ECG 782 HW 4

Carlo Lopez-Tello

2015-10-15

1 Problem 10.49

1.1 Maximum exposure time

We are given the dimensions of the image 256 by 256 pixels. We now that the bullet takes up 10 percent of the width of the image and that the bullet is 3 cm long. This implies that the image is 30 cm by 30 cm. Thus, each pixel is .12 cm by .12 cm. The exposure time has to be fast enough so that the bullet can only move .12 cm or 1 pixel. We can calculate the exposure time using $.12cm = speed(exposure)$. We know the maximum speed of the bullet is 900 m/s so we can calculate the exposure time to be 1.33 us.

1.2 Minimum FPS

We want the time between frames to be fast enough so that the bullet is always captured in two consecutive frames. In other words the bullet cannot move more than half the width of the image between frames. The problem with this limit is that we can have half a bullet in one frame if we capture exactly as the bullet hits the midpoint of the image. Thus the bullet cannot travel more than half the width minus the length of the bullet minus the two pixels from motion blur $128 - 26 - 2 = pixels$. We can convert 100 pixels to the actual distance of 12 cm. Thus, the maximum distance that a bullet can move between frames is 12 cm. We can calculate the frame rate using $12cm(FPS) = speed$. For a bullet traveling at the max speed of 900 m/s, the frame rate should be higher than 7500 FPS.

1.3 Bullet Segmentation

Assuming we have control over the scene. Meaning that we have constant illumination and background. We can use simple frame differencing between the recording and an image of the background.

1.4 Calculation bullet speed

We can easily calculate the speed of the bullet if we know how many pixels it moved between frames. In order to do this we need to find a point corresponding to the same part of the bullet in both images. We can use a corner of the bullet and try to match it to the next frame. Once we know the number of pixel the bullet moved we can calculate the distance and divide it by the time between frames.

2 Background Subtraction

2.1 simple frame differencing

Simple frame differencing works well on the white cars but not on the black one. The problem is that the color of the car is close to the color of the background.

2.2 static background

Using a static background works better than frame differencing on the white cars. However, it performs worse on the black car. This occurs because there is a big contrast difference between the white cars and the background.

2.3 static average background

current frame 1.png

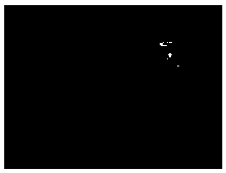
background frame 1.png

object frame 1.png

current frame 45.png

background frame 45.png

object frame 45.png



current frame 91.png

background frame 91.png

object frame 91.png

The average of the frames is almost identical to the static background used before. Therefore the results are similar.

2.4 adaptive background

current frame 1.png

background frame 1.png

object frame 1.png

current frame 45.png

background frame 45.png

object frame 45.png

current frame 91.png

background frame 91.png

object frame 91.png

The alpha value used was .25. Using an adaptive background produces results similar to frame differencing. With the current alpha value, the procedure works better on the white cars than the black one. One could adjust the alpha value, but the results would get worse for the white cars.

3 Gaussian Mixture Model

3.1 foreground extraction

current frame 1.png

foreground frame 1.png

current frame 45.png

foreground frame 45.png

current frame 91.png

foreground frame 91.png

3.2 clean foreground

current frame 1.png

foreground frame 1.png

current frame 45.png

foreground frame 45.png

current frame 91.png

foreground frame 91.png

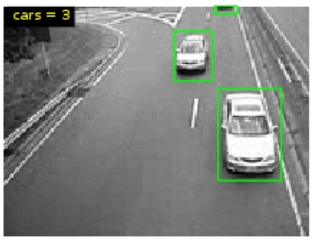
In order to make the object boundaries clearer I applied dilation with a square structuring element, and then erosion with a rectangular one.

