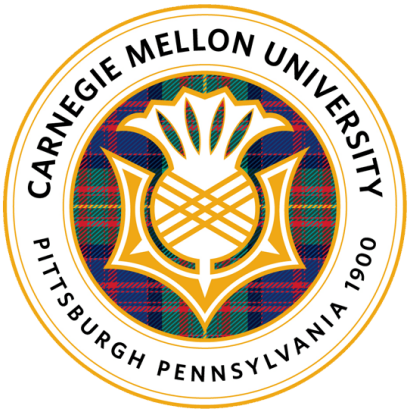
Generalizable Learning-based Registration

Hunter Goforth

CMU-RI-TR-19-38

May 2019



The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee
Prof. Simon Lucey, *Chair*
Prof. Michael Kaess
Leonid Keselman

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Abstract

This thesis explores the application of deep learning algorithms to 2D (image) and 3D (point cloud) registration scenarios, especially where the challenging nature of the data precludes the use of classical methods for establishing correspondence and alignment.

In the 2D case, we apply a recently-proposed learning-based image registration method to the problem of aligning outdoor imagery taken across seasons and times of day. Further, we extend the method to perform GPS-denied UAV geolocalization by aligning UAV images and satellite imagery. We also propose a novel joint optimization of motion estimates from visual odometry and geolocalization, to increase localization accuracy in cases where parts of the satellite map may be missing or too dissimilar from the UAV imagery.

In the 3D case, we develop three novel point cloud registration algorithms based on state-of-the-art point cloud processing deep networks. We show the benefits of the learned representation for registration on partial data, intra-category data (same category, different shape), and real-world data.

Acknowledgements

I want to thank my advisor Simon Lucey, for his advice and guidance which have helped me to become a researcher and contributor to the field of computer vision. I want to thank my other committee members, Michael Kaess and Leo Keselman, for reviewing my work and providing valuable feedback.

I want to thank some great co-contributors that I have worked with at CMU without whom much of this work could not have occurred, including Yasuhiro Aoki, Arun Srivatsan, Vinit Sarode, Xueqian Li, and Chaoyang Wang.

I want to thank my parents, my wonderful girlfriend, and rest of my family for their continued support of my education.

I want to thank all the great friends I have made at CMU, and I am sure we will see each other again!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Questions

There are many existing methods which address 2D image registration and 3D point cloud registration. These existing methods, and the problem statement of registration itself, are discussed in more detail in the Background Chapter. For a vast number of scenarios, an existing method will be able to provide reasonable accuracy and speed. Existing methods struggle to produce good results in cases where the appearance is quite different between the reference dataset (template) and the data to be aligned to the reference (source).

One example of such a registration scenario is the problem of aligning images of a location which were taken at different times of day, or across different seasons. This is a more difficult problem than aligning images of a location taken within a small temporal window, as the appearance has changed drastically. Especially for outdoor environments, appearance difference is manifested in lighting conditions, shadows, removal or occlusion of objects like buildings and cars, changes in foliage, and weather conditions.

In the point cloud case, appearance difference between template and source is manifested through noise affecting the measured position of points, partial measurement due to specific viewpoint of an object, or differences due to multi-modal measurement on different sensors for template and source.

The central hypothesis of this thesis is these appearance differences may be handled with a strong prior gained through learning on similar data. We argue that by designing algorithms which learn from data, and providing an appropriate training dataset which captures the distribution of these appearance differences, we can perform well at test time on this type of data. **To test this hypothesis, this thesis asks several questions such as, how should such learning algorithms (deep neural networks) be designed, how should the training datasets be generated, and what types of appearance difference are most challenging and currently out-of-reach of existing methods. We also ask, how does**

the registration speed and accuracy compare to existing methods.

1.2 Contributions

The specific contributions of this work include:

1. Adapting recent deep image registration techniques for use in aligning UAV imagery to satellite images, providing UAV geolocalization (Chapter 3).
   (a) Training networks on offline satellite imagery, and showing that these networks can generalize to challenging UAV datasets unseen during training.
   (b) Developing a joint optimization of motion parameters from geolocalized position estimates and visual odometry, which improves localization accuracy in cases where the map is too dissimilar from the UAV imagery.
2. Developing three novel point cloud registration algorithms based on a recent deep network design which processes point clouds directly (Chapter 4).
   (a) Showing favorable performance of developed methods as compared to common registration baselines.
   (b) Showing how the novel point cloud registration networks may be adapted for partial point cloud registration, and how performance of partial registration generalizes to real-world data.
   (c) Showing the potential for deep registration methods as a means of registering intra-category objects.

1.3 Overview of Chapters

Chapter 2 covers preliminary topics for 2D image and 3D point cloud registration, such as classical methods, introduction to relevant deep neural network architectures, and context within higher level problems like Structure-from-Motion (SfM) and Simultaneous Localization and Mapping (SLAM).

Chapter 3 covers methods, experiments, and results for deep learning registration in the 2D image case, particularly for aligning images of the same location taken across large temporal gaps.

Chapter 4 covers methods, experiments, and results for deep learning registration in the 3D point cloud case, particularly for data with various levels of noise and partiality.

Chapter 5 provides discussion about the connection between 2D and 3D case, conclusions, and recommendations for future work.

Chapter 2

Background

In this chapter, we present topics and related work which are preliminaries for the subsequent chapters.

First, we define registration as used in the computer vision and robotics community. **Registration refers to taking 2 sets of data (a template and a source), establishing the ground truth correspondence between the template and source, and from this correspondence finding a transformation in 2D or 3D space which precisely relates template to source.** In practice, establishing correspondence equivalent to ground truth can rarely be achieved due to the noisy process of measurement by which the template and/or source are acquired. The typical approach of a registration algorithm is to establish a reasonable cost function between template and source which, if minimized, is likely to lead to a solution near to the ground truth registration. It turns out that a great deal of the problems faced by the computer vision and robotics communities are problems of registration at the core. Fig. 2.1 shows example of the registration paradigm for both 2D and 3D cases.

In Section 2.1, we elaborate on common image registration scenarios, and describe classical solutions to these problems. In Section 2.2, we provide context on how the fundamental image registration problem fits within larger topics in computer vision and robotics, such as structure from motion (SfM) and simultaneous localization and mapping (SLAM). In Section 2.3, we present a brief history of and concepts involved with Convolutional Neural Networks (CNN), and provide understanding of how CNNs may be combined with image registration.

In Section 2.4, we present an analogous section for 3D point cloud registration. We present the fundamental problem at hand, how the problem fits within broader context of current problems in robots and computer vision, and classical solutions to the problem. In Section 2.5, we present a brief summary of rotational representations in 3D. Finally, we present in Section 2.6 current deep neural networks approaches to point cloud representations, and give intuition about how state-of-the-art deep representations for point clouds may allow efficient, accurate point cloud alignment.

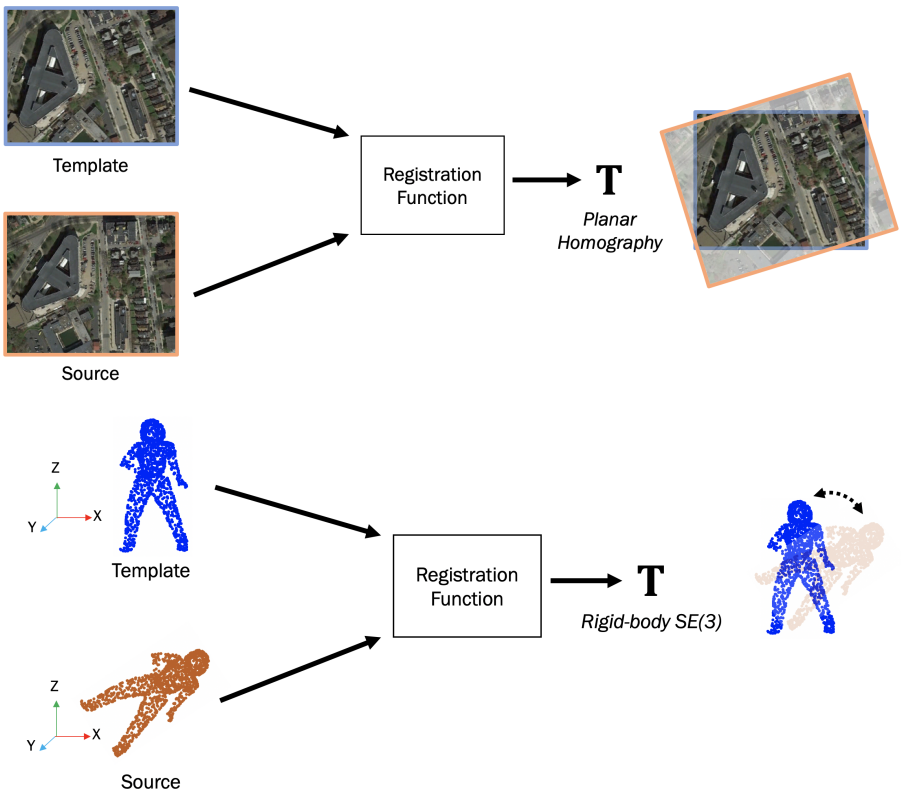


Figure 2.1: The set-up of typical registration problems in the case of 2D (image) and 3D (point cloud) data. In any registration problem, we are given template and source measurements for which we attempt to find the ground truth alignment through a registration function. For example in the 2D image case (top), the aligning transformation $\mathbf{T}$ may be parameterized as a planar homography (or other type of planar warping). In the point cloud case, the aligning transformation $\mathbf{T}$ may be a rigid-body transformation (part of the $SE(3)$ transformation group).

2.1 Image Registration

Consider panoramic image stitching [16, 81], a very well studied problem in the computer vision literature. Panoramic image stitching is the problem of a making a larger image by stitching together many smaller images. A common approach to panoramic image stitching is as follows.

To start, we can find interest points in the first image from the image sequence. Many methods have been developed in the literature for finding interest points, such as a difference-of-Gaussians function [37] or the Harris corner detector [39]. Interest points are typically those points in an image which have high gradient information, such as corners or other textured areas. After identifying these points, we also form a descriptor of each point, which is a vector unique to the orientation and appearance of the particular point. Popular existing methods for computing unique descriptors include the Histogram of Oriented Gradients method [22] and BRIEF [18].

Next, we look at the second image in the sequence. We repeat the process of finding interest points in the second image. We also compute descriptors for these points. Based on the assumption that the viewpoint of the second image affords some overlap with the first image, some of these interest points will correspond to the same location in the scene.

To determine which points are seen in both images, we can compare and compute distances between the descriptors computed for the points in both images. Descriptors which have lower distance and appear in both images are likely to be the same point in the scene. We can therefore form an initial list of the possible corresponding points in the two images. Existing pipelines which combine keypoint detection and description include SIFT [48], SURF [8], and ORB [72]. Now that we have the initial list of possible matching points, we can determine the transformation relating these two images. For a planar scene, this transformation from pixels in one image to pixels in the second image is represented by the planar homography matrix having 8 degrees of freedom ([40], Chap. 13). For a non-planar scene, we must use the more general essential or fundamental matrix (depending on knowledge of camera intrinsics), and the accompanying epipolar geometry relations ([40], Chap. 9). We may use RANSAC [29] to determine the value of either transformation matrix. We may repeat this process for each subsequent image, establishing the correspondence of feature points, and estimating the transform between images, which allows to create a panoramic view of all of the images. An example of panoramic image stitching using homography estimation with SIFT and RANSAC, is shown in Fig. 2.2.

Panoramic image stitching at the core is a problem of registration. We can see how the panoramic image stitching example fits in the overall paradigm of registration. First, the two datasets to be aligned are each image pair. The transformation to be estimated in the 8 parameter homography (or essential matrix). The cost function used for alignment is the distance between corresponding interest points after applying the transformation. Further, the panoramic stitching problem can be generalized into the Structure from Motion (SfM) problem. In the SfM problem, the data are again two images, and the cost function used for alignment is the distance

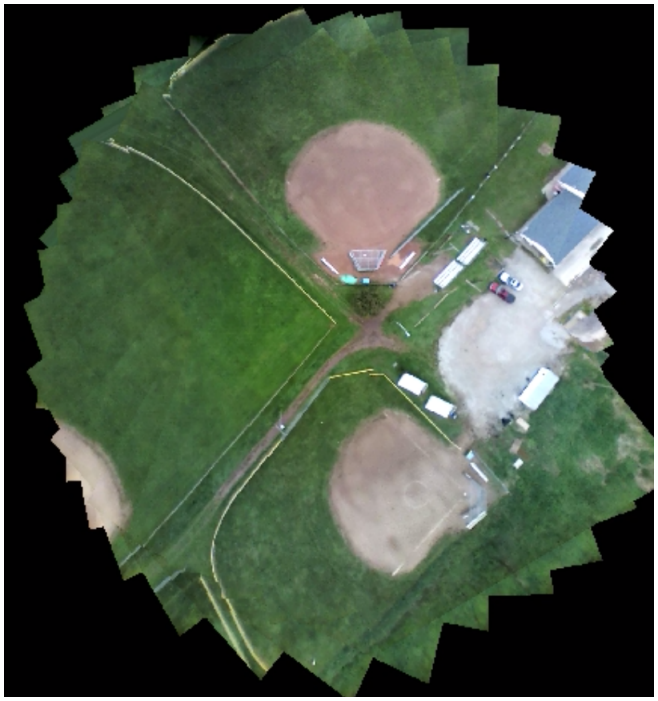


Figure 2.2: An example of image stitching using SIFT [48] and homography estimation with RANSAC. The video is captured from a spinning UAV above relatively flat terrain (baseball field), which allows use of the flat-ground assumption and thus planar homography.

between corresponding interest points across views. However, because the scene now has structure and is non-planar, we cannot use the homography anymore. The transformation must now be the 5 degree-of-freedom essential matrix.

Dense registration. The interest point methods described above are known as sparse alignment methods. This is because the interest points are sparsely scattered throughout the images, and these sparse points are matched across images. There are methods which are referred to as dense alignment methods, as opposed to these sparse methods.

One of the most well-known dense registration methods is the classical Lucas-Kanade (LK) alignment algorithm [50]. Instead of identifying interest points and computing descriptors, the LK alignment algorithm seeks to minimize the difference of all, or most, of the pixels in two images. The pixel-wise difference is also known as the photometric difference. The LK algorithm works by defining an optimization on the cost function of photometric difference, and parameterizing this optimization using the parameters of the appropriate transformation (the 8-parameter homography, in the case of planar scenes). The cost function can then be minimized using a Gauss-Newton optimization over the homography parameters. For an extensive and detailed review and derivation of the classical LK algorithm and more recent variants, the reader is referred to [7].

The dense LK registration method is different from the sparse interest point methods in terms of the cost function used for alignment. In the LK algorithm, the cost function is the photometric difference for all pixels, whereas in the sparse interest

point methods, the cost function is the distance between corresponding interest points after warping the image. However the data (images) and transformations (homography or essential matrix) remain the same. Instead of directly comparing pixels, it is also possible to extract a dense descriptor at each pixel, and use LK on these dense representations instead [3].

Section 3.2 of this thesis incorporates the LK registration method, showing how it has been combined in recent work with convolutional neural networks to have enhanced image registration capabilities.

2.2 Bundle Adjustment (BA)

Consider our panoramic image stitching example again. In the simple explanation in the previous section, we describe how two images may be aligned to each other. This process can be repeated pair-wise across the whole sequence of images. However, only estimating the pair-wise motion parameters is not guaranteed to lead to visually accurate panorama. This is because each pair-wise estimation introduces some slight error which deviates from the true motion of the camera. This slight error accumulates over each pairwise estimation, so the full panorama does not look visually accurate (and the motion parameters between each frame do not reflect the true camera motion).

To solve this problem, we must estimate motion using interest point correspondence information from more than just directly adjacent frames. This process is called bundle adjustment ([40], Chap. 18). In bundle adjustment, we consider interest points which persist across multiple frames, which are often also referred to as feature tracks. We use the same cost function as in the original pair-wise matching with interest points, which is the distance between corresponding feature points after applying the transformation. When finding this distance between two frames which are not directly adjacent, the distance is referred to as the reprojection error. The reprojection cost function for an image sequence may be minimized using a number of techniques such as Gradient descent, Newton's method, or Levenberg-Marquardt.

LK and BA. Analogously, we can think about bundle adjustment from an LK perspective rather than sparse interest points. In this perspective we would seek to minimize directly the photometric error instead of distance between corresponding interest points. Instead of discrete interest points locations, we can consider small patches of the image itself. These small patches are chosen through a similar method as the interest point detection, however the patch of pixels itself is used in bundle adjustment, instead of the interest point location in pixel coordinates. This approach is relatively recent and is referred to as photometric bundle adjustment [23, 2].

Section 3.3 of this thesis develops an approach which follows a similar pattern to interest point bundle adjustment and photometric bundle adjustment. In our work however, instead of using image patches, we are able to use all image pixels since the scene is planar in our case and all pixels are related by homography. We use this approach to jointly optimize motion estimates from UAV visual odometry and

comparisons to a satellite map.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) have seen great success in the computer vision in recent years. Although the concept is not new, CNNs have always required a great deal of data and processing power to learn effectively and generalize on difficult tasks. With the advent of this computing capability in modern GPUs, and the creation of datasets containing millions of examples, it has become attainable to train CNNs for image-based tasks such as object classification, localization, segmentation, and more.

Levels of training supervision. When training a CNN for a task, the user may be able to provide a precise ground truth answer for each training example. This ground truth answer is used to define the loss function for backpropagation. For example, when training a network to recognize what kind of animal a picture contains, the correct label ('dog' or 'cat') is provided to the network for each training example. This is called supervised learning.

In other tasks, we may be able to do weakly-supervised learning. This refers to giving the network only a partial knowledge of the ground truth during training, instead of the full answer. For instance, in weakly-supervised object localization, only the label of an image ('dog') is provided, but the network is asked to learn to predict a bounding box around the animal in the image. The way this is achieved, is that the network learns to recognize which features occur in images (like ears, tongue, eyes, tail) that are unique to the class label ('dog'). Since it has identified these features, it is also able to group these features together and put a bounding box around them. This specific example of weakly supervised learning is used for object localization [13], though there are other examples of weak supervision for tasks such as place recognition [5] and semantic segmentation [65].

Taking this one step further, we are also able to do completely unsupervised learning for some tasks. A notable example is monocular depth prediction. Predicting depth from a single camera image is an ill-defined problem since there are many scales and orientations of a scene which can produce the same image. However, as humans we typically have biases and assumptions which help us to understand the scale of scene and make accurate guesses of depth. It is therefore expected that a CNN could learn good assumptions which could make it proficient for single-image depth estimation as well. The problem can be formulated using supervised learning, but this requires many training examples with ground truth depth which may be difficult to obtain. A recent method [34] has shown that instead of collecting ground truth depth using a physical depth sensor, we can instead do unsupervised learning using left and right stereo camera images to learn depth prediction. The approach relies on the fact that, given a depth prediction at each pixel in the left image, the reprojection of the pixel into the right image using the estimated depth should correspond to a similarly colored pixel (same scene point). This is the stereo vision constraint which is inherent to the geometry of scenes and cameras, and allows us to take advantage

of an unsupervised approach to depth estimation.

In this thesis, the primary focus will be on supervised training. This is because for the applications presented here, it is feasible to gather the target ground truth for many training examples, either through gathering of real-world data or through generation of simulated data and ground truth transformations. This is also because it is difficult in most cases to formulate image registration or point cloud registration in a weakly or unsupervised learning framework.

Network structures. CNNs architectures have different structure (number and type of layers) based on the task. Image classification networks such as VGG [80] have one of the simpler types of structures. For image classification, the network input is an RGB image of some normalized sized such as (224,224,3). The image is then convolved with a set of filters (64 filters for example), with each filter having a kernel size and a stride with which it is applied. The result is a new data volume with spatial dimensions determined by the kernel size and stride, and number of channels equal to the number of filters in the filter set. This convolution of filter sets is repeated over many layers with a nonlinearity activation function between some of the convolution operations (typically reLU [59]). After all the convolutional layers, the resulting data volume is then passed through several fully-connected layers. A fully-connected layer can be thought of as a matrix of size (output dimension, input dimension). Fully-connected layers are also followed by reLU. The final fully-connected layer has an output dimension equal to the number of possible classes for classification (1000 for ImageNet [24]). For image classification with a single class per image, the softmax activation is applied at the output of the final fully-connected layer. The softmax activation function serves to map the network output into probabilities for each predicted class. After softmax, the most likely predicted class is that which has the highest value in the activation vector.

This thesis explores CNN architectures for image alignment. As will be further described in Section 3.2, there are convolutional layers in the image alignment CNN, but there are not fully-connected layers. Instead of fully-connected layers, there is a layer which implements Lucas-Kanade image alignment. This implementation of LK takes as input the data volume output from the final convolutional layer, as opposed to single-channel grayscale or 3-channel RGB images in traditional implementations of LK.

Transfer learning with pretrained features. The backbone of training a convolutional neural network (and most other kinds of networks) is the backpropagation algorithm [73] for updating the internal network parameters. Since the earliest era of neural network learning, backpropagation has been found to allow network to learn powerful internal feature representations and complex inter-relationships in data that are useful for a task. In recent years, further exploration has revealed that the features learned for one task, may also be quite transferable to a different task [78, 95].

Large convolutional networks for image classification, such as VGG [80], which have been trained on the ImageNet dataset, have been found to contain information in their learned convolutional filters which is helpful for other related tasks in vision.

The filters within these large networks can capture notions of low level features in images, like edges and corners, all the way up to high level image-specific features like a door or specific animal. Since lower level features like edges and corners occur across all natural images, we can speed up training for novel networks by initializing some or all of the weights with pretrained weights. In several sections of this thesis, transfer learning is utilized to speed up learning using deep neural networks, as opposed to randomly initialized network weights.

2.4 Point Cloud Registration

Analogous to 2D image registration, we can consider the problem of 3D point cloud registration. Point cloud are unstructured point sets in 3D with each point described by its 3D location. Common sensors which collect point cloud data include LiDAR sensors on autonomous vehicles, infrared distance sensors, laser range finders, and RGBD cameras. Point clouds inherently lack information about connectivity and orientation of surfaces, as opposed to other volumetric representations such as voxels or polygonal meshes. However point cloud data is very common due to the large number of sensors which collect this type of data, as well as the point clouds requiring low memory overhead for storage and processing.

Since point cloud data is abundant, point cloud registration is a very common task. Point cloud registration is the task of finding the transformation in 3D space which best aligns one point cloud with another. Analogous to the 2D image registration case, we can call one point cloud template and the other source. The transformation is often estimated as a rigid body transformation in $SE(3)$, meaning that the scene in the point cloud does not contain non-rigid motion or scale changes between template and source. The rigid transformation assumption is applicable especially to robotic mapping and localization, where it can be assumed that the environment is static in many cases.

A plethora of methods exist in the point cloud registration literature. One of the most popular and commonly used algorithms is Iterative Closest Point (ICP) [12]. The fundamental process of ICP consists of two steps:

1. For each point in the template, find its nearest neighbor in the source.
2. Estimate the $SE(3)$ transformation which minimizes the Root Mean Square Error (RMSE) for the distances found in Step 1.
3. Transform the source using the solution and repeat Steps 1 and 2 until some convergence threshold.

Due to its simplicity, ICP is widely used and there exists many variants which improve upon the fundamental process [74]. However, ICP is notably sensitive to the initial transformation in some cases and may be unable to recover from a local minima, and the correspondence step is costly.

Recalling the interest point methods image alignment such as SIFT, we can think of possible solutions to the local minima problem of ICP. If we could have local interest point detectors for 3D point clouds, it may be easier to escape local minima. Indeed, much literature focuses on interest point detection, computation, and matching for 3D point clouds, analogous to the 2D interest point detection methods in the image processing community. For instance, scale invariant curvature descriptors [31], oriented descriptors [33], extended Gaussian images [52], fast point feature histograms [75], color intensity-based descriptors [35], global point signatures [20], heat kernels [64], etc.

