ESTIMATING MOTION OF OBJECT CONTOURS FROM DISTANCE TRANSFORMS

by

Kyle Soeder
A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Science

Committee:

_____	Dr. Zoran Durić, Thesis Director
_____	Dr. Jana Košecká, Committee Member
_____	Dr. Yotam Gingold, Committee Member
_____	Dr. Sanjeev Setia, Chairman, Department of Computer Science
_____	Dr. Kenneth Ball, Dean Volgenau School of Engineering
Date: _____	Spring Semester 2015 George Mason University Fairfax, VA

Estimating Motion of Object Contours from Distance Transforms

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Kyle Soeder
Bachelor of Science
Loras College, 2011

Director: Dr. Zoran Durić, Associate Professor
Department of Computer Science

Spring Semester 2015
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to my family

Acknowledgments

I would like to thank the three people who helped make this thesis possible. My advisor Dr. Zoran Durić provided advice, encouragement, and the support I needed to complete this thesis. I would also like to to thank Sam Gelman and Nalini Vishnoi for providing the preprocessing software and the Kinnect sequences of human movement. Sam also provided the segmentation of the gait sequence.

Table of Contents

List of Figures

Abstract

ESTIMATING MOTION OF OBJECT CONTOURS FROM DISTANCE TRANSFORMS

Kyle Soeder, M.S.

George Mason University, 2015

Thesis Director: Dr. Zoran Durić

plain This thesis develops a method for estimating motion for object contours in depth video sequences. First the depth image is thresholded to find the foreground object. Contours are then extracted from the thresholded image. Given the contours in two frames my method estimates the motion that transforms the contour in frame 1 to the contour in frame 2. First the Euclidean Distance Tansform (DT) is computed for the contour in frame 2. The DT and its gradient at each of the points along the contour in frame 1 are used to estimate the motion between the contours in the frames. I consider the obtained motion an estimate of the normal flow between the frames. Finally the normal flow is used to estimate affine motion between the contours. I demonstrate the method on several synthetic images corresponding to contours of various shapes. Finally I demonstrate the effectiveness of my method on several image sequences including household objects, a person raising their leg and a long sequence of a human walking.

Chapter 1: Introduction

Motion between images has traditionally been estimated using image derivatives or feature matching ([1]). I am interested in calculating the motion between contours in objects in depth images, such as images collected from a Kinnect depth camera. In this thesis my method for calculating the motion will utilize distance transforms and their gradients as opposed to image derivatives and feature matching.

The motivation for this work was Chamfer Matching, where the distance transform was used to determine closely matched contours. However, Chamfer Matching can be computationally expensive since it requires computing many correlations between the contour and the distance transform of a template. For a perfect match one would expect to get a low value for resulting correlation between the contours and the distance transform. If the match is not perfect, the matching score could be high even if the contours are very similar and just slightly displaced.

It can be observed that if one computes the gradients of the distance transform, those gradients point in the direction normal to the contour. By combining unit vectors obtained from those gradients and the negative values of the DT, one can obtain vectors that correspond to the normal displacement between pairs of contours.In addition, if it is assumed the displacement between two contours corresponds to rigid motion, one can compute a parametric motion model, such as Affine motion between those contours. In this thesis I have developed this method for calculating motion models in python using OpenCV libraries.

I applied this method first to a few household objects being moved across a flat surface in a depth video. I thresholded the images, extracted the contours and then applied the normal flow calculation to the objects. With the normal flow I then applied the calculation to obtain the motion model for each of the objects. This experiment is described in further

detail in Section 4.1.

The second application of this method was applied to a human raising their leg off the ground while standing. For this work I obtained depth images from Sam Gelman that had the background removed already. Applying the same thresholding I again extracted the contours from the sequence of images. Then I calculated the normal flow for two frames apart, since the motion between single frames was too small to provide useful results. I followed the same method for motion calculation after obtaining the normal flows. This experiment is outlined in further detail in Section 4.2.

The final application of this method was performed on a sequence of a person walking. For this sequence the images were segmented first into 4 clusters using a K-Means clustering based on the depth of the pixels. Each cluster was then compared to the cluster of the same depth in the pairs of images. Each segment was converted to a binary image and had the contours extracted. The fact that there were several depths caused occlusion to occur, these occluding contours were removed using a distance calculation against the closer contours in the image. Once the occluded contours were removed, normal flow and affine motion was calculated. This final experiment is discussed in further detail in Section 4.3.

Chapter 2: Related Work

There are many different forms of distance transforms that have been established and used in various algorithms for image matching and image feature computations. Rosenfeld in ([2]) discusses several different distance transforms that can be computed including City Block, Square, Hexagonal, Octagonal and Euclidean distances. For my thesis I observed all of these different forms of distance transforms prior to deciding to use Euclidean distance as my distance transform. Euclidean distance requires lots of computations to calculate, however there have been several efforts involved in speeding up the process of computing Euclidean Distance.

Felzenszwalb in ([3]) discusses one method for speeding up the calculation of Euclidean distance. In this effort Felzenszwalb introduces a linear time algorithm for the calculation of Euclidean squared distance. The process followed to perform this calculation is a series of minimizations described in 3. For my work I am using Euclidean distance since it has been efficiently implemented in several libraries, such as OpenCV which is what I used for my calculations.

With more efficient ways to calculate distance transforms, there have been many efforts using the transforms for different matching algorithms. In ([4]) Borgefors discusses a technique using distance transforms and minimization of distances to match images. Since matching is computationally expensive at very high resolutions, Borgefors uses a series of matches starting at low resolution, eliminating irrelevant features in the image, incrementally increasing resolution to high resolutions. The large amount of calculations required for high resolution is reduced since the low resolution matching eliminated all the irrelevant features and calculations were not performed on those features at the high resolution.

Another application of distance function occurs in ([5]). In this effort Huttenlocher

uses Hausdorf distance to compare images. This is of interest to me as both Hausdorf and Chamfer matching are computationally expensive and I would like to improve the efficiency of image matching. This effort discusses how Hausdorf matching requires a matching to be performed against a model on an image, moving the model across the image collecting the matching score for every possible position of the model on the image. As is evident the effort of moving the model over the image and recalculating the match at each location is very computationally expensive. My method eliminates the need for many of those comparisons.

Derivatives of distance functions are also useful for comparisons of images. In ([6]) Fitzgibbon describes the use of the gradients of distance functions to register point sets and in turn match point sets to models of objects, an example in the paper is matching point sets to specific letters in images. This was of interest to me because I am also using gradients to determine direction of motion, although I am not using gradient descent, I am simply doing one gradient calculation for direction.

One of the motivations for this effort was to speed up the process of Chamfer Matching. Several other proposals have been made to speed up this process. Gavrila in ([7]) proposed a method for grouping similar templates together to speed up Chamfer Matching by reducing the number of comparisons needing to be made against the different templates. The templates are also grouped at different resolutions, so the higher the resolution the fewer the number of templates needed to be matched to the image.

Another hierarchical approach to speeding the Chamfer Matching process is discussed in ([8]). Svensson in this effort does not use the binarisation process that several other efforts have implemented, and instead uses distance weighted gradient magnitude in an effort to keep as much information as possible about the image throughout processing. This proposal increased the fidelity of the results while still keeping the computations at a minimum through the use of the hierarchical comparisons that occur.

One other important effort that I wanted to discuss was ([9]). This effort takes into account distance transforms and directional integral images to perform more accurate and more efficient matches. This effort takes into account more directions then my own, I only

take into account lateral movement, but this effort could be used in the future to further my own work by taking into account more directions than just the lateral directions.

The last related work I wanted to discuss was ([10]). The motion models and normal flow calculations in this research are the basis for my work. I use the normal flow calculation described in this research, as well as the parameter estimation to give my final affine motion model once I have calculated the normal flows.

Chapter 3: Technical Background

The first key component to the calculation of both the motion models and performing the matching in this work is the distance transform. A distance transform of a binary image specifies the distance of each image pixel to the nearest non-zero pixel. Distance transforms have been used extensively in computer vision, image processing, and pattern recognition.

We will use a definition of distance transform given in [3, 11, 12]. Let $\mathcal{G}$ be a regular grid and $P \subseteq \mathcal{G}$ a set of points on the grid. The distance transform associates to each grid location the distance to the nearest point in P,

$$\mathcal{D}_P(p) = \min_{q \in \mathcal{G}} (d(p,q) + \mathbf{1}(q)),$$

where $\mathbf{1}(q)$ is an indicator function for membership in P,

$$\mathbf{1}(q) = \begin{cases} 0 & \text{if } \ q \in P \\ \infty & \text{otherwise} \end{cases}$$

and $d(p,q)$ is the Euclidean distance, i.e. $d(p,q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$. The algorithm for computing this distance transform is shown in Algorithm 1.

