On Learning from Videos

by

Yingnan Cui

Submitted to the Department of Mechanical Engineering
on June 22, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

The robot phone disassembly task is difficult in many ways: It has requirements on high precision, high speed, and should be general to all types of cell phones. Previous works on robot learning from demonstration are hardly applicable due to the complexity of teaching, huge amounts of data and difficulty in generalization. To tackle these problems, we try to learn from videos and extract useful information for the robot. To reduce the amounts of data we need to process, we generate a mask for the video and observe only the region of interest. Inspired by the idea that spatio-temporal interest point (STIP) detector may give meaningful points such as the contact point between the tool and the part, we design a new method of detecting STIPs based on optical flow. We also design a new descriptor by modifying the histogram of optical flow. The STIP detector and descriptor together can make sure that the features are invariant to scale, rotation and noises. Using the modified histogram of optical flow descriptor, we show that even without considering raw pixels of the original video, we can achieve pretty good classification results.

Thesis Supervisor: Kamal Youcef-Toumi
Title: Professor
Acknowledgments

I would like to first thank my family, who have been providing support for my studying overseas. In the past two years, they constantly sent to me their love and concern, giving me the courage to face the challenges.

I am extremely grateful to my advisor, Prof. Kamal Youcef-Toumi, for the opportunity he has provided and guidance of my research at Mechatronics Research Lab. Thanks to Kamal’s help, I feel I’ve made a lot of improvements in my research.

Thanks also to my labmates, who have been giving me lots of valuable advice both for my personal life and research. Thanks to Bo Jiang, who has always been an inspiration for me. Thanks to Wilhelmena Figueiredo, for helping me get familiar with the RecycleBot and providing all of the resources I need for the thesis. Thanks to Ali Jahanian, for helping me with the optical flow code. You all keep me happy.
### Contents

1 Introduction 15
   1.1 Impact of E-waste 15
   1.2 A Close Look at Phone Recycling 16
   1.3 Previous Work on Phone Disassembly 16
   1.4 Recybot: A Phone Disassembly Framework 17
   1.5 Motivations for Using Computer Vision in Phone Disassembly Project 19
   1.6 Organization of the Work 20

2 A Review of Image Alignment 23
   2.1 The Goal of the Lucas-Kanade Algorithm 23
   2.2 The Cost Function 24
   2.3 Derivation of The Lucas-Kanade Algorithm 25
   2.4 Requirements on the Set of Warps 26
   2.5 Summary 26

3 Finding the Region of Interest 27
   3.1 Detecting Changes in the Scene 27
   3.2 Noise Cancellation 29
   3.3 Conclusion 32

4 Interest Points Detection 35
   4.1 An Introduction to Space-time Interest Points 35
   4.2 A Detector Working on Optical Flow 37
List of Figures

1-1 The Recybot at the Mechatronics Research Lab, MIT. .................. 18

1-2 The tools of Recybot. .................................................. 19

1-3 The process of video classification that will be discuss in Chapter 4, 5
and 6. ............................................................................. 21

3-1 The first frame of the video. ................................................. 28

3-2 The part is being removed using a screw driver. ...................... 29

3-3 The part has been removed. .................................................. 29

3-4 Subtracting the background. .................................................. 30

3-5 First using Otsu’s method to find the global threshold, then binarize
the image. ........................................................................ 31

3-6 Remove small objects in the binary image. Note the noise in Fig. 3-5
disappeared. .................................................................... 31

3-7 The mask that is to be applied to the original video. ................. 32

3-8 One frame of the masked video. .......................................... 33

3-9 One frame of the processed video. The yellow circle shows the detected
interest point, which in this case is the contact point between the screw
driver and the part that is being removed. ............................ 33
4-1 Result of detecting the strongest spatio-temporal interest points in a football sequence with a player heading the ball (a) and in a hand clapping sequence (b). From the temporal slices of space-time volumes shown here, it is evident that the detected events correspond to neighborhoods with high spatio-temporal variation in the image data or 'space-time corners'. Figure from [15].

4-2 Two successive frames from a walking video, which is part of Weizmann Human Activity Video dataset [13].

4-3 The computed optical flow for two consecutive frames in Fig. 4-2.

4-4 For each octave of scale space, the initial two channels of the flow image are repeatedly convolved with Gaussians to produce the set of scale space images shown on the left (only one channel is shown). Adjacent Gaussian smoothed flow images are then subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian smoothed flow image is down-sampled by a factor of 2. Figure from [17].

4-5 The constructed scale space for the flow image shown in Fig. 4-3. The original flow image is split into two channel and computed separately. Each row of images corresponds to one octave. For every row, from left to right, the original flow image is convolved with Gaussian kernels whose variances are becoming larger and larger (spaced by k). Therefore, as can be observed, the flow images become more and more blurred going from left to right. Going from up to down, since the flow images are subsampled after each octave, the images are smaller and smaller. The whiter the pixel, the higher is the value, and the larger is the motion.
4-6 The difference-of-Gaussian flow images for the two channels. These images are obtained by subtracting every two smoothed flow images that are in adjacent scale. For each row (octave), there are 5 Gaussian-smoothed flow images in Fig. 4-5, so after subtraction there are 4 difference-of-Gaussian flow images each row (octave). The whiter the pixel, the larger is the difference between two adjacent scales in that pixel. From the figure, the contour of the shape of the person is usually significantly whiter, meaning higher difference.

4-7 Extrema of the difference-of-Gaussian flow images are detected by comparing a pixel (marked by ×) with its 26 neighbors in 3 × 3 regions at the current and adjacent scales (marked with circles). Figure from [17].

4-8 The solid line of (a) shows the percent of keypoints that are repeatably detected at the same location and scale in a transformed image as a function of the number of scales sampled per octave. The dotted line shows the percent of keypoints that have their descriptors correctly matched to a large database. Fig. (b) shows the total number of keypoints detected in a typical image as a function of the number of scale samples. Figures from [17].

4-9 The solid line in the graph shows the percent of keypoint locations that are repeatably detected in a transformed image as a function of the prior image smoothing for the first level of each octave. The dotted line shows the percent of descriptors correctly matched against a large database. Figure from [17].

4-10 Randomly chosen two frames from a walking video in [13].

4-11 The detected extrema on randomly chosen two frames without pruning.

4-12 The remaining points after pruning using contrast thresholding. The contrast threshold is set to be 0.03.

4-13 The remaining points after pruning using edge response thresholding. The ratio $r$ is set to be 10.
4-14 The remaining points after pruning using contrast thresholding. The contrast threshold is set to be 0.15.

4-15 The remaining points after pruning using edge response thresholding. The ratio $r$ is set to be 10.

5-1 An illustration of the rotation invariance of the flow pattern. Suppose the two local flow domains are surrounding the same keypoint, i.e., nothing is changed from the left flow image to the right flow image except it is rotated, the descriptor should be able to recognize they are the same pattern.

5-2 A keypoint descriptor is created by first computing the flow magnitude and orientation at each image sample point in a region around the keypoint location, as shown on the left. These are weighted by a Gaussian window, indicated by the overlaid circle. These samples are then accumulated into orientation histograms summarizing the contents over $4 \times 4$ subregions, as shown on the right, with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This figure shows a $2 \times 2$ descriptor array computed from an $8 \times 8$ set of samples, whereas the experiments in this work use $4 \times 4$ descriptors computed from a $16 \times 16$ sample array. Modified from [17].

6-1 A sample frame for each action in the Weizmann action dataset [13].

6-2 Silhouette plot of the divided clusters. When using $k$-means clustering we set $k = 10$. Most of the components are within the corresponding clusters, though clusters 3, 4, 5, 6, 7, 8, 9 all have a few outliers.

6-3 The confusion matrix computed by using $n$-fold cross validation. For the divided 3 sets, we train on every two sets and test on the other. One simplification in implementation is that clustering is done only once, using all of the data.
List of Tables

6.1 Change of classification error rate as the number of clusters changes.

The classifier being used is 1-nearest neighbor classifier. . . . . . . . . . . . 60
Chapter 1

Introduction

1.1 Impact of E-waste

Each year there are up to 50 million metric tons of E-waste globally [6]. The United States is the world leader in producing electronic waste, tossing away about 3 million tons each year. China already produces about 2.3 million tons (2010 estimate) domestically, second only to the United States [7].

The processes of dismantling and disposing of electronic waste in developing countries led to a number of environmental impacts. The dumping of cathode ray tubes can allow lead, barium and other heavy metals to leach into the ground water and release toxic phosphor. For printed circuit board, open burning and acid baths to remove metals can have air emissions and discharge into rivers of glass dust, tin, lead, brominated dioxin, beryllium cadmium, and mercury. Chemical stripping using nitric and hydrochloric acid and burning of chips will lead to PAHs, heavy metals, brominated flame retardants discharged directly into rivers acidifying fish and flora [22].

Unprotected workers at the recycling sites are exposed to PAH-containing dust and gaseous fumes via inhalation, ingestion and dermal contact. It was also shown that PAH contamination extends to the surroundings of e-waste recycling areas [20]. Those PAH emissions represent a health hazard for e-waste workers and residents living on or near e-waste recycling areas, since several PAH metabolites have been
recognized as carcinogenic. The association between high PAH exposure and the risk of lung cancer has been well established [18], and e-waste recycling activities in southern China have been associated to an increased inhalation cancer risk [21].

1.2 A Close Look at Phone Recycling

Humans toss millions of cell phones each year in favor of newer technology. According to the U.S. Environmental Protection Agency (EPA), 141 million mobile phones were discarded in 2009 and only 12 million of those were collected for recycling [5]. Most cell phones contain precious metals and plastics that can be recycled to save energy and resources that would otherwise be required to mine or manufacture. When placed in a landfill, these materials can pollute the air and contaminate soil and drinking water [3]. Cell phone coatings are typically made of lead, which is a toxic chemical that can result in adverse health effects when exposed to it in high levels. The circuit board on cell phones can be made of copper, gold, lead, zinc, beryllium, tantalum, coltan, and other raw materials that would require significant resources to mine and manufacture [4]. This is why it is important to recycle old cell phones and source these increasingly scarce materials whenever possible.

Mobile phones have value well after their intended use. Yet the value of these phones to recyclers is marginal and relies on high volume to become profitable. The average cost in 2006 to extract the precious metals for the U.S. cell phone recycling company ECS Refining was $0.18 while the average revenue from the recycled metals was $0.75 [19]. With a profit margin significantly smaller than refurbished units, this method of gaining economic value from the recycling of cell phones is significantly more volume dependent.

1.3 Previous Work on Phone Disassembly

Previous efforts on cell phone disassembly are not satisfactory. Most phone manufacturers don’t have phone disassembling robots. While Apple designed Liam for
disassembling the iPhone, it works only for iPhone 6 model and still needs human’s assistance for some parts. Liam has one fully detailed model and is not generalized enough for other models. It is a warehouse-sized disassembly machine, taking three years to make. It can disassemble up to 1.2 million iPhones a year, while Apple sold 230 million iPhones in 2015 alone [2].

