EE 417 Term Project

Topic: Visitor Traffic Counter, Based on Optical Flow

Team Members: Murat Aslan 26746, Işıl Sefünç 26698

Date:14 January 2022

Importance of the Problem and Problem Definition

In this project we are aiming to count the number of people that cross a counting line. In today's world, the usage of surveillance technology has expanded, particularly after Covid 19. Places such as the library in Sabancı University or malls or large stores may want to count the number of people entering and exiting a place for various reasons.

Problem Formulation and Solution Method

In our approach we used optical flow to detect the number of people, since motion is one of the major signs for entering or exiting a building. Using consecutive frames, we calculate Optical Flow to determine the motion vector field. Calculation of optical flow between video frames gives us the velocities of an object in a video. We assumed a counting line in front of the entrance of the building. Since when someone is entering or exiting the building he or she has to be present in the rectangular area around the counting line, we are focusing on that rectangular area using a window.

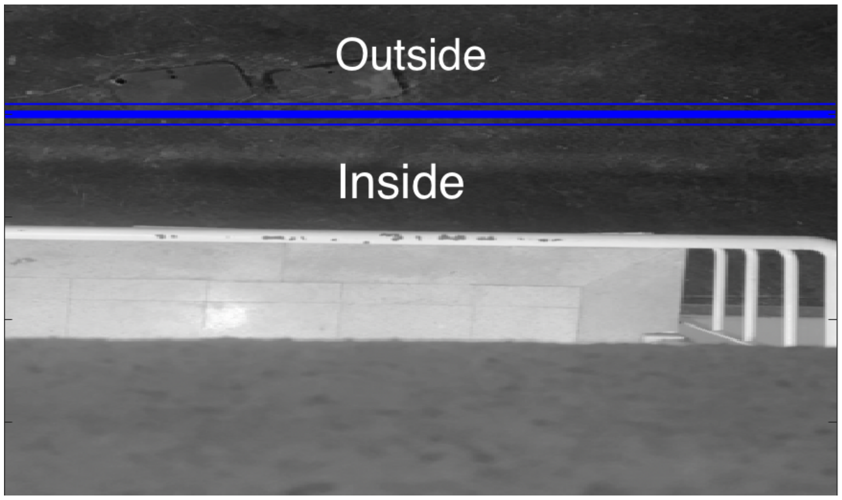


Figure 1: The counting line and the rectangular area shown

We use the optical flow vectors in that rectangular area to extract features. If there are multiple arrows across the counting line this means there is a person crossing the door. The direction of the arrows indicates whether the person is exiting or entering the building.

Your own implementation and providing your own results

While performing the optical flow algorithm developed by Lucas-Kanade, we use the u = [V x; V y] = G1 * b formula to estimate velocities, where G equals the Harris matrix with spatial derivatives and b with temporal derivatives. The algorithm focuses on the movement of features between frames as a response to the scene and camera moving relative to each other.

To perform the Lucas Kanade algorithm we first need to smooth the images using a gauss filter. Since we will compute the Image x and Image y derivatives we need to smooth the image, since the derivative is sensitive to noise. For each consecutive frame we are subtracting ImPrev from ImCurrent to get the difference image which shows the relative motion. By solving the linear equations for each pixel, we then calculate the eigenvalues. Using the Eigenvalues we got, we solve the equations and display the optical flow vectors on the image with red color. In this step we can display the velocity vectors on the image frames and see the moving object and their direction. (classification of the vectors).

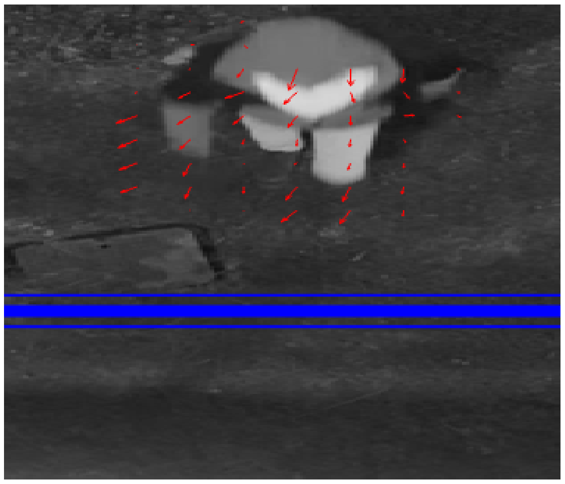
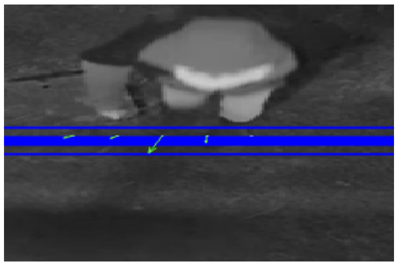


Figure 2: Velocity vectors

After that, we plotted the counting line on the image to represent the entrance. To show the optical flow vectors close to the counting line we drawed them in green color. The green arrows are the critical features that we look to identify the people count.



