Universiteit Leiden
The Netherlands

Opleiding Informatica

Dynamic Real-time Videostream Stitching

Using Multiple Uncalibrated Cameras

Mitchel Colli

Supervisors:
Dr. E. M. Bakker & Dr. W. A. Kosters

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

July 1, 2019

Abstract

In this paper the problem of real-time stitching using two or more uncalibrated cameras is addressed. Three different feature detecting algorithms (SIFT, SURF and ORB) using two of our own filters (so-called *Domain Filter* and *Slopes Filter*) are evaluated to determine their ability to find suitable image pairs for stitching in various scenarios. In static high contrast scenarios many features are able to be detected and many matches can be made, whereas in moving low-contrast scenarios this is severely reduced. Without enough matches it is not possible to perform homography estimation. The panoramic video is formed by stitching two or more videostreams together. Our system is designed as a lightweight system where the computational requirements are kept relatively low and low-cost off-the-shelf cameras are used. This in contrast to specialized systems that are either too bulky or too expensive, and that require cameras to be calibrated. In our case we researched the ability to obtain high-quality results without distortions, in real-time, but without calibration. This goal enables us to perform stitching on-the-fly without having to spend much time beforehand.
Performance analysis shows that our system is eligible for real-time stitching by doing so only when a suitable image pair is found, and does not spend any resources on cases where it is not possible.

Contents

Chapter 1

Introduction

One camera lens can capture only one image at a specific point in time. The size of this image is limited to the angle of view of that camera. If we want to create a larger image, we could take different images at different points in time and put them together to create a so called panorama. Creating such a panorama allows for photos with a wider angle than is typically allowed by your lenses. This broad perspective is especially useful for photographers who like larger and more detailed prints, like architectural and landscape photographers. However, taking shots at different moments with moving objects in the scene would create many distortions and we also wouldn't be able to create a video from the area. To create a video whose dimension is larger than the one image taken from a single camera, multiple cameras / lenses are required.

After the invention of photography in 1839, the phenomenon of *"being there"* [1] started to develop. In order to create this *"realistic immersive feeling"* like you're actually there viewing the scene, people thought of putting multiple photos of the environment together: creating a panoramic image. Soon enough, in the late nineteenth century, cameras were manufactured to produce panoramas themselves and more techniques were established [2]. But this progress led to several problems to be solved. One of the early problems was that of visible seams caused by different exposures[1], as can be seen in Figure 1.1.

Figure 1.1: View from the top of Lookout Mountain, Tenn., February, 1864 George Barnard albumen silver print; 10.5 x 42 in. PH - Barnard, G., no. 86 (F size) P&P. Note the visible seams as a result of different exposures. Image taken from [63].

[1] These and other related problems will be discussed more thoroughly in the next chapter.

Nowadays a lot of research has been put into creating panoramas, which resulted in stitching software like AutoStitch [42], Microsoft Photosynth[2] [58] and PTGui [43]. However, these programs are not suitable for use during live recordings. Also various devices have made it to the market, e.g., the Samsung Gear 360 [44], GoPro Odyssey [45] and Nokia OZO [46]. These are often expensive specialized calibrated cameras making them unsuitable for the masses (approx. $15,000 and $60,000 for the Odyssey and OZO, respectively). Furthermore, the built-in stitching software often does not perform as expected (Gear 360 [32]). The Insta360 Pro 2 [47] supports 360 live-streaming at 4K resolution — in both 3D and monoscopic formats, but costs $4999.95.
Research on this topic is quite important, as it finds applications in a wide variety of areas, such as: robotics, texture synthesis, industrial inspection, street view services, surveillance systems, the military, virtual reality and many more. Imagine a group of soldiers wanting to diffuse a bomb. To keep the men safe, they send in a robot equipped with a camera to do the job for them. The Daksh is such a device with multiple cameras, as seen in Figure 1.2. To reduce the risk of making mistakes, the captured images must combine into one with as little delay as possible and without any visible errors.

Figure 1.2: Daksh is an electrically powered Remotely Operated Vehicle (ROV) designed and developed by the Indian state-owned Defence Research and Development Organisation (DRDO) at the Research and Development Establishment (Engineers), Pune, India [49].

These previously described examples, however, use calibrated cameras. An application of stitching images from uncalibrated cameras is when you have persons and vehicles equipped with wearable cameras in one environment. For a central operator concatenated images of the scene would be very useful. But also for landscape and architectural photographers who want to create panoramas using one or more cameras. Another example would be robotic platforms with multiple slightly moving cameras.

In this paper we address if an obtained image pair is eligible for stitching by comparing the effects of three different state-of-the-art feature detecting algorithms without the use of any form of calibration, i.e., you have only a restricted control over its position, orientation, field-of-view, resolution and exposure. In addition, the framework is suitable for anyone who owns a computer/laptop of 2019's standards.

[2]However, this website and service was shut down on February 6, 2017.

The rest of the paper is organized as follows. Chapter 2 gives background information about video stitching. In Chapter 3 we review related state-of-the-art work. Then in Chapter 4 we discuss several algorithms and methods that are applied for real-time video stitching. In Chapter 5 we show how we tested our approach. In Chapter 6 acquired results are displayed. Lastly we draw our conclusions and explore future work in Chapter 7.

This paper was written as a Bachelor project at the Leiden Institute of Advanced Computer Science (LIACS) of Leiden University under the supervision of dr. E. M. Bakker and dr. W. A. Kosters.

Chapter 2

Background: the image stitching pipeline

During the design of previous image stitching systems, different approaches have been taken. Techniques for stitching are broadly classified into two approaches: direct (pixel-based) and feature-based. Various researches [20, 21] have shown advantages and disadvantages of these approaches. However, direct techniques require information about the camera motion model. In this paper we deal with uncalibrated cameras (i.e., you have limited knowledge of the cameras' internal characteristics and restricted control over their positioning in the real world), therefore we do not have such information. For feature-based techniques, on the other hand, calibration is not mandatory, this makes them suitable for our problem and will be the focus of our research.

Feature-based image stitching generally follows the various steps of an image stitching pipeline, as depicted in Figure 2.1.

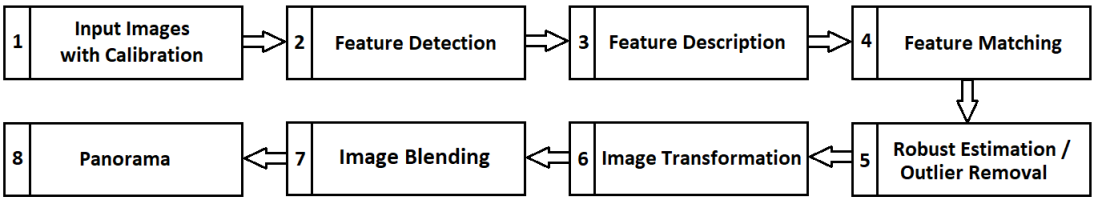


Figure 2.1: An image stitching pipeline.

Various researches [3, 4] have been using this processing pipeline for calibrated image stitching. Calibration is used to reduce the dissimilarities between what is seen in the real world and what is captured by the two or more cameras. Each lens slightly distorts incoming light depending on their purpose, causing straight lines in the real world to appear curved. For example, wide-angle lenses typically produce images with barrel distortion (Figure 2.2) because they have a wide field of view. It is possible to calibrate each camera by calculating how the image should actually look like, and always use that information to rectify the image from that camera before further processing. Another part of calibration is to characterize the position and

the direction between cameras and use this information to reconstruct the three-dimensional structure of a pixel coordinates scene of its image points. In several studies [39, 40] this data has been used to improve homography estimation, which relates the transformation between two planes. In our case we can not perform such calculations, because we are not performing any lens calibration.

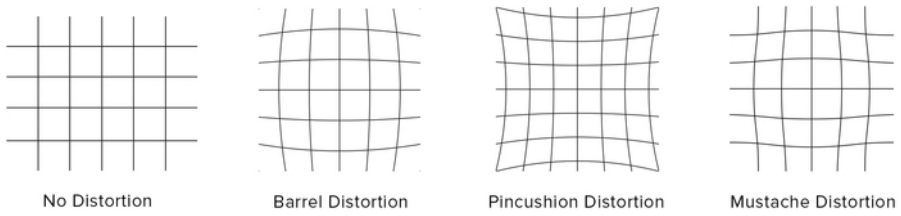


Figure 2.2: Most common lens distortions [50].

In the following, we describe the problems that have to be solved at each stage of the processing pipeline in more detail. Once the system has started, the image registration technique is employed. It takes images captured from the cameras and finds points in the images that have distinct properties (e.g. edges, textures, colors, etc.) and tries to match the same points in each of those images. Afterwards, those matches are used to extrapolate and calculate the relationship between all the points in the images. This enables us to stitch the images together. This technique of collecting matches usually consists of the first five steps of the processing pipeline: 1) Input Images with Calibration, 2) Feature Detection, 3) Feature Description, 4) Feature Matching and 5) Robust Estimation / Outlier Removal. The calculation of the relationship between the planes is done in step 6) Image Transformation. Lastly, the images become one in step 7) Image Blending, resulting in our final panoramic image.

1) Input Images with Calibration

Simply capture images using the selected cameras.

2) Feature Detection (Interest Point Detection / Corner Detection)

An algorithm that chooses points from the captured images based on some criterion. The word 'interest' says it all, a method like this finds the most interesting, most informative points, such as corners, edges, textures, etc. A few examples are: Harris [5], DoG [60], DoH [62], FAST [6] and Min. Eigen [7].

3) Feature Description

"A descriptor is a vector of values that describes the image patch around found interest points. They could be as simple as the raw pixel values, or it could be more complicated, such as a histogram of gradient orientations. A few examples are: SIFT [8], SURF [9], BRISK [12], BRIEF [10], FERNS [11], ORB [13], FREAK [14] and LIOP [15]. Many of these examples also come with their own feature detector. SIFT's detector is based on the Difference-of-Gaussian (DoG), which is an approximation of the Laplacian [61]. The DoG detector detects

centers of blob-like structures. SURF is meant to be a fast approximation of SIFT. BRISK's detector is a corner detector. Its descriptor is a binary string representing the signs of the difference between certain pairs of pixels around the interest point. A challenge of being a good useful descriptor is invariance. The descriptor generally should be scale, shift and rotational invariant.

An interest point together with its descriptor is usually called a local feature. Local features are used for many computer vision tasks, such as image registration, 3D reconstruction, object detection and object recognition" [64]. "In [16] detectors were evaluated by their repeatability ratios and total number of correspondences over several views of scenes and with various image distortion types. In [17] descriptors were evaluated by their matching rates for the same views" [18].

4) Feature Matching

Here the local features are used to establish correspondences between the images. Methods could be broadly divided into two classes:

- "Tree-based methods, like *k*-d tree [24], Best Bin First [25] and Bag-of-Words [26, 27]. These schemes, however, need to set up an indexing tree before feature searching. Because of their additional time costs in building and updating the indexing tree, these methods may not be suitable for real-time applications."
- "Brute-force searching: a straightforward way to find the most similar feature correspondences. Given a certain feature in one image, its search time is computed by the similarity measure time between two features multiplied by the total number of feature candidates in another image" [4].

5) Robust Estimation / Outlier Removal

Once the matches are made, faulty ones still exist. "A simple example is fitting a line in two dimensions to a set of observations. Assuming that this set contains both inliers, i.e., points which approximately can be fitted to a line, and outliers, points which cannot be fitted to this line, a simple least squares method for line fitting will generally produce a line with a bad fit to the data including inliers and outliers. The reason is that it is optimally fitted to all points, including the outliers. RANSAC [28], on the other hand, attempts to exclude the outliers and find a linear model that only uses the inliers in its calculation. This is done by fitting linear models to several random samplings of the data and returning the model that has the best fit to a subset of the data. Since the inliers tend to be more linearly related than a random mixture of inliers and outliers, a random subset that consists entirely of inliers will have the best model fit. In practice, there is no guarantee that a subset of inliers will be randomly sampled, and the probability of the algorithm succeeding depends on the proportion of inliers in the data as well as the choice of several algorithm parameters" [59]. A few other examples are: LMedS [29], MSAC [30] and BaySAC [31]. RANSAC is the most widely used technique to remove them, because of its ability to do robust estimation. Its processing speed is controlled by the sampling time, which is determined by the percentage of outliers inside the input data.

