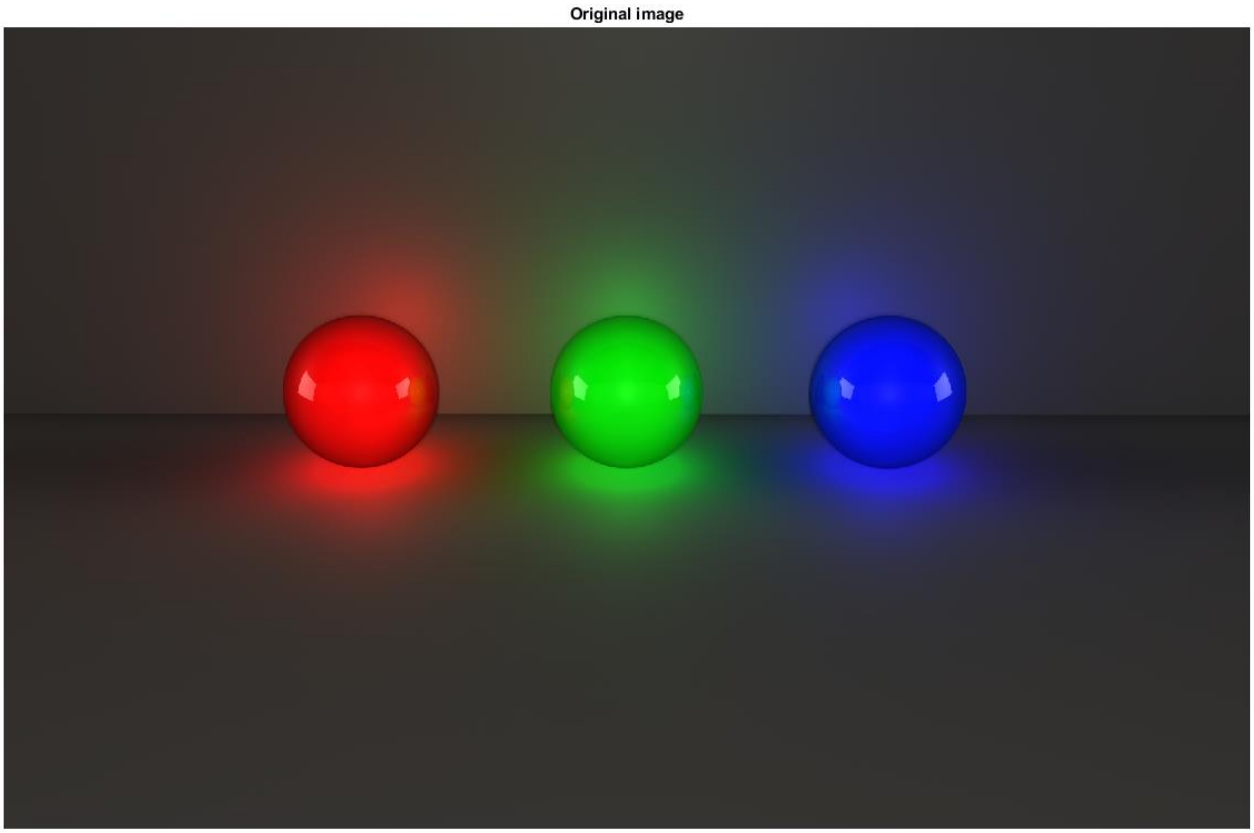
Introduction to Robotics and Mechatronics

