Object Tracking Using Local Binary Descriptors

Henry A. Spang V

Follow this and additional works at: https://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Object Tracking Using Local Binary Descriptors

by

Henry A Spang V

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Engineering

Supervised by

Dr. Andreas Savakis
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, NY
May, 2014

Approved By:

_____________________________________________        ___________
Dr. Andreas Savakis
Primary Advisor – R.I.T. Dept. of Computer Engineering

_____________________________________________        _________
Dr. Raymond Ptucha
Secondary Advisor – R.I.T. Dept. of Computer Engineering

___________________________________________
Dr. Sonia Lopez Alarcon
Secondary Advisor – R.I.T. Dept. of Computer Engineering
I would like to dedicate this thesis to my parents, Austin and Linda Spang, my grandparents, Austin and Marti Spang, and my fiancée, Bridget Lally. Without their support this work would not have been possible.
Acknowledgements

I would like to thank my advisor Dr. Andreas Savakis for all of his assistance throughout this process as well as my committee members Dr. Raymond Ptucha and Dr. Sonia Lopez Alarcon. Additionally I would like to thank my colleagues in the Computer Vision Lab Breton Minnehan, Ethan Peters, and Riti Sharma for their work on the original RealD project which served as a starting point of this thesis.
Abstract

Visual tracking has become an increasingly important topic of research in the field of Computer Vision (CV). There are currently many tracking methods based on the Detect-then-Track paradigm. This type of approach may allow for a system to track a random object with just one initialization phase, but may often rely on constructing models to follow the object. Another limitation of these methods is that they are computationally and memory intensive, which hinders their application to resource constrained platforms such as mobile devices. Under these conditions, the implementation of Augmented Reality (AR) or complex multi-part systems is not possible.

In this thesis, we explore a variety of interest point descriptors for generic object tracking. The SIFT descriptor is considered a benchmark and will be compared with binary descriptors such as BRIEF, ORB, BRISK, and FREAK. The accuracy of these descriptors is benchmarked against the ground truth of the object’s location. We use dictionaries of descriptors to track regions with small error under variations due to occlusions, illumination changes, scaling, and rotation. This is accomplished by using Dense-to-Sparse Search Pattern, Locality Constraints, and Scale Adaptation. A benchmarking system is created to test the descriptors’ accuracy, speed, robustness, and distinctness. This data offers a comparison of the tracking system to current state of the art systems such as Multiple Instance Learning Tracker (MILTrack), Tracker Learned Detection (TLD), and Continuously Adaptive MeanShift (CAMSHIFT).
Table of Contents

Chapter 1 Introduction ........................................................................................................................................... 1

Chapter 2 Background ........................................................................................................................................ 3
  2.1. Descriptors ............................................................................................................................................... 3
    2.1.1 SIFT Descriptor ............................................................................................................................... 6
    2.1.2 BRIEF Descriptor ............................................................................................................................ 8
    2.1.3 ORB Descriptor ............................................................................................................................... 11
    2.1.4 BRISK Descriptor ............................................................................................................................ 13
    2.1.5 FREAK Descriptor ........................................................................................................................... 17
  2.2. Tracking Objects .................................................................................................................................... 22
    2.2.1 Continuously Adaptive Mean Shift ................................................................................................. 24
    2.2.2 Multiple Instance Learning ............................................................................................................. 25
    2.2.3 Tracker Learned Detection ............................................................................................................ 27
    2.2.4 SURFTrac ......................................................................................................................................... 29
    2.2.5 Dictionary Tracking ........................................................................................................................ 31

Chapter 3 Tracking with Binary Local Descriptors ....................................................................................... 34
  3.1. Tracking System ...................................................................................................................................... 34
    3.1.1 Dictionary of Descriptors ............................................................................................................... 36
    3.1.2 Search Area Description ............................................................................................................... 37
    3.1.3 Locality Constraints ....................................................................................................................... 41
3.1.4 Scale Adaptation ................................................................. 43
3.1.5 Summary of Tracker Flow .................................................. 45

3.2 Benchmarking System .............................................................. 46
  3.2.1 Accuracy ........................................................................ 46
  3.2.2 Distinctness ................................................................. 47
  3.2.3 Speed ......................................................................... 48
  3.2.4 Robustness ................................................................. 49

Chapter 4 Experimental Results ...................................................... 50
  4.1 Effects of Scaling ................................................................ 50
  4.2 Accuracy ........................................................................... 53
    4.2.1 Parametric Sweep ......................................................... 54
    4.2.2 Unified Parameter Set ..................................................... 62
  4.3 Distinctness ........................................................................ 65
  4.4 Speed .................................................................................. 70
  4.5 Robustness ......................................................................... 79

Chapter 5 Conclusions and Future Work ........................................... 90
  5.1 Descriptor Selection ............................................................. 90
  5.2 Multiple Object Tracking ..................................................... 91
  5.3 GPU Acceleration ............................................................... 92
  5.4 Mobile Implementation ....................................................... 93
5.5. Additional Features ........................................................................................................94
5.6. Conclusions ..................................................................................................................95
List of Figures

Figure 1 BRIEF Sampling Pattern ................................................................. 10
Figure 2 BRISK Scale Pyramid ................................................................. 15
Figure 3 BRISK Circular Sampling Pattern ............................................... 16
Figure 4 FREAK Sampling Pattern, blue circles are sampling points, red circles represent smoothing ................................................................. 19
Figure 5 FREAK Orientation Pattern, blue circles are sampling points, red circles are smoothing, and lines represent pairs .................................................. 21
Figure 6 General Program Flow of the Tracker ......................................... 34
Figure 7 Specific flow of tracking process .................................................. 35
Figure 8 Comparison of search area with and without spacing .................. 39
Figure 9 Descriptor that has been adjusted for the size of the face ............ 41
Figure 10 Gaussian Weights used to enforce locality constraints .............. 42
Figure 11 Scale adaptation method that is performed every frame .......... 44
Figure 12 Comparison of tracker running with and without scale adaptation on Cliffbar 51
Figure 13 Comparison of tracker running with and without scale adaptation on Tiger1 . 52
Figure 14 Comparison of proposed tracker to MILTrack using different descriptors on Cliffbar ....................................................................................... 56
Figure 15 Comparison of proposed tracker to MILTrack using different descriptors on Dollar ......................................................................................... 57
Figure 16 Comparison of proposed tracker to MILTrack using different descriptors on Tiger1 ....................................................................................... 58
Figure 17 Representative frames of tracker running on various videos. From top to bottom the videos are Cliffbar, David, Dollar, and Surfer ................................................. 60
Figure 18 Objects used for distinctness tests. From left to right is the Large Well-Defined Object, Small Well-Defined Object, and Small Poorly Defined Object ..................... 66
Figure 19 SIFT error surface for different size objects ........................................................................ 66
Figure 20 BRIEF error surface for different size objects ..................................................................... 67
Figure 21 ORB error surface for different size objects ....................................................................... 68
Figure 22 BRISK error surface for different size objects ..................................................................... 68
Figure 23 FREAK error surface for different size objects ..................................................................... 69
Figure 24 Object used for all rotation and scale invariance tests ....................................................... 80
Figure 25 Rotational Invariance for all descriptors ............................................................................. 81
Figure 26 Rotational Invariance with the maximum error normalized to directly compare 64-Byte and 32-Byte error ......................................................................................... 83
Figure 27 Scale Invariance for all descriptors ....................................................................................... 84
Figure 28 Scale Invariance with changing Descriptor scales ............................................................... 87
Figure 29 Ground truth comparison for different descriptors on Cliffbar ........................................... 102
Figure 30 Ground truth comparison for different descriptors on Coke Can ........................................ 103
Figure 31 Ground truth comparison for different descriptors on David ............................................. 104
Figure 32 Ground truth comparison for different descriptors on Dollar ............................................. 105
Figure 33 Ground truth comparison for different descriptors on Occluded Face 1 .................. 106
Figure 34 Ground truth comparison for different descriptors on Occluded Face 2 .................. 107
Figure 35 Ground truth comparison for different descriptors on Girl .............................................. 108
Figure 36 Ground truth comparison with different descriptors on Surfer .................................. 109
Figure 37 Ground truth comparison with different descriptors on Sylvester ............... 110
Figure 38 Ground truth comparison with different descriptors on Tiger 1.................. 111
Figure 39 Ground truth comparison with different descriptors on Tiger 2............... 112
Figure 40 Ground truth comparison with different descriptors on Twinings.............. 113
List of Tables

Table 1 Summary of the attributes of the descriptors being used in the proposed system 22
Table 2 Steps of detection phase of the tracker ................................................................. 45
Table 3 Steps of the tracking phase of the tracker ............................................................... 45
Table 4 Effect of scaling on time taken for different portions of the tracking algorithm. 53
Table 5 Results of average distance from ground truth using best parameter set of proposed tracker with different descriptors in use as compared to results of MILTrack. 59
Table 6 Percentage of frames closer than 20 pixels to ground truth ................................. 60
Table 7 The highest accuracy for each of the descriptor video pairings for parameters Radius (R), Threshold (T), and Sigma (S). ................................................................. 61
Table 8 Results of average distance from ground truth using unified parameter set........ 63
Table 9 Percentage of frames closer than 20 pixels to ground truth using unified parameter set .................................................................................................................. 64
Table 10 General Parameter Set for each descriptor ............................................................. 65
Table 11 Time in milliseconds of SIFT Descriptor based tracking ................................. 72
Table 12 Time in milliseconds of BRIEF Descriptor based tracking ......................... 73
Table 13 Time in milliseconds of ORB Descriptor based tracking ....................... 74
Table 14 Time in milliseconds of BRISK Descriptor based tracking ...................... 75
Table 15 Time in milliseconds of FREAK Descriptor based tracking .................. 76
Table 16 Tracker speed over various search radii ......................................................... 78
Table 17 Time in milliseconds of description creation for varying sizes of descriptor.... 79
Glossary

**SIFT**  Scale-Invariant Feature Transform

**BRIEF**  Binary Robust Independent Elementary Feature

**ORB**  Oriented Fast and Rotated BRIEF

**BRISK**  Binary Robust Invariant Scalable Keypoints

**FREAK**  Fast Retinal Keypoints

**CAMSHIFT**  Continuously Adaptive Mean Shift

**MIL**  Multiple Instance Learning

**MILTrack**  Multiple Instance Learning Tracker

**TLD**  Tracker Learned Detection
Chapter 1  Introduction

Visual tracking has long been an important topic in the field of Computer Vision. Significant advances took place in the 90’s with pioneering work of the Lucas-Kanade (KLT) [1] and the advent of CAMSHIFT [2]. More recently, there has been an explosion of methods for tracking that have been profiled in [3], [4], [5]. What has generally been shown by these new studies is that there is an extremely diverse set of trackers currently available, both in model and model-free forms. Though these systems may generate extremely accurate results, they demand computational power. Thus, these tracking methods may not be viable for embedded applications such as mobile devices. There has also been an apparent increase in tracking methods that rely on separating the object from the background such as MILTrack in [6] and Tracker-Learned-Detection shown in [7] and [8].

In a related research area, a lot of effort has been put into describing salient image locations in a manner that is invariant to scaling, rotation, and illumination changes. These features are aptly named local descriptors and they have stimulated a blossoming effort in recent years. Applications of local descriptors range from image stitching [9] to object recognition [10]. Motivated by these contributions, this thesis aims at evaluating a full-fledged tracker based on local descriptors. Object tracking using descriptors is related to image stitching or object recognition. Instead of using the descriptor to match the images, they are used to follow an image patch throughout the video stream. The low computational effort of descriptors would lead to a computationally efficient algorithm that has potential to be ported to a mobile platform.
The main contribution of this thesis is the benchmarking of binary local descriptors for tracking based on a dictionary of templates approach. Furthermore, locality constraints are incorporated via a penalty term, as in [11]. Another contribution is the introduction of a new approach to scale adaptation based on matching descriptors.

Following this introduction, Chapter 2 details the background efforts that have contributed to the proposed system. Chapter 3 explains how the system itself is going to track objects. Chapter 4 provides an overview the experimental results of the tracking system. Finally, Chapter 5 presents the conclusions and future work.
Chapter 2  Background

Image description and tracking are both important fields of computer vision that have a wide and varied approach base. Examination of the current tracking algorithms leads to a clear understanding of their advantages and current shortcomings. Also, in understanding descriptors, it becomes apparent where they might be beneficial for tracking. The following sections discuss local descriptors as well as general tracking algorithms.

2.1. Descriptors

Almost as long as there have been images stored on computers, people have been trying to find ways to accurately describe their content. An important consideration is to decide what points in the image were important to describe. One of the earliest local descriptors is the Harris Corner Detector [12]. Corners are detected in an objective fashion across images and the criteria they are trying to meet are often quantitative so they can be ranked based on the level of fit. This is accomplished via a modified version of the Moravec Corner Detector [13]. This detector examines local windows of an image and calculates their average intensities while undergoing shifts. With the following shifts: \{(1,0), (1,1), (0,1), (−1,1)\}, the change in average intensity can be seen below where \(w_{u,v}\) is the image window, \(I_{u,v}\) is the intensity of a pixel at location \((u, v)\) in the image \(I\), and \(x\) and \(y\) are the shifts:

\[
E_{x,y} = \sum_{u,v} w_{u,v} |I_{x+u,y+v} - I_{u,v}|^2
\]  

(2.1)
This equation is then modified by Harris et al. to become more robust. Their first change was to use an analytic expansion in order to cover all possible small shifts instead of the small set given earlier. This was done via the following equations where \( O(x, y) \) represents an analytic expansion about the origin and \( \otimes \) is the cross-product which is used to estimate the gradient:

\[
E_{x,y} = \sum_{u,v} w_{u,v} [xX + yY + O(x^2, y^2)]^2 \quad (2.2)
\]

\( X \) and \( Y \) are defined in Equation 2.3 and Equation 2.4 as the convolution between an image \( I \) and a default vector:

\[
X = I \otimes (-1,0,1) \approx \frac{\delta I}{\delta x} \quad (2.3)
\]

\[
Y = I \otimes (-1,0,1)^T \approx \frac{\delta I}{\delta y} \quad (2.4)
\]

Continuing along these lines, \( E \) can be represented as:

\[
E(x,y) = Ax^2 + 2Cxy + By^2 \quad (2.5)
\]

Where \( A, B, \) and \( C \) are defined in Equation 2.6.

\[
A = X^2 \otimes w
\]

\[
B = Y^2 \otimes w
\]

\[
C = (XY) \otimes w
\]

From this, a more robust version of \( E \) can be found which causes the corner measure to utilize variation in \( E \) in regards to the direction shift:

\[
E(x,y) = (x,y)M(x,y)^T \quad (2.7)
\]

\[
M_{2x2} = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \quad (2.8)
\]

From this, \( \alpha \) and \( \beta \) can be found as the eigenvalues of \( M \) and \( k \) is a scaling factor. Utilizing these values, a measure of corner quality can be derived. The response can be shown as follows:
\[ Tr(M) = \alpha + \beta = A + B \]  \hspace{1cm} (2.9)

\[ Det(M) = \alpha \beta = AB - C^2 \]  \hspace{1cm} (2.10)

\[ R = Det - k(Tr)^2 \]  \hspace{1cm} (2.11)

\( R \) yields a value that, if positive, indicates the presence of a corner. Essentially, what \( R \) states is that should \( \alpha \) and \( \beta \) be of the same sign and their product is greater than their sum, a corner is found. Now that points can be found with good repeatability, they can be described.

Scale-Invariant Feature Transform (SIFT) [14] is one of the most important descriptors and is considered a benchmark in the field. SIFT is based on the Difference of Gaussian (DoG), and creates a description that is scale, rotation, and illumination invariant. It is also robust to noise thanks to Gaussian filtering. SIFT provides a tool with a high accuracy rate in terms of matching one object or point between different images [9], [14]. However, its floating point based description is computationally expensive. Thus, many advances in the field have been focused on faster feature creation and matching.

The introduction of Local Binary Patterns (LBP) [15] has led to the advent of binary descriptors. These descriptors utilize a series of intensity comparisons to create a binary string, the string being a concatenation of all of the comparison results. This allows for the descriptors to be matched via the Hamming Distance. The Hamming Distance is simply the number of high bits in the result of a bit-wise XOR of two descriptors. This method of scoring can provide a significant speedup over a standard Euclidean Distance or any generic clustering algorithm. By converting a floating point descriptor such as SIFT into a binary string [16], some computational gains can be
achieved. However this approach still has slower creation time compared to a true binary descriptor.

Currently, descriptors are used for a wide range of applications. One of the primary applications for descriptors is image stitching, where points are matched between two images that share a common area [9]. This methodology allows for panoramic pictures to be created. Additionally, the exploration of descriptors in a time sensitive domain such as tracking is important to examine.

