Measuring 3D-Length from an Image Using Robust and Accurate Curve Estimation

Maolong Tang

A dissertation

submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

University of Washington

2019

Reading Committee:

Ming-Ting Sun, Chair

Linda G. Shapiro

Jenq-Neng Hwang

Program Authorized to Offer Degree:

Department of Electrical Engineering

University of Washington

Abstract

Measuring 3D-Length from an Image Using Robust and Accurate Curve Estimation

Maolong Tang

Chair of the Supervisory Committee:
Professor Ming-Ting Sun
Department of Electrical Engineering

Accurate and robust curve estimation is important for many practical applications. In this work we develop an application for measuring an infant's 3D length with regular 2D cellphone photos using accurate and robust curve estimation.

Measuring 3D length of a deformable object from a 2D RGB image is a very challenging task. The 2D length that appears in the 2D image could represent very different actual 3D lengths depending on the distance of the object to the camera and the camera pose. The depth information of the object parts is lost when the object is projected from the 3D world to the 2D image. Furthermore, a deformable and dynamic object such as an infant moves all the time, which could introduce severe motion blur in the 2D image. The infant's skin is relatively textureless and lacks easily identifiable feature points. There is no existing reliable method that

could estimate its physical length directly from a 2D image. 3D pose estimation based on learning usually does not infer 3D length with enough accuracy.

To reliably and accurately calculate the 3D length from a 2D image, we propose a new technique which uses easily obtainable and easy-to-apply circular stickers with a known size to help create feature points and provide reference physical lengths at the 3D locations of the feature points. A circular sticker in the 3D space will be projected into an ellipse in the 2D image. The sticker's 3D location could be recovered with high accuracy from a single RGB image if its ellipse projection in the 2D image could be precisely estimated. With the help of the ellipse estimation, the length of a deformable object such as an infant could be calculated with decent accuracy. However, estimating the exact location of an elliptical curve from real world images, which contain various kind of noise and motion blur, is not a trivial task. Few existing ellipse estimation methods can reliably detect the ellipses and accurately estimate the ellipse parameters to obtain the 3D lengths with enough accuracy to meet our requirement (for infant length measurement). To solve the problem, we develop a deformable contour based method called ellipse growing for reliably detecting the curve of the ellipse from the images and estimate the ellipse parameters accurately.

For images with strong motion blur, we could deblur the images before the ellipse parameter estimation. However, existing deblurring algorithms may introduce artifacts that make ellipse parameter estimation even harder and less accurate. Instead, we propose a CNN-based method to help restore the ellipses. It is fast and able to handle typical motion blur of real-world images. Neural networks require a lot of training data. We propose an approach to automatically generate enough realistic training data for training the neural networks to obtain accurate results. The proposed approach can be generalized for estimating other shapes in the images.

Inspired by the CNN's effectiveness, we further generalize the CNN to recover more general smooth curves from motion blurred and noisy images without restoration. We developed novel loss functions, new differentiable layers, and synthetic data generation methods. Our model is light weighted, and is very effective on real-world photos. The overall pipeline could be easily used for more general curve estimation tasks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am indebted to many people during the graduate study at UW. First, I would like to thank my dissertation advisor, Prof. Ming-Ting Sun, for his patience and supportiveness. He is not only the teacher that taught me the very first image processing class, but also the academic mentor that coach me in researcher survival skills. It is my great fortune to be his student. Second, I would like to thank my committee members: Prof. Linda Shapiro, Prof. Jenq-Neng Hwang and Prof. Michael McCarthy for their suggestions and timely help.

Besides my dissertation committees, I owe a debt of gratitude to Multimedia signal processing lab mates and friends at UW, including Yang Zhao, Jun Xie, Haoming Chen, Xiang Zhang, Prof. Junfeng Yao, Prof. Zhibin Gao, Prof. Cheng Jin, Mayoore Jaiswal, Dianqi Li, Kevin Lin, Kun Yang, Lianqiang Li, Alex Hu, Shifeng Zhu, Di Sun. I also would like to thank my internship mentor Pavel Komlev and colleagues Kai Wang, Simon Chen and Shawn Poindexter for the wonderful three months. I learned many engineering techniques that are impossible to learn in school.

Lastly, I would like to thank my parents for their endless love and support. Without their encouragement, completing the graduate study would not have been possible.

Chapter 1. INTRODUCTION

Many practical applications involve measuring 3D lengths of deformable objects, for example, measuring an infant's length for the infant's growth monitoring. Traditionally, measuring an infant's length is performed with an infantometer where the infant is put into the infantometer with the head aligned with the headboard and the footboard is adjusted to measure the infant's length. However, the infant struggles and cries in the measuring process, and it often needs three persons to hold the infant and position the infant's head, legs, and the boards of the infantometer during the process. Thus, it is not practical for a parent to perform this measurement at home regularly. It is desirable to be able to measure an infant's length from a cellphone photo. This would make the measurement of the infant's length easy which is important for infant growth velocity monitoring.

In general, measuring 3D lengths from a 2D RGB image is a very challenging task. The 2D distance that appears in the 2D image could represent a very different actual 3D distance depending on the distance of the object to the camera and the camera pose. The depth information of the object is lost when the object is projected from the 3D world to the 2D image. Furthermore, a deformable and dynamic object such as an infant moves all the time, which could introduce severe motion blur. The infant skin is relatively textureless and lacks easily identifiable feature points. There is no reliable existing method that could estimate its physical length directly from a 2D image. 3D pose estimation based on learning, usually does not infer 3D lengths with enough accuracy.

To make the measurement of an infant's length from a cellphone image possible, we developed a technique in which fixed-size circular stickers are put on the body joints of the infant before taking photos. The circular stickers will be projected onto the image plane into ellipses with different parameters depending on the direction and distance from the camera. By estimating the

parameters of the ellipses in the picture, we can calculate the 3D positions of the sticker centers and the length of the infant. Our algorithm automatically calculates the 3D positions of the body parts and the total length of the infant. The circular stickers can be put on the infant's body easily in a few seconds, before the pictures are taken. This new technology would make frequent measurements of the infant's length and the tracking of the infant's growth velocity possible.

In order to obtain accurate 3D lengths, we need to be able to detect the ellipses in the image robustly and the parameters of the ellipses need to be estimated accurately. We propose a fast and robust ellipse detection and parameter estimation method based on ellipse growing, which could give good results and is robust to real photos compared to those using existing ellipse detection and parameter estimation algorithms.

A difficulty with the proposed approach is that the infant moves during the picture taking, which could cause severe motion blur. We propose to use a CNN (Convolutional Neural Network) to restore the ellipses before ellipse fitting and ellipse parameter estimation. To generate realistic training data, which are needed for training the CNN to accurately restore the ellipses for length measurements, we propose to simulate the whole ellipse formation pipeline. The whole training process includes generating ellipses of random sizes, motion blurs, and illumination changes, and adding noise, demosaicing, and gamma and inverse gamma corrections. Simulation results show that better accuracy of measurements can be achieved with the proposed training methods.

Intrigued by this ellipse-based 3D length estimation application and the effectiveness of CNN, we study more general curve detection from heavily degraded photos. We develop a more sophisticated synthetic data generation pipeline for training the CNN, a novel differentiable layer and location aware loss functions to facilitate accurate edge recovery. The overall process does not need human labeling. The results show that our proposed method could detect practical motion-

blurred, noisy and faint curves with just one trained model. The whole pipeline could also be easily adjusted for other edge/curve detection tasks.

The organization of the rest of this dissertation is as follows. In Chapter 2 we briefly review existing publications related to our work. In Chapter 3 we introduction our proposed infant length measurement technique, which involves detecting elliptical curves and estimating ellipse parameters of the elliptical markers. Chapter 4 proposes our ellipse growing method that could robustly detect pure color circular markers in real photos. Chapter 5 discusses using a CNN to recover motion blurred markers. Chapter 6 extends the work of Chapter 5 to general curve segments. It proposes a CNN-based smooth curve detector that is capable to handle many degradation effects such as motion blur, heavy noise and faint contrast in real-world photos. Chapter 7 summaries this dissertation and proposes future research directions.

Chapter 2. RELATED WORK

An intuitive approach to measure the length of an object in the 3D space from images is to construct the 3D structure of the object. With the 3D structure, the 3D coordinates of the feature points and the total length of the object can be calculated. The most popular technique to estimate the 3D structure from 2D images is Structure from Motion (SfM) [1]. SfM requires a rigid textured object, and the availability of multiple photographs from different angles. However, an infant is not a rigid object, and it is difficult to extract enough feature points from the relatively textureless skin. There are two main methods to handle non-rigid objects: one is the template-based method [2] and the other one is the non-rigid structure from motion (NRSfM) method [2]. The template-based method usually requires not only the texture but also a known template, which makes it not applicable for our goal. The NRSfM method does not require a known model, however, this method requires accurate matched feature point pairs [2]. Reliable feature matching such as SIFT [3] does not work well on skin with no apparent texture. Dense matching by the state-of-the-art optical flow algorithms usually give very noisy feature pairs, so they are not suitable for accurate infant length estimation. Structure from shading [4] is a method that does not rely on feature matchings. However, due to unknown light sources and the complexity of cloth material and skin, even the most recent structure from shading algorithm [4] cannot produce satisfactory 3D results from infant photographs.

To accurately determine the 3D length using our proposed approach based on the circular markers as mentioned in Chapter 1, the accuracy of the ellipse parameter estimation is very important. There are many publications and open sourced software about ellipse estimation. Instead of a thorough review, we briefly summarize a few representative methods. Hough transform [5][6] is one of the oldest and probably most well-known ellipse detection method. It

detects an ellipse using a voting strategy, ellipses with high votes will be selected as candidates. This method has many open source implementations such as OpenCV [7] and scikit-image [8]. Hough transform usually returns many false positives, and its running time is relatively long. Another popular ellipse detection method is based on edge chaining [9], which first extracts candidate elliptic curve segments, then group them into higher quality ellipses. This method is usually faster, since curve extraction is a local operation. However, curve extraction is vulnerable to poor image quality.

To detect accurate locations of makers in real-world images, unavoidably, we need to consider the effects of motion blur. This is especially important for our target application, since the infant moves all the time during the picture taking process. Recently, people have applied neural networks to deblur images. A complete review is out of the scope of this report. We briefly summarize three representative works. In [10], it uses blurred patches cropped from actual images to train a CNN that classifies the motion blur-kernel into 73 types. After post-processing, the number of blur-types is increased to 361. Finally, a Markov Random Field (MRF) model is used to calculate a dense and smooth blur-kernel distribution. In [11], instead of training the network with spatial domain input, it takes the image patch's Discrete Fourier Transform (DFT) as input. Convolution and deconvolution in the spatial domain are simplified as multiplication and division in the frequency domain. It achieves the state-of-the-art results with much higher processing speed. In [12], it uses a high-speed camera to take sharp video frames, then synthesizes motion blurred images by averaging consecutive clear frames. It adopts simplified residual network blocks [13] as the basic building blocks. We adopt a similar strategy, but use a much smaller network. Our training data are all synthetic (produced with our proposed method), however, the trained nework works very well on real test images. Most debluring papers focus on recover sharp good looking

images. Few papers discuss the CNN's ability of recovering the object boundary's exact location. In order to calculate the 3D length, the sharp, good-looking image is not enough, we need the correct location of the curves in order to get accurate ellipse parameters. Our result validates that it is possible to use our proposed synthetic data to train a CNN to recover accurate curve locations.

Adopting neural network to recover or extract curves accurately and consistently from low quality images is a direction that has not been fully explored by existing literature yet. Most of the existing CNN curve detectors try to detect semantically meaningful edges. The detected edges are usually served as input features for image segmentation and object detection [14]. Typical neural-network-based edge detectors, such as [15][16][17], treat edge detection as a classification problem, and the training process is supervised by human labeled datasets. The state of the art CNN curve detectors are able to produce results close to or even better than human [16]. However, for low-level image processing tasks, for example curve detection, the probability map style detector is not ideal. Probability is a scalar, it does not provide directional information to chain edge pixels into curve segments, nor does it provide a straightforward way to extract subpixel edge locations. In fact, many probability map style detectors produce very thick edges [18], which make the detectors inappropriate for location critical applications. In our proposed method, the edge strength is represented by the magnitude of maximum contrast of the CNN feature map, the direction of the maximum contrast, which describes the edge orientation, and the edge subpixel information that could be obtained by fitting a parabola curve along the direction of the maximum contrast [19]. For model training, it is not practical to ask a human to accurately label low-level edges with subpixel accuracy on thousands of images. We must rely on synthetic data for training, and special cares need to be taken for generating good training data to generalize the model to real

world images. In our work, we address the issue of generating good synthetic training data for achieving good results.

The CNN has also been very successful in blind image deblurring. Instead of summarizing all deblurring algorithms, we only review closely related work here. Our approach is similar to [12][20][21], in which a neural network is trained to directly recover the latent image without estimating the blur kernel. The key is how to prepare enough clear and blur image pairs for training. For realistic motion blur synthesizing, people usually average consecutive frames from slow motion camera [12]. Our ground truth is synthetic low-level curves with subpixel accuracy. The motion blurred image is synthesized by randomly generating motion trajectories, then shift and average clear images along the trajectories. Our loss function is quite different from that in those CNN deblurring algorithms. Instead of building a loss function from the difference between the prediction and the ground truth pixel values, our loss function tries to maximize the maximum contrast value at the ground truth edge locations and minimize the angle difference between the linked curve pixels.

Our work is also related to noisy or faint edge detection [22][23][24]. To the best of our knowledge, [24] is the only work that applies a CNN to detect curves from noisy images. It proposes an edge preservation loss function that minimizes the first order gradient difference between the processed image and the ground truth image. There are two pitfalls of this proposed method: 1. The ground truth image itself may be a degraded image; 2. The first order gradient of a pixel does not represent the edge orientation very well [25]. Compared with [24], our method considers all possible degradation issues related to real world images such as noise, motion blur, luminance change, JPEG compression artifacts and image demosaicing. We provide a complete pipeline to show how to use image subpixel edge information for training and how subpixel edges

could be extracted from the CNN feature map. Our model can sense the smooth curve hidden by noise and motion blur beyond the capability of first order gradient detectors such as the Canny edge detector.

Chapter 3. MEASURING INFANT LENGTH WITH AN IMAGE

It is important to measure an infant's length and growth velocity regularly to make sure that the infant is growing normally. Traditionally, measuring an infant's length requires performing a manual measurement using an infantometer, which consists of a flat surface with a headboard and an adjustable footboard. The infant is placed on the flat surface, and then the head and the legs are straightened so that both touch the boards that are parallel, providing a measure of the infant's length. However, the infant struggles and cries in the process, and it often needs three persons to position the head and the legs of the infant, and to adjust the footboard of the infantometer. During the process, it could also accidently hurt the infant.

Figure 3.1 shows an example of the process of measuring an infant's length using an Infantometer [26].

Figure 3.1. Measuring the infant's length using an infantometer.

Given the complexity of the infant length measurement procedure and the staff requirement, a parent cannot be expected to perform this process alone regularly at home. It is very desirable to have a simple process so that the infant's length could be measured easily by a single person without special training or special equipment and the need to hold and straighten out the infant.

One possibility is to use a photograph taken by a typical cellphone camera with the infant free-moving, and to use computer vision techniques to automatically calculate the infant's length by an algorithm.

However, this is a very difficult problem. A free-moving infant is a dynamic deformable object. The body parts of the infant (e.g., the head and legs) can bend in the 3D space. Also, the lengths of the body parts will appear differently in the 2D image depending on their distances and viewing directions relative to the camera. In order to accurately measure the length of the infant, we need the 3D positions of the joints of the infant's body parts relative to the camera, and physical reference lengths at these 3D positions since the same physical length could appear very differently in 2D images when at different 3D positions. We also need to be able to detect these body part joints accurately and robustly under various lighting conditions. Unfortunately, many body parts important for determining the length of the infant, such as the top of the head (which may be covered by hair), the neck, the shoulder, the joints between the body and the legs, the knee, and the heel, all lack clear and unique feature points. Another severe problem is that infants' movements may cause severe motion blurs of the body parts and joints in the image.

In this chapter, we present an approach aiming to measure an infant's length accurately and automatically from an image that is captured by a regular cellphone camera. In our proposed approach, we use circular stickers of a known-size to mark the feature points (joints of the body parts) and to provide the reference lengths at the 3D positions of those feature points. A circular sticker in the 3D space will be projected into an ellipse with different sizes and shapes in the 2D image depending on its 3D location and orientation relative to the camera. We can thus estimate the 3D positions of the centers of the stickers from the ellipses in the 2D image, and calculate the distances between the centers of the stickers on the infant. We develop a minimal spanning tree

method to automatically determine the order of the ellipses for calculating the total length of the infant. We also developed a blurred-ellipse detection algorithm which can automatically reject the ellipses with motion blur to obtain accurate results. The stickers can be put on the infant easily in a few seconds, and the calculation of the length is fully automatic.

We tested our algorithm on a model (a baby doll) as well as on human infants. The results show that our proposed approach can provide very accurate results and is easy to apply in practical situations.

The contributions of our work in this chapter include:

1. We propose a new approach which enables automatic calculation of an infant's length from a picture taken from a regular cellphone camera.

2. We propose to use a minimal spanning-tree based algorithm to automatically calculate the infant's length.

3. We propose a blurred-ellipse detection method to reject the motion-blurred ellipses caused by the infant's motion while taking the pictures.

4. We perform experiments to show the effectiveness and the accuracy of the proposed methods.

3.1 Proposed Approach

For a deformable object such as an infant, determining its 3D length from a single 2D picture is an ill posed problem due to the reasons mentioned above. Our idea is that feature points are needed, but they do not need to be "natural" feature points on the infant. We proposed to use known-size round stickers placed on the joints on the infant's body to serve as easily identifiable feature points and provide the needed reference physical lengths at the 3D positions of those feature points. With different distances and viewing angles from the camera, the round stickers will appear as ellipses

in different sizes and shapes. By analyzing the ellipses' geometric parameters, we can recover the sticker centers' 3D positions relative to the camera with only one image. Stickers can easily be put on the infant's joints (e.g., heel, knee, hip, shoulder, and head) in just a few seconds, and with the 3D positions of the ellipse centers, the distance between stickers could be calculated. The infant's length could then be calculated by adding up the length of each body part. Since the hair may cover the infant's head, we put a sticker on a hard surface (such as a cardboard or a book) to be pressed against the head to provide the length measurement between the top of the head and the shoulder. With this approach, only off-the-shelf round stickers are needed in the measurement. There is no need to try to detect feature points in the textureless areas, or to provide reference physical lengths at the 3D positions of those feature points. Essentially, the stickers solve the problems of defining feature points, detecting the 3D locations of the feature points, and providing the physical reference lengths at the 3D locations all at once.

To efficiently and reliably estimate the ellipses, we developed a pure color ellipse estimation algorithm called ellipse growing that is more efficient and has a higher success rate than popular open sourced ellipse estimation algorithms such as Hough transform [6], ELSDc [27], and arc segment based detector [28]. This method's advantages will be discussed in next chapter.

After finding the ellipse and the ellipse parameters, the 3D locations of the markers can be estimated. The details about the circular marker's 3D position estimation can be found in [29]. The key steps are described briefly below. Besides using center location (x_c, y_c), major/minor axis length $(2a, 2b)$ and orientation β, an ellipse can be described by the following implicit equation where $A\ B\ C\ D\ E$ and F are constants:

$$Ax^2 + 2Bxy + Cy^2 + 2Dx + 2Ey + F = 0 \tag{3.1}$$

A matrix Q is created as:

$$\boldsymbol{Q} = \begin{pmatrix} A & B & -\frac{D}{f} \\ B & C & -\frac{E}{f} \\ -\frac{D}{f} & -\frac{E}{f} & \frac{F}{f^2} \end{pmatrix} \tag{3.2}$$

where f is the camera's focal length.

Q can be decomposed as:

$$Q = (\boldsymbol{v_1} \quad \boldsymbol{v_2} \quad \boldsymbol{v_3}) \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \begin{pmatrix} \boldsymbol{v_1^T} \\ \boldsymbol{v_2^T} \\ \boldsymbol{v_3^T} \end{pmatrix} \tag{3.3}$$

where $\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}$ and λ_1, λ_2, λ_3 are normalized eigen vectors and eigen values, respectively. Due to the property of the ellipse equation, the eigen values can be arranged to satisfy the following inequalities:

$$\begin{cases} \lambda_1\lambda_2 > 0 \\ \lambda_1\lambda_3 < 0 \\ |\lambda_1| \geq |\lambda_2| \end{cases} . \tag{3.4}$$

The sticker center's 3D position $\boldsymbol{C}$ in the camera's coordinates is then:

$$\begin{cases} z_0 = S_3 \frac{\lambda_2 r}{\sqrt{-\lambda_1\lambda_3}} \\ \boldsymbol{C} = z_0(\boldsymbol{v_1} \quad \boldsymbol{v_2} \quad \boldsymbol{v_3}) \begin{pmatrix} S_2 \frac{\lambda_3}{\lambda_2}\sqrt{\frac{\lambda_1-\lambda_2}{\lambda_1-\lambda_3}} \\ 0 \\ -S_1 \frac{\lambda_1}{\lambda_2}\sqrt{\frac{\lambda_2-\lambda_3}{\lambda_1-\lambda_3}} \end{pmatrix} \end{cases} \tag{3.5}$$

where S_1, S_2 and S_3 are undetermined signs. By requiring the stickers to be in front of the camera, and the sticker surface facing camera, the number of solutions is reduced to two. More information can be found in [29]. As shown in Figure 3.2, those two solutions will have different surface orientations, but very close positions. Specially designed patterns are needed to help to resolve the ambiguity. Otherwise, multiple images are needed to help to find the true position. For our situation, the round sticker's size is very small compared to its distance to the camera center. Thus,

there is no need to differentiate those two solutions: for a sticker of diameter 1.905 cm, camera to sticker distance larger than 20 cm, the difference of those two solutions is less than 0.5 mm. The 3D distance accuracy will not be affected by the sticker's orientation. It is mainly affected by the accuracy of the detected ellipse parameters. After obtaining each sticker's 3D position, we can simply calculate the Euclidean distance of the sticker pairs to get the length between the two sticker centers.

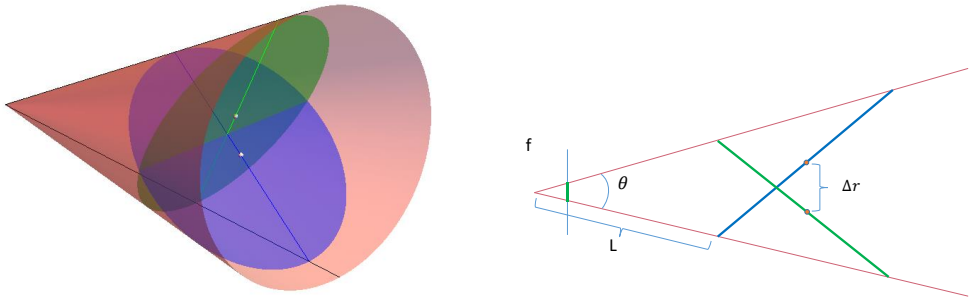


Figure 3.2. The circular marker's 3D location can be estimated from its elliptical shape projection on the image. Left: two circular makers of different orientations could form the same cone relative to the camera center; Right: a 2D side view. For an ellipse on image, there are two possible 3D markers solutions. However, for a typical setup (marker size=1.905 cm, cellphone to camera distance L larger than 20 cm) those two solutions are very close to each other. Their difference (Δr) is usually smaller than 1 mm, thus there is no need to differentiate them.

To measure the infant's length automatically, we need to determine the connecting order of the ellipses automatically. To do this, we use a Minimal Spanning Tree (MST) based algorithm to obtain the connectivity of the stickers.

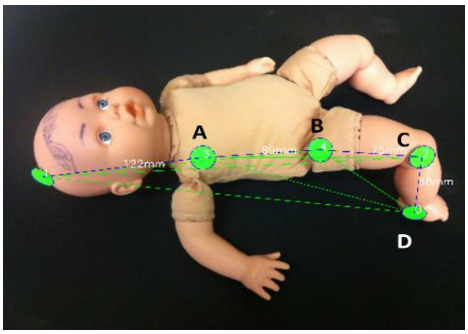


Figure 3.3. Placing round stickers on the infant and using a minimal-spanning-tree based algorithm for automatically calculating the infant's length from an image.

We formulate the connections of stickers as a graph, in which each sticker represents one node. Let N_i be the ith sticker. Since any two stickers are connected by exactly one path in our configuration, we can extract an MST to obtain the connectivity of the stickers. The optimal MST encourages that the path between any two connected stickers should be covered by the infant's body as much as possible. For example, in Figure 3.3, we encourage the connection between Node C and D, while discouraging the connection of A-D and B-D. Therefore, the path between two stickers will be weighted by the portion of the path located on the body as well as the Euclidian distance between the two stickers in 3D. More formally, the weight between any two nodes N_i and N_j is defined as:

$$E(i,j) = \left\| N_i^{3d} - N_j^{3d} \right\| \cdot (1 + \alpha p) \tag{3.6}$$

$$p = \frac{|Pixels\ outside\ the\ body|}{|Pixels\ along\ line(N_i,N_j)|} \tag{3.7}$$

where N_i^{3d} stands for the 3D position of node N_i and $\| * \|$ is the Euclidian distance between the two points. $| * |$ stands for the cardinal number, and α is a constant. To count the number of the pixels outside the body, we first segment out the infant's body from the background based on the skin color. The total length of the MST is the infant's body length. Figure 3.3 shows an example where the green lines are all possible paths, and the blue lines are the selected MST path. Besides

this MST-based method, an alternative method is to take multiple pictures with each picture containing two stickers only, and the total length is the sum of the individual length in each picture.

3.2 Blurred Marker Detection

Sometimes the ellipses in the picture appear blurry due to the infants' movements. The blurred ellipses will have an impact on the accuracy of the final results. To improve accuracy, we develop a blur detection method to automatically reject the images with blurred ellipses. Since we can take multiple images, we have enough clear ellipses for the measurement. To increase the possibility of getting good images while the infant is moving, we can use the burst mode of the camera to take pictures.