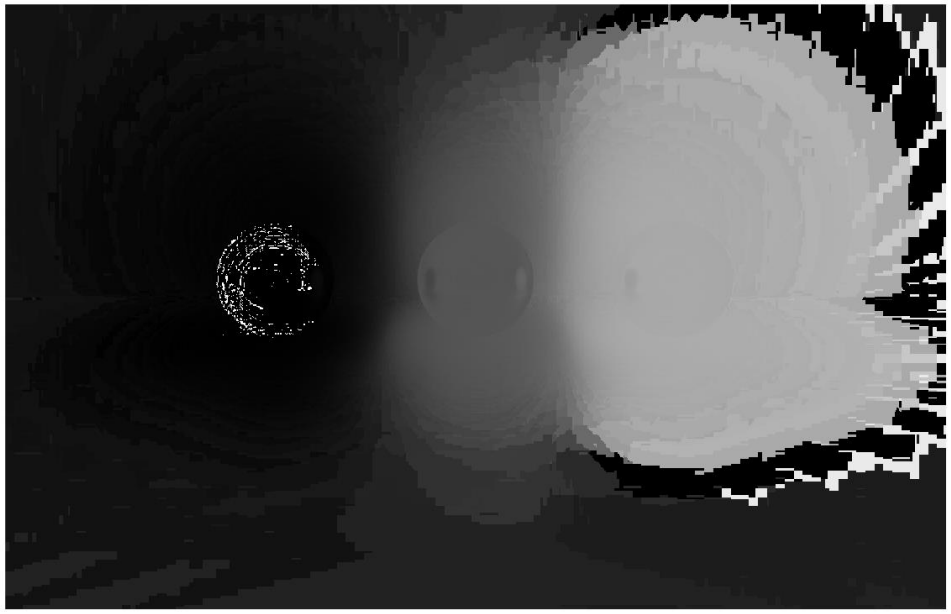
GROUP 2.5
Benson Chalethu
Pietro Griffa
Ingvar Groza

