Motion detection algorithm based on Background Subtraction

Shivam Shah[1], Vivek Adhikari[2], Vineet Pokhriyal[3]

[1]Research Scholar, Uttaranchal Institute of Technology, UTU
[2]Research Scholar, Uttaranchal Institute of Technology, UTU
[3]Research Scholar, Uttaranchal Institute of Technology, UTU

Abstract: This paper presents an algorithm to detect moving objects within a scene captured by a stationary camera. This algorithm is based on background subtraction, where we build a background model of the scene and compare each frame of the scene with the background model to estimate the amount of motion i.e. difference between the background model and the frame. At each step we update our background model by moving it closer to the current frame. This allows us to effectively estimate the regions of motion in the captured scene. This approach has huge advantage in terms of effectiveness over previously used frame subtraction method. It runs with minimized cost in memory and relatively less computational complexity.

Keywords: Background model, Background Subtraction, Background Updation, Computer Vision, Motion detection, Moving Object Detection, Motion Detection Algorithm

1 INTRODUCTION

An important stream of research within Computer Vision that has gained a lot of importance is Motion Detection. Motion detection[1] is the process of detecting a change in position of an object relative to its surroundings or the change in the surroundings relative to an object. The growing interest in detecting changes within a video, understanding and interpreting human gestures in a video is strongly motivated by recent improvements in Computer Vision and the availability of low-cost hardware such as video camera.

Detecting moving objects in an image sequence is a basic, and fundamental task for many computer vision applications such as video surveillance, traffic monitoring, human gesture recognition. At present method used in moving object detection are mainly the frame subtraction method (brief working of which is presented in a later section), background subtraction method (later discussed in detail), background estimation[2] method and the optical flow method.

Any motion detection system that is based on the above mentioned techniques needs to handle a number of critical situations. These situations induce motion in irrelevant areas inside the scene. Proper handling and management of these situations is required if we want to accomplish a near exact motion detection system. These critical situations arise in many different forms such as:

1. Noise in the image, due to poor quality image\video source.
2. Small movements of non-static objects such as tree branches and bushes blowing in the wind.
3. Variations in lighting conditions in different parts of the same object.
4. Gradual and sudden changes in the light conditions.
5. Objects moving so fast that they are captured in only a single frame of the whole scene.

Some of these situations require pre-processing of the frame, such as reduction of noise and lighting adjustment. However it adds up to the computation that needs to be done to detect moving objects.

The system designed for detecting motion must also keep working without any human interference for a long amount of time. In order to achieve it, the system should adapt to gradual and sudden illumination or light changes and also new objects settling in the scene e.g. A new car (new object) being parked in a parking area (scene). This means that the background should be temporally adaptive.

Additionally, the system should also discard irrelevant motion and slight oscillations of the camera e.g. the camera mount shakes due to turbulence caused by the wind. This means that there must be a local estimation for the confidence in the background value.

A video[3] \scene is made up of a sequence of images called frames. These frames capture the static view that is being observed by the camera. Viewing these frames at high speed gives the illusion of motion. Frame rate, the number of frames per unit of time of video, ranges from 6 frames per second (frame/s) to 120 or more frame/s. The minimum frame rate to achieve a comfortable illusion of a moving image is about 16 frames per second. The device used here for motion detection take 30 frames/s.

Since a video consists of a sequence of images, an in-depth knowledge of image is required to successfully manipulate it. An image[4] is a pictorial representation of a physical object. An image, or more precisely a Raster Image, have a finite set of digital values, called picture elements or pixels. It contains a fixed number of rows and columns of pixels. Pixels are the smallest individual element in an image, holding quantized values that represent the brightness of a given color at any specific point. The size of a pixel, that represents the amount of color that can be displayed, may vary from 8 bits to 64 bits. Therefore, a 24 bits per pixel (bpp) image consists a pixel size of 24 bits. Presently, 32 bpp images are used widely where a pixel is of 32 bits (8 bit each for alpha, red, green and blue). The device being used generates 24 bpp (8 bit each for red, green and blue) images. There is no alpha channel since it presents some problems in manipulation which is discussed later.