3.3 vehicle detection

current frame 1.png

current frame 45.png

current frame 91.png



4 Optical Flow

4.1 lucas kanade

current frame 1.png

current frame 45.png

current frame 91.png

4.2 horn shunk

frame1.png

frame45.png

frame91.png

Both algorithms work better on the white cars than the black one. LK appears to produce tighter flows, meaning the flow vectors are closer together. One the black car the results look better using LK, since even when the vectors are in the wrong place they are grouped together. Using HS on the black car produces a flow diagram where the vectors are all over the place.

5 Code

Part 2.A partA.cpp

```
#include "opencv2/opencv.hpp"
#include <string>
using namespace cv;
using namespace std;

////////////////////////////////functions
Mat readFrame(VideoCapture& capture, int frameNumber)
{
        Mat frame;
        capture.set(CV_CAP_PROP_POS_FRAMES,frameNumber);
    bool success = capture.read(frame);
    if (!success)
    {
        cerr << "Cannot read frame " <<  frameNumber << endl;
    }
    return frame;
}

////////////////////////////////main
int main(int argc, char** argv)
{
    if (argc != 2)
    {
        cout << "need a filename\n";
        return 1;
    }
    std::string filename = argv[1];

    VideoCapture capture(filename);

    if (!capture.isOpened())
    {
        cerr << "Failed to open file\n";
        return 1;
    }
    //int totalFrames = static_cast<int>(capture.get(CV_CAP_PROP_FRAME_COUNT))
        ;

    int frameIndex[3] = {1,45,91};

    Mat currentFrame;
    Mat backgroundFrame;
    Mat frameDif;
    char buffer[100];

    for(int i=0;i<3;i++)
    {
        currentFrame = readFrame(capture, frameIndex[i]);
        cvtColor( currentFrame, currentFrame, CV_BGR2GRAY );

        backgroundFrame = readFrame(capture, frameIndex[i]-1);
```

```
        cvtColor( backgroundFrame, backgroundFrame, CV_BGR2GRAY );

        frameDif = currentFrame - backgroundFrame;
        threshold(frameDif, frameDif, 50, 255, THRESH_BINARY);

        snprintf(buffer,100,"object frame %d.png", frameIndex[i]);
        string filename = buffer;
        imwrite( filename, frameDif);

        snprintf(buffer,100,"current frame %d.png", frameIndex[i]);
        filename = buffer;
        imwrite( filename, currentFrame);

        snprintf(buffer,100,"background frame %d.png", frameIndex[i]);
        filename = buffer;
        imwrite( filename, backgroundFrame);
    }
     waitKey(0);
     return 0;

}
```

partA.m

```
system("cmake .");
system("make");
system("rm -rf CMakeFiles");
system("rm CMakeCache.txt");
system("rm cmake_install.cmake");
system("rm Makefile");
system("./partA ../../traffic.avi");
figure(1)
k = 1;
for i = [1 45 91];
        subplot(3,3,k);
        str = sprintf("current frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
        subplot(3,3,k);
        str = sprintf("background frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
        subplot(3,3,k);
        str = sprintf("object frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
end
```