A hierarchy of parametric flow models has been proposed including pure translation, image rotation, 2D affine flow, and 2D homography (6-parameter or simplified quadratic flow). We will consider all those models here. 6-parameter flow corresponds to the instantaneous projected image motion field generated by a moving plane. Other models used here can be obtained by setting some of the eight parameters to zero. In the 6-parameter model

Algorithm 1 Algorithm 1: The distance transform algorithm for the the squared Euclidean distance in one-dimension.

```
1: k ← 0                    (* Index of rightmost parabola in lower envelope *)
2: v[0] ← 0                 (* Locations of parabolas in lower envelope *)
3: z[0] ← −∞                (* Locations of boundaries between parabolas *)
4: z[1] ← +∞
5: for q = 1 to n − 1                (* Compute lower envelope *)
6:     s ← ((f(q) + q^2) − (f(v[k]) + v[k]^2))/(2q − 2v[k])
7:     if s ≤ z[k]
8:        then k ← k - 1
9:             goto 6
10:    else k ← k + 1
11:    v[k] ← q
12:    z[k] ← s
13:    z[k + 1] ← +∞
14: k ← 0
15: for q = 0 to n - 1 (Fill in values of distance transform)
16: while z[k + 1] < q
17: k ← k + 1
18: D_f(q) ← (q − v[k])^2 + f(v[k])
```

coordinates of a point (x, y) in the first frame will move to (x', y') in the next frame:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} w_1 \\ w_4 \end{pmatrix} + \begin{pmatrix} w_2 & w_3 \\ w_5 & w_6 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.1}$$

Eq. (3.2) relates corresponding points in successive image frames. To obtain the displacement $\vec{u}(x, y) = (\delta x \;\; \delta y)^T$ of (x, y) we subtract $(x \;\; y)^T$ from both sides of (3.2) to obtain

$$\begin{pmatrix} \delta x \\ \delta y \end{pmatrix} = \begin{pmatrix} x' - x \\ y' - y \end{pmatrix} = \begin{pmatrix} w_1 \\ w_4 \end{pmatrix} + \begin{pmatrix} w_2^1 & w_3 \\ w_5 & w_6^1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{3.2}$$

where $w_2^1 = w_2 - 1$ and $w_6^1 = w_6 - 1$. The normal displacement field at (x, y) is given by $u_n(x, y) = \delta\vec{r}_n \cdot \vec{n} = n_x\delta x + n_y\delta y = w_1 n_x + w_2^1 x n_x + w_3 y n_x + w_4 n_y + w_5 x n_y + w_6^1 y n_y = \mathbf{w} \cdot \mathbf{p}$, where $\vec{n} = n_x\vec{\imath} + n_y\vec{\jmath}$ is the gradient direction, $\mathbf{p} = (n_x \;\; x n_x \;\; y n_x \;\; n_y \;\; x n_y \;\; y n_y)^T$, and

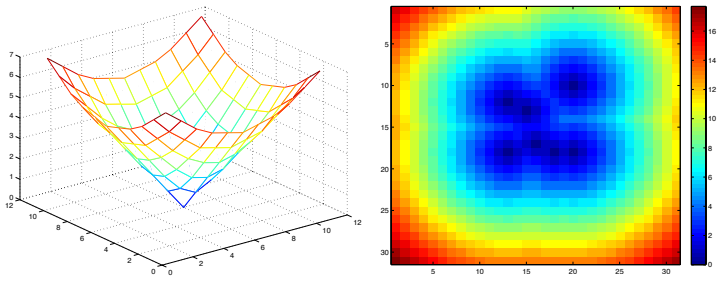

Figure 3.1: Distance transform computed with respect to a single binary point on image (left). The point is the smallest value in the chart. Distance transform performed on a point cloud (right).

$\mathbf{w} = (w_1 \;\; w_2^1 \;\; w_3 \;\; w_4 \;\; w_5 \;\; w_6^1)^T$ is the vector of affine parameters.

We seek the affine model $\mathbf{w}$ that minimizes $\|\mathbf{e}\| = \|\mathbf{b} - P\mathbf{w}\|$; the solution satisfies the system $P^T Pw = P^T \mathbf{b}$ and corresponds to the linear least squares (LS) solution. We will describe the use of this motion model in the sections to follow.

3.1 Computing Flow from Distance Transform

Utilizing the distance trasnform, along with the motion models, I was able to find the affine motion model between two frames. In order to do so I utilized the gradients to find normal flows and then performed the aforementioned calculation as follows. It can be seen that the distance transform $\mathcal{D}_P(p)$ is a smooth function on $\mathcal{G}$ and therefore it has a gradient at each point $p \in \mathcal{G}$. Fig. 3.3 shows the distance transform computed for a small binary image and its gradients. It can be seen that the gradients point away from the contour and that they are normal to it. Fig. 3.4 shows examples of image flow computed from a real image sequence using the proposed method.

The algorithms for computing $\mathcal{D}_P(p)$ can be modified to compute the nearest point $q \in P$ for each point $p \in \mathcal{G}$. However, it can be observed that the nearest point q of any

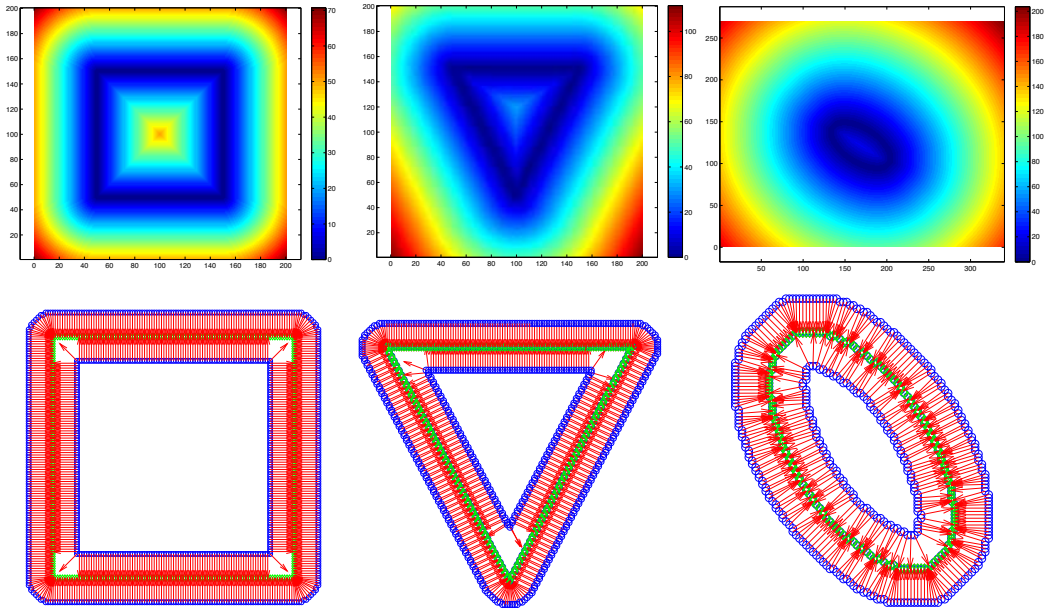

Figure 3.2: Distance transform and examples of displacements computed for three simple shapes: square, triangle, ellipse. Distance transforms are shown in the top row. The bottom row shows displacements computed from blue contours with respect to green contours. The blue contours are 10 ± 0.5 pixels away from the green contour (original binary pattern).

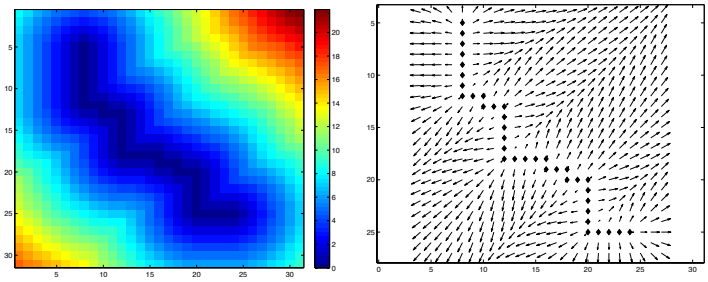

Figure 3.3: Distance transform computed with respect to a binary contour (left) and its gradients (right). The contour is shown in dark blue, $\mathcal{D}_P(p) = 0$, and all gradients are normalized to 1.

point p is in the direction of negative gradient

$$\nabla \mathcal{D}_P(p) = \frac{\partial \mathcal{D}_P(p)}{\partial x}\vec{i} + \frac{\partial \mathcal{D}_P(p)}{\partial y}\vec{j}$$

and its distance is $\mathcal{D}_P(p)$, therefore we can write

$$q \approx p - \mathcal{D}_P(p)\frac{\nabla \mathcal{D}_P(p)}{\|\nabla \mathcal{D}_P(p)\|} = p - \vec{v}_P(p), \tag{3.3}$$

where $\vec{v}_P(p)$ can be treated as a(n) (approximate) *displacement* of p with respect to some $q \in P$. Similarly, given a binary pattern R we can compute the distance function $\mathcal{D}_R(p)$ and define the displacement $\vec{v}_R(p)$ with respect to R.

