Extracting the foreground objects in a video

When a fixed camera observes a scene, the background remains mostly unchanged. In this case, the interesting elements are the moving objects that evolve inside this scene. In order to extract these foreground objects, we need to build a model of the background, and then compare this model with a current frame in order to detect any foreground objects. This is what we will do in this recipe. Foreground extraction is a fundamental step in intelligent surveillance applications.

If we had an image of the background of the scene (that is, a frame that contains no foreground objects) at our disposal, then it would be easy to extract the foreground of a current frame through a simple image difference:

```
// compute difference between current image and background
cv::absdiff(backgroundImage,currentImage,foreground);
```

Each pixel for which this difference is high enough would then be declared as a foreground pixel. However, most of the time, this background image is not readily available. Indeed, it could be difficult to guarantee that no foreground objects are present in a given image, and in busy scenes, such situations might rarely occur. Moreover, the background scene often evolves over time because, for instance, the lighting condition changes (for example, from sunrise to sunset) or because new objects can be added or removed from the background.

Therefore, it is necessary to dynamically build a model of the background scene. This can be done by observing the scene for a period of time. If we assume that most often, the background is visible at each pixel location, then it could be a good strategy to simply compute the average of all of the observations. However, this is not feasible for a number of reasons. First, this would require a large number of images to be stored before computing the background. Second, while we are accumulating images to compute our average image, no foreground extraction will be done. This solution also raises the problem of when and how many images should be accumulated to compute an acceptable background model. In addition, the images where a given pixel is observing a foreground object would have an impact on the computation of the average background.

A better strategy is to dynamically build the background model by regularly updating it. This can be accomplished by computing what is called a **running average** (also called **moving average**). This is a way to compute the average value of a temporal signal that takes into account the latest received values. If pt is the pixel value at a given time t and μ_{t-1} is the current average value, then this average is updated using the following formula:

$$\mu_t = (1-\alpha)\mu_{t-1} + \alpha p_t$$

The α parameter is called the **learning rate**, and it defines the influence of the current value over the currently estimated average. The larger this value is, the faster the running average will adapt to changes in the observed values. To build a background model, one just has to compute a running average for every pixel of the incoming frames. The decision to declare a foreground pixel is then simply based on the difference between the current image and the background model.

How to do it...

Let's build a class that will learn about a background model using moving averages and that will extract foreground objects by subtraction. The required attributes are the following:

```
class BGFGSegmentor : public FrameProcessor {

  cv::Mat gray;         // current gray-level image
  cv::Mat background;   // accumulated background
```

```
cv::Mat backImage;    // current background image
cv::Mat foreground;   // foreground image
// learning rate in background accumulation
double learningRate;
int threshold;        // threshold for foreground extraction
```

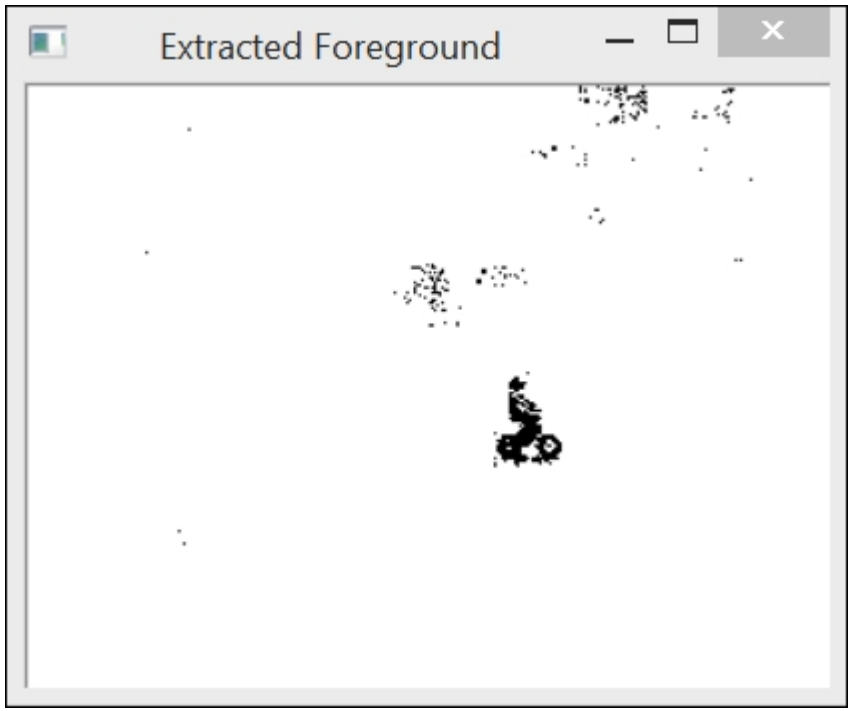
The main process consists of comparing the current frame with the background model and then updating this model:

```
// processing method
void process(cv:: Mat &frame, cv:: Mat &output) {

  // convert to gray-level image
  cv::cvtColor(frame, gray, CV_BGR2GRAY);

  // initialize background to 1st frame
  if (background.empty())
    gray.convertTo(background, CV_32F);

  // convert background to 8U
  background.convertTo(backImage,CV_8U);

  // compute difference between image and background
  cv::absdiff(backImage,gray,foreground);
  // apply threshold to foreground image
  cv::threshold(foreground,output,threshold,255,cv::THRESH_BINARY_INV);

  // accumulate background
  cv::accumulateWeighted(gray, background,
    // alpha*gray + (1-alpha)*background
    learningRate,  // alpha
    output);       // mask

  }
```

Using our video-processing framework, the foreground extraction program will be built as follows:

```
int main()
{
  // Create video procesor instance
  VideoProcessor processor;

  // Create background/foreground segmentor
  BGFGSegmentor segmentor;
  segmentor.setThreshold(25);

   // Open video file
   processor.setInput("bike.avi");

  // set frame processor
  processor.setFrameProcessor(&segmentor);

  // Declare a window to display the video
  processor.displayOutput("Extracted Foreground");

  // Play the video at the original frame rate
  processor.setDelay(1000./processor.getFrameRate());

  // Start the process
  processor.run();
}
```

One of the resulting binary foreground images that will be displayed is as follows:



How it works...

