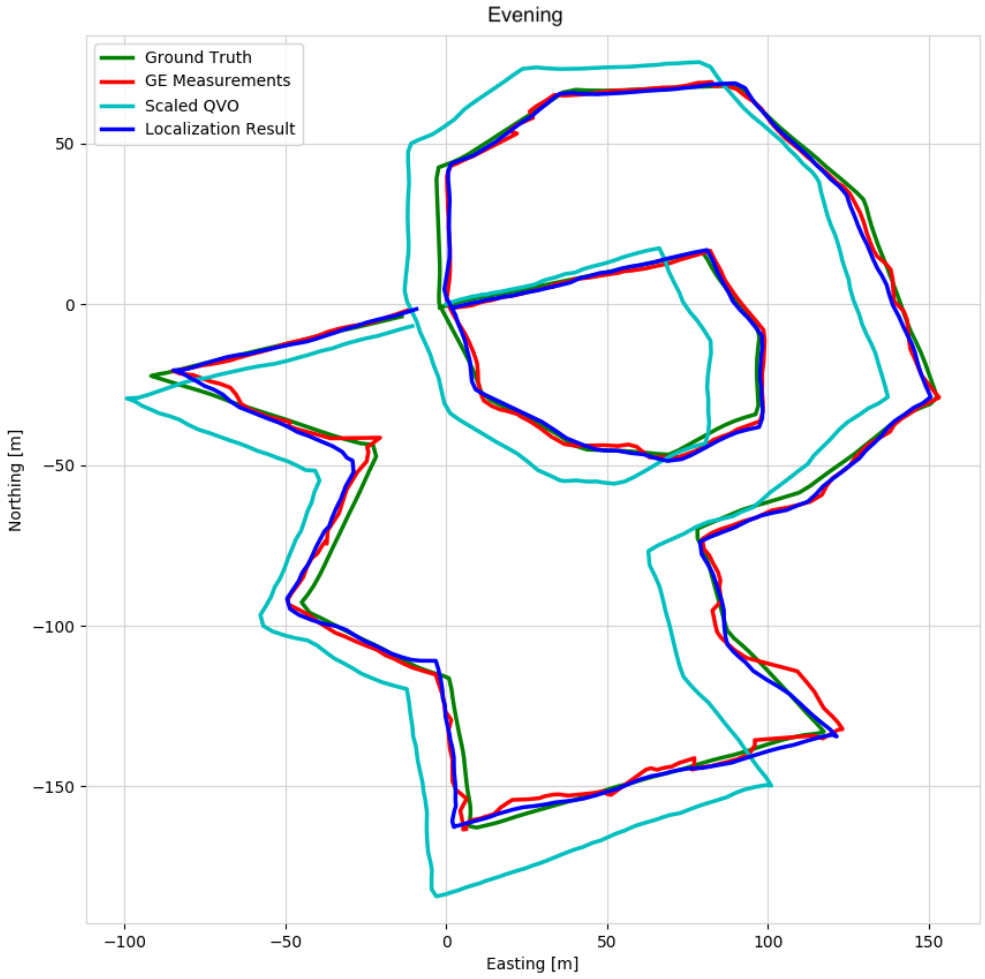


Figure 4.14: Position estimates for the 1132km evening path. The ground truth path is shown in dark green. The raw measurements from the GE localizer are shown in red. The compounded, scaled QVO estimates are shown in light blue. The optimized localization results that combine these two sources are shown in dark blue. For the evening path, we have the worst GE measurements. There are quite a few areas where the red line jumps around or is far from the path. This resulted in a more heavy reliance on VO in those areas.