More recently, Apple debuted a new robot named Daisy, which is capable of taking apart nine different versions of the iPhone, and it can disassemble up to 200 iPhone devices an hour [1]. But still, Daisy is incapable of disassembling other cell phone models.

1.4 Recybot: A Phone Disassembly Framework

The Recybot at the Mechatronic Research Lab is a high-speed intelligent robotic system for low-cost electronic recycling. Unlike Apple’s Liam and Daisy, the aim of this robot is to be capable of disassembling all types of cell phones. This is a challenging task, considering that different cell phone types can have completely different layouts and components. If using model based approach, all of the phone models’ detailed information would be required and the disassembly procedures for each type of phone would be defined, which is impractical. Instead, we seek to adopt robot learning in the application of phone disassembly. In this particular work, we aim to have the robot learn by observing what humans are doing in videos.

The Recybot has a straightforward design with low-cost in mind. It uses a 3D printer, Anet A8, as its framework, which has 3 translational degree of freedom with stepper motors. The heat bed was replaced with a base to hold the phone. The extruder was replaced with an Arduino controlled tool holder, which adds another two degree of freedoms: roll and yaw. The total cost of Recybot is only $315. Fig. 1-1 shows the Recybot. Fig. 1-2 shows its tools.
Figure 1-1: The Recybot at the Mechatronics Research Lab, MIT.
1.5 **Motivations for Using Computer Vision in Phone Disassembly Project**

We are interested in getting all kinds of information from a video for robot manipulation. Specifically, our long-term goal is to let a robot for phone disassembly learn to disassemble a phone from processing videos in which a human is disassembling a phone. To this end, in this thesis we explore different techniques that may be helpful for the robot to learn.

Robot learning from videos has great potentials in the area of robot learning. It requires minimal efforts in getting the training data. The teacher no longer have to grab the tool of a robot to teach the task, but can do the same task like a human and use the recorded videos as the training data. As there are a lot of videos online, we can potentially use these videos as the training data. Pre-existing knowledge can be incorporated in the learning, which may help accelerate the process.

Previous works on learning from demonstration rarely use raw pixels as the only
kind of input. Only recently do people pay attention to robot learning from videos, but still other information, like robot configuration, is required in the training stage. In their work, Finn et al. used a convolutional neural network (CNN) to represent the policy. They claim that after training the CNN, the robot is able to repeat the task in a given video [12]. The problem of their method is that it needs a lot of data. In their real-world placing experiment, they collect 1293 demonstrations and use 1197 of them as the training set. Also, they have to use pre-existing data: they initialize the first convolution layer from VGG-19 and keep it fixed during training.

Our work is addressing robot learning from video in a direct and efficient way. Specifically, this work is trying to solve two problems. The first is how to find region of interest and points of interest for robot manipulation. The second is how to recognize what kind of action is happening in a given video.

For the first problem, we show how to find the region of interest by creating a mask for the region. To find the points of interest, we give a new method of finding interest points by detecting local extrema in the optical flow domain.

For the second problem, we modified the previous work on histogram of optical flow to serve as descriptors for the interest points detected. And we do video classification using a small dataset. We show that even the dataset is small, our method can give satisfying results.

1.6 Organization of the Work

In Chapter 2 we introduce image alignment, which is the foundation of this work. In Chapter 3, we present how to get the region of interest of a video. In Chapter 4 we give a new method of getting space-time interest points. In Chapter 5, we talk about the rotation invariant descriptor. In Chapter 6, we combine everything and show how to do video classification using the designed detector and descriptor. Fig. 1-3 shows the process of video classification that will be discussed from Chapter 4 to Chapter 6.
Input Videos with Labels

Optical Flow

Flow Images

Interest Points Detector

Interest Points

Optical Flow Descriptor

Interest Point Descriptors

\(k\)-means Clustering

Assignment Based on \(\chi^2\) Distance

Clusters

Video Descriptors with Labels

\(k\)-Nearest Neighbors

Classifier

Figure 1-3: The process of video classification that will be discuss in Chapter 4, 5 and 6.
Chapter 2

A Review of Image Alignment

In this chapter we give an introduction to image alignment. We will show the Lucas-Kanade algorithm. The image alignment algorithm will be used to compute optical flow between two consecutive frames, which will be the foundation of our work.

2.1 The Goal of the Lucas-Kanade Algorithm

The goal of the Lucas-Kanade algorithm is to align a template image $T(x)$ to an input image $I(x)$, where $x = (x, y)^T$ is a column vector including the pixel coordinates. When using the Lucas-Kanade algorithm to compute the optical flow, the template image $T(x)$ is a sub-region of the frame at time $t$, and the input image $I(x)$ is the frame at $t+1$. Let $W(x; p)$ be the parameterized set of warps, where $p = (p_1, ..., p_n)^T$ is a vector of parameters. The warp $W(x; p)$ takes the pixels $x$ in the coordinate frame of template $T$ and maps it to the sub-pixel location in the coordinate frame of the image $I$. If we are tracking a large image patch moving in a 3D way we can consider the set of affine warps:
where there are six parameters \( p = (p_1, p_2, p_3, p_4, p_5, p_6)^T \). In general, \( n \) can be arbitrarily large and \( W(x; p) \) can be arbitrarily complex.

### 2.2 The Cost Function

The goal of the Lucas-Kanade algorithm is to minimize the sum of squared error between two images: template \( T \) and image \( I \) warped back onto the coordinate frame of the template:

\[
\sum_x [I(W(x; p)) - T(x)]^2
\]

To estimate Eq. (2.2), the Lucas-Kanade algorithm assumes that a current estimate of \( p \) is known and then iteratively solves for increments to the parameters \( \Delta p \), that is,

\[
\sum_x [I(W(x; p + \Delta p)) - T(x)]^2
\]

is optimized with respect to \( \Delta p \), and then the parameters are updated:

\[ p \leftarrow p + \Delta p \]  

The two steps are iterated until the estimation of the parameters \( p \) converge to a defined threshold \( ||\Delta p|| \leq \epsilon \).
2.3 Derivation of The Lucas-Kanade Algorithm

Do a first-order Taylor expansion on Eq. (2.3):

$$\sum_x \left[ I(W(x; p)) + \nabla I \frac{\partial W}{\partial p} - T(x) \right]^2$$  \hspace{1cm} (2.5)

where $\nabla I = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})$ is the gradient of image $I$ evaluated at $W(x; p)$. And $\frac{\partial W}{\partial p}$ is the Jacobian of the warp. If $W(x; p) = (W_x(x; p), W_y(x; p))^T$, then

$$\frac{\partial W}{\partial p} = \begin{pmatrix} \frac{\partial W_x}{\partial p_1} & \frac{\partial W_x}{\partial p_2} & \cdots & \frac{\partial W_x}{\partial p_n} \\ \frac{\partial W_y}{\partial p_1} & \frac{\partial W_y}{\partial p_2} & \cdots & \frac{\partial W_y}{\partial p_n} \end{pmatrix}$$  \hspace{1cm} (2.6)

For example, the affine warp in Eq. (2.1) has the Jacobian:

$$\frac{\partial W}{\partial p} = \begin{pmatrix} x & 0 & y & 0 & 1 & 0 \\ 0 & x & 0 & y & 0 & 1 \end{pmatrix}$$  \hspace{1cm} (2.7)

The partial derivative of the equation in Eq. (2.5) with respect to $\Delta p$ is:

$$2 \sum_x \left[ \nabla I \frac{\partial W}{\partial p} \right]^T \left[ I(W(x; p)) + \nabla I \frac{\partial W}{\partial p} - T(x) \right]$$  \hspace{1cm} (2.8)

where we refer to $\nabla I \frac{\partial W}{\partial p}$ as the steepest descent images.

Setting Eq. (2.8) equal to 0 and solving it gives the closed form solution for the minimum of the expression as:

$$\Delta p = H^{-1} \sum_x \left[ \nabla I \frac{\partial W}{\partial p} \right]^T \left[ T(x) - I(W(x; p)) \right]$$  \hspace{1cm} (2.9)

where $H$ is the $n \times n$ Gauss-Newton approximation to the Hessian matrix:

$$H = \sum_x \left[ \nabla I \frac{\partial W}{\partial p} \right]^T \left[ \nabla I \frac{\partial W}{\partial p} \right]$$  \hspace{1cm} (2.10)

We call $\sum_x \left[ \nabla I \frac{\partial W}{\partial p} \right]^T \left[ T(x) - I(W(x; p)) \right]$ the steepest descent parameter updates. The Lucas-Kanade algorithm then consists of iteratively applying Eq. (2.9) and Eq. (2.4).
In summary, the Lucas-Kanade algorithm is as follows:

Algorithm 1 The Lucas-Kanade Algorithm

repeat
1. Warp $I$ with $W(x; p)$ to compute $I(W(x; p))$
2. Compute the error image $T(x) - I(W(x; p))$
3. Warp the gradient $\nabla I$ with $W(x; p)$
4. Evaluate the Jacobian $\frac{\partial W}{\partial p}$ at $(x; p)$
5. Compute the steepest descent images $\nabla I \frac{\partial W}{\partial p}$
6. Compute the Hessian matrix using Eq. (2.10)
7. Compute $\sum_x [\nabla I \frac{\partial W}{\partial p}]^T [T(x) - I(W(x; p))]$
8. Compute $\Delta p$ using Eq. (2.9)
9. Update the parameters $p \leftarrow p + \Delta p$
until $||\Delta p|| \leq \epsilon$

2.4 Requirements on the Set of Warps

The only requirement on the warps $W(x; p)$ is that they are differentiable with respect to the warp parameters $p$. The condition is required to compute $\frac{\partial W}{\partial p}$.

2.5 Summary

In this chapter, we have shown the Lucas-Kanade algorithm. Its application to optical flow will become the foundation of this work: both the space-time interest point detector and the descriptor will be based on the computed optical flow.
Chapter 3

Finding the Region of Interest

When processing videos, one important task is to find the region of interest (ROI). In this chapter we show an example of extracting interesting part of a self-made disassembling video based on ROI. The methods suits only for this specific scenario: static camera, one part is removed at a time.

3.1 Detecting Changes in the Scene

Extracting interesting parts of the video is important in processing. Today, high-resolution cameras are widely used. The typical HD video $(1920 \times 1080)$ has more than two million pixels per frame. Processing such kind of videos can be extremely computationally expensive. So instead of directly processing the whole video, we can first do pre-processing and find the region that is of interest. Usually the interesting part in a video is the part that is moving. But sometimes we don’t care how that part is moving. Instead, we care more about the result of the changes. If we can first find a region and observe only that region, sometimes we can get as much important information as observing the whole scene.

