EYE BALL DETECTION IN A REAL TIME IMAGE USING MATLAB

Parul Goyal[1] and Saxena .D.K[2]
Dehradun Institute of Technology, Dehradun, India
E-mail : [1]parulgoya2007@yahoo.com

Abstract

This paper addresses the problem of detection and tracking an eye ball using image processing algorithms. After analyzing all the collected data by means of geometric, mathematic and physical procedures; the ball will be detected. The detection and tracking of ball in a video will be our objective. This characteristic has potential applications for automatic editing, broadcasting, archiving, browsing and training. The suggested algorithm is based on finding the biggest object present in each frame. The results are promising when the moving objects are few and of different sizes. In other words, when there are a number of objects of the same size, present in the video, this algorithm fails to produce accurate results .To cope with the limitations ,the video chosen has very few moving objects.

Key words: Image Processing, Eye Ball, Tracking

I. DETECTION AND TRACKING

1. Detection Methods

1.1. Convolution and Correlation

Convolution

Linear filtering of an image is accomplished through an operation called convolution. In convolution, the value of an output pixel is computed as a weighted sum of neighboring pixels. The matrix of weights is called the convolution kernel, also known as the filter.

For example, suppose the image is:

$$A = \begin{vmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 20 & 22 \\ 10 & 12 & 19 & 21 & 3 \\ 11 & 18 & 25 & 2 & 9 \end{vmatrix}$$

The convolution kernel is:

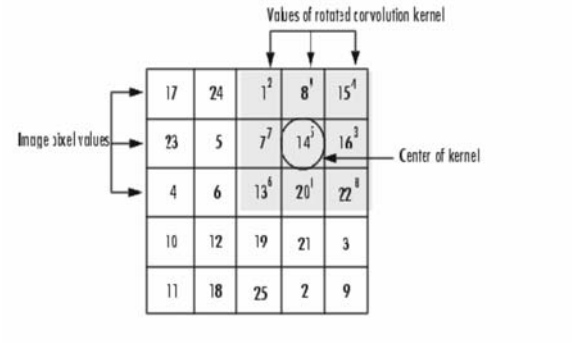
$$h = \begin{vmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{vmatrix}$$

The following matrix box shows how to compute the (2,4) output pixel using these steps:

1. Rotate the convolution kernel 180 degrees about its center element.
2. Slide the center element of the convolution kernel so that it lies on top of the (2, 4) element of A.
3. Multiply each weight in the rotated convolution kernel by the pixel of A underneath.
4. Sum the individual products from step 3.

Hence the (2, 4) output pixel is:

$$1 \cdot 2 + 8 \cdot 9 + 15 \cdot 4 + 7 \cdot 7 + 14 \cdot 5 + 16 \cdot 3 + 13 \cdot 6 + 20 \cdot 1 + 22 \cdot 8 = 575$$



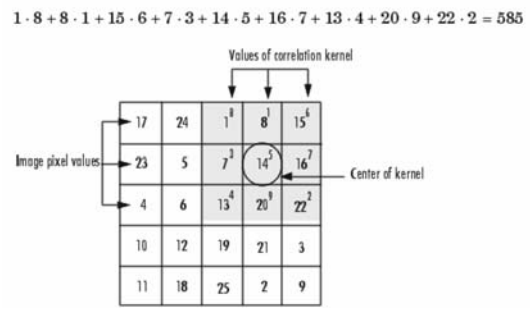
Correlation

The operation called correlation is closely related to convolution. In correlation, the value of an output pixel is also computed as a weighted sum of neighboring pixels. The difference is that the matrix of weights, in this case called the correlation kernel, is not rotated during the computation.

The following figure shows how to compute the (2, 4) output pixel of the correlation of A, assuming h is a correlation kernel instead of a convolution kernel, using these steps:

1. Slide the center element of the correlation kernel so that lies on top of the (2, 4) element of A.
2. Multiply each weight in the correlation kernel by the pixel of A underneath.
3. Sum the individual products from step 3.

