COMP3419 Assignment

Niravit Theng

October 2018

1 Introduction

In this report we will outline the implementation on how the tracking of the monkey works and any animation techniques such as replacing the background with a video. First the video has a size of 586 x 320 but our output video will be size of 1280 x 720. The monkey has 5 red markers on it in total and we will use that marker to track the movement of the monkey.

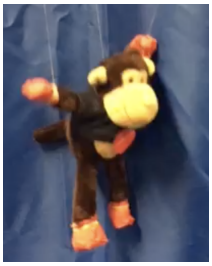

Figure 1: The monkey to capture

2 Implementation

The main focus of the implementation will be in several stages. First will be motion capturing then replacing background and monkey, then finally adding in intelligent objects and soundtrack.

2.1 Motion capture

As the red markers are there for us to track the monkey's movement. It is really difficult to keep track of it as the marker are not always visible (especially the one on the body) and the colour are not always red as shadows cast over it. This made it really difficult to capture the monkey movements.
Our first goal is to remove any other colour we don't want. First we tried out a basic filter which is $red > 150 and blue < 60 and green < 40$. This of course did

not work out well as the face of the monkey contain some red in some frame. This approach do give us basic tracking.
So the next approach is to introduced more filter which can filter out the face colours and browns. To get close to filtering out any unnecessary colours as possible we employ the use of eyedropper tool in photoshop to check the RGB value of each pixel in each frame. We sample only from problematic frames only instead of going through all the frames. This include some guess working also to try produce the closet result to the motion as possible.

```
red = red(c) > 150  && green(c) > 37 && green(c) < 200 &&
blue(c) > 30 && blue(c) < 125;

face = red(c) > 190 && red(c) < 255 && green(c) > 130 &&
green(c) < 200 && blue(c) > 45 && blue(c) < 125;

brown = red(c) > 150 && red(c) < 200 && green(c) > 80 &&
green(c) < 165 && blue(c) > 40 && blue(c) < 115;
```

Above is the values use to filter the pixel and if the pixel returns true for all of these it will be draw as white pixel on a new image.

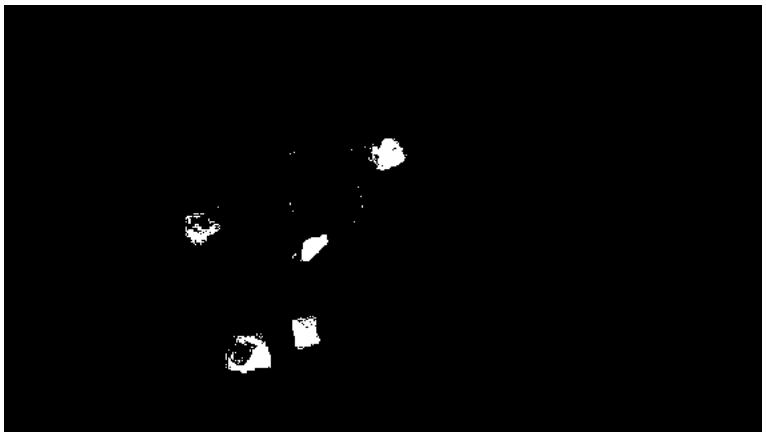

Figure 2: The filtered out pixels drawn on new image

After obtaining the frames with segmented markers which is in binary image. This makes the tracking a lot easier as the colour in binary are only in black or white. As seen in the figure above there are still left over pixels which are not needed for our tracking. Thus as suggested by the assignment specification to use morphological operations, we have tested all operations such as erosion, dilation, opening and closing. The results is that doing erosion then dilation seems to be the best result but we don't just perform it once but uses erosion 3 times and dilation 8 times to achieve a good result. The erosion uses in the code is a simplify version of the one in tutorial 2. Instead of using a kernel of 3x3 with 0 and 1 in it, we use a boolean 1D array of size 5 to store the pixel,

centre, left, right, up and down as we will only look at those neighbour pixels when at a certain pixel. If all the elements inside boolean array is true, the pixel in the new image is colour white with the same co-ordinate. For dilation it is similar, we don't employ a kernel or array but simply if we see a white pixel we will colour it white in the new image.

Figure 3: The improved segmented image

After obtaining a good quality marker for our monkey we will now simplify the markers into just 1 co-ordinate we can use to track our movement.
First we will iterate through the frame to find the minimum and maximum point for x and y as we can think of the monkey movement as a whole as a box with 4 corners being, (minimum x, minimum y) for the top left corner, (maximum x, minimum y) for the top right corner, (minimum x, maximum y) for the bottom left corner and (maximum x, maximum y) for the bottom right corner. These points are found by only checking for white pixels. The for the centre point we just find by (maxX + minX)/2 and same with y co-ordinate.