If computational efficiency were of no concern, another option for point cloud registration is obviously a brute-force search of the transformation space. We could even consider a nearly-brute force solution which is combined with distance metric for each query transformation i.e. we randomly transform the source and compute the RMSE of closest points to the template, keeping track of poses which produce low RMSE. Some approach in literature have moved in this direction, searching for the globally optimal registration despite the computational disadvantage. A representative example of this type of globally optimal registration is Go-ICP [91].

Point cloud registration problems are often characterized by different surface samplings between template and source, a template which is a full 3D model and source which is a partial observation, or Gaussian-like noise in template or source. Aligning partial or noisy point clouds is a difficult task for any registration algorithm. One goal, and something which will be further discussed in Chapter 4, would be to have an alignment algorithm which produces a smooth cost manifold with minimal cost for the ground truth transformation on difficult registration scenarios. It is thought that perhaps we can harness deep learning based methods to build such an algorithm.

2.5 Representations of 3D Rotation

3D point cloud registration involves the optimizing a cost function with respect to some parameterization of a rigid body transformation, and in particular, a parameterization of 3D rotation. The group of 3×3 orthonormal matrices representing 3D rotation is also referred to as $SO(3)$. There are several well-known parameterizations of 3D rotation, including Euler angles, quaternions, 3×3 rotation matrices, and the exponential map representation. For an excellent review of the benefits and drawbacks of each parameterization the reader is referred to [38]. Following is a brief summary of the major findings in [38], with the final conclusion that either quaternion or exponential map representation are the best choice for 3D point cloud registration.

Euler angles parameterize rotation using 3 angles, a minimal parameterization. The main shortcoming of Euler angles, which precludes their use in the optimization process of point cloud registration, is the presence of singularities within the Euclidean parameter space ("gimbal lock"). In short, this means that it is common for an Euler angle parameterization to enter a point in its parameterization space

where the dimensionality reduces from 3 to 2 i.e. of the parameters loses its effect on the physical orientation of the object. This singularity effect makes Euler angles ill-suited for registration.

Rotation matrices have the favorable property of being a group under multiplication. However, they are highly over-parameterized, representing orientation with 9 parameters. There exists 6 nonlinear constraints on rotation matrices, which must be manually imposed throughout the course of optimization to ensure valid rotations are obtained. This manual task is labor intensive as well as susceptible to numerical errors, rendering rotation matrices insufficient for the optimization required in point cloud registration.

Quaternions are a reasonable choice for optimization applications. They are almost minimally parameterized, having 4 parameters, do not have any singularity problems, and are generally numerically well-conditioned. There is one minor drawback, which is that there must be steps taken to re-normalize the 4-vector of all quaternions during optimization.

Finally, the exponential map is likely the best choice for point cloud registration. Given a vector in $\mathbb{R}^3$, the exponential map converts this vector to an $SO(3)$ rotation matrix based on both the direction and magnitude of the initial vector. The exponential map has the property of being minimally parameterized (3 parameters in an $\mathbb{R}^3$ vector). It also has the property that the singularities of the Euclidean space are far from the locations that are used in almost every scenario that would be encountered in point cloud registration. Finally, there is no computational overhead for re-normalization as was the case for quaternions and rotation matrices.

The exponential map representation may be extended to $SE(3)$ as well, including the translation component and expanding the vector parameter space to $\mathbb{R}^6$. The definition for mapping this $\mathbb{R}^6$ vector to the $SE(3)$ group of 4×4 parameterization will be seen in Section 4.1.3. The groups $SO(3)$ and $SE(3)$ are also known as Lie groups, and the vector spaces $\mathbb{R}^3$ and $\mathbb{R}^6$ from which they are mapped are known as Lie algebras. [25] provides an in depth review of these mappings, their properties, and their usefulness in robotics and computer vision.

2.6 PointNet

Since the recent explosion of deep learning methods which began in 2012 with AlexNet [46], it was inevitable that deep learning would be applied to point cloud data at some point. However, there was an inherent challenge in applying deep learning to point cloud data, which was not the case for image data. Image data is structured, since pixel information is naturally stored in blocks of $H \times W \times C$ and data which is adjacent within this structure is also adjacent in scene i.e. two adjacent pixels inherently capture information about two adjacent 3D locations. This spawned the idea of convolutional networks which process the pixel data using local sliding windows (convolutional filters). This approach is valid since adjacent pixel information tends to be very tightly correlated in local neighborhoods. However,

the same is not true for unstructured point cloud data. A point cloud, stored in an $N \times 3$ block, is not guaranteed to have any ideas of ordering based on distance or other metrics. The other problem is that N can be arbitrary, and in the case of point cloud registration, N for the template and source may be different numbers.

Deep networks typically have taken the form of either convolutional or fully-connected (also known as Multi-Layer Perceptron (MLP)) layers. However, it would be naïve to process the $N \times 3$ block using a sliding-window convolutional operator, since there is no concept of order. It is infeasible to use a fully-connected approach because a fully-connected network requires a fixed number of input nodes and weights, which could not be fulfilled for the point cloud input unless some form of fixed-number subsampling is used. However, arbitrary sampling would be very limiting and it would be preferable to utilize all given points.

The tactic used by PointNet [68] is to instead learn a single MLP through which all points in the cloud are passed. For a single 3D point which is size 3×1, we perform the MLP operation with several matrix multiplications, with nonlinearity (reLU) between each multiplication. The first matrix has size 64×3, transforming the 3D point into a $64 - D$ embedding space, followed by reLU. This continues with several layers up to a $1024 - D$ embedding space. Processing all the points in the point cloud with the same MLP, we have an $N \times 1024$ matrix of points transformed to the embedded space. Next we apply a pooling (such as average or maximum) along the N axis to get a single, fixed-length vector of size 1024. This fixed-length vector can be thought of as a latent, global descriptor of the entire point cloud. In the case of point cloud classification, we would want this descriptor to occupy separate parts in the embedded space for each different class of object (e.g. car, table, chair, person). It is possible to plot these descriptors using a high-dimensional visualization tool such as t-SNE [51], to see how the latent vector is discriminatory for the object class.

As an aside, the notion of pooling across a variable dimension of the input data is not novel in the point cloud learning case. In the Natural Language Processing (NLP) community for instance, variable sentence length has dictated the need for a pooling operation along the dimension of sentence length [45].

Of course, these descriptors must be learned from labeled data. In the original PointNet paper, during training of the network, the $1024 - D$ descriptors are passed through another MLP which outputs a vector of probabilities for each class (e.g. car, table, chair, person). This network may be trained with a standard single-label classification loss such as soft-max cross entropy. PointNet achieves state-of-the-art classification accuracy with this architecture. They also find that the architecture can be extended for semantic segmentation of point cloud, for which they achieve state-of-the-art performance as well.

The success of PointNet has motivated many further extensions in applications. The authors extend the architecture in a second work to find and assign descriptors to local parts of the point cloud, rather than a global descriptor vector as in the original work [69]. They also apply the PointNet method for 3D object detection in [67], combining the PointNet semantic segmentation capabilities with existing 2D image

object detection to find 3D bounding boxes in RGB-D data. Other work has used PointNet to complete partial point cloud data, and specifically partial KITTI car point clouds, to form full point clouds from only partial input [97].

PointNet has seen success with object classification, semantic segmentation, object detection, generative point completion, and even place recognition [4]. However, the application of PointNet to point cloud registration has not been well-explored. In the original PointNet work, the MLP network also includes feature transformation (T-net) which are learned transformations of the points in the embedded space. The authors postulate that these learned transformations serve to re-orient the points as they travel through the network, orienting them to some concept of canonical orientation which may enhance classification accuracy. Indeed, they show through ablation studies that the inclusion of the T-net gives minor improvement. Though it is not well understood exactly the type of transformation estimated in these subnetworks, or if they are indeed orienting objects according to some canonical orientation.

Expanding on the T-net idea, another work [96] is more explicit in estimating a transformation of the point cloud which aids in classification and segmentation. IT-Net, as it is called, uses an auxiliary PointNet followed by fully connected layers which estimates a rigid body $SE(3)$ transformation which is applied to the point cloud. The transformed points are then passed again through the same network, estimating a new $SE(3)$ transformation, and iteratively continuing until a convergence threshold is reached. This works also directly compares against the use of the original T-net, showing how explicitly estimating an $SE(3)$ transformation gives better performance than transformations in the embedding space.

While IT-Net applies $SE(3)$ transformations to point clouds iteratively, there is only a single point cloud input, as opposed to a full registration pipeline where two point clouds (template and source) are input to a network and their relative pose difference is estimated. IT-Net can be thought of as a registration module which takes only a source point cloud and has no need for a template because all source cloud inputs will be aligned to a single canonical "template" orientation; therefore no actual template cloud is required for estimating the transformation. However, this approach has the obvious drawback that it cannot estimate transformations between a template and source which have arbitrary positions within the coordinate system.

Missing so far from literature is a method using PointNet which takes as input both source and template point cloud, and achieves registration in this more general setting. However, an important distinction is that there does indeed exist methods for point cloud registration which utilize deep learning, though there are not many. In [27], a deep auto-encoder is used to compute descriptors on 3D interest points, followed by descriptor matching. In [93], a network is used to perform both interest point detection and descriptor computation. However, since PointNet has allowed many state-of-the-art advances in the point cloud domain, and it is inherently efficient and lightweight, it is natural to ask how this architecture may be applied to the classical registration problem. Chapter 4 describes several methods that we have explored for adapting PointNet for the task of point cloud registration.

Chapter 3

Aligning Images Across Large Temporal Gaps

This chapter explores the use of deep network representations in matching and aligning imagery which is captured across large temporal gaps, such as across time-of-day or seasons. In this challenging registration scenario, we hypothesize that deep representations based on learned priors can enhance the state-of-the-art performance.

Unmanned Aerial Vehicle (UAV) geolocalization is a specific application which is enabled by the ability to align imagery across temporal gaps. In this application, we use pre-existing satellite imagery as a map to which UAV imagery is compared and aligned, thus localizing the vehicle. Section 3.3 will further discuss this application.

To develop networks which are capable of the UAV geolocalization task, we begin with some preliminary experiments using deep features for matching and alignment between satellite images of a location.

In particular, Section 3.1 presents preliminary experiments where deep features are extracted from satellite imagery (using a network not trained on any satellite images) and these features are matched across images using a correlation metric. This experiment motivates further sections by providing an initial intuition for where deep features may provide good matching of salient image features.

In Section 3.2, alignment of the extracted deep features from satellite imagery is added, via an additional Lucas-Kanade layer in the deep network. We train this network on a large dataset of satellite imagery, and show impressive alignment performance at test-time. We also present experiments where the network is tested instead on imagery taken from outdoor webcams, showing promising generalizability of the features for alignment in related tasks.

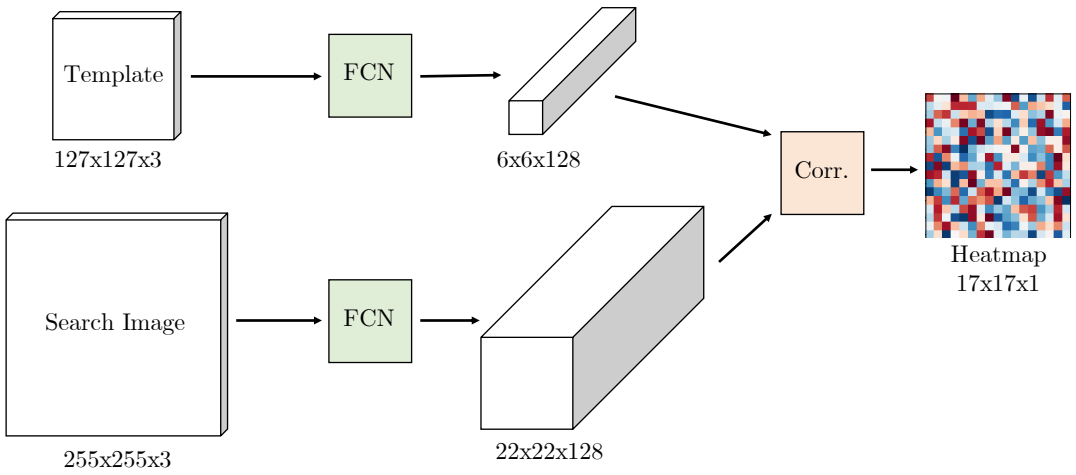


Figure 3.1: Fully-convolutional cross-correlation matching pipeline from [11]. The pipeline is originally used for object tracking, where the template is an image patch containing the object in a previous frame, and the search image is the entire current frame. To find the template inside the search image, both are passed through Fully-Convolutional Networks (FCN) with shared weights. The resulting data volumes are cross-correlated, leading to a heatmap for the template locations in the search image.

3.1 Deep Cross-Correlation Matching

To initially explore the use of deep convolutional features for the satellite image matching task, we adopt the method from [11], where Fully-Convolutional Networks (FCN) are used for object tracking. The method is explained in Fig. 3.1.

The cross-correlation method can be adapted from an object-tracking application to the satellite image matching task as follows. Instead of the template and source being taken from sequential frames in a object-centric video, the template and source may be extracted from satellite images taken at different time stamps. An example of this type satellite data is shown in Fig. 3.2. Satellite data is gathered from the United States Geographical Survey Earth Explorer.

We perform a cross-correlation experiment on satellite image data using convolutional weights from a VGG-16 network [80] which is pre-trained on ImageNet. We compare the use of these weights in transfer learning, compared to the pre-trained weights using the original tracking work [11]. We also compare against a simple baseline, which is the cross-correlation of normalized pixel patches (no FCN used). The results of this cross correlation experiment are shown in Table 3.1.

In summary, we find that using pre-trained VGG16 conv3 features for the FCN produces good results for satellite image matching on the representative satellite dataset from Fig. 3.2. This is despite using pre-trained weights, which are not trained on satellite image matching. This illustrates the transfer capability of deep features for other unrelated tasks. With this knowledge, we use VGG16 conv3 features in many subsequent experiments in later chapters.

Figure 3.2: Example satellite data used in the cross-correlation experiment. Both satellite images are captured of the same area, but taken several years apart. A 125×125 patch from the left image (template) is extracted and a search is conducted for the same patch in the right image. This image pair contains many representative changes seen over temporal gaps, which are highlighted in Fig. 3.3.

Figure 3.3: Examples of differences seen in satellite images captured across large temporal gaps, such as different angles of buildings due to satellite viewpoint, seasonal changes to foliage, shadow angles, and construction, among others. These photometric and environment differences create difficult conditions for matching or registration.

Method	**Template Size**	**Acc.** (%)	**Template Corr. Volume**	**Source Corr. Volume**
Normalized pixel x-corr.	125×125	3	$125 \times 125 \times 3$	$1739 \times 1760 \times 3$
Pre-trained FCN from [11]	125×125	3	$32 \times 32 \times 128$	$435 \times 440 \times 128$
	250×250	12	$64 \times 64 \times 128$	$435 \times 440 \times 128$
	350×350	23	$128 \times 128 \times 128$	$435 \times 440 \times 128$
VGG16 conv1	125×125	10	$63 \times 63 \times 64$	$870 \times 880 \times 64$
VGG16 conv2		25	$32 \times 32 \times 128$	$435 \times 440 \times 128$
VGG16 conv3		**90**	$16 \times 16 \times 256$	$218 \times 220 \times 512$
VGG16 conv4		32	$8 \times 8 \times 512$	$109 \times 110 \times 512$
VGG16 conv5		5	$4 \times 4 \times 512$	$54 \times 55 \times 512$

Table 3.1: Results for cross-correlation matching experiment on satellite data. We attempt to localize a template patch from one satellite image within a separate search image, see Fig. 3.2 for the data which is used. We compute **accuracy** as the percent of patches which are localized (highest value in cross-correlation heatmap) within 20 pixels of the ground truth location in the search image. We find that by far the best performing method is using pre-trained VGG16 conv3 weights for the FCN, at 90% localization accuracy. Accuracy is computed for all possible 125×125 template patches from the template image. This reinforces the common belief that mid-layer features from pre-trained networks are often the most transferable for other tasks, which has been found in prior literature [95]. With this insight, we continue using VGG16 conv3 features in subsequent experiments.

Lastly, we visualize which areas of the imagery are most easily matched between two satellite images. We would expect that salient, persistent features such as large buildings would be most easily matched between images, while less textured parts such as forests or fields may not be easily matched. We visualize this using the heatmap, computed using VGG16 conv3 features, shown in Fig. 3.4.

We reiterate that this section has not trained any deep features on satellite imagery; we have used only weights pre-trained on other tasks, as a preliminary experiment. This experiment, while still a type of registration, is a global search technique as opposed to a frame-by-frame image registration such as visual odometry with sparse interest points. The transformation for registration in this cross-correlation experiment is simply a translation in x, y; scale, rotation, or projectivity are not estimated, as in homography estimation.

In subsequent sections, we will consider more complex registration scenarios, where scale, rotation, and projectivity will be estimated through planar homography. Instead of a global search, we will assume the approximate position of the template is known in the search image, but this position estimate needs to be refined. We will see that this experimental set-up is exactly what is needed for GPS-denied UAV flight. In GPS-denied flight, a rough position of the vehicle may be estimated used the last known GPS location and on-board sensors. The vehicle location can be further refined through visual comparison with a satellite map.

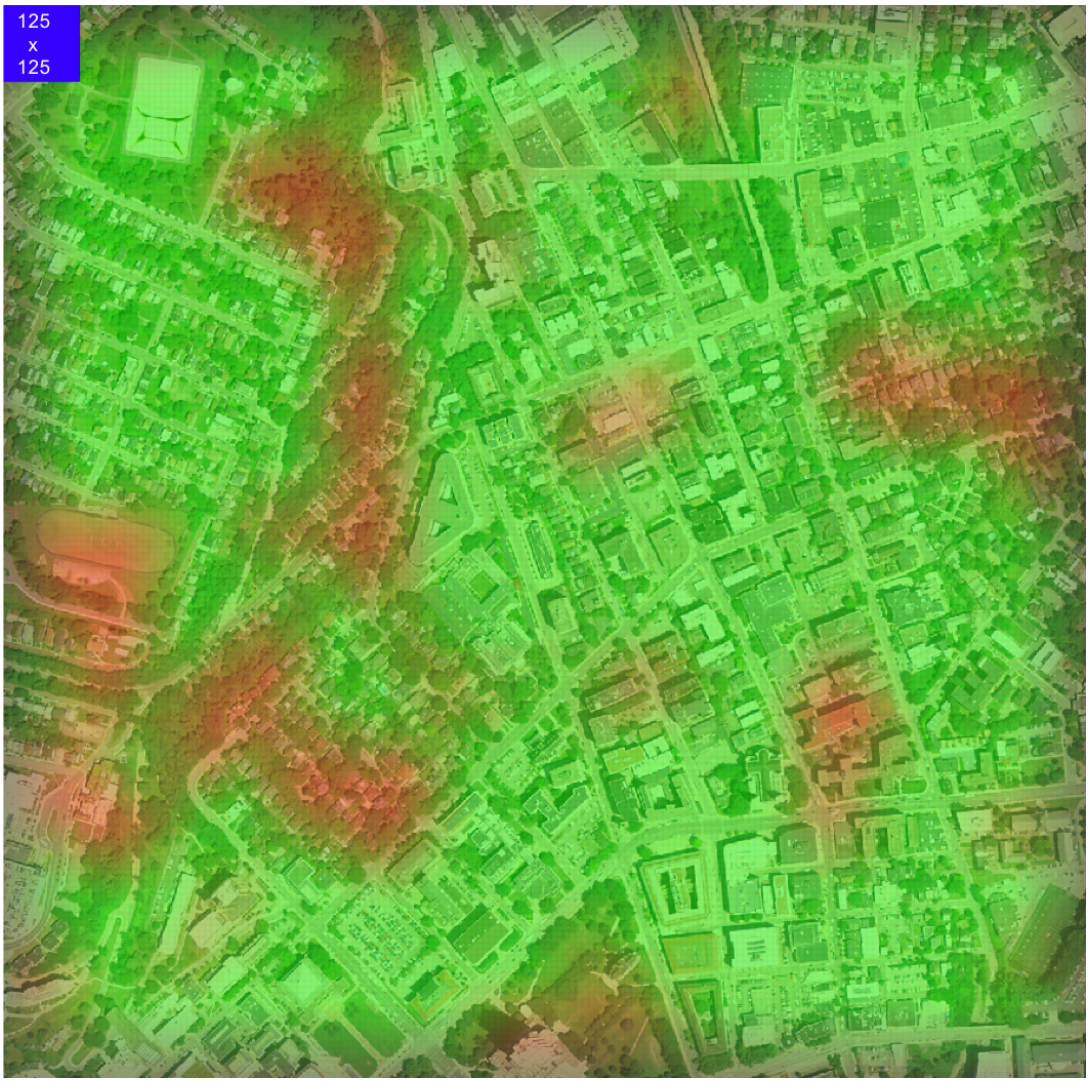


Figure 3.4: Results for cross-correlation experiments in Section 3.1. We visualize the heatmap for localization accuracy, showing which areas of the image are most easily matched vs. not easily matched. The green areas show where matching is easier, due to salient and persistent features such as buildings and intersections. The red areas highlight low-texture, repetitive areas such as forested areas and fields. Red areas which overlap with buildings or man-made structures are most commonly due to construction, or near tall buildings which have large angle variation due to viewpoint difference.

3.2 DeepLK for Aligning Satellite Imagery

Building on Section 3.1, we will look at the case where the template position is approximately known in the search image. This is the scenario in the GPS-denied UAV localization application, where the UAV location is approximately known, but we would like to refine this position through comparison with a satellite map.

In the case where the template position in the search image is approximately known, and fine registration is required, we must estimate all motion parameters for image registration. These parameters include translation, scale, rotation, and projectivity. When estimating all possible planar motion parameters, we use the homography transformation. To estimate this transformation, we adopt the Lucas-Kanade algorithm.

In the following sections, we discuss prior work related to satellite image registration (Section 3.2.1), we motivate the use of the Lucas-Kanade algorithm as opposed to sparse feature matching (Section 3.2.2), describe how to combine deep features and Lucas-Kanade (Section 3.2), and finally describe registration experiments using this architecture (Section 3.2.4).

3.2.1 Prior Work in Satellite Image Registration

Satellite image registration is a common task within the remote sensing community. Early methods for this task are often based around the matching of simplistic representations of landmark outlines (buildings, rivers, streets) [55] or early feature point methods [83]. The nature of registration problems in remote sensing is often alignment of entire satellite mosaics covering up to several kilometers. Thus, these images typically contain ample texture, and sparse feature-based methods are adequate. For instance, Yang *et al.* [92] present an automatic satellite image registration algorithm which utilizes SIFT distance, a Shape Context feature [9], and Euclidean distances of pixel intensity. This methodology makes their approach reliant on ample texture in imagery. Other representative works in remote sensing include [10, 36], which employ similar methods.

A closely related work is Verdie *et al.* [85], which introduces a Temporally Invariant Learned DEtector (TILDE) for registering imagery from outdoor webcams which captured images across time of day and seasons. The method is based on learned interest point detection and matching, which thus makes it still reliant on persistent, salient texture across images. This is satisfied in the outdoor webcam dataset, which has ground-level imagery of buildings, streets, intersections, and other texture environments. We present results on the same outdoor webcam dataset which is used in this work, in Section 3.2.4, showing that our method can adapt for both low and high texture environments.