Although the topic of image blur analysis has attracted much attention, most previous works focus on detecting blurred images. In [30], Su et al. constructed a new blur metric: singular value feature, and use it to detect the blurred regions of an image. In [31], Liu et al. designed four local blur features for blur confidence and type classification. In [32], Shi et al. studied a few blur feature representations in image gradient, Fourier domain, and data-driven local filters. In our case, even if there are some blurred regions in an image, as long as a sticker in the image is clear, it can still be used to calculate the 3D distance.

The motion blur can be modeled as a convolution with a motion blur kernel. The stickers are put on various backgrounds which have very different pixel values. Due to the convolution and the different pixel values across the ellipse boundary, the regions immediately inside and outside the ellipse boundary usually have relatively large gradients. Since the stickers could be put on many different backgrounds, the gradients outside the ellipse boundary have many variations. On the other hand, since the inside of the ellipse is just pure green color in our case, we can ensure that the gradients inside the ellipse are small and consistent except the region immediately inside

the ellipse boundary. Thus, it is easy to extract a ring with relatively large gradients between the ellipse boundary and the inside region of the ellipse. Depending on the width of the ring, we could determine the extent of the sticker blurriness.

After we extract the sticker masks by setting a color threshold, we calculate the magnitudes of the gradients of the grey-scale image for the pixels inside the sticker masks. The Sobel operator is used in the calculation of gradients**.** After that, we process the gradient images by binarization with an empirical threshold, followed by the Closing operation using a 9x9 kernel, which is an all-ones matrix, to fill the holes. An example for a blurred sticker is shown in Figure 3.4.

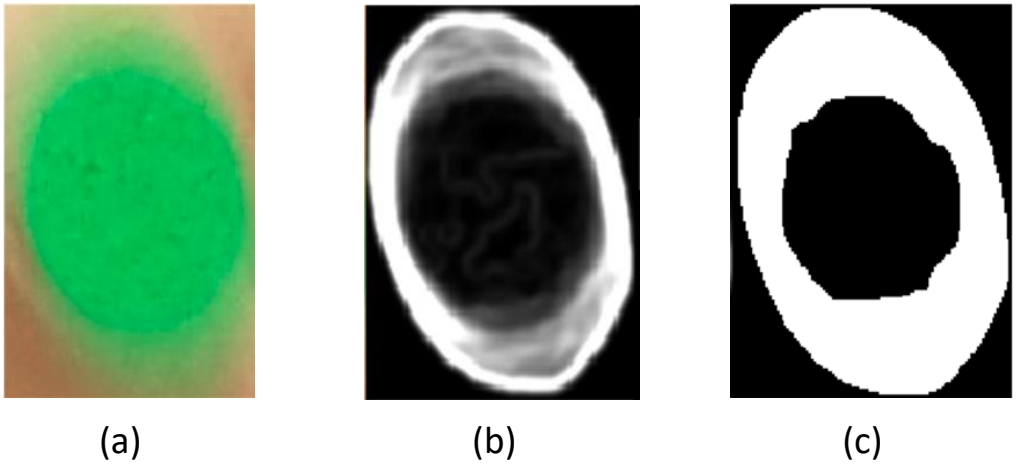


Figure 3.4. Blurred sticker sample. (a) is the original blurred sticker image, (b) is the gradient image, and (c) is the mask of the gradient image after binarization and morphology processing.

To make the process simple, we approximate the gradient mask (e.g., in Figure 3.4 (c)) by an elliptical ring, which is the blue part shown in Figure 3.5. We define the inner ellipse as the internal contour of the ring, and the outer ellipse as the exterior contour of the ring. If the elliptical ring is wider, the corresponding sticker is more likely to be blurred.

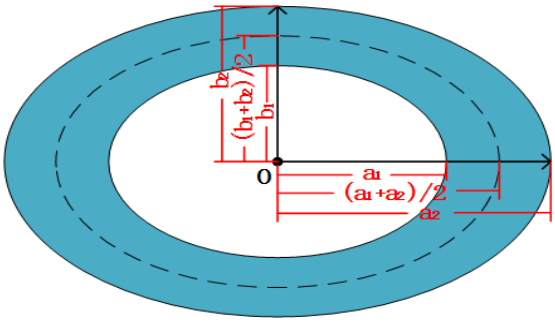


Figure 3.5. The elliptical ring used to decide blurriness.

Suppose the long axis of the inner and outer ellipse are a_1 and a_2, and the short axis of the inner and outer ellipse are $b_1$and b_2, respectively, where $a_1 \geq b_1 > 0$, $a_2 \geq b_2 > 0$, $a_2 > a_1$ and $b_2 > b_1$. The middle dashed ellipse can be regarded as the skeleton of the elliptical ring, which can be obtained by skeletonization of the elliptical ring, and the long and the short axis of it are $(a_1 + a_2)/2$ and $(b_1 + b_2)/2$. Thus, the area A of the elliptical ring is

$$A = \pi(a_2 b_2 - a_1 b_1) \tag{3.8}$$

The perimeter P of the dashed ellipse is

$$P = \pi(b_1 + b_2) + 2(a_1 + a_2 - b_1 - b_2) \tag{3.9}$$

and the Area/Perimeter (A/P) ratio is

$$A/P = \frac{\pi(a_2 b_2 - a_1 b_1)}{\pi(b_1+b_2)+2(a_1+a_2-b_1-b_2)} \quad . \tag{3.10}$$

To simplify the equations, we denote $a_1 = n\, b_1$, $a_2 = n\, b_2$, $a_2 = m\, a_1$ and $b_2 = m\, b_1$, where $n \geq 1$ and $m \geq 1$, so the A/P ratio in Equation (2.10) can be simplified into

$$A/P = \frac{(m-1)n\pi b_1}{\pi+2(n-1)} \tag{3.11}$$

In Equation (3.11), for a given elliptical ring, n and $b_1$are fixed. If m is larger, the elliptical ring is wider. A/P estimates how thick the ellipse ring is. For accurate ellipse estimation, smaller A/P is preferred.

In our experiments, we tested our ellipse blur detection method on 30 infant images from the dataset we collected in our field trials. To evaluate the accuracy, we manually estimated the stickers and classified the stickers as clear stickers or blurred stickers. First, we classified the images. If there were only clear stickers in an image, the image was classified as a clear image. Otherwise, the image was classified as a blurred one. Among the 30 images we tested, 15 images were clear and 15 images were blurred. Next, we classified the stickers. There were 86 stickers in these 30 infant images. If the Blur Ratio > threshold, the sticker was regarded as blurred; otherwise, the sticker was regarded as clear. We set the threshold as 20. The accuracy for the sticker classification was 95.3%, and only 4 stickers were incorrectly estimated, which may be due to the illumination and shadow effects.

Although four stickers were incorrectly classified, the 30 infant images tested were all correctly classified, so the accuracy of image classification was 100%. This is because the image is classified as a clear image only if all the stickers are classified as clear.

3.3 Experimental Results

3.3.1 *Calibration of the proposed method as a virtual ruler*

Before comparing the measurements of length using the proposed method and using an infantometer, we established the precision of the proposed sticker method. For this purpose, we put six stickers on an Infantometer. Three stickers were put on the headboard to determine the headboard plane, and the other three at 20 cm, 30 cm, and 40 cm positions on the flat base, respectively. Here the sticker's correct orientation out of two possible solutions could be selected automatically using the fact that the headboard stickers and the ruler stickers are coplanar. By

selecting the correct solution, the maximum 0.5 mm error mentioned previously can be circumvented.

We measured the distance from each of the three stickers on the headboard plane to each of the three stickers on the base of the infantometer to estimate the precision of our proposed method. As shown in Table 3.1, our proposed method is very accurate. The maximum measurement error is 0.11 cm, and the average is 0.058 cm (below 1 mm). This demonstrates that with clear images and precise ellipse detection, our proposed approach could achieve very high accuracy.

Table 3.1. length measurement (in cm) using stickers compared to infantometer. Three images were taken from different angles

Sticker Location	Image No	Dist2Headboard	Err	Err%
20 cm	1	19.89	0.1	0.6%
	2	19.98	0.02	0.1%
	3	19.97	0.02	0.1%
30 cm	1	29.92	0.08	0.3%
	2	29.96	0.04	0.1%
	3	30.07	0.07	0.2%
40 cm	1	39.94	0.06	0.15%
	2	40.03	0.03	0.08%
	3	40.11	0.1	0.3%

Next, to evaluate the accuracy in a controllable environment, we put stickers on a baby doll's joints. We used a book touching the doll's head. The length between the head to the shoulder is estimated by calculating the shoulder sticker's distance to the book plane. In this baby doll experiment, only one image is used to estimate the total length, since all stickers were captured in one image. One person could take the pictures easily.

The final infant length is:

$$L_{total} = L_{head2shoulder} + L_{shoulder2hip} + L_{hip2knee} + L_{knee2heel} + r_{sticker} . \quad (3.12)$$

As shown in Table 3.2, the final result has only about 0.97% error compared to the ground truth measured by a ruler.

Table 3.2. Baby doll measurement with ¾” green stickers. (result in mm), Ground truth (GT) obtained by real ruler.

NO.	Head2Breast	Breaset2Hip	Hip2Knee	Knee2Heel	Total
1	111.55	95.2	73.83	65.83	346.41
2	111.11	94.40	71.91	65.69	343.11
3	113.67	95.09	73.03	65.14	346.93
4	114.47	96.71	73.41	63.91	348.50
5	112.03	97.18	72.19	64.43	345.83
6	111.09	96.44	72.21	67.28	347.02
7	112.15	94.46	74.62	66.83	348.06
8	112.91	95.44	73.29	65.32	346.96
AVG	114.24	95.50	73.00	65.49	348.24
STD	1.22	1.04	0.93	1.13	1.64
GT	114	96	73	67	350
diff%	1.43%	0.40%	0.084%	2.16%	0.97%

3.3.2 *Field Test*

We also apply the proposed method to real infants in practical settings. Since it is more difficult to take a single clear picture that contains all stickers, we take pictures of knee to heel, hip to knee, shoulder to hip, and head to shoulder. Different types of pictures were uploaded to different folders and the computer automatically calculates the individual length and sums the individual lengths to get the final total length.

The size of the sticker in the image can also affect the accuracy of the measurement. Although theoretically we could recover circular stickers of any orientation and distance, the result depends on the accuracy of the ellipse parameters. If a circular sticker is too small in the image, then even a small deviation in the ellipse estimation will cause noticeable errors in the 3D position recovery. Thus, it is preferred to use a reasonably large sticker. However, if the sticker is too large, it is not convenient to use. We used a 3/4” (1.905 cm) sticker, which is a suitable size. Also, we made the stickers appear large in the pictures taken.

Another issue is the location of the stickers. For example, a sticker could be put on the outer side of a leg or on the inner side of the leg. The principle is to place the stickers close to the joints and observe if the distance between two neighbor sticker pairs is affected when the baby moves.

We compared the infant length measurements using both the proposed method and the infantometer method of the same day (see Table 3.3). We averaged the results if there were more than one clear photo available. The errors are close to 3%.

Table 3.3. Field test compared with infantometer (in cm). sticker diameter is 1.905 cm.

Sub	Head to Shoulder	Shoulder to Hip	Hip to Knee	Knee to Heel	Total	GT	Err
1	16.77	18.30	11.88	13.98	61.89	60.0	3.15%
2	16.74	20.44	15.41	13.25	66.80	66.5	0.5%
3	19.7	31.46 *		14.27	66.38	65.0	2.1%
4	18.51	16.03	13.58	13.70	61.90	64.0	3.3%
5	15.00	19.23	11.92	13.43	59.65	64.3	7.2%
6	17.70	20.58	13.96	14.07	67.27	67.45	0.27%

* Combined result.

Chapter 4. ELLIPSE GROWING FOR ESTIMATING CIRCULAR MARKER'S 3D LOCATIONS

Accurate estimation of the ellipse parameters is critical for the accurate estimation of the 3D length in our proposed approach. In this chapter, we describe our ellipse growing algorithm used for ellipse estimation, and compare our proposed algorithm to other state-of-the-art ellipse estimation algorithms.

The key to improve the marker's location estimation in actual images is to alleviate the effects of blurriness and noise. For small linear motion blur, majority of the ellipse boundaries could still be accurately located by the edge. Thus, we propose a method called ellipse growing that is tolerant to mild motion blur.

We take advantage of the sticker's color to develop an ellipse estimation algorithm which performs better and faster than using the Hough transform approach [6] and the edge chaining base method [28]. Our ellipse growing method could be simply regarded as a variant of the active contour model [33]. The major difference is that the curve remains the ellipse shape all the time during evolving. Below are details of our proposed ellipse growing method.

4.1 PROPOSED ELLIPSE GROWING METHOD

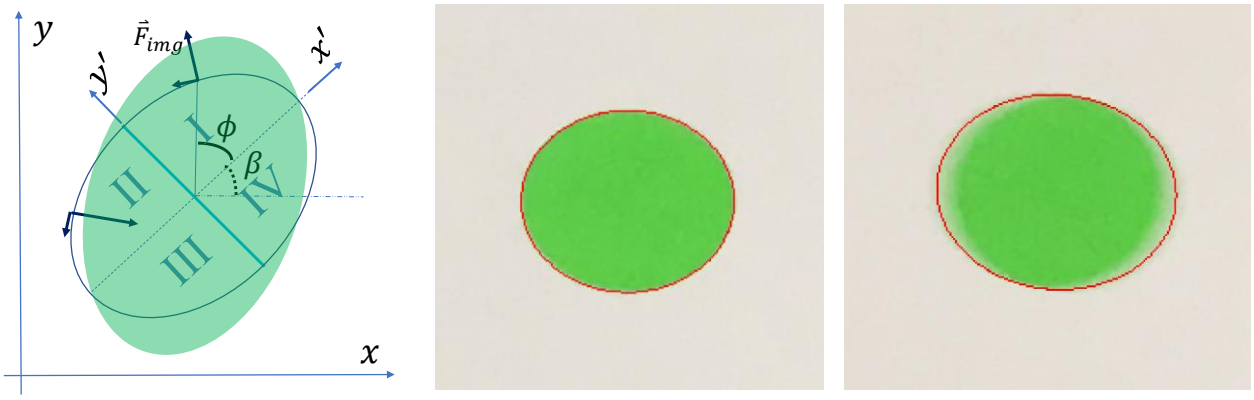
First, we set up initial seed ellipses in the image. There are several ways to set up the initial seeds. Since we use green stickers, one way is to blur the image's green channel with large Gaussian kernel, for example 91×91 kernel for 3264×1836 images, then use local peaks as initial seeds. Another way is to set seeds densely distributed in the image, then remove those non-green seeds. Since different seeds could grow into the same ellipse, the repeated ellipses will be removed later.

We let the seeds expand until they fit the contour of stickers. The contour of a sticker is defined as the edge of green color after thresholding.

Let (x_e, y_e) denote a point on the ellipse in the image coordinate system for the image plane, which has the coordinate (x'_e, y'_e) in the ellipse's local coordinate. Each ellipse starts with a very small circle, for example a 5 pixels radius circle. An ellipse is described by its major and minor radius: a and b, orientation angle β, and ellipse center coordinate: x_c and y_c. It can be represented as:

$$\vec{r}_e(\phi) = \begin{pmatrix} x_e(\phi) \\ y_e(\phi) \end{pmatrix} = \begin{pmatrix} cos\,\beta & -sin\,\beta \\ sin\,\beta & cos\,\beta \end{pmatrix} \begin{pmatrix} a\,cos\,\phi \\ b\,sin\,\phi \end{pmatrix} + \begin{pmatrix} x_c \\ y_c \end{pmatrix} \tag{4.1}$$

$$\phi \in [0, 2\pi)$$



(a) (b) (c)

Figure 4.1. Ellipse growing and the result on real images. (a) Illustration of image force exerted on ellipse curve. β is the orientation of major ellipse axis. Ellipse curve in its local coordinate is represented as $(x'_e, y'_e)^T = (a\,cos\,\phi\,,\ b\,sin\,\phi)^T$. In global (image) coordinates it is represented as in Equation (4.1). (b) Detection result of a clear marker. (c) Detection result of a blurred marker. We can see that the clear side of the boundary prevents the ellipse from shrinking too much.

The ellipse is modeled as an expandable and movable loop. A force is exerted on this loop to make it expand and move until it finds the optimal size and location that covers the sticker. The choice of force is quite flexible. In our implementation, for example Figure 4.1(a), for each point

on the ellipse boundary, the image force can be broken into two directions: tangential and perpendicular. The perpendicular force points outward if there are green color pixel values larger than a threshold outside the curve. It points inward if there are non-green color pixel values inside the curve. Perpendicular force magnitude could be chosen as a constant. For our case, we use 10. To determine the tangential force, the ellipse is divided into four parts which belong to four quadrants of the ellipse's local coordinate system. In quadrant I and III, the tangential force is a 90-degree counterclockwise rotation of the perpendicular force; while in quadrant II and IV the tangential force is a 90-degree clockwise rotation of the perpendicular force. The tangential force is used to adjust the ellipse orientation. Its magnitude is set to be a small value to avoid oscillations during updates, and usually ten percent of the magnitude of the perpendicular force would be good. After the ellipse boundary reaches the green color threshold, $\vec{F}_{img}$ could be replaced with the gradient of the blurred edge to encourage the final ellipse rest on the sticker's edge.

With a properly chosen image force, this method could be directly applied to fit a pure color ellipse or used in combination with other methods to perform fine parameter tuning.

The complete updating equations are:

$$\begin{aligned}
\pi(a^2+b^2)\ddot{\beta}+\tfrac{\gamma_\beta}{2}(a^2+b^2)\dot{\beta} &= \textstyle\int_0^{2\pi}\vec{F}_{img}(\vec{r}_e(\phi))\cdot\frac{\partial\vec{r}_e(\phi)}{\partial\beta}d\phi \\
\ddot{x}_c+\tfrac{\gamma_{x_c}}{2\pi}\dot{x}_c &= \textstyle\int_0^{2\pi}\vec{F}_{img}(\vec{r}_e(\phi))\cdot\frac{\partial\vec{r}_e(\phi)}{\partial x_c}d\phi \\
\ddot{y}_c+\tfrac{\gamma_{y_c}}{2\pi}\dot{y}_c &= \textstyle\int_0^{2\pi}\vec{F}_{img}(\vec{r}_e(\phi))\cdot\frac{\partial\vec{r}_e(\phi)}{\partial y_c}d\phi \\
\pi\ddot{a}+\tfrac{\gamma_a}{2}\dot{a} &= \textstyle\int_0^{2\pi}\vec{F}_{img}(\vec{r}_e(\phi))\cdot\frac{\partial\vec{r}_e(\phi)}{\partial a}d\phi \\
\pi\ddot{b}+\tfrac{\gamma_b}{2}\dot{b} &= \textstyle\int_0^{2\pi}\vec{F}_{img}(\vec{r}_e(\phi))\cdot\frac{\partial\vec{r}_e(\phi)}{\partial b}d\phi
\end{aligned} \tag{4.2}$$

The integration part is replaced by summation in the implementation, " ˙ " and " ¨ " are the first and second order derivative to time, γ_β, γ_{x_c}, γ_{y_c}, γ_a and γ_b are coefficients of "friction", that slow down the updating to avoid oscillations. In our implementation, we use 100 as starting "friction" then increase it by 0.2 for each iteration. This set of differential equations can be solved by the simple finite difference method:

Assume $\phi \in [0,2\pi)$ is evenly divided into N parts, and each part is of size $\Delta\phi = \frac{2\pi}{N}$; the discretized version of the first equation in (4.2) is:

$$\pi(a^2+b^2)\,\frac{\beta^{t+1}-2\beta^t+\beta^{t-1}}{\Delta t^2}+\frac{\gamma_\beta}{2}(a^2+b^2)\,\frac{\beta^{t+1}-\beta^{t-1}}{2\Delta t}=\sum_{i=1}^{N}\vec{F}_{img}(\vec{r}_e(\phi_i))\cdot\frac{\partial\vec{r}_e(\phi_i)}{\partial\beta} \qquad (4.3)$$

In Equation (4.3), $\phi_i = 2\pi(i-0.5)/N$. Other equations will have a similar format, except different coefficients. When choosing the time step, it is preferred to diminish the time step during updating. Δt could start with value 0.5, then reduced to 0.1 in the last 1/3 of iterations. Usually 2000 iterations are sufficient for an ellipse size smaller than 200 pixels. The comparison between clear and blurred detected markers are shown in Figure 4.1 (b) and (c). We could see that our ellipse growing strategy could prevent the ellipse curve from being overly distorted by mild motion blur.

4.2 EXPERIMENTS AND COMPARISONS

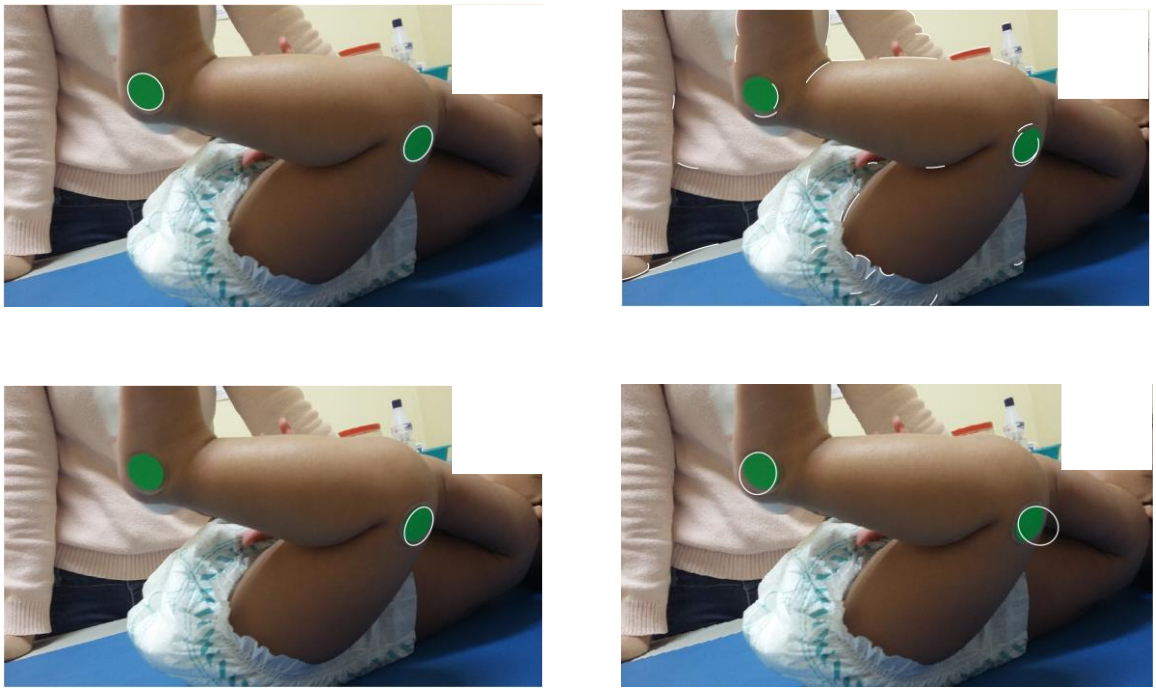


Figure 4.2. Comparison of different ellipse detection methods: (a) Our ellipse growing method; (b) ELSDc [27], we only keep the detected arcs and overlay them onto the image; (c) One of the state-of-the-art ellipse detector [28]. We tried Gaussian smoothing with $\sigma = \{1.0, 2.0, 3.0\}$ and median filter of size {1,2,4,5,6,7}. (c) is the best result we can get. (d) Hough transform [6], the algorithm returns too many false positives, we only display the one with the highest vote. Note: except ELSDc, all other three methods detect ellipse in a small patch near the marker. This small patch could be obtained by drawing a window around the green color seed (see paragraph 4.1).

Figure 4.2 shows the detection result by three methods (Ellipse growing, ELSDc [27], ellipse detector proposed in [28], and Hough transform[6]) on real world images used in the infant length measurement project. This image does not even have noticeable motion blur. However, only our method works well. All other methods do not perform well.

To compare the quantitative performances among different methods, we take several pictures of three green circular markers. The distances between markers were measured with rulers. Since the relation between ellipse parameters and the 3D location has a closed math form, accurate ellipse parameters will lead to an accurate 3D distance estimation. We evaluate the performance of

different methods by comparing the estimated 3D distances to the ground truth measure by rulers. We compare the results produced by the Hough transform [6], one of the state-of-the-art ellipse detectors described in [28], and the ellipse growing method. There are three green markers for each image. The ground truth is obtained by measuring the distance between each marker using a ruler (as shown in Figure 4.3). Hough transform returns too many false positives. We have to manually set the threshold to pick proper ellipses to perform the 3D distances calculation. The ellipse detector in [28] also failed to automatically detect the ellipse if the original resolution image is used. We have to crop out a small rectangular region. Ellipse growing recovers very accurate 3D distances when the input image is clear, and it is still able to recover decent results when motion blur is small. The quantitative comparison results are shown in Table 4.1 and Table 4.2. The value in each entry is calculated as:

$$\frac{\left|d_0-d_{0_{gt}}\right|+\left|d_1-d_{1_{gt}}\right|+\left|d_2-d_{2_{gt}}\right|}{3} \tag{4.4}$$

where d_0, d_1 and d_2 are estimations, $d_{0_{gt}}$, $d_{1_{gt}}$ and $d_{2_{gt}}$ are the ground truths.