\subsection{2.1.1 SIFT Descriptor}

SIFT is one of the pioneering works at local description, but it is one of the more complicated algorithms as compared to the other descriptors discussed in this thesis. The descriptions are based on the Difference of Gaussians (DoG). The Gaussians are generated by running two 1D kernels over the image. The Gaussian kernel used can be seen below where \( \sigma = \sqrt{2} \):

\[
g(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{2.12}
\]

Turning this into a kernel with 7 samples yields:

\[
G = [0.0297, 0.1038, 0.2197, 0.2821, 0.2197, 0.1038, 0.0297]
\]

Using this kernel for convolution allows us to find the DoG (\( \Delta_{\text{Gaussian}} \)) as follows where \( G \) is the kernel above, \( I \) represents an entire image, and * represents convolution:

\[
I_A = I \ast G \tag{2.12}
\]

\[
I_B = I_A \ast G' \tag{2.13}
\]

\[
\Delta_{\text{Gaussian}} = I_A - I_B \tag{2.14}
\]
This method compensates for noise and slight variations in appearance.

SIFT also takes scaling into account by utilizing the concept of image pyramids. This is done by creating several different versions of the given image at different scales. Each new scale is created via bilinear interpolation by (a factor of 2), which causes each pixel at every scale to be a combination of all of its neighbors at a higher scale. By finding the maxima and minima of these pyramids, the same object of a different scale can be roughly located.

The last step in describing a pixel is to find its orientation and gradient magnitude. This information is used to handle in-plane rotation and illumination variations. The following equations are used to describe every pixel in the image where $A_{ij}$ is an individual pixel, $M_{ij}$ is the gradient magnitude, and $R_{ij}$ is the orientation:

$$M_{ij} = \sqrt{(A_{ij} - A_{i+1,j})^2 + (A_{ij} - A_{i,j+1})^2} \quad (2.15)$$

$$R_{ij} = \arctan(A_{ij} - A_{i+1,j}, A_{i,j+1} - A_{ij}) \quad (2.16)$$

Using $M_{ij}$ the illumination variance can be found by putting all the magnitudes through a threshold. By only allowing a variation of 0.1 times the maximum gradient value in the image, changes can be mitigated to a decent extent. By creating a histogram of local image gradient orientations, the peak of the orientation can be found for each key location by examining the peak of the histogram. By multiplying each orientation by the magnitude of the gradient, the orientations that have a more intense or defined value will be more strongly considered in the histogram.

Matching these descriptors to one another is no small task as there is a lot of information to decipher. In order to go through this efficiently, it is proposed to use a modified version of the best-bin-first algorithm [17] and the Hough Transform [18]. Both
of these algorithms are based on clustering/finding neighbors. With the best-bin-first, the higher scale levels of the pyramid are given a larger weight as a way to filter the lower levels which contain less information.

The main concern that has risen from SIFT is that its speed becomes a limiting factor when used within a real-time system. Rectifying the issue of SIFT’s speed has been attempted in several different works such as [19], which is an entirely new descriptor, and [20], which is a modified SIFT for mobile phones. In this thesis, instead of using either one of these faster approaches, the original SIFT is used to provide a base level comparison against the binary descriptors.

2.1.2 BRIEF Descriptor

Binary Robust Independent Elementary Features (BRIEF) [21] is the first of several binary descriptors that are discussed and examined in this thesis. The main reason for the investigation of binary descriptors is due to their ease of matching. For all descriptors of this type, a Hamming Distance (or bit count) can be used. This method of matching can be completed with bitwise XOR methods or even architecture supported bit counting commands. This replaces the often computationally expensive Euclidean Distance or L2 Norm, and greatly increases matching speed. Prior to the efforts seen in BRIEF, binary descriptors had consisted of floating point descriptors being created and converted into a binary string. Although this does increase the speed at which the descriptors can be matched, it does cause the descriptor creation to take even longer. Thus, BRIEF is proposed as a method to create the binary string directly.

This is done by examining a given patch of an image and performing a series of threshold tests based on the pixel intensity. By taking each test as a single bit, the
concatenation of these values into a string yields a descriptive element. The length of this element directly affects the amount of descriptive/matching ability. It is important to find a length that allows for unique keys while not simultaneously causing an exaggerated matching or creation time.

One of the most important considerations that must be made while constructing a binary descriptor is the sampling pattern of the tests. As BRIEF was an early effort in this field, a series of patterns were tried to examine their effectiveness. Figure 1 shows the pattern that has been selected. Each line in the pattern represents an intensity comparison between two pixels. The first four patterns considered were created by randomly selecting pairs from a given distribution. The fifth pattern considered was the only pattern to utilize a systematically generated, symmetrical pattern. The first pattern was a uniform distribution where comparison points are randomly selected, the second pattern was a Gaussian Distribution based on the center of the patch, the third pattern was two Gaussian Distributions (one around the center, one around the other), and the fourth pattern was a random sampling of a coarse polar grid.
The results gathered from these patterns have shown almost equal performance for the first four patterns and substantially lower performance for the final pattern. This shows that in order for a binary descriptor to develop enough distinctness to be useful, it must sample a wide range of areas. When the pattern is symmetric, it only picks up on certain portions of the object, which leads to a less descriptive string. However, utilizing a random pattern gives a wide area for sampling and is able to more effectively describe the entire patch or object.

Much like SIFT, before any of the descriptor is created, there is a level of smoothing that must occur before any intensity tests can be completed. This is to help make the descriptor more robust to noise. Unfortunately, BRIEF by itself has no rotation or scale-invariance, as it is simply a series of comparisons. It relies on an external
detector to be rotationally/scale-invariant. Without the ability to match an object that has
been rotated or scaled, the number of applications for the descriptor becomes limited.
Thus, many binary descriptors that have recently appeared attempt to become more
robust, with respect to rotation and scaling, and remove any dependencies on external
detectors.

2.1.3 ORB Descriptor

Oriented FAST and Rotated BRIEF (ORB) [22] is one such attempt at creating a
binary descriptor that is more robust and invariant to rotation, scale, and illumination.
Instead of removing an external detector entirely, the approach chosen is to integrate an
efficient detector that is not normally orientation aware and modify it to be so.

In order to understand the inner workings of ORB and how it generates
descriptors, it is important to come to terms with the operations of FAST [23] [24]. FAST
is a computationally efficient method for finding corners. It finds corners by performing a
series of comparisons between a pixel and a ring around the radius. Based on the
construction of the ring a consecutive number of pixels must all be higher or lower than
the center pixels by a given threshold. If the number of pixels is met, then the center point
is declared a corner. As it stands, FAST does not provide any measure of the strength of
the corner or any notation of the direction of a corner. In order to create an amount of
robustness to in-plane rotation, the creators of ORB have modified FAST to do so.

The first modification made to FAST is to incorporate the Harris corner measure
as seen in [12]. By filtering all of the corners through the Harris measure, only the
strongest corners (keypoints) are retained. As both FAST and the Harris measure have
variable thresholds, both are used in an iterative fashion. When a specific number of
When descriptors is desired, the threshold for FAST is set low enough to create more than the required number of keypoints. The Harris filter is then used to order all the found points which are then added to the list of kept points. Should too few be found, the threshold is lowered until the desired amount is kept.

The second modification to FAST is the one that gives the orientation component. This is done by finding the centroid of the image patch that is currently being described. The centroid is calculated as follows where \( x \) and \( y \) represent a pixel location \( m_{pq} \) is the moment, \( I(x,y) \) is the intensity of a given point, \( C \) is the centroid, and \( \theta \) is the orientation:

\[
\begin{align*}
    m_{pq} &= \sum_{x,y} x^p y^q I(x,y) \\
    C &= \left( \frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \\
    \theta &= \arctan \left( \frac{m_{01}}{m_{10}} \right)
\end{align*}
\]

In this case, the \( \tan^{-1} \) function must be quadrant aware. What these equations provide is the angle of each patch orientation, which in turn is the orientation component of each keypoint. The modified version of FAST that this creates is referred to as oFAST. Utilizing this orientation, a customization of BRIEF is created, henceforth referred to as steered BRIEF. The customization simply rotates the features created by BRIEF. This is done through some basic linear algebra. By creating a matrix of all of the generated features, a rotation matrix can be generated based on the angle of the keypoints. By multiplying the feature matrix and the rotation matrix, a rotationally modified set of features is created.
Unfortunately, by forcing the descriptors to follow angles, some of the distinctness is removed. Removing distinctness from a descriptor leads to a diminished ability to find the correct matches between points. Thus, in order to make steered BRIEF viable, a learning algorithm was created by Rublee et al. to select the binary tests with the highest variance and lowest common correlation. The method itself is straightforward and is run on a very large dataset. By running every possible, non-overlapping test on every patch in the test set, an extremely high number of tests can be found. These tests are ordered based on their distance from a mean of .5 (the highest possible variance for a binary string). The tests are then iteratively added into a new vector that represents the tests to be kept. The new test is tested against the entire vector to see if it is highly correlated. If the correlation is above a threshold, the test is thrown out. These tests are added until the desired number of bits can be achieved (in this case 256). The ordered tests comprise the descriptor now referred to as rBRIEF. The combination of oFAST and rBRIEF create ORB.

2.1.4 BRISK Descriptor

Binary Robust Invariant Scalable Keypoints (BRISK) [25] is another binary descriptor, though unlike ORB it is not improving upon an old descriptor, it is proposing a new approach.

Much like ORB, BRISK does rely on an implementation of FAST in order to generate keypoints. However, it does use an optimized version of FAST known as AGAST which has been optimized for speed. However, unlike ORB, BRISK does not rely on the Hessian corner score to sort its keypoints. Instead it defines its own method for determining which points to keep. This is done by finding the maximum FAST score
which is defined as the highest threshold that a point can have and still be considered a corner. Points that are kept are the ones that have the highest score out of all of their 8 neighbors. In order to achieve scale-invariance, BRISK also implements a pyramid based method. The pyramids necessitate that each selected keypoint is the maximum score when compared to its neighbors in the pyramid levels above and below it. Unlike ORB, the pyramid scales are not all at a constant scaling factor. BRISK pyramids have extra layers in between the main layers. The factors are 2 for the main level and 1.5 for the inter-levels. This pyramid can be seen in Figure 2.
By sampling and finding keypoints with this multi-resolution pyramid, a mild tolerance to scale can be created which helps to enhance the robust nature of the
The other main variance of BRISK from ORB/BRIEF is that BRISK does not rely on a random search pattern, instead it uses a highly structured search area, though it does still rely on smoothing to help diminish the effect of noise on the descriptor. The sampling pattern can be seen in Figure 3.

Each blue circle represents a point that can be sampled and each red ring represents a Gaussian smoothing. The size of the ring is proportional to the size of $\sigma$ that is being used in the distribution. As with BRIEF, the descriptor itself is created via the brightness comparison of pairs of points. However, instead of a random sampling pattern, the comparisons are generated systematically via the following equations where $A$ represents all the possible combinations, $S$ represents the short range pairings and $L$ represents the long range pairings:

$$A = \{(p_i, p_j) \in \mathbb{R}^2 \times \mathbb{R}^2 | i < N \land j < i \land i, j \in \mathbb{N}\} \quad (2.20)$$

$$S = \{ (p_i, p_j) \in A | \|p_j - p_i\| < \delta_{max}\} \subseteq A$$
$$L = \{ (p_i, p_j) \in A | \|p_j - p_i\| > \delta_{min}\} \subseteq A \quad (2.21)$$
Selecting from these equations to finds pairs yields informative binary strings, however they are not rotation invariant until the gradient is taken into account. This is done via utilization of the long range pairings only, as it has been found that short range pairings do not calculate accurate gradients. The gradient is then utilized to determine the rotation of the descriptor itself, which is another deviation from ORB, whose orientation is that of the detector. The gradient and rotation are calculated with the following equations where \( g(p_i, p_j) \) is the local gradient, \( p \) represents a pixel, and \( I(p, \sigma) \) is the smoothed intensity:

\[
g(p_i, p_j) = (p_j - p_i) \frac{I(p_j, \sigma_j) - I(p_i, \sigma_i)}{\|p_j - p_i\|^2} \tag{2.22}
\]

\[
g = (g_x, g_y) = \frac{1}{L} \sum_{(p_i, p_j) \in L} g(p_i, p_j) \tag{2.23}
\]

By calculating the gradient of every long distance pair, the orientation of the keypoint itself can be found with a simple quadrant aware inverse tangent. Rotating the pattern then allows for the bit string to achieve a level of uniformity. The culmination of this effort is a 512-bit long string that is robust to noise, scale, illumination, and rotation. As with ORB and BRIEF it is matched via the Hamming Distance so it does not suffer any extended matching times.

### 2.1.5 FREAK Descriptor

Fast Retina Keypoints (FREAK) [26] is one of the recent works in the field of descriptors. It is a binary descriptor that shares several common characteristics with the other descriptors. Much like BRISK, it uses the AGAST detector for keypoint generation.
and has a fixed sampling pattern instead of a random distribution of pairs. In a similar style to ORB, it orders its immense possible number of pairs via a learning algorithm, though the actual process does differ from ORB. It also uses the Hamming Distance just like all of the other binary descriptors, though its specific method of matching does have some differences that highlight FREAK’s innovations.

Although FREAK does have a lot of similar aspects when compared with the other binary descriptors, it does propose several new ideas in an attempt to become more effective. FREAK has been motivated by studying the way a human eye processes images. The human eye is designed to see higher resolutions in the center of vision (due to a higher number of photoreceptors and ganglion cells) and lower resolution on the periphery of vision (due to the lower density). This allows for the eye to detect movement easily on the edges and then focus on it to discern what is actually there. Utilizing this, FREAK creates a specific form of search which can be seen in Figure 4.
Figure 4 FREAK Sampling Pattern, blue circles are sampling points, red circles represent smoothing

This pattern does appear to be similar to that of BRISK, with rings of points around a center with different levels of Gaussian Smoothing. However, much larger difference in smoothing levels for the outer rings than the inner rings and the points share overlapping data. The overlapping data allows for more information to be drawn from the individual points. For example, comparing three fields as follows:

\[ I_A > I_B, I_B > I_C, I_A > I_C \]  \hspace{1cm} (2.24)
This only provides extra information if the fields share some data. Thus, overlapping areas allow for less receptive fields in the pattern to be used. By using larger smoothing kernels on the outer rings, it removes more information, thus giving a general idea of what is there instead of specifics, which are found in the inner rings with smaller kernels.

In order to choose a valid combination of comparisons, FREAK employs a method similar to that of ORB, where all possible combinations are examined to see which gives the most diverse results. This is done by first generating a full length (all possible pairs) descriptor; the researchers have used 50 thousand keypoints [26], and put them all into a matrix. Each row in this matrix is a descriptor where each column represents a specific pair comparison. The mean of each column is then calculated and the matrix is then rearranged to have columns with means closer to 0.5 at the front of the matrix. By selecting the column with the mean closest to 0.5 and then iteratively adding columns with a mean close to 0.5 and a low correlation to the already added columns, the pairs for descriptor construction can be found. It was seen that this method has an automatic bias where the comparisons start with the outer, less descriptive area and then work towards the more descriptive area, thus enforcing a coarse-to-fine pattern.

Finally, the coarse-to-fine nature of FREAK must be implemented in the matching method. Thankfully, this is done very easily and does not require any deviation from the standard Hamming Distance. The only change that needs to be made is that the number of bits that have been compared must be monitored. If the distance is already greater than a given threshold after a certain number of bits, the rest of the string can be ignored. The reason this is valid goes back to the construction of the descriptor.
Essentially if the less descriptive section of the descriptor does not match, there is no way that the more descriptive section could because it is already the wrong object. This allows for a higher level of accuracy and a faster matching computation as non-matching descriptors are abandoned earlier.

![Figure 5 FREAK Orientation Pattern](image)

The final portion of FREAK is the rotation invariance, which also parallels that of BRISK. However, instead of using the same points that are being used to generate the descriptor itself (as BRISK does), FREAK generates a new sampling pattern specifically
for orientation, as seen in Figure 5. This pattern is symmetric and is used to generate a gradient with the following equation where $M$ is the number of pairs, $P_o^r$ is the 2D coordinate vector of the center of the receptive field:

$$O = \frac{1}{M} \sum_{P_o \in G} \left( I(P_o^{r_1}) - I(P_o^{r_2}) \right) \frac{P_o^{r_1} - P_o^{r_2}}{\|P_o^{r_1} - P_o^{r_2}\|}$$

(2.25)

This equation allows for the rotation of the FREAK descriptor, which in turns lends an amount of robustness to the descriptor. This is done with 45 points compared to the several hundred of BRISK so it is more efficient in its calculation. This leads to an additional speedup for FREAK. Combining the speedup received in its matching with that of the calculation and FREAK becomes a very efficient algorithm.

### 2.1.6 Summary of Descriptors

Table 1 represents a brief summary of the attributes of the different descriptors that are used within the proposed system.