Given the displacement calculated from the distance tranform and it's gradients, the flow between two contours can also be calculated. Normal flow is the projection of image motion (optical flow) onto the edge gradient direction [13]. It is usually computed from image derivatives resulting in very noisy measurements. In addition, since it corresponds to edge motion in the normal direction only it gives rise to the *aperture problem*—i.e., in

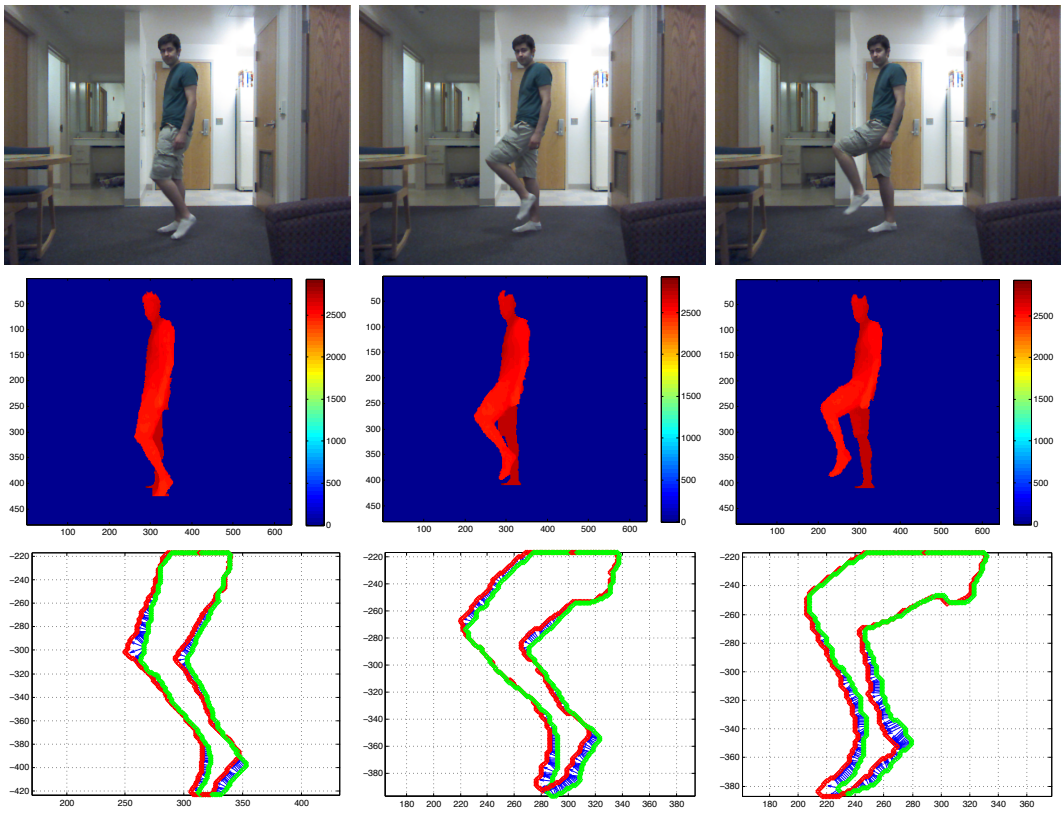

Figure 3.4: Computing flow from distance transform on a sequence of a moving leg. Top row: Color frames 0,10,20 from an RGBD video of a moving leg. Middle row: Depth frames 0,10,20 from the same sequence. Note that the background subtraction was used to eliminate background pixels. Bottom row: The image flow computed for the moving leg using the distance transform and its gradient. Green pixels belong to the current frame, the red pixels belong to the next (reference) frame. The leg was segmented using depth and anthropomorphic measures.

a small region along a straight edge it does not contain any information about tangential motion of the edge. This problem is usually solved by assuming that the motion in a sufficiently large region, that includes edges of varying orientations, obeys some simple model so that the information over the whole region can be used to recover the missing information. The method used here estimates the normal flow from pairs of successive color image frames without image derivatives [14]. The normal flow computed from images is shown in Fig. 3.5.

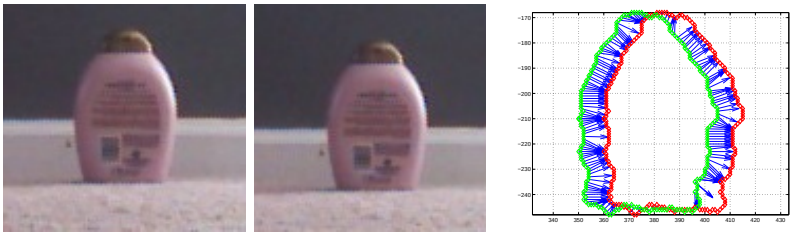

Figure 3.5: Normal flow computed from pairs of frames of a bottle.

Chapter 4: Experiments

I performed several different experiments to validate the usefulness of the methods described above. The first experiment is focused on common household items being moved across a flat surface. The second and third experiments focused on two different sequences of human movements.

4.1 Experiment 1: Depth Videos of Common Household Items

Using a Microsoft Kinnect depth camera, I took 30 frames of different household objects at a distance of approximately 2 meters from the camera. The object was moved about a centimeter per frame. The resulting images are depth images from the Kinnect that describe the depth of each pixel in the frame. I have provided examples of all the different objects in 2 separate frames as well as a single color frame in Fig. 4.1.

Using these depth images, I thresholded the image to obtain the objects in a binary image format. From the binary image I extracted the boundaries. The extracted boundaries for two sample frames of the bottle are shown in Fig. 4.2 (resulting boundary images). The implementation of this boundary finding process is shown below.

```
Thresholding Process:
        Load the image
        Set all values greater than max depth threshold to zero
        Set all values less than min depth threshold to zero
        Set all values within threshold to pixel value of 244
Contour Finding Process:
        Run Canny image detection
        Use OpenCV find contours algorithm to return the contours
```

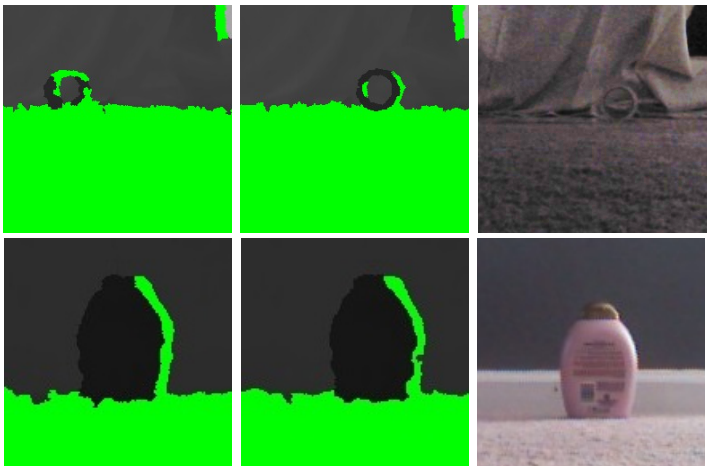

Figure 4.1: Sample depth and color images of common household objects (roll of tape top, conditioner bottle bottom)

The next step applies the flow calculation formula described in section 3. The implementation in Python is shown in (code for python implementation). The resulting flow from frames in Fig. 4.2 are shown in Fig. 4.3. The methods described in section 3 are used to estimate the motion models between frames for the bottle. The resulting motion models, flows and residuals are given in Fig. 4.3.

4.2 Experiment 2: Depth Videos of Human Raising Leg

Using a Microsoft Kinnect depth camera, two sequences of 20 frames were captured of a person raising their left leg. The images are the side profile of the person and were provided by Sam Gelman. The person was stationary and approximately 4 meters from the camera. The person raised their left leg, bending at the knee moving the leg approximately 2 centimeters per frame. The resulting images are depth images from the Kinnect that describe the depth of each pixel in the frame. I have provided images of the sequences in

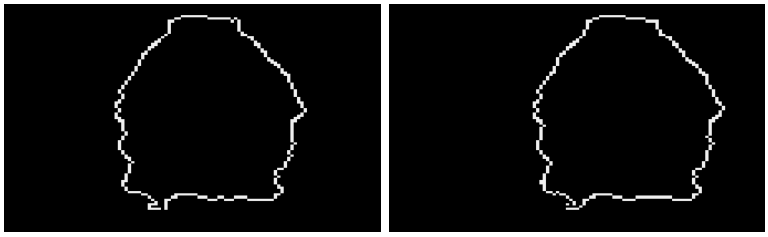

Figure 4.2: Contours of bottle after thresholding.

