Localization Using Feature Matching in Near-Random Textures

Siyu Liu

A Master Thesis
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Master of Science in Engineering

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Szymon M. Rusinkiewicz

June 2013

© Copyright by Siyu Liu, 2013.

All rights reserved.

Abstract

Precise vehicle localization, whether performed via visual scene registration or specialized sensors such as GPS, is currently limited in robustness and accuracy. We propose a computer vision-based localization system that improves precision by matching features in small patches of asphalt or carpet imaged with a downward-facing camera. The key insight is that such images will have enough near-random detail to retrieve their correct position in the environment.

In our system, the environment is first processed to construct a knowledge base. Then during the retrieval phase, we apply multilevel search to precisely localize the observed patch in the known environment. We adjust the patch registration algorithms previously developed for natural scenes to the case of ground environments with near-random textures. Our system is marker-free and robust against orientation as well as illumination changes. By combining temporal coherence with classification techniques, we can instantaneously provide accurate localization predictions. We also envision several potential applications including vehicle navigation system and precise control for 3D fabrication.

Acknowledgements

First and foremost, I would like to thank my advisor, Szymon Rusinkiewicz, for his exceptional guidance and patient mentorship during my graduate studies. I would like to thank my reader, Thomas Funkhouser, for his support and feedback. I would like to thank the Computer Science department for providing this research opportunity to develop this work. I would like to thank everyone at Princeton Graphics Group for the generous help and the wonderful time we shared together during the past two years. I would also like to thank all the fellow master students and faculty members I worked with in the Computer Science department. Finally, I would like to thank my family for their continuous moral support and stand by me every day.

To my grandparents and my parents.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Localization is a challenging problem of finding accurate positions in a known environment. It is mostly used for robot navigation tasks which has been a prominent field since 1980s. There are two main approaches proposed in literature. One is sensor facilitaed localization system. Inertial measurements and GPS sensors combined with temporal based models such as Kalman Filter are used to recover the outdoor position of the robot [4]. On the other hand, there are also computer vision based localization systems focusing on landmark registration in 3D scenes. The main idea is to recognize and track reference objects in the scene and use stereo vision to retrieve the position in the environment [39]. Unfortunately, the former approach is often less accurate and suffers from long term drift in sensor readings; the later approach relies on correct scene registrations which could be difficult due to occlusions, viewing angles and dynamic changes in the environment.

In this work, we propose a computer vision based localization system based on feature matching with a down-facing camera looking at the ground. Therefore, we simplify the problem by keeping our observations relatively at the same scale and from the same viewing angle. In addition, it is also easier to control occlussions and lighting conditions on the ground. Besides, comparing to complicated 3D scenes, dynamic changes due to human activity can be also reduced in our setting.

However, in spite of the advantages it brings, ground surfaces such as carpet, wood or concrete usually contain near-random textures. Thus, there are less intuitively unique structures to differentiate one part from the rest, at least from the large scale. Fortunately, given the fact that the ground surface is not consisted of perfect replicas of the same pattern, there still exist enough details to distinguish one patch from the others. One early success of this idea is now widely used inside the optical mouse. A built-in low resolution camera takes a series of images as the mouse moves; cross correlation is used to solve for changes in positions due to motion. Nonetheless, this system usually requires taking large amount of images with small offsets between consecutive frames. Another previous attempt appears in the paper [9] where they perform image feature matching on fine-scaled scans of the votes in order to authenticate voting paper. Motivated by these successes, we decide to evaluate this idea in larger and more general enviroments for localization purpose.

The localization system presented in this work is divided into two phases, training and testing. In training phase, we first preprocess input image data with different image filters and then extract features from it to build up our knowledge base. We also apply a feature selection algorithm to reduce the size of the knowledge base to keep it compact. In testing phase, the observed patches will first go through the same preprocessing steps. Then we will extract the same image features from them and match against our knowledge base.

In the following chapters of this thesis, we will first review some related work. Secondly, we will introduce the data acquisition and ground truth derivation process. Then we will describe our main algorithm step by step in more details. Finally, we will conclude this work by analyzing our experimental results and discussing possible future directions.

Chapter 2

Related Work

Our localization system involves techniques in image filtering, feature extraction, feature matching using nearest neighbor search and patch registration. The pipeline is depicted in Figure 4.1. In this chapter, we will discuss some previous works relating to these areas.

2.1 Image Filtering

To improve the consistency among input images, preprocessing becomes a necessity to reduce the variations in appearance. During the years, many image filtering algorithms have been developed to address this issue. Many of them focus on frequency analysis in Fourier space. The particular algorithm we chose for this project is bandpass filtering [13]. The idea is to select an appropriate frequency band to decrease the influences from both high-frequency noise and low-frequency overall intensity changes. Furthermore, we also experimented with other filtering techniques such as median filter and laplacian filter [13]. In addition, to adjust the different luminance level due to different lighting conditions, we also tried to apply histogram equalization [19] and luminance remapping [17] as a normalization procedure between our knowledge base and observed testing patch to achieve better matching result.

2.2 Feature Detection

In order to efficiently and effectively describe our environment, we extract features using feature detection algorithms. Over the past decade, different feature detectors have been proposed focusing on different aspects. Some look for corners: Harris Corner Detector [16, 40, 52] thresholds on high response in both directions of the gradient; Smallest Univalue Segment Assimilating Nucleus (SUSAN) Corner Detector [44] uses circular mask and relies on brightness dissimilarity; Wavelet Based Multi-Scale Analysis [22] thresholds on high-high component of an image decomposed by a B-spline wavelet in several scales. Some other detectors try to identify interesting regions: Maximally Stable Extremal Region Detector (MSER) [27] imposes thresholds on different intensity levels to slice input image into a serie of binary images and merging blobs of local minimums; Salient Region Detector [18] relies on probability density function of intensity values computed over an elliptical region and select the entropy extrema over scales; Scale Invariant Feature Transform (SIFT) [26] detects local spatial extrema between consecutive layers within a Gaussian pyramid. In this work as the first attempt, we choose SIFT detector for its stability in detection and compatibility with SIFT descriptor.

2.3 Feature Description

After identifying the interest points using feature detectors, we would like to design a concise representation to describe them. Therefore, accompanied with detection algorithms, many feature discriptors are developed on the side. Filter-Based Descriptors such as Steerable Filters [14] and Gabor Filters [10] focus on single filter response whereas Textons [24, 48] usually consist of a set of filter responses from a selected filter bank; Phase-Based Local Features [8] uses both phase and amplitude responses from complex-valued steerable bandpass filters; Distribution-Based Discriptors such as SIFT descriptor [26] builds a histogram over quantized gradient orientations and Rotation Invariant Feature Transform (RIFT) [50] constructs a similar orientation histogram yet within concentric rings after circular normalization; Derivative-Based Descriptors such as Local Grayvalue Invariants [38] computes a group of differential invariants up to third-order local derivatives; Color-Based Descriptors [49] collects normalized RGB as well as opponent/spherical angles. In this project, we choose SIFT descriptor to work with SIFT detector.

2.4 Nearest Neighbor Search

We applied nearest neighbor search for feature matching process. Unfortunately, exhaustive search is usually time-consuming which prevents us from developing real-time applications. Therefore, we choose approximate nearest neighbor (ANN) to accelarate the localization system. There are two common methods for ANN search proposed in literature. One attempts to partition the space while the other tries to cluster the data. In this work, we choose FLANN (Fast Library for Approximate Nearest Neighbors) based matching algorithm [32] which consists of randomized kd-trees and hierarchical k-means trees.

The first kd-tree model proposed by Feidman et al. dates back to 1977 [15] where they split data in half at each level of the tree along the dimension with largest variance. A recent continuing work constructs a randomized kd-tree [42] who chooses randomly from the first D dimensions where data varies the most. In addition, the search space is quantized into bins and a randomized tree is grown in each bin. Then, during searching phase, a priority queue is maintained across all randomized kd-trees and the search is ordered by increasing distances to the boundary of each bin.

On the other hand, the hierarchical k-means tree [23] is a direct extension from k-means clustering algorithm with clustering performed recursively at each level of the tree. Simliar to searching among randomized kd-trees, a prioirty queue of unexplored branches is maintained at each node to generate a heuristic according to which the search is performed in a greedy manner of best-bin-first.

2.5 Patch Registrtion

Feature matching using nearest neighbor search can sufficiently narrow down the search space for localization problem. Unfortunately, matches among multiple interest points against the knowledge base may not always agree with each other. Greedily selecting the one with the least feature space distance is shown to be misleading sometimes. Therefore, template matching techniques are used as an extra selection mechanism to determine which nearest neighbor match is most reliable.

In general, patch registration algorithms often choose to calculate the difference between given patch (or the template) and its potential matches. Widely used similarity metrics include cross correlation, bi-directional distance [43] and etc. In order to achieve rotation and scale invariant, people also attempted to map original data from spatial domain to log-polar space using Fourier-Mellin transform [11]. In this work, we choose to use normalized cross correlation [51] to refine nearest neighbor search results from feature matching step.

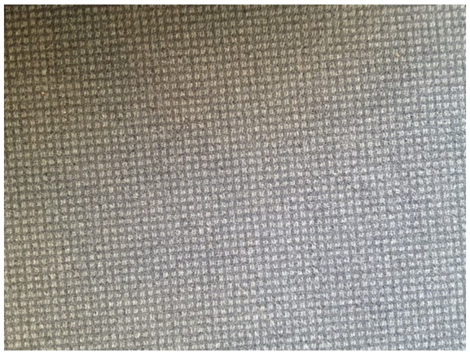
Chapter 3

Data

The data used in this work is captured by camera phones with different resolutions. The image data include different materials, such as carpet, wall, concrete and etc., under different lighting conditions. All of them are taken with cameras fixed at roughly the same height from the captured surface. Furthermore, the data set is divided into two parts for training and testing purpose. The testing image is a shifted and rotated version of the training one with limited changes in scale. The lighting conditions may or may not vary between training and testing sets. Some of the samples are shown in Figure 3.1. The data also comes with two different resolutions, 640×480 and 1632×1224, covering different size of areas e.g. 126cm$\times$94.5cm for carpet data, which means in the low resolution, each pixel is a square of side length roughly 2mm. During testing phase, a patch of size 30×30 pixels (120mm$\times$120mm) is clipped around an arbitrary position from the testing image. The overlapped region between training and testing images varies from 50% to 90% across different pairs.

3.1 Preprocessing

In order to reduce the influence from different intensity level due to different illumination conditions, two normalization techniques, histogram equalization [19] and


(a) Carpet Data

(b) Woolen Data

(c) Concrete Data

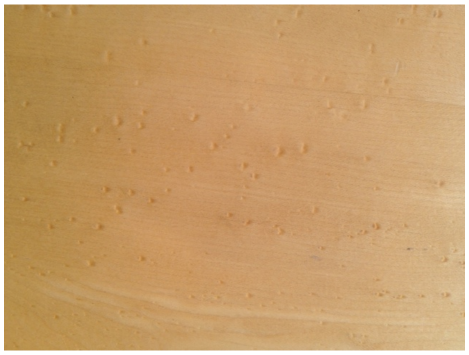

(d) Wood Data

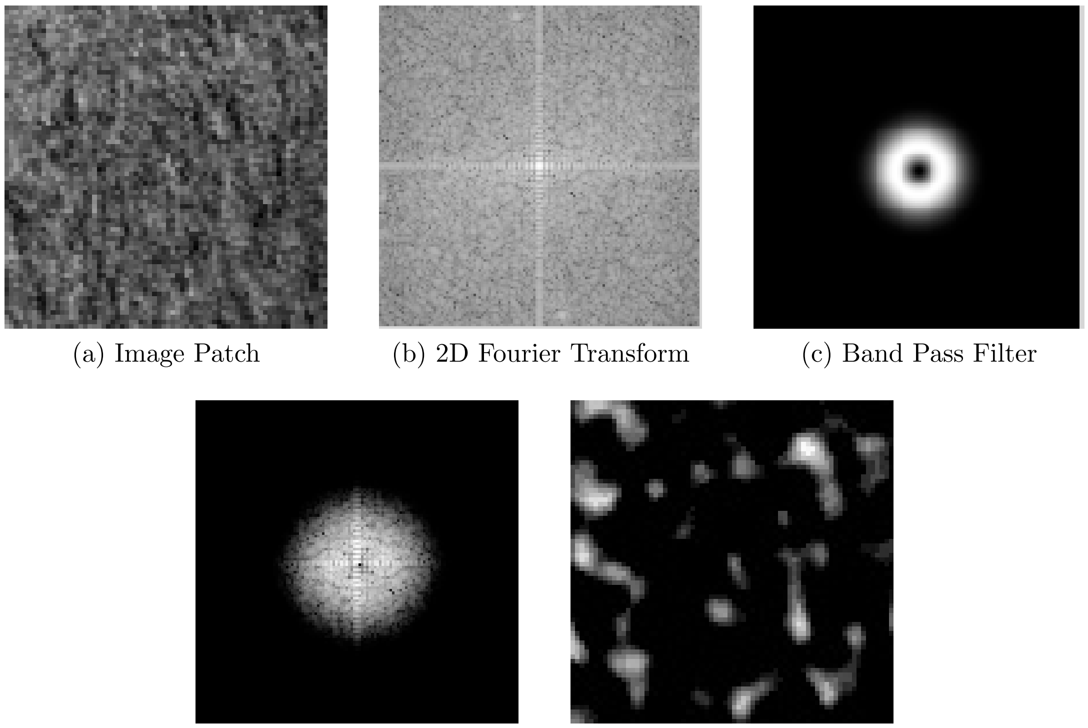
Figure 3.1: Data Images

luminance remapping [17] are implemented and compared in this work. For simplicity purpose, all input images are converted to grayscale first and applied a band pass filter to attenuate the noise and remove the base color of the surface.

3.1.1 Band Pass Filter

Treating images as signals enables us to analyze structures at different scales in the frequency domain (Figure 3.2b). High frequency information would correspond to sudden intensity changes such as sharp edges or undesirable noise from capturing process. In addition, specific to this work, we would like to tolerate certain amount of subtle surface geometry changes, for instance, stepping on the carpet. Therefore, smoothing out some high frequency information allows us to match the rough shape of the textures themselves. On the other hand, we would also like to filter out some low frequency information to adjust the overall illumination changes for directional lighting and remove the base color of the surface.

To achieve this goal, we convolve our input images with a band pass filter kernel (Figure 3.2c). For time efficiency, we apply 2D fast Fourier transform on the input image (Figure 3.2b) and then perform convolution as a multiplication in Fourier space (Figure 3.2d). As shown in Figure 3.2e, we can extract certain patterns after applying band pass filter to the noisy input image patch.


(a) Image Patch (b) 2D Fourier Transform (c) Band Pass Filter

(d) Apply Band Pass Filter (e) Inverse Fourier Transform

Figure 3.2: Band Pass Filter on Carpet Image Patch

3.1.2 Histogram Equalization

The first intensity normalization algorithm we experimented is histogram equalization [19]. The idea is similar to inverse sampling. Its purpose is to improve the contrast and stretch the intensity distribution. The density function of intensity is first accumulated to form its cumulative distribution. Then this distribution is normalized to the range from 0 to 255. Finally, we remap it back to achieve equalized intensities. The result is shown in Figure 3.3 and as we can see, more details are revealed after contrast enhancement. Unfortunately, the normalization is not uniform across the entire image. Assuming the intensities follow a Gaussian distribution (which they usually do), after remapping, the central mass spreads out to the two sides causing loss in contrast in brighter and darker regions. Thus, on the right side of the wall with less light reached, we lose details after histogram equalization. This observation shows that this technique is very sensitive to light directions. Although filtering out low frequency information could help to alleviate this drawback, enhancing contrast might introduce more high frequency artifects. As stated in previous section, high frequency information could lead to difficulties in matching. Therefore, this technique is not included in the final version of our system.

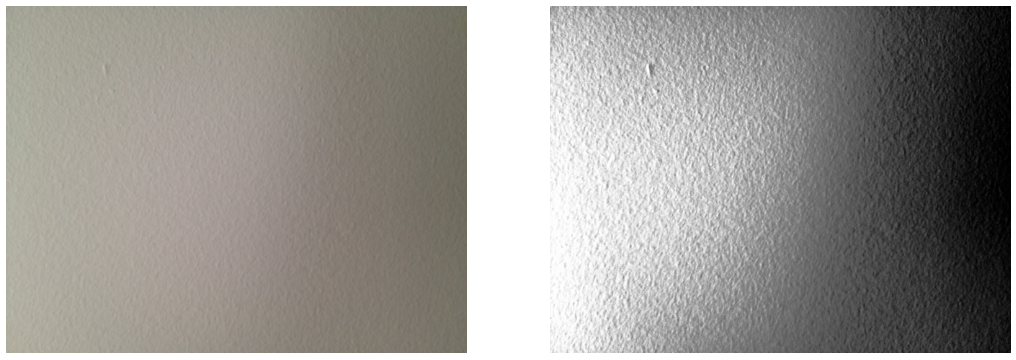

(a) Wall Data (b) Apply Histogram Equalization

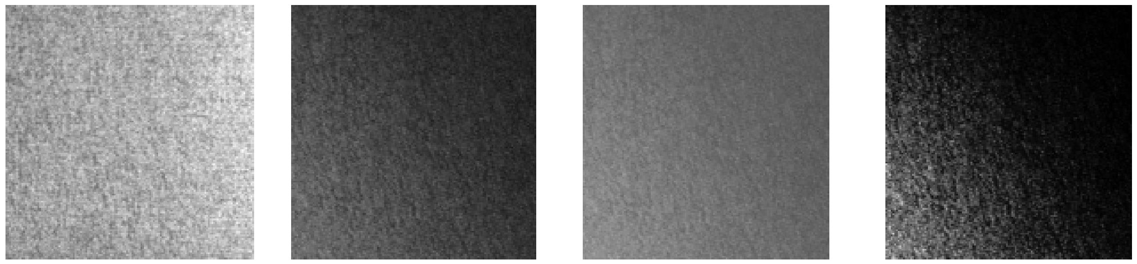
Figure 3.3: Intensity Normalization Using Histogram Equalization

3.1.3 Luminance Remapping

Instead of normalizing every intensity distribution to 0 to 255 range, luminance remapping [17] tries to map one distribution to the other according to Eq 3.1. Visually, luminance remapping will make a darker image brighter (Figure 3.4b and Figure 3.4c). Unlike the non-uniform behavior from histogram equalization in Figure 3.4d, luminance remapping preserves details in darker areas (upper-right corner).