Part 2.B partB.cpp

```
#include "opencv2/opencv.hpp"
#include <string>
using namespace cv;
using namespace std;

//////////////////////////////////functions
Mat readFrame(VideoCapture& capture, int frameNumber)
{
        Mat frame;
        capture.set(CV_CAP_PROP_POS_FRAMES,frameNumber);
    bool success = capture.read(frame);
    if (!success)
    {
        cerr << "Cannot read frame " <<  frameNumber << endl;
    }
    return frame;
}

//////////////////////////////////main
int main(int argc, char** argv)
{
    if (argc != 2)
    {
        cout << "need a filename\n";
        return 1;
    }
    std::string filename = argv[1];

    VideoCapture capture(filename);

    if (!capture.isOpened())
    {
        cerr << "Failed to open file\n";
        return 1;
    }
    int totalFrames = static_cast<int>(capture.get(CV_CAP_PROP_FRAME_COUNT));


    Mat backgroundFrame = readFrame(capture, totalFrames-1);
    cvtColor( backgroundFrame, backgroundFrame, CV_BGR2GRAY );
    imwrite( "backgroundFrame.png", backgroundFrame);

    Mat currentFrame;
    Mat frameDif;

    char buffer[100];

    int frameIndex[3] = {1,45,91};

    for(int i=0;i<3;i++)
    {
        currentFrame = readFrame(capture, frameIndex[i]);
```

```
        cvtColor( currentFrame, currentFrame, CV_BGR2GRAY );

        frameDif = currentFrame - backgroundFrame;
        threshold(frameDif, frameDif, 50, 255, THRESH_BINARY);

        snprintf(buffer,100,"object frame %d.png", frameIndex[i]);
        string filename = buffer;

        //imshow( filename, frameDif);
        imwrite( filename, frameDif);

        snprintf(buffer,100,"current frame %d.png", frameIndex[i]);
        filename = buffer;

        //imshow( filename, currentFrame);
        imwrite( filename, currentFrame);

        snprintf(buffer,100,"background frame %d.png", frameIndex[i]);
        filename = buffer;
        imwrite( filename, backgroundFrame);
    }
     waitKey(0);
     return 0;

}
```

partB.m

```
system("cmake .");
system("make");
system("rm -rf CMakeFiles");
system("rm CMakeCache.txt");
system("rm cmake_install.cmake");
system("rm Makefile");
system("./partB ../../traffic.avi");
figure(1)
k = 1;
for i = [1 45 91];
        subplot(3,3,k);
        str = sprintf("current frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
        subplot(3,3,k);
        str = sprintf("background frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
        subplot(3,3,k);
        str = sprintf("object frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
end
```

Part 2.C partC.cpp

```
#include "opencv2/opencv.hpp"
#include <string>
using namespace cv;
using namespace std;

//////////////////////////////////functions
Mat readFrame(VideoCapture& capture, int frameNumber)
{
        Mat frame;
        capture.set(CV_CAP_PROP_POS_FRAMES,frameNumber);
    bool success = capture.read(frame);
    if (!success)
    {
        cerr << "Cannot read frame " <<  frameNumber << endl;
    }
    return frame;
}

//////
//string ty =  type2str( M.type() );
//printf("Matrix: %s %dx%d \n", ty.c_str(), M.cols, M.rows );
string type2str(int type) {
  string r;

  uchar depth = type & CV_MAT_DEPTH_MASK;
  uchar chans = 1 + (type >> CV_CN_SHIFT);

  switch ( depth ) {
    case CV_8U:  r = "8U"; break;
    case CV_8S:  r = "8S"; break;
    case CV_16U: r = "16U"; break;
    case CV_16S: r = "16S"; break;
    case CV_32S: r = "32S"; break;
    case CV_32F: r = "32F"; break;
    case CV_64F: r = "64F"; break;
    default:     r = "User"; break;
  }

  r += "C";
  r += (chans+'0');

  return r;
}

//////////////////////////////main
int main(int argc, char** argv)
{
    if (argc != 2)
    {
        cout << "need a filename\n";
        return 1;
    }
```

```
std::string filename = argv[1];

VideoCapture capture(filename);

if (!capture.isOpened())
{
    cerr << "Failed to open file\n";
    return 1;
}
char buffer[100];
//capure returns uint8 matrices
int totalFrames = static_cast<int>(capture.get(CV_CAP_PROP_FRAME_COUNT));

Mat sum = readFrame(capture, 0);
sum.convertTo(sum,CV_32F);

for(int i=1;i<totalFrames;i++)
{
    Mat frame;
    frame = readFrame(capture, i);
    frame.convertTo(frame,CV_32F);
    sum += frame;
}

Mat backgroundFrame = sum/totalFrames;

cvtColor( backgroundFrame, backgroundFrame, CV_BGR2GRAY );
imwrite( "backgroundFrame.png", backgroundFrame);

Mat currentFrame;
Mat frameDif;

int frameIndex[3] = {1,45,91};
for(int i=0;i<3;i++)
{
    currentFrame = readFrame(capture, frameIndex[i]);
    currentFrame.convertTo(currentFrame,CV_32F);
    cvtColor( currentFrame, currentFrame, CV_BGR2GRAY );

    frameDif = currentFrame - backgroundFrame;
    threshold(frameDif, frameDif, 50, 255, THRESH_BINARY);

    snprintf(buffer,100,"object frame %d.png", frameIndex[i]);
    filename = buffer;

    //imshow( filename, frameDif);
    imwrite( filename, frameDif);

    snprintf(buffer,100,"current frame %d.png", frameIndex[i]);
    filename = buffer;

    //imshow( filename, currentFrame);
    imwrite( filename, currentFrame);
```

```
        snprintf(buffer,100,"background frame %d.png", frameIndex[i]);
        filename = buffer;
        imwrite( filename, backgroundFrame);
    }
     waitKey(0);
     return 0;

}
```