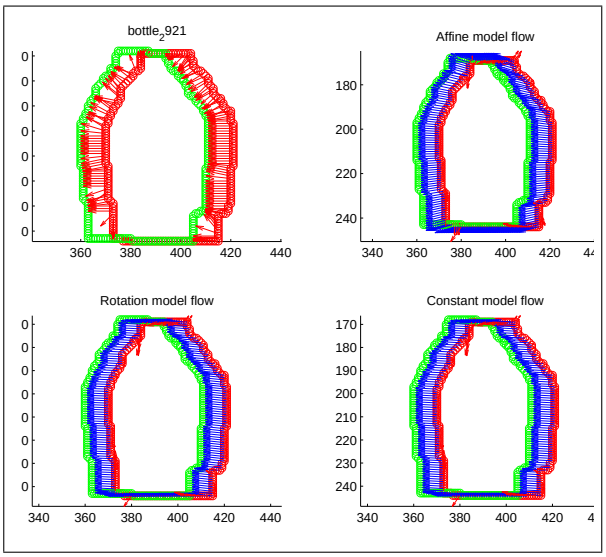

Figure 4.3: Resulting motion models and residuals for bottle.

Fig. 4.4.

Using these depth images, I thresholded the image to obtain the image of the body in a binary image format. From the binary image I extracted the boundaries. For my next step I applied the same flow calculation formula described in experiment 1. The resulting flow from the sequence in Fig. 4.4 is shown in Fig.(4.5, 4.6, and 4.7). The methods described in section 3 are used to estimate the motion models between frames for the leg raising. The resulting flows are given in Fig. 4.5- 4.7.

4.3 Experiment 3: Segmented Depth Videos of Human Walking

Using a Microsoft Kinnect depth camera, a sequence of 55 frames, approximately 2 seconds of RGBD images of walking, was captured of a person walking in front of the camera. The images are the side profile of the person walking. The distance from the camera approximately 4 meters away. The images were then segmented using a K-Means clustering to cluster parts of the image based on the depth of each pixel in the image. The resulting segments are as follows, segment 1 is the left arm and is blue in the depth images, segment two is the left leg and the body it is green in the depth images, segment 3 is the right leg and is orange in the depth images, lastly depth 4 is the right arm and is red in the depth images. The segmentation is used to generate the contours much like the previous experiments except that this time the specific depth (1,2,3 or 4) is used to create the binary image, where all the other values except the specific depth being tested is 0 and the depth being tested is set to 1. I have provided the depth images of the sequence in Fig. 4.8 and Fig. 4.9.

One of the issues that I had to resolve in these frames was the occlusion that occurs when looking at depths 2 through 4 in the image. Since there were up to four segments in each frame, I had to work through occlusion at the deepest 3 levels. The first depth didn't have any occlusion so I followed the same process to get the flows and motion as was followed

Figure 4.4: Segmented depth images of person raising leg.

