Annie Bete, Jeff Bonner, and Hannah Johnson
CSE 5524 Computer Vision
Final Project
Nov 26, 2022

Using Computer Vision Techniques to Analyze the Correctness of a Runner's Form

Introduction

A runner's form can be analyzed by looking at their cadence (the number of steps taken in one minute), change in their vertical direction (the displacement of their head over time), and whether they are overstriding. We will film a runner running in a straight line using a stationary camera, so that multiple of their strides are recorded. From there, we will use background subtraction to determine the runner's form each stride and analyze their overall form.

Data Collection

The data was collected using a Sony Alpha 7C camera on a tripod. The runner started running before entering the frame so that they would be at full speed for the duration of the video. The video was cropped so that there was approximately one second before the runner entered the recorded scene and one second after the runner left the recorded scene, to ensure that there were plenty of background frames that could be used for background subtraction.

All of the camera settings were manually set to ensure that the focus and aperture were consistent for the entire video. The video was reshot, analyzed, and the settings were adjusted until the video had a sufficient number of clear frames to use to analyze the runner's form.

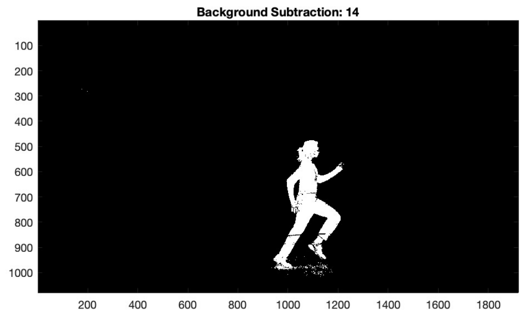
Methodology and Results

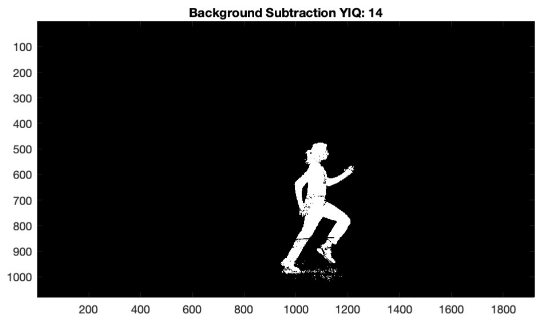
Data Setup

First, the video was loaded into MATLAB. The frames in the first and last seconds of the video were extracted to be the background frames and the rest of the frames contained the runner's run. All frames were vertically cropped to remove extra background. For the run, every other frame was sampled to be processed, resulting in 68 frames. For the background, every fifth frame was sampled, resulting in 24 frames. Image compression was experimented with, but cropping and sampling the frames decreased the runtime enough. It was good that an alternative to image compression was found, since it affects how the motion between each frame is portrayed.

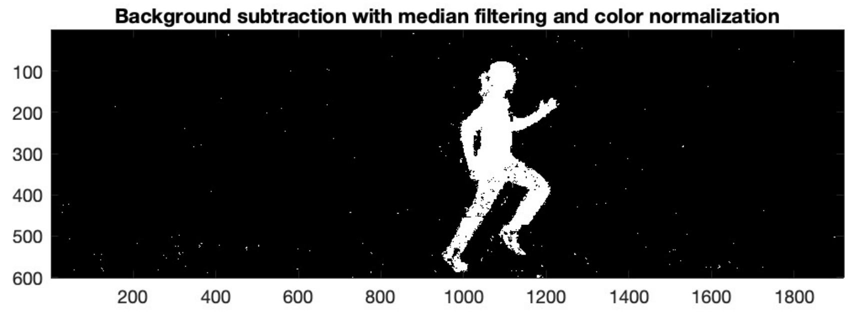
Background Subtraction

The initial background subtraction, shown below, did a very good job of removing the background, with the exception of the reflection/shadow on the floor directly beneath the runner. The YIQ method of focusing more on intensity than on the colors was attempted to improve the results. The initial background subtraction is shown below on the left and the YIQ method shown below on the right. It didn't remove a lot of the noise from the ground.

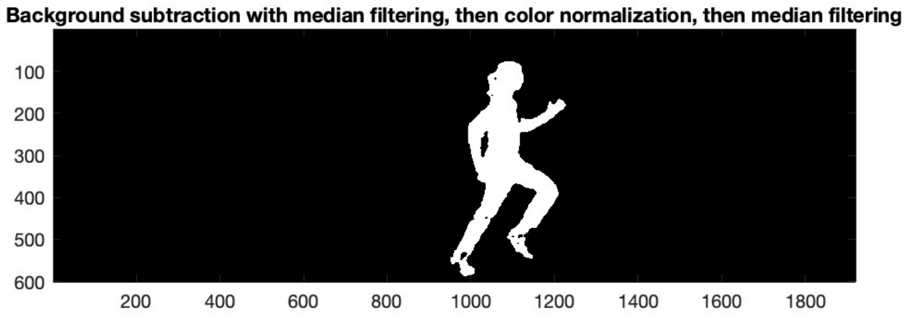




The method that worked the best to remove the noise on the floor was color normalization. Median filtering using a 3x3 filter was used before the color normalization to improve the results. An example frame is shown below.

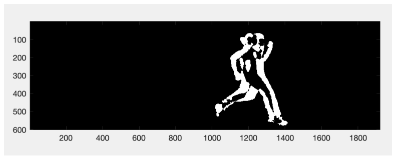

Background subtraction with median filtering and color normalization

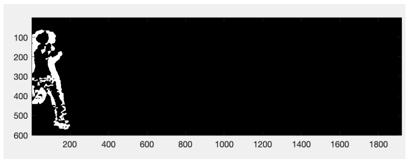
Finally, additional noise was removed using a second 9x9 median filter after the background subtraction and by removing all 8-connected components that had fewer than 500 pixels. An example frame produced from the final background subtraction and noise removal is shown below.