6) Image Transformation

Images undergo the necessary geometric transformations, i.e., rotations, scaling, etc. to align them with each other.

7) Image Blending

Stitching is employed across the stitch, in order to result in seamless stitching. A few examples are: Alpha "feathering" blending, Gaussian Pyramid, Poisson blending and Multiband blending [57]. Stitching techniques are broadly classified into two approaches:

- Direct techniques (Pixel-based): each pixel's intensity in an image is compared with every other pixel in another image. These methods are variant to image scale and rotation.
- Feature-based techniques: each feature point in an image is compared with every other feature point in another image, using their local descriptor. These methods are likely invariant to noisy images, image scale, translation and rotation.

Feature-based video stitching follows the same steps as feature-based image stitching – as video comprises of multiple images shown quickly after each other — but with additional difficulty [34]. Our goal is to achieve an errorless stitched panoramic video in real-time using multiple cameras — in our case two cameras[3]. But using multiple cameras means obtaining multiple frames. These images can differ a lot from each other, which causes a lot of problems during the stitching process. So we will first have to determine which negative effects occur and secondly we will create methods to eliminate them.

The following are important topics which are needed to be solved when stitching as they are artifacts that are visually unacceptable:

- *Distortions:* straight lines in a scene that don't remain straight in an image. A picture taken from a fisheye-lens nicely shows this phenomenon (Figure 2.3).

Figure 2.3: Barrel distortion. Image taken from Android Authority.

[3]A single camera with multiple lenses which produce different images is considered the same thing.

- *Illumination:* light is a very important factor in creating images. If one camera is more exposed to the source than the other, a visible glare can really disrupt the outcome of a panorama (Figure 2.4).

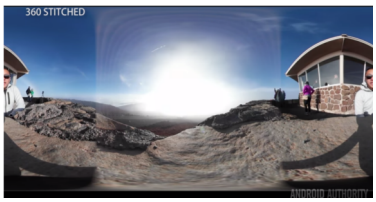


Figure 2.4: Light exposure. Image taken from Android Authority.

- *Parallax:* a displacement in the apparent position of an object viewed along two different lines of sight, and is measured by the angle of inclination between those two lines (Figure 2.5).

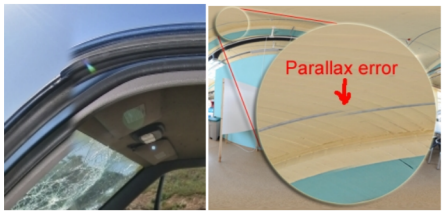


Figure 2.5: Parallax. Image taken from [51].

- *Ghosting:* a ghost is a replica of the transmitted image, offset in position, that is superimposed on top of the main image (Figure 2.6).

Figure 2.6: Ghosting. Image taken from [52].

- *Visible seams:* images stitched together may still leave visible seams behind that have to be removed (Figure 2.7).

Figure 2.7: Visible seams. Image taken from [53].

- *Vignetting:* a reduction of an image's brightness or saturation toward the periphery compared to the image center (Figure 2.8).

Figure 2.8: Vignetting. Image taken from [54].

- *High-cost computations:* a higher computational cost means it takes more time to complete a task.

These disturbances decrease the overall speed of a system and should be avoided as best as possible.

For our processing pipeline we choose SIFT, SURF and ORB as our feature detectors/descriptors and RANSAC as our outlier remover. Other aspects of the processing pipeline are not within the scope of this paper. Details about these methods are described in Chapter 4.

Chapter 3

Related Work

In the previous chapter we described the used techniques and methods for image stitching. Here we focus on work that is closely related to our research goal of determining how to dynamically stitch real-time videostreams using multiple uncalibrated cameras.

Brown et al. [20] formulated ”stitching as a multi-image matching problem, and uses invariant local features to find matches between all of the images”. Because of this, their method is ”insensitive to the ordering, orientation, scale and illumination of the input images. It is also insensitive to noise images that are not part of a panorama, and can recognise multiple panoramas in an unordered image dataset”. They have ”developed a C++ implementation of the algorithm described in their paper, called Autostitch. A demo of this program can be downloaded from `http://www.autostitch.net`”. Our framework uses a similar setup to this and is described in Chapter 4.

Li et al. [4] presented a high-speed aerial video stitching system. Their method – using fast feature detection and binary feature description, a spatial and temporal coherent filter and dynamic key frame updating – was able to obtain results at 150 frames per second (fps) on a what the authors call ”normal personal computer” (2014), rather than an ”advanced vision platform such as FPGA”. This was implemented using C++ and the OpenCV library with image resolutions of 320×240.

Rydholm [36] implemented his vehicle mounted system for creating panoramic video using cylindrical projection – which reduces the problem to finding a translation between the images – and stitching using feature detection and matching with the ORB algorithm. To reduce artifacts such as ghosting, a simple but effective alpha blending technique was used. With three cameras in colour with resolutions of 1024×768, he was able to achieve a panoramic video at approximately 15 fps. This was accomplished using C++ and the OpenCV library on a computer of what the author calls ”2012’s standards”.

He et al. [37] presented ”an efficient parallax-robust stitching algorithm to build wide field of view (FOV) videos for surveillance applications. This method consists of two parts: alignment work done at the stitching model calculation stage and change detection based frame updating at the video stitching stage. The system uses layered warping method to pre-align the background scene which is robust to scene parallax. As for each new frame, an efficient change detection based seam updating method is adopted to avert ghosting and artifacts caused by moving foregrounds”. They obtained results varying at around 12, 22 and 31 fps for resolutions of 1280×720, 720×480 and 352×288, respectively, using C++. All of their experiments were conducted on a PC with Intel® Core™ i3 CPU @ 2.33GHz × 2 and 2GB RAM.

Su et al. [38] proposed ”a robust video stitching approach that can explicitly detect the potential occlusions. Then, based on the occlusion maps”, they chose ”a proper strip in the overlapped region as the blending area. With spatial-temporal Bayesian view synthesis, spatial ghost-like artifacts are significantly eliminated and the output videos kept stable” at around 4 fps with a resolution of 4800×1080. All of their experiments were performed on a workstation with Visual Studio 2013, OpenCV 3.0 and CUDA6.5, Intel® Xeon® CPU @ 3.0GHz. 64GB RAM, Quadro5000 and GPU-Z is 3GB.

Yoon et al. [35] proposed a ”real-time video stitching method using camera path estimation and homography refinement”. Their method can ”stably stitch multiple frames acquired from moving two cameras with a resolution of 1280×720 in real-time at a rate exceeding 13 fps using C++”. Their methods were tested on a PC with an Intel® i5-4590 CPU @ 3.4GHz and 4GB RAM. To reduce processing time, they proposed ”the use of camera path estimation, which is an estimation method using a grid and Lucas-Kanade optical flow” instead of using existing methods like feature extraction, feature matching and homography estimation. To remove the error of alignment, they ”propose the homography refinement, which is a refinement method that uses block matching”.

Ha et al. [41] proposed a condition of suitable image pairs for feature-based image stitching. They introduced a new criterion which is based on the distribution relationship of corresponding features. Applying several sets of sequenced images, they confirmed the usability of their proposed condition. Through the conditions, they expect that the failure of feature-based image stitching will be reduced by controlling strange images that do not satisfy the conditions in advance. We will use this condition to test our framework’s ability to find suitable images pairs. Details about this method is described in Chapter 5.

All previous described methods for stitching require calibrated cameras to perform their calculations, hence we can not adopt these methods. All other techniques for the reduction of artifacts and alignment errors are not within the scope of this paper.

Chapter 4

Videostream Stitching Framework for Multiple Uncalibrated Cameras

In this chapter we describe our framework for videostream stitching using a similar setup to [20] as described in Chapter 2. The videostream pipeline can be seen in Figure 4.1.

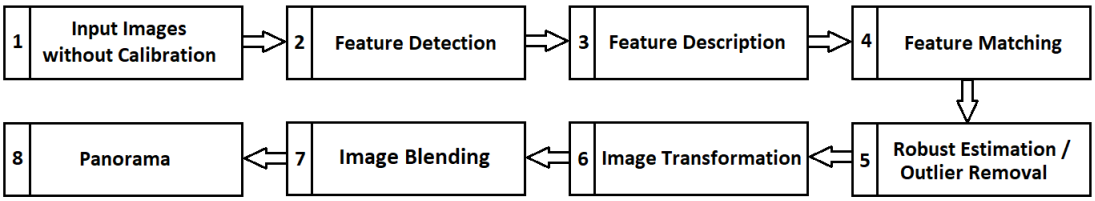


Figure 4.1: A videostream pipeline.

Our main focus is on the first five stages of this pipeline. This is where the matching and filtering is happening. Afterwards, calculating image transformations, etcetera, only has to happen when an image pair is suitable for stitching.

Implementation

Our framework is implemented using C++11 and OpenCV version 3.3.1 (Open Source Computer Vision Library), which "is an open source library for image and video analysis" [22]. "OpenCV is aimed at making computer vision accessible to programmers and users in the area of real-time human-computer interaction and mobile robotics. Thus, the library comes with source code and hand-tuned assembly language binaries optimized for Intel processors, so that users can both learn from the library and make use of its performance" [23].

At each stage of the pipeline one or more algorithms are executed. First we have to select which cameras that are attached to the system we want to use. Then the images are captured and converted into grayscale, because the feature detectors are designed to obtain the intensity of the pixels. Next, interest points are captured, described and matched. Wrong matches are then filtered out of the collection. Using the leftover good matches we can calculate a homography transformation and finally put the images together. So in short, our framework consists of the following algorithms executed in sequence:

- Camera Selector
- Input Images without Calibration
- Feature Detection
- Feature Description
- Feature Matching
- Domain Filter *(optional)*
- Slopes Filter *(optional)*
- Image Transformation *(visualization only)*
- Image Blending *(visualization only)*

Now we will explain what is done at each stage of our pipeline from Figure 4.1 and which algorithms belong to it.

1) Input Images without Calibration

In this module we count and select cameras in order for them to capture images. Because C++ / OpenCV does not have a function to determine the amount of attached cameras, we had to write our own method. Since this method tries to "open" a camera at index i starting from 0 and increments it after each successful attempt, it eventually hits an index that can't be used and will produce an inevitable error as shown in Algorithm 1. The user is then able to select which two cameras to use. Then images can be captured. Pseudo-code for this is shown in Algorithm 2.

Algorithm 1: Camera Selector

Input: -
Output: Indexes used to represent the left and right camera.

```
/* Part I: count if enough cameras are attached to the system.                    */
attachedCount = 0;
while true do
    if a camera doesn't exist at index attachedCount then
        Break;
    end
    if the current camera can't be opened then
        Exit program;
    end
    attachedCount++;
end
/* Part II: ask user which cameras to use.                                        */
if attachedCount < 2 then
    Exit program;
end
while true do
    Ask which camera to use for leftIndex;              /* Index ∈ [0, attachedCount] */
    Ask which camera to use for rightIndex;        /* Can't be the same number as leftIndex */
end
```

Algorithm 2: Input Images without Calibration

Input: leftCamera and rightCamera
Output: Captured images and their grayscale counterpart

```
Capture leftFrame;
Capture rightFrame;
if leftFrame.empty() || rightFrame.empty() then
    Print an error;
    Exit program;
end
Show captured images;
Convert images to grayscale;
```

2) Feature Detection

In this module the features are detected by either SIFT, SURF or ORB. The user can choose which method to use by setting a variable (*mode*) and is used to easily test the different methods during our experiments. Because of the uncalibrated cameras we want to be able to find interest points that are invariant to scale and rotation. In addition, we want to fulfill the real-time requirement. From the vast selection of feature detectors/descriptors we choose SIFT, SURF and ORB, because they all meet the requirements and for their accessibility within OpenCV. Pseudo-code for this module is shown in Algorithm 3.

Algorithm 3: Feature Detection

```
Input: Input frames (grayscale), mode
Output: Feature points for the frames.
switch mode do
    case 0 do
        Detect feature points with SIFT;
    end
    case 1 do
        Detect feature points with SURF;
    end
    case 2 do
        Detect feature points with ORB;
    end
    otherwise do
        Error, wrong mode used;
        Exit program;
    end
end
```

3) Feature Description

In this module the features are described based on the same variable *mode* from the previous algorithm. Pseudo-code for this is shown in Algorithm 4.