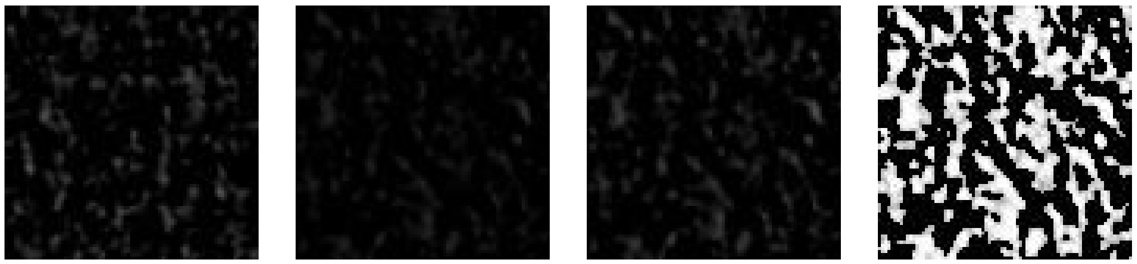
$$\mathbf{Y}(p) \leftarrow \frac{\sigma_{\mathbf{B}}}{\sigma_{\mathbf{A}}}(\mathbf{Y}(p) - \mu_{\mathbf{A}}) + \mu_{\mathbf{B}} \tag{3.1}$$

***Y**(p) denote the pixels in image A; μ_A, μ_B and σ_A, σ_B are the means and standard deviations of the intensities in image A and B respectively.*



(a) Image 1 (b) Image 2 (c) Image 2 (Luminance Remapping) (d) Image 2 (Histogram Equalization)

Figure 3.4: Comparison between Luminance Remapping and Histogram Equalization



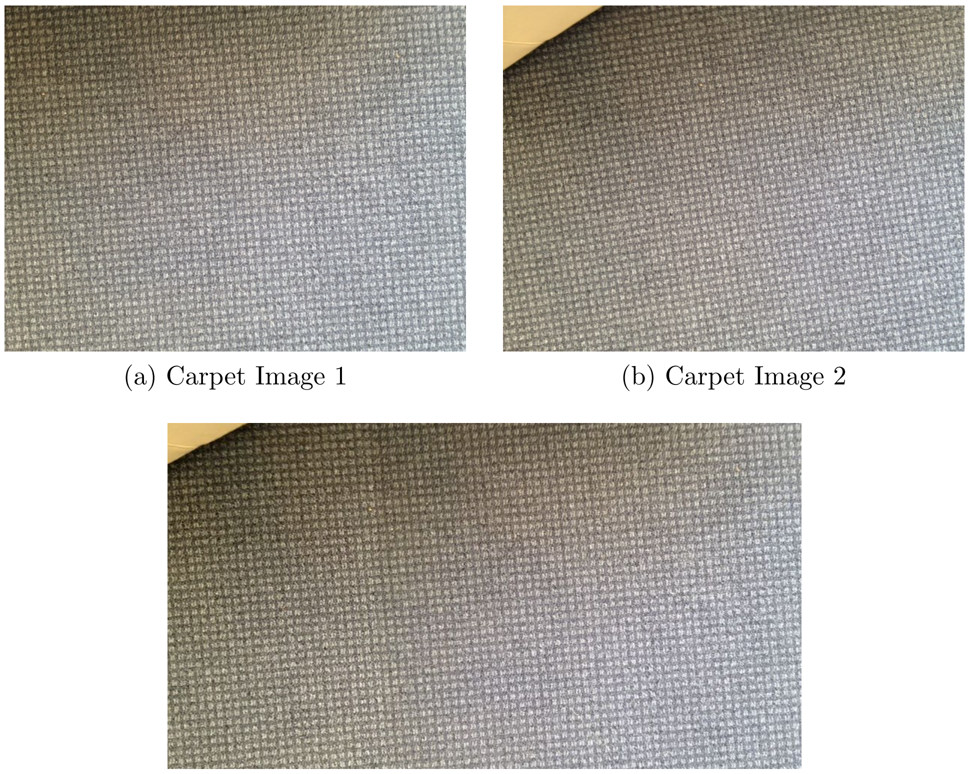
(a) Image 1 (b) Image 2 (c) Image 2 (Luminance Remapping) (d) Image 2 (Histogram Equalization)

Figure 3.5: Comparison between Luminance Remapping and Histogram Equalization after Band Pass Filter

In addition, the undesirable increase in contrast from histogram equalization magnifies the subtle changes in surface geometry as shown in Figure 3.5. These reintroduced high frequency structures go against the purpose of applying band pass filter at the first place. In general, our experiments found luminance remapping preserves enough details from the input images while adjusting them to the same intensity level.

3.2 Ground Truth Derivation

In order to evaluate the performance of our system, we first need to generate the groud truth mapping between training and testing images. Under similar lighting conditions, matching SIFT features with RANSAC [13] on the raw input images will usually provide reasonably high quality stitching result as in Figure 3.6c.


(a) Carpet Image 1

(b) Carpet Image 2

(c) Stitched Image

Figure 3.6: Image Stitching Using SIFT Matching with RANSAC

Nonetheless, to ensure the reliability of the ground truth, we also use more sophisticated state-of-the-art image composition tools such as i2k [1] and ICE [2] with markers while stitching.

Chapter 4

Algorithm

Our localization system is divided into two phases, training and testing, as shown in Figure 4.1. To begin with, we preprocess raw input images with image filters described in Section 3.1.1. During training phase, we first extract image features using feature detectors and represent them using feature discriptors; we then perform feature selection using local search algorithm; finally, we use the feature descriptors and compressed input images to construct the knowledge base. During testing phase, we use nearest neighbor search and patch registration techniques to match testing patch against the knowledge base. In this chapter, we will discuss each step of our algorithm in details.

4.1 Feature Extraction

In this work, we experiemtned with three different feature detectors, Harris corner detector [16], Scale Invariant Feature Transform (SIFT) [26] and Maximally Stable Extremal Region Detector (MSER) [27]. On the other hand, we start from using simple point feature descriptor, i.e. gradient magnitude and orientation, and then continue to more complicated region feature descriptors e.g. SIFT descriptor, polar

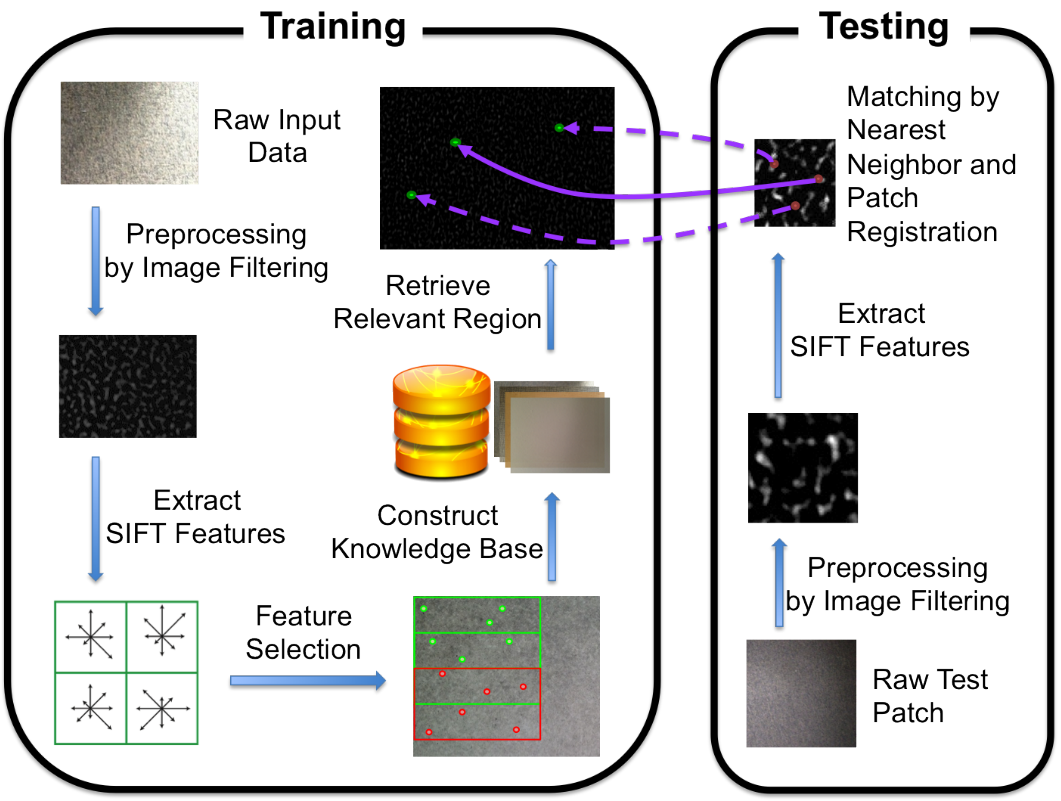


Figure 4.1: System Pipeline

descriptor and Fourier shape descriptor. In this section, we will describe each of the detection and description algorithm in details.

4.1.1 Feature Detection

Harris Corner Detector

Propsed by Harris and Stephens in 1988 [16], the Harris Corner detector tries to identify key points in the image corresponding to corners. Given a grayscale image $\boldsymbol{I}$, we perfom a swap window over an area (u, v) weighted by $w(u, v)$ centered at (x, y). The weighted sum-squared difference is given by Eq. 4.1.

$$\boldsymbol{S}(x, y) = \sum_{u,v} w(u, v)(\boldsymbol{I}(x + u, y + v) - \boldsymbol{I}(u, v))^2 \tag{4.1}$$

Using Tylor expansion, $\boldsymbol{I}(x+u, y+v)$ can be approximated as Eq. 4.2, where $\boldsymbol{I}_x$ and $\boldsymbol{I}_y$ are the partial derivatives of $\boldsymbol{I}$ approximated using finite difference along x and y direction respectively.

$$\boldsymbol{I}(x+u, y+v) \approx \boldsymbol{I}(u, v) + \boldsymbol{I}_x(u, v)x + \boldsymbol{I}_y(u, v)y \tag{4.2}$$

Then, plug Eq. 4.2 into Eq. 4.1 we get,

$$\boldsymbol{S}(x, y) \approx \sum_{u,v} w(u, v)(\boldsymbol{I}_x(u, v)x + \boldsymbol{I}_y(u, v)y)^2 \tag{4.3}$$

In martix notation, we have,

$$\boldsymbol{S}(x, y) \approx \begin{pmatrix} x & y \end{pmatrix} \boldsymbol{M} \begin{pmatrix} x \\ y \end{pmatrix}, \ \boldsymbol{M} = \sum_{u,v} w(u, v) \begin{pmatrix} \boldsymbol{I}_x^2 & \boldsymbol{I}_x\boldsymbol{I}_y \\ \boldsymbol{I}_x\boldsymbol{I}_y & \boldsymbol{I}_y^2 \end{pmatrix} \tag{4.4}$$

To determine the possibility of a corner, we threshold on the response in Eq. 4.5.

$$\boldsymbol{R} = det(\boldsymbol{M}) - k(trace(\boldsymbol{M}))^2 \tag{4.5}$$

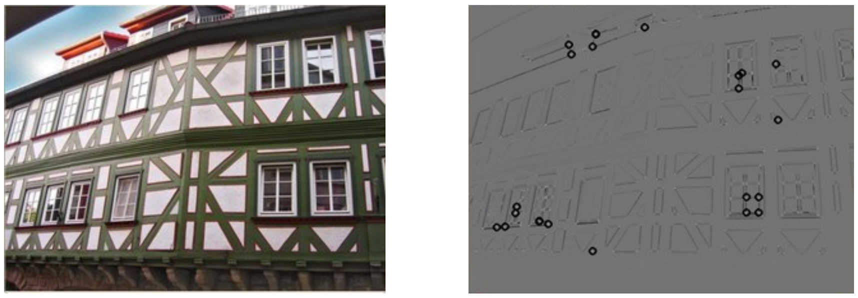
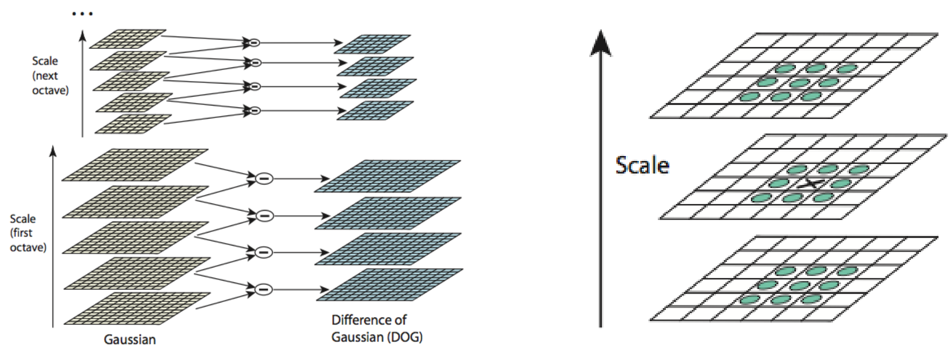

(a) Original Image (b) Detected Corners

Figure 4.2: Corners Detected Using Harris Detector

Note that $det(\boldsymbol{M})$ and $trace(\boldsymbol{M})$ are the product and sum of two eigenvalues λ_1 and λ_2 of $\boldsymbol{M}$ respectively. Therefore, if the intensity changes rapidly in two different directions i.e. the magnitudes of both λ_1 and λ_2 are sufficiently large, the current window would probably contain a corner (Figure 4.2).

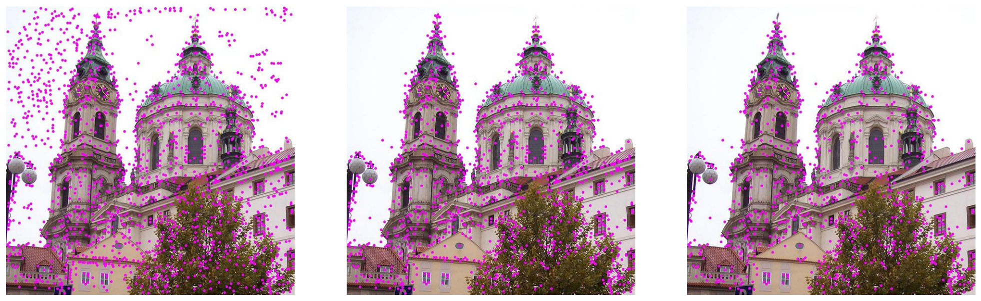
SIFT Detector

Scale Invariant Feature Transform or SIFT [26] is a commonly used image feature for image stitching. It is shown to be reliable from previous works. To find the interest points, we first need to build the Gaussian pyramid, a pyramid by repeatedly smoothing and subsampling from the input image. By subtracting two consecutive layers, we can obtain the Difference-of-Gaussian (DoG) pyramid as in Figure 4.3a.



(a) Gaussian Pyramid and DoG Pyramid (b) Spatial Extrema

Figure 4.3: SIFT Feature Detector



(a) All Interest Points (b) Discard Low Contrast Interest Points (c) Filter out Edge-Like Interest Points

Figure 4.4: Key Points Using SIFT Dectector

Then, the interest points are the spatial extrema of its 26 neighbors, i.e. 8 neighbors in the same layer, and 9 neighbors from each of the two adjacent layer (Figure 4.3b). The extracted interest points are unique to the scale they were found. A sample result on a natural scene image is shown in Figure 4.4.

MSER Detector

Maximally Stable Extremal Region (MSER) [27] is another common image feature for finding correspondences in nature scenes. It is also argued to sometimes have better localization accuracy than SIFT detectors. To describe the detection process, we first need to define some terminologies. A region $\boldsymbol{R}$ is defined to be a connected component of pixels. The outer boundary of $\boldsymbol{R}$ is denoted as $\partial\boldsymbol{R}$ which is a set of pixels having at least one adjacent pixel to $\boldsymbol{R}$ but not belonging to $\boldsymbol{R}$. An extremal region is a region $\boldsymbol{R}$ such that for all pixels $p \in \boldsymbol{R}$ and $q \in \partial\boldsymbol{R}$, the intensities $\boldsymbol{I}(p) > \boldsymbol{I}(q)$. The stability of a region $\boldsymbol{R}$ is measured by the relative area variation r_i of $\boldsymbol{R}$ at intensity level i with the intensity threshold increasing (decreasing) by the parameter Δ. Formally, the relative area variation r_i is defined as Eq. 4.6, where $|\cdot|$ denotes the cardinality or the size of the region $\boldsymbol{R}$ and $\boldsymbol{R}_{i+\Delta}$ ($\boldsymbol{R}_{i-\Delta}$) is the region $\boldsymbol{R}_i$ with Δ increase (decrease) from intensity level i. The MSER detector tries to identify the regions with the intensity level i' where $r_{i'}$ reaches a minimum.

$$r_i = \frac{|\boldsymbol{R}_{i+\Delta} \setminus \boldsymbol{R}_{i-\Delta}|}{|\boldsymbol{R}_i|} \tag{4.6}$$

A comparison study by Mikolajczyk et al. [31] shows that MSER is arguably better than other region detectors, Harris-affine [29, 37], Hessian-affine [30], edge-based regions [46], intensity extrema [47], and salient regions[21]. MSER can detect interest points with relatively robustness to scale, viewpoint and illumination changes

(Figure 4.5). However, MSER tends to focus smaller regions which may cause more confusions for matching especially in the context of near-random textures.

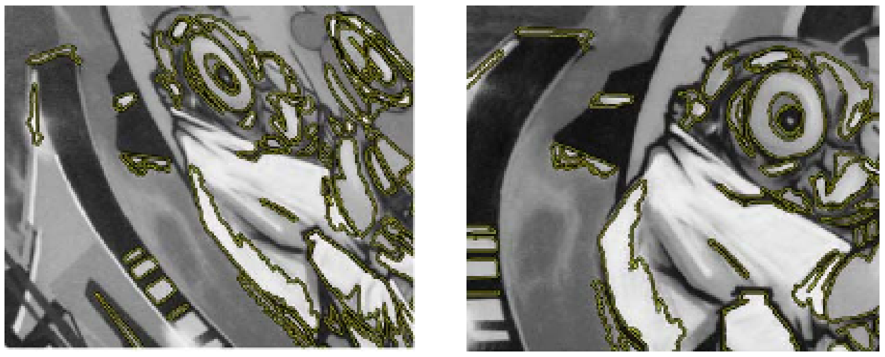

(a) Viewpoint 1 (b) Viewpoint2

Figure 4.5: Originally Detected Shape Regions by MSER

4.1.2 Feature Description

After identifying the locations of keypoints (interest points), we would like to describe them before matching process. In this work, we experimented with four different kinds of feature descriptors, gradient descriptor using gradients at a keypoint, SIFT descriptor using a weighted histogram of gradient orientations over certain neighborhood around a keypoint, polar descriptor sampling intensities along concentric circles centered at a keypoint and Fourier descriptor on texture shape contours near a keypoint. In this section, we will explain them in details.

Local Gradient Descriptor

Starting at the finest scale, our first attempt is to use the local gradients at the keypoint itself to construct a descriptor. Given the structure tensor $\boldsymbol{M}$ defined in Eq. 4.4, our descriptor is a 4 dimensional vector consisting of the orientations of two eigenvectors $\overrightarrow{\boldsymbol{e}_1}, \overrightarrow{\boldsymbol{e}_2}$ and the magnitudes of two eigenvalues $|\lambda_1|, |\lambda_2|$ of $\boldsymbol{M}$.

One obvious drawback of this descriptor is that it is hardly reproducible. Thus, if the training and testing images are captured roughly at the same time, this simple descriptor may produce reasonably good matching results. However, our experiments found that it is very sensitive to lighting and high frequency information changes such as noise or slight variations in surface geometry. Therefore, we only used it at the very early stage of this work to validate the idea.

SIFT Descriptor

Discover the flaws in only considering the keypoint itself, we decided to use a region descriptor instead. The one we experimented the most is the SIFT descriptor [26].

Given a keypoint, we first compute the gradient magnitude and orientation at each image sample point within certain neighborhood. A Gaussian weight is assigned to each sample point according to its distance to the keypoint at the center indicated by the overlaid circle in Figure 4.6. Then, these samples are accumulated into orientation histograms summarizing the structures over a 4×4 grid in each quadrant. Each arrow in the histograms is of length equal to the Gaussian weighted sum of gradient magnitudes in the corresponding orientation bin of the quadrant. Then we can construct a 2×2 descriptor shown on the right of Figure 4.6.