2 Frame Subtraction Approach

As mentioned earlier, the Frame Subtraction method is the most used easy approach for motion detection. In this method, the presence of moving objects is determined by comparing 2 successive frames. The previous frame is compared and then subtracted with the current frame. This allows us to obtain only those areas in the scene where motion is detected. The calculation is simple and it has a wide adaptability.

2.1 Algorithm

We have 2 images:-

currentFrame – A grayscale image of the current frame of the scene,
previousFrame – A grayscale image of the previous frame of the scene, and
threshold – The threshold that determine whether the movement is motion or not.

1. *Calculate the Difference between the currentFrame and previousFrame*
2. *Using the threshold value as a Threshold for the image calculated in (1), we calculate the areas which have changed in the currentFrame from the previousFrame.*
3. *Resulting image from (2) is then highlighted in the currentFrame to indicate areas of motion.*

The above algorithm forms a basis of background subtraction method. We modify the above algorithm for space and time to achieve a more complex but efficient motion detection algorithm.

3 Background Subtraction Approach

This approach builds up on the foundation set by the frame subtraction approach. The principle of this method is to build a model of the static scene (i.e. without moving objects) called background, and then compare every frame of the sequence to this background in order to discriminate the regions of motion, called foreground (the moving objects). A procedural view of how this method works is shown in Fig. 1

This approach requires image manipulation to differentiate the foreground from the background. In general, the following manipulations are required. Assuming we have 2 images X and Y, we are manipulating these images to obtain image Z.

3.1 Difference

The difference[1] of two images of the same size and pixel format, produces an image, where each pixel equals to absolute difference between corresponding pixels from provided images.

For each pixel (x) in Image Z:

$$red = |\ X.getPixel(x).R - Y.getPixel(x).R\ |$$
$$green = |\ X.getPixel(x).G - Y.getPixel(x).G\ |$$
$$blue = |\ X.getPixel(x).B - Y.getPixel(x).B\ |$$

$$Z.setPixel(x) = Color(red, green, blue)$$

The reason why 32bpp images are not used is because if images with alpha channel are used, visualization of the result image may seem a bit unexpected – perhaps nothing will be seen (in the case if image is displayed according to its alpha channel)., the reason being the fact that after differencing the entire alpha channel will be zeroed (zero difference between alpha channels), what means that the resulting image will be 100% transparent.

3.2 Threshold

It does image binarization using specified threshold value. All pixels with intensities equal or higher than threshold value are converted to white pixels. All other pixels with intensities below threshold value are converted to black pixels.

For each pixel (x) in Image Z:
If X.getPixel(x).Intensity > threshold
Z.setPixel(x) = White
Else
Z.setPixel(x) = Black

3.3 Algorithm

We have the following images:-

backgroundFrame – A grayscale image of the first image of the scene\video.
currentFrame – A grayscale image of the current frame of the scene.
threshold – The threshold that determine whether the movement is motion or not.

1. *We Calculate the Difference between the backgroundFrame and the currentFrame. The resulting image is shown in Fig. 2.a*

   For each pixel (x)
   I(x) -> | backgroundFrame(x) – currentFrame(x) |

The image that we obtain is the one where all the pixels having same values (i.e. pixels that don't change) are zeroed out, and all the pixels that change (i.e. regions of motion) are highlighted. The work doesn't ends here, the resulting image will contain both relevant and irrelevant areas of motion. Now we have to filter those out.

2. *Using the threshold value as a Threshold for the image calculated in (1), we filter the areas of motion to obtain Fig. 2.b*
   For each pixel(x)
   If I(x) > threshold
   I(x) -> White
   Else
   I(x) -> Black

By using an appropriate threshold value, we can filter out and neglect irrelevant areas i.e. movement of tree leaves in wind etc.

