CRAB COUNTER

Chuangang Ren and Wei Lin

Dec. 15, 2007

Boston University
Department of Electrical and Computer Engineering

Technical report No. ECE-2007-05

BOSTON
UNIVERSITY

CRAB COUNTER

Chuangang Ren and Wei Lin

Boston University
Department of Electrical and Computer Engineering
8 Saint Mary's Street
Boston, MA 02215
www.bu.edu/ece

Dec. 15, 2007

Technical report No. ECE-2007-05

Summary

This report describes a project completed in the course EC520 entitled “Digital Image Processing and Communications”. The goal of this project was to develop an algorithm for the detection of hermit crabs roaming a beach using video cameras. An algorithm consisting of three steps: background subtraction, post-processing and counting, was implemented. The background subtraction step was implemented in one version using median temporal filtering, and in another version using probability density estimation. The algorithm based on median filtering produced more accurate crab counts, however turned out to be more sensitive to background movements. Further work is needed to accomplish accurate crab counting from video sequences.

Contents

List of figures and tables

1. Introduction

The task we undertook was to design a system to help scientists in monitoring and enumeration of crabs on a beach using video cameras.

Key steps of the proposed algorithm are:

1) Obtain binary image sequence to enable the detection of moving objects. We present two methods: Background Subtraction and Density Estimation.
2) Post-processing.
3) Enumeration of the objects.

2. Proposed approach

Step I: Obtain binary image sequence

Method A: Background Subtraction

Background subtraction is a method typically used to segment moving regions in image sequences taken from a static camera by comparing each new frame to a model of the scene background. For example, one of the simplest background subtraction techniques is to calculate a median image of the scene, subtract each new frame from this image and threshold the result. Fig.1 shows one frame (#16) from the original video sequence with 800 frames and the result of background subtraction.

Fig. 1. Frame from original video sequence and result of background subtraction

The binary image above is obtained using the same threshold at every pixel in one frame. This is not flexible when we deal with different areas (water and land) in an image, thus leading to wrong detections (this is obvious in water area in Fig.1, where ripples on water surface are wrongly detected as objects).

In order to fix this problem, we look into intensity value of one pixel over time, and determine a particular threshold for this pixel according to the feature of intensity value changing in time. In this crab-counting case, different parts in the video have obviously different features. Fig.2. shows the intensity values of two different pixels (one pixel, drawn as the red curve, is on land, the blue curve is in the water) changing from Frame #400 to 800).

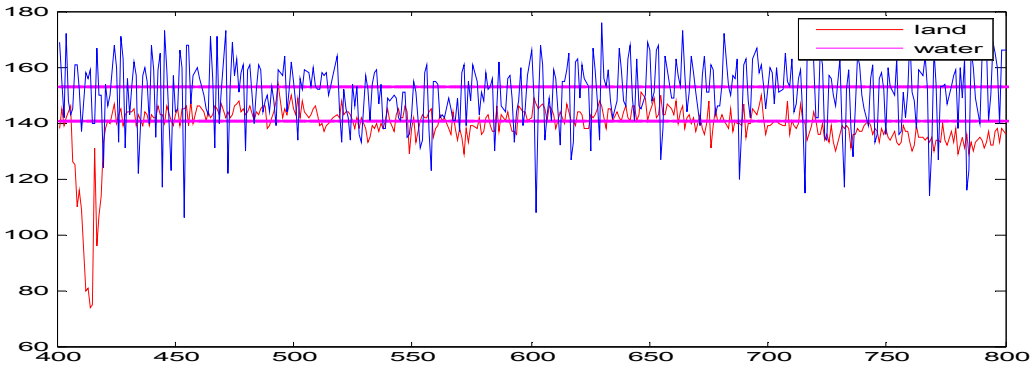

Fig. 2. Intensity value over time of pixels representing land and water (red line: pixel on land with coordinates (100,400); blue line: pixel in water with coordinates (450,400)).

Obviously, intensity fluctuations of the water pixel are greater than those of the land pixel. Based on this, we use the variance of intensity to control the magnitude of threshold, thus allowing adaptive thresholding. In order to eliminate the sudden jumps in magnitude caused by noise or small background object movement, we filter the above time-variant signal using a low-pass filter.

Let $P_{i,j}(n)$ be intensity value of pixel with coordinates (i,j) over time. $B_1, B_2 ... B_n$ are the image sequence outputs. Then, we have the output B_n:

$$U_{i,j}(n) = (P_{i,j}(n) - M_{i,j}) * H;$$

$$V_{i,j}(n) = \begin{cases} 0, U_{i,j}(n) \le \alpha\sigma \\ 1, U_{i,j}(n) > \alpha\sigma \end{cases}$$

$$B_n = \sum_i \sum_j V_{i,j}(n)$$

In which, $M_{i,j}$ is the median of $P_{i,j}(n)$. σ is the standard deviation of $P_{i,j}(n)$, with multiplication of coefficient α as the threshold. H is a length 15 low pass filter. $U_{i,j}$ is the output of low pass filter, and $V_{i,j}$ is $U_{i,j}$ after threshold. Fig 3 shows a block diagram of our system.