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>Type</th>
<th>Size</th>
<th>Invariance</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIFT</td>
<td>Float</td>
<td>128 Dimensions</td>
<td>Scale, Rotation, and Global Illumination</td>
</tr>
<tr>
<td>BRIEF</td>
<td>Binary</td>
<td>256-Bit</td>
<td>Global Illumination</td>
</tr>
<tr>
<td>ORB</td>
<td>Binary</td>
<td>256-Bit</td>
<td>Rotation and Global Illumination</td>
</tr>
<tr>
<td>BRISK</td>
<td>Binary</td>
<td>512-Bit</td>
<td>Scale, Rotation, and Global Illumination</td>
</tr>
<tr>
<td>FREAK</td>
<td>Binary</td>
<td>512-Bit</td>
<td>Scale, Rotation, and Global Illumination</td>
</tr>
</tbody>
</table>

Table 1 Summary of the attributes of the descriptors being used in the proposed system

### 2.2. Tracking Objects

In the simplest sense, tracking is simply identifying a region of an image and then looking for it repeatedly in successive images. Upon closer examination, this leads to a
problem of specifying the matching criterion. Thus, a lot of effort has gone into adequately finding and matching specific objects or patches of images. Many different methods exist in the field such as Mean Shift, Point Tracking, Template Methods, and Discriminative Foreground-Background methods.

Point Tracking states that the tracking of an object can be modeled via the correspondence of points throughout a sequence. Generally, Point Trackers fall under the category of deterministic [27] or probabilistic [28]. Deterministic methods make use of assumptions such as objects not moving far, or moving in a smooth motion. They also employ assumptions about the shape of the object not changing once it has been defined. Probabilistic methods are less full of assumptions in that they attempt to model the properties of the object (speed, acceleration, position). By trying to form a statistical link between frames, these methods are generally more robust to occlusions and rotations. However, Point Tracking in general is computationally expensive and not as accurate as more recent methods.

Mean Shift is an established algorithm that has been adapted for tracking in [2] and [29]. It has also been applied to the embedded field with applications optimized for the GPU [30]. The general theory of Mean Shift is to form a procedure which helps to identify maxima in a density function. In a general sense it can be defined as:

\[
m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x)x_i}{\sum_{x_i \in N(x)} K(x_i - x)}
\]  

(2.26)

Where \( x \) is the initial estimate of the maxima location of the density function and \( K \) is a kernel function. Utilizing this equation, it is possible to determine the weight of nearby points. Then \( x \) can be iteratively estimated until convergence is established.
Template Methods served as an inspiration to this work. In a general sense, template based tracking simply states that a template is created which represents the object or patch in question. The data contained in the template is then followed over consecutive frames of a video. Many different methods of template based tracking have appeared over years of research. Templates can be implemented using raw intensity of pixels as their templates [31], [32], and [33]. However, they can also utilize descriptors to make the template invariant to scale and rotation [34]. Depending on the information stored within the template, the matching method will change. These two factors contribute to the computational complexity of template tracking.

Some modern trackers consider tracking as a classification problem. By examining both the foreground (object) and the background (not the object) a classifier can be made to discriminate between the two areas of the image. This is shown in both [6] and [34]. By separating the background from the foreground minor changes in the object can be handled as they are slowly added into the foreground classifier and the small changes in the object are almost always closer to the foreground than the background.

2.2.1 Continuously Adaptive Mean Shift

The desire to track efficiently and robustly has led to a lot of different approaches. One of the main areas of research revolves around statistical analysis as a method of tracking. An algorithm that has received a lot of attention in the field has been the Continuously Adaptive Mean Shift (CAMSHIFT) [2]. CAMSHIFT is similar to the Mean Shift algorithm which operates on probability distributions and attempts to find the peak of the distribution. However, Mean Shift is designed for static or fixed frame images, so
CAMSHIFT has modified it to be able to move around in an image by dynamically changing the observation window, thus making it able to track. The other modifications that have been made have revolved around making the algorithm examine criteria like local color correlations, image gradients, and derivations for a specific region (object) they can be found in later frames. Utilizing histograms to find these distributions also allows for orientation to be calculated, which makes the tracking fairly robust.

Though this method has been shown to work well, it may not meet the speed requirements of a modern system. This is primarily due to its computational expense. In the original paper, it is cited to achieve 30 frames per second (FPS) using a (slow today) processor of 300MHz. However, this is computed on a 160x120 video stream which is much lower than modern video outputs. For instance most modern webcams utilize a resolution of 640x480, which has 16 times more pixels than the lower resolution. As CAMSHIFT is deemed to be $O(\alpha N^2)$ where N is the image dimensions, the slowdown to bring CAMSHIFT up to a modern resolution would be very large.

2.2.2 Multiple Instance Learning

Another method that has surfaced gained popularity and provided excellent tracking results has been the Multiple Instance Learning (MIL) Tracker (MILTrack) [6]. This tracker is based on the world of object detection, which is based in statistical analysis. The main goal of the MIL tracker is to remove some of the ambiguity that revolves around detecting (and tracking) an object. Essentially, when a detector is being trained via a supervised method, it is given a series of specific bounding boxes that declare their contents as a positive example of a desired object (such as a face). Unfortunately, this approach can lead to errors as it is extremely hard to have the objects
cropped exactly the same in every instance. This leads to an ambiguity with how the object is defined and can then lead to the detector becoming confused. MIL attempts to solve this dilemma by creating the detector with the ambiguity already included in it. Instead of providing specific examples of an object, one would simply enter the center of the object and let an algorithm pick out several rectangles around it. Adding these entries into a ‘bag’ of positive examples, the detector is trained on slightly obfuscated data. This allows for the detector to work better in real world scenarios where perfect examples of an object are much less likely to exist.

MILTrack attempts to rectify the issue of maintaining information about an object and being able to track it as it changes throughout a video stream. The tracker runs, only with information about the object from the first frame being used to initialize it. This lack of overhead is invaluable in tracking, as it allows for the tracker itself to be run on a much wider array of videos and objects.

MILTrack functions by creating a set of positive examples at each frame of the image. These are all samples from around where the object was found. These examples are all put into a single positive bag. In addition to positive examples, the appearance model of MILTrack must be updated with negative samples. These are recovered in the same manner; however they are all sampled from outside the sampling range of the positive samples. Since this can lead to a large number of negative samples, they are randomly selected from their complete set before being passed in as negative bags. These bags are used to update a series of weak classifiers which are boosted into one strong classifier. This is then used to find the object in the next frame and the algorithm repeats. The constant updating of the appearance model in MILTrack allows it to handle the
changing nature of tracking an object without any supervision necessary. This feature makes it ideal for the robust tracking, though its method is complicated and thus there is room for a speed increase.

### 2.2.3 Tracker Learned Detection

Tracker Learned Detection (TLD) is a system that utilizes a tracker and a detector. Though this portion may be common, it is the interactions between the two that are unique. Each frame the tracker is used to estimate the motion of the object. From this estimation of motion, there is a chance of drift or error, so a detector is used to help correct the tracker on each frame. In addition to the detector correcting the tracker each frame (if needed), it is retrained every frame in order to lessen error over time.

The training method proposed in this tracker is referred to as P-N Learning [7]. This method of learning is a supervised technique which means that the data going into classification training is selected by the algorithm instead of randomly selected. P-N Learning introduces the concept of Experts. These Experts have the ability to monitor the current state of detector training and then estimate any errors that have occurred. The error estimation is used in the next iteration of training to try and improve the detector further. There are two types of Experts, both of which are independent from one another. P-Experts handle error cases that are false negatives and N-Experts handle false positives. For both of the experts, when a false case is found it is relabeled to be the opposite and then added back into the training set with this new label. The P-Experts help to increases the classifier’s generality and the N-Experts increase the classifier’s discriminability. Their independence allows for each expert to mitigate the error of the other. The
classifier is then trained again and the steps repeat. This whole process is repeated \( k \) times.

In addition to the Experts, TLD contains an object model, an object detector, and a tracker. The model itself is simply a series of image patches that are classified as either positive or negative. With the addition of new patches, a nearest neighbor (NN) classifier is used to decide where each patch should go by finding whether they most closely resemble the positive or negative examples. The model itself is only updated if certain thresholds are met so that the model does not become too large to utilize and so that bad data does not enter the model.

The detector runs on every frame and due to search regions it has a fair number of patches to classify. Thus, efficiency is essential or a real-time speed is not obtainable. This is handled by giving the detector three distinct phases. Each of these phases has the ability to reject or pass a patch through, thus limiting the total number of patches that must be classified by the detector. The first stage examines the gray variance of the patches. If the patch has a variance smaller than 50% of the desired object patch it is rejected. This has a high rejection ratio and can be done very efficiently using integral images which is defined in Equation 2.27, where \( f(x, y) \) is a pixel in the image and \( I(x, y) \) is an element of the integral image. The integral image allows for the efficient computation of the sum of an area in an image. This can be seen in Equation 2.28 where \( A, B, C, \) and \( D \) are the corner points of a rectangle.

\[
I(x, y) = \sum_{x' \leq x} \sum_{y' \leq y} f(x', y')
\]  

(2.27)
The second phase is an Ensemble Classifier. The purpose of this stage is to perform a series of intensity comparisons of points that were randomly generated offline. These comparisons create a binary code which indexes a series of posteriors. If the average posterior is greater than 50%, the patch is considered a pass and is sent to the final phase which is an NN classifier. This stage simply takes the remaining phases and decides which is closest to the positive templates stored in the model. If it is higher than a set threshold, it is determined to be the object.

The tracker itself is based on a Median Flow tracker but it also has failure detection. This method estimates the motion of the object and estimates the number of points that will still be within a bounding box. The motion is estimated by a Lucas-Kanade tracker which utilizes two pyramid levels to mitigate scale-based errors. The failure detection is handled by comparing a single point movement to the median movement. If the absolute value of their difference is greater than a given threshold, it is declared a failure. This helps to handle rapid movement and occlusion.

2.2.4 SURFTrac

SURFTrac is an excellent example of a descriptor being utilized for tracking/continuous object recognition. One of the main points of this tracker is the decoupling the descriptor from the feature point detector. Whereas most descriptors come with a feature point detector and all of those points are described. SURFTrac has utilized a point detector, but then has removed the description of all points that is usually seen. This was done to save computational cycles and allow for the tracking to run in real-time.
The detection phase is also the tracking phase in this system because the detected interest points are what is actually being followed. The detection occurs in scale-space and utilizes image pyramids as seen in such attempts as Hessian-Affine, Harris-Affine, and approximate Hessian. This allows for the interest points that are detected to be filtered so that only the best responses are received. The interest points are generated from a series of video frames via an approximation of their motion. Given a previous frame of $I_{k-1}$ with an interest point $p_{k-1}$ defined as $p_{k-1} = (x_{k-1}, y_{k-1}, \sigma_{k-1})$, the interest point in the next frame can be transformed based on the motion between the two frames $M_{k}^{k-1}$.

$$p_{k} = (x_{k}, y_{k}, \sigma_{k}) = M_{k}^{k-1}(p_{k-1})$$  \hfill (2.29)

However, this is not enough to get consistent points so instead, a 3D search is generated using $\Delta_{\sigma}$ as the search range and $\gamma_{\sigma}$ as the motion prediction error. Using these constants the interest point neighborhood can be found.

$$P_{k} = \left\{ (x_{k}', y_{k}', \sigma_{k}'): \begin{array}{l} |\sigma_{k}' - \sigma_{k}| \leq \Delta_{\sigma}, \\
|\sigma_{k}' - \sigma_{k}| \leq \gamma_{\sigma_{k}'}, \\
|y_{k}' - y_{k}| \leq \gamma_{\sigma_{k}'} \end{array} \right\}$$  \hfill (2.30)

Going forward from this, the interest points are tracked, but again, without using the SURF descriptor. Instead, the interest point tracking relies on information that is already available in the scale-space where they were found. First, the simplest option is to remove all points that are not predicted to be in the region of interest. If there is more than one point in the region of interest, it must be described in order to find the actual match. This is done utilizing the Hessian Matrix $H$. Two ratios are defined when matching the interest points the first $r_{1}$ is defined as the eigenvectors of the Hessian...
Matrix \( r_1 = \frac{\lambda_1}{\lambda_2} \). The second ratio is defined as where \( \text{trace}(H) \) is the sum of the diagonal elements of the matrix \( H \):

\[
\frac{r_2}{r_1} = \frac{\text{trace}(H)^2}{\det(H)}
\]

(2.31)

As \( H \) has already been found, \( r_2 \) requires less computation and is thus used over \( r_1 \). So, a pair of interest points between frames is defined as the pair with the lowest difference in \( r_2 \).

Another method that was utilized to match interest points was Normalized-Cross-Correlation (NCC). This method was modified so that it could function in the Hessian domain instead of utilizing pixel intensities which are often not descriptive enough when in a tight clustering around a point. Instead, the Hessian determinant was used as a signature. In order to find the best match, a frustum around each interest point was created as a 5x5x3 grid. The L2 norm between the current frame’s points and the previous frame’s points was used to find the best match.

Finally, SURF itself is used only as an object recognition tool. When there are free cycles, the feature points being tracked can be described and then matched against an image database. This is only done when there is spare time because doing it every frame would be extremely cost prohibitive. So, this method does show that descriptors can be used to repeatedly recognize an object, however it does not utilize the descriptor itself for tracking.

### 2.2.5 Dictionary Tracking

Distance Metric Learning (DML) [34] served as an inspiration for this work with its use of descriptors in a dictionary setting in addition to other techniques. The dictionary
for this method is populated with a series of templates that are a representation of the image. In an attempt to make this both efficient and invariant to change (illumination/orientation), SIFT keypoints are used to keep track of the patch. Over a 50x50 patch, 9 separate SIFT descriptors are created yielding a $9 \times 128 = 1152$ dimension description of the patch. In order to further diminish the computational load, dimensionality reduction is applied. Instead of using a standard statistical method such as Principle Component Analysis (PCA), Random Projections (RPs) are utilized. RPs take the form of a matrix that is populated with the following elements:

$$
\begin{align*}
    r_{ij} &= \sqrt{3} \begin{cases} 
        1, & \text{with probability } \frac{1}{6} \\
        0, & \text{with probability } \frac{2}{3} \\
        -1, & \text{with probability } \frac{1}{6}
    \end{cases} 
\end{align*}
$$

(2.32)

Multiplying this matrix with a data sample (in this case the 1152 dimension template), two thirds of the data is thrown out. This allows for a significant reduction in dimensionality without the need to train or perform statistical analysis on the data as the matrix is made once at the start.

In addition to templates carrying SIFT descriptors to describe patches, this method also utilizes DML. The purpose of DML is to create a new distance metric that can more accurately score the information being given. This tracker has utilized Information Theoretic Metric Learning (ITML). Based on the general sense of DML where $G$ represents a distance metric, ITML takes $G_0$ as an initial Mahalanobis distance matrix. From this initial matrix it attempts to create a new Mahalanobis distance matrix $G$.
that will meet the constraints of being similar to $G_0$ and also satisfy the class labels constraints.

The similarity is handled by relating the matrix $G$ to a multivariate Gaussian distribution given by:

$$p(x; G) = \frac{1}{Z} e^{-\frac{1}{2}d_G(x, \mu)}$$

(2.33)

In this $\mu$ is the mean and $Z$ is a normalization constant. The Kullback-Leibler (KL) can be leveraged as a distance metric between $G$ and $G_0$ and is defined as follows:

$$KL(p(x; G_0)||p(x; G)) = \int p(x; G_0)\log \frac{p(x; G_0)}{p(x; G)} dx$$

(2.34)

From this the matrix can be tailored to fit the class labels by minimizing the distance between certain class boundaries. The distance between members of the same class and members of a different class should always be lower for the same class members.

When applied to the realm of tracking, the problem becomes a binary classification problem. As the tracker runs, patches are taken that depict the foreground and background. This is done simply by selecting a winner and then taking the patches around the winner. By taking these two classes, the tracker can be updated overtime to look more for the foreground than the background. This updating method is simply an online DML method which allows for a real-time update instead of the offline method used to bootstrap the original metric.
Chapter 3  Tracking with Binary Local Descriptors

The usage of descriptors as a tracking tool and the necessary modules needed to make them work are explored in this chapter. The benefits and disadvantages of this tracking system are used to benchmark all of the individual descriptors. The portions of the algorithm proposed by this thesis can be seen in Section 3.1.1, Section 3.1.2, and Section 3.1.4. In addition this thesis also contributes the results drawn from the system defined in Section 3.2.

3.1. Tracking System

In order to operate in real-time, this tracking system is based on of the matching abilities of binary descriptors. Thus, a framework must be developed around the description process to enable tracking/matching.

![Image of General Program Flow of the Tracker]

Figure 6 General Program Flow of the Tracker
A basic representation of what the algorithm is doing can be seen in Figure 6. It is clear from the diagram that there are separate execution paths. The topmost is for detection and it is generally only run once on the test videos. This allows the tracker to run on any video stream with only a single frame of knowledge about the image patch to be tracked. The lower path is the tracking path which is run on all other frames. Once static dictionary is full and a location of the object has been provided, the system can begin to track it. To begin tracking the search grid must be defined around the object. The points in the search grid are then described and matched to the dictionaries. The candidate with the lowest score gets accepted as the match and the process repeats itself. Abstracted to this high level, the algorithm is not very complicated, which helps with both development time and actual runtime speed.