The (2, 4) output pixel from the correlation is:



1.2. Shape Recognition

The aim of shape recognition is to make the computer recognize particular shapes or patterns in a big source bitmap. Like the audio signal with speech recognition, this is far from being a trivial issue. Lots of powerful but also very complex methods exist to recognize shapes in an image. The one we are going to study here is very basic and makes a number of assumptions to work correctly. The shape we are going to try to situate on a source image must not be rotated nor distorted on this actual image. This limits a lot the use of this algorithm because in real situations the pattern you search in an image is always a bit rotated or distorted. In one dimensional signal correlation between two signals quantifies how similar they are; in two dimensions the principle is the same. Computing the correlation of the source image and the pattern we are looking for while give us an output image with a peak of white corresponding to the position of the pattern in the image. In fact we make our source image pass through a filter which kernel is the pattern we are looking for. When computing the output image pixel by pixel at one point the kernel and the position of the searched pattern in the image will correspond perfectly, this will give us a peak of white we will be able to situate.

Therefore the implementation of the algorithm is very simple; we compute the matrix convolution of the source image with the kernel, constituted of the pattern we want to search. In this case you will almost be forced to use the FFT convolution, indeed the searched patterns are often more then 30 pixels large, and for kernels of over 30x30 it's faster to use an FFT rather than straight convolution. I tried this algorithm without the FFT optimization and it is very slow: about one transform per minute!

However the results themselves are quite remarkable, the position of the pattern is well recognized, sometimes with a bit of noise though. A nice way to enhance the results and make the peaks even more precise is to make the kernel pass through an edge enhancement filter; this will make the kernel more selective and precise. Here are the results of the algorithm:

Fig 1.1. Source Image.

Fig 1.2. The Pattern The Algorithm Will Be Looking For.

Fig 1.3. The White Peak Corresponding to the Ball's Position.

Fig 1.4. Source Image.

Fig 1.5. The Pattern the Algorithm Will be Looking For.

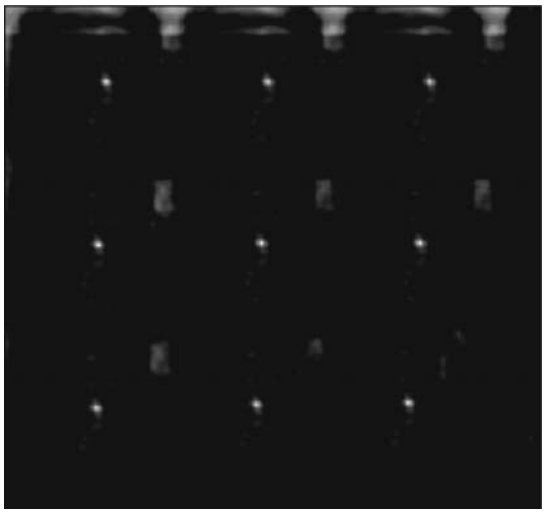


Fig 1.6. The White Peak Corresponds to
The Positions of The Patterns.

2. Miscellaneous Mathematical Techniques

2.1 Gaussian Filter

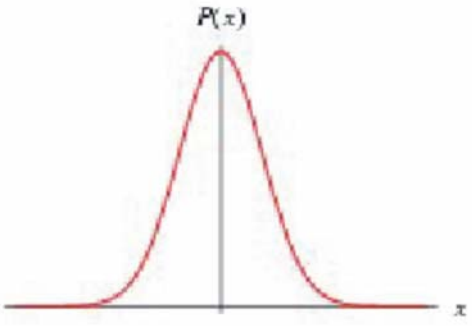


Fig 1.7. A Typical Gaussian Distribution

A Gaussian distribution, N (μ,σ^2)in a variant X with mean μ and variance σ^2 has probability function defined by,

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/(2\sigma^2)}, \quad x \in (-\infty, \infty)$$

The continuous distribution graph is symmetric and has a similar shape to a bell-shaped curve with a MGF (Moment Generating Function)

$$M(\theta) = \exp\left(\mu\theta + \frac{1}{2}\sigma^2\theta^2\right)$$

Normal distributions have many convenient properties. Random variants with unknown distributions are often assumed to be normal, especially in physics and astronomy. This distribution has been found to regularly occur in real-world. Sets of data, and it is often the limit to which the sum of a large number of random Variables. Many common attributes such as test scores and height follow roughly normal distributions, with few members at the high and low ends and many in the middle. Although this can be a dangerous assumption, it is often a good approximation due to a surprising result known as the central limit theorem, which states that the mean of any set of variants with any distribution having a finite mean and variance tends to the normal distribution.

