Assignment 2

Video Stream Analytics

Li Pei, AndrewID: lip
Shuting Xi, AndrewID: sxi

Optical Flow and Color Heat Map

1. Optical Flow

There are two algorithms to implement optical flow, Horn-Schunck Algorithm and Lucas-Kanade Algorithm. We searched both of these two algorithms via the Internet and found several different versions of codes. After running these codes and comparing their performance with those of Matlab's Simulink and its function's sample code, we decided to use Matlab's vision package according to demos from Mathwork to visualize the optical flow with arrow map. With a reference to Matlab's sample code, our revised code for this problem is listed below. In order to have better and more obvious performance, we subtract the backgrounds of these videos and use their foregrounds as our video inputs, using Approximate Median algorithm.

```
converter = vision.ImageDataTypeConverter;
shapeInserter = vision.ShapeInserter('Shape','Lines','BorderColor','Custom',
'CustomBorderColor', 255);
opticalFlow = vision.OpticalFlow('ReferenceFrameDelay', 1);
opticalFlow.OutputValue = 'Horizontal and vertical components in complex
form';

input_video_gray1 = 'N-52.3-1.avi';
videoReader1 =
vision.VideoFileReader(input_video_gray1,'ImageColorSpace','Intensity','Vide
oOutputDataType','uint8');
input_video_gray2 = 'N-52.3-2.avi';
videoReader2 =
vision.VideoFileReader(input_video_gray2,'ImageColorSpace','Intensity','Vide
oOutputDataType','uint8');
input_video_gray3 = 'S-52.9-1.avi';
videoReader3 =
vision.VideoFileReader(input_video_gray3,'ImageColorSpace','Intensity','Vide
oOutputDataType','uint8');
input_video_gray4 = 'S-52.9-2.avi';
videoReader4 =
vision.VideoFileReader(input_video_gray4,'ImageColorSpace','Intensity','Vide
oOutputDataType','uint8');
input_video_gray5 = 'N-53.4-1.avi';
videoReader5 =
vision.VideoFileReader(input_video_gray5,'ImageColorSpace','Intensity','Vide
oOutputDataType','uint8');
input_video_gray6 = 'N-53.4-2.avi';
videoReader6 =
vision.VideoFileReader(input_video_gray6,'ImageColorSpace','Intensity','Vide
oOutputDataType','uint8');

videoPlayer = vision.VideoPlayer('Name','Motion Vector');

% t = 0; % set the stop time
while ~isDone(videoReader1)
%   t = t+1;
    frame1 = step(videoReader1);
    frame2 = step(videoReader2);
    frame3 = step(videoReader3);
    frame4 = step(videoReader4);
    frame5 = step(videoReader5);
    frame6 = step(videoReader6);

    im1 = step(converter, frame1);
    im2 = step(converter, frame2);
    im3 = step(converter, frame3);
    im4 = step(converter, frame4);
    im5 = step(converter, frame5);
    im6 = step(converter, frame6);

    of1 = step(opticalFlow, im1);
    of2 = step(opticalFlow, im2);
    of3 = step(opticalFlow, im3);
    of4 = step(opticalFlow, im4);
    of5 = step(opticalFlow, im5);
    of6 = step(opticalFlow, im6);

    lines1 = videooptflowlines(of1, 20);
    lines2 = videooptflowlines(of2, 20);
    lines3 = videooptflowlines(of3, 20);
    lines4 = videooptflowlines(of4, 20);
    lines5 = videooptflowlines(of5, 20);
    lines6 = videooptflowlines(of6, 20);

    if ~isempty(lines1)
        out1 =  step(shapeInserter, im1, lines1);
        out2 =  step(shapeInserter, im2, lines2);
        out3 =  step(shapeInserter, im3, lines3);
        out4 =  step(shapeInserter, im4, lines4);
        out5 =  step(shapeInserter, im5, lines5);
        out6 =  step(shapeInserter, im6, lines6);

        figure(1),subplot(3,2,1),imshow(out1); title('N-52.3-1.avi');
        subplot(3,2,2),imshow(out2); title('N-52.3-2.avi');
        subplot(3,2,3),imshow(out3); title('S-52.9-1.avi');
        subplot(3,2,4),imshow(out4); title('S-52.9-2.avi');
        subplot(3,2,5),imshow(out5); title('N-53.4-1.avi');
        subplot(3,2,6),imshow(out6); title('N-53.4-2.avi');
    end
%     if t == 50
%          break;
%     end
end
%Close the video reader and player

release(videoPlayer);
release(videoReader);
```

We put all six videos' optical flow together as one output in this part. We also write a code for only one video output. Since the method is the same, we put this more complicated version here and leave out the simpler one. It is worth mentioning that we align the videos according to the real positions of cameras. The output result of our code, from right-to-left, top-to-bottom, are representing camera position from north to south.

Here are some screenshots of out optical flow implementation.

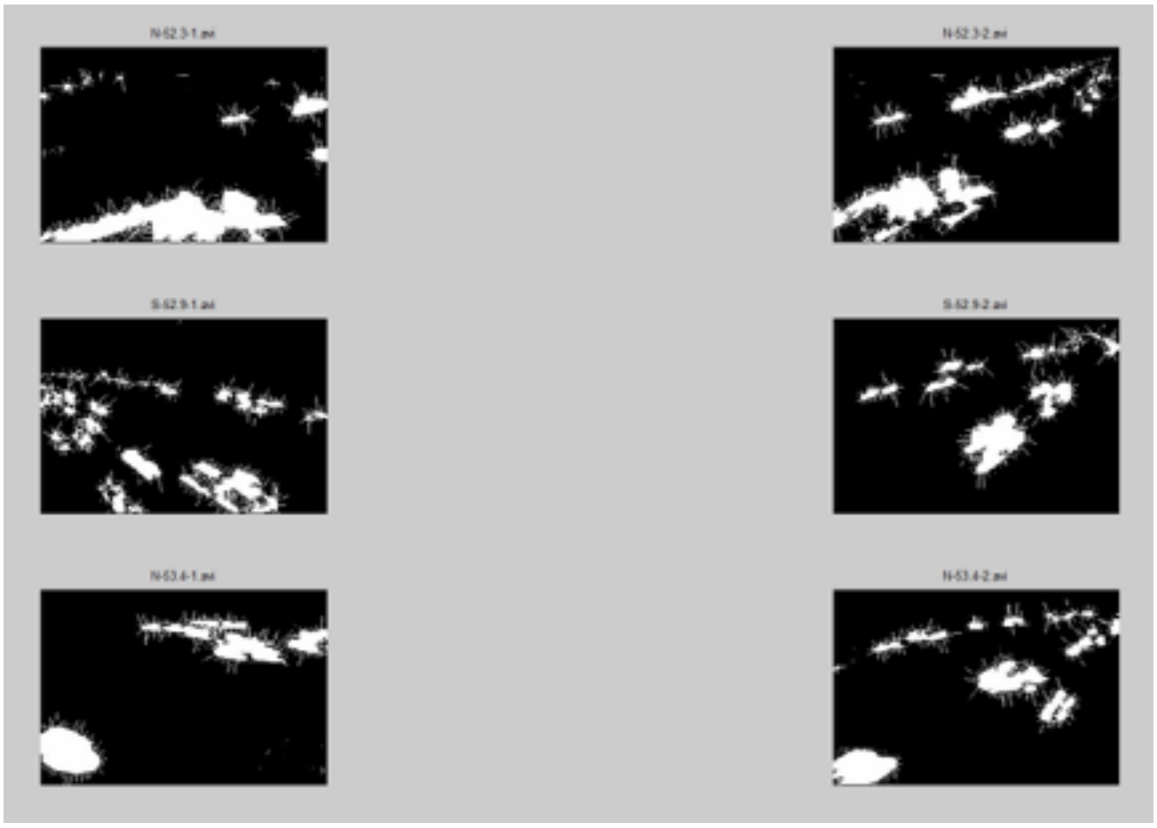


Fig 1.11 Optical flow of all six videos (frame 50)

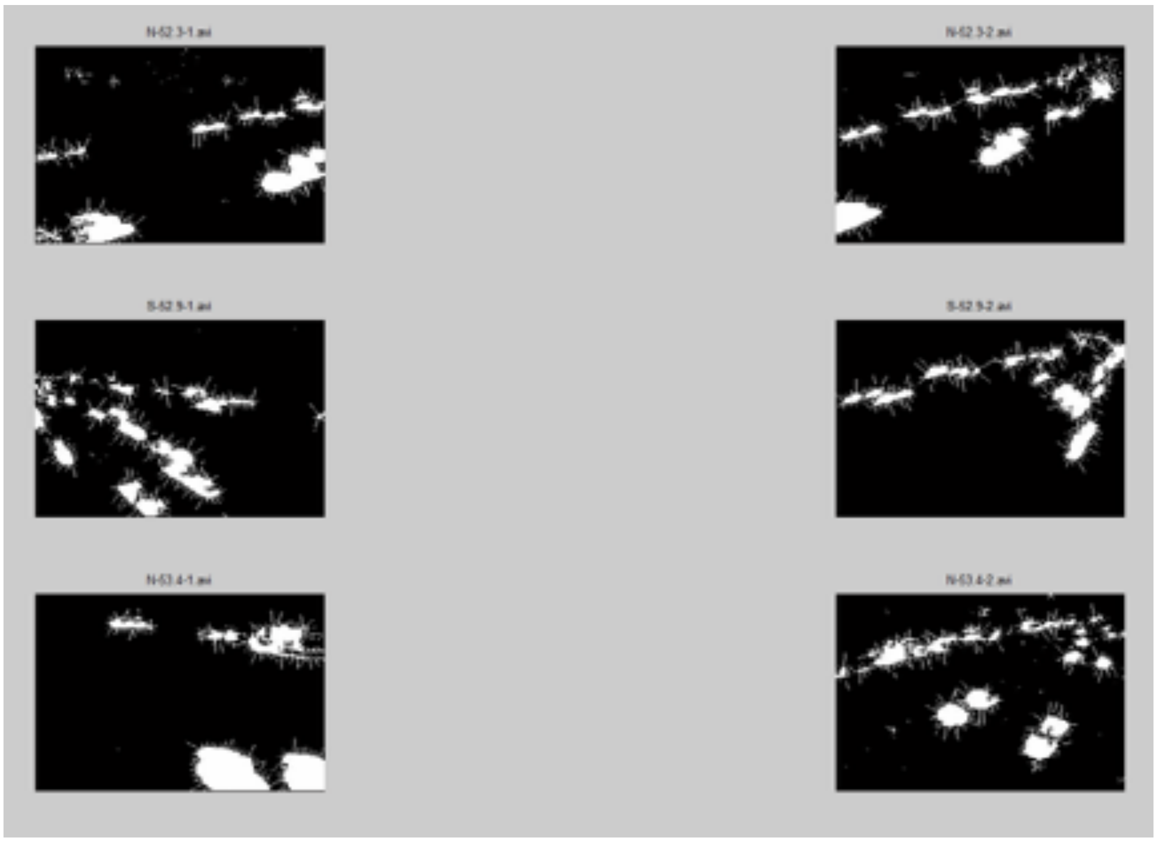


Fig 1.12 Optical flow of all six videos (frame 100)

