Analysis of U.S. Olympic Boxing Videos
Colorado School of Mines
Computer Vision

Justin Brewer

May 5, 2014
Contents

1 Introduction 4

2 Previous Work and General Background 5
  2.1 Computer Vision in Sports .................................................. 5
  2.2 Assumptions and the Environment ........................................ 6
  2.3 Colors and Textures ......................................................... 7

3 Scientific Background and Approach 8
  3.1 Background Subtraction ...................................................... 8
    3.1.1 Shadow Detection ..................................................... 10
    3.1.2 Morphological Operations .......................................... 10
  3.2 Mean Shift ................................................................. 10
    3.2.1 CamShift .............................................................. 11
  3.3 Template Matching ......................................................... 11

4 Proposed Tracking Algorithm 12

5 Implementation 14

6 Results 14
  6.1 Gaussian Mixture Model Background Subtraction Results ............ 14
  6.2 CamShift Results ......................................................... 16

7 Conclusion 17

Appendix:

A Adaptive MoG Background Subtraction Code 19

B CamShift Code 20

C Matlab Tracking and Plotting Code 25

D Matlab Heat Map Generation Code 27

List of Figures

1 Illustration of Hawk-Eye computer vision system tennis ball tracking system
   (Courtesy of Hawk-Eye Innovations). ........................................ 5
2 Illustration of heat map produced by SportVu computer vision system [], 5
3 Illustration of lens distortion imposed on captured frame. ............... 6
4 Illustration of attire worn by boxers in an official boxing match. ....... 7
5 First frame in video (left). Background image (right). .................. 9
6 Result of basic background subtraction from Figure 5 .................... 9
Cartoon of mean shift algorithm moving window to region of highest probability.

Sudo-code block diagram of boxer tracking system.

Good segmentation results from adaptive Gaussian mixture model.

Non-ideal result from adaptive Gaussian mixture model.

Non-ideal result from adaptive gaussian mixture model.

Good results from CamShift Algorithm.

Bad results from CamShift Algorithm.

(X,Y) position plot of boxer.

Heat map produced from (X,Y) data in figure 15.
1 Introduction

Boxing, in particular analysis of Olympic boxing videos is the main focus of this project. This project is a proof of concept project for the United States Olympic Team and many thanks go out to the US Olympic committee for providing video data sets and support.

Computer Vision is the process of using computers to acquire images, transform images and extract symbolic descriptions from images. Many computer vision projects involve identifying, tracking or segmenting humans from videos. Once a human is found and segmented from an image a human’s location or pose can be used to further understand the scene.

Olympic boxing is fought on a 20 by 20 ft. square enclosure called the ring, this is where the boxing takes place. The ring in enclosed by elastic ropes and covered with canvas. Part of the canvas is located outside of the ropes called the apron providing support to keep the boxers from falling out of the ring and a place for coaches to stand at the end of a round. During the fight three people are present on the ring, two boxers and a referee. Each boxer is required by Olympic rules to wear boxing gloves and a head guard in order to reduce the impact from a punch and protect the boxers. Each of these pieces of equipment are generally colored either blue or red.

Analyzing overhead recordings of a boxing match may prove to be a useful for boxers, coaches and television hosts if interesting and useful information can be extracted from the videos and then presented. An overhead view of boxing provides a unique opportunity to track the location of each person in the ring as the match is fought. If this is done the (X,Y) location with time each person will be known. Using simple physics this data could be easily used to compute the velocity of each person with respect to time and then the acceleration with respect to time. Position, velocity and acceleration data may prove to be a quantitative way to see fatigue, reaction time, or the impact of a punch on a boxer. The position with respect to time data can directly be used to create a heat map showing where a person or all the people on the ring have been and how much time they have spent in certain locations. This type of heat map might be a cool visualization to for sports analysts or prove to be a valuable tool for boxers and coaches to see a areas within the ring that certain boxers are most a least comfortable motivating areas for improvement of influencing fighting strategies before a boxing match.

The rest of this document will provide a feasibility discussion and a general approach for analyzing overhead boxing videos with intent to fulfill the following goals:

1. Extract (X,Y) coordinates with respect to time of the three people in the ring (2 boxers and 1 referee). Taking the (X,Y) origin to be the lower left corner of the ropes.
2. Produce (X,Y) plots for each person in the ring aiming for an accuracy of 20 cm.
3. Produce individual heat maps for each person, representing where that person has been and how much time they spent in that location.
4. Produce overlaid heat maps for each person, representing what areas of the ring have been touched and how much time has been spent in the touched locations.
2 Previous Work and General Background

2.1 Computer Vision in Sports

The use of computer vision to aid athletes and televised broadcasts of sports is becoming more prevalent. The most common approaches to using computer vision in sports are to aid the athlete in training, aid the referees and officials in calling a fair game, or to increase viewership of televised broadcasts by aiding broadcasters in illustrating, analyses or explain sporting events. Such applications of computer vision rely on calibrated cameras to capture images or videos which can then be processed by programs and algorithms to extract information from the scene such as tracking ball locations, segmenting players from scenes and gathering 3D information. One example of a system designed to help referees can be seen in Figure 1 uses a system calibrated video cameras to track a tennis ball and extract the balls 3D location in space. The hawk eye system reconstructs an image of the balls flight path ultimately showing if the tennis ball landed in bounds or out of bounds. This system is used with the referee to call a fair game as it helps him interpret split second plays. The generated graphics are also shown on television.