partC.m

```
system("cmake .");
system("make");
system("rm -rf CMakeFiles");
system("rm CMakeCache.txt");
system("rm cmake_install.cmake");
system("rm Makefile");
system("./partC ../../traffic.avi");
figure(1)
k = 1;
for i = [1 45 91];
        subplot(3,3,k);
        str = sprintf("current frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
        subplot(3,3,k);
        str = sprintf("background frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
        subplot(3,3,k);
        str = sprintf("object frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
end
```

Part 2.D partD.cpp

```
#include "opencv2/opencv.hpp"
#include <string>
using namespace cv;
using namespace std;

//////////////////////////////////functions
Mat readFrame(VideoCapture& capture, int frameNumber)
{
        Mat frame;
        capture.set(CV_CAP_PROP_POS_FRAMES,frameNumber);
    bool success = capture.read(frame);
    if (!success)
    {
        cerr << "Cannot read frame " <<  frameNumber << endl;
    }
    return frame;
}

//////
//string ty =  type2str( M.type() );
//printf("Matrix: %s %dx%d \n", ty.c_str(), M.cols, M.rows );
string type2str(int type) {
  string r;

  uchar depth = type & CV_MAT_DEPTH_MASK;
  uchar chans = 1 + (type >> CV_CN_SHIFT);

  switch ( depth ) {
    case CV_8U:  r = "8U"; break;
    case CV_8S:  r = "8S"; break;
    case CV_16U: r = "16U"; break;
    case CV_16S: r = "16S"; break;
    case CV_32S: r = "32S"; break;
    case CV_32F: r = "32F"; break;
    case CV_64F: r = "64F"; break;
    default:     r = "User"; break;
  }

  r += "C";
  r += (chans+'0');

  return r;
}

//////////////////////////////main
int main(int argc, char** argv)
{
    if (argc != 2)
    {
        cout << "need a filename\n";
        return 1;
    }
```

```
std::string filename = argv[1];

VideoCapture capture(filename);

if (!capture.isOpened())
{
    cerr << "Failed to open file\n";
    return 1;
}
float alpha = .25;
char buffer[100];
//capure returns uint8 matrices
int totalFrames = static_cast<int>(capture.get(CV_CAP_PROP_FRAME_COUNT));

Mat background[totalFrames];

background[0] = readFrame(capture, 0);
cvtColor( background[0], background[0], CV_BGR2GRAY );
background[0].convertTo(background[0],CV_32F);

for(int i=1;i<totalFrames;i++)
{
    Mat frame;
    frame = readFrame(capture, i);
    cvtColor( frame, frame, CV_BGR2GRAY );
    frame.convertTo(frame,CV_32F);
    background[i] = (1-alpha)*background[i-1] + alpha*frame;
}


Mat currentFrame;
Mat frameDif;
Mat backgroundFrame;

int frameIndex[3] = {1,45,91};
for(int i=0;i<3;i++)
{
    currentFrame = readFrame(capture, frameIndex[i]);
    currentFrame.convertTo(currentFrame,CV_32F);
    cvtColor( currentFrame, currentFrame, CV_BGR2GRAY );


    backgroundFrame = background[frameIndex[i]];

    frameDif = currentFrame - backgroundFrame;
    threshold(frameDif, frameDif, 50, 255, THRESH_BINARY);

    snprintf(buffer,100,"object frame %d.png", frameIndex[i]);
    filename = buffer;

    //imshow( filename, frameDif);
    imwrite( filename, frameDif);

    snprintf(buffer,100,"current frame %d.png", frameIndex[i]);
```

```
        filename = buffer;

        //imshow( filename, currentFrame);
        imwrite( filename, currentFrame);

        snprintf(buffer,100,"background_frame_%d.png", frameIndex[i]);
        filename = buffer;
        imwrite( filename, backgroundFrame);
    }
     waitKey(0);
     return 0;

}
```