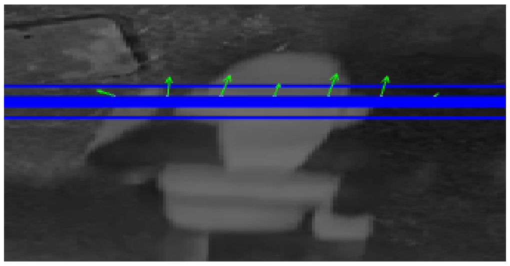


Figure 3: Person going in

Figure 4: Person going out

We set a threshold to eliminate the noise in the image for the aim of lowering it. We don't consider velocity vectors with magnitudes smaller than that threshold in this way.

In the second part to count the people, we created an empty array to count the number of arrows in each frame. Each instance of the array will represent the number of green arrows (as shown in Figure 3 and 4) in that frame. In this array a green arrow pointing down is considered positive, while a green arrow pointing up is considered negative. We iterate through the pixels that are between the top and the bottom blue line in Figure 4, and count the number of green arrows in each frame. This means that we only consider optical flow that is close to the door, and we keep track of the direction of the optical flow vectors.

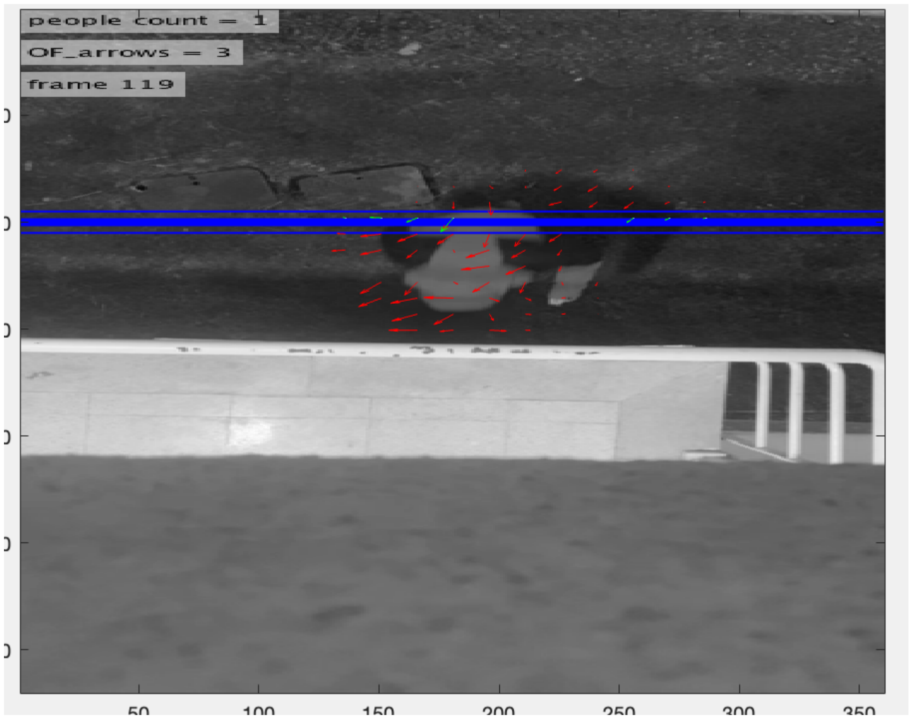
To indicate the number of people going in and going out we use this array. When a person is walking through the line, first the number of arrows - with positive or negative direction - will increase and then decrease in each frame, since the person will go away from the counting line eventually. So, to detect and count the incoming and outgoing people we examine the array and if the number of arrows are 0, we determine no one is moving near the exit, and if the number of arrows stay positive and above a certain threshold for consecutive frames then we determine someone has come inside, if the number of arrows are negative we determine that someone has gone out.

Otherwise, if it is in the opposite direction of line this means a person is exiting the building and we decrease the counter. The counter is placed on the upper left corner of the image.

