Samuel Petit
Student no: 17333946
Exam no: 46429

Computer Vision CSU44053 - Final Examination

Question 1a - Application

Let's clarify some details about this problem before jumping into my solution. To begin, professional tennis players are able to play while the tennis ball is moving at very high speeds, hence for some level of accuracy here we will require a video which contains a high frame rate. A frame rate that is too low may result in us missing some bounces. Secondly, with a single point of view we may not be able to detect every single bounce, which could be due to different reasons, one of them being a player blocking our point of view, another one could be when the ball is hidden by one of the nets polls. In reality we would have multiple points of views to solve this problem hence we will be ignoring those edge cases and treat them as acceptable misses.

Now, jumping into my solution, we are dealing with a video hence in this case we would repeat the following instructions on each frame we are iterating over.
The first step would be to remove the background, such as to identify the moving objects of the scene. In this case we would expect to find the players and the ball.
The first step to do so would be to convert the current image to grayscale as we do not need to deal with colours here to solve our problem. We can do so by converting our image, that is for each pixel in the image we can map it to a grayscale value, essentially converting our image from multiple channels (typically 3 for RGB) into a single channel (grayscale) image, conversion from RGB into grayscale uses the following formula:

- $Y = 0.2126R + 0.7152G + 0.0722B$

Now we can proceed with the background detection (or removal), given the circumstances, it seems to me that a gaussian mixture background model would work well here as it should deal with the crowd and slight movements due to wind quite nicely and give us the outputs we are looking for, which in this case would be the ball and the 2 players. To do so, we provide the gaussian mixture model with the grayscale image obtained the previous step. The output from this technique would be a binary image which represents the pixels that are in the foreground. It is worth noting that this background detection model may struggle with fast variations, however given that a fast frame rate is somewhat required for this problem this should not be an issue. The final detail with this technique is that it requires a learning rate (as it is a form of unsupervised learning). Typical values range from 0.01 to 0.1, in this case with the amount of possible movement with the crowd and such I would expect a value closer to 0.1 to work however I could be wrong and we could use cross-validation using a confusion matrix on a range of learning rates in order to identify an ideal value.

We can clean this binary image using morphological dilations and erosions, depending on how that binary image is looking we could apply either an opening operation which would cleanup the noise. In the case that this binary image contains separate close regions we could use a closing operation instead. Once we have identified which technique to use based on the binary images we are observing we should have a clean foreground binary

image. These morphological operations take as input a binary image and outputs a binary image too.

We should see 3 different objects in the binary image, that is the two players and the ball. In order to detect the ball we can attempt to detect the shape of a ball within these objects. To do so I will use Hough circle detection, to do so, we need to obtain the set of edges within our foreground scene. To do so, we can use the binary image as a mask on our original grayscale image such as to obtain an image where every positive pixel in our binary image is replaced with its corresponding grayscale value. This will effectively use our binary image as input and output a grayscale image which would be in this case our foreground image.

Now that we have this foreground grayscale image, we can use it to detect the edges within that image, to do so I will use Canny edge detection. This technique takes a grayscale image as input and will output an edge map as the output array. In terms of parameters there are 2 thresholds that this method uses as parameters, a high and low threshold. These are used by setting anything below the low threshold to a non edge, anything above the high threshold to an edge and values lying within the thresholds range will depend on how connected they are to other edges.
In this case I believe that a relatively high low threshold could be used such as to only really get outlines and not obtain additional edge details that we may not care about. Once again this is quite subjective and in order to find ideal parameters we should use cross-validation using a range of values such as to obtain a set of ideal parameters. Briefly, canny edge detection works by computing the gradient and orientation of the image using its first derivative, then using the second derivation in order to remove non-maximas. These results are then passed through the thresholding explained above.

We now have our edges and can proceed with detecting the balls location using Hough transform. Hough Circle detection will use our edges as inputs and output a set of circles, that is each returned point includes a center (x,y) and a radius r. It will use a circles equation and each parameter will result in a dimension in the accumulator used. Here our Hough circle detection will use 3 parameters x, y: the possible circle centers and r: the circle radius. Here we could attempt to use a constant circle radius however with the perspective of our images, the balls size may change and hence we will use a small range of radiuses. Given our 3D accumulator, we ten need to pick a size for each of the dimensions and then consider each edge point in the accumulator, the ones which mach possible circle centers are incremented, this is where our circle equation comes into play. The final step is to look for maximums in the accumulators.

