



ASSIGNMENT 2

282.762 Robotics and Automation

Caelan Vandermey, 14087877

Table of Contents

Table of Figures	i
1. Introduction	1
2. Program Explanation.....	1
2.1. Global Setup.....	1
2.2. Main Function	1
2.2.1. JSON Data Container	1
2.2.2. Image Choosing.....	2
2.3. Input and Output Control	2
2.4. Gathering Circles.....	3
2.4.1. Masks	3
2.4.2. Circle Finding and Drawing	3
3. Program Inputs and Outputs	5
References	6

Table of Figures

Figure 1 Global Setup Code.....	1
Figure 2 JSON Data Type Initialisation	2
Figure 3 Image Choosing Code in Main	2
Figure 4 getCircles Function Code	3
Figure 5 Masks. a) Red Masks, b) Yellow Mask, c) Blue Mask	3
Figure 6 getBlue Function Code.....	4
Figure 7 Image Inputs and Outputs	5

1. Introduction

The task that has been assigned is to take five images to find the coloured tennis balls and to identify the colour. This requires a two-step process; Identifying the coloured regions and locating the tennis balls within them. The complete process of this is shown in Section 2: Program Explanation. The images that were used in this assignment are shown in Section 3: Program Inputs and Outputs, as well as the output images from the program.

2. Program Explanation

The created program can be broken down into four distinct parts; Global Setup, Main Function, Input and Output Control and Gathering Circles. These parts are discussed below.

2.1. Global Setup

The global setup has three parts to it. These parts are the includes of other libraries, declaring namespaces and function prototypes.

In Figure 1, line one includes the OpenCV libraries that allow for the image processing functions (i.e. `cvtColor`, `inRange`, `dilate`, `erode`, etc.) and line two includes a library called “JSON for Modern C++” (nlohmann, 2017) which allows for the use of JavaScript Object Notation (JSON) formatted data storage in C++, of which, is very crucial in the structure of the program.

Lines four and five declare the `cv` and `std` namespaces respectively. Line six makes using the JSON data type easier to use in the following code.

Lines eight to eleven are the function prototypes for the functions that are called later in the program.

```
1  #include <opencv2\opencv.hpp>
2  #include "json.hpp"
3
4  using namespace cv;
5  using namespace std;
6  using json = nlohmann::json;
7
8  void getCircles(json vals, char* s);
9  void getRed(Mat* imageIn, Mat* image, json vals);
10 void getYel(Mat* imageIn, Mat* image, json vals);
11 void getBlu(Mat* imageIn, Mat* image, json vals);
```

Figure 1 Global Setup Code

2.2. Main Function

The main function has two major parts; The JSON Data Container and Image Choosing.