![Diagram](image.png)

**Figure 7 Specific flow of tracking process**

Figure 7 is a more specific representation of the main flow of the system. It is shown here is that there are really only 5 sections of the algorithm. The search grid will be defined later on in Section 3.1.2, describing the candidates will be based on binary local descriptors defined in Section 2.1, scale adaptation will be defined in Section 3.1.4, and finally the locality constraints will be described in Section 3.1.3.
3.1.1 Dictionary of Descriptors

As seen in [34], the dictionary is responsible for holding a series of descriptors that were created under scrutiny to represent the object at hand. The initialization can be as simple as clicking on an object in a video stream, to a more sophisticated method such as using an object detector. This allows for any object (via clicking or detection) to be tracked without the usage of a ground truth file for initialization. Assuming that a detector or method for assigning an object exists (both readily available through open source projects, e.g. OpenCV [35], or tools such as MATLAB), this tracker has an impressive amount of generic tracking ability.

An issue with setting up a dictionary to hold “useful” descriptors at the start of a video stream is that most objects will not stay exactly the same throughout the duration of a video. The tracked object is likely to move forward or backward (change scale), rotate, succumb to occlusion, or undergo illumination changes. This can be mitigated to a mild degree by populating the static dictionary with several different views of the object. However, this is not perfect as the object will most likely not change very much in the first few frames of motion. The static dictionary can take new entries over the length of the video, but detection is a slow process so avoiding it is beneficial to the tracker processing speed. Thus, in order to obtain true robustness to the average difficulties faced in a video, the dictionary is divided into two distinct sets. There is the static dictionary, which contains the original descriptors found at the start of the program. Then there is the dynamic dictionary, which contains the most recent object matches. By updating this on a per frame basis, it is possible to account for rotation and scale changes as they will occur over time so each new frame is only slightly different than the last.
The final steps that need to be considered for this method to be fully viable are to deal with drift and occlusion. To prevent drift and occlusion from affecting the tracker, a threshold that bars entry into the dictionary set is established. By utilizing the Hamming Distance, defined as $d_H(c_i, t_j)$ where $c_i$ is the candidate of the search grid and $t_j$ is the template in the dictionary, the score can be found.

$$S^l_H = \arg\min_{(t_j \in D)}\{d_H(c_i, t_j)\}$$

Comparing this with a threshold yields the new dictionary set.

$$D = \begin{cases} 
    t_0 = c_{win} \text{ and } t_1 \ldots t_n = t_0 \ldots t_{n-1}, & \text{if } S^w_H \leq \text{Threshold} \\
    D, & \text{if } S^w_H > \text{Threshold}
\end{cases}$$

This stops issues like occlusion from stopping the tracker from functioning as an occluded object’s Hamming Distance will be much too high for it to be considered a proper match. It also helps to prevent drift as any candidate that has become too different from the original object (high Hamming Distance) will not be entered into the dictionary.

### 3.1.2 Search Area Description

The search area is one of the most important portions of efficiently implementing the tracking algorithm. The accuracy may be determined by the uniqueness of the descriptor, however, without a large enough search area a fast moving object may not be found. Unfortunately, increasing the search area can have a debilitating cost on the speed of the algorithm. Thus, without an appropriately defined search area, the tracker can lose the object by virtue of not searching far enough. Conversely, if the tracker has too large of a search area it may fall below real-time speeds which could cause skipped frames and lead to losing the object.
A solution based on a fine-to-coarse search methodology was developed for the tracker to run efficiently while tracking fast moving objects. What this yields is a grid of candidate points to search where the center of the grid checks every point (fine) and the outer regions of the grid search with a separation between the points (coarse). The rationale of the approach is that the object is not expected to move very far in any given frame, thus a fine search is needed. To account for large movements it is not efficient to simply define a very large, densely sampled search area. Thus, the search area is defined in two regions, the fine and the coarse. The fine region is a small region around the center of the search area. The coarse region starts at a given distance from the center point and simply increases the inter-pixel width of the region. In terms of an actual implementation, the search area is defined as a series of deltas that are applied to the center location, as shown in Figure 8. They are defined once during the start of the program and can be seen below where $r$ is the radius of the search region, $\alpha$ is the spread-scaling constant, and $i, j$ are loop variables:

$$\Delta_{i,j} = (i + \delta_x, j + \delta_y)$$

$$\delta_x = \begin{cases} 
0, & \text{if } -\frac{r}{2} < i < \frac{r}{2} \\
\alpha \left( i - \frac{r}{2} \right), & \text{if } i \geq \frac{r}{2} \\
\alpha \left( \frac{r}{2} + i \right), & \text{if } i \leq -\frac{r}{2} 
\end{cases}$$

$$\delta_y = \begin{cases} 
0, & \text{if } -\frac{r}{2} < j < \frac{r}{2} \\
\alpha \left( j - \frac{r}{2} \right), & \text{if } j \geq \frac{r}{2} \\
\alpha \left( \frac{r}{2} + j \right), & \text{if } j \leq -\frac{r}{2} 
\end{cases}$$

By utilizing these equations, the search area can be extended to handle a wide range of movement speed without sacrificing FPS. For example, if $r$ is 20 and $\alpha$ is 1,
without these equations the maximum distance from the center would be ±20. Using the equations it goes to ±30, which is an increase of 150% without any slowdown in the program and without sacrificing the interior region which still has a spacing of one pixel. Figure 8 shows the effect of the spacing with the parameters set to the above values. Each mark represents a point that will be described. Despite having a larger spacing between pixels, the entire area of the larger spaced region appears covered. This is due to the inherent nature of the search radius having a large number of points. The total number of points in the search area can be represented as \((2r + 1)^2\), so with \(r = 20\) the total number of points is 1681. Increases in the radius to cover the larger search area without adjusting the spacing would yield 3721 points. So, an increase of 50% in the actual radius would yield an increase of over 100% for the number of points utilized, causing a significant slowdown.

![Figure 8 Comparison of search area with and without spacing](image)

Utilizing a search region, allows for a specific matching to be accomplished. With this method, there is always a constant number of candidates per frame with one being selected as the winner. However, descriptors like ORB rely on a detector to create a
series of keypoints (points to describe). These keypoints are generated to a specific target number of them; however there are instances where it is simply not possible. This would cause serious issues in the matching, as there would be scoring discrepancies between two frames with different number of points. This style of keypoint generation may cause an issue for some descriptors, such as ORB, which use their detected keypoints to find orientation, or sort them to find the best possible values to describe. Without this level of sorting or removing points based on corner scores, it is possible that a lot of bad descriptors or non-discriminative descriptors could be generated.

One of the benefits of the descriptors is their ability to operate at different scales without being affected too drastically. This is especially true of descriptors such as BRISK or FREAK, which can simply expand the range of their patterns without much penalty in speed. Thus, by first defining the size of the object that will be tracked, the descriptor region can easily be modified to fit the object. This also allows for the search area to remain constant for all video resolutions, as the descriptor itself will reach further out if needed for high resolution videos. Figure 9 represents a descriptor that has been sized to fit the face according to the result of face detection. The size of this descriptor is based on the response of a Viola-Jones face detector. This allows the tracking algorithm to be completely generic for faces with no operator intervention. Should generic object tracking be the desired approach, the initial size must be given via a detector output, or ground truth information or by clicking on a frame of the video.
3.1.3 Locality Constraints

One of the main concerns with increasing the search area is that the descriptor will no longer be distinct enough to form an appropriate match. In order to combat this, a form of locality constraints is utilized in order to stop errant matches, as introduced in [11]. This is also done in order to reduce jitter in the tracked object, operating on the assumption that the object does not make large jumps between consecutive frames. The constraints are represented as a series of Gaussian weights. By adding these weights as a penalty to the scoring function, they can be decoupled from the actual tracker. This allows for the object to move freely and to undergo transformations, as there is no strict constraint. The locality constraint is simply a way to bias the object location towards the center of the search radius. The actual weights are generated as follows:
\[ weight(x, y) = scalingFactor * \left( 0.5 - e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} \right) \] (3.6)

This allows for an array of weights to be used to add a penalty (or benefit) to certain areas of the search region. The closer to the center of the region the point is, the better are the chances of it getting selected. This specific Gaussian is designed so that the center point actually subtracts from the Hamming Distance instead of adding to it.

Figure 10 shows the locality constraint for a search radius of 20. The Gaussian distribution is designed so that 99% of the data will be under the maximum penalty. This is done by setting sigma as follows:

\[ \sigma^2 = \frac{radius}{3} \] (3.7)
When creating these weights, the scaling factor must be set appropriately so that there is still a chance of finding the object at the outer reaches of the search region. If the penalty is too high, the tracker could lag behind a fast moving object. Conversely, if it is too low, then the tracker has the ability to match a false positive. Thus, it is important to find the best $\sigma^2$ value for the Gaussian weights. A value that is too low will cause the weights to have a very light penalty, which will not mitigate jitter. However, a value that is too high will result in the tracker that is not able to keep up with objects which are moving quickly. Instead, the tracker will lag behind these objects and eventually lose them if the motion is continued for long enough.

3.1.4 Scale Adaptation

Another challenge for the tracker to overcome will be to handle changes in the scale of the object. Though the tracker already attempts to handle some amount of scaling based on the dynamic dictionaries, this is not a perfect approach. Instead it would be helpful to have the descriptor size change over time with the object. This allows for the tracker to fully handle scale, instead of just trying to mitigate some of the error introduced by it.

This thesis proposes a scale adaptation method that relies, like so much of the tracker, on matching the descriptors. Instead of examining specific points on the object and calculating their distances from each other to infer scale, the descriptor is matched at different scales. At every frame, the candidate with the lowest score is selected. At the winner location, new descriptors are created at the scale level above and the scale level below, the current descriptor. These are then compared to the static dictionaries and
should one be lower than the current winner, that scale is adopted for future frames. An example of this method can be seen in Figure 11.

The rationale behind this technique is simple. If the descriptor is describing the object as accurately as possible, it will give a result close to the original descriptor. If the object has changed pose in addition to scale, this method would not match the static very well. However, this was intentional as it attempts to limit the frames in which the scale could be adapted which should stop rapid and false changes. Describing only the winning location at the scale level above and below was chosen for its computational efficiency. Instead of describing the entire search grid, which can take a fair amount of time, description is required for only one point, which is almost unnoticeable. This method also grants freedom from requiring specific portions of the object to remain unchanged. With the descriptor, a base level of invariance is already achieved, so the scaling method should work even when an object is rotated or has undergone some other transformation.
3.1.5 Summary of Tracker Flow

In summary, the tracker can be broken into two distinct phases, the detection phase, and the description phase. Both of these phases are outlined below in Table 2 and Table 3:

### Detection Algorithm – Local Binary Descriptor Tracker

**Input:** Video Frame

1. Find desired object (face detection, classification, ground truth, etc.)
2. Resize descriptor to match object size
3. Describe object from center point
4. Add descriptor to appropriate dictionary element
5. Repeat until frame number exceeds dictionary size

<table>
<thead>
<tr>
<th>Table 2 Steps of detection phase of the tracker</th>
</tr>
</thead>
</table>

### Tracking Algorithm – Local Binary Descriptor Tracker

**Input:** Video Frame

1. Create all keypoints from search area
2. Describe all keypoints
3. Compute Hamming Distance of each described keypoint to each dictionary
   a. Keep track of minimum score
   b. Add constant to score of dynamic dictionaries to bias towards static dictionary
4. Compare winning descriptor against all dictionaries
   a. If it is above matching threshold return to step 1
5. Update the location of the object with the location of the winning candidate
6. Check upper and lower scales
   a. Change scale if needed
7. Display or output location, return to step 1

| Table 3 Steps of the tracking phase of the tracker |
3.2. Benchmarking System

The large number of descriptors and the new style of tracking have led to the necessity of having a system that can give information on how well it is working. Thus benchmarking software was created to report on all the areas of importance. The fields that will be examined include accuracy, distinctness, speed, and robustness. By examining a large sum of data, the best parameters for the tracking software can be determined, allowing for a robust algorithm that does not require any parameter tweaking to perform well on different videos. Without this, human intervention would be necessary to tune the tracker parameters, which is not an ideal situation should the tracker be used in the wild.

3.2.1 Accuracy

Accuracy is the most important of all the fields to test. A tracker could be incredibly fast, but without enough accuracy to actually maintain the location of the tracked object, the program would become an expensive way to display a video stream.

One of the easiest ways to measure the accuracy of a tracker is simply to have it output all of the locations of the tracked object. Then, the distance from the ground truth is plotted versus time and an informative graph detailing tracker accuracy is created. This will be the primary method of finding the best parameter set for the tracker. There are many different parameter sets that can be tested. Most likely, there will be a specific set for each video that will be tested, however having to tune the tracker for each video is not ideal. Thus, the average distance from the ground truth will be observed for each set. This will allow the parameter set that works best for a range of videos to be determined. The resulting set may not be the absolute best for any given video, but it will allow the tracker
to perform its function admirably without requiring a new setup for each video. The parameters that will be tested include: search radius, matching threshold, and $\sigma$ for the Gaussian weights.

### 3.2.2 Distinctness

This field is entirely for the descriptors as they are the entities that need to be unique in order for matching to occur. Unfortunately, this cannot be tested quite as easily as accuracy, since the ground truth does not carry any information about the descriptors. However, the Hamming Distance offers a basic measure of how distinct a specific descriptor is. Instead of computing the distances and then keeping track of the lowest score, all of the scores are output and then plotted into a 3D surface. This shows the distances between the various keypoints. An ideal pattern would be for the points closest to the center to have the lowest distance. As the spatial distance increases, so too should the Hamming Distance. A good way to rank the descriptors in this field is to examine the rate at which the distance between two points grows. If only a single bit is different from two neighboring keypoints, it would be very likely that these descriptors are not distinct enough. So, the higher the rate at which distances increase, the better the descriptor will be in terms of remaining unique amongst its peers.

This 3D surface plot will be created for a very large search radius that would normally not be used in tracking. This will help to show any weaknesses in the descriptors. It will also be run over a series of descriptor sizes. The expectation is that the larger the area that is sampled, the more unique is the information provided by a descriptor. So, by scaling the descriptor sizes, the breakdown point can be seen when a descriptor is no longer valid for tracking.
3.2.3 Speed

Speed is a very easy statistic to observe, as it is simply the rate at which the tracker processes a video stream in terms of frames per second (FPS). In order to get meaningful results, the tracker will be run without showing a video on the display. This allows for the algorithm itself to be the object under test instead of the rate at which the computer can display an image, which is often a limiting factor.

In order to test for speed, a parameter set should first be found that allows for a high level of tracking. This establishes a baseline where the tracker is functioning at maximum accuracy, but not necessarily speed. This in itself will be compared between the different descriptors. Next, the search radius will be modified (both larger and smaller than the radius for maximum accuracy) in order to see its effects on how the tracker speeds up and slows down. By examining the rate at which a descriptor speeds up and slows down, a lot of information can be gathered about its complexity. For instance, the rate at which two different descriptors increase could indicate a possible crossing of timings in for high enough cases based on how each descriptor handle large numbers of keypoints. Watching the rate at which the descriptors are generated as the number of keypoints wanes could help show the difference between setup time and description time. This would help identify the point where it is no longer worth trimming keypoints.

Another metric that should be examined is the effect on speed that descriptors of different sizes have. In theory, there should be no difference in speed for a large binary descriptor and a small binary descriptor, as they are both simply making a series of point comparisons. However, this should still be tested as it will show how long it takes to generate one descriptor, and to ensure that they are working as theorized.
3.2.4 Robustness

Robustness is one of the hardest metrics to measure as it has a fairly subjective meaning. However, the main goal of this metric is to determine exactly how far an object can be distorted before the descriptor is not a match anymore. This can be done with a series of images that have been designed and edited to test the robustness of the descriptors. Editing allows minor changes to be made to the image that correspond to rotations, occlusions, and other such variations in appearance.

This can all be done while processing a video stream as long as the base object can be found initially either through ground truth or detection. Once the object is found, it is cropped down so that only a sub-image of the object is considered. This sub-image undergoes a series of transformations to test the robustness of the descriptor. OpenCV contains the functionality required to rotate an image a specific angle which allows for the rotation tests to be completed easily. It can also resize an image above and below its current size. The distances of the descriptor for these transformations will be recorded and monitored to determine at which point the object is no longer a good match.

In addition to these tests specifically designed for robustness, a series of test videos will be used in order to glean information on the overall ability of the tracker. These videos can include challenges such as occlusion, change in scale, rotation, and poor image quality. For these reasons the standard test videos will be used and the tracker’s accuracy on them will indicate how well a specific descriptor performed.
Chapter 4  Experimental Results

As with any system, the tracker has several parameters that can be used to fine-tune its performance. This occurs because of the large amount of variability in the videos that were used for testing which were the same as used in MILTracker [6]. Each video has its own unique set of challenges ranging from different sizes, different resolutions, and different rates of motion. Each of these variables can change drastically the way in which an object needs to be tracked. In this section, the effects of changing parameters will be examined on the standard test videos. From these parameter sweeps, the best results for the tracker will be found and discussed.