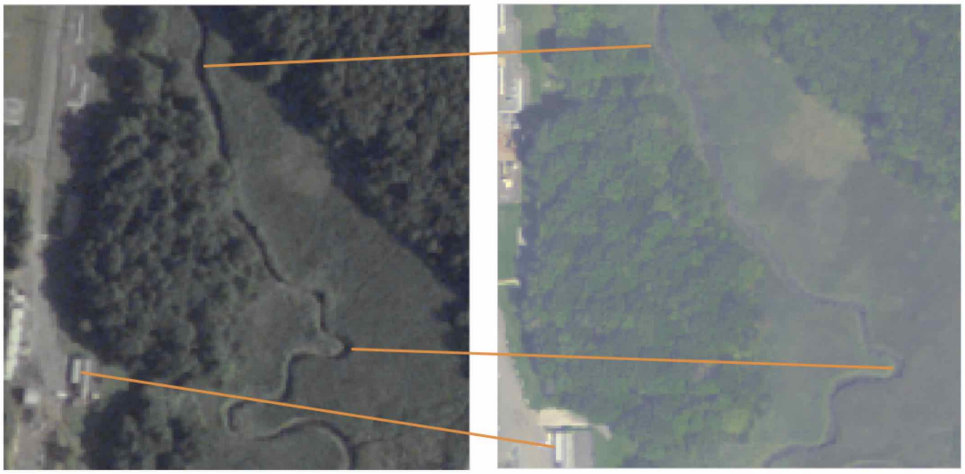


Figure 3.5: An example of low texture in satellite imagery, where the orange markers indicate a few ground truth correspondences. Sparse feature-based methods such as SIFT [48] fail on this example due to a lack of persistent, salient features. A dense registration algorithm such as Lucas-Kanade [7] is required, which considers all or most of the pixels in the image during registration. We show that deep features, when combined with LK, can overcome both the photometric differences and the low texturedness in examples like this one.

3.2.2 LK for Low-Texture Environments

As discussed in the Background chapter, image registration is addressed using methods which broadly fall into one of two categories. The first category covers sparse feature-based registration methods, examples of which include SIFT [48], SURF [8], or ORB [72]. The second category covers dense, direct (pixel-based) registration methods, exemplified by Lucas-Kanade [7].

The main advantage of sparse feature-based registration methods is that they function particularly well in scenes with high levels of unique texture, where feature points can be extracted. In this case, it possible to achieve registration using perhaps a couple dozen feature correspondences. However, when the texture in the scene is lower, a dense registration approach is more appropriate. In cases of low texture, it becomes necessary to consider most or all of the pixel information in each image to do registration, rather than a few sparse correspondences. Some methods are both dense and feature-based, such as dense SIFT or bitplanes [1, 3].

In the satellite image registration problem, we find that the data often has low texture. An example of low texture data is shown in Fig. 3.5. This example helps motivate the use of a dense registration algorithm such as Lucas-Kanade for the satellite image registration task. In the example, sparse feature-based methods fail during the correspondence step, having detected too few interest points to perform successful registration. However, we are able to use a combination of deep features and the Lucas-Kanade method to register the images in Fig. 3.5.

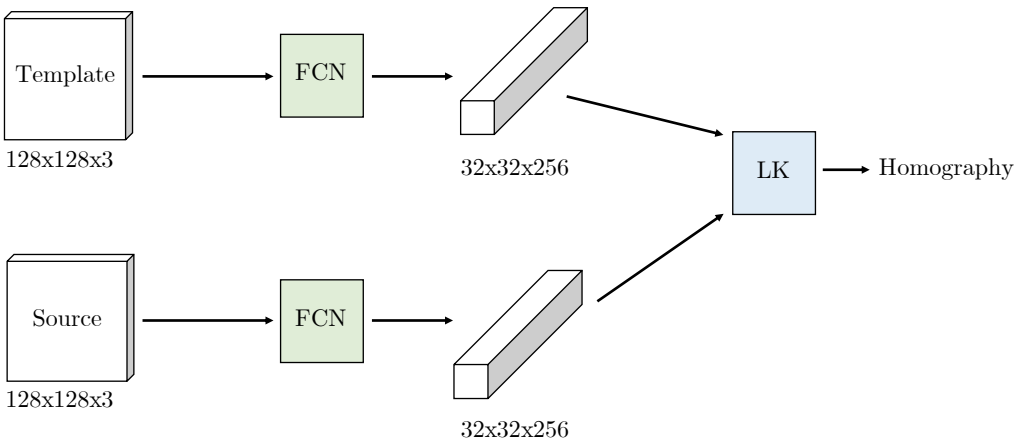


Figure 3.6: The basic DeepLK architecture. Sizes of image volumes are for reference only, spatial dimensions are arbitrary. A template and source image are provided, and passed through an FCN which reduces their spatial dimension and creates many feature channels. These feature volumes are passed through a differentiable implementation of LK, which estimates the planar homography for alignment of source to template.

3.2.3 DeepLK Architecture

Combining deep features and Lucas-Kanade alignment was first shown in two recent, concurrent works, by Wang *et al.* [86] and Chang *et al.* [19]. We will reference these approaches generally as DeepLK. In DeepLK, the template and source image are first passed through a Fully Convolutional Network (FCN) as in the cross-correlation experiment. Next, instead of cross-correlation, the resulting feature volumes are passed into a differentiable, multi-channel implementation of LK which estimates the 8-DoF homography transformation between template and source. The architecture allows for FCN weights to be learned which enhance alignment performance in the LK layer. A diagram of the basic DeepLK architecture is shown in Fig. 3.6.

Inverse Compositional Lucas-Kanade (ICLK). We review the formulation of ICLK, and the parameterization of the warp function (homography). In ICLK, the role of the template and image are reversed from the original LK formulation. This reversing allows for much greater computational efficiency, which will be shown in the following derivation. The formulation of the ICLK seeks the minimization of squared photometric difference between a template T and source image I:

$$\sum_x \|T(\mathbf{W}(\mathbf{x};\Delta\mathbf{p})) - I(\mathbf{W}(\mathbf{x};\mathbf{p}))\|_2 \tag{3.1}$$

Here we use the notation $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$ to mean sampling an image I at image coordinates $\mathbf{x}$ that have been warped with warping function $\mathbf{W}$, using a projective transformation (homography) parameterized by $\mathbf{p} \in \mathbb{R}^8$. The warp function $\mathbf{W}$ for homography, parameterized by $\mathbf{p} = (p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8)$ and for a pixel

located at $\mathbf{x} = (x, y)$, is defined as:

$$\mathbf{W}(\mathbf{x}; \mathbf{p}) = \frac{1}{1 + p_7 x + p_8 y} \begin{pmatrix} (1 + p_1)x + p_3 y + p5 \\ p_2 x + (1 + p_4)y + p6 \end{pmatrix} \tag{3.2}$$

Note that the template and source image can have an arbitrary number of channels; in our case, we will use feature maps of template and source image with up to 256 feature channels that have been output from fully-convolutional layers.

In each iteration of ICLK, an incremental warp parameter update is computed which we denote as $\Delta\mathbf{p}$. To solve equation (3.1), the ICLK approach performs a first-order Taylor expansion and solves the resultant least-squares form. The resulting expression for the warp update can be written:

$$\Delta\mathbf{p} = \mathcal{H}^{-1} \sum_{\mathbf{x}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} [I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})] \tag{3.3}$$

Where ∇T is the gradient of the template image and $\mathcal{H}$ is the Hessian matrix:

$$\mathcal{H} = \sum_{\mathbf{x}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right]^{\mathrm{T}} \left[\nabla T \frac{\partial \mathbf{W}}{\partial \mathbf{p}} \right] \tag{3.4}$$

To compute $\Delta\mathbf{p}$ using Equation 3.3, we must compute the warp Jacobian $\frac{\partial \mathbf{W}}{\partial \mathbf{p}}$ at $(\mathbf{x}; 0)$. For homography, the warp Jacobian can be written as:

$$\frac{\partial \mathbf{W}}{\partial \mathbf{p}}(\mathbf{x}; 0) = \begin{pmatrix} x & y & 1 & 0 & 0 & 0 & -x^2 & -xy \\ 0 & 0 & 0 & x & y & 1 & -xy & -y^2 \end{pmatrix} \tag{3.5}$$

Once the solution for $\Delta\mathbf{p}$ has been obtained via Equation 3.3, it can be converted to a homography matrix, and applied via inverse composition to the current warp parameters. Specifically, if $\mathbf{H}_\Delta$ is the homography with parameters $\Delta\mathbf{p}$, and $\mathbf{H}_\mathbf{p}$ is the homography with parameters $\mathbf{p}$, then the updated parameters can be extracted from a new homography calculated by:

$$\mathbf{H}_\mathbf{p} = \mathbf{H}_\mathbf{p} \mathbf{H}_{\Delta\mathbf{p}}^{-1} \tag{3.6}$$

For further details on how to develop ICLK from the original LK, the reader is encouraged to reference [7].

Loss function. A question arises of the correct loss function to use for the task of estimating homography parameters. The loss function $\mathcal{L}(\mathbf{p}, \mathbf{p}_{GT})$ compares the estimated warp parameters $\mathbf{p}$ and ground truth warp parameters $\mathbf{p}_{GT}$, outputting a single value which is used for backpropagation. In [86], the authors define the Conditional Loss, which is a robust Huber loss [43] computed on the difference

between the ground truth warp parameters and the estimated parameters. The problem with this approach is that each of the 8 parameters within $\mathbf{p}$ do not equally affect the magnitude of the geometric warp. For instance, the projective parameters p_7 and p_8 carry much more weight in terms of the visual effect of the warp, than do the translation terms p_3 and p_6. Therefore, a loss function is needed which captures the visual correctness of the regressed parameters $\mathbf{p}$.

We use the Corner Loss proposed in [19], which is a better measure of visual correctness than the Conditional Loss. The Corner Loss computes the squared distance between the four corners of a ground truth un-warped image $I(\mathbf{W}(\mathbf{x}; \mathbf{p}_{GT}))$ and the prediction of the un-warped image $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$. Defining the 4 corners of the warped image I as $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$, $\mathbf{c}_4$, we have the Corner Loss defined as:

$$\mathcal{L}(\mathbf{p}_{GT}, \mathbf{p}) = \sum_{i=1}^{4} \|\mathbf{W}(\mathbf{c}_i; \mathbf{p}) - \mathbf{W}(\mathbf{c}_i; \mathbf{p}_{GT})\|_2^2 \tag{3.7}$$

Stopping criteria. ICLK solves a non-linear least squares problem by first-order Taylor expansion and successive iterations. For each unique image and template pair, there is a variable number of iterations until convergence of the algorithm. The criterion for convergence is often a heuristic threshold on the magnitude of the change in warp parameters, $\Delta\mathbf{p}$. This is the chosen method of convergence used in [19]. In [86] however, the authors use the magnitude of the average error residual at each iteration as the stopping criterion. The error residual is calculated as $I(\mathbf{W}(\mathbf{x}; \mathbf{p})) - T(\mathbf{x})$ in Equation 3.3. We theorize that the choice of stopping criterion is highly dependent on the problem domain and the type of imagery. We experimented with using both the average residual method, and the magnitude of $\Delta\mathbf{p}$ threshold method. We found that using a heuristic threshold on the magnitude of warp parameters provides the best trade-off between accuracy of the final alignment and number of iterations.

Dynamic iterations during training. In both prior works [86] and [19], the authors only allow the ICLK layer of the network to iterate a single time during the training stage, although during testing the ICLK is able to iterate to convergence. This approach is taken due to limitations of the implementation framework (MatConvNet [84], Torch [21]), which make it very difficult to perform back-propagation through multiple ICLK iterations during training. Therefore, the authors design loss functions which suit the single-iteration training regime, and augment their dataset to mimic the middle output of LK iterations. However, we use the more optimal strategy of iterating a dynamic number of times in the ICLK layer during training, as this is readily implemented using a more recent framework (PyTorch [66]). Iterating dynamically during training also allows us to utilize the simple formulation of Corner Loss in Equation 3.7 as the loss function of our network output.

We show that unfolding dynamic LK iterations during training gives a dramatic performance improvement over single LK iteration during training, for our task at test time. Results and further information can be found in Fig. 3.11, where we compare single-iteration Corner Loss minimization and dynamic-iteration Corner Loss minimization.

3.2.4 Experiments

For training and testing, we require datasets containing a large amount of aligned images of a variety of outdoor environments, across many different periods of time which showcase seasonal changes and other temporal differences. One source of such data is aligned orthographic satellite imagery, which can be found from at the United States Geographical Survey Earth Explorer. Another source of data is the Archive of Many Outdoor Scenes (AMOS) dataset [44], which is also used by the TILDE work [85]. AMOS provides data from webcams positioned at many outdoor scenes across the world, effectively generating large amounts of time-lapse data of outdoor scenes. In the following sections, we describe experiments and results on these two datasets.

We compare a DeepLK alignment strategy against three baselines: two other dense alignment strategies, and one sparse key-point strategy. These strategies are:

1. **ICLK**: ICLK algorithm on raw pixels, without applying any dense descriptors. We normalized the pixels to have zero-mean and unit variance per-channel.
2. **ICLK on untrained VGG16 conv3 features**: ICLK algorithm on feature maps extracted from the untrained VGG16 conv3, similar to Deep Cross-Correlation experiments.
3. **SIFT+RANSAC**: We extract sparse SIFT keypoints and perform RANSAC to estimate the homography for alignment. We use the implementation included in OpenCV [14]. It should be noted that in [19], the authors find that SIFT+RANSAC provides the second best alignment method, behind only their implementation.

We employ a Corner Error metric for showing performance of our algorithm, which is similar to that of [19]. The Corner Error is related to the Corner Loss, except that it reports the average Euclidean distance between the four corners of the ground truth un-warped image $I(\mathbf{W}(\mathbf{x};\mathbf{p}_{GT}))$ and the prediction of the un-warped image $I(\mathbf{W}(\mathbf{x};\mathbf{p}))$. It is measured in pixels, and is equal to:

$$\mathcal{E}(\mathbf{p}_{GT},\mathbf{p}) = \frac{1}{4}\sum_{i=1}^{4} \|\mathbf{W}(\mathbf{c}_i;\mathbf{p}) - \mathbf{W}(\mathbf{c}_i;\mathbf{p}_{GT})\|_2 \tag{3.8}$$

Since we test with variable sizes of square image patches, we instead report the Corner Error as a percentage of the image width so that we can compare warps for different image patch sizes. We provide a visualization of Corner Error in Fig. 3.7.

Satellite imagery dataset. For training and testing, we use images from a suburban area of New Jersey, USA. We chose the location for its abundance of data, and its even mix between high and low texture. Some example images from this dataset are shown in Figure 3.8. We obtain aligned images taken during spring,

Figure 3.7: A visualization of Corner Error. The far left picture is ground truth. The rest of the pictures illustrate warps which result in a Corner Error of 10% of the image width from the ground truth.

Figure 3.8: Some representative patch examples from our *New Jersey* satellite image dataset. The examples illustrate challenges of the temporally-invariant alignment problem; namely, varying levels of texture, image quality, stylistic changes, degradation in natural environments, the addition or subtraction of structures such as buildings, and shadows. We find that the trained DeepLK is the only method of the ones we experimented with which is able to align all of the above examples with reasonable accuracy.

summer, and fall, taken in 2006, 2008, 2010, 2013, 2015, and 2017 (10 images total). The images are each 7582×5946 pixels, at a resolution of 1 meter per pixel, meaning we train on about 50 sq. km. of geographical area. We withhold 20% of the geographical area (across all 10 images) of this dataset for testing.

We dynamically create data pairs from the satellite imagery data during training and testing. That is, during the training loop, we randomly choose two of the 7582×5946 images from the New Jersey dataset, and randomly choose a location in the image and a patch size to sample. Keeping one of the patches static (the template T), we apply a random warping to the other patch (the source image I). The parameters for random patch size, and the random warp parameters, can be found in the Implementation Details section.

We test on 5000 data pairs from the *New Jersey* satellite image dataset, from geographical areas which are unseen during training. The results of this experiment are shown in Fig. 3.9. We report the results in terms of Corner Error as a percent of image width, versus the ratio of training data. The results indicate the superior performance of DeepLK with dynamic training iterations for aligning satellite imagery, in the face of large temporal and seasonal variations. Please see the description in Fig. 3.9 for more information on the performance metrics. In Fig. 3.10, we provide

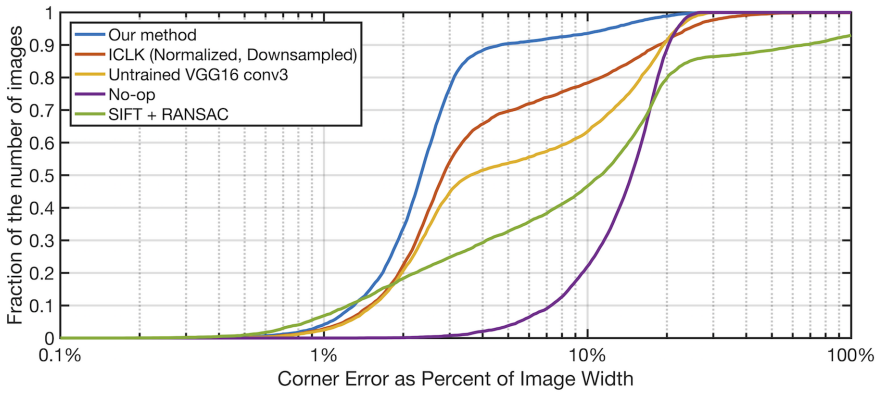


Figure 3.9: Results of testing on 5000 test patches from our *New Jersey* satellite image dataset. The Corner Error as a percentage of image width is reported. The results compare our method (blue), normal ICLK, ICLK computed on untrained VGG16 conv3 features, and SIFT+RANSAC. There are several interesting aspects of the results; for one, we can see that SIFT+RANSAC can align about 7% of the dataset to less than 1% Corner Error. However, at the 1.5% Corner Error threshold, we see that our method (blue) rapidly surpasses the performance of SIFT+RANSAC and the other methods. Our method aligns 80% of the dataset to less than 3% Corner Error. Purple illustrates the results for an algorithm which always returns $\mathbf{p} = 0$ (no-op). We can see that all methods except ours (blue) pass through the purple line, indicating that they perform worse than no-op for some ratio of the dataset.

some notable qualitative alignment results.

Single-iteration vs. dynamic-iteration training. As previously mentioned, there is a choice of how many iterations to do in the ICLK layer of the DeepLK methods, during training. Previous works have designed the system around a single-iteration training scheme due to limitations of the implementation framework. However, we show that dynamic iterations provide a dramatic performance boost for our task in Fig. 3.11. This indicates that for the satellite image alignment task, a dynamic iteration scheme is required to achieve adequate performance. We attribute this requirement to the often dramatic visual differences in satellite imagery, which is likely not as prevalent in frame-to-frame tracking data used in in [86] or in the synthetic data used in [19].

AMOS dataset. Our goal is to show that it is possible to learn temporal invariance for outdoor scenes in general. Thus, we propose to train our network using the satellite imagery dataset *only*, and test performance on both satellite imagery and the AMOS dataset. We hypothesize that by training on satellite imagery, we can learn invariances for both low and high texture, across the paradigms of most outdoors scenes (urban or rural), and across the variations that occur from large temporal differences in the images.

For the AMOS Dataset, we first test on 2000 data pairs from a webcam located in St. Louis, Missouri, USA. Some representative examples of this dataset are shown in Fig. 3.12. The alignment results for this dataset are shown in Fig. 3.14. Notably, we find that our alignment method, which has been trained only on satellite data,

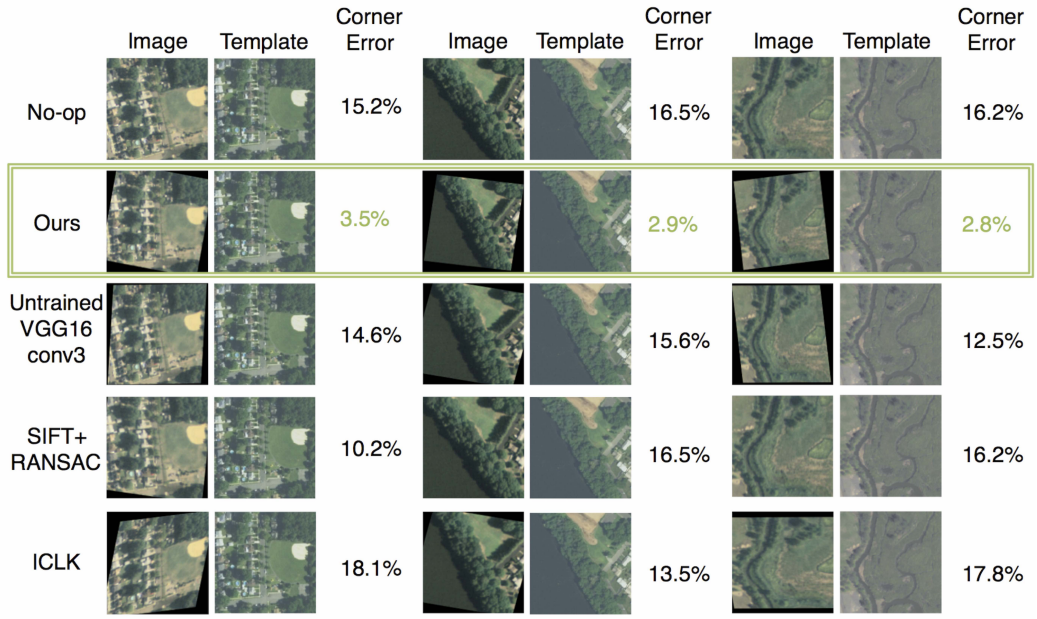


Figure 3.10: Notable results from testing on the *New Jersey* satellite image dataset. Our method has the ability to perform alignment on low texture scenes, despite significant style differences due to temporal effects. In many experiments, the other methods have final Corner Error that is greater than the Corner Error of no-op. We also find that SIFT+RANSAC is unable to estimate any warp at all due to lack of texture in some cases.

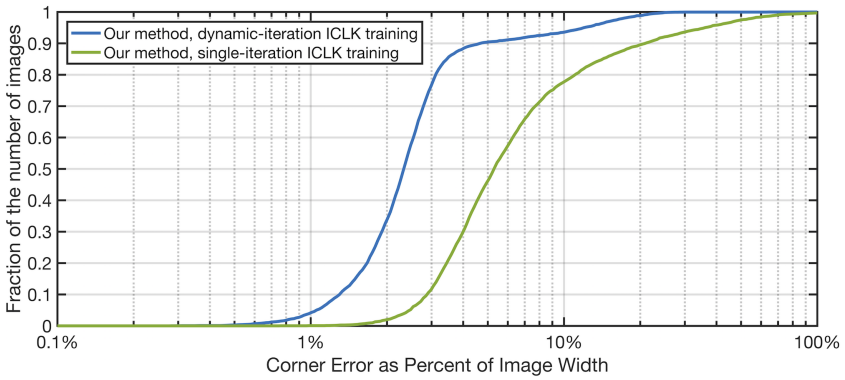


Figure 3.11: Performance difference for single-iteration training methods of deep alignment on satellite imagery, vs. dynamic iterations. The testing data is identical to the testing data shown in Fig. 3.9 (*New Jersey* satellite data). The single-iteration trained network has much higher Corner Error percentage throughout the testing dataset. We attempted training several times using both single-iteration and dynamic-iteration, and present the best results for both.

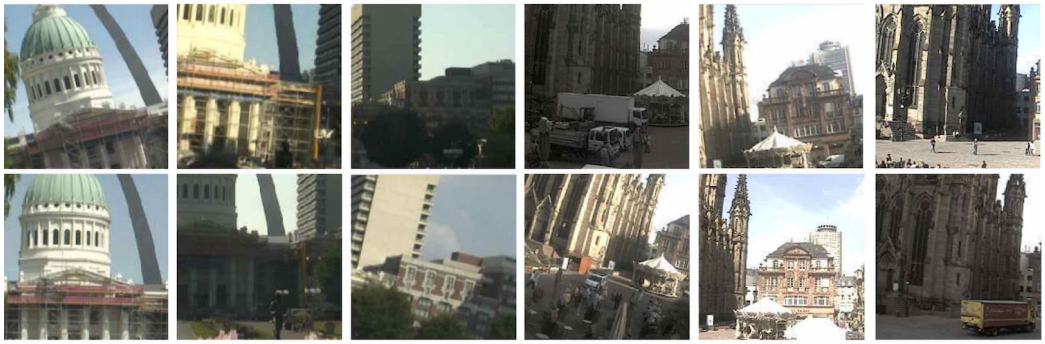


Figure 3.12: Some representative patch examples from the time-lapse webcam dataset [44]. The three example data pairs on the left are from the *StLouis* testing dataset, and the three example pairs on the right are from the *Courbevoie* dataset. We show that invariance that is learned from the satellite data can be transferred to the webcam alignment task; our algorithm performs the best among methods we experimented with on aligning the above examples, despite training only on satellite data.

has learned invariances to outdoor scenes which allow it to be effective at alignment on the AMOS dataset. Some alignment examples are captured in Fig. 3.13.