PreLab 06

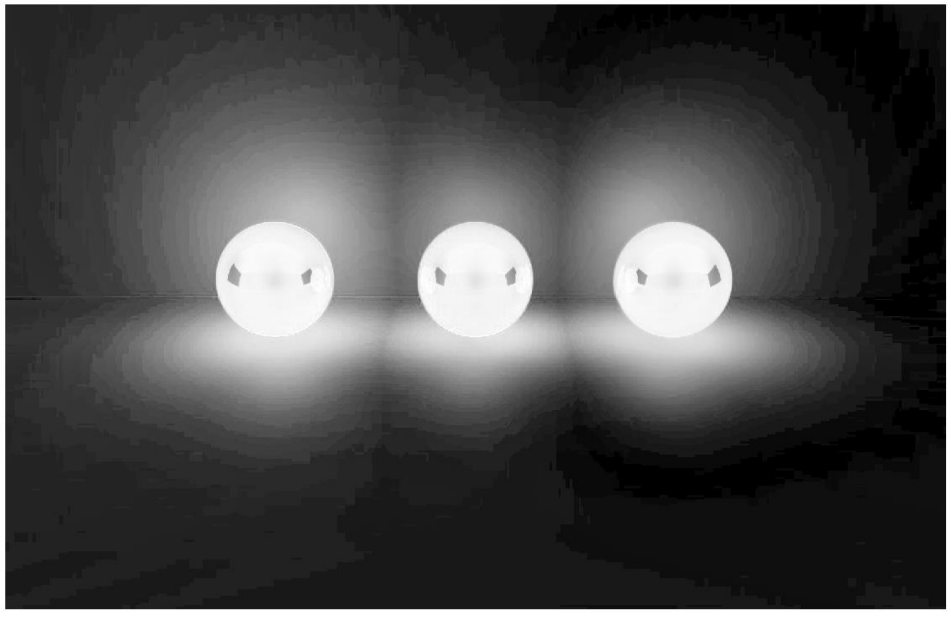
Q1



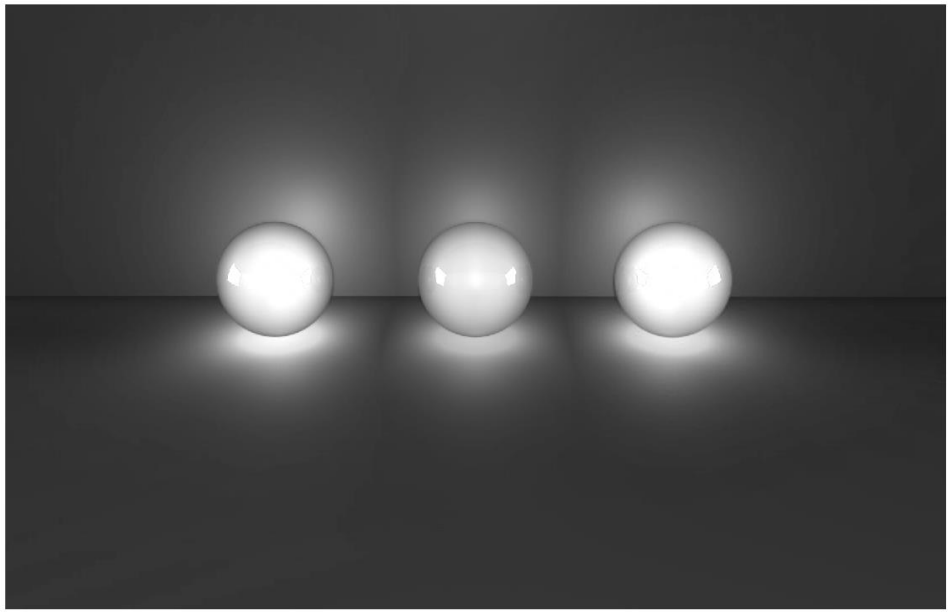
Hue



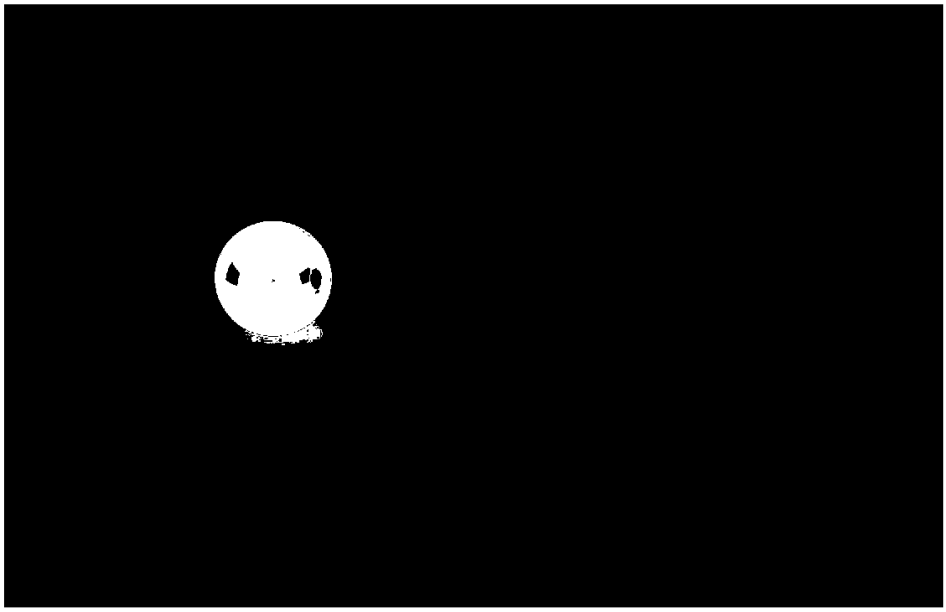
Saturation



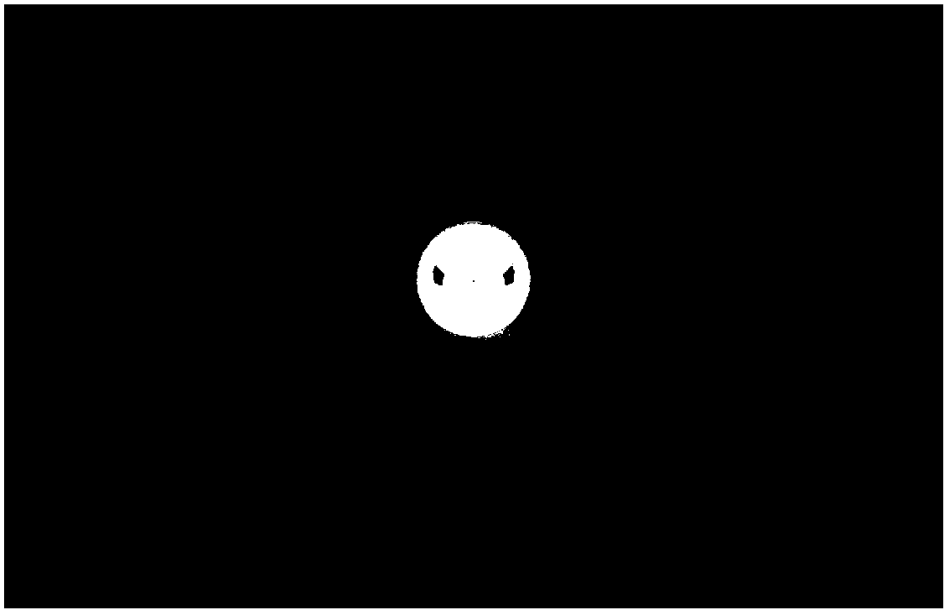
Value



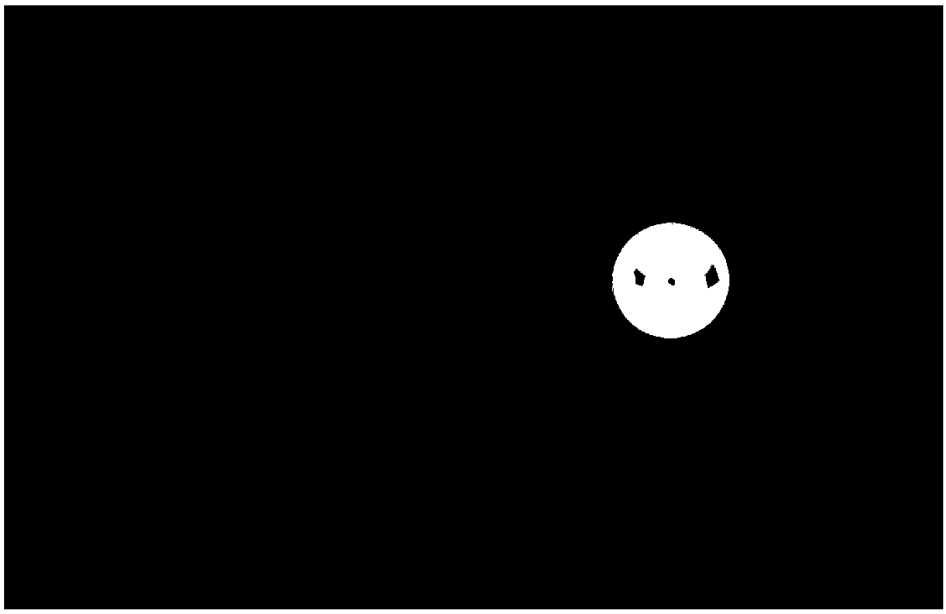
Red

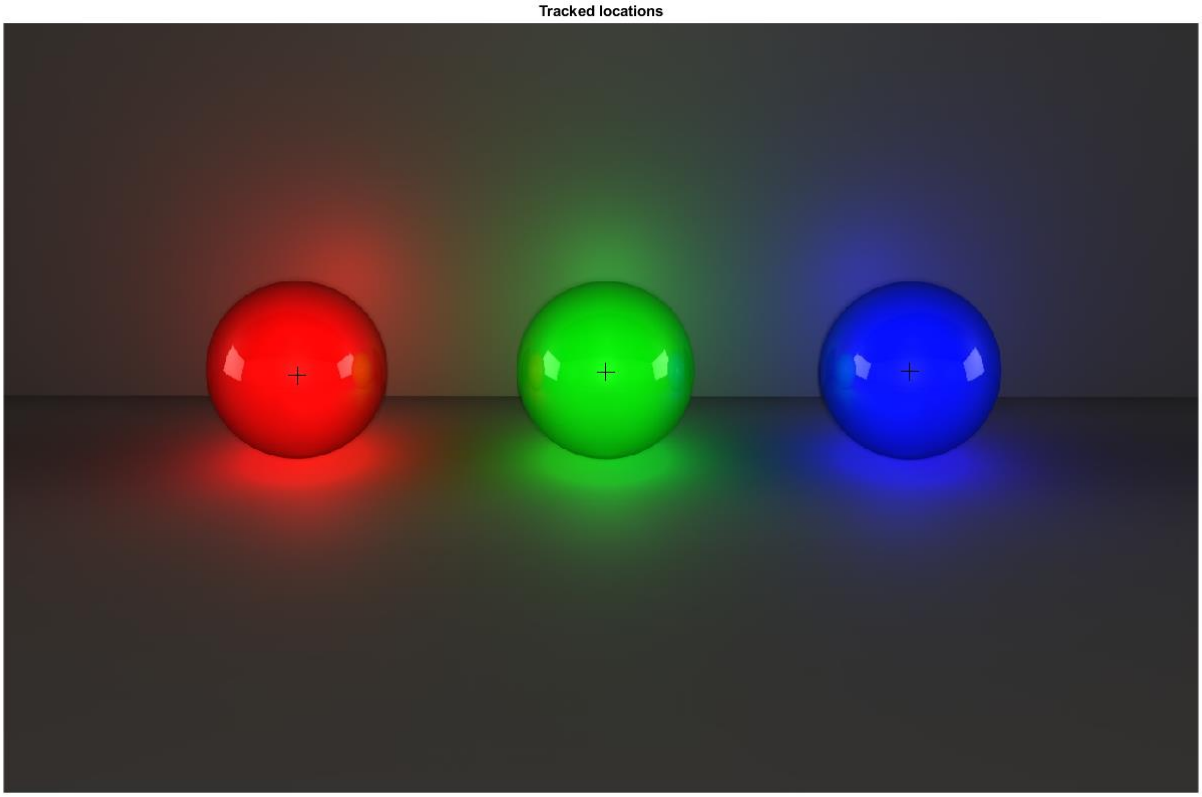


Green



Blue





color_tracker.m

```
% tabula rasa:
clc; clear all; close all;

%% This is to solve Prelab Q1: %%

% Read the image given to you (use function imread())
img = imread('image1.jpg');

% plot the original image (use imshow())
figure(1)
imshow(img);
title('Original image');

% convert your image into hsv color space (use function rgb2hsv())
HSV = rgb2hsv(img);

% plot the grayscale images of hue, saturation and value of your image seperately (use imshow() again)
[h s v] = imsplit(HSV);
figure(2)
imH = imshow(h);
title('Hue');
hp = impixelinfo(imH);  % used to find thr right treshhold values on the plot
set(hp,'Units','Normalized','Position',[0.08 0.9 0.15 0.02]);
figure(3)
imS = imshow(s);
title('Saturation');
sp = impixelinfo(imS);  % used to find thr right treshhold values on the plot