Figure 1: Illustration of Hawk-Eye computer vision system tennis ball tracking system (Courtesy of Hawk-Eye Innovations).

Another system called SportVU is advertised as ”revolutionizing the way sports contests are viewed, understood, played and enjoyed.” This systems says its capable of providing real-time (X,Y,Z) position data and heat maps similar to the proposed heat maps stated in the goals of the introduction for soccer (futbal) and basketball matches. Figure 2 shows an example graphic from this system. The heat map shows the location and frequency of shots throughout a game [5].

Figure 2: Illustration of heat map produced by SportVu computer vision system [ ].
Little information is provided about the SportVu player tracking system on their webpage but the provided information indicates that this system with little effort may be converted to the tracking of boxers as well. Many computer vision systems for sports applications exist but the current capabilities of these systems show that implementing a reliable system for tracking boxers should be feasible.

### 2.2 Assumptions and the Environment

Each computer vision system makes assumptions about the environment, scene, and variables of the camera setup and the images the camera/s are taking. Many variables complicate this process such as lighting, environment, color, image quality, distortions etc. But analyzing videos from a stationary camera removes the need to constantly update the relative poses of the camera with respect to stationary background objects. The proposed algorithm will assume a single overhead stationary, color camera, capable of capturing the entire boxing ring in one frame. Cameras fulfilling this assumption will impose lenses distortions on the captured image as seen in a frame from the video data provided by the U.S. Olympic Committee in Figure 3.

![Figure 3: Illustration of lens distortion imposed on captured frame.](image)

In the video frame seen in Figure 3 the camera has applied barrel distortion to the captured scene. The barrel distortion is seen as the unnatural outward (away from center) curvature applied to objects in the scene, and is more evident toward the edges of the frame. Barrel distortion will add inaccuracies in locating the position of each person in the frame. This can be seen in Figure 3 by looking at the people in the ring. The boxer standing on the USA Boxing logo appears to be standing directly underneath the camera and when segmented from the image tracking the top center point of his head also corresponds to
a "precise" location on the ring. The other boxer and referee are positioned away from center appear to be slightly tilted in the video. The tilted appearance is due to the barrel distortion imposed by the camera lens and when segmented from the frame it is more difficult to "precisely" determine their position in the ring.

Distortion is best corrected for by calibrating the camera used to capture the scene. Camera calibration allows the intrinsic camera parameters, focal length, pixel size and, distortion coefficients of the camera to be found if these parameters are not published with the owner’s manual of the camera. Finding the 9 intrinsic parameters of a camera requires a non-linear optimization such as iterative least squares fit to minimize residual errors [\(_\text{[1]}\)]. This type of calibration works best when preformed with a number of images of a calibration object taken at a variety of view point (at least three). Once the calibration is preformed and the intrinsic camera parameters are known each frame of the video can be rectified to remove the lens distortion. For all intents and purposes lens distortion removal will be skipped in the proposed algorithm due to not posing information about the camera used to shoot the boxing videos that are being analyzed. Skipping this step will not affect our ability to prove or show feasibility.

2.3 Colors and Textures

As noted in the Introduction each boxer is required to wear boxing gloves and a head guard. Olympic regulations state that each boxer must wear either blue or red in an official match as seen in Figure 4.

![Figure 4: Illustration of attire worn by boxers in an official boxing match.](image)

In some cases boxers are also required to wear boxing gloves that have a white patch on the main hitting surface of the glove. Figure 4 illustrates important information that can be drawn from a boxing match held under official standards. The red and blue colors of the boxers could be used to aid in tracking and segmenting each boxer. The white patch on each of the boxer’s gloves could provide a distinct feature to aid in information extraction such as when a boxer is punching. The distinct pattern on the top of each boxers head caused by the head guard as seen in Figure 4 could be used to help determine direction or body rotation data of each boxer through the fight.

Being able to assume various properties of the captured scene such as colors being worn by people or patterns that will always be present can provide the necessary restraints or assumptions needed to establish a robust computer vision system or can extend the capabilities of the system by enabling maximum information extraction. Due the variability of
the colors and patterns in the data being analyzed no assumptions about colors (like blue or red) or assumptions about patterns will be made but colors and patterns within each data set will be used by the proposed algorithm.

3 Scientific Background and Approach

The goal of this algorithm is to segment and track the three people (2 boxers and 1 referee) in the boxing ring throughout the duration of a fight. Image segmentation is the process of finding groups of pixels that belong together. The result of segmentation is often clusters of pixels. A good segmentation of a boxer from a boxing video would provide the locations of the majority of pixels in an image belonging to the boxer or the contour information outlining the boxer. Videos provide useful information such as the data gathered in the previous frame. Colors and textures are also useful properties of objects in a scene that aid in segmentation. In segmenting boxers from videos there are three main techniques that should be analyzed and tested.