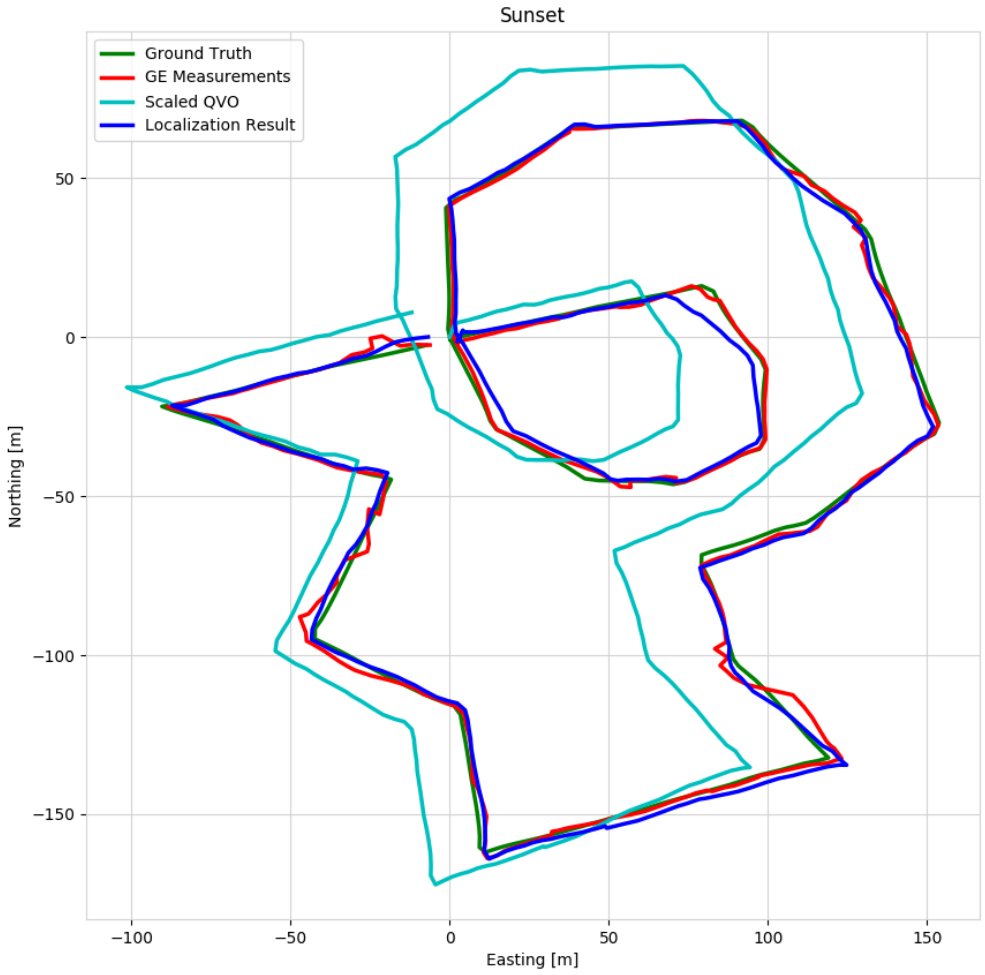


Figure 4.15: Position estimates for the 1132km sunset path. The ground truth path is shown in dark green. The raw measurements from the GE localizer are shown in red. The compounded, scaled QVO estimates are shown in light blue. The optimized localization results that combine these two sources are shown in dark blue. For the sunset path, we have quite a few bad GE measurements. However, by combining with the scaled QVO, our localization results do a reasonable job smoothing them out and following the ground truth path.

