Workshops in Creative Computing
Computer Vision
Lab 3: Blob Detection and Tracking

Parag K Mital

February 6, 2012

Introduction

In the lecture we explored a method of locating foreground objects called Blob Detection that assumes objects are distinct from a learned background model. In the first week's lab, we used frame-differencing to find the motion between each frame. Instead of finding the difference between each frame, we could store one image as a background, and compute the difference between this stored frame and every incoming frame (background subtraction) in order to find blobs which are separate from the stored background. However, this approach does not account for changes in the scene from the camera moving or in lighting and as well we have to also specify a good threshold which may also change over time.

In part 1 of this lab, we will start with the basics by implementing our own background subtraction algorithm using the code from week 1. In part 2, we will explore a much more powerful way of detecting and tracking blobs using a set of tools that are already built for you: a background model using GMMs, Gaussian Mixture Models (credit: zivkovic); a simple distance based tracker (credit: stefanix); and one library I have built which combines the functionalities of both of these libraries, and as well computes the orientation of the tracked blobs by inferring the tangential motion using optical flow and vector math. We will learn how to use this library directly for our own purposes, and also work with OpenSoundControl for sending these messages to another program such as Max, Processing, etc..

Part 1

Modify your code from the first week's lab to compute background subtraction using a single stored image. Recall that we created a variable "`previousImage`" and updated this variable on every iteration of `update()`. Instead of updating this variable in `update()`, have the user press a key such as the spacebar, and update this variable only when the spacebar is pressed. Try this with a complicated background. Try it again with the camera pointing to the ceiling.

To create a nicer image for blob detection, you can play with morphology and smoothing using the following OpenCV functions: `cv::erode, cv::dilate, cv::filter2D, cv::GaussianBlur, cv::morphologyEx`. Following these operations you can also play with thresholding using: `cv::threshold` and `cv::adaptiveThreshold`. Look these functions up in the OpenCV reference in order to get familiar with how they are used. Or try to infer from the syntax of the function declaration how they might be called. Try performing some of these operations on your difference image in different orders and see how they affect the output image. Remember, these functions require the OpenCV matrix in `Mat` form.

When you have a clean image which encompasses the blobs you'd like to track, you can describe them using blob detection. openFrameworks includes a library for doing this fairly easily called ofxCvContourFinder. Have a look at the opencvExample included with openFrameworks and try to understand the code. See if you can include your operations of morphology, blurring, and thresholding to get a better blob image.

Part 2

We'll now use the libraries mentioned in the Introduction for performing blob detection and tracking using a Gaussian Mixture Model for each pixel and send the tracking data using OpenSoundControl (OSC). OSC is a very convenient interface for sending and receiving messages. openFrameworks includes an addon for OSC and examples in the addonsExamples folder of how to send (oscSenderExample) and receive (oscReceiveExample) OSC messages. Since we will use the OpenCV addon as well as the ofxOsc* addon, we will need our project to know about both of these libraries. Up until now, we would copy the opencvExample folder, though now we can copy the allAddonsExample folder, which will include the libraries for all of the openFrameworks addons including both OpenCV and OSC.

Starting with the allAddonsExample, make sure you include the additional files listed on the course website. Extract all the files into your "src" directory, and drag them into your project so they are compiled and linked to your program. You'll want to remove all the erroneous code from your testApp.h and testApp.cpp files, and make sure you include "pkmBlobTracker.h". This file will include all the other files for you. You'll also want to **inherit** from "ofCvBlobListener" by modifying your class definition like so:

```
1  class testApp : public ofBaseApp, public ofCvBlobListener {
2
3    ...
4
5  };
```

This allows us to use our "testApp" as a **listener** for 3 methods which will get called whenever a blob is detected, moved, and undetected. Think of these methods in the same way as you think about the keyboard and mouse callbacks. They are called whenever an action is performed. In this case, the action refers to the events of a blob being detected, moved, and being un-detected. First we must declare these functions in our class like so:

```
1  class testApp : public ofBaseApp, public ofCvBlobListener {
2
3    ...
4
5    void blobOn( int x, int y, int id, int order );
6    void blobMoved( int x, int y, int id, int order );
7    void blobOff( int x, int y, int id, int order );
8
9    ...
10
11 };
```

And we also have to define them in our .cpp file, which for now can be empty definitions:

```
1  void blobOn( int x, int y, int id, int order )
2  \{
3
4  \}
5  void blobMoved( int x, int y, int id, int order )
6  \{
7
8  \}
9  void blobOff( int x, int y, int id, int order )
10 \{
11
12 \}
```

It is up to our application to use these functions in whatever way we wish. For instance, we could assign each blob to an `ofSoundPlayer`, so that whenever a blob is detected, a sound starts to play. When they move, the sound could play back with the speed of the blob, or be spatialized according to where the blob is situated. These are some techniques I have used in my own installations, "People Making Noise" and "Future Echoes".

Each time these functions are called, we receive the x and y centroids of the blobs in image pixel coordinates, an id which is a unique number identifying each blob, and the order, which is an integer

sorting the blobs by their area. If we wanted to calculate speed, we could keep a hash table of all blob id's in order to find the speed (i.e. delta position = speed). If you are interested in doing this, a `map` is a very good way of creating a hash table.

In order to use the `pkmBlobTracker` class, we must first create a variable in our class definition, e.g. `tracker`. Then we allocate the object by calling the method `tracker.allocate(int width, int height)` in our `setup()` using the width and height of the image to find blobs on (e.g. camera resolution). We also declare that we will want testApp to handle callbacks for the blobs by writing `tracker.setListener(this);` [1]. We give the image to find blobs on in our `update()` method by using `void update(unsigned char *pixels, int w, int h)`. As well, we can draw an interesting visualization of the blob algorithm by calling `tracker.draw(0,0)`. To see anything, we'll want our window size to be at least 3 times the width of the input image and another 80 pixels for padding. The pkmBlobTracker also defines some keyboard interactions for changing different parameters of the algorithm. Have a look at the class method and call the `keyPressed` method if you aren't getting good results with your blob detection in order to change some of the parameters.

Try using OpenSoundControl to send the x,y positions of blobs when they move to another program such as MaxMSP. You can view the example oscSenderExample to get an idea of how to send osc message. Make sure you use the allAddonsExample in order to link the libraries of both ofxOSC and OpenCV.

[1] `this` is a pointer to our instance of the object `testApp` which is created in `main()`, e.g. `new testApp()`