H. Chase Stevens, s1107496

Joris Sebastijonas Urbaitis, s1104617

Introduction to Vision and Robotics Practical 1 Report

# Introduction

In order to detect the objects found in the videos, our Matlab program processed each frame and detected moving objects. The information from different frames was linked together to form a coherent picture of the movement of the objects being thrown. We then detected whether these objects were balls and used basic kinematics to detect where the apex of their trajectory would be.

# Methods

## Thresholding

In order to detect the objects, we first have to preprocess each frame. First, the image is converted to grayscale. We then take the absolute difference between the frame and the reference background image. We use the result to create a histogram of greyscale values and use the algorithm provided during labs (findthresh.m) to find a threshold. The difference image is then filtered and converted to black and white using this threshold and, as a final step, bwmorph’s clean method is used to remove lone pixels. The remaining blobs in the image are then expanded, with pixel within 5 pixels of a white pixel being made white. This means that nearby blobs become connected. The downside is that they become more circular and make ball detection trickier.

Initially, the first frame of the video is selected to be the reference background image and converted to greyscale. Each frame, the current reference background image is split into a set number of horizontal slices. The total difference between the pixels of a background slice and the corresponding slice in a greyscaled version of the current frame is checked. If this distance is under a set threshold, that slice is updated by setting it to be the current frame. This method was chosen to avoid cases in which parts of the background had been reliably altered, but a moving object elsewhere in the frame was blocking this part of the background from being updated. The total effect of these efforts was that if the background reference is no longer providing sufficient information about the moving objects in the frame, it is updated. This allows us to quickly get rid of noise appearing in various parts of the image that have nothing to do with the objects being thrown up, for example, when the background cloth moves.

## Detecting connected components

To detect the connected components in the thresholded black and white image we used the Matlab function bwconncomp, with the default connectivity of 8. This checks a 3x3 region around every pixel for non-zero values. It then uses the number of these to classify the pixel either as part of a larger component or, if it is not connected to any known component, as a novel component. The function ultimately returns a list of pixel indices for every component that we then use for object and movement detection.

## Linking data from different frames

In order to link data between different frames we first filter the connected component data in order to remove objects that are very small or very large and likely to be noise. We then find the center of each object and calculate a pseudo-Euclidean distance in three-dimensional space (x coordinate, y coordinate, and time) between this object and every object detected in the last few frames, using the following formula:

The closest of these within a specified threshold is deemed to be the same object; if no past object falls within this threshold we create a new object with a unique identifier. It should be noted that, in our implementation, it is possible for multiple connected components to sometimes be classified as the same object, allowing for error in their detection.

## Apex of the trajectory

Kinematics is used to detect if an object has reached the highest point on the Y-axis of its trajectory. After linking data from different frames we can calculate the Y-axis velocity of the object. We know that the velocity at the apex of the trajectory of the ball will be zero. We also know that speed can be calculated using the following physics formula:

Since the relevant acceleration force in this case is gravity and can be assumed to be 0 (as we are calculating relative to the current moment in time), we can derive the time *t* at which the velocity will be 0 by calculating:

Here *g* is the gravitational constant. It should be noted that although we use the accepted value of 9.8 meters per second squared for the gravitational constant, the relationship this bears to our final time calculation is tenuous at best.

After calculating the time *t* at which the velocity will be zero, we can use it to decide whether the object is close to its apex by considering the size of *t*. If *t* is under a certain threshold, we mark the object as being at its apex. This can have the result of marking the apex very slightly before the actual highest point, but this is mostly invisible to the human eye in our experiments. The upside is that we do not consider future frames or delay displayed frames to calculate this.

## Ball detection

We take the area of the connected component we are trying to classify. We ‘overlay’ a circle of the same size on top of it and check what percentage of the object falls within the circle, giving us a compactness metric. Using a pre-set threshold we can then decide whether the object is a ball or not.

# Results

We didn’t use one of the videos for testing, saving it to be run only with the final version of our code. The results of correct ball detection in that run are as follows:

|  |  |  |
| --- | --- | --- |
|  | Positives | Negatives |
| True | 6 | 2 |
| False | 2 | 0 |

In all but one of the true positive cases, the apex was detected at the correct time – in the outlier case, the apex was detected earlier than was appropriate.

## Image pipeline

Here, we outline the steps images go through when being processed. This process begins with a raw, full-color frame (see Figure 1) and a background image (see Figure 2), and returns a mask, which is then in turn used (in conjunction with previous data on objects) to infer current objects. We take the full-color frame and translate it into greyscale, then take the absolute difference of this and the background (see Figure 3). We then threshold this resultant image to create the binary image seen in Figure 5 which, after applying bwdist, becomes Figure 6. After linking the connected components in this binary image to previously seen objects, we can draw traces of these objects’ previous histories while using the binary image as a mask for the frame, resulting in Figure 8.