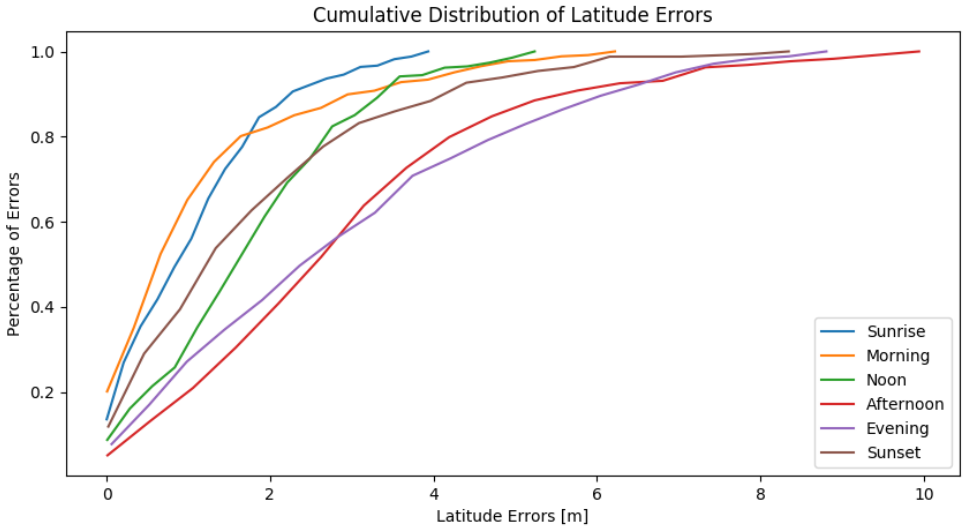


Figure 4.16: For the sunrise and morning runs, 80% of the latitude errors are less than 2m. 80% of the latitude errors for the noon and sunset runs are less than 3m. The afternoon and evening runs are the worst-performing runs with only 60% of the latitude errors being less than 3m.

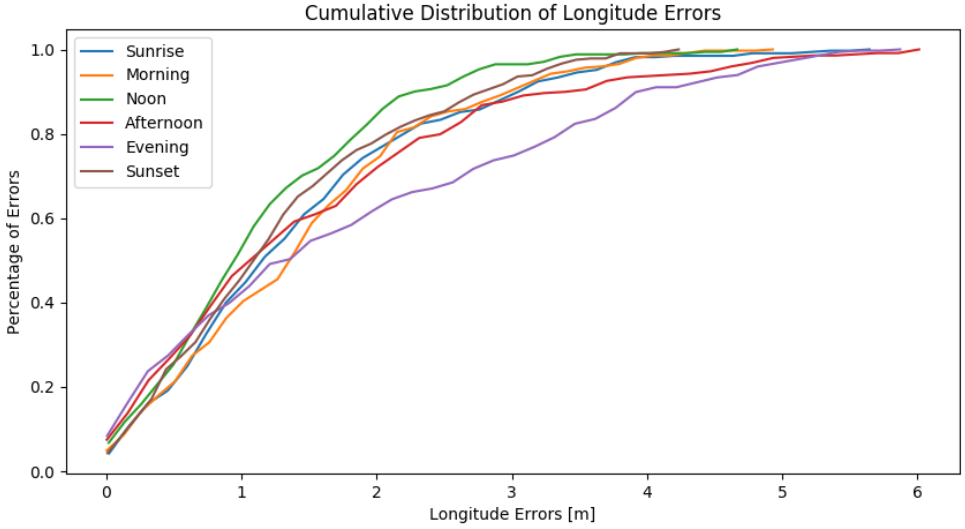


Figure 4.17: For every run except the evening run, approximately 80% of the longitude errors are less than 2m. For the evening run (purple), 75% are less than 3m.

4.6 Summary

In this chapter, we detailed a procedure for generating entirely offline a map that is compatible with the VT&R system using only images rendered from GE. We also modified the VT&R system to add the GE image-based measurement to the localization optimization. This involved converting the encoder model from Python to C++, adding a `ge_localizer` module to run during the localization pipeline, and including the GE image-based measurement in the localization optimization. We also added a VO scale factor estimator to improve the prior estimate from VO.

We evaluated this modified version of VT&R on a dataset of real UAV images collected at six different times of day. We were able to achieve less than 4m RMSE and for most of the runs, 80% of the errors were less than 2m. By adding the GE image-based measurement to the VT&R system, we can take advantage of the existing infrastructure onboard the UAV including being able to easily integrate with the controller in future work.