- Boxer segmentation and tracking by motion detection. As seen in Improved Adaptive Gaussian Mixture Model for Background Subtraction [7].

- Boxer segmentation and tracking using color space or histogram data as explained in Mean Shift: A Robust Approach Toward Feature Space Analysis [2].

- Boxer recognition using iterative template matching example of which is explained in Template matching Based Object Recognition with Unknown Geometric Parameters [3].

3.1 Background Subtraction

Background subtraction is a useful technique for detecting moving or intruding objects in a stationary background scene. The background or stationary scene can be described by a statistical model to which a moving object can be detected by seeing areas or pixels that don’t fit the model. A basic implementation of background subtraction involves averaging the grayscale frames of a video containing moving objects to obtain a background image. Moving objects then found by doing a frame by frame comparison against the background image. A pixel is determined to be background or foreground if the difference between a pixel in the current frame and the corresponding pixel in the background is greater than a threshold value. Figures 5 & 6 show the results of implementing this simple background subtraction model and will be used to compare with more sophisticated background subtraction techniques such as that proposed in [7]. In Figures 5 & 6 it is seen that each person in the ring appears as a white cluster or blob. It is important to note that objects that enter the scene but stay in the same place for long amounts of time just as the people in the upper left hand corner of this video did start to appear in the background image but are still
Figure 5: First frame in video (left). Background image (right).

Figure 6: Result of basic background subtraction from Figure 5

considered to be foreground. The main concern is that this method of background subtraction segments the shadows imposed by the boxers on the canvas as foreground which would add error when extracting the position data of each boxer.

A Gaussian mixture model is statistical alternative to the simple background subtraction model described above. The proposed benefits of a Gaussian mixture model background subtraction model are to account for controlled but consistent color variations for a pixel in the image. An example of this would be a leaf blowing in a breeze causing that leaf to move into and out of some of the pixels in the image. Even though the leaf is moving in the scene it is not moving object of interest. This type of method also accounts for illumination changes in the scene such as switching a light on or daytime becoming nighttime for the outdoor case. The adaptive Gaussian mixture model proposed in [7] is a nice implementation. This method uses an adaptive background model that is defined by using a training set of images \( \chi \) capped at a certain number of images corresponding to a certain length of video time. Once the training set is full each additional frame of the video will be added to the end of training set and the first frame in the training set will be removed. The capped length of the
training set makes this method adaptive meaning that intruding objects that enter the scene but stay in the same location for enough time will eventually become part of the background scene.

Determining if a pixel belongs to the background or foreground is done using a Bayesian decision comparing the RGB color of a pixel in the current frame with a Gaussian mixture background model. The Gaussian mixture background model is a probability density function corresponding to the probability that the pixel is a certain RGB color. The number of Gaussian components in the Gaussian mixture model is variable and the Gaussian mixture model of a pixel is found using the variance and means of that pixel across all the elements in the training set $\chi$.

3.1.1 Shadow Detection

The result from the basic background algorithm seen in Figure 5 detected the shadows that the people in the ring cast on the canvas. Using a Gaussian mixture model for background subtraction has the benefit of also enabling a shadow detection feature to easily be implemented. The idea of shadow detection as explained in, Detecting Moving Shadows: Formulation, Algorithms and Evaluation is that shadows have similar chromaticity but lower brightness than the background model. Chrominance is a measure of the difference between the expected color of a pixel and the value of the pixel. The expected value of the pixel is taken directly from the Gaussian mixture model used for background subtraction chromaticity is calculated using the mean and variances of each pixel the pixel is grouped into three categories foreground, background or shadowed background. For tracking boxers shadows are best grouped into the background category.

3.1.2 Morphological Operations

The result of background subtraction is a binary or black and white image. Ideally this image would only contain three white blobs or the boxers and the referee in the scene. As seen in Figure 5 noise and other objects outing the ring can appear in the subtracted image as well. Morphological operators are used to shape features of a binary image using different structuring elements. The fundamental morphological operations are erosion and dilation. Erosion is used to generally decrease to size of blobs in a binary image by eroding away the edges. Erosion is useful when trying to de-connect two blobs that are connected at an edge. It can be used to remove small blobs that appear as noise in the image. Dilation is used to expand blobs in a binary image. Dilation can be used to connect two blobs that may be separated. Both dilation and erosion can be used to ”clean up” the binary resultant from background subtraction.

3.2 Mean Shift

Mean shift algorithms are used for a variety of purposes in the academic literature. In computer vision mean shift is most used for segmentation and tracking. Interested in tracking, the mean shift algorithm is a mode finding algorithm. Modes in this sense of the word are a set of data samples manifesting an underlying probability density function in a feature
space such as color. In the color feature space the normalized histogram of an object is the underlying probability density function describing that object. Knowing the expected color histogram of an object the mean shift algorithm will iteratively converge on the mode or region within the image that best matches the probability density function of the object being searched for. This is done through the use of a predetermined window size and shape that is iteratively moved until the region of highest probability converges towards the center point of the widow. A cartoon of this is seen in Figure 7. Comparison of the target histogram with the current frame is done using a method called histogram back projection. The result of histogram back projection is a grayscale image for which the lightness of a pixel directly corresponds to the probability that pixel belongs to the target.