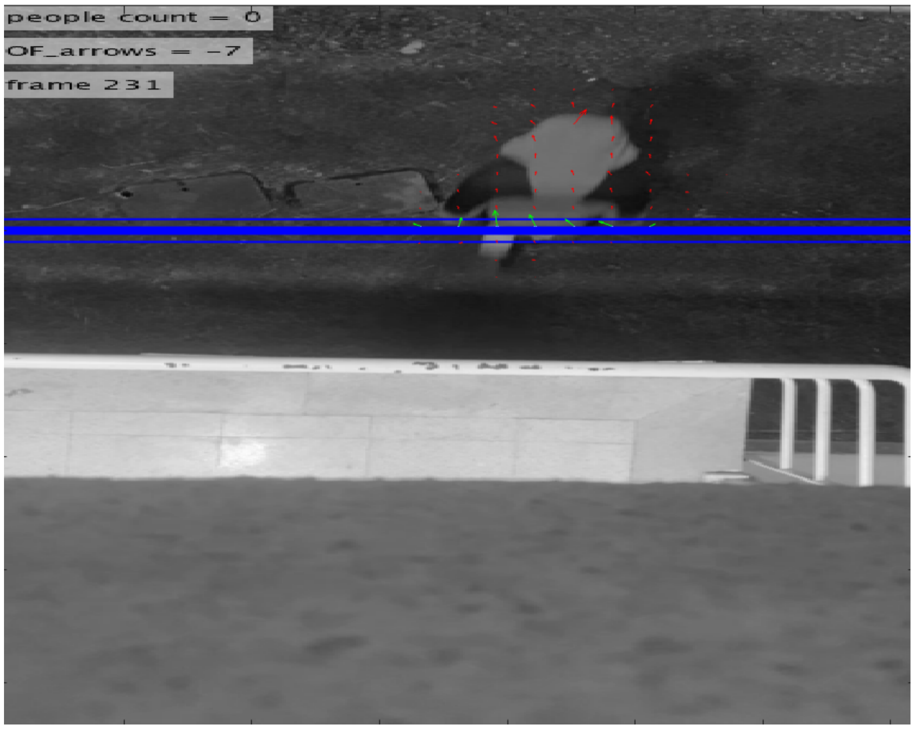
Discussion of the results

Our method is composed of two main steps, first one is classification of the optical flow vectors and the second phase is to analyze the motion of the object looking at their velocity vectors and count the people according to it.

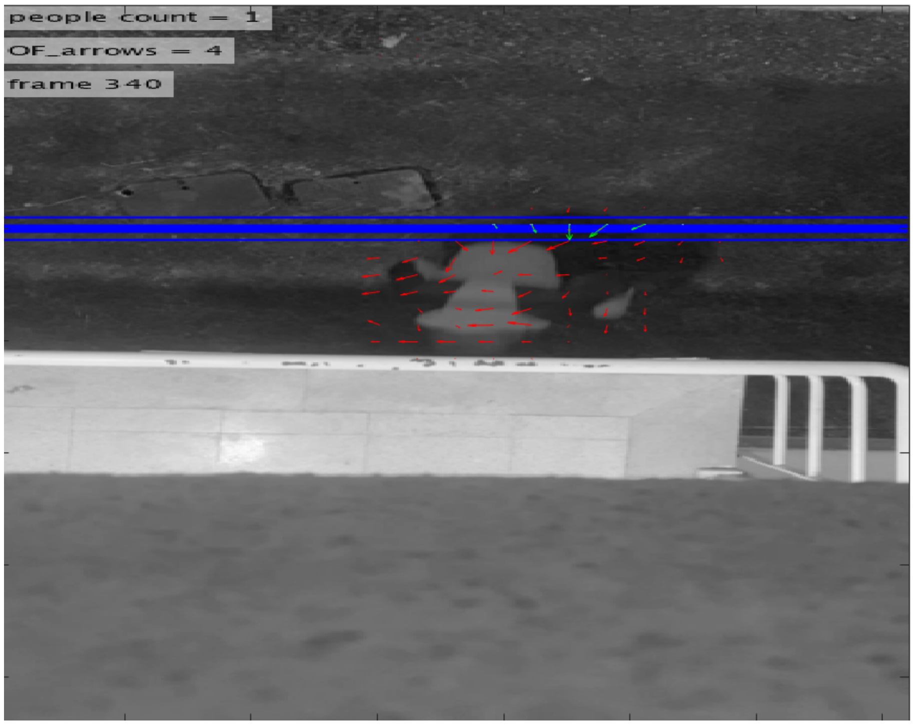
The output of our method is shown below.