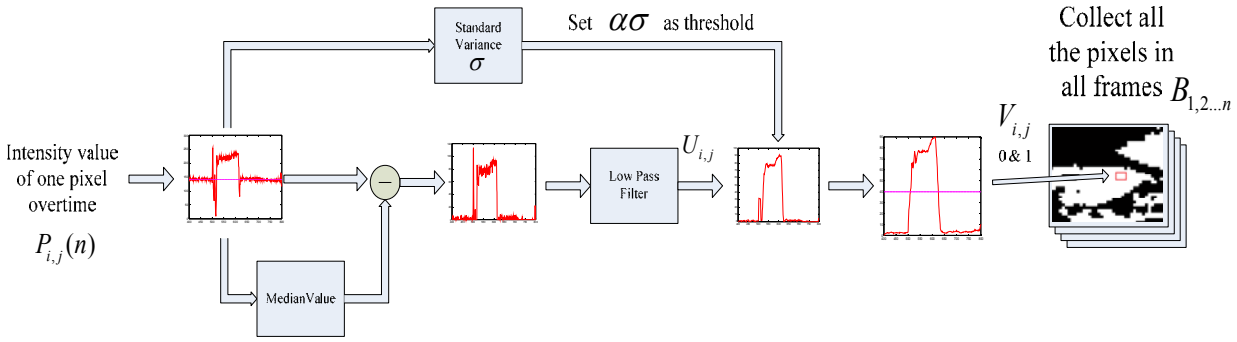

Fig. 3. Block diagram of the proposed background subtraction

For the pixels shown in Fig.2. with coordinates (100,400) and (450,400), Fig. 4 shows the absolute deviation from the median. Fig.5 shows the land pixel (100,400) after filtering against the threshold of $\alpha\sigma$ =8.7. Fig.6 shows the water pixel (450,400) after filtering against its own threshold of $\alpha\sigma$ =11.9. In both cases α =1.5.

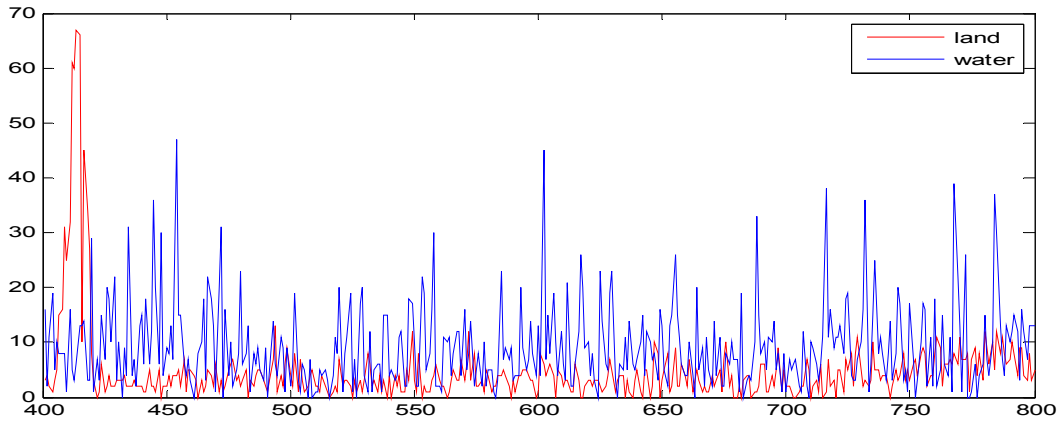

Fig. 4. Absolute deviation from the median (blue line: pixel in water(100,400); red line: pixel on land (450,400)).

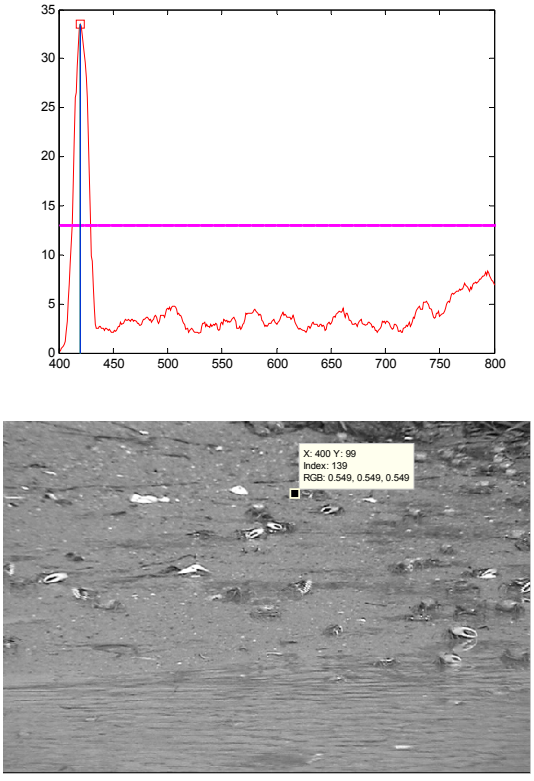

Fig. 5. Absolute deviation from the median after low-pass filtering shown against the computed threshold (8.7), and original frame with pixel at (100,400) identified. At frame#420, an object is passing through this pixel.