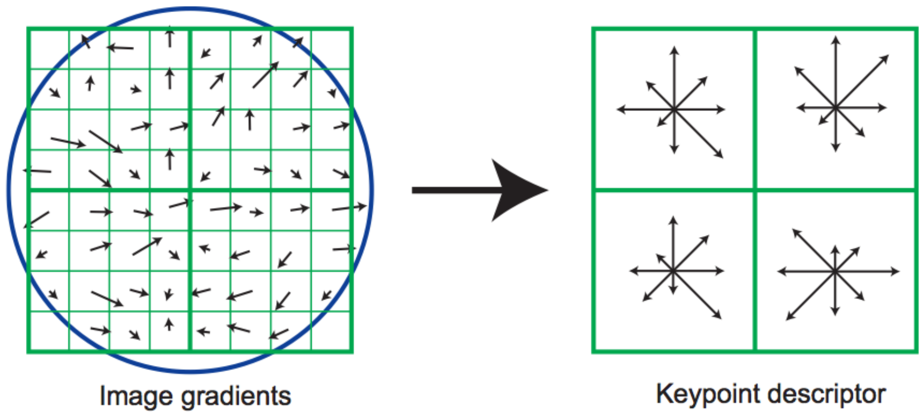


Figure 4.6: SIFT Descriptor

In this work, we used 4×4 descriptors computed from 16×16 grid. Within each subregion of the descriptor, the orientation is quantized into 8 equally spaced bins. Furthermore, we normalized the histogram according to the dominant direction of the gradient of the keypoint at the center to achieve rotation invariance.

Polar Descriptor

Comparing to gradient-based feature descriptors, we also attempted to build descriptors using only intensities. The one we experimented is the polar descriptor which is to transform a patch around a keypoint to polar space (Figure 4.7). However, we observed that under conventional polar transform, the sampling is not uniform. The rings with smaller radius get denser samples comparing to those with larger radii.

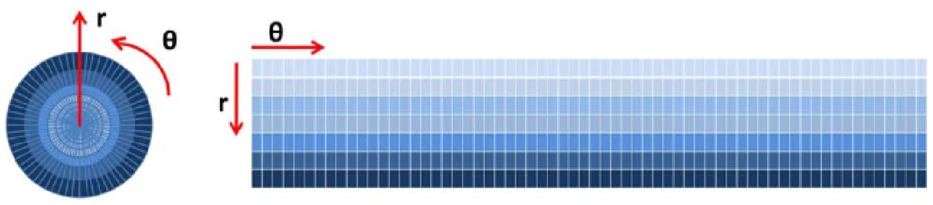


Figure 4.7: Polar Transform Sampling

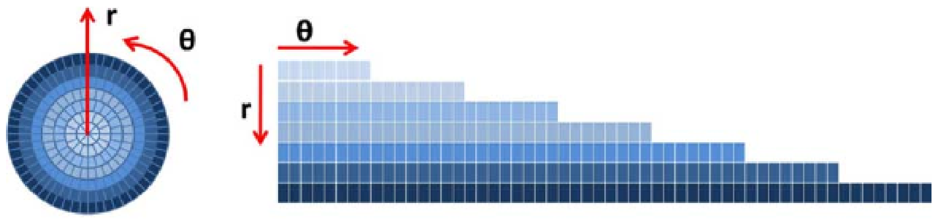


Figure 4.8: Adaptive Polar Transform Sampling

To adjust the bias, we applied the adaptive sampling strategy proposed by Matungka et al. [28]. At each radius R_i, the number of pixels the corresponding ring covers is roughly $2\pi R_i$. Assuming we want to sample n points along the unit ring, then for a ring of radius R_i, the angular sampling rate θ_i would be $\frac{2\pi}{nR_i}$. This result yields an adaptive polar transform as in Figure 4.8. One example on the Lena image is shown in Figure 4.9.

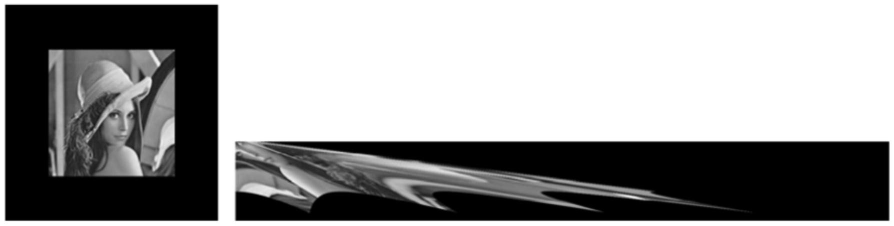


Figure 4.9: Adaptive Polar Transform on Lena Image

In this work, we sampling along three different radii, 25%, 50% and 75% of the given patch radius. Since the patches are preprocessed with band pass filter, each sample is a weighted average of nearby pixels in the original image. Finally, we stack them together to form a 2-dimensional feature descriptor. Then, we can use Fast Fourier Transform with phase correction [34] to recover the transformation.

Although adaptive polar transform performs well in patch registration for natural images, due to the nature of near-random textures, its performance often depends on the number of different radii used for sampling. In addition, matching two different descriptors requires FFT which is an expansive operation for pairwise comparisons. Having considered that, we decide not to use it in the final version of our system.

Fourier Shape Descriptor

The last descriptor we tried is Fourier descriptor developed for matching shape contours. The contour-finding algorithm we choose for this work is one proposed by Suzuki and Abe [45]. We calculate the center of contour points and take angular samples evenly along the contour with linear interpolation on subpixels. Finally, we use FFT and extract the Fourier coefficients to form a shape descriptor. Moreover, we normalize the scale by dividing the magnitudes of other Fourier coefficients by the first one, i.e. the DC component. The main problem with this approach is that the contour may break or merge when the patch shifted (Figure 4.10b, Figure 4.10c). Furthermore, the shape contours of the surface textures may vary under different lighting conditions and subtle changes on surface geometry. Therefore, it is not as robust as we required from our experiments.

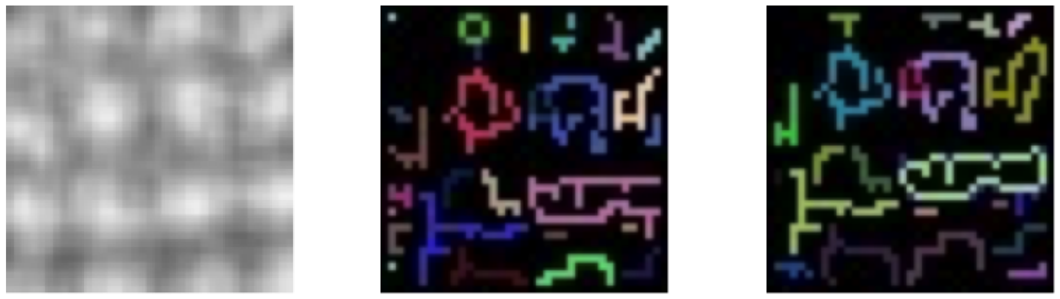

(a) Input Image Patch (b) Contours (c) Contours from Shifted Patch

Figure 4.10: Contour Extraction

4.2 Feature Selection

In order to keep our knowledge base compact and reduce the matching time, we also implemented a local search algorithm for feature selection purpose. There are two major sources of redundancy embedded in the collection of feature descriptors from previous step: some detected keypoints are clustered in image space; some clustered in feature space, which causes confusions in matching. Therefore, we form an optimization problem by selecting a subset of features to address this issue.

Objective Function

The objective function we maximizing in Eq. 4.7 aims to optimize the keypoint distribution in both image space and feature space.

$$\mathbf{S}(f_1, f_2, ..., f_n) = \alpha S_f(f_1, f_2, ..., f_n) + (\mathbf{1} - \alpha) S_s(f_1, f_2, ..., f_n) \tag{4.7}$$

$$S_f(f_1, f_2, ..., f_n) = \sum_{i=1}^{i \leq n} D_f(f_i) \tag{4.8}$$

$$S_s(f_1, f_2, ..., f_n) = \sum_{i=1}^{i \leq n} \sigma(D_s(f_i) - D_{cutoff}) \tag{4.9}$$

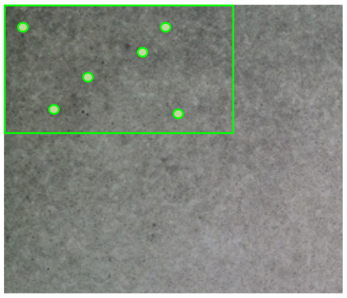
S is the score function for a given set of features. $f_1, f_2, ..., f_n$ are the n selected features in the current set. S_f is the score function for a given set of features in feature space. S_s is the score function for a given set of features in image space. α is the weight between the two score functions. $D_f(f_i)$ is the normalized distance from feature f_i to its nearest neighbor in feature space. $D_s(f_i)$ is the distance from feature f_i to its nearest neighbor in image space. $\sigma(D_s(f_i) - D_{cutoff})$ is the the standard sigmoid function ($\sigma(x) = 1/(1 + e^{-x})$) sifted by D_{cutoff} such that we penalize the score when the distance in image space is less then the threshold given by D_{cutoff}.

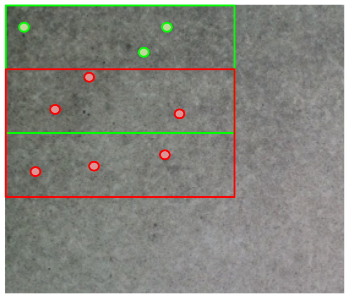
Optimization

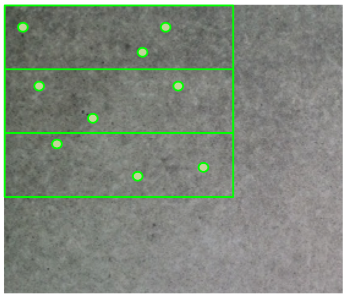
Given the objective function, we search for an optimal (or often a sub-optimal) solution using local search algorithm with Simulated Annealing [20]. Let us denote the set of all features detected by some algorithm in Section 4.1 as $\boldsymbol{U}$ and the subset we selected as $\boldsymbol{S} \subseteq \boldsymbol{U}$. We start the search by randomly selecting m features to form the initial collection $\boldsymbol{S}_0$. In iteration i, we randomly swap n features from the current collection $\boldsymbol{S}_i$ with n features from $\boldsymbol{U} \setminus \boldsymbol{S}_i$. If the objective function increases, we keep the current swaps or otherwise we discard them with probability $1 - p$. As the search proceeds, both parameters n and p decrease towards zero. A typical choices for n and p are 5 and 0.1 at the beginning. The searching process terminates if the relative increment of the objective function falls under some threshold (e.g. 0.01) or the search reaches a predefined maximum number of iterations (e.g. 1000). Moreover, to raise the chance of overcoming a local optimal, we maintain several search chains and select the one yielding highest score as the final output.

Sliding Window

Performing local search over the entire environment is too time-consuming to be practical. Thus we use a sliding window algorithm to accelarate the searching process. We quantize the space into $N \times N$ halfly overlapped windows. Therefore, we only need to optimize within each window while keeping consistency among overlapped regions. We start sliding window from the upper-left corner of the environment and sweep the space from top to bottm, left to right. After search terminates with m features seleted, we identify those inside the overlapped region with the next window and use them as part of the initialization for future search (Figure 4.11).


(a) Search End in First Window


(b) Search in Next Overlapped Window


(c) Search End in Second Window

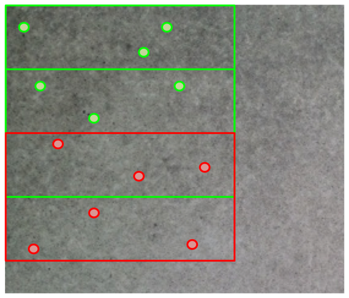

(d) Search in Next Overlapped Window

Figure 4.11: Sliding Window Feature Selection

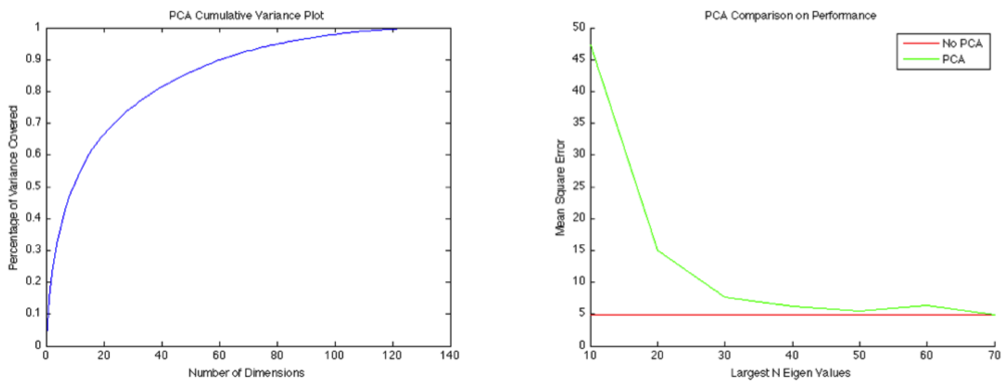
4.3 Knowledge Base Construction

The next stage of our system is to construct the knowledge base. To further decrease the memory cost, we first perform the Principal Component Analysis (PCA) [33] to reduce the dimensionality of SIFT features. Then, we feed them to FLANN-based ANN matcher to build up kd-trees and k-means trees. In addition, to support patch registration using template matching, we can also compress and store the preprocessed image data and dynamically expand them during testing phase.

4.3.1 Principal Component Analysis

The idea of PCA is to use an orthogonal transformation to convert a set of observations with possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. Given a data matrix $\boldsymbol{X}$ whose m rows represent observations with zero empirical mean and n columns represent each dimension of the data, we perform Singular Value Decomposition to find an $m \times m$ matrix $\boldsymbol{W}$ with eigenvectors of $\boldsymbol{X}\boldsymbol{X}^T$, an $n \times n$ matrix $\boldsymbol{V}$ with eigenvectors of $\boldsymbol{X}^T\boldsymbol{X}$ and an $m \times n$ singular value matrix $\boldsymbol{\Sigma}$ such that $\boldsymbol{X} = \boldsymbol{W}\boldsymbol{\Sigma}\boldsymbol{V}^T$. Then to find the d dimensional representation $\boldsymbol{X}_d$ that minimizes the sum of squares reconstruction error, we project $\boldsymbol{X}$ into the space spanned by d eigenvectors of $\boldsymbol{W}$ corresponding to the d largest singular values in $\boldsymbol{\Sigma}$, i.e. $\boldsymbol{X}_d = \boldsymbol{W}_d^T\boldsymbol{X}$. The parameter d is usually selected from empirical analysis such that the reconstruction error is below some threshold.

Specific to our data, we used two plots to find this parameter d, the cumulative singular values plot (Figure 4.12a) and the empirical matching error plot (Figure 4.12b). As a tradeoff between performance and time cost for nearest neighbor search, we choose to reduce SIFT descriptors from 128 to $d = 35$ dimensions for our system. Even though this value is purely determined from carpet data (Figure A.1), it also performs reasonably well with other data we have. Furthermore, considering the performance degradation of kd-tree based nearest neighbor search algorithms on high dimensional data, dimensionality reduction on SIFT decsriptors also contributes to the overall localization performance. We then input the low dimentional data to the FLANN based matcher [32] described in Section 2.4.



(a) Cumulative Singular Value Plot (b) Low-Dim Projection Performance

Figure 4.12: Principal Component Analysis on Carpet Data

4.3.2 Image Compression

Explicitly storing all the image data for a large environment takes up too much space in practice. Therefore, we use image compression algorithms to compress our preprocessed image data. We first quantize the environment into partially overlapped bins. Then the image patch in each bin is compressed using 2D discrete wavelet compression algorithm [5]. The basic idea is to recursively perform wavelet transform on the image where we only store the differences while passing on the sum to the next level. One example of wavelet compression used in JPEG2000 is shown in Figure 4.13. The three large images are derived from applying high pass filtering to the original image. Then the original image is low pass filtered and downsampled to generate the next level. By repeatedly alternating between these two steps, we can achieve a predefined compression ratio. However, there is still a tradeoff between how much we can compress and the reconstruction quality. In this work, we haven't thoroughly explored the factor of compression ratio to our localization performance.

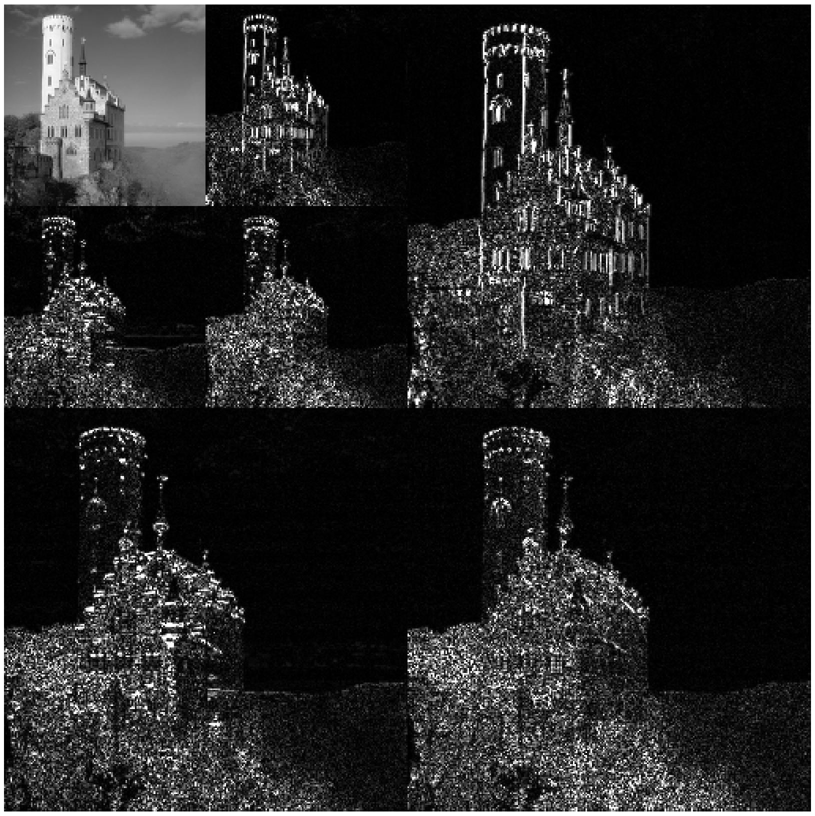


Figure 4.13: Wavelet Compression

The decompression process can be performed dynamically as we move around the environment. In the outdoor scenario, we can use GPS or landmarks to acquire a rough guess on the initial position such that we only need to expand the image data associated within the neighborhood. On the other hand, for indoor cases, we can use sensors or heuristics, such as which floor or room we are currently in, to initialize the search space for our localization system. After we acquired the initial position, we can dynamically decompress and fetch the nearby image(s) as we move around.

4.4 Matching Process

The major component of computer vision based localizaiton system is feature matching. In natural images, feature matching using SIFT descriptors usually performs reasonably well. However, our experiments found that in the context of near-random textures with lighting and slight surface geometry changes, naively choosing the closest nearest neighbor match on SIFT descriptors may not always yield expected precision. Therefore, we propose a two-stage matching algorithm shown in Figure 4.14. In this section, we will decribe the matching process in more details.