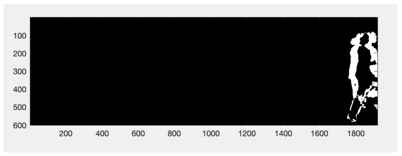

Background subtraction with median filtering, then color normalization, then median filtering

Motion Difference, MEI, and MHI

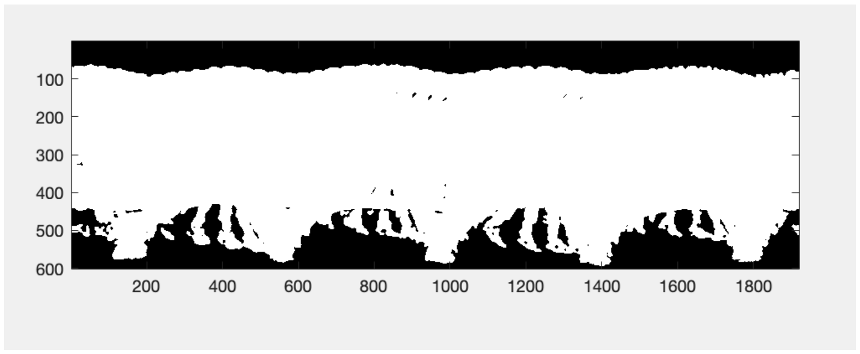
After background subtraction, the binary images were used to get the motion difference images. The ith motion difference image was the absolute value of subtracting the ith frame from the (i-1)th frame. Then it was thresholded and processed for noise removal. bwareaopen() was used to remove regions fewer than 100 pixels with the 8 connected criteria. Then median filtering was used to remove any possible scattered noise. The image processing steps in the motion difference algorithm are not necessary in this application because the frames were already clean from noise after the background subtraction steps, but they were performed just in case. The results are shown below.


imagesc(motionDiff(:,:, 42))

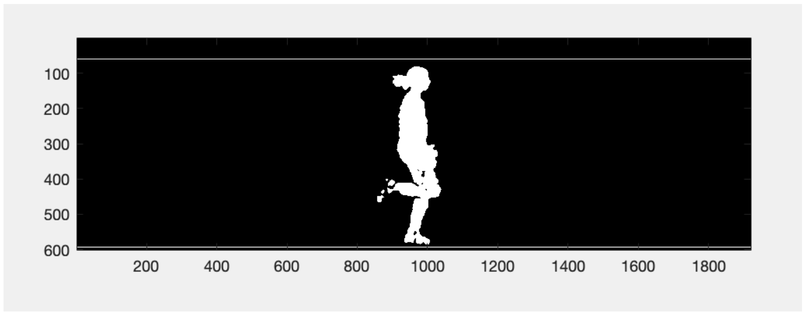

imagesc(motionDiff(:,:, 12))


imagesc(motionDiff(:,:, 58))

Next, all of the motion difference images were used to calculate an overall MEI for the entire running period, shown below.



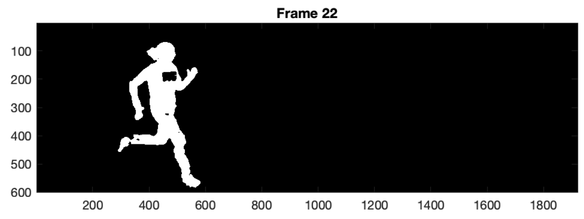
From visually inspecting the MEI, it is clear that the runner took a total of 5 strides in the 2 second interval. The MEI was used to calculate the floor by getting the max row value in the MEI figure. The top most head point was calculated in a similar way by getting the min row value. The background subtracted image with the floor and top head points is shown below.

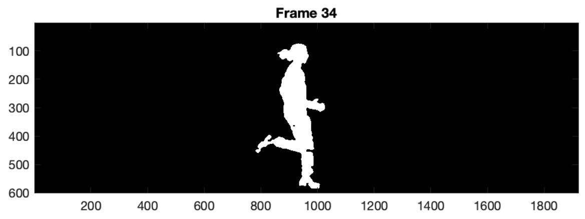


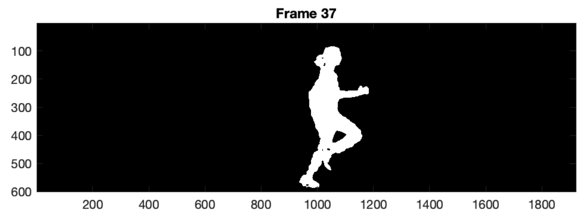
Next, the MEI and floor row was used to find the frames that corresponded to steps. This proved to be much harder than expected. When just the difference between bottom point and floor was thresholded with a value picked by inspection were used to get the step frames, the result was many frames that are close to each other in sections. For example, frames 22, 23, 26, 27, 28, 34, 35, 36, 37, 38, 39, 43, 44, 45, 46, 47, 48, 49, 50, 57, and 59. Thus it was necessary to group the resulting frames into step groups and then take the minimum step frame of each group. To group into step groups I used

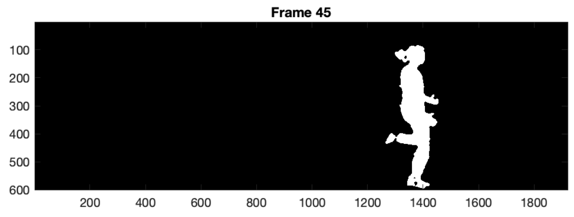
```
[~,~,bins] = histcounts(stepFrameCand,numSteps);
```

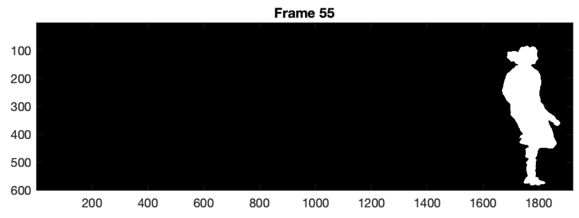
which groups the candidate step frames into a specified number of bins. However, first the number of bins, numSteps, needed to be calculated. numSteps was calculated by counting the number of times the bottom of the figure in the frames crossed bottom 20th of the image. This was achieved by calculating the difference between the bottom of each frame figure and the floor and then counting sign changes and dividing the result by 2. The resulting step frames are shown below.