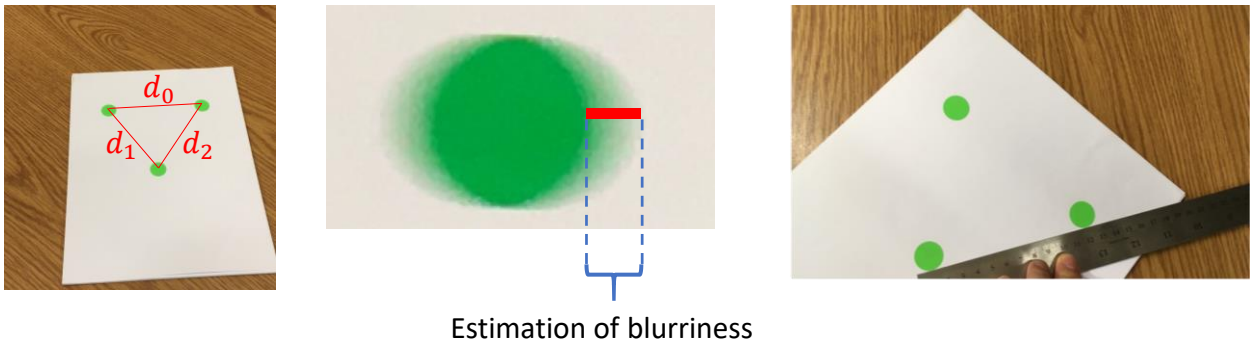


Figure 4.3. Real testing images and the estimation of blurriness. (a) An example image with three circular markers, d_0, d_1 and d_2 are 3D relative distances that need to be estimated from the marker's ellipse parameters. Note, this is a clear image, other images may have different extent of motion blur. (b) How blurriness is defined. (c) Using a ruler to get the ground truth distance. Test image resolution: 3024×4032.

Table 4.1. 3D distance average recovery error comparison (Equation (4.4)). (unit is mm)

Image number	Blurriness	Hough[6] with manual selection	[28]	Ellipse Growing
0	Clear	5	0.75	**0.7**
1	8.5 pixels	30	5.25	**2.7**
2	9.8 pixels	10	**5.25**	6.55
3	11.2 pixels	14.45	**5.25**	7.3
4	12.6 pixels	8.5	**7.8**	10
	Average	13.6	**4.9**	5.5
	STD	9.8	**2.5**	3.7

Table 4.2. Running time comparison.

	Hough[6]	[28]	Ellipse Growing
Running time	5 minutes	8 seconds	3.5 seconds

Our method is the most accurate when motion blur is very small (<9.8 pixels). When motion blur is larger (>=9.8 pixels), [28] is more accurate. However, [28] could not handle field test images even when there is no obvious motion blur (Figure 4.2). Overall, our ellipse growing method is a reliable and accurate method to estimate the shape of pure color markers in real photos when motion blur is not very strong. To handle stronger motion blur, we develop another method based on a CNN (Convolutional Neural Network) which will be discussed in the next chapter.

Chapter 5. RESTORATION OF ELLIPSES USING A CNN

5.1 PROBLEM STATEMENT

In the field trials we described in Chapter 2 to test our proposed approach, we found it can provide accurate measurements, but several factors can affect the accuracy of the proposed technique. A problem is that the infant moves during the picture taking, which could cause severe motion blur. The severe motion blur could cause the ellipse-parameter estimation inaccurate. One way to solve this problem is to reject the blurry ellipses we used in Chapter 3. Another way is simply taking more pictures to increase the chances of having clear images (for example when the object changes motion direction). A technically more interesting way is to deblur the image before the ellipse parameter estimation, since few publications studied how deblurring algorithm can improve geometric feature estimation. However, traditional deblurring methods may introduce ringing effects [34][35], which makes accurate ellipse edge estimation difficult. Besides, most of the state of the art deblurring algorithms fail if the image contains strong noise. Applying a smoothing algorithm before deblurring usually affects the deblurring accuracy [36]. For accurate recovery of the ellipse parameters, the deblurring algorithm itself should be noise robust. Only a few publications explicitly considered noisy blurred images [36][37][38]. However, [36] relies on inverse Radon transform which requires straight edges in the latent image. In [37], it requires more than one image, and [38] assumes that the blur kernel is known. Also, an important issue is that in our application, we need the correct locations of the ellipses, not just to deblur the ellipses.

Another problem is that sometimes, because of poor lighting, the detected stickers' boundaries may not be continuous due to camera's sensor noise. Mild Gaussian smoothing could help to remove noise without causing serious edge shift. However, for motion-blurred and poor-lighting

images, edge locations are not easy to recover. Almost all ellipse detection and fitting methods rely on accurate edges as input. Directly applying existing ellipse estimation methods will not result in good ellipse parameter estimations. Both edge-grouping and Hough-transform-based methods failed. Errors caused by the motion blur under poor lighting could not be removed by outlier handling methods such as [39], since all edges' locations are affected.

Motivated by the success of Convolutional Neural Network (CNN), in this chapter, we propose to use a CNN to restore the ellipses before ellipse detection and parameter estimation. Through simulations, we found that for our infant-length measurement application, the data used in training the CNN affect the accuracy of the measurements significantly. We propose a synthetic data generation pipeline that takes into account effects in generating realistic training data, to restore the ellipses for automatic infant length measurements. Since our task is to accurately recover the ellipse edge locations, not the whole latent image, our CNN structure is light-weighted. The results confirm the effectiveness of our proposed approach.

5.2 PROPOSED NETWORK

The proposed CNN structure for restoring the degraded ellipses is shown in Figure 5.1. Our CNN structure is inspired by [40]. The main differences are that we added sigmoid functions after most of the convolutions layers to increase the nonlinearity, and we do not use a very deep structure. Our use of the sigmoid function is inspired by [41]. It resembles the purpose of $tanh$ function in the paper but gives us better results.

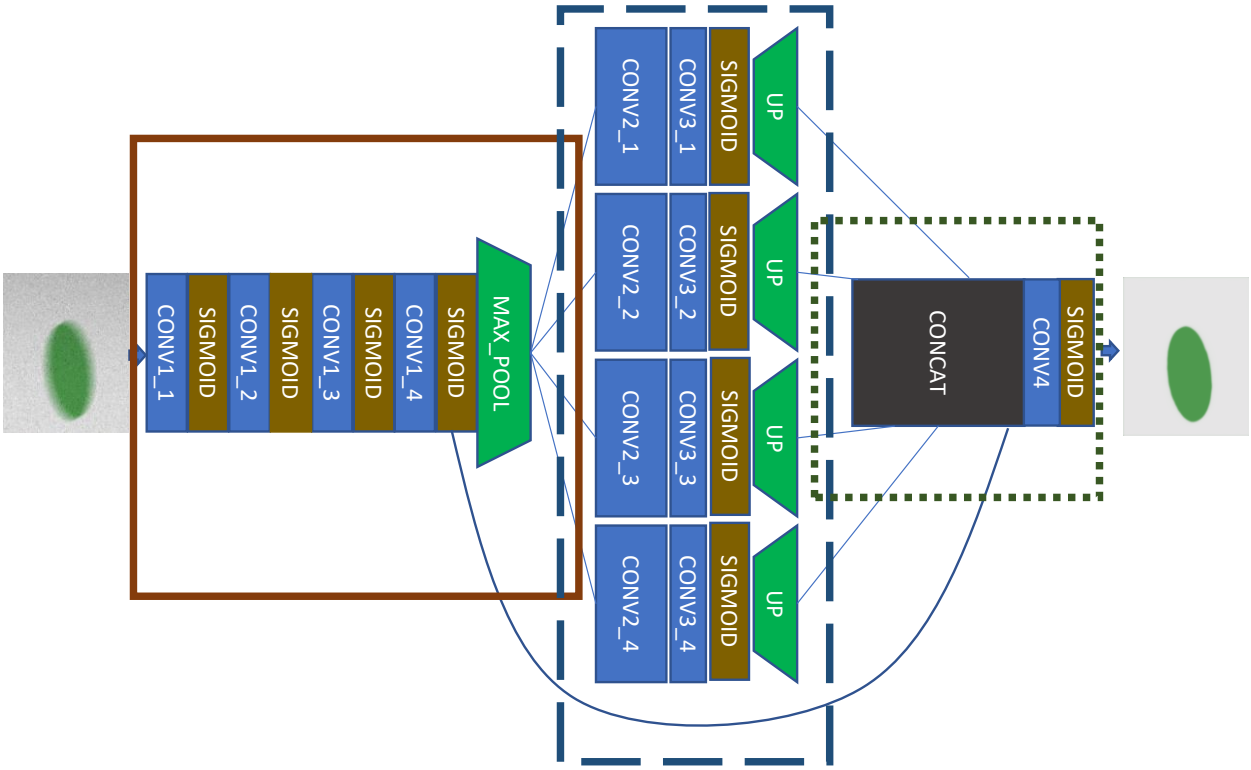


Figure 5.1. Proposed CNN structure.

The structure could roughly be considered by the three parts enclosed by solid, dashed and squared dotted rectangles as shown in Figure 5.1. The first part of the structure consists of four convolutional layers. After max pooling, the image's resolution is reduced to half in both horizontal and vertical dimensions of its original. The second part contains four parallel convolution layers to extract information from different scales. Since the resolution has been halved in both dimensions, four up-sampling layers are used at the end of the second part. The third part concatenates the results of part one and part two, then followed by a three-channel convolution layer and a sigmoid function. Parameters of each convolutional layers are listed in Table 5.1.

Table 5.1. Convolution layer parameters

Layer Name	CONV1_1	CONV1_2	CONV1_3	CONV1_4
Filter size	7 × 7	5 × 5	5 × 5	3 × 3

Filter numbers	16	32	32	64
Stride	(1,1,1,1)	(1,1,1,1)	(1,1,1,1)	(1,1,1,1)
Receptive field	7×7	11×11	15×15	17×17

Layer Name	CONV2_1	CONV2_2	CONV2_3	CONV2_4
Filter size	1×1	3×3	3×3	3×3
Filter numbers	16	16	16	16
Stride	(1,1,1,1)	(1,1,1,1)	(1,1,1,1)	(1,1,1,1)
Dilation	1	2	4	8
Receptive field	18×18	20×20	24×24	32×32

Layer Name	CONV3_1	CONV3_2	CONV3_3	CONV3_4	CONV4
Filter size	1×1	1×1	1×1	1×1	5×5
Filter numbers	1	1	1	1	3
Stride	(1,1,1,1)	(1,1,1,1)	(1,1,1,1)	(1,1,1,1)	(1,1,1,1)
Receptive field	18×18	20×20	24×24	32×32	NA

The motivation of using this structure is that an ellipse is much simpler than a general image. Thus, the network does not need to be too deep or too complicated as in [40], [12], or [41]. An ellipse has smoothly curved boundaries. To recover the edge location, there is no need to use the global information of the image. However, the receptive field still needs to be large enough to cover the blur-kernel size and to minimize the effect of noise. For our case, a 32×32 receptive field is large enough to handle the blurred ellipses. We could remove the max pooling and up-sampling layers. However, this will increase the computation time, and the results do not get better.

5.3 SYNTHESIZE TRAINING DATA

For deep learning, one of the most important parts is collecting and labeling a sufficient amount of data for training. Collecting and labeling data could be very time-consuming. For us, accurate pixel and even sub-pixel level edge location is needed. We need the blurred ellipse and the corresponding clear ellipse pairs for training. The strategy of [12] to take several pictures to synthesize the blurred ones does not apply to our case.

To synthesize an ellipse and its degraded version close to that observed in the practical situations, we simulate the process of the ellipse formation and degradation. We limit our image size to 256×256, the stickers' color to green, and the background color to white. We synthesize the training data with the following four steps considering the actual ellipse generation and degradation process:

Step 1. Generate the motion-blurred ellipse mask. We first generate random ellipse parameters, motion directions, and motion magnitudes. We evenly divided the motion range to get 101 locations, then generate a clear 0/1 ellipse mask for each location. The motion-blurred ellipse mask is obtained by averaging all those 101 clear masks. The final ellipse mask has a value between 0 and 1. To account for the partial area effect [42], the mask is first generated in a high resolution (we use 2560×2560). It is then blurred by a Gaussian kernel of size 7x7 with standard deviation of 2, then downsampled to 256×256.

An ellipse can be described as:

$$h(x, y, \theta, x_0, y_0, a, b) = \frac{(cos\theta(x-x_0)+sin\theta(y-y_0))^2}{a^2} + \frac{(-sin\theta(x-x_0)+cos\theta(y-y_0))^2}{b^2} = 1 \quad (5.1)$$

where θ is the ellipse's orientation, (x_0, y_0) is the ellipse's center, and a and b are parameters for major and minor axes. So, the clear ellipse mask can be written as:

$$f_{mask}(i, j, \theta, x_0, y_0, a, b) = \begin{cases} 1, h(i, j, \theta, x_0, y_0, a, b) \leq 1 \\ 0, h(i, j, \theta, x_0, y_0, a, b) > 1 \end{cases} \quad (5.2)$$

where (i, j) is the pixel's location. The motion blurred mask is:

$$f_{blurred}(i, j) = \frac{1}{2N+1} \sum_{n=-N}^{N} f_{mask}(i, j, \theta, x_0', y_0', a, b) \quad (5.3)$$

$x_0^{'} = x_0 + n \cdot \delta l \cdot cos\phi$ and $y_0^{'} = y_0 + n \cdot \delta l \cdot sin\phi$, $(cos\phi, sin\phi)$ is the motion direction, and the total moved length (in pixels) is $l = 2 \cdot N \cdot \delta l$ (in our case, *N=50*).

Step 2. Add noise and light intensity variation. We add zero-mean Gaussian noise with a magnitude of 0.2, standard deviation of 1, and use Error function (integral of a Gaussian function) as the shape of light intensity variation (caused by shadow). Image pixel values first are multiplied by the light intensity profile. Then, each R, G, B channel is applied with independent Gaussian noise. We also randomly control the percentage of pixels that are affected by noise.

Step 3. Image demosaicing. For a regular camera, at each pixel location only one of R, G, B is directly read from the sensor. Other values are estimated from its neighbors' values by interpolation. The reason to simulate demosaicing is because demosaicing will add artifact to otherwise smooth edges, and the noises of the neighbor pixels are no longer independent. Pixel or subpixel accuracy of the estimated ellipse is desirable. It is better to simulate those artifacts, so that the CNN can learn how to correct it.

Step 4. Gamma correction and pixel value quantization. Pixel values in all previous operations were represented using float numbers within the range [0,1]. For a JPEG image, modern digital cameras use the sRGB space where the RGB values went through a Gamma correction process with $\gamma = 2.2$ before the quantization and compression process. To make the synthesized data as close to those in the actual images as possible, we also perform the same Gamma correction process (i.e., the RGB colors were first gamma corrected in each channel, $R_\gamma = R^{1/\gamma}$, $G_\gamma = G^{1/\gamma}$, $B_\gamma = B^{1/\gamma}$). Then, all values are quantized to integers between 0 to 255, represented by eight-bit unsigned integers. When feeding the training data into our model, we first change the data type to float and rescale the pixel values to the range $[0, 1]$, then apply inverse gamma correction to make the pixel values proportional to the luminance. We show from simulations that the results are better with this Gamma correction.

The whole process of automatically generating the degraded ellipses for training the CNN is shown in Figure 5.2. We could see that after applying the demosaicing effect, noise became less (because noise corresponds to the missing (RGB) color is dropped), and noise granularity become larger because of interpolation. Also, the ellipse boundary is less sharp than before. After quantization, the pixel values are not as accurate as before. However, this is not noticeable to human eyes. The resultant degraded ellipses look very similar to those observed in the practical images we encountered.

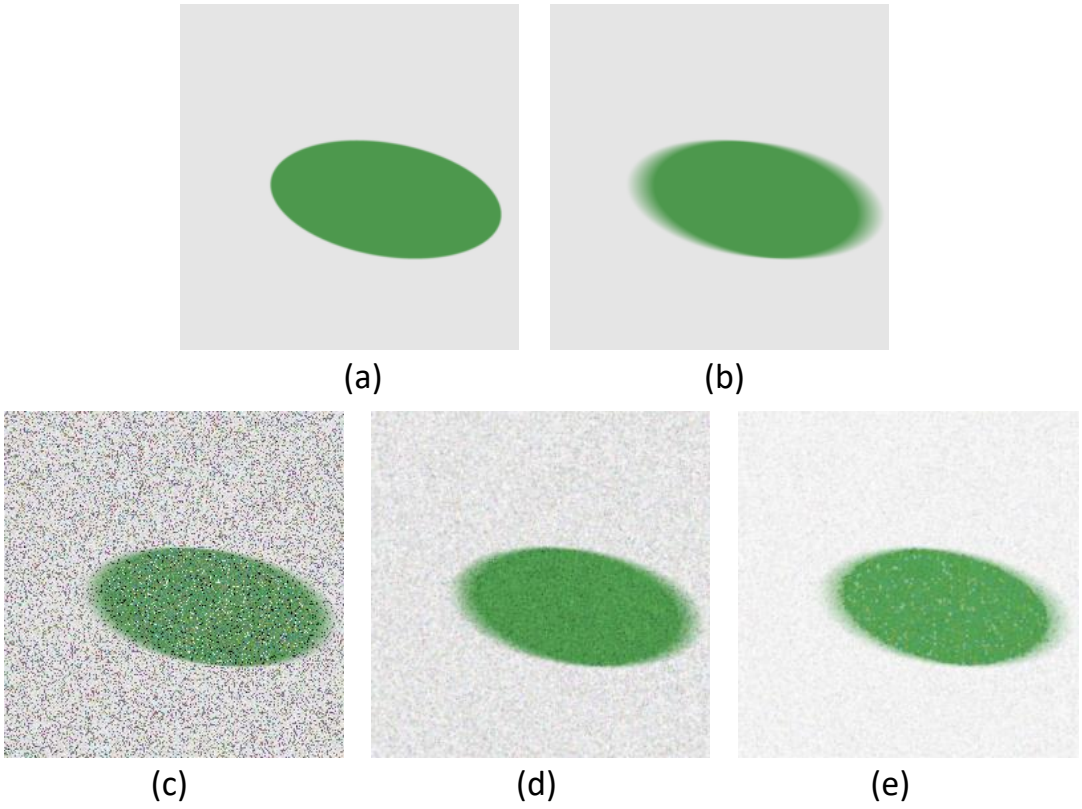


Figure 5.2. From clear ellipse to final degraded input image. (a) Clear ellipse, (b) blurred, no noise, (c) blurred image with noise, (d) after considering the demosaicing effect, (e) after gamma correction and pixel value quantization. Note: Image saturation adjusted for better viewing in black and white.

5.4 TRAINING AND EXPERIMENTAL RESULTS

To train the CNN model, we generated 30,000 blurred ellipses of random sizes, orientations, motion directions, and motion ranges. We choose the batch-size to be 16. Every time 16 images are randomly drawn from images containing the 30000 blurred ellipses, then they are applied with noise, random lighting variation, demosaicing effect, Gamma correction and pixel value quantization. All parameters in the CNN were randomly initialized. The energy function is chosen to be:

$$loss = \frac{\sum_{All\ pixels}\left(x_{recv}(i,j) - x_{gt}(i,j)\right)^2}{L_x \times L_y} \tag{5.4}$$

where $x_{recv}(i,j)$ is the pixel value at the (i,j) position in the recovered image, $x_{gt}(i,j)$ is the pixel value at the (i,j) position in the ground truth image, and $L_x \times L_y$ is the image size. Since the shape of the sticker is simple, the training only takes about 4,000 steps with Adam optimizer.

5.4.1 *Restoration with synthetic data*

A comparison of noisy blurred input, restored ellipses, and the ground truth ellipses is shown in Figure 5.3. Subjectively, there is no obvious difference between the restored ellipses and the ground truths. The only noticeable difference is that the recovered ellipse edge is not as sharp as the ground truth. Figure 5.4 (a) and (b) show the difference between blurred ellipses, the restored ellipses, and the ground truths. For the restored ellipses, the errors mainly locate at near the ellipse boundaries.

After obtaining the restored ellipses, we use the Taubin's method [43] with outlier removal [39] to estimate the parameters of all ellipses for comparisons. Figure 5.4 (c) shows ellipse fitting

results of the CNN processed image and the motion blurred image. Ellipses fitted from the restored images almost entirely overlap with the ellipse fitted from the ground truth images.

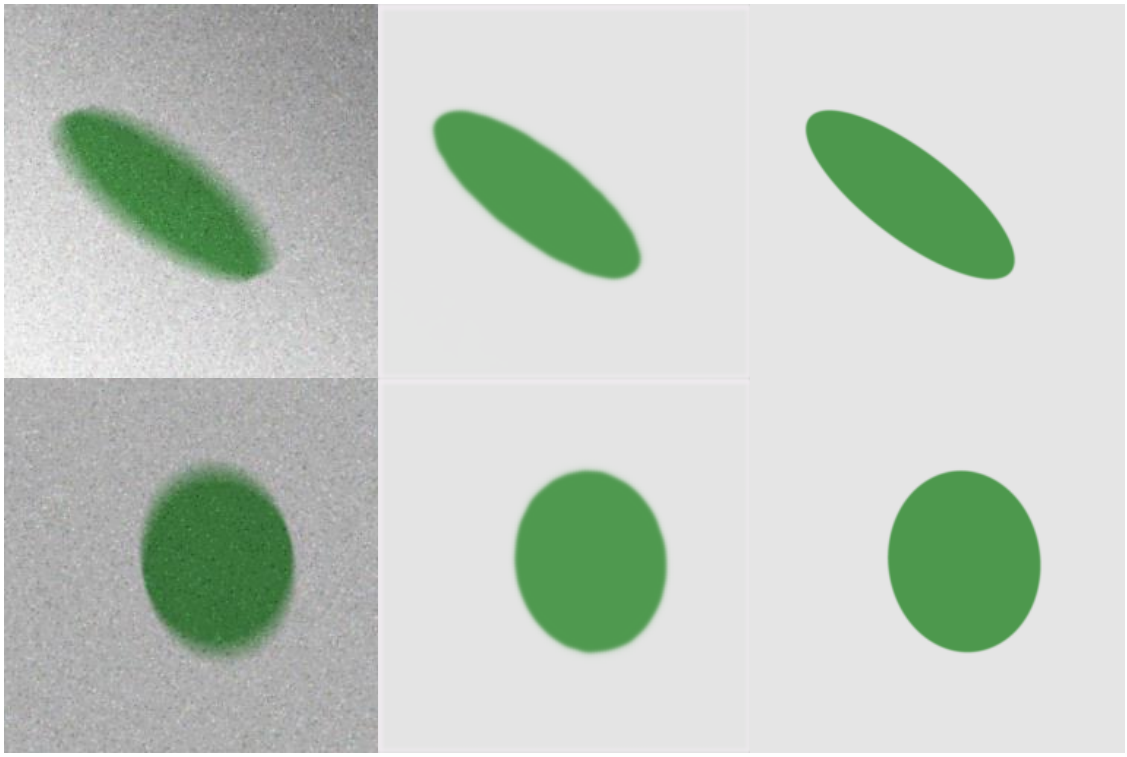


Figure 5.3. Synthetic ellipse recovery. (a) Noisy blurred input, (b) restored by CNN, (c) Ground Truth. Note: saturation adjusted for better viewing in black and white.

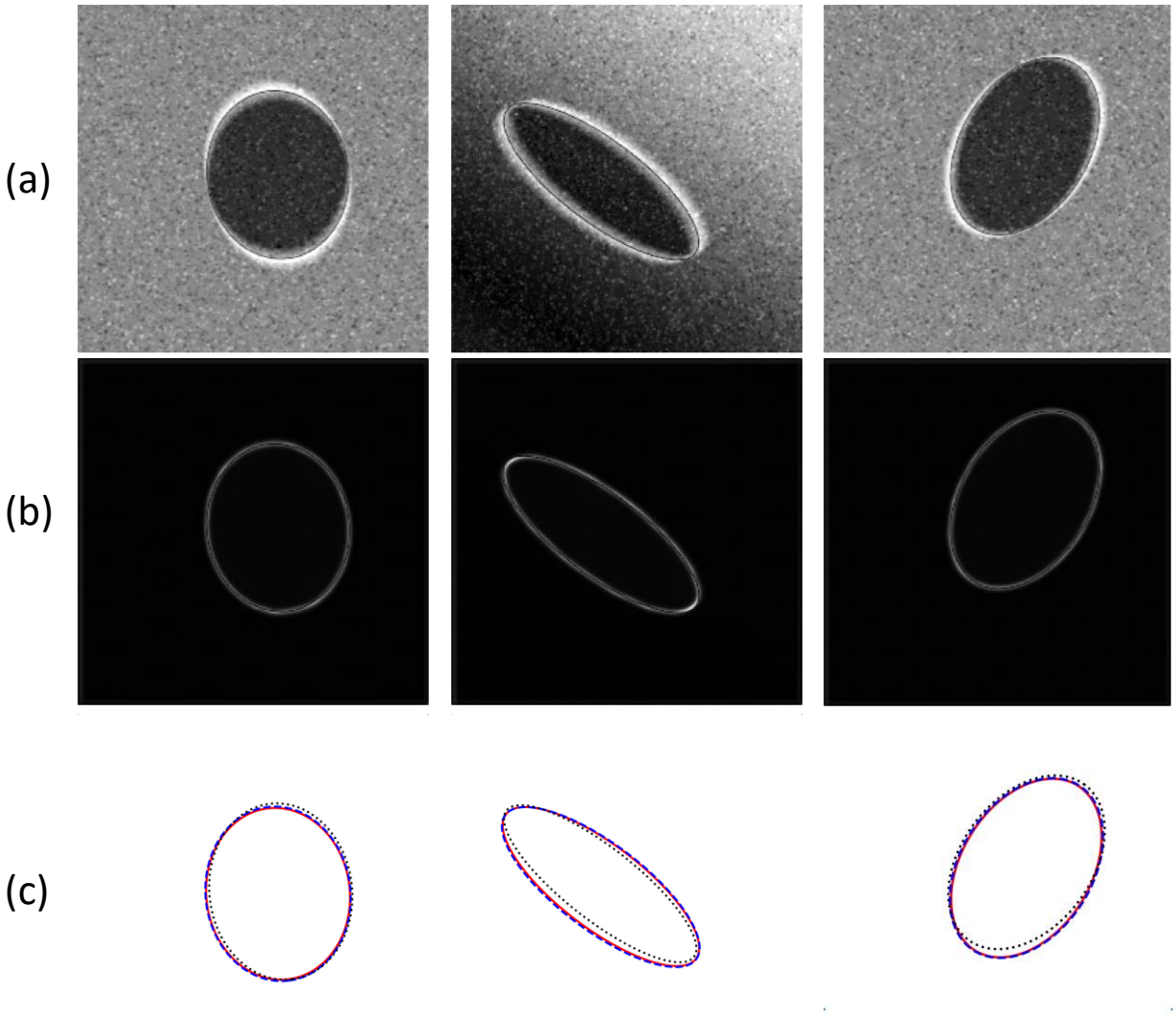


Figure 5.4. Synthetic image edge recovery and ellipse fitting. (a) |blurred-GT|, (b) |recovered-GT|, brighter means the absolute pixel value difference is larger. (c) Fitted ellipses comparison. Solid line is the ground truth, dashed line is the fitting result of CNN restoration and Canny edge, dotted line is the result of [28]. (Best viewed in color)

5.4.2 *Results with actual images*

To show the results in the practical situations, we scan the image patches near those green stickers. We use overlapped patches, each patch has a size 256×256. After recovery, only the center

200×200 pixels are stored and merged for ellipse detection. Circular stickers have a known diameter 19.05 mm.