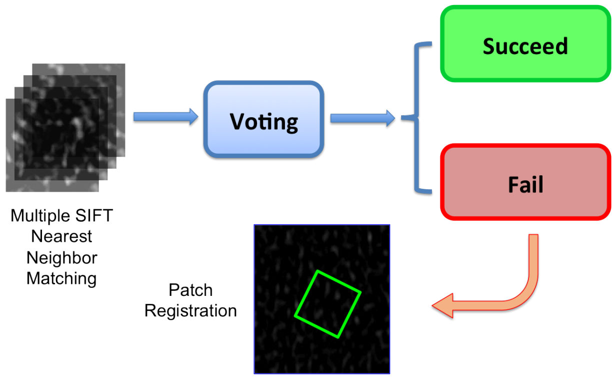
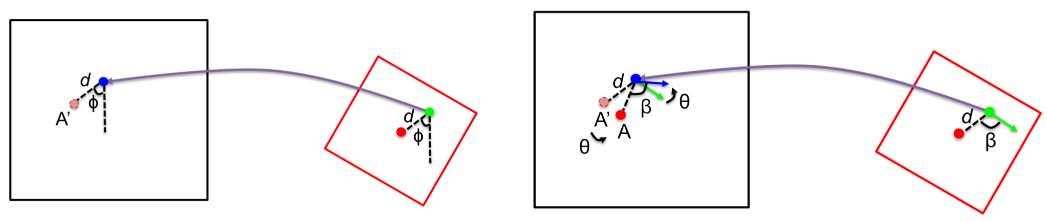


Figure 4.14: Matching Process

4.4.1 Nearest Neighbor Matching

For each testing patch, we can extract multiple keypoints. We perform nearest neighbor search for each of them using FLANN based ANN matcher [32]. Then, for each matching pair, we use the position of the matching keypoint to predict the central position of an arbitrary testing patch in the known environment.

Figure 4.15 shows a situation where the green SIFT feature in the red testing patch on the right matches to the blue keypoint in the environment on the left. If we only use the relative position from the green keypoint to the red patch center, i.e. the angle ϕ and distance d, we will predict the matching patch center at A' as in Figure 4.15a. However, by using the orientations assigned to the green and blue keypoint (the green and blue arrows), we can recover the rotation offset θ and correct the prediction to A (Figure 4.15b).



(a) Nearest Neighbor Matching (b) Matching with Angle Correction

Figure 4.15: Nearest Neighbor Prediction

4.4.2 Voting

Observed that the closest nearest neighbor may not always yield a prediction close enough to the ground truth, a voting strategy is used to identify possible consensus among different nearest neighbor predictions. Given the nature of near-random textures, a wrong match could potentially map a testing patch to any position in the environment with equal probability. Therefore, when multiple predictions agree with each other, it is highly likely that their predictions would be close to the ground truth. There are many possible algorithms to implement voting. For instance, we can quantize space into bins and let each of the SIFT predictions vote for one bin and the nearby 8 neighbors. Then we can use a hashtable to store the votes. In this work, we propose a simple algorithm described in Algorithm 1.

The idea is to calculate the pairwise distances from a SIFT prediction candidate $\boldsymbol{n}$ to the others. If the pixel-wise distance between candidate $\boldsymbol{n}$ and $\boldsymbol{m}$ is below some threshold τ_{dist}, we count as one vote from $\boldsymbol{m}$ to $\boldsymbol{n}$. Finally, if any candidate gets enough votes to pass the threshold τ_{vote}, we will consider the voting successful.

Algorithm 1 Nearest Neighbor Voting

```
procedure NNVoting(points[], $\tau_{\text{vote}}, \tau_{\text{dist}}$)
    n ← Length(points[])
    marked[] ← true
    maxVoteCount ← 0
    minAvgDist ← $\infty$
    for i = 1 to n do
        if marked[i] == false then Continue
        (votes[], center, voteCount) ← Vote(i, points[], marked[], $\tau_{\text{dist}}$)
        if voteCount == 0 then
            marked[i] ← false
        else if voteCount $\geq \tau_{\text{vote}}$ then
            avgDist ← GetAvgDist(votes[], center)
            if avgDist < minAvgDist then
                minAvgDist ← avgDist
                maxCenter ← center
                maxCount ← voteCount
    return maxCount, maxCenter, minAvgDist
```

Algorithm 2 Voting

```
procedure Vote(i, points[], marked[], $\tau_{\text{dist}}$)
    n ← Length(points[])
    center ← $(0, 0)$
    for j = 1 to n do
        if marked[j] == false then Continue
        dist ← Distance(points,points[j])
        if dist $\leq \tau_{\text{dist}}$ then
            voteCount ← voteCount + 1
            center ← center + points[j]
            Add(votes[], points[j])
    center ← center / Length(votes[])
    return votes[], center, voteCount
```

We also group each candidate $\boldsymbol{n}$ and its voters into a set of predictions $\boldsymbol{V_n}$ and calculate the average distance d_n from the gravity center c_n to each of member in $\boldsymbol{V_n}$.

In case of ties in voting, we would prefer the one with smaller within-group average distance. In addition, d_n is also used as the search range to refine the prediction of c_n using template matching.

The time complexity of Algorithm 1 is $O(n^2)$, yet if n is small, e.g. 10 or 20, the performance may not compromise too much comparing to the cost for computing hash values in the hashtable implementation. The parameters for this algorithm are estimated from experiments. Empirically, if voting only involves the first 10 SIFT predictions, the typical parameter values are $\tau_{\text{vote}} = 2, \tau_{\text{dist}} = 15$.

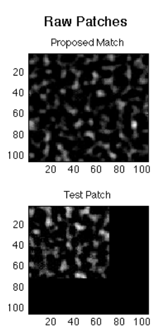
4.4.3 Template Matching

In case of voting failure, we fall back to the idea of applying template matching around every SIFT predictions, i.e. given a patch clipped around a SIFT prediction as template $\boldsymbol{T}$, we want to register it to an image $\boldsymbol{I}$. In this work, we choose Normalized Cross Correlation (NCC) as the matching distance metric Eq. 4.10. However, calculating NCC score on intensities in Cartesion space is subject to orientation and scale. Therefore, we attempted two different approaches to compensate for this drawback.

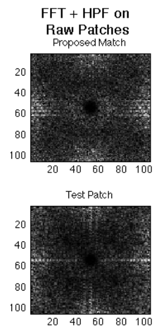
$$NCC(x,y) = \frac{\sum_{x',y'}(\boldsymbol{T}(x',y') \cdot \boldsymbol{I}(x+x',y+y'))}{\sqrt{\sum_{x',y'} \boldsymbol{T}(x',y')^2 \cdot \sum_{x',y'} \boldsymbol{I}(x+x',y+y')^2}} \tag{4.10}$$

Fouier-Mellin Transform

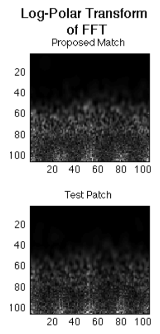
To achieve rotation invariance, we can apply polar transform as described in Section 4.1.2. For scale invariance, we can perform sampling in log space. Therefore in log-polar space, rotation and scale become translations, which can be effeciently recovered using Fast Fourier Transform (FFT). To overcome the initial translation in Cartisian space, we can apply FFT on original image patches as well. Hence, we yield the entire pipeline of Fourier-Mellin Transform [11] as demonstrated in Figure 4.16.



(a) Input



(b) Fourier Transform



(c) Log Polar Transform

Figure 4.16: Fourier-Mellin Transform

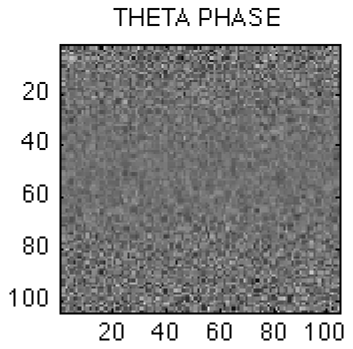
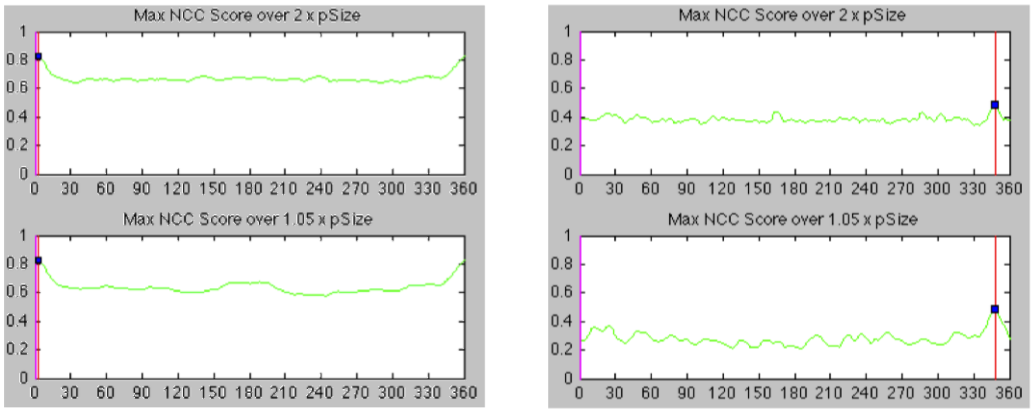


Figure 4.17: Phase Image

To start with, we first pad zeros to the template to make it the same size as the matching image patch. To reduce the additional low frequency from padding (Figure 4.16a), we apply a high pass filter after Fourier transform (Figure 4.16b). Log-polar transform will map rotation and scale into translations as in Figure 4.16c. Then, we can apply phase correction using FFT. However, this algorithm discards the rotation information recovered from SIFT matching. Moreover, our experiments show that the global optimum in the phase image (Figrue 4.17) sometimes diverges from the ground truth. One possible explanation is that the confusions caused by the nature of near-random textures increase the difficulty for matching. In addition, the excessive cutoff on low frequency information after high pass filtering may also be another reason to consider. Granted, one can argue that by carefully selecting the kernel size of the high pass filter, this method is still possible to work.

Brute Force Search

An alternative approach is to apply brute force search for the correct rotation offset by rotating the testing patch with bilinear interpolation on subpixel resolution. However, if we search over all possible rotations, i.e. 0 to 2π, we found that the global maximum NCC score does not seem to be a good indicator for the actual prediction error. Fortunately, our empirical study suggests that the rotation recovered from SIFT prediction $\hat{\theta}$ is often close enough to the ground truth θ. In addition, the local maximum of the NCC score around $\hat{\theta}$ can be used to select the best SIFT prediction with reasonably small error.



(a) NCC Curve with No Rotation (b) NCC Curve with Rotation

Figure 4.18: NCC Curve from Brute Force Search in Rotation

One example of such NCC curve is shown in Figure 4.18. The x axis denotes the rotation on testing patch whereas the y axis labels the NCC score. The red vertical line indicates the SIFT predicted rotation $\hat{\theta}$. The upper two plots correspond to template matching over a larger area around SIFT prediction (around 30 to 35 pixels radius in offset). The bottom two plots depict the same template matching scenario but within a much smaller neighborhood (around 3 to 5 pixels radius in offset).

In an easier case with only translation Figure 4.18a, we observe that $\hat{\theta}$ coincides with the ground truth, i.e. 0 degree. Furthermore, searching over different sizes of neighborhood does not affect much to the shape of the NCC curve. On the other hand, in Figure 4.18b, the ground truth rotation θ is around -20 degrees whereas the local maximum of NCC score occurs a bit left to the SIFT predicted rotation $\hat{\theta} =$ -15, yet it is still not too far away from θ. Nonetheless, the lower and more flactuated NCC curve could result from slight surface geometry changes and subpixel interpolations while rotating the testing patch.

Despite of the differences in the two situations above, we notice that there is always a local peak near SIFT predicted rotation $\hat{\theta}$ (the red vertical line in all four plots). Therefore, to improve the time efficiency, we restrict the search neighborhood for rotation to be $(\hat{\theta} - \delta, \hat{\theta} + \delta)$. More interestingly, the maximum NCC score within a smaller window also tends to be more robust than that in global scale. Our experiments found that sometimes the global maximum NCC score from a wrong SIFT prediction can overrun that from the correct one.

4.4.4 Learning Error

After manually examined a collection of NCC curves accociated with correct SIFT predictions, we found some common shape characteristics shared among them. A close up look is shown in Figure 4.19.

From the observations, we can make the following hypothesis: for a good SIFT prediction, there is usually a salient peak θ_{max} near the SIFT prediction $\hat{\theta}$. Then, to testify this hyphothesis, our first experiment is to train a linear regression model mapping the curve itself to the corresponding prediction error value. However, the experiment was unsuccessful due to low signal to noise ratio in the NCC data and lack of clear correspondences between NCC curve shapes and exact error values.

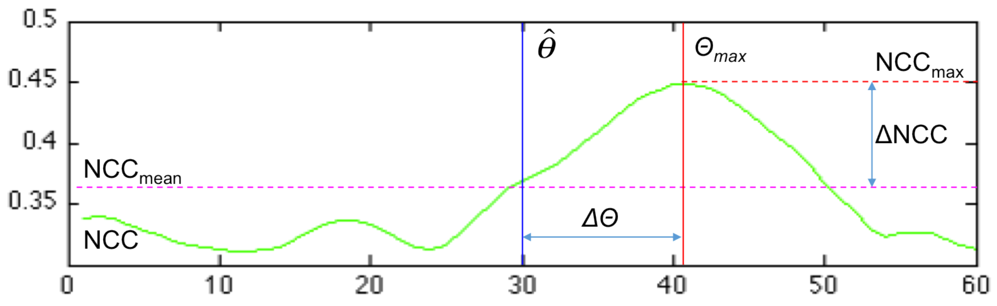


Figure 4.19: NCC Curve Close Up

Fortunately, in the context of precise localization, what we care about the most is whether a prediction will generate error small enough. Therefore, we can simplify the continuous regression problem to a binary classification problem. To state formally, given a NCC curve from template matching over rotations, classify whether the prediction error is small or large comparing to a predefined threshold.

In this work, we select two different kinds of classifiers: logistic regression [7], which produces linear decision boundary, and neural network, [36] which is a more sophisticated non-linear classifier. We also conduct a comparison study on using different kinds of input as the training data: raw NCC curve, simple heuristics (e.g. $\Delta\theta$ and ΔNCC in Figure 4.19), derivatives and cum-sum features.

Cum-Sum Feature

To better characterize the behavior of the NCC curve, we also designed a feature called "cum-sum" feature as an alternative to the raw NCC curve as training input (Figure 4.20). We first smooth the raw NCC curve (upper-left plot) and subtract the mean (bottom-left plot). Then, we apply a Gaussian weight (upper-right plot) and use Riemann sum to approximate the area under the curve. The cumulative summation begins from the center, i.e. the SIFT predicted rotation $\hat{\theta}$, and then grows on both sides to generate the cumulative sum curve (bottom-right plot). Therefore, if the hypothesis is true, the cum-sum curve would increase rapidly at first, then gradually become flat and finally end with large overall sum.

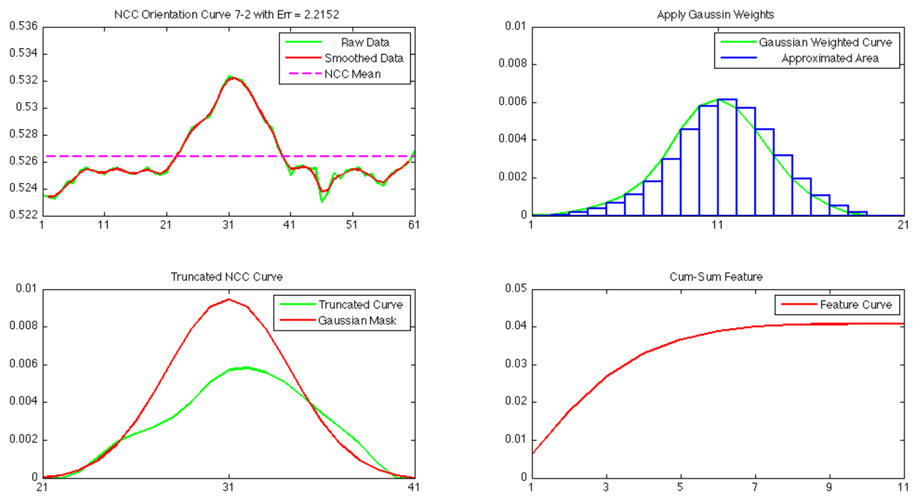


Figure 4.20: Cumulative Sum Feature Generation

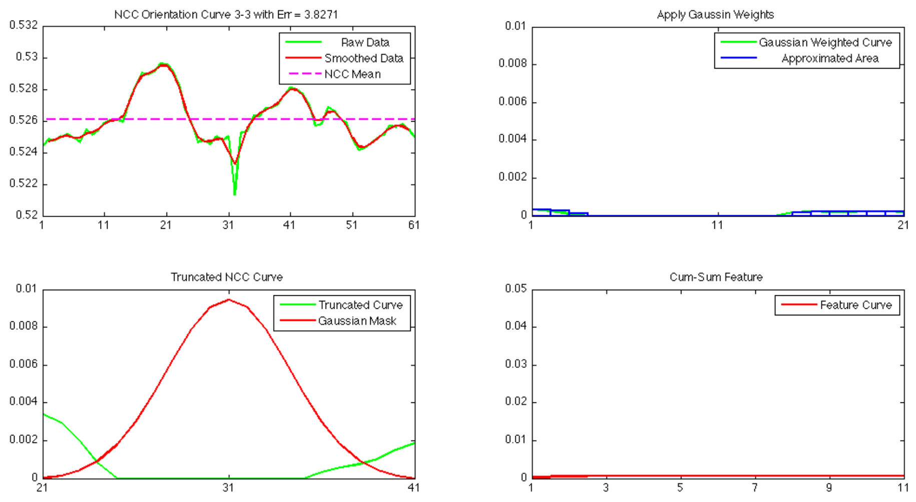
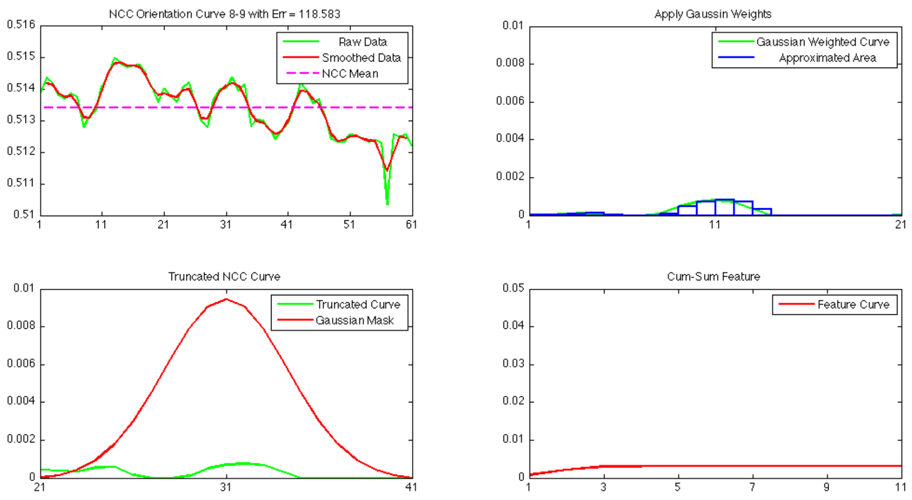
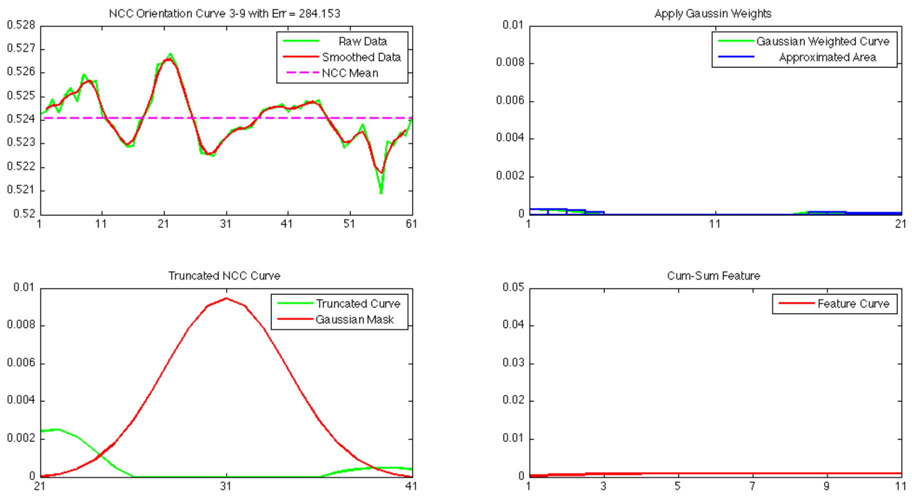


Figure 4.21: Cum-Sum Feature Failure Case

Our experiment shows that the cum-sum feature can very well characterize our hypothesis. For a positive example with prediction error 2.21 pixels in Figure 4.20, the cum-sum curve behaves correctly as described before. On the other hand, when the NCC curve is flactuating (Figure 4.22a with error 118.58) or the nearest peak is far away from $\hat{\theta}$ (Figure 4.22b with error 284.15), the corresponding cum-sum feature would end up with smaller final sum as we expected.