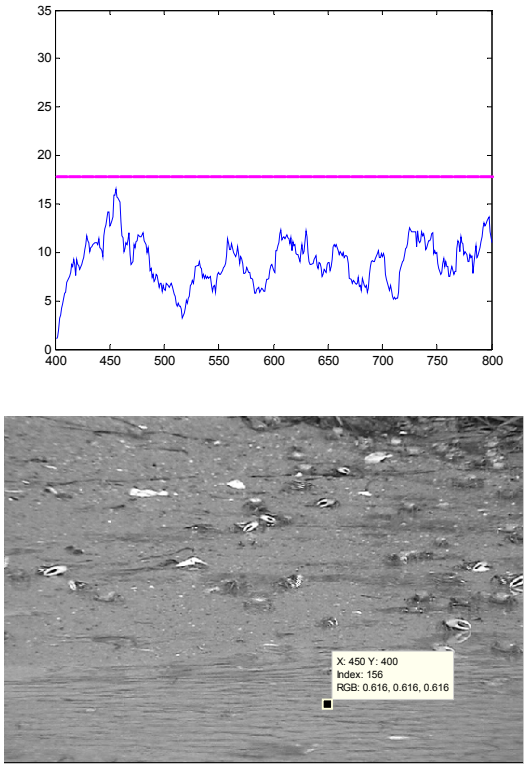

Fig. 6. Absolute deviation from the median after low-pass filtering shown against the computed threshold (11.9), and original frame with pixel at (400,450) identified.

Apparently, in Fig.5, a crab passed by through pixel (100,400) around frame #420, which appears to be a peak in magnitude of intensity value. In Fig.6, there are no crabs crossing pixel (450,400) in the duration of the 400 frames, which corresponds to no intensity exceeding the threshold.

If we collect all the pixels in image planes and in time, we can construct a binary image sequence, and subsequently apply noise suppression and enumeration. Fig 7 displays one of such binary image frames.

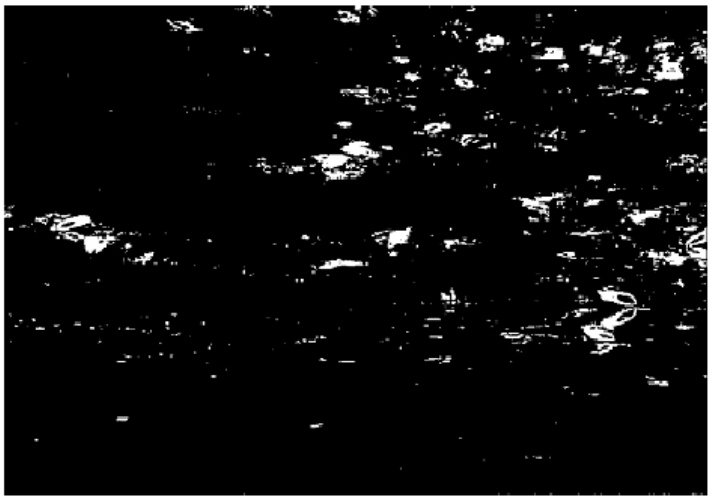

Fig. 7. Final binary result of step 1 in method A

Method B: Density Estimation Model

Stage I.

The method mentioned above could be perfectly implemented for absolutely static background videos. However, the background of the scene contains many non-static objects and even very small movement of the camera could result in large false detections. Thus, we implement another method. In this method, the probability of observing a particular pixel intensity is estimated based on sample intensity values for each pixel captured in time. That is, each single sample of the N samples is considered to be a Gaussian distribution $N(0,\Sigma)$ by itself. This allows us to estimate the density function more accurately and depending only on recent information from the sequence.

Let $x_1, x_2, ..., x_N$ be recent samples of intensity values for a pixel. Using these samples, the probability density function that this pixel will have intensity value x_t at time t can be non-parametrically estimated using the kernel estimator K as follows:

$$P_r(x_t) = \frac{1}{n}\sum_{i=1}^{N} K(x_t - x_i)$$

If we assume independence between the different color channels with different kernel bandwidths, then

$$\Sigma = \begin{pmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_3^2 \end{pmatrix}$$

And the density estimation is reduced to:

$$P_r(x_t) = \frac{1}{N}\sum_{i=1}^{N}\prod_{j=1}^{d}\frac{1}{\sqrt{2\pi\sigma_j^2}}e^{-\frac{1}{2}\frac{(x_{t_j}-x_{i_j})^2}{\sigma_j^2}}$$

Above, the Kernel width is estimated based on the median m of $|x_i - x_{i+1}|$ for each consecutive pair (x_i, x_{i+1}) of samples. They are calculated independently for each color channel. The standard deviation of the first distribution can be estimated as

$$\sigma = \frac{m}{0.68\sqrt{2}}$$

We set the threshold as a global threshold over all images that can be adjusted to achieve a desired percentage of false positives.