We put the green circular stickers on the headboard and footboard of an infantometer as shown in Figure 5.5 (a), and take pictures of the stickers. The estimated footboard to headboard distance is 45.2 cm, compared to the infantometer reading of 45 cm. The difference is only 0.2 cm. This verifies that our method can accurately recover noisy motion-blurred circular sticker's parameters with the CNN.

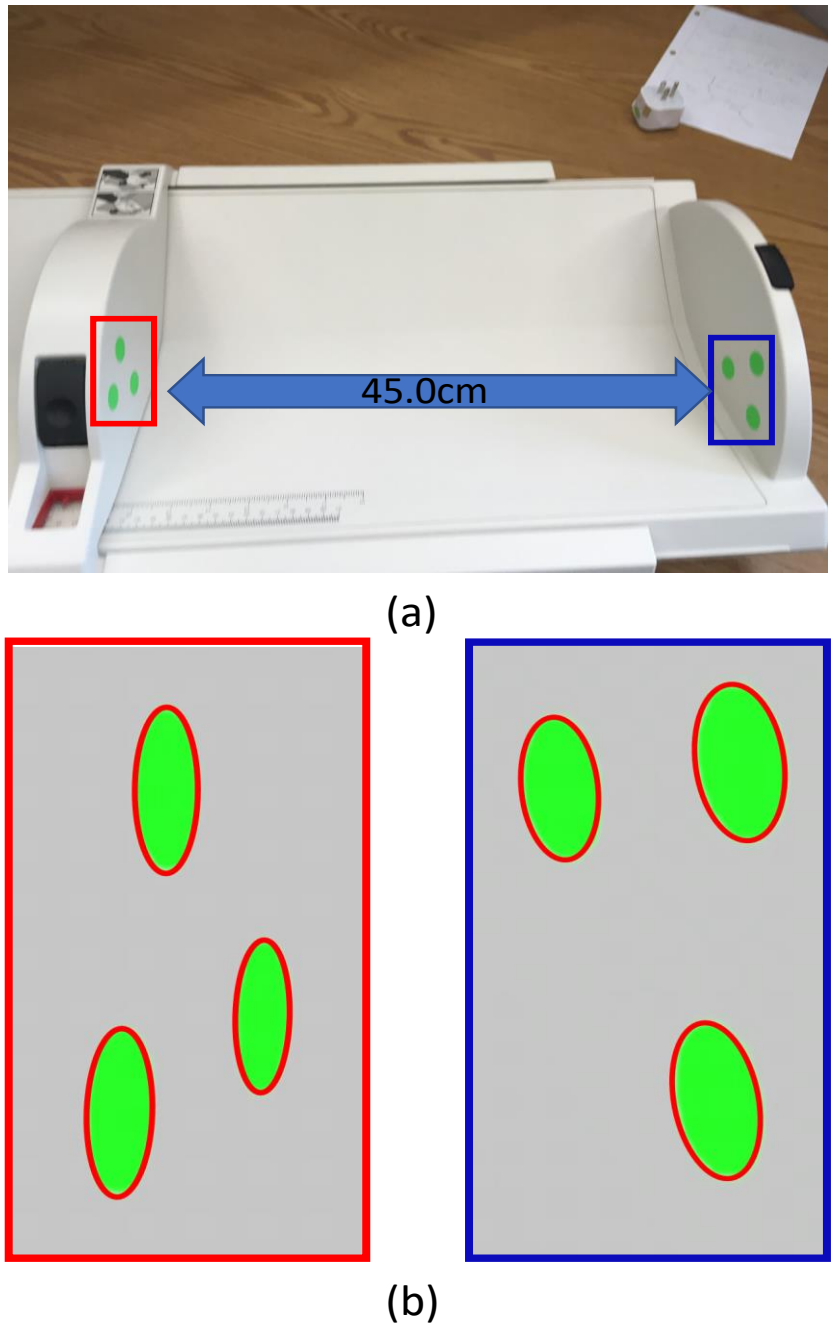


Figure 5.5. Infantometer with motion blurred circular stickers on the headboard and footboard. (a) Test image; (b) CNN processed and detected markers, the image's saturation is adjusted for better black and white view. The distance reading from the infantometer is 45.0 cm. Recovered clean ellipses and fitting. The calculated two plane distance is 45.2 cm, compared to 45.0 cm read from the infantometer.

Unlike previous deblur algorithms, which try to recover all textures of an image, our model suppresses textures that do not belong to the label and background. The reason is that we only trained the model with green circular stickers and white background, the ground truths are synthesized ideal ellipses. All other unrelated texture will be considered as noise by the CNN model. We could compare the CNN processed image Figure 5.5 (b) with the original image Figure 5.5 (a), dark edges of the background were gone in the recovered images. This does not invalidate our method, because the goal is to recover the edge location, not a good visual result. The model tries to recover ellipses with smooth and clear edge boundaries that can facilitate edge tracing. Such effect could not be achieved if actual images are used as ground truths of training data due to the unavoidable noise and artifacts.

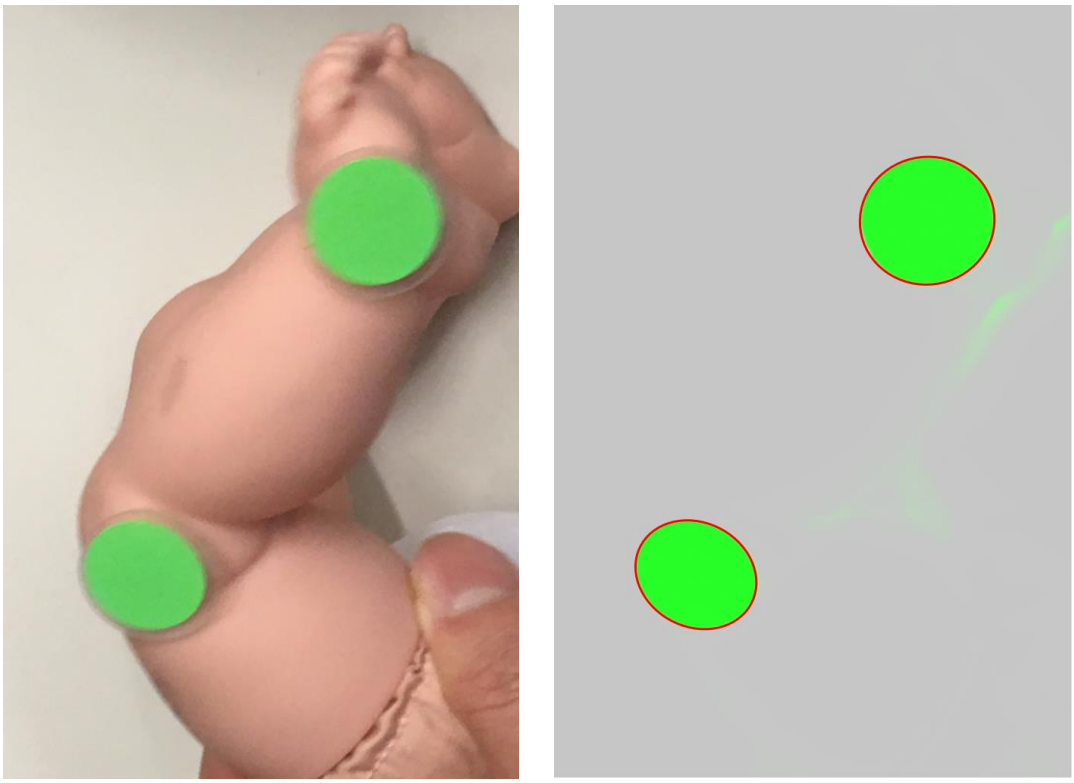
Table 5.2. Results of applying different methods on an image where the ellipses are blurred. "This work*" means "This work without gamma correction." Without CNN processing: Ellipse growing (Chapter 4) is used for comparison here, since it is the only method that reliably detects markers in real images. We also test [27] [28] [9] and [44] using green color seeds and Gaussian and median filter of different size, none of them is able to detect sufficient number of ellipses for distance estimation.

Estimation	This work	This work*	Without CNN processing
1	45.1 cm	45.8 cm	42.6 cm
2	45.3 cm	46.3 cm	41.9 cm
3	45.0 cm	45.8 cm	42.2 cm
4	45.0 cm	45.8 cm	41.9 cm
5	45.4 cm	46.2 cm	42.6 cm
6	45.3 cm	46.1 cm	42.7 cm
Avg	**45.2 cm**	46.0 cm	42.3 cm
Med	**45.2 cm**	46.0 cm	42.4 cm
STD	**0.1 cm**	0.2 cm	0.3 cm
Err	**0.2 cm**	1.0 cm	2.6 cm

Table 5.2 shows the comparison of 3D distance recovery using parameters of the detected ellipses. The relation between 3D positions and ellipse parameters is briefly summarized in the

previous chapter and its details could be found in [29]. The 3D distance estimation is very sensitive to ellipse parameters. It is impossible to obtain ground truth parameters from the blurred images, thus, the accuracy of recovered 3D distance could serve as a good metric to evaluate the algorithm. The ground truth of sticker's 3D distances could be obtained using physical tools. Here image blur is caused by camera motion. The distance estimated without CNN processing is 42.3 cm, 2.7 cm shorter than the infantometer reading. The distance estimation with our proposed CNN is 45.2 cm, only 0.2 cm longer than the infantometer reading. The results also show the necessity of simulating gamma and inverse gamma correction during training: the error decreases from 1 cm to 0.2 cm. Notes about Avg, Med, and STD: for one image we have six length estimations, three headboard stickers' distances to the footboard, and 3 footboard stickers' distances to the headboard. Avg means the average value of those six estimations; Med means the median value; STD means standard deviation of those six values. The result validates our synthetic training data generation method. The good performance on synthetic data is transferable to realistic images.

Figure 5.6 shows the recovered clear ellipses from the image that contains motion-blurred circular stickers. Here motion blur was simulated by moving the baby doll's leg while taking the picture. The ground truth distance is 6.70 cm. The estimated distance by the method proposed in this paper is 6.78 cm. The difference is less than 1 mm.



(a) (b)

Figure 5.6. Blurred stickers on baby doll. (a) Circular stickers were blurred due to foot motion, (b) recovered clear ellipses (saturation adjusted for black and white view). Conventional methods (such as [6][9]) failed when directly applied to blurred images.

Our current trained CNN cannot handle stickers of general shapes. However, our CNN model and the data generation method do not make assumptions about the shape of the stickers. For different shapes of stickers, we just need to synthesize the training data for this specific shape of sticker. For sticker-based pose estimation or many other 3D reconstruction problems, accuracy of the edge location is very important. If we could integrate the smoothed curve deblurring problem into the deep learning structure, this could facilitate 3D reconstruction with low-quality images.

Chapter 6. EXACT CURVE LOCATION ESTIMATION FROM NOISY AND MOTION BLURRED IMAGES USING CONVOLUTIONAL NEURAL NETWORKS

6.1 PROBLEM STATEMENT

Estimating exact curve locations is a very import initial step for 2D and 3D shape estimations, since in many applications, the extracted curve locations will be used to determine the parameters of the higher-level curves such as ellipses, circles and lines. The estimated 2D curve parameters could also be further used for 3D pose estimation, for example, a circular marker's 3D location could be estimated from its 2D ellipse projection on image as discussed in the previous chapters. For more general 3D shape reconstruction with multi-view images, curve segments could be incorporated into the structure from motion (SfM) pipeline to recover 3D models that are denser and more accurate than point based methods [45][46][47][48]. All those applications require detectors to detect curves with good location accuracy. Furthermore, to make the detector practical, the detector should be able to handle real world image degradations such as noise and motion blur.

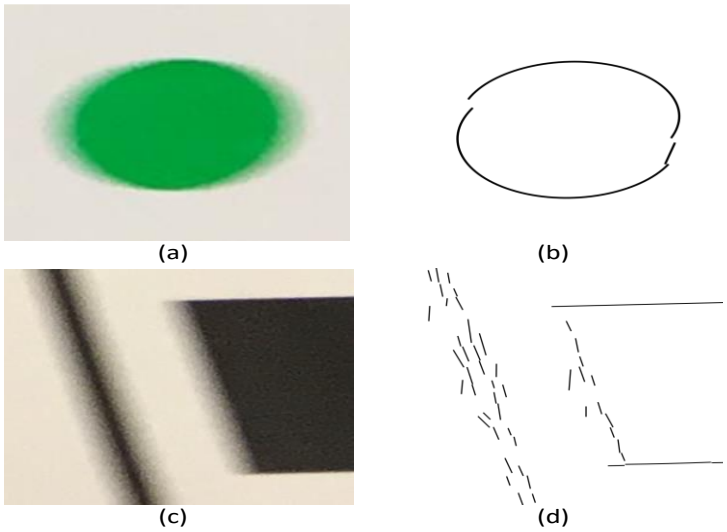


Figure 6.1. Example of failed curve segment estimation when image contains motion blur. (a) A blurred elliptical marker; (b) Detected ellipse by applying ELSDc [27] on (a); (c) An image

containing a blurred line and a blurred rectangle; (d) Result of ELSDc on (c). Note that ELSDc outputs the ellipse curve and line segments, both are colored as black.

Regular low-level edge detectors are very sensitive to noise and blurriness. Curve detectors based on those edge detectors are fragile to degraded image qualities. As an example, Figure 6.1 shows the detecting results using the Ellipse and Line Segment Detector with the Continuous Validation (ELSDc) [27] algorithm on blurred images. ELSDc is one of the most popular curve segment detectors. It preprocesses images with Gaussian filters, chains pixels with similar gradient orientations, then applies the Helmholtz principle to refine the results with statistically significant curves. We could see from Figure 6.1 (a), (b) that a single blurred ellipse is detected as two separate ellipse segments which are far from ideal. Also the locations of the detected curve segments are not accurate which will affect the accuracy of estimating the parameters of the ellipse, which is important for our application discussed in the previous chapters. In Figure 6.1 (c), (d) long blurred straight edges are detected as many short scattered line segments with not very accurate locations.

A straightforward way to solve the degraded image issue is to preprocess images with high quality restoration algorithms before edge detection and chaining. However, Image denoising and deblurring algorithms usually do not directly facilitate accurate edge location recovery, nor do they benefit smooth long curve extraction. For example, in Figure 6.2 (a), we use a blurred circular marker as the input. Figure 6.2 (b) shows the outputs of two state of the art blind deblurring algorithms [49][21] together with detected edges. The restored images contain many sharp artifacts which makes smooth curve extraction difficult. The edges of deblurred images are detected by Canny edge detector with $\sigma = 1$ Gaussian smoothing. The curve segments are extracted by linking the edge pixels within a 30-degree tolerance range. The result of optimization based deblurring algorithm [49] is very noisy (left image of Figure 6.2 (b)), we are not able to extract a

complete ellipse curve. Neural network based deblurring algorithms are also problematic. Figure 6.2 (b) middle and left are the results of one of the state of the art CNN based deblurring methods [21]. We applied the same smoothing and chaining parameters. Although closed curves are detected, we could see obvious distortion from a perfect ellipse (right two images in Figure 6.2 (b) and (c)). This could hurt the performance of algorithms that utilize curve's convexity for higher level shape detection. Figure 6.3 shows the results of applying the same procedure to a table texture patch as the input (Figure 6.3(a)), we could not extract smooth curve segments longer than 50 pixels from any of those three restored images (Figure 6.3 (b) and (c)).

Good image restoration algorithms do not necessarily restore edges for easy chaining and accurate locations, since the criteria of good image restoration is different from good chainable edge pixels and accurate edge locations. Image restoration algorithms usually aim to recover pixels' values. However, to extract smooth curves, only edges' geometric features such as locations and orientations are important, pixel values are not even needed. To overcome existing methods' limitations, we propose to use CNN to directly recover edge pixels' geometric features from noisy and motion blurred images without image restoration. Neural network has been very successfully used in many low-level image processing tasks. With carefully prepared training data, it is possible to train CNN to directly detect edge with accurate edge locations from low quality images without restoration. Unlike regular CNN edge detectors (such as [15] and [16]) that output an edge probability map, we use the feature map's maximum contrast as an indicator of the edge's strength, and the direction of the maximum contrast for edge chaining. After detecting chainable edge pixels, curve segments are directly extracted by chaining pixels with maximum contrast values higher than a threshold. The intuition of our proposed approach is based on the observation that the CNN feature map is just a trainable multi-channel image generator. Given proper training

data and strategy, any classical edge detectors that work for color image should also work for trained feature map. We could train the CNN to generate images that are good for recovering edge geometrical features instead of pixel values. Although we only adapt maximum contrast to CNN feature map in this chapter, our proposed pipeline should also work with other color edge detectors with little modifications.

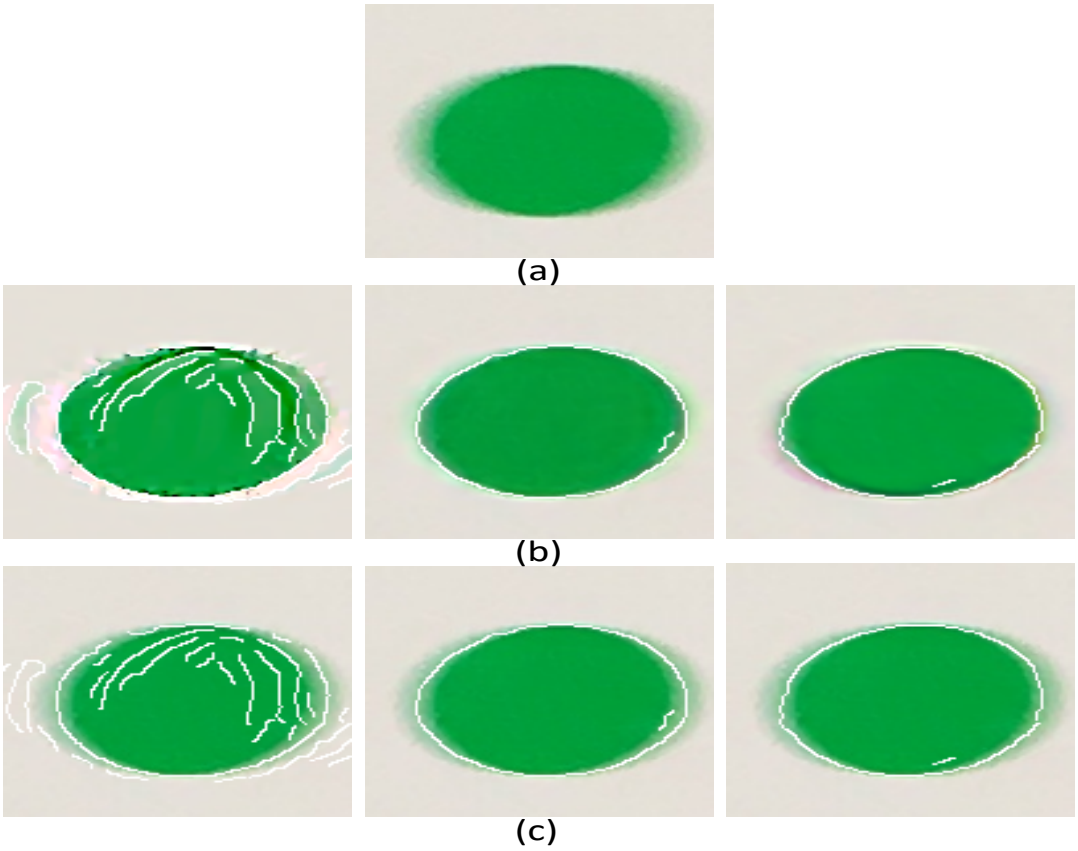


Figure 6.2. Green elliptical marker image. From top to bottom: (a) Input image, (b) Restored image overlaid with the detected curves, (c) Input image overlaid with the detected curves. From left to right: dark channel deblurring [49], SRN color [21], SRN LSTM [21].

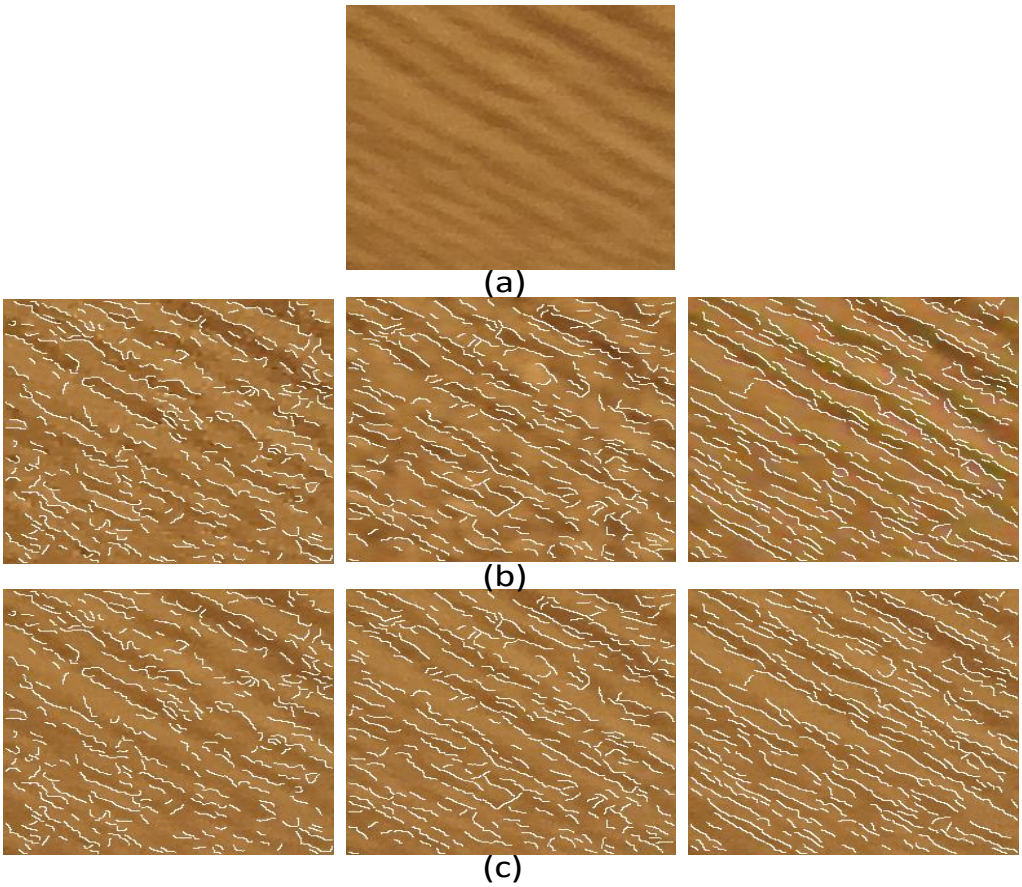


Figure 6.3. Table texture image. From top to bottom: (a) Input image, (b) Restored images overlaid with the detected curves, (c) Input image overlaid with the detected curves. From left to right: dark channel deblurring [49], SRN color [21], SRN LSTM [21].

The novelties of our work include: 1. To the best of our knowledge, we are the first one that studies edge detection and chaining from motion blurred images using CNN without restoration. 2. We adapt moving variance normalization as a differentiable layer to improve the network's capability to detect low contrast curves. 3. We propose to use multi-channel maximum contrast for the CNN edge detector and its related multi-channel loss functions for training. 4. We propose a parallelable edge linking method that uses edge's subpixel and tangent information. 5. We also contribute a synthetic data generation method that is able to generate sufficiently sophisticated

images with subpixel edge and orientation labels on curve segments for training the CNN to achieve good results for real-world images.

In the rest of the chapter, we first discuss the related work and background. Then we introduce our proposed subpixel curve segments extraction pipeline and the resulting curve detector. Finally, we discuss the training strategy that makes the network works for real images, and evaluate our methods both qualitatively and quantitatively.