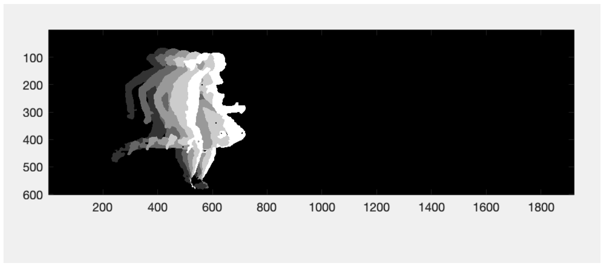
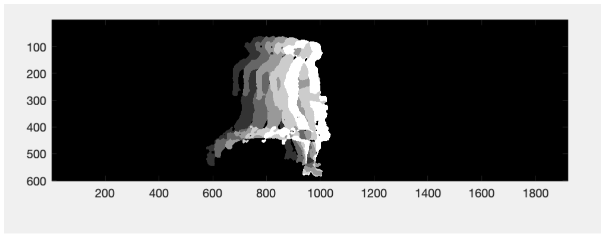
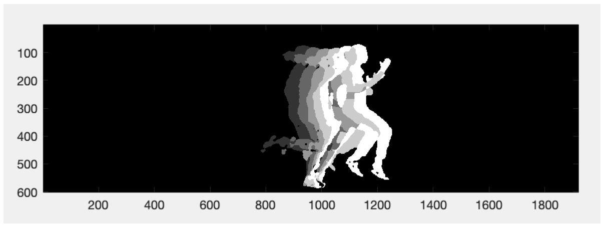
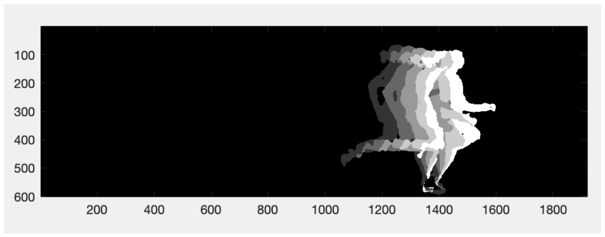
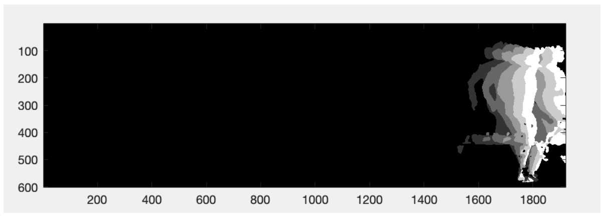




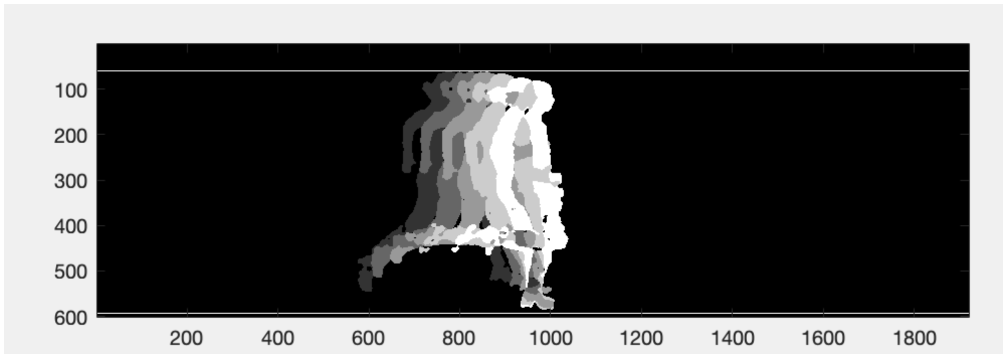




Then the MHI for each step frame was computed based on the previous 5 motion difference image frames before the step frame, shown below.

Finally the MHI for all frames was computed and a video with the head and floor lines was made using implay(). An example of the MHI for frame 35, with the head and floor bars, is shown below.

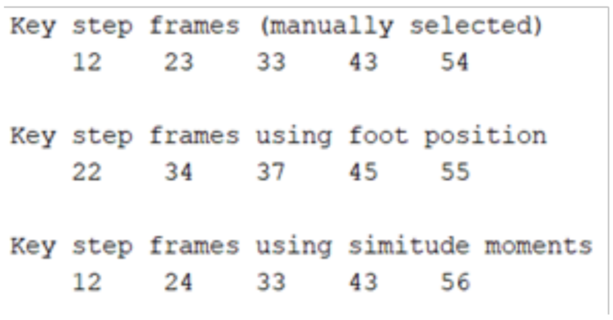


Similitude Moment Analysis

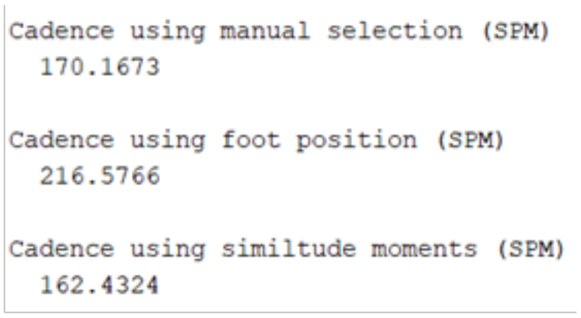
In addition to using the foot placement to determine a step frame, we also manually selected one of the step frames to use as an “ideal” step reference frame for moment analysis. Frame 34 was chosen as this reference frame. The seven similitude moments were calculated for each of the background subtracted frames in the video. The difference was calculated between these moments and the moments of the reference frame. These values were sorted in ascending order. To select the step frames, this list of frame numbers was traversed in ascending order, and each frame was given a buffer of 7 frames on either side so no two frames within 7 frames apart would be chosen (7 frames seemed to give the best results based on the test footage). This buffer ensured that only one frame was chosen in a given “step” that could have multiple frames of the step match up closely. A 10 frame buffer was applied to the start and end of the footage so as to not select frames before the runner is clearly visible in the footage.

The cadence of the runner was calculated by taking the average frame difference between step frames and dividing this value by the framerate of the video in order to figure out how many seconds were in an average stride. This secondsPerStride value was then divided by 2 to get the average stride time just for one foot, and then divided into 60 to get the average number of strides per minute (SPM).

The selected frames based on manual selection, similitude moment analysis, and the foot placement methods can be found below, in addition to their respective extrapolated cadence calculations.

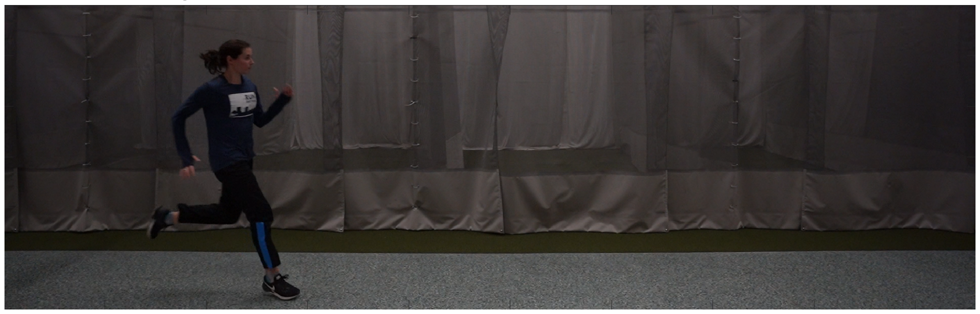


```
Key step frames (manually selected)
    12    23    33    43    54

Key step frames using foot position
    22    34    37    45    55

Key step frames using simitude moments
    12    24    33    43    56
```

