# CV FINAL REPORT

## Squatting tracking

DECEMBER 1, 2017

CORNELL UNIVERSITY

Group member:<Beitong Tian bt346><Jianhua Fan jf773><Qian Qiao qq39><Yanfei Xu yx427>

## Abstract

No known specific algorithm is developed to track human doing squat with a real and big barbell on. The existence of a large barbell and drastic movement like squatting can make human body in some frames quite different from normal human body like in figure 1. Thus, specifically modified methods are needed to solve this task. In the following parts we will introduce our fully developed tracking algorithms to track the squat movement from video sequences.


## Literature s

In [1], an approach to tracking the human motion of balancing on each foot was developed. In this research, the author first defined a hierarchical 2D human body model, which includes six major components: head, body and four limbs. Each of the four limbs can be further decomposed to include primary component (upper arms and legs) and secondary component (lower arms and legs), respectively. Each of these body model components is represented by a quadrangle and every component is connected to another one by a joint. By making use of inherent correlation between different components, the author designed a top-down updating framework and an adaptive algorithm with constraints of foreground region for efficient tracking of human body for balancing applications. For each component of human body, consecutive two frame's difference can be represented by angle change, then with the constraints of foreground region, Average Absolute Difference Image (AADI) was computed between current frame and the former frame. As the optimal angle change is consistent with the minimum of AADI, the adaptive algorithm iteratively computes the minimum AADI to achieve the optimal state. Experiment results showed that the average tracking time is 0.632s which is fast and each part of human body can be tracked accurately.

In [2], a system for automated human body tracking and modeling based on a monocular camera was developed. In this system, eleven joint points including head, shoulder, hip, elbows, knees, hands and feet are extracted separately to build a 2D human body model. The head is extracted by analyzing negative minimum curvature (NMC) points on a parameterized silhouette. The torso, along with its angle and size, is determined by integrating multiple frame information with connectivity constraint. Hands and feet can be identified correctly based on a modified star skeleton approach and the nearest-neighbor tracking mechanism. The rest of joint points can also be located by taking advantage of the connectivity constraints.

In [3], this paper explores foreground detection, tracking foreground, analysis technique procedures. First, foreground objects are segmented from the background by a four-stage process: thresholding, noise cleaning, morphological filters (boundary detection), and object detection. For the tracking process, the center of the detected foreground region in the sequences is stored in a trajectory map. For the motion analysis, it fuses features together into a fused motion analysis using a weighted averaging process. The ($\theta$, $\alpha$, $\beta$, acceleration variations on the centroid of the motion blob) are implemented for analysis. In addition, the speed of the bounding box surrounding of the silhouette detected could be used. They consider shadow to be a problem for the silhouette based motion identification and analysis, because the structure may be fluctuated by the shadow types.

In [4], the tracking of moving target can be divided as following stages: for target pre-processing stage, a moving target region is morphologically dilated (twice), eroded and then its border is extracted. The author use blob to track target. The position and velocity of each blob is determined from the last time step t last and used to predict a new image position at the current time. Targets that are "close enough" in cost space are considered to be potential matches.

## Introduction

Our project is based on squat movement, what we do is extracting angles between certain body parts from original 2D video of a person doing squat. Squat can be roughly described by two angles: the angle at knee between thigh and calf, the angle at hip between back and thigh. Thus, our mainly object is to track these two angles.

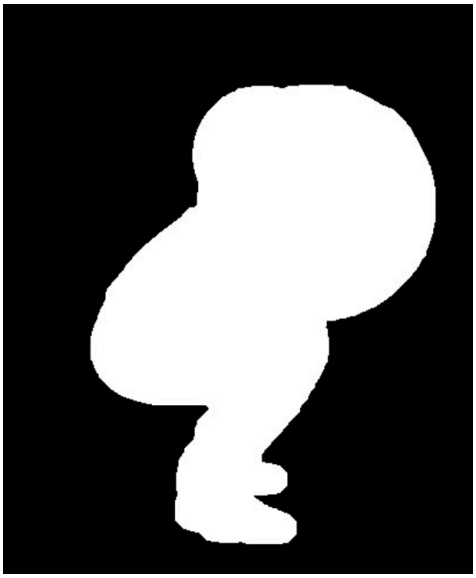


Figure1 Squat

We develop two methods to solve this task, one is fast enough to do real time analysis but is limited to relative simple background scenario, the other is robust enough to handle complex background situation but has too much computation and can hardly be adopted by a real time system

## Hypothesis

Our hypothesis is closely connected with the dataset. The general hypothesis includes none camera motion, only one person in work space at one time, movement parallel to the camera-plane, slow and continuous movements static background. For the sake of computational time in the Hough transform, we also assume that the radius of barbell is between 10 percent and 20 percent of the image's width. In the studio like video we assume background has no color closely related to the human clothing, background color is unanimous. In the complex background video, high frequency background noise is allowed and we can handle lines, which has the same color as human clothing, in the background.

## Dataset & dataset documentation

Dataset of this project is built by ourselves. Essentially, we recorded video of a person doing squat and details will be discussed as follows. Videos are recorded in a gym subject to our assumption, two different kinds of background are covered. One is a studio like background where color of background is of high contrast to foreground color. The other has relative complicated background, high frequency noise from ground and wall can exist, same part of the background connected with foreground in the 2D image can have same color with foreground, this indeed brings in lots of trouble for segmentation, because foreground and background are connected in the view of a single frame. Our dataset is video based so we can only provide typical frames in this report. Fig 2 is from simple background video and fig 3 is from relative complicated background.

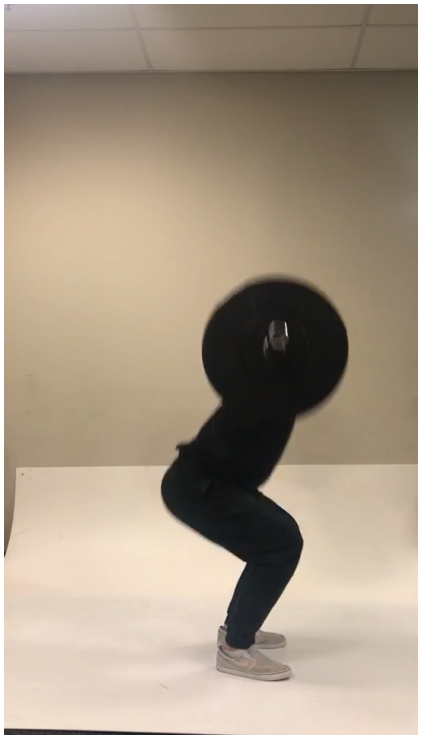


Figure 2 simple background

Figure 3 complicated ground

No previous specific dataset for squatting with ground truth exist, so we define our own criterion for the ground truth. There are two candidate criterion performs as our ground truth. The first one is we put obvious markers on the point we want to extract and track, and simply consider the markers as the ground truth. Figure 4 is an example of this criterion. The second criterion is more efficient for evaluation and will have less problem with the possible influence of marker can have on the tracking algorithm. It's based on the tracking of target point we want, we just compare the point we extract from the image with the original color image, deal the results of each point like a classification point, vote for yes if the point is near the target points and no if the target point is far away. This is purely decided by human common sense, and because falsely detected points are usually far away from the target, this guarantees this criterion do make some sense.


Figure 4 with marker

## Conduction of experiment

In general, however simple or hard the dataset is, we will start from segmentation with the video. In the segmentation part, we tested different algorithms finally chose two methods to handle simple and complicated background respectively. After a segmented video has been obtained, we do feature extraction on this. Different ideas to model human body have been explored and experimented, some has poor results and some are too ideal to implement on this practical task.

**Background extraction** : To handle complicated background video better, a background extraction method is developed. We suppose to subtract each frame with the first frame to remove the background. While our first dataset has the barbell in each frame. And the barbell will move in the later frame. So, we cannot get a total static background from our dataset. We want to extract a static background and use the synthetic background to be the background template.

First of all, to simplify the problem and decrease the computation time. We split the image into patches. Then, we picked some important patch such as the patch on the barbell, some patches on the static object and patches on the people. We calculated the average brightness of picked patch in all frames. And we plot the brightness time curve to analysis how to write the algorithm.

We found that all the patches will have a flat part at the first several frames. Because now the user was not shown in the video. And we can see that the patches on the barbell has a low brightness. Because the color of the barbell is black. So, we can use this property to find out the rough area of the barbell. And other area is the static background which we can apply them in the final template.

Next, we find when the user was shown in the video. The curve will change periodically. We focus on the barbell area. Some of those patches will change its brightness because the barbell will move and the background will be exposed and the brightness of the background is different to the barbell brightness. So

we will change all the pixels in the patch to the exposed background value. So, part of the barbell area will be like “transparent”. We can see the back of it.

Now there are still some patches are not be processed. These patches might be some dark static background. There are some dark mats in the video. So they are marked as barbell area at first. And even the barbell has moved away, the back of the barbell is still dark. So actually they are some dark area. And we set all the pixels in these patches to the original values like in the first several frames. Because they are supposed to be some dark area, no matter it belongs to barbell or some dark background.

The above is our thoughts after analysing the brightness time curve of each patch.

We follow these thoughts and ideas and write this algorithm in Matlab:

First, we create a template background matrix which has the same size of each frame of the video. And then we split it into the 4*4 patches. And we set all the brightness to 0. So the image is now whole black.