Figure 7: Cartoon of mean shift algorithm moving window to region of highest probability.

### 3.2.1 CamShift

Continuously Adaptive Mean Shift Algorithm (CamShift) is an adaption to the mean shift algorithm. CamShift unlike the mean shift algorithm allows for variability in the widow size and rotation [4]. This adaptively is useful for objects that are changing perspective by moving towards or away from the camera. If the widow is allowed to be elliptical the CamShift algorithm will essentially fit an ellipse to the object being tracked. This is valuable when tracking humans from an overhead perspective because the width of the shoulders will in most cases will result in the major axis of the ellipse and the corresponding minor axis will be parallel with the direction the person is facing.

### 3.3 Template Matching

Another method of tracking objects is through the use of template or image patch feature matching. This method can be applied when the object being tracked has a distinct region or the object as a whole is distinct. Distinct images patches are locally unique and contain distinct patterns and textures. Template matching is mentioned here but not used in the implementation of the boxer tracking system described later. The potentially distinct features of a boxer that pose as a good feature to track is the pattern seen on top of the boxers head as described in the Colors and Textures section. Template matching is potentially a very effective way to track the head of a boxer, when this pattern is present, doing so would require an iterative template matching system that in some way checks for in-plane and out-of-plane rotations that a boxers head frequently experiences during a fight. For this reason this method is mentioned but not used.
Figure 8 shows a sudo-code block diagram of the proposed computer vision algorithm for tracking boxers from an overhead perspective. The proposed algorithm requires minimal user input and ultimately stores the (X,Y) location of each person in the boxing ring.

Figure 8: Sudo-code block diagram of boxer tracking system.
The proposed algorithm uses both an adaptive Gaussian mixture model background subtraction and CamShift to track each person frame by frame. Using an adaptive Gaussian mixture model to perform background subtraction takes advantage of how much boxers move during the duration of a match. The algorithm is initiated by reading the first frame of the boxing video being analyzed. Once the first frame is read the algorithm requires the user to provide the initial pixel locations of each person that is going to be tracked and the pixel locations of the four corners of the ring. During this step the algorithm saves a histogram representing each person in the ring by looking at a region of a specified size at the user input locations. The algorithm next sets up the mixture of Gaussian background model. Parts of the background model could potentially be passed into the program if they have been initialized before. If an initial background model is not passed in the algorithm would initialize a background model by retrieving a number of consecutive frames to fill the requirements of training set needed for the adaptive background model. The algorithm then proceeds to calculate the background subtraction of the first frame in the video. The binary result of the background subtraction is processed using morphological erosion to first remove as many small blobs as possible, which are noise. Next dilation is preformed to slightly expand the regions that are left back to their original size. Connected component labeling is done to create a list of blobs, the area of each blob and the centroid of each blob. Next the largest blob within a threshold distance of pixels away from the previous location of each person is found. This is done because a person is only capable of moving a certain distance between capturing of frames. An improvement to the algorithm would include a prediction of the location in the next frame based on the previous position, velocity and acceleration such as the prediction a Kalman filter would provide.

This point in the code is where the real tracking begins. The code checks to see if the blobs are larger that a specified number of pixels. This is done because upon experimentation, if the blobs are greater than this size background segmentation has captured the majority of the person and the centroid of a blob that meets this criteria is likely to fall within the 20 cm required accuracy. This breaks down when less than three blobs are detected or one or more of the blobs fails to meet the required pixel number size. If this happens the algorithm uses the previous location and histogram data stored in data to create a sub-image of predetermined size cropped from the current frame, for each failed case. Using the stored relevant stored histogram of the person being track CamShift is applied to the relevant sub image. The result of the CamShift algorithm is an ellipse with the center of the ellipse position in the location of best probability to encapsulate the person. This location is checked against the previous position data, and the diameter of the minor and major axis are checked for reliability. If the size or location of the fitted ellipse fails to meet a pre-determined specification the algorithm should pause and ask the user to "click" on the location of each person. Finally, the location and histogram data of each person is updated and the process repeats loading a new frame of the video until the end of the video is reached.

The algorithm will store the frame-by-frame (X,Y) location of each person in memory. The required (X,Y) plots can either be generated frame-by-frame as the algorithm processes the video or can be produced after all the frames have been processed.
5 Implementation

Unfortunately, due to time constrains the algorithm has not been implemented as one program. The current state of the algorithm is split between two programming languages Matlab and c++. The main computer vision techniques used by the algorithm, background subtraction and CamShift have been implemented and tested for functionality but the algorithm as a whole has not been tested in its entirety. In the future the algorithm should be implemented only using c++ for best processing speeds.