Algorithm 4: Feature Description

```
Input: Input frames (grayscale), detected features, mode
Output: Descriptors for the features.
switch mode do
    case 0 do
        Describe feature points with SIFT;
    end
    case 1 do
        Describe feature points with SURF;
    end
    case 2 do
        Describe feature points with ORB;
    end
    otherwise do
        Error, wrong mode used;
        Exit program;
    end
end
```

4) Feature Matching

In this module the features are matched based on the variable *mode*. For SIFT and SURF we had a choice between the L^1-norm, i.e., a Manhattan distance function, or the L^2-norm, i.e., a Euclidean distance. We select the L^1-norm here, because of the real-time requirement. ORB uses binary descriptors, so their distances are represented using the Hamming distance. Pseudo-code for this is shown in Algorithm 5.

Algorithm 5: Feature Matching

```
Input: Detected features, mode
Output: Vector of matches
switch mode do
    case 0 do
        Match descriptors using Brute-Force NORM_L1 matcher;
    end
    case 1 do
        Match descriptors using Brute-Force NORM_L1 matcher;
    end
    case 2 do
        Match descriptors using Brute-Force NORM_HAMMING matcher;
    end
    otherwise do
        Error, wrong mode used;
        Exit program;
    end
end
```

5) Robust Estimation / Outlier Removal

Once matches have been made, false ones still exist. These are the matches that contain interest points that are not the same in the real world, but still get matched because of the similar descriptors. In this module we have to eliminate these wrong matches to improve homography estimation or else we will obtain an incorrect result. To do this, we introduce two different methods. The goal is to experimentally compare the suitability of the image pairs (using our methods and the measure of [41], which will be explained in Chapter 5) with the different algorithms: SIFT, SURF and ORB. We choose these algorithms due to their invariance in scale and rotation, fast performance rates and easy access with C++ and OpenCV.

Domain Filter

For this method we look at different intervals in the distance reaches of SIFT, SURF and ORB. Every feature-based algorithm keeps track of how accurate a made match is. This is done by looking at the "distance" between two feature points. For floating-point features a Manhattan or Euclidean measure is maintained, for binary representations the Hamming distance is used. The closer these "distances" are to zero, the more similar the two points look alike. If the maximum distance is reached, the points are the exact opposite of each other. These maximum values are not desirable and practical, so we define new practical maxima based on an empirical experiment. We chose a high contrast environment so we could obtain many matches using SIFT, SURF and ORB and looked at the distances we obtained from them. We obtained an approximate maximum

reach for SIFT to be 3742, for SURF this is 3 and for ORB it is 98. In case higher values are possible in lower contrasted scenarios we round up these numbers to the closest power of two. So we define a domain of [0-4096], [0-4] and [0-128] for SIFT, SURF and ORB, respectively. These domains are then divided in ten equal lengths and the maximum value is then lowered to the next interval where the performance is measured. Pseudo-code for this is shown in Algorithm 6.

Algorithm 6: Domain Filter

```
   Input: Vector of matches, mode
   Output: Vector of matches
1  maxDistanceSIFT = 4096.0;
2  maxDistanceSURF = 4.0;
3  maxDistanceORB = 128;
4  domainFactor; //Some value in [1-10]
5  if mode == 0 then
6  |  maxDistance = (domainFactor / 10) * maxDistanceSIFT;
7  end
8  if mode == 1 then
9  |  maxDistance = (domainFactor / 10) * maxDistanceSURF;
10 end
11 if mode == 2 then
12 |  maxDistance = (domainFactor / 10) * maxDistanceORB;
13 end
14 for every match do
15 |  if currentMatches.distance > maxDistance then
16 |  |  Remove match;
17 |  end
18 end
```

Slopes Filter

When looking at unfiltered matches in an image we see a lot on lines criss-cross throughout the image with a wide variety of lengths. In case two images only have a horizontal displacement, all lines are ought to be parallel with approximately the same lengths. When one image is slightly rotated lines do cross with different lengths. In order to remove lines that do not look similar to the biggest group this method removes them based on their slope and length. We calculate the standard deviation for both values and remove all matches whose slope or length is not within one standard deviation of the mean. This interval also decreases by a tenth and the performance is measured per algorithm. Pseudo-code for this is shown in Algorithm 7.

These methods are compared to a state-of-the-art outlier removal technique with and without the filters active. For this technique we use RANSAC, because it is capable of interpreting/smoothing data containing a significant percentage of gross errors, and is thus ideally suited for applications in automated image analysis where interpretation is based on the data provided by error-prone feature detectors [28] and because of its easy use as its implementation is within OpenCV funcionality.

Algorithm 7: Slopes Filter

Input: List of matched pixels, left input frame.
Output: Vector of matches

```
for every match do
    Obtain coordinates of left pixel;
    Obtain coordinates of right pixel, adding the width of the left input frame to the x-value;
    Calculate difference for x and y-values;
    Calculate the slope and length for this match and save this data;
end
Calculate the means and standard deviations for the slopes and lengths;
double stdScalar; //Some value in [0-1]
for every match do
    if !(slope > mean1 - stdScalar*stddev1 && slope < mean1 + stdScalar*stddev1) || !(length > mean2 -
     stdScalar*stddev2 && length < mean2 + stdScalar*stddev2) then
        Remove match;
    end
end
```

6) Image Transformation

In this module the homography between the filtered matches is calculated. Afterwards, the frames warp to their destination. Pseudo-code for this is shown in Algorithm 8. This module is for visualization only and is not important to optimize further at this stage.

Algorithm 8: Image Transformation

Input: List of matched pixels, both input frames.
Output: Transformed frame.

```
Calculate the homography matrix between the rightPixels and leftPixels;
Warp rightFrame to leftFrame using the homography matrix with a perspective transformation;
```

7) Image Blending

The resulting images from the Image Transformation are used to properly determine the overlapping region. That part of the image is getting a simple adjustment based on the position of that area. The blending method as described in [36] yields the best result as it avoids as much ghosting as possible, while still making the seam between the frames as smooth as possible. Pseudo-code for this is shown in Algorithm 9. This module is for visualization only and is not important to optimize further at this stage.

Algorithm 9: Image Blending

Input: leftFrame and transformed rightFrame
Output: Panorama image

```
Determine overlapping region;
Perform non-linear alpha blending on that region;
Show the created panorama;
```

Chapter 5

Experiments

In this chapter we describe the experimental setups used to validate the performance and limits of our system. To evaluate the performance of the system, a variety of experiments have to be conducted. The goal is to see if the system is able to find suitable image pairs in an easy scene (e.g., lot of contrast so the system can find more feature points) and a difficult scene (e.g., not much contrast so not many feature points will be detected). Hard- and software specifications used for the experimental setup are shown in Table 6.1 in Chapter 6.

We first define performance as a combination of two factors: speed and quality.
We know the frame rate of the capture device, for real-time we want our system to match that frame rate. Pseudo-code for these measurements can be seen in Appendix A in Algorithm 11 and Algorithm 12, respectively.
Quality of the captured image pair is measured by adopting the method described in [41] and works as follows. After the feature points of both input images have been matched, the size of the area where the matches are distributed on the image with the four matches located at the outermost positions is calculated. These are the pixel coordinates corresponding to the minimum and maximum values in the x and y direction at the pixel position of all matches. Then, the ratio that the matches area occupies in the entire image is calculated (Eq. 5.1). After the good matches have been extracted, the area where they are distributed and their ratio is also calculated (Eq. 5.2). Finally, the percent score is calulated by Eq. 5.3.

$$Matches\ ratio\ (\%) = \frac{Matches\ area}{Total\ size\ of\ image} \times 100 \tag{5.1}$$

$$Good\ matches\ ratio\ (\%) = \frac{Good\ matches\ area}{Total\ size\ of\ image} \times 100 \tag{5.2}$$

$$Percent\ score\ (\%) = \frac{Good\ matches\ ratio}{Matches\ ratio} \times 100 \tag{5.3}$$

Figure 5.1 represents the idea behind this quality measure.

Figure 5.1: Interest points in corresponding feature analysis. (a) is matches area, which is the area occupied by the outermost corresponding features. (b) is good matches area, which is the area occupied by the outermost good matches [41].

Pseudo-code for this is shown in Algorithm 10.

Algorithm 10: Percent Score

```
   Input: rightFrame, rightPixels
   Output: Percent Score
   /* Define four out of bounds minimum and maximum coordinates                    */
1  Point2f p1(0, MAX_RESOLUTION);
2  Point2f p2(-1, 0);
3  Point2f p3(0, -1);
4  Point2f p4(MAX_RESOLUTION, 0);
5  for every rightPixel do
6      Point2f currentPoint = rightPixels[i];
7      if currentPoint.y < p1.y then
8          p1 = currentPoint;
9      end
10     if currentPoint.x > p2.x then
11         p2 = currentPoint;
12     end
13     if currentPoint.y > p3.y then
14         p3 = currentPoint;
15     end
16     if currentPoint.x < p4.x then
17         p4 = currentPoint;
18     end
19 end
20 Store x and y values of all points in separate variables with the same number;
   /* Calculate the area according to the four outermost coordinates               */
21 area = abs(( (x1*y2 - y1*x2) + (x2*y3 - y2*x3) + (x3*y4 - y3*x4) + (x4*y1 - y4*x1) ) / 2.0);
22 matchesRatio = (area / size_of(rightFrame)) * 100.0;
23 percentScore = (matchesRatio / initialRatio) * 100.0;
   /* The initial ratio is calculated in the same way after all matches have been made */
```

Figure 5.2 shows where in the pipeline these measurements are taken.

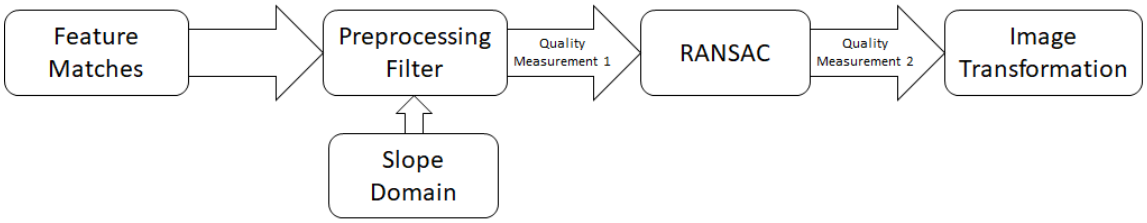


Figure 5.2: Quality measurements.

According to [41], whenever the percent score falls between 5% and 80%, the images are a suitable pair for stitching. If the percent score falls below 5%, the overlapping area is too small to find enough good matches. If the percent score exceeds 80%, not enough bad matches are removed from the initial acquired matches. This in turn affects the calculation of the homography between the images and makes stitching unsuccessful. We shall use these values as well to evaluate our own filters.

Secondly, we define limits as the maximum angle between the cameras' viewing direction where they still obtain percent scores which show the captured image pair is suitable for stitching. If the test constantly returns a negative suitable pair has been found, the limit has been reached or there is no overlap at all between both images.

Experimental execution

The default setup is defined as two cameras with their lenses being 14cm apart and having parallel viewing directions 122cm away from the captured scene, as can be seen in Figure 5.3.

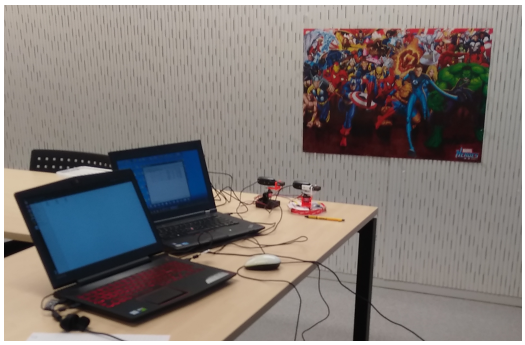


Figure 5.3: The experimental setup. The poster used during our experiments is ©MARVEL[4].