(a) NCC Curve with Flactuation



(b) NCC Curve with Futher Mode

Figure 4.22: Cum-Sum Feature on Negative Cases

Unfortunately, the size of the Gaussian weight kernel still requires to be estimated. This parameter is essentially the estimated error of SIFT rotation prediction which we found a little hard to fit in practice. One failure case is shown in Figure 4.21.

Logistic Regression

Unlike linear regression which produces unbounded continuous predictions, logistic regression uses the logistic function (also sometimes referred to as the sigmoid function) in Figure 4.23 to map any number in $\mathbb{R}$ to $[0, 1]$. Similar to linear regression, each dimension of the input data x_i is weighted by a coefficient β_i and then a bias β_0 is added before feeding to the logistic function as in Eq. 4.11.

$$y = \frac{1}{1 + e^{-(\beta_0 + \boldsymbol{\beta x})}} \tag{4.11}$$

The parameters $\beta_0, \boldsymbol{\beta}$ can be learned using maximum likelihood estimation (MLE). Unfortunately, the formulation of the logistic function prevents us from finding a closed-form solution to the coefficients through MLE. Therefore, we apply gradient descent to iteratively learn the parameter values.

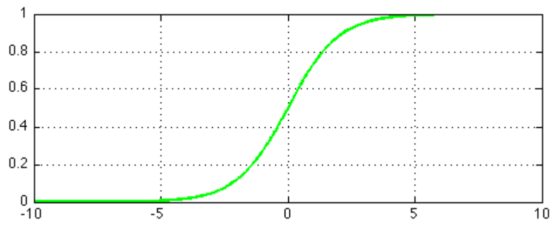


Figure 4.23: Logistic Function

Neural Network

A more complicated non-linear classifier is neual network [36]. The basic structure is depicted in Figure 4.24. In this example, there are three layers of nodes in this network, input layer, hidden layer and output layer.

Each dimension of the input data is fed into each input node in the input layer. Then we compute a linear combination of each input node and use them as the input for the hidden layer. The non-linearity is added by putting each node in the hidden layer through a non-linear activation function, usually sigmoid or arctangent. Finally, a linear combination of the output from the hidden layer is fed into each node in the output layer as the network outputs. To form a binary classifier, we can utilize softmax to convert the outputs into a probability distribution.

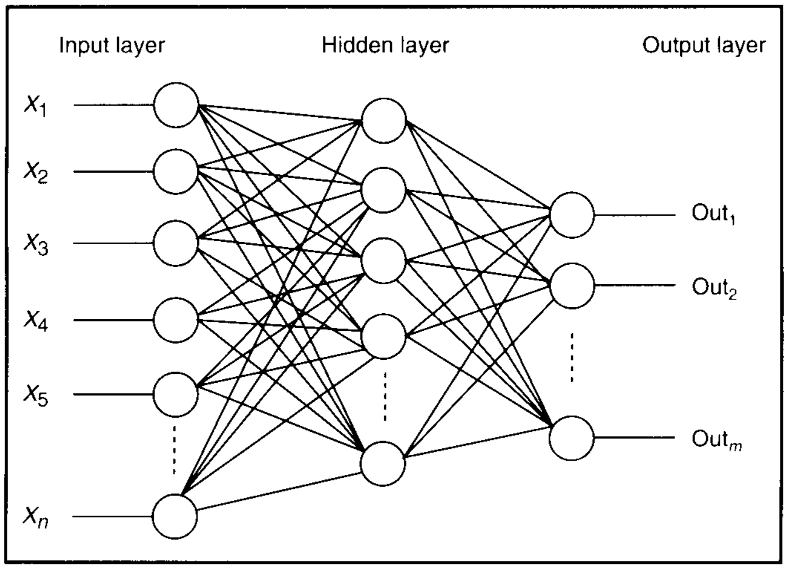
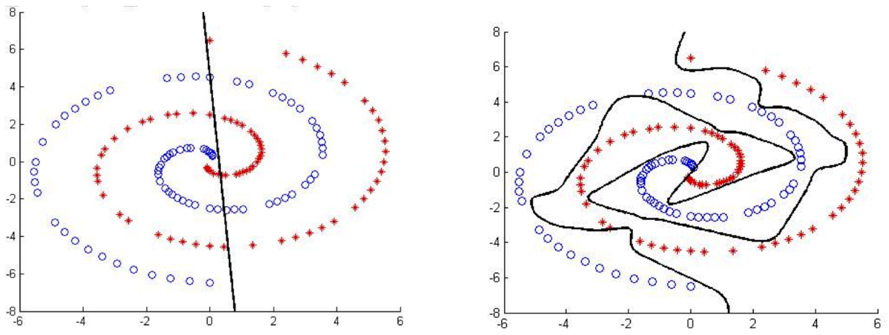


Figure 4.24: Neural Network



(a) Linear Decision Boundary (LR) (b) Non-Linear Decision Boundary (NN)

Figure 4.25: Comparision between Logistic Regression and Neural Network

Same as the learning process for logistic regression, we need to use MLE to fit the model parameters, i.e. the weights between each pair of nodes. We use gradient descent with backpropagation [36] to train the network. Comparing to logistic regression, neural network can produce much more complicated decision boundary. One example is shown in Figure 4.25 (Figure 4.25a by logistic regression and Figure 4.25b by neural network).

Although the non-linearity enables us to separate data that are not linear separable, this gain in learning power also comes with the risk of overfitting. To reduce this risk, we performed n-fold cross validation [7] as a model selection mechanism.

Emsemble Method

Having experimented with both classifiers, we discovered that the improvement in localization performance is not significant (the detailed results will be further analyzed in Section 5.2). By examining the failure cases during classification, we found the two classes are largely overlapped. There seems to be no clear linear or non-linear decision boundary that could easily separate one from the other. Thus, an overkilled approach would be to use the ensemble method. The pipeline is shown in Figure 4.26.

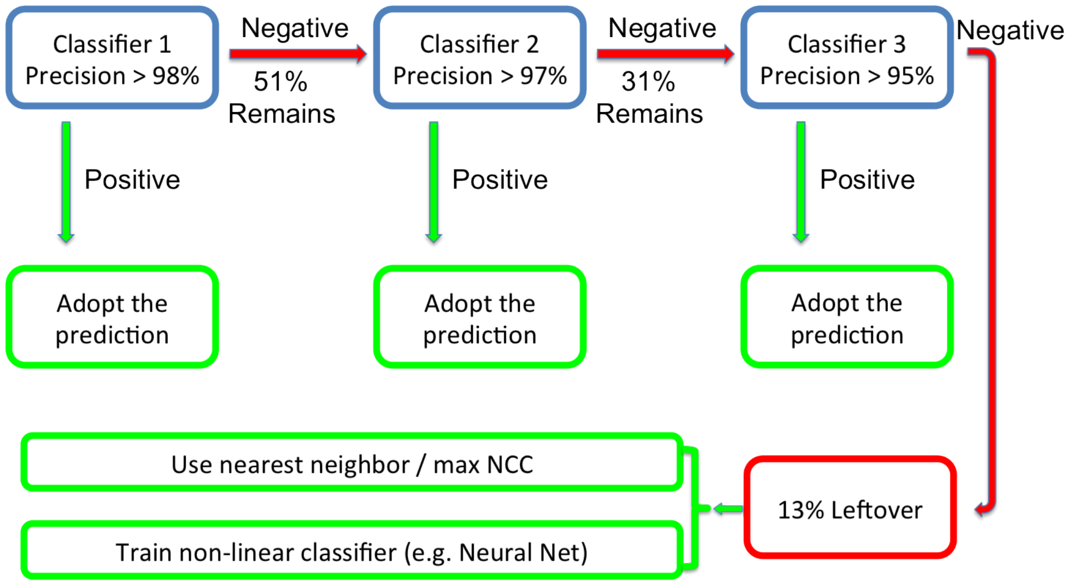


Figure 4.26: Pipeline of Ensemble Method

The idea is to collect a series of classifiers each learning different aspects of the data in a cascading fashion. Then, a testing example is piped through each classifier one by one and accepted immediately at the one having high positive confidence on.

In the context of our system, we use logistic regression as the base classifier and interpret their continuous outputs as confidence levels. Nevertheless, thresholding on high confidence yields higher precision in prediction but at the cost of lower recall. As a consequence, each of the classifier may not be able to cover many positive examples during learning. In our experiment, we focus on the top 20 SIFT matches from 1000 randomly selected patches in the testing image. If there is any high confidence positive prediction from a classifier c_i among the 20 SIFT matches for a patch p_n, we consider patch p_n to be accepted by the classifier c_i. As it turns out, we can train the first classifier c_1 with over 99% precision but only 67% recall, yet with an acceptance rate of 49%. In other words, 49% of the training cases can be successfully accepted by c_1. Then, we train the second classifier c_2 on the remaining 51% training data and repeat this process on the leftover data from c_2. We stop adding new classifiers until the acceptance rate is less than 50% on the current training set with precision rate over 95%. To clean up the remaining data (usually 13% to 17% after three rounds of training), we can either greedily select the one with the maximum NCC score or again train a non-linear classifier on them e.g. neural network. The performance of this approach will be discussed in further details in the next chapter.

Chapter 5

Result

In this chapter, we are going to discuss the preformance of the proposed system from some experimental results. We will first describe and analyze the baseline implementaion which always predicts according to the closest SIFT nearest neighbor match. Then we will compare it to our whole system pipeline on more challenging cases.

5.1 Baseline Implementation

Surprisingly, even without preprocessing and the final matching mechanism, greedily selecting the closest nearest neighbor from SIFT matches could already achieve high precision in localization predictions. The summary statistics are listed in Table 5.1[1]. The testing image pairs are P-Carpet Figure A.1, Wall Figure A.2, Wood Figure A.3 and Concrete Figure A.4. In addition, we compare the preformances of different feature detectors as mentioned in Section 4.1.1 with the SIFT descriptor. To obtain an estimated upper bound on how well these detectors could perform, we intend to set the quality threshold for keypoints lower, e.g. the response in Harris Corner detector, the number of nearby layers in MSER and the contrast threshold in SIFT.

[1]The measurement in actual scale is up to pixel level accuracy only; all experiments reported in this chapter are conducted with at least 3000 testing cases

Our experiments suggest that Harris Corner detector can usually detect more keypoints from the input image, yet all of them are of the same size. MSER detector is more sensitive to image contrast and usually focuses on small features. SIFT detector appears to be more compatible with SIFT descriptor considering that it often extracts less features yet providing competitive performance. In terms of running time, both Harris Corner and MSER detectors are often faster than SIFT, yet none of the three goes beyond any reasonable time limit.

Data Images	**Harris Corner**	**MSER**	**SIFT**
Number of Features Detected			
P-Carpet	**7007**	6444	5253
Wall	**8200**	23	7968
Wood	**6900**	156	2925
Concrete	**7854**	3676	2615
Average Prediction Error (mm)			
P-Carpet	1.34 ± 0.01	81.34 ± 0.79	$\mathbf{1.19 \pm 0.01}$
Wall	0.47 ± 0.01	–[2]	$\mathbf{0.40 \pm 0.01}$
Wood	2.91 ± 0.36	65.73 ± 1.25	$\mathbf{2.47 \pm 0.08}$
Concrete	$\mathbf{1.96 \pm 0.02}$	220.13 ± 2.83	6.03 ± 0.72
Small Error Percentage (error $\leq$ 5 pixels)			
P-Carpet	$\mathbf{100.00 \pm 0.00\%}$	$63.11 \pm 0.51\%$	$\mathbf{100.00 \pm 0.00\%}$
Wall	$\mathbf{99.76 \pm 0.20\%}$	–	$\mathbf{99.63 \pm 0.12\%}$
Wood	$\mathbf{94.75 \pm 0.43\%}$	$8.89 \pm 0.37\%$	$93.76 \pm 0.17\%$
Concrete	$\mathbf{98.75 \pm 0.19\%}$	$11.75 \pm 0.24\%$	$97.03 \pm 0.44\%$

Table 5.1: Localization Performance on Baseline Implementation

We evaluate the performance of localization using two measurements. One is the average error from the predicted position to the ground truth; the other is the percentage of smaller error, i.e. predictions within 5-pixel radius to the ground truth. For the P-Carpet and Wall image data, both Harris Corner and SIFT can produce high precision prediction results. Harris Corner detecter outperforms SIFT on Concrete

[2]No statistics for MSER performance due to insufficient number of features detected

data mainly due to more detected keypoints. However, none of the three detecters performs equally well with Wood data, though most of the times Harris Corner and SIFT would generate predictions sufficiently close to the ground truth.

Unfortunately, these preliminary results appear to be a little too optimistic. In ideal situations where there are almost no lighting or surface geometry changes, this simple baseline implementation could achieve satisfactory accuracy. But, when more variations are involved between testing patch and the knowledge base, the performance degrades a lot. We will discuss this issue in further detail later.

Feature Selection

We also evaluate our feature selection algorithm using the baseline implementation. As described in Section 4.2, there are two main parameters for the selection process, N the number of overlapped windows in one row ($N \times N$ of them in total) and m the number of features we want to keep within each window. The summary plot on the P-Carpet data is shown in Figure 5.1[3].

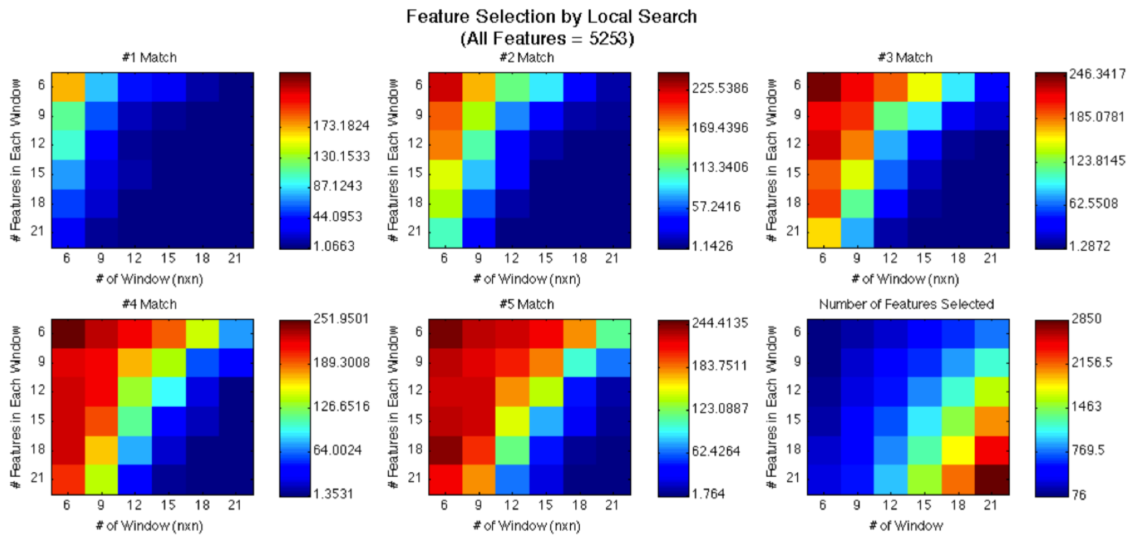


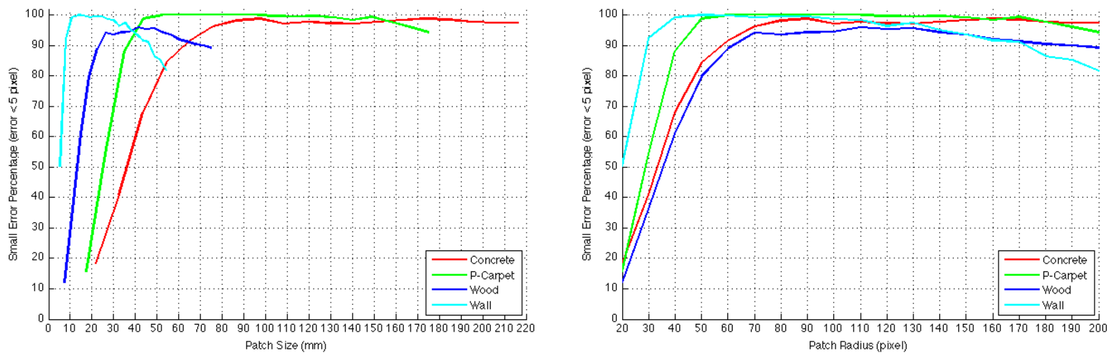
Figure 5.1: Feature Selection Performance

[3]#i Match denotes the i^{th} nearest neighbor match of SIFT predictions

From 5253 detected SIFT features, the local search algorithm selects 2125 of them to achieve average prediction error of 1.37 using the closest nearest neighbor matching. This result is generated from selecting 21 features within each 18×18 halfly overlapped windows. In addition, we notice that the prediction performance drops down as the nearest neighbor matching error in feature space goes up. This observation indicates that the ordering of SIFT nearest neighbor matches may correspond to the prediction error they generate.

Patch Size Selection

Another important parameter we need to estimate is the the size of the testing patch. The desired patch size may vary according to different properties of the surface. The size summary plot of the four image data we experiemented is shown in Figure 5.2.



(a) In Actual Scale (mm)   (b) In Relative Scale (pixel)

Figure 5.2: Testing Patch Size Plot in Diameter

Notice that in both plots, increase in testing patch size may not always bring performance boost. One common cause is that there are always some high quality false positive matches in each image data, usually in high frequency. Therefore, larger testing patch size would be more likely to include such matches. Because these false matches having smaller error in feature space, they often will become the closest nearest neighbor and overrun other "correct" nearest neighbor predictions.

On the other hand, different image data do have their unique additional reasons for this performance degradation as well. For instance, we observed that for Wall data (cyan line), when testing patch size goes up, some larger scale features are detected and selected as the closest nearest neighbor. Most of the time, they will match to the ones with similar feature size in the knowledge base, yet sometimes they could also match to features with much smaller size. This could be due to the fact that the resolution of SIFT descriptor may not be sufficiently high to capture enough details in larger scale. Since the plain wall is uniformly colored, there are not much helpful patterns we can extract from color contrast. Thus, potentially there could be confusions among observations at different scales.

Another interesting observation is that from relative scale (Figure 5.4b), the irregular high frequency patterns from the surface geometry in the Wall data improve the distinctiveness of a single patch even with small radius. Unfortunately, our following experiments show that due to lack of color contrast, image features purely created by shadows can be very sensitive to light directions. One example is shown in Figure 5.3. In this case, the stitching result could not be generated without markers.