2.2.1. JSON Data Container

A JSON data container is an object based data type that is easy for computers to parse/generate and for people to read (JSON, 2017). The JSON data container is initialised in lines 14 to 36 of Figure 2 and has five objects contained within it, one for each image with the same name as the images. Each of the image named objects contains three objects, named after the colour of the tennis balls that it will be used to find. Within these colour separated objects are six more objects that contain the integers that will be used in the following functions. All of this is much easier to read from the raw code available in Figure 2.

```

14 json values = {
15     { "Image 1.jpg", {
16         { "R", { { "d1",1 }, { "e1",1 }, { "d2",4 }, { "e2",1 }, { "gaus",11 }, { "cMin",70 }, { "dist",20 } } },
17         { "B", { { "d1",4 }, { "e1",2 }, { "d2",2 }, { "e2",1 }, { "gaus",9 }, { "cMin",56 }, { "dist",30 } } },
18         { "Y", { { "d1",2 }, { "e1",1 }, { "d2",2 }, { "e2",1 }, { "gaus",11 }, { "cMin",56 }, { "dist",30 } } } } },
19     { "Image 2.jpg", {
20         { "R", { { "d1",4 }, { "e1",1 }, { "d2",4 }, { "e2",2 }, { "gaus",11 }, { "cMin",70 }, { "dist",18 } } },
21         { "B", { { "d1",2 }, { "e1",2 }, { "d2",3 }, { "e2",1 }, { "gaus",9 }, { "cMin",56 }, { "dist",30 } } },
22         { "Y", { { "d1",1 }, { "e1",4 }, { "d2",1 }, { "e2",2 }, { "gaus",9 }, { "cMin",56 }, { "dist",17 } } } } },
23     { "Image 3.jpg", {
24         { "R", { { "d1",4 }, { "e1",1 }, { "d2",4 }, { "e2",1 }, { "gaus",11 }, { "cMin",56 }, { "dist",20 } } },
25         { "B", { { "d1",4 }, { "e1",1 }, { "d2",3 }, { "e2",1 }, { "gaus",9 }, { "cMin",56 }, { "dist",21 } } },
26         { "Y", { { "d1",1 }, { "e1",2 }, { "d2",1 }, { "e2",1 }, { "gaus",9 }, { "cMin",56 }, { "dist",23 } } } } },
27     { "Image 4.jpg", {
28         { "R", { { "d1",1 }, { "e1",1 }, { "d2",4 }, { "e2",2 }, { "gaus",11 }, { "cMin",56 }, { "dist",20 } } },
29         { "B", { { "d1",4 }, { "e1",1 }, { "d2",2 }, { "e2",1 }, { "gaus",9 }, { "cMin",56 }, { "dist",21 } } },
30         { "Y", { { "d1",1 }, { "e1",2 }, { "d2",1 }, { "e2",1 }, { "gaus",9 }, { "cMin",56 }, { "dist",24 } } } } },
31     { "Image 5.jpg", {
32         { "R", { { "d1",2 }, { "e1",3 }, { "d2",3 }, { "e2",2 }, { "gaus",11 }, { "cMin",70 }, { "dist",22 } } },
33         { "B", { { "d1",1 }, { "e1",2 }, { "d2",1 }, { "e2",1 }, { "gaus",9 }, { "cMin",56 }, { "dist",22 } } },
34         { "Y", { { "d1",1 }, { "e1",2 }, { "d2",1 }, { "e2",1 }, { "gaus",9 }, { "cMin",56 }, { "dist",26 } } } } }
35 };

```

Figure 2 JSON Data Type Initialisation

2.2.2. Image Choosing

The remainder of the main function is dedicated to string manipulation and passing this string on to the “getCircles” function, that results in an image being written to the computer’s hard drive. The string manipulation is used to ensure that each image has the tennis balls identified.

This is achieved through a “for” loop that modifies the seventh character in the string “s” (line 41 in Figure 3. Which is the character that would be one or two etc, denoting the name of the input image and the part of the object that will be passed on the “getCircles” (e.g. “Image 1.jpg” is the image filename and the reference for the appropriate data in the “values” object).

```

37 char s[] = "Image x.jpg";
38 int i = 0;
39
40 for (i = 0; i < 5; i++) {
41     s[6] = i + 49;
42     getCircles(values[s], s);
43 }
44

```

Figure 3 Image Choosing Code in Main

2.3. Input and Output Control

The input/output control is run through the “getCircles” function, shown in Figure 4. Where it reads the image in colour on line 50 and then converts the image from Blue Green Red (BGR) to Hue Saturation Value (HSV) (line 51) and stored in another Mat, named “image”.

Lines 53, 54 and 55 call the circle gathering functions “getRed”, “getYel” and “getBlu”, respectively, which draw coloured circles on the input image. These functions are given the locations of the original and converted images (so that the functions can read and edit these images) and the respective settings from the JSON data container. These functions do not return anything as they change the images being given to them.

Line 57 changes the output file name (“outName”) from “Image x out.jpg” to the appropriate filename (i.e. “Image 1 out.jpg” for image one). Line 58 writes the now edited input image under the filename stored in “outName”.

```

46 void getCircles(json vals, char* s) {
47     Mat imageIn, image;
48     char outName[] = "Image x out.jpg";
49
50     imageIn = imread(s, CV_LOAD_IMAGE_COLOR);
51     cvtColor(imageIn, image, COLOR_BGR2HSV);
52
53     getRed(&imageIn, &image, vals["R"]);
54     getYel(&imageIn, &image, vals["Y"]);
55     getBlu(&imageIn, &image, vals["B"]);
56
57     outName[6] = s[6];
58     imwrite(outName, imageIn);
59 }

```

Figure 4 getCircles Function Code

2.4. Gathering Circles

The circle gathering functions are “getRed”, “getYel” and “getBlu” which find and draw the circles for the colours red, yellow and blue respectively. These functions each have their own set of masks to find the correct colours in the image. Because of this, these functions have two sections in this report; Masks and Circle Finding and Drawing.