Second, we read the video and use its first frame as the history. And we use the media filter to filter the image. Because this filer can get the better result for the barbell area. The edge will not lose. We can get a better region of the barbell area. And we split the first frame in to 4*4 patch.

And for the following frames, we split the frame into 4*4 patch. We calculate the brightness difference for each patch in the frame sequence. If the brightness is larger than 100 and in the continuous 10 frames the values of brightness are stable in a specific range, we assume this patch should belong to the static background. So we stick this patch into the result image. And now the result image should be filled with some patches. And some of the patches in the result image are still black. This region is the rough barbell region which may include dark background and barbell.

When the user is moving the barbell. Some patches belong to the barbell will move and the real background will be exposed. So when the real background is exposed. The patches belong to the barbell will be qualified to be sticked to the result image. As we can see the barbell will disappear and the real background will be exposed.

Now, most of the patches in the real background image are set. And we set a stop point, when the frame number hit the stop point. For example, the current frame is 200. We will stop process this algorithm and stick all the patches in the current frame to the real background result.

The algorithm will not affect our analysis of the movement. Because we will recovery the real background before the user start to do squatting. The user will lift the barbell at first and when the user is ready, actually all the real background patches are exposed and captured by our algorithm. So we will recovery the real background before the user start to do the squatting. And we can subtract the frame with the result of the recovered background to get our desired foreground and do the following process.

As we can see from the result as shown in figure 5.1, the result is not perfect. We can see that some regions are very good. But some regions’ brightness is different to their neighbors. We think there are two regions. The first one is that the barbell is not totally black, the bar of the barbell is silver which has a high brightness. So we will recognize this part to be the real background. And the second reason is the brightness of the whole video will change because the shadow and the camera. So, we will have some discontinuity of the result. And we stick the patches to the result at different time so that our result will not have some discontinuity.

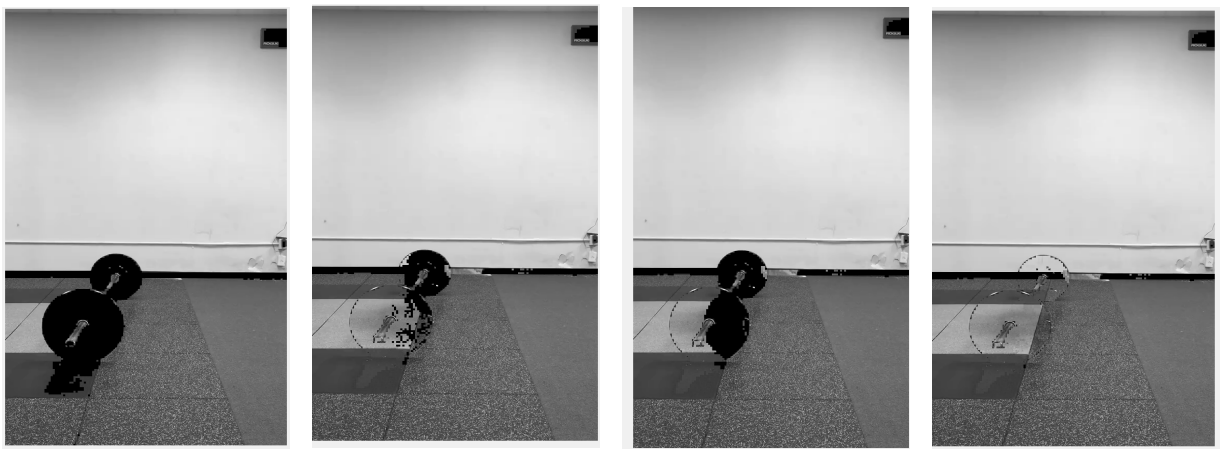


Figure 5.1

To solve the first problem, we need to locate the area of the bar and stick new patch to it when its three direction neighbor changes. So at this time, this patch must move and the real background will be exposed. We can use some algorithm like label to label all the area. First we can dilate the barbell area. And we use label algorithm to get the area of each regions. And the regions with smallest areas are the bar regions which are surrounded by the barbell piece as shown below. The region 1 and 2 have large area. The region 3 and 4 have small area. We can use threshold to select region 3 and 4. And set them to barbell area(black). So we can detect bar area and barbell area as shown in figure 5.2.

To solve the second problem, we can calculate the average brightness of each frame and multiply the patch with a factor to solve the global brightness change problem.

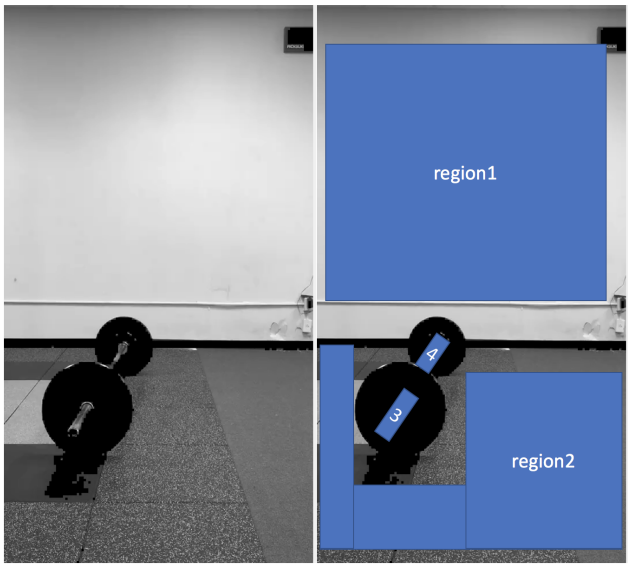



Figure 5.2

We use this recovered background to do the background subtraction. We simply subtract this template from each frame and scale the brightness. The result is shown in figure 5.3.


Figure 5.3

We can see that because the pants color is similar to the background such as the white wall.

And the recovered background is not perfect. So there are some errors in the result.

We can take a better dataset and refine the recovered background algorithm to get a better result later.

With the background extraction, we continued our experiments on pre-processing with the input video by means of the following techniques.

The objective of the preprocessing experiment is to identify the best result with least noise and most complete contour of the person who is doing squatting in the video. We experimented different pre-processing techniques and execute it on our specific dataset.

Four main types of techniques were experimented:

- Background Subrtraction: Five programs were developed in total in this part: We wrote our own background subtraction algorithm as a baseline, then implemented other four programs based on four type of background subtractors in openCV; In the end we combined them together into one program: back_sub.py. For optimal result, we chose Gaussian Mixture-based segmentation.

- Edge detection: As an alternative for pre-processing techniques, we implemented Canny Operator to generat the edge video; as an improvement, we implemented autocanny program to remove some false positive;

- Component Extraction: As a further processing for the results from the above two methods, three noise filters were experimented in this part to remove noise. Also, we implemented a program to find the largest component in the resulting video.

- Optic flow motion detection.

As a baseline of optimizing background subtraction, we developed a program using frame differences. Since background subtraction is an idea that detects the moving objects from the difference between the current frame and a reference frame, we implemented our simple background subtraction algorithm accordingly. As the result of our background subtraction algorithm was far from ideal, subsequent four algorithms were implemented using OpenCV.

**BackgroundSubtractorCNT:** Background subtraction based on counting. This is a fast background subtraction solution in OpenCV.

The four parameters involved in this functions are: maxStability is the stability number of frames with same pixel color to consider stable, useHistory determines if we're giving a pixel credit for being stable for a long time, maxStability is the maximum allowed credit for a pixel in history, isParallel determines if we're parallelizing the algorithm

The parameters that performed the best are: (5, True, 15*60, True)

**BackgroundSubtractorGMG:** This algorithm combines statistical background image estimation and per-pixel Bayesian segmentation. By default, it uses first 120 frames for background modeling (thus we get a black window during first few frames). Specifically, this algorithm employs probabilistic foreground segmentation algorithm that identifies possible foreground objects using Bayesian inference. The estimates are adaptive; newer observations are more heavily weighted than old observations to accommodate variable illumination.

The three parameters involved in this functions are history is the length of the history, nmixtures is Number of Gaussian mixtures, backgroundRatio is the background ratio, noiseSigma is the noise strength.

The parameters that performed the best are: (2,0.5)

**BackgroundSubtractorMOG:** This algorithm is a Gaussian Mixture-based Background / Foreground Segmentation Algorithm.

The parameters involved in this functions are: length of history, number of gaussian mixtures, threshold etc. The algorithm uses a method to model each background pixel by a mixture of K Gaussian distributions (K = 3 to 5).

The parameters that performed the best are: (5, 3, 0.7, 0)

**BackgroundSubtractorMOG2:** It is also a Gaussian Mixture-based Background/Foreground Segmentation Algorithm. Compared with BackgroundSubtractorMOG, it provides better adaptability to varying scenes due to illumination changes etc.

For the parameters in this algorithm, we have the option of selecting whether shadow to be detected or not. If detectShadows = True (which is so by default), it detects and marks shadows, but decreases the speed. Shadows will be marked in gray color.

The parameters that performed the best are: (5, 15, True)

The processed results of various background subtractor:

For the same frame in the video with the optimal parameters that we chose, we can get below processed results, which laid the foundation of our final background subtractor choice:

*Figure6.1 Background subtractor processed image (a)CNT (b) GMG (c) MOG (d) MOG2*

After comparing the optimal results, we decide to choose BackgroundSubtractorMOG2 as our main program to segment the input video and compare with the edge video.

**Edge operators:** As an alternative to background subtraction, we tested edge operators in openCV on the input video to get the edge data for comparison. We experimented with Canny operator mainly.