Finally, a parameter set will be determined that works well for the entire range of test videos. This is important because a system that must be fine-tuned for every video is not efficient or convenient for the user. Thus, a final set of parameters will be used to find and compare results against MILTrack and TLD. These results will provide a more telling example of how the tracker works as a generic entity.

4.1. Effects of Scaling

One of the more difficult portions of tracking is handling forward and backward motion, because when an object approaches or moves away from the capture source, the object’s size changes. Descriptors have a level of scale-invariance; however, the more original the object appears to the descriptor, the closer the description should be to the original. The challenge with changing the scale of an object while tracking is that fixed point comparisons are often computationally complex, or may not be robust to changes in the object’s shape. Conversely, a loose definition of scale may cause the scale to change
rapidly, when this is not the ideal case. Scale adaptation as shown in Section 3.1.4 has provided a minor improvement in some videos where scaling is prominent. Figure 12 illustrates the effect of scale adaption on the tracker running on the video ‘cliffbar’. This video already worked very well for the tracker, but scale adaptation was able to improve the accuracy. It is clear that the accuracy is not always better, but it did bring the average distance down ~8% for this video (3.95 to 3.65). Overall, it has slightly lower and fewer distance spikes, showing an improved ability to maintain the location of the object.

![Effect of Scaling on Cliffbar](image)

**Figure 12 Comparison of tracker running with and without scale adaptation on Cliffbar**

Figure 13 is the accuracy comparison with and without scale adaptation on the video Twinings. Unlike the results for Cliffbar, the tracker does not perform as well for Twinings, maintaining an average distance of 13.67. Turning on the scale adaption feature reduced the error by 10% (similar to the reduction seen by Cliffbar). It is
important to note that despite the improvement, there is still a period of time where the error distance is higher 20 pixels. However, when scale adaptation is used, the number of frames with tracking error above 20 is reduced significantly. It also shows a general ability to mitigate the spikes in distance throughout the entire video.

Effect of Scaling on Twinings

![Graph showing the effect of scaling on twinings](Image)

Figure 13 Comparison of tracker running with and without scale adaptation on Tiger1

The effect of scaling on speed is something that is important to consider. No matter the benefits in terms of accuracy, if a method causes the tracker to become slow to a fault, it is not worth pursuing. It is for this reason that the scaling was chosen to be completed only on the one winning candidate, and not across an entire search grid. This allows for the additionally computation of only 2 descriptors, which is much better than a few hundred to several thousand extra computations. For some of the more basic descriptors, a single point will take only a few microseconds to generate. However, more
complex descriptors such as BRISK and FREAK will take approximately a millisecond to generate their single descriptor.

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>BRIEF</th>
<th>FREAK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Timing</td>
<td></td>
</tr>
<tr>
<td>Radius 10</td>
<td>Frame Time (ms)</td>
<td>Description Time (ms)</td>
</tr>
<tr>
<td>Radius 20</td>
<td>2.50</td>
<td>6.76</td>
</tr>
<tr>
<td>Radius 30</td>
<td>8.04</td>
<td>16.39</td>
</tr>
</tbody>
</table>

Table 4 shows a numeric breakdown of the tracker’s speed throughout several different search radii. It is clear that BRIEF barely notices the time added by the using scale adaptation. Conversely, FREAK takes a large penalty for scaling, but only at the lower end of the radii scale. At a radius of 10, FREAK’s scale adaptation takes twice as long as the actual description of the entire search grid. This main seem counterintuitive, however, this is because scaling requires two separate instances of FREAK to describe a single point. When the entire search grid is being described, there is heavy optimization so multiple points (or bytes) can be calculated simultaneously. The orientation phase is the portion that takes the longest. Due to this, it is expected that describing a single point will appear to take a majority of the total frame time. Looking at later results, it becomes clear that FREAK does end up using its optimizations well, as it eventually becomes faster than BRIEF despite being more complex.

4.2. Accuracy

A tracker without accuracy does not serve any purpose, which is why this field is such a critical one to analyze. Within the system, there are several parameters that play a key role in tracking efficiency. The primary parameter is the type of descriptor utilized to
describe the image regions. This one parameter defines how distinct varying points are from one another, and how robust the tracker is to disturbances. The other parameters that were adjusted to find the best possible accuracy were the matching threshold, the search radius, and the sigma scale. By running through a large sweep of all of these parameters, the best set could be found for each video, showing that the proposed system can track an object very well.

It should be noted that, despite SIFT’s ability to perform well in many different applications, it was not suited for this system. Thus, to make the results appear less cluttered, it has not been included in the accuracy result tables or figures. This also goes for CAMSHIFT which did not perform well enough to warrant comparison.

### 4.2.1 Parametric Sweep

As per any system, a series of variables define the inner workings of the tracker. For this system, the variables were the search radius, $\sigma^2$ for the Gaussian locality constraints, and a threshold for dictionary entry. These variables control how far away from the center point to search, how free the motion is, and how high of a distance is still deemed a match.

Each video and descriptor combination most likely has a unique combination of parameters. Each descriptor has its own characteristics which make them tend towards a particular parameter set. However, the videos themselves introduce a lot of variability into the system. One video may have a poorly defined (blurry) object, while another may have a very crisp image. Some objects move very quickly, and some move very little. Thus, a balance must be struck between search radius, locality constraints, and matching thresholds. By running the tracker with large parameter sweeps on many different
standard videos, the validity of the system can be immediately proven by showing the system works on a variety of different videos.

Figure 14 is an example of a video where the system has performed very well. This particular video involves tracking a Cliffbar as it is moved from side to side, front to back, and rotated. The object is very well defined in this video with few instances of it becoming blurred. The definition of this object makes for easy tracking when a descriptor is used as it can be a very distinct entity. Though every descriptor has a lower average distance than MIL (as shown in Table 5), it is clear that some descriptors have done better than others. ORB clearly has a point when tracking has gone far from the object. A reason for this occurring could be ORB’s attempt to make it invariant to change. With the amount of mitigation that is done by the tracking system, this can actually lead to a disadvantage if done improperly. BRIEF, which is a simpler (non-rotational) form of ORB, actually does better on this video and several others throughout the testing process. As BRIEF is the simplest descriptor, this result shows that the tracker is doing enough processing to handle basic changes to a point where being an advanced descriptor is not necessarily an advantage.
Continuing on the trend of videos where every descriptor is tracking well, Figure 15 shows the tracking error over time on the video Dollar. For this video it is clear that all of the descriptors have been able to accurately track the object without any major issues. There are no spikes where the error distance becomes large, and throughout the duration of the video the tracker’s error is less than that of MILTrack. In this video, a coupon book (or dollar bill) is shown and initialized as the first frame. From this point onward the dollar is folded in half, and then an exact replica is shown next to it. The purpose of the video is to continue to track the folded dollar despite the perfect one next to it. This is playing at the tracker’s weakness to return to the static dictionary (original object). However, thanks to the dynamic dictionaries and the locality constraints, the tracker is able to track the object almost perfectly.
Turning from videos that track near perfectly to ones that present challenges, Figure 16 shows the ground truth comparison for Tiger1, a video that the tracker was not able to track very well. In this video, a toy tiger is moved around the frame at a fairly rapid rate both in front of and behind a plant’s leaves. The level of occlusion and rotation are both very high for this video. Interestingly the descriptor that performs best for this video is BRIEF, the simplest of the descriptors. The reason for this may be the fact that the tracking system is already attempting to mitigate all of the changes and errors that may occur while tracking. When the descriptor attempts to do this itself, it may actually be lowering its distinctness, or may over compensate and differ from the dictionaries.
Table 5 is a comprehensive set of tracking error results. Each row is a standard test video and each column a method of tracking with MIL being the baseline. The individual values are simply the average distance from the ground truth (calculated as the mean square error). This metric for testing accuracy was shown in [6] and has been used here to give a reference of the proposed system to MILTrack. From this data it is clear that the proposed system has the ability to perform better on all of the standard test videos. Though different descriptors have received the lowest error for different videos, single descriptors have still managed to win on almost all the videos. FREAK, for example, has a lower accuracy in all but one video (Tiger1). FREAK has shown the best overall results with the lowest average error (9.94 pixels). BRISK was a close second at 9.99 pixels while ORB and BRIEF were last with 12.23 and 11.61 pixels respectively.
Table 5 Results of average distance from ground truth using best parameter set of proposed tracker with different descriptors in use as compared to results of MILTrack

Table 6 is based on another performance metric seen in [6]. Instead of reporting the average error throughout the entire video sequence, the percentage of frames that are within 20 pixels of the ground truth is reported. This metric helps highlight videos that were consistent and ones that had large spikes in tracking error. The main fact to take from this table is that the proposed tracker has several videos where, throughout the duration of the video, the tracker is under 20 pixels from the ground truth. This table also shows an excellent comparison of the descriptors in terms of stability. FREAK, which is the most advanced of the descriptors, has the most videos at 100%. Following FREAK is BRISK, BRIEF, and ORB. This is counter-intuitive as ORB was created as an improvement over BRIEF. The creators of ORB do state in [22] that the rotational component of BRIEF did remove some of its uniqueness. This would definitely have a negative effect on the performance of the tracker for some cases, which could be why it has the least stable tracking pattern of any of the descriptors used.
<table>
<thead>
<tr>
<th></th>
<th>Video</th>
<th>MIL</th>
<th>TLD</th>
<th>BRIEF</th>
<th>ORB</th>
<th>BRISK</th>
<th>FREAK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cliffbar</td>
<td>88%</td>
<td>61%</td>
<td>94%</td>
<td>94%</td>
<td>98%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Coke Can</td>
<td>55%</td>
<td>68%</td>
<td>21%</td>
<td>72%</td>
<td>66%</td>
<td>69%</td>
</tr>
<tr>
<td></td>
<td>David</td>
<td>52%</td>
<td>91%</td>
<td>100%</td>
<td>84%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Dollar</td>
<td>69%</td>
<td>42%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Occluded Face</td>
<td>43%</td>
<td>34%</td>
<td>86%</td>
<td>91%</td>
<td>93%</td>
<td>99%</td>
</tr>
<tr>
<td></td>
<td>Occluded Face</td>
<td>60%</td>
<td>60%</td>
<td>99%</td>
<td>76%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Girl</td>
<td>31%</td>
<td>70%</td>
<td>41%</td>
<td>70%</td>
<td>49%</td>
<td>82%</td>
</tr>
<tr>
<td></td>
<td>Surfer</td>
<td>93%</td>
<td>100%</td>
<td>88%</td>
<td>99%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td></td>
<td>Sylvester</td>
<td>90%</td>
<td>94%</td>
<td>100%</td>
<td>98%</td>
<td>100%</td>
<td>97%</td>
</tr>
<tr>
<td></td>
<td>Tiger 1</td>
<td>81%</td>
<td>89%</td>
<td>93%</td>
<td>56%</td>
<td>74%</td>
<td>81%</td>
</tr>
<tr>
<td></td>
<td>Tiger 2</td>
<td>83%</td>
<td>70%</td>
<td>89%</td>
<td>76%</td>
<td>95%</td>
<td>74%</td>
</tr>
<tr>
<td></td>
<td>Twinings</td>
<td>78%</td>
<td>37%</td>
<td>74%</td>
<td>97%</td>
<td>88%</td>
<td>75%</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>69%</td>
<td>68%</td>
<td>82%</td>
<td>84%</td>
<td>89%</td>
<td>90%</td>
</tr>
</tbody>
</table>

Table 6 Percentage of frames closer than 20 pixels to ground truth

Figure 17 shows a series of frames from different test videos with the tracker’s location drawn on each frame. What these show is the ability for the tracker to maintain the location of an object over the entire video stream. It is evident that all of the different descriptors perform fairly consistently with respect to each other. Generally, FREAK is
closest to the actual ground truth, although for the videos shown, no descriptor does poorly. The topmost frames are from the Cliffbar video. They show the tracker’s excellent ability to handle variations as even BRIEF, the simplest of the descriptors considered, can continue to track when the object is rotated and scaled. David and Dollar show similar traits for the tracker while Surfer shows that the tracker can work on a poorly defined image. This is actually the most impressive video for the tracker to handle because descriptors function much better when given a higher resolution, larger object to describe. More insight on this is given in Section 4.3.

<table>
<thead>
<tr>
<th>Video</th>
<th>BRIEF</th>
<th>ORB</th>
<th>BRISK</th>
<th>FREAK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R</td>
<td>T</td>
<td>S</td>
<td>R</td>
</tr>
<tr>
<td>Cliffbar</td>
<td>22</td>
<td>60</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>Coke Can</td>
<td>12</td>
<td>50</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>David</td>
<td>22</td>
<td>50</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>Dollar</td>
<td>24</td>
<td>513</td>
<td>5</td>
<td>26</td>
</tr>
<tr>
<td>Occluded Face 1</td>
<td>28</td>
<td>60</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>Occluded Face 2</td>
<td>22</td>
<td>513</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>Girl</td>
<td>10</td>
<td>50</td>
<td>8</td>
<td>22</td>
</tr>
<tr>
<td>Surfer</td>
<td>28</td>
<td>90</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>Sylvester</td>
<td>28</td>
<td>90</td>
<td>3</td>
<td>28</td>
</tr>
<tr>
<td>Tiger 1</td>
<td>26</td>
<td>80</td>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>Tiger 2</td>
<td>28</td>
<td>80</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Twinings</td>
<td>22</td>
<td>100</td>
<td>3</td>
<td>30</td>
</tr>
</tbody>
</table>

Table 7 The highest accuracy for each of the descriptor video pairings for parameters Radius (R), Threshold (T), and Sigma (S).

Table 7 is the collection of parameters that were used to obtain the results shown. The parameters shown are Radius (R), Threshold (T), and Sigma (S). They represent how far away from the center point the tracker can search, the level of error it accepts before it declares a loss, and the freedom it has to move within its search radius. Interestingly enough, these parameters have shown some of the characteristics of the descriptors. The radii are generally fairly similar across all of the descriptors. However, the thresholds
have shown which descriptors are generally better at having consistent descriptions. When the description isn’t consistent, the threshold must be higher in order to account for the difference between the dictionary and the candidates. When it is lower, it means that the candidates have become consistent and generally match the dictionary. The sigma is also a defining characteristic for these descriptors and what must be noted about it is that only one descriptor (FREAK) does not get to the high sigma levels. This means that FREAK is generally distinct enough so that it does not need to be restricted in its motion.

### 4.2.2 Unified Parameter Set

Tracking for a very specific parameter set is possible for most tracking algorithms. However, tracking under a single, unified, parameter set is much more difficult. By using a unified parameter set, the tracker is attempting to handle all the variability that can occur in the different videos. There will be different rates of movement, different levels of definition, occlusions, and rotations, all of which must be taken into consideration.

Examining Table 8 shows the effects of using a unified parameter set for each descriptor. Each descriptor does receive their own parameter set as they work too differently to function with similar parameters. This shows a distinctly different set of errors from the best parameter set. It is clear that each descriptor works best for a few specific videos, and the remaining videos have gone much worse than before. This is an excellent representation however, of the benefit of a more complex descriptor. BRIEF, which performed admirably in the best parameter set cases, has now become the worst descriptor by a fair margin. This emphasizes that a more advanced descriptor is able to have a more general parameter set. BRIEF still performs the best on the Sylvester video;
however it does not anywhere else, even videos where it had the lowest average distance previously. FREAK, however, still performs best overall. It may not have had the most videos with the lowest error, but it did have the lowest overall error, which means it functions better as a generic tracking system.