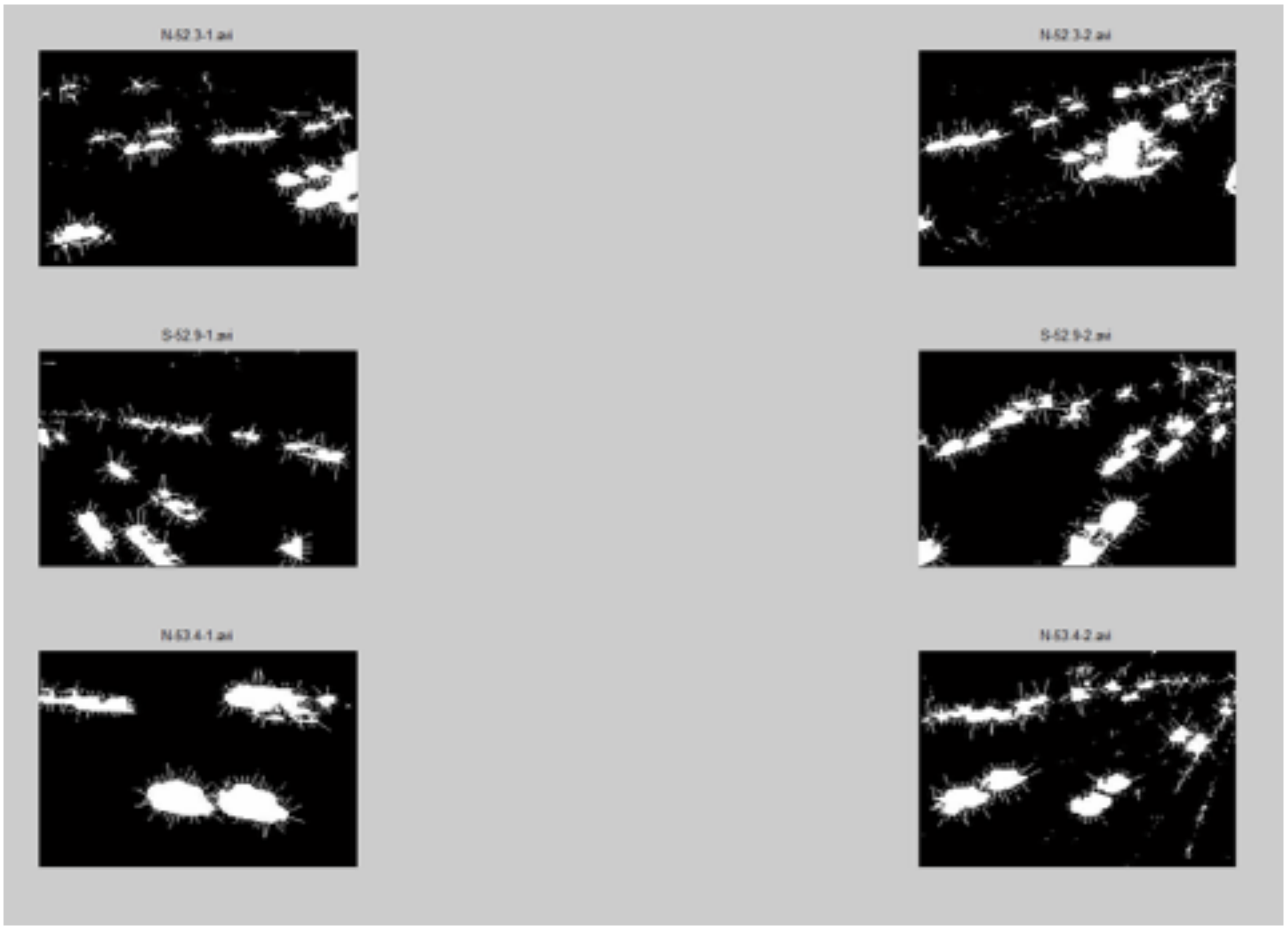


Fig 1.13 Optical flow of all six videos (frame 200)

2. Color Heat Map

As for the heat map, we used the code for assignment 1 directly. Since both of us have already wrote reports about this method, we simply posted the source code and the screenshots of its results here without further analysis. Here is the code.

```
%Removing background first first
%Accumulate the color in heat map
%Cover the color on original frame
%---------read in video and get the attributes-------------
inputObj = VideoReader('AID-495-N-52.3-1.mp4');
nFrames = inputObj.NumberOfFrames;% Total number of frames


backGround = read(inputObj,1); % read in 1st frame as background frame
backGround_Gray = rgb2gray(backGround);


[height,width,d] = size(backGround);%get the size of each frame

%-------------------------processing!---------------------------
threshold =25;
foreGround = zeros (height, width);

for k = 1:nFrames
    inputFrame = read(inputObj, k);
    inputGray = rgb2gray(inputFrame);
    frameDiff = abs(double(inputGray) - double(backGround_Gray));
    for i=1:width
        for j=1:height
            if ((frameDiff(j,i) > threshold))
                foreGround(j,i) = foreGround(j,i)+8;
            end
        end
    end
    backGround_Gray = inputGray;
    figure(1)
    imshow(inputFrame)
    hold on
    heat=imagesc(foreGround,[0,255]); %turn double value into color
    set(heat,'AlphaData', 0.5);
    imagesc(heat);
    hold off
end
```

Actually, we put the heat map of all six videos together as one output too. However, since we just repeat every line of this code for six times, we subtract the main code here instead of posting the complete version of our code. The more sophisticated one can be found in our uploading files.

Here are some screenshots of the heat map results. We also put these six videos according to the cameras' real positions in the road.

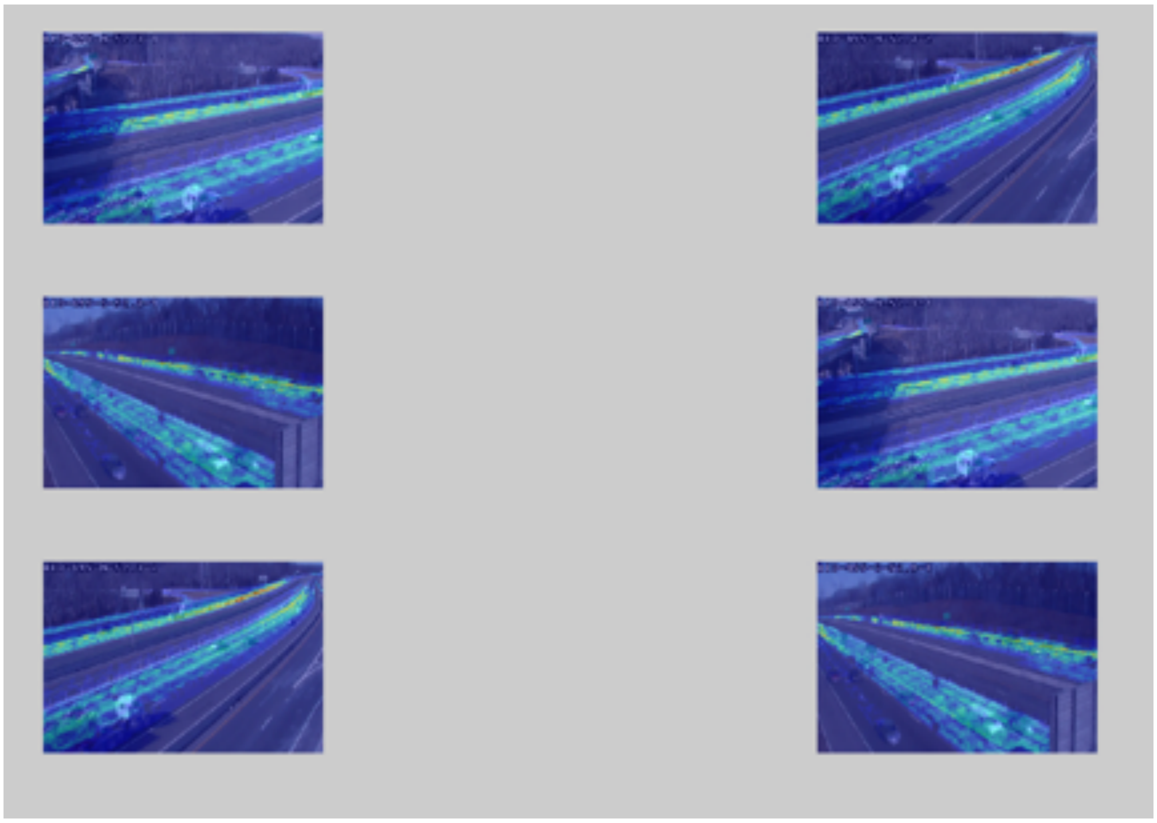


Fig 1.21 Heat map of all six videos (frame 50)

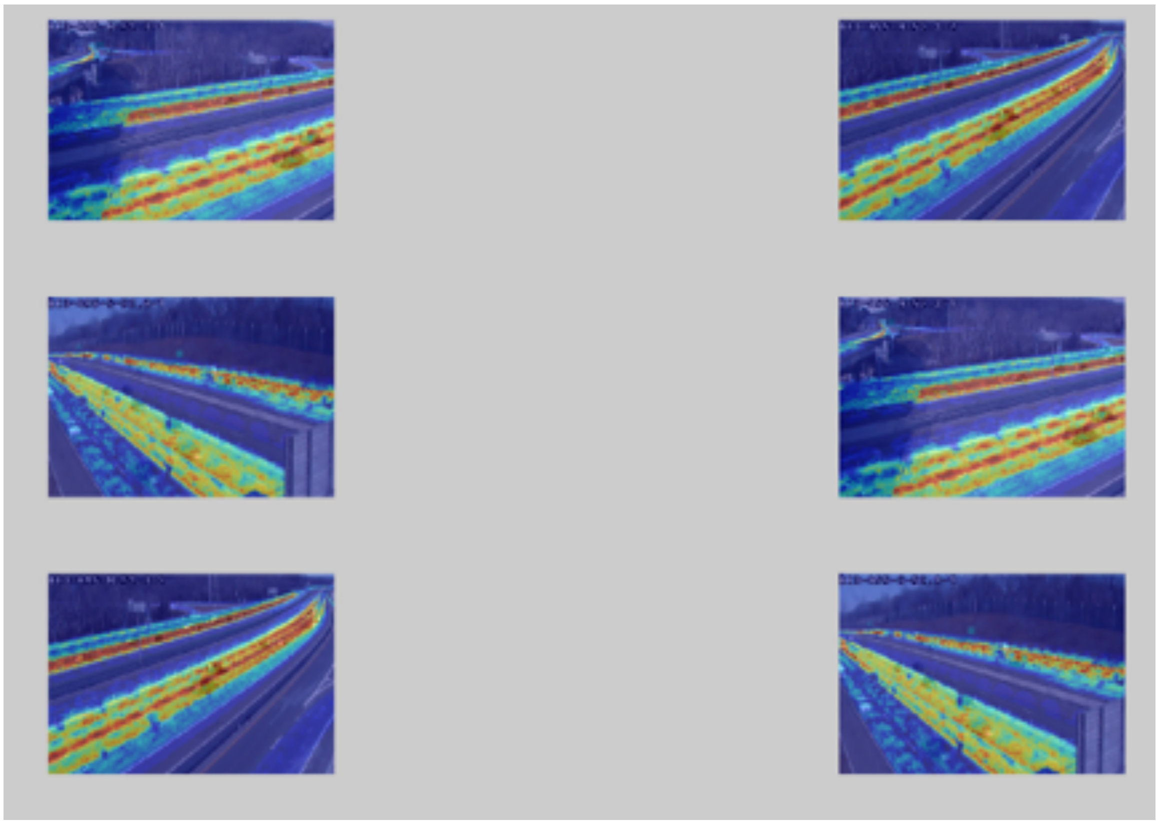


Fig 1.22 Heat map of all six videos (frame 100)