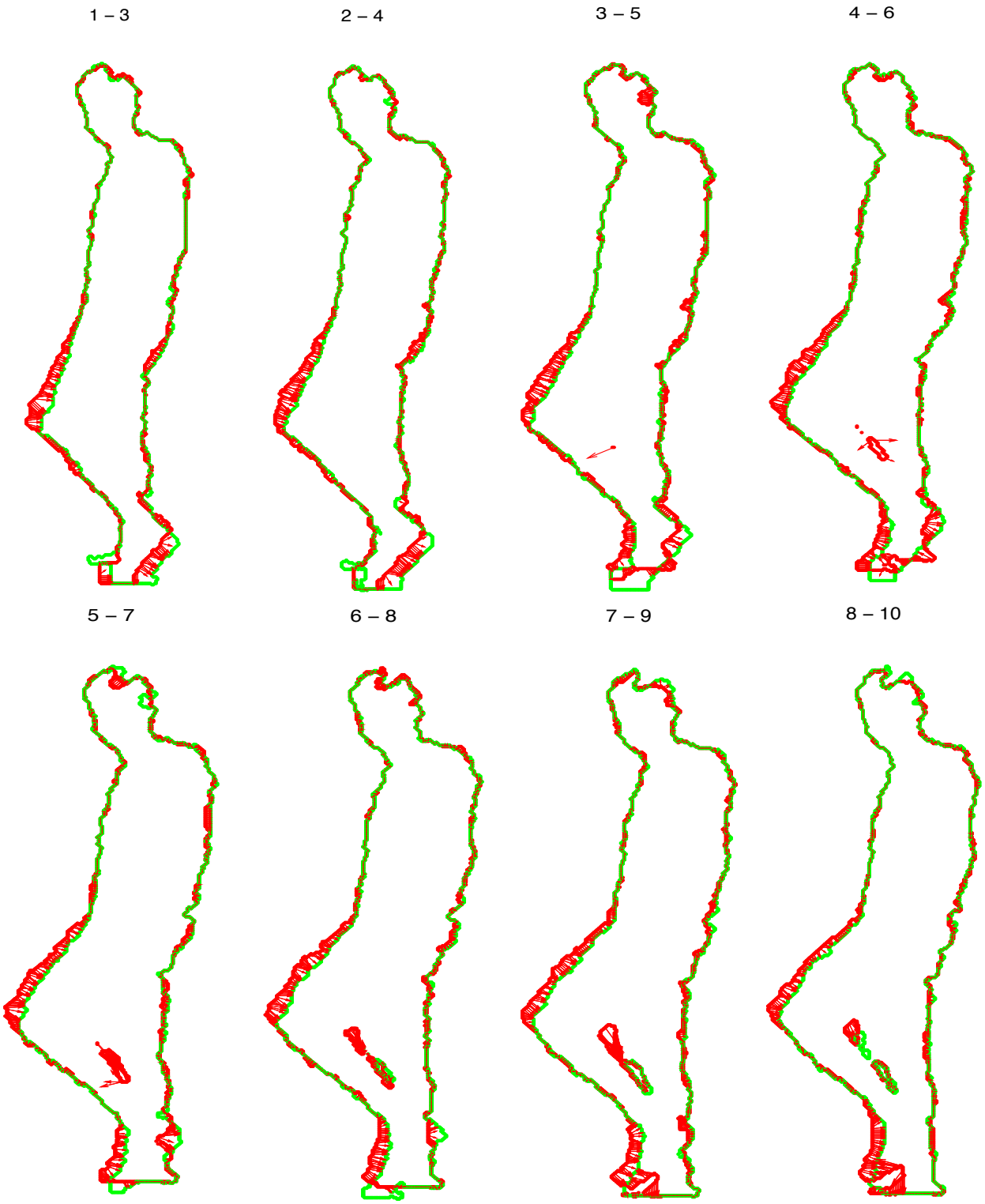

Figure 4.5: Resulting flows of person raising leg. Frames 1-9

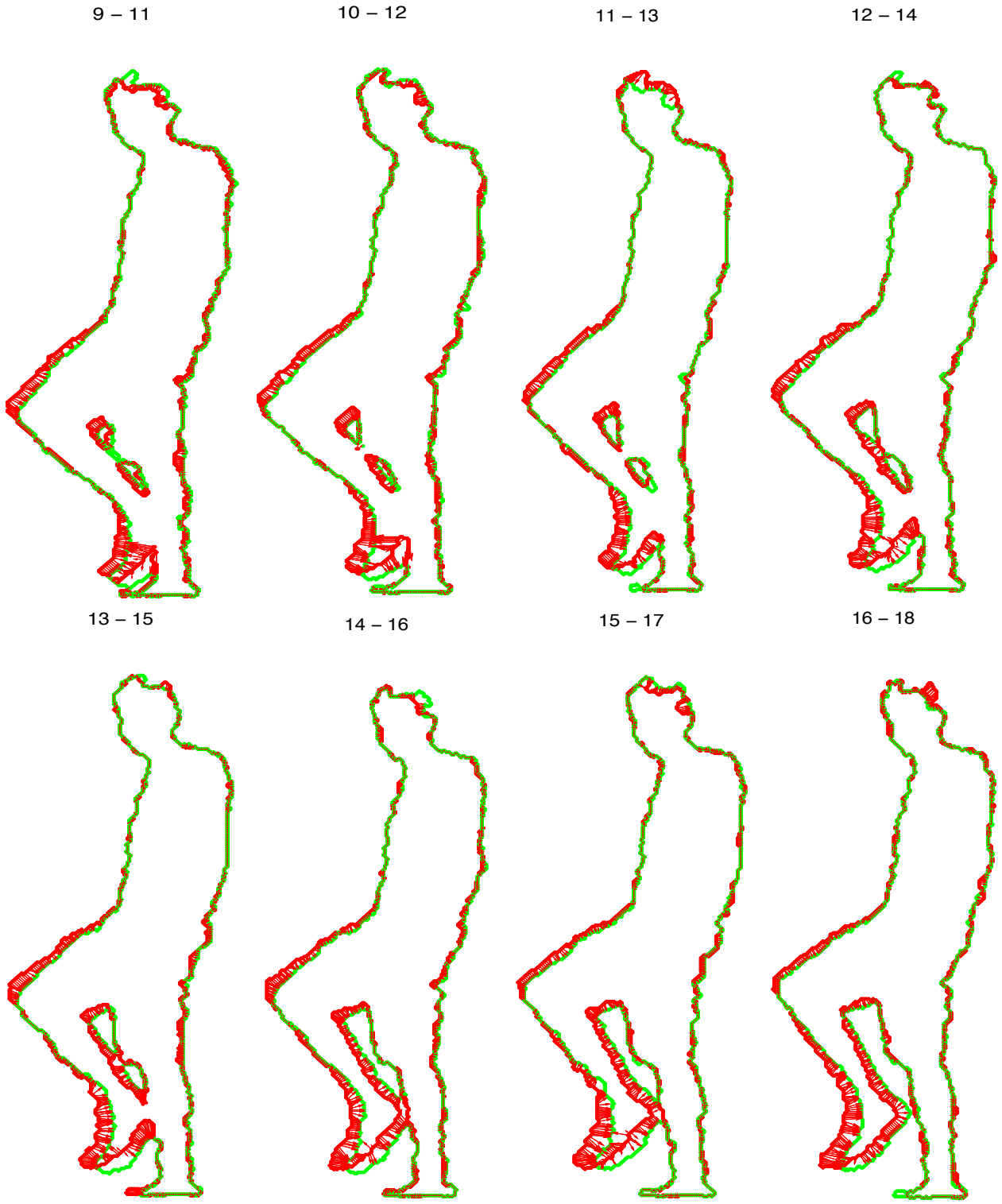

Figure 4.6: Resulting flows of person raising leg. Frames 9-17

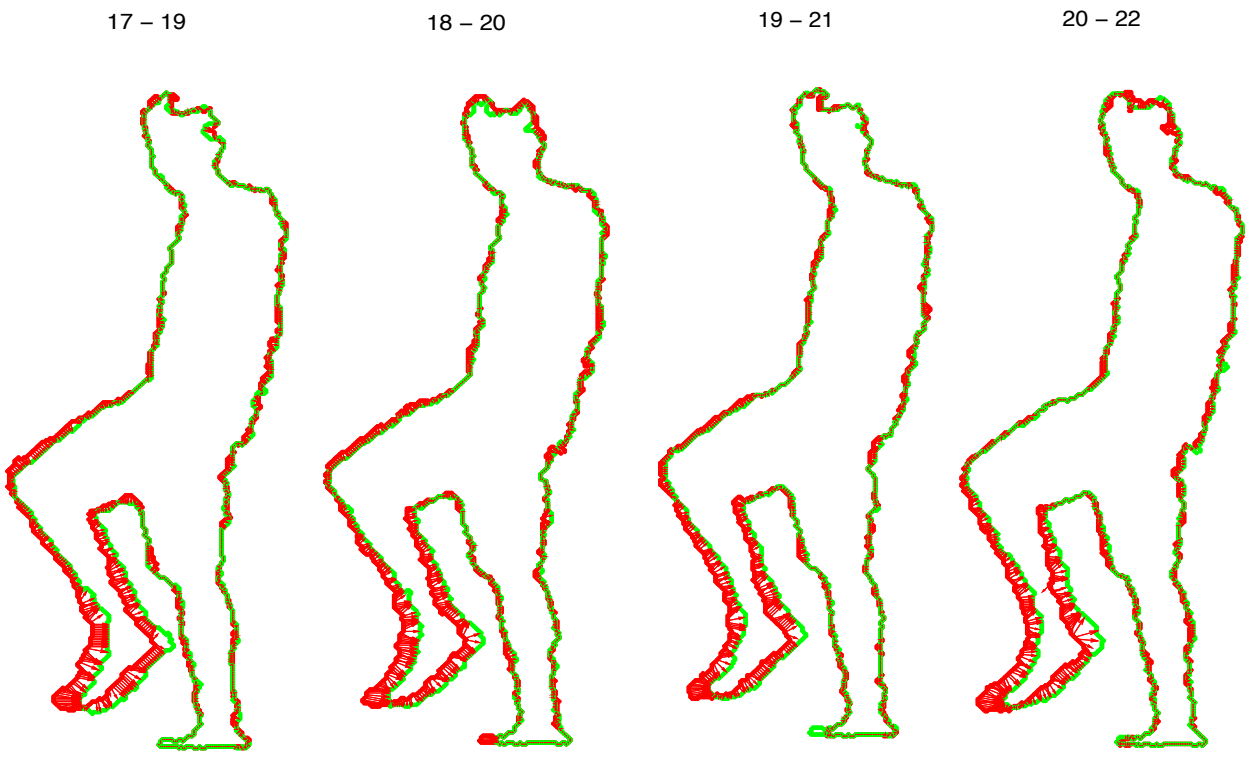

Figure 4.7: Resulting flows of person raising leg. Frames 17-21

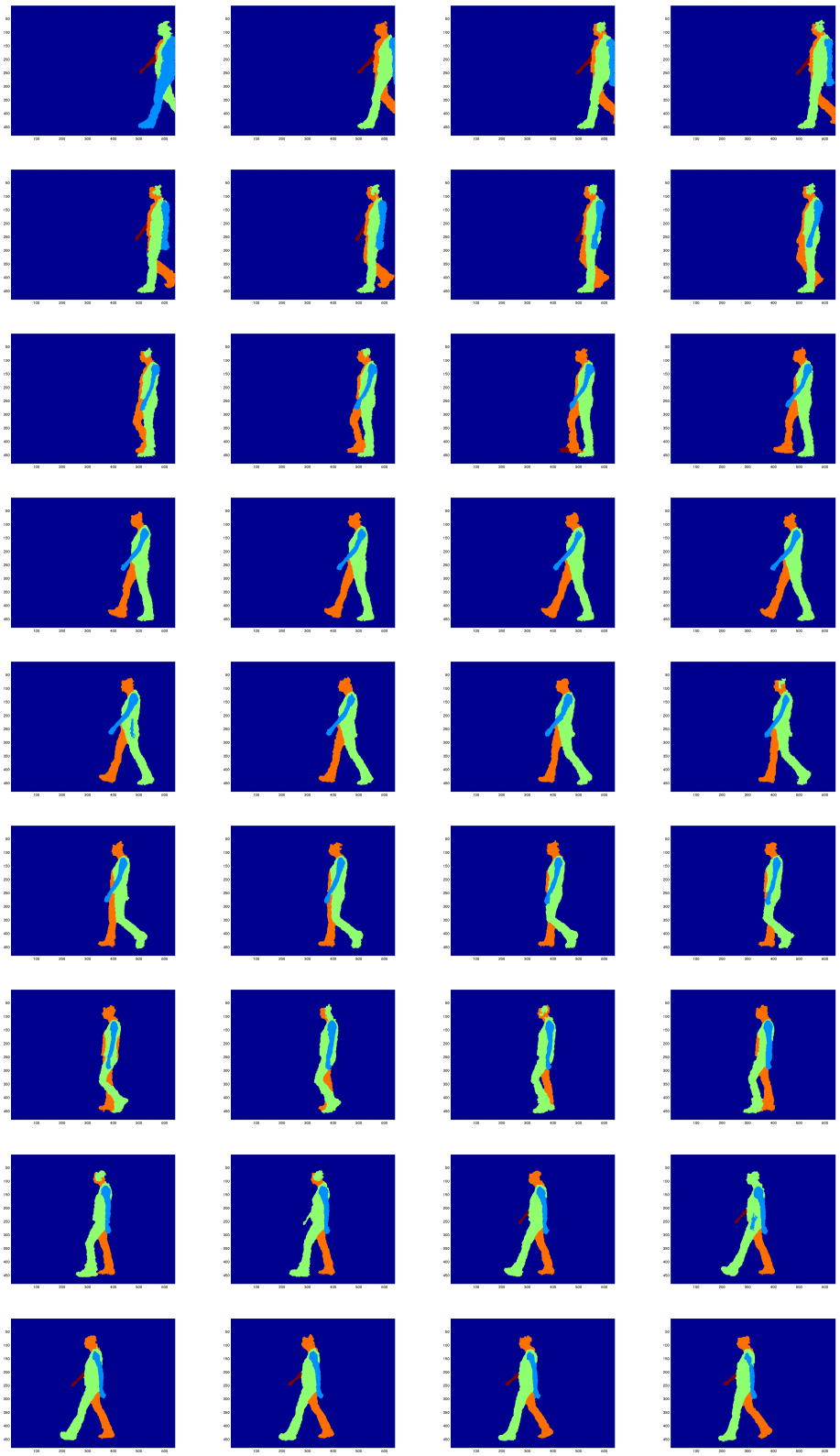

Figure 4.8: Sample depth images of person walking. (Frames 25-60)