Computing the running average of an image is easily accomplished through the `cv::accumulateWeighted` function that applies the running average formula to each pixel of the image. Note that the resulting image must be a floating point image. This is why we had to convert the background model into a background image before comparing it with the current frame. A simple thresholded absolute difference (computed by `cv::absdiff` followed by `cv::threshold`) extracts the foreground image. Note that we then used the foreground image as a mask to cv `::accumulateWeighte` d in order to avoid the updating of pixels declared as foreground. This works because our foreground image is defined as being false (that is, `0`) at foreground pixels (which also explains why the foreground objects are displayed as black pixels in the resulting image).

Finally, it should be noted that, for simplicity, the background model that is built by our program is based on the gray-level version of the extracted frames. Maintaining a color background would require the computation of a running average in some color space. However, the main difficulty in the presented approach is to determine the appropriate value for the threshold that would give good results for a given video.

There's more...

The preceding simple method to extract foreground objects in a scene works well for simple scenes that show a relatively stable background. However, in many situations, the background scene might fluctuate in certain areas between different values, thus causing frequent false foreground detections. These might be due to, for example, a moving background object (for example, tree leaves) or a glaring effect (for example, on the surface of water). Casted shadows also pose a problem since they are often detected as part of a moving object. In order to cope with these problems, more sophisticated background modeling methods have been introduced.

The Mixture of Gaussian method

One of these algorithms is the **Mixture of Gaussian** method. It proceeds in a way that is similar to the method presented in this recipe but adds a number of improvements.

First, the method maintains more than one model per pixel (that is, more than one running average). This way, if a background pixel fluctuates between, let's say, two values, two running averages are then stored. A new pixel value will be declared as the foreground only if it does not belong to any of the most frequently observed models. The number of models used is a parameter of the method and a typical value is `5` .

Second, not only is the running average maintained for each model, but also for the running variance. This is computed as follows:

$$\sigma_t^2 = (1-\alpha)\sigma_{t-1}^2 + \alpha(p_t - \mu_t)^2$$

These computed averages and variances are used to build a Gaussian model from which the probability of a given pixel value to belong to the background can be estimated. This makes it easier to determine an appropriate threshold since it is now expressed as a probability rather than an absolute difference. Consequently, in areas where the background values have larger fluctuations, a greater difference will be required to declare a foreground object.

Finally, when a given Gaussian model is not hit sufficiently often, it is excluded as being part of the background model. Reciprocally, when a pixel value is found to be outside the currently maintained background models (that is, it is a foreground pixel), a new Gaussian model is created. If in the future this new model becomes a hit, then it becomes associated with the background.

This more sophisticated algorithm is obviously more complex to implement than our simple background/foreground segmentor. Fortunately, an OpenCV implementation exists, called `cv::BackgroundSubtractorMOG` , and is defined as a subclass of the more general `cv::BackgroundSubtractor` class. When used with its default parameter, this class is very easy to use:

```
int main()
{
  // Open the video file
  cv::VideoCapture capture("bike.avi");
  // check if video successfully opened
  if (!capture.isOpened())
    return 0;
  // current video frame
  cv::Mat frame;
  // foreground binary image
  cv::Mat foreground;
  cv::namedWindow("Extracted Foreground");
  // The Mixture of Gaussian object
  // used with all default parameters
  cv::BackgroundSubtractorMOG mog;
  bool stop(false);
  // for all frames in video
```

```
  while (!stop) {
    // read next frame if any
    if (!capture.read(frame))
      break;
    // update the background
    // and return the foreground
    mog(frame,foreground,0.01)
    // learning rate
    // Complement the image
    cv::threshold(foreground,foreground,128,255,cv::THRESH_BINARY_INV);
    // show foreground
    cv::imshow("Extracted Foreground",foreground);

    // introduce a delay
    // or press key to stop
    if (cv::waitKey(10)>=0)
      stop= true;
  }
}
```

As it can be seen, it is just a matter of creating the class instance and calling the method that simultaneously updates the background and returns the foreground image (the extra parameter being the learning rate). Also note that the background model is computed in color here. The method implemented in OpenCV also includes a mechanism to reject shadows by checking whether the observed pixel variation is simply caused by a local change in brightness (if so, then it is probably due to a shadow) or whether it also includes some change in chromaticity.

A second implementation is also available and is simply called `cv::BackgroundSubtractorMOG2`. One of the improvements is that the number of appropriate Gaussian models per pixel to be used is now determined dynamically. You can use this in place of the previous one in the preceding example. You should run these different methods on a number of videos in order to appreciate their respective performances. In general, you will observe that `cv::BackgroundSubtractorMOG2` is much faster.

See also

- The article by C. Stauffer and W.E.L. Grimson, *Adaptive Background Mixture Models for Real-Time Tracking*, in *Conf. on Computer Vision and Pattern Recognition, 1999*, gives you a more complete description of the Mixture of Gaussian algorithm.