Figure 3-1, 3-2 and 3-3 show three frames of a video, in which a part is removed from the Samsung phone using a screw driver. Note that the camera is static. This will be our video for testing. We want to detect what has changed through the process.
If one compares the first few frames with the last few frames, the change should be very obvious: only one part is removed from the phone and nothing else changes. Whatever happens during the process does not matter very much. For example, a person may use a screwdriver or hand to remove the part. The person might have to try several times to successfully remove the part. During the process, the screwdriver might accidentally drop and the person picked it up. All of these details are not as important as the fact that the particular part got removed in the video.

This observation is important in robot learning from videos. Because ultimately, the robot will want to reproduce the result of the changes in a video, not the process. In fact, for phone disassembly task, the robot is unlikely to reproduce what exactly a human is doing. By focusing on the major changes rather than the less important ones, the robot is more likely to successfully reproduce the results in a video.

Figure 3-1: The first frame of the video.

To detect what has changed during this video, we subtract the last frame from the first frame. The result is shown in Fig. 3-4.
3.2 Noise Cancellation

Now we can clearly see the removed part in Fig. 3-4. But due to the noise in camera and illumination changes, there is a lot of noise in the resulting image. We want to eliminate the noise so that only the major changes are remaining. So we do global image thresholding using Otsu's method, which chooses the threshold to minimize the
intraclass variance of the black and white pixels. After getting the global threshold, the image is binarized \(^1\), as shown in Fig. 3-5.

Observing Fig. 3-5, we can see the noise more clearly than before, as now all pixels shown in the binarized image have the value of either 0 or 1. Simply doing subtraction won’t work, because although what has been removed is clearly shown, the noise can interfere with the observation. So we need to get rid of the noise.

Since the clusters of noise pixels are usually much smaller than the cluster of pixels where real change lies in, we can get rid of the noise by deleting those clusters whose number of pixels are below a threshold. Fig. 3-6 shows the result of this operation. Now the noise is gone and only the changed part is remaining.

Because of the previous step for deleting noise, some pixels of the removed part are gone. We want to get them back so that later on we can create a mask and observe the whole changed part. So we dilate the binarized image as shown in Fig. 3-7.

The obtained binarized image can serve as a mask. When we apply such a mask to the original video, it will suppress all of the irrelevant parts of the video and only

\(^1\)Changing pixels to either 0 or 1.
Figure 3-5: First using Otsu’s method to find the global threshold, then binarize the image.

Figure 3-6: Remove small objects in the binary image. Note the noise in Fig. 3-5 disappeared.

preserve the changed part. One frame of the masked video is shown in Fig. 3-8.

After getting the masked video, we can do analysis on it, instead of on the original video, which can help save a lot of computational resources.

Figure 3-9 shows one frame of the video processed using Laptev’s method on
detecting space-time interest points [15], which we will discuss in detail in the next chapter.

3.3 Conclusion

In this chapter we showed how to use background subtraction to find the region of interest. We applied the mask to the original video and extracted only a fraction of the original video, thus significantly reducing the data that we need to process. In the next chapter, we will continue to explore the idea of spatio-temporal interest point, and develop a new way of detecting such kind of points.
Figure 3-8: One frame of the masked video.

Figure 3-9: One frame of the processed video. The yellow circle shows the detected interest point, which in this case is the contact point between the screwdriver and the part that is being removed.
Chapter 4

Interest Points Detection

In this chapter we show how to detect keypoints in the 3D space-time domain. In section 4.1 we take a look of the pre-existing methods in detecting spatio-temporal interest points. In section 4.2 we show how to detect space-time points in optical flow images using the difference-of-gaussian function.

4.1 An Introduction to Space-time Interest Points

Local image features or interest points provide compact and abstract representations of patterns in an image. If we extend the notion of spatial interest points into the spatio-temporal domain, the resulting features often reflect interesting events that can be used for compact representation of video data as well as for interpretation of spatio-temporal events.

The importance of space-time interest points in the application of phone disassembly is twofold. First, such points provide potential candidate locations for the robot to apply its tools. In the case of prying, for example, the prying point is often detected as one of the space-time interest points. In certain scenarios, the prying point can be detected as the only space-time interest point. Thus detecting space-time interest points in a prying video may give potential prying locations. There are many uncertainties when using this method to detect candidate locations: maybe the right point is not detected; or maybe no point is detected. But at least this approach
Figure 4-1: Result of detecting the strongest spatio-temporal interest points in a football sequence with a player heading the ball (a) and in a hand clapping sequence (b). From the temporal slices of space-time volumes shown here, it is evident that the detected events correspond to neighborhoods with high spatio-temporal variation in the image data or 'space-time corners'. Figure from [15].

gives a possibility.

Second, since the detected space-time interest points can be used for compact representation of video data, we can recognize what’s happening in a video by observing the detected points. In phone disassembly, in order to imitate the actions in a video, knowing what kinds of actions are there in the video can be important. Chapter 4, 5 and 6 together show the steps for action recognition in a video.

The space-time interest point does not have a unified definition, different people have different interpretations of what it is and how it should be detected. In the earliest work on this topic, Laptev argues that many interesting events in video are characterized by strong variations in the data along both the spatial and the temporal dimensions. For example, consider a football sequence with a player heading the ball and hand clapping, as shown in Fig. 4-1. More generally, points with non-constant motion correspond to accelerating local image structures that may correspond to accelerating objects in the world. Hence, such points can be expected to have information about the forces acting in the physical environment and changing its structure [15].

Dollar et al. apply 1D Gabor filters temporally to the Gaussian smoothed images. The detector is tuned to fire whenever there are periodic frequency components in local image intensities. It seems unreasonable to assume that only periodic motions
are interesting. Periodic motions like a bird flapping its wings will induce strong responses. But this detector can also respond strongly to a range of other motions, including at spatio-temporal corners. Generally speaking, any region with spatially distinguishing characteristics undergoing a complex motion will induce a strong response [11].

Willems et al. localize features both in the spatio-temporal domain and over both scales simultaneously using the determinant of the Hessian as saliency measure. They create an efficient implementation of the detector by approximating all 3D convolutions using box filters [23].

Ke et al. build on the concept of integral video to achieve realtime processing of video data. Rather than relying on interest points, they use dense spatio-temporal Haar-wavelets computed on optical flow. Discriminative features are selected during a training stage. The resulted application dependent features are not scale-invariant [14].

4.2 A Detector Working on Optical Flow

Previous works on spatio-temporal interest point detector either extend the pre-existing 2D feature detector to the temporal domain [15], or first use a 2D feature detector to detect keypoints on spatial domain and then prune the detected points based on temporal information [9, 8]. The purpose of pruning is to get rid of the static points detected by a spatial detector. The extension of the spatial feature detector to the temporal domain can only work when there are sudden changes in velocity, like when a ball is hitting a player’s head, the contacting point will be detected. But it is hard for this method to detect other kinds of points. Dollar’s method of using 1D Gabor filter to detect periodic frequency components works fine but is not very intuitive. While recent works that detecting on spatial domain and pruning across frames seem like not utilizing temporal information well enough.

In this section, we try to give a method that is detecting points of interest directly in the flow domain. When doing feature detection, optical flow is often used in the
descriptor, but rarely in the detector. Yet the flow image can give rich information on temporal information: each pixel’s motion is estimated, so temporal information about velocity between consecutive frames is given. Figure 4-3 shows the computed optical flow for two adjacent frames of a video shown in Fig. 4-2. Our goal is to discover unique local flow patterns to represent the whole video. We adapt Lowe’s method of detecting local extrema using difference-of-Gaussian function to the flow images and obtain the features of interest.

Figure 4-2: Two successive frames from a walking video, which is part of Weizmann Human Activity Video dataset [13].

Figure 4-3: The computed optical flow for two consecutive frames in Fig. 4-2.
4.2.1 The Goal of the Detector

We would like the detected features to be invariant to noise, scaling, rotation and minor changes in viewing direction. In this section we will address the problem of scale invariance and we will consider rotation invariance in the descriptor.

But first let's discuss what scale invariance means. For a 2D image, scale invariance means that whenever we zoom in or zoom out, as long as the patterns are similar, the detector might as well pick up the patterns and later on the descriptor will recognize that they are the same pattern. Scale invariance of the flow patterns, however, means regardless of the magnitudes of velocities, as long as the flow patterns are desired, the detector should be able to pick them up. Technically speaking, we want to detect local flow features whose characteristics can be determined by observing local values in a predefined way.

4.2.2 Computing Difference-of-Gaussian Flow Images

The first step of keypoint detection is to find locations and its scales that can be repeatably assigned under different views of the same moving object. The scale space function can be used to detect locations that are invariant to scale change of the flow image. To get the scale space of the flow image, the flow image is first split into two channels: one is the horizontal component $F_h(x, y)$, the other the vertical component $F_v(x, y)$. This will make it easier for convolution. The two channels will then be combined. As an abuse of terminology, we also call the two components flow images. The scale space of the original flow image will be two functions, $L_h(x, y, \sigma)$ and $L_v(x, y, \sigma)$, that are produced from the convolution of the Gaussian function, $G(x, y, \sigma)$, with the two flow channels:

$$L_h(x, y, \sigma) = G(x, y, \sigma) * F_h(x, y)$$
$$L_v(x, y, \sigma) = G(x, y, \sigma) * F_v(x, y)$$

(4.1)
where \(*\) is the convolution operation, and

\[
G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}
\]  

(4.2)

To efficiently detect stable keypoint locations in scale space, we use scale-space extrema in the difference-of-Gaussian functions convolved with the flow images, \(D_h\) and \(D_v\). \(D_h\) and \(D_v\) can be computed from the difference of two nearby scales separated by a constant multiplicative factor \(k\):

\[
D_h(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) \ast F_h(x, y) = L_h(x, y, k\sigma) - L_h(x, y, \sigma)
\]

\[
D_v(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) \ast F_v(x, y) = L_v(x, y, k\sigma) - L_v(x, y, \sigma)
\]

(4.3)

The difference-of-Gaussian function is very efficient to compute, since the Gaussian smoothed flow images \(L_h\) and \(L_v\) need to be computed in any case for scale space feature description, \(D_h\) and \(D_v\) are computed by simple subtraction. It is in essence a close approximation to the scale-normalized Laplacian of Gaussian, \(\sigma^2 \nabla^2 G\). Lindeberg showed that the normalization of the Laplacian with factor \(\sigma^2\) is required for true scale invariance [16].

To understand the relationship between \(D\) and \(\sigma^2 \nabla^2 G\), from the heat diffusion equation (in terms of \(\sigma\)):

\[
\frac{\partial G}{\partial \sigma} = \sigma \nabla^2 G
\]

(4.4)

From the above equation, we see \(\nabla^2 G\) can be yielded from finite difference approximation to \(\partial G/\partial \sigma\), by computing the difference of nearby scales at \(k\sigma\) and \(\sigma\):

\[
\sigma \nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma}
\]

(4.5)

therefore,

\[
G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2 \nabla^2 G
\]

(4.6)

This shows that the \(\sigma^2\) scale normalization for the scale-invariant Laplacian is
incorporated by the difference-of-Gaussian function when it has scales differing by a constant factor.