Stage II. False alarm suppression:

If the background moves to occupy a new pixel, but it was not part of the model for that pixel, then it will be detected as a foreground object. This happens mostly in the case of slight movement of the camera. We decide if a detected pixel is caused by a background object that has moved by considering the background distributions in a small neighborhood of the detection.

In this stage, we check whether a pixel detected as object is caused by moving objects or the changes of background. Let x_t be the observed value of a pixel, and if x_t be detected as the foreground pixel by the first stage of the background subtraction at time t. We define $N(x_t)$ as the neighbor of x_t, the estimated probability computed in stage I as $P_r(x_t)$.

We replace $P_r(x_t)$ with probability $P_N(x_t)$ estimated in a small neighborhood. $P_N(x_t)$ is defined as:

$$P_N(x_t) = \max_{y \in \{N(x) \cup x_t\}} P_r(y)$$

where $P_r(y)$ is calculated using the kernel function estimation in stage I. We define the neighbor area $N(x)$ as a $n \times n$ rectangular. Fig.8 shows one result of the above false positives suppression.

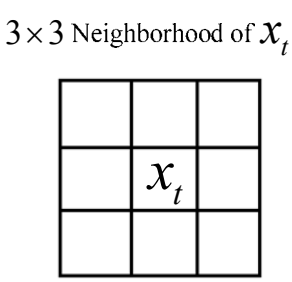

Fig. 8. Result of background subtraction using the density estimation model before (top) and after (middle) false positives suppression using a 3*3 neighborhood, and the original image (bottom) with the used neighborhood of x_t

Fig. 9 shows another example when serious movement of the camera occurs (extracted from Frame #16).

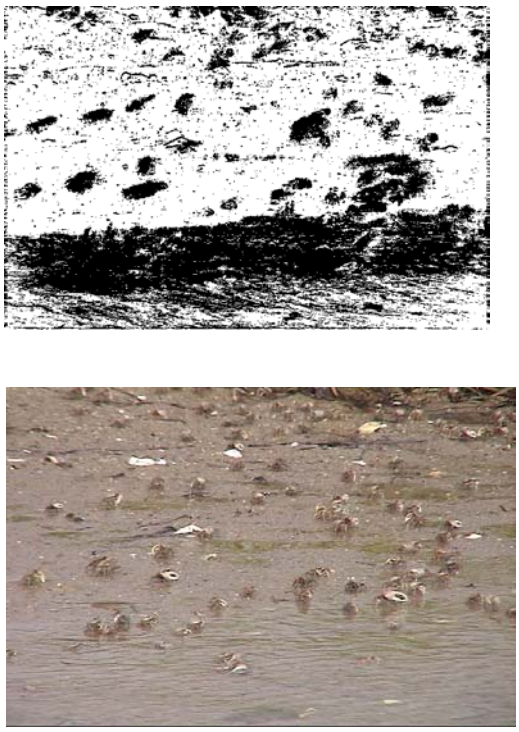

Fig. 9. Result of background subtraction using the density estimation model before (top) and after (middle) false positives suppression using a 3*3 and 7*7 neighborhood, and the original image (bottom).

The model can handle situations where the background of the scene is cluttered and not completely static but contains small motions. The model adapts quickly to changes in the scene which enables very sensitive detection of moving targets.

Step II: Post Processing

Once we get the binary images, we can see that there is still a lot of noise present. To reduce the noise, we use the filter mask K to filter the images.

$$K=\begin{bmatrix} 1/8 & 1/8 & 1/8 \\ 1/8 & 0 & 1/8 \\ 1/8 & 1/8 & 1/8 \end{bmatrix}$$

K is an averaging filter.

For pixel R(i,j) in the image, we can define its 3-by-3 neighborhood as follows:

$$R(i,j)=\begin{bmatrix} R(i-1,j-1) & R(i-1,j) & R(i-1,j+1) \\ R(i,j-1) & R(i,j) & R(i,j+1) \\ R(i+1,j-1) & R(i+1,j) & R(i+1,J+1) \end{bmatrix}$$

We convolve the image with the filter K, to get a filtered image; every pixel A(i,j) in the new image is the average value of the neighborhood of the corresponding pixel R(i,j) in the old image.

Then, we compare the two corresponding pixels:

$$R(i,j)=\begin{cases} 1-R(i,j) & \text{if } |R(i,j)-A(i,j)| > 0.5 \\ R(i,j) & \text{otherwise} \end{cases}$$

The equation above indicates that if a pixel is surrounded by pixels with mostly different values, we change the value of this pixel, however if the pixel has the same value as most of the pixels in the neighborhood, it will remain the same. Fig.9 shows the result based on the first and the second method.