<table>
<thead>
<tr>
<th>Video</th>
<th>MIL</th>
<th>TLD</th>
<th>BRIEF</th>
<th>ORB</th>
<th>BRISK</th>
<th>FREAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cliffbar</td>
<td>11</td>
<td>34.62</td>
<td>19.50</td>
<td>24.79</td>
<td>17.74</td>
<td>4.016</td>
</tr>
<tr>
<td>Coke Can</td>
<td>20</td>
<td><strong>16.74</strong></td>
<td>31.67</td>
<td>23.33</td>
<td>30.17</td>
<td>33.64</td>
</tr>
<tr>
<td>David</td>
<td>23</td>
<td>11.27</td>
<td>7.417</td>
<td>19.64</td>
<td><strong>4.217</strong></td>
<td>11.40</td>
</tr>
<tr>
<td>Dollar</td>
<td>15</td>
<td>68.06</td>
<td>7.384</td>
<td><strong>3.350</strong></td>
<td>4.437</td>
<td>7.034</td>
</tr>
<tr>
<td>Occluded Face</td>
<td>27</td>
<td>23.47</td>
<td>12.72</td>
<td>20.72</td>
<td>9.441</td>
<td><strong>7.751</strong></td>
</tr>
<tr>
<td>Occluded Face</td>
<td>20</td>
<td>19.91</td>
<td>19.69</td>
<td>22.41</td>
<td><strong>10.59</strong></td>
<td>12.08</td>
</tr>
<tr>
<td>Girl</td>
<td>32</td>
<td><strong>19.77</strong></td>
<td>103.8</td>
<td>22.78</td>
<td>50.26</td>
<td>25.33</td>
</tr>
<tr>
<td>Surfer</td>
<td>11</td>
<td><strong>9.947</strong></td>
<td>27.20</td>
<td>18.63</td>
<td>22.11</td>
<td>11.54</td>
</tr>
<tr>
<td>Sylvester</td>
<td>11</td>
<td>8.559</td>
<td><strong>5.669</strong></td>
<td>35.10</td>
<td>40.63</td>
<td>18.21</td>
</tr>
<tr>
<td>Tiger 1</td>
<td>16</td>
<td><strong>11.07</strong></td>
<td>25.60</td>
<td>32.77</td>
<td>25.63</td>
<td>33.03</td>
</tr>
<tr>
<td>Tiger 2</td>
<td>18</td>
<td>20.68</td>
<td>30.73</td>
<td>31.93</td>
<td>30.53</td>
<td>33.44</td>
</tr>
<tr>
<td>Twinings</td>
<td>15</td>
<td>61.12</td>
<td>19.19</td>
<td>21.06</td>
<td><strong>11.52</strong></td>
<td>13.28</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>18.25</td>
<td>25.43</td>
<td>25.88</td>
<td>23.04</td>
<td>21.44</td>
<td>17.56</td>
</tr>
</tbody>
</table>

Table 8 Results of average distance from ground truth using unified parameter set

Table 9 shows the stability of the tracker as seen in Table 6 for a unified parameter set. These results show that the tracker loses a significant amount of stability when using a unified parameter set. Each descriptor still has specific videos that it tracks very well, but the performance for other videos can suffer so dramatically that the average performance is pulled down. This is very clear when examining FREAK’s results. Despite having two videos that track very well, Coke Can and Tiger 1 have both suffered tremendously and have drifted more than 20 pixels off of the object for more than 80% of the video. So, despite tracking well for the majority of videos, these two videos cause a dramatic decrease in performance.

Contrary to the results in Table 8, BRIEF has performed remarkably well for this metric considering it has the highest average distance. Instead of tracking as close as
possible and falling off, BRIEF appears to have simply tracked from a farther average distance, but was not too far away from the actual object.

Comparing the proposed tracker to MIL and TLD in this case shows that the best case is a tie with TLD in average accuracy. This again goes back to the fact that there are a few videos where the proposed tracker does not do well. FREAK does have the highest number of videos at 100% of the frames being under a distance of 20; however its performance for the Coke Can video is 12%, which is much lower than MIL or TLD. Thus, despite the fact that FREAK has a lower average distance than both MIL and TLD, it does have the tendency to track poorly on some videos when using a unified parameter set. This tendency can decrease the percentage of frames under a distance of 20 from the ground truth.

<table>
<thead>
<tr>
<th>Video</th>
<th>MIL</th>
<th>TLD</th>
<th>BRIEF</th>
<th>ORB</th>
<th>BRISK</th>
<th>FREAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cliffbar</td>
<td>88%</td>
<td>61%</td>
<td>58%</td>
<td>58%</td>
<td>66%</td>
<td>100%</td>
</tr>
<tr>
<td>Coke Can</td>
<td>55%</td>
<td>68%</td>
<td>28%</td>
<td>38%</td>
<td>28%</td>
<td>12%</td>
</tr>
<tr>
<td>David</td>
<td>52%</td>
<td>91%</td>
<td>95%</td>
<td>57%</td>
<td>100%</td>
<td>97%</td>
</tr>
<tr>
<td>Dollar</td>
<td>69%</td>
<td>42%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Occluded Face</td>
<td>43%</td>
<td>34%</td>
<td>79%</td>
<td>49%</td>
<td>87%</td>
<td>97%</td>
</tr>
<tr>
<td>Occluded Face</td>
<td>60%</td>
<td>60%</td>
<td>53%</td>
<td>42%</td>
<td>89%</td>
<td>77%</td>
</tr>
<tr>
<td>Girl</td>
<td>31%</td>
<td>70%</td>
<td>19%</td>
<td>48%</td>
<td>12%</td>
<td>61%</td>
</tr>
<tr>
<td>Surfer</td>
<td>93%</td>
<td>100%</td>
<td>55%</td>
<td>81%</td>
<td>57%</td>
<td>81%</td>
</tr>
<tr>
<td>Sylvester</td>
<td>90%</td>
<td>94%</td>
<td>100%</td>
<td>34%</td>
<td>35%</td>
<td>79%</td>
</tr>
<tr>
<td>Tiger 1</td>
<td>81%</td>
<td>89%</td>
<td>56%</td>
<td>36%</td>
<td>41%</td>
<td>17%</td>
</tr>
<tr>
<td>Tiger 2</td>
<td>83%</td>
<td>70%</td>
<td>47%</td>
<td>47%</td>
<td>38%</td>
<td>35%</td>
</tr>
<tr>
<td>Twinings</td>
<td>78%</td>
<td>37%</td>
<td>59%</td>
<td>44%</td>
<td>74%</td>
<td>58%</td>
</tr>
<tr>
<td>Average</td>
<td>69%</td>
<td>68%</td>
<td>62%</td>
<td>53%</td>
<td>61%</td>
<td>68%</td>
</tr>
</tbody>
</table>

Table 9 Percentage of frames closer than 20 pixels to ground truth using unified parameter set

Table 10 represents the average parameter sets that have been found for each of the descriptors. The average parameter sets were found for each descriptor instead of the system as a whole, because each one has too many of its own characteristics to work well with another’s parameters. With these parameters the accuracy for each one has fallen,
however it is important to note that FREAK still performed fairly well. This means that
generally, FREAK is a suitable descriptor for a generic application. It should even be able
to run at a fairly high rate of speed, which will be discussed in Section 4.4.

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>General Parameter Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>BRIEF</td>
<td>Radius: 22, Threshold: 60, Sigma: 1</td>
</tr>
<tr>
<td>ORB</td>
<td>Radius: 30, Threshold: 70, Sigma: 4</td>
</tr>
<tr>
<td>BRISK</td>
<td>Radius: 30, Threshold: 90, Sigma: 5</td>
</tr>
<tr>
<td>FREAK</td>
<td>Radius: 26, Threshold: 100, Sigma: 5</td>
</tr>
</tbody>
</table>

Table 10 General Parameter Set for each descriptor

4.3. **Distinctness**

Distinctness of a descriptor is a key factor in the performance of the tracking system. A
larger distinctness allows for a larger search radius, which in turn allows for a higher
speed object to be tracked. Thus, each of the descriptors was evaluated by creating an
extremely large search radius and then mapping out all of their Hamming Distances over
the search region. The ideal surface would appear similar to a Gaussian distribution
where the lowest distances were at the center of the search area. They should also only
have one prominent null, as multiple low distance areas could lead to a false-positive
match. It should also be noted that the error surfaces were all generated without the aid of
the Locality Constraints. This was done so that the descriptors themselves were
evaluated, not the rest of the tracker. Figure 18 shows all of the images used for the
distinctness testing. From left to right they represent a large, well-defined object, a small,
well-defined object, and a small, poorly defined object.
Figure 18 Objects used for distinctness tests. From left to right is the Large Well-Defined Object, Small Well-Defined Object, and Small Poorly Defined Object

Figure 19 SIFT error surface for different size objects

Figure 19 shows the error plots for SIFT. These error plots are unique in the fact that they are for the Euclidean Distance as opposed to the Hamming Distance. This does mean that the actual distances will be higher for SIFT, so it should be considered when examining these surfaces. Aside from this fact though, it is clear that SIFT is not designed for this style of application. Instead of having a single minimum in the error distance surface, it has many different points that can reach a low distance error. This characteristic, coupled with the computational complexity of SIFT, does not bode well for SIFT in this tracking framework. It is possible that this is due to the nature of SIFT itself. Currently, the tracker is simply telling it to describe a dense grid of points, all of which are the same size. Normally SIFT would generate its own points to describe, each with its own size and location, and in a sparse pattern. By ignoring this and simply describing the dense, uniformly sized region, it is possible that SIFT is forced to operating under non-ideal
conditions. This shows that without any extra considerations, SIFT is not a good fit for the system. When considering the other descriptors, it is clear that the effort is not worth the reward in pursing SIFT for this system.

Figure 20 BRIEF error surface for different size objects

Figure 20 shows BRIEF’s error surfaces. As the first of the binary descriptors, and the simplest one used, it provides promising results. For the large, well-defined object (surface on the far left), it is clear that BRIEF has identified the object well. There is a distinct drop in distance at the middle of the surface and then a rise to some high distance peaks as the search grid is traversed. This is still true for smaller objects, as shown in the middle surface, though the surface appears less smooth. However, this is to be expected as a smaller object will have the search area move away from it faster than a large object would. Thus, instead of gradually falling off the object, the error surface generated is an abrupt one. Unfortunately, when the object is small and poorly defined, as shown in the far right surface, the distances become extremely erratic. This error surface was developed by examining the coke can video, which is a very small object, and one that is not tracked very well. With this error plot, it is easy to see why. Despite there being areas with a small distance, there are so many, that are so close together, that it would be easy for the tracker to become confused. Adding into this equation all of the other difficulties
of tracking on coke can (rotations, occlusions, lighting) it becomes clear that a video such as this is not ideal for a BRIEF-based tracker.

As pointed out in Section 2.1.3, ORB is simply a rotated version of BRIEF. Thus, it still uses a randomized pattern to generate its comparison points. Figure 21 shows the error surfaces for ORB. It should be noted that they are extremely similar to the ones generated by BRIEF. This makes sense, because the search pattern is very similar. However, it is noted in [22] that the rotation of BRIEF has removed some of its uniqueness. This can be seen with these error maps as the lowest error no longer reaches quite as far down as it did with BRIEF, though there is still clearly a spike of low distance. It does maintain a valid surface for the large and small well-defined objects and it suffers a poor surface for the ill-defined object. So, it is very similar to BRIEF in almost every regard.

BRISK is the first of the descriptors that uses a predefined pattern. Its error surfaces can be seen in Figure 22. The error surfaces for BRISK are very similar to the
surfaces for BRIEF and ORB. This validates binary descriptors as a whole because they all get fairly good results. There are some differences between BRISK and the other descriptors. Mainly, its error surfaces appear slightly more uniform than those of the random pattern based BRIEF and ORB. Instead of one main dip, and several smaller dips or even regions of lower general distance, BRISK clearly has one spike downward and the rest of the distances are much higher.

One interesting item to note is that overall, the error surfaces for BRISK are much smoother than those of BRIEF and ORB. This could be due to the specific pattern that BRISK uses for all of its comparisons. This would cause regions of repeat in an object. The descriptor itself remains unique because it is very unlikely that all 512-bits will align, but having a predefined set of circles around the object can lead to gathering regions of information. Instead describing randomly across the region, locations (eyes or nose on a face for example) will be continuously sampled. The small poorly defined object (far right surface) is a prime example of this. In BRIEF and ORB, it appeared extremely sporadic. However, with BRISK, there are very clear ridges of information throughout the surface. This could be a reason why BRISK performed fairly well on Coke Can, however ORB did marginally better, so this may not be true for the entire video sequence.

![Figure 23 FREAK error surface for different size objects](image_url)
FREAK is a very unique case in terms of the error surfaces. One of FREAK’s unique points is the ability to handle coarse-to-fine matching. Instead of matching the entire 64-Byte descriptor, if a certain distance was met before a cut-off point (16-Bytes) that descriptor would be thrown out. This is evident in Figure 23. As in the previous error surfaces, the lowest point is in the center of the search grid. However, instead of a smooth rise towards the higher distances at the outer edges of the search grid, there is a very sharp rise as the descriptors are thrown out. This does not necessarily improve accuracy, as poor results would result in a high distance regardless. It does effectively show the number of descriptors that FREAK throws out.

These results have shown that the average descriptor manages to remain fairly unique over the entire large search radius. This is a very good indicator as the radius that was utilized to perform these tests was above any of the radii used for accuracy or speed testing. This is primarily due to the fact that a radius of 30 was found to be the general cap for 60 FPS. As the tracker is designed for real-time usage, pursuing larger radii was not deemed necessary, and as these results show, a much larger radius was not going to be possible and could have hindered the accuracy of the tracker.

4.4. Speed

When designing a tracking system, real-time is often a lofty goal that may not always be met. For instance, examining the results of [6], [7] and [8], these state of the art trackers are generally limited to FPS rates around 20. MILTrack claims speeds of up to 25 FPS, however while running TLD many videos were shown in mid-teens for FPS. CAMSHIFT came in much slower than the other methods averaging approximately 8 FPS during testing. However, there is also not too dire a need for a tracker, running on
the CPU, to become faster than 30 FPS as this is the limit in terms of speed for many consumer grade webcams.

In the pursuit of speed, the proposed system has utilized descriptors in an attempt to harness their efficiency in both the creation and matching process. By using a model-free system, the speeds that can be achieved become quite high. Within the tracker subsystem, there are several points that determine the actual speed of the tracker. These sections are the rate at which an object can be described, the rate at which it is scored, and the rate at which it can perform scale adaptation. All of these times are integral in the overall speed of the system, and each one can vary wildly based on the descriptor that is used.

Table 11 demonstrates the tracker’s speed while using the SIFT descriptor. Though SIFT has had a lot of success in the field of descriptors, it is not the best suited descriptor for this particular system. Even from the smallest of radii, the tracker is running at an extraordinarily slow speed of 1.1 FPS. At this rate, a standard video would take 25 times as long as normal which is not an issue if time is readily available. However, bringing this onto a webcam would lead to an extremely poor tracking implementation. In one second, a person could move clear across the entire frame, thus the tracker will have lost the object.
Radius | Frame Avg | Description Avg | Scoring Avg | Scale Avg | FPS  
---|---|---|---|---|---  
10 | 911.99 | 903.40 | 0.72 | 7.69 | 1.10  
12 | 1284.10 | 1275.23 | 1.02 | 7.70 | 0.78  
14 | 1723.06 | 1713.73 | 1.44 | 7.75 | 0.58  
16 | 2214.23 | 2204.74 | 1.79 | 7.52 | 0.45  
18 | 2745.54 | 2735.57 | 2.32 | 7.48 | 0.36  
20 | 3279.86 | 3269.41 | 2.78 | 7.52 | 0.30  
22 | 3917.35 | 3906.31 | 3.34 | 7.55 | 0.26  
24 | 4637.72 | 4625.80 | 4.00 | 7.78 | 0.22  
26 | 5436.38 | 5423.94 | 4.80 | 7.67 | 0.18  
28 | 6274.91 | 6261.20 | 5.31 | 8.27 | 0.16  
30 | 7293.07 | 7278.68 | 6.10 | 8.16 | 0.14  

Table 11: Time in milliseconds of SIFT Descriptor based tracking

However, the scaling and scoring play an extremely small role in the overall frame time, especially at larger radii. The scoring is expected to be insignificant compared to the descriptor as it is simply an L2 Norm in the case of SIFT, however the scale adaptation could have posed a fairly harsh penalty. Thankfully, as it is based solely on the one winning candidate, its time does not increase when the radius of the search grid does. This helps to alleviate the issue of consuming too much time for scale’s sake, and it also confirms that performing scale adaptation on two entirely separate search grids would simply take too long. Overall, although SIFT may work excellently for other applications, it does not work well for a tracker based on a dense search grid.

Table 12 shows the average times for the main components of the tracker when it is using the BRIEF descriptor. This descriptor has a lot working for it in terms of speed. Like the remaining descriptors it is a binary descriptor. However, unlike the remaining descriptors, it does not make any attempts at being rotationally invariant. Now, this may result in a descriptor with lowered accuracy, but it also has a less complex creation. This is extremely beneficial for speed, and it has been shown that the tracker itself does a fair
amount to mitigate small amounts of rotation so the lack of invariance is not a terrible thing.