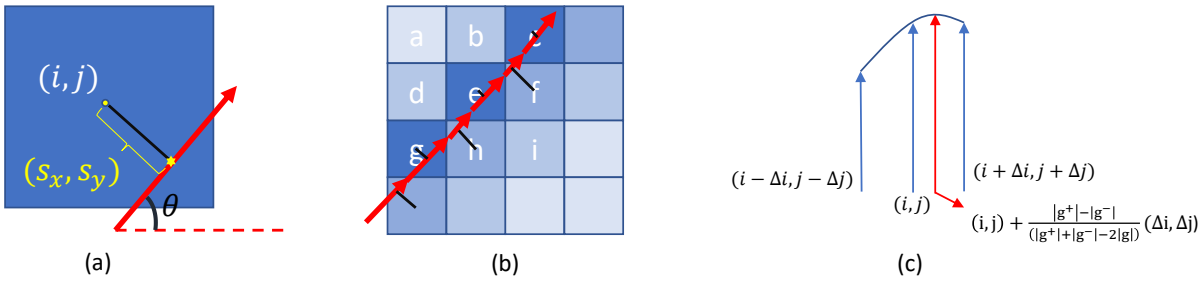
6.2 BACKGROUND

6.2.1 *Subpixel edge representation and estimation*

The model of a subpixel edge is shown in Figure 6.4(a), it is described by a 2D vector: subpixel displacement and tangent. (s_x, s_y) is the subpixel displacement relative to the nearest integer grid, tangents $(cos\theta, sin\theta)$ represents the edge's orientation. One simple method to extract subpixel edge from gray image is to first detect edge using Canny edge detector [50], then do a parabola fit of gradient magnitude at integer locations and find the peak [19]. The procedure and limitations are reviewed in [51]. Figure 6.4 (c) briefly shows how subpixel location is calculated, (i, j) is an edge point detected by Canny edge detector, $(i - \Delta i, j - \Delta j)$ and $(i + \Delta i, j + \Delta j)$ are two neighbor grids that are along the gradient direction. The subpixel edge locations could be estimated as:

$$(i, j) + \frac{|g^+| - |g^-|}{(|g^+| + |g^-| - 2|g|)}(\Delta i, \Delta j) \qquad (6.1)$$

$\Delta i, \Delta j \in \{0, -1, 1\}$. Usually it is very hard to recover edges to subpixel accuracy when the image is blurry and noisy. However, it is still beneficial to use the subpixel edge model. It makes the formulation of the loss function and the linking criteria easier. A truly reliable subpixel estimation usually assumes a known parameterized shape, which does not belong to the scope of our discussion. Our method could be considered as a curve candidate generator.



(a) (b) (c)

Figure 6.4. Subpixel edge representation. (a) An edge pixel, the subpixel edge is represented using red arrow; (b) An example of linked subpixel edges, red arrows are the subpixel edges, black line segments are the subpixel displacements relative to the integer pixel centers, and darker pixel colors mean higher gradient magnitudes. (c) Red arrow points to the peak location, its location is calculated as: $(i,j)+\frac{|g^+|-|g^-|}{(|g^+|+|g^-|-2|g|)}(\Delta i,\Delta j)$, where $|g|$, $|g^+|$ and $|g^-|$ are gradient magnitudes at (i,j), $(i+\Delta i, j+\Delta j)$ and $(i-\Delta i, j-\Delta j)$, respectively.

6.2.2 *Edge detection with a multi-channel image*

Estimating subpixel locations from a multi-channel image or CNN feature map can be done by replacing the gray image gradient with a multi-channel gradient [52] also known as image's maximum contrast [25]. We will use maximum contrast and multi-channel gradient magnitude interchangeably. Although a CNN feature map can be interpreted as a multi-channel image, unlike a regular color image, a CNN feature map does not serve the purpose of visualization. It is supposed to be engineered to facilitate edge chaining and location recovery. Thus, there is no restriction on the number of output feature channels. There are many good multi-channel edge detectors, however due to the scope of this chapter, they will not be reviewed here. Instead, [53] could be referred as brief summary. Here we only present the essential steps of deriving and calculating the maximum contrast, and the detail can be found in [25].

Assuming the feature map has n channels, and each pixel is represented by a vector $f(x,y) = (f_0, f_1, f_2 \dots, f_{n-1})^T$ (for RGB image this is simply $f(x,y) = (f_R, f_G, f_B)^T$). If we pick a direction $\eta = (\eta_x, \eta_y)$, the derivative of multi-channel pixel value is also a vector:

$$\frac{\partial f(x,y)}{\partial t} = \left(\frac{\partial f_0}{\partial t}, \frac{\partial f_1}{\partial t} \dots, \frac{\partial f_{n-1}}{\partial t}\right)^T \quad . \tag{6.2}$$

Each element of the vector could be written as

$$\frac{\partial f_i}{\partial t} = \eta_x \frac{\partial f_i}{\partial x} + \eta_y \frac{\partial f_i}{\partial y} = \begin{pmatrix} f_{i_x} & f_{i_y} \end{pmatrix} \begin{pmatrix} \eta_x \\ \eta_y \end{pmatrix} \quad . \tag{6.3}$$

The Contrast is defined as:

$$S(x,y,t) = \left|\frac{\partial f(x,y)}{\partial t}\right|^2 = \frac{\partial f(x,y)}{\partial t}^T \cdot \frac{\partial f(x,y)}{\partial t} \quad . \tag{6.4}$$

This could be further written into:

$$S(x,y,t) = \begin{pmatrix} \eta_x & \eta_y \end{pmatrix} \begin{pmatrix} f_{0_x} & \cdots & f_{n-1_x} \\ f_{0_y} & \cdots & f_{n-1_y} \end{pmatrix} \begin{pmatrix} f_{0_x} & f_{0_y} \\ \cdots & \cdots \\ f_{n-1_x} & f_{n-1_y} \end{pmatrix} \begin{pmatrix} \eta_x \\ \eta_y \end{pmatrix} = \begin{pmatrix} \eta_x & \eta_y \end{pmatrix} \boldsymbol{M} \begin{pmatrix} \eta_x \\ \eta_y \end{pmatrix} \tag{6.5}$$

$$\boldsymbol{M} = \begin{pmatrix} f_{0_x} f_{0_x} + f_{1_x} f_{1_x} \dots + f_{n-1_x} f_{n-1_x} & f_{0_x} f_{0_y} + f_{1_x} f_{1_y} \dots + f_{n-1_x} f_{n-1_y} \\ f_{0_x} f_{0_y} + f_{1_x} f_{1_y} \dots + f_{n-1_x} f_{n-1_y} & f_{0_y} f_{0_y} + f_{1_y} f_{1_y} \dots + f_{n-1_y} f_{n-1_y} \end{pmatrix} \tag{6.6}$$

where $\boldsymbol{M}$ is a 2×2 symmetrical matrix. The maximum contrast would be the direction that makes $S(x,y,t)$ the largest value. This can be solved by finding the eigen vector of the largest eigen value of $\boldsymbol{M}$. The eigen vector and eigen value of the 2×2 symmetrical matrix have close form solutions. They can be directly written down as [25]:

$$\lambda^+ = \frac{M_{00} + M_{11} + \sqrt{(M_{00} - M_{11})^2 + 4M_{01}^2}}{2} \tag{6.7}$$

$$\eta^+ = \begin{pmatrix} \sqrt{\frac{1 + \frac{M_{00} - M_{11}}{\sqrt{(M_{00} - M_{11})^2 + 4M_{01}^2}}}{2}} \\ sign(M_{10}) \sqrt{\frac{1 - \frac{M_{00} - M_{11}}{\sqrt{(M_{00} - M_{11})^2 + 4M_{01}^2}}}{2}} \end{pmatrix} \quad . \tag{6.8}$$

where $\sqrt{\lambda^+}$ is the equivalent gradient magnitude for the multi-channel image. The Maximum contrast layer can be directly cascaded with the CNN since it has a close form solution. During training, the gradient will back propagate using the loss function, through the maximum contrast layer then to the CNN.

6.3 PROPOSED METHOD

The overall idea of our proposed method is to use a CNN to generate a multi-channel image aka feature map for better edge detection and geometric properties (subpixel locations and tangents) estimation. Edge linking and curve extraction will be based on the estimated subpixel edge information. The overall training pipeline is shown in Figure 6.5.

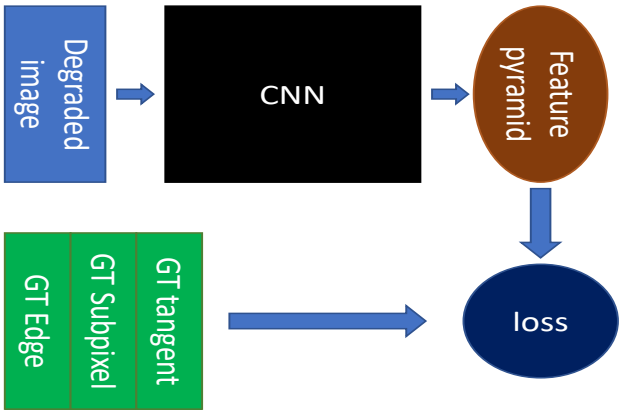


Figure 6.5. The overall pipeline of the proposed method. Synthesized degraded image will be fed into the network, the output feature pyramid and the ground truth edge, subpixel and tangent labels will be fed into the loss function. See Figure 6.6 for one possible CNN network structure, and Section 6.3.3 for detail description of our proposed loss function.

6.3.1 *Network structure*

The CNN structure we chose is very generic. The basic rule is that the last layer neuron's receptive field should be larger than the extent of motion blur. Figure 6.6 is an example network that could

handle 20-30 pixel motion blur. It resembles [7] and [13], but instead of training with three resolutions of images, we only feed the network with the highest resolution image. The network is a U-shape network modified from [21]. The size of the network could be adjusted by adding or removing Resnet blocks [13] for each scale of encoder and decoder. For 20 to 30 pixels motion blur, we found that two Resnet blocks for each resolution is good enough. For each scale output of decoder, we apply two 1×1 convolutions to extract an 8-channel feature map for contrast calculation. The lower resolution feature maps will be up sampled by transposed convolution then concatenated with higher resolution features. We choose the feature channel size to be 32 for the original resolution. A moving variance normalization (black block in Figure 6.6) is applied to the feature map. The detail of moving variance normalization will be discussed in the next paragraph. When the resolution is reduced by half, the channel size is doubled. The lowest resolution feature map has 128 channels. Each scale of output feature map (out_1, out_2, out_3) has 8 channels. The detailed network structure is shown in Figure 6.6. This CNN network has the fewest number of parameters compared with any existing motion deblurring network. It is also much smaller than the CNN noisy edge detector in [24] (see Table 6.1). The reason for being able to reduce the model to this small is because we do not need to recover the pixel values of the whole image.

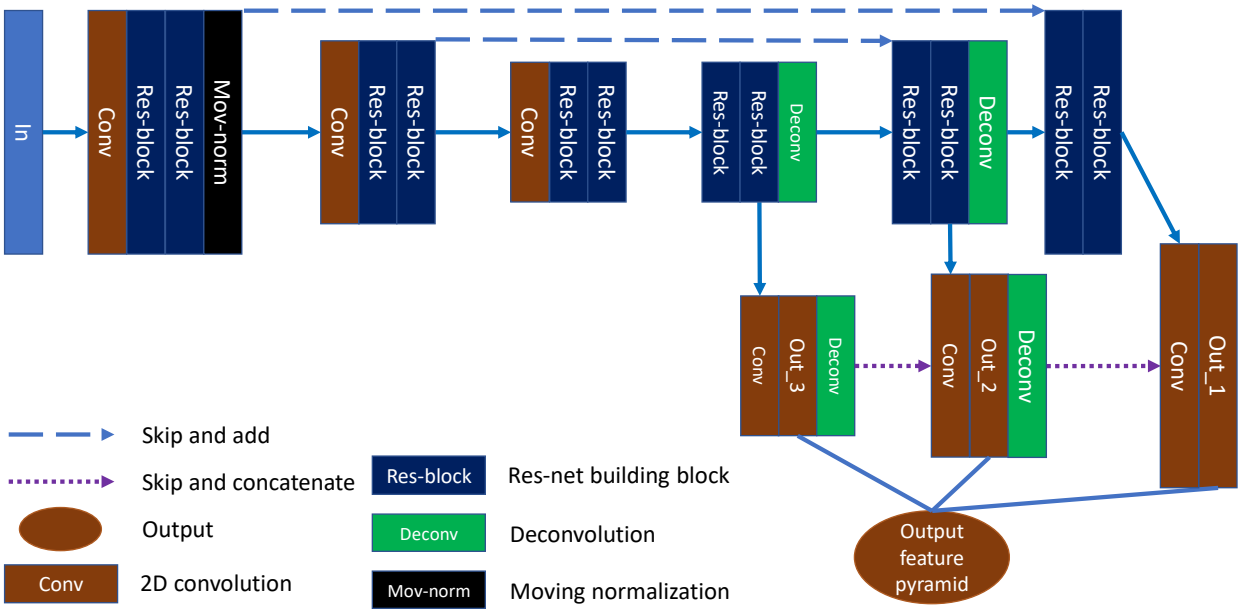


Figure 6.6. Encoder decoder network with skip connections. When testing, only out_1 will be used for edge detection and chaining.

Table 6.1. Comparison of model sizes. The SRN [21] color model has the fewest number of parameters among all existing CNN deblurring methods. Our model is even smaller than it.

Method	Ours	SRN [21]	CNN edge preservation losses [24]
Model Size	2.16 M	2.73-3.76 M	7.78 M

6.3.2 *Moving variance normalization*

One desirable feature of an edge detector is invariance of luminance. We wish the network to detect the similar edge pattern for the same object under different lighting. To achieve this, we could of course increase the network size and training data complexity, hoping the model to learn almost all possible patterns under all possible luminance. However, a more complicated model not only consumes more computational resource, but also increases the chance of overfitting.

The effect of non-uniform luminance could be effectively reduced using image's local properties [54]. Here, we choose to normalize the feature map using the moving variance and moving average. Moving variance could be built from the moving average operation.

$$Var_w(x) = Avg_w[x^2] - Avg_w[x]^2 \tag{6.9}$$

where Avg_w and Var_w are average and variance over window w, respectively. An image or feature map could be normalized as:

$$x_{norm} = \frac{x - Avg_w[x]}{\sqrt{Var_w(x) + \epsilon}} \tag{6.10}$$

where ϵ is a small value to avoid dividing aby 0. We choose $\epsilon = 0.1$ in our experiments. The 2D window size is empirically chosen as 33×33. The moving average could be efficiently implemented as:

$$Avg_w[x_n] = Avg_w[x_{n-1}] + \frac{x_n - x_{n-w}}{w} \quad . \tag{6.11}$$

Here we use $Avg_w[x_n]$ to represent the 1D moving average. It is worth noting that its backward propagation (gradient) could also be implemented efficiently in a similar way:

$$\frac{\partial Avg_w[x_n]}{\partial x_i} = \begin{cases} \frac{\partial Avg_w[x_{n-1}]}{\partial x_i}; & n - w < i < n \\ \frac{\partial Avg_w[x_{n-1}]}{\partial x_i} + \frac{1}{w}; & i = n \\ \frac{\partial Avg_w[x_{n-1}]}{\partial x_i} - \frac{1}{w}; & i = n - w \\ 0; & others \end{cases} \quad . \tag{6.12}$$

For large w, the above equation could help to avoid calculating average for each pixel repeatedly. The 2D moving average operator could be separated into two 1D moving average operators. By careful implementation, this moving variance normalization operation has almost negligible add-on to the overall running time. The benefits of moving average compared to naïve average and separated two 1D average are shown in Table 6.2 and Table 6.3. We could see that this moving style implementation is especially efficient when the window size is large.

Table 6.2. Comparison of forward running time of different moving average implementations. For each entry, from left to right: the running time (in seconds) of naïve 2D convolution, separable convolution and our implementation based on Equation (6.11) and (6.12). The first two implementations directly use existing (CUDA [55] based) convolution functions from an open source neural network frame work, Tensorflow [56]. Our implementation is written in CUDA, then wrapped as a custom Tensorflow differentiable operator. The input batch size is 16. The image/feature map channel size is 32. Time is recorded for 100 times of running.

Image size / Window size	64 × 64			128 × 128			256 × 256			512 × 512		
7 × 7	0.30	0.23	**0.19**	1.53	1.40	**0.74**	5.62	5.33	**2.59**	22.8	21.2	**10.3**
11 × 11	0.26	025	**0.20**	1.96	1.38	**0.76**	7.68	5.70	**2.63**	30.7	22.5	**10.4**
15 × 15	0.67	0.25	**0.21**	3.00	1.34	**0.81**	11.2	5.83	**2.65**	43.3	23.7	**10.4**
25 × 25	1.1	0.30	**0.20**	3.17	1.48	**0.63**	33.0	6.48	**2.64**	134	25.9	**10.7**
35 × 35	1.5	0.34	**0.20**	3.24	1.79	**0.64**	65.1	7.12	**2.62**	261	27.5	**10.3**

Table 6.3. Comparison of backpropagation running time (in seconds) of different moving average implementations. The setting is the same as Table 6.2.

Image size / Window size	64 × 64			128 × 128			256 × 256			512 × 512		
7 × 7	**0.27**	0.31	0.31	2.26	1.84	**1.63**	12.5	6.29	**5.93**	49.2	30.0	**23.4**
11 × 11	0.36	**0.31**	0.32	6.29	1.73	**0.98**	22.9	6.78	**5.92**	95.6	33.8	**23.7**
15 × 15	0.51	0.33	**0.32**	9.92	1.87	**0.98**	40.4	7.16	**5.78**	157	37.6	**23.7**
25 × 25	2.4	0.33	**0.29**	21.2	2.15	**0.92**	105	8.71	**5.72**	778	45.9	**22.4**
35 × 35	4.2	0.38	**0.27**	39.8	2.60	**0.94**	207	14.7	**5.47**	1248	57.5	**22.4**

This moving variance normalization layer could be directly applied to the input image. However, since this operation could be efficiently implemented as a differentiable layer, it is better to apply the moving variance normalization to the learned feature map. A neural network will learn

a feature map that is more suitable for moving variance normalization than the original pixel values.

6.3.3 *Maximum contrast aligning loss*

The loss function should be designed to facilitate edge geometric feature recovery. Intuitively, we try to align the output feature pyramid's maximum contrast with the ground truth subpixel edges. The edge pixel's maximum contrast should be sufficiently large, non-edge pixel's maximum contrast should be sufficiently, small and the ridge of the profile should be aligning with the ground truth edge. An ideal curve and its contrast profile are shown in Figure 6.7. To train the network toward the ideal maximum contrast profile, our loss function is made by three parts: 1. Tangent loss that aligns predicted edge's orientation with the ground truth edge. 2. Absolut maximum contrast loss that increases the edge pixel's multi-channel gradient magnitude and suppresses non-edge pixel's gradient magnitude. 3. Relative maximum contrast loss function that shapes the contrast profile near a ground truth subpixel edge for better parabola fitting.

Edge tangents are very important information for edge chaining. The desired dot product between the predicted tangents and the ground truths should be as close to one as possible to satisfy the smoothness constraints. The tangent loss is defined as:

$$tangent\ loss = \frac{\sum_{edge\ pixel} \max\left(0, 0.98 - \left|\eta_x^+ \cdot \eta_x^{GT} + \eta_y^+ \cdot \eta_y^{GT}\right|\right)}{edge\ pixel\ number} \tag{6.13}$$

where $\eta^+ = \left(\eta_x^+, \eta_y^+\right)^T$ and $\eta_{GT} = \left(\eta_x^{GT}, \eta_y^{GT}\right)^T$ are predicted and ground truth edge tangents. This loss pushes the learned orientation as parallel to the ground truth as possible. However, if the tangent aligning is good enough, for example, $\left|\eta_x^+ \cdot \eta_x^{GT} + \eta_y^+ \cdot \eta_y^{GT}\right| > 0.98$, there is no need to over optimize this dot product.

The absolute maximum contrast loss functions for edge and non-edge pixels are defined as:

$$edge\ pixel\ loss = \frac{\sum_{edge\ pixel} \max\left(0, \alpha - \sqrt{\lambda^+(x,y)}\right)}{edge\ pixel\ number} \tag{6.14}$$

$$non\text{-}edge\ pixel\ loss = \frac{\sum_{\text{non-edge pixel}} \max\left(0, \sqrt{\lambda^+(x,y)} - \beta\right)}{non\text{-}edge\ pixel\ number} \tag{6.15}$$

where $\lambda^+(x,y)$ is the maximum contrast at location (x,y) (here we apply square root to make the contrast to have the same fundamental unit as gradient magnitude), α is a large value, and β is a small value (in this work, we choose $\alpha = 1.0$ and $\beta = 0.1$). Hinge loss is adopted here for efficient multiple loss function training. If an edge pixel's contrast is larger than α, it will be considered as a candidate curve pixel, there is no need to further explicitly increase it. Similarly, when a non-edge pixel's maximum contrast is smaller than β, it will not be considered as an edge pixel, and we do not need to further explicitly minimize it. Hinge loss could help to prevent over-minimizing the loss function. It saves the headroom for other loss functions.

Once we have the edge pixel detected, we could go further to fit a parabola to estimate the subpixel edge location. To facilitate parabola fitting, the relative maximum contrast loss function is proposed as:

$$relative\ contrast\ loss^+ = \frac{\sum_{edge\ pixel} \max\left(0, 0.1 - \left(\sqrt{\lambda^+(x,y)} - \sqrt{\lambda^+(x^+,y^+)}\right)\right)}{edge\ pixel\ number} \tag{6.16}$$

$$relative\ contrast\ loss^- = \frac{\sum_{edge\ pixel} \max\left(0, 0.1 - \left(\sqrt{\lambda^+(x,y)} - \sqrt{\lambda^+(x^-,y^-)}\right)\right)}{edge\ pixel\ number} \tag{6.17}$$

where (x^+, y^+) and (x^-, y^-) (see Figure 6.7) are two pixels across the curve (their center (x,y) is the ground truth subpixel edge location), and $(x^+ - x^-, y^+ - y^-)$ is perpendicular to the edge. In our experiments, we choose (x^+, y^+) and (x^-, y^-) to be 0.8-pixel length away from the ground truth subpixel edge.

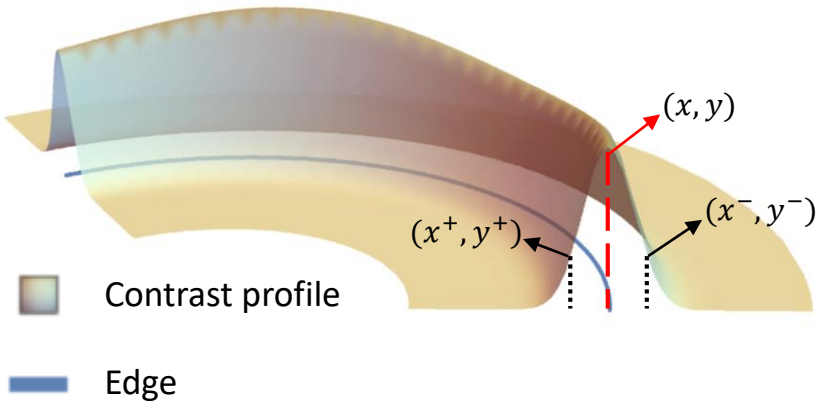


Figure 6.7. An example of ideal edge and its contrast profile. The blue curve represents the ground truth edge we want to estimate. The surface height represents the square root of the maximum contrast. $x^+ = x_{GT} + \eta_x^{GT} \cdot 0.8$, $y^+ = y_{GT} + \eta_y^{GT} \cdot 0.8$, $x^- = x_{GT} - \eta_x^{GT} \cdot 0.8$, *and* $y^- = y_{GT} - \eta_y^{GT} \cdot 0.8$. Note that, (x_{GT}, y_{GT}), (x^+, y^+) and (x^-, y^-) usually are not integers, bicubic interpolation is used to calculate their corresponding pixel values and gradients.

It is beneficial to apply the maximum contrast aligning loss to all scales of the feature pyramid. Multi-scale loss will make network training faster. It could also help to prevent from trapping into poor performing local minimum. Assume the highest resolution feature scale number is 0, and the scale-number increases while resolution reduces, for a ground truth edge pixel (x, y) at the original resolution, its corresponding coordinate at Scale n will be $(0.5^n \cdot x, 0.5^n \cdot y)$. Non-integer coordinates cannot be avoided due to the down sampling operation and the subpixel related loss functions. To calculate contrast, we need to first calculate each feature channel's 2D gradient. Interpolation is usually needed at the off-grid location. The most popular image interpolation method is bilinear interpolation. It could be implemented as a differentiable operation that allows back propagation during training [57]. For us, since we need interpolated values for pixel value gradients at non-integer locations, bicubic interpolation is a better choice since it is smoother and has a larger receptive field than the bilinear interpolation. It could also be efficiently implemented

as a differentiable operation. The detail of the implementation of differentiable bicubic interpolation for arbitrary location is summarized in the appendix.

6.3.4 *Synthesize training data*

To train the network, we need the degraded image and its ground truth subpixel edge labels. It is difficult to label subpixel information for real images, not to mention preparing enough data for the neural network training. We must rely on synthetic images for accurate subpixel edge labels. Fortunately, for a low-level edge detector, there is no need to synthesize semantically meaningful objects. Almost all objects' contour could be locally approximated by straight lines or elliptic arcs. Thus, we could start from very simple 2D shapes such as rectangles and ellipses of random sizes, then apply some simple operations such as "or", "and", and "subtraction" to create sophisticated shapes suitable for training the CNN. In our work, we choose ellipses and rectangles as initial binary 2D shapes, then randomly pick operations from addition, subtraction and multiplication to synthesize more sophisticated shapes. Several layers of shapes could be overlaid to synthesize a sufficiently complicated image for training the CNN.

6.3.4.1 Synthesize images with subpixel weights from simple 2D shapes for training the CNN

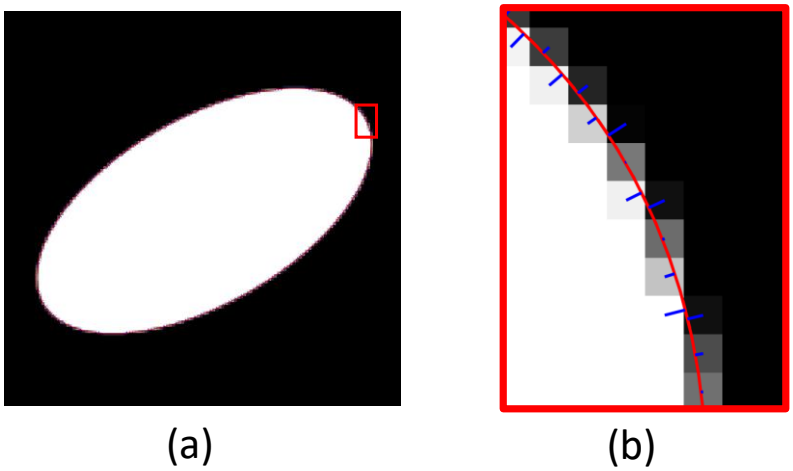


Figure 6.8. An example of a 2D shape mask with subpixel edge and displacement. (a) a 2D shape mask with subpixel edge; (b) Zoom-in view of subpixel edge and displacement. Subpixel edge is represented by thin red curve, its displacement relative to grid is shown as short blue line segments.