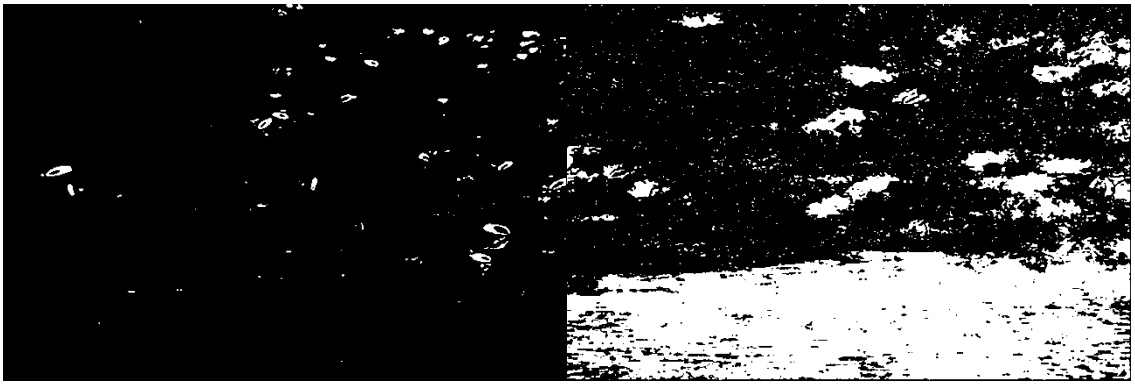

Fig. 10. Post-processed results of background subtraction using median subtraction (left) and probability density estimation (right).

After filtering, we apply morphology operations to the images. Morphology is a broad set of image processing operations that process images based on shapes. The most basic morphological operations are dilation and erosion.

Dilation adds pixels to the boundaries of objects, while erosion removes pixels from object boundaries. The combination of dilation and erosion is usually called the “open” operation. In this case, we first erode the image, and then dilate it. Between these two steps, we also remove some isolated pixels.

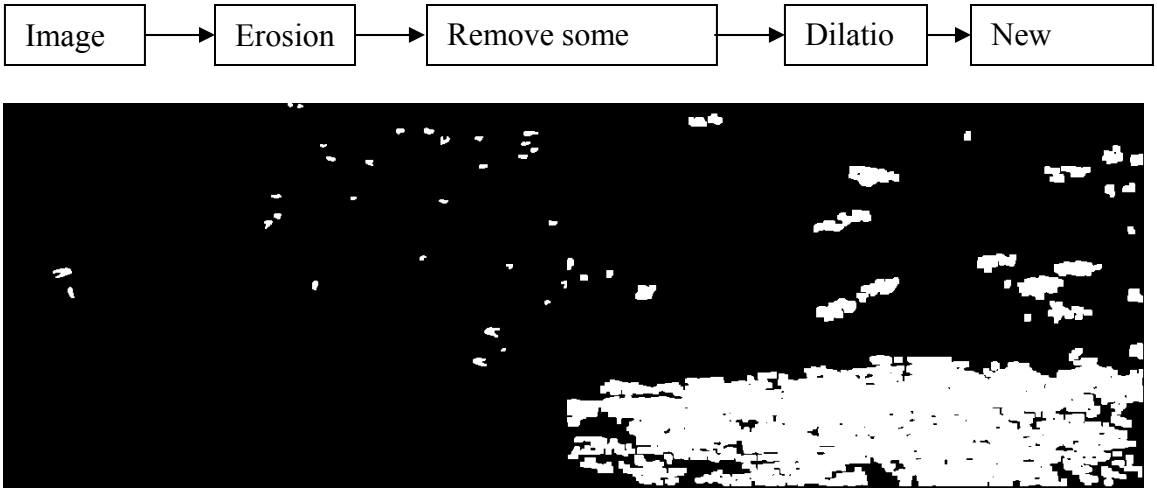

Fig. 11. Post-processed and morphologically-corrected results of background subtraction using median subtraction (left) and probability density estimation (right).

Step III: Enumeration and result comparison.

Based on the images we obtained in the previous steps, we can count the number of white regions that indicate locations of the crabs. The main idea of counting is labeling every white region in the images. The results of this counting for the 2 methods are shown in Fig.10 and Fig.11.