<table>
<thead>
<tr>
<th>Radius</th>
<th>Frame Avg</th>
<th>Description Avg</th>
<th>Scoring Avg</th>
<th>Scale Avg</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2.50</td>
<td>1.63</td>
<td>0.40</td>
<td>0.01</td>
<td>400.00</td>
</tr>
<tr>
<td>12</td>
<td>3.35</td>
<td>2.29</td>
<td>0.55</td>
<td>0.02</td>
<td>298.51</td>
</tr>
<tr>
<td>14</td>
<td>4.31</td>
<td>3.05</td>
<td>0.74</td>
<td>0.00</td>
<td>232.02</td>
</tr>
<tr>
<td>16</td>
<td>5.41</td>
<td>3.95</td>
<td>0.92</td>
<td>0.01</td>
<td>184.84</td>
</tr>
<tr>
<td>18</td>
<td>6.62</td>
<td>4.99</td>
<td>1.09</td>
<td>0.02</td>
<td>151.06</td>
</tr>
<tr>
<td>20</td>
<td>8.04</td>
<td>6.13</td>
<td>1.35</td>
<td>0.01</td>
<td>124.38</td>
</tr>
<tr>
<td>22</td>
<td>9.47</td>
<td>7.40</td>
<td>1.55</td>
<td>0.01</td>
<td>105.60</td>
</tr>
<tr>
<td>24</td>
<td>11.17</td>
<td>8.72</td>
<td>1.89</td>
<td>0.01</td>
<td>89.53</td>
</tr>
<tr>
<td>26</td>
<td>13.01</td>
<td>10.24</td>
<td>2.19</td>
<td>0.02</td>
<td>76.86</td>
</tr>
<tr>
<td>28</td>
<td>15.17</td>
<td>12.02</td>
<td>2.57</td>
<td>0.02</td>
<td>65.92</td>
</tr>
<tr>
<td>30</td>
<td>17.02</td>
<td>13.55</td>
<td>2.86</td>
<td>0.02</td>
<td>58.75</td>
</tr>
</tbody>
</table>

Table 12 Time in milliseconds of BRIEF Descriptor based tracking

Even at its slowest point, BRIEF maintains a FPS of well over 30 FPS (nearing 60). At a radius of 10, BRIEF is a massive 363 times faster than SIFT. Speedup increases even further at a radius of 30 where BRIEF becomes 420 times faster than SIFT. Not only does this show that it is an immensely fast descriptor, but also that it scales better with a larger radius. Breaking down the individual components it is clear that this simplistic descriptor’s scale adaptation is extremely efficient. Without any orientation concerns, the scale adaptation can be seen simply as a series of 512 comparisons (length of two descriptors) and that is it. There is no moment to generate, no gradient to find, no extra calculations that are often more costly than the actual description. In terms of the effect on the FPS of the system, this only adds 10 microseconds to the overall frame, which is a lesser impact than that of reading the video frame itself. This is also the first instance of the Hamming Distance being used for matching. Though not drastically faster than the L2 Norm, it does save several milliseconds for the higher radii.


<table>
<thead>
<tr>
<th>Radius</th>
<th>Frame Avg</th>
<th>Description Avg</th>
<th>Scoring Avg</th>
<th>Scale Avg</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>9.52</td>
<td>8.74</td>
<td>0.38</td>
<td>0.06</td>
<td>105.04</td>
</tr>
<tr>
<td>12</td>
<td>13.28</td>
<td>12.52</td>
<td>0.52</td>
<td>0.06</td>
<td>75.30</td>
</tr>
<tr>
<td>14</td>
<td>17.51</td>
<td>17.13</td>
<td>0.67</td>
<td>0.05</td>
<td>57.11</td>
</tr>
<tr>
<td>16</td>
<td>22.64</td>
<td>21.81</td>
<td>0.86</td>
<td>0.05</td>
<td>44.17</td>
</tr>
<tr>
<td>18</td>
<td>28.42</td>
<td>27.14</td>
<td>1.05</td>
<td>0.04</td>
<td>35.19</td>
</tr>
<tr>
<td>20</td>
<td>34.96</td>
<td>33.11</td>
<td>1.25</td>
<td>0.05</td>
<td>28.60</td>
</tr>
<tr>
<td>22</td>
<td>41.58</td>
<td>39.47</td>
<td>1.48</td>
<td>0.05</td>
<td>24.05</td>
</tr>
<tr>
<td>24</td>
<td>49.05</td>
<td>46.79</td>
<td>1.67</td>
<td>0.07</td>
<td>20.39</td>
</tr>
<tr>
<td>26</td>
<td>57.56</td>
<td>54.97</td>
<td>1.98</td>
<td>0.04</td>
<td>17.37</td>
</tr>
<tr>
<td>28</td>
<td>67.16</td>
<td>64.23</td>
<td>2.29</td>
<td>0.05</td>
<td>14.89</td>
</tr>
<tr>
<td>30</td>
<td>76.17</td>
<td>72.99</td>
<td>2.57</td>
<td>0.06</td>
<td>13.13</td>
</tr>
</tbody>
</table>

Table 13 Time in milliseconds of ORB Descriptor based tracking

Table 13 shows the timing results for ORB, which is the first of the binary descriptors that attempt to deal with orientation. What is immediately apparent is that this is much slower than BRIEF, though it does still maintain a fairly impressive speedup over SIFT. One beneficial sign of this data is that the Hamming Distance was extremely close in time as compared to BRIEF. This shows that it is consistent in being an efficient method for scoring these binary descriptors and doesn’t suffer the variability of some clustering algorithms or statistical analysis methods. The main component of ORB that is causing this slowdown appears to be its orientation. As compared to BRIEF, which took on average ~0.01 milliseconds to scale, ORB took ~0.05 milliseconds. These times may seem insignificant, however, considering that the scale time is essentially the time it takes to describe a point, it could add up over an entire search region. This is abundantly clear when examined over the rest of the data and it appears that ORB is tracking at 105 FPS to BRIEF’s 400 for the same radius. Performing the linear algebra based orientation for ORB is a major time-sink, and as the results in Section 4.2, it is not particularly needed for this system. Despite this, ORB has still proven itself able to provide a real-time experience for most radii.
Table 14 Time in milliseconds of BRISK Descriptor based tracking

<table>
<thead>
<tr>
<th>Radius</th>
<th>Frame Avg</th>
<th>Description Avg</th>
<th>Scoring Avg</th>
<th>Scale Avg</th>
<th>FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5.77</td>
<td>2.79</td>
<td>0.56</td>
<td>1.90</td>
<td>173.31</td>
</tr>
<tr>
<td>12</td>
<td>7.14</td>
<td>3.79</td>
<td>0.82</td>
<td>1.99</td>
<td>140.06</td>
</tr>
<tr>
<td>14</td>
<td>8.46</td>
<td>4.92</td>
<td>1.02</td>
<td>1.95</td>
<td>118.20</td>
</tr>
<tr>
<td>16</td>
<td>9.86</td>
<td>6.08</td>
<td>1.36</td>
<td>1.88</td>
<td>101.42</td>
</tr>
<tr>
<td>18</td>
<td>11.19</td>
<td>7.27</td>
<td>1.68</td>
<td>1.72</td>
<td>89.37</td>
</tr>
<tr>
<td>20</td>
<td>13.12</td>
<td>8.78</td>
<td>2.01</td>
<td>1.74</td>
<td>76.22</td>
</tr>
<tr>
<td>22</td>
<td>14.66</td>
<td>10.38</td>
<td>2.34</td>
<td>1.37</td>
<td>68.21</td>
</tr>
<tr>
<td>24</td>
<td>17.43</td>
<td>12.30</td>
<td>2.83</td>
<td>1.76</td>
<td>57.37</td>
</tr>
<tr>
<td>26</td>
<td>20.07</td>
<td>14.47</td>
<td>3.22</td>
<td>1.78</td>
<td>49.83</td>
</tr>
<tr>
<td>28</td>
<td>22.03</td>
<td>16.14</td>
<td>3.64</td>
<td>1.68</td>
<td>45.39</td>
</tr>
<tr>
<td>30</td>
<td>24.79</td>
<td>18.38</td>
<td>4.16</td>
<td>1.70</td>
<td>40.34</td>
</tr>
</tbody>
</table>

Table 14 represents BRISK’s speed results and it has some interesting points within it. BRISK is the first of the two descriptors (BRISK and FREAK) that utilize a predefined pattern and use it for their orientation. It is also one of the first to show the benefits of optimizations. This is clearly evident in the results, because the scaling really tells the tale of how long it takes to become ready to describe a point. BRISK has to decide on octaves, rotations, and other parameters of that sort, so when it describes a single point, there is a significant portion of time that passes before any comparisons are made. This causes the scaling time to appear massive as compared to BRIEF and ORB. Instead of having a scale adaptation on the microsecond level, it takes well over a millisecond (almost two). Despite this, it is still significantly faster than ORB because its actual description time is so much lower. This allows for BRISK to maintain above real-time speed for the entirety of the testing set.

BRISK is also the first out of the binary descriptors that utilizes 64-Bytes. This did allow BRISK to maintain a higher average accuracy as shown in Section 4.2. However, it did have an effect on speed. The most apparent operation that this has affected is the speed of the Hamming Distance. Essentially, it takes twice as long to score
the 64-Byte BRISK as compared to BRIEF and ORB. This does show that the Hamming Distance scales exactly as expected. However, despite the increase in size, the description time for BRISK has shown to be fairly short, though still longer than BRIEF. This again, could be due to optimizations that have taken place in the actual implementation as bitwise comparisons can be handled in mass with special instruction sets.

Table 15 shows the timing results for FREAK. FREAK has shown throughout the entire set of tests to be an excellent descriptor and speed is no different. Much like BRISK, FREAK is a 64-Byte binary descriptor with a pre-defined pattern instead of random point comparisons. It has by far the slowest scale adaptation ranging anywhere from ~2.4 milliseconds all the way to ~4 milliseconds. Despite this, it has proven to be one of the fastest descriptors in the test set. It may not start as the fastest, but it scales the best by a fair margin. Starting from the 3rd fastest descriptor, it quickly becomes faster than BRISK, and by the end of the radii sweep, it has become faster than BRIEF, making it the fastest out of all the descriptor for large radii. This trend is almost entirely due to optimizations. The actual orientation portion of the description takes a fairly long time to complete, which is clear when comparing the scale adaptation time to the description.
time. For a radius of 10, the total description time is less than 2 milliseconds, while the scale adaptation time is closer to 4 milliseconds. This means, that in one call it is faster to describe an entire search grid, then two points described from separate function calls. Thus, there is clearly a lot of setup that must be accomplished before the actual description can take place. In the OpenCV implementation, the actual comparisons are made utilizing the SSE instruction set, so they can be performed in 128-bit chunks (or $\frac{1}{4}$ of the entire descriptor). This is what allows FREAK to only lose 10 or less FPS per radii increment, while others lose significantly more.

The other unique factor about FREAK is its coarse-to-fine matching technique. Where all of the other descriptors are matching the entirety of their descriptor, FREAK examines the first 16-Bytes and then makes a decision about whether to continue or not. This allows for both an increase in accuracy, and an increase in speed. This is shown in the time it takes for the scoring to complete. For smaller radii, it is taking a time that is more in line with a 32-Byte descriptor, and for the higher radii, it is actually taking less time, as a larger search radius generally leads to more points that are not going to match the dictionary. Thus, more points will fail the coarse-to-fine search pattern and will end up being thrown out. Combining this and previous optimizations, it is clear that FREAK is going to be able to utilize a larger search region than the other descriptors without sacrificing speed for it.

Table 16 is a more compact form of the speeds that were found for the tracking system. It is clear that SIFT is never at a point where it is valid for a real-time system. However, all of the binary descriptors have proven their merit as high efficiency descriptive entities. BRIEF had the highest overall speed, while FREAK scaled the best
with an increasing radius. Early, it was mentioned that when using a webcam, the program will be locked at 30 FPS, unless a special device is procured. So, all of the speed that the tracker maintains is not really necessary for a live application. However, instead of simply ignoring the extra speed, the processing resources available could be utilized. Since there are open cycles, instead of waiting for the webcam to return an image, the system could perform additional calculations. So, these results have proven that not only is the tracking system fast enough, it is actually fast enough to be expanded upon to fairly large degree.

<table>
<thead>
<tr>
<th>Search Radius</th>
<th>SIFT FPS</th>
<th>BRIEF FPS</th>
<th>ORB FPS</th>
<th>BRISK FPS</th>
<th>FREAK FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1.10</td>
<td>400.00</td>
<td>105.04</td>
<td>173.31</td>
<td>147.93</td>
</tr>
<tr>
<td>12</td>
<td>0.78</td>
<td>298.51</td>
<td>75.30</td>
<td>140.06</td>
<td>137.17</td>
</tr>
<tr>
<td>14</td>
<td>0.58</td>
<td>232.02</td>
<td>57.11</td>
<td>118.20</td>
<td>129.37</td>
</tr>
<tr>
<td>16</td>
<td>0.45</td>
<td>184.84</td>
<td>44.17</td>
<td>101.42</td>
<td>117.79</td>
</tr>
<tr>
<td>18</td>
<td>0.36</td>
<td>151.06</td>
<td>35.19</td>
<td>89.37</td>
<td>113.38</td>
</tr>
<tr>
<td>20</td>
<td>0.30</td>
<td>124.38</td>
<td>28.60</td>
<td>76.22</td>
<td>103.31</td>
</tr>
<tr>
<td>22</td>
<td>0.26</td>
<td>105.60</td>
<td>24.05</td>
<td>68.21</td>
<td>92.17</td>
</tr>
<tr>
<td>24</td>
<td>0.22</td>
<td>89.53</td>
<td>20.39</td>
<td>57.37</td>
<td>82.78</td>
</tr>
<tr>
<td>26</td>
<td>0.18</td>
<td>76.86</td>
<td>17.37</td>
<td>49.83</td>
<td>74.07</td>
</tr>
<tr>
<td>28</td>
<td>0.16</td>
<td>65.92</td>
<td>14.89</td>
<td>45.39</td>
<td>68.45</td>
</tr>
<tr>
<td>30</td>
<td>0.14</td>
<td>58.75</td>
<td>13.13</td>
<td>40.34</td>
<td>61.01</td>
</tr>
</tbody>
</table>

Table 16 Tracker speed over various search radii

Table 17 is a brief summary of how the different descriptors handle objects of different sizes. Instead of actually describing objects of different size, these tests were run on a set of sizes that was more controlled. What this shows, yet again, is that SIFT is not suited for this style of work. As an object grows bigger, it slows down even further, so it would not have a consistent experience across many videos. ORB is the only one out of the binary descriptors that did not perform entirely as expected. This is due to its method of finding the orientation which is based on linear algebra. Due to this, its description time increases a small amount when extremely large sizes are encountered. This is
nowhere near the increase in time that was seen by SIFT, however it does mean that ORB may have trouble tracking on high-definition videos where the objects will be larger. BRIEF, BRISK, and FREAK all show the anticipated behavior of not suffering any slowdown due to scale. As they are simply a series of point comparisons, the absence of slowdown makes sense, because the distances between points can just be scaled. This means that BRIEF, BRISK, and FREAK can provide a stable experience across many different image resolutions, including high-definition.

<table>
<thead>
<tr>
<th>KeyPoint Size</th>
<th>SIFT (ms)</th>
<th>BRIEF (ms)</th>
<th>ORB (ms)</th>
<th>BRISK (ms)</th>
<th>FREAK (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.000</td>
<td>4.153</td>
<td>0.004</td>
<td>0.005</td>
<td>1.063</td>
<td>1.806</td>
</tr>
<tr>
<td>8.000</td>
<td>4.266</td>
<td>0.004</td>
<td>0.005</td>
<td>1.089</td>
<td>1.794</td>
</tr>
<tr>
<td>16.00</td>
<td>4.544</td>
<td>0.004</td>
<td>0.005</td>
<td>1.076</td>
<td>1.811</td>
</tr>
<tr>
<td>32.00</td>
<td>6.221</td>
<td>0.004</td>
<td>0.006</td>
<td>1.076</td>
<td>1.813</td>
</tr>
<tr>
<td>64.00</td>
<td>8.329</td>
<td>0.004</td>
<td>0.013</td>
<td>1.085</td>
<td>1.731</td>
</tr>
<tr>
<td>128.0</td>
<td>9.66</td>
<td>0.004</td>
<td>0.040</td>
<td>1.093</td>
<td>1.711</td>
</tr>
<tr>
<td>256.0</td>
<td>9.915</td>
<td>0.004</td>
<td>0.142</td>
<td>1.065</td>
<td>1.728</td>
</tr>
</tbody>
</table>

Table 17: Time in milliseconds of description creation for varying sizes of descriptor

4.5. Robustness

Despite all attempts made to harden the system to all the challenges in tracking, a large portion of its robustness comes from the descriptors themselves. Each descriptor was shown to have its own unique response to varying stimuli, which could help to contribute to its overall performance. For the rotation test, an object of interest was defined, described, and then the image was rotated around the object (so it would remain in the same location). By not moving the object, the same location could be kept and only a single description/comparison was done for each new angle of rotation. Figure 24 is the object used for all rotation and scale testing. This image was chosen because, as shown in Section 4.3, all the descriptors had shown a high distinctness on this image. This means that they should all be able to describe and match it well.
Figure 24 Object used for all rotation and scale invariance tests

Figure 25 shows the different descriptors’ invariance to rotation. What should be noted immediately is that SIFT is plotted onto the right (secondary) axis as the Euclidean distance would have skewed the Hamming Distance results. It should also be noted that it actually behaves fairly similarly to the binary descriptors in that it has peaks and troughs as the object that is being described rotates. However, it is clear that SIFT is not entirely rotationally invariant. Instead of having a low distance for all the angles of rotation, it has large spikes and even the lowest points are fairly high.
Moving onto the binary descriptors, some interesting events occur. First, there is BRIEF, which actually follows the same pattern of a lot of rotationally invariant descriptors. It does reach a fairly high distance away from the non-rotated object, but it does have a lower distance at certain increments of the rotation angle. So, despite not actively trying to mitigate angles, the pattern of random points appears to have helped it perform better at certain angles as compared to other ones. ORB, being the rotationally invariant form of BRIEF, does better than BRIEF throughout most of the test. However, there is a specific section where BRIEF actually has a lower error. Testing on multiple images has shown a similar behavior for ORB, thus it can be concluded that ORB’s orientation method is not able to perform well in all cases.
Moving onto BRISK and FREAK, the two descriptors with defined patterns, it is clear that their results are both drastically different than BRIEF and ORB. BRISK has shown an immense amount of error in this comparison, greater even than BRIEF, which claims no rotational invariance. This could be a result of BRISK’s ring based pattern. The rings could be more sensitive to rotation simply because they are circular and the comparisons are all based on where the center point of each ring’s circles are. Unfortunately, this could also show that BRISK simply does not have a good method for mitigating rotation. It does display a fair amount of skill in lowering the error for specific angle sets, much like all of the other descriptors. At these points (~135° and ~270°) it does show the highest level of mitigation. That also could be due to the fact that it has the highest error beforehand. Of course, this large error may be due to the fact that BRISK is twice as long as BRIEF and ORB. With twice the bits to compare, it makes sense that twice as many errors would occur. Figure 26 shows the effect of normalizing the Hamming Distance so that the 64-Byte descriptors have the same maximum distance as the 32-Byte descriptors. This confirms that BRISK would perform just comparably to ORB and BRIEF. Thus, if it was a 32-Byte descriptor their results would have been similar initially.
FREAK, despite having a ring-based sampling pattern like BRISK, maintains the lowest error throughout the entire rotation test. This is truly a testament to FREAK’s ability as a descriptor as its error very rarely approaches 50, let alone the 150 or 300 of other descriptors. It also never has a distance of 512, which means that it never fails its own coarse-to-fine test. What this all boils down to is that FREAK has an excellent method for rotation invariance. FREAK’s method for rotation invariance is actually fairly similar to BRISK’s, however instead of using several hundred pairs for calculating the local gradient, FREAK uses 45. The pairs are not selected from the description pattern, but instead come from a defined orientation pattern. Overall this leads to an excellent method that has proven its merit in being rotationally invariant.
Scale invariance is critical in the continued tracking of an object, as it is very rare for an object to remain at the same distance through the entire video sequence. Thus, it is important to find out which descriptors are able to best handle scale. This allows for good performance while tracking a video with a lot of motion towards or away from the camera.