The Canny operator in openCV is powerful in that it embeds several stages including some pre-processing and post-processing techniques.

Looking into its implementations, we can find stages of this function includes:

Step 1: Smooth the image using a Gaussian filter to remove high-frequency noise.

Step 2: Compute the gradient intensity representations of the image.

Step 3: Apply non-maximum suppression to remove “false” responses to edge detection.

Step 4: Apply thresholding using a lower and upper boundary on the gradient values.

Step 5: Track edges using hysteresis by suppressing weak edges that are not connected to strong edges.

Four arguments can be tuned in this function. The first argument is our input image. Second and third arguments are our minVal and maxVal respectively. The third argument is aperture_size. It is the size of Sobel kernel used for find image gradients. By default, it is 3. The last argument is L2gradient which specifies the equation for finding gradient magnitude. If it is True, it uses the equation mentioned the Euclidean formula for the gradient magnitude, otherwise, it uses this function: Edge_Gradient(G)=|Gx|+|Gy|.

To find the optimal value for the arguments in the Canny operator and save the parameter tuning time, autoCanny program was implemented which takes the median of the image, and then constructs upper and lower thresholds based on a percentage of this median.

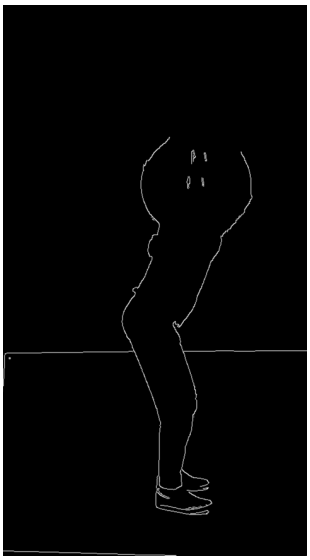


*Figure6.2 auto_canny Edge Operator processed image*

**Component Extraction:** This program was developed with the motivation to eliminate the background noise from the MOG2 processed video and Canny operator processed video. It intends to identify the largest component and extract a region of interest after segmentation, which in our project is supposed to be the contour of the person.

Theoretically, as the small noise components in the image have the smaller size, we can calculate the component size in the image and eliminate the smaller ones. The function connectedComponentsWithStats was used to remove the components with the size of fewer than 300 pixels (See below program Component_extraction.py for reference).

Unfortunately, this algorithm failed in that it will eliminate some edges of the person which are not continuous when applied. This made the future analysis of the key points(knees, feet) more difficult.

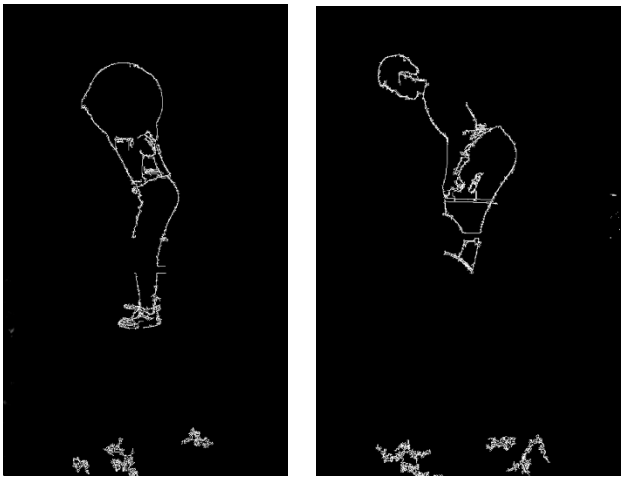


*Figure6.3 Component Extraction processed image*

Another method to localize feature points called negative minimum curvature (NMC) has been experimented, but due to limited reference paper on that, the method is by no means robust and can only be abandoned. However, starting from the idea of curvature, we initialize an idea based on combination of curvature and prior human body knowledge. In this method, we first represent human body as polylines, then according to the poly lines, tangent values of the curve can be approximated.

**Optical flow**: We use the canny edge detector to find all the edges in each frame. While except the body area edge, there are many other edges like we also detect the wall, the mats on the ground. We change the parameter to remove these edges. But the effect is not very good. And we tried to use hough transform to detect some special edges and remove them. Because there are many kinds of edges, so that we cannot remove all the edges. And we tried to use morphological filter to remove the useless edges. For the same reason, this method doesn't work.

We find we just want the edges in a small region. We can find the people area and do the canny edge process in this area. The spatial differences between people and the background are their brightness, their shape and so on. And we cannot separate people with the background perfectly with this spatial information. So we want to use temporal information to find the people area. Because mostly, the user is moving. So, firstly, we think we can subtract two continue frames to get the difference between two continuous frames and use some threshold and filter to get the result. While the result is not very good. Because there are too many noises in the video which we cannot filter. And the difference method is very sensitive to noise. We tried different patch size and filter. While the result is not very good. So this method cannot work.

Then we think we can use optical flow to track the moving object. We choose to use Horn-Schunck method to do the optical flow process. We can get the movement vector of each patch so that we can know the movement speed of each patch. We can use a threshold to detect the moving object. We set the threshold to filter the noise and low speed part. The result is better than the difference method. While many low speed body parts are lost. For example, the upper body part is very good. Because it will move all the time. While

the foot is not clear. Because the foot will not move some time when you are moving. So we cannot get a good extraction of the whole body. Because the principle of the method is to detect moving object.

We think edge information may make up the faults. Because the edge detect result can always show all the body parts. We design a algorithm to make up the lost part. If some moving patch turns to stop. From the image, these patches turns black from white. We assume these parts may have a low speed at this time. So we will check the edge detection result whether the patch in the same location contains edges or not. If so, we think it is part of body, so that we will set this patch to white. And keep this patch to white for a while. For example, we will keep this patch to be white for 30 frames(1s).

The result is very good, we can get a rough area which contains almost only the user. And then, we did some dilation and closing process on the result. And we fill the hole in the result. So we can get a white region of people. And then we did edge detection on this region. We will get a good result and remove all other edges. The reason why we don't use the optical flow result is the result of the single method is not accurate. And we did one more process in our algorithm. We sum up the first 20 frames edge information as edge history. And if our edge output has the same edge information as the history. We will abandoned it. Because all the edges in the history should not belong to the body edge. The edge detection on a specific area method becomes much better. The result is shown in figure 7.1 and 7.2.

The top left image is the movement vector and the origin image. And the top middle image is the result after speed threshold and edge compensate. And the top right image is the result of morphological process which is a closing process with a 30 pixel diameter disk. And the bottom left image is the original edge detection result. And the bottom middle image is the edge result after processing edge detection on the top right image with history subtraction method. And bottom right result is the morphological result of the edge detection which is a closing process with a disk kernel(rad = 2) . Because the algorithm is slow so that we cannot process every frame. So we will get one best frame. With this result we can find the important knots and track these knots. While the tracking result is not good so that this method can be used in other application. We can refine this algorithm later to make it faster.