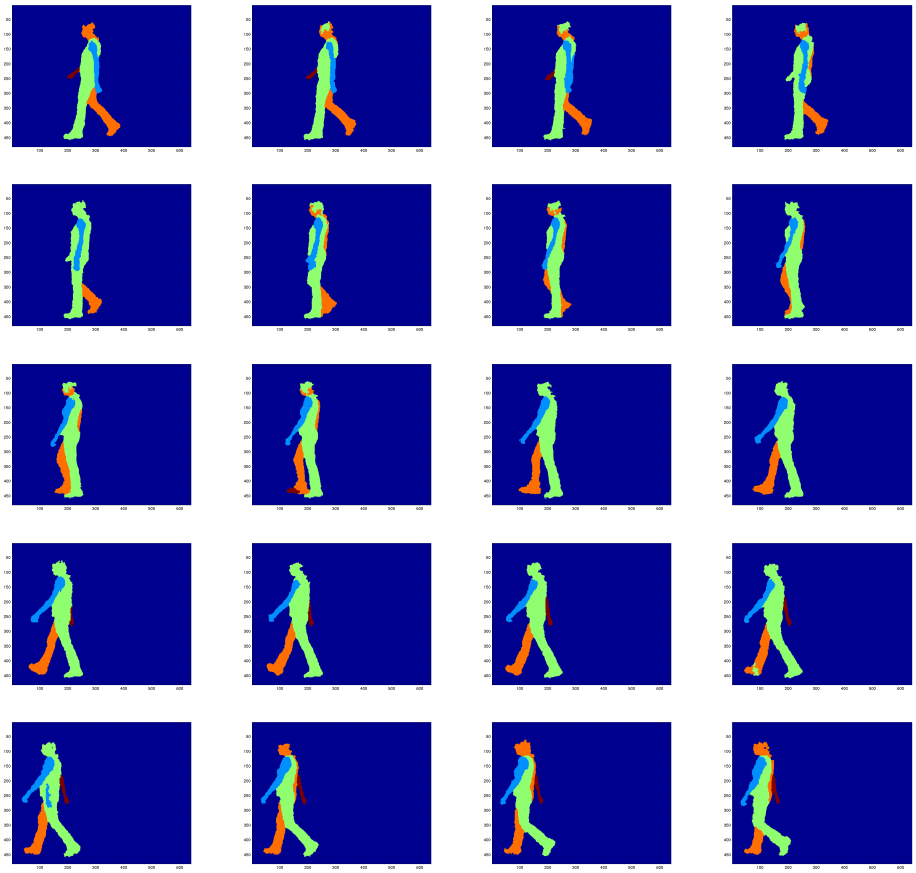

Figure 4.9: Sample depth images of person walking.(Frames 61-80)

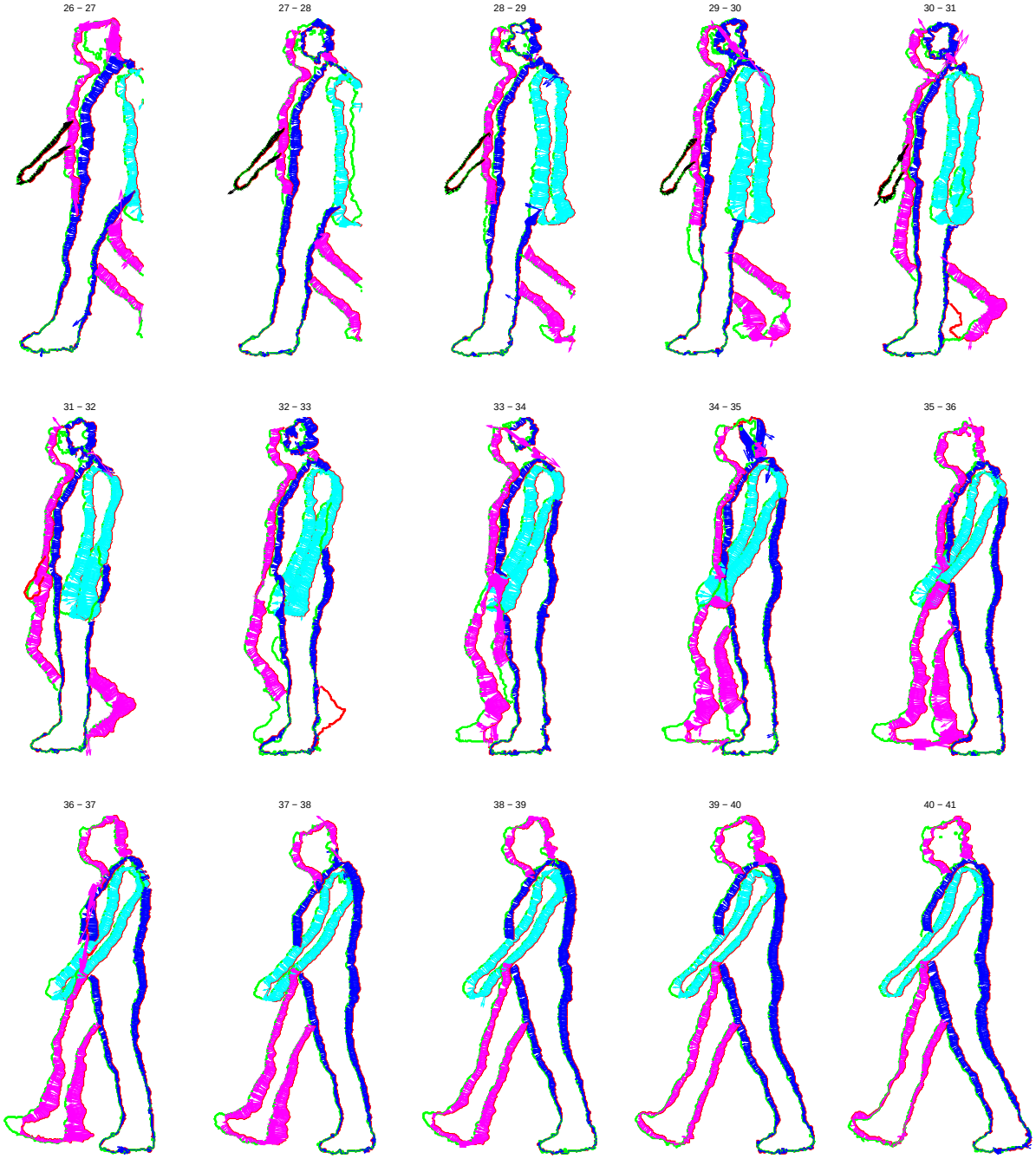

Figure 4.10: Computed flows for frames 26-41 of segmented depth sequence of person walking

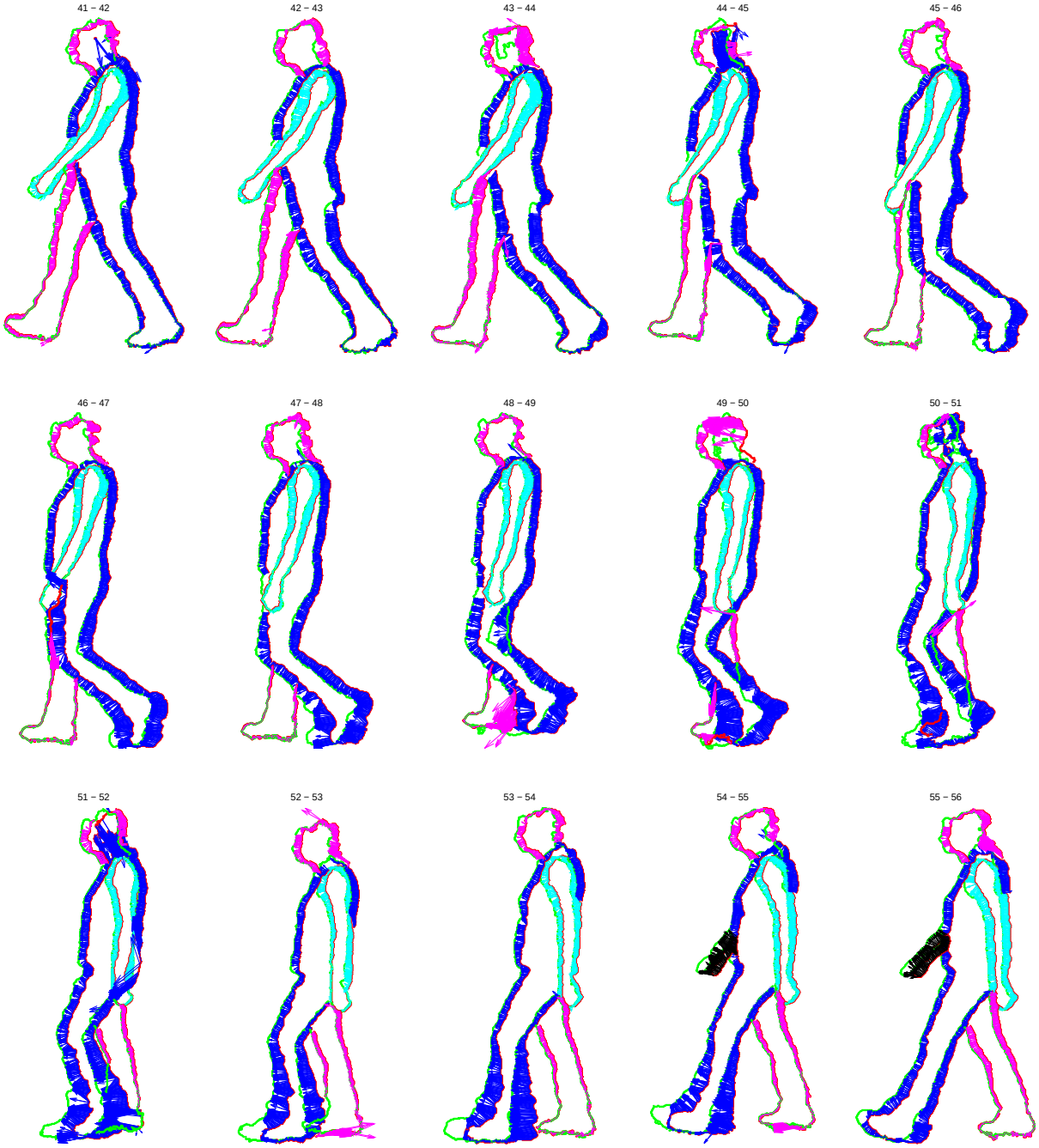

Figure 4.11: Computed flows for frames 41-56 of segmented depth sequence of person walking.