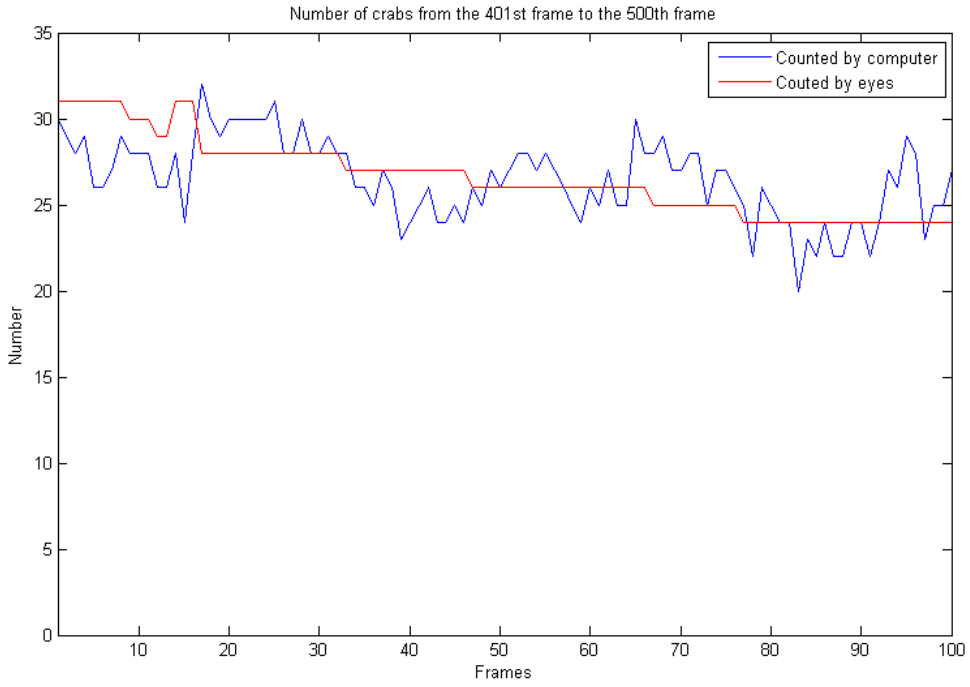

Fig. 12. Number of crabs estimated over time. Blue curve: Result computed by background subtraction method; Red curve: Result obtained by us.

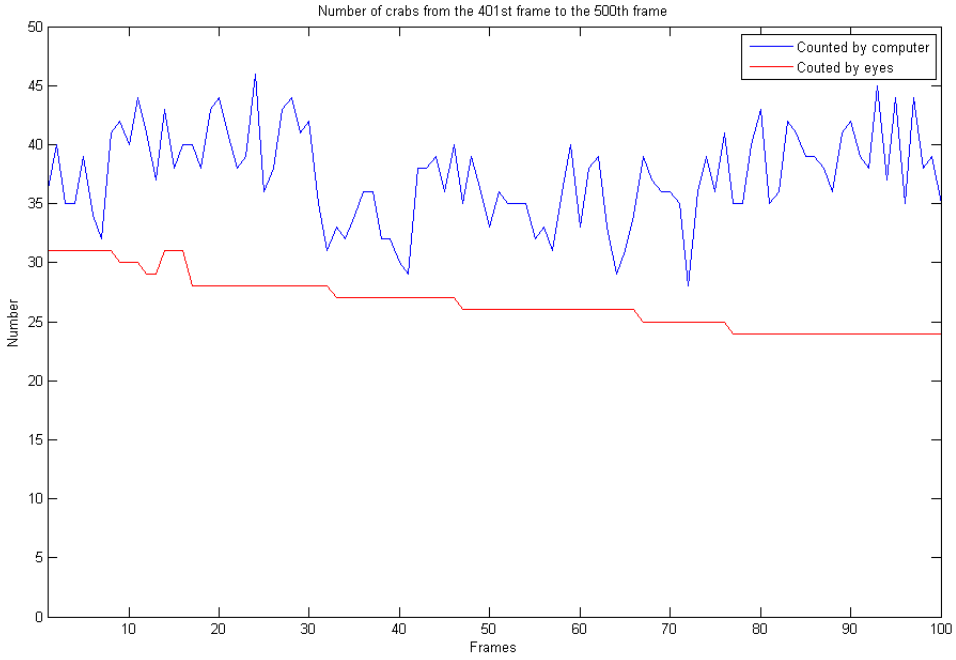

Fig. 13. Number of crabs estimated over time. Blue curve: Result computed by probability density estimation method; Red curve: Result obtained by us

3. Comparison of two methods and further improvement

The proposed adaptive threshold background subtraction method can obtain very high accuracy (accuracy of 90% compared with the probability estimation method that shows only 70% accuracy) but is sensitive to background movement. Probability density estimation method is robust to background change. However, it appears to be blind to water. The reason is because the Gaussian model we established in this project is unable to adapt quickly enough to the background change. The selection of model parameters is crucial here since we face the following tradeoff. If the background model adapts too slowly to changes in the scene, we will construct an inaccurate model that will have low detection sensitivity. On the other hand, if the model adapts too quickly, the moving objects would be very likely detected as background movement. In order to solve these problems, more experiments and analysis would need to be carried out.

4. References

[1] Otsu, N. "A Threshold Selection Method from Gray-Level Histograms," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 9, No. 1, 1979, pp. 62-66.

[2] Elgammal A, Harwood D, Davis L. “Non- parametric Model for Background Subtraction”. Proceedings of the 6th European Conference on Computer Vision, 2000:751-767.