Figure 7.1

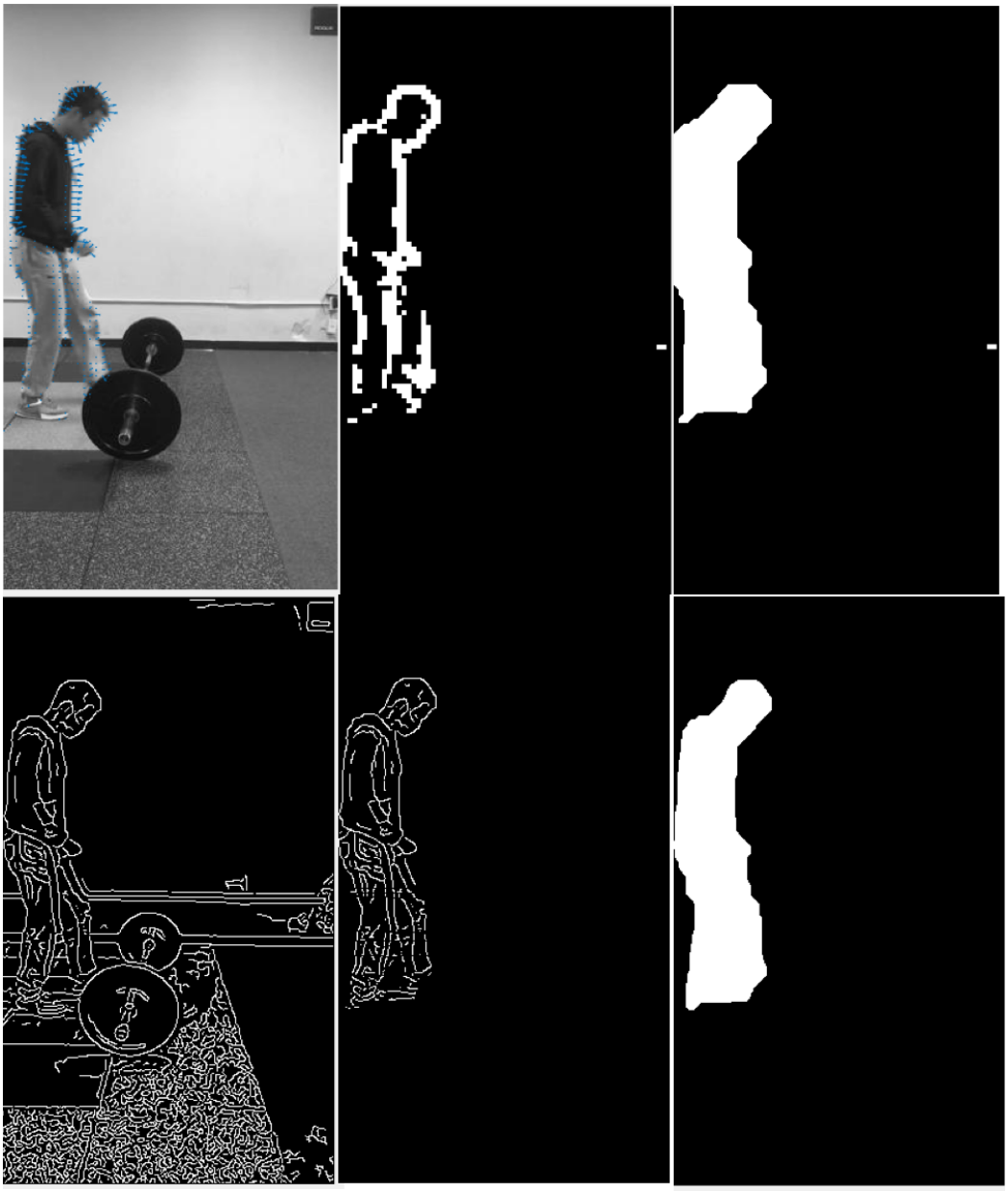


Figure 7.2

## Algorithm specific design

Mixture of Gaussian is our main principle to handle simple background while optical flow and morphology filtering is the guiding idea for the program to deal with complicated background. These two methods with their parameters have been explained in detail in the experiment conduction part.

After segmented segmentation has been done Two methods are developed and a comparison between them is given. One method is to use a general feature extraction method in which we extract the four points.

Vtrack is a candidate to track the four target points. First we find the frame where the person has come into the working space and stood straight, four target points are extracted in that certain frame, then vtrack from visionx is used to perform tracking. Different parameters have been tuned but the results are poor. Table1 has shown the parameters and corresponding result figure number.

| Figure number | V(number of pixel in horizontal search) | S(number of pixel in vertical search) | Search technique | Size of correlation patch matrix |
|---|---|---|---|---|
| 8 | 8 | 8 | grid | 15 |
| 9 | 20 | 20 | vel | 15 |
| 10 | 20 | 20 | Lin | 20 |
| 11 | 20 | 25 | lin | 25 |
| 12 | 30 | 30 | lin | 30 |

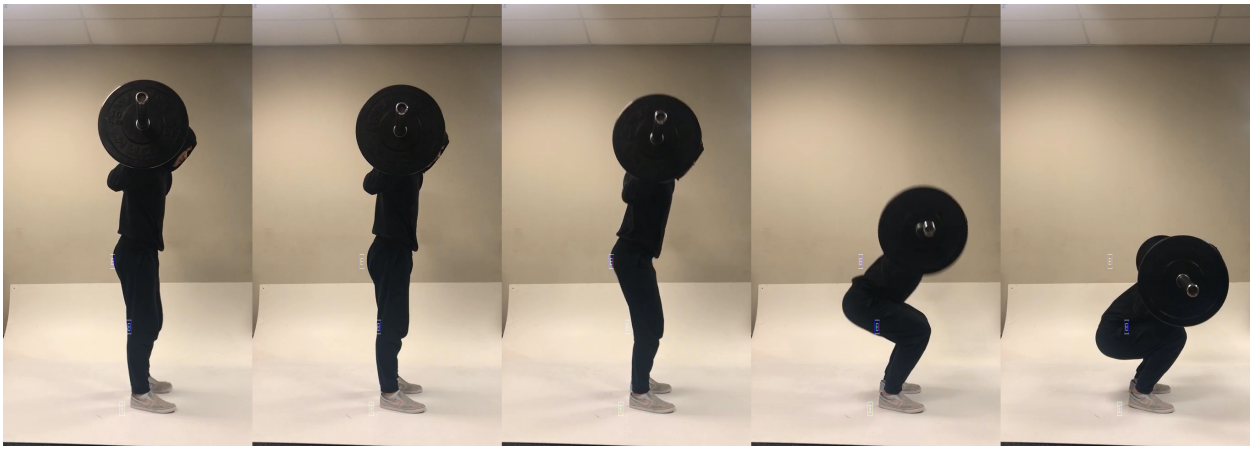


Figure 8 vtrack results

Figure 9 vtrack results

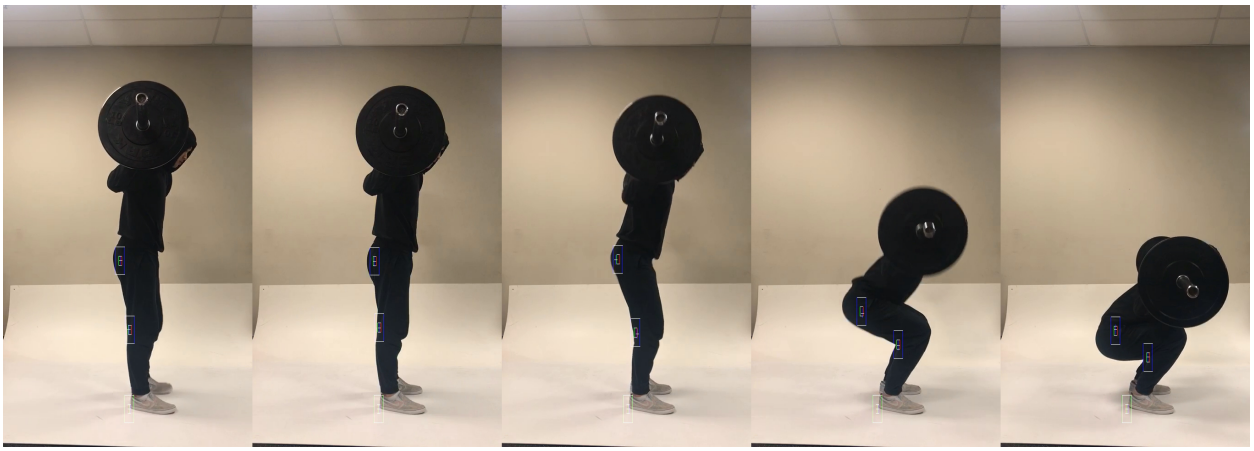


Figure 10 vtrack results

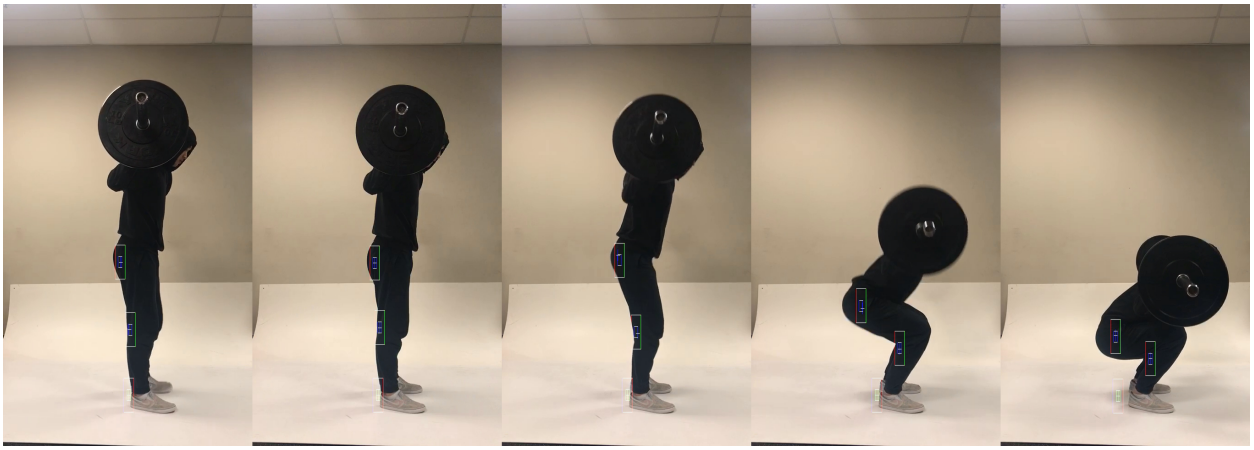


Figure 11 vtrack results

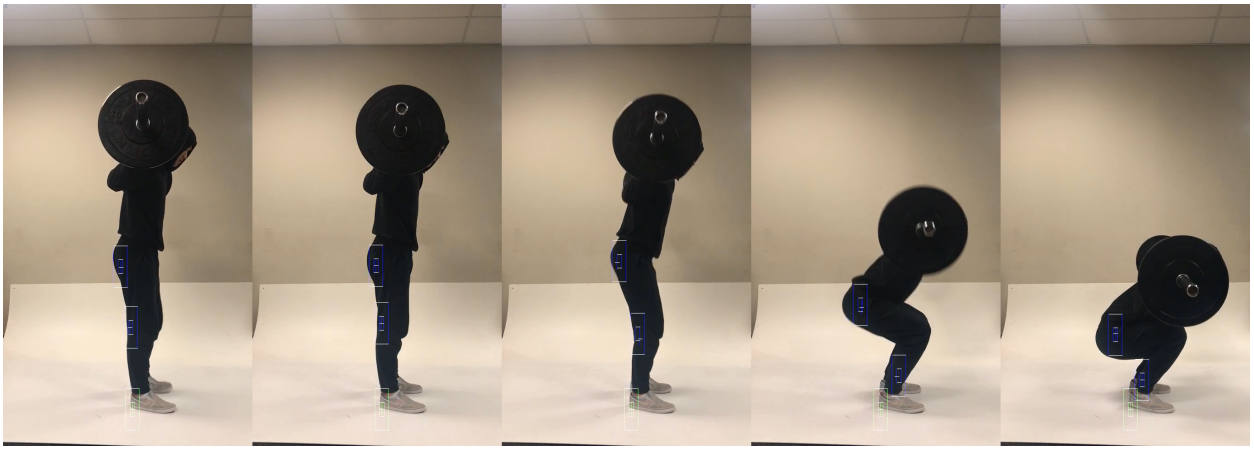


Figure 12 vtrack results

It can be concluded that when the size of searching area goes bigger, the results are not terrible but still far from our original needs. However, such tracking method is still under consideration because it is the fast way that we can come up with to track the points after they extracted in one frame.