2.2. Method of Least Squares

The method of least squares is a statistical approach that is based on maximum likelihood principle for estimating the parameters of a model from a set of data. The method is computationally simple and often leads to closed form (i.e.: no iterative) solutions.

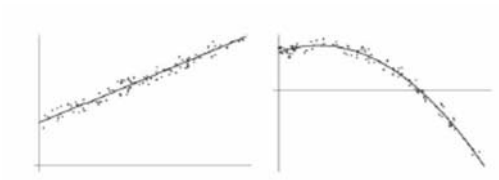


Fig 1.8. Examples the Least Square Fitting of Straight And Polynomial Lines to Noise Effected Data.

Estimations of the optimal parameters are achieved by minimizing a least squares sum of residuals between the observed data and the model prediction in the following way. If a set of n parameters are defined in a vector x, as $x = (x_1, x_2, ..., x_n)$, then given a system it is stated that a relationship between this parameters vector and measurements z of the system is defined by,

$$A_x - z = 0$$

However, given noisy measurements, it is unlikely that a solution to (2.10) will exist. Therefore, for any estimate of the parameters there will be some degree of error.

$$A_x - z = E$$

As previously stated, the idea of least squares attempts to minimized the squared error over n measurements, and so system representation to be minimized becomes,

$$f(x) = (Ax - z)^T (Ax - z)$$

and minimizations is achieved by differentiating (2.12) with respect to x and finding the stationary position, which produces the following least square definition,

$$x = (A^T A)^{-1} A^T z$$

This functional form is justified by probability theory on the basis that each piece of observed data has independent errors drawn from a Gaussian distribution.

However, it is claimed that the method is generally not suitable for data which may be contaminated by other sources of error, such as outliers. Solutions to these types of problems are normally referred to as robust, and include the Hough transform, and can never be solved in closed form.

2.3. B-Spline

B-Spline curves originate from flexible strips used to create smooth curves in traditional drafting applications. Much like Bezier curves they are formed mathematically from piecewise approximations of cubic polynomial functions with zero, first and second order continuity. B-Splines are one type of spline that is frequently used in graphics applications. From N + 1 control points, it is possible to derive a continuous function P(v) such that,

$$P(v) = \sum_{k=0}^{n} P_k N_{k,t}(v), \quad v = 0 \to n - t + 2$$

Where,
Pk =Control points.
N (v) = Blending functions.
t =Degree (normally 3 or 4).
uk= Knots (the break points where they occur on the curve).

The blending functions are defined as,

$$N_{k,1}(v) = \begin{cases} 1, & if\, u(k) \le u(k+1) \\ 0, & otherwise \end{cases}$$

$$N_{k,t}(v) = \frac{v - u(k)}{u(k+t-1) - u(k)} N_{k,t-1}(v) + \frac{u(k+t) - v}{u(k+t) - u(k+1)} N_{k+1,t-1}(v)$$

Such curves have many advantages including:-

1. Changes to a control point only affect the curve in that locality.
2. Any number of points may be added without increasing the degree of the Polynomial.
3. As with Bezier curves, adding multiple points at or near a single position draws the curve towards that position.
4. Closed curves may be created by making the first and last points the same, although continuity will not be maintained automatically.

2.4. Runge-Kutta Numerical Integration

The methods most commonly employed to numerically integrate ordinary differential equations were first developed by the German mathematicians C.D.T., Runge and M.W. Kutta in the latter half of the nineteenth century. The basic reasoning behind so-called Runge-Kutta methods is the employment of a trial step at the midpoint of an interval to cancel out lower-order error terms. The Runge-Kutta approach is to aim for the desirable features of the Taylor series method, but with the replacement of the requirement for the evaluation of higher order derivatives with the requirement to evaluate f(x, y) at some points within the step x_i to x_{i+1}. The method assumes that the correct value of the slope over the step can be written as a linear combination of the function f(x, y) evaluated at certain points in the step.