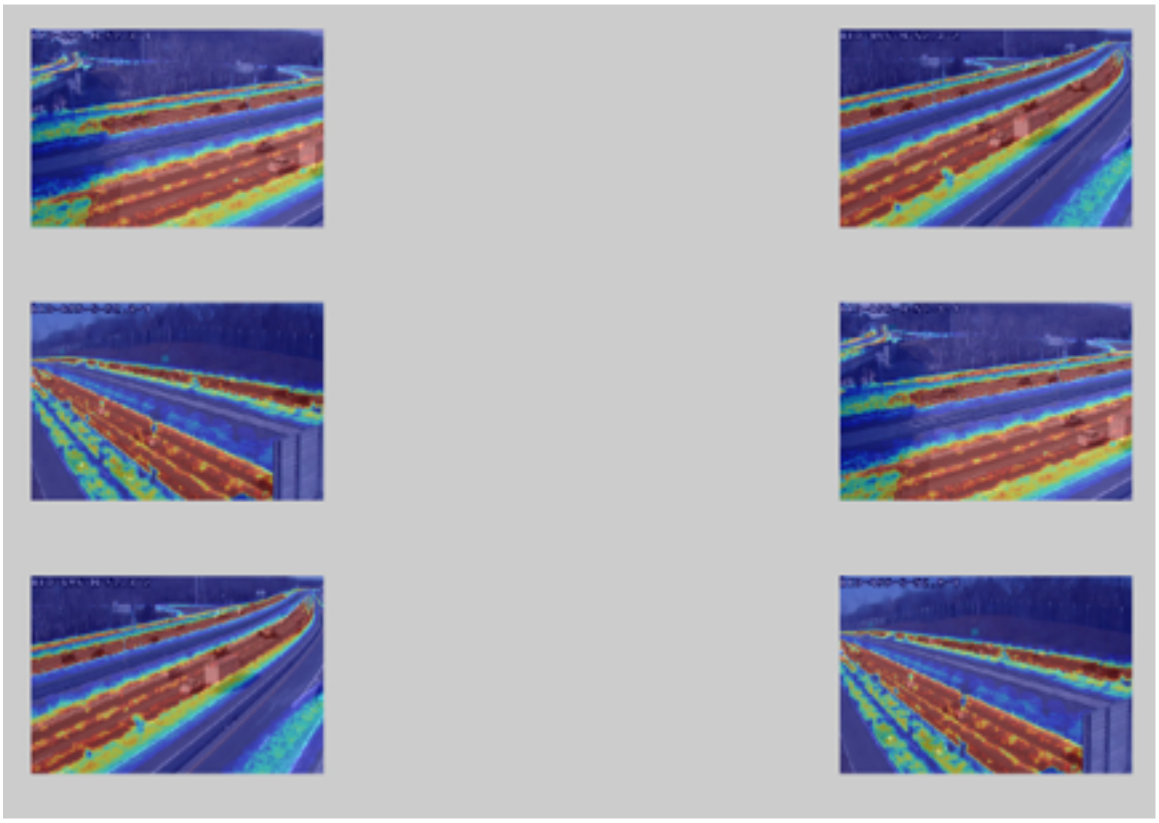


Fig 1.23 Heat map of all six videos (frame 200)

Count Vehicles

At first, we tried to subtract the background and extract the foreground of the video and then use the blob technique to find each vehicle and then count them. However, it works well for some frames, but has bad performance on others.

So we use Matlab's Blob Analysis Block to count cars. With the port BBox, a M-by-4 matrix of [x y width height] bounding box coordinates, we can know the number of blobs, the width, height and the upper left corner of the bounding box, which means we can count vehicles directly according to the output of this port and track them in each frame. It's worth mentioning that in order to count only one way vehicles, we set a bound to BBox's output. When the x or y is out of that boundary, we simply don't count the cars.

Here is our implementation code.

```
foregroundDetector = vision.ForegroundDetector('NumGaussians', 10, ...
    'NumTrainingFrames', 50);

inputObj = VideoReader('N-52.3-1.avi');
nFrames = inputObj.NumberOfFrames;

videoReader = vision.VideoFileReader('AID-495-N-52.3-1.mp4');
blobAnalysis = vision.BlobAnalysis('BoundingBoxOutputPort', true, ...
    'AreaOutputPort', false, 'CentroidOutputPort', false, ...
    'MinimumBlobArea', 100);
for i = 1:nFrames
    frame = step(videoReader); % read the next video frame
    foreground = step(foregroundDetector, frame);
    se = strel('square', 5);
    filteredForeground = imopen(foreground, se);

    bbox = step(blobAnalysis, filteredForeground);
    n = 0;
      for j = 1 : size(bbox,1)
         if(bbox(j,2) < 120)
             result = insertShape(frame, 'Rectangle', bbox, 'Color', 'red');
             n = n + 1;
         end
      end
    result = insertText(result, [10 10], n, 'BoxOpacity', 1, ...
            'FontSize', 14);

    figure(1); imshow(result); title('Clean Foreground');
end

release(videoReader); % close the video file
```

The screenshots of each video are as follows. In these two videos, we count the number of vehicles on the upper line. However, just to clarify, we track cars on both lines with red bounding box. When we tried to only track cars on the upper line, there was always an error and we couldn't run our code somehow.

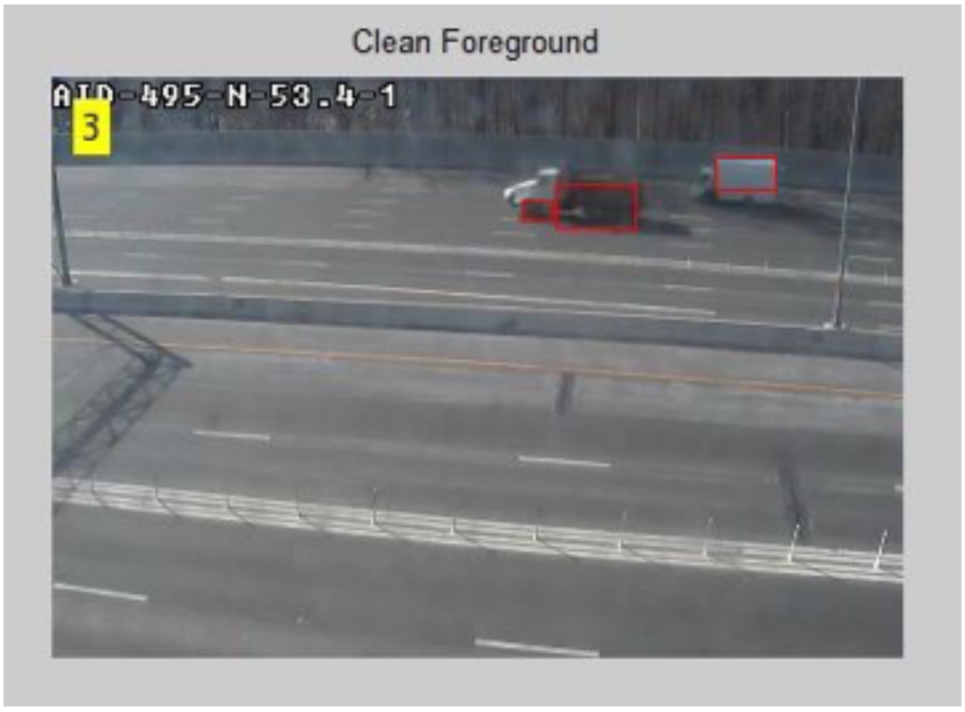


Fig 2.1 Counting vehicles in video AID-495-N-53.4-1 (Frame 15)

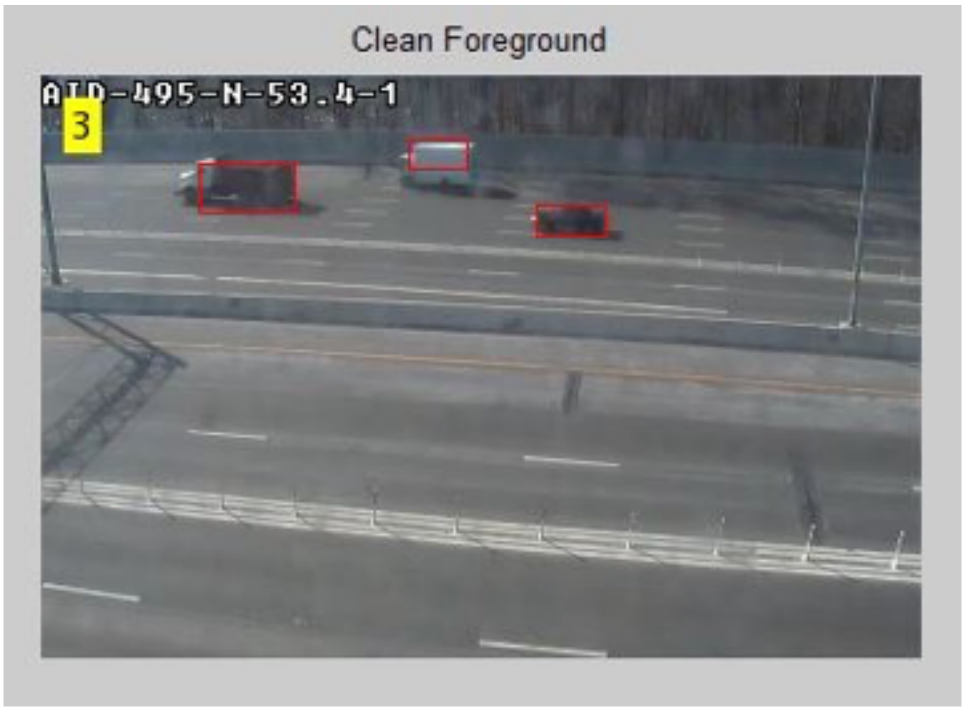


Fig 2.2 Counting vehicles in video AID-495-N-53.4-1 (Frame 20)