**Extract points in every single frame:** The other method we have adopted so far is to search each frame separately, this method is relatively slow because a mixture of circular Hough transform, prior knowledge based searching are needed for each frame. Provided with enough computational ability or a more strict assumptions. This can still be a real time task.

Following is the detailed information of this method. Our prime target in this procedure is to find the four points that can guide us find the two angles that we want. After lots of consideration and discussion, we found the fact that if we just search the leftmost of the foreground in the middle (in vertical direction) part of the image, chances are that we find the point representing hip. And it's quite obvious that the foot of a human is the lowest point of him/her. Two points are fixed in this way, then we will find the point representing knee and back. To search the foot-knee-hip connection, we take advantage of the connectivity idea to locate the elbow and knee joints from the hip point and foot point. [ref] Based on the human kinematic constraints, if we make a straight line L from hip from to foot, the knee has a movement limitation of $L_{\perp}$which intersects with L at the midpoint of foot to hip. Therefore, by searching all the points on $L_{\perp}$, the knee point can be obtained by minimizing the equation below:

$$E_{total} = argmin\{E(knee, foot) + E(hip, knee)\}$$

Where E is the connectivity energy function. For the back, we utilize a trick of squatting. Normally, the barbell is set on the top of the back behind the neck. Thus the center of the barbell can be considered as the top point of back. A circular Hough transform is designed, we tune different parameters and decide a set of parameters that can find the most suitable circle center. The result got from this algorithm is visually all percent great and a sample from the video will be shown in the result part.

## Results & observation

The result of tracking four target points from the simple background condition is shown in figure 13. Visually, we can say the tracking is one hundred percent right. Originally, we think the dataset with labels may be considered as cheating, so after the report with our results from dataset without labels have been submitted. We will take video with obvious markers on so that we can develop an algorithm to automatically calculate in each frame, how distant the target points our program found is from the ground truth we marked. And videos of different person with slightly different poses will also be analysed. Then we can put that new results in addendum report.

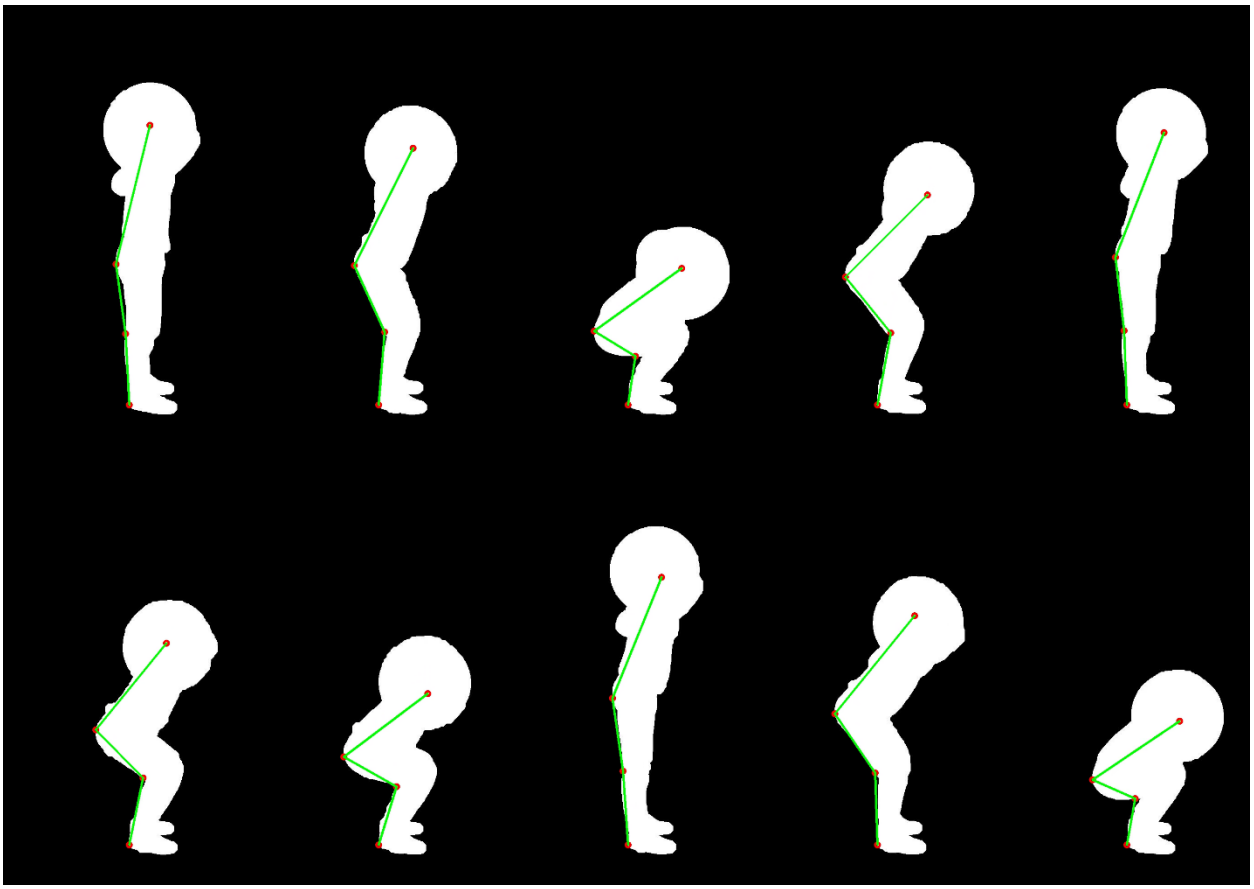
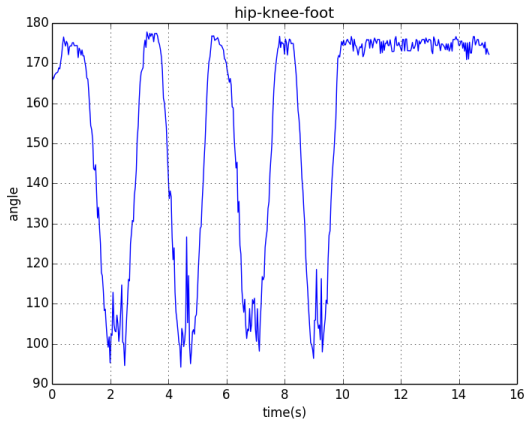


Figure 13 tracking result

In this report, we present the two angles we calculate from the four points we tracked in the relatively simple background. Fig 10 is the angle at knee and figure 15 is the angle we get at hip. In the hypothesis part, we assumed that the movement is continuous and slow. So combined with the video result, we can say that the smooth part of the angle curve is definitely right, we check the frame corresponding to the sharp change of the angle curve and also found the tracking is right. So we can now conclude that our tracking on a simple background video for squatting is basically right. Fig 16, 17, 18 illustrates the trajectory of knee, hip and top of the back respectively.



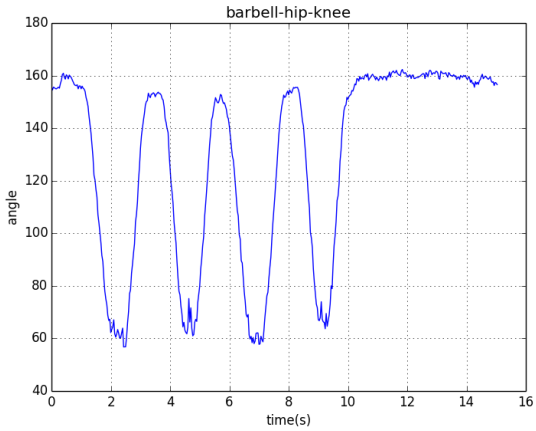




Figure 14 knee

Figure15 hip

However, in the experiment process of program like optical, we can observe that program with lots of process is time consuming to do segmentation on the complicated background video.