The general expression for a Runge Kutta method is,

$$x_{n+1} = x_n + T\sum_{r=1}^{R} c_r K_r$$
$$k_1 = f(t_n, x_n)$$
$$k_r = f\left(t_n + Ta_r, x_n + T\sum_{s=1}^{r-1} b_{rs}k_s\right), \quad r = 2, 3, \ldots R$$
$$a_r = \sum_{s=1}^{r-1} b_{rs}$$
$$\sum_{r=1}^{R} c_r = 1$$

Since it is not initially known at this point in the interval these evaluations should be considered, it is possible to choose these points in such a way that the result is consistent with the Taylor series solution to some particular, known as the order of a based Runge-Kutta method.

The second order Runge-Kutta algorithm requires the known derivative function f at the endpoints and midpoint of the interval, and the unknown function y at the previous point. Since we start with initial conditions, the algorithm is self starting. Note to that it is applicable with a general function f (for example nonlinear), and simple to program. The second-order form is defined as,

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right)$$
$$y_{n+1} = y_n + k_2 + O\left(h^3\right)$$

In a trade-off of accuracy against computational effort, the fourth order algorithm is perhaps the most efficient of any method and is given by,

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right)$$
$$k_3 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2\right)$$
$$k_4 = hf(x_n + h, y_n + k_3)$$
$$y_{n+1} = y_n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 O\left(h^5\right)$$

3 . Tracking Methods

3.1. Kalman Filter

The Kalman filter is an efficient recursive filter which estimates the state of a dynamic system from a series of incomplete and noisy measurements, developed by Rudolf Kalman.

Rudolph E. Kalman devised the Kalman filter in 1960 and it has its roots in applied statistical theory. It is a commonly used technique for the removal of measurement errors and estimating system parameters in a wide variety of fields especially economics, physics and of course computer science.

In general, the method proposed by Kalman assumes that at a given point in time 't' the current state x_t of a physical system can be estimated, and that this system state is linearly related to the state at time t + 1 by some matrix A in the following manner,

$$x_{t+1} = A_t x_t + w_t \quad (1)$$

Where the vector w represents the process noise and is specified by a co-variance matrix Q. The confidence in the estimate of each variable in x_t is expressed in terms of a Gaussian distribution, with the variance of each Gaussian held in a co-variance matrix P.

Kalman also stated that must also be a linear relationship between any measurements taken, and the state of the physical system,

$$z_t = H_t x_t + v_t \quad (2)$$

Where z_t represents a vector containing measurements of the physical system to be estimated and vector v_t symbolizes the measurement noise and is defined by a co-variance structure R.

Once (1) & (2) and are established, a recursive approach estimates x_t in an optimal fashion, where optimal is defined as the minimization of the mean squared error of (2) over all the measurements. The algorithm employed consists of two main steps, predict and update.

Predict

At each time step, a prediction is made about the new state vector and state co-variance matrix using the following relationships,

$$\hat{x}_{t+1} = A_t x_t \quad (3)$$

$$\hat{P}_{t+1} = A_t P_t A_t^T + Q_t \quad (4)$$

The prediction of the new state is calculated simply from the linear relationship between the state at t and t + 1 (the noise is ignored as it is assumed to have zero mean). The variance of the state variables is also increased by the process noise, defined by co-variance matrix Q.

Update

Following this prediction, a measurement of the physical system z_t is taken. The state vector is then updated using this measurement by means of the following equations,

$$x_t = \hat{x}_t + K_t (z_t - H_t \hat{x}_t) \quad (5)$$

$$K_t = P_t H_t^T H_t \hat{P}_t H_t^T + R_t^{-1} \quad (6)$$

$$P_t = (I - K_t H_t) \hat{P}_t \quad (7)$$

The update of the state vector at time t provides an evaluation of the level of inaccuracy of the estimation based upon to the measurement taken. This is multiplied by a matrix K, which is a measure of strength of trust in the measurement, and what influence it has on the estimation of the state. Finally, the scaled difference between the predicted measurement and the measurement is added to the predicted state vector x_t. The final stage of the predict-update cycle is to update the variance of the state Gaussians.

3.2. Condensation Algorithm

The condensation algorithm (Conditional Density Propagation) is a computer vision algorithm. The principal application is to detect and track the contour of objects moving in a cluttered environment. Object tracking is one of the more basic and a difficult aspect of computer vision and is generally a prerequisite to object recognition. Being able to identify which pixels in an image make up the contour of an object is a non-trivial problem. Condensation is a probabilistic algorithm that attempts to solve this problem.