In its current state the adaptive Gaussian mixture model background subtraction method has been implemented using functions from c++ computer vision package OpenCV. The main functions used for this portion of the code are BackgroundSubtractorMOG2() and getBackgroundImage(). BackgroundSubtractorMOG2() implements the adaptive mixture of Gaussian background subtractor model explained in the background section above. The maximum number of Gaussian components per pixel that proved to give the best results was three. the function getBackgroundImage() is used to calculate and visualized the background image. Visualization of the background image is useful when initializing/optimizing input parameter values.

All morphological operations are done using Matlab functions. The Matlab function regionprops() performs connected component labeling creating a matlab datastructure called a struct which is similar to a dynamic array. The function regionprops() returns a struct containing all of the "blobs" in the image as well as each blobs area, centroid and bounding box.

The CamShift portion of the algorithm is currently implemented using the c++ computer vision package OpenCV. The heart of this code uses OpenCV’s CamShift class and the function calcBackProject(). calcBackProject() uses the normalized histogram to scan over pixels in the current frame coloring them based on the probability of them belonging to the object being searched for. CamShift() performs the iterative mean shift algorithm described in the background.

6 Results

6.1 Gaussian Mixture Model Background Subtraction Results

Using an adaptive Gaussian mixture model gave very promising results. Figure 9 shows good results from the background subtraction algorithm. The red contours were found from the resulting binary background subtracted image and then overlaid onto the current frame being analyzed. It can be seen that the people in the image are segmented very well. The segmentation captured the outline of the whole person and in both cases very few pixels belonging to the people fall outside the segmentation.

The adaptive Gaussian mixture model background subtraction does breakdown in a few areas. Because the algorithm is probabilistically comparing the colors in the background model with colors of the pixels in the current frame, when a boxer is wearing a similar color clothes as pixels colors that show up in the background model. The target will not be fully segmented from the background when these similar colored pixels are overlapping. An
example of this is seen in Figure 10 the boxer wearing the white colored shirt "blends" into the white words on the boxing ring.

Another breakdown in the adaptive Gaussian mixture model background subtraction occurs in both Figures 10 & 11. In these Figures it can be seen that one boxer is punching the other boxer. The result is that in these cases when the people are closely interacting or touching they look like one intruding object with respect to the background model.

Figure 9: Good segmentation results from adaptive Gaussian mixture model.

Figure 10: Non-ideal result from adaptive Gaussian mixture model.

Figure 11: Non-ideal result from adaptive gaussian mixture model.
6.2 CamShift Results

In order to compensate for the results where the background does not detect a whole person and the locations where the boxers merge together the idea was to use the CamShift algorithm. This idea ties in nicely with the background subtraction method because both algorithms can be used together to boost the overall robustness of the algorithm. The current implementation of the CamShift algorithm computes histogram back projection using the hue component of the objects histogram. Because of this objects that that have distinct hue values in the image are good objects to track. This can be seen by comparing Figures 12 & 13. In Figure 12 the referee who is wearing a red shirt is very distinct compared to the rest of the background. This is seen in the histogram back projection as the referee appears as the largest, "most probable" blob with very little other areas appearing in the back projection. In Figure 13 the CamShift algorithm is trying to track the white boxer. Because the hue value associated with white is not distinct in the background image the result of the histogram back projected is an image full highlighted areas that are not the boxer. This case fails to track the boxer for more than a few seconds.

Figure 12: Good results from CamShift Algorithm.

Figure 13: Bad results from CamShift Algorithm.
7 Conclusion

Overall, using an adaptive Gaussian mixture model for background subtraction gave very promising results. This method proved to be reliable in segmenting the boxers from the videos. Figure 14 shows the (X,Y) plot of a boxer. The boxer was tracked using a sequence of frames where no interaction with the other boxer occurred so there were no merged blobs. The boxer was then tracked using morphological operations and the tracking methods stated in the proposed algorithm. Figure 15 is a heat map of this same sequence of frames. The proposed algorithm uses CamShift to deal with the locations where the boxers interact merging into one blob.

![Figure 14: (X,Y) position plot of boxer](image1)

![Figure 15: Heat map produced from (X,Y) data in figure 15](image2)

The first implementation of the CamShift algorithm proved to be very successful for tracking objects or people that have a very distinct hue values compared to the background image. Using the previous location of the people in the ring to crop out a portion of the image before the CamShift algorithm is preformed may give prove to give better results for these objects with non-distinct hue values. Also, The Camshift algorithm can be done using any properties from the feature space of an image. Different implementations of the CamShift algorithm need to be done to fully prove or disprove if CamShift is a viable solution to aid in the tracking of boxers.
References