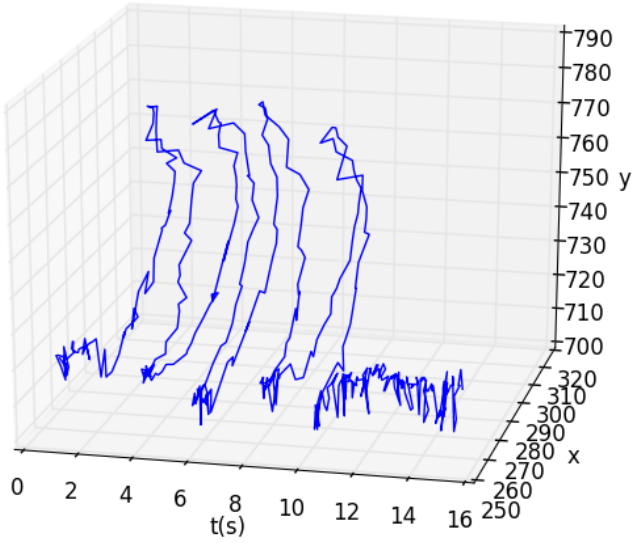



Figure 16 knee

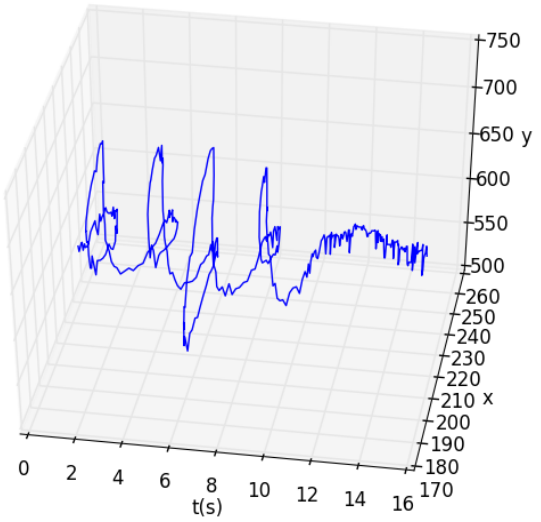



Figure 17 hip

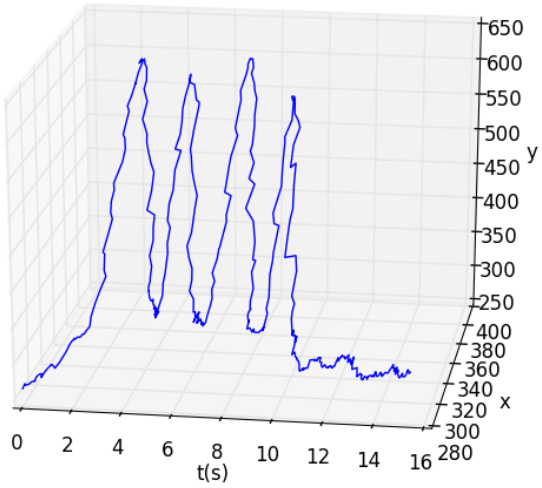



Figure 18  top of back

## Conclusion

Basically, we have developed a complete set of programs to finish the squat tracking analysis based on video subject to our hypothesis. Video with markers will be recorded and numerical analysis based on the distance between our detected points and markers will be provided in the addendum report. Feature extraction methods based on curvature can also be developed in future and results of complicated background will also be analyzed in the addendum report.

## Program Listing & documentation

```
#########################################################################
#   NAME:       back_sub.py                                 #
#   Description: Background subtraction experiments                 #
#   Author:     Jianhua Fan(jf773)  Qian Qiao(qq39)                 #
#               Beitong Tian(bt346) YanFei Xu(yx427)                #
#   Data:       11/20/2017                                  #
#########################################################################

import cv2
import numpy as np
import sys
import os
from matplotlib import pyplot as plt

def back_Sub(fpath, mode):
    cap = cv2.VideoCapture(fpath)
    total_frame = cap.get(cv2.CAP_PROP_FRAME_COUNT)
    print(total_frame)
    if mode == 'MOG2':
        fgbg = cv2.createBackgroundSubtractorMOG2(5, 15, True) #history, varThreshold, bShadowDetection
    elif mode == 'MOG':
        fgbg = cv2.bgsegm.createBackgroundSubtractorMOG(5, 3, 0.7, 0) #history, nmixture, backgroundRatio, noiseSigma
    elif mode == 'GMG':
        fgbg = cv2.bgsegm.createBackgroundSubtractorGMG(2,0.5)#initializationFrame, decisionThreshold
    elif mode == 'CNT':
        fgbg = cv2.bgsegm.createBackgroundSubtractorCNT(5, True, 15*60, True) #minStability, useHistory, maxStability, isParallel
    else:
        print('We only support methods of MOG2, MOG, GMG, CNT now!')
        exit(1)

    fourcc = cv2.VideoWriter_fourcc('m','p','4','v')
    output_video = '../results/video/{}.m4v'.format(mode)
```

```
    try:
        os.remove(output_video)
    except OSError:
        pass
    out = cv2.VideoWriter(output_video, fourcc, 20.0, (544, 960), False)


    try:
        while cap.isOpened():
            current_frame = int(cap.get(cv2.CAP_PROP_POS_FRAMES))
            rate = cap.get(cv2.CAP_PROP_FPS)
            ret, frame = cap.read()
            if ret:
                height, width, layers = frame.shape
                fgmask = fgbg.apply(frame)
                if current_frame == 800:
                    cv2.imwrite('../results/image/original_{}.png'.format(current_frame), frame)
                    cv2.imwrite('../results/image/{}_{}.png'.format(mode, current_frame), fgmask)
                out.write(fgmask)
                cv2.imshow('frame', fgmask)
                print(current_frame)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
            else:
                break
        cap.release()
        out.release()
        cv2.destroyAllWindows()
    except KeyboardInterrupt:
        print('Stopped for ctr-c')
        cap.release()
        out.release()
        cv2.destroyAllWindows()


def main():
    video_name = 'output_CannyEdge.m4v'
```

```
    mode = sys.argv[1]
    input_path = '../results/video/'
    fpath = input_path + video_name
    back_Sub(fpath, mode)


if __name__ == "__main__":
    main()
```

```
#########################################################################
#   NAME:       backsub_ours.py                                         #
#   Description: Background subtraction combines statistical background  #
#                image estimation and per-pixel Bayesian segmentation.   #
#   Author:     Jianhua Fan(jf773)  Qian Qiao(qq39)                     #
#               Beitong Tian(bt346) YanFei Xu(yx427)                    #
#   Data:       11/20/2017                                              #
#########################################################################
import cv2
import numpy as np


cap = cv2.VideoCapture('../data/trim_start_end.mp4')
i = 0
while True:
    i += 1
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    if i == 100:
        background = gray
        cv2.imwrite('../results/ours_image_{}.jpg'.format(i), background)
        break


while True:
    i += 1
    ret, frame = cap.read()
    height, width, layers = frame.shape
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gray = gray - background
    cv2.imshow('frame', gray)
    if i == 200:
        cv2.imwrite('../results/ours_image_{}.jpg'.format(i), gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
out.release()
```

```
cv2.destroyALLWindows()
```

```
#########################################################################
#   NAME:       autocanny.py                                            #
#   Description: Automatic threshold for Canny edge operator            #
#   Author:     Jianhua Fan(jf773)  Qian Qiao(qq39)                     #
#               Beitong Tian(bt346) YanFei Xu(yx427)                    #
#   Data:       11/20/2017                                              #
#########################################################################
import cv2
import numpy as np
from matplotlib import pyplot as plt


cap = cv2.VideoCapture('../data/input.mp4')
fourcc = cv2.VideoWriter_fourcc('m','p','4','v')
out = cv2.VideoWriter('../results/CannyEdge.m4v',fourcc, 20.0, (544,960), False)
i = 0
while True:
    i += 1
    ret, frame = cap.read()
    blur = cv2.GaussianBlur(frame, (5, 5), 0)   # noise removal
    sigma = 0.33
    v = np.median(frame)     # median pixel
    lower = int(max(0, (1.0 - sigma) * v))
    upper = int(min(255, (1.0 + sigma) * v))
    edges = cv2.Canny(blur, lower, upper)
    cv2.imshow('frame', edges)
    out.write(edges)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

```
#########################################################################
#   NAME:       Component_extraction.py                                 #
#   Description: Identify the largest component in the image            #
#   Author:     Jianhua Fan(jf773)  Qian Qiao(qq39)                     #
#               Beitong Tian(bt346) YanFei Xu(yx427)                    #
#   Data:       11/20/2017                                              #
#########################################################################
import cv2
import numpy as np
import sys
import os
from matplotlib import pyplot as plt


def noise_Remove(fpath):
    cap = cv2.VideoCapture(fpath)
    total_frame = cap.get(cv2.CAP_PROP_FRAME_COUNT)
    print(total_frame)

    fourcc = cv2.VideoWriter_fourcc('m','p','4','v')
    output_video = '../results/video/MOG2_noise_Remove.m4v'

    try:
        os.remove(output_video)
    except OSError:
        pass
    out = cv2.VideoWriter(output_video, fourcc, 20.0, (544, 960), False)

    try:
        while cap.isOpened():
            current_frame = int(cap.get(cv2.CAP_PROP_POS_FRAMES))
            rate = cap.get(cv2.CAP_PROP_FPS)
            ret, frame = cap.read()

            if ret:
                height, width, layers = frame.shape
```

```
            fgmask = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            nb_components, output, stats, centroids = cv2.connectedComponentsWithStats(fgmask, connectivity=8)    #Component statistics
            sizes = stats[1:, -1]; nb_components = nb_components - 1
            min_size = 300
            img2 = np.zeros((output.shape), np.uint8)
            for i in range(0, nb_components):
                if sizes[i] >= min_size:
                    img2[output == i + 1] = 255
                    plot.imshow(img2)
                    plot.show()
            out.write(img2)
            cv2.imshow('frame', img2)
            print(current_frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        else:
            break
    cap.release()
    out.release()
    cv2.destroyAllWindows()
  except KeyboardInterrupt:
    print('Stopped for ctr-c')
    cap.release()
    out.release()
    cv2.destroyAllWindows()

def main():
  video_name = 'MOG2.m4v'
  input_path = '../results/video/'
  fpath = input_path + video_name
  noise_Remove(fpath)



if __name__ == "__main__":
```

```
main()
```

Tracking.py:

```
from __future__ import print_function
import cv2
import numpy as np
import sys
import os
from matplotlib import pyplot as plt
import sys


def com  angle(bar, hip, knee, foot):
    first = hip - bar
    second = hip - knee
    three = foot - knee
    cosine  angle  bhk = np.dot(first, second) / (np.linalg.norm(first) * np.linalg.norm(second))
    cosine_angle_hkf = np.dot(second, three) / (np.linalg.norm(second) * np.linalg.norm(three))
    print(cosine_angle_hkf)
    angle_bhk = np.arccos(cosine_angle_bhk)
    angle  hkf = np.arccos(cosine  angle  hkf)
    print(np.degrees(angle_hkf))
    return np.degrees(angle_bhk), np.degrees(angle_hkf)

def find  joint(img, prev  x, prev  y):

    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # ret,img = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)

    (height, width) = img.shape


    ###hip bottom - 1/2
    x_hip = width
    y_hip = -1
    for i in range(height-1, height/2, -1):
        for j in range(width):
            if img[i, j] ==255:
                break
        if j < width - 1:
            if j < x  hip:
                x_hip = j
                y_hip = i

    # print(x  hip,y  hip)

    ####foot  bottom to 1/8
    x_foot = width
    y  foot = -1
    # for i in range(height-1, height*4/5, -1):
    #     for j in range(width):
    #         if img[i, j] == 255:
    #             break
    #     if j < width - 1:
    #         if j < x_foot:
    #             x_foot = j
    #             y  foot = i
    x_foot = 275
    y_foot = 872
    # print(x_foot,y_foot)

```

```
59.     ######knee  in the middle of hip and foot
60.     x_knee = 0
61.     y  knee = 0
62.     ########method 1
63.     # for i in range(height*4/5, y_hip, -1):
64.     #     for j in range(width):
65.     #         if img[i, j] == 255:
66.     #             break
67.     #     if j < width - 1:
68.     #         if j > x_knee:
69.     #             x  knee = j
70.     #             y_knee = i
71.     # real_x_knee = x_knee
72.     # real_y_knee = y_knee
73.     # if prev  y  knee != 0 and abs(prev  y  knee - y  knee) > 10:
74.     #     y_knee = prev_y_knee
75.     #     x_knee = prev_x_knee
76.
77.     # print(x  knee, y  knee)
78.
79.     ####method 2
80.     # for i in range(height*4/5, y_hip+50, -1):
81.     #     for j in range(width-1, -1, -1):
82.     #         if img[i, j] == 255:
83.     #             break
84.     #     if j > 0:
85.     #         if j > x  knee:
86.     #             x_knee = j
87.     #             y_knee = i
88.     # print(x_knee, y_knee)
89.
90.     #####method 3
91.     k = float(y_hip-y_foot)/(x_hip-x_foot)
92.     middle_k = (0 - 1 / k)
93.     middle  x = (x  hip+x  foot)/2
94.     middle_y = (y_hip+y_foot)/2
95.     for j in range(width):
96.         i = middle_y+middle_k*(j - middle_x)
97.         i = int(i)
98.         if j > 0 and j < height and img[i, j] == 255:
99.             x_knee = j
100.            y_knee = i
101.            break
102.    # print(x_knee, y_knee)
103.
104.
105.    ####head
106.    circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,1,
107.                               param1=30,param2=20,minRadius=90,maxRadius=150)
108.    # circles = circles[0,:]
109.    if  circles is not None:
110.        circles = np.uint16(np.around(circles))
111.        # circles = circles[circles[:,2].argsort()]
112.        # x, y, r = circles[0]
113.
114.        x_bar = y_bar = 0
115.        min_diff = sys.maxint
116.        for (x,y,r) in circles[0, :]:
117.            diff = (x-prev  x)*(x-prev  x)+(y-prev  y)*(y-prev  y)
118.            if diff < min_diff:
119.                min_diff = diff
```

```
                x_bar = x
                y_bar = y
        x = x  bar
        y = y_bar
    else:
        x = prev_x
        y = prev  y


    img = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
    cv2.circle(img,(x_foot, y_foot),5,(0,0,255),3)
    cv2.circle(img,(x_hip, y_hip),5,(0,0,255),3)
    cv2.circle(img,(x  knee, y  knee),5,(0,0,255),3)
    cv2.circle(img,(x, y),5,(0,0,255),3)

    # cv2.circle(img,(x, y),r,(0,255,0),2)
    # cv2.circle(img,(x, y),2,(0,0,255),3)

    cv2.line(img,(x_foot,y_foot),(x_knee,y_knee),(0,255,0),3)
    cv2.line(img,(x_hip,y_hip),(x_knee,y_knee),(0,255,0),3)
    cv2.line(img, (x  hip,y  hip), (x, y), (0,255,0), 3)

    bar = np.array([x_bar, y_bar])
    hip = np.array([x_hip, y_hip])
    knee = np.array([x  knee, y  knee])
    foot = np.array([x_foot, y_foot])

    with open('../results/txt/bar.txt', 'a') as f_bar:
        print(bar, file=f  bar)
    with open('../results/txt/hip.txt', 'a') as f_hip:
        print(hip, file=f_hip)
    with open('../results/txt/knee.txt', 'a') as f_knee:
        print(knee, file=f  knee)
    with open('../results/txt/foot.txt', 'a') as f_foot:
        print(foot, file=f_foot)

    bhk, hkf = com  angle(bar, hip, knee, foot)

    return (img, x, y, bhk, hkf)

def read  video(fpath):

    cap = cv2.VideoCapture(fpath)
    total_frame = cap.get(cv2.CAP_PROP_FRAME_COUNT)


    fourcc = cv2.VideoWriter_fourcc('m','p','4','v')
    output_video = '../results/video/joints_maximum.m4v'
    try:
        os.remove(output_video)
    except OSError:
        pass
    out = cv2.VideoWriter(output  video, fourcc, 30.0, (544, 960), True)


    try:
        x = y = 0
        while cap.isOpened():
            current_frame = int(cap.get(cv2.CAP_PROP_POS_FRAMES))
```

```
            rate = cap.get(cv2.CAP_PROP_FPS)
            # print(rate)
            ret, frame = cap.read()
            if ret and current_frame < 500:
                height, width, layers = frame.shape

                result, x, y, bhk, hkf = find  joint(frame, x, y)
                with open('../results/angle.txt', 'a') as f:
                    print(current_frame, bhk, hkf, file=f)

                cv2.namedWindow('image',cv2.WINDOW  NORMAL)
                cv2.resizeWindow('image', 600,600)
                cv2.imshow('image', result)
                out.write(result)
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
            else:
                break
        cap.release()
        out.release()
        cv2.destroyAllWindows()
    except KeyboardInterrupt:
        print('Stopped for ctr-c')
        cap.release()
        out.release()
        cv2.destroyAllWindows()

def main():
    video_name = 'new_trim.m4v'
    input_path = '../data/'
    fpath = input  path + video  name
    read_video(fpath)


if __name__ == "__main__":
    main()
```

## UNIX style Program Description:

**NAME**

back_sub – pass a key word to perform corresponding background subtraction

**SYNOPSIS**

back_sub [-method]

DESCRIPTION

Back_sub program pass the choices of background_subtractor to perform background subtraction on the input video, the choice provided in our program are: MOG2, MOG, GMG, CNT.

CONSTRAINTS

The input should be a video with frame size 544*966.

OPTIONS

[value] Specify the background subtractors to choose: MOG2, MOG, GMG, CNT

AUTHOR(S)

Jianhua Fan(jf773) Qian Qiao(qq39)

SEE ALSO

backsub_CNT.py, backsub_GMG.py, backsub_MOG.py, backsub_MOG2.py

**NAME**

autocanny – automatically generate optimal edge frames in a video

**SYNOPSIS**

autocanny

DESCRIPTION

This program takes the median of the image from a video, then constructs upper and lower thresholds based on a percentage of this median, then pass to canny edge operator to generate the optimal edge image.

CONSTRAINTS

The input should be a video with frame size 544*966.

OPTIONS

None

AUTHOR(S)

Jianhua Fan(jf773) Qian Qiao(qq39)

SEE ALSO

None

**NAME**

Component_Extraction –Identify the largest component

**SYNOPSIS**

Component_Extraction

**DESCRIPTION**

This program takes a video as input, identify the largest component in each frame, and output .

**CONSTRAINTS**

The input should be a video processed after segmenttion, with each frame size 544*966. .

**OPTIONS**

None

**AUTHOR(S)**

Jianhua Fan(jf773) Qian Qiao(qq39)

SEE ALSO

None

**Name:**

<OE_bg_subtraction>-<This program will return a perfect body region>

**SYNOPSIS:**

OE_bg_subtraction [add = address] [ps = patchsize] ] [th = threshold]

**Description:**

This program can automatically recognize the frame when the user is fully shown in the video. And automatically remove noises and extract the body region with black and white output image.

**Constraints:**

The dataset(video) should have some constraints. The user should step into the video from one side.

**Options :**

Address =<absolute address>

Specifies the address of the video file

patchsize = <int>

specifies the patch side length, default is 4

Threshold = <int>

Specifies the minimum speed that a patch can be recognized as a body part

**Authors**:

Squat detection team

**NAME**

tracking.py – track the key features of people doding squatting from video.

**SYNOPSIS**

python tracking.py

**DESCRIPTION**

This program is used to track and compute the key features from video sequence, including hip, knee, foot and barbell centroid. In addition, it computes the angle between the line of bar-hip and hip-knee, the line of hip-knee and knee-foot.

**CONSTRAINTS**

The input video is 544*960

**AUTHOR**

Jianhua Fan

**Name:**

<Background_extraction>-<This program recovers the real background>

**SYNOPSIS:**

Background_extraction [add = address] [ps = patchsize] ] [th = threshold] [tt = tmpth] [cb = countbg]

**Description:**

This program can recover the real background from a video. It can remove some simple object in the image.

**Constraints:**

The dataset(video) should contain an object whose brightness is different from the background it blocks. And the object should be able to move and expose the real background it blocks totally.

**Options :**

Address =<absolute address>

Specifies the address of the video file

patchsize = <int>

specifies the patch side length, default is 4

Threshold = <int>

specifies the max brightness difference that we can infer two frames are stable, default is 2

Tmpth = <int>

specifies the max brightness that a patch can be inferred to foreground

Countbg = <int>

specifies the minimum stable frame number that a patch can be set to result

**Author**s:

Squat detection team

Background extraction