We test also on 2000 data pairs from a webcam located in Courbevoie, France. Some representative examples of this dataset are shown in Figure 3.12, with the alignment results show in Figure 3.15. Again, we find that the network trained only on satellite images is able to generalize for this alignment task. Specifically, we find that our network can align 80% of image pairs in the Courbevoie dataset to within 5% corner error. The SIFT+RANSAC method, and ICLK on untrained VGG16 conv3 features, both align about 60% of this dataset to within 5% corner error.

3.2.5 Implementation Details

For generating image pairs, we randomly select two aligned images from a given dataset. We extract square patches in the images which range from 175 pixels to 300 pixels wide for the satellite image dataset, and square patches between 150 pixels and 220 pixels wide from the AMOS dataset. We extract a padded version of the image I, so that when it is warped, there are not cutoff regions around the edges. We warp the image I, choosing projective warp parameters from uniform random distributions. We choose warp parameters such that if the algorithm were to predict $\mathbf{p} = 0$ for every test example (no-op), the maximum Corner Error would be about 30% of the image width.

We transfer the conv3 layer of the VGG16 network for our convolutional pipeline, and fine-tune only conv3. We implement the algorithm using the open-source PyTorch framework, on an NVIDIA GeForce Titan X GPU. We trained on 15,000 dynamically generated training pairs from the New Jersey satellite image dataset. We implemented a mini-batch training approach, calculating the Corner Loss on a mini-batch of 5 training pairs before applying the stochastic gradient update. We

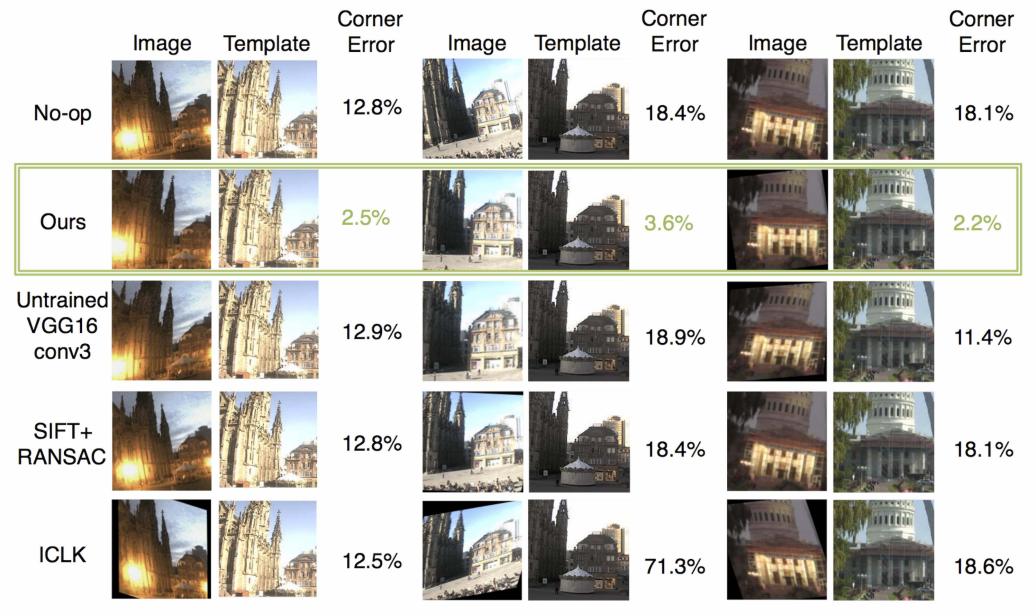


Figure 3.13: Notable alignment results using our algorithm to align time-lapse AMOS data, using features trained only on satellite imagery. These results show that deep features can be used to learn temporal-invariance in one domain (satellite imagery) and transfer this invariance to a similar but unseen domain (time-lapse imagery from near ground-level).

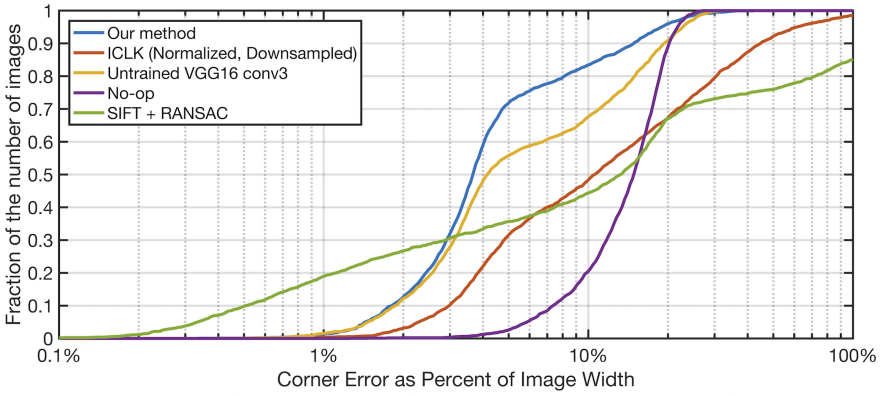


Figure 3.14: Results for Corner Error when testing on the *StLouis* webcam dataset, after training only on satellite images. We can see that our method is able to align 70% of the test pairs to less than 5% Corner Error. This indicates that we have learned invariance to outdoor scenes from satellite data, and have transferred that invariance to the task of webcam data alignment.

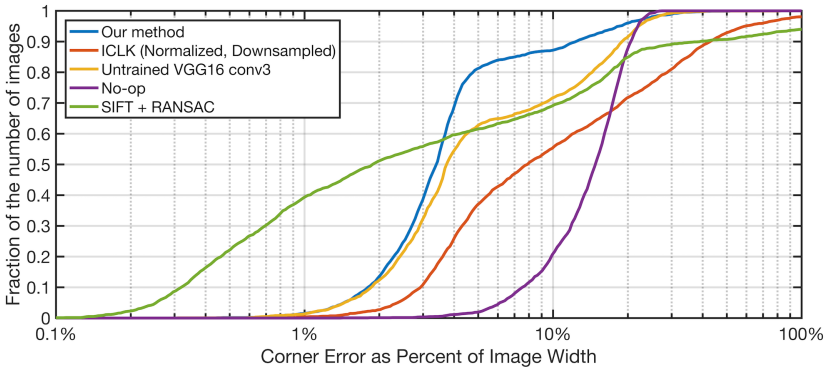


Figure 3.15: Results for Corner Error when testing on the *Courbevoie* time-lapse webcam dataset, after training only on satellite images. Although SIFT+RANSAC performs well because of the high-texture scenes, our method our method can align 80% of the test pairs to less than 5% Corner Error, the best of all other methods at the 5% Corner Error threshold. The invariance that is learned by training the conv3 layer on satellite images is persistent when testing on the unseen data; we can see this from the fact that our method outperforms the untrained VGG16 conv3 layer.

generate a validation batch of 20 image pairs randomly at train time from the source data (*New Jersey* satellite data), and keep the model which generates the lowest validation loss during training.

3.2.6 Discussion

We have shown that deep features can be used effectively to align imagery across large temporal gaps, such as with satellite imagery and time-lapse webcam data. We have further shown that a temporal invariance can be learned from satellite images which transfer with some effectiveness to a different but similar task of time-lapse webcam data.

In some cases, SIFT can be competitive or better than the proposed algorithms for registering images across large gaps in time, and it is important to understand in what cases this is true. SIFT is inherently reliant upon local textures, and the similar visual appearance of these local texture between template and source image. In the webcam dataset especially, there are examples where there is enough texture and the appearance difference is not so drastic, and in these cases SIFT registration can be successful. Also, in these cases SIFT can be quite accurate since corresponding local features allow precise registration. But the limitations of SIFT and most other baseline methods, are that they cannot handle drastic appearance difference (night vs. day, summer vs. winter) or low texture outdoor settings (lack of urban landmarks) which are best learned through larger datasets.

The DeepLK approach for aligning with temporal invariance can be extended naturally to the UAV geolocalization application. In this application, DeepLK can be used to match the imagery from a UAV with a downward facing monocular RGB camera with a pre-existing satellite map, thus localizing the vehicle. This extension

will be discussed in the following Section 3.3.

3.3 UAV Geolocalization with Satellite Imagery

The commercial and consumer use of UAVs outdoors has grown exponentially in the last few years. UAVs now find use in search and rescue [76, 70], industrial inspection [63, 17], land surveying and mapping [79, 62], precision agriculture [82], monitoring of remote environments [49, 57], the study of wildlife populations [42], and many more. Nearly all applications require precise latitude and longitude estimates of the UAV during flight, with some also requiring accurate altitude or 6 degrees-of-freedom (DoF). This level of localization may not be available in GPS-denied or GPS-spoofed situations, and requires the use of often costly GPS hardware and IMUs. A vision-based system that achieves comparable accuracy would be beneficial as it would lower the cost of UAV platforms, and could replace GPS when there are signal issues. This idea is further motivated by the fact that there is ample free, GPS-aligned, satellite imagery covering many parts of the globe available online. This satellite imagery can provide a map prior for a flight, against which we can perform template matching to localize.

The main challenge of using satellite imagery as a map prior is overcoming the differences in imaging conditions between the satellite images and the incoming video stream from the UAV. An example of this is shown in Fig. 3.16. Since the UAV images are captured at much lower altitude than the satellite map, there is a larger perspective effect of structures higher than the ground plane. The correspondence of typical feature descriptors like SIFT [48], which are commonly used in the remote sensing community for satellite-to-satellite image matching [36, 10, 92], usually fails with such perspective differences. Another challenge can come from temporal aspects, such as seasonal effects, time-of-day, and the removal or addition of buildings or cars. Overcoming these temporal aspects has been attempted using learned, sparse descriptors in the TILDE work [85]. However, sparse descriptors are still reliant on local texture and inherently cannot generalize to more rural, low texture flight environments.

To tackle the registration problem, we will use the DeepLK method developed in the previous section. The previous tests on satellite imagery and webcam data are indicative of the generalization performance of the learned deep features. We make the assumption that learning features that are robust between temporally varying satellite images, is a similar task to learning features that are robust between a satellite image and imagery from a high-altitude UAV. This assumption is shown to be true through our experiments.

3.3.1 Prior Work in UAV Geolocalization

There have been a few attempts in recent literature to develop a full UAV localization system based on satellite image matching. In [77], the authors use HOG features

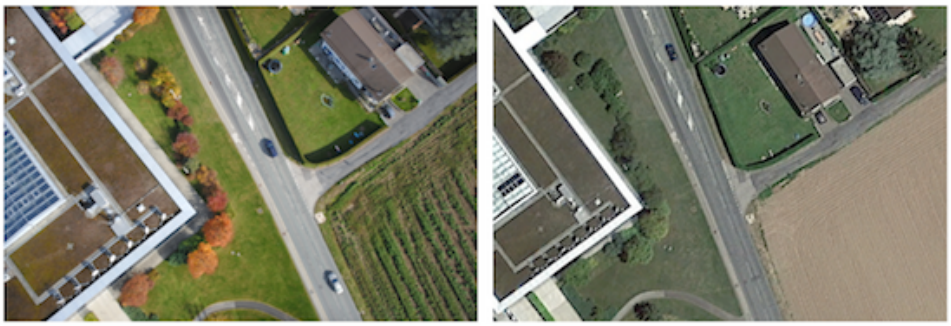


Figure 3.16: A typical example of alignment between UAV imagery (left) and a satellite image (right). Notice differences such as seasonal variation of vegetation, shadow angles, perspectives of buildings, presence of vehicles, and variations due to different imaging hardware. Our goal is overcoming these variations to enable precise UAV localization.

for alignment between UAV imagery and the map, making their approach reliant on well-defined, temporally consistent texture such as buildings and roadways. In [94], the authors accomplish alignment using a mutual information metric. They show the method working only on a UAV flight over a textured urban environment, and the satellite map they use is photometrically very similar to the UAV imagery captured. In [60], the authors develop a complex pipeline for alignment which includes SIFT matching and semantic segmentation of buildings. This makes their method heavily reliant upon the presence of texture, as well as on the photometric similarity of UAV images and the map. They must also train their neural network semantic segmentation on data extracted from the exact location and with similar imaging qualities as used at test time.

To give an idea of the performance of these systems, each achieves average localization error of less than approximately 10 meters across flight distances of between 300-1500 meters, and at altitudes of 100-300 meters. All previous work makes the flat-world assumption in order to parameterize motion using the planar homography, a model that we will also use.

3.3.2 Formulation of Geolocalization

Our goal is to obtain accurate, absolute pose of the UAV at any frame F during the flight sequence. We make the flat world assumption and parameterize the motion of the UAV in terms of planar homographies with respect to the flat world. Assuming we have a satellite map M that is aligned to GPS coordinates, then the absolute pose of the UAV at any frame can be encoded in a homography relating the satellite map image to the current frame, which we call $\mathbf{H}_{abs}^{F}$. Then, the goal is to estimate $\mathbf{H}_{abs}^{F}$ for all frames.

We assume the position of the UAV is approximately known at the time of capturing the initial frame $F = 1$. This is a reasonable assumption in applications where the approximate take-off position is known, or where a single GPS data point is given,

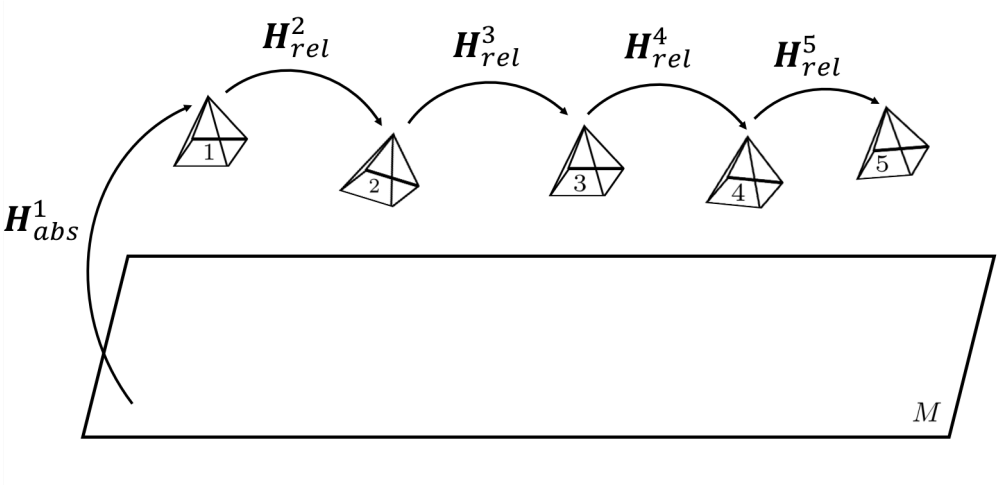


Figure 3.17: Visualization of the geolocalization formulation. $\mathbf{H}^1_{abs}$ parameterizes the approximately known initial pose of the UAV with respect to the satellite map M, at the time that GPS-denied flight begins. Estimates of motion via visual odometry ($\mathbf{H}^F_{rel}$) can be combined with the initial absolute position ($\mathbf{H}^1_{abs}$) to estimate the absolute position ($\mathbf{H}^F_{abs}$) at each frame F.

before beginning GPS-denied flight. There is an absolute homography $\mathbf{H}^1_{abs}$ relating the initial view of the UAV to the satellite map image M at frame $F = 1$, whose parameters can be determined based on the approximate heading and GPS location of the UAV. As the UAV moves, each frame F is related to the last frame $F - 1$ by a relative homography $\mathbf{H}^F_{rel}$ computed using image pairs from the UAV (visual odometry). Therefore, the absolute homography relating the view from the UAV at frame F to the satellite map is the composition of the intial absolute homography $\mathbf{H}^1_{abs}$ with all relative homographies up to frame F:

$$\mathbf{H}^F_{abs} = \mathbf{H}^F_{rel} \cdot \mathbf{H}^{F-1}_{rel} \cdot ... \cdot \mathbf{H}^2_{rel} \cdot \mathbf{H}^1_{abs} \tag{3.9}$$

In Fig. 3.17, we visualize the motion parameters, map M, and camera frustums.

3.3.3 Localizing Each Frame

One option for geolocalization is to compare each incoming UAV frame to the map M, and estimate the UAV position for each frame individually. Beginning with the first frame, we would compare and align the frame to the satellite map M, and use this alignment to refine $\mathbf{H}^1_{abs}$. For the next frame, we can estimate $\mathbf{H}^2_{rel}$ between frame one and two using a registration method of choice (e.g. SIFT or LK). The estimated absolute position of the UAV at frame two is then described by $\mathbf{H}^2_{abs} = \mathbf{H}^2_{rel} \cdot \mathbf{H}^1_{abs}$. Then, we can compare frame two to the map M to refine $\mathbf{H}^2_{abs}$. We could repeat this process for every incoming frame. In general, this approach can be taken as long a two assumptions are satisfied: the map M exists at the current location (there is no missing data or "holes" in the map, and we have not flown outside the map extent) and the map M can be sufficiently registered with the

current UAV frame (there is not such drastic appearance difference, or corrupted map pixels, between M and the current image). In the Experiment section, we show how well this approach works on two flight datasets. However, as discussed in the next section, there is a question of how to handle the cases where the two aformentioned assumptions are not satisfied.

3.3.4 Pose Optimization for Missing Map Sections

Motivation. Consider the two assumptions made when localizing against the map M for *every* UAV frame: the map M exists at the current location (there is no missing data or "holes" in the map, and we have not flown outside the map extent) and the map M can be sufficiently registered with the current UAV frame (there is not such drastic appearance difference, or corrupted map pixels, between M and the current image).

When either of these two assumptions is broken, it is not possible to geolocalize the frame. In this case, we must use the visual odometry estimate of motion $\mathbf{H}_{rel}^{F}$ for that particular frame only, and cannot refine this motion estimate by comparing to the map M. This is not particularly bad if we do not need to know the absolute position for this frame and can afford to skip it. However, it would be beneficial if we could use the absolute position of past and future frames to help in estimating the absolute position of the frame which could not be compared to the map.

Motivated by this problem, we devise a novel, LK-based optimization over multiple poses at once, which incorporates photometric constraints from both visual odometry and map comparisons. Our goal is to gracefully handle cases where some subset of UAV frames cannot be compared effectively to M, whereas other neighbording frames in the UAV sequence *can* be effectively registered to the map.

Formulation. We define certain frames throughout the UAV sequence as template frames T, to which other temporally adjacent frames are aligned. We define visibility neighborhoods V around each template, which contain the frames adjacent to T. V should be chosen so that there is sufficient overlap between all frames in V with T so as to allow direct registration such as with Lucas-Kanade. The precise choice of template frames and visibility neighborhoods depends on many characteristics of the particular flight hardware and the speed of the vehicle, and must be tuned for a given application.

In Fig. 3.17 for instance, we could define a single template T_3 as the third frame, and a visibility neighborhood for this template containing the other four frames: $V(T_3) := 1, 2, 4, 5$.

To align with the map, we extract the deep feature representation of all T using our trained convolutional layers from Section 3.2. We also extract the patch from the map M which corresponds to T based on the current estimate of all motion parameters in the sequence, and extract the deep features of this map patch. Then, we simultaneously optimize for the motion parameters based on the minimization of error between all templates T and the images within their corresponding visibility

neighborhoods V, as well as the error between the deep feature extractions of the templates T and the map M. We express this minimization objective as:

$$\min_{\mathbf{p}} \sum_{k} \sum_{F \in V(k)} \left\| I_F(\mathbf{W}(\mathbf{x}; \mathbf{p}_{F,k})) - T_k(\mathbf{x}) \right\|_2^2 + \lambda \sum_{k} \left\| M^{\phi}(\mathbf{W}(\mathbf{x}; \mathbf{p}_{M,k})) - T_k^{\phi}(\mathbf{x}) \right\|_2^2 \tag{3.10}$$

Where k is the set of frames which are templates, I_F is the UAV image at frame F, M^{ϕ} and T^{ϕ} are deep feature extractions of the map and template respectively, $\mathbf{p}_{F,k}$ is the composition of motion parameters from frame F to template k, and $\mathbf{p}_{M,k}$ is the composition of motion parameters from the satellite map M to the template k. As previously mentioned, we use the notation $I(\mathbf{W}(\mathbf{x}; \mathbf{p}))$ to mean sampling an image I at image coordinates $\mathbf{x}$ that have been warped with warping function $\mathbf{W}$, using a projective transformation parameterized by $\mathbf{p} \in \mathbb{R}^8$.

λ is a tunable parameter used to weight the contribution of the map alignment; if it is zero, the optimization does not use the satellite map at all, but optimizes the motion parameters based only on UAV imagery (visual odometry).

Derivation of motion parameter update. The minimization in (3.10) can be solved using the iterative Gauss-Newton method, in a similar manner to the derivation of the original (forward compositional) Lucas-Kanade algorithm. We start by defining $\mathbf{p}_f$ of a particular relative homography in the UAV sequence $\mathbf{H}_{rel}^{f}$ that we optimize with respect to. We rewrite both $\mathbf{p}_{M,k}$ and $\mathbf{p}_{F,k}$ as functions of $\mathbf{p}_f$ with the addition of a small perturbation $\Delta\mathbf{p}_f$ to the motion parameters:

$$\begin{aligned} \sum_{k} \sum_{F \in V(k)} & \left\| I_F(\mathbf{W}(\mathbf{x}; \mathbf{p}_{F,k}(\mathbf{p}_f + \Delta\mathbf{p}_f))) - T_k(\mathbf{x}) \right\|_2^2 + \\ & \lambda \sum_{k} \left\| M^{\phi}(\mathbf{W}(\mathbf{x}; \mathbf{p}_{M,k}(\mathbf{p}_f + \Delta\mathbf{p}_f))) - T_k^{\phi}(\mathbf{x}) \right\|_2^2 \end{aligned} \tag{3.11}$$

Linearizing with respect to $\mathbf{p}_f$ yields

$$\sum_{k} \sum_{F \in V(k)} \left\| \mathbf{J}_I \Delta\mathbf{p}_f + \mathbf{r}_I \right\|_2^2 + \lambda \sum_{k} \left\| \mathbf{J}_M \Delta\mathbf{p}_f + \mathbf{r}_M \right\|_2^2 \tag{3.12}$$

where

$$\mathbf{J}_I = \nabla I_F \frac{\partial \mathbf{W}}{\partial \mathbf{p}_{F,k}} \frac{\partial \mathbf{p}_{F,k}}{\partial \mathbf{p}_f} \tag{3.13}$$

$$\mathbf{J}_M = \nabla M^\phi \frac{\partial \mathbf{W}}{\partial \mathbf{p}_{M,k}} \frac{\partial \mathbf{p}_{M,k}}{\partial \mathbf{p}_f} \tag{3.14}$$

$$\mathbf{r}_I = I_F(\mathbf{W}(\mathbf{x}; \mathbf{p}_{F,k}(\mathbf{p}_f))) - T_k(\mathbf{x}) \tag{3.15}$$

$$\mathbf{r}_M = M^\phi(\mathbf{W}(\mathbf{x}; \mathbf{p}_{M,k}(\mathbf{p}_f))) - T_k^\phi(\mathbf{x}) \tag{3.16}$$

We note that $\frac{\partial \mathbf{p}_{F,k}}{\partial \mathbf{p}_f}$ is nonzero only if $f \in V(k)$ and $|f-k| \leq |F-k|$, and $\frac{\partial \mathbf{p}_{M,k}}{\partial \mathbf{p}_f}$ is nonzero only if $f \leq k$. Finally, taking the derivative w.r.t. $\Delta\mathbf{p}_f$, setting the minimization equal to zero, and solving for $\Delta\mathbf{p}_f$ yields:

$$\Delta\mathbf{p}_f = -\mathcal{H}^{-1}\left(\sum_k \sum_{F \in V(k)} \mathbf{J}_I^{\mathrm{T}} \mathbf{r}_I + \lambda \sum_k \mathbf{J}_M^{\mathrm{T}} \mathbf{r}_M\right) \tag{3.17}$$

where

$$\mathcal{H} = \sum_k \sum_{F \in V(k)} \mathbf{J}_I^{\mathrm{T}} \mathbf{J}_I + \lambda \sum_k \mathbf{J}_M^{\mathrm{T}} \mathbf{J}_M \tag{3.18}$$

The optimal update $\Delta\mathbf{p}_f$ for all relative homographies $\mathbf{H}_{rel}^f$ can be computed in parallel using this iterative solution, with the update at each iteration $\mathbf{p}_f \leftarrow \mathbf{p}_f + \Delta\mathbf{p}_f$. The iteration continues until the maximum value of $\Delta\mathbf{p}_f$ across all frames is below a certain threshold.