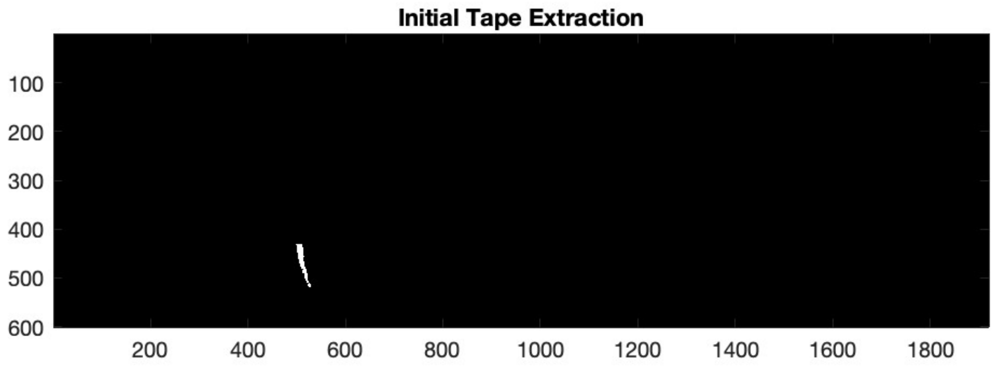


```
Cadence using manual selection (SPM)
  170.1673

Cadence using foot position (SPM)
  216.5766

Cadence using similtude moments (SPM)
  162.4324
```

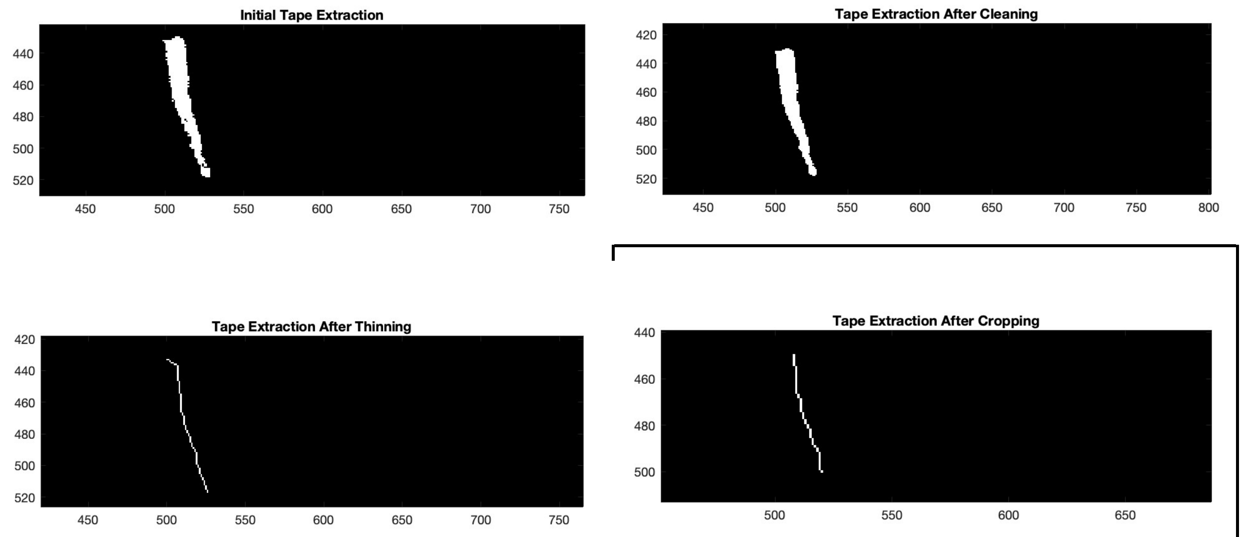
Stride Angle Analysis
The stride angle analysis was aided by a piece of blue tape, shown in the photo below. This piece of tape was applied to the runner's pants between the knee and the ankle to clearly show the angle of the lower leg.



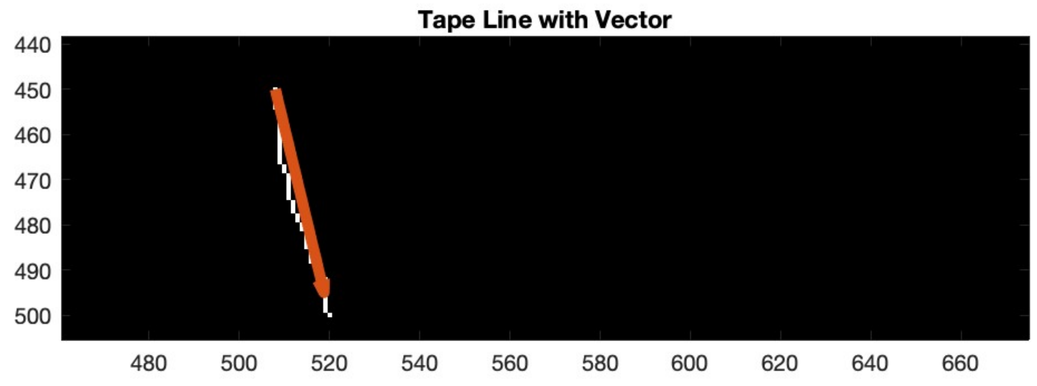
First, the piece of tape was extracted, using simple color thresholding. Since this tape was blue, the color red=20, green=40, blue=90 was used for the base, with a tolerance of 20 on either side of the value. The resulting extraction is shown below.



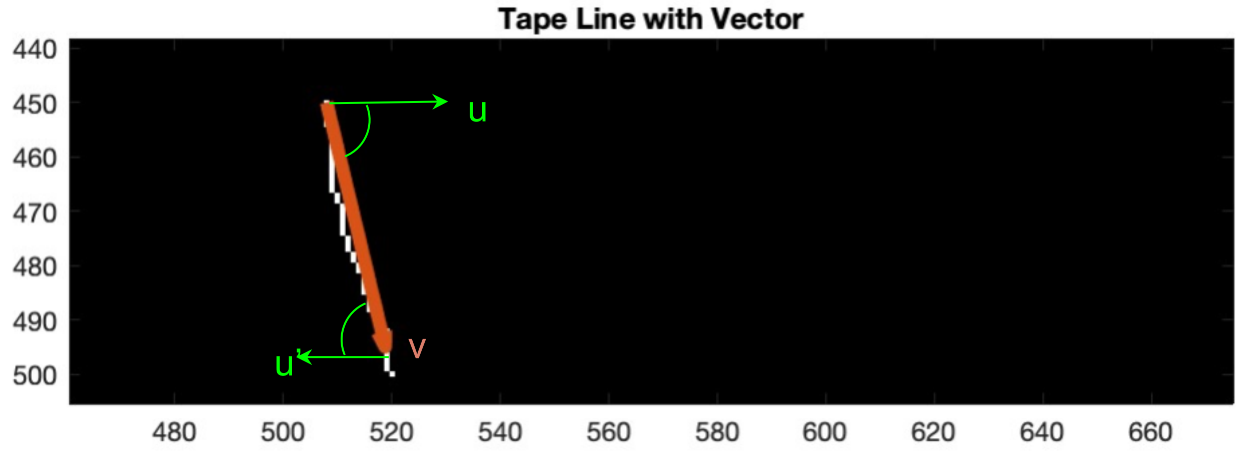
An example of an initial tape extraction is the top left image below. To improve the selection and make it easier to get a line from, the region was first cleaned using a median filter, morphological cleaning, and morphological filling. The resulting image is the top right image below. Next, morphological thinning, which removes the boundary pixels, was repeatedly applied until only a line of pixels remained, as shown in the bottom left image below. Finally, the middle region was extracted, since the top and bottom edges were not always smooth and affected the line. This is the bottom right image shown below.