Chapter 5

Conclusions and Future Work

5.1 Summary and Contributions

The primary goal of this thesis was to make advances towards a system that generates an offline map using satellite images from which a UAV is able to localize against in real time. The primary benefit of such a system is that a UAV would be able to fly autonomously in new, never-before-flown areas without having to map it under manual or GPS control first. Towards this goal, we have made four main contributions.

The first contribution is the development of an autoencoder trained entirely on GE rendered images for a specific path that is capable of generalizing to real never-before-seen images. By rendering 42 images in GE per meter of the desired path, we obtain enough training data to teach the encoder to compress a 320×160 image containing 51,2000 pixels into a 1000 dimensional vector while maintaining the core structural information. This significant reduction in size has positive implications for storage requirements and computation time. The major benefit is that the network does not require any real UAV images during training.

The second contribution is a method for computing a localization measurement provided a subset of encoded reference images from GE and a live image. This method uses inner product kernels to make a

series of comparisons and generate a set of weights to indicate how well the reference image matches with the live image. These weights are then used alongside the known reference image coordinates to produce a weighted average localization measurement. We experimentally evaluated this method on a 1.1 km dataset consisting of real UAV images captured at six different times of day. We were able to achieve comparable performance to the current best method for localizing using GE images [27], i.e., RMSE errors under 3m. However, unlike [27], our method is likely capable of running in real time taking only 1% of the computation time of [27]. We also have lower storage requirements using approximately half the storage needed by [27]. Together with the first contribution, this resulted in a journal publication [7].

The third contribution is the means to automatically generate offline a VT&R compatible map containing the encoded GE reference images. Provided with the global coordinates every 1m along a desired path and a monocular GE image rendered at each of these coordinates, a script has been developed using the Robochunk library to produce a VT&R compatible map including recording the visible landmarks at each vertex. Also given a collection of encoded reference images and accompanying coordinates, each reference image will automatically be stored in the closest map vertex.

The fourth contribution is integrating the new GE image-based measurement into the existing VT&R code. This includes converting the encoder which was originally written in Python to C++ and creating a new `ge_localizer` module to run during the localization pipeline. An additional VO scale factor computation has been added as well to improve the prior estimate from QVO. The GE image-based measurement has been added to the localization optimization. The modified version of VT&R has been evaluated on a real dataset of UAV images

collected at six different lighting conditions using the offline generated map detailed above. There is some tradeoff between smoothness of the path and accuracy, but we are able to obtain less than 4m RMSE on all runs. For most lighting conditions, 80% of the errors are less than 2m.

5.2 Lessons Learned and Areas of Future Work

One of the largest lessons learned during this thesis is the importance of a good prior estimate. This system relies on the prior estimate to select the subset of GE reference images and when it is inaccurate, it can quickly lead into a bad feedback loop. If we choose the wrong subset of reference images based on the prior, we will get a bad GE measurement. This means the next selection will be even further off and the estimate will quickly devolve from the path.

During the initial trials, we did not compute a scale factor for QVO. In the harder areas of the path, the pose would quickly diverge from the path after relying on VO. To work around this, we added a separate scale estimation based on a sliding window of past measurements. In the future instead of estimating the scale separately, the scale could be added as a state variable into the localization optimization. This would likely add some robustness to the scale estimation.

Most of the errors in the GE measurements occur in areas where there are large shadows present in the live images but not in the GE images or vice versa. Some more work could be done on making the encoder more invariant to lighting conditions. A possible idea for this is using a denoising autoencoder as in [33]. By adding noise to the input training image, but not to the reconstruction target, the encoder learns to be more invariant to changes in the input image. Sundermeyer et al.

[33] have used this to match real images to simulated images, similar to our application.

Finally, the next step is to test this system onboard the UAV. Beginning with open-loops tests to confirm the system is capable of running in real time on the UAV hardware and then based on the onboard performance, potentially moving onto closed-loop tests with the controller.