Figure 6.8 is an example of an ellipse mask with subpixel edge labels. The subpixel information is represented using a displacement vector $(\delta x, \delta y)$ and a tangent vector (η_x, η_y). $(\delta x, \delta y)$ is the displacement between the center of the pixel and the closest point on the curve, (η_x, η_y) is curve's tangent vector. It is straightforward to calculate the closest point and tangent direction for a rectangle. For an ellipse, there is no close solution. Instead, we implement an efficient iterative algorithm proposed by [58]. Once the closest point is found, the tangent vector calculation is trivial. The mask pixel value is 1 if the pixel is fully inside the 2D shape, it is 0 if fully outside. A boundary pixel usually has a mask value between 0 and 1 due to subpixel effect. To estimate the boundary mask value, we first sample the pixel center and the four corners to determine whether this pixel is a boundary pixel. If all five points are either inside or outside of the shape, then the mask value is either 1 or 0; Otherwise it is a boundary pixel, and 8×8 subpixel locations will be sampled

from the 1×1 square to estimate how many percent is inside the 2D shape, this percent value is the boundary pixel's mask value.

Starting with simple 2D shapes such as ellipse and rectangle, more sophisticated 2D shapes could be generated by addition, subtraction and multiplication operations. For the synthesized more sophisticated 2D shapes, it is easy to determine fully inside or outside pixels. However, boundary pixels' mask values, subpixel displacement vectors and tangents need to be carefully handled. Most of the boundary pixels inherit the properties of one of their parent shapes. However, the boundary of two parent shapes may cross and create joints. A pixel is a joint pixel if it is at the boundary of both parent shapes. Its mask value needs to be recalculated. Since there are only very small number of joint pixels, and it is hard to define the subpixel and tangent vectors, we exclude the joint pixels from edge pixel labels.

Once we have the 2D shapes of decent complexity, we could synthesize an image by blending two images into one using the mask of the 2D shape:

$$I_{blend} = I_{background} \cdot (1 - mask) + I_{foreground} \cdot mask \quad . \tag{6.18}$$

Background and foreground could be pure colored images or images randomly cropped from popular datasets such as Pascal VOC [59] and DIV2K [60]. We could blend more layers of 2D shapes to synthesize even more complicated images. Below is an example of synthesizing an image using two layers of 2D shapes:

$$I_{final} = \left(I_{background} \cdot (1 - mask_1) + I_1 \cdot mask_1\right) \cdot (1 - mask_2) + I_2 \cdot mask_2 \tag{6.19}$$

where I_{final} is the final image we get, $I_{background}$ is the background image, I_1 and I_2 are images for Layer 1 and Layer 2, and $mask_1$ and $mask_2$ are the corresponding masks for the two layers of the 2D shapes. For edge subpixel and tangent information, the front layer overrides the below layers.

6.3.4.2 Synthesize motion blur

Since the motion in the real-world may not be linear, in this chapter we use a more general motion model than the one we used in the previous chapter. To generate random motion trajectories, we assume that the object has a random initial speed. Here we assume the random initial speed is uniformly distributed between 5 pixels/s and 15 pixels/s, and its initial orientation is uniformly sampled from 360 degrees. We divide 1 second into 20 equal intervals. We first generate 4 random 2D acceleration vectors at different time instances, use bicubic interpolations to interpolate those four 2D points, and then sample 20 points from the interpolated curve (Figure 6.9 (a)). We then integrate these acceleration vectors (plus the initial speed) to get the object's 2D speed vector at each sampled time (Figure 6.9 (b)); Integrating the sampled speeds we obtain the object's 2D trajectory as shown in Figure 6.9(c). To generate the motion blurred image, the original image is first blurred by a Gaussian kernel with σ randomly chosen from [0.5,1.5] and then shifted to different positions according to the trajectory and averaged. Boundary with 16 pixel thickness will be cropped out to remove the artifacts cause by the motion blur simulation.

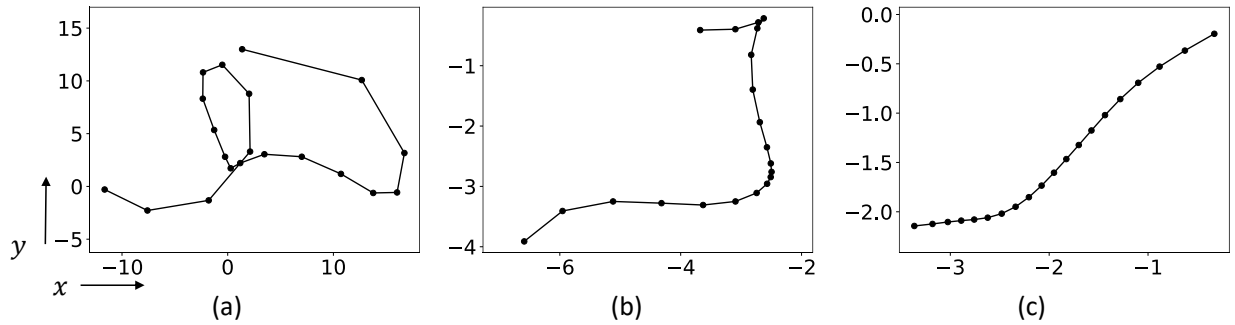


Figure 6.9. Motion trajectory generation. (a) Randomly generated 2D accelerations (pixels/second^2); (b) 2D Speed (pixels/second); (c) Object's moving trajectory. Each dot represents a 2D vector. The horizontal and vertical axes show the x and y components of the vector (unit in pixels).

6.3.4.3 Noise, demosaicing effect, gamma correction and random JPEG quality

In the previous chapter, we simulated the image degradation effect using Gaussian white noise and bilinear demosaicing. However, Gaussian white noise is quite different from noise in real images, and bilinear demosaicing is rarely used in real image processing pipeline. In this chapter, we further consider a more realistic noise model and demosaicing algorithm.

A realistic noise model [61] is broken down into irradiance dependent and independent parts. We use the same parameters as [62]. For each pixel, the noise could be modeled by a Gaussian distribution:

$$N(0, L \cdot \sigma_s^2 + \sigma_c^2) \quad . \tag{6.20}$$

It is necessary to simulate the image demosaicing effect, since demosaicing increases the granularity. After adding noise, fake raw images will be generated by keeping only one of RGB values based on the Bayer pattern. Since each camera manufacturer may have their own proprietary image processing pipeline, it is hard to accurately simulate the image formation process for a certain brand of camera. What we can do is to make the network less sensitive to the type of demosaicing algorithms. For simplicity, we consider two types of demosaicing algorithms: 1. a very basic bilinear interpolation method that uses only pixel values within a 3×3 range (same as we used in the previous chapter); 2. a higher quality interpolation method that uses pixels values within a 5×5 range [63]. With proper padding, both methods could be written down as a stride 2 convolution that has single channel input and 12 channel output, the result could be reassembled into a full resolution RGB image. The only difference between those two algorithms is the convolution kernel, the assembling process is the same. Assume the convolution kernel of bilinear demosaicing is $K_{bilinear}$, the kernel of [63] is K_{hq}, then

$$t \cdot K_{bilinear} + (1 - t) \cdot K_{hq} \tag{6.21}$$

is also a valid demosaicing kernel ($t \in [0,1.0]$). For each image we randomly pick a value t from uniform distribution, then use the linear blended kernel to demosaic the synthesized raw image. We should note that, using the linear blended kernel is equivalent to first demosaic image by both methods, then use t to linearly blend the result. However, using the linear blended kernel, we only need to compute the convolution once.

Besides noise and demosaicing, we apply a random gamma correction (1.7~2.7) with a broader range compared to that we used in the previous chapter, to the image. Since most images are stored with JPEG compression, we also account for the image degradation produced by the JPEG lossy compression. We use the JPEG compression operator in Tensorflow and randomly choose JPEG quality from 60 to 100, then apply it onto the synthesized image. The comparison of different stages of synthesized images is shown in Figure 6.10.

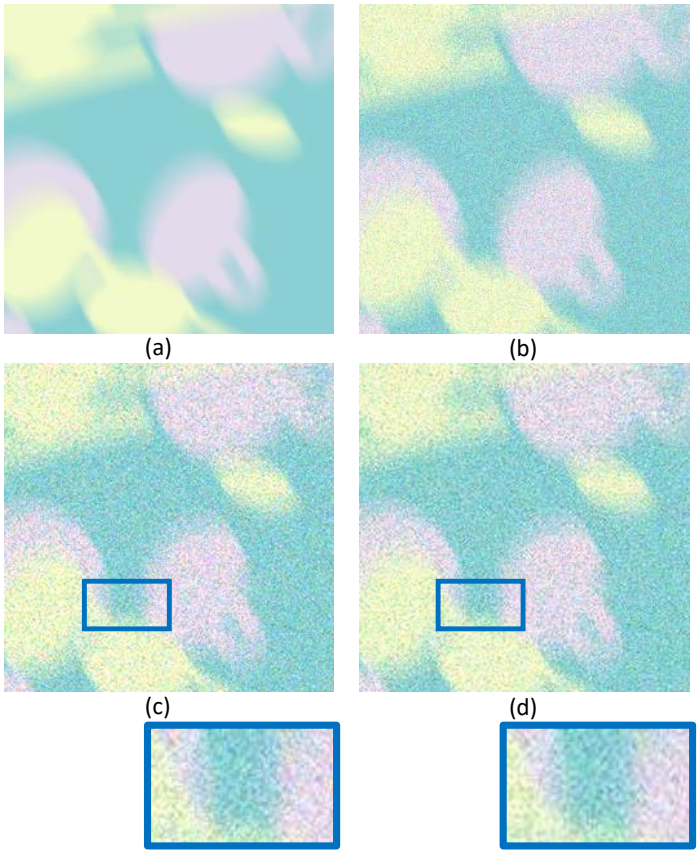


Figure 6.10. An example of different stages of image synthesizing. (a) With motion blur; (b) Add signal dependent noise; (c) Demosaicing; (d) With degradations from JPEG compression. Zoom in images shows the difference before and after JPEG compression. Note: image pixel values are stretched for display, the absolute brightness is not accurately displayed.

6.3.4.4 Real Blank images

One problem of using a neural network is that since it has too many parameters, overfitting is very likely to happen. When the network gets overfitted to the training data, it could behave abnormally to subtle patterns in testing images. It is very likely that the network performs very well to the synthetic data, but the test results on real images produce total garbage since real images may contain subtle patterns which may not be visually obvious.

One example of those subtle patterns is the patterns that may exist when we display a blank image on a monitor. As shown in Figure 6.11, when we display pure color images on a monitor,

we can see some distortion patterns. We found these kinds of patterns are useful to train the CNN to suppress the effect of those undesirable patterns. We generate some pure blank images with different colors, display them on a 4K resolution monitor, then take pictures of the monitor with random camera angle and shaking. We then feed the images into the network and suppress the gradient magnitudes that are larger than a threshold to make the network inactive to useless real image patterns. Since the monitor is made up by millions of small light emitting units, the pictures may also contain spatial aliasing patterns. The loss function used for blank images during the training of the CNN is:

$$blank\ image\ loss = \frac{\sum_{all\ pixels} \max(0, \lambda - 0.1)}{pixel\ number} \tag{6.22}$$

where λ is the maximum contrast of the output feature map. This loss function is applied to all scales of the output feature pyramid. With this loss function, the maximum contrast will be suppressed if its value is larger than 0.1. Using these blank images could help to prevent the network from being too sensitive to those visually weak patterns caused by the camera's internal image process pipeline. It encourages the network to produce small gradient magnitudes as default responses.

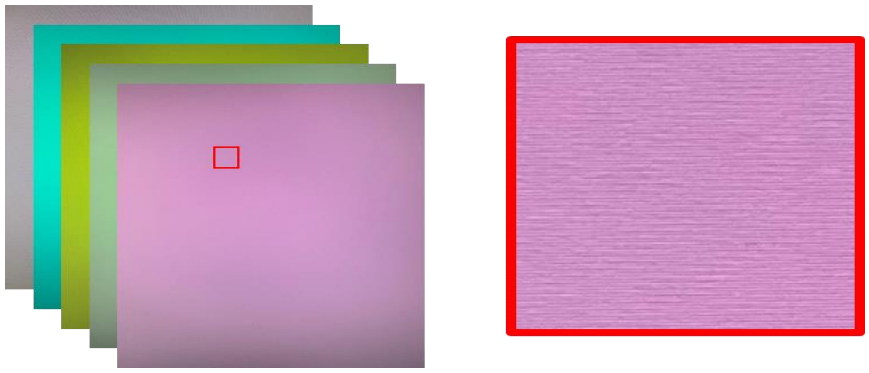


Figure 6.11. Blank images that are used to suppress abnormal feature map response. Left: typical blank images, they are obtained by taking pictures of a 4k monitor with random RGB

colors. Right: the zoom-in view of the red rectangle. In the zoom-in view, we could see that for a blank image, it may contain patterns that are caused by spatial aliasing, image processing pipelines, or the 4k monitor's own pattern. The network is trained to be less sensitive to those weak patterns.

6.3.5 *Curve pixel detection and extraction*

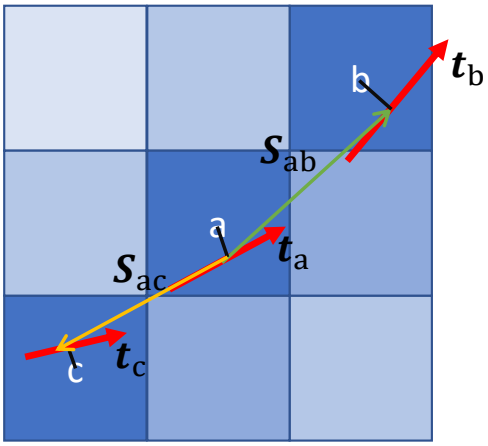
After getting the output feature map, next step is to detect curve pixels. A pixel is a curve pixel if it belongs to a curve which is sufficiently long to be considered statistically significant. Our curve pixel detection method is parallelable and it runs efficiently on GPU. For curve extraction, it basically reassembles the curve pixels as lists of (subpixel) coordinates. Each smooth curve is represented as a list. Since it involves list data structures and the length of each curve is unknown before extraction, it could not be processed using GPU. However, this process is parallelable and could take advantage of parallel capability of modern CPUs.

6.3.5.1 Curve pixel detection

Curve pixel detection resembles the procedure of edge drawing [64]. It starts with some seed pixels, then try to trace curve from them. Any pixel belongs to the curve will be marked as a curve pixel. However, instead of using the routing strategy proposed by [64], we utilize the subpixel edge model described earlier, and set a smoothness constraint using subpixel tangents and displacements as shown in Figure 6.12. In Figure 6.12, $\boldsymbol{t}_a$ is the tangent vector at subpixel location *"a"*. Our method works for both gray and multi-channel images and is suitable for parallel implementation.

Candidate pixels are first generated by procedure similar to the Canny edge detector [50]. We set a low threshold to maximum contrast (same as gradient magnitude for a gray image), then apply non-maximum suppression. A subpixel edge representation could be obtained using methods mentioned in 6.2.1 and 6.2.2. After getting candidate edge pixels, we check all possible

connections that satisfy the smoothness constraint (shown in Figure 6.12 (a)). The 3×3 search range could be increased if needed. A curve pixel in the curve body (for example edge a in Figure 6.12 (a)) is usually connected with two other edges, one in the tangent's forward direction, the other one in the backward direction. It is possible that in either forward or backward direction, an edge pixel has multiple valid connections. Only the one with the closest distance will be kept. A bipartite graph could be built from all chainable pixels. This bipartite graph is recorded as a table (see Figure 6.12 (b) for an example). Using this table, we could traverse the whole curve this pixel belongs to.



Edge id	Forward edge id	Backward edge id
a	b	c
b	-1	a
c	a	-1
…	…	…

(a) (b)

Figure 6.12. Smoothness constraint and chainable pixel table. (a) Smoothness constraint: forward direction: $|\boldsymbol{t}_a \cdot \boldsymbol{t}_b| > cos\ (\theta_t)$ and $\boldsymbol{t}_a \cdot \frac{\boldsymbol{S}_{ab}}{|\boldsymbol{S}_{ab}|} > cos\ (\theta_s)$; backward direction: $|\boldsymbol{t}_a \cdot \boldsymbol{t}_c| > cos\ (\theta_t)$ and $-\boldsymbol{t}_a \cdot \frac{\boldsymbol{S}_{ac}}{|\boldsymbol{S}_{ac}|} > cos\ (\theta_s)$. Red arrow represents subpixel edge, the black line segments represent subpixel edge's relative displacement to the pixel center. θ_t and θ_s are angle thresholds. (b) Chainable pixels are recorded as a table, assume no other chainable pixels connects to c and d. -1 means no valid linkable pixel.

After obtaining the chainable pixel table, each thread of GPU will pick a seed, and start traversing. Seeds could be generated by setting a high threshold to the maximum contrast just like

the Canny edge detector. Different curves may have different lengths. The processing time could vary significantly. For efficient GPU implementation, it is better to let all threads perform similar tasks that cost similar amount of time. Thus, we fix the number of traversing steps to 20, even for a 4K large image. A 20-pixel long smooth curve is considered statistically significant. The traversing process will start from the forward chaining direction, stop when it meets a pixel that has already been marked as a curve pixel, or an invalid edge id or a visited pixel. Then start backward direction if the forward direction traversing hits invalid pixel before 20 steps. After traversing, all those pixels will be marked as curve pixels if the traverse process did not hit any invalid pixel id and no loop detected. This process does not need to be synchronized across all threads, since setting a pixel as a curve pixel multiple times does not change the result. In the end, we obtained a curve pixel map. A pixel will be marked as 1 if it belongs to a curve that is longer than 20 pixels, otherwise it will be marked as 0. This method's complexity is linear to the number of edge pixels.

6.3.5.2 Curve extraction

If curve pixel detection or edge detection is our goal, then the process is done. However, sometimes we may need to extract each curve as a list of (subpixel) coordinates for further processing, such as parameterized curve fitting, curve grouping and classifications. Curve extraction is usually operated on a CPU since GPU does not support the list data structure. Extracting a curve as a list could be as simple as sequentially traverse through the whole curve. However, this method does not maximize the parallel processing power of modern CPUs. Curve extraction could be thought as a bipartite graph edge contraction problem if each curve is represented as a vertex, and curve to curve connection is represented as an edge. Each curve could be assigned a score. This score could be the smallest contrast of the whole curve. Each curve has a head and a tail. Thus there are four

types of possible connections between curves: head-tail, head-head, tail-head and tail-tail. Each curve is initialized from a single edge pixel. Every time a curve with the locally highest score (compared with head-tail contacted curves) is chosen as mergeable curves, then all those curves are merged by the head-tail, head-head, tail-head and tail-tail order. The whole curve extraction process is fully parallel. For example, during a head-tail merging, all those operations could run simultaneously. One curve can only have at most one valid head-tail connection. Thus, there is no risk of a race condition. The merging process will be applied for several rounds until no mergeable curves exist. For every new round of merging, there is no need to check every curve for mergeability. New seeds could only exist in previous altered curves or their neighbors. Overall, this parallel curve merging method complexity is linear to the number of edge pixels. Besides the parallelable curve merging process, another way to exploit parallelism is to divide a (large) image into several small regions. The curve merging process could be applied independently within each region, then across different regions.

6.4 EXPERIMENTS

To train the network, we use four types of images: 1. Images randomly cropped from blank images (subsection 6.3.4.4), no augmentation is applied; 2. Images synthesized by random operations between 2D shapes (subsection 6.3.4.1). The 2D shapes and background are randomly colored, and all image augmentation effects (blur, noise, demosaic and random JPEG quality) are applied; 3. Images synthesized similarly as Type 2, but no motion blur; 4. Images formed similarly as Type 2, but instead of coloring with random RGB colors, background and foreground images are randomly cropped from high quality image datasets such as DIV2K [60] or Pascal VOC [59]. For image Type 1, only blank image loss is used. For image Type 2, edge/non-edge contrast loss, relative contrast loss and orientation loss are applied. For image Type 3, edge contrast loss, relative

contrast loss and orientation loss are applied. Image Type 4 uses the same loss function as image Type 3. The loss-weights for the four types of training data are $0.6, 1.0, 0.01,$ and 1.0, respectively. The reasons of choosing training data like these are: Type 1 images are used to suppress the network's response to meaningless patterns in real images. However, we don't want the suppression to be the dominant force during training. Thus we choose weight 0.6, which is smaller than 1.0. For image Type 2, we have complete labels, thus all related loss functions are applied; For Type 3 images, they are used to prevent the network from ignoring sharp edges. Since the majority of training data involve motion blur, it is very likely that an overfitted network becomes insensitive to clear edges. For image Type 4, we only have partial edge information since we do not know the ground truth edge labels of the cropped images, thus we only applied the loss function to edges synthesized.

The network's performance largely depends on what training data is used. Usually, we can't expect one trained model to solve all situations. For example, a model trained with motion blur will not be optimal for clear image edge detection. It usually does not detect fine details in a clear image. Here we assume a practical situation: an image contains both noise and motion blur, motion blur is around 20 pixels, and the chained edge length is larger than the blurriness (see Figure 4.3 (b) for how blurriness is determined). To prepare the training data for this practical situation, we randomly generate combinations of ellipse and rectangular in the size range of $[30, 100]$ pixels, long thin rectangles, elliptic rings and combinations of ellipses. We precomputed 10000 of random 2D shapes with an image size of 288×288. The inside or outside of the mask could be represented using a boolean image with values "0" or "1". Edge related labels (mask value, subpixel displacement and tangents) could be sparsely represented to save the disk space. During training, we randomly pick two layers of the precomputed shapes, recover their dense edge labels,

randomly pick one background image and two foreground images, and then synthesize the image using methods described in 6.3.4.1.

For training, one batch contains 48 images, 12 for each of those four types. Due to the image augmentation, the image size will be reduced to 192×192. For 60 epochs of training (different image types have their own 2D shape pools), it takes about 13 hours with a single GTX 1080ti GPU and Intel i9 7940X CPU.

6.4.1 *Qualitative comparison*

6.4.1.1 Image with artificial white noise

ELSDc [27] is the most popular ellipse and line segment detector. We compare our method with it using ELSDc's testing images. ELSDc is parameter free. Our model needs parameters such as the threshold value for contrast in the Canny edge detector. However, thanks to the moving variance normalization layer, we could fix the parameters without any change. During testing, choose 3.0 and 0.1 as contrast and relative contrast thresholds to select seeds, then detect edges that belong to 10 pixels long curves. We keep the setting the same for all testing images. The test images are camera calibration patterns with an array of circles (Figure 6.13) or rings (Figure 6.14). We add Gaussian white noise of different standard deviation to the image, then clip pixel value to [0.0,1.0]. Gaussian white noise is different from the signal dependent noise model used in our training pipeline, thus it does not have direct bias toward our model. Besides ELSDc [27], we also add one of the recent noisy edge detector [23] into comparison.

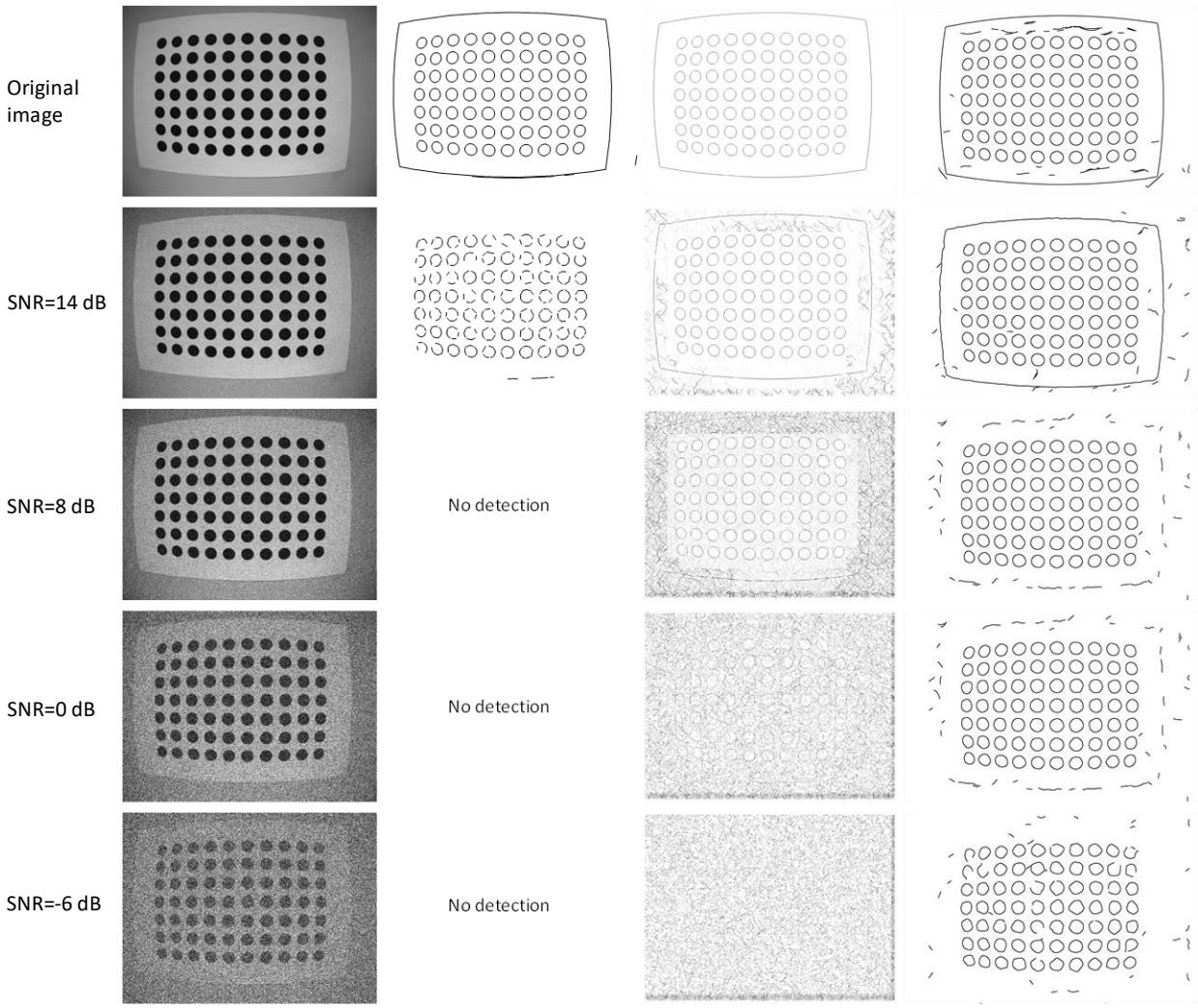


Figure 6.13. Detecting curves from a circular calibration pattern. From left to right: inputs images (with different noise level), results of ELSDc [27], faint curved edge detector [23] and our method. $\mathrm{SNR} = 20 \log_{10} \frac{\text{Avarage pixel value}}{\text{Noise Standard Deviation}}$ Note: Source code of faint curved edge detector [23] is very memory demanding. We divide a large image into many small overlapping patches and process them individually. We can see grid artifacts of the detected edges. The default parameters of faint curved edge detector are used.