Next, the vector corresponding to this line is created by taking the top coordinates and the bottom coordinates and connecting them. An example vector is shown below.



In this case, the floor is assumed to be a horizontal vector. Two horizontal vectors are shown on the drawing below. You can see how the angle that the leg is at when it contacts the floor, the angle between u' and v in the drawing below, is the same angle, by vertical angles, as a u and v. Angle u will be used in the calculation, for simplicity, since it has the same origin as v. The angle between two vectors, u and v, can be calculated by $A = cos^{-1}((u \cdot v)/(||u||||v||))$, in radians. Multiplying by 180/pi gives the angle in degrees. The user can enter an angle that would be the cutoff for when a stride starts to overstride. If the angle was 45°, then any angle less than 45° would be overstriding, and any angle greater than 45° would be a good stride.



Discussion

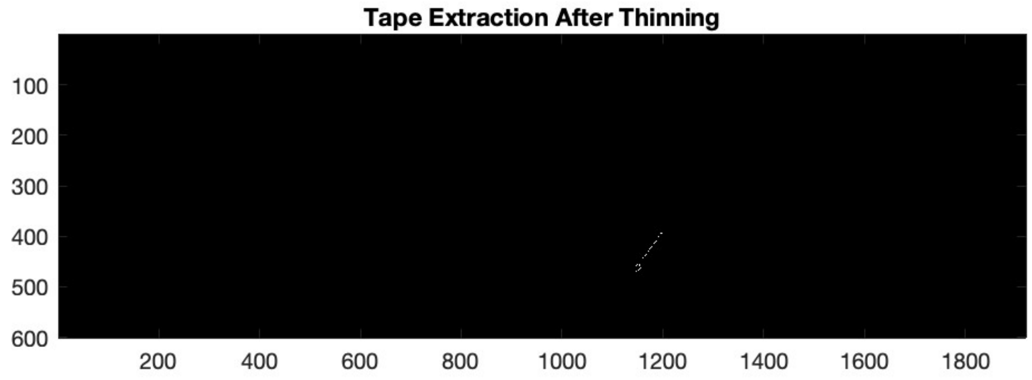
The background subtraction worked very well for this purpose. The top of the head and the lower leg/feet all needed to be clear, but the rest of the body didn't need to be perfect, since it wasn't used in the later algorithms. This means that some things, like the graphic on the runner's shirt being taken out during background subtraction, didn't affect the outcome of the experiments.

The MEI was very useful in visually determining the number of steps the runner took, and the pattern of the runner's gait was very clear. However there was a problem in determining step frames. The issue was that the step frame algorithm can and does pick up on both the end and beginning of a step. In order to fix this, a step frame must be better defined. If we define a step as when the non weight bearing leg is fully bent and raised, then our step frame algorithm had a ⅗ success rate. To improve our results, further computer vision techniques like moment similitude or covariance tracking techniques would need to be applied. We decided to explore moment similitude.

The MHI helps show a continuous representation of the runner's movement just before each step. By looking at these images, the runner's form can be analyzed. In addition, the video that was created out of the MHI's can give further insight on the runner's form. The floor and head line are visual anchors for assessing the runner. Comparing the runner's head with the head line shows the fairly regular periodic behavior of the runner's vertical motion.

The similitude moment analysis for the step frames proved to be very accurate after some tuning of the frame buffer to select only one frame per step, as can be seen by comparing the cadences of each method of step frame extraction. After this fine tuning, the calculated cadence of the similitude moments had a smallder error to the manually selected cadence than the foot placement method. However, this analysis starts with a provided background subtracted step frame, which may not be available in all cases. Combining these two methods to hone in the step frame even closer could serve to be even more accurate than each of the individual efforts taken.

For the overstriding analysis, The method of using the tape was straightforward, but not perfect. In some cases, the jagged top and bottom edges, along with holes in the tape extraction that weren't fixed in cleaning, created odd angles in the final line. An example of this is shown below, where the line splits towards the bottom left side.



The overstriding analysis also depends on the correct step frames being selected. It is very important that the frames being entered into it are the very first frames where the foot makes contact with the floor, since the angle of the leg changes so much while the foot is on the ground.

In the future, we could also check for correct body weight distribution and form as well as calculate the amount of time in contact with the ground. Further analysis into exactly how the foot is contacting the ground would improve the step detection, and thus the overstriding analysis, along with giving more information about the runner's form.

Another way to improve this analysis would be to use different videos shot in different settings. Due to time constraints, this is a proof of concept, but further development with more testing would make this into a robust analysis.

Breakdown of work

Annie recorded the data, created helper functions to transition between 4D data and its color channels, experimented with image compression, performed image preprocessing, completed the background subtraction, and designed the stride analysis algorithm. Hannah computed the total MEI, determined the top most head level, the floor, and the step frames, and computed the

MHI’s. Jeff calculated cadence based on the extrapolated average stride duration and used similitude moments to find an alternative set of step frames to compare against the step frame extraction method Hannah worked on.