```
patchsize = 4;
threshold = 8;
count_th = 60;
countbg = 3;
threshold2 =2;
tmpth = 60;
patcharea = patchsize * patchsize;
curr_temp =zeros(patchsize);
prev_temp =zeros(patchsize);
filename = 'computer visoin project 2017-11-11 10.18.55.mp4';
mov = VideoReader(filename);
numFrames = mov.NumberOfFrames;
FramesX = mov.Height;
FramesY = mov.Width;
cell_number_Y = mov.Height/patchsize;
cell_number_X = mov.Width/patchsize;
histogram1 = zeros(cell_number_Y,cell_number_X,numFrames);
status = zeros(cell_number_Y,cell_number_X);
count = zeros(cell_number_Y,cell_number_X);
patch_matrix_Y = ones(1,cell_number_Y)*patchsize;
patch_matrix_X = ones(1,cell_number_X)*patchsize;
numFramesWritten = 0;
PrevFrame = rgb2gray(read(mov,1));
PrevFrame_split = mat2cell(PrevFrame,patch_matrix_Y,patch_matrix_X);
HIS = PrevFrame_split;
Frame_extract = zeros(FramesX,FramesY,'uint8');
Frame_extract = mat2cell(Frame_extract,patch_matrix_Y,patch_matrix_X);
BGhistory = zeros(cell_number_Y,cell_number_X);
for i = 1 : cell_number_Y
    for j = 1 : cell_number_X
        BGhistory(i,j) = sum(sum(PrevFrame_split{i,j} )) / patcharea;
    end
end
BGhistory_count = zeros(cell_number_Y,cell_number_X);

for t = 2 : 270
currFrame = rgb2gray(read(mov, t));
currFrame_split = mat2cell(currFrame,patch_matrix_Y,patch_matrix_X);
current = t
for i = 1 : cell_number_Y
    for j = 1 : cell_number_X
        tempave = sum(sum(currFrame_split{i,j} )) / patcharea;
        temp = abs(tempave - BGhistory(i,j));
        if(abs(temp)<threshold2 && BGhistory_count(i,j)<countbg && status(i,j) ==
0 && tempave> tmpth)
            BGhistory_count(i,j) = BGhistory_count(i,j) + 1;
        elseif (abs(temp)>=threshold2 && status(i,j) == 0)
            BGhistory(i,j) = tempave;
            BGhistory_count(i,j) = 0;
        end
        if (BGhistory_count(i,j) ==countbg && status(i,j) == 0)
            Frame_extract{i,j} = currFrame_split{i,j};
```

```
            status(i,j) = 1;
        end
    end
end

Frame_output = cell2mat(Frame_extract);
figure(1),
imshow (Frame_output);
end

patchsize = 1;
cell_number_Y = mov.Height/patchsize;
cell_number_X = mov.Width/patchsize;
patch_matrix_Y = ones(1,cell_number_Y)*patchsize;
patch_matrix_X = ones(1,cell_number_X)*patchsize;
patcharea =patchsize * patchsize;
Frame_extract_split = mat2cell(Frame_output,patch_matrix_Y,patch_matrix_X);
PrevFrame_split_2 = mat2cell(PrevFrame,patch_matrix_Y,patch_matrix_X);

for i = 1 : cell_number_Y
    for j = 1 : cell_number_X
        tempave = sum(sum(Frame_extract_split{i,j} )) / patcharea;
        if tempave <= 50
            Frame_extract_split{i,j} = PrevFrame_split_2{i,j};
        end
    end
end

Frame_output_2 = cell2mat(Frame_extract_split);
figure(2),
imshow (Frame_output_2);
```

Optical flow based background subtraction with edge detect compensation

```
count = 1;
show =1;

patchsize = 4;

tempave = 0;

frame = 0;


filename = 'input2.avi';

threshold = 0.027;

vidReader = VideoReader(filename);

numFrames = vidReader.NumberOfFrames;

count_cell = zeros(1,numFrames);


FramesX = vidReader.Height;

FramesY = vidReader.Width;


cell_number_X = vidReader.Height/patchsize;

cell_number_Y = vidReader.Width/patchsize;

cell_num = cell_number_X * cell_number_Y;


patcharea = patchsize * patchsize;


increase_threshold = 10;

low_threshold = cell_num * 0.04;


patch_matrix_Y = ones(1,cell_number_Y)*patchsize;

patch_matrix_X = ones(1,cell_number_X)*patchsize;


result = zeros(FramesX,FramesY,'uint8');

result_combine = zeros(FramesX,FramesY,'uint8');


result_split = mat2cell(result,patch_matrix_X,patch_matrix_Y);
```

```
opticFlow = opticalFlowHS('Smoothness',1);


status = zeros(cell_number_X,cell_number_Y,'uint8');
count2 = ones(cell_number_X,cell_number_Y,'uint8');
status_for_edge_history = zeros(cell_number_X,cell_number_Y,'uint8');




for frame = 1:numFrames
    count = count + 1
    frameRGB = read(vidReader, frame);
    frameGray = rgb2gray(frameRGB);
    flow = estimateFlow(opticFlow,frameGray);
    flow_m_split = mat2cell(flow.Magnitude,patch_matrix_X,patch_matrix_Y);



    if(show == 1)
    figure(1),
    imshow(frameGray);
    hold on
    plot(flow,'DecimationFactor',[5 5],'ScaleFactor',25);
    hold off
    end

    for i = 1 : cell_number_X
        for j = 1 : cell_number_Y
            tempave = sum(sum(flow_m_split{i,j} )) / patcharea;
            if(tempave >= threshold)
                result_split{i,j} = uint8(ones(patchsize)*255);
                if(count >2)
                count_cell(count) = count_cell(count) + 1;
                end
            else
```

```
            result_split{i,j} = uint8(zeros(patchsize));
        end
    end
end

if(count > 2)
    if(count > 30 && count_cell(count) - count_cell(count-1) < -3)
        result = cell2mat(result_split)
        figure(3),
        imshow(result_previous);

se = strel('disk',30);
result_previous = imclose(result_previous,se);

figure(4),
imshow(result_previous);
result_split_previous = mat2cell(result_previous,patch_matrix_X,patch_matrix_Y);

[result_edge,thr] = edge(frameGray_previous,'Canny');
trick = edge(rgb2gray(read(vidReader, 1)),'Canny');
figure(10),
imshow(result_edge);
frameGray = rgb2gray(frameRGB);
for i = 1:FramesX
    for j =1:FramesY
        if(result_previous(i,j) == 0 || trick(i,j)==1)
            result_edge(i,j) = 0;
        end
    end
end

figure(5),
imshow(result_edge);
```

```
se = strel('disk',20);
result_edge = imclose(result_edge,se);
figure(6),
imshow(result_edge);

result_fill =  imfill(result_edge,'holes') ;
figure(7),
imshow(result_fill);

se = strel('disk',5);
result_open =  imopen(result_fill,se);
figure(8),
imshow(result_open);

se = strel('disk',2);
result_close =  imclose(result_open,se) ;
figure(9),
imshow(result_close);

imwrite(result_close,strcat('preprocess_input1.png'));
break;

   end
end

if(show == 1)
    figure(2),
    result_combine = cell2mat(result_split);
    result_split_previous = result_split;
    imshow (result_combine);
    result_previous = result_combine;
    frameGray_previous = frameGray;
```

```
    else
        result_combine = cell2mat(result_split);
    end

    imwrite(result_combine,strcat('..\imgdata\',num2str(count),'.png'));
end

myObj = VideoWriter('result6.avi');
writerObj.FrameRate = 30;
open(myObj);
for i=1:count - 1
   fname=strcat('..\imgdata\',num2str(i),'.png');
   frame = imread(fname);
   writeVideo(myObj,frame);
end
close(myObj);
```

**References**

[1] Lifang Wu, et.al, An adaptive approach to human motion tracking from video, Visual Communication and Image Processing, 2010

[2] Chih-Chang Yu, et.al, Automatic human body tracking and modeling from monocular video sequences, IEEE, 2007

[3] Eyup Gedikli, Murat Ekinci, Human Motion Detection, Tracking and Analysis For Automated Surveillance

[4] Robert T. Collins, et.al, A System for Video Surveillance and Monitoring