A Adaptive MoG Background Subtraction Code

```cpp
#include <iostream>
#include <stdio.h>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>
#include "opencv2/video/background_segm.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <vector>

using namespace cv;

// Function prototypes
std::vector<cv::Point2d> findTargets(cv::Mat Image);
std::vector<cv::Point2d> orderTargets(std::vector<cv::Point2d> allTargets);
void drawPose(cv::Mat rotVec, cv::Mat transVec, cv::Mat K, cv::Mat dist,
... cv::Mat imageInput);

int main(int argc, char* argv[])
{
    // string filename = "Clark vs Torres.mpg";
    // string filename = "C:/Users/Justin Brewer/Documents/MATLAB
    // .../Jackson vs Esparza.mpg";
    string filename = "C:/Users/Justin Brewer/Documents/MATLAB
    .../Clark vs Rodriguez.mpg";

    VideoCapture cap;
    bool update_bg_model = true;
cap.open(filename);
    cv::VideoWriter output;
    cv::VideoWriter output2;

    output.open("C:/Users/Justin Brewer/Documents/Senior Year Files/
    ...ComputerVision/OpeCVprojects/Final/segmentedVideo.wmv",
    ...CV_FOURCC('W','M','V','1'), 29.97, cv::Size(720,528),false);
    output2.open("C:/Users/Justin Brewer/Documents/Senior Year Files/
    ...Computer Vision/OpeCVprojects/Final/contourVideo.wmv",
    ...CV_FOURCC('W','M','V','1'), 29.97, cv::Size(720,528),true);

    cv::BackgroundSubtractorMOG2 bg;//(100, 3, 0.3, 5);
    // BackgroundSubtractorMOG2 bg = BackgroundSubtractorMOG2(100, 3, false);
    bg.set("nmixtures", 3);
    // bg.set("bShadowDetection",true);
    // bg.set("fTau",0.5);
    // bg.set("nShadowDetection",255);

    std::vector < std::vector < cv::Point >> contours;

    cv::namedWindow ("Frame");
    cv::namedWindow ("Background");
```
Mat frame, fgmask, fgimg, backgroundImage;
backgroundImage = imread("C:/Users/Justin Brewer/Documents/...Senior Year Files/Computer Vision/OpeCVprojects/Final/backgroundImage.jpg");
for(;;)
{
cap >> frame;
bg.operator()(frame, fgimg, -0.5);
bg.getBackgroundImage(backgroundImage);
cv::erode (fgimg, fgimg, cv::Mat ());
cv::erode (fgimg, fgimg, cv::Mat ());
cv::dilate (fgimg, fgimg, cv::Mat ());
cv::dilate (fgimg, fgimg, cv::Mat ());
threshold(fgimg,fgimg, 200, 255,THRESH_BINARY);
cv::imshow ("Background", fgimg);
output.write(fgimg);

cv::findContours (fgimg, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);
cv::drawContours (frame, contours, -1, cv::Scalar (0, 0, 255), 2);

cv::imshow ("Frame", frame);
output2.write(frame);
//cv::imshow ("Background", fgimg);
//imwrite("C:/Users/Justin Brewer/Documents/Senior Year Files/...Computer Vision/OpeCVprojects/Final/backgroundImage.jpg",backgroundImage);
//imwrite("C:/Users/Justin Brewer/Documents/Senior Year Files/...Computer Vision/OpeCVprojects/Final/foreGroundImageImage.jpg",fgimg);
char k = (char) waitKey(5);
if( k == 27 ) break;
}

system("PAUSE");
return EXIT_SUCCESS;

B CamShift Code

#include "opencv2/video/tracking.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"

#include <iostream>
#include <ctype.h>

using namespace cv;
using namespace std;

Mat image;
bool backprojMode = false;
bool selectObject = false;
int trackObject = 0;
bool showHist = true;
Point origin;
Rect selection;
int vmin = 10, vmax = 256, smin = 30;

static void onMouse(int event, int x, int y, int, void *)
{
    if(selectObject)
    {
        selection.x = MIN(x, origin.x);
        selection.y = MIN(y, origin.y);
        selection.width = std::abs(x - origin.x);
        selection.height = std::abs(y - origin.y);

        selection &= Rect(0, 0, image.cols, image.rows);
    }

    switch(event)
    {
        case CV_EVENT_LBUTTONDOWN:
            origin = Point(x,y);
            selection = Rect(x,y,0,0);
            selectObject = true;
            break;
        case CV_EVENT_LBUTTONUP:
            selectObject = false;
            if(selection.width > 0 && selection.height > 0)
                trackObject = -1;
            break;
    }
}

static void help()
{
    cout << "This is a demo that shows mean-shift based tracking\n";
    cout << "You select a color objects such as your face and it tracks it.\n";
    cout << "This reads from video camera (0 by default, or the camera number the\nuser enters)\n";
    cout << "Usage: \n";
    cout << "/camshiftdemo [camera number]\n";

    cout << "Hot keys: \n";
    cout << "\tESC - quit the program\n";
    cout << "\tc - stop the tracking\n";
    cout << "\tb - switch to/from backprojection view\n";
    cout << "\th - show/hide object histogram\n";
    cout << "\tp - pause video\n";
    cout << "To initialize tracking, select the object with mouse\n";
}

const char* keys =
{
    "{1| 0 | camera number}"
int main( int argc, const char** argv )
{
    help();