partD.m

```
system("cmake .");
system("make");
system("rm -rf CMakeFiles");
system("rm CMakeCache.txt");
system("rm cmake_install.cmake");
system("rm Makefile");
system("./partD ../../traffic.avi");
figure(1)
k = 1;
for i = [1 45 91];
        subplot(3,3,k);
        str = sprintf("current frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
        subplot(3,3,k);
        str = sprintf("background frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
        subplot(3,3,k);
        str = sprintf("object frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
end
```

Part 3.A partA.cpp

```
#include "opencv2/opencv.hpp"
#include <string>
#include "opencv2/bgsegm.hpp"

using namespace cv;
using namespace bgsegm;
using namespace std;

////////////////////////////////functions
Mat readFrame(VideoCapture& capture, int frameNumber)
{
        Mat frame;
        capture.set(CV_CAP_PROP_POS_FRAMES,frameNumber);
    bool success = capture.read(frame);
    if (!success)
    {
        cerr << "Cannot read frame " <<  frameNumber << endl;
    }
    return frame;
}

////////////////////////////////main
int main(int argc, char** argv)
{
    if (argc != 2)
    {
        cout << "need a filename\n";
        return 1;
    }
    std::string filename = argv[1];
    //matlab defaults
    int history = 150;
    int nmixtures = 5;
    double backgroundRatio = .7;
    double noiseSigma = 30^2;

    VideoCapture capture(filename);
    Ptr<BackgroundSubtractorMOG> pmog;
    pmog = createBackgroundSubtractorMOG(history, nmixtures, backgroundRatio,
        noiseSigma);



    if (!capture.isOpened())
    {
        cerr << "Failed to open file\n";
        return 1;
    }
    int totalFrames = static_cast<int>(capture.get(CV_CAP_PROP_FRAME_COUNT));

    //int frameIndex[3] = {1,45,91};
```

```
    Mat currentFrame;
    Mat foregroundFrame;

    char buffer[100];

    for(int i=0;i<totalFrames;i++)
    {

        currentFrame = readFrame(capture, i);
        cvtColor( currentFrame, currentFrame, CV_BGR2GRAY );
        pmog->apply(currentFrame, foregroundFrame);

        if(i == 1 || i == 45 || i == 91)
        {
                        snprintf(buffer,100,"current frame %d.png", i);
                        filename = buffer;
                        imwrite( filename, currentFrame);

                        snprintf(buffer,100,"foreground frame %d.png", i);
                        filename = buffer;
                        imwrite( filename, foregroundFrame);
                }

    }
     waitKey(0);
     return 0;

}
```

partA.m

```
system("cmake .");
system("make");
system("rm -rf CMakeFiles");
system("rm CMakeCache.txt");
system("rm cmake_install.cmake");
system("rm Makefile");
system("./partA ../../traffic.avi");
figure(1)
k = 1;
for i = [1 45 91];
        subplot(3,2,k);
        str = sprintf("current frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        str = sprintf("foreground frame %i.png",i);
        k = k+1;
        subplot(3,2,k);
        str = sprintf("foreground frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
end
```