Figure 4-4: For each octave of scale space, the initial two channels of the flow image are repeatedly convolved with Gaussians to produce the set of scale space images shown on the left (only one channel is shown). Adjacent Gaussian smoothed flow images are then subtracted to produce the difference-of-Gaussian images on the right. After each octave, the Gaussian smoothed flow image is down-sampled by a factor of 2. Figure from [17].

The difference-of-Gaussians $D_h(x, y, \sigma)$ and $D_v(x, y, \sigma)$ can be constructed as shown in Fig. 4-4. The initial flow images are incrementally convolved with Gaussians to produce fields separated by a constant factor $k$ in scale space, as shown in the left column. Each octave of scale space (doubling of $\sigma$) is divided into $s$ intervals, where $s$ is an integer. For each flow channel, we must produce $s + 3$ flow images in the stack of blurred fields for each octave. This is because $s + 3$ blurred flow images generate $s + 2$ difference-of-Gaussian flow images, plus two flow images are needed (the highest and lowest scales of the octave) for extrema detection. In implementation we choose $s = 2$. Difference-of-Gaussian flow images can be yielded by subtracting adjacent flow image scales, as shown on the right. Once a complete octave has been processed, we resample the Gaussian flow image that has twice the initial value of $\sigma$ (2 images from the top of the stack) by taking every other pixel in each row and
Computation can be greatly reduced by sampling. Fig. 4-5 shows the constructed Gaussian pyramids from the original flow image shown in Fig. 4-3. Fig. 4-6 shows the corresponding difference-of-Gaussian flow images. Since two channels of flow images are being processed, we combine them by taking root sum squared and ultimately produce \( D(x, y, \sigma) \):

\[
D(x, y, \sigma) = \sqrt{D^2_h(x, y, \sigma) + D^2_v(x, y, \sigma)}
\]  

\[(4.7)\]

(a) The scale space construction of horizontal flow image.

(b) The scale space construction of vertical flow image.

Figure 4-5: The constructed scale space for the flow image shown in Fig. 4-3. The original flow image is split into two channel and computed separately. Each row of images corresponds to one octave. For every row, from left to right, the original flow image is convolved with Gaussian kernels whose variances are becoming larger and larger (spaced by \( k \)). Therefore, as can be observed, the flow images become more and more blurred going from left to right. Going from up to down, since the flow images are subsampled after each octave, the images are smaller and smaller. The whiter the pixel, the higher is the value, and the larger is the motion.

### 4.2.3 Local Extrema Detection

Local extrema are points that have largest or smallest values compared with its neighbors. Such points are preferred in the detection procedure because they are unique among its neighbors and can be repeatably detected. To detect the local extrema of
Figure 4-6: The difference-of-Gaussian flow images for the two channels. These images are obtained by subtracting every two smoothed flow images that are in adjacent scale. For each row (octave), there are 5 Gaussian-smoothed flow images in Fig. 4-5, so after subtraction there are 4 difference-of-Gaussian flow images each row (octave). The whiter the pixel, the larger is the difference between two adjacent scales in that pixel. From the figure, the contour of the shape of the person is usually significantly whiter, meaning higher difference.

\[ D(x, y, \sigma), \] each sample point is compared to its eight neighbors in the current flow image and nine neighbors in the scales above and below, as shown in Fig. 4-7. It will be selected only if the value of the sample point is larger or smaller than all of its neighbors. Since most of the sample points will be eliminated following the first few checks, the cost of the computation is reasonably low.

There is no minimum spacing of samples that will detect all extrema, since the extrema can be arbitrarily close together. To determine the frequency of sampling in the image and scale domains that is needed to reliably detect the extrema, Lowe did experiments on feature matching using a collection of real images drawn from a diverse range. The result is shown in Fig. 4-8. As shown by Fig. 4-8b, more and more points are detected as the number of scales sampled increases. Figure 4-8a shows that
Figure 4-7: Extrema of the difference-of-Gaussian flow images are detected by comparing a pixel (marked by ×) with its 26 neighbors in $3 \times 3$ regions at the current and adjacent scales (marked with circles). Figure from [17].
repeatability does not continue to improve as more scales are sampled. The reason of that is as more scales are sampled, the extra local extrema detected are on average less stable and therefore less likely to be detected in the transformed image.

Figure 4-9 shows an experimental determination of the amount of prior smoothing, $\sigma$, that is applied to each image level before building the scale space representation for an octave. From the result, we see that the repeatability continues to increase with $\sigma$. But there is a cost in terms of efficiency using a large $\sigma$. The author chooses to use $\sigma = 1.6$. In our implementation, we also use this value.

Pre-smoothing of the flow images before extrema detection means we are effectively discarding the highest spatial frequencies. To make full use of the input, the size of the input flow image was doubled using linear interpolation before building the first level of the pyramid. This image doubling technique leads to a very efficient implementation and can increase the number of stable keypoints by a factor of 4.
Figure 4-9: The solid line in the graph shows the percent of keypoint locations that are repeatably detected in a transformed image as a function of the prior image smoothing for the first level of each octave. The dotted line shows the percent of descriptors correctly matched against a large database. Figure from [17].

4.2.4 Rejecting Low Contrast and Edge Responses

Once a keypoint has been found by local extrema detection, a detailed fit to the nearby data for location, scale and ratio of principal curvatures needs to be done so that points with low contrast or are poorly located on an edge will be rejected. Brown and Lowe developed a method for fitting a 3D quadratic function to the local sample points to determine the interpolated location of the maximum. Using Taylor’s expansion (up to the quadratic terms) of the scale space function, \( D(x, y, \sigma) \), shifted so that the origin is at the sample point:

\[
D(x) = D + \frac{\partial D^T}{\partial x} + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x \tag{4.8}
\]

where \( D \) and its derivatives are evaluated at the sample point and \( x = (x, y, \sigma)^T \) is the offset from this point. The extremum \( \hat{x} \) can be found by taking the derivative of the function with respect to \( x \) and setting it to zero:

\[
\hat{x} = -\frac{\partial^2 D^{-1} \partial D}{\partial x} \tag{4.9}
\]
The Hessian and derivative of $D$ are approximated by computing differences of neighboring sample points. If the offset $\hat{x}$ is larger than 0.5 in any dimension, it means the extremum is closer to a different sample point. In this case, it will be changed and interpolation performed about the changed point. The final offset $\hat{x}$ will be added to the location of the sample point to get the estimation for the location of the extremum.

The function value at the extremum, $D(\hat{x})$, can be used for rejecting unstable extrema with low contrast. Substituting Eq. (4.9) into Eq. (4.8), we can get:

$$D(\hat{x}) = D + \frac{1}{2} \frac{\partial D^T}{\partial x} \hat{x}$$ (4.10)

For the experiments in this thesis, two values, 0.03 and 0.15, are set as the threshold for $D(\hat{x})$.

Since the difference-of-Gaussian function has a strong response along edges, it is unstable to small amounts of noise. Lowe observed that a poorly detected extremum will have a large principal curvature across the edge but a small one in the perpen-
Figure 4-12: The remaining points after pruning using contrast thresholding. The contrast threshold is set to be 0.03.

Figure 4-13: The remaining points after pruning using edge response thresholding. The ratio $r$ is set to be 10.

Figure 4-14: The remaining points after pruning using contrast thresholding. The contrast threshold is set to be 0.15.
Figure 4-15: The remaining points after pruning using edge response thresholding. The ratio $r$ is set to be 10.

dicular direction. The principal curvature can be computed at the location and scale of the keypoint by computing a $2 \times 2$ Hessian matrix, $H$:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$ (4.11)

The derivatives can be estimated by taking differences of neighboring sample points.

The eigenvalues of $H$ are proportional to the principal curvatures of $D$. Let $\alpha$ be the eigenvalue with the larger magnitude and $\beta$ with the smaller one, the sum of the eigenvalues and the product can be determined as:

$$\text{Tr}(H) = D_{xx} + D_{yy} = \alpha + \beta$$
$$\text{Det}(H) = D_{xx}D_{yy} - D_{xy}^2 = \alpha\beta$$ (4.12)

where ‘Tr’ is the trace of $H$ and ‘Det’ is the determinant of $H$. When the determinant is negative, the curvatures have different signs so the point is discarded as not being an extremum. Let $r$ be the ratio between $\alpha$ and $\beta$, so that $\alpha = r\beta$. Since

$$\frac{\text{Tr}(H)^2}{\text{Det}(H)} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r + 1)^2}{r}$$ (4.13)

depends only on $r$, to check the ratio of principal curvatures is below some threshold,
\[ \frac{\text{Tr}(H)^2}{\text{Det}(H)} < \frac{(r+1)^2}{r} \] (4.14)

Our implementation uses \( r = 10 \).

Figure 4-11 shows all of the points that are detected by comparing the sample points with their neighbors. As can be seen, without pruning, there will be a lot of points that are irrelevant. When the contrast threshold is set to be 0.03, the result is shown in Fig. 4-12. Although most of the irrelevant points are gone by now, there are still some outliers. When we set the contrast threshold to be 0.15, as shown in Fig. 4-14, only several keypoints are remaining, but most of them are exactly on the moving part. As for rejecting edge responses, from Fig. 4-12 to Fig. 4-13, we can see this procedure helps a little; yet from Fig. 4-14 to Fig. 4-15, no sample points are rejected. In this case, it's because the remaining points after contrast thresholding pruning are already so few, and coincidentally all of them are not recognized as edge responses. But overall, we can see that edge responses pruning does not work as well in the flow images as in the original images.

4.3 Summary

Inspired by Lowe's work, we designed a new STIP detector that works on the optical flow images. To make this detector scale invariant, we apply difference-of-Gaussian function and detect extrema in scale space. In order to improve stability, we prune the detected sample points using contrast and edge response thresholding. We did experiment on tuning the parameters, and discover that a too low contrast threshold results in too much noise. Now we have the interest point detector, we hope to get more information out of the video based on this detector. In the next chapter, we will construct descriptors for the detected keypoints.
Chapter 5

The Descriptor for Spatio-temporal Interest Points

A descriptor is used to describe an interesting point so that later on one can retrieve the keypoint in a large database. In this chapter we show how to use a descriptor to describe the local flow image region that is highly distinctive yet is as invariant as possible to variations in rotation and changes in 3D viewpoint. To compute the descriptor for a keypoint, first the flow image magnitudes and orientations are sampled around the keypoint location, using the scale of the keypoint to select the level of Gaussian blur for the image. To achieve orientation invariance, the flow orientations and the coordinates of the descriptor are rotated relative to the keypoint orientation.

5.1 Orientation Assignment

By assigning consistent orientation to each keypoint based on local flow image properties, the descriptor can be computed relative to this orientation and therefore rotation invariance can be achieved. Since we are working on the optical flow field, a natural choice would be to build a descriptor based on the local flow field surrounding a keypoint.

Rotation invariance in the flow domain requires tolerance of rotational changes of the local optical flow patterns. To illustrate this, Fig. 5-1 shows two patterns. The
second pattern is obtained by rotating the first one. Our goal is to be able to detect such similar patterns as essentially the same.