Starting from this position, the percent score is measured by performing a rotation with the rightmost camera until no possible overlap exists divided in 10 steps. This translates to a maximum rotation of 45°. An abstract representation of explaining this setup can be seen in Figure 5.4.

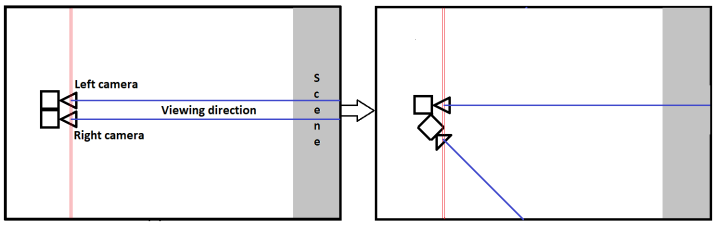
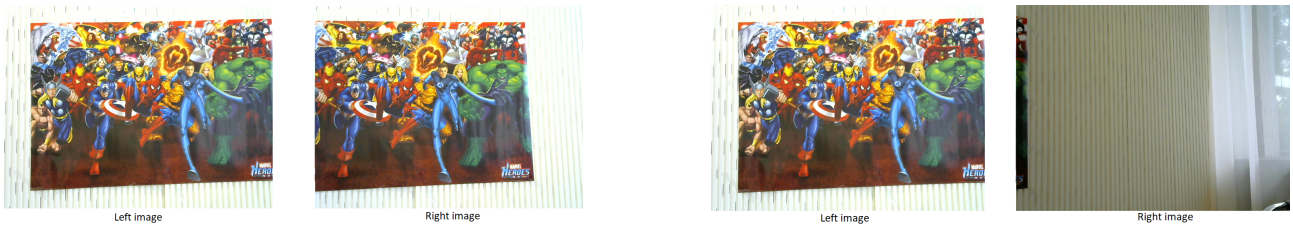


Figure 5.4: Abstract representation of the experimental setup.

The captured images for the starting and ending position can be seen in Figure 5.5. At each of these intervals of 4.5° the percent score is calculated for every algorithm and every filter. The cameras are attached to a servomechanism so we can accurately rotate according to the intervals.



(a) (b)

Figure 5.5: Captured images. (**a**) Starting position; (**b**) Final position.

[4]MARVEL, all related characters: ™ & © 2008 Marvel Entertainment, Inc. and its subsidiaries. Licensed by Marvel Characters B.V. `www.marvel.com` All rights reserved.

Chapter 6

Results

In this chapter we evaluate our achieved results from the experiments described in Chapter 5. Our experiments were conducted using the following hard- and software details, as shown in Table 6.1:

OS	Ubuntu 16.04.2 LTS
Kernel	4.13.0-38-generic
RAM	15,6 GiB
CPU	Intel® Core™ i7-7700HQ CPU @ 2.80GHz × 8
GPU	Intel® HD Graphics 630 (Kaby Lake GT2)
Cmake	3.5.1
Make	4.1
C++	C++11
Compiler	5.4.0 (GNU C++ Compiler)
OpenCV	3.3.1
USB	3.0
Resolution	640×480
FPS	15

Table 6.1: Hard- & software details.

With these specifications and the used cameras the system is able to capture and process 15 frames per second (FPS) without any other calculations. Table 6.2 shows the average speeds of the used algorithms at which our system handles the captured images.

Algorithm	SIFT	SURF	ORB
FPS	4	13	10

Table 6.2: Average speeds of the used algorithms.

This means SURF is the fastest and SIFT the slowest algorithm.

In order to compare our filters with RANSAC, we have to know the performance of only RANSAC. Figure 6.1 shows the performance of RANSAC without our own filters using the different algorithms.

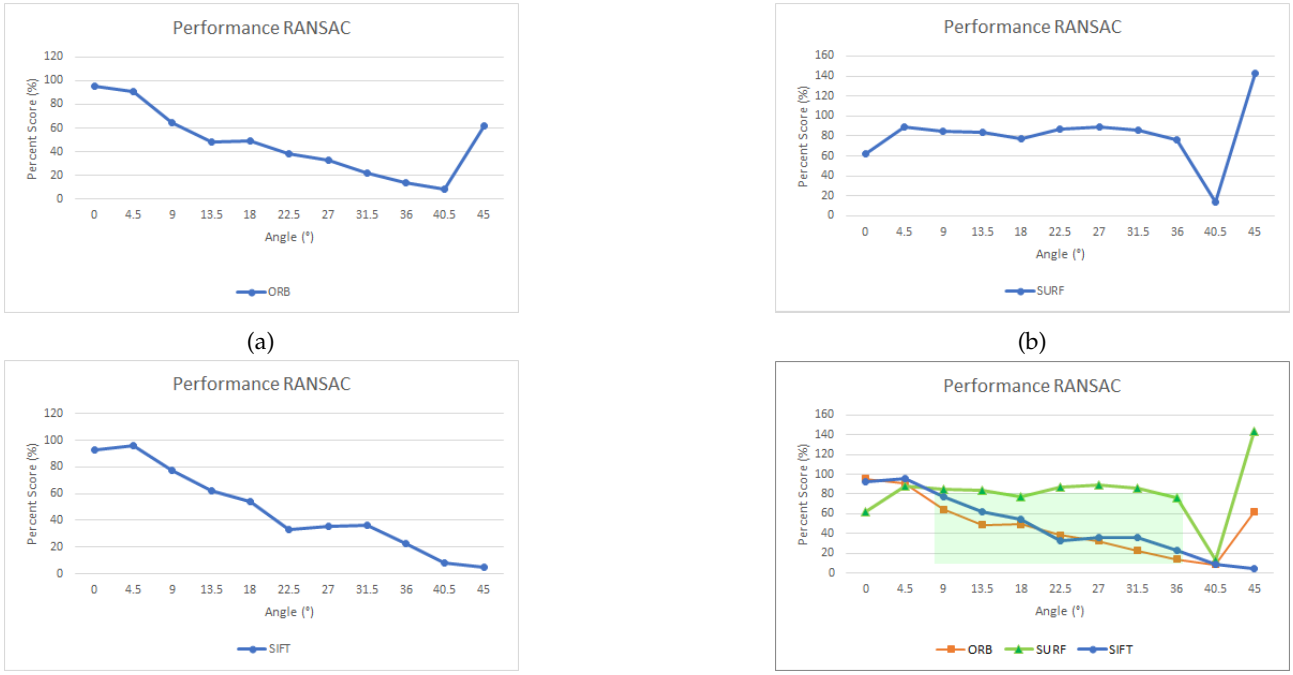
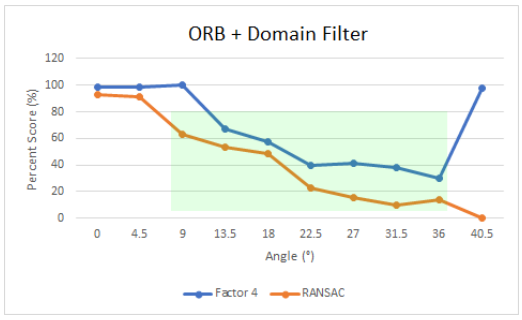


Figure 6.1: Performance of RANSAC without our own filters using the various algorithms. (**a**) Using ORB; (**b**) Using SURF; (**c**) Using SIFT; (**d**) Comparison of all three cases.

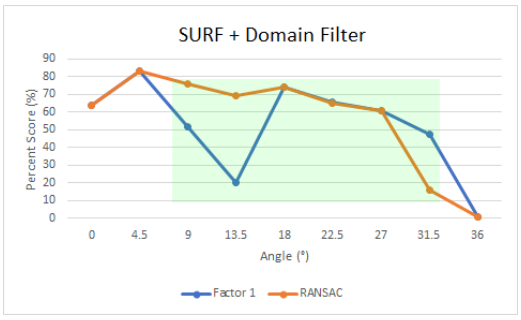
According to these results, RANSAC with SURF rarely ever returns a value that states the given image pair is suitable for stitching. However, Figure 6.2 shows a screenshot of a suitable image pair of said combination at an angle of 22.5°. Also, the 40.5° angle is the final step which maintains a little bit of overlap between the two images. At a 45° angle no overlap exists, but the scores might be positive — this gives unreliable scores.

Figure 6.2: Result using RANSAC with SURF at a 22.5° angle. (**a**) Input images; (**b**) Resulting panorama. The area highlighted in green is the sweet spot.

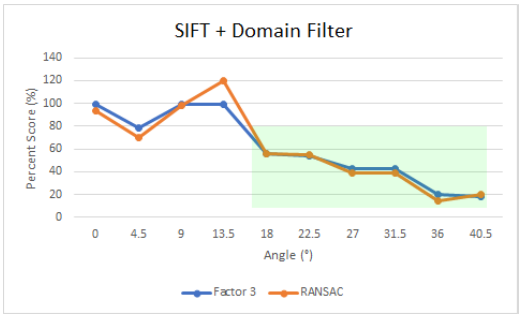
The performances of our own filters using the different algorithms are shown in Appendix B throughout Figures B.1-B.6. Here we will highlight the best case for each of those results. Figure 6.3 and Figure 6.4 show the best cases for the Domain Filter and Slopes Filter, respectively.



(a)

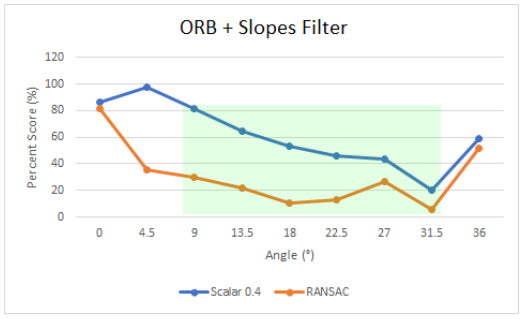


(b)

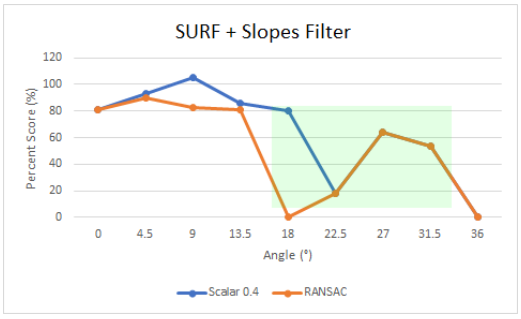


(c)

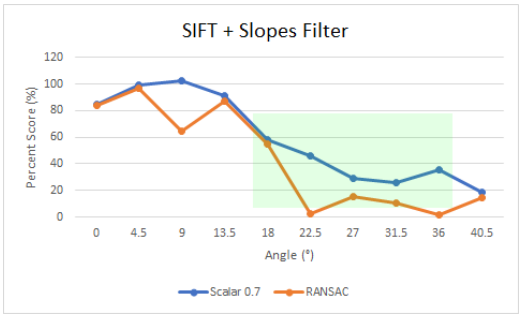
Figure 6.3: Best results of Domain Filter. (**a**) ORB with factor 4; (**b**) SURF with factor 1; (**c**) SIFT with factor 3. The area highlighted in green is the sweet spot.



(a)



(b)



(c)

Figure 6.4: Best results of Slopes Filter. (**a**) ORB with scalar 0.4; (**b**) SURF with scalar 0.4; (**c**) SIFT with scalar 0.7. The area highlighted in green is the sweet spot.

What immediately stands out here, is that no data exists now for a 45° angle (no overlap). This is because so many matches have been filtered out, there are not enough matches left to even perform a homography estimation so there is no need to calculate the percent score. The influence of RANSAC after matches have been preprocessed by our own filters is minimalistic.

Chapter 7

Conclusion and Future Work

The goal of this paper was to determine how to dynamically stitch real-time videostreams using multiple uncalibrated cameras. Three state-of-the-art feature-based algorithms were used to collect interest points and create matches. We introduced two different methods to filter out good matches and compared them to RANSAC, an other widely used filtering technique. Our experimental setup was designed to compare several scenarios. To evaluate the quality of our methods we adopted a performance measurement introduced by Ha et al [41].