The optimization can be applied in a sliding-window or a all-in-one batch fashion. A sliding-window approach would be more suitable for applications with a requirement for localization at online speeds. We present experiments using a sliding-window, with more details in the Experiments section.

3.3.5 Converting Between Pose and Homography

Up to this point, we have dealt with pose of the vehicle in terms of homography relationships in pixel space. We now address the methods that are used to convert homography relationships to 6 degree-of-freedom rigid body transformations (or vice versa) to determine the geo-localized position of the UAV. Converting from homography to pose is done after pose optimization as a final step.

First, we describe the map coordinate system. Consider the image coordinate system of the satellite map M such that x, y are defined as the horizontal and vertical image axis and $(0, 0)$ is the top left image corner, and consider an additional z axis also with its origin at top left image corner and its orientation "into the image". For

simplicity, assume that the map M is aligned to the GPS coordinate system such that x in the image axis corresponds to longitude and y to longitude, and that we are given the longitude and latitude coordinates corresponding to the four image corners that can be interpolated to find the latitude and longitude of each pixel. Of course, this makes the assumption that latitude and longitude are rectilinear and orthogonal as opposed to spherical; however, this is a valid assumption for the map sizes used in our experiments, which are about one kilometer across.

Now, we describe how to calculate the initial absolute homography $\mathbf{H}^1_{abs}$. We assume the initial pose of the UAV is roughly known using on-board sensors or by using an estimate based on the initial launch position of the UAV. From this information, we have an estimate of the longitude and latitude of the UAV, altitude, and possibly orientation (roll, pitch, yaw), totaling 6 degrees of freedom. We can convert the approximately known initial UAV position to a 3×3 vehicle rotation matrix $\mathbf{R}$ and 3×1 translation $\mathbf{t}$ which are with respect to the coordinate system x, y, z defined previously for the map M. Combining these rotation and translation into a 3×4 pose matrix yields

$$\mathbf{P} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \end{pmatrix}. \tag{3.19}$$

.

Using this pose matrix and the UAV camera intrinsic matrix $\mathbf{K}$, we have that a homogeneous 3×1 pixel location in the UAV camera image $\mathbf{x}_I$ corresponds to a homogeneous 4×1 3D point on the map $\mathbf{X}_M$ as

$$\mathbf{x}_I = \mathbf{K}\mathbf{P}^{-1}\mathbf{X}_M. \tag{3.20}$$

Note however that with the map coordinate system as defined, all "3D" points actually lie on the plane $z = 0$, and all 3D points $\mathbf{X}_M$ thus have $z = 0$. This means that we can ignore the 3rd column of the matrix $\mathbf{P}^{-1}$, instead writing

$$\mathbf{x}_I = \mathbf{K}\mathbf{P}^{-1}_{3\times3}\mathbf{x}_M, \tag{3.21}$$

with $\mathbf{P}^{-1}_{3\times3}$ defined as all columns of $\mathbf{P}^{-1}$ except the third, and $\mathbf{x}_M$ defined as a pixel location in M. Thus, $\mathbf{H}^1_{abs} = \mathbf{K}\mathbf{P}^{-1}_{3\times3}$.

Now, we describe how to calculate $\mathbf{R}, \mathbf{t}$ for the current frame from the estimated absolute homography for the current frame $\mathbf{H}^F_{abs}$. First, recall that $\mathbf{H}^F_{abs} = \mathbf{H}^F_{rel} \cdot \mathbf{H}^{F-1}_{abs}$, and that if we begin with the first frame then $\mathbf{R}$ and $\mathbf{t}$ for $\mathbf{H}^{F-1}_{abs}$ will be known. Then, the goal is to determine $\mathbf{R}$ and $\mathbf{t}$ of the relative homography $\mathbf{H}^F_{rel}$, and combine this relative transformation with the currently known pose of $\mathbf{H}^{F-1}_{abs}$. This process involves the decomposition of homography into pose. To decompose homography, we use the implementation provided in OpenCV [14], which in turn implements the method described in [53]. After decomposing, the relative pose update may be combined with the current absolute pose.

3.3.6 Experiments

As few attempts have been made at UAV localization systems similar to the one presented here, there are no freely available datasets or baseline implementations. A proper dataset for this task consists of a UAV sequence using a monocular, downward-facing camera, with precise ground truth pose with respect to the Earth, and an accompanying globally aligned satellite image of the flight location. The altitude of the UAV must also be such that the flat-world assumption can reasonably be made. Of the few previous works [77, 94, 60], none have made their datasets or source code publicly available for comparison. Therefore we gather two datasets, in part using UAV imagery from the free datasets offered from the senseFly professional drone mapping company[1]. The first dataset is captured overhead the Swiss village of Merlischachen, at an altitude of 0.2km, over 0.85km flight distance. This is an urban environment similar to those used in all prior works, with ample texture and landmarks such as buildings and roadways. The second dataset is more challenging, captured overhead a rural gravel quarry at 0.22km altitude and flight distance of 0.61km. This location offers substantially less texture and landmarks for alignment. We use this dataset to showcase the capabilities of our system to align with low-texture imagery, an advantage over past approaches.

Village Dataset. Some example UAV frames from the Village dataset and an overview of the flight are in Fig. 3.18. The UAV used is the senseFly eBee drone, equipped with a downward facing Canon IXUS 125 HS camera. We extract the GPS-aligned satellite map from Google Earth Pro. The UAV imagery was captured in April, 2013, and the satellite imagery in May, 2012. High-accuracy RTK GPS (latitude, longitude, and altitude) is included in the metadata of each UAV frame in the dataset. We process the GPS metadata of the initial frame of the dataset in order to form $\mathbf{H}^1_{abs}$, the homography relating the initial UAV pose to the satellite map. To form initial estimates of the frame-to-frame relative motion $\mathbf{H}^F_{rel}$, we use SURF [8] feature extraction and correspondence to estimate homographies between UAV frames. We find SURF provides odometry performance comparable to other popular sparse-feature based methods, and is thus used as a representative baseline.

Matching every frame. With estimates for $\mathbf{H}^1_{abs}$ and $\mathbf{H}^F_{rel}$, we compare different approaches for map alignment based on the different aforementioned assumptions. In the first case, we compare every UAV frame to the map (assuming that the map exists for every frame, and there is not any registrations which fail between UAV images and map). **Using this approach, we find that the average localization error is 6.89 meters in the x-y plane, and 7.92 meters in altitude error, or 10.49 meters in 3D.** This average is across all frames in the Village dataset. In other words, we do find that our method can successfully retain "tracking" between the UAV imagery and the map, when considering all available UAV frames in the dataset. This error is competitive with or less than the errors seen in prior work.

Handling skipped frames with pose optimization. However, we now consider the case where we must skip map alignment for some frames, and we will use the aforementioned pose optimization to account for the skipped frames. For the

[1] https://www.sensefly.com/education/datasets/

Village dataset, we find empirically that using a sliding-window approach for the pose optimization produces the lowest average localization error. This is due to the low frame-rate of the UAV dataset, where there is as low as 50% frame-to-frame overlap. This makes the composition of $\mathbf{H}_{rel}^{F}$ from the SURF odometry inaccurate when predicting pose after many frames. It is this predicted position which is used to extract patches from the map M in the optimization to use for alignment, but if these patches do not have significant overlap with the UAV frames then the optimization fails. We use a sliding-window containing 4 UAV frames, of which the 2nd and 4th frame images are the templates T, with $V(2) = 1, 3$ and $V(4) = 3$ as the visibility neighborhoods of the two templates. This window is slid forward by two frames after the optimal parameters are found. We start with the first 4 frames of the video, and progress through the whole sequence. With this sliding window, only half of all video frames are directly compared with the map (assuming that the other half of frames cannot be compared to the map for some reason, such as map parts missing, corrupted map pixels, or very large appearance difference between map and UAV image). **For this experiment, we find that the average error measured in 3D distance is 10.67 meters (result shown in Fig. 3.19), as compared to 11.03 meters if the odometry position estimates only are used for the skipped frames.** This result shows that our optimization approach is able to reduce the localization error as compared to a naive approach to localization with skipped frames.

After solving for the optimal motion parameters, we convert them to GPS coordinates (latitude, longitude, and altitude) and plot error from ground truth, versus SURF odometry. Localization results on the Village dataset are shown in Fig. 3.19, and some example alignments in Fig. 3.20. Despite relatively little training data as described in Section 3.2, the learned features generalize well to the unseen UAV imaging conditions and satellite map used in both the Village and Gravel Pit datasets. We find other alignment methods simply do not come close to proper alignment on these datasets, including ICLK on normalized images, or ICLK on vanilla conv3 VGG16 features. SIFT produces very few (3 to 5) if any correct correspondences between UAV images and satellite images, not enough for effective RANSAC estimation.

Gravel pit dataset. Example frames from the Gravel Pit dataset are shown in Fig. 3.21, along with a flight overview. The same UAV and camera are used from the Village dataset. The UAV imagery was captured in April, 2013, and the satellite map we extract is from November, 2014. We chose this dataset to showcase the abilities of our method of alignment for use on very low-texture environments lacking landmarks. The effects of seasonal variation are more pronounced in this dataset due to increased vegetation as well.

We repeat the process from the Village dataset for extracting initial motion parameter estimates. For the motion parameter optimization, we experiment by eliminating the inclusion of the odometry term in (3.10) and using only the minimization of the map with templates. Further, we use a sliding-window of size 1 (a single template), and move the sliding window by 1 frame each step. This way, the optimization will seek to fully align all UAV frames with the map. This is equivalent to what is done in prior work, where visual odometry is not taken into consideration for

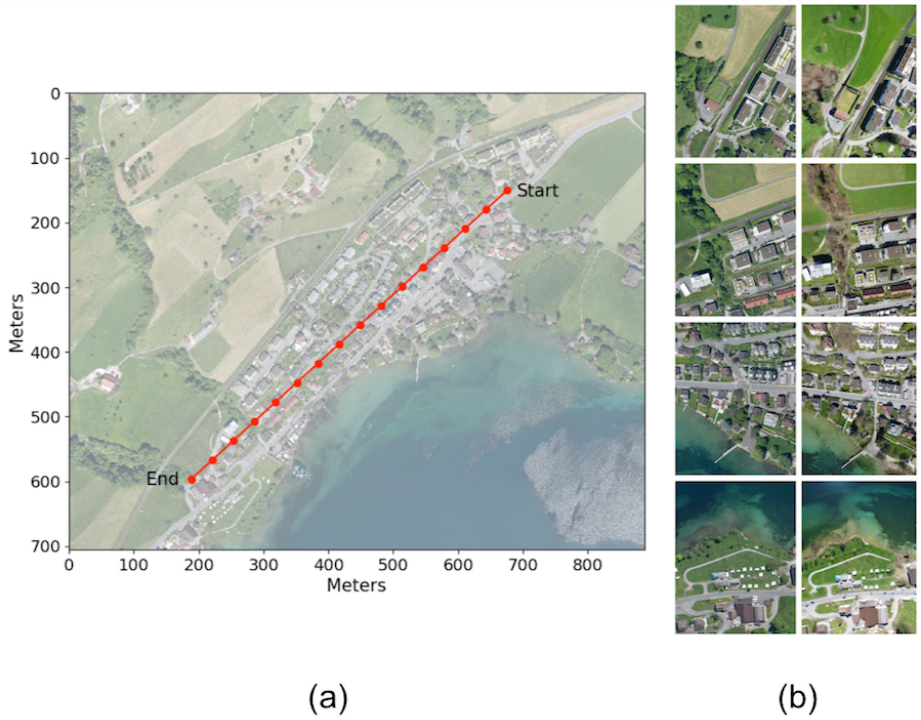


Figure 3.18: (a) Overview of the Village dataset flight path. (b) Some examples of UAV frames (right) and their corresponding satellite map patches (left) for the Village dataset.

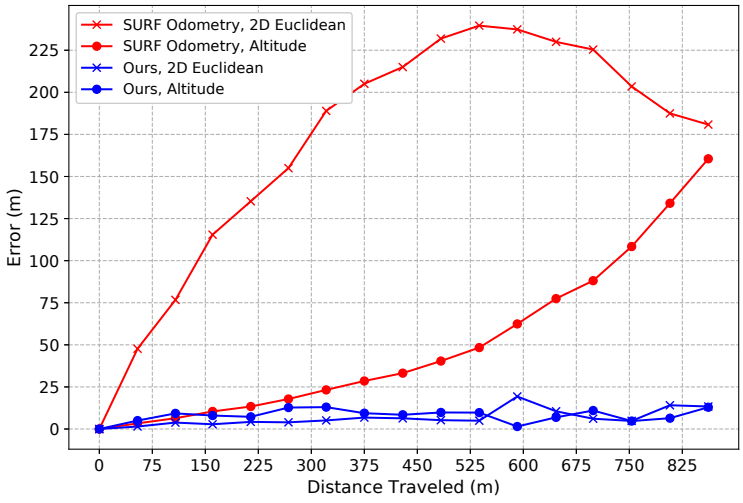


Figure 3.19: Our results on the Village dataset. Results labeled 2D Euclidean are error distances measured in the x-y plane. Markers represent template frames. The average 2D euclidean error of our method is 7.06m. The average altitude error of our method is 8.01m. The ground truth altitude of the UAV for the entire sequence is approximately 0.2km. We find that alignment methods including SIFT+RANSAC or ICLK on vanilla VGG16 conv3 features fail to align UAV images with the satellite map for this dataset, and thus do not improve on the error from SURF odometry.

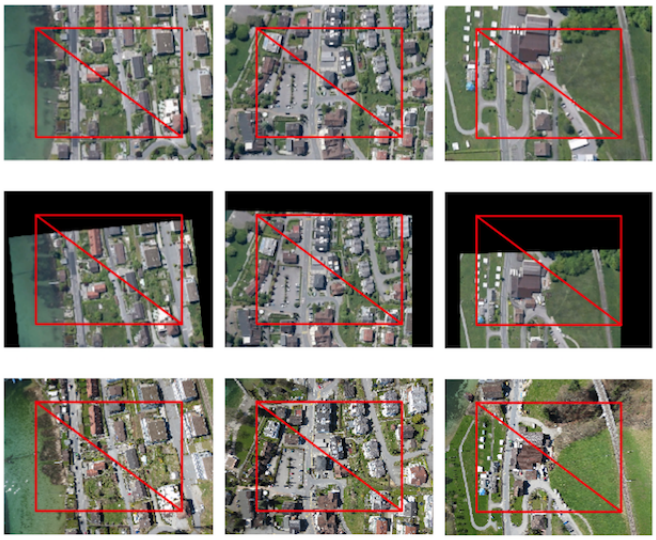


Figure 3.20: Some examples of the alignment capabilities of our system on the Village dataset, which occur during optimization of motion parameters. Crops from the satellite map (top) are warped and aligned (middle) with the corresponding UAV imagery (bottom). The red rectangle is static in all images, allowing the reader to visually compare aligned landmarks.

computing the optimal motion parameters. We use this approach to illustrate that the algorithm recovers from misalignment on one frame and successfully relocalizes in the next frame, even with large UAV motion between frames. Results are shown in Fig. 3.22, with some example alignments in Fig. 3.23. We observe that the registration performance does degrade due to the difficulty of the low-texture; however, there is still clear visual evidence that the registration has not lost "tracking" with the map, which is most clearly seen in Fig. 3.23.

3.3.7 Implementation Details

We use an OpenCV [14] implementation of SURF detection and computation, with brute-force matcher and cross-check for smallest distance match. We also use the OpenCV's `findHomography` function with RANSAC, with an error threshold of 5 pixels. The satellite map for the Village dataset is 4800×2861 pixels at 0.45 meters per pixel width. The map for the Gravel Pit dataset is 3355×1852 at 0.32 meters per pixel width. Native UAV image resolution is 4608×3456. UAV images and map patches are square-cropped and scaled to 200 pixels each during pose optimization. The output of VGG16 conv3 has 256 feature channels and is 3-times spatially downsampled from the input. On a 2.9 GHz Intel Core i5 laptop with 16 GB RAM, optimization using 2 templates with 4 frames in each visibility neighborhood, and two overlapping frames between neighborhoods, takes 8.41s on average over 10 trials. We empirically find $\lambda = 0.35$ best for optimization experiments on Village dataset, although the automatic selection of best lambda we leave for future work.

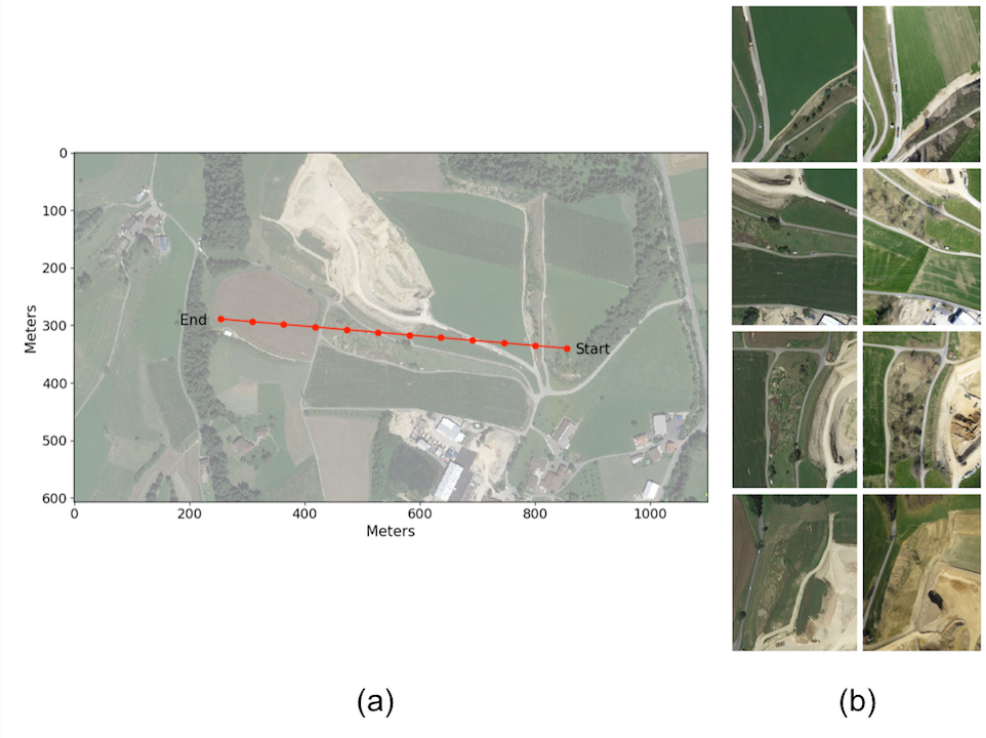


Figure 3.21: (a) Overview of the Gravel Pit dataset flight path. (b) Some examples of UAV frames (right) and their corresponding satellite map patches (left) for the Gravel Pit dataset.

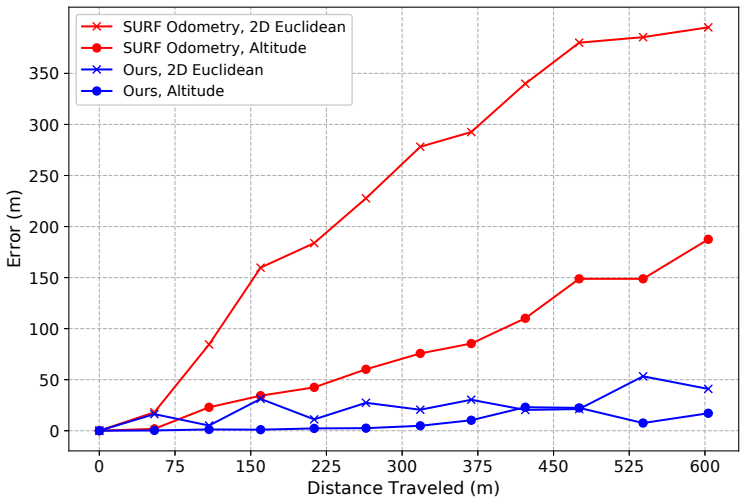


Figure 3.22: Our results on the more challenging, low texture Gravel Pit dataset. The average 2D euclidean error of our method is 25.00m. The average altitude error of our method is 7.70m. Precise x-y localization is difficult on this dataset due to low texture, leading to ambiguous alignment. However, no other methods we try for UAV-map matching are able improve on the odometry error as ours has. We note that altitude error may be lower since the planar assuption was particularly well-satisfied for this dataset i.e. the ground is quite fronto-parallel to the nadir UAV camera as opposed to the Village dataset, where there is a consistent sloping landscape.

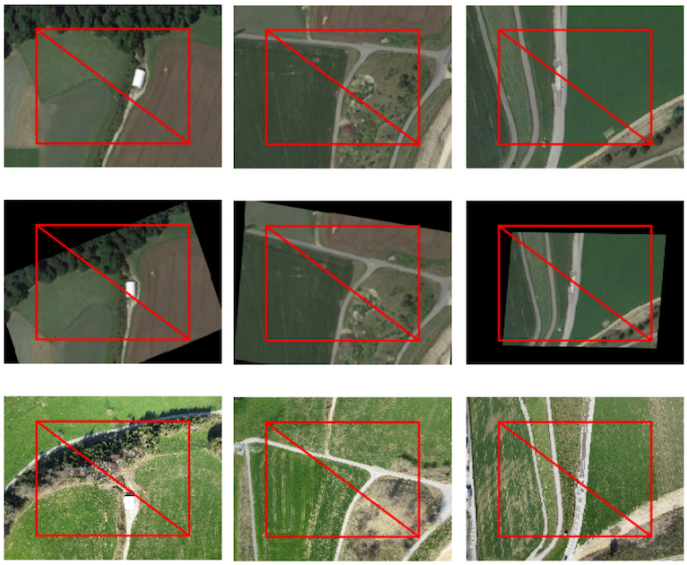


Figure 3.23: Some examples of our alignment on the Gravel Pit dataset. Crops from the satellite map (top) are warped and aligned (middle) with the corresponding UAV imagery (bottom). The red rectangle is static in all images, allowing the reader to visually compare aligned landmarks. These examples show that we are able to maintain tracking and registration to the map, as there is clear correspondence and alignment taking place between visual landmarks.

3.4 Chapter Discussion

In this chapter, we have studied the difficult problem of registration across large temporal gaps. First, we showed in Section 3.1 that deep convolutional features from pre-trained networks could be effective in providing temporal invariance, even without training on a specific dataset. Next, in Section 3.2, we showed the DeepLK method could find application in this problem, and we showed that a novel change in the training procedure of DeepLK (dynamic training iterations) could dramatically improve registration results on satellite imagery. Finally, we showed in Section 3.3 how the DeepLK approach could be integrated into a full UAV geolocalization pipeline based on matching UAV imagery to satellite imagery, and showed experiments on UAV flight datasets. We further developed an optimization which combined deep features and visual odometry constraints within a Bundle-Adjustment-like optimization, inspiring further work in the integration of deep feature representations into existing SLAM pipelines.

In the next chapter, we explore deep learning as it can be applied to point cloud registration instead of image registration. Analogously to the image case, where we addressed temporal invariance, we will address challenging registration scenarios in 3D. Namely, noisy, partial, and real-world data which proves difficult to register with existing approaches. The connection between this chapter and next, is the exploration of the usefulness of deep network representations for difficult registration scenarios.

Chapter 4

Learning for Point Cloud Registration

Work from this chapter was done in collaboration with Yasuhiro Aoki, Rangaprasad Arun Srivatsan, Vinit Sarode, and Xueqian Li.