Appendix: Matlab source code of the key methods we used or tried:

Matlab source code we used to collect frames and calculate background model.

```
function [Frames , bg] = collectframes2(filename,high)
% format: e.g. [Frames , bg] = collectframes2('crabs',50);
% the # of elements is double 'high'.

for n = 1:high
    readimages = aviread(filename,n);
    imshow(readimages.cdata);
    Frames(n) = getframe;
end
movie(Frames,0,12);

% Background Calculation
figure;
for i = 1:length(Frames(1).cdata(:,1,1))
    for n=1:high
        M1(n,:) = Frames(n).cdata(i,:,1);
        M2(n,:) = Frames(n).cdata(i,:,2);
        M3(n,:) = Frames(n).cdata(i,:,3);
    end
    bg(i,:,1)=round(median(M1));
    bg(i,:,2)=round(median(M2));
    bg(i,:,3)=round(median(M3));
end
imshow(uint8(bg));
```

Matlab source code that has the same effect for collecting frames and background calculation, meanwhile, it removes sawtooth edges caused by camera shooting.

```
function [Frames , bg] = collectframes(filename,high)
% format: e.g. [Frames , bg] = collectframes('crabs',50);
% the # of elements is double 'high'.

for n = 1:high
    readimages = aviread(filename,n);
    imshow(readimages.cdata);
    PreFrames(n) = getframe;
end
movie(PreFrames,1,12);
% ensure image height evens
for n= 1: high
    PreFrames(n).cdata(length(PreFrames(n).cdata(:,1,1)),:,:)=[];
end

% Lowpass Filter
f = [0 0.45 0.55 1];
a = [1 1 0 0];
lowpass = firpm(17,f,a); % experiments proove that order's better be odd!

% reconstruct vedio and saving to 'Frames'
Frames=PreFrames;
for n = 1:high

        Frames(2*n).cdata(:,:,1) = reconstr(PreFrames(n).cdata(:,:,1),lowpass);
        Frames(2*n).cdata(:,:,2) = reconstr(PreFrames(n).cdata(:,:,2),lowpass);
        Frames(2*n).cdata(:,:,3) = reconstr(PreFrames(n).cdata(:,:,3),lowpass);
```

```
            Frames(2*n-1).cdata(:,:,1)=
reconstr2(PreFrames(n).cdata(:,:,1),lowpass);
            Frames(2*n-1).cdata(:,:,2)=
reconstr2(PreFrames(n).cdata(:,:,2),lowpass);
            Frames(2*n-1).cdata(:,:,3)=
reconstr2(PreFrames(n).cdata(:,:,3),lowpass);
end
movie(Frames,1,24);

% Background Calculation
figure;
for i = 1:length(Frames(1).cdata(:,1,1))
      for n=1:high
            M1(n,:) = Frames(n).cdata(i,:,1);
            M2(n,:) = Frames(n).cdata(i,:,2);
            M3(n,:) = Frames(n).cdata(i,:,3);
      end
      bg(i,:,1)=round(median(M1));
      bg(i,:,2)=round(median(M2));
      bg(i,:,3)=round(median(M3));
end
imshow(uint8(bg));
```

Calculate Difference Images from the background, the simplest background subtraction method.

```
function Diff = Calcu_DiffImage ( Frames, bg, high )
% Calculate difference images
% e.g. Diff = Calcu_DiffImage ( Frames, bg400, high )
```

```
Diff=Frames;
for n = 1:high
    Diff(n).cdata(:,:,1) = Frames(n).cdata(:,:,1) - bg(:,:,1);
    Diff(n).cdata(:,:,2) = Frames(n).cdata(:,:,2) - bg(:,:,2);
    Diff(n).cdata(:,:,3) = Frames(n).cdata(:,:,3) - bg(:,:,3);
end
movie(Frames,0,24);hold;
movie(Diff,0,24);
```

Background subtraction method using adaptive threshold.

```
function bw = gray_time (Frames, windowlength)
% bw = gray_time(Frames,20);

[row,col] = size(Frames(1).cdata);
for i=1:row
    for j=1:col
        for n=1:400
            curve(n)=Frames(n).cdata(i,j);
        end
        mean = median(curve);
        thresh = var(curve);

        curve=abs(curve-mean);
        windowSize = windowlength;
        filt_curve = filter(ones(1,windowSize),windowSize,double(curve));

        mm=zeros(1,length(curve));
        mm(find(filt_curve>2*sqrt(thresh))) = 1;
        bw(i,j,:)= mm(20:10:60);
```

```
    end
end
```