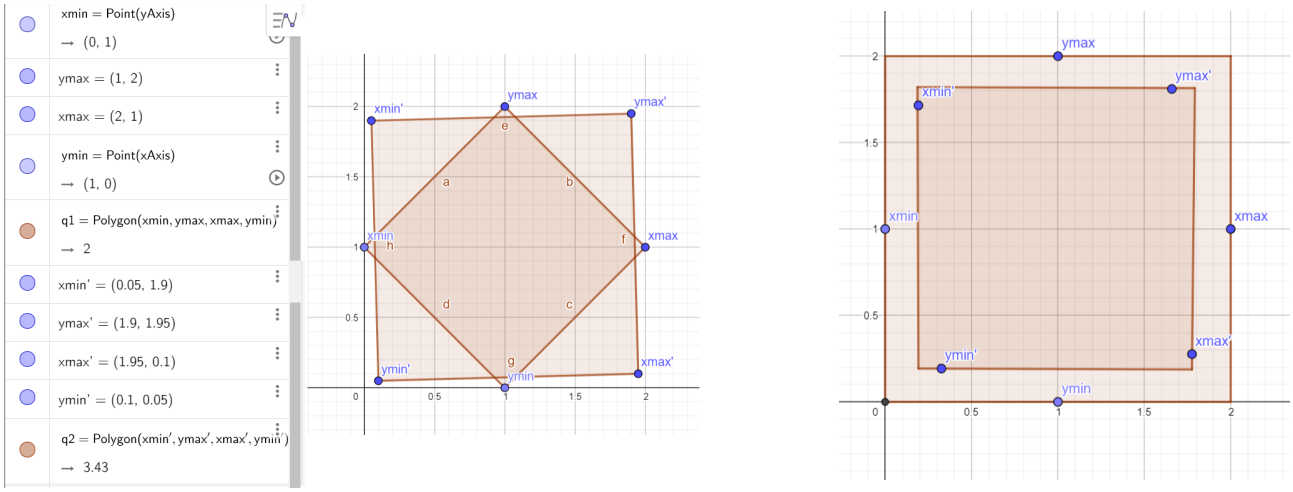
Results quickly show that our methods are able to score on par compared to the use of only RANSAC. However, RANSAC's scores are unreliable once no overlap between two images exist due to the fact that RANSAC handles data without context and will always want to try to keep the greatest common denominator of data. Whereas our methods filter based on thresholds, so it is possible to be left without enough matches to even perform homography estimation. This in turn will tell that two images are not suitable for stitching even without the use of the performance measurements. Applying RANSAC after our own filters is redundant.

Based on our findings the optimal setup would be to use ORB with the Slopes Filter having its scalar set to 0.4.

Some improvements and inconsistencies exist within our results. The first five graphs or so in Figures B.1-B.6 show our filters score a 100%, meaning no points were filtered. So instead of rounding up our experimental maxima to the nearest power of two, we could have started with half of the limits — but this should be tested.

Also notice the fact that the percent score is able to exceed 100%. Once all matches have been made, the outermost points are determined and the area they encompass is calculated. After a filter is applied, this process is repeated. However, it is possible for the outermost points to arrange themselves in such a way the area they then surround could actually be larger than smaller. Our suggestion is to include a bounding box containing these points. This way, when the outermost points change, the bounding box is always smaller or

equal to its previous version. Figure 7.1 shows this behaviour.



(a) (b)

Figure 7.1: Areas of minima/maxima, points obtained after a potential filter are denoted with an apostrophe. (**a**) Proof that the first obtained area is smaller than the next one; (**b**) Suggested bounding box.

To see if our filters are state-of-the-art worthy, we would like to test our approach in a variety of other scenarios. These include setups with less contrast, more camera movements (e.g., horizontal/vertical displacements), more depth in the scenes to test its robustness when parallax occurs and moving objects. For other future work, we would like to see our addition of the bounding box to the performance measurements tested. With further research and improvements — such as applying multithreading or using optical flow — the system might be even faster and more accurate. We believe using more than two cameras would still work, but that should be tested.

Bibliography

[1] J. J. Cummings and J. N. Bailenson, "How Immersive Is Enough? A Meta-Analysis of the Effect of Immersive Technology on User Presence," *Media Psychology*, vol. 19, no. 2, pp. 272–309, 2016.

[2] T. Luhmann, "A Historical Review on Panorama Photogrammetry," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 34, no. 5/W16, p. 8, 2004.

[3] S. Pravenaa and R. Menaka, "A Methodical Review on Image Stitching and Video Stitching Techniques," *International Journal of Applied Engineering Research*, vol. 11, no. 5, pp. 3442–3448, 2016.

[4] J. Li, T. Yang, J. Yu, Z. Lu, P. Lu, X. Jia, and W. Chen, "Fast Aerial Video Stitching," *International Journal of Advanced Robotic Systems*, vol. 11, no. 10, p. 167, 2014.

[5] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," *Proceedings of the 4th Alvey Vision Conference*, pp. 147–151, 1988.

[6] E. Rosten, R. Porter, and T. Drummond, "Faster and Better: A Machine Learning Approach to Corner Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 105–119, 2010.

[7] J. Shi and C. Tornasi, "Good Features to Track," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 593–600, 1994.

[8] D. G. Lowe, "Distinctive Image Features From Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[9] H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded Up Robust Features," *IEEE Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[10] M. Calonder, V. Lepetit, M. Özuysal, T. Trzcinski, C. Strecha, and P. Fua, "Brief: Computing a Local Binary Ddescriptor Very Fast," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1281–1298, 2012.

[11] M. Özuysal, M. Calonder, V. Lepetit, and P. Fua, "Fast Keypoint Recognition using Random Ferns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 3, pp. 448–461, 2010.

[12] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary Robust Invariant Scalable Keypoints," *International Conference on Computer Vision*, pp. 2548–2555, 2011.

[13] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An Efficient Alternative to SIFT or SURF," *International Conference on Computer Vision*, pp. 2564–2571, 2011.

[14] A. Alahi, R. Ortiz, and P. Vandergheynst, "Freak: Fast Retina Keypoint," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 510–517, 2012.

[15] Z. Wang, B. Fan, and F. Wu, "Local Intensity Order Pattern for Feature Description," *International Conference on Computer Vision*, pp. 603–610, 2011.

[16] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool, "A Comparison of Affine Region Detectors," *International Journal of Computer Vision*, vol. 65, no. 1/2, pp. 43–72, 2005.

[17] K. Mikolajczyk and C. Schmid, "A Performance Evaluation of Local Descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, 2005.

[18] A. Hietanen, J. Lankinen, J.-K. Kämäräinen, A. G. Buch, and N. Krüger, "A Comparison of Feature Detectors and Descriptors for Object Class Matching," *Neurocomputing*, vol. 184, pp. 3–12, 2016.

[19] G. S. Maddala, "Outliers," *Introduction to Econometrics (2nd ed.)*, pp. 88–96, 1992.

[20] M. Brown and D. G. Lowe, "Automatic Panoramic Image Stitching using Invariant Features," *International Journal of Computer Vision*, vol. 74, no. 1, pp. 59–73, 2007.

[21] R. Szeliski, "Image Alignment and Stitching: A Tutorial," *Foundations and Trends® in Computer Graphics and Vision*, vol. 2, no. 1, pp. 1–108, 2006.

[22] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, "A Brief Introduction to OpenCV," *Proceedings of the 35th International Convention MIPRO*, pp. 1725–1730, 2012.

[23] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Sotware Tools*, 2000.

[24] J. L. Bentley, "Multidimensional Binary Search Trees Used for Associative Searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[25] J. S. Beis and D. G. Lowe, "Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces," *Conference on Computer Vision and Pattern Recognition*, 1997.

[26] L. Fei-Fei and P. Perona, "A Bayesian Hierarchical Model for Learning Natural Scene Categories," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 524–531, 2005.

[27] T. Botterill, S. Mills, and R. Green, "Speeded-up Bag-of-Words Algorithm for Robot Localisation Through Scene Recognition," *International Conference of Image and Vision Computing New Zealand*, pp. 1–6, 2008.

[28] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications To Image Analysis and Automated Cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–396, 1981.

[29] P. J. Rousseeuw, "Least Median of Squares Regression," *Journal of the American Statistical Association*, vol. 79, no. 388, pp. 871–880, 2012.

[30] P. H. S. Torr and A. Zisserman, "A New Robust Estimator With Application to Estimating Image Geometry," *Journal of Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 138–156, 2000.

[31] T. Botterill, S. Mills, and R. Green, "New Conditional Sampling Strategies for Speeded-Up RANSAC," *Proceedings of the British Machine Vision Conference*, 2009b.

[32] T. Ho and M. Budagavi, "Dual-Fisheye Lens Stitching for 360-degree Imaging," *Proceedings of the 42nd IEEE International Conference on Acoustics, Speech and Signal Processing*, 2017.

[33] W. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, "Bundle Adjustment: A Modern Synthesis," *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*, pp. 298–372, 1999.

[34] T. Tsen, "Video Stitching Literature Review," 2014.

[35] J. Yoon and D. Lee, "Real-Time Video Stitching Using Camera Path Estimation and Homography Refinement," *Symmetry*, vol. 10, p. 4, 2018.

[36] N. Rydholm, "Panoramic Video Stitching," 2015.

[37] B. He and S. Yu, "Parallax-Robust Surveillance Video Stitching," *Sensors*, 2015.

[38] J. Su, H. Cheng, L. Yang, and A. Luo, "Robust Spatial-Ttemporal Bayesian View Synthesis for Video Stitching with Occlusion Handling," *Machine Vision and Applications*, vol. 29, pp. 219–232, 2017.

[39] J.-H. Suk, C.-G. Lyuh, S. Yoon, and T. M. Roh, "Fixed HomographyBased Real-Time SW/HW Image Stitching Engine for Motor Vehicles," *ETRI Journal*, vol. 37, no. 6, pp. 1143–1153, 2015.

[40] C. Zou, P. Wu, and Z. Xu, "Research on Seamless Image Stitching based on Depth Map," *6th International Conference on Pattern Recognition Applications and Methods*, pp. 341–350, 2017.

[41] Y.-J. Ha, S.-H. Park, and H.-D. Kang, "Suitable Image Pairs for Feature-based Image Stitching," *10th International Conference on Human System Interactions (HSI)*, 2017.

[42] "AutoStitch: A New Dimension in Automatic Image Stitching." http://matthewalunbrown.com/autostitch/autostitch.html. Accessed: 2018-05-14.

[43] "PTGui." https://www.ptgui.com/. Accessed: 2018-05-14.

[44] "Samsung Gear 360." https://www.samsung.com/us/support/owners/product/gear-360-2016. Accessed: 2018-05-14.

[45] "GoPro Odyssey." https://gopro.com/odyssey. Accessed: 2018-05-14.

[46] "Nokia OZO." https://ozo.nokia.com/. Accessed: 2018-05-14.

[47] "Insta360 Pro 2." https://www.dpreview.com/news/7672468603/the-insta360-pro-2-is-an-8k-3d-360-degree-camera. Accessed: 2018-08-24.

[48] "Al-Vista." https://www.loc.gov/collections/panoramic-photographs/articles-and-essays/a-brief-history-of-panoramic-photography/. Accessed: 2018-08-23.

[49] "Daksh." https://www.army-technology.com/projects/remotely-operated-vehicle-rov-daksh/. Accessed: 2018-08-23.

[50] "Distorions." http://www.drewgrayphoto.com/learn/distortion101. Accessed: 2018-08-25.

[51] "Parallax." http://3dpanorama.co.uk/faq/how-to-fix-parallax. Accessed: 2018-08-25.

[52] "Ghosting." https://en.wikipedia.org/wiki/Ghosting_(television)). Accessed: 2018-08-25.

[53] "Visible Seams." https://www.photo.net/discuss/threads/help-removing-seams-while-stitching-panorama.89177/). Accessed: 2018-08-25.

[54] "Vignetting." https://commons.wikimedia.org/wiki/File:Example_of_vignetting_and_dusty_scan.jpg. Accessed: 2018-08-25.

[55] "Image Transformation Techniques." http://content.inflibnet.ac.in/data-server/eacharya-documents/548158e2e41301125fd790c3_INFIEP_58/6/ET/58-6-ET-V1-S1__image_transformation_techniques.pdf. Accessed: 2018-05-14.

[56] "Local Reference Frame." https://en.wikipedia.org/wiki/Local_reference_frame. Accessed: 2018-05-14.