![Diagram of rotation invariance of flow pattern](image)

Figure 5-1: An illustration of the rotation invariance of the flow pattern. Suppose the two local flow domains are surrounding the same keypoint, i.e., nothing is changed from the left flow image to the right flow image except it is rotated, the descriptor should be able to recognize they are the same pattern.

The scale of the keypoint is used to select a Gaussian smoothed flow image, which is composed by two channels: $L_h(x, y, \sigma)$ and $L_v(x, y, \sigma)$. Once the Gaussian smoothed flow image is selected, all of the following computations will be performed in a scale-invariant manner because of this selection. For each flow image sample, $L_h$ and $L_v$, at this scale, the flow magnitude, $m(x, y)$, and orientation, $\theta(x, y)$, are computed as follows:

$$
m(x, y) = \sqrt{L_h(x, y, \sigma)^2 + L_v(x, y, \sigma)^2}$$
$$
\theta(x, y) = \tan^{-1}(L_v(x, y, \sigma)/L_h(x, y, \sigma)) \tag{5.1}
$$

An orientation histogram will be formed from the flow orientations of sample points within a region around the keypoint. The orientation histogram has 36 bins covering the 360 degree range of orientations. Each sample added to the histogram is weighted by its flow vector magnitude and by a Gaussian-weighted circular window with a $\sigma$ that is 1.5 times of the scale of the keypoint.

The peaks in the orientation histogram correspond to dominant directions of local flow patterns. The highest peak is selected and any other peak that is within 80%
of the highest peak is also used to create a keypoint with that orientation. So for locations with multiple peaks of similar magnitude, multiple keypoints will be created at the same location and scale with different orientations. Finally, a parabola is fit to the 3 histogram values closest to each peak to interpolate the peak position for better accuracy.

5.2 The Local Flow Image Descriptor

In Fig. 5-2, the flow vectors are illustrated by small arrows at each sample location on the left side. The computed descriptor is shown on the right side.

![Flow field and Keypoint descriptor](image)

Figure 5-2: A keypoint descriptor is created by first computing the flow magnitude and orientation at each image sample point in a region around the keypoint location, as shown on the left. These are weighted by a Gaussian window, indicated by the overlaid circle. These samples are then accumulated into orientation histograms summarizing the contents over $4 \times 4$ subregions, as shown on the right, with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This figure shows a $2 \times 2$ descriptor array computed from an $8 \times 8$ set of samples, whereas the experiments in this work use $4 \times 4$ descriptors computed from a $16 \times 16$ sample array. Modified from [17].

A Gaussian function with $\sigma$ being one half of the descriptor window is adopted to assign a weight to the magnitude of each sample point. The circular window on the left side of Fig. 5-2 shows this Gaussian function. The purpose of doing this is to give less emphasis to the flow vectors that are far from the center of the descriptor.
window, and also to avoid sudden changes in the descriptor with small changes in the position of the window.

The keypoint descriptor is shown on the right side of Fig. 5-2. By creating orientation histograms over $4 \times 4$ sample regions, it allows for significant shift in flow positions. There are eight directions for each orientation histogram, with the length of each arrow corresponding to the magnitude of the histogram entry. Even a flow pattern on the left shifts up to 4 sample positions, it can still contribute to the same histogram on the right, so that the objective of allowing for larger local positional shifts can be achieved.

To avoid boundary effects, trilinear interpolation is used to distribute the value of each sample into adjacent histogram bins. Boundary effects can happen when the descriptor suddenly changes as a sample smoothly shifts from being in one bin to another or from one orientation to another. Concretely, each entry into a bin is multiplied by a weight of $1 - d$ for each dimension, where $d$ is the distance of the sample from the central value of the bin measured in units of the histogram bin spacing.

The histogram is constructed as a vector including the values of all the orientation histogram entries, corresponding to the lengths of the arrows on the right side of Fig. 5-2. For illustration, the figure only shows a $2 \times 2$ array of orientation histograms, while in our implementation we use a $4 \times 4$ array of histograms with 8 orientation bins in each. So each keypoint has a histogram of $4 \times 4 \times 8 = 128$ bins.

Finally, to make the histograms more stable to match, the vector is normalized to unit length. This can help reduce the effect of overall magnitude changes. Following the SIFT descriptor, we also reduce the influence of large flow magnitudes by thresholding the values in the unit feature vector to each be no larger than 0.2.

5.3 A Note of Combining Several Descriptors

Some argue that the histogram of optical flow alone is not good enough to form a descriptor. In fact, papers like [9] are using more than one descriptor like histogram of
gradient and histogram of optical flow. Indeed, combining several descriptors can give us better results. But the purpose of the descriptor is to connect with the previously designed detector. Since the detector is detecting points on flow images, using our modified version of histogram of optical flow should give the max consistency between the detector and descriptor. For the simplicity of implementation, we are not using any other kind of descriptors.

5.4 Summary

We present a modified version of histogram of optical flow (HOF) as our descriptor. Unlike previous works that are extracting features around the statically detected keypoints only on the flow image and ignoring the requirements of scale and rotation invariance, our work aims to tackle the challenge of scale and rotation invariance on the flow domain. In the design of the descriptor, we also consider the problem of robustness and large magnitude, and solve these problems by normalization and thresholding.
Chapter 6

Video Classification

To test the detector and descriptor, we do video classification on a dataset and show the results. First we use $k$-means algorithm to cluster the descriptors in the training set and get $n$ clusters. For training, we try to assign each descriptor to the cluster that is closest. Based on the assignment, we build a histogram for every video, which will serve as the descriptor for that video. We then use $k$-nearest neighbors ($k$-NN) classifier to do classification and do testing using cross validation.

6.1 Dataset

The Weizmann Human Activity Video Dataset is used for classification [13]. This dataset has 10 categories with each 9 videos, so there are in total 90 videos. For training and testing, we divide this dataset into three sets, each containing 10 categories with each 3 videos. We test the performance of behavior recognition using $n$-fold cross validation. In this case we have 2 sets for training and 1 set for testing, so there are in total 3 training and testing cases. The ten categories of the dataset are shown in Fig. 6-1.
6.2 Clustering

We create a library of descriptor prototypes by clustering a large number of descriptors extracted from the training data using $k$-means algorithm. $k$-means clustering is a type of unsupervised learning, which is suitable for us because the types of the descriptors are unlabeled. The goal of this algorithm is to find groups in the data, with the number of groups being $k$. Based on the features that are provided, the algorithm works iteratively to assign each descriptor to one of the $k$ groups. The descriptors will be grouped based on feature similarity. It will return the labels for the training data and the centroids of the $k$ clusters. The parameter $k$ is given by experience. Silhouette is a method of interpretation and validation of consistency within clusters of data. It provides a succinct graphical representation of how well each object lies within its clusters. Fig. 6-2 shows the silhouette plot of the clusters, using all of the extracted point descriptors in the dataset, when setting $k = 10$.

6.3 Classification

After detecting keypoints and assigning their corresponding descriptors, we want to go one step further and try to classify the videos based on their descriptors. The descriptors of a video can be thought of having already conserved the essential infor-
Figure 6-2: Silhouette plot of the divided clusters. When using \( k \)-means clustering we set \( k = 10 \). Most of the components are within the corresponding clusters, though clusters 3, 4, 5, 6, 7, 8, 9 all have a few outliers.

...
<table>
<thead>
<tr>
<th>Clusters</th>
<th>50</th>
<th>100</th>
<th>200</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Rate</td>
<td>0.1556</td>
<td>0.1778</td>
<td>0.1667</td>
<td>0.1444</td>
<td>0.1333</td>
</tr>
</tbody>
</table>

Table 6.1: Change of classification error rate as the number of clusters changes. The classifier being used is 1-nearest neighbor classifier.

Depending on how many clusters we have, usually it takes less than 2 minutes for each iteration. We use Piotr’s Computer Vision Matlab Toolbox and his video classification framework for implementation [10, 11].

Fig. 6-3 shows the confusion matrix of the classification results. Table 6.1 shows the changes of error rate when different numbers of clusters are being used. Too many clusters can increase the computation time. In this case, choosing $K = 500$ is good enough.

![Confusion Matrix](image)

Figure 6-3: The confusion matrix computed by using $n$-fold cross validation. For the divided 3 sets, we train on every two sets and test on the other. One simplification in implementation is that clustering is done only once, using all of the data.

6.4 Summary

We test the detector and descriptor on the Weizmann action dataset. We use $k$-means to get the clusters and $k$-nearest neighbors to train the classifier. We tested
the performance using cross validation and get fine results. For this small dataset, the training and testing cases are only so few. We believe using a larger dataset and better classifier, better result can be yielded.
Chapter 7

Conclusion and Recommendations

7.1 Conclusion

The aim of this work is to try to get enough information from the video for the robot to learn. Specifically, for the phone disassembly task, we are interested in finding a region where things have changed, a point for the robot to act on, and the kind of action in a given video. To this end, we got the region of interest by background subtraction and created a mask for the video, which could significantly reduce the amount of data we need to process. We explored the idea of space-time interest point and created a new STIP detector working on the optical flow images. We made changes to the histogram of optical flow descriptor and made it invariant to rotation and noise. Finally, video classification was done based on the clustered descriptors, using $k$-means and $k$-NN.

7.2 Recommendations

An immediate future work is to build our own dataset specifically for the phone disassembly task. The dataset should include several different tasks like prying, pulling, sliding, etc., so that when given an input video, the system can recognize the task and change tools and do path planning accordingly. Also, ultimately we want the system to work as a whole, meaning that given an input video, the robot should mimic the
person in a video and do the same task using automatically chosen tools. So a lot of work should still be done to integrate the system.

Model reconstruction and tracking can help us to extract more information from the video. In the phone disassembly project, if we can reconstruct the rough shape of the moving part and track how it is moved, a lot of important information can be obtained. The model reconstruction and tracking can be complementary to each other and serve as mutual improvements.