for the first 2 experiments. For the second depth and deeper I had to take into account the front contours and remove any occluded contours created by the closer contours. For example, if the front arm covered part of the front of the body, that would get included as a contour if I didn't remove the contour created from the overlap. By applying a distance transform to the front contour and removing all the contour points in the second depth that were within 3 pixels of the front depth I excluded any occluded contours and calculate motion on the specific contours for each segment. I followed this same process for depths 3 and 4 excluding any contours that were within 3 pixels to any of the front contours. Once all the contours were accurately defined and the occluded contours were removed, I ran into an issue when motion grew beyond the width of the arms and legs in the frames. To resolve this issue, I split the contours into left and right sides, and compared only the left contours in frame 1 to the left contours in frame 2 and the same for the right contours. To perform this split, I did a search to find the first non-zero pixel on the left side of the frame and the first non-zero pixel on the right side of the frame for each row in the frame. Then I took a distance transform from those pixels and sorted the contour points based on their proximity to those non-zero pixels. The flows were then calculated according to the correct sides of the contours being compared to one another. The resulting flows are show in Fig. 4.10 - Fig. 4.13.

Once the flows were calculated for each image, I applied the same calculation to determine the six parameter affine motion model as was previously used in Experiment 1 and 2. A few examples of the motion models are shown in Fig. 4.14 The motion for the front arm is cyan, the front leg is blue, the back leg is magenta, and the back arm is black.

4.4 Discussion

The primary goal of these experiments was to demonstrate the application of the discussed method to estimate the motion of several different objects. In the first experiment I demonstrated how motion of household objects which have rigid contours can be obtained using the normal flow calculation. There were some minor issues with this experiment as the

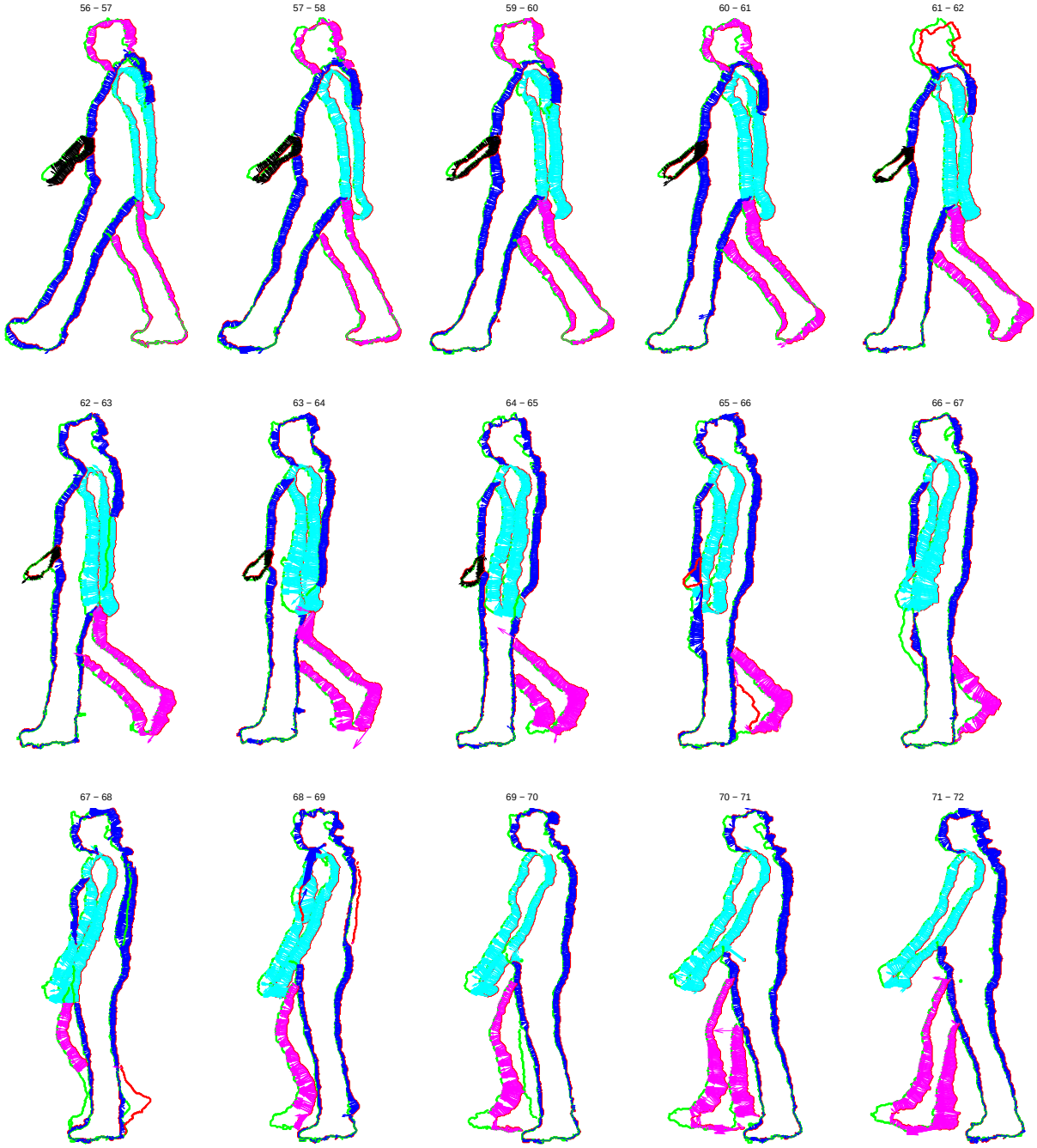

Figure 4.12: Computed flows for frames 56-72 of segmented depth sequence of person walking