set(sp,'Units','Normalized','Position',[0.08 0.9 0.15 0.02]);
figure(4)
imV = imshow(v);
title('Value');
```

```
    % use the hue image you just plotted to find the hue lower and upper bounds for each color
    rh_t = [0.03    0.99];
    gh_t = [0.25    0.5];
    bh_t = [0.61    0.67];

    % use the saturation image you just plotted and find one single lower and upper bound for all your colors
    s_t = [0.89 1];

    % use these tresholds to create a mask for each color, plot your three masks seperately (for each
    % color you should have a black-white image showing only the blob of that color)
    hm_r = h <= rh_t(1) | h >= rh_t(2);
    hm_g = h <= gh_t(2) & h >= gh_t(1);
    hm_b = h <= bh_t(2) & h >= bh_t(1);
    sm = s <= s_t(2) & s >= s_t(1);
    mR = hm_r & sm;
    mG = hm_g & sm;
    mB = hm_b & sm;
    figure()
    imshow(mR);
    title('Red');
    figure()
    imshow(mG);
    title('Green');
    figure()
    imshow(mB);
    title('Blue');

    % find the centroid of the three colors using their respective masks ( use function regionprops();
    % be aware that it can return more than one centroid  )
    propsR = regionprops(mR,'Centroid');
    cR = propsR.Centroid;
    propsG = regionprops(mG,'Centroid');
    cG = propsG.Centroid;
    propsB = regionprops(mB,'Centroid');
    cB = propsB.Centroid;

    % plot the original image with the center of the centroid (use function insertMarker())
    figure()
    imgR = insertMarker(img,cR,'+','color','black','size',10);
    imgRG = insertMarker(imgR,cG,'+','color','black','size',10);
    imgRGB = insertMarker(imgRG,cB,'+','color','black','size',10);
    imshow(imgRGB);
    title('Tracked locations');
```

Q2

```
int EdgeDetect (IplImage* img, int thresh)
{   // ********** Prelab Q2 ***********
    // note: you can find the function definitions online under https://docs.opencv.org/2.4/index.html

    // Create a new image for converting the  original image to gray image. Use cvCreateImage() and assign
    // it to variable called "gray_img" of type IplImage*. Use cvGetSize(img) within cvCreateImage() to get the right size
    // If you are not sure, you can refer to the example in the lab manual.
    IplImage *gray_img, *smooth_gray_img, *edge_detect, *edge_img;
    gray_img = cvCreateImage(cvGetSize(img), 8, 1);