Combining external knowledge with video processing might also help a lot in augmenting the quality of the output information. One possible approach is to incorporate semantic image segmentation. If we can know not only what has been removed from the phone in a video, but also its properties by resorting to external knowledge database, there should be a huge improvement for the performance of the system. An example is that if one is removing a camera from the phone in a video, and if we have the knowledge of the properties of a camera: easy to break, cannot withstand pressure, etc., we might infer from that knowledge the right way to disassemble the camera.
Appendix A

MATLAB Code for Generating Masked Videos

```matlab
clear; close all; clc
fontSize = 22;
videoFile = './videos/frames3.avi';
videoObj = VideoReader(videoFile);
vidHeight = videoObj.HEIGHT;
vidWidth = videoObj.WIDTH;
s = struct('cdata', zeros(vidHeight, vidWidth, 3, 'uint8'), ... 'colormap', []);
k = 1;
while hasFrame(videoObj)
    s(k).cdata = rgb2gray(readFrame(videoObj));
    k = k + 1;
end
k = k - 1;
sz = size(s);
diff = s(1).cdata - s(k).cdata; % diff is the difference between ...
    the first and last frame
```
%% Thresholding and binarizing
level = graythresh(diff);
BW = imbinarize(diff, level);
imshow(BW)

%% Get rid of the noise
shrink = bwareaopen(BW, 20, 8);
imshow(shrink)

%% Image dilate
se = strel('rectangle', [20, 20]);
J = imdilate(shrink, se, 'same');
imshow(J)

%% Nail the region
[row, col] = find(J);
topLeftRow = min(row);
topLeftCol = min(col);
bottomRightRow = max(row);
bottomRightCol = max(col);

%% Use the mask to get the ROI of the video
d = s;
shrinkedS = struct('cdata', zeros(bottomRightRow - topLeftRow, ...
   bottomRightCol - topLeftCol, 1, 'uint8'), 'colormap', []);

for i = 1:k
   d(i).cdata = uint8(double(s(i).cdata) .* J);
   shrinkedS(i).cdata = d(i).cdata(topLeftRow:bottomRightRow, ...
   topLeftCol:bottomRightCol);
end

% Get the masked video
vidWriter = VideoWriter('maskedVid.avi');
open(vidWriter);
for i = 1:k
   writeVideo(vidWriter, shrinkedS(i).cdata);
end
close(vidWriter);
54 %
55 addpath('./stip/stip');
56 stdemo()
Appendix B

MATLAB Code for the STIP Feature Detector and Descriptor

function [pos, scale, orient, desc] = detector(im, img, octaves, ...
   intervals, object_mask, contrast_threshold, ...
   curvature_threshold, interactive)

% The aim of this detector is to detect local extrema
% in the optical flow images, and give the descriptors for the
% detected keypoints.
% This implementation is borrowed from Thomas F. El-Maraghi's
% implementation of SIFT and inspired by:
% [1] David G. Lowe, "Distinctive Image Features from ...
% Scale-Invariant Keypoints",
% accepted for publication in the International Journal of ...
% [2] David G. Lowe, "Object Recognition from Local ...
% Scale-Invariant Features",
% Proc. of the International Conference on Computer Vision, ...
% Corfu,
% September 1999.
% Input:
im - the optical flow image

img - the input image, with pixel values normalize to lie between [0,1].

octaves - the number of octaves to search for keypoints ...
(default = 4).

intervals - the number of geometrically sampled intervals to divide each octave into when searching for keypoints (default = 2).

object_mask - a binary mask specifying the location of the ... object in the image to search for keypoints on. If not specified, the ... whole image is searched.

contrast_threshold - the threshold on the contrast of the DOG ... extrema before classifying them as keypoints (default = 0.03).

curvature_threshold - the upper bound on the ratio between the ... principal curvatures of the DOG extrema before classifying it as a keypoint (default = 10.0).

interactive - ≥ 1 displays progress and timing information,
≥ 2 displays intermediate results of the algorihtm (default = 1).

Output:

pos - an Nx2 matrix containing the (x,y) coordinates of the ... keypoints stored in rows.

scale - an Nx3 matrix with rows describing the scale of each ... keypoint (i.e., first column specifies the octave, second column specifies ... the interval, and third column specifies sigma).

orient - a Nxl vector containing the orientations of the ... keypoints [-pi,pi).

desc - an Nx128 matrix with rows containing the feature descriptors corresponding to the keypoints.

if ~exist('octaves','var')
octaves = 4;
end
if -exist('intervals','var')
    intervals = 2;
end
if -exist('object_mask','var')
    object_mask = ones(size(im));
end
if -exist('contrast_threshold','var')
    contrast_threshold = 0.1;
end
if -exist('curvature_threshold','var')
    curvature_threshold = 10.0;
end
if -exist('interactive','var')
    interactive = 1;
end

% Check that the flow filed is normalized to [-1, 1]
if (min(im(:)<-1) || (max(im(:))>1))
    fprintf(2, 'Warning: image not normalized to [-1,1].
')
end

% Split the input m x n x 2 flow field into to channels: horizontal
% and vertical, and treat them differently
imh = im(:,:,1);
inv = im(:,:,2);

% Blur the flow field with a standard deviation of 0.5 to prevent
% aliasing and then upsample the image by a factor of 2 using
% linear interpolation.
if interactive > 1
    fprintf(2, 'Doubling image size for first octave...
');
end
tic;
antialias_sigma = 0.5;
if antialias_sigma == 0
    signalh = imh;
    signalv = imv;
else
    g = gaussian_filter(antialias_sigma);
    signalh = conv2(g, g, imh, 'same');
    signalv = conv2(g, g, imv, 'same');
end

% signal = im;
[X, Y] = meshgrid(1:0.5:size(signalh, 2), 1:0.5:size(signalh, 1));
signalh = interp2(signalh, X, Y, '*linear'); % up-sampling
signalv = interp2(signalv, X, Y, '*linear'); % up-sampling
subsample = [0.5]; % subsampling rate for doubled image is 0.5

% Now generate the Gaussian and Difference-of-gaussian (DOG)
% pyramids. The pyramids will be stored as several cell arrays,
% gauss_pyr_h{orient,interval}, gauss_pyr_v{orient,interval}, and
% DOG_pyr_h{orient,interval}, DOG_pyr_v{orient,interval}. In order
% to detect keypoints on s intervals per octave, s+3 blurred images
% in the gaussian pyramid should be generated: s+3 blurred images
% generate s+2 DOG images, plus two images are needed (the highest
% and lowest scales of the octave) for extrema detection.

% Generate the first image of the first octave of the gaussian
% pyramid by preblurring the doubled image with a gaussian with a
% standard deviation of 1.6.
if interactive > 1
    fprintf(2, 'Preblurring image...
');
end
preblur_sigma = sqrt(sqrt(2)^2 - (2*antialias_sigma)^2);
if preblur_sigma == 0
    gauss_pyr_h{1,1} = signalh;
    gauss_pyr_v{1,1} = signalv;
else
    g = gaussian_filter(preblur_sigma);
    gauss_pyr_h{1,1} = conv2(g, g, signalh, 'same');
end
gauss_pyr_v{1,1} = conv2(g, g, signalv, 'same');

end

clear signal
pre_time = toc;
if interactive >= 1
    fprintf(2, 'Preprocessing time %.2f seconds.
', pre_time);
end

% Initial blurring for the first image of the first octave of the
% pyramid.
initial_sigma = sqrt((2*antialiassigma)^2 + preblur_sigma^2);

% Keep track of the absolute sigma for the octave and scale
absolute_sigma = zeros(octaves, intervals+3);
absolute_sigma(1, 1) = initial_sigma * subsample(1);

% Keep track of the filter sizes and standard deviations used to
generate the pyramid
filter_size = zeros(octaves, intervals+3);
filter_sigma = zeros(octaves, intervals+3);

% Generate the remaining levels of the geometrically sampled
% gaussian and DOG pyramids
if interactive >= 1
    fprintf(2, 'Expanding the Gaussian and DOG pyramids...
');
end
tic;
for octave = 1:octaves
    if interactive >= 1
        fprintf(2, ['\tProcessing octave %d: image size %d x %d ' ...
            'subsample %.1f
'], octave, ...
            size(gauss_pyr_h{octave, ... 1},2), size(gauss_pyr_h{octave, 1},1), ...
            subsample(octave));
        fprintf(2, '\t\tInterval 1 sigma %f
', ...
            absolute_sigma(octave, 1));
    end
end
end

sigma = initial_sigma;
g = gaussian_filter(sigma);
filter_size(octave, 1) = length(g);
filter_sigma(octave, 1) = sigma;
DOG_pyr_h{octave} = zeros(size(gauss_pyr_h{octave,1},1), ... 
    size(gauss_pyr_h{octave,1},2), ... 
    intervals+2);
DOG_pyr_v{octave} = zeros(size(gauss_pyr_v{octave,1},1), ... 
    size(gauss_pyr_v{octave,1},2), ... 
    intervals+2);
for interval = 2:(intervals+3)
    % Compute the standard deviation of the gaussian filter 
    % needed to produce the next level of the geometrically 
    % sampled pyramid. Here, sigma_k+1 = k*sigma. By definition 
    % of successive convolution, the required blurring sigma_f 
    % to produce sigma_i+1 from sigma_i is:
    %
    % sigma_i+1^2 = sigma_f,i^2 + sigma_i^2
    % (k*sigma_i)^2 = sigma_f,i^2 + sigma_i^2
    %
    % therefore:
    %
    % sigma_f,i = sqrt(k^2-1)*sigma_i
    % where k = 2^(1/intervals) to span the octave, so:
    %
    % sigma_f,i = sqrt(2^(2/intervals)-1)*sigma_i
    
sigma_f = sqrt(2^(2/intervals)-1)*sigma;
g = gaussian_filter(sigma_f);
sigma = (2^(1/intervals))*sigma;

    % Keep track of the absolute sigma
    absolute_sigma(octave,interval) = sigma * subsample(octave);

    % Store the size and standard deviation of the filter for
% later use
filter_size(octave, interval) = length(g);
filter_sigma(octave, interval) = sigma;
gauss_pyr_h(octave, interval) = conv2(g, g, ...)
        gauss_pyr_h(octave, interval-1), 'same');
gauss_pyr_v(octave, interval) = conv2(g, g, ...)
        gauss_pyr_v(octave, interval-1), 'same');
DOG_pyr_h(octave)(::,::,interval-1) = ...
        gauss_pyr_h(octave, interval) - ...
        gauss_pyr_h(octave, interval-1);
DOG_pyr_v(octave)(::,::,interval-1) = ...
        gauss_pyr_v(octave, interval) - ...
        gauss_pyr_v(octave, interval-1);
if interactive >= 1
        fprintf(2, 'Interval %d sigma %f
', interval,
        absolute_sigma(octave, interval));
end
if octave < octaves
    % Subsample this image by a factor of 2 to produce the
    % first image of the next octave.
    sz = size(gauss_pyr_h{octave, intervals+1});
    [X, Y] = meshgrid(1:2:sz(2), 1:2:sz(1));
    gauss_pyr_h{octave+1, l} = interp2(gauss_pyr_h{octave, ...}
        intervals+1},X,Y,'*nearest');
    gauss_pyr_v{octave+1, l} = interp2(gauss_pyr_v{octave, ...}
        intervals+1},X,Y,'*nearest');
    absolute_sigma{octave+1, l} = absolute_sigma{octave, ...}
        intervals+1};
    subsample = [subsample subsample(end)*2];
end
pyr_time = toc;
if interactive >= 1
    fprintf(2, 'Pyramid processing time %.2f seconds.
', pyr_time);
end
% Combine the horizontal and vertical DOG
DOG_pyr = DOG_pyr_h;
for octave=1:octaves
    for interval=2:(intervals+3)
        DOG_pyr{octave}(:,:,interval-1) = sqrt( ... 
            DOG_pyr_h{octave}(:,:,interval-1).^2 + ... 
            DOG_pyr_v{octave}(:,:,interval-1).^2 );
    end
end