Part 3.B partB.m

```
system("cmake .");
system("make");
system("rm -rf CMakeFiles");
system("rm CMakeCache.txt");
system("rm cmake_install.cmake");
system("rm Makefile");
system("./partA ../../traffic.avi");
figure(1)
k = 1;
for i = [1 45 91];
        subplot(3,2,k);
        str = sprintf("current frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);
        k = k+1;
        subplot(3,2,k);
        str = sprintf("foreground frame %i.png",i);
        image = cleanImage(image);
        imshow(image);
        title(str);
        k = k+1;
end
```

cleanImage.m

```
function [morphImage] = cleanImage(image)
        structuralElement = strel ("square", 9);
        morphImage = imdilate(image,structuralElement);
        structuralElement = strel ("square", 5);
        morphImage = imerode(morphImage,structuralElement);
end
```

Part 3.C partC.m

```
system("cmake .");
system("make");
system("rm -rf CMakeFiles");
system("rm CMakeCache.txt");
system("rm cmake_install.cmake");
system("rm Makefile");
system("./partA ../../traffic.avi");

k = 1;
figure(1)
for i = [1 45 91];
        subplot(3,1,k)
        str = sprintf("current frame %i.png",i);
        image = imread(str);
        imshow(image);
        title(str);

        str = sprintf("foreground frame %i.png",i);
        image = imread(str);
        morphImage = cleanImage(image);
        labeledImage = bwlabel(morphImage);
        boundingBoxes = regionprops(labeledImage ,'BoundingBox');
        for j=1:rows(boundingBoxes)
                rectangle('Position',boundingBoxes(j,1).BoundingBox,'EdgeColor
                    ','green');
        end
        str = sprintf("cars = %i",rows(boundingBoxes));
        rectangle('Position',[2 1 35 10],'FaceColor','black');
        text(5,5,str,'color','yellow');
        k = k+1;

end
print(figure(1),'asd.png');
```

Part 4.A partA.m

```
v = VideoReader('traffic.avi');
allFrames = read(v);

k = 1;
index = [1,41,91];
optical = opticalFlowLK;
for i=2:size(allFrames,4)
    if(i == 2 || i == 46 || i == 92)
        currentFrame = allFrames(:,:,:,i);
        currentFrame = rgb2gray(currentFrame);
        pastFrame = allFrames(:,:,:,i-1);
        pastFrame = rgb2gray(pastFrame);
        flow = estimateFlow(optical,pastFrame);
        flow = estimateFlow(optical,currentFrame);
        figure(k)
        hold on
        imshow(currentFrame);
        plot(flow);
        hold off
        str = sprintf('image%i',k);
        saveas( gcf,str, 'png' );
        k = k+1;
    end
end
```

Part 4.B partB.m

```
v = VideoReader('traffic.avi');
allFrames = read(v);

k = 1;
optical = opticalFlowHS;
for i=2:size(allFrames,4)
    if(i == 2 || i == 46 || i == 92)
        currentFrame = allFrames(:,:,:,i);
        currentFrame = rgb2gray(currentFrame);
        pastFrame = allFrames(:,:,:,i-1);
        pastFrame = rgb2gray(pastFrame);
        flow = estimateFlow(optical,pastFrame);
        flow = estimateFlow(optical,currentFrame);
        figure(k)
        hold on
        imshow(currentFrame);
        plot(flow,'ScaleFactor',10)
        hold off
        str = sprintf('image%i',k);
        saveas( gcf,str, 'png' );
        k = k+1;
    end
end
```