[57] "Computer Vision: Algorithms and Applications." http://szeliski.org/Book/. Accessed: 2018-05-14.

[58] "Microsoft Photosynth." http://photosynth.net. Accessed: 2018-05-14.

[59] "Random Sample Consensus." https://en.wikipedia.org/wiki/Random_sample_consensus. Accessed: 2018-05-14.

[60] "Molecular Expressions Microscopy Primer: Digital Image Processing Difference of Gaussians Edge Enhancement Algorithm." http://micro.magnet.fsu.edu/primer/java/digitalimaging/processing/diffgaussians/index.html. Accessed: 2018-05-07.

[61] "LoG Filter." http://academic.mu.edu/phys/matthysd/web226/Lab02.htm. Accessed: 2018-05-07.

[62] "The Hessian." https://www.khanacademy.org/math/multivariable-calculus/applications-of-multivariable-derivatives/quadratic-approximations/a/the-hessian. Accessed: 2018-05-07.

[63] "Lookout Mountain." https://www.loc.gov/collections/panoramic-photographs/articles-and-essays/a-brief-history-of-panoramic-photography/. Accessed: 2018-05-07.

[64] "Interest Point." https://dsp.stackexchange.com/questions/24346/difference-between-feature-detector-and-descriptor/24370. Accessed: 2018-05-07.

Appendix A

Time Analysis

The speed at which our system operates has a significant role in the evaluation of our framework. To determine at which speeds the cameras capture frames and how fast these frames are processed a time analysis is conducted. Algorithm 11 is used to determine the amount of FPS the cameras capture images. Algorithm 12 is used to determine the amount of FPS are processed by the pipeline. For real-time processing these frame rates should be closely matched.

Algorithm 11: Time Analysis Default Frame Rate

Input: Input frames
Output: Speed at which frames are captured

```
#include <chrono>
auto begin = chrono::high_resolution_clock::now();
for i = 0; i < 100; i++ do
    Capture leftFrame;
    Capture rightFrame;
    Show frames;
end
auto end = chrono::high_resolution_clock::now();
auto dur = end - begin;
auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(dur).count();
auto fps = frames/(ms/1000);
```

Algorithm 12: Time Analysis System Frame Rate

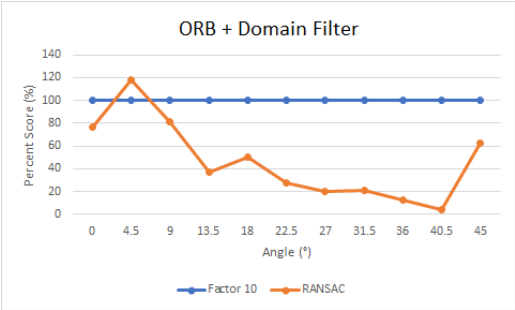
Input: Input frames
Output: Speed at which frames are processed

```
1 #include <chrono>
2 auto begin = chrono::high_resolution_clock::now();
3 for i = 0; i < 100; i++ do
4 |   Enable module(s);
5 end
6 auto end = chrono::high_resolution_clock::now();
7 auto dur = end - begin;
8 auto ms = std::chrono::duration_cast<std::chrono::milliseconds>(dur).count();
9 auto fps = frames/(ms/1000);
```

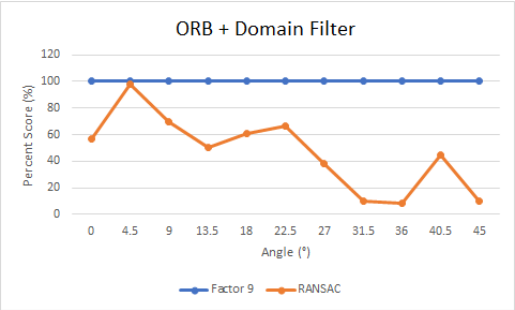
Appendix B

Experimental Results

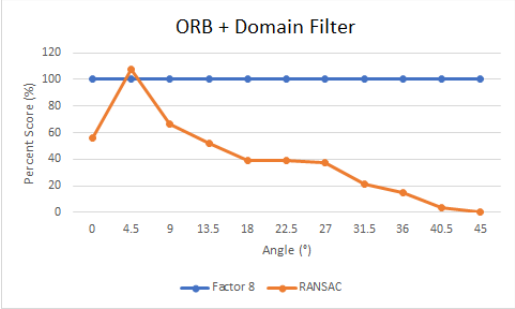
On the next few pages we compare the performances of our filters to RANSAC using the three algorithms. In some of the graphs the angle is limited to a value lower than 45°, this is because after that point not enough matches were found to continue our calculations.



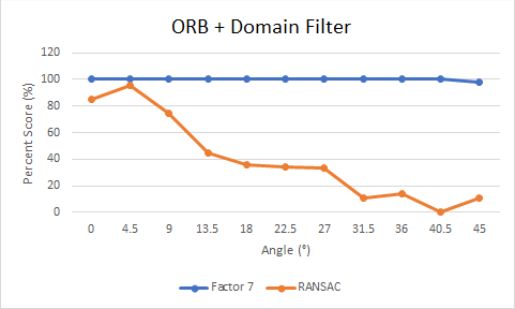
(a)



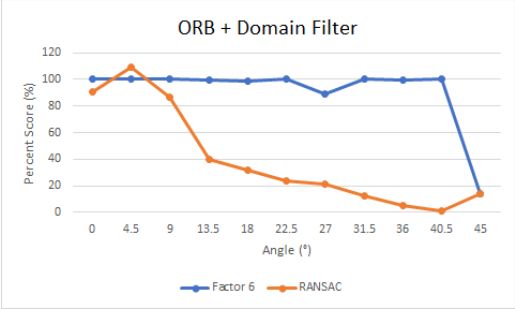
(b)



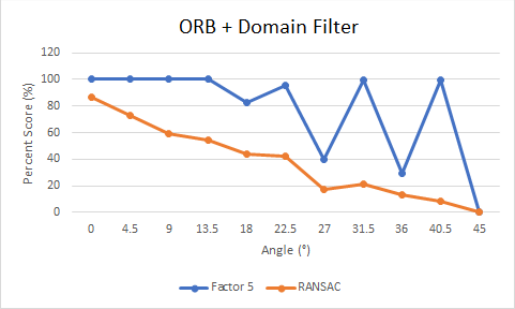
(c)



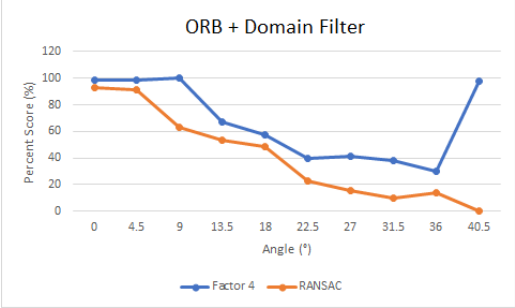
(d)



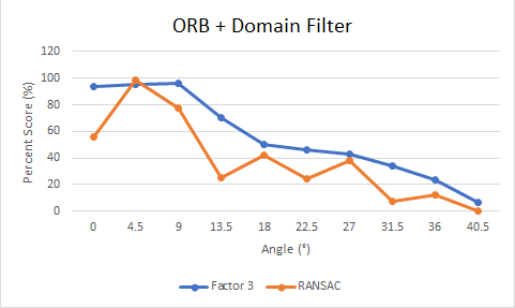
(e)



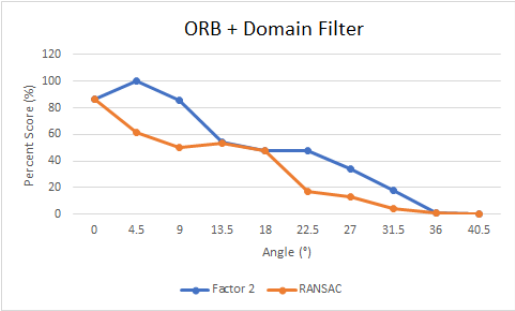
(f)



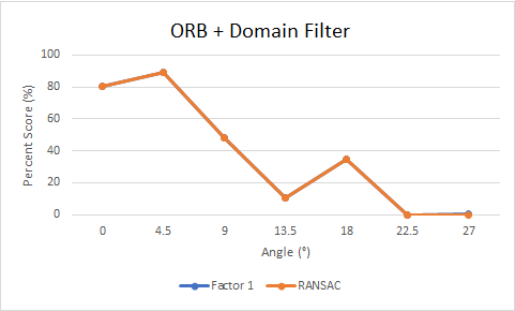
(g)



(h)

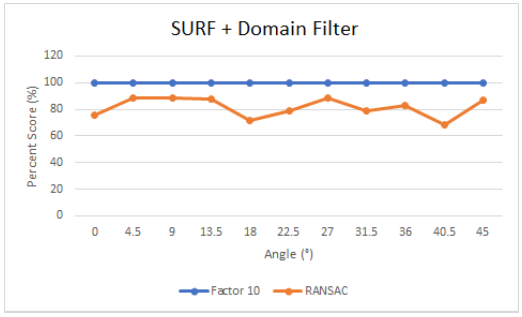


(i)

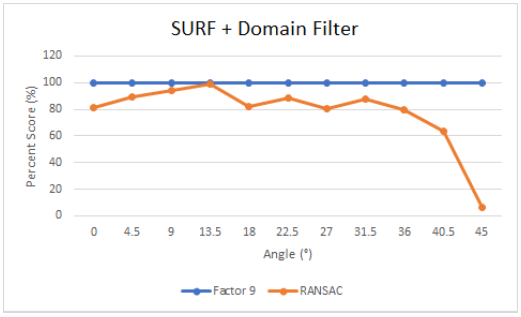


(j)

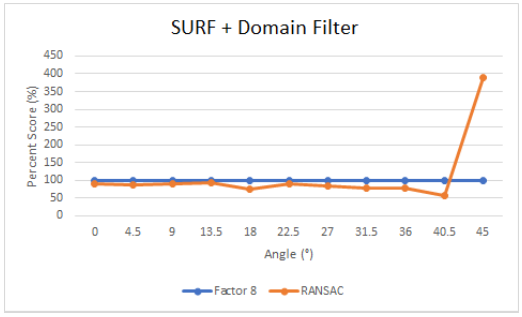
Figure B.1: Performance of ORB with Domain Filter without and with RANSAC at each interval. (**a**) - (**j**): Factor 10 - Factor 1.