% Display the gaussian pyramid when in interactive mode
if interactive > 2
    sz = zeros(1,2);
    sz(2) = (intervals+3)*size(gauss_pyr_h{1,1},2);
    for octave = 1:octaves
        sz(1) = sz(1) + size(gauss_pyr_h{octave,1},1);
    end
    pic_h = zeros(sz);
    pic_v = zeros(sz);
    y = 1;
    for octave = 1:octaves
        x = 1;
        sz = size(gauss_pyr_h{octave,1});
        for interval = 1:(intervals+3)
            pic_h(y:(y+sz(1)-1),x:(x+sz(2)-1)) = ...
                gauss_pyr_h{octave, interval};
            pic_v(y:(y+sz(1)-1),x:(x+sz(2)-1)) = ...
                gauss_pyr_v{octave, interval};
            x = x + sz(2);
        end
        y = y + sz(1);
    end
    fig_h = figure;
    clf;
    showIm(pic_h);
resizeImageFig(fig_h, size(pic_h), 0.25);
fprintf(2, ['The gaussian pyramid (0.25 scale) for horizontal ...
    velocity.
Press any key to continue.\n']);
pause;
close(fig_h);
fig_v = figure;
cif;
showIm(pic_v);
resizeImageFig(fig_v, size(pic_v), 0.25);
fprintf(2, ['The gaussian pyramid (0.25 scale) for vertical ...
    velocity.
Press any key to continue.\n']);
pause;
close(fig_v);
end

% Display the DOG pyramid when in interactive mode
if interactive > 2
    sz = zeros(1,2);
    sz(2) = (intervals+2)*size(DOG_pyr_h{1}(:,:,1),2);
    for octave = 1:octaves
        sz(1) = sz(1) + size(DOG_pyr_h{octave}(:,:,1),1);
    end
    pic_h = zeros(sz);
    pic_v = zeros(sz);
    y = 1;
    for octave = 1:octaves
        x = 1;
        sz = size(DOG_pyr_h{octave}(:,:,1));
        for interval = 1:(intervals + 2)
            pic_h(y:(y+sz(1)-1),x:(x+sz(2)-1)) = ...
                DOG_pyr_h{octave}(:,:,interval);
            pic_v(y:(y+sz(1)-1),x:(x+sz(2)-1)) = ...
                DOG_pyr_v{octave}(:,:,interval);
            x = x + sz(2);
        end
        y = y + sz(1);
fig_h = figure;
clf;
showIm(pic_h);
resizeImageFig(fig_h, size(pic_h), 0.25);
fprintf(2, ['The DOG pyramid (0.25 scale) horizontal.\nPress ... any key to continue.\n']);
pause;
close(fig_h);
fig_v = figure;
clf;
showIm(pic_v);
resizeImageFig(fig_v, size(pic_v), 0.25);
fprintf(2, ['The DOG pyramid (0.25 scale) vertical.\nPress ... any key to continue.\n']);
pause;
close(fig_v);

% The next step is to detect local maxima in the DOG pyramid. When
% a maximum is found, two tests are applied before labeling it as a
% keypoint. First, it must have sufficient contrast. Second, it ...
% should
% not be an edge point (i.e., the ratio of principal curvatures ...
% at the
% extremum should be below a threshold).

% Compute threshold for the ratio of principle curvature test ... applied to
% the DOG extrema before classifying them as keypoints.

curvature_threshold = ((curvature_threshold + ... 1)^2)/curvature_threshold;

% 2nd derivative kernels
xx = [ 1 -2  1 ];
yy = xx';

xy = [ 1 0 -1; 0 0 0; -1 0 1 ]/4;

raw_keypoints = [];
contrast_keypoints = [];
curve_keypoints = [];

% Detect local maxima in the DOG pyramid
if interactive > 1
    fprintf(2, 'Locating keypoints...
');
end

tic;

loc = cell(size(DOG_pyr)); % boolean maps of keypoints
for octave = 1:octaves
    if interactive > 1
        fprintf( 2, 'Processing octave %d\n', octave );
    end
    for interval = 2:(intervals+1)
        keypoint_count = 0;
        contrast_mask = abs(DOG_pyr{octave}(:,:,interval)) > ...
            contrast_threshold;
        loc{octave,interval} = ... 
            zeros(size(DOG_pyr{octave}(:,:,interval)));
        edge = ceil(filter_size(octave,interval)/2);
        for y=(1+edge):(size(DOG_pyr{octave}(:,:,interval),1)-edge)
            for x=(1+edge):(size(DOG_pyr{octave}(:,:,interval),2)-edge)
                % Only check for extrema where the object mask is 1
                if object_mask(round(y*subsample(octave)), ...
                    round(x*subsample(octave))) == 1
                    % When not displaying intermediate results, ...
                    % perform the check that the current location

% in the DOG pyramid is above the contrast ...
threshold before checking
% for an extrema for efficiency reasons. ...
Note: we could not make this
% change of order if we were interpolating ...
the locations of the extrema.
if( (interactive >= 2) || (contrast_mask(y,x) ... == 1) )

% Check for a max or a min across space ...
and scale
tmp = ...

DOG_pyr{octave}((y-1):(y+1),(x-1):(x+1), ... ...
(interval-1):(interval+1));
pt_val = tmp(2,2,2);
if( (pt_val == min(tmp(:))) || (pt_val == ... max(tmp(:,:,))) )
% The point is a local extrema of the ...
DOG image. Store its coordinates for ...
% displaying keypoint location in ...
interactive mode.
raw_keypoints = [raw_keypoints; ... ...
  x*subsample(octave) ... ...
  y*subsample(octave)];

if abs(DOG_pyr{octave}(y,x,interval)) ... 
>= contrast_threshold
% The DOG image at the extrema is ...
above the contrast threshold. ...
Store ...
% its coordinates for displaying ...
keypoint locations in ...
interactive mode.
contrast_keypoints = ...
[contrast_keypoints; ... ...
raw_keypoints(end,:)];
% Compute the entries of the Hessian matrix at the extrema location.
Dxx = sum(DOG_pyr{octave}...
    (y,x-1:x+1,interval) .* ...
    xx);
Dyy = sum(DOG_pyr{octave}...
    (y-l:y+1,x,interval) .* ...
    yy);
Dxy = sum(sum(DOG_pyr{octave}...
    (y-l:y+1,x-l:x+l,interval) ...
    .* xy));

% Compute the trace and the determinant of the Hessian.
Tr_H = Dxx + Dyy;
Det_H = Dxx*Dyy - Dxy^2;

curvature_ratio = (Tr_H^2)/Det_H;

if ((Det_H > 0) && ...
    (curvature_ratio < ... curvatures is below the threshold (i.e.,
    % it is not an edge point). ... Store its coordianates for ...
    % keypoint locations in ... interactive mode.
    curve_keypoints = ... [curve_keypoints; ...
raw_keypoints(end,:));

% Set the loc map to 1 to at ...
this point to indicate a ...
keypoint.
loc(octave,interval)(y,x) = 1;
keypoint_count = ...
keypoint_count + 1;

end
end
end
end
end
end
if interactive ≥ 1
    fprintf( 2, '	%d keypoints found on interval ...%d
', keypoint_count, interval );
end
end
keypoint_time = toc;
if interactive ≥ 1
    fprintf(2, 'Keypoint location time %f seconds.
', keypoint_time);
end
% Display results of extrema detection and keypoint filtering in ...
    interactive mode.
if interactive ≥ 2
    fig = figure;
clf;
imshow(img);
hold on;
plot(raw_keypoints(:,1),raw_keypoints(:,2),'y+');
resizeImageFig( fig, size(im), 2 );
fprintf( 2, 'DOG extrema (2x scale).
Press any key to ...
    continue.\n' );

pause;
close(fig);
fig = figure;
clf;
imshow(img);
hold on;
plot(contrast_keypoints(:,1),contrast_keypoints(:,2),'y+');
resizeImageFig( fig, size(im), 2 );
fprintf( 2, 'Keypoints after removing low contrast extrema ...(2x scale).
Press any key to continue.\n' );

pause;
close(fig);
fig = figure;
clf;
imshow(img);
hold on;
plot(curve_keypoints(:,1),curve_keypoints(:,2),'y+');
resizeImageFig( fig, size(im), 2 );
fprintf( 2, 'Keypoints after removing edge points using ... principal curvature filtering (2x scale).
Press any key ...
    to continue.\n' );

pause;
close(fig);

end

clear raw_keypoints contrast_keypoints curve_keypoints

% Now we want to assign orientations to the keypoints, in order to achieve rotation invariance. Note that here the computation of magnitude and orientation is different from the original SIFT algorithm. We are working on the flow domain and are mainly interested in the flow pattern. What we want to achieve in the invariance is simply the flow invariance, i.e., the direction of flow vector. After detecting a keypoint, its flow vector should
% already be known, we've already known the magnitude and
% orientation. But what we need is to associate its own flow vector
% with those surrounding it.

% Now assign orientations to the keypoints. We histogram the
% gradient orientation over a region about each keypoint.
\texttt{g = \texttt{gaussian\_filter}(1.5 * \texttt{absolute\_sigma}(1,\texttt{intervals}+3) / ...}
\texttt{subsample(1));}
\texttt{zero\_pad = \texttt{ceil(length(g)/2));}

% Compute the gradient direction and magnitude of the gaussian
% pyramid images
\texttt{if interactive \geq 1}
\texttt{fprintf(2, 'Computing the orientations for the gauss ...}
\texttt{ pyramid...\n');}
\texttt{end}
\texttt{tic;}

\texttt{mag\_pyr = \texttt{cell(size(gauss\_pyr\_h))};}
\texttt{grad\_pyr = \texttt{cell(size(gauss\_pyr\_h));}}
\texttt{for octave = 1:octaves}
\texttt{    for interval = 2:(\texttt{intervals}+1)}
\texttt{        horizontal = gauss\_pyr\_h{octave, interval};}
\texttt{        vertical = gauss\_pyr\_v{octave, interval};}

\texttt{        mag = \texttt{zeros(size(gauss\_pyr\_h{octave,interval}))};}
\texttt{        mag(:,:) = \texttt{sqrt(horizontal.^2+vertical.^2);}}

\texttt{        mag\_pyr{octave,\texttt{intervals}+1} = \texttt{zeros(size(mag)+2*zero\_pad));}
\texttt{        mag\_pyr{octave,\texttt{intervals}+1}((zero\_pad+1):(end\_zero\_pad),...}
\texttt{                      (zero\_pad+1):(end\_zero\_pad)) = mag;}

84
% Compute the orientation of the flow
grad = zeros(size(gauss_pyr_h{octave,interval}));
grad(:, :) = atan2(vertical, horizontal);
grad(grad == pi) = -pi;

% Store the orientation of the flow in the pyramid
grad_pyr{octave,interval} = zeros(size(grad)+2*zero_pad);
grad_pyr{octave,interval}((zero_pad+1):(end-zero_pad),... 
    (zero_pad+1):(end-zero_pad)) = ...
    grad;
end
end
clear mag grad
grad_time = toc;
if interactive > 1
    fprintf( 2, 'Gradient calculation time %.2f seconds.
', ...
    grad_time );
end

% Assign orientations to the keypoints that have been located. This
% is done by looking for peaks in histograms of flow orientations
% in regions surrounding each keypoint. A keypoint may
% be assigned more than one orientation. If it is, two identical
% descriptors are added to the database with different
% orientations.