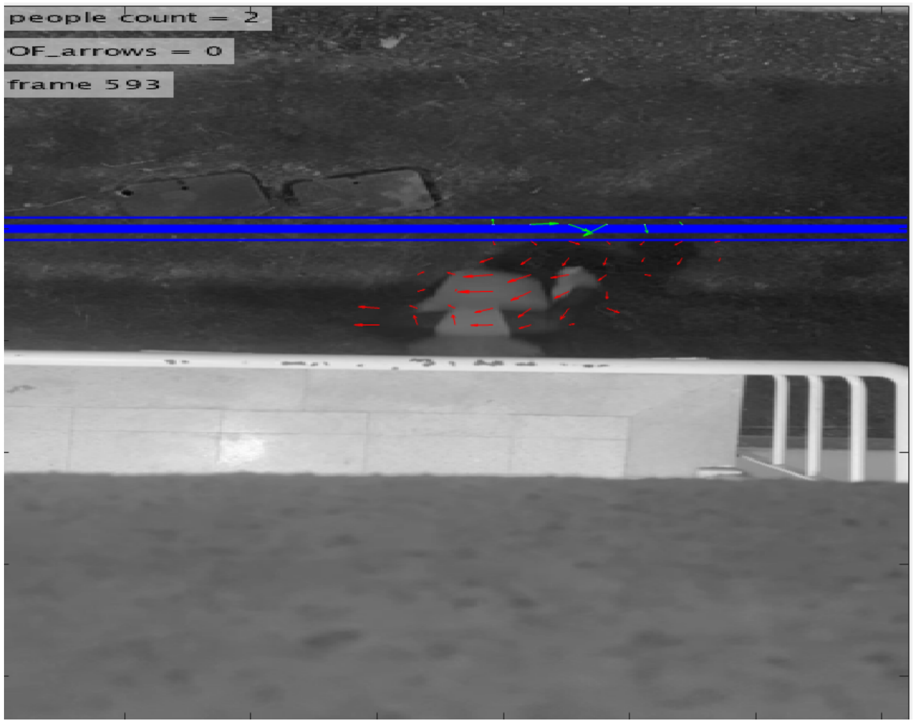
Count = 1



Count = 0



Count = 1



Count = 2

Because our approach isn't dependent on recognizing individuals, it doesn’t discriminate between objects, this may cause error if there are many objects moving in the environment. Instead of a person going through the door, multiple small objects going through the door would trick our algorithm.

Furthermore, our system currently only works for one person crossing at a time. The window size affected the performance of the algorithm most, also the following line took the most of the time during our executions:

“[xramp,yramp] = meshgrid(1:1:xdim,1:1:ydim); quiver(xramp,yramp,Vxg,Vyg,10,"g");”

but it is not a concern, as this is only to display the green arrows in the frame, not integral to the algorithm itself.

The Lukas Kanade optical flow algorithm has 2 important parameters, “K = window size”, and the “Threshold” parameter we use to determine if the current window we are iterating over is a significant image image feature (corner or not), this numeric value of this threshold value should depend on the window size. Other than that, because the video is noisy, using the Lucas Kanade optical flow algorithm results in still corners of the image looking as if they have a very minute amount of motion, we set a second threshold parameter that filters out parts of the image with low temporal difference so that we can ignore these optical flow vectors, and only consider strong optical flow vectors that are generated by a person going through the door. We had to create this second threshold parameter, because even after we filter out the noise, if the intensity of the same window in 2 consecutive frames is different by a magnitude of 1, we would falsely consider this as motion (albeit a small one, but our algorithm counts the number of vectors, not the sum of their magnitudes so we have to ignore these vectors). One other parameter we have is during the count_ins function

Appendix:

Proj_main.m:

```
%%
clear all; close all; clc;
% Load the files given in SUcourse as Seq variable [row,col,num]=size(Seq);
load("walk3.mat");
Seq = walk3;
[row,col,dummy,num]=size(Seq);
```

```
Threshold = 30000;

k = 15;

% ImPrev = Seq(:,:,1);

% ImCurr = Seq(:,:,2);

% lab7OF2(ImPrev,ImCurr,k,Threshold);

%j = 2;

%ImPrev = Seq(:,:,j-1);

%ImCurr = Seq(:,:,j);

%lab7OF_P(ImPrev,ImCurr,k,Threshold);

%%

% Define k and Threshold

arrow_per_frame = zeros(1,num); %

step_size = 1; % must be 1

tic

for j=1+step_size:step_size:num


    ImPrev = Seq(:,:,:,j-step_size);

    ImCurr = Seq(:,:,:,j);

    ImPrev = rgb2gray(ImPrev);

    ImCurr = rgb2gray(ImCurr);

    arrow_per_frame = proj_OF_loop(ImPrev,ImCurr,k,Threshold,arrow_per_frame,j-1,step_size);

    pause(0.001);

end

toc

%display(arrow_per_frame);

%person_count = count_ins2(arrow_per_frame,step_size);

%display(person_count);

%%
```

Proj_OF_loop.m

```
function arrow_per_frame_out = proj_OF_loop(ImPrev,ImCurr,k,Threshold,arrow_per_frame,frame_index,step)
```

```
% Smooth the input images using a Box filter

% write a kernel and use matlab's built in convolution function

thresh2 = 25;

thresh3 = 25;

box_filt_kernel =  [1/9,1/9,1/9;

                              1/9,1/9,1/9;

                               1/9,1/9,1/9];

gauss_kernel = [1,4,7,4,1;

   4,16,26,16,4;

   7,26,41,26,7;

   4,16,26,16,4;

   1,4,7,4,1;];

gauss_kernel = gauss_kernel/273;

% ImPrev = conv2(ImPrev,box_filt_kernel);

% ImCurr = conv2(ImCurr,box_filt_kernel) ;

%ImPrev = medfilt2(ImPrev,[5 20]);

%ImCurr = medfilt2(ImCurr,[5 20]);

ImPrev = double(imgaussfilt(ImPrev));

ImCurr = double(imgaussfilt(ImCurr));

%ImPrev = medfilt2(ImPrev);

%ImCurr = medfilt2(ImCurr);

% Calculate spatial gradients (Ix, Iy) using Prewitt filter

% write prewitt kernel and use built-in convolution function

prewitt_x_kernel = [1,0,-1;

                    1,0,-1;

                    1,0,-1];

prewitt_y_kernel = [1,1,1;

                     0,0,0;

                    -1,-1,-1];

Gx= conv2(ImCurr,prewitt_x_kernel);

Gy = conv2(ImCurr,prewitt_y_kernel);
```

```
% Calculate temporal (It) gradient
ImTemporal = double(ImCurr - ImPrev);
%ImTemporal= wiener2(ImTemporal,[5 20]);
%ImTemporal = imgaussfilt(ImTemporal);
[ydim,xdim] = size(ImCurr);
Vx = zeros(ydim,xdim);
Vy = zeros(ydim,xdim);
Vxg = zeros(ydim,xdim);
Vyg = zeros(ydim,xdim);
G = zeros(2,2);
b = zeros(2,1);
cx=k+1;
%disp(max(ImTemporal,[],'all'));
if max(ImTemporal,[],'all') < thresh2
  %disp("false");
else
for x=k+1:k:xdim-k-1
  cy=k+1;
      for y=k+1:k:ydim-k-1
          %sub_array = ImCurr(i:i+2*k,j:j+2*k);
          sum_x_squared = sum(Gx(y-k : y+k , x-k:x+k).*Gx(y-k : y+k , x-k:x+k),'all');
          sum_x_y = sum( Gx( y-k : y+k , x-k:x+k ).* Gy(y-k : y+k , x-k:x+k),'all' );
          sum_y_squared = sum( Gy(y-k : y+k , x-k:x+k).*Gy(y-k : y+k , x-k:x+k),'all' );
          G = [sum_x_squared,sum_x_y;
                 sum_x_y,sum_y_squared];
      % Calculate the elements of G and b
           eigen = eig(G);
          if (min(eigen) < Threshold)
              Vx(cy,cx)=0;
               Vy(cy,cx)=0;
          else
```

```
            % Calculate u
            curr_temp_window = ImTemporal(y-k : y+k , x-k:x+k);
            if max(curr_temp_window,[],'all') > thresh3
                b11 = sum(    Gx(y-k : y+k , x-k:x+k) .* ImTemporal(y-k : y+k , x-k:x+k),'all' );
                b12 = sum(    Gy( y-k : y+k , x-k:x+k ) .* ImTemporal(  y-k : y+k , x-k:x+k) ,  'all'  );
                b = [b11;