    VideoCapture cap;
    string filename = "C:/Users/Justin Brewer/Documents/MATLAB/
    ...Jackson vs Esparza.mpg";
    //string filename = "C:/Users/Justin Brewer/Documents/MATLAB/
    ...Clark vs Rodriguez.mpg";
    Rect trackWindow;
    int hsize = 16;
    float hranges[] = {0,180};
    const float* phranges = hranges;
    CommandLineParser parser(argc, argv, keys);
    int camNum = parser.get<int>("1");

    cap.open(filename);
    //cv::VideoWriter output;
    //cv::VideoWriter output2;

    output.open("C:/Users/Justin Brewer/Documents/Senior Year Files/
    ...Computer Vision/OpeCVprojects/Final/camshiftVideo.wmv",CV_FOURCC('W','M','V','3')
    ...29.97, cv::Size(720,528),false);
    //output2.open("C:/Users/Justin Brewer/Documents/Senior Year Files/
    ...Computer Vision/OpeCVprojects/Final/contourVideo.wmv",CV_FOURCC('W','M','V','1')
    ...29.97, cv::Size(720,528),true);

    if( !cap.isOpened() )
    {
        help();
        cout << "***Could not initialize capturing...***\n";
        cout << "Current parameter’s value: \n";
        parser.printParams();
        return -1;
    }

    namedWindow( "Histogram", 0 );
    namedWindow( "CamShift Demo", 0 );
    setMouseCallback( "CamShift Demo", onMouse, 0 );
    createTrackbar( "Vmin", "CamShift Demo", &vmin, 256, 0 );
    createTrackbar( "Vmax", "CamShift Demo", &vmax, 256, 0 );
    createTrackbar( "Smin", "CamShift Demo", &smin, 256, 0 );

    Mat frame, hsv, hue, mask, hist, histimg = Mat::zeros(200, 320, CV_8UC3),
    ...backproj;
    bool paused = false;

    for(;;)
    {
        if( !paused )
        {
            cap >> frame;
            if( frame.empty() )
    22
break;
}
frame.copyTo(image);
if( !paused )
{
cvtColor(image, hsv, COLOR_BGR2HSV);
if( trackObject )
{
    int _vmin = vmin, _vmax = vmax;
    inRange(hsv, Scalar(0, smin, MIN(_vmin, _vmax)),
             Scalar(180, 256, MAX(_vmin, _vmax)), mask);
    int ch[] = {0, 0};
    hue.create(hsv.size(), hsv.depth());
    mixChannels(&hsv, 1, &hue, 1, ch, 1);
    if( trackObject < 0 )
    {
        Mat roi(hue, selection), maskroi(mask, selection);
        calcHist(&roi, 1, 0, maskroi, hist, 1, &hsize, &phranges);
        normalize(hist, hist, 0, 255, CV_MINMAX);
        trackWindow = selection;
        trackObject = 1;
        histimg = Scalar::all(0);
        int binW = histimg.cols / hsize;
        Mat buf(1, hsize, CV_8UC3);
        for( int i = 0; i < hsize; i++ )
        {
            buf.at<Vec3b>(i) = Vec3b(saturate_cast<uchar>(
            ...((i*180./hsize), 255, 255));
            cvtColor(buf, buf, CV_HSV2BGR);
            for( int i = 0; i < hsize; i++ )
            {
                int val = saturate_cast<int>((hist.at<float>
            ...((i)*histimg.rows/255));
                rectangle( histimg, Point(i*binW,histimg.rows),
                          Point((i+1)*binW,histimg.rows - val),
                          Scalar(buf.at<Vec3b>(i)), -1, 8 );
            }
        }
    }
    calcBackProject(&hsv, 1, 0, hist, backproj, &phranges);
    backproj &= mask;
    RotatedRect trackBox = CamShift(backproj, trackWindow,
                                      TermCriteria( CV_TERMCRIT_EPS |
                                                   ...CV_TERMCRIT_ITER, 100, 1 ));
    if( trackWindow.area() <= 1 )
    {
        int cols = backproj.cols, rows = backproj.rows, r =
...(MIN(cols, rows) + 5)/6;
trackWindow = Rect(trackWindow.x - r, trackWindow.y - r,
trackWindow.x + r, trackWindow.y + r) &
Rect(0, 0, cols, rows);
}

if( backprojMode )
    cvtColor( backproj, image, COLOR_GRAY2BGR );
ellipse( image, trackBox, Scalar(0,0,255), 3, CV_AA );
}
}
else if( trackObject < 0 )
    paused = false;

if( selectObject && selection.width > 0 && selection.height > 0 )
{
    Mat roi(image, selection);
    bitwise_not(roi, roi);
}

imshow( "CamShift Demo", image );
    output.write(image);
    imshow( "Histogram", histimg );

char c = (char)waitKey(10);
if( c == 27 )
    break;
switch(c)
{
    case 'b':
        backprojMode = !backprojMode;
        break;
    case 'c':
        trackObject = 0;
        histimg = Scalar::all(0);
        break;
    case 'h':
        showHist = !showHist;
        if( !showHist )
            destroyWindow( "Histogram" );
        else
            namedWindow( "Histogram", 1 );
        break;
    case 'p':
        paused = !paused;
        break;
    default:
        ;
}

output.release();
return 0;
% Initial variable and read in videos
movieObj = VideoReader('Jackson vs Esparza.mpg'); % open file
writerObj = VideoWriter('refTrack'); open(writerObj);
movieObj = VideoReader('contourVideo.mpeg'); % open file
background = imread('averageBackground1.png');
framerate = movieObj.FrameRate;
SegmentedVideo = VideoReader('segmentedVideo.mpeg'); % open file
RingImage = read(movieObj,1);
RingSize = size(RingImage);
height = RingSize(1);
width = RingSize(2);
refimage = zeros(height,width);