% Set up the histogram bin centers for a 36 bin histogram.
num_bins = 36;
hist_step = 2*pi/num_bins;
hist_orient = -pi:hist_step:(pi-hist_step);

% Initialize the positions, orientations, and scale information of
% the keypoints to empty matrices.
pos = [];
orient = [];
scale = [];

% Assign orientations to the keypoints.
if interactive >= 1
    fprintf( 2, 'Assigning keypoint orientations...
' );
end
tic;
for octave = 1:octaves
    if interactive >= 1
        fprintf( 2, 'Processing octave %d
', octave );
    end
    for interval = 2:(intervals + 1)
        if interactive >= 1
            fprintf( 2, 'Processing interval %d ', interval );
        end
        keypoint_count = 0;
        % Create a gaussian weighting mask with a standard deviation of 1/2 of
        % the filter size used to generate this level of the pyramid.
        g = gaussian_filter( 1.5 * ...
            absolute_sigma(octave,interval)/subsample(octave) );
        hf_sz = floor(length(g)/2);
        g = g'*g;
        % Zero pad the keypoint location map.
        loc_pad = zeros(size(loc{octave,interval})+2*zero_pad);
        loc_pad((zero_pad+1):(end-zero_pad),(zero_pad+1):(end-zero_pad)) ... 
            = loc{octave,interval};
        % Iterate over all the keypoints at this octave and ... orientation.
        [iy, ix]=find(loc_pad==1);
        for k = 1:length(iy)
            % Histogram the gradient orientations for this keypoint ... weighted by the
% gradient magnitude and the gaussian weighting mask.
x = ix(k);
y = iy(k);
wght = g.*mag_pyr(octave,interval)((y-hf_sz):(y+hf_sz),...
     (x-hf_sz):(x+hf_sz));
grad_window = ... 
    grad_pyr(octave,interval)((y-hf_sz):(y+hf_sz),...
     (x-hf_sz):(x+hf_sz));
orient_hist=zeros(length(hist_orient),1);
for bin=1:length(hist_orient)
    % Compute the difference of the orientations mod pi
    diff = mod( grad_window - hist_orient(bin) + pi, 2*pi ... 
               ) - pi;
    % Accumulate the histogram bins
    orient_hist(bin)=orient_hist(bin)+sum(sum(wght.*max(1 ...
                - abs(diff)/hist_step,0)));
end

% Find peaks in the orientation histogram using nonmax ...
    suppressed
    peaks = orient_hist;
    rot_right = [ peaks(end); peaks(1:end-1) ];
    rot_left = [ peaks(2:end); peaks(1) ];
    peaks( peaks < rot_right ) = 0;
    peaks( peaks < rot_left ) = 0;

% Extract the value and index of the largest peak.
[max_peak_val, ipeak] = max(peaks);

% Iterate over all peaks within 80% of the largest peak ... 
    and add keypoints with 
    % the orientation corresponding to those peaks to the ... 
        keypoint list.
peak_val = max_peak_val;
while( peak_val > 0.8*max_peak_val )
% Interpolate the peak by fitting a parabola to the three histogram values
% closest to each peak.
A = []; b = [];
for j = -1:1
  A = [A; (hist_orient(ipeak)+hist_step*j).^2 ... (hist_orient(ipeak)+hist_step*j) 1];
  bin = mod( ipeak + j + num_bins - 1, num_bins ) + 1;
  b = [b; orient_hist(bin)];
end
c = pinv(A)*b;
max_orient = -c(2)/(2*c(1));
while( max_orient < -pi )
  max_orient = max_orient + 2*pi;
end
while( max_orient ≥ pi )
  max_orient = max_orient - 2*pi;
end

% Store the keypoint position, orientation, and scale information
pos = [pos; [(x-zero_pad) ... (y-zero_pad)]*subs samp (octave) ];
orient = [orient; max_orient];
scale = [scale; octave interval ... absolute_sigma(octave,interval)];
keypoint_count = keypoint_count + 1;

% Get the next peak
peaks(ipeak) = 0;
[peak_val, ipeak] = max(peaks);
end
end
if interactive ≥ 1
  fprintf( 2, '(%d keypoints)
', keypoint_count );
end
end

clear loc loc_pad
orient_time = toc;
if interactive > 1
  fprintf( 2, 'Orientation assignment time %.2f seconds.\n', ...
    orient_time );
end

% Display the keypoints with scale and orientation in interactive ...
  % mode.
if interactive > 2
  fig = figure;
  clf;
  imshow(img);
  hold on;
  display_keypoints( pos, scale(:,3), orient, 'y' );
  resizeImageFig( fig, size(img), 2 );
  fprintf( 2, 'Final keypoints with scale and orientation (2x ... scale).\nPress any key to continue.\n' );
  pause;
  close(fig);
end

% The final of the SIFT algorithm is to extract feature descriptors
% for the keypoints. The descriptors are a grid of gradient
% orientation histograms, where the sampling grid for the
% histograms is rotated to the main orientation of each
% keypoint. The grid is a 4x4 array of 8 bin orientation
% histograms. This produces 128 dimensional feature vectors.
%
% The orientation histograms have 8 bins
orient_bin_spacing = pi/4;
orient_angles = [-pi:orient_bin_spacing:(pi-orient_bin_spacing)];
% The feature grid has 4x4 cells. feat_grid describes the cell center positions.
grid_spacing = 4;
[x_coords, y_coords] = meshgrid([-6:grid_spacing:6]);
feat_grid = [x_coords(:) y_coords(:)]';

% Loop over all of the keypoints.
if interactive > 1
    fprintf( 2, 'Computing keypoint feature descriptors for %d ... 
    keypoints', size(pos,1) );
end
for k = 1:size(pos,1)
    x = pos(k,1)/subsample(scale(k,1));
    y = pos(k,2)/subsample(scale(k,1));
    % Rotate the grid coordinates.
    M = [cos(orient(k)) -sin(orient(k)); sin(orient(k)) ... 
        cos(orient(k))];
    feat_rot_grid = M*feat_grid + repmat([x; y],1,size(feat_grid,2));
    feat_rot_samples = M*feat_samples + repmat([x; ... 
        y],1,size(feat_samples,2));
    % Initialize the feature descriptor.
    feat_desc = zeros(1,128);
    % Initialize the descriptor list to the empty matrix.
desc = [];

% Histogram the gradient orientation samples weighted by the gradient magnitude and a gaussian with a standard deviation of 1/2 the feature window. To avoid boundary effects, each sample is accumulated into neighbouring bins weighted by 1-d
% in all dimensions, where \( d \) is the distance from the center of 
% the bin measured in units of bin spacing.

```matlab
for s = 1:size(feat_rot_samples,2)
    x_sample = feat_rot_samples(1,s);
    y_sample = feat_rot_samples(2,s);

    % Interpolate the gradient at the sample position
    [X, Y] = meshgrid((x_sample-1):(x_sample+1), ... 
                     (y_sample-1):(y_sample+1));
    G_h = interp2(gausspyrh{scale(k,1),scale(k,2)}, X, Y, ... 
                  '*linear');
    G_v = interp2(gausspyrv{scale(k,1),scale(k,2)}, X, Y, ... 
                  '*linear');
    G_h(isnan(G_h)) = 0;
    G_v(isnan(G_v)) = 0;
    diff_x = 0.5*(G_h(2,3) - G_h(2,1));
    diff_y = 0.5*(G_v(3,2) - G_v(1,2));
    mag_sample = sqrt(diff_x^2 + diff_y^2);
    grad_sample = atan2(diff_y, diff_x);
    if grad_sample == pi
        grad_sample = -pi;
    end
```

% Compute the weighting for x and y dimensions
```matlab
    x_wght = ... 
    max(l-(abs(feat_rot_grid(1,:)-x_sample)/grid_spacing),0);
    y_wght = ... 
    max(l-(abs(feat_rot_grid(2,:)-y_sample)/grid_spacing),1);
    pos_wght = reshape(repmat(x_wght.*y_wght,8,1),1,128);
```

% Compute the weighting for the orientation, rotating the 
% gradient to the main orientation of the keypoint first, 
% then computing the difference in angle to the histogram 
% bin mod pi.
```matlab
diff = mod(grad_sample-orient(k)-orient_angles+pi, 2*pi) ... 
      - pi;
```
orient_wght = max(1-abs(diff)/orient_bin_spacing,0);
orient_wght = repmat(orient_wght,1,16);

% Compute the gaussian weighting

\[
g = \exp\left(-\frac{(x_{\text{sample}}-x)^2+(y_{\text{sample}}-y)^2}{2\times \text{feat_window}^2}\right) / (2\pi \text{feat_window}^2);
\]

% Accumulate the histogram bins

feat_desc = feat_desc + pos_wght.*orient_wght*g*mag_sample;
end

% Normalize the feature descriptor to a unit vector to make the
% descriptor invariant to affine changes in illumination.

feat_desc = feat_desc / norm(feat_desc);

% Threshold the large components in the descriptor to 0.2 and
% then renormalize to reduce the influence of large gradient
% magnitudes on the descriptor

feat_desc(feat_desc>0.2) = 0.2;
feat_desc = feat_desc / norm(feat_desc);

% Store the descriptor

desc = [desc; feat_desc];
if (interactive ≥ 1) & (mod(k,25) == 0)
    fprintf(2, '.');
end
end
desc_time = toc;

% Adjust for the sample offset

sample_offset = -(subsample - 1);
for k = 1:size(pos, 1)
pos(k,:) = pos(k,:) + sample_offset(scale(k,1));
end
% Return only the absolute scale
if size(pos,1) > 0
    scale = scale(:,3);
end

% Display summary in interactive mode
if interactive > 1
    fprintf( 2, 'nDescriptor processing time %.2f seconds.
', desc_time);
    fprintf( 2, 'Processing time summary:
' );
    fprintf( 2, 'Preprocessing: %.2f s
', pre_time );
    fprintf( 2, 'Pyramid: %.2f s
', pyr_time );
    fprintf( 2, 'Keypoints: %.2f s
', keypoint_time );
    fprintf( 2, 'Gradient: %.2f s
', grad_time );
    fprintf( 2, 'Orientation: %.2f s
', orient_time );
    fprintf( 2, 'Descriptor: %.2f s
', desc_time );
    fprintf( 2, 'Total processing time %.2f seconds.
', pre_time +
        pyr_time + keypoint_time + grad_time + orient_time + ...
        desc_time );
end
Bibliography