                        b12];
                u = (-inv(G)*b);
                Vx(cy,cx)=u(1);
                Vy(cy,cx)=u(2);
            else
                Vx(cy,cx)=0;
                Vy(cy,cx)=0;
            end
            end
        cy=cy+k;
      end
    cx=cx+k;
end
end
%divider = line([256,256],[0,512],'Color','r','LineWidth',2);%line([x1,x2],[y1,y2],'Color','r','LineWidth',2)
%x = 256, y = 0 to 512 % vertical line
right_of_counter = 0;
line_x = 200;
%disp(size(Vy))
%disp(size(Vyg))
%disp("q");
for i = line_x - 10:1:line_x + 10
   for j = 1:1:360
       if Vy(i,j) > 0
            right_of_counter = right_of_counter + 1;
            Vxg(i,j) = Vx(i,j);
            Vyg(i,j) = Vy(i,j);
            %Vx(j,i) = 1;
            %Vy(j,i) = 1;
            ImPrev(i,j) = 255;
         elseif Vy(i,j) < 0
            right_of_counter = right_of_counter -1;
            Vxg(i,j) = Vx(i,j);
            Vyg(i,j) = Vy(i,j);
            %Vx(j,i) = 1;
            %Vy(j,i) = 1;
            ImPrev(i,j) = 255;
         else
           a = 2; % dummy line
         end
   end
end
% for i = line_x :-1:line_x + 100
%     disp("a");
%     for j=1:1:400
%
%         if (abs(Vy(i,j)) > 0) %% a small value; and if it is smaller than
%             disp(j);
           % a specific negative value, then it is an outgoing arrow;
%               right_of_counter = right_of_counter + 1;
%               Vxg(i,j) = Vx(i,j);
%               Vyg(i,j) = Vy(i,j);
%               %Vx(j,i) = 1;
%               %Vy(j,i) = 1;
%               ImPrev(i,j) = 255;
%         else
%             a = 2;

%         end

%

%     end

% end

%disp(right_of_counter);

% right_of_counter = optical flow arrows going to right

person_count = count_ins2(arrow_per_frame,step);

ImPrev = uint8(ImPrev);

ImPrev = insertText(ImPrev,[0 0],sprintf('people count = %d',person_count));

ImPrev = insertText(ImPrev,[0 30],sprintf('OF_arrows = %d',right_of_counter));

ImPrev =  insertText(ImPrev,[0 60],sprintf('frame %d', frame_index));

ImPrev = rgb2gray(ImPrev);

arrow_per_frame_out = arrow_per_frame;

arrow_per_frame_out(1,frame_index) = right_of_counter;

%disp(size(ImPrev));

cla reset;

%imagesc(uint8(ImTemporal));

imagesc(ImPrev); hold on;

line([0,360],[200,200],'Color','b','LineWidth',4);%line([x1,x2],[y1,y2],'Color','r','LineWidth',2)

line([0,360],[190,190],'Color','b','LineWidth',1);%line([x1,x2],[y1,y2],'Color','r','LineWidth',2)

line([0,360],[210,210],'Color','b','LineWidth',1);%line([x1,x2],[y1,y2],'Color','r','LineWidth',2)

%[xramp,yramp] = meshgrid(1:1:xdim,1:1:ydim); quiver(xramp,yramp,Vx,Vy,10,"r");

if max(Vxg,[],'all') > 0

  [xramp,yramp] = meshgrid(1:1:xdim,1:1:ydim); quiver(xramp,yramp,Vxg,Vyg,10,"g");

end

colormap gray;

end
```

Count_ins2.m:

```
function Iout = count_ins2(arrow_per_frame_array,step)

[row,col] = size(arrow_per_frame_array);

frame_count = col;

%disp(row);

%disp(col);

Iout = 0;

k = 1;

while (k < frame_count)

  flag = 0;

  if( (arrow_per_frame_array(1,k)  >  0)  & (k < frame_count)   )

      total = 0;

      for i=k:step:k+20

          total = total + arrow_per_frame_array(1,i);

      end

      if total > 40

          flag = 1;

          k = k+20;

      end


  elseif( (arrow_per_frame_array(1,k)  <  0)  & (k < frame_count)   )

     total = 0;;

     for i=k:step:k+20


          total = total + arrow_per_frame_array(1,i);

      end

      if total < -40

          flag = -1;

          k = k+20;

      end

  end
  Iout = Iout + flag;

  k = k + step;

end


end
```