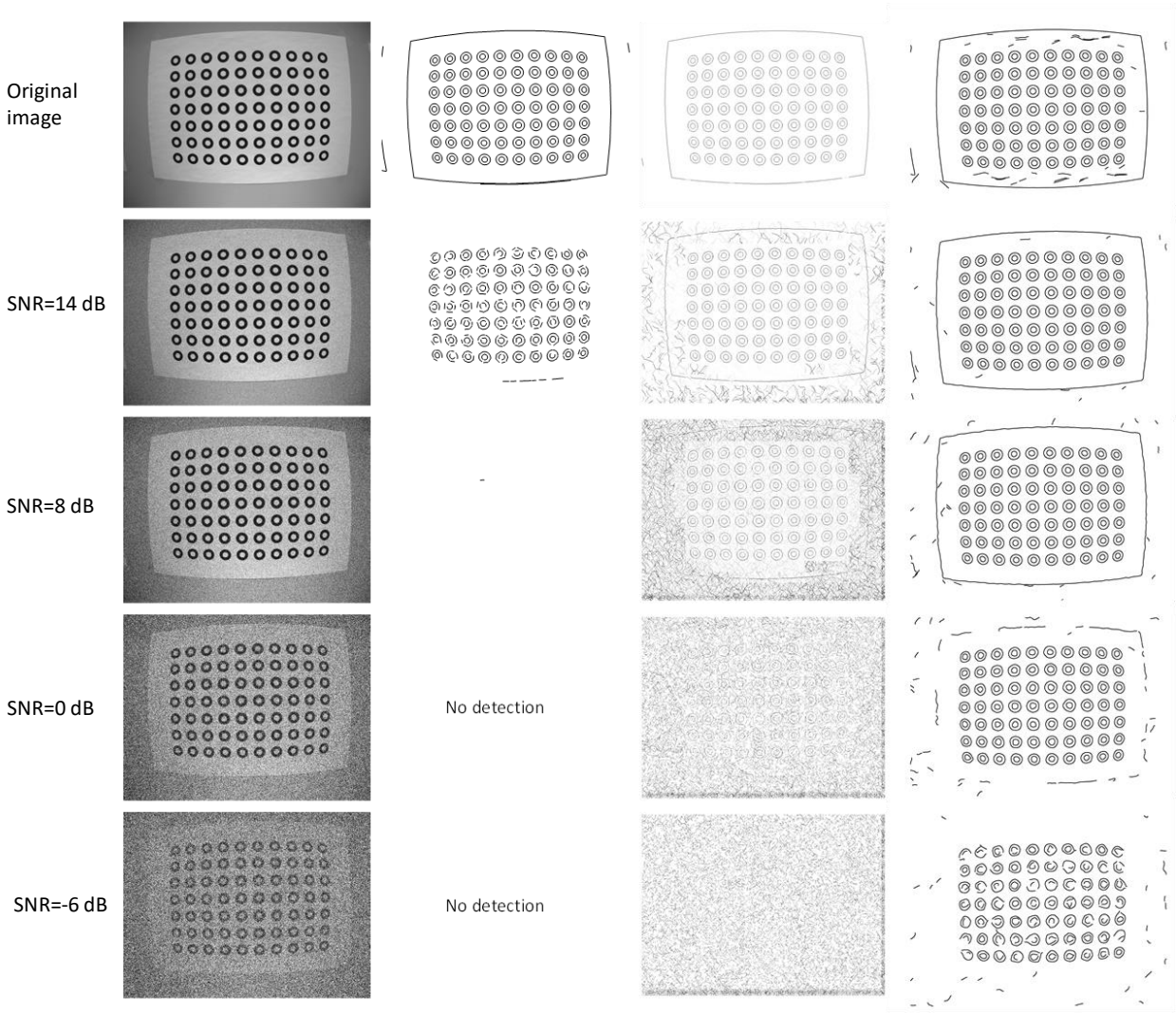


Figure 6.14. Detecting curves from a ring calibration pattern. See Figure 6.13 for explanation of each sub-image.

From the comparison we could see that ELSDc is very sensitive to noise. ELSDc has visually the best result when image is clean, however, its performance decreases dramatically with noise involved. It fails to detect anything when SNR drops below 8 dB, while our method still detects meaningful curve segments even when noise is larger than the signal. Compare to noisy edge detector [23], our method could handle more severe noise and detect cleaner edges.

6.4.1.2 Real images with blur or faint edges

To qualitatively show the benefits of our proposed method in handling real world images, we apply our model to several cellphone images that contain motion blur and faint edges. Testing images are separated into two groups, one group contains three motion blurred images and the other group contains four images with faint edges. We compare our model with dark channel deblur [49] and SRN [21] on the blurred images. For images with faint edges, we compare our method with faint edge detector [23] and ELSDc [27]. Since ELSDc [27] and faint edge detector [23] only take gray images as input, we first convert color image into gray, then apply the detectors. Comparison results are shown in Figure 6.15, Figure 6.16, Figure 6.17 and

Figure 6.18.

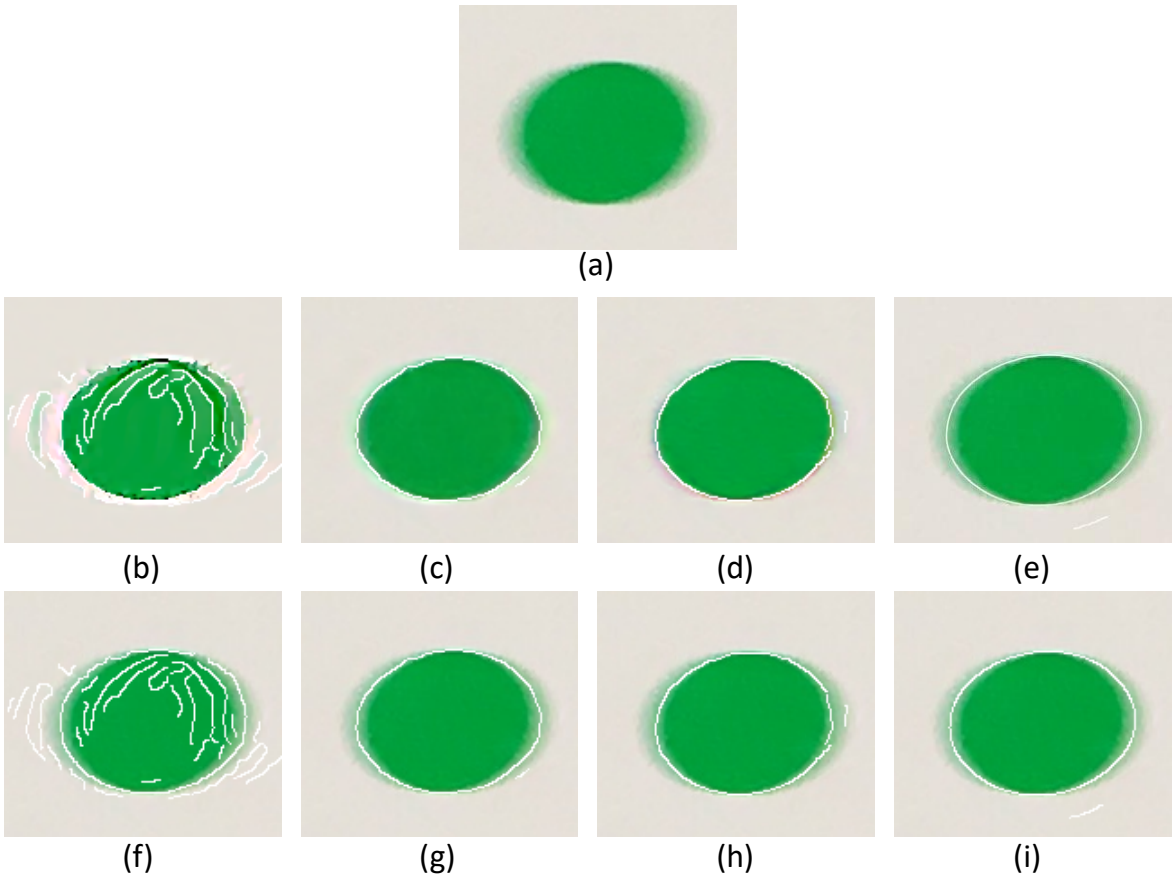


Figure 6.15. Detecting edge from a lightly blurred elliptical marker. (a) input image; (b), (c), (d) restoration result of dark channel deblur [49], SRN color [21] and SRN LSTM [21], overlaid with detected edge pixels. (e) Input image overlaid with our curve extraction result. (f), (g), (h) input image overlaid with edges detected in (b), (c), (d). (i) Input image overlaid with our edge detection result. Note: We use white color to represent detected edges. For restored images, the threshold of contrast is 0.1. Images are smoothed by a Gaussian filter with $\sigma = 1.0$ before edge detection. Smoothness criteria are $\theta_s = 60$ and $\theta_t = 45$ degrees (see explanation of the smoothness constraint in Figure 6.12 (a)). All images use the same smoothness criteria and only edges that can form longer than 10-pixel curves are kept.

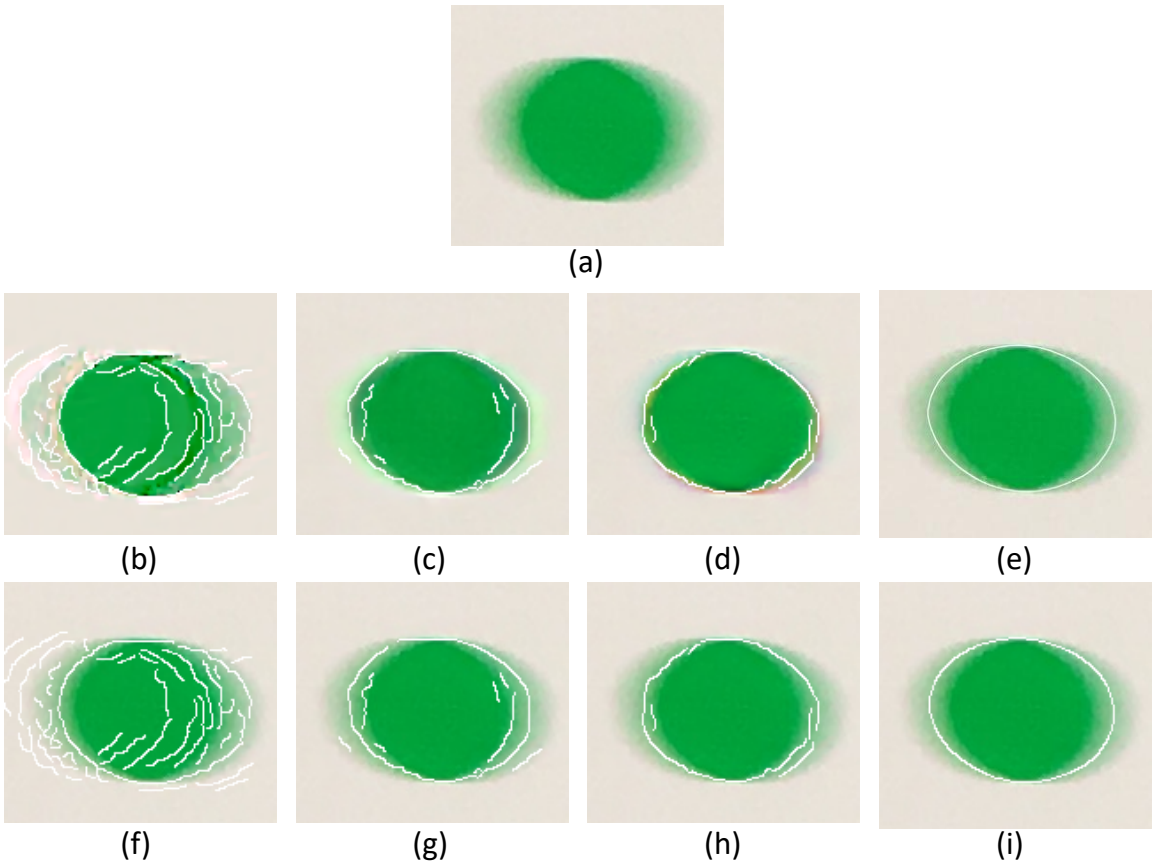


Figure 6.16. Detecting edges from a heavily blurred elliptical marker. See Figure 6.15 for the explanation of each sub-image.

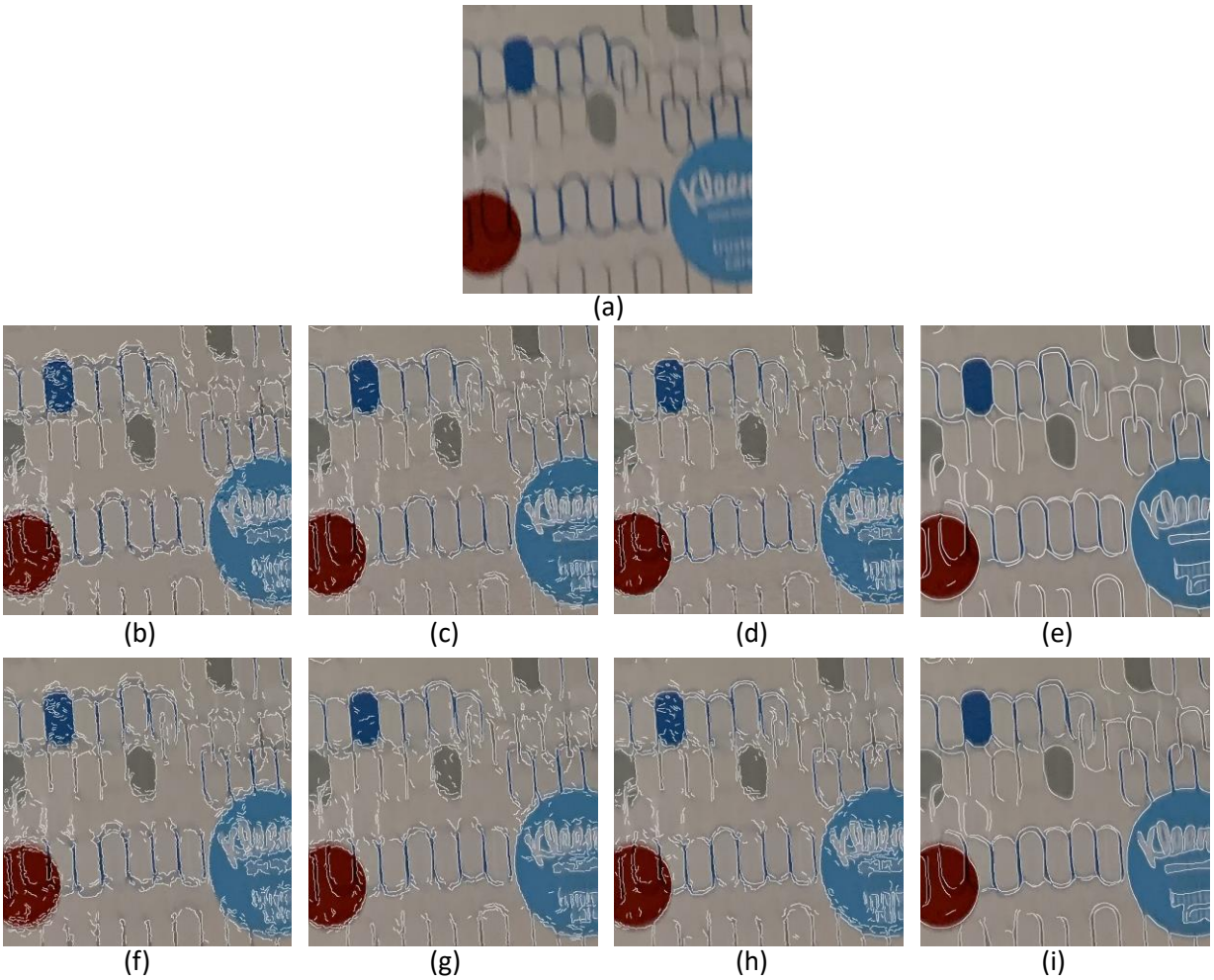


Figure 6.17. Qualitative comparison on real motion blurred images. See Figure 6.15 for explanation of each sub-image.

For blurred images, we could see that edges detected by our method fall onto the center of the blur profile (for example Figure 6.15 (e) (i) and Figure 6.16 (e) (i)). Our method also extracts long, smooth and clean curves, which usually facilitates higher level curve operations such as grouping and splitting. While artifacts of deblurring algorithms create noisy curves which increase the burden of future curve processing.

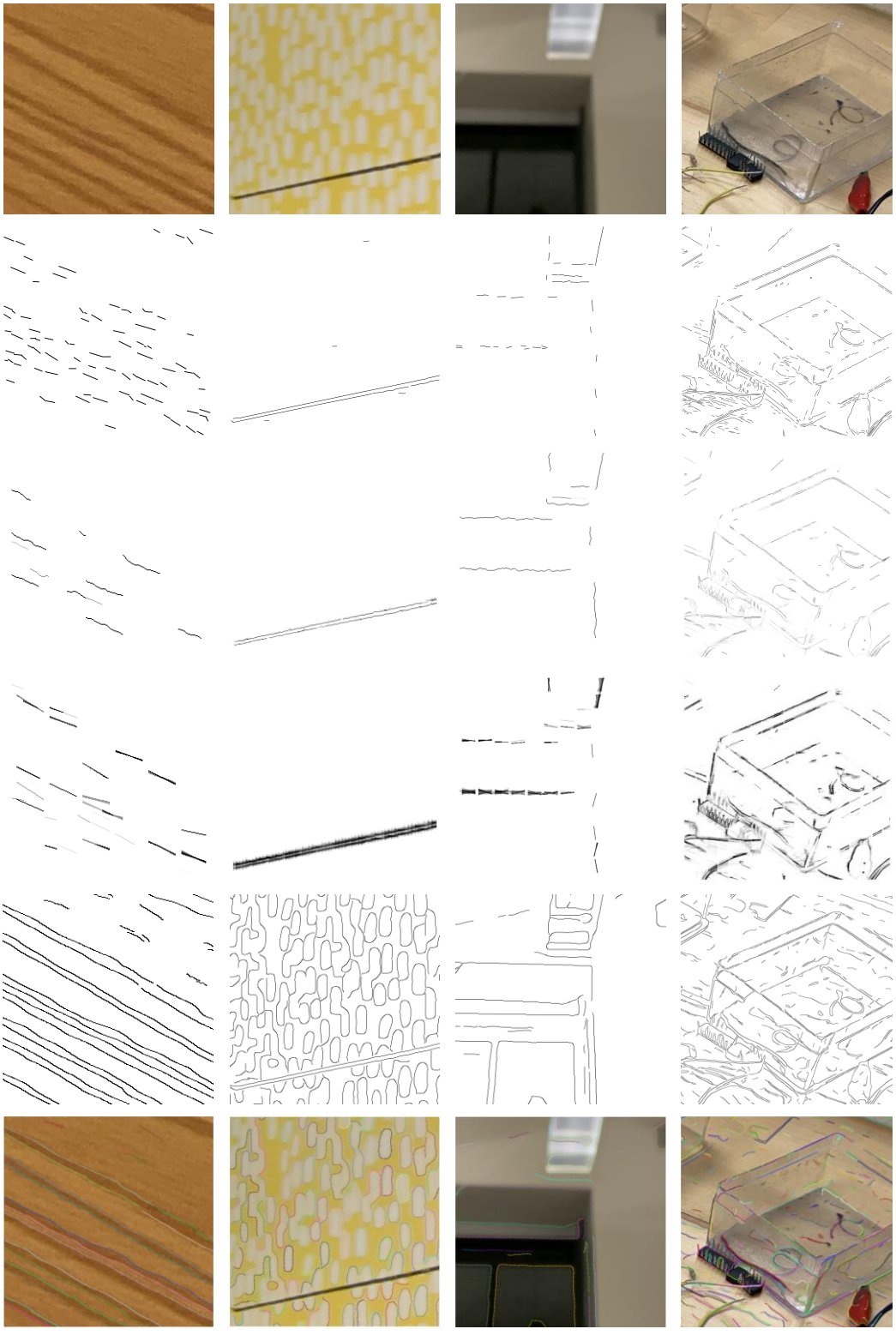


Figure 6.18. Real cellphone images with faint edges. From top to bottom: input images, ELSDc [27], faint curved-edge edge detector [23], faint straight-edge detector [23], our curve pixel

detector and our curve extraction (different color represents different curves). Note: ELSDc is parameter free. Faint curved edge detector's default parameter is used here. For faint straight edge detector we use sigma $= 0.05$, and for lower threshold sigma $= 0.01$ see Figure 6.19. For our method, we fix the parameters for all examples in this work, no parameter tuning for individual image. For faint edges in real photos (Figure 6.18), those four testing images are (from left to right): table texture, tissue box, out of focus image of window and transparent plastic box. Those are all very common objects in real world. For the table texture image, our method is the only method that extracts long smooth curves from the table texture. For the tissue box, both ELSDc [27] and faint edge detector [23] (both curved and straight edge versions) failed to detect the patterns on the box. For the faint straight edge detector, we could lower $sigma$ to get better response (Figure 6.19). However, the result becomes noisier. For the out of focus window image, only our method and faint straight edge detector with $sigma = 0.01$ (Figure 6.19) detect the edges of window. However, our method has better details than low threshold faint straight edge detector. For the transparent plastic box image, our method detects equal amount of details as low threshold faint straight edge detector but stays as clean as ELSDc. Overall, our method has the best detail, equal or better cleanness and better edge location.

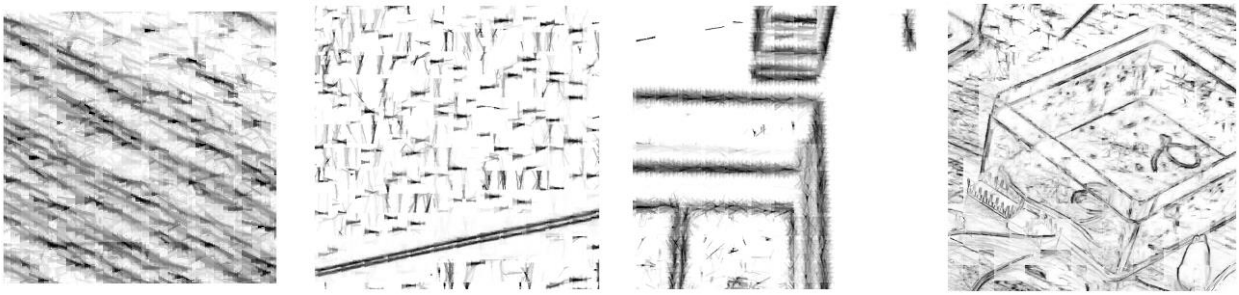


Figure 6.19. Faint straight edge detector [23] with sigma = 0.01.

6.4.2 *Quantitative comparison: 3D location estimation with circular markers*

To quantitatively evaluate the accuracy of the detected curve segments on real images, we took several pictures of three circular markers with different motion blur. Given camera parameters and the circular marker size, the 3D positions of the markers could be accurately calculated from their 2D ellipse parameters on the images. Using the relationship between the ellipse parameters and

the 3D positions of the markers, the 3D distances between the marker centers can be calculated. We have three sets of images, each image is similar to Figure 4.3(a) except for different motion blur. One set contains clear images. The 2nd set contains images of around 10 pixels motion blur. The third set contains images of around 20 pixels or larger motion blur.

For our method and SRN deblurring, curve segments are extracted by chaining edge pixels if the angle difference is smaller than 45 degree. Then long curves with green pixels (for easy detection) will be fitted as an ellipse using the Taubin's method [43] with outlier removing [39]. We did not use more sophisticated ellipse detection methods that group and split curves, since what we propose here is a curve segment extractor not ellipse detector. Since the images are relatively simple, a good curve segment detector should be able to extract a sufficiently long curve for accurate ellipse parameter estimation. Markers' 3D locations (relative to camera) and orientations are estimated from the ellipse parameters. Since three circular markers are on the same plane, we will exhaustively search for all possible three marker combinations that satisfy:

$$\boldsymbol{n_i} \cdot \boldsymbol{N} < 0.98 \tag{6.23}$$

where $\boldsymbol{n_i}$ is the orientation of one of the three circular markers, and N is the orientation of the plane formed by centers of all three markers. Their relative 3D distances (d_0, d_1 and d_2 in Figure 4.3(a)) are compared with ground truths measured by a physical ruler.