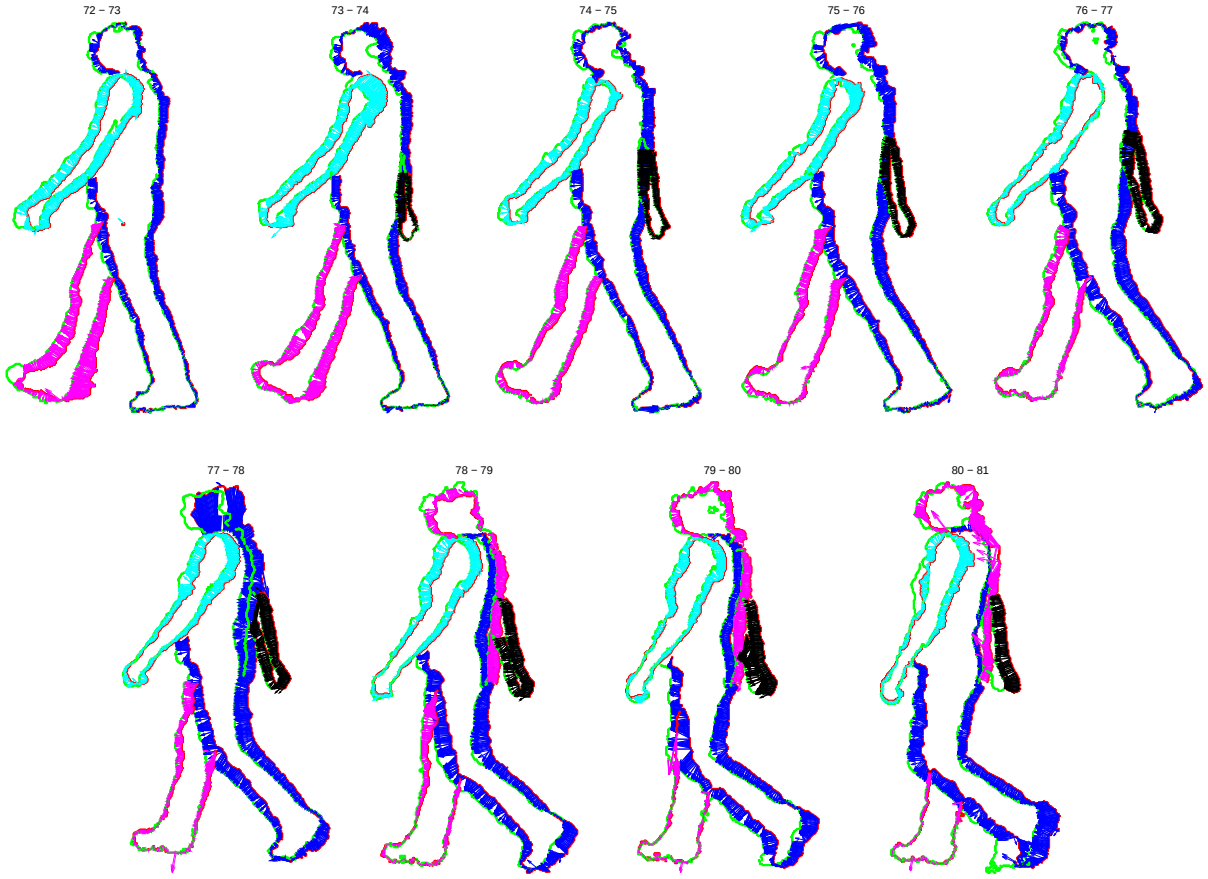

Figure 4.13: Computed flows for frames 72-81. Frames 25-80 are corresponding to approximately 2 seconds of RGBD images of walking.

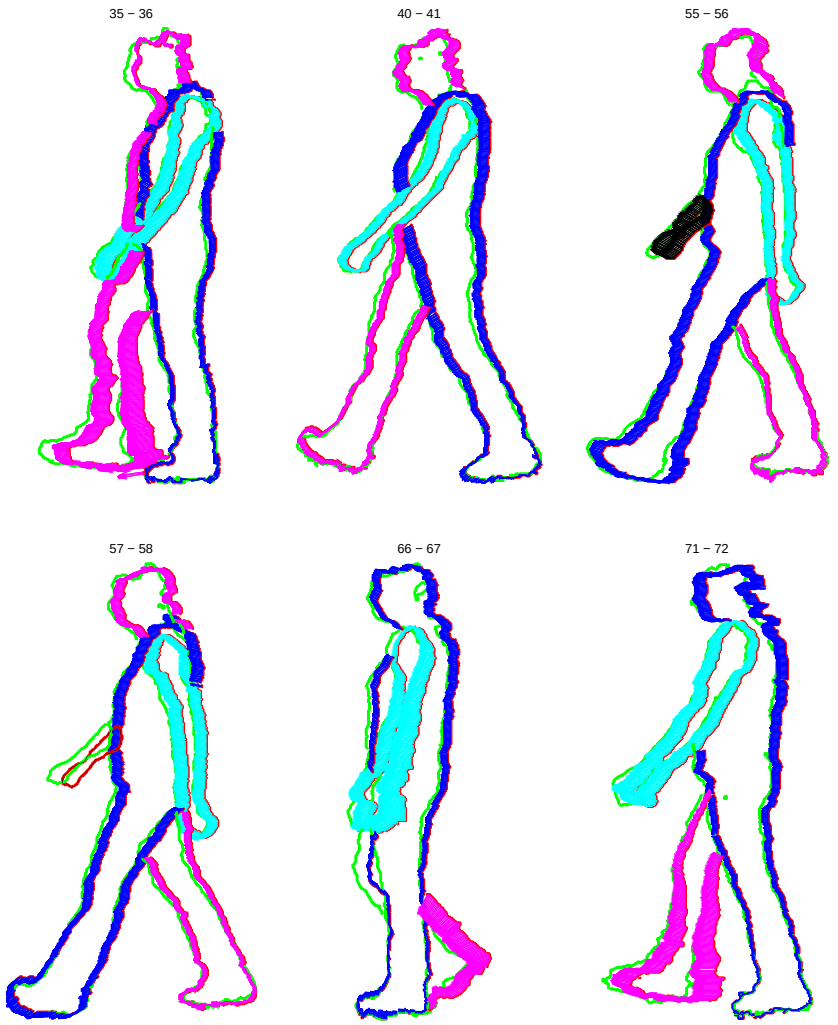

Figure 4.14: Computed affine motion for several frames of human walking.

Kinnect camera has a low resolution. So these results could be improved upon with a better resolution camera. The results did should clear and accurate motion estimation for the objects.

The second experiment was used to demonstrate the validity of this method on a non-rigid object, a human, moving. This sequence of frames shows a person raising his left leg while not moving laterally at all. This concentration on only one part of the body moving allowed me to test my method against a relatively simple motion. The difficulty in this sequence was that the leg contour changed significantly as the gap between the front leg and back leg developed, creating a frame that didn't exist in one frame that did exist in another. This was only visible in a couple frames and the end result didn't cause major anomalies in the calculated flows. The results could be improved if segmentation was performed on the different parts of the leg and foot that were rotating and moving in different directions. For example the foot is moving backwards at some points when the leg is moving up, that causes some discrepancy when the final motion model is calculated.

The last experiment was the most effective in demonstrating the method on non-rigid, as well as multi-directional movement. Since the frames were segmented prior to processing, I was able to perform flow calculations on specific parts of the body. These images demonstrate the capability of this method to be able to describe large motion. there were some frames in which the entire arm moved outside of the previous contour, and with my splitting of the contours I showed how well this method can describe larger motion. These results show clear, accurate motion of the different phases of gait. This method could be used potentially to create models for phases of gait estimation in the future.

Chapter 5: Conclusion and Future Work

Through these experiments I demonstrated my motion estimation method on several image sequences with varying complexities of motion. In doing so I demonstrated the viability of this method on rigid and non-rigid contours extracted from RGBD videos collected from a Kinnect depth camera. I also showed how one can use a Euclidean distance transform and its derivatives on an image to estimate motion of objects between frames. The results from the experiments have shown that this method is fast and reliable and can be used for fairly large displacement, especially when contour orientation is incorporated into the method.

Although the results were very positive, there are some improvements that could be made in the future. First off, the resolution of the camera taking the images needs to be improved upon, and the latest version of the Kinnect camera has resolved this issue, so all new image collection could use the new camera to collect better data. Between the new camera and some improved contour smoothing, these results could be improved upon. Another improvement that could be made is to take into account contour orientation for comparison, as I only split into left and right halves the contours could be split into more pieces and compared accordingly. Lastly I would suggest performing motion calculations on each segment in the sequence instead of just an overall motion calculation. Given these improvements, future efforts in this area could lead to even more accurate and efficient results.

Bibliography

Bibliography

[1] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[2] A. Rosenfeld and J. L. Pfaltz, "Distance functions on digital pictures," *Pattern recognition*, vol. 1, no. 1, pp. 33–61, 1968.

[3] P. Felzenszwalb and D. Huttenlocher, "Distance transforms of sampled functions," Cornell University, Tech. Rep., 2004.

[4] G. Borgefors, "Hierarchical chamfer matching: A parametric edge matching algorithm," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 10, no. 6, pp. 849–865, 1988.

[5] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge, "Comparing images using the hausdorff distance," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 15, no. 9, pp. 850–863, 1993.

[6] A. W. Fitzgibbon, "Robust registration of 2d and 3d point sets," *Image and Vision Computing*, vol. 21, no. 13, pp. 1145–1153, 2003.

[7] D. M. Gavrila and V. Philomin, "Real-time object detection for smart vehicles," in *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, vol. 1. IEEE, 1999, pp. 87–93.

[8] S. Svensson and I.-M. Sintorn, "Hierarchical chamfer matching based on propagation of gradient strengths," in *Discrete Geometry for Computer Imagery*. Springer, 2006, pp. 308–319.

[9] M.-Y. Liu, O. Tuzel, A. Veeraraghavan, and R. Chellappa, "Fast directional chamfer matching," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 1696–1703.

[10] H. Wechsler, Z. Duric, F. Li, and V. Cherkassky, "Motion estimation using statistical learning theory," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 4, pp. 466–478, 2004.

[11] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient belief propagation for early vision," *International journal of computer vision*, vol. 70, no. 1, pp. 41–54, 2006.

[12] ——, "Distance transforms of sampled functions." *Theory of computing*, vol. 8, no. 1, pp. 415–428, 2012.

[13] Y. Aloimonos and Z. Duric, "Estimating the heading direction using normal flow," *International Journal of Computer Vision*, vol. 13, no. 1, pp. 33–56, 1994.

[14] Z. Duric, F. Li, Y. Sun, and H. Wechsler, "Using normal flow for detection and tracking of limbs in color images," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, vol. 4. IEEE, 2002, pp. 268–271.

Curriculum Vitae

Kyle Soeder received his Bachelor of Science in Computer Science and Mathematics from Loras College in 2011. He worked as a Software engineer for a couple years before applying to the Masters of Science in Computer Science Program at George Mason University. He worked as a Software Engineer through the process of completing his Master of Science degree.