Figure 1. Raw frame 768 from video GOPR0008.



Figure 2. Background image used at time of frame 768.

|  |  |
| --- | --- |
| C:\Users\Chase Stevens\AppData\Local\Microsoft\Windows\INetCache\Content.Word\fig_3.png  Figure 3. Absolute background-frame difference. | C:\Users\Chase Stevens\AppData\Local\Microsoft\Windows\INetCache\Content.Word\fig_4.png  Figure 4. Histogram of Figure 3. |

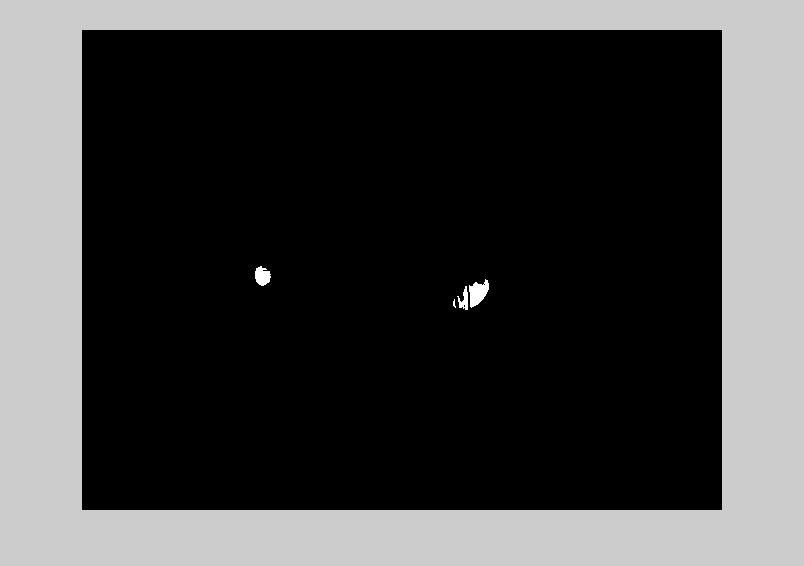


Figure 5. Thresholded background-frame difference.

|  |  |
| --- | --- |
| C:\Users\Chase Stevens\AppData\Local\Microsoft\Windows\INetCache\Content.Word\fig_6.png  Figure 6. "Thickened" thresholded image. | I:\Users\Chase Stevens\Documents\GitHub\ivr1\fig_7.png  Figure 7. Figure 6 used as a mask for frame 768. |

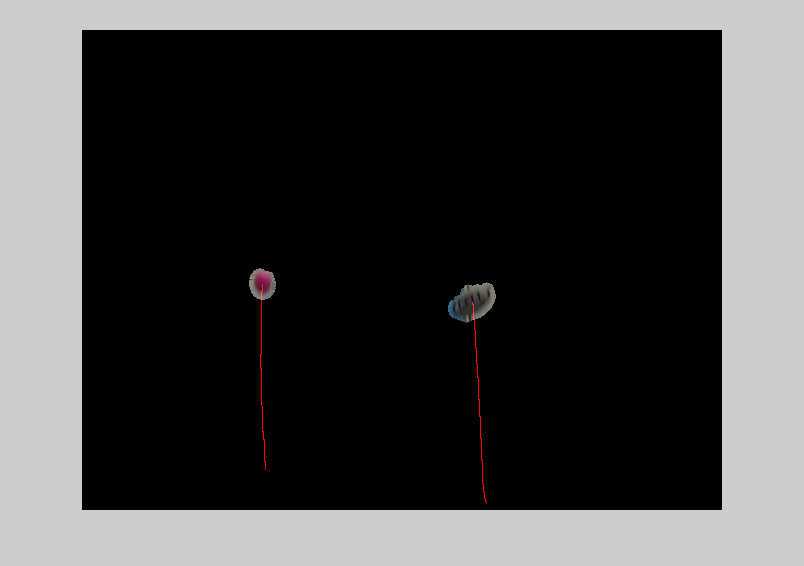


Figure 8. Masked frame with object traces.

* For one frame:
  + Histogram
  + Background used
  + Mask pre-expansion
  + Mask post-expansion
  + Object paths
* Errors
  + Misdetection of objects -
  + Misclassification -
  + Incorrectly timed apex -
  + Object as multiple connected components/poor keying -

# Discussion

Filtering movement of irrelevant objects could be better.

# Work Distribution

# Code