- The parameters we provide the hough circle detection are the minimum and maximum circle radius we are attempting to detect, in our case we know that the tennis ball will always have a diameter of 10px hence we can set the minimum radius to 4px to be safe. Then the maximum radius parameter can be set to 7px in order to always be able to detect the ball throughout the image.

This method should provide us with a single circle: that is the center and radius of the tennis ball for this frame. We keep track of the center position (x,y) of the ball for each frame in an array.

Our next step is to be able to detect when the ball bounces on the ground, to do so we can use the past ball positions to detect changes in movement. That is when the ball has been going down (a y value lowering) and the next frame sees an increase in y position, we can be pretty certain that one of two things happened:

- The ball bounced on the floor
- A player hit the ball

We can detect the cases where a player hit the ball by checking that the ball and player are not overlapping (in those cases a ball may potentially not be detected). If it is not the case then we know that we have a bounce on the floor.

We can get approximately the bounce position by taking the average position of the current frame and the previous frame (when the ball's y position was lowering). Our next step is to locate where the ball bounced off in the plan view.

To do so we can start by manually trimming the image used for the plan view such as to remove the area that is outside the court is no longer included in the image.
Then, we will need to do a perspective transformation on our foreground grayscale image from a few steps above.

So, we are given the 4 points limiting the tennis court, which means we can use those and the grayscale foreground image as input to our perspective transformation. The output will be the same grayscale foreground image where the perspective has been changed such that it is now flat (like the plan view).

All that is left now, is to scale the resulting grayscale image such that it has the same height and width as the plan view. Since they have the same size and outlines, we can use the position of the circle in the grayscale image which we have changed perspective as the position where the ball bounced in the plan view.

We are now able to identify where the tennis ball is bouncing in the court.

Question 1b - Compare & Contrast

Let's start with a quick introduction to these 3 methods before starting to compare them. Gaussian smoothing is a technique that will attempt to smooth the image by reducing noise, it works by iterating over each of the image's pixels, using a weighted mask (which uses a gaussian distribution) in order to compute the new value of the current pixel. Median smoothing is also a technique that attempts to smooth an image by reducing noise. The method used to do so is by computing the median value of all the pixels within the kernel window. Finally, opening is a method that attempts to clean binary images by using two techniques called dilation and erosion together. The two effects of an opening operation are noise reduction and narrow bridge removal.

Let's jump into comparisons of these techniques, the first thing we notice is all 3 of these techniques share a common use case: they are used to reduce noise, or in other words,

clean images by reducing noise. The first major difference we notice though is in terms of the inputs and outputs of these techniques, both gaussian smoothing and median smoothing take as input an image (either single or multi channel - that is a black and white or colour image can both be used) and will output the processed image in the same format. The opening technique on the other hand takes as input a binary image and will output a binary image: it is used for cleaning binary images unlike both median and gaussian smoothing.

Another difference between these methods is their linearity, gaussian smoothing is a linear technique as a linear function is used to compute the value of the current pixel using the pixels within the kernel. This is not the case for both median smoothing which will compute the median of the pixel values within the kernel, and for the opening operation which will perform a dilation and erosion operation which are non linear operations.

We also find that the opening technique and median smoothing share a similar effect to the output image. That is, after performing an opening operation, we find that it will remove narrow bridges, and while it attempts to maintain the region size roughly, that shape and its size will still be affected and be modified slightly. This is similar to the median smoothing operation, as each pixel value is replaced with the median of its local neighbors this results which damages thin lines, sharp corners as well as modifies the region's shape to some extent. That is not the case with gaussian smoothing, which will use a gaussian distribution on the kernel used in order to obtain the output pixel. This approach puts more focus on the center pixels which somewhat avoids transforming shapes, the side effect will be that edges will become blurry instead of sharp.

Going slightly deeper into how these techniques work, we find that they all use a sliding window approach which is a common point. However they differ in how their kernel (or sliding window) is used. Gaussian smoothing and median smoothing are quite similar in their use of the kernel, they will both apply a single operation to the pixels within the kernel. That is for gaussian smoothing we compute the weighted average using the kernels weights (which follow a gaussian distribution), for median smoothing the values within the kernel are used in order to retrieve the median value which is used as the output pixel. This is slightly different for the opening operation which will apply to techniques chained together in order to compute the output pixel. The opening technique consists of an erosion operation followed by a dilation operation. This is written down as : $f \circ s = (f \ominus s) \oplus s$; we can notice the two operations which are set operations, more specifically Minkowski set addition and set subtraction.