The algorithm itself is described in detail by Isard and Blake in a publication in the International Journal of Computer Vision in 1998. One of the most interesting facets of the algorithm is that it does not compute on every pixel of the image. Rather, pixels to process are chosen at random, and only a subset of the pixels ends up being processed. Multiple hypotheses about what is moving where are supported naturally by the probabilistic nature of the approach. The evaluation functions come largely from previous work in the area and include many standard statistical approaches. The original part of this work is the application of particle filter estimation techniques.

4. Algorithms

4.1. Ball Detection Algorithm

- Loading images

```
% loop over all images
for i = 1 : 60
% load image
Im = (imread(['C:/DATA/',int2str(i), '.jpg']));
imshow(Im)
end
```

- Background subtraction

```
% computes the background image
Imzero = zeros(240,320,3);
for i = 1:5
Im{i}= double(imread(['C:/DATA/',int2str(i),'.jpg']));
Imzero = Im{i}+Imzero;
end
```

- Extraction of ball from an image

```
% background subtraction and selection of pixels
with highest difference
fore = zeros(MR,MC);
fore = (abs(Imwork(:,:,1)-Imback(:,:,1)) > 10) ...
| (abs(Imwork(:,:,2) - Imback(:,:,2)) > 10) ...
| (abs(Imwork(:,:,3) - Imback(:,:,3)) > 10);

% removal of small noise
foremm = bwmorph(fore,'erode',2);
```

- Selection of largest object

```
labeled = bwlabel(foremm,4);
stats = regionprops(labeled,['basic']);%basic
mohem nist
[N,W] = size(stats);
  if N < 1
return

end
```

- Computing center of mass

```
centroid = stats(1).Centroid;
radius = sqrt(stats(1).Area/pi);
```

```
cc = centroid(1);
cr = centroid(2);
flag = 1;
return
```

4.2. Ball Tracking Algorithm

- Initializing kalman filter

```
% Kalman filter initialization
R=[[0.2845,0.0045]',[0.0045,0.0455]'];
H=[[1,0]',[0,1]',[0,0]',[0,0]'];
Q=0.01*eye(4);
P = 100*eye(4);
dt=1;
A=[[1,0,0,0]',[0,1,0,0]',[dt,0,1,0]',[0,dt,0,1]'];
g = 6; % pixels^2/time step
Bu = [0,0,0,g]';
kfinit=0;
x=zeros(100,4);
```

- Updating position of object by kalman filter

```
% Kalman update
if kfinit==0
xp = [MC/2,MR/2,0,0]'
else
xp=A*x(i-1,:)' + Bu
end
kfinit=1;
PP = A*P*A' + Q
K = PP*H'*inv(H*PP*H'+R)
x(i,:) = (xp + K*([cc(i),cr(i)]' - H*xp))';
x(I,:)
[cc(i),cr(i)]
P = (eye(4)-K*H)*PP
```

II. RESULTS

Ball Detected


Ball Tracked



III. CONCLUSION

In this paper we have shown that two important Steps of video surveillance, namely object detection and tracking in sports, can be integrated using a kalman filter based algorithm. This approach has the advantage that no thresholds are necessary to re-find previously detected objects. This also leads to better and easier detection of (partial) occlusion.

The core of the object is tracked comparing the size of all the moving objects and finally selecting the biggest object. Then the center of mass of the biggest object is computed.

This work could have been tackled in many different ways. The literature review discusses the past work on similar topics, and hence the techniques that could have been used in this coursework.

Results show that this algorithm is quiet effective when the difference between sizes of moving objects is high. But this algorithm fails when the difference between sizes is very less

This gives scope for future works to build on this software. The software has been designed and implemented for the specific task of detecting cricket balls from a series of images. Other algorithms could be used to improve the functionality of the program.

Also the software could be made more general so it can be used for other research that the Sports Engineering Research Group. The software has certain criteria that need to be fulfilled in order for a region of the picture to be detected. It would be possible to give the user a selection of criteria that could be used for detection of other objects in an image. It may also be possible for the software to detect the cricket ball in a single image without the need of comparison with another image.