Code

main.m

```
% beginning of Annie's additions (part 1)
%% load video and split it into background and running clips
file = 'input/test1_cropped.mp4';
vidObj = VideoReader(file);
beginningBackground = double(read(vidObj,[1 floor(vidObj.FrameRate)]));
run = double(read(vidObj,[floor(vidObj.FrameRate) floor(vidObj.NumFrames -
vidObj.FrameRate)]));
endingBackground = double(read(vidObj,[floor(vidObj.NumFrames - vidObj.FrameRate)
Inf]));
background = cat(4,beginningBackground,endingBackground);
run = run(400:1000, :, :, 1:2:end);
background = background(400:1000, :, :, 1:5:end);
%% Compress image
% [compressedBackground1, compressedBackground2, compressedBackground3] =
image_compression(background);
% [compressedRun1, compressedRun2, compressedRun3] = image_compression(run);
%% Show results of background subtraction with noise removal
t=6;
sub = background_subtraction(background, run, t);
med = medfilt3(sub, [9 9 1]);
med = bwareaopen(med,500,8);
implay(med)
% end of Annie's additions (part 1)
%beginning of hannah additions
%% Getting Mei over all frames and calculating ground and top most head point
%Calculating Motion Difference frames
motionDiff= motionDifference(med);
% getting the Mei and displaying it
mei=MEI(motionDiff); % note the Mei is already normalized by how its computed
imagesc(mei);
axis('image'); % set the ratio so image is not stretched out
colormap("gray");
pause;
%finding the floor and the topmost head point
[nonzeroRows,nonzeroCol]=find(mei==1);
Floor=max(nonzeroRows);
tophead=min(nonzeroRows);
%displaying the floor and topmost head point
floorAndHead=med;
floorAndHead(Floor,:,:)=1;
floorAndHead(tophead,:,:)=1;
imagesc(floorAndHead(:,:,35));
axis('image'); % set the ratio so image is not stretched out
colormap("gray");
pause;
```

```matlab
implay(floorAndHead);
%% finding frames which correspond to steps
stepFrames=findSteps(med,Floor);
[~,numSteps]=size(stepFrames);
%display step frames
for i=1:numSteps
   imagesc(med(:,:,stepFrames(i)));
   axis('image'); % set the ratio so image is not stretched out
   colormap("gray");
   pause;
end
%% Get step frames with Moments to compare accuracy to foot placement frames
sampleStepFrame = med(:,:,33);
Nsample = similitudeMoments(sampleStepFrame);
momentDifferences = [];
for i=1:nFrames
   Nframe = similitudeMoments(med(:,:,i));
   d = Nsample - Nframe;
   momentDifferences(end+1) = sqrt(d * d');
end
%sort differences, and get the sorted original indexes to order the frames
[sortedMomentDifferences, sortedMomentStepFrames] = sort(momentDifferences,'ascend');
%get rid of frames if better frame exists within 3 frames on either side
keyMomentStepFrames = [sortedMomentStepFrames(1)];
for i=1:length(sortedMomentStepFrames)
   goodframe=1;
   for j=1:length(keyMomentStepFrames)
       if(abs(sortedMomentStepFrames(i)-keyMomentStepFrames(j)) < 8 ...
               || sortedMomentStepFrames(i) < 10 ...
               || sortedMomentStepFrames(i) > nFrames-10)
           goodframe=0;
           break;
       end
   end
   if(goodframe)
       keyMomentStepFrames(end+1) = sortedMomentStepFrames(i);
   end
end
sortedKeyMomentStepFrames = sort(keyMomentStepFrames);
disp("Key step frames (manually selected)")
manualSteps =[12,23,33,43,54];
disp(manualSteps)
% fprintf("SSE for foot position technique: %f\n", sum((manualSteps - stepFrames) .^
2) / length(manualSteps))
disp("Key step frames using foot position")
disp(stepFrames)
% fprintf("SSE for similitude moments technique: %f\n", sum((manualSteps -
sortedKeyMomentStepFrames) .^ 2) / length(manualSteps))
```

```
disp("Key step frames using simitude moments")
disp(sortedKeyMomentStepFrames)
%display step frames
for i=1:length(sortedKeyMomentStepFrames)
   imagesc(med(:,:,sortedKeyMomentStepFrames(i)));
   axis('image'); % set the ratio so image is not stretched out
   colormap("gray");
   pause;
end
%% Get cadence based on average frame difference (moments) and framerate
vidObj = VideoReader(file);
avgStrideFrameDifferenceManual = (manualSteps(end) - manualSteps(1)) /
double(numSteps-1);
secondsPerStride = avgStrideFrameDifferenceManual / vidObj.FrameRate;
cadenceFoot = 60.0 / secondsPerStride / 2; % halved because measuring only one leg
disp("Cadence using manual selection (SPM)")
disp(cadenceFoot)
avgStrideFrameDifferenceFoot = (stepFrames(end) - stepFrames(1)) / double(numSteps-
1);
secondsPerStride = avgStrideFrameDifferenceFoot / vidObj.FrameRate;
cadenceFoot = 60.0 / secondsPerStride / 2; % halved because measuring only one leg
disp("Cadence using foot position (SPM)")
disp(cadenceFoot)
avgStrideFrameDifferenceMoment = (sortedKeyMomentStepFrames(end) -
sortedKeyMomentStepFrames(1)) / double(numSteps-1);
secondsPerStride = avgStrideFrameDifferenceMoment / vidObj.FrameRate;
cadenceMoment = 60.0 / secondsPerStride / 2; % halved because measuring only one leg
disp("Cadence using similitude moments (SPM)")
disp(cadenceMoment)
%% Get Mhi for each step frame
[nRow,nCol,nFrames]=size(med);
mhi=zeros(nRow,nCol,numSteps);
for i=1:numSteps
   delta=5; %number of frames before step that are included in mhi
   %compute mhi from all motion difference images upto
   % and including ith step
   mhi(:,:,i)=MHI(motionDiff(:,:,1:stepFrames(i)), delta);
end
%display mhi images
for i=1:numSteps
   imagesc(mhi(:,:,i));
   axis('image'); % set the ratio so image is not stretched out
   colormap("gray");
   pause;
end
%% Mhi for all frames
[nRow,nCol,nFrames]=size(med);
mhi=zeros(nRow,nCol,nFrames);
```

```
for i=1:nFrames
   delta=5; %number of frames before step that are included in mhi
   %compute mhi from all motion difference images upto
   % and including ith step
   mhi(:,:,i)=MHI(motionDiff(:,:,1:i), delta);
end
%% display mhi images
floorAndHeadMHI=mhi;
floorAndHeadMHI(Floor,:,:)=1;
floorAndHeadMHI(top head,:,:)=1;
imagesc(floorAndHeadMHI(:,:,35));
axis('image'); % set the ratio so image is not stretched out
colormap("gray");
pause;
implay(floorAndHeadMHI);
%end of hannah additions
% beginning of Annie's additions (part 2)
%% stride analysis
% get steps (manually)
stepFramesToTest = 20:30;
stepsManualBackgroundSub = med(:,:,stepFramesToTest);
stepsManualColorImage = run(:,:,:,stepFramesToTest);
[red, green, blue] = FourDtoColors(stepsManualColorImage);
red = red.*stepsManualBackgroundSub;
green = green.*stepsManualBackgroundSub;
blue = blue.*stepsManualBackgroundSub;
color_backgroundSubtraction = colorsTo4D(red, green, blue);
for i=1:length(stepFramesToTest)
   line = extractTape(color_backgroundSubtraction(:,:,:,i), 20, 40, 90, 25);
   if sum(sum(line))~=0
       imagesc(line);
       axis('image'); % set the ratio so image is not stretched out
       colormap("gray");
       overstriding = strideAnalysis(line, 45);
   end
   pause
end
% for each example frame, save the extracted line and print stride result
% for i=1:length(stepFramesToTest)
%     line = extractTape(color_backgroundSubtraction(:,:,:,i), 20, 40, 90, 25);
%     if sum(sum(line))~=0
%         overstriding = strideAnalysis(line, 45);
%         imagesc(line);
%         axis('image'); % set the ratio so image is not stretched out
%         colormap("gray");
%         saveas(gca, sprintf('output/strideStep%d.png', stepFramesToTest(i)))
%         sprintf("Frame %d, Overstriding: %d (False=0, True=1)",
stepFramesToTest(i), overstriding)
```

```
%     end
% end
%% Play colored background subtracted video with white background
[r, c, f] = size(med);
med4D = zeros(r,c,3,f);
med4D(:,:,1,:) = med;
med4D(:,:,2,:) = med;
med4D(:,:,3,:) = med;
[red, green, blue] = FourDtoColors(run);
[~,~,~,d] = size(run);
red = red.*med;
green = green.*med;
blue = blue.*med;
colorSum = red + green+ blue;
red(colorSum==0) = 255;
green(colorSum==0) = 255;
blue(colorSum==0) = 255;
color_backgroundSubtraction = colorsTo4D(red, green, blue);
implay(color_backgroundSubtraction./255)
%% Plays the background subtracted video with a red line representing where the tape
was in each frame.
tape = repmat(double(med),[1 1 1 3]);
tape = permute(tape, [1 2 4 3]);
[~,~,~,d] = size(tape);
[red, green, blue] = FourDtoColors(run);
backgroundSub = tape(:,:,1,:);
[r, c, ~, depth] = size(backgroundSub);
backgroundSub=reshape(backgroundSub, [r, c, depth]);
red = red.*backgroundSub;
green = green.*backgroundSub;
blue = blue.*backgroundSub;
color_backgroundSubtraction = colorsTo4D(red, green, blue);
for i=1:d
   line = extractTape(color_backgroundSubtraction(:,:,:,i), 20, 40, 90, 20);
   if sum(sum(line))~=0
       overstriding = strideAnalysis(line, 45);
       [r,c] = find(line==1);
       if overstriding ==1
           sprintf("Frame %d, Overstriding: %d (False=0, True=1)", i, overstriding)
       end
       for j=1:length(r)
           tape(r(j)-2:r(j)+2,c(j)-2:c(j)+2,1,i) = 1;
           tape(r(j)-2:r(j)+2,c(j)-2:c(j)+2,2,i) = 0;
           tape(r(j)-2:r(j)+2,c(j)-2:c(j)+2,3,i) = 0;
       end
   end
end
implay(tape);
```

```
% end of Annie's additions (part 2)
```