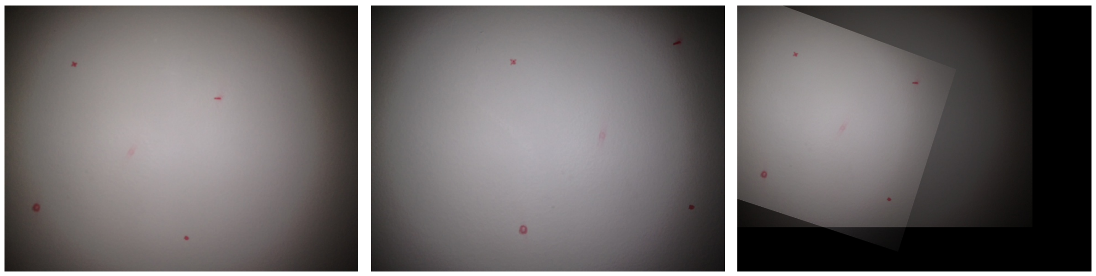

(a) Training Data (b) Testing Data (c) Stitch Result

Figure 5.3: Wall Data with Flash Light

The SIFT features detected in this data highly depend on lighting and surface geometry. We can get a more clear view when we remove the background color using high pass filter to reveal the circular structure around the highlight spot (Figure 5.4).

As a result, the captured surface changes its appearance as light source moves around, which makes image feature matching problematic.

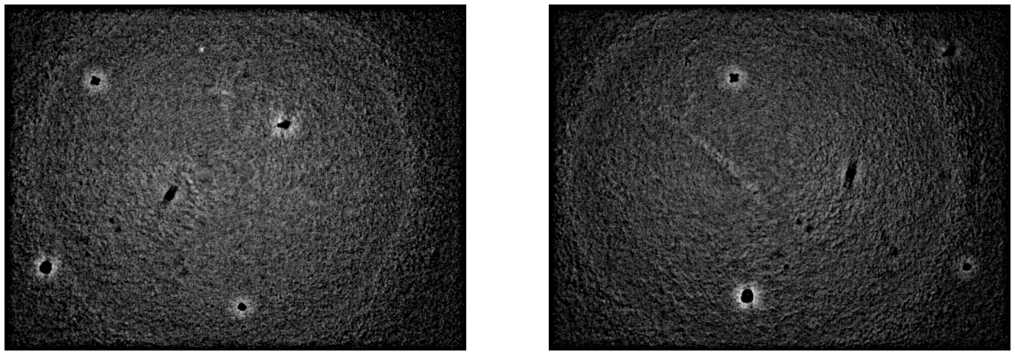

(a) High Pass Filter on Training Data (b) High Pass Filter on Testing Data

Figure 5.4: High Pass Filter on Wall Data with Flash Light

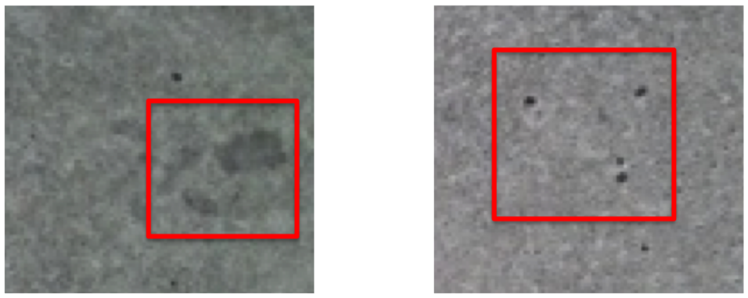

(a) Color Difference (b) Surface Concavity

Figure 5.5: Concrete Surface Close Look

Comparing to Wall data, Concrete data (red line) seems to be more robust to patch size changes. If we look closely, there are some irregular patterns caused by color differences on the surface (Figure 5.5a). Moreover, we also found that surface concavities are often helpful too (Figure 5.5b). Because of the concave geometry, not enough light can be reflected into the camera. Therefore, we can observe some dark dots distributed irregularly across the surface. Since these dots are usually darker than its surrounding area, they can be easily detected. Furthermore, the unique patterns they formed can be captured by SIFT descriptors. However, the distribution of these "marker-like" dots are often sparse. Thus, it would require observations at larger scale to cover at least some of them for localization purpose.

5.2 System Evaluation

Even though the initial results using the simple baseline implementation is encouraging, when more variations happen on the lighting conditions and surface geometry, greedily selecting the closest nearest neighbor does not always appear to perform well. In this section, we will discuss experiments on more challenging image data and evaluate the whole pipeline we proposed.

Preprocessing

As mentioned in Section 3.1.1, the purpose of applying band pass filtering is to reduce the noise and adjust for the overall intensity changes due to lighting. We repeated the same experiment on the four data sets we used to evaluate the baseline implementation. The resulting statistics are listed in Table 5.2[4].

(a) Concrete Patch without Band Pass Filter (b) Concrete Patch with Band Pass Filter

Figure 5.6: Comparison on Band Pass Filtering

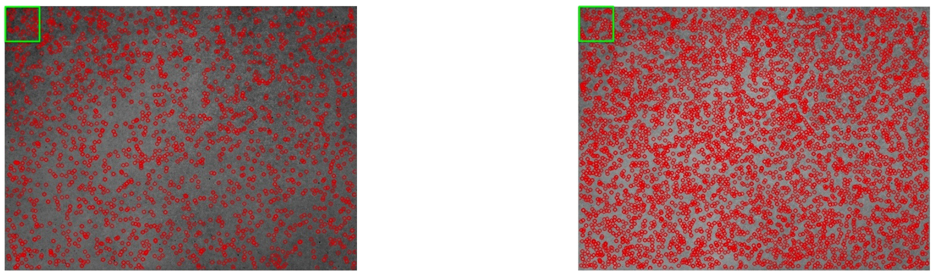

(a) Detected Keypoints without Band Pass Filter (b) Detected Keypoints with Band Pass Filter

Figure 5.7: Comparison on Keypoint Distribution (Green box indicates patch size)

[4]We only include experiments using the SIFT detector and descriptor for space-saving purpose

Data Images	Band Pass Filter	No Band Pass Filter
	Number of Features Detected	
P-Carpet	**9158**	5253
Wall	**10565**	7968
Wood	**5973**	2925
Concrete	**6077**	2615
	Average Prediction Error (mm)	
P-Carpet	$\mathbf{1.08 \pm 0.01}$	1.19 ± 0.01
Wall	$\mathbf{0.40 \pm 0.01}$	$\mathbf{0.40 \pm 0.01}$
Wood	2.99 ± 0.38	$\mathbf{2.47 \pm 0.08}$
Concrete	$\mathbf{1.92 \pm 0.02}$	6.03 ± 0.72
	Small Error Percentage (error $\leq$ 5 pixels)	
P-Carpet	$\mathbf{100.00 \pm 0.00\%}$	$\mathbf{100.00 \pm 0.00\%}$
Wall	$\mathbf{99.93 \pm 0.04\%}$	$\mathbf{99.63 \pm 0.12\%}$
Wood	$\mathbf{95.26 \pm 0.56\%}$	$93.76 \pm 0.17\%$
Concrete	$\mathbf{99.83 \pm 0.05\%}$	$97.03 \pm 0.44\%$

Table 5.2: Comparison on Preprocessing Using Band Pass Filter

Our first observation is that the increase in number of features detected. One possible reason is that after removing some low frequency information (sometimes corresponding to the base color of the surface as in Figure 5.6), details become much more salient. In addition, we discover that for the same testing patch, the matching error in feature space for the closest nearest neighbor decreases after band pass filtering. Therefore, it seems that the newly detected features are more distinctive and stable than what we had before.

Besides band pass filtering, we did not find intensity normalization process necessary i.e. Histogram Equalization or Luminance Remapping, since SIFT is based on gradients which makes it robust to overall intensity level changes.

Voting

Unfortunately, the closest nearest neighbor may not always yield a close prediction to the ground truth, especially while facing lighting changes and data corruptions. For instance, let us consider the same concrete surface Figure A.4a (Figure A.5a) as in the previous experiment. If the testing patch is captured from different time (Figure A.5b, which is taken after more than half a year later from Figure A.5a), the baseline implementation tends to perform worse (Table 5.3). Therefore, instead of the greedy strategy, we decide to employ a collaborative approach, i.e. voting. Our voting experiment only involves the top 20 closest nearest neighbor SIFT predictions with over 3 votes as success threshold τ_{vote} as described in Section 4.4.2.

Data Images	**Voting**	**Without Voting**
	Average Prediction Error (mm)	
Concrete2	84.41 ± 2.83	84.47 ± 2.97
Concrete2 (BPF[5])	28.90 ± 0.54	34.6 ± 0.30
	Small Error Percentage (error $\leq$ 5 pixels / 5.47mm)	
Concrete2	$59.85 \pm 0.42\%$	$60.94 \pm 0.24\%$
Concrete2 (BPF)	$\mathbf{85.52 \pm 0.35\%}$	$80.08 \pm 0.17\%$
	Small Error Average (mm)	
Concrete2	2.52 ± 0.12	2.61 ± 0.14
Concrete2 (BPF)	$\mathbf{1.69 \pm 0.05}$	2.58 ± 0.07

Table 5.3: Performance Summary on Voting Strategy

In the case with band pass filtering (BPF), i.e. second rows, around 60% of the times voting will succeed; otherwise, the closest nearest neighbor is selected as final prediction. In contrast, without BPF, the voting success rate is merely above 10%. This result indicates that pre-filtering forces the SIFT detector to focus on the actual near-random textures rather than unstable high frequency details; therefore more consensus can be established among different SIFT nearest neighbor predictions.

[5]BPF: preprocessed with Band Pass Filtering

Template Matching

Observed that the size of SIFT features extracted from testing patch may not be large enough to cover the entire patch. Thus, only considering partial information from the patch via SIFT feature matching could be misleading. This may not be an issue if there are less environmental changes between the image we builing our knowledge base from and the one we using for testing. But, when changes in lighting or surface conditions are involved, we may not be able to extract the same features as we seen before. Therefore, the localization performance may suffer from only trusting the SIFT nearest neighbor predictions. Granted, voting strategy would leverage this drawback by collectively considering multiple matches, yet this inconsistency in extracted features will also jeopardize the voting success rate. Therefore, in situations where voting fails and cloest nearest neighbor may not be reliable, we need an additional mechanism to help us choosing the "correct" prediction if there is any.

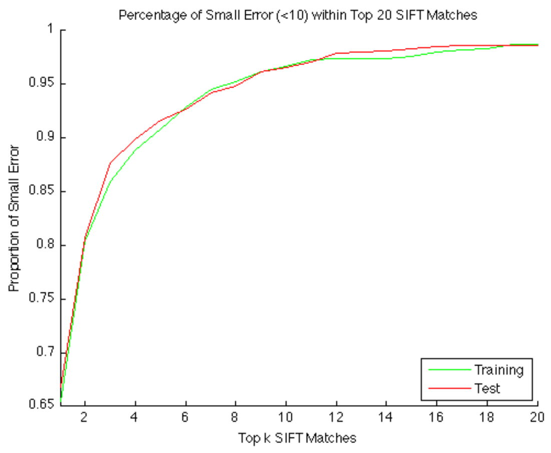


Figure 5.8: Small Error Percentage among First k SIFT Predictions

Following the idea of matching the entire testing patch as a whole rather than part of it, we apply template matching techniques to solve this problem. As described in Seaction 4.4.3, we choose NCC as our distance metric. However, performing template matching over the entire environment may not be practical due to time efficiency.

Thus, we only focus on small neighborhoods around each of the SIFT nearest neighbor predictions. The reason why we can use SIFT predictions to narrow down the search space is based on the observation in Figure 5.8. Notice that around 99% of the time, the top 20 SIFT matches will contain at least one prediction close enough to the ground truth. Therefore, we can even further narrow down the search by only focusing on top 20 SIFT matches instead of all of them, which could be up to 80 or more. Additionally, to compensate for the fact that template matching in image space is not robust to rotation, we utilize the SIFT predicted rotation as a prior. Then, we can rotate the testing patch within a small neighborhood aroung this prior and generate the NCC curve as Figure 4.18.

Data Images	**V-MaxNCC**[6]	**MaxNCC**	**V-CNN**[7]
	Average Prediction Error (mm)		
Concrete2	214.77 ± 2.23	240.62 ± 0.68	84.41 ± 2.83
Concrete2 (BPF)	$\mathbf{25.36 \pm 3.15}$	36.75 ± 3.17	28.90 ± 0.54
	Small Error Percentage (error $\leq$ 5 pixels / 5.47mm)		
Concrete2	$15.94 \pm 0.16\%$	$8.10 \pm 0.44\%$	$59.85 \pm 0.42\%$
Concrete2 (BPF)	$\mathbf{91.61 \pm 0.84\%}$	$88.52 \pm 1.15\%$	$85.52 \pm 0.35\%$
	Small Error Average (mm)		
Concrete2	$\mathbf{2.36 \pm 0.16}$	2.52 ± 0.05	2.52 ± 0.12
Concrete2 (BPF)	$\mathbf{1.32 \pm 0.01}$	$\mathbf{1.31 \pm 0.01}$	1.69 ± 0.05

Table 5.4: Comparison between Max NCC and Voting Strategies

One greedy strategy is to select the match who possesses the highest NCC score from associated NCC curve. Reusing the same concrete data from previous section, we can compare its performance shown in the middle column of Table 5.4.

Without preprocessing, the unstable high frequency details prevent NCC template matching from finding the correct match, yet with proper amount of filtering in

[6]V-MaxNCC: Voting + Max NCC to resolve voting failure

[7]V -CNN: Voting + Closest Nearest Neighbor to resolve voting failure

frequency space, NCC becomes much more effective in prediction. Furthermore, in addition to using NCC curve for selection purpose, our experiments show that the position recovered from the maximum response in template matching is often more accuate then the corresponding SIFT prediction. Finally, combined with voting strategy, we can achieve more than 90% small error with 1.32mm over all predictions.

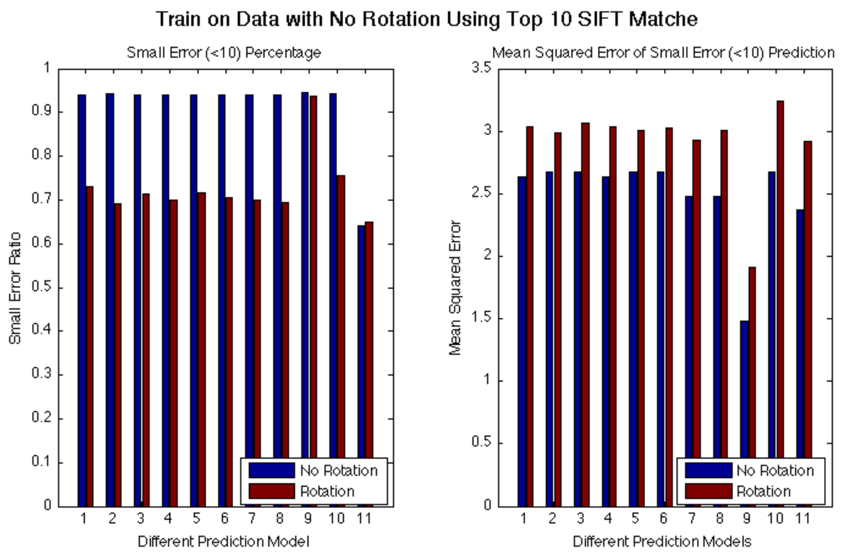
Error Learning

We also attempt to learn a potential mapping from the shape of the NCC curve mentioned in last section to the error range the patch generates in prediction. Thus, we conduct a series of experiments using different kinds of input data, i.e. raw NCC curve, simple heuristics[8], derivatives and cum-sum features (Section 4.4.4). We compare the performance between using logistic regression and neural network as the classifier. The image data we used in our analysis is Carpet-FR data (Figure A.8). In this case, the training image is captured under natural lighting and testing image is taken under flash light. The rotation is about -20° with a scale change of $5\% - 10\%$. Moreover, the carpet has no obvious pattern. In addition, we captured a similar pair of carpet data without rotation and scale (Figure A.7). We use one pair for learning and the other for validation to prevent overfitting. We randomly sampled 15000 NCC curves from 1500 testing patches (i.e. top 10 SIFT predictions) from each image pair. The size ratio between training set and testing set is 1:2 (we experimented with 2:1 ratio, but the performance did not improve significantly). We test our model on both testing sets from the same image pair and validation pair (Figure 5.9[9]).

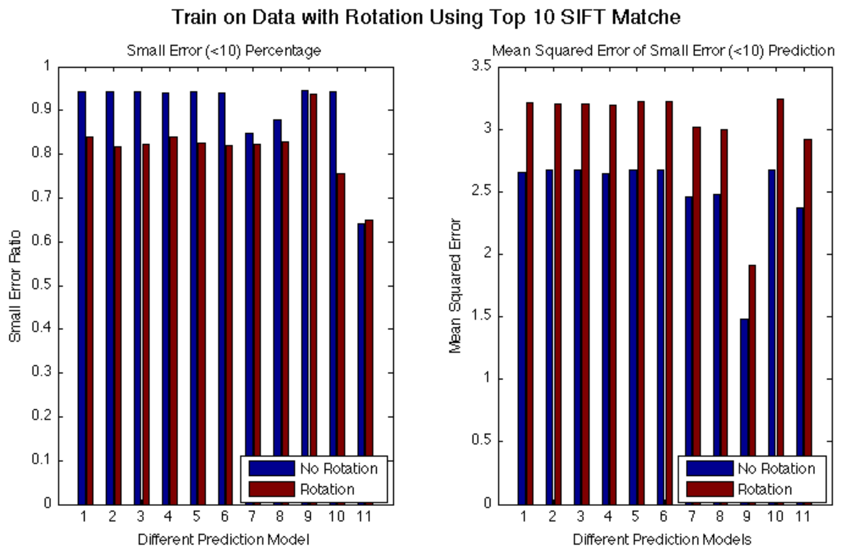
The result shows that the classifiers tend to overfit easier training data with no rotation, so that they do not generalize well to rotation data as in Figure 5.9a. On the other hand, when train on rotation data, the classifiers can also generalize to no

[8]Heuristics include the distance from SIFT predicted rotation $\hat{\theta}$ to the local mode of the NCC curve, the ratio between max and mean of the NCC scores and the absolute max value of NCC curve

[9]Figure 5.9a is generated when training on no rotation data and validate on rotation data and Figure 5.9b is derived the other way around



(a) Train on No Rotation



(b) Train on Rotation

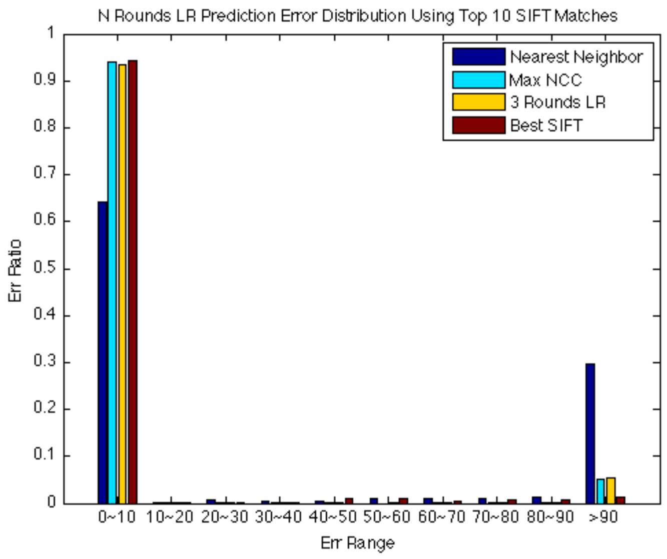
Figure 5.9: Comparison on Different Training Strategies

The corresponding experiments for each pair of bars are: 1. Logistic regression on raw data; 2. Logistic regression on simple heuristics; 3. Logistic regression on derivatives; 4. Neural network on raw data; 5. Neural network on simple heuristics; 6. Neural network on derivatives; 7. Logistic regression on cum-sum features; 8. Neural network on cum-sum features; 9. Best SIFT prediction within top 10 predictions; 10. Maximum NCC predictions; 11. Closest nearest neighbor predictions rotation cases, i.e. Figure 5.9b. This observation also coindices with our intuition: no rotation data is generally more consistent and ideal. As a result, the learned classifier would not be able to tolerate small changes in rotation and scale. Furthermore, it seems like the Cum-Sum feature we designed (bar 7 and 8) tends to cause overfitting. More importantly, with the extra overhead in training and classifying, the overall performance does not improve significantly comparing to greedily selecting the match with absolute maximum NCC score (bar 10). Therefore, one may argue that the extra training process does not worth its efforts.