Point cloud registration in 3D is at the core of many applications such as localization and mapping [58, 61], reconstruction [47], object matching [30], detection and shape completion [89], and tracking [71, 87], among others. Existing methods of point cloud registration, such as ICP, and 3D sparse feature methods, are discussed in the Background chapter.

Across these applications, there are varying levels of difficulty involved in performing accurate registration. As discussed in the Background chapter, a simple registration scenario might be between a template and source which consists of an identical sampling of points, uniformly sampled from the entire surface of an object, with a small perturbation between template and source. This is typically an easy registration scenario for two primary reasons: the cost function for registration will be zero when template and source are perfectly aligned since the sampling is the same, and the perturbation is small and unlikely to lead to a registration algorithm finding a local minimum. Thus, we can understand the level of difficulty of a registration in terms of these two aspects: *sampling* and *perturbation*.

The difficulty of registration generally increases as the magnitude of the *perturbation* increases, and in particular when the magnitude of the difference in orientation (rotation) increases. The difficulty of registration also increases when the *sampling* between template and source are no longer equal. For example, the source may be a different surface sampling than template, may have added isotropic or anisotropic noise at each point, or may be a sampling from only a partial viewpoint of the scene.

In the following chapter, we explore the use of deep learning in point cloud registration, and specifically address how deep learning can improve performance in registration scenarios at varying levels of sampling and perturbation difficulty. We show that in some key scenarios, deep learning can improve upon state-of-the-art methods in difficult registration scenarios, such as the presence of noise, partial

viewpoint, and real-world data which is far removed from the training distribution used during learning.

4.1 PointNet for Registration

Point clouds are unstructured, having no inherent order. This lack of structure makes it difficult to create deep networks which can process the data. However, the introduction of PointNet [68] provides a unique solution to this dilemma, in providing a learnable representation for point clouds. PointNet is the first deep neural network which processes point clouds directly, as opposed to alternative representations such as 2D image projections of objects [90, 15, 32], voxel representations [54, 89, 98] or graph representations [88]. By processing point clouds directly instead of through an auxilliary representation, PointNet can achieve extremely fast processing speeds. Within larger network architectures, PointNet has proven to be useful for tasks including classification, semantic segmentation, object detection [67], and completion of partial point clouds [97].

Although it has helped achieve state-of-the-art performance in other point cloud tasks, PointNet has not yet been applied to the problem of point cloud registration. In the following sections, we outline **three** methods that we experimented with for introducing PointNet into a registration pipeline:

1. **PCRNet.** PCRNet can be thought of as a one-shot regression, from template and source point cloud to the pose parameters which align them. The tasks of both point cloud representation and registration are learned from data.

2. **Iterative PCRNet.** Iterative PCRNet uses the basic PCRNet structure, but combines multiple incremental estimates over time to align template and source.

3. **PointNetLK.** An iterative framework where only the point cloud representation is learned, and not the task of registration. Registration is instead formulated as the minimization of distance between representation vectors, which is solved using a Gauss-Newton optimization similar to Lucas-Kanade.

Notation. We denote matrices with uppercase bold such as $\mathbf{M}$, constants as uppercase italic such as C, and scalar variables with lowercase italic such as s.

Let ϕ denote the PointNet function, $\phi : \mathbb{R}^{3\times N} \to \mathbb{R}^K$, such that for an input point cloud $\mathbf{P} \in \mathbb{R}^{3\times N}$, $\phi(\mathbf{P})$ produces a K-dimensional vector descriptor. The function ϕ applies a Multi-Layer Perceptron (MLP) to each 3D point in $\mathbf{P}$, such that the final output dimension of each point is K. Then a symmetric pooling function, such as maximum or average, is applied, resulting in the K-dimensional global descriptor.

Let $\mathbf{P}_\mathcal{T}$, $\mathbf{P}_\mathcal{S}$ be template and source point clouds respectively. We will seek to find the rigid-body transform $\mathbf{T} \in SE(3)$ which best aligns source $\mathbf{P}_\mathcal{S}$ to template $\mathbf{P}_\mathcal{T}$.

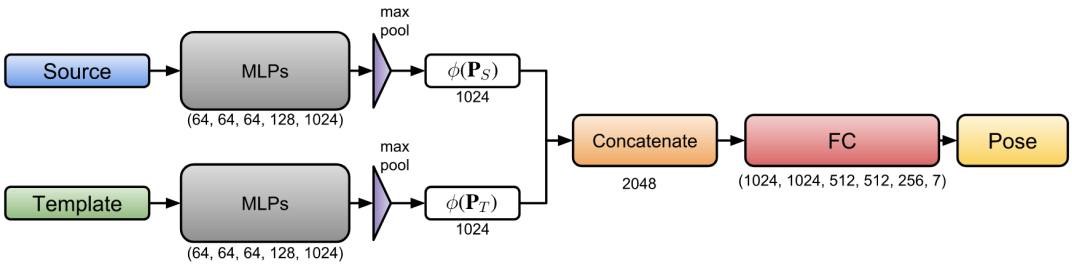


Figure 4.1: PCRNet, consisting of five MLPs having size $(64, 64, 64, 128, 1024)$. The source and template point clouds are sent as input through a twin set of MLPs, arranged in a Siamese architecture. Using a max-pooling function, we obtain global features. These features are concatenated and provided as an input to five fully connected layers having size $(1024, 1024, 512, 512, 256)$, and an output layer of size 7. The first three output values represent the translation and the last four, after normalization, represent the rotation quaternion.

4.1.1 PCRNet

This section introduces the PCRNet architecture. A block diagram of the architecture is shown in Fig. 4.1. The model consists of five multi-layered perceptrons (MLPs) similar to the PointNet architecture having size $(64, 64, 64, 128, 1024)$. The MLPs are arranged similar to a Siamese architecture [41]. Both source $\mathbf{P}_\mathcal{S}$ and template $\mathbf{P}_\mathcal{T}$ are given as input to the MLPs, and a symmetric max-pooling function is used to find the global feature vectors $\phi(\mathbf{P}_\mathcal{S})$ and $\phi(\mathbf{P}_\mathcal{T})$. Weights are shared between MLPs used for source and template.

The global features are concatenated and given as an input to a number of fully connected layers. In this work, we choose five fully connected layers, as they seemed to be sufficient enough for robust performance. In initial experiments, using lesser number of FC layers, the performance of the network was strictly worse.

The FC layer shown by the red block in Fig. 4.1 has five hidden layers of size $(1024, 1024, 512, 512, 256)$ and an output layer of size 7 whose parameters will represent the estimated transformation $\mathbf{T}$. The first three of the output values we use to represent the translation vector $\mathbf{t} \in \mathbb{R}^3$ and last four represents the rotation quaternion $\mathbf{q} \in \mathbb{R}^4$, $\mathbf{q}^T\mathbf{q} = 1$. In this way, the transformation $\mathbf{T}$ which aligns $\phi(\mathbf{P}_\mathcal{S})$ and $\phi(\mathbf{P}_\mathcal{T})$ is estimated with a single forward pass, or single-shot, through the network. The single-shot design lends itself particularly well to high-speed applications, which will be discussed further in Section 4.2.3.

4.1.2 Iterative PCRNet

In this section, we present a network with an iterative scheme similar to ICP and Lucas-Kanade for image alignment as shown in Fig. 4.2. We retain the structure of PCRNet, but modify the number of layers. For the iterative implementation, we use three hidden FC layers having size $(1024, 512, 256)$, and an output layer of size 7. Also, we include an additional dropout layer before the output layer, to avoid over-

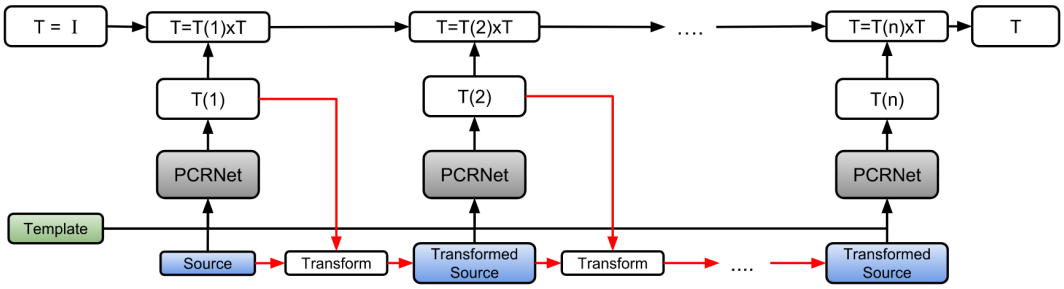


Figure 4.2: Iterative PCRNet, which uses a modified form of PCRNet described in Fig. 4.1 and iteratively improves the estimate of the PCRNet. In the first iteration, the source and template point clouds are given to PCRNet which predicts an initial misalignment $\mathbf{T}_1$. The source point cloud is transformed using $\mathbf{T}_1$ and the original template are given as input to the PCRNet, in the next iteration. After performing n iterations, we combined the poses from each iteration to find the overall transformation between the original source and template.

fitting. We empirically observe that introducing iterations allows us to use lesser number of hidden layers compared to PCRNet, and yet obtain robust performance.

In the first iteration, the original source and template point clouds are given to PCRNet, which predicts an initial misalignment $\mathbf{T}_1$ between them. For the next iteration, $\mathbf{T}_1$ is applied to the source point cloud and then the transformed source and the original template point clouds are given as input to the PCRNet. After performing n iterations, we find the overall transformation between the original source and template point clouds by combining all the poses in each iteration:

$$\mathbf{T} = \mathbf{T}_n \times \mathbf{T}_{n-1} \times \cdots \times \mathbf{T}_1. \tag{4.1}$$

4.1.3 PointNetLK

In the case of PointNetLK, we seek to learn ϕ such that, for the best aligning $\mathbf{T}$, we have $\phi(\mathbf{P}_\mathcal{T}) = \phi(\mathbf{T} \cdot \mathbf{P}_\mathcal{S})$. We formulate an optimization as follows. Let $\mathbf{P}_\mathcal{T}$, $\mathbf{P}_\mathcal{S}$ be template and source point clouds respectively. We will seek to find the rigid-body transform $\mathbf{T} \in SE(3)$ which best aligns source $\mathbf{P}_\mathcal{S}$ to template $\mathbf{P}_\mathcal{T}$. The transform $\mathbf{T}$ will be represented by an exponential map as follows:

$$\mathbf{T} = \exp\left(\sum_i \xi_i \mathbf{G}_i\right) \qquad \boldsymbol{\xi} = (\xi_1, \xi_2, ..., \xi_6)^T, \tag{4.2}$$

where $\mathbf{G}_i$ are the generators of the exponential map with twist parameters $\boldsymbol{\xi} \in \mathbb{R}^6$. The 3D point cloud alignment problem can then be described as finding $\mathbf{T}$ such that $\phi(\mathbf{P}_\mathcal{T}) = \phi(\mathbf{T} \cdot \mathbf{P}_\mathcal{S})$, where we use the shorthand $(\cdot)$ to denote transformation of $\mathbf{P}_\mathcal{S}$ by rigid transform $\mathbf{T}$. This equation is analogous to the quantity being optimized in the classical LK algorithm for 2D images, where the source image is warped such that the pixel intensity differences between the warped source and

template are minimized. It is worth noting that we do not include the T-net in our PointNet architecture, since its purpose was to transform the input point cloud in order to increase classification accuracy [68]. However, we instead use the LK layer to estimate the alignment, and the T-net is unnecessary.

Another key idea that we can borrow from the LK algorithm is the Inverse Compositional (IC) formulation [7]. The IC formulation is necessitated by the fact that the traditional LK algorithm has a high computational cost for each iteration of the optimization. This cost comes from the re-computation of an image Jacobian on the warped source image, at each step of the optimization. The insight of the IC formulation is to reverse the role of the template and source: at each iteration, we will solve for the incremental warp update to the template instead of the source, and then apply the *inverse* of this incremental warp to the source. By doing this, the Jacobian computation is performed for the template instead of the source and happens only once before the optimization begins. This fact will be more clearly seen in the following derivation of the warp update.

Restating the objective, we seek to find $\mathbf{T}$ such that $\phi(\mathbf{P}_\mathcal{T}) = \phi(\mathbf{T} \cdot \mathbf{P}_\mathcal{S})$. To do this, we will derive an iterative optimization solution.

With the IC formulation in mind, we take an inverse form for the objective:

$$\phi(\mathbf{P}_\mathcal{S}) = \phi(\mathbf{T}^{-1} \cdot \mathbf{P}_\mathcal{T}) \tag{4.3}$$

The next step is to linearize the right-hand side of (4.3):

$$\phi(\mathbf{P}_\mathcal{S}) = \phi(\mathbf{P}_\mathcal{T}) + \frac{\partial}{\partial \boldsymbol{\xi}} \left[\phi(\mathbf{T}^{-1} \cdot \mathbf{P}_\mathcal{T})\right] \boldsymbol{\xi} \tag{4.4}$$

Where we define $\mathbf{T}^{-1} = \exp(-\sum_i \xi_i \mathbf{G}_i)$.

We will denote the Jacobian $\mathbf{J} = \frac{\partial}{\partial \boldsymbol{\xi}} \left[\phi(\mathbf{T}^{-1} \cdot \mathbf{P}_\mathcal{T})\right]$, where $\mathbf{J} \in \mathbb{R}^{K \times 6}$ matrix. At this point, computing $\mathbf{J}$ would seem to require an analytical representation of the gradient for the PointNet function with respect to the twist parameters of $\mathbf{T}$. This analytical gradient would be difficult to compute and quite costly. The approach taken in the classical LK algorithm for ND images is to split the Jacobian using the chain rule, into two partial terms: an image gradient in the ND image directions, and an analytical warp Jacobian [7]. However, in our case this approach will not work either, since there is no graph or other convolutional structure which would allow taking gradients in x, y and z for our 3D registration case.

We instead opt to compute $\mathbf{J}$ using a stochastic gradient approach. Specifically, each column $\mathbf{J}_i$ of the Jacobian can be approximated through a finite difference gradient computed as

$$\mathbf{J}_i = \frac{\phi(\exp(-t_i \mathbf{G}_i) \cdot \mathbf{P}_\mathcal{T}) - \phi(\mathbf{P}_\mathcal{T})}{t_i} \tag{4.5}$$

Where t_i are infinitesimal perturbations of the twist parameters $\boldsymbol{\xi}$. This approach to computing $\mathbf{J}$ is what allows the application of the computationally efficient inverse compositional LK algorithm to the problem of point cloud registration using PointNet features. Note that $\mathbf{J}$ is computed only once, for the template point cloud, and does not need to be recomputed as the source point cloud is warped during iterative alignment.

For each column $\mathbf{J}_i$ of the Jacobian, only the i^{th} twist parameter has a non-zero value t_i. Theoretically, t_i should be infinitesimal so that $\mathbf{J}$ is equal to an analytical derivative. In practice, we find empirically that setting t_i to some small fixed value over all iterations yields the best result.

We can now solve for $\boldsymbol{\xi}$ in (4.4) as

$$\boldsymbol{\xi} = \mathbf{J}^{+} \left[\phi(\mathbf{P}_{\mathcal{S}}) - \phi(\mathbf{P}_{\mathcal{T}}) \right] \tag{4.6}$$

Where $\mathbf{J}^{+}$ is a Moore-Penrose inverse of $\mathbf{J}$.

In summary, the iterative algorithm consists of a looping computation of the optimal twist parameters using (4.6), and then updating the source point cloud $\mathbf{P}_{\mathcal{S}}$ as

$$\mathbf{P}_{\mathcal{S}} \leftarrow \Delta\mathbf{T} \cdot \mathbf{P}_{\mathcal{S}} \qquad \Delta\mathbf{T} = \exp\left(\sum_i \xi_i \mathbf{G}_i \right) \tag{4.7}$$

The final estimate $\mathbf{T}_{est}$ is then the composition of all incremental estimates computed during the iterative loop:

$$\mathbf{T}_{est} = \Delta\mathbf{T}_n \cdot \ldots \cdot \Delta\mathbf{T}_1 \cdot \Delta\mathbf{T}_0 \tag{4.8}$$

The stopping criterion for iterations is based on a minimum threshold for $\Delta\mathbf{T}$. A graphical representation of our model is shown in Fig. 4.3.

4.1.4 Loss Functions

The loss function used during training for registration is an important consideration, and there may not be a single loss function which works equally well for training across all types of data (noise-free, noisy, partial view). We consider two loss functions, one which is data-dependent, and the other which is dependent only on the estimated transforms.

A data-dependent loss function is targeted at minimizing the point-point distance between two point clouds. One such loss function we experiment with is the Earth Mover Distance (EMD), which was introduced in [28]. The Earth Mover Distance is

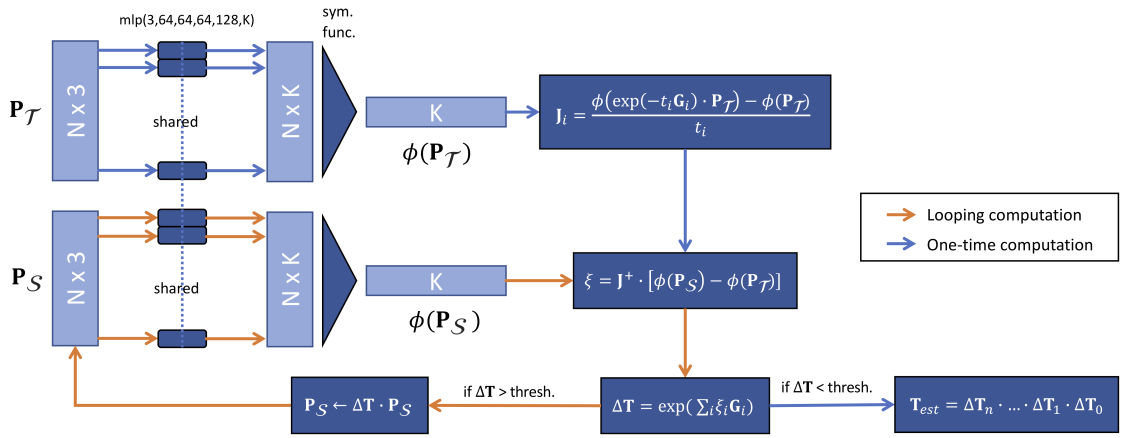


Figure 4.3: The PointNetLK architecture. Point cloud inputs source $\mathbf{P}_\mathcal{S}$ and template $\mathbf{P}_\mathcal{T}$ are passed through a shared MLP, and a symmetric pooling function, to compute the global feature vectors $\phi(\mathbf{P}_\mathcal{S})$ and $\phi(\mathbf{P}_\mathcal{T})$. The Jacobian $\mathbf{J}$ is computed once using $\phi(\mathbf{P}_\mathcal{T})$. The optimal twist parameters are found, which are used to incrementally update the pose of $\mathbf{P}_\mathcal{S}$, and then the global feature vector $\phi(\mathbf{P}_\mathcal{S})$ is recomputed. During training, a loss function is used which is based on the difference in the estimated rigid transform and the ground truth transform.

a bijective point-point distance between template and source point clouds, defined as

$$\text{EMD}(\mathbf{P}_\mathcal{S}^{\text{est}}, \mathbf{P}_\mathcal{T}) = \min_{\psi:\mathbf{P}_\mathcal{S}^{\text{est}} \to \mathbf{P}_\mathcal{T}} \frac{1}{|\mathbf{P}_\mathcal{S}^{\text{est}}|} \sum_{x \in \mathbf{P}_\mathcal{S}^{\text{est}}} \|x - \psi(x)\|_2, \tag{4.9}$$

where $\mathbf{P}_\mathcal{T}$ is the template point cloud and $\mathbf{P}_\mathcal{S}^{\text{est}}$ is the source point cloud $\mathbf{P}_\mathcal{S}$, transformed by the estimated transformation $\mathbf{T}$. This function finds a bijection ψ and minimizes the distance between corresponding points based on ψ.

A transformation-dependent loss function is targeted at minimizing the difference between the estimated transform $\mathbf{T}_{est}$ and the ground truth transformation $\mathbf{T}_{gt}$. This could be expressed as the Mean Square Error (MSE) between the twist parameters $\boldsymbol{\xi}_{est}$ and $\boldsymbol{\xi}_{gt}$. Instead, we use

$$||(\mathbf{T}_{est})^{-1} \cdot \mathbf{T}_{gt} - \mathbf{I}_4||_F, \tag{4.10}$$

which is more efficient to compute as it does not require matrix logarithm operation during training, and follows in a straightforward way from the representation of $\mathbf{T}_{est}, \mathbf{T}_{gt} \in SE(3)$.

For the following experiments, we have found that the PCRNet architectures are more easily trained using the data-dependent loss function, while PointNetLK is more easily trained using the transformation-dependent loss function. We attribute this to the fact that PCRNet must learn the registration task from scratch in the

final FC layers and is more object-specific since registration is entirely data-driven, and therefore benefits from the more object-specific EMD loss which is a function of object shape as well. This is opposed to PointNetLK, which does not learn the registration task but rather only fine-tunes the PointNet latent representations. In this more general framework, it may be better to take a data-independent loss approach so that the PointNet latent representations may be unspecific to objects as possible.

4.2 Experiments

In this section, we compare performance of our networks on various test data. ICP is used frequently as a basline (point-to-point implementation).

We experiment with synthetic data of multiple object categories, a specific object category, or a specific object, to exhibit various traits of the 3 proposed networks. We show how PCRNet variants are generally better suited to object-specific and category-specific scenarios, while PointNetLK is better-suited to general registration tasks with no assumption on the test data.

We experiment on synthetic data with added noise, repeating the object specificity experiments from above. We find that PCRNet architectures generally handle noisy data better PointNetLK, most likely due to their increased model capacity.

Based on the observation that PointNetLK is less object-specific, we also extend PointNetLK to perform partially visible registration on multiple synthetic categories, and also on partial real-world Kinect data.

We allow a maximum of 20 iterations for both iterative PCRNet and PointNetLK while performing tests, while the maximum iterations for ICP (point-to-point) was chosen as 100. We found these cutoffs to be more than enough in a vast majority of cases. In addition to maximum iterations, for all algorithms we also use the convergence criteria

$$\left\| \mathbf{T}_i \mathbf{T}_{i-1}^{-1} - \mathbf{I} \right\|_F < \epsilon, \tag{4.11}$$

where $\mathbf{T}_i, \mathbf{T}_{i-1} \in SE(3)$ are the transformations predicted in current and previous iterations, and the value of ϵ is chosen to be 10^{-7}.

In order to evaluate the performance of the registration algorithms, a metric we use is area under the curve (AUC). Plots showing success ratio versus success criteria on rotation error (in degrees) are generated for ICP, iterative PCRNet and PointNetLK. Fig. 4.4 shows examples of these curves. The area below the curves in these plots, divided by 180 to normalize between 0 and 1, is defined as AUC [1]. We measure the misalignment between predicted transformation and ground truth transformation

[1] We define success ratio as the number of test cases having rotation error less than success criteria.

and express it in axis-angle representation and we report the angle as rotation error. As for the translation error, we report the L2 norm of the difference between ground truth and estimated translation vectors.

4.2.1 Synthetic Data

We use models from the ModelNet40 dataset [89] for the following experiments. We sample 1000 points from the model faces uniformly weighted on face area, template point clouds are normalized into a unit box, and their mean is shifted to origin. We randomly choose 5070 transformations with Euler angles in the range of $[-45°, 45°]$ and translation values in the range of [-1, 1] units. We apply these rigid transformations on the template point clouds to generate the source point clouds.