Table 6.4. Average error of estimating circular markers' relative (3D) distances (in mm) when images are clear

Blurriness	Lu et al. [65]	ELSDc [27]	SRN color [21]	SRN LSTM [21]	Dark channel deblur [49]	Ours
clear	0.5	**0.3**	**0.3**	0.7	0.8	0.46
clear	1.8	2.6	1.7	**1.6**	1.7	2.8
clear	2.8	2.7	2.6	**2.3**	2.6	2.9
Avg	1.7	1.9	**1.5**	**1.5**	1.7	2.1

Std	1.2	1.4	1.2	**0.8**	0.9	1.4

Table 6.5. Average error of estimating circular marker's relative (3D) distances (in mm) when image's motion blurriness is around 10 pixels. Note: for Lu et al. [65], we keep increasing median filter's size until it detects three markers. For ELSDc [27], we tried Gaussian filter ($\sigma = 1.0 \sim 5.0$) and median filter (with size $3 \sim 7$). If still no coplanar markers detected, we mark it as 'Failed'. For SRN [21] and dark channel deblur [49] restored images, we apply the same smoothing strategy as ELSDc.

Blurriness	Lu et al. [65]	ELSDc [27]	SRN color [21]	SRN LSTM [21]	Dark channel deblur [49]	Ours
12.6	5.2	Failed	3.1	3.2	3.3	**2.2**
9.8	3.5	Failed	19.8	6.8	2.6	**2.0**
11.2	3.5	13	20.6	6.1	6.2	**2.8**
8.5	3.5	Failed	Failed	1.1	1.4	**0.7**
14.2	6.5	8.4	3.7	6.3	Failed	**1.3**
Avg	4.44	NA	NA	4.7	NA	**1.8**
Std	1.4	NA	NA	2.5	NA	**0.8**

Table 6.6. Average error of estimating circular marker's relative (3D) distances (in mm) when image's motion blurriness is around 20 pixels or more. Note: * means three coplanar markers are manually selected, since no three ellipses satisfy the coplanar criteria. Smoothing strategy is the same as Table 6.5.

Blurriness	Lu et al. [65]	ELSDc [27]	SRN color [21]	SRN LSTM [21]	Dark channel deblur [49]	Ours
19.4	7.6*	Failed	4.7	7.7	5.1	**1.8**
23.9	9.9*	Failed	25.6	15.0	Failed	**3.4**
27.2	9.9	Failed	9.3	Failed	Failed	**3.4**
23.4	12	Failed	7.3	9.1	2.4	**1.5**
30	58.4	Failed	5.4	32.23	Failed	**4.3**
Avg	19.6	NA	10.46	NA	NA	**2.9**
Std	21.8	NA	8.7	NA	NA	**1.2**

For both Lu et al. [65] and ELSDc [27], we use their internal ellipse fitting algorithms. Since the major error is caused by inaccurate edge detection (caused by motion blur), the ellipse fitting algorithm is not the major source of error. Lu et al. [65] is the only method that applies curve grouping. For SRN [21] and dark channel deblur [49], except that smoothing is applied, they use the same curve extraction as our method. The reasons of choosing those baselines are: 1. We want to show how motion blur could fail algorithms that implicitly require good quality images as input; 2. State of the art deblurring does not necessarily recover edges with good location. See the description of each table to find how we calculate the result.

Our method is not the most accurate method when the input image is clear. However, all methods estimate the distance with decent accuracy. When there is motion blur, our method achieves the best result on every testing image.

6.4.3 *Extension: Extract fine curves from low noise images*

We should note that the loss function, synthetic data generation and curve pixel detection and extraction proposed in this chapter do not just work for blur and noisy edge detection. With slightly adjusted training data, we could train a neural network that can help to extract curve from small objects. The modifications are: 1. Change large 2D shapes in training data to thin small shapes; 2. Since we do not consider motion blur, remove the motion blur effect; 3. Change the network to a smaller network and add a transpose convolution to generate twice the resolution of output; this step is not mandatory, however, a smaller network and a higher output resolution make training faster and curve extraction easier. The network we use here is shown in Figure 6.20. Loss function, edge detection and curve extraction are the same. However, we lower the curve length threshold to 10, since small objects have short smooth contours.

Usually, the number of corner edge pixels is much smaller than flat edge pixels. We wish the network could be sensitive to the bending of edge. Thus, we propose to weight the edge related loss by the corner response. Assume M is the 2×2 symmetric matrix in Harris corner detector [66]:

$$M = \begin{pmatrix} A & C \\ C & B \end{pmatrix} \tag{6.24}$$

Then the weight is defined as:

$$w = \frac{2\lambda_{min}}{A+B+\epsilon} \tag{6.25}$$

ϵ is a small value to avoid dividing by 0; λ_{min} is the smaller eigen value of the matrix. w is 0 for edge or flat region; 1 for perfect corner. e^w is used to weight edge pixels. The corner response map is usually blurred to cover pixels that are close to corners.

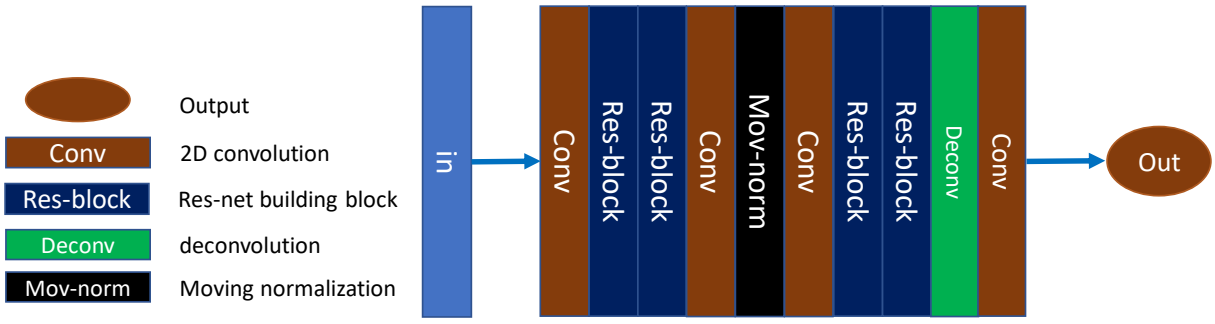


Figure 6.20. Network used for thin object curve pixel detection. Res-block channel sizes are: 32, 64, 64, and 32. The output has 8 channels and twice the resolution of the input.

Figure 6.21. Extracting fine curve pixels. From left to right: input image, ELSDc [27] and our curve pixel detection result.

Figure 6.21 right column shows our curve pixels detection result on clear images. Compared with ELSDc [27], our method extracts more details and it is robust to JPEG compression artifacts.

Chapter 7. CONCLUSION AND FUTURE WORK

Measuring 3D distance from a 2D image is a challenging problem, especially when it involves a deformable dynamic object. In this work, we first propose a new technique to measure an infant's length from one 2D photograph. The method is easy to apply and requires only circular stickers and a regular cellphone camera. Thus, a parent could easily apply this method regularly to keep track of the infant's growth. Compared to the traditional infantometer method, this new method can be applied by one person instead of three persons. It also does not need special equipment (i.e., an infantometer). Our experimental results show that it has a good accuracy of about 3% in actual field trials.

To reliably detect markers during this study, we developed an ellipse estimator based on ellipse growing. Simulation results compared with other state-of-the-art methods show that it can produce results with good accuracy for real-world images and have much better robustness and faster speed.

To handle markers' motion blur that encountered in the applications, we proposed a CNN-based approach for ellipse restoration. We also proposed a fully automatic synthetic training data generation method which can generate realistic data for training the CNN in order to obtain good results. The method considers different artificial effects during the image formation. We demonstrated the effectiveness of our method using actual images taken by a regular cellphone camera.

Inspired by neural network's effectiveness in recovering circular marker's boundary curve location, we study using CNN to detect more general low-level curves from degraded photos. We developed a pipeline to extract curves from an image under different kinds of degradation such as heavy noise and motion blur. It performs better than existing curve detection and extraction

algorithms when the image contains heavy noise and motion blur. It also performs better than first apply restoration then apply curve extraction. We propose a contrast aligning loss that is directly related to the edge location and orientation. We also propose a data synthesizing method and show how to generalize the model to real-world photos. Efficient curve detection and extraction algorithms are developed to accompany the CNN model. The effectiveness and flexibility are verified by qualitative and quantitative comparisons with other related state of the art methods.

To further improve the performance of curve detection, there are several directions to explore. One direction is better subpixel edge or curve segment representation that is compatible with neural networks. In this dissertation we only explore maximum contrast as orientated edge representation, there are still many other alternatives in classical curve detection literatures. Different representation may have its unique benefits. Another direction is new network structures. It is worth to keep experimenting if we could further reduce the network size without sacrificing the performance. Also, the effect of network parameters such as the channel size and the convolution type are still not fully studied yet. Besides the model, there is still much space to improve in training data. Several effects have not been simulated yet, for example surface reflection, over exposure, and bad pixels from the sensor.

It is also worth exploring the combination of curve-based 3D reconstruction and faint/blurred curve/edge detectors. Edge/curve-based 3D reconstruction, although proposed very early [67][1], does not receive as much attention as point-based reconstruction. For photos with motion blur or weak textures, it is easier to recover long curve than recovering the corner features. However, compare with point-based features, curve only provides good localization in its perpendicular direction. Matching curves from different images is a challenging task. Point features could be used to initialize curve matching [45], however this usually implicitly assumes good quality input

images. There are still some practical problems that need to be solved before integrating the faint/blurred curve detector into the curve/edge-based structure-from-motion pipeline. A successful adaptation of the proposed curve detector will not only make 3D reconstruction more accurate but also more robust to regular image degradation effects such as motion blur and noise.

BIBLIOGRAPHY

[1] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge uiniversity press, 2003.

[2] M. Salzmann and P. Fua, *Deformable Surface 3D Reconstruction from Monocular Images*, vol. 2. Morgan & Claypool, 2010.

[3] D. G. Low, D. G. Lowe, and D. G. Low, "Distinctive Image Features from Scale-Invariant Keypoints," *Int. J. Comput. Vis.*, pp. 91–110, 2004.

[4] J. T. Barron and J. Malik, "Shape, Illumination, and Reflectance from Shading," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 8, pp. 1–5, 2015.

[5] D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern Recognit.*, vol. 13, no. 2, pp. 111–122, 1981.

[6] Q. Xie, Yonghong; Ji, "A new efficient ellipse detection method," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, 2002, p. Vol. 2.

[7] "OpenCV." [Online]. Available: https://opencv.org/.

[8] "scikit-image." [Online]. Available: https://scikit-image.org/docs/dev/api/skimage.html.

[9] D. K. Prasad, M. K. H. Leung, and S. Y. Cho, "Edge curvature and convexity based ellipse detection method," *Pattern Recognit.*, vol. 45, no. 9, pp. 3204–3221, 2012.

[10] J. Sun, W. Cao, Z. Xu, and J. Ponce, "Learning a Convolutional Neural Network for Non-uniform Motion Blur Removal."

[11] A. Chakrabarti, "A neural approach to blind motion deblurring," in *European Conference on Computer Vision*, 2016.

[12] S. Nah, T. H. Kim, and K. M. Lee, "Deep multi-scale convolutional neural network for dynamic scene deblurring," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 257–265.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770--778.

[14] P. Dollár and C. L. Zitnick, "Fast edge detection using structured forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 8, pp. 1558–1570, 2015.

[15] S. Xie and Z. Tu, "Holistically-nested edge detection," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1395--1403.

[16] Y. Liu, M. M. Cheng, X. Hu, K. Wang, and X. Bai, “Richer convolutional features for edge detection,” in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, vol. 2017-Janua, pp. 5872–5881.

[17] Z. Yu, C. Feng, M. Y. Liu, and S. Ramalingam, “CASENet: Deep category-aware semantic edge detection,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5964--5973.

[18] Z. Yu *et al.*, “Simultaneous Edge Alignment and Learning,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 388--404.

[19] F. Devernay, “A Non-Maxima Suppression Method for Edge Detection with Sub-Pixel Accuracy,” *Rapp. Rech. Natl. Rech. En Inform. En Autom.*, 1995.

[20] S. Su, M. Delbracio, J. Wang, G. Sapiro, W. Heidrich, and O. Wang, “Deep video deblurring for hand-held cameras,” in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, vol. 2017-Janua, pp. 237–246.

[21] X. Tao, H. Gao, X. Shen, J. Wang, and J. Jia, “Scale-Recurrent Network for Deep Image Deblurring,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 8174–8182, 2018.

[22] N. Ofir, M. Galun, B. Nadler, and R. Basri, “Fast detection of curved edges at low SNR,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 213–221, 2016.

[23] N. Ofir, M. Galun, S. Alpert, A. Brandt, B. Nadler, and R. Basri, “On Detection of Faint Edges in Noisy Images,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PP, no. c, pp. 1–1, 2019.

[24] N. Ofir and Y. Keller, “Multi-scale Processing of Noisy Images using Edge Preservation Losses,” *arXiv Prepr.*, 2018.

[25] B. B. Kimia, X. Li, Y. Guo, and A. Tamrakar, “Differential Geometry in Edge Detection: Accurate Estimation of Position, Orientation and Curvature,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 7, pp. 1573–1586, 2019.

[26] M. de Onis, A. W. Onyango, J. Van den Broeck, W. C. Chumlea, and R. Martorell, “Measurement and standardization protocols for anthropometry used in the construction of a new international growth reference,” *Food Nutr. Bull.*, vol. 25, no. 1 SUPPL. 1, pp. S27-36, 2004.

[27] V. Pătrăucean, P. Gurdjos, and R. Grompone Von Gioi, “Joint A Contrario Ellipse and Line Detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 788–802, 2017.

[28] C. Lu, S. Xia, M. Shao, and Y. Fu, “Arc-support Line Segments Revisited: An Efficient High-quality Ellipse Detection,” *IEEE Trans. Image Process.*, 2019.

[29] Q. Chen, H. Wu, and T. Wada, “Camera Calibration with Two Arbitrary Coplanar Circles,” in *European Conference on Computer Vision*, 2004, pp. 521–532.

[30] B. Su, S. Lu, and C. L. Tan, “Blurred image region detection and classification,” in *ACM international conference on Multimedia*, 2011.

[31] R. Liu, Z. Li, and J. Jia, “Image partial blur detection and classification,” in *Computer Vision and Pattern Recognition*, 2008.

[32] J. Shi, L. Xu, and J. Jia, “Discriminative blur detection features,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 2965–2972, 2014.

[33] M. Kass, A. Witkin, and D. Terzopoulos, “Snakes: Active Contour Models,” *Int. J. Comput. Vis.*, 1988.

[34] Z. Hu, N. Ahuja, M.-H. Yang, J.-B. Huang, and W.-S. Lai, “A Comparative Study for Single Image Blind Deblurring,” pp. 1701–1709, 2016.

[35] L. Xu, J. S. Ren, C. Liu, and J. Jia, “Deep Convolutional Neural Network for Image Deconvolution,” in *Advances in neural information processing systems*, 2014, no. 413113, pp. 1–9.

[36] L. Zhong, S. Cho, D. Metaxas, S. Paris, and J. Wang, “Handling noise in single image deblurring using directional filters,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 612–619, 2013.

[37] L. Yuan *et al.*, “Image deblurring with blurred/noisy image pairs,” in *ACM Transactions on Graphics*, 2007, vol. 26, no. 3, p. 1.

[38] M. Jin, S. Roth, and P. Favaro, “Noise-Blind Image Deblurring,” pp. 3510–3518, 2017.

[39] M. Shao, Y. Ijiri, and K. Hattori, *Grouped outlier removal for robust ellipse fitting*. IEEE, 2015, pp. 138–141.

[40] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, “Pyramid scene parsing network,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6230–6239, 2017.

[41] C. J. Schuler, M. Hirsch, S. Harmeling, and B. Scholkopf, “Learning to Deblur,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 7, pp. 1439–1451, 2016.

[42] A. Trujillo-Pino, K. Krissian, M. Alemán-Flores, and D. Santana-Cedrés, “Accurate subpixel edge location based on partial area effect,” *Image Vis. Comput.*, vol. 31, no. 1, pp. 72–90, 2013.

[43] G. Taubin, “Estimation of Planar Curves, Surfaces, and Nonplanar Space Curves Defined by Implicit Equations with Applications to Edge and Range Image Segmentation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, no. 11, 1991.

[44] X. Yonghong and J. Qiang, “A new efficient ellipse detection method,” *Proc. - Int. Conf. Pattern Recognit.*, vol. 16, no. 2, pp. 957–960, 2002.

[45] I. Nurutdinova and A. Fitzgibbon, “Towards pointless structure from motion: 3d reconstruction and camera parameters from general 3D curves,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, vol. 2015 Inter, pp. 2363–2371.

[46] D. Rao, S. J. Chung, and S. Hutchinson, “CurveSLAM: An approach for vision-based navigation without point features,” *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 4198–4204, 2012.

[47] S. Maity, A. Saha, and B. Bhowmick, “Edge SLAM: Edge Points Based Monocular Visual SLAM,” in *Proceedings - 2017 IEEE International Conference on Computer Vision Workshops, ICCVW 2017*, 2018, pp. 2408–2417.

[48] L. Liu, D. Ceylan, C. Lin, W. Wang, and N. Mitra, “Image-based reconstruction of wire art,” *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–11, 2017.

[49] J. Pan, D. Sun, H. Pfister, and M. H. Yang, “Blind image deblurring using dark channel prior,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1628–1636.

[50] John Canny, “A Computational Approach To Edge Detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–714, 1986.

[51] R. Grompone Von Gioi and G. Randall, “A sub-pixel edge detector: An implementation of the canny/devernay algorithm,” *Image Process. Line*, vol. 7, pp. 347–372, Nov. 2017.

[52] S. Di Zenzo, “A note on the gradient of a multi-image,” *Comput. Vision, Graph. Image Process.*, vol. 33, no. 1, pp. 116–125, Jan. 1986.

[53] C. Akinlar and C. Topal, “ColorED: Color edge and segment detection by Edge Drawing (ED),” *J. Vis. Commun. Image Represent.*, vol. 44, pp. 82–94, 2017.

[54] R. C. Gonzalez and R. E. Woods, *Digital image processing*. 2007.

[55] Nvidia, *Programming guide*. 2010.

[56] M. Abadi *et al.*, *Tensorflow: A system for large-scale machine learning*. 2016.

[57] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, “FlowNet 2.0: Evolution of optical flow estimation with deep networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2462--2470.

[58] N. Chernov and H. Ma, “Least squares fitting of quadratic curves and surfaces,” *Comput. Vis.*, pp. 287–302, 2011.

[59] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman, “The PASCAL Visual Object Classes Challenge - a Retrospective,” *Int. J. Comput. Vis.*, vol. 111, pp. 98--136, 2015.

[60] E. Agustsson and R. Timofte, “NTIRE 2017 Challenge on Single Image Super-Resolution: Dataset and Study,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2017, vol. 2017-July, pp. 1122–1131.

[61] A. Foi, M. Trimeche, V. Katkovnik, and K. Egiazarian, “Practical Poissonian-Gaussian noise modeling and fitting for single-image raw-data,” *IEEE Trans. Image Process.*, vol. 17, no. 10, pp. 1737–1754, 2008.

[62] S. Guo, Z. Yan, K. Zhang, W. Zuo, and L. Zhang, “Toward Convolutional Blind Denoising of Real Photographs,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1712–1722.

[63] H. S. Malvar, L. W. He, and R. Cutler, “High-quality linear interpolation for demosaicing of Bayer-patterned color images,” *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, vol. 3, pp. 5–8, 2004.

[64] C. Topal and C. Akinlar, “Edge Drawing: A combined real-time edge and segment detector,” *J. Vis. Commun. Image Represent.*, vol. 23, no. 6, pp. 862–872, 2012.

[65] C. Lu, S. Xia, M. Shao, and Y. Fu, “Arc-support Line Segments Revisited: An Efficient High-quality Ellipse Detection,” *IEEE Trans. IMAGE Process.*, 2019.

[66] C. G. Harris and M. Stephens, “A combined corner and edge detector,” in *Alvey vision conference*, 1988, pp. 10--5244.

[67] R. Berthilsson, K. Åström, and A. Heyden, “Reconstruction of general curves, using factorization and bundle adjustment,” *Int. J. Comput. Vis.*, vol. 41, no. 3, pp. 171–182, 2001.

[68] A. Paszke, S. Gross, S. Chintala, and G. Chanan, “Pytorch,” *Comput. software. Vers. 0.3*, vol. 1, 2017.

APPENDIX A

A.1. Differentiable sampling/warping via bicubic interpolation

Many image processing tasks (such as resizing, subpixel edge estimation and optical flow calculation) may involve estimating off-grid pixel values. Bilinear interpolation is the easiest and most popular way to get sub-grid information. As shown in [57], this operation is differentiable with respect to not only image pixel values but also the input non-integer coordinates. However, the first order gradient of bilinear interpolation is not smooth, a better alternative is bicubic interpolation. Bicubic interpolation could help us to estimate smooth first order gradient at arbitrary coordinates. Here, we show how to implements bicubic interpolation as a differentiable layer, it could also be used as drop in replacement of bilinear image warping.

Assume the input image is $\boldsymbol{I}$ and the sampling coordinate variables are x and y. They could be represented as the integer part and non-integer part:

$$x = \lfloor x \rfloor + \alpha$$

$$y = \lfloor y \rfloor + \beta$$

where $\lfloor x \rfloor$ and $\lfloor y \rfloor$ are the integer parts of the coordinates, and α, β ($\in [0,1]$) are the subpixel parts. The bicubically sampled/warped image is:

$$I'(x,y) = \sum_{i=0,1,2,3} \sum_{j=0,1,2,3} \alpha^i \beta^j a_{ij} I(x_i, y_j)$$

where $x_{\{0,1,2,3\}} = \{\lfloor x \rfloor - 1, \lfloor x \rfloor, \lfloor x \rfloor + 1, \lfloor x \rfloor + 2\}$, $y_{\{0,1,2,3\}} = \{\lfloor y \rfloor - 1, \lfloor y \rfloor, \lfloor y \rfloor + 1, \lfloor y \rfloor + 2\}$ and a_{ij}s are bicubic interpolation coefficients (constants). The gradients (for backpropagation calculation) could be defined as:

$$\begin{cases} \dfrac{\partial I'(x,y)}{\partial I(x_i,y_j)} = \alpha^i \beta^j a_{ij} \\ \dfrac{\partial I'(x,y)}{\partial x} = \displaystyle\sum_{i=1,2,3} \sum_{j=0,1,2,3} i \cdot \alpha^{i-1} \beta^j a_{ij} I(x_i,y_j) \\ \dfrac{\partial I'(x,y)}{\partial y} = \displaystyle\sum_{i=0,1,2,3} \sum_{j=1,2,3} j \cdot \alpha^i \beta^{j-1} a_{ij} I(x_i,y_j) \end{cases}$$

This differentiable layer could be directly implemented using the build-in clipping operation provided by many neural network framework (such as TensorFlow [56] and Pytorch [68]). A more efficient way is to implement the whole operation in CUDA [55]. Empirically we found that a CUDA implementation could save 90% of the running time.

VITA

MAOLONG TANG

EDUCATION

Ph.D. candidate in Electrical Engineering, University of Washington, WA — Current

MS in Physics, North Carolina State University, Raleigh, NC — June, 2014

BS in Optics, University of Science and Technology of China, Hefei, China — June, 2011

PUBLICATIONS

- Maolong Tang, Ming-Ting Sun, Leonardo Seda, James Swanson, and Zhengyou Zhang, "Measuring infant length using images," APSIPA, 2018.
- Maolong Tang and Ming-Ting Sun, "Using CNN for ellipse estimation in an infant length measurement application," APSIPA, 2018.
- Xiang Zhang, Phillip Zhou, Ming-Ting Sun, Maolong Tang, Shanshe Wang, Siwei Ma, and Wen Gao, "Surface Light Field Compression using a Point Cloud Codec," IEEE JETCAS 2019.
- Xiang Zhang, Phillip Zhou, Ming-Ting Sun, Maolong Tang, Shanshe Wang, Siwei Ma, and Wen Gao, "A Framework for Surface Light Field Compression," ICIP, 2018.
- Eliot Gann, Brian Collins, Maolong Tang, John R. Tumbleston, Subhrangsu Mukherjee, and Harald Ade," Origins of Polarization-dependent anisotropic X-ray scattering from organic thin films," Journal of Synchrotron Radiation, 2016.

In preparation

- Maolong Tang and Ming-Ting Sun, “Exact curve location estimation from noisy and motion blurred images.”

RESEARCH EXPERIENCE

Multimedia Signal Processing Lab, University of Washington — Jan. 2016-Present

- Faint curve detection in heavy noise, motion blurred images
- Restoring elliptical markers with CNN for accurate 3D distance estimation
- CNN optical flow for edge pixels

- Infant length measurement with images

Ade's Research Group, NC State University, Raleigh, NC Jan. 2013-
- Light scattering theory and numerical simulation Jul. 2014
- Lattice Boltzmann method for polymer spinodal decomposition simulation
- Characterizing organic material with X ray scattering

Cao's Research Group, NC State University, Raleigh, NC May. 2012-
- Analytical methods for light emission simulation Dec. 2012
- MoS2 transistor characterization

Key Lab of Quantum Information of Chinese Academy of Science Oct. 2009-
University of Science and Technology of China, Hefei, China Jun. 2011
- FDTD method for microcavity whispering gallery mode simulation

TEACHING EXPERIENCE

University of Washington, Seattle, WA 2014-Present
- 9 quarters of Physics TA: mechanics, electromagnetics, optics, and circuits.
- 4 quarters of EE TA: digital image processing
- 1 quarter as EE instructor: digital image processing

NC State University, Raleigh, NC 2011-2014
- 8 semesters of Physics TA

WORKING EXPERIENCE

Machine Learning Software Engineer Intern, Facebook, Seattle Jun. 2018-
Aug. 2018