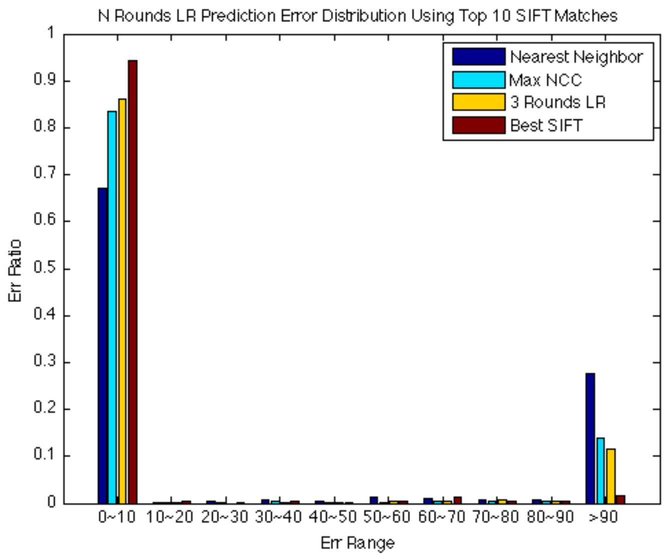
We also experiment with the ensemble method to see if we can make any improvements using more sophisticated learning mothod. As described before in Section 4.4.4, we deliberately hold the confidence level of a classifer so high that any positive prediction from it would most likely to be a true positive case. Therefore, by combining multiple linear classifiers, we can create a complicated decision bounary to partition the space. Figure 5.10[10] shows an experiment with 3 rounds of logistic regression fitting with maximum NCC strategy on leftover data. The model is trained on rotation data and test on both testing sets. Unfortunately, the resulting classifier does not buy us much improvements. Therefore, we decided not to go for this overkilled approach.

To sum up, our learning experiments suggest that the positive examples learned by the classifiers are usually the same as those with highest NCC scores. However, we can still use these classifier in a different way to improve the time efficiency. High confidence level trades recall to precision. Therefore, when a classifier is almost certain on the current example being positive, we can trust its judgement and stop the template matching process with the remaining SIFT predictions in the list. This early-stop trick may not seem to earn us much benefits at the first glance, yet our study shows around 60% of the time, the classifier would be satisfied with the closest

[10]The performance of max NCC strategy improved comparing to those in Figure 5.9 since we search over a smaller neighborhood around SIFT predicted rotation $\hat{\theta}$



(a) Test on No Rotation



(b) Test on Rotation

Figure 5.10: Three Rounds Ensemble Method Using Logistic Regression

nearest neighbor. The more detailed histogram on pruning is given in Figure 5.11. Thus, in practice, this simple strategy does improve the running time.

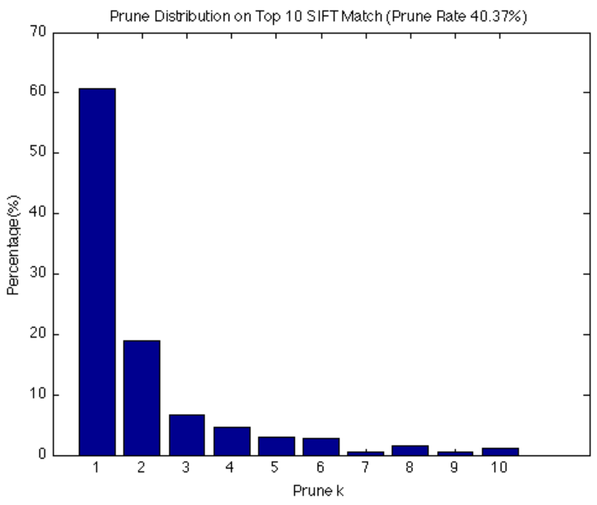


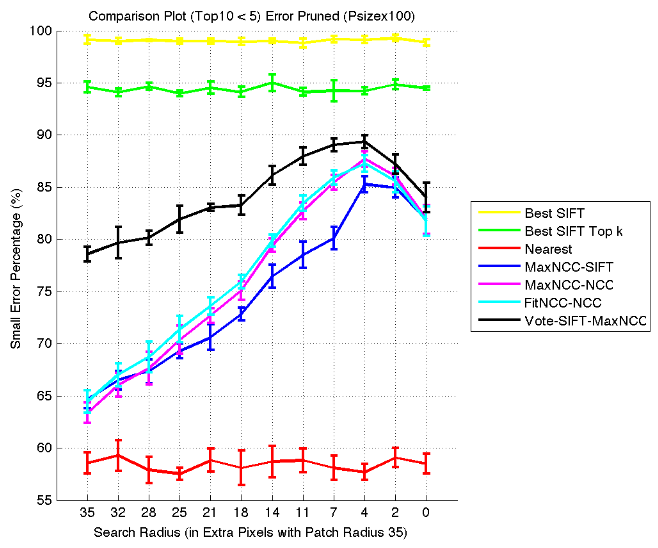
Figure 5.11: Early Stop Pruning Distribution

More Results

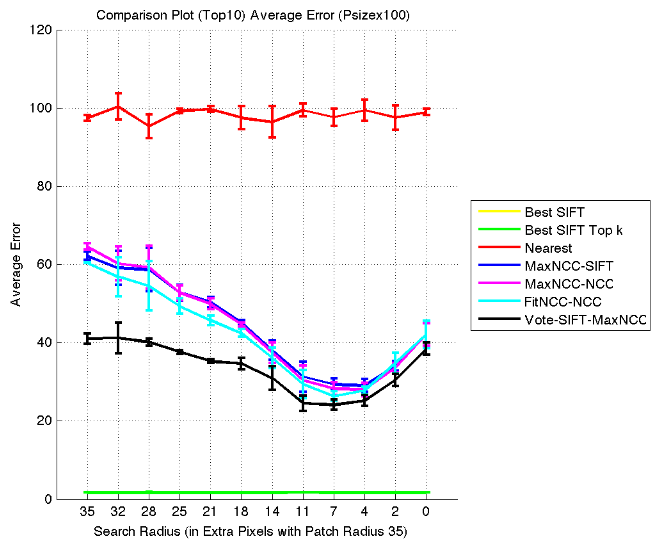
A more comprehensive plot for the performance comparison on Carpet-FR data with near-random texture (Figure A.8) is shown in Figure 5.12.

The result shows that we can achieve near 90% of small error among all predictions (about 6300) using our current best predication strategy, i.e. Vote-SIFT-MaxNCC (black line). The template mathing neighborhood size can be interprated as the estimated error in best SIFT prediction. If we only consider top 10 closest nearest neighbor predictions from SIFT given the ground truth, our final algorithm can provide competitive accuracy with only 5% less than optimum whereas closest nearest neighbors from baseline implementation can merely get close to 60% overall. In addition, the early stop version of max NCC prediction (cyan line) yields roughly the same as selecting the max NCC after examing all 10 predictions (magenta line).

In terms of running time, the summary plot is shown in Figure 5.13. Notice that, early stop reduces the running time by nearly half comparing to examing all predictions. Furthermore, voting appears to be the most time efficient strategy since when it succeeds, only one prediction needs to be further examined.



(a) Train on No Rotation



(b) Train on Rotation

Figure 5.12: Performance Comparison on Different Prediction Strategies

The x-axis denotes the neighborhood size template matching is searching over; the y-axis denotes the percentage of small error, i.e. less than 5 pixels or 11.72mm. Best SIFT (yellow line): the best SIFT prediction among all predictions; Best SIFT Top k (green line): the best SIFT prediction among top 10 closest nearest neighbor predictions; Nearest (red line): the closest nearest neighbor; MaxNCC-SIFT (blue line): using the SIFT prediction selected from max NCC; MaxNCC-NCC (magenta line): using template matching prediction selected from max NCC; FitNCC-NCC (cyan line): MaxNCC-NCC with early stop; Vote-SIFT-MaxNCC (black line): voting combined with max NCC

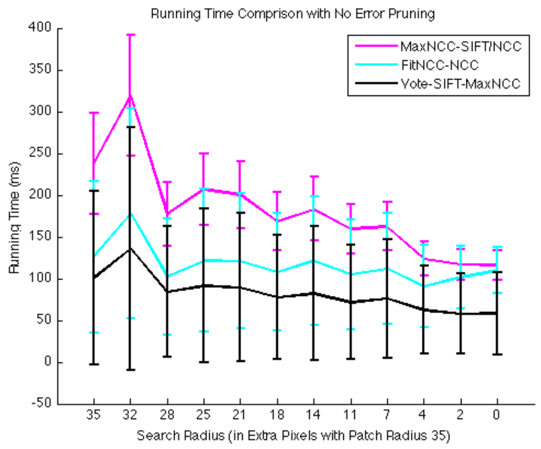


Figure 5.13: Running Time Comparison on Different Prediction Strategies

The same experiment is repeated on other image data, Wood (Figure A.3), Concrete (Figure A.4), Carpet-NL (Figure A.6), Concrete2 (Figure A.5), Carpet-F (Figure A.7) and Carpet-E (Figure A.9). The system performance is summarized in Table 5.5. The experiments are conducted using the whole system pipeline. The statistics reflect the performance of our current best localization strategy.

Data Images	**Average Error (mm)**	**Small Error Percentage**
Wood	0.50 ± 0.04	$99.73 \pm 0.03\%$
Concrete	1.26 ± 0.14	$99.97 \pm 0.05\%$
Carpet-NL	1.83 ± 0.01	$99.98 \pm 0.02\%$
Concrete2	25.36 ± 3.15	$91.61 \pm 0.84\%$
Carpet-F	29.12 ± 2.32	$94.74 \pm 0.37\%$
Carpet-E	332.01 ± 2.61	$51.33 \pm 0.45\%$

Table 5.5: Evaluation on Other Image Data

The results demonstrate certain level of robustness of our system under environmental changes. With similar lighting condition (Wood, Concrete, Carpet-NL), our system can produce high precision localization results. Under more challenging cases with lighting variations (Concrete2, Carpet-F), we can still achieve over 90% of small error predictions. In an extreme situation under insufficient lighting (Carpet-E), the performance drops down to around 51%, yet still better than closest nearest neighbors with 21%. More importantly, the loss in localization accuracy is not a consequence of incorrect algorithm we proposed but the fact that SIFT matches become unreliable. In this case, among top 20 SIFT predictions, the best we can get is merely 51% of small error percentage as well. One possible reason for this degradation in SIFT matching performance is that the influence of insufficient lighting overruns the contributions from colored near-random textures on the surface itself.

Chapter 6

Conclusion

In this study, we presented a novel computer vision based localization system using feature matching in near-random textures. The proposed system is accurate in predictions, robust to environmental changes and marker-free. Our experimental results in this work indicate the potential of applying such algorithm in real world applications. In addition, the multi-level search pipeline we described can be adopted by plugging in other image features for feature matching or other distance metrics for template matching, e.g. bi-directional distance [43]. In this chapter, we will envision some potential applications and discuss some future directions for continuing work.

6.1 Applications

Indoor Robot Navigation

Most of the existing indoor robot localization systems are based on sensors [41]. For instance, people have tried to use ultrasonic sensors to measure the distance between the robot and the reflecting surface. Moreover, Espinace et al. [12] proposed a multi-sensor approach combined with indoor structure information to estimate the current position. In addition, researchers also attempted to measure WIFI signal strength to identify locations [3]. However, sensor based navigation system usually suffer from the noise in readings and reception strength. Even in the idea case, the precision is still not as good as expected. Another solution to this problem is based on applying computer vision techniques in 3D scene registration using markers [25], yet the accuracy is often compromised by occussion and dynamic changes in the scene.

However, our system can more precisely recover the position, usually within 2cm or less comparing to 10-30cm for sensors. It is also easier to match near-random textures on the ground than recognize markers in 3D scene as argued before. In addition, by constantly re-estimating the position, it is also possible to avoid the cumulative error in the long run. Nevertheless, the robustness of our system still requires further evaluations, yet the current results do show potential for this application.

3D Fabrication

The paper from Rivers et al. in 2012 [35] suggests another potential application for our system: the 3D fabrication. In that study, they built a computer vision based control system to cut out interesting objects from wood by following human guildance and markers. Our proposed system can be more flexible by removing the dependency to markers and reducing the amount of human interventions. If our algorithm is robust enough, we can even incorporate it into other reduction based fabrication systems e.g. milling machines to create reliefs.

Dynamic Map

Another way to look at the localization system is to identify whether a robot or vehicle goes to a place it has been to before. For example, in 3D scanning (especially to scann large outdoor objects), it would be helpful to know the exact offset while aligning multiple scans. More interestingly, the map or the knowledge base needs not to be acquired in advance. We can dynamically expand our knowledge base about the environment on the fly, which is a well-known problem called Simultaneous Localization and Mapping or SLAM. An overview of this probalem can be found in [6]. Admitted, the challenges still remain in how to design a reliable yet “marker-free” expansion system to grow the map.

6.2 Future Work

In this section, we will discuss some future directions for continuing work. Furthermore, we will also show our preliminary results for some of them.

Higher Resolution

Although we have higher resolution image data, we haven’t thoroughly examined our system using them. One experiment we did with the high resolution version of the Concrete2 data (Figure A.5) suggests that the localization accuacy may be improved with more accruate acquisitions. The performance summary is shown in Table 6.1.

Data Images	**Avg. Err mm**	**Small Error %**	**Avg. SE mm**[1]
CNN[2]	52.48 ± 1.98	74.74 ± 0.29%	1.03 ± 0.01
MaxNCC	25.59 ± 0.60	89.80 ± 0.23%	0.78 ± 0.01
F-MaxNCC[3]	32.82 ± 1.70	87.03 ± 0.37%	0.77 ± 0.01
V-MaxNCC	23.88 ± 0.42	90.92 ± 0.08%	0.79 ± 0.01

Table 6.1: High Resolution Performance

Comparing to the performance with lower resolution version (Table 5.4), higher resolution actually gains us higher precision in all our error measures. However, we also observe that the early stop strategy with trained classifier (F-MaxNCC) does not generalize as good as before on the same carpet data, yet its performance is still

[1]Avg. SE mm: average among small error in mm
[2]CNN: closest nearest neighbor prediction from SIFT feature matching
[3]F-MaxNCC: MaxNCC with early stop using classifiers

comparable to MaxNCC strategy in this case. More interestingly, we do not have to double the testing patch size in pixel, which means a smaller patch in actual size with higher resolution would be sufficient for localization purpose. Unfortunately, we cannot gain the same performance without doubling the testing patch size on Wall data (Figure A.2). But, when we do double the size, we can achieve a similar performance of over 96% of small error rate with around 0.2mm average error overall.

Reliable Region

Another interesting line of experiments is to identify the statistical properties of the reliable regions for our system. For example, in Figure 6.1, we plot the color-coded map for our system performance. Green regions correspond to those with small prediction error by our algorithm; red regions correspond to those with low SIFT prediction quality; yellow regions correspond to those failure cases of our algorithm, i.e. the final output is worse than the best SIFT prediction(s) we considered in our algorithm. Therefore, we would like to explore if there exist certain patterns either in the actual testing patches or the SIFT descriptors themselves to help us detect failure localization predictions.

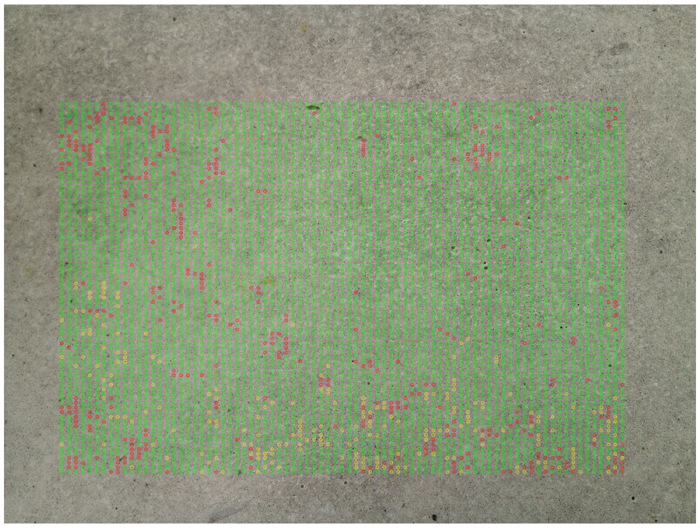


Figure 6.1: Reliable Regions

Temporal Coherence

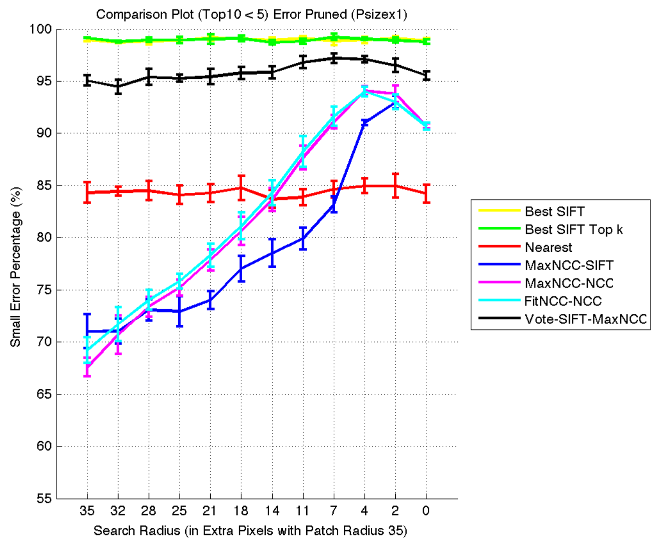
One effective augmentation to improve the performance of our system is to incorporate temporal coherence from dynamic models or sensor readings. By restricting the overall search area, we can bound the localizatoin error from above. In addition, given the nature of near-random textures, a false positive match in feature space could lead to localization predictions in any random position with roughly equal probabilities. Therefore, narrowing down search space could also improve the precision in localizaion. Some preliminary results are shown in Figure 6.2 given some hypothetical temporal coherence on the Carpet-FR data (Figure A.8) usd in Section 5.2.

If we can effectively narrow down our search space to within 164mm from the ground truth, we can achieve over 97% of small error ($error < 11.72mm$) over all predictions (Figure 6.2a). Moreover, given a rough guess of half a meter radius, we can still improve the small error percentage from near 90% to above 93% (Figure 6.2b).

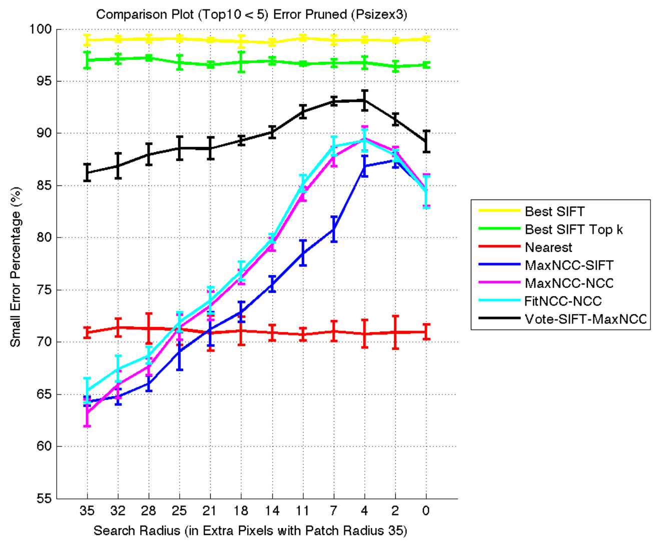
Other Directions

Our current experiments in this work are mostly conducted within small environment. Therefore, the next step would be to solve the scalability issue. However, if we can obtain a rough guess to the scale of the current testing size (listed in Appendix A), our experimental results could provide an estimate on how well our system would perform. In addition, we would also like to experiment with greater variaty of surfaces such as sidework, bricks walls and etc. Furthermore, we would hope to develop a mechanism to detect failure cases of our system such that we can respond accordingly, especially to corrupted data. One possible proposal to handle corrupted data is to use larger patch for SIFT voting yet smaller patch for refinement using tempalte matching.

Finally, if we can deploy our system on a real moving vehicle, the captured image would also be blurred due to motion. Moreover, there would also be physical errors



(a) Search Range of 70 pixel / 164mm



(b) Search Range of 210 pixel / 492mm

Figure 6.2: Performance with Temporal Coherence

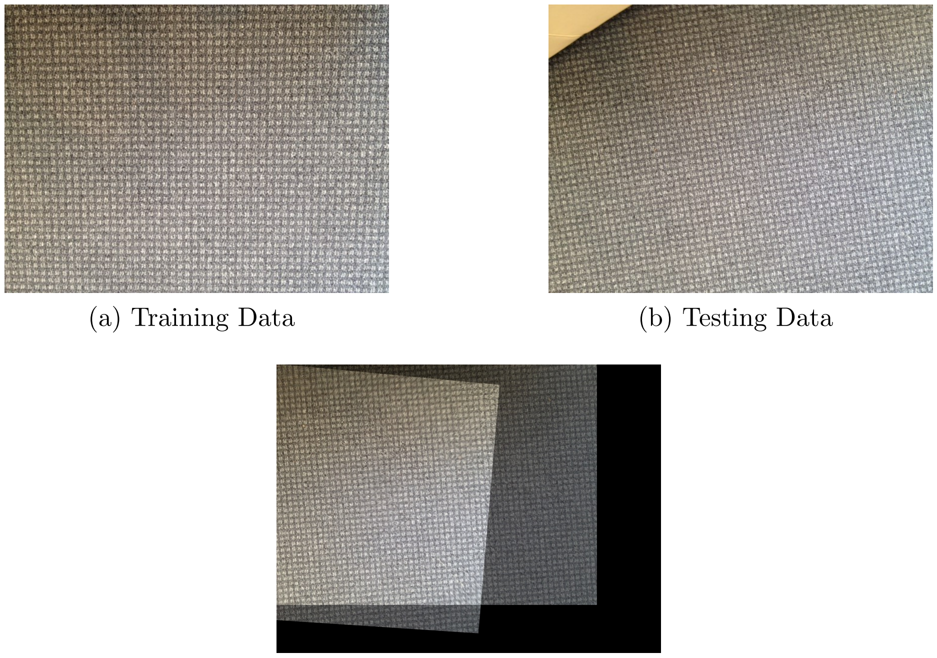
in actual motions from the motors. In addition, dynamically building and updating the knowledge base would also be an interesting topic to further explore.

In conclusion, it seems that our proposed algorithm would still face many challenges both in precision and time efficiency under real world applications, yet our current results do show the potential for continuing research.

Appendix A

Testing Environments

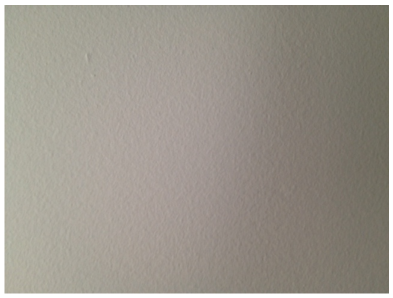
Here is a subset of the testing images we used in this work. All images are captured using a camera phone. The homography is estimated using markers but not included in the image data below. Low Resolution is 640×480 and High Resolution is 1632×1224.

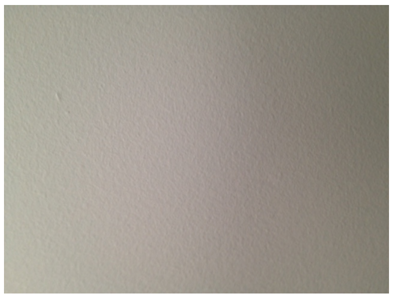

(a) Training Data

(b) Testing Data

(c) Stitch Result

Figure A.1: Patterned Carpet Low Resoltion (0.56m×0.42m)


(a) Training Data


(b) Testing Data

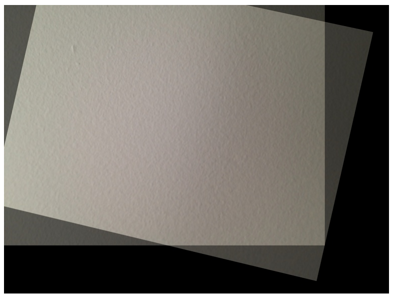
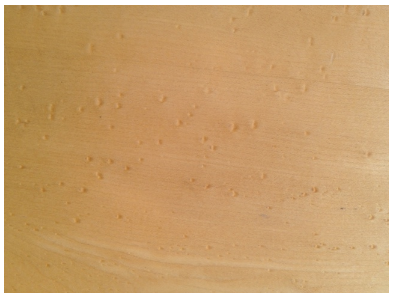
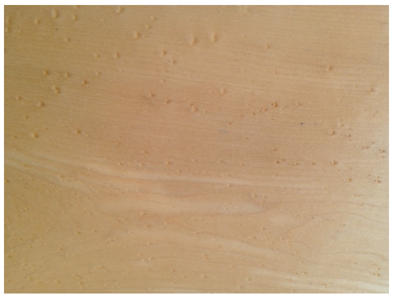

(c) Stitch Result

Figure A.2: Wall Low Resoltion (0.15m×0.13m)


(a) Training Data


(b) Testing Data

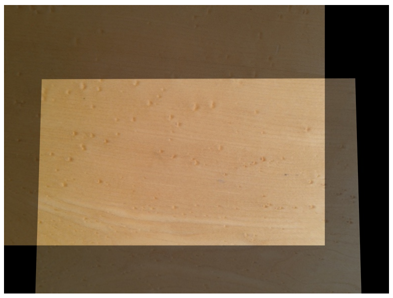

(c) Stitch Result

Figure A.3: Wood Low Resoltion (0.24m×0.18m)

(a) Training Data

(b) Testing Data

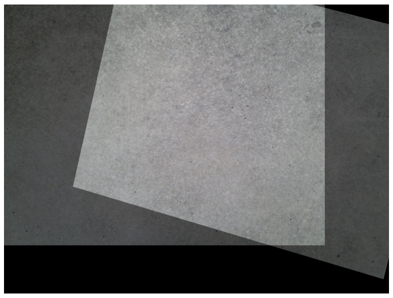

(c) Stitch Result

Figure A.4: Concrete Low Resoltion (0.7m×0.52m)

(a) Training Data

(b) Testing Data

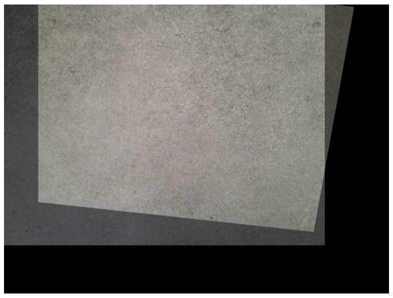
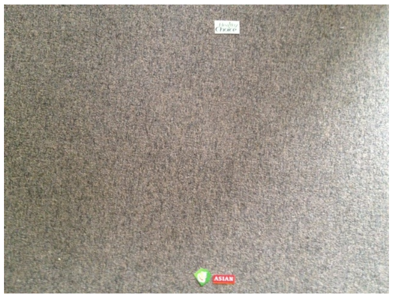

(c) Stitch Result

Figure A.5: Concrete Low Resoltion under Different Natural Lighting (0.7m×0.52m)

(a) Training Data


(b) Testing Data

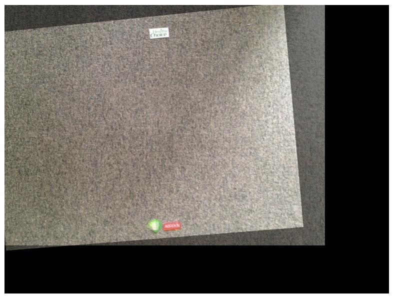
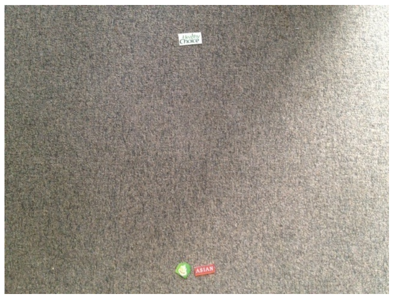
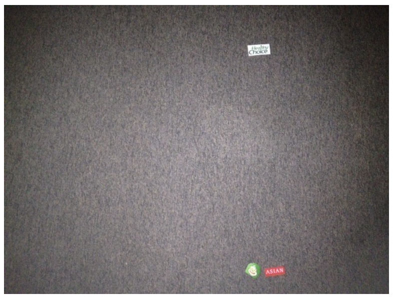

(c) Stitch Result

Figure A.6: Random-Textured Carpet Low Resoltion under Different Natural Lighting (1.2m×0.8m)


(a) Training Data


(b) Testing Data

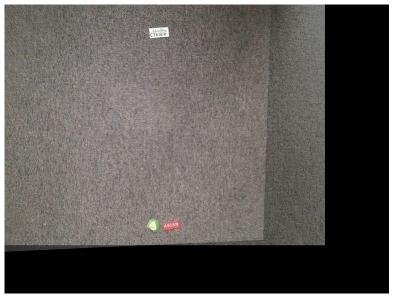
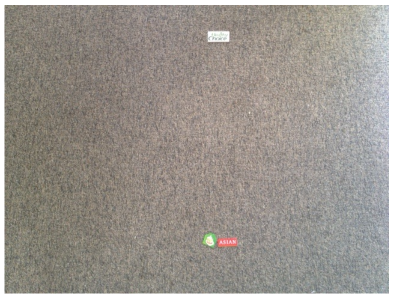

(c) Stitch Result

Figure A.7: Random-Textured Carpet Low Resoltion with Flash Light (1.2m×0.8m)


(a) Training Data

(b) Testing Data

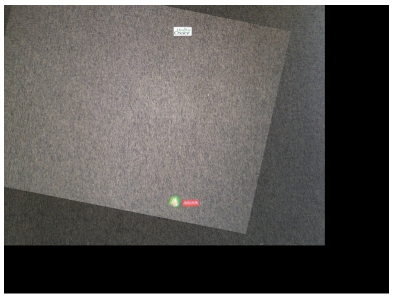
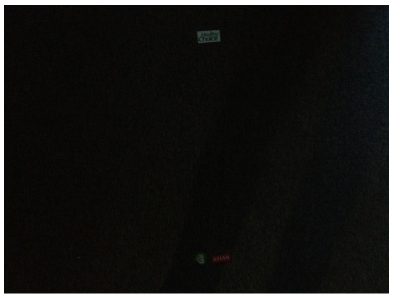

(c) Stitch Result

Figure A.8: Random-Textured Carpet Low Resoltion with Flash Light (1.2m×0.8m)

(a) Training Data


(b) Testing Data

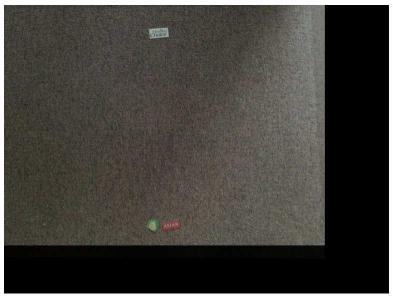

(c) Stitch Result

Figure A.9: Random-Textured Carpet Low Resoltion with No Light (1.2m×0.8m)

Bibliography

[1] i2k Automatic Image Alignment Tool. http://www.dualalign.com/index.php.

[2] Microsoft Image Composite Editor (ICE). http://research.microsoft.com/en-us/projects/ice/.

[3] Netstumbler on Android System. http://www.stumbler.net.

[4] M. Agrawal and K. Konolige. Real-time Localization in Outdoor Environments using Stereo Vision and Inexpensive GPS. In *IEEE International Conference on Pattern Recognition (ICPR 2006)*, volume 3, pages 1063–1068, 2006.

[5] A. Akansu and R. Haddad. *Multiresolution Signal Decomposition: Transforms, Subbands, and Wavelets*. Academic Press, Inc., Orlando, FL, USA, 1992.

[6] J. Aulinas, Y. Petillot, J. Salvi, and X. Lladó. The slam problem: a survey. In *Proceedings of the 2008 conference on Artificial Intelligence Research and Development: Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*, pages 363–371, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[7] C. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[8] G. Carneiro and A. Jepson. Multi-scale phase-based local features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 736–743, 2003.

[9] W. Clarkson, T. Weyrich, A. Finkelstein, N. Heninger, J.A. Halderman, and E.W. Felten. Fingerprinting Blank Paper Using Commodity Scanners. In *IEEE Symposium on Security and Privacy*, pages 301–314, 2009.

[10] J.G. Daugman. Complete Discrete 2-D Gabor Transforms by Neural Networks for Image Analysis and Compression. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36:1169–1179, 1988.

[11] S. Derrode and F. Ghorbel. Robust and efficient fourier-mellin transform approximations for gray-level image reconstruction and complete invariant description. *Comput. Vis. Image Underst.*, 83(1):57–78, July 2001.

[12] P. Espinace, A. Soto, and M. Torres-Torriti. Real-time robot localization in indoor environments using structural information. In *Proceedings of the 2008 IEEE Latin American Robotic Symposium*, LARS '08, pages 79–84, Washington, DC, USA, 2008. IEEE Computer Society.

[13] D.A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach.* Prentice Hall Professional Technical Reference, second edition, 2002.

[14] W.T. Freeman and E.H. Adelson. The Design and Use of Steerable Filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(891–906), 1991.

[15] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3(3):209–226, September 1977.

[16] C. Harris and M. Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[17] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, number 14 in SIGGRAPH '01, pages 327–340. ACM, 2001.

[18] T. Kadir and Brady M. Scale, Saliency and Image Description. *International Journal of Computer Vision*, 45:83–105, 2001.

[19] M. Kaur, J. Kaur, and J. Kaur. Survey of Contrast Enhancement Techniques based on Histogram Equalization. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 2(7), 2011.

[20] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing, 1983.

[21] S. Lazebnik, C. Schmid, and J. Ponce. A sparse texture representation using affine-invariant regions. In *In Proc. CVPR*, pages 319–324, 2003.

[22] J. Lee, Y. Sun, and C. Chen. Multiscale corner detection by using wavelet transform. *IEEE Transaction on Image Processing*, 4:100–104, 1995.

[23] B. Leibe, K. Mikolajczyk, B. Schiele, and T. Darmstadt. Efficient clustering and matching for object class recognition. In *In Proc. BMVC*, 2006.

[24] T. Leung and J. Malik. Representing and Recognizing the Visual Appearance of Materials using Three-dimensional Textons. *International Journal of Computer Vision*, 43:29–44, 2001.

[25] M. Li, B. Hong, and R. Luo. Novel method for monocular vision based mobile robot localization. In *Computational Intelligence and Security, 2006 International Conference on*, volume 2, pages 949–954, 2006.

[26] D. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(91–110), 2004.

[27] J. Matas, O. Chum, U. Martin, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *In Proceedings of the British Machine Vision Conference*, volume 1, pages 384–393, 2002.

[28] R. Matungka, Y.F. Zheng, and R.L. Ewing. Image registration using adaptive polar transform. *Image Processing, IEEE Transactions on*, 18(10):2340–2354, 2009.

[29] K. Mikolajczyk and C. Schmid. An affine invariant interest point detector. In *In Proceedings of the 7th European Conference on Computer Vision*, pages 1–7, 2002.

[30] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1615–1630, 2005.

[31] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *International Journal of Computer Vision*, 65:2005, 2005.

[32] M. Muja and D. Lowe. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. In *In VISAPP International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009.

[33] K. Pearson. *On Lines and Planes of Closest Fit to Systems of Points in Space.* University College, 1901.

[34] B.S. Reddy and B. N. Chatterji. An fft-based technique for translation, rotation, and scale-invariant image registration. *Image Processing, IEEE Transactions on*, 5(8):1266–1271, 1996.

[35] A. Rivers, I. Moyer, and F. Durand. Position-correcting tools for 2d digital fabrication. *ACM Trans. Graph.*, 31(4):88:1–88:7, July 2012.

[36] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, 2 edition, 2003.

[37] F. Schaffalitzky and A. Zisserman. Automated location matching in movies. *Computer Vision and Image Understanding*, 92(2-3):236–264, Nov 2003.

[38] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:530–535, 1997.

[39] S. Se, D. Lowe, and J. Little. Vision-based mobile robot localization and mapping using scale-invariant features. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA 2001)*, volume 2, pages 2051–2058, 2001.

[40] J. Shi and C. Tomasi. Good feartures to track. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[41] F.N. Sibai, H. Trigui, P.C. Zanini, and A.R. Al-Odail. Evaluation of indoor mobile robot localization techniques. In *Computer Systems and Industrial Informatics (ICCSII), 2012 International Conference on*, pages 1–6, 2012.

[42] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, 2008.

[43] D. Simakov, Y. Caspi, E. Shechtman, and M. Irani. Summarizing visual data using bidirectional similarity. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2008.

[44] S. Smith and Brady J. SUSAN - a new approach to low level image processing. *International Journal of Computer Vision*, 23:45–78, 1997.

[45] S. Suzuki and K. Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, pages 32–46, 1985.

[46] T. Tuytelaars, L. Van Gool, L. D'haene, and R. Koch. Matching of affinely invariant regions for visual servoing. In *In Int. Conference Robotics and Automation ICRA 99*, pages 1601–1606, 1999.

[47] T. Tuytelaars and L. Van Gool. Matching widely separated views based on affine invariant regions. *Int. J. Comput. Vision*, 59(1):61–85, August 2004.

[48] M. Varma and A. Zisserman. Classifying Images of Materials: Achieving Viewpoint and Illumination Independence. In *Proceedings of the 7th European Conference on Computer Vision (ECCV)*, pages 255–271, 2002.

[49] J. Weijer and C. Schmid. Coloring local feature extraction. In *Proceedings of the 9th European conference on Computer Vision (ECCV)*, volume Part II, pages 334–348, 2006.

[50] J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid. Local Features and Kernels for Classification of Texture and Object Categories: A Comprehensive Study. *International Journal of Computer Vision*, 73:213–238, 2007.

[51] F. Zhao and W. Huang, Q ad Gao. Image Matching by Normalized Cross-Correlation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 2, 2006.

[52] Z. Zheng, W. Song, and W. Li. Image coding based on flexible contour model. *Machine Graphics Vision*, pages 83–94, 1999.