![Figure 27 Scale Invariance for all descriptors](image)

Figure 27 is the scale invariance tests for all of the descriptors. Again, SIFT has been plotted on a secondary axis that represents Euclidean distance to help not skew the results of the Hamming Distance. In this set of tests, SIFT has performed fairly well. Instead of being very jumpy in terms of distances, it seems to settle into a fairly consistent distance. This is especially true of the higher scales, where it is seen to be fairly consistent in error. It does better with scaling up than scaling down, which makes
sense because scaling upwards will wash out data by simply expanding a small region of what was already being described. However, when scaling downwards, parts of the image that were not previously described are now entered into the equation. This entirely new data serves to pollute the descriptor and causes a worse match.

BRIEF, in all of its simplicity, has proven to be an excellent candidate for remaining invariant to scale. Where the rotation caused BRIEF to sample vastly different points which yielded a high distance, scale appears to have not caused this at all. Throughout the test it has maintained a lower distance than all other descriptors aside from a single scale in which FREAK did better. BRIEF does not do any octave matching to try and determine scale, so the fact that it has beaten all other descriptors is very impressive. Of course, this could also mean that the pattern itself isn’t advanced enough to pick up on the fact that the object is changing in scale. However, based on the accuracy results, it simply works that the pattern BRIEF is using samples fairly well as an object increases in size.

ORB functions in a similar fashion to BRIEF for this test, just as it did for the rotation. However, in this case it is slightly worse than BRIEF for scaling. This could be due to its attempt at rotating the descriptors as size increases. As the object becomes larger, it will slowly start to pollute the descriptor with similar information, as areas stretch and eventually occupy the whole screen. What this does for the image patch (and the moment used to determine angle) is a sort of averaging effect. Instead of having different sections of the object which come in different intensities, it has much fewer (the patch could become one color in the extreme case) which would cause the moment to return a poor result. This would cause the descriptor to be rotated improperly and then
increase the distance to the dictionary. This effect would eventually settle down which would explain why as the scale increased, ORB settled onto a particular distance and then remained very close to it for the remainder of the testing.

As with the rotation tests, BRISK has proven to have the highest distance from the dictionary as more and more change occurred. It has approximately twice the error of BRIEF and ORB, which could be due again to the fact that BRISK is twice the length of BRIEF and ORB, or simply that it is more sensitive to scale. Despite is larger error it does still maintain the same general pattern that the previous descriptors have shown. The error increases to a point, and then settles down to a fairly constant value. BRISK may be experiencing a high error rate because of the method used to handle scale. The actual scaling method takes place during the detection phase for BRISK. So, instead of letting it find the same keypoints each time, thus ensuring a similar pattern, it is instead being forced to use a set of keypoints. Thus, the scale method isn’t even considered, so very little is actually being done to handle variations in scale.

As shown in Section 3.2.2, FREAK has the ability to do coarse-to-fine matching. This plays a role in scale invariance, as it does in so many other parts of the system. Rotation was easily dealt with by FREAK; however, scale appears to be a bigger challenge. It is clear that after a scaling factor of only 1.3, the coarse-to-fine matching fails and throws out the descriptor. What this means for the tracker is that FREAK is going to start mismatching very rapidly when scale is brought into the equation. This can be mitigated to some extent with the dynamic dictionary which helps keep references of the intermediate scales. However, if any very rapid change in scaling occurs, then it is likely that FREAK will produce a very large distance and may lose the object.
These scaling results have pointed to the necessity for scaling the descriptor over time. Instead of relying on the descriptors themselves, the system can benefit by allowing the descriptor region to resize instead of remaining static, which should mitigate at least part of the error.

Figure 28 Scale Invariance with Changing Descriptor Scales

Figure 28 displays the results of what happens to scale invariance when the descriptors are allowed to resize with the object. In this case, the size is actually set based on the scaling factor, so it should in theory be a perfect match of the object. This is proven to be untrue, though it did cause the error to go down a significant amount.

BRIEF and BRISK both took excellently to scaling, causing their error to diminish to the point that it averaged close to 10 instead of 100 or 200 respectively. This is especially impressive for BRISK, as it had twice the error of BRIEF before scaling but
now has approximately half the error of BRIEF. This shows that BRISK’s pattern takes very well to scaling. This could be due to its circular nature, or other contributing factors, but overall it is an impressive result. The results for BRIEF make sense because they are simply just moving the pattern to a new scale, which in theory would cause the descriptor to sample the same points each time. Based on the actual implementation of the image stretching, this may not be true, but it does at least get the descriptor a lot closer than before.

ORB appears relatively unaffected by the changing of scale. This is counterintuitive, as the change in scale should in theory mitigate the actual object changing in size. There should not be any changes in the angle the gradient displays, nor should any differences in relative location of the pairs on the object occur. Nevertheless, its distances remain relatively unchanged between the scaled and non-scaled versions.

FREAK is, as usual, a unique case for this test. Much like BRIEF and BRISK, for the first half of the test it shows that the distances have been lowered. However, there is a random spike of very high distance (the result itself is thrown out) and then afterwards, there is a maintained high distance. Granted, it is lower than when scaling was not used, but it does not maintain the low distance that other descriptors have. Essentially, what this means is that even with scaling, FREAK will reach a point where it is likely to stop matching the object.

Overall, these results have shown that each descriptor may have its own specific performance, but they generally follow the same general pattern. Each descriptor has also shown different levels of robustness. What this translates into is the descriptors base ability to handle the changes that occur over a video. However, it has already been shown
that even the simplest of descriptor works well in the system based on all the mitigation attempts that it makes. It should not be ignored that FREAK, which performed well in the rotation tests and the beginning of the scaling test, has the best overall accuracy. This is because the areas where it started to break down (high scales) are generally not seen in typical videos. Usually, an object will stay within the same general plane and its size will usually not increase by over 2 times.
Chapter 5  Conclusions and Future Work

As a tracking system, the proposed tracking solution based on binary descriptors has proven to be an excellent alternative to today’s state-of-the-art trackers. Furthermore, it has potential for future work. Thanks to its ability to run at above real-time speeds, there is room for more to be done with the algorithm. This could range from attempts to improve accuracy, to entirely new actions.

5.1. Descriptor Selection

As the average user would not be expected to choose a descriptor before running the tracker, it is important to select one descriptor to be used in the system. Examining all of the results given in Chapter 4 it is clear that, though all descriptors may perform well under certain circumstances, one performed better than the rest. For this system, FREAK was able to obtain the best overall results. This was clear in all measures of ability. Throughout the accuracy test, FREAK has shown the best overall results in terms of average error in both the tuned parameter set and the unified parameter set. This makes it the best for specific video tracking as well as generic tracking. It has also shown that, despite not being the fastest overall, its speed did scale the best with grid size. This allows for a larger radius than the other descriptors which could be necessary in videos with excessive motion. In addition to these tracker-based metrics, FREAK also did very well in both the distinctness tests and the robustness tests. For these reasons, FREAK should be the default choice for this tracking system.
5.2. *Multiple Object Tracking*

One of the main areas that could be easily improved in this system is the ability to track multiple objects, which is becoming a popular topic. An area where this is usually applied is tracking multiple faces or people in a crowd. By monitoring many different people, flows could be determined, or a single person of interest could be filtered out of the others. Attempts at multiple object tracking can be seen in [36] and [37]. Both of these attempts utilize a complicated method which does work, but at a high computational cost.

Tracking multiple objects would be easy to implement with the proposed system, because all that would be required is to keep track of a dictionary for each object in question. Based on the descriptor that is used, this could be as little as 32-Bytes of data for a static dictionary. Thus, for less than a KB per object, this system could scale very effectively.

Examining the speed results and assuming a linear scale rate (this appears to be valid), it is clear that in its fastest configuration; the system takes 2.5ms per frame to track. Translating this into a 30FPS state, it is clear that 13 objects could theoretically be tracked at real-time speeds using a CPU. Transitioning to the slowest FREAK configuration (~60FPS), this would allow for 2 objects to be tracked at a real-time speed. So, even in this simple, non-optimized method, the system still has the theoretical ability of tracking multiple objects at real-time speeds.
5.3. **GPU Acceleration**

Graphical Processing Units (GPUs) have been used in an increasing amount of research in terms of both applications and hardware design. The reason that GPUs are now used more is because of their ability to do parallel operations. A standard GPU will utilize a Single Instruction Multiple Data (SIMD) style of execution. This allows for programs with a lot of data to execute the algorithm across its entire dataset in parallel. There have already been attempts at making GPU based tracking algorithms. Some have taken the method of creating one designed specifically for the GPU such as [38] while some have gone the other direction and simply accelerated an existing algorithm [39]. These methods have both shown speeds averaging in the several hundred FPS all the way to 1000 FPS, so there is clearly a benefit to putting an algorithm into a GPU.

This system would take very well to GPU acceleration because of the independence of all of the candidates. A prototype GPU implementation is presented in [11]. Instead of the current setup, which each point in the search grid is processed independently, they could very easily be processed in parallel via a GPU. Going even further than this, each point inside the descriptor internally could be computed in parallel as well. This is all because every candidate and every point in the descriptor are all independent from each other. Even the Hamming Distance can be broken down into a parallel computation as each Byte is independent of the other Bytes. Finally, even the candidate selection can be handled in parallel by using parallel reduction search algorithms. In short, the vast majority of the calculations in this algorithm can be done in parallel, which makes it a good candidate for an implementation.
Moving past the base algorithm implementation, the GPU has a very promising outlook for multi-object tracking. As adding additional objects to track is simply a matter of describing more candidates and then matching them against a specific dictionary, each of these new objects could be tracked in parallel. This could allow for an extremely large number of objects to be tracked simultaneously, which would be an extremely valuable asset in a traffic monitoring or crowd tracking program.

Now, despite the benefits that GPUs may provide, they may not actually be the best platform for this specific algorithm. The reason for this is that although the algorithm does have many parallel elements, it is not massively parallel, essentially it does not contain millions of elements to be processed. Thus, the GPU itself may not be fully utilized and could spend time stalling or filling hardware with pointless operations. Thus, it may be a better idea to utilizing threading in a multi-core environment and let a single CPU core take care of an object. These cores are much larger, and can handle the more complicated operations instead of having to break down the algorithm into simple kernels. It would also help to mitigate the transfer time onto the GPU, which is not insignificant when frames are transferred. Without having to worry about transfer time would also allow for the tracker to be run on high definition videos. These high definition frames would take much too long to transfer onto a GPU, so the shared memory benefits of a threaded system would be much better in this case.

5.4. Mobile Implementation

The mobile field has been blossoming in a fairly spectacular fashion in recent years. With the advent of more and more powerful smartphones and tablets, the amount of complex algorithms and computer vision that has been implemented has skyrocketed.
Generally these mobile platform methods are based on a low complexity approach, so that it can be run on the device. There are examples of these methods in [40] and [41].

One of the main benefits of this tracking system is its low complexity. The operations are all based on simple bitwise comparisons. The only place where the algorithm gets fairly complicated is in the orientation calculations for the descriptors. Since the algorithm was already designed to be lightweight, it should be possible to port it on a mobile phone environment with no loss of precision. Once implemented on a mobile platform, it could join the field of virtual reality or augmented reality, both of which often need the location of objects of interest.

5.5. **Additional Features**

Since the tracking algorithm runs faster than 30 frames per second, there is room for extra computations without compromising real-time performance. Additional features could be considered for improving accuracy or for calculating additional information.

One piece of new information that could be generated from what the tracker is already providing is object location in 3D space or pose-estimation should the object be a face. The logic behind this improvement is that if multiple sub-objects on a single main object are tracked, they could provide information on how they move together. Their relative locations and the distances between them could give them the ability to measure motion going forwards and backwards. It could also be used to determine overall object rotation. If this was done with eyes on a face, for example, the two eyes could provide distance, assuming a frontal pose. This could then be extrapolated into where the user is looking at the screen, which would play back into the idea of augmented or virtual reality.
Another area that could be explored for the sake of increased accuracy is multiple channel descriptions. There has already been work done in this field [42] with the advent of descriptors that take into account color images and depth images (RGB-D). These cameras have become increasingly common especially since Microsoft has released their Kinect sensor which provides RGB-D images. Utilizing the extra information could definitely improve the accuracy of the tracker because there is more discriminative information. There are already binary descriptors which can describe RGB-D objects; an example of this is BRAND [43]. However, there may not always be a depth camera available to increase the amount of information available. So, instead of utilizing the depth channel of an image, a binary descriptor could be made to describe all three RGB channels individually. This is three times the amount of data that a grayscale image provides. By simply extending the method the descriptors utilize (threshold comparisons to construct the binary string) on each channel; the information should increase without a drastic increase in time or complexity.

5.6. Conclusions

Throughout the testing of this system it has become clear that the proposed tracker performs excellently in terms of overall accuracy and speed. When compared to state-of-the-art tracking systems, it has the ability to track better on all of the videos used when tuned and better on over half of the videos when using a generic parameter set. What is important to take away from these results, however, is the future potential for this system. With the tracker already running at an above real-time level, there is a fair amount of room to add onto this system, or integrate it into an already existing system. This system also has the huge advantage of being able to move directly into a high
resolution video set without any hits to speed. This means that the proposed system is completely ready for a time when a standard camera is capturing video in high definition. Once high definition videos are standard, this would allow the tracker to easily follow an object and even have a better defined sub-object set. This means that as resolution increases, the ability for augmented reality or virtual reality increases as pose-estimation or 3D location would be even more accurate.

In summation, the tracker already works as an excellent system, but it may actually be an even more impressive starting algorithm than a complete system. There are so many application areas and avenues for future work that the possibilities are almost endless.
Bibliography


ECCV, 2006.


[38] J. F. Henriques, R. Caseiro, P. Martins and J. Batista, "Exploiting the Circulant


Appendix

Figure 29 Ground truth comparison for different descriptors on Cliffbar
Figure 30 Ground truth comparison for different descriptors on Coke Can
Figure 31 Ground truth comparison for different descriptors on David
Figure 32 Ground truth comparison for different descriptors on Dollar
Figure 33 Ground truth comparison for different descriptors on Occluded Face 1
Figure 34 Ground truth comparison for different descriptors on Occluded Face 2
Figure 35 Ground truth comparison for different descriptors on Girl
Figure 36 Ground truth comparison with different descriptors on Surfer
Figure 37 Ground truth comparison with different descriptors on Sylvester
Figure 38 Ground truth comparison with different descriptors on Tiger 1
Figure 39 Ground truth comparison with different descriptors on Tiger 2
Figure 40 Ground truth comparison with different descriptors on Twinings