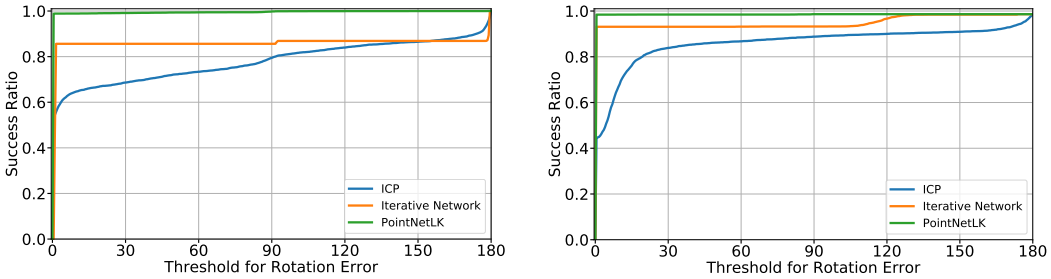
For this experiment, Iterative PCRNet and PointNetLK are trained on 20 different object categories from ModelNet40 with total of 5070 models. We do not use one-shot PCRNet for this experiment, since we found it has limited success with multi-category or single-category registration, and is better-suited to single-model registration. We perform tests using 100 models chosen from 5 object categories which are not in training data (referred to as unseen categories) with no noise in point clouds. Results for this experiment are shown in Fig. 4.4. Also shown are results from training the networks with objects from the same category (car) as the test-set. Here, we see a large improvement for Iterative PCRNet, while ICP and PointNetLK retain similar performance.

These results emphasize that the Iterative PCRNet, when retrained with category-specific data, provides improvement over ICP and is comparable to PointNetLK. However, PointNetLK shows better generalization than Iterative PCRNet across various object categories. We attribute this to the inherent limitation of the learning capacity of Iterative PCRNet to large shape variations, while PointNetLK only has to learn the PointNet representation rather than the task of alignment. However, in the next set of experiments, we demonstrate the advantages of PCRNet-based architectures over PointNetLK and other baselines, especially in the presence of noisy data.

4.2.2 Noisy Synthetic Data

In order to evaluate the robustness of our networks to noise, we perform experiments with Gaussian noise in the source points. For our first test, we use the same training dataset as described in the previus section Sec. 4.2.1. We sample noise from Gaussian distribution for each point in source point cloud with 0 mean and a standard deviation varying in the range of 0 to 0.04 units. For these results, we trained Iterative PCRNet and PointNetLK with noisy source point clouds using 20 different object categories and a total of 5070 models.

During testing, we compare ICP, PointNetLK and Iterative PCRNet. Fig. 4.5a shows results for noisy point cloud testing on **multiple object categories**. We



(a) Multiple categories, no noise.

(b) Single category (car), no noise.

Figure 4.4: Results for Sec. 4.2.1. PointNetLK has an advantage over PCRNet-based architectures for no-noise registration scenarios. PointNetLK performs better across multiple object categories, and has remarkable ability to handle large initial perturbation. When training and testing on a single object category (car), we find that Iterative PCRNet has improved performance although still cannot match PointNetLK.

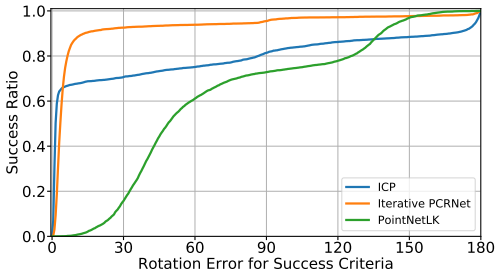
observe that Iterative PCRNet has a higher number of successful test cases with smaller rotation error as compared to ICP and PointNetLK, which shows that Iterative PCRNet is robust to Gaussian noise. PointNetLK performs the worst and is very sensitive to noisy data, and in fact, during training PointNetLK appears to have difficulty converging for noisy training data. The results emphasize that Iterative PCRNet works quite well in the presence of noise in the source data, and across multiple categories.

For the second test, we used the dataset as described in Sec. 4.2.1 and added Gaussian noise in source point clouds as described above. We train the networks on a **specific object category** (car) and test them on the same category, using 150 models of cars. The result in Fig. 4.5b shows that Iterative PCRNet performs the best and has the highest number of successful test cases.

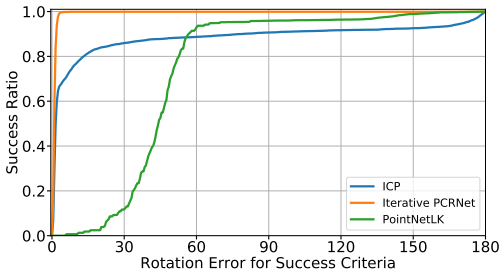
We compare the success ratio of networks when training and testing on only **one noisy car model** (see Fig. 4.5c). Iterative PCRNet once again exhibits a high success ratio, which is better than ICP and PointNetLK. Finally, we compare PCRNet that is trained without noise and tested on noisy data, with ICP and PointNetLK. While not being as good as ICP, the result is still competitive, and performs better than PointNetLK (See Fig. 4.5d).

We show some qualitative results of the performance of Iterative PCRNet in Fig. 4.6.

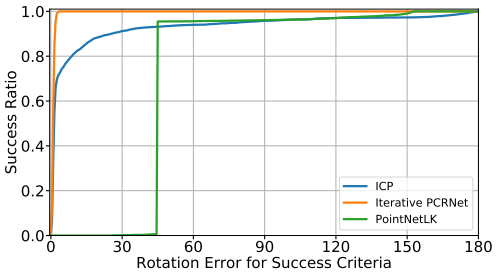
Fig. 4.7 shows success ratio versus change in the amount of noise added to source point clouds during testing. Both Iterative PCRNet and PointNetLK are trained on multiple object categories with Gaussian noise having a maximum standard deviation of 0.04. We observe a sudden drop in the PointNetLK performance as the standard deviation for noise increases above 0.02. On the other hand, iterative PCRNet performs best in the neighbourhood of the noise range that it was trained on (0.02-0.06), and produces results comparable to ICP beyond that noise level. This shows that PCRNet-based network are more robust to noise as compared to



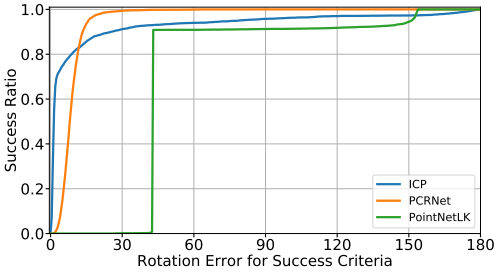
(a) Training and testing: Multiple object categories with noise.



(b) Training and testing: Multiple models of a category with noise

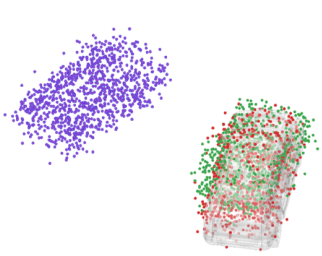


(c) Training and testing: Only one model with noise

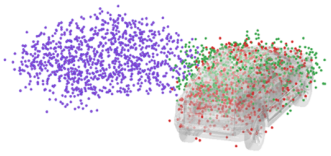


(d) Trained on one model without noise and tested on data with noise

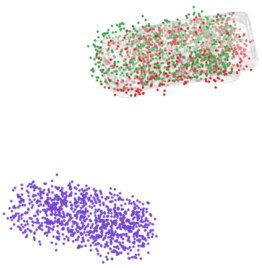
Figure 4.5: Results for Section 4.2.2, noise testing. The y-axis is the ratio of experiments that are successful and the x-axis shows value of the maximum rotation error that qualifies the estimation to be a success. (a), (b) and (c) show results for comparisons of Iterative PCRNet with ICP and PointNetLK using three different types of datasets. We observe superior performance of Iterative PCRNet in most cases as compared to ICP and PointNetLK. We observe that PointNetLK is quite sensitive to noise in this range (up to 0.04 std. dev.) and has difficulty converging during training. (d) PCRNet, which has not seen noise during training, is tested with noisy data and shows good performance while being faster than ICP and PointNetLK. Speed considerations are discussed in Sec. 4.2.3.



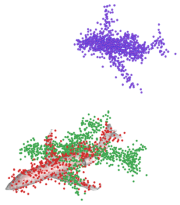
(a) Trained on one car. Iterative PCRNet: (2.14°, 0.0056)



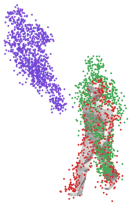
(b) Trained on multiple cars. Iterative PCRNet: (2.14°, 0.0056).



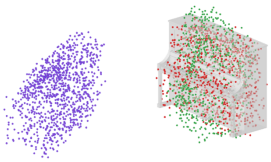
(c) Trained on multiple categories. Iterative PCRNet: (3.07°, 0.0107).



(d) Trained on multiple categories. Iterative PCRNet: (0.34°, 0.0048). ICP: (43.62°, 0.2564).



(e) Trained on multiple categories. Iterative PCRNet: (5.55°, 0.0042). ICP: (45.15°, 0.1767).



(f) Trained on multiple categories. Iterative PCRNet: (5.96°, 0.0035). ICP: (75.02°, 0.0683).

Figure 4.6: Qualitative results for Section 4.2.2, with final (rot., trans.) errors reported. For each example, the template is shown by a gray CAD model, purple points show initial position of source, red points show converged results of iterative PCRNet trained on data with noise, and green points show results of ICP. (a), (b), (c), (d) and (e) show the results for objects from seen categories, while (f) shows results of unseen category.

PointNetLK.

4.2.3 Computational Efficiency

In this experiment, we use a testing dataset with only one model of car from ModelNet40 dataset, with Gaussian noise in the source data. We apply 100 randomly chosen transformations with Euler angles in range of $[-45°, 45°]$ and translation values in range of [-1, 1] units. The networks are all trained using multiple models of same category (i.e. car). We compared the performance of Iterative PCRNet, PCRNet, PointNetLK, ICP and GO-ICP, as shown in Table 4.1. We report the rotation and translation error after registration, computation time (all on CPU), and the AUC of the success ratio curve.

The results demonstrate that Go-ICP converges to a globally optimal solution with a very small rotation error and translation error, but the time taken is three orders of magnitude more than Iterative PCRNet and five orders of magnitude more than PCRNet. The AUC value of Go-ICP is 1, meaning that it has converged in all test cases, while Iterative PCRNet has the second best AUC value. This experiment shows how the Iterative PCRNet is similar to Go-ICP in terms of accuracy,

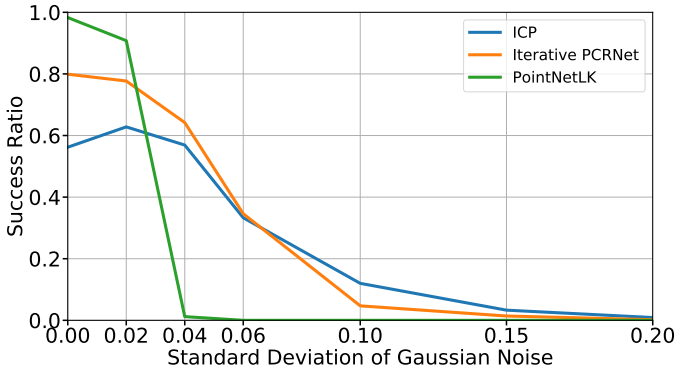


Figure 4.7: Results for Sec. 4.2.2. Iterative PCRNet and PointNetLK are trained on multiple object categories with Gaussian noise, having maximum value of std. dev. equal to 0.04. The x-axis shows different values of standard deviation in noise used in testing. PointNetLK is most accurate in the absence of noise, while iterative PCRNet is robust to noise around the levels that it has observed during training (0.02-0.06). PointNetLK suffers at higher noise level, likely due to the fact that the PointNet representation can be sensitive to outliers without the additional ability to learn corrective alignment in later layers.

Table 4.1: Results from Section 4.2.3. Accuracy and computation time comparisons for registering noisy data (car category). Notice that both PCRNet models achieve nearly the same AUC as Go-ICP while being orders of magnitude faster. (All computations on CPU).

	Rot. Err. (deg)		Trans. Err.		Time (ms)		AUC
Algorithm	Mean	SD	Mean	SD	Mean	SD	
PCRNet	8.8	4.8	0.0077	0.0008	19.7	0.30	0.95
Iterative PCRNet	1.0	2.6	0.0085	0.0024	307	81	0.99
PointNetLK	51.8	29.6	0.8783	0.0054	256	38	0.70
ICP [12]	11.8	31.9	0.0282	0.0392	407	128	0.93
GO-ICP [91]	0.4	0.2	0.0016	0.0007	$2.7 \cdot 10^5$	$1.5 \cdot 10^5$	1.00

but computationally much faster, allowing for use in many practical applications. Further, while PCRNet is less accurate than Iterative PCRNet and Go-ICP, the accuracy may be good enough as a pre-aligning step in applications such as object detection and segmentation as was explored in [96].

4.2.4 Partially Visible Data

As PointNetLK is the architecture with the greatest generalizability across object categories, we extend it for partial visibility experiments. In the real world, oftentimes the template is a full 3D model and the source a 2.5D scan. One approach in this case is to input the 2.5D source and 3D template directly into an alignment algorithm and estimate the correspondence and the alignment. A second approach is to use an initial estimate of camera pose with respect to the 3D model to sample

visible points on the model, which can be compared with the 2.5D scan. The camera pose can be iteratively updated until the visible points on the 3D model match the 2.5D scan.

We take the latter approach for testing PointNetLK, because the cost function $\phi(\mathbf{P}_{\mathcal{T}}) - \phi(\mathbf{T} \cdot \mathbf{P}_{\mathcal{S}})$ can tend to be large for input point clouds which are a 3D model and 2.5D scan. Instead, it makes more sense to sample visible points from the 3D model first based on an initial pose estimate, so that the inputs to PointNetLK are both 2.5D. This way, a correct final alignment is more likely to lead to the cost function $\phi(\mathbf{P}_{\mathcal{T}}) - \phi(\mathbf{T} \cdot \mathbf{P}_{\mathcal{S}})$ being close to zero.

Sampling visible points is typically based on simulating a physical sensor model for 3D point sensing, which has a horizontal and vertical field-of-view, and a minimum and maximum depth [56, 26]. We adapt ModelNet40 data for partially visible testing using a simplistic sensor model as follows. We sample faces from ModelNet shapes to create a template, place the template into a unit box $[0, 1]^3$, set the template equal to the source, and warp the source using a random perturbation. Next we translate the source and template both by a vector of length 2 in the direction $[1, 1, 1]^T$ from the origin. Then we assign the visible points of the template $\mathbf{P}^v_{\mathcal{T}}$ as those satisfying $(\mathbf{P}_{\mathcal{T}} + 2 \cdot [1, 1, 1]^T) < \text{mean}(\mathbf{P}_{\mathcal{T}} + 2 \cdot [1, 1, 1]^T)$. This operation can be thought of a placing a sensor at the origin which faces the direction $[1, 1, 1]^T$ and samples points on the 3D models which lie in front of it, up to a maximum depth equal to the mean of the point cloud. We set the visible source points $\mathbf{P}^v_{\mathcal{S}}$ in the same manner. This operation returns about half of the points in both template and source being visible for a given point cloud. We input the 2.5D visible point sets $\mathbf{P}^v_{\mathcal{T}}$ and $\mathbf{P}^v_{\mathcal{S}}$ into PointNetLK, allowing a single iteration to occur for estimation of the aligning transform $\mathbf{T}_{est}$. We then warp the original full source model $\mathbf{P}_{\mathcal{S}}$ using the single-iteration guess $\mathbf{T}_{est}$, and re-sample $\mathbf{P}^v_{\mathcal{S}}$. We repeat the single-iteration update and visibility re-sampling until convergence. We repeat the same procedure for testing ICP.

We test on the ModelNet40 test set, using random translation $[0, 0.3]$ for all tests. The results are shown in Fig. 4.9. Notably, we find that PointNetLK is able to learn to register objects using our sensor model, and generalizes well when the sensor model is applied to unseen object categories. Example template and source pairs for partially visible alignment are shown in Fig. 4.8 for ModelNet test dataset. We observe that our approach generalizes well to unseen shapes as shown in Fig. 4.10 which is generated from RGBD sensor data [6].

4.2.5 Same Category, Different Models

We hypothesize that PointNetLK features could be useful for registering point clouds of objects which are different but of the same category. An example of this is shown for two airplane models in Fig. 4.11. We would hope that the registration error for PointNetLK $|\phi(\mathbf{T} \cdot \mathbf{P}_{\mathcal{S}}) - \phi(\mathbf{P}_{\mathcal{T}})|$ is minimized when the airplane models, despite being different, are aligned in orientation. This reaffirms that the feature vectors learned for alignment are capturing a sense of the object category, and the canonical

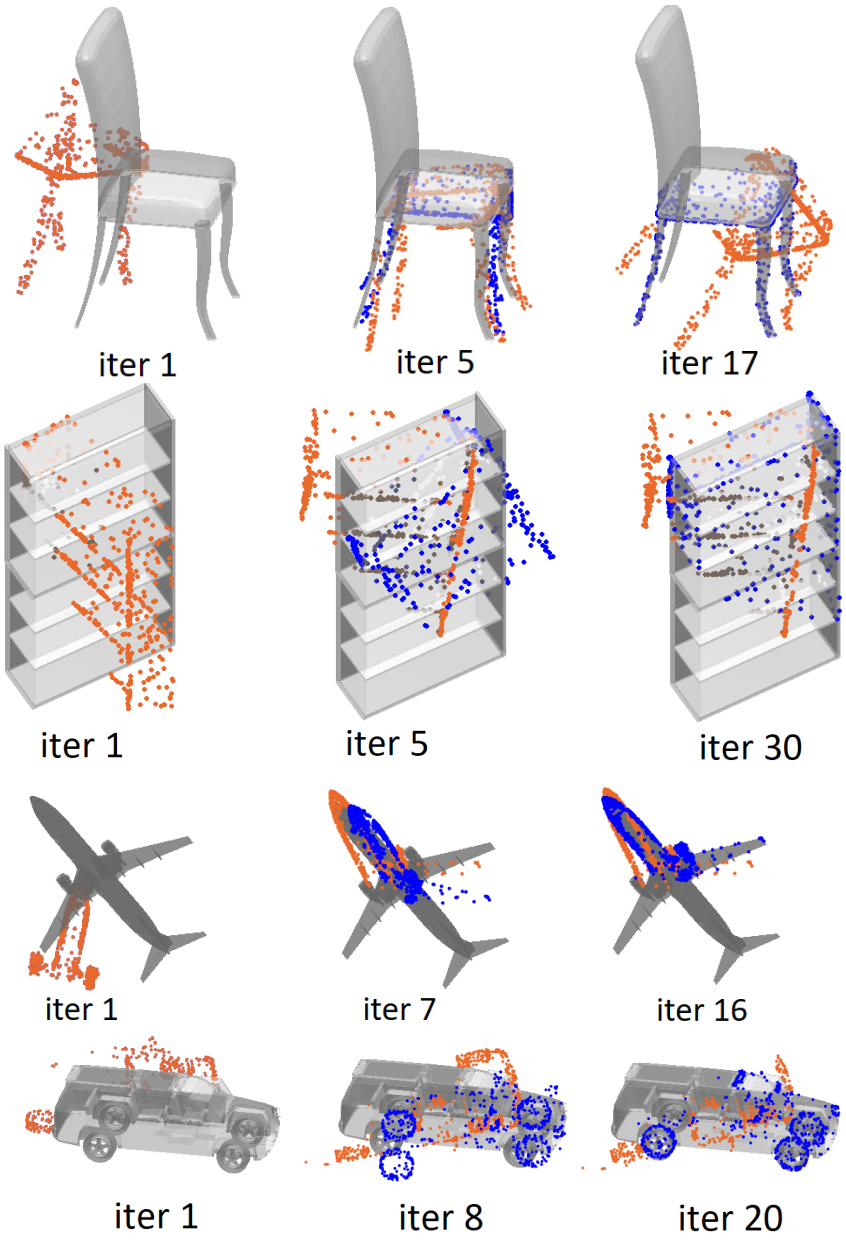


Figure 4.8: Results for Section 4.2.4. We test registration of partially visible ModelNet data, comparing ICP (shown by orange points), and PointNetLK trained on partially visible data (shown by blue points). We show that PointNetLK is able to learn the distribution of partial data seen with a simulated sensor model, and perform better at test-time.

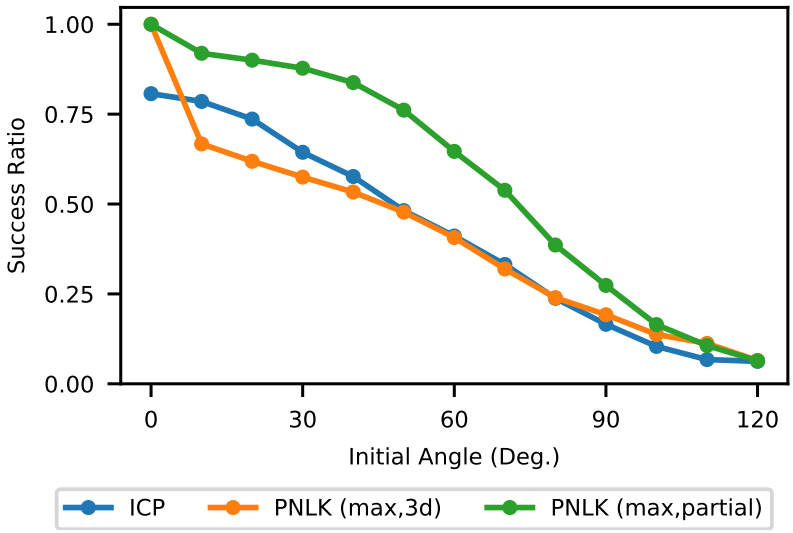


Figure 4.9: Results for Section 4.2.4. We test registration of partially visible ModelNet data, comparing ICP (blue), PointNetLK trained on 3D data (orange), and PointNetLK trained on partially visible data (green). Both PointNetLK models are trained with max pool. We use 20 object categories for testing, which are unseen during training. We find that training with partially visible data greatly improves performance, even surpassing ICP. A registration is counted as successful if the final alignment rotation error is less than 5 degrees and translation error is less than 0.01. Notice that PointNetLK has perfect performance at zero initial angle since we subtract the mean of each point cloud, whereas ICP does not.

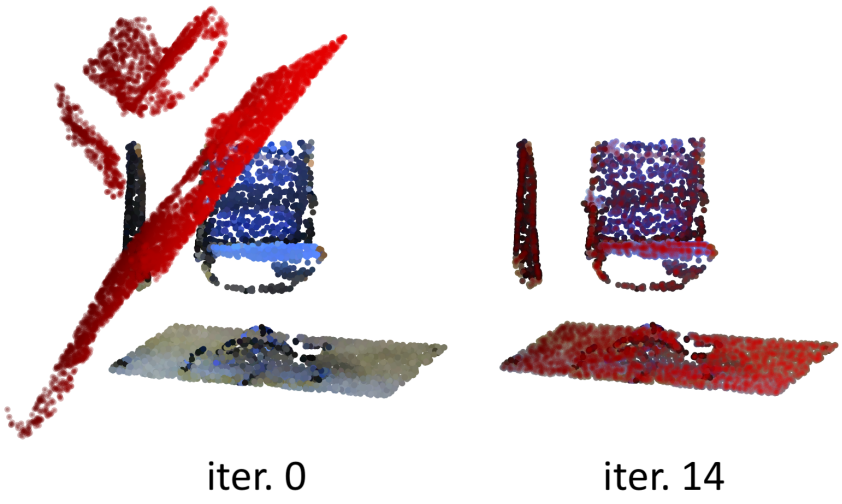


Figure 4.10: Results for Sec. 4.2.4. Registration of raw indoor RGBD scan from Stanford S3DIS [6] with PointNetLK. The red cloud represents the full 3D template, while the colored cloud represents the partial source scan to be registered. We use a simulated sensor model as described in Sec. 4.2.4.

orientation of that object. The network used for this experiment is trained using max pool on full 3D models. We find that in many cases, such as in the airplane example of Fig. 4.11, the PointNetLK cost function is globally minimized when the correct orientation is attained, while the ICP cost function is not necessarily minimized. In practice, this approach could work particularly well to identify the correct orientation of objects within a category if the orientation is known up to one or two axes of rotation.

4.3 Chapter Discussion

In this chapter, we have presented several methods for point cloud registration using deep networks. These methods are each built upon the original PointNet architecture, allowing our registration networks to process the point clouds directly as opposed to through voxelization or other representations.