Bibliography

[1] In: Google Earth. Image Credits: Google, Landsat/Copernicus. Location: UTIAS, North York, Ontario, Canada. Accessed: Aug 24, 2020.

[2] In: H. Bai, "Variational Autoencoder for face image generation in PyTorch", Github Repository. [Online]. Available: https://github.com-/bhpfelix/Variational-Autoencoder-PyTorch. Accessed: Jul. 10, 2020.

[3] P. Agarwal and L. Spinello. "Metric Localization using Google Street View". In: *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robotic Systems*. 2015, pp. 3111–3118.

[4] P. Agarwal et al. "Robust map optimization using dynamic covariance scaling," in: *In Proceedings of the IEEE International Conference on Robotics and Automation*. 2013, pp. 62–69.

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

[6] H. Bay et al. "Speeded-up robust features (SURF)". In: *Computer Vision and Image Understanding* 110.3 (2008), pp. 346–359.

[7] M. Bianchi and T. D. Barfoot. "UAV Localization Using Autoencoded Satellite Images". In: *"IEEE Robotics and Automation Letters (RAL)"* 6.2 (2021), pp. 1761–1768.

[8] M. Blöesch et al. “Vision based MAV navigation in unknown and unstructured environments”. In: *In Proceedings of the International Conference of Robotics and Automation*. 2010, pp. 21–28.

[9] Michael Calonder et al. “Brief: Binary robust independent elementary features”. In: *In Proceedings of the European Conference on Computer Vision*. 2010, pp. 778–792.

[10] Xi Chen et al. “Variational lossy autoencoder”. In: *In Proceedings of the International Conference on Computer Vision* (2016).

[11] G. Conte and P. Doherty. “An Integrated UAV Navigation System Based on Aerial Image Matching”. In: *In Proceedings of the IEEE Aerospace Conference*. 2008.

[12] P. Furgale and T. D. Barfoot. “Visual Teach and Repeat for Long Range Rover Autonomy”. In: *Journal of Field Robotics* 27.5 (2010), pp. 534–560.

[13] Kristen Grauman and Trevor Darrell. “The pyramid match kernel: Discriminative classification with sets of image features”. In: *In Proceedings on the IEEE International Conference on Computer Vision*. 2005, pp. 1458–1465.

[14] M Gridseth and T. D Barfoot. “DeepMEL: Compiling Visual Multi-Experience Localization into a Deep Neural Network”. In: *In Proceedings of the IEEE International Conference on Robotics and Automation*. 2019.

[15] X. Hou et al. “Deep feature consistent variational autoencoder”. In: *In Proceedings of the Applied Computer Vision IEEE Winter Conference*. 2017, pp. 1133–1141.

[16] Alex Kendall, Matthew Grimes, and Roberto Cipolla. “Posenet: A convolutional network for real-time 6-dof camera relocalization”.

In: *In Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*. 2015, pp. 2938–2946.

[17] C Linegar, W Churchill, and P Newman. "Made to measure: Bespoke landmarks for 24-hour, all-weather localisation with a camera". In: *In Proceedings of the IEEE International Conference on Robotics and Automation*. 2016, pp. 787–794.

[18] D. G. Lowe. "Distinctive image features from scale-invariant keypoints". In: *International Journal of Computer Vision* 60.2 (2004), pp. 91–110.

[19] A. L. Majdik, Y. Albers-Schoenberg, and D. Scaramuzza. "MAV urban localization from Google street view data". In: *In Proceedings of the International Conference on Intelligent Robotic Systems*. 2013, pp. 3979–3986.

[20] A. L. Majdik et al. "Air-ground Matching: Appearance-based GPS-denied Urban Localization of Micro Aerial Vehicles". In: *Journal of Field Robotics* (2015).

[21] A. Makhzani et al. "Adversarial autoencoders". In: *In Proceedings of the International Conference on Learning Representations* (2015).