2.4.1. Masks

The three functions each have their own mask as the colours are in different ranges. Being in the HSV colour space makes colour identification much easier. The hue determines the colour where the saturation and value determine intensity and brightness. By adjusting the hue, finding the correct colour range is much easier. The saturation and value can then be a way of trimming the image to make circle detection easier (for when the circles are on objects of a similar colour).

The “inRange” function is used to show what pixels fit within the ranges given. Yellow, for example, has a hue range of 25-80, a saturation range of 50-255 and a saturation range of 106-255 (this is shown in Figure 5b. The blue ranges follow the same format but have different ranges, as shown in Figure 5c.

The red ranges, however, are spread across both ends of the hue spectrum. This is because red is at the maximum and minimum on a HSV scale. Because of this, there are two “inRange” functions used (shown in Figure 5a); one for the hue range of 170-178 and one for the hue range of 0-5. These two ranges each have different saturation and value ranges and this is used to reduce the undesired red from passing through, rather than having one set each for those ranges. The two mask images are then added together in line 67. Adding in this way is applicable as both images are binary, with either black or white in the picture. If a pixel value exceeds the maximum then it will be considered white anyway, resulting in no issue occurring.

- | | |
|----|--|
| a) | <pre> 65 inRange(*image, Scalar(170, 180, 120), Scalar(178, 215, 255), imageR); 66 inRange(*image, Scalar(0, 132, 50), Scalar(5, 255, 180), redTmp); 67 imageR = imageR + redTmp; </pre> |
| b) | <pre> 90 inRange(*image, Scalar(25, 50, 106), Scalar(80, 255, 255), imageY); </pre> |
| c) | <pre> 113 inRange(*image, Scalar(105, 175, 30), Scalar(120, 255, 240), imageB); </pre> |

Figure 5 Masks. a) Red Masks, b) Yellow Mask, c) Blue Mask

2.4.2. Circle Finding and Drawing

All three of the circle finding and drawing functions follow a very similar format, where the only difference is that each one has a different mask. Line 113 in Figure 6 is the same line as what is shown in Figure 5c and this line could be swapped with the line of code in Figure 5b and the program will now mask for yellow. Assuming the correct JSON data has been given, this would result in finding the yellow tennis balls, rather than the blue ones.

The first two lines of the functions are also very similar as a temporary Mat is required to store the mask (a different name is used in each function that represents the colour being masked for [red,

yellow or blue]) and the vector is required as this is where the found circles will be recorded (in the format of x, y, radius).

After the mask for the colour has been made, the image is adjusted to remove noise. This is achieved using the dilate and erode functions on the binary images created. The dilate function increases the boundaries of the white space, making them larger. The erode function does the opposite of this and reduces the boundaries of the white space, making them smaller. The use of these two functions makes finding the circles that represent the tennis balls easier and more reliable. Rather than changing the size of the matrices used in this, the functions allow for multiple runs of the dilation/erosion without repeatedly calling the function. This is achieved by adjusting the third to last parameter. The JSON object has four integers for this, each representing one function call. This can be seen in lines 115 through 118 in Figure 6 where the "vals[*]" (the "*" represents the name of the value being used) parts get the integer from the JSON container.

The image is then blurred using the "GaussianBlur" function to blur the edges of the image (shown on line 120 in Figure 6). This is also adjusted by integers in the JSON data container. These values change the size of the kernel used to blur the image.

Line 122 shows the "HoughCircles" function, which is a variant of the Hough transformation. This is used as a tennis ball will likely be a circle or an arc on a circle. Using this to spot the circles in the blurred image allows the location of the tennis balls to be confirmed. When a circle is found, it has the coordinates of the centre and the radius of the circle stored in the "circles" vector array. The fifth argument in the function and alters the minimum distance between the circles and the seventh argument changes the accumulator threshold. Both values are provided by the JSON data container.