% Initialize Ring Points and initial position of each boxer.
Origin = [143 488];
Corners = [[119 63] [549 31] [143 488] [566 471] ];
Centers = [[337 18] [100 282] [594 238] [366 509] ];
Boxer1In = [480 316];
Boxer2In = [413 169];

% Define the frames to be analyzed.
numFrame = 150:800;

% Initialize data structure to store position data.
refXY = zeros(length(numFrame),2);
refXY(1,:) = RefIn;
boxRef = zeros(length(numFrame),4);
count = 1;

% For loop to loop through each frame being analyzed
for i = numFrame
    realimage = read(movieObj,i);
    image = read(SegmentedVideo,i);
    image = im2bw(image,0.5);
    regions = regionprops(image);
    number = size(regions);
    regarea = zeros(1,number(1));
    centroid = zeros(number(1),2);
    for j=1:number(1)
        regarea(j)=regions(j).Area;
        centroid(j,:) = regions(j).Centroid;
    end
end

% Preform connected component labeling.
regions = regionprops(image);
number = size(regions);
regarea = zeros(1,number(1));
centroid = zeros(number(1),2);
for j=1:number(1)
    regarea(j)=regions(j).Area;
    centroid(j,:) = regions(j).Centroid;
end

[B,I]=sort(regarea');

% Find the largest blobs.
blob1 = regions(I(number(1)));
blob2 = regions(I(number(1) - 1));
blob3 = regions(I(number(1) - 2));
c = blob1.Centroid;
d = blob2.Centroid;
e = blob3.Centroid;

% Show the location of the three largest blobs.
% imshow(image,[])
% line([c(1)-5 c(1)+5], [c(2) c(2)], 'Color', 'g');
% line([c(1) c(1)], [c(2)-5 c(2)+5], 'Color', 'g');
% line([d(1)-5 d(1)+5], [d(2) d(2)], 'Color', 'r');
% line([d(1) d(1)], [d(2)-5 d(2)+5], 'Color', 'r');
% line([e(1)-5 e(1)+5], [e(2) e(2)], 'Color', 'b');
% line([e(1) e(1)], [e(2)-5 e(2)+5], 'Color', 'b');

% Check if the largest blobs are located near the previous position
% locations
if norm(blob1.Centroid-refXY(count,:)) < 15
    refXY(count+1,:) = blob1.Centroid;
    boxRef(count,:) = blob1.BoundingBox;
elseif norm(blob2.Centroid-refXY(count,:)) < 15
    refXY(count+1,:) = blob2.Centroid;
    boxRef(count,:) = blob2.BoundingBox;
elseif norm(blob3.Centroid-refXY(count,:)) < 15
    refXY(count+1,:) = blob3.Centroid;
    boxRef(count,:) = blob3.BoundingBox;
else
    refXY(count+1,:) = refXY(count,:);
    boxRef(count,:) = boxRef(count-1,:);
end

% refimage = zeros(height,width);
% refimage = realimage;
% refimage = colorMap;

% Display location on image
refimage(round(refXY(count,2))-1:round(refXY(count,2))+1,round(refXY(count,1))-1:round(refXY(count,1)+1,:),:) = 255;

imshow(refimage,[])

line([119 337],[63 18],'Color','r')
line([337 549],[18 31],'Color','r')
line([549 594],[31 238],'Color','r')
line([594 566],[238 471],'Color','r')
line([566 366],[471 509],'Color','r')
D Matlab Heat Map Generation Code

```matlab
% heatmap plotting
dim = size(refXY);
colorMap = zeros(height, width);

for k = 1:dim(1)-1
    h = fspecial('gaussian', [boxRef(k,4) boxRef(k,3)], 18);
    gaus = size(h);
    colorMap(floor(boxRef(k,2)):floor(boxRef(k,2))+gaus(2)-1),
    ... floor(boxRef(k,1)):floor(boxRef(k,1))+gaus(1)-1)) =
    colorMap(floor(boxRef(k,2)):floor(boxRef(k,2))+gaus(2)-1),
    ... floor(boxRef(k,1)):floor(boxRef(k,1))+gaus(1)-1)) + h';
end

% colormap(hot)
% imageesc(colorMap)
% imshow(colorMap, [])
% colorbar

% I = HeatMap(colorMap, 'X', 'Y');

heatmap(colorMap)

% Draw Outline of Boxing Ring
line([119 337], [63 18], 'Color', 'r')
line([337 549], [18 31], 'Color', 'r')
line([549 594], [31 238], 'Color', 'r')
line([594 566], [238 471], 'Color', 'r')
line([566 366], [471 509], 'Color', 'r')
line([366 143], [509 488], 'Color', 'r')
line([143 100], [488 282], 'Color', 'r')
line([100 119], [282 63], 'Color', 'r')
```