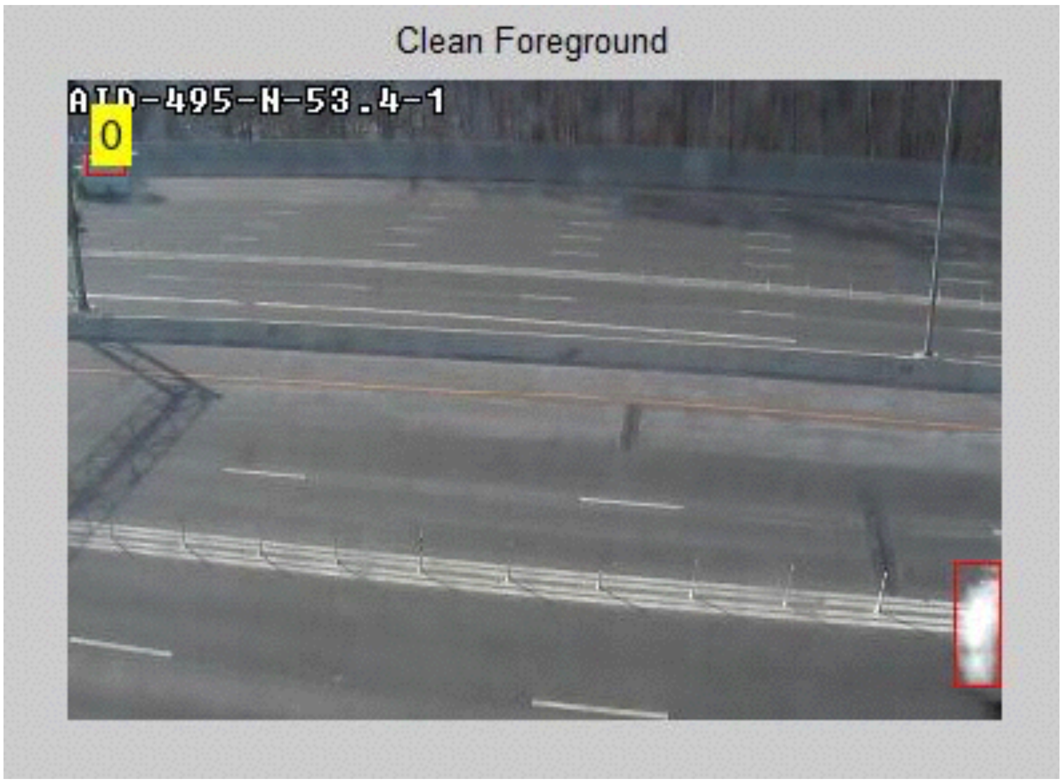


Fig 2.3 Counting vehicles in video AID-495-N-53.4-1 (Frame 30)

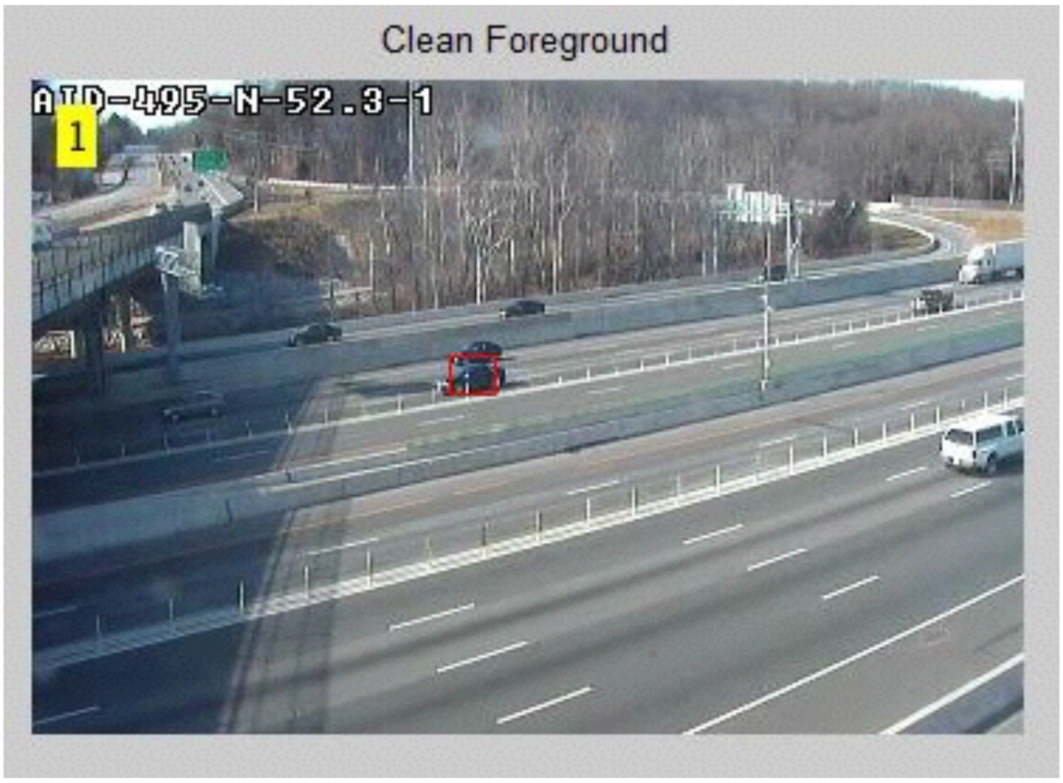


Fig 2.4 Counting vehicles in video AID-495-N-52.3-1 (Frame 15)

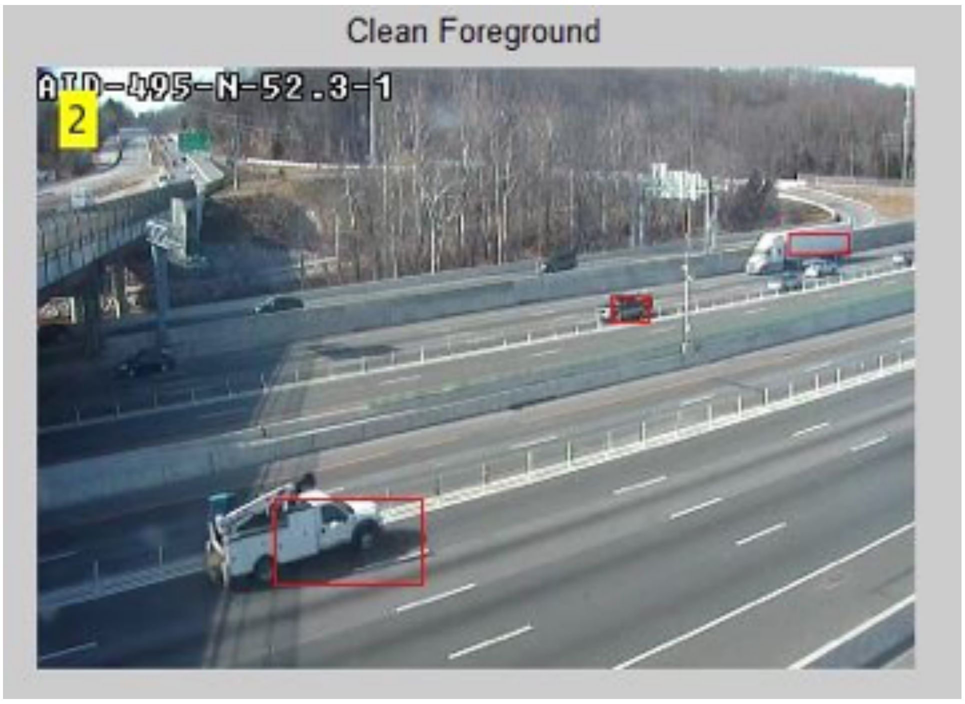


Fig 2.5 Counting vehicles in video AID-495-N-52.3-1 (Frame 20)

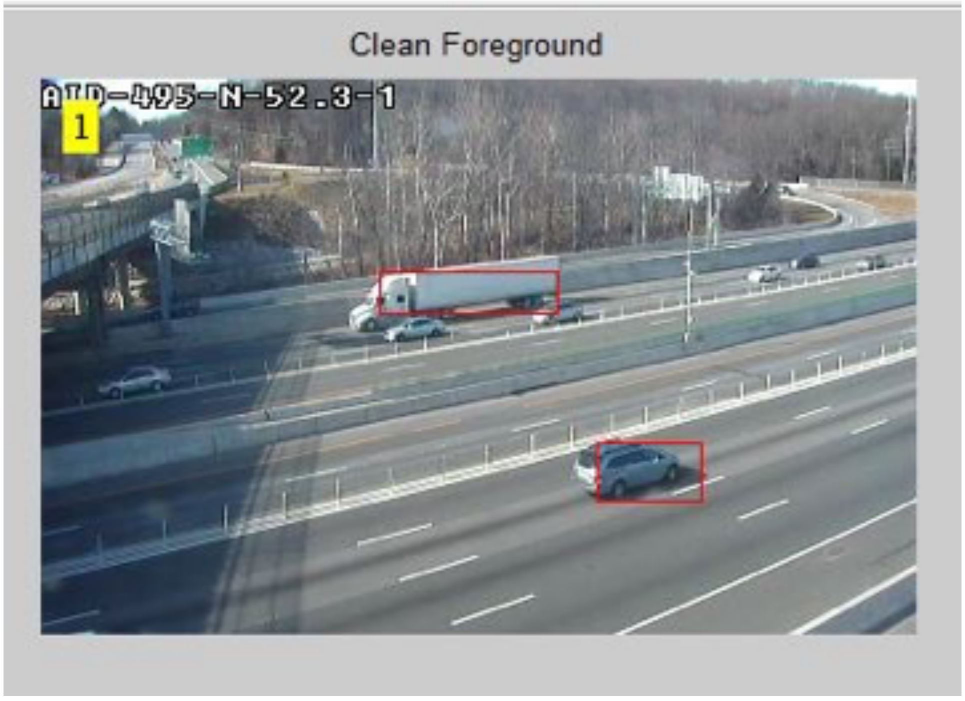


Fig 2.6 Counting vehicles in video AID-495-N-52.3-1 (Frame 30)

As we can see from the pictures, the counting is not always accurate when there are many small cars. That may caused by inaccuracy of the blob method. We tried to optimize the results by setting different parameters, but there still can be some mistakes.

Estimate the Speed of Each Vehicle

With the Matlab's package, we can get the U, V matrix from the output of the 'step' function. Then we calculate sqrt(U^2 + V^2) and generate a velocity matrix. In order to get rid of noise, we set a threshold 0.2 to the velocity. Any speed lower than 0.2 is considered as noise. Then we multiply the velocity by a constant to get a reasonable speed which is closer to the reality world. After several tries, it turns out that 300 is a good choice. Then we use Blob Analysis Block to locate each vehicle by reading the output results of the port BBox. With the bounding box parameters [x y width height], we can calculate the mean of each car's velocity. Finally, we use insertShape and insertText to label each car with its estimated speed. To clarify, we use the video which has been subtracted the background as the input.