Probability Density Estimation Method

```
function P = Self_Adapt_Diff(Frames, high, N, threshold)
% P = Self_Adapt_Diff(Frames,35,30,0.1);
% N: # of frames used to calculate probability distribution
for i= N+1:high
% calculate kernel width estimation
    for n=1:N
        I1{n}=(abs(Frames(i+1-n).cdata(:,:,1)-Frames(i-n).cdata(:,:,1)));
        I2{n}=(abs(Frames(i+1-n).cdata(:,:,2)-Frames(i-n).cdata(:,:,2)));
        I3{n}=(abs(Frames(i+1-n).cdata(:,:,3)-Frames(i-n).cdata(:,:,3)));
    end
            % calculate median of |xi-xi+1|
    for m = 1:length(Frames(1).cdata(:,1,1))
        for n=1:N
            A1(n,:) = I1{n}(m,:); %Frames(n).cdata(m,:,1);
            A2(n,:) = I2{n}(m,:);
            A3(n,:) = I3{n}(m,:);
        end
        M1(m,:)=double(mean(A1));
        M2(m,:)=double(mean(A2));
        M3(m,:)=double(mean(A3));
    end
    Q1=(M1./(0.68*1.414));
    Q2=(M2./(0.68*1.414));
    Q3=(M3./(0.68*1.414));

    for m= 1:length(Frames(1).cdata(:,1,1))
```

```
            for n= 1:length(Frames(1).cdata(1,:,1))
                if Q1(m,n)==0
                    Q1(m,n)=0.1;
                end
                if Q2(m,n)==0
                    Q2(m,n)=0.1;
                end
                if Q3(m,n)==0
                    Q3(m,n)=0.1;
                end
            end
        end

        P{i}=0;
        for n=1:N
            II1 =
(1./sqrt((2*pi*Q1.^2))).*exp((-1/2)*double(Frames(i).cdata(:,:,1)-Frames(i-n).cdata
(:,:,1)).^2./Q1.^2);
            II2 =
(1./sqrt((2*pi*Q2.^2))).*exp((-1/2)*double(Frames(i).cdata(:,:,2)-Frames(i-n).cdata
(:,:,2)).^2./Q2.^2);
            II3 =
(1./sqrt((2*pi*Q3.^2))).*exp((-1/2)*double(Frames(i).cdata(:,:,3)-Frames(i-n).cdata
(:,:,3)).^2./Q3.^2);
            III = II1.*II2.*II3;
            P{i} = P{i}+III;
        end
        P{i}=P{i}./N;
end
```

Noise Suppression

```
for m =1+mar : row-mar%%%%%%%%%%%
        for n = 1+mar : col-mar%%%%%%%%%%%%
            if P_bw{i}(m,n)
                continue;
            else
                k=1;
                for p = m-mar : m+mar
                    for q = n-mar : n+mar
                        mask(k) = Frames(i).cdata(p,q);
                        k=k+1;
                    end
                end
                for p = 1:k-1
                    X(p) = 0;
                    for q=1:N
                        II =
(1./sqrt((2*pi*Q1(m,n).^2))).*exp((-1/2)*double(mask(p)-Frames(i-q).cdata(m,n)).^
2./Q1(m,n).^2);
                        X(p) = X(p)+II;
                    end
                    X(p) = X(p)./N;
                end
                P{i}(m,n) = max(X);
            end
        end
    end
    P{i}=im2bw(P{i},threshold);
end
```

```
%post processing
N=zeros(1,100);
for k=1:1:100
     K=[1/8 1/8 1/8
        1/8 0 1/8
        1/8 1/8 1/8];
    D=double(S(:,:,k));
    A=conv2(D,K);
    C=A(2:1:482,2:1:722);
    for i=1:1:481
        for j=1:1:721
            if abs(D(i,j)-C(i,j))>0.5
              S(i,j,k)=1-S(i,j,k);
            end
        end
    end
    %morphology operation
    SE=strel('square',3);
    S(:,:,k)=imerode(S(:,:,k),SE);
    S(:,:,k)=bwareaopen(S(:,:,k),8);
    S(:,:,k)=imdilate(S(:,:,k),SE);
    [l num]=bwlabel(S(:,:,k));
    N(1,k)=num;
end
%play

%Draw the plot
plot(N,'b')
axis([1 100 0 35])
title 'Number of crabs from the 401st frame to the 500th frame'
xlabel('Frames')
```

```
ylabel('Number')
hold;
t(1,1:8)=31;t(1,9:11)=30;t(1,12:13)=29;t(1,14:16)=31;t(1,17:32)=28;t(1,33:46)=27;
t(1,47:66)=26;t(1,67:76)=25;t(1,77:100)=24;
plot(t,'r');
legend('Counted by computer','Couted by eyes')
```

Method 2

```
for i=1:1:100
    S(:,:,i)=P{i+15};
end
for i=1:1:481
    for j=1:1:721
        S(i,j,:)=1-S(i,j,:);
    end
end
%postprocessing
N=zeros(1,100);
for k=1:1:100
    K=[1/8 1/8 1/8
       1/8 0 1/8
       1/8 1/8 1/8];
  D=double(S(:,:,k));
  A=conv2(D,K);
  C=A(2:1:482,2:1:722);
  for i=1:1:481
      for j=1:1:721
          if abs(D(i,j)-C(i,j))>0.5
            S(i,j,k)=1-S(i,j,k);
          end
      end
```

```
    end
    %morphology operation
    SE=strel('square',6);
    S(:,:,k)=imerode(S(:,:,k),SE);
    S(:,:,k)=bwareaopen(S(:,:,k),12);
    S(:,:,k)=imdilate(S(:,:,k),SE);
    [l num]=bwlabel(S(:,:,k));
    N(1,k)=num;
end
%Draw the plot
plot(N,'b')
axis([1 100 0 50])
title 'Number of crabs from the 401st frame to the 500th frame'
xlabel('Frames')
ylabel('Number')
hold;
t(1,1:8)=31;t(1,9:11)=30;t(1,12:13)=29;t(1,14:16)=31;t(1,17:32)=28;t(1,33:46)=27;
t(1,47:66)=26;t(1,67:76)=25;t(1,77:100)=24;
plot(t,'r');
legend('Counted by computer','Couted by eyes')
```