background_subtraction.m

```
function [subtracted_image] = background_subtraction(background, run, threshold)
% background_subtraction for a colored video. Applies median filter and
%           color normalization to improve results.
% inputs: foreground video frames, background video frames, threshold for
%           background subtraction
% output: bitmap
   % get separate color channels for the background frames
   [h, w, ~, d] = size(background);
   background_red = reshape(background(:,:,1,:), [h,w,d]);
   background_green = reshape(background(:,:,2,:), [h,w,d]);
   background_blue = reshape(background(:,:,3,:), [h,w,d]);

   % median filtering for the background frames
   background_red = medfilt3(background_red, [3 3 1]);
   background_green = medfilt3(background_green, [3 3 1]);
   background_blue = medfilt3(background_blue, [3 3 1]);
   % normalization of each background color channel
   b_red = background_red./(background_red+background_green+background_blue);
   b_green = background_green./(background_red+background_green+background_blue);
   b_blue = background_blue./(background_red+background_green+background_blue);
   % get separate color channels for the foreground frames
   [h, w, ~, d] = size(run);
   f_red = reshape(run(:,:,1,:), [h,w,d]);
   f_green = reshape(run(:,:,2,:), [h,w,d]);
   f_blue = reshape(run(:,:,3,:), [h,w,d]);
   % median filtering for the foreground frames
   f_red = medfilt3(f_red, [3 3 1]);
   f_green = medfilt3(f_green, [3 3 1]);
   f_blue = medfilt3(f_blue, [3 3 1]);
   % normalization of each foreground color channel
   fg_red = f_red./(f_red+f_green+f_blue);
   fg_green = f_green./(f_red+f_green+f_blue);
   fg_blue = f_blue./(f_red+f_green+f_blue);
   % get mean and standard deviation for each background color channel
   sigma_red = std(b_red, 0, 3);
   u_red = mean(b_red, 3);
   sigma_green = std(b_green, 0, 3);
   u_green = mean(b_green, 3);
   sigma_blue = std(b_blue, 0, 3);
   u_blue = mean(b_blue, 3);
   % perform background subtraction
   red_subtracted = ((fg_red - u_red).^2./(sigma_red.^2)) > threshold^2;
   green_subtracted = ((fg_green - u_green).^2./(sigma_green.^2)) > threshold^2;
   blue_subtracted = ((fg_blue - u_blue).^2./(sigma_blue.^2)) > threshold^2;
   % OR all of the color channels to get the logical bitmap
   subtracted_image=red_subtracted|green_subtracted|blue_subtracted;
end
```

colorsTo4D.m

```
function [FourD] = colorsTo4D(red,green, blue)
% Given three color channels, colorsTo4D arranges the colors in one array
% such that the dimensions are (width, height, channels, # frames).
   FourD = cat(4, red, green, blue);
   FourD = permute(FourD, [1 2 4 3]);
end
```

extractTape.m

```
function [tape] = extractTape(image, r, g, b, threshold)
% extractTape extracts the (r,g,b) +- threshold values from an image and
% returns the bitmap that corresponds to a line representing the angle of
% the tape.
   % get separate color channels
   [red, green, blue] = FourDtoColors(image);

   % extract the colors of the tape, use median filtering, and clean/fill the
resulting shape.
   tape = (red>=r-threshold) & (red<=r+threshold) & (green>=g-threshold) &
(green<=g+threshold) & (blue>=b-threshold) & (blue<=b+threshold);
   tape = medfilt2(tape);
   tape = bwmorph(tape,'clean');
   tape = bwmorph(tape,'fill');

   % keep reducing the tape shape until it is just a line
   line = bwmorph(tape, 'thin');

   while ~isequal(line, tape)
       tape = line;
       line = bwmorph(tape, 'thin');
   end
   tape = line;

   % extract the middle section of the line, since the top and bottom edges
   % sometimes aren't perfect.
   [rows, ~] = find(tape==max(max(tape)));
   maxRow=max(rows);
   minRow = min(rows);
   tape(minRow:minRow+floor(.2*(maxRow-minRow)), :) = 0;
   tape(maxRow-floor(.2*(maxRow-minRow)):maxRow, :) = 0;
end
```