Following is our implementation code.

```
input_video_gray = 'N-52.3-1.avi';
inputObj = VideoReader('AID-495-N-52.3-1.mp4');
nFrames = inputObj.NumberOfFrames; % Total number of frames
videoReader =
vision.VideoFileReader(input_video_gray,'ImageColorSpace','Intensity','VideoOu
tputDataType','uint8');
converter = vision.ImageDataTypeConverter;
opticalFlow = vision.OpticalFlow('ReferenceFrameDelay', 1);
opticalFlow.OutputValue = 'Horizontal and vertical components in complex
form';

%Convert the image to single precision, then compute optical flow for the
video. Generate coordinate points and draw lines to indicate flow. Display
results.
blobAnalysis = vision.BlobAnalysis(...
'CentroidOutputPort', false, 'AreaOutputPort', false, ...
'BoundingBoxOutputPort', true, ...
'MinimumBlobAreaSource', 'Property', 'MinimumBlobArea', 100);

speed_detection_limit = 10;

while ~isDone(videoReader)

    frame = step(videoReader);
    im = step(converter, frame);
    of = step(opticalFlow, im);
    lines = videooptflowlines(of, 20);

    velocity = sqrt(of.*conj(of));
    [row,col] = size(velocity);

    for i = 1:row
        for j = 1:col
            if velocity(i,j) > 0.2
                v(i,j) = velocity(i,j) * 300;
            else
                v(i,j) = 0;
            end
        end
    k = 1;
    for j = 1:size(bbox,1)
        x = bbox(j,1);
        y = bbox(j,2);
        width = bbox(j,3);
        height= bbox(j,4);

        if width*height > 0
            if (x+ width < col) && (y + height  < row)
                m =  mean(mean(v(y:y+height,x:x+width)));
                if m > speed_detection_limit %set threshold for detecting cars
                    display_box(k,:) = bbox(j,:);
                    velocity_mark(k,:) = [x,y,m];
                    k = k + 1;
                end
            end
        end
    end
    result = insertShape(frame, 'Rectangle', display_box, 'Color', 'green');
    for p = 1:size(velocity_mark,1)
        if velocity_mark(p,3) > speed_detection_limit
            result = insertText (result,[velocity_mark(p,1),velocity_mark(p,
2)],velocity_mark(p,3),...
            'BoxOpacity', 1,'FontSize', 10);
        end
    end
    inputFrame =  read(inputObj, t);
   figure(1),subplot(2,1,1),imshow(inputFrame);
    subplot(2,1,2),imshow(result)

    display_box = zeros(k,4);
    velocity_mark = zeros(k,3);
end
%Close the video reader and player

release(videoReader);
```

Some screenshots of our results are presented below.

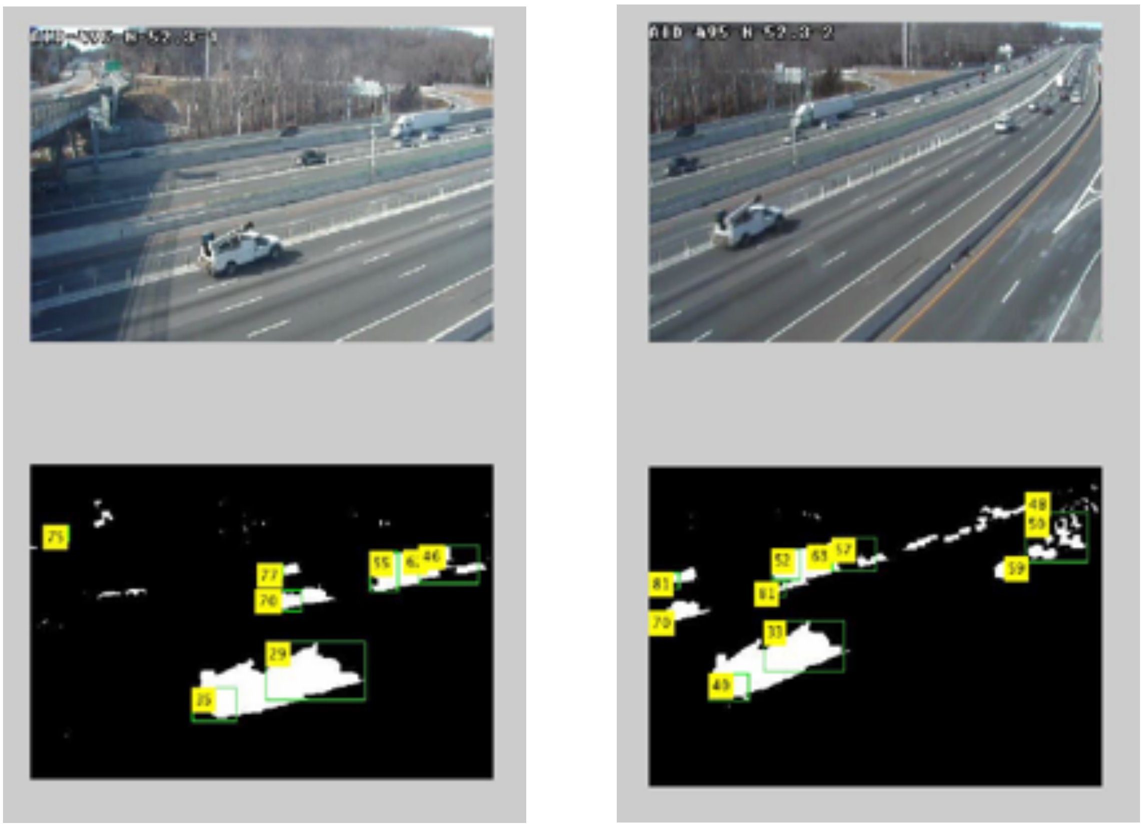


Fig 3.1 Speed estimation of video AID-495-N-52.3-1(2) (Frame 20)

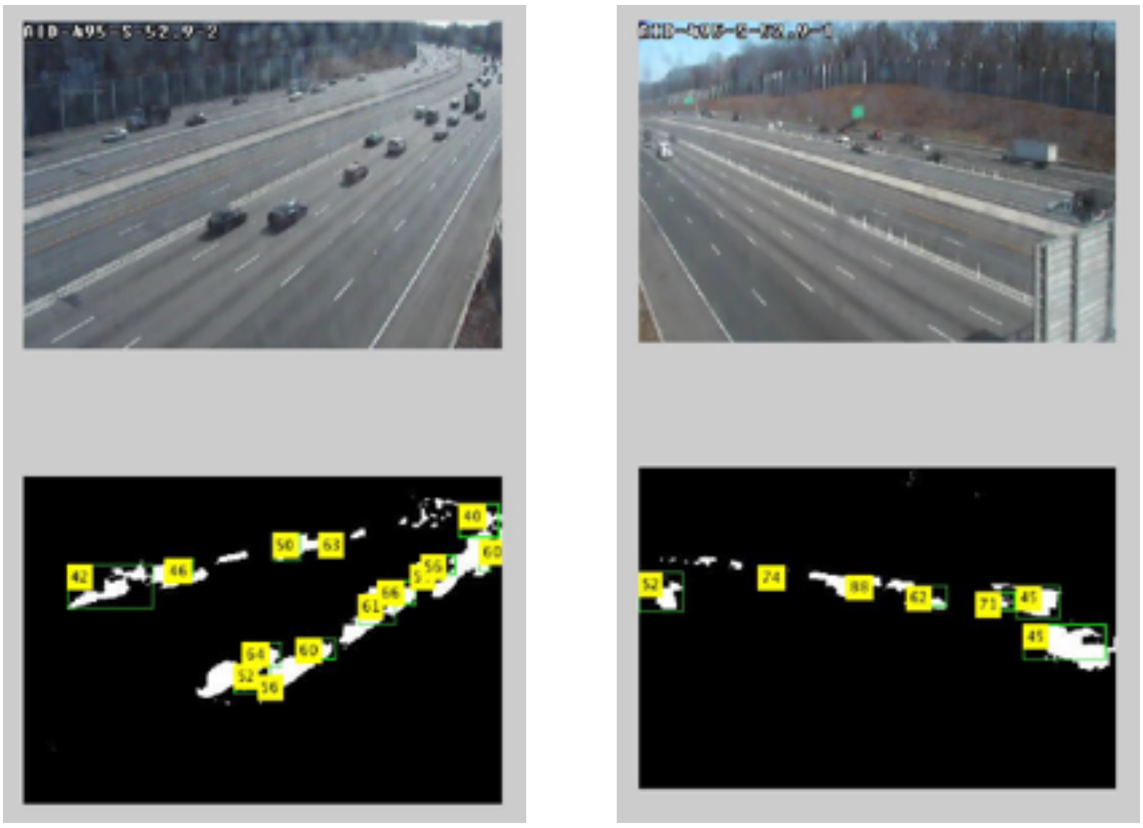


Fig 3.2 Speed estimation of video AID-495-S-52.9-1(2) (Frame 20)

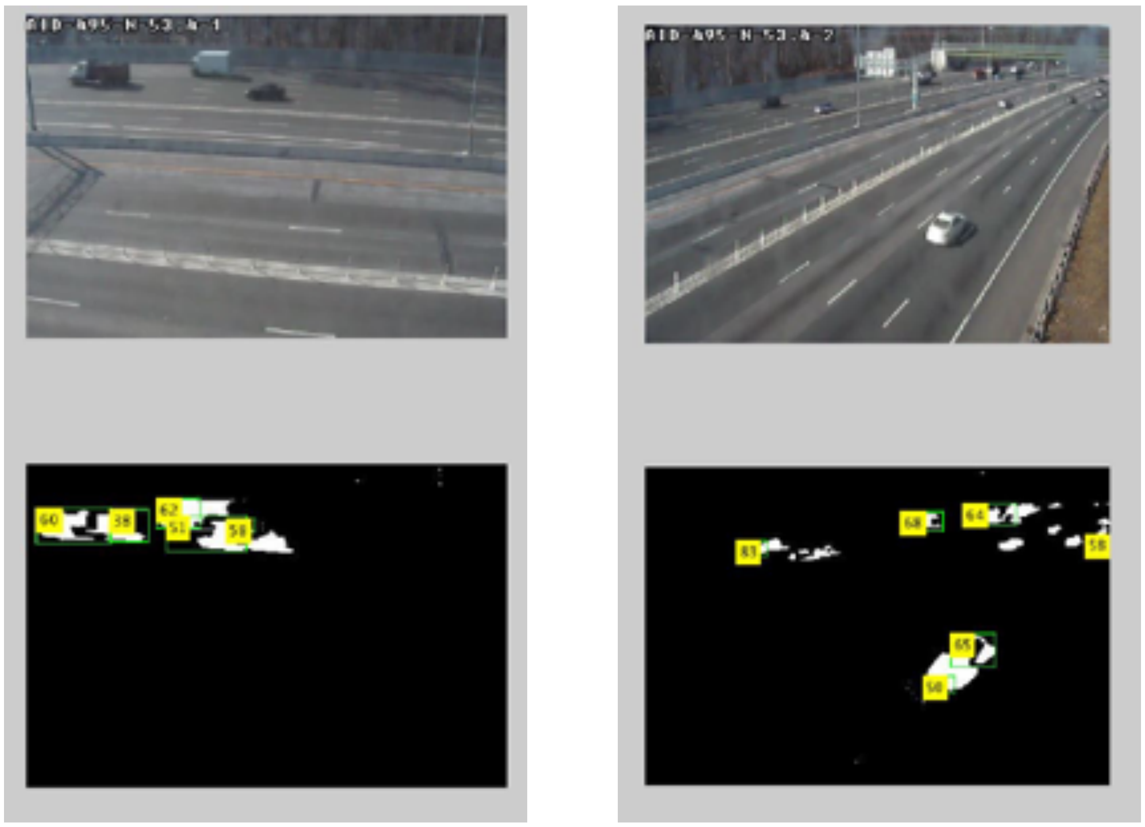


Fig 3.3 Speed estimation of video AID-495-N-53.4-1(2) (Frame 20)