The final part of these functions is the drawing of the circles on the input image. This is achieved by running a loop for each set of data in the "circles" vector array. In each pass of the loop, the data is retrieved from the array and then the circles are drawn on the image by the "circle" function. The first call of the "circle" function (on line 127) draws the centre of the circle in the opposite of the colour being searched for, which is yellow for the blue tennis balls. The second "circle" call draws the outline of the circle that was found, and is in a slightly different colour to the tennis ball so that it is visible but still shows that it is a ball of the right colour. A red tennis ball would get a cyan centre and a red outline and a yellow ball would get a yellow/green outline and a purple centre (this is because the yellow tennis ball is more of a green in some images).

Now that the circles have been drawn, this function ends and the next colour tennis ball is searched for or the image is written to the hard drive.

```
109 void getBlu(Mat* imageIn, Mat* image, json vals) {
110     Mat imageB;
111     vector<Vec3f> circles;
112
113     inRange(*image, Scalar(105, 175, 30), Scalar(120, 255, 240), imageB);
114
115     dilate(imageB, imageB, Mat(), Point(-1, -1), vals["d1"], 1, 1);
116     erode(imageB, imageB, Mat(), Point(-1, -1), vals["e1"], 1, 1);
117     dilate(imageB, imageB, Mat(), Point(-1, -1), vals["d2"], 1, 1);
118     erode(imageB, imageB, Mat(), Point(-1, -1), vals["e2"], 1, 1);
119
120     GaussianBlur(imageB, imageB, Size(vals["gaus"], vals["gaus"]), -1, -1);
121
122     HoughCircles(imageB, circles, CV_HOUGH_GRADIENT, 1, vals["cMin"], 200, vals["dist"], 30, 110);
123
124     for (size_t i = 0; i < circles.size(); i++) {
125         Point c(cvRound(circles[i][0]), cvRound(circles[i][1]));
126         int r = cvRound(circles[i][2]);
127         circle(*imageIn, c, 10, Scalar(0, 255, 255), -1, 8, 0);
128         circle(*imageIn, c, r, Scalar(255, 0, 0), 10, 8, 0);
129     }
130 }
```

Figure 6 getBlue Function Code

3. Program Inputs and Outputs

This section of the report shows the inputs and outputs from the program. Figure 7 shows the five test images and their corresponding outputs from the program. Column A shows the input and Column B shows the output. Each row is labelled to show the image that is being processed (i.e. the first row with images is labelled with a one, meaning it is showing image one's input and output).







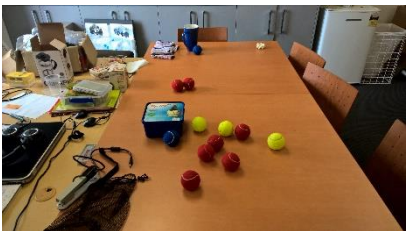



	A	B
1		
2		
3		
4		
5		

Figure 7 Image Inputs and Outputs

The above image pairs show that the program can identify the tennis balls in the images, even if they have been grouped up to the point that the balls are blocking each other from view or if they are on/in front of objects of the same colour as the ball is. These situations have been solved by limiting the HSV ranges to disallow most of the extra objects from passing through the “inRange” function.

Some of the circles are not perfectly centred and this is likely down to two major factors. One is that the dilation and erosion does not completely recreate the circles from the binary image. And the Other is that the “HoughCircles” function looks for curved edges in an image and does not allow circles to be too close together. Some of the circles found are from the white patterns on the tennis ball itself, rather than the whole ball. This could be changed by increasing the minimum circle size, but that would remove the circles from the tennis balls in the back of the image. It could also be changed by dilating and eroding more in the program, but the more the image is morphed, the less representative the output becomes.

References

JSON. (2017, May 10). Retrieved from Introducing JSON: <http://json.org/>

nlohmann. (2017, May 12). *GitHub - nlohmann/json: JSON for Modern C++*. Retrieved from GitHub: <https://github.com/nlohmann/json>