[22] C McManus, B Upcroft, and P Newmann. "Scene signatures: Localised and point-less features for localisation". In: *In Proceedings of the Robotics: Science and Systems Conference*. 2014.

[23] I. Melekhov, J. Kannala, and E. Rahtu. "Relative Camera Pose Estimation Using Convolutional Neural Networks". In: *In Proceedings of the International Conference on Advanced Concepts for Intelligent Vision Systems*. 2017, pp. 675–687.

[24] A. Nassar et al. “A Deep CNN-Based Framework for Enhanced Aerial Imagery Registration With Applications to UAV Geolocalization”. In: *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018.

[25] G. Pascoe, W. Maddern, and P. Newman. “Robust Direct Visual Localisation using Normalised Information Distance”. In: *In Proceedings of the British Machine Vision Conference* (2015), pp. 70.1–70.13.

[26] G. Pascoe et al. “FARLAP: Fast robust localisation using appearance priors”. In: *In Proceedings of the IEEE International Conference on Robotics and Automation*. 2015, pp. 6366–6373.

[27] B. Patel, T. D. Barfoot, and A. P. Schoellig. “Visual localization with Google Earth images for robust global pose estimation of UAVs”. In: *In Proceedings of the International Conference on Robotics and Automation*. 2020.

[28] M Paton et al. “Bridging the appearance gap: Multi-experience localization for long-term visual teach and repeat”. In: *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2016, pp. 1918–1925.

[29] V Prasad and B Bhowmick. “SfMLearner++: Learning Monocular Depth & Ego-Motion using Meaningful Geometric Constraints”. In: *In Proceedings of the IEEE Winter Conference on Applications of Computer Vision*. 2018.

[30] K. Ridgeway et al. “Learning to generate images with perceptual similarity metrics”. In: *In Proceedings of the International Conference on Image Processing*. 2015.

[31] Shaojie Shen, Nathan Michael, and Vijay Kumar. "Tightly-coupled monocular visual-inertial fusion for autonomous flight of rotorcraft MAVs". In: *In Proceedings of the International Conference on Robotics and Automation*. 2015, pp. 5303–5310.

[32] A. D Stewart and P Newman. "LAPS - localisation using appearance of prior structure: 6-DoF monocular camera localisation using prior pointclouds". In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2012, pp. 2625–2632.

[33] M. Sundermeyer et al. "Implicit 3D orientationLearning for 6D Object Detection from RGB Images". In: *In Proceedings of the European Conference on Computer Vision*. 2018, pp. 699–715.

[34] P. Vincent et al. "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion". In: *Journal of Machine Learning Research* 11 (2010), pp. 3371–3408.

[35] P Viola and W.M. Wells III. "Alignment by Maximization of Mutual Information". In: *International Journal of Computer Vision* 25.2 (1997), pp. 137–154.

[36] M. Warren et al. "There's no place like home: Visual teach and repeat for emergency return of multirotor UAVs during GPS failure". In: *IEEE Robotics and Automation Letters* 4.1 (2019), pp. 161–168.

[37] S. Weiss et al. "Monocular Vision for Long-Term Micro Aerial Vehicle State Estimation: A Compendium". In: *Journal of Field Robotics* 30.5 (2013), pp. 803–891.

[38] R. W. Wolcott and R. M. Eustice. "Visual localization within LIDAR maps for automated urban driving". In: *In Proceedings of*

the IEEE/RSJ International Conference on Intelligent Robots and Systems. 2014, pp. 176–183.

[39] J. Zhang et al. "Local features and kernels for classification of texture and object categories: A comprehensive study". In: *International Journal of Computer Vision* 73.2 (2007), pp. 213–238.

[40] T Zhou et al. "Unsupervised Learning of Depth and Ego-Motion From Video". In: *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[41] T Zhou et al. "View Synthesis by Appearance Flow". In: *In Proceedings of the European Conference on Computer Vision*. 2016.