Motion History Image

Since we have already done Motion History Image for assignment 1, we simply applied that code to this question and omit the analysis part here. The source code is as follows.

```
%removing back ground to make people in blobs
%adding all frames before to make motion energy image
%small substraction for frames before to make it fade
inputObj = VideoReader('AID-495-S-52.9-2.mp4');
nFrames = inputObj.NumberOfFrames; % Total number of frames

frame = read(inputObj,1); % read in 1st frame as background frame

%-----get the average value of every pixel in the frame-------
[height,width,d] = size(frame);
backgroundAvg = zeros(height,width);

for i = 1:nFrames
    background = double(rgb2gray(read(inputObj,i)));
    backgroundAvg = backgroundAvg + background;
end
backgroundAvg = backgroundAvg/nFrames;

[height,width,d] = size(frame);%get the size of each frame
% --------------------- process frames -----------------------------------
threshold = 25;
foreGround = zeros(height, width);
for k = 2:nFrames
    inputFrame = read(inputObj, k);
    inputGray = rgb2gray(inputFrame);
    framDiff = abs(double(inputGray) - double(backgroundAvg));
    for i=1:width
        for j=1:height
            if ((framDiff(j,i) > threshold))
                foreGround(j,i) =255;
            else
                if (foreGround(j,i)>0)
                    foreGround(j,i)=foreGround(j,i)-20;
                else
                    foreGround(j,i)=0;
                end
            end
        end
    end
    figure(1),subplot(2,1,1),imshow(inputFrame);
    subplot(2,1,2),imshow(uint8(foreGround))
end
```

Here are some screenshots of the results. In order to compare Motion History Image with Optical Flow, we use the same frame to analyze.

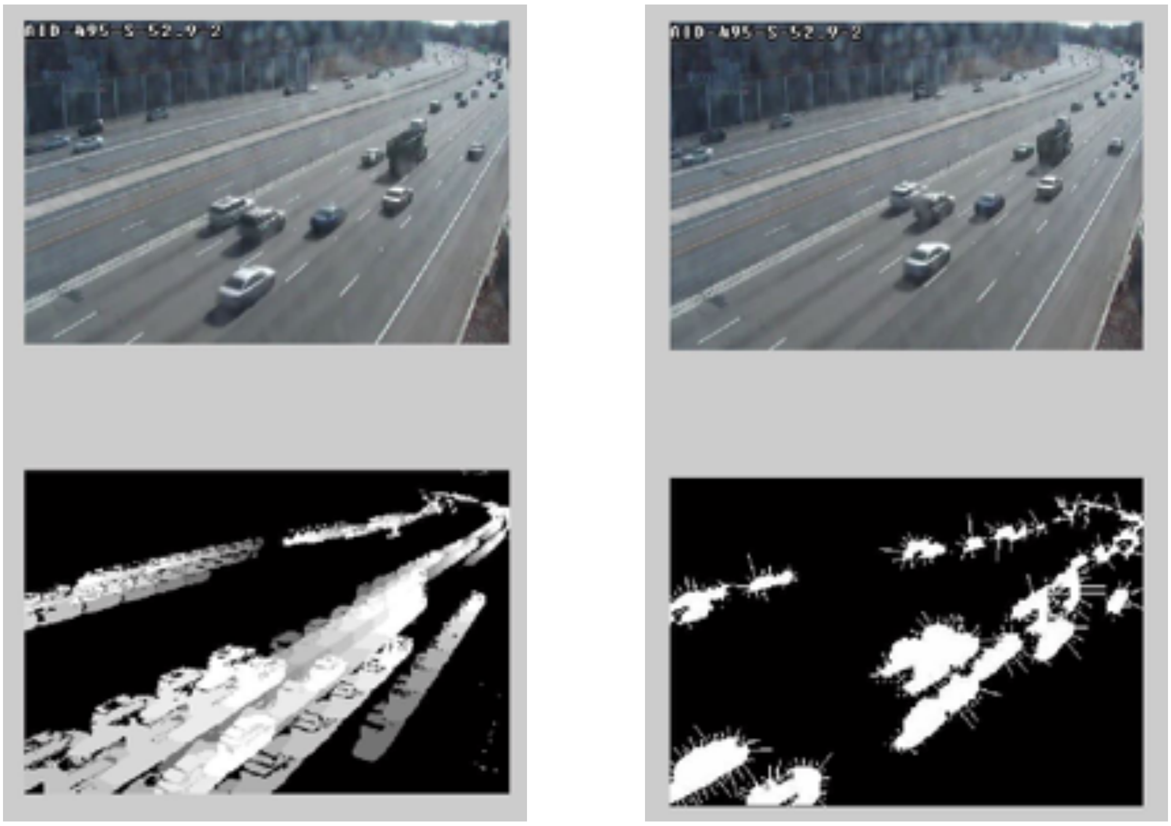


Fig 4.1 Screenshots of Motion History Image and Optical Flow Map (Frame 10)

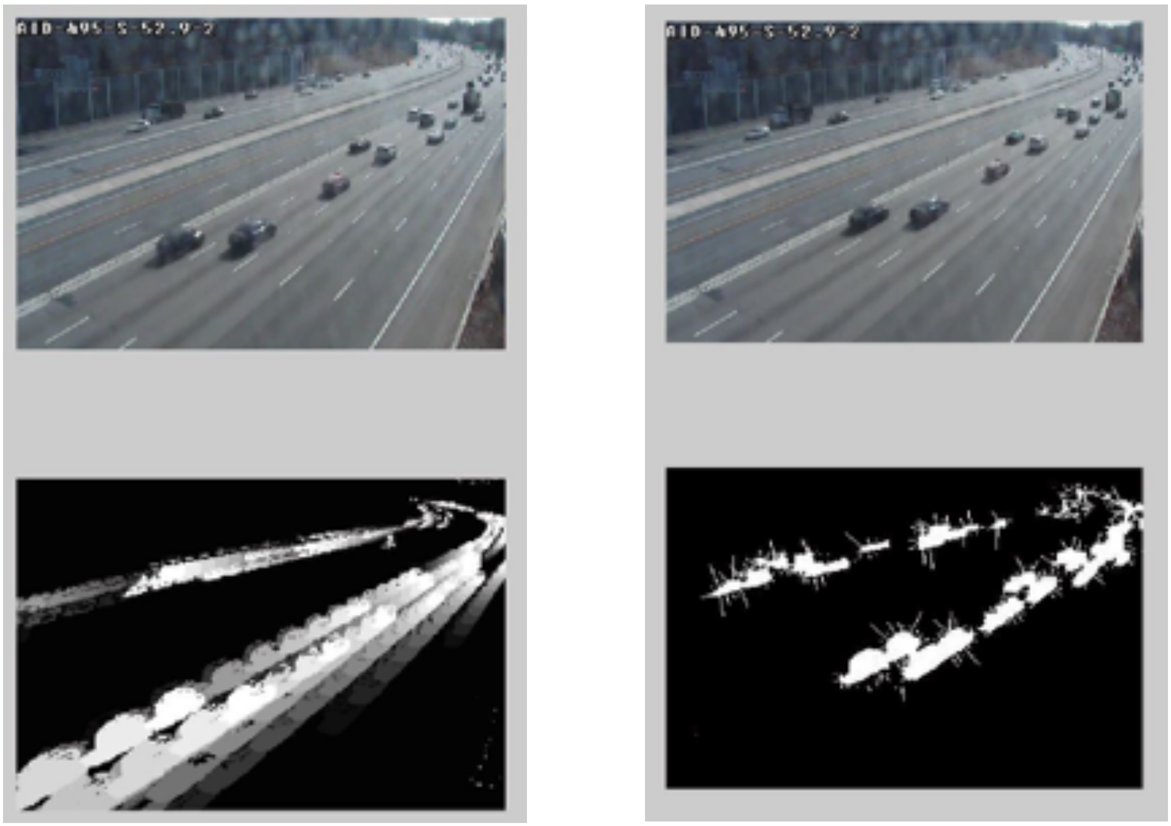


Fig 4.2 Screenshots of Motion History Image and Optical Flow Map (Frame 20)

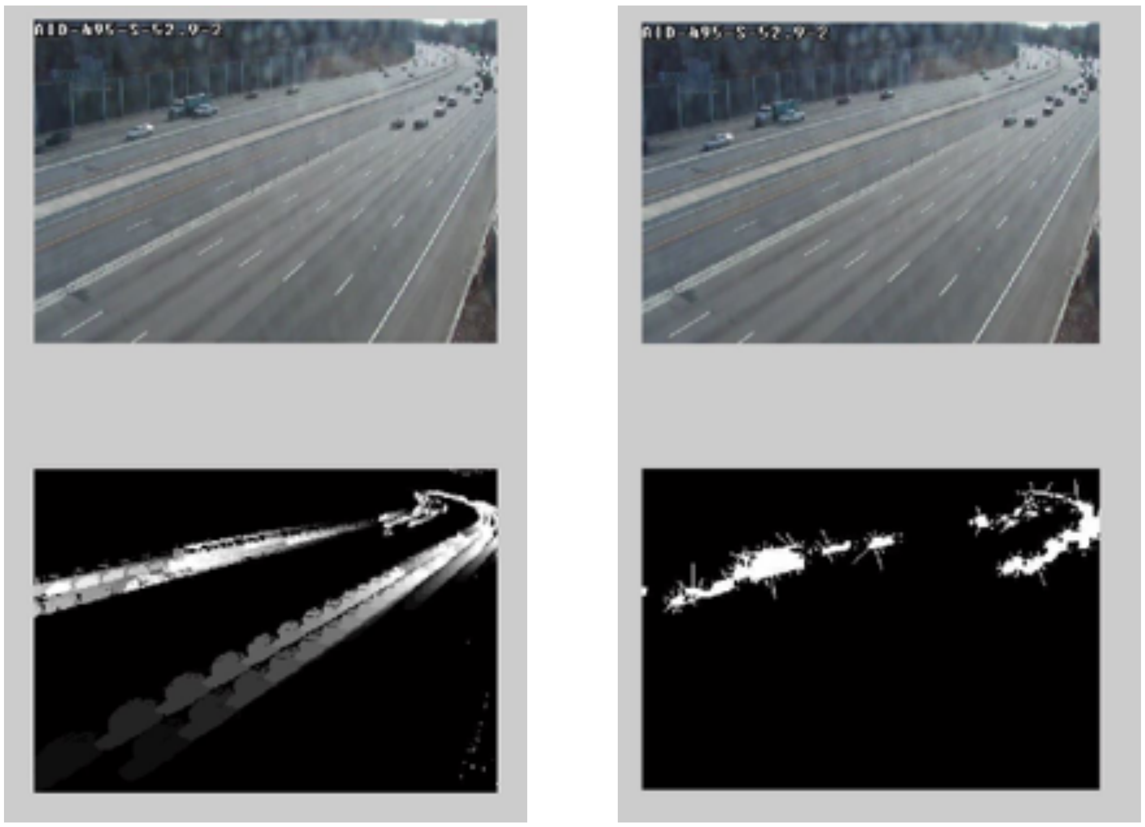


Fig 4.3 Screenshots of Motion History Image and Optical Flow Map (Frame 30)

Analysis: On the one hand, Motion History Image can tell us vehicles' moving trails while optical flow can tell us the contour of vehicles'. On the other hand, the vectors for optical flow imply vehicles' moving velocity and direction, which contains more information than Motion History Image.