After obtaining those points, we will begin searching from each corner to the middle of the box. This ensure that it will only search each quadrant of the box and does not overlap with other parts. Each of the corner will search its way to the middle and stop there. If it see a white pixel, we will use that point for our tracking thus the first white pixel discovered in the quadrant will be use only. If for some reason the search cannot find a white pixel, we will use the min and max points of that corner instead.
The centre point is still calculated with min and max x,y and does not need the quadrant search. Sometimes the body marker disappears in some frames thus using this method ensure it will always be present. Since we only need 5 points per frame, an ArrayList of size 5 of type PVector is use to store all the points founded. The figure below is the image representation of the algorithm.

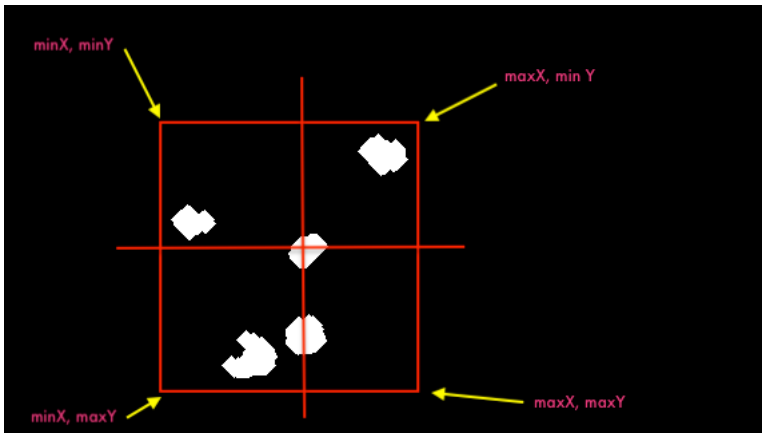

Figure 4: The quadrant to search

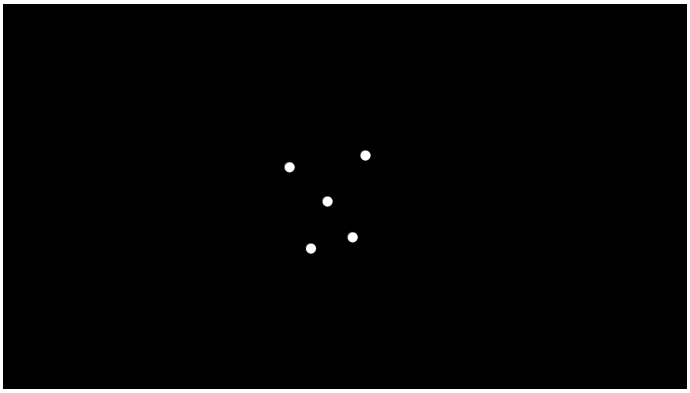

Figure 5: The points from quadrant search visualise with an ellipse

2.2 Replacing background and monkey

As we use Processing, replacing the background is trivial as we start with a blank canvas and replace the background with a video. This is done by reading the frame of the background video and displaying them in succession.

For replacing the monkey it is also trivial as we have the 5 points needed to replace each body part of the monkey, we just display an image at those co-ordinate accordingly in each frame.

2.3 Intelligent Object

The drawing of these added object were trivial. The image loading does not need to be discuss. The spawning on the object has 2 variations. Either they spawn at 0 - image width or 1280 + image height depending on a random generated

boolean. We also randomly generate the y co-ordinate too. For the second variation, we spawn from any four corner of the screen and travel diagonally.

They interact with the replaced monkey and another intelligent object by collision. To determine collision we just need to check if the x values between two object overlaps or not. If it does it means it collided. We also taken into the account of sprite length rather than colliding the the original dot point in figure 5.

2 interaction happens when collision happens. The chocolate coronet when hit with the marionette(monkey) will trigger a sound effect and disappear while the other intelligent object ignores the marionette but when collide with coronet object it transform and play a different sound effect.

2.4 Soundtrack

Soundtrack is trivial as we use Processing sound library and simply play it when collision happens.

3 Conclusion

Overall the techniques used in this report did successfully achieved what we wanted to do. To capture the motion of the monkey as close as we can. Perhaps there might be a better and efficient way of achieving this but due to time constraint, the result will use techniques outline by this report. It would also help if the colour of the monkey and its movement are more clear.