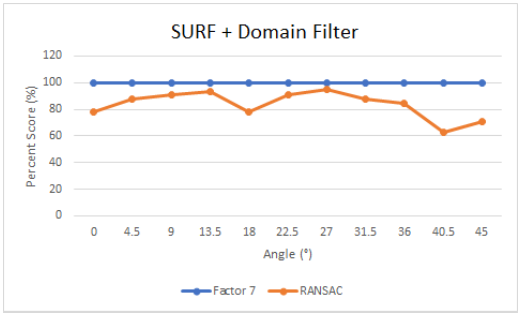
(a)



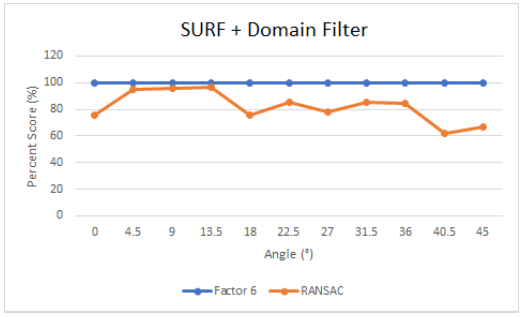
(b)



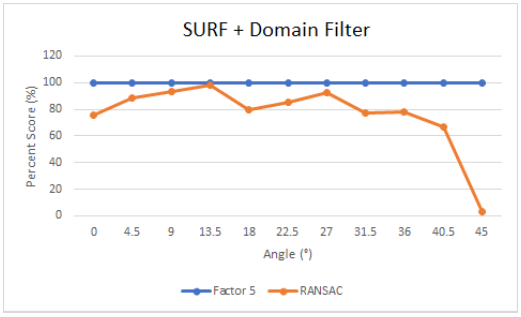
(c)



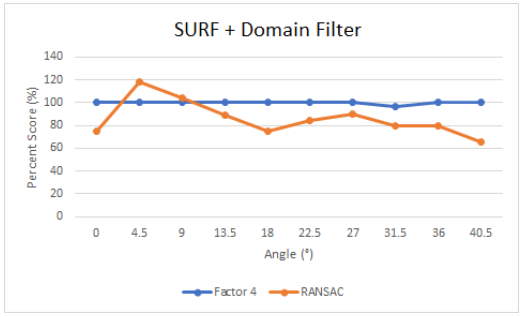
(d)



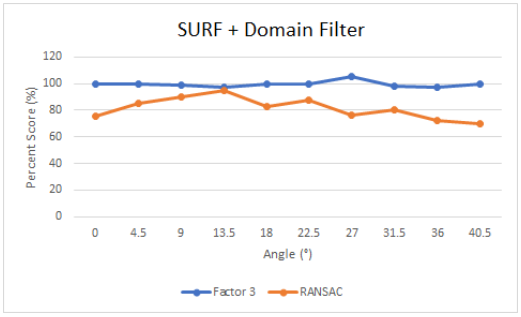
(e)



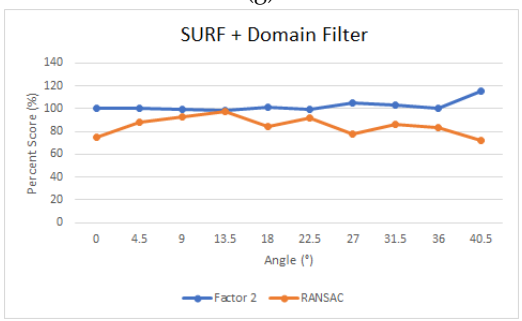
(f)



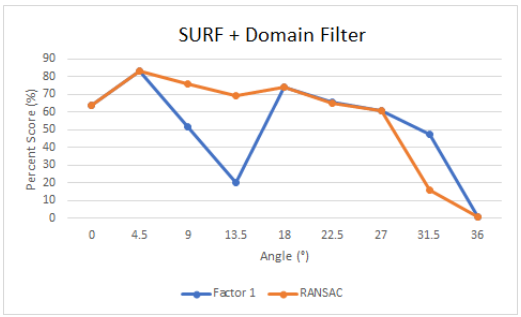
(g)



(h)

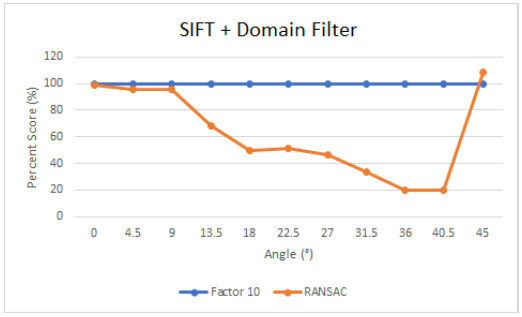


(i)

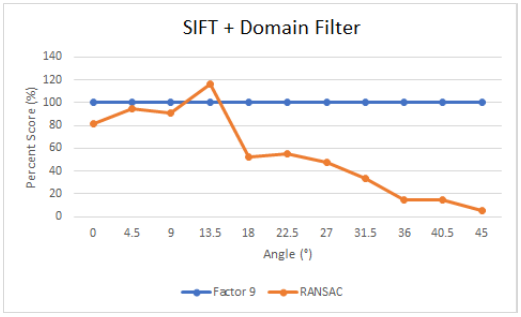


(j)

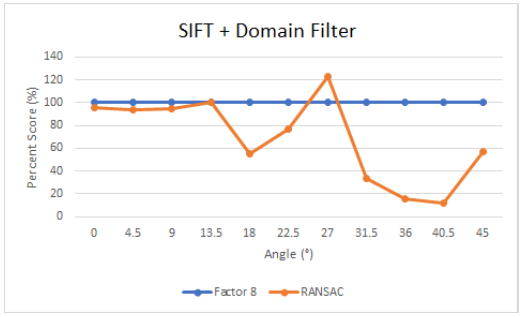
Figure B.2: Performance of SURF with Domain Filter without and with RANSAC at each interval. (**a**) - (**j**): Factor 10 - Factor 1.



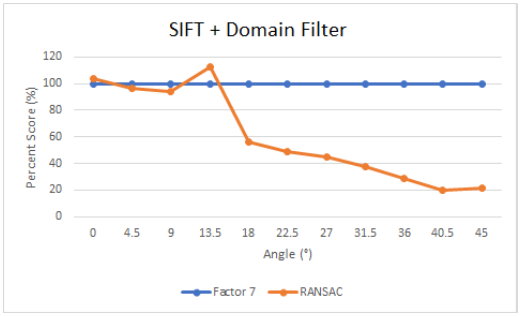
(a)



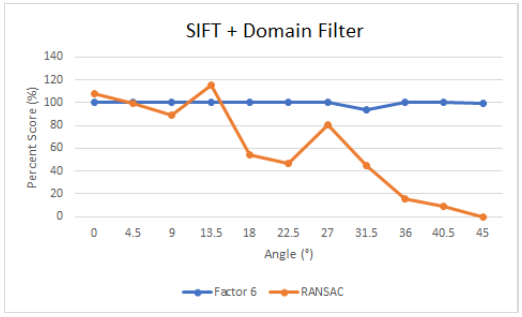
(b)



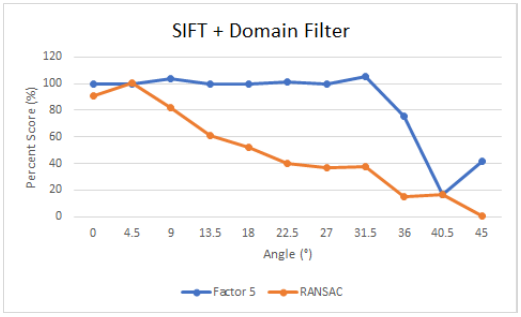
(c)



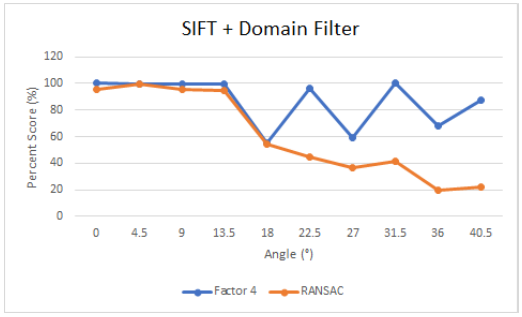
(d)



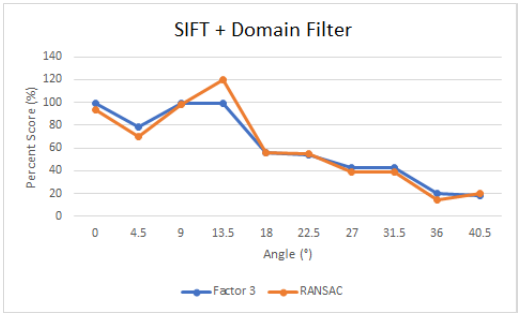
(e)



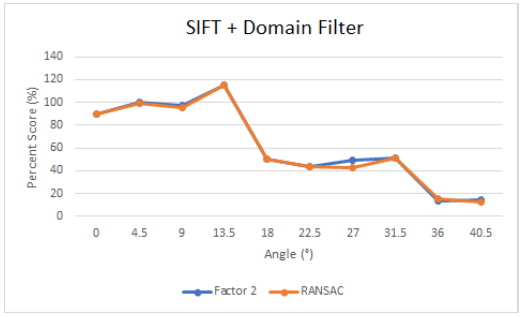
(f)



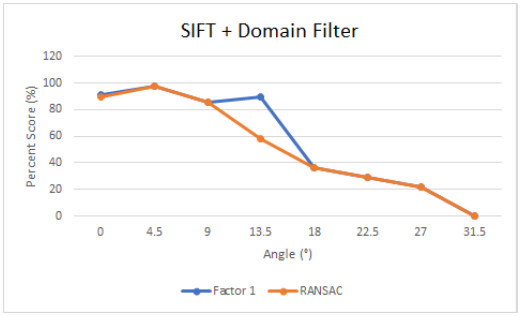
(g)



(h)

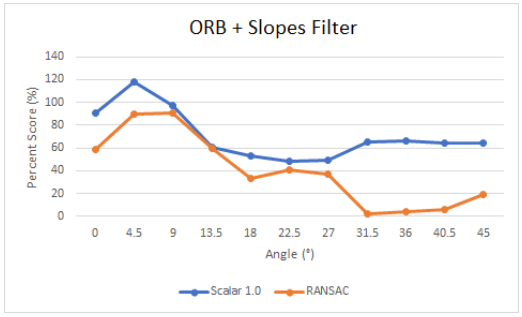


(i)

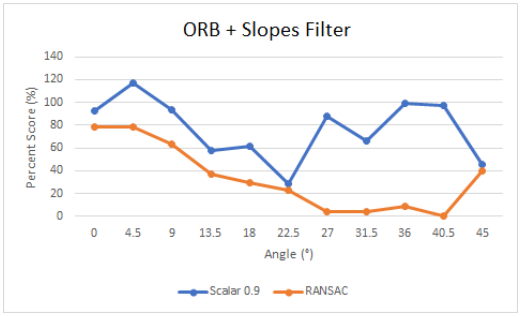


(j)

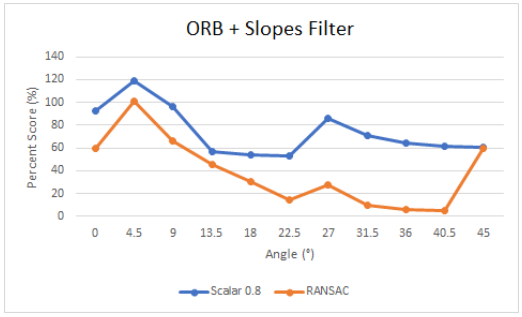
Figure B.3: Performance of SIFT with Domain Filter without and with RANSAC at each interval. (**a**) - (**j**): Factor 10 - Factor 1.



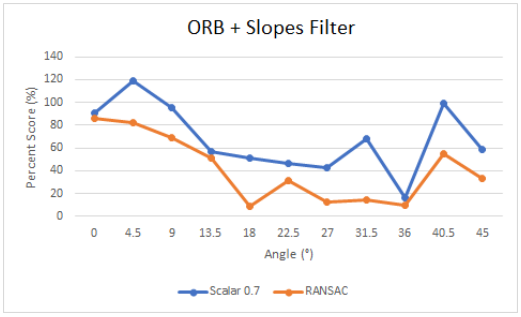
(a)



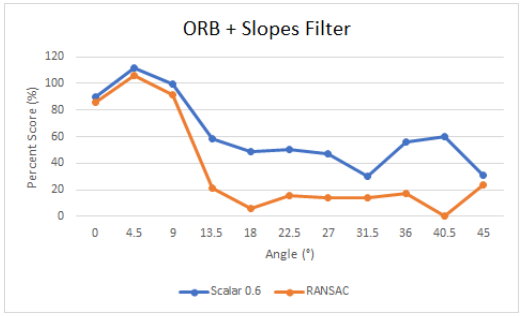
(b)



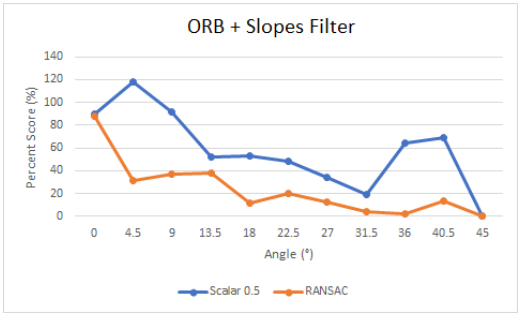
(c)



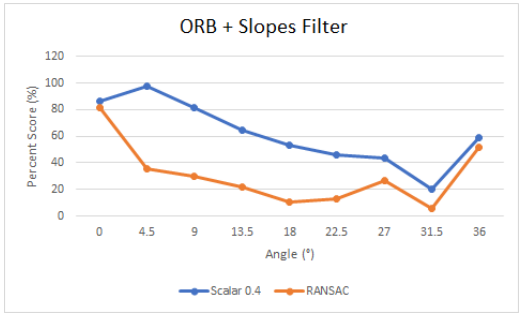
(d)