Detect the Anomalous Events

Since we have implement velocity detection method in question 3, we can look through the speed matrix for vehicles and set threshold to detect anomalous events represented by abnormal speed. The threshold includes two part, the first part is a lower bound, which distinguishes all vehicles from other moving objects, and the other part, upper bound, is used to distinguishes anomalous vehicles from other vehicles with normal speed.

In this question , we don’t have to distinguish the direction or lane of highway, because elements in velocity matrix are scale, not vector. We don’t need to detect the direction of abnormal vehicle, because its abnormal low speed is enough to distinguish it from other normal vehicles. Here is the source code.

```
input_video_gray = ’S-52.9-2.avi';
inputObj = VideoReader('AID-495-S-52.9-2.mp4');
nFrames = inputObj.NumberOfFrames; % Total number of frames
videoReader =
vision.VideoFileReader(input_video_gray,'ImageColorSpace','Intensity','VideoOu
tputDataType','uint8');
converter = vision.ImageDataTypeConverter;
opticalFlow = vision.OpticalFlow('ReferenceFrameDelay', 1);
opticalFlow.OutputValue = 'Horizontal and vertical components in complex
form';

%Convert the image to single precision, then compute optical flow for the
video. Generate coordinate points and draw lines to indicate flow. Display
results.
blobAnalysis = vision.BlobAnalysis(...
'CentroidOutputPort', false, 'AreaOutputPort', false, ...
'BoundingBoxOutputPort', true, ...
'MinimumBlobAreaSource', 'Property', 'MinimumBlobArea', 100);

speed_detection_limit_upper_bound = 40;
speed_detection_limit_lower_bound = 30;

while ~isDone(videoReader)
    frame = step(videoReader);
    im = step(converter, frame);
    of = step(opticalFlow, im);
    lines = videooptflowlines(of, 20);

    velocity = sqrt(of.*conj(of));
    [row,col] = size(velocity);
    for i = 1:row
        for j = 1:col
            if velocity(i,j) > 0.2
                v(i,j) = velocity(i,j) * 300;
            else
                v(i,j) = 0;
            end
        end
    end
    bbox = step(blobAnalysis, boolean(v));
    k = 1;
    for j = 1:size(bbox,1)
        x = bbox(j,1);
        y = bbox(j,2);
        width = bbox(j,3);
        height= bbox(j,4);

        if width*height > 0
            if (x+ width < col) && (y + height  < row)
                if m > speed_detection_limit_lower_bound &&  m <
speed_detection_limit_upper_bound   %%set threshold for detecting cars
                    display_box(k,:) = bbox(j,:);
                    velocity_mark(k,:) = [x,y,m];
                    k = k + 1;
                end
            end
        end
    end

    result = insertShape(frame, 'Rectangle', display_box, 'Color', 'green');
    for p = 1:size(velocity_mark,1)
        if velocity_mark(p,3) > speed_detection_limit_lower_bound &&
velocity_mark(p,3) < speed_detection_limit_upper_bound
            result = insertText (result,[velocity_mark(p,1),velocity_mark(p,
2)],velocity_mark(p,3),...
            'BoxOpacity', 1,'FontSize', 10);
        end
    end
    inputFrame =  read(inputObj, t);
    figure(1),subplot(2,1,1),imshow(inputFrame);
    subplot(2,1,2),imshow(result)

    display_box = zeros(k,4);
    velocity_mark = zeros(k,3);
end
%Close the video reader and player

release(videoReader);
```

Here are the screenshots of the last two videos with the anomalous car.

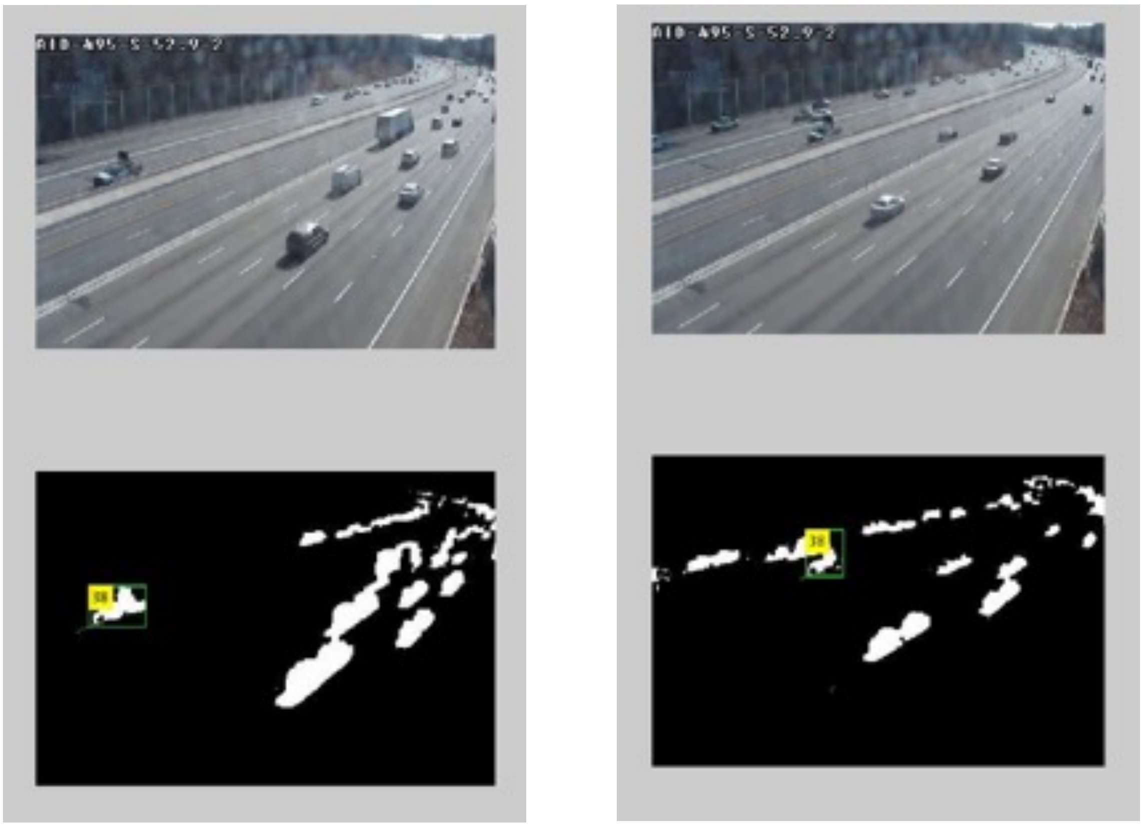


Fig 5.1 Detect the anomalous car in the video AID-495-S-52.9-2 at frame 136 and 171

Label the Camera Locations

As for this question, we tried to use the anomalous vehicle to judge the location of each camera. But we failed to find that car in all the six videos. Instead, we use a specific truck which has a bright color and big size to learn cameras' locations. We tried to use our code to find a specific car by setting a series of parameters. When it finds such object, it breaks at that frame. By comparing the frame number and its velocity direction, we can roughly estimate the location of each camera. However, since there are some noise and our code cannot find a car precisely, we cannot get the exact truck for all six videos. So for those videos which failed to finding the truck, we roughly use the successful ones to estimate the time it may show up in the failing videos.

Here are the frames we extract from five videos (We didn't find such a car in one video due to the camera's shooting direction). We use this yellow big truck to solve this problem.