findSteps.m

```
function  stepFrames= findSteps(images, floorRow)
    %returns an array with the frames that correspond to a step by finding
    %the ones that have bottom closest to floor
     [nrows,~,numIm]=size(images);
     bottoms(1:numIm)=0;
     frames=1:numIm;
     %get bottoms for all frames- if frame is empty sets bottom to 0
     for i=1:numIm
       if(sum(images(:,:,i),"all")==0)
       bottoms(i)=0;
       else
       [nonzeroRows,~]=find(images(:,:,i)==1);
       bottoms(i)=max(nonzeroRows);
       end
     end
     %get number of steps by counting number of times bottom crosses
     %bottom 20th of the picture, do this by counting sign changes
     % (zero is counted as a sign change)
     bottomSec=floor(nrows*19/20);
     diff=bottomSec-bottoms;
     count=0;
     for i=1:numIm-1
         if sign(diff(i))*sign(diff(i+1)) ~= 1
             count=count+1;
         end
     end
     numSteps=floor(count/2);
     %difference between bottom and floor
     diff=abs(floorRow-bottoms);
     %threshholding
     threshhold=10 ;
     stepFrameCand=frames(diff<=threshhold);
     %seperate stepframe candidate into bins as multiple frames in same
     %section of video might be step frames
     [~,~,bins] = histcounts(stepFrameCand,numSteps);
     stepFrames(1:numSteps)=0;
     for i=1:numSteps
       stepFrames(i)=floor(median(stepFrameCand(bins==i)));
     end

end
%[N,edges] = histcounts(X) partitions the X values into bins, and returns
% the count in each bin, as well as the bin edges. The histcounts function
% uses an automatic binning algorithm that returns bins with a uniform
% width, chosen to cover the range of elements in X and reveal the
% underlying shape of the distribution.
```

```
%[N,edges,bin] = histcounts(___) also returns an index array, bin, using
% any of the previous syntaxes. bin is an array of the same size as X
% whose elements are the bin indices for the corresponding elements in X.
% The number of elements in the kth bin is nnz(bin==k), which is the same
% as N(k).
```

FourDtoColors.m

```matlab
function [red, green, blue] = FourDtoColors(image)
% Given a 4D matrix of images in the form (width, height, colors, frames),
% it extracts the color channels red, green, blue to be in the form (width,
% height, frame).
   [h, w, ~, d] = size(image);
   red = reshape(image(:,:,1,:), [h,w,d]);
   green = reshape(image(:,:,2,:), [h,w,d]);
   blue = reshape(image(:,:,3,:), [h,w,d]);
end
```

MEI.m

```matlab
function  mei= MEI(diffImages)
    %returns an 2d matrix with the area of motion turned on with value 1
    %and the rest of the pixels turned off
    % mathematically we are unioning over all difference images, so if a
    % pixel is turned on in any image differences then it is turned on for
    % the mei
    % We can fake this by adding element wise and then thresholding
    [~, ~, numIm]=size(diffImages);
    mei=diffImages(:,:,1);
    for i=2:numIm
       mei=mei+diffImages(:,:,i);
    end
    %thresholding
    mei=mei>0;
end
```

MHI.m

```
function  mhi= MHI(diffImages, delta)
    %returns an 2d matrix with a gradient motion where white represents
    %most recent motion and black represents no motion
    %delta is how many frames before matters
    [~, ~, numIm]=size(diffImages);
    diffImages=double(diffImages);
    mhi=double(diffImages(:,:,1));
    minFrameNum=2;
    for i=minFrameNum:numIm
        [rowNonzero, colNonzero]=find(diffImages(:,:,i));
        for j= 1:size(rowNonzero)
           mhi(rowNonzero(j),colNonzero(j))=i;
        end
        [rowTooOld, colTooOld]=find(mhi<i-delta);
        for j= 1:size(rowTooOld)
           mhi(rowTooOld(j),colTooOld(j))=0;
        end
    end
    %normalize the MHI
    mhi=max(0,(double(mhi-(numIm-delta)))/double(delta));
end
```

motionDifference.m

```
function  motionDiff= motionDifference(images)
    %returns an 3d matrix with the difference between the i and i-1 image
    % at position 1 there is a zero matrix
    %at postion 2 there is the difference between the 1st and 2nd image
     motionDiff(:,:,1)=zeros(size(images(:,:,1)));
     [~,~,numIm]=size(images);
     for i=2:numIm
       motionDiff(:,:,i)=abs(images(:,:,i-1)-images(:,:,i));
     end
     %thresholding
     threshhold=0.15 ;
       motionDiff= motionDiff>=threshhold;
       %performing noise removal
       for i=2:numIm
          % Removes regions fewer than 100 pixels (8-connected)
           motionDiff(:,:,i)=bwareaopen(motionDiff(:,:,i), 100, 8);
           motionDiff(:,:,i)=medfilt2(motionDiff(:,:,i));
       end

end
```

strideAnalysis.m

```matlab
function [overstriding] = strideAnalysis(line, angle)
% strideAnalysis is given a line (corresponding to the tape) and a
% threshold angle of when to count an angle as overstriding. Outputs 0 for
% not overstriding and 1 for overstriding.
   % get coordinates from the input line. Top left and bottom right
   % corners.
   [r, c] = find(line==1);
   coords = [r,c];
   lineStart = coords(1,:);
   lineEnd = coords(end,:);

   % get vectors u and v. Currently, the ground is assumed to be
   % horizontal
   u = [lineEnd(2)-lineStart(2), (lineEnd(1)-lineStart(1))];
   v=[20,0];

   % hold on
   % plot(lineStart(2),lineStart(1), 'or')
   % plot(lineEnd(2),lineEnd(1), 'or')
   % quiver(lineStart(2),lineStart(1),lineEnd(2)-lineStart(2),lineEnd(1)-
lineStart(1))
   % get angle between u and v
   A = acos(dot(u,v)/(norm(u)*norm(v)))*180/pi;

   % determine if overstriding based on the angle
   if A <= angle
       overstriding = 1;
   else
       overstriding = 0;
   end
end
```