Some key takeaways from our analysis of deep registration methods are as follows:

- PointNetLK performs well in cases with low amounts of Gaussian-like noise and when the template and source have similar sampling, and can both handle large initial perturbations and small, fine alignments. When more noise is present, PCRNet is a better option, likely because of the ability to learn the registration task as opposed to PointNetLK.
- When the object category or specific object for registration is known, then PCRNet has an advantage over PointNetLK. This is due to the ability to learn object-specific registration representations from data.
- Our PointNet registration methods are competitive or better than ICP in many cases, with comparable or less computation time even on CPU thanks to the efficient computation of PointNet descriptors and lack of costly correspondence step. Further, we can achieve results competitive with Go-ICP for a fraction of the computational cost.
- For PointNetLK, we have shown the ability to learn better performance for partially visible data, assuming that the sensor model is known during the training phase. Further, we have shown that the network trained with the sensor model on synthetic data (ModelNet40 [89]) can generalize to partially visible data from real sensors (S3DIS [6]).
- For PointNetLK, we have shown the potential for registering different objects of the same category. This has potential applications as part of a larger object classification or semantic segmentation pipeline, where objects can be registered to a canonical position to make object classification easier as was explored in [96].
- An interesting extension of this work would be to use descriptors from the PointNet++ network [69]. PointNet++ extends the global descriptor of PointNet, concatenating additional descriptors for local parts of a point cloud as

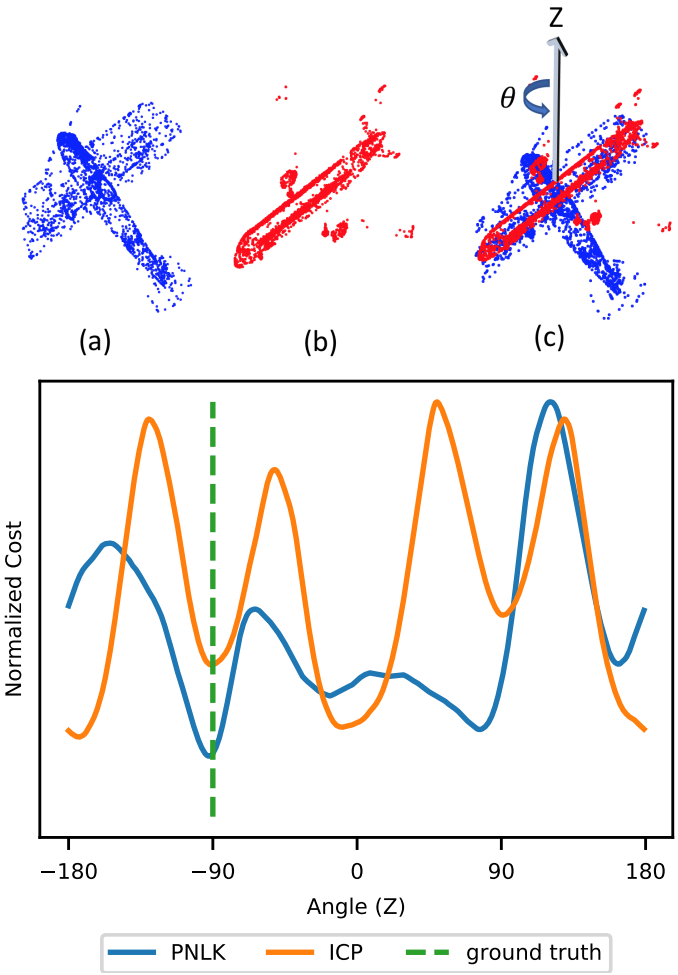


Figure 4.11: Results for Section 4.2.5. PointNetLK can achieve a global minimum when two different objects of the same category have the same orientation, whereas ICP can fail. We use two different airplane models from ModelNet40, a biplane (a) and a jetliner (b). (c) shows the initial (incorrect) configuration for alignment, where the centroids each model are at the same location. The jetliner is then rotated about the Z-axis through its centroid. The cost function for standard ICP and PointNetLK during this rotation are plotted. The airplanes have the same orientation at $-90°$ (ground truth). PointNetLK has a global minimum here, whereas ICP has global minimum at $180°$. Points are sampled from object vertices.

well. These local descriptors may enable a multi-scale registration approach similar to that which is used frequently in image registration applications. In such an approach, a coarse registration could be done on global descriptors, followed by fine registration on progressively more local descriptors.

Chapter 5

Conclusion

This thesis has presented an exploration of registration techniques which utilize deep learning. In the 2D case, we have shown the applicability of state-of-the-art deep features for registration of UAV and satellite imagery for the task of UAV localization. In the 3D case, we have shown how novel deep architectures can be developed for point cloud registration, and how these architectures can lead to impressive registration performance even on data which is far from the training distribution.

5.1 Benefits of Learning for Registration

Deep methods for registration continue to be an active topic of research, as we try to ascertain the value of learning in registration. In registration, we have many baseline methods which do not utilize machine learning and do not require large datasets ahead of test time. It is therefore important to ask, what benefits have been gained from adopting a machine learning approach for the classical problems of registration?

A definite conclusion that we can draw about registration with deep learning, is that it offers selectivity in the salient features for registration. Specifically, we can train networks which place **less weight on certain visual appearance characteristics** of template and source, and **more weight on a shape similarities**. For instance, in the 2D case, we learned invariance to visual style (seasons, time of day) and retained saliency of shape (buildings or other landmarks). In the 3D case, we learned invariance to appearance variations such as noise, partiality, or even inter-class variation, and retained saliency on overall object shape or object category to perform successful registration. Indeed, this selectivity of shape over appearance is often key in these and other registration problems, and should be considered a key benefit of deep learning methods for registration.

Among other benefits of learning for registration, we have also seen a benefit of speed as compared to many baseline methods. There is no need of costly point correspondence and matching step as occurs in SIFT or ICP. This leads to deep

registration methods having $O(n)$ runtime as opposed to $O(n^2)$ in the number of salient features. Further, deep methods are differentiable and can thus be integrated into larger learning pipelines.

5.2 Future Work

As deep learning continues to penetrate classical robotics problems such as Simultaneous Localization and Mapping (SLAM), we will see increased interest in learning pipelines for image and point cloud registration. Future work in this area is likely to include more investigation into the possibility of Bundle Adjustment (BA) with deep features, or differentiable BA which uses a final reconstruction to propagate loss back into a learnable system.

Another important question to be addressed in future work is whether the loss function for registration should be based solely upon the data, solely upon the transformation, or somewhere in between. For example, in the 2D case, loss functions could be based on the pixel-wise (or feature-wise) error, on the error measured between the 8 parameters of estimated and ground truth homography, or the loss we chose: the 4-corner loss, a proxy for the homography loss which is more conducive to stochastic gradient descent since it is linear in the warp parameters. In the 3D case, we have used both a Frobenius norm loss based on transformation parameters (for PointNetLK) and an EMD loss computed directly on the data (for PCRNet and Iterative PCRNet). These losses were chosen for each network specifically, and we found that the Frobenius norm loss generally did not work well for PCRNet, and the EMD loss did not work well for PointNetLK, however it is not abundantly clear why. This discrepancy likely stems from the fact that PCRNet learns the task of registration from scratch as opposed to PointNetLK, however we leave further investigation of these properties to future work.

Bibliography

[1] Hatem Alismail, Brett Browning, and Simon Lucey. "Direct visual odometry using bit-planes". In: *arXiv preprint arXiv:1604.00990* (2016).

[2] Hatem Alismail, Brett Browning, and Simon Lucey. "Photometric bundle adjustment for vision-based SLAM". In: *Asian Conference on Computer Vision*. Springer. 2016, pp. 324–341.

[3] Hatem Alismail, Brett Browning, and Simon Lucey. "Robust tracking in low light and sudden illumination changes". In: *2016 Fourth International Conference on 3D Vision (3DV)*. IEEE. 2016, pp. 389–398.

[4] Mikaela Angelina Uy and Gim Hee Lee. "PointNetVLAD: Deep point cloud based retrieval for large-scale place recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4470–4479.

[5] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. "NetVLAD: CNN architecture for weakly supervised place recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 5297–5307.

[6] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. "3D Semantic Parsing of Large-Scale Indoor Spaces". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[7] Simon Baker and Iain Matthews. "Lucas-Kanade 20 years on: A unifying framework". In: *International journal of computer vision* 56.3 (2004), pp. 221–255.

[8] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features". In: *European conference on computer vision*. Springer. 2006, pp. 404–417.

[9] Serge Belongie, Jitendra Malik, and Jan Puzicha. "Shape context: A new descriptor for shape matching and object recognition". In: *Advances in neural information processing systems*. 2001, pp. 831–837.

[10] Youcef Bentoutou, Nasreddine Taleb, Kidiyo Kpalma, and Joseph Ronsin. "An automatic image registration for applications in remote sensing". In: *IEEE transactions on geoscience and remote sensing* 43.9 (2005), pp. 2127–2137.

[11] Luca Bertinetto, Jack Valmadre, Joao F Henriques, Andrea Vedaldi, and Philip HS Torr. "Fully-convolutional siamese networks for object tracking". In: *European conference on computer vision*. Springer. 2016, pp. 850–865.

[12] Paul J Besl and Neil D McKay. "Method for registration of 3D shapes". In: *Sensor Fusion IV: Control Paradigms and Data Structures*. Vol. 1611. International Society for Optics and Photonics. 1992, pp. 586–607.

[13] Hakan Bilen and Andrea Vedaldi. "Weakly supervised deep detection networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2846–2854.

[14] Gary Bradski. "The opencv library". In: *Dr Dobb's J. Software Tools* 25 (2000), pp. 120–125.

[15] Hilton Bristow, Jack Valmadre, and Simon Lucey. "Dense semantic correspondence where every pixel is a classifier". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 4024–4031.

[16] Matthew Brown and David G Lowe. "Automatic panoramic image stitching using invariant features". In: *International journal of computer vision* 74.1 (2007), pp. 59–73.

[17] Jonathan Cacace, Alberto Finzi, Vincenzo Lippiello, Giuseppe Loianno, and Dario Sanzone. "Aerial service vehicles for industrial inspection: task decomposition and plan execution". In: *Applied Intelligence* 42.1 (2015), pp. 49–62.

[18] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. "BRIEF: Binary robust independent elementary features". In: *European conference on computer vision*. Springer. 2010, pp. 778–792.

[19] Che-Han Chang, Chun-Nan Chou, and Edward Y Chang. "Clkn: Cascaded lucas-kanade networks for image alignment". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2213–2221.

[20] Chin Seng Chua and Ray Jarvis. "Point signatures: A new representation for 3D object recognition". In: *International Journal of Computer Vision* 25.1 (1997), pp. 63–85.

[21] Ronan Collobert, Koray Kavukcuoglu, Clément Farabet, et al. "Torch7: A matlab-like environment for machine learning". In: *BigLearn, NIPS workshop*. Vol. 5. Granada. 2011, p. 10.

[22] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *international Conference on computer vision & Pattern Recognition (CVPR'05)*. Vol. 1. IEEE Computer Society. 2005, pp. 886–893.

[23] Amaël Delaunoy and Marc Pollefeys. "Photometric bundle adjustment for dense multi-view 3D modeling". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1486–1493.

[24] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[25] Ethan Eade. "Lie groups for 2D and 3D transformations". In: *URL http://ethaneade.com/lie. pdf, revised Dec* (2013).

[26] Benjamin Eckart, Kihwan Kim, and Kautz Jan. "EOE: Expected Overlap Estimation over Unstructured Point Cloud Data". In: *2018 International Conference on 3D Vision (3DV)*. IEEE. 2018, pp. 747–755.

[27] Gil Elbaz, Tamar Avraham, and Anath Fischer. "3D point cloud registration for localization using a deep neural network auto-encoder". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 4631–4640.

[28] Haoqiang Fan, Hao Su, and Leonidas J Guibas. "A point set generation network for 3D object reconstruction from a single image". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 605–613.

[29] Martin A Fischler and Robert C Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[30] Thomas Funkhouser, Patrick Min, Michael Kazhdan, Joyce Chen, Alex Halderman, David Dobkin, and David Jacobs. "A search engine for 3D models". In: *ACM Transactions on Graphics (TOG)* 22.1 (2003), pp. 83–105.

[31] Natasha Gelfand, Niloy J Mitra, Leonidas J Guibas, and Helmut Pottmann. "Robust global registration". In: *Symposium on geometry processing*. Vol. 2. 2005, p. 5.

[32] Georgios Georgakis, Srikrishna Karanam, Ziyan Wu, and Jana Kosecka. "Matching RGB Images to CAD Models for Object Pose Estimation". In: *arXiv preprint arXiv:1811.07249* (2018).

[33] Jared Glover, Gary Bradski, and Radu Bogdan Rusu. "Monte Carlo pose estimation with quaternion kernels and the Bingham distribution". In: *Robotics: Science and Systems*. Vol. 7. 2012, p. 97.

[34] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. "Unsupervised monocular depth estimation with left-right consistency". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 270–279.

[35] Guy Godin, Marc Rioux, and Rejean Baribeau. "Three-dimensional registration using range and intensity information". In: *Videometrics III*. Vol. 2350. International Society for Optics and Photonics. 1994, pp. 279–291.

[36] Maoguo Gong, Shengmeng Zhao, Licheng Jiao, Dayong Tian, and Shuang Wang. "A novel coarse-to-fine scheme for automatic image registration based on SIFT and mutual information". In: *IEEE Transactions on Geoscience and Remote Sensing* 52.7 (2014), pp. 4328–4338.

[37] Rafael C Gonzalez, Richard E Woods, et al. "Digital image processing". In: *Publishing house of electronics industry* 141.7 (2002).

[38] F Sebastian Grassia. "Practical parameterization of rotations using the exponential map". In: *Journal of graphics tools* 3.3 (1998), pp. 29–48.

[39] Christopher G Harris, Mike Stephens, et al. "A combined corner and edge detector." In: *Alvey vision conference*. Vol. 15. 50. Citeseer. 1988, pp. 10–5244.

[40] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[41] David Held, Sebastian Thrun, and Silvio Savarese. "Learning to track at 100 FPS with deep regression networks". In: *European Conference on Computer Vision*. Springer. 2016, pp. 749–765.

[42] Jarrod C Hodgson, Shane M Baylis, Rowan Mott, Ashley Herrod, and Rohan H Clarke. "Precision wildlife monitoring using unmanned aerial vehicles". In: *Scientific reports* 6 (2016), p. 22574.

[43] Peter J Huber. "Robust estimation of a location parameter". In: *Breakthroughs in statistics*. Springer, 1992, pp. 492–518.

[44] Nathan Jacobs, Nathaniel Roman, and Robert Pless. "Consistent temporal variations in many outdoor scenes". In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2007, pp. 1–6.

[45] Yoon Kim. "Convolutional neural networks for sentence classification". In: *arXiv preprint arXiv:1408.5882* (2014).

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[47] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, et al. "The digital Michelangelo project: 3D scanning of large statues". In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 2000, pp. 131–144.

[48] David G Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2 (2004), pp. 91–110.

[49] Yi Lu, Dominique Macias, Zachary S Dean, Nicole R Kreger, and Pak Kin Wong. "A UAV-mounted whole cell biosensor system for environmental monitoring applications". In: *IEEE Trans. Nanobiosci* 14 (2015), pp. 811–817.

[50] Bruce D Lucas and Takeo Kanade. "An iterative image registration technique with an application to stereo vision". In: *Proceedings of the 7th international joint conference on Artificial intelligence-Volume 2*. Morgan Kaufmann Publishers Inc. 1981, pp. 674–679.

[51] Laurens van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of machine learning research* 9.Nov (2008), pp. 2579–2605.

[52] Ameesh Makadia, Alexander Patterson, and Kostas Daniilidis. "Fully automatic registration of 3D point clouds". In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2006, pp. 1297–1304.

[53] Ezio Malis and Manuel Vargas. "Deeper understanding of the homography decomposition for vision-based control". PhD thesis. INRIA, 2007.

[54] Daniel Maturana and Sebastian Scherer. "VoxNet: A 3D convolutional neural network for real-time object recognition". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 922–928.

[55] Gerard Medioni and Ramakant Nevatia. “Matching images using linear features”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6 (1984), pp. 675–685.

[56] Ravish Mehra, Pushkar Tripathi, Alla Sheffer, and Niloy J Mitra. “Visibility of noisy point cloud data”. In: *Computers & Graphics* 34.3 (2010), pp. 219–230.

[57] Max Messinger and Miles Silman. “Unmanned aerial vehicles for the assessment and monitoring of environmental contamination: An example from coal ash spills”. In: *Environmental pollution* 218 (2016), pp. 889–894.

[58] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. “FastSLAM: A factored solution to the simultaneous localization and mapping problem”. In: *Aaai/iaai* 593598 (2002).

[59] Vinod Nair and Geoffrey E Hinton. “Rectified linear units improve restricted Boltzmann machines”. In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[60] Ahmed Nassar, Karim Amer, Reda ElHakim, and Mohamed ElHelw. “A Deep CNN-Based Framework for Enhanced Aerial Imagery Registration With Applications to UAV Geolocalization”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 1513–1523.

[61] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W Fitzgibbon. “Kinectfusion: Real-time dense surface mapping and tracking.” In: *ISMAR*. Vol. 11. 2011. 2011, pp. 127–136.

[62] Jun-nosuke Okamoto and Hiroto Shimazaki. “Land surveying performance of commercially-available inexpensive small UAV”. In: *Proceedings of 36th Asian Conference on Remote Sensing 2015 (ACRS 2015), Fostering Resilient Growth in Asia*. Citeseer. 2015.

[63] Sammy Omari, Pascal Gohl, Michael Burri, Markus Achtelik, and Roland Siegwart. “Visual industrial inspection using aerial robots”. In: *Applied Robotics for the Power Industry (CARPI), 2014 3rd International Conference on*. IEEE. 2014, pp. 1–5.

[64] Maks Ovsjanikov, Quentin Mérigot, Facundo Mémoli, and Leonidas Guibas. “One point isometric matching with the heat kernel”. In: *Computer Graphics Forum*. Vol. 29. 5. Wiley Online Library. 2010, pp. 1555–1564.

[65] George Papandreou, Liang-Chieh Chen, Kevin P Murphy, and Alan L Yuille. “Weakly-and semi-supervised learning of a deep convolutional network for semantic image segmentation”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1742–1750.

[66] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. “Automatic differentiation in PyTorch”. In: *NIPS-W*. 2017.

[67] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. “Frustum PointNets for 3D object detection from RGB-D data”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 918–927.

[68] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. "PointNet: Deep learning on point sets for 3d classification and segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 652–660.

[69] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. "PointNet++: Deep hierarchical feature learning on point sets in a metric space". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5099–5108.

[70] Juntong Qi, Dalei Song, Hong Shang, Nianfa Wang, Chunsheng Hua, Chong Wu, Xin Qi, and Jianda Han. "Search and Rescue Rotary-Wing UAV and Its Application to the Lushan Ms 7.0 Earthquake". In: *Journal of Field Robotics* 33.3 (2016), pp. 290–321.

[71] Chen Qian, Xiao Sun, Yichen Wei, Xiaoou Tang, and Jian Sun. "Realtime and robust hand tracking from depth". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, pp. 1106–1113.

[72] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. IEEE. 2011, pp. 2564–2571.

[73] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. "Learning representations by back-propagating errors". In: *Cognitive modeling* 5.3 (1988), p. 1.

[74] Szymon Rusinkiewicz and Marc Levoy. "Efficient variants of the ICP algorithm." In: *3dim*. Vol. 1. 2001, pp. 145–152.

[75] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. "Fast point feature histograms (FPFH) for 3D registration". In: *IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 3212–3217.

[76] Jürgen Scherer, Saeed Yahyanejad, Samira Hayat, Evsen Yanmaz, Torsten Andre, Asif Khan, Vladimir Vukadinovic, Christian Bettstetter, Hermann Hellwagner, and Bernhard Rinner. "An Autonomous Multi-UAV System for Search and Rescue". In: *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*. DroNet '15. Florence, Italy: ACM, 2015, pp. 33–38. ISBN: 978-1-4503-3501-0. DOI: 10.1145/2750675.2750683. URL: http://doi.acm.org/10.1145/2750675.2750683.

[77] M. Shan, F. Wang, F. Lin, Z. Gao, Y. Z. Tang, and B. M. Chen. "Google map aided visual navigation for UAVs in GPS-denied environment". In: *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2015, pp. 114–119. DOI: 10.1109/ROBIO.2015.7418753.

[78] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. "CNN features off-the-shelf: an astounding baseline for recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2014, pp. 806–813.

[79] Sebastian Siebert and Jochen Teizer. "Mobile 3D mapping for surveying earthwork projects using an Unmanned Aerial Vehicle (UAV) system". In: *Automation in Construction* 41 (2014), pp. 1–14.

[80] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556.

[81] Richard Szeliski, Heung-Yeung Shum, Heung-Yeung Shum, and Heung-Yeung Shum. "Creating full view panoramic image mosaics and environment maps". In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 1997, pp. 251–258.

[82] Pratap Tokekar, Joshua Vander Hook, David Mulla, and Volkan Isler. "Sensor planning for a symbiotic UAV and UGV system for precision agriculture". In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1498–1511.

[83] Jezching Ton and Anil K Jain. "Registering Landsat images by point matching". In: *IEEE Transactions on Geoscience and Remote Sensing* 27.5 (1989), pp. 642–651.

[84] A. Vedaldi and K. Lenc. "MatConvNet – Convolutional Neural Networks for MATLAB". In: *Proceeding of the ACM Int. Conf. on Multimedia*. 2015.

[85] Yannick Verdie, Kwang Yi, Pascal Fua, and Vincent Lepetit. "TILDE: a temporally invariant learned detector". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 5279–5288.

[86] Chaoyang Wang, Hamed Kiani Galoogahi, Chen-Hsuan Lin, and Simon Lucey. "Deep-lk for efficient adaptive object tracking". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 627–634.

[87] Chieh-Chih Wang, Charles Thorpe, Sebastian Thrun, Martial Hebert, and Hugh Durrant-Whyte. "Simultaneous localization, mapping and moving object tracking". In: *The International Journal of Robotics Research* 26.9 (2007), pp. 889–916.

[88] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. "Dynamic graph CNN for learning on point clouds". In: *arXiv preprint arXiv:1801.07829* (2018).

[89] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. "3D ShapeNets: A deep representation for volumetric shapes". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1912–1920.

[90] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. "PoseCNN: A convolutional neural network for 6d object pose estimation in cluttered scenes". In: *arXiv preprint arXiv:1711.00199* (2017).

[91] Jiaolong Yang, Hongdong Li, Dylan Campbell, and Yunde Jia. "Go-ICP: A globally optimal solution to 3D ICP point-set registration". In: *IEEE transactions on pattern analysis and machine intelligence* 38.11 (2016), pp. 2241–2254.

[92] Kun Yang, Anning Pan, Yang Yang, Su Zhang, Sim Ong, and Haolin Tang. "Remote sensing image registration using multiple image features". In: *Remote Sensing* 9.6 (2017), p. 581.

[93] Zi Jian Yew and Gim Hee Lee. "3DFeat-Net: Weakly supervised local 3D features for point cloud registration". In: *European Conference on Computer Vision*. Springer. 2018, pp. 630–646.

[94] A. Yol, B. Delabarre, A. Dame, J. Dartois, and E. Marchand. "Vision-based absolute localization for unmanned aerial vehicles". In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 3429–3434. DOI: 10.1109/IROS.2014.6943040.

[95] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

[96] Wentao Yuan, David Held, Christoph Mertz, and Martial Hebert. "Iterative Transformer Network for 3D Point Cloud". In: *arXiv preprint arXiv:1811.11209* (2018).

[97] Wentao Yuan, Tejas Khot, David Held, Christoph Mertz, and Martial Hebert. "PCN: Point Completion Network". In: *3D Vision (3DV), 2018 International Conference on*. 2018.

[98] Yin Zhou and Oncel Tuzel. "VoxelNet: End-to-end learning for point cloud based 3d object detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 4490–4499.