3. *Resulting image from (2) is then highlighted in the currentFrame to indicate areas of motion as shown in Fig. 2.c*

4. *The last step is updating the background. This is done by moving the background to the current frame by a specified amount. If we replace our background with the currentFrame, this method becomes frame subtraction.*

Updating the background is usually achieved by morphing the background slightly toward the currentFrame. The easiest form of morphing can be achieved by combining the two images by taking specified percent of pixels' intensities from the first image and the rest from second image. The value background percent value is set to 0.75.

For each pixel(x) in Image Z
Z.setPixel(x) =
*0.75 * background.getPixel(x) +*
*(1 - .75) * currentFrame.getPixel(x)*

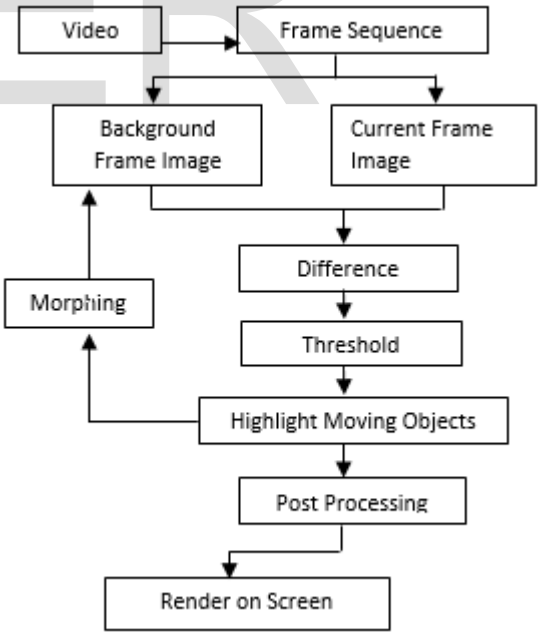

FIG 1. BACKGROUND SUBTRACTION PROCESS

FIG 2.A DIFFERENCE BETWEEN BACKGROUND AND THE CURRENT FRAME

FIG 2.B RESULTING IMAGE AFTER APPLYING THRESHOLD FOR MOTION

FIG 2.C AREAS OF MOTION BEING HIGHLIGHTED BY RED EDGES

4 CONCLUSIONS

The above algorithm was able to find near exact places of motion. It worked effectively on many environments both indoors and outdoors. The threshold value played an important part in categorizing relevant and irrelevant motion. Keeping the threshold value high allowed us to neglect various irrelevant moving objects such as leaves in the tree. However, if we kept it value too high, the algorithm was not able to detect the blink of an eye also. If we kept the threshold value low, we were able to detect very slowly moving objects such as the movement of a tie. Thus finding the critical threshold value satisfying the requirement is also necessary.

The background initialization process used in this algorithm is pretty straight forward and easy. This required almost no initial computation (a little bit of noise reduction, gamma and color correction was required). Most background subtraction methods differ on how they initialize the background model. Some methods are too complex and require factoring the image into objects using cascade classifiers and the determining which of these objects are moveable and which are not. By identifying these objects, the computation required in subsequent phases may speed up but it requires the algorithm to predict where the object can move to in the next frame. As the scene proceeds, the prediction get narrowed down and hence the foreground object and its motion is detected.

Another approach is using background estimation to generate a near exact background and then find areas of motion.

There are various places where this algorithm can be optimized for computation. One instance of optimisation is the fact that the pixels are repeated inside the image, e.g. if we encounter a blue pixel, it can be assumed that its' neighbours will also be blue (to some extent). Hence we can replace the value of the neighbouring pixels with the one we just manipulated. This will reduce each pixel computation and speed up processing. The source code of the project is available online[5] for review and further optimizations.

5 REFERENCES

1. From Wikipedia, http://en.wikipedia.org/wiki/Motion_detection
2. A robust and computationally efficient motion detection algorithm based on ∑-Δ background estimation by A. Manzanera and J. C. Richefeu. http://www.ensta-paristech.fr/~manzaner/Publis/icvgip04.pdf
3. From Wikipedia, https://en.wikipedia.org/wiki/Video
4. From Wikipedia, http://en.wikipedia.org/wiki/Digital_image
5. Github project, Motionizer, https://github.com/raze1392/Motionizer