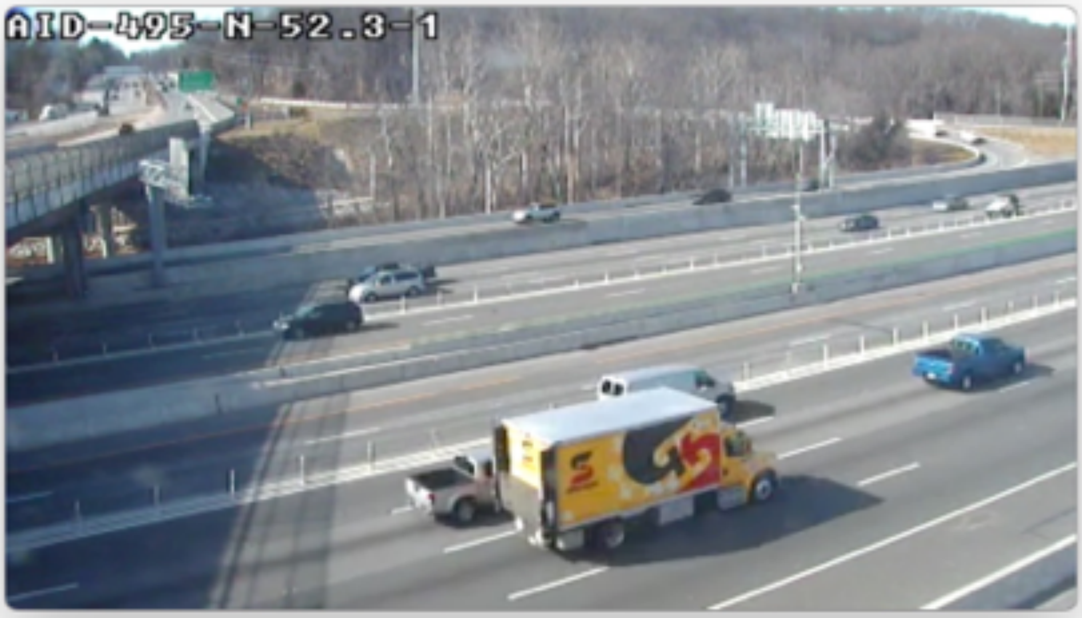


Fig 6.1 This yellow truck showed up in video AID-495-N-52.3-1 at time 02:02

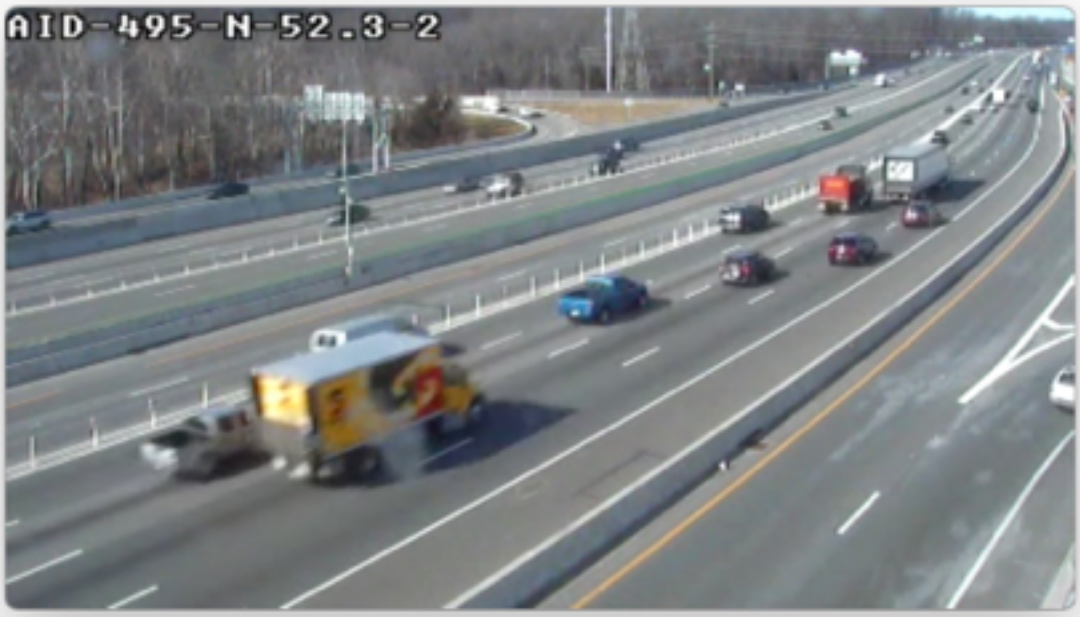


Fig 6.2 This yellow truck showed up in video AID-495-N-52.3-2 at time 02:03

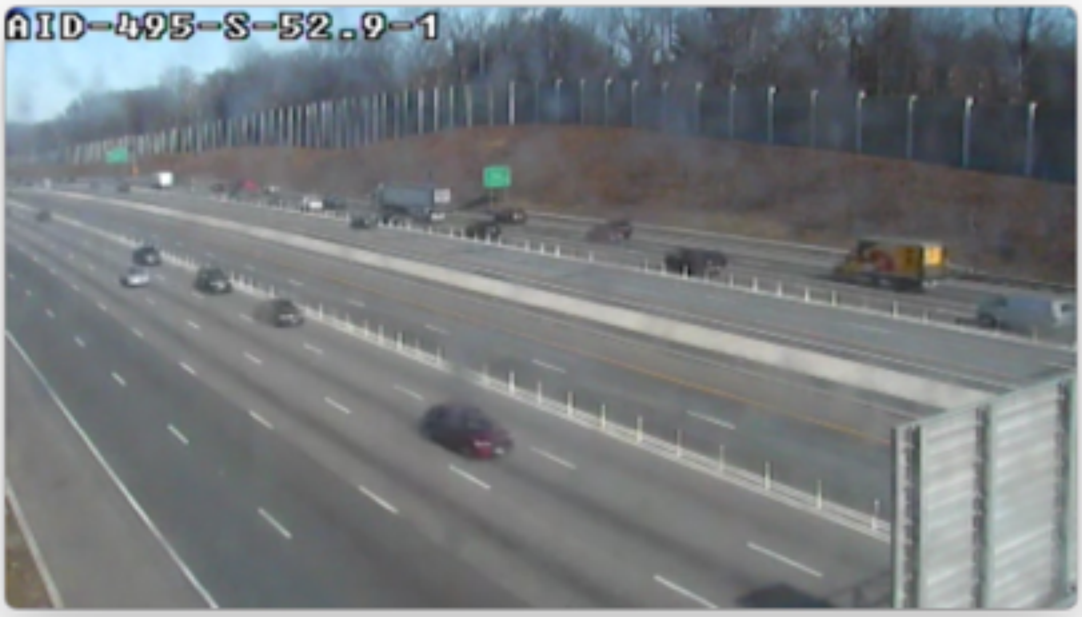


Fig 6.3 This yellow truck showed up in video AID-495-S-52.9-1 at time 01:27

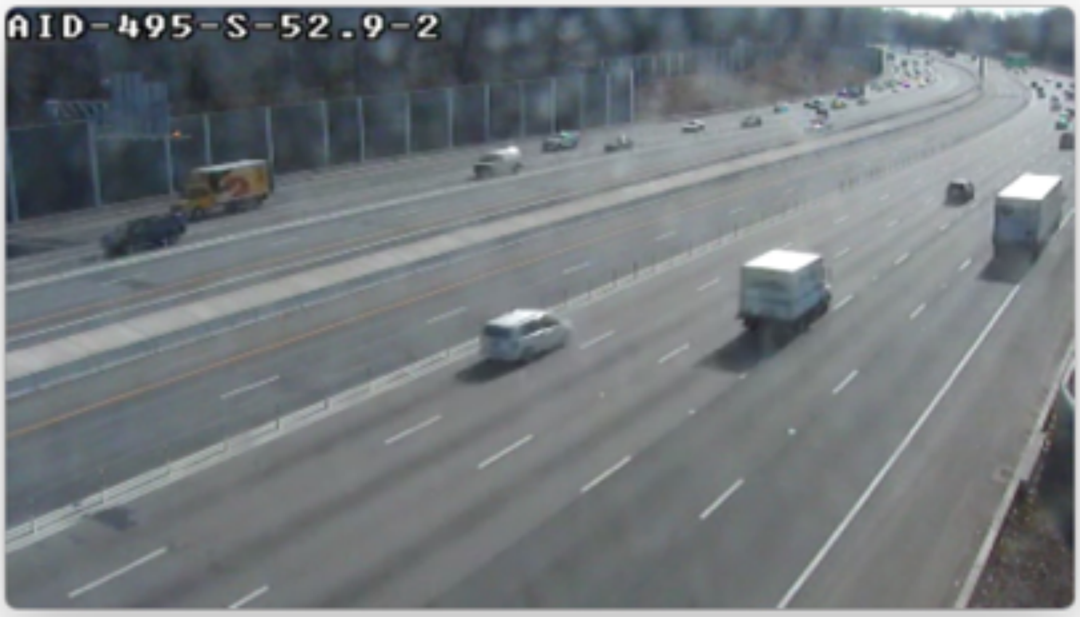


Fig 6.4 This yellow truck showed up in video AID-495-S-52.9-2 at time 01:22

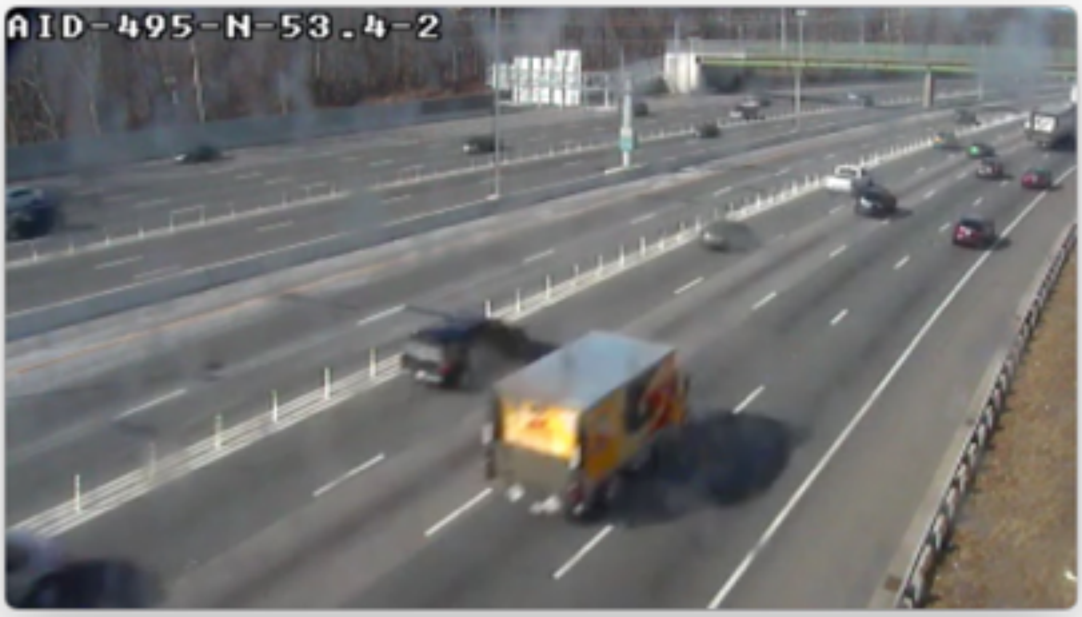
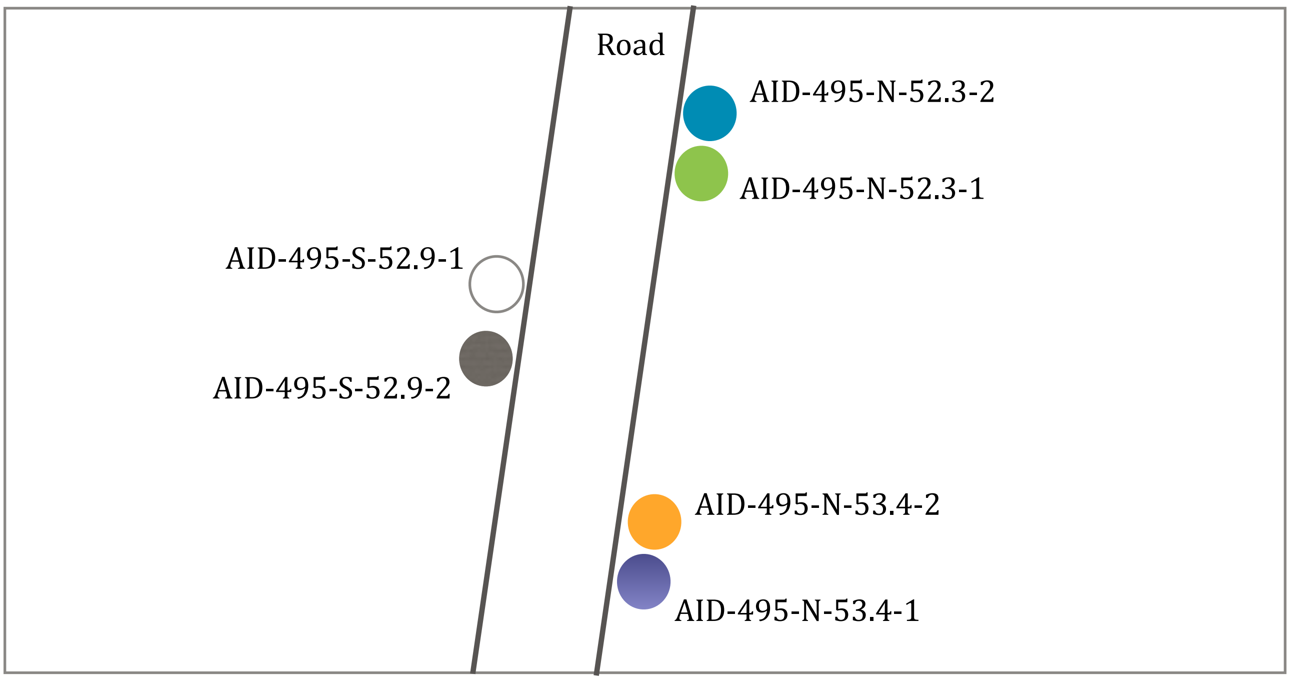


Fig 6.5 This yellow truck showed up in video AID-495-N-53.4-2 at time 00:55

According to the timeline and the direction, we can get a conclusion as follows. In order to make it more clear, below is a picture showing our judgement about these six cameras' locations.


Road
AID-495-N-52.3-2
AID-495-N-52.3-1
AID-495-S-52.9-1
AID-495-S-52.9-2
AID-495-N-53.4-2
AID-495-N-53.4-1