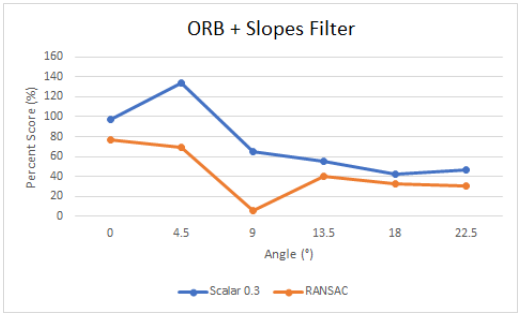
(e)



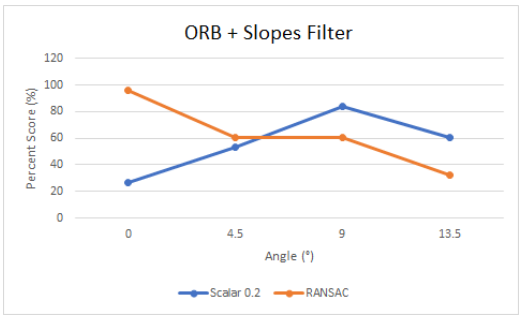
(f)



(g)

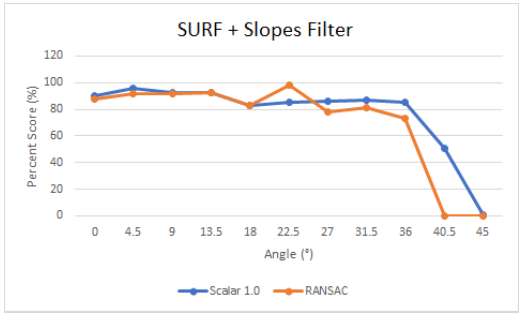


(h)

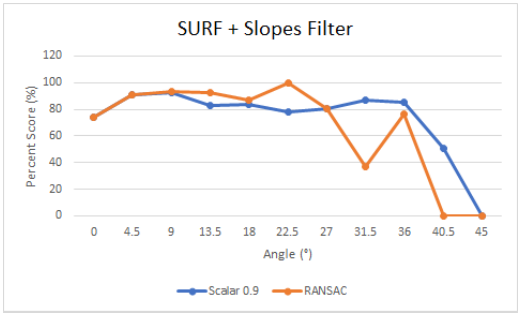


(i)

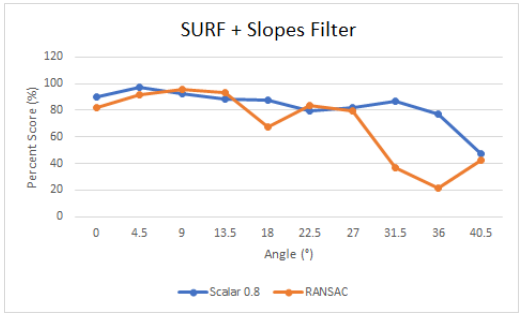
Figure B.4: Performance of ORB with Slopes Filter without and with RANSAC at each interval. (**a**) - (**i**): Scalar 1.0 - Scalar 0.2.



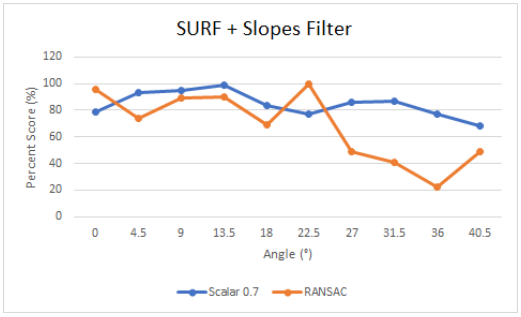
(a)



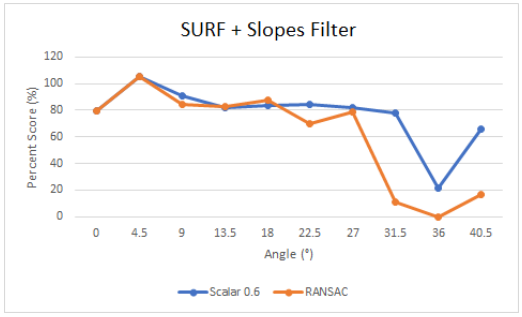
(b)



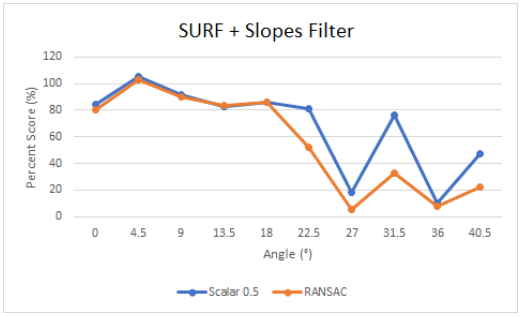
(c)



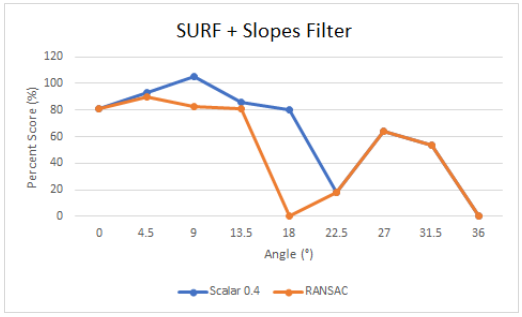
(d)



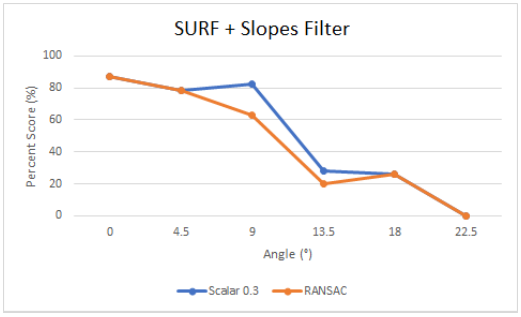
(e)



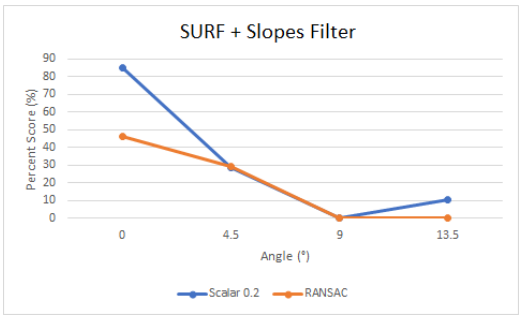
(f)



(g)

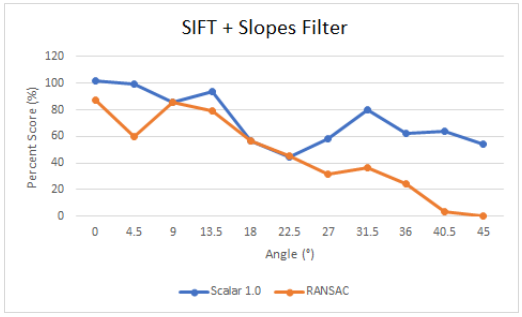


(h)

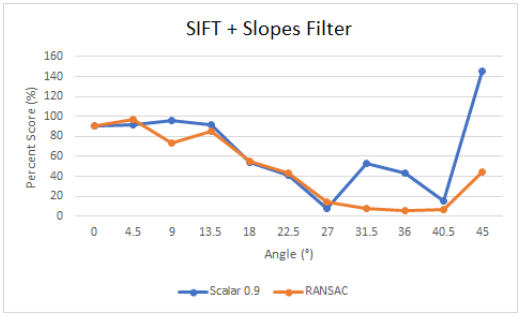


(i)

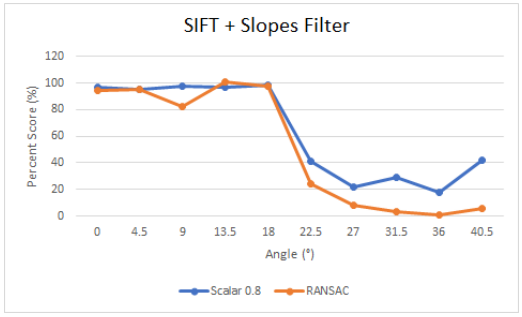
Figure B.5: Performance of SURF with Slopes Filter without and with RANSAC at each interval. (**a**) - (**i**): Scalar 1.0 - Scalar 0.2.



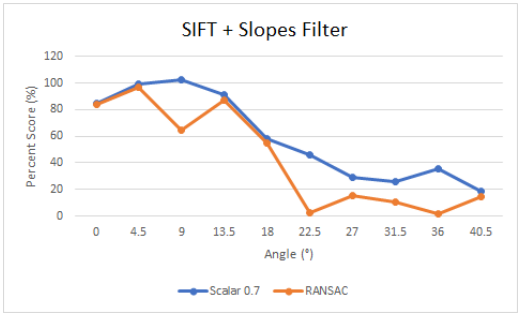
(a)



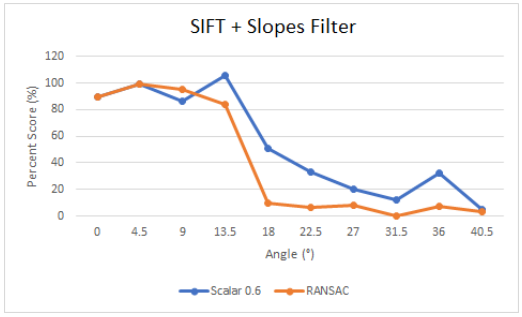
(b)



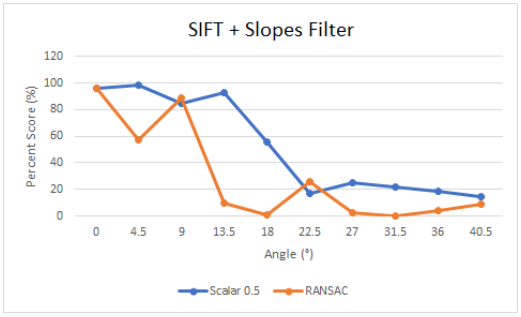
(c)



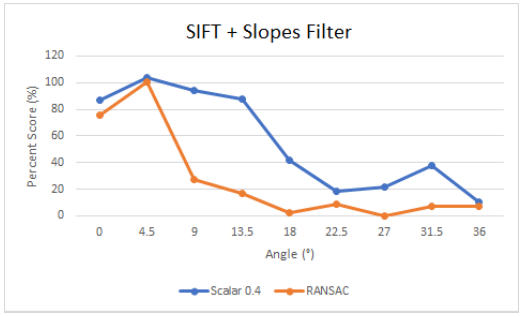
(d)



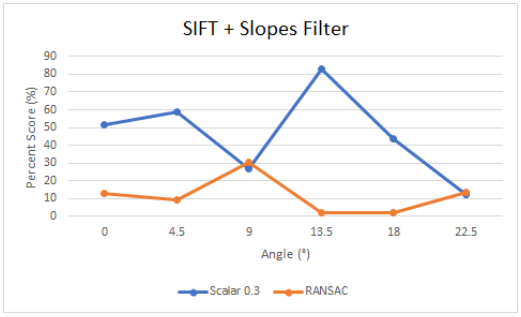
(e)



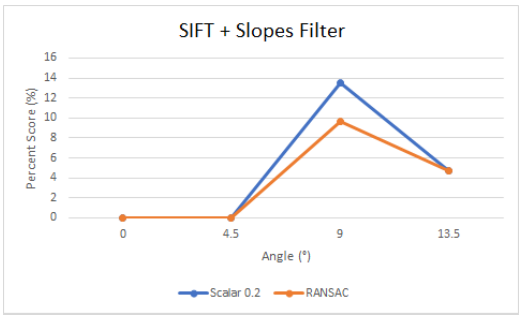
(f)



(g)



(h)



(i)

Figure B.6: Performance of SIFT with Slopes Filter without and with RANSAC at each interval. (**a**) - (**i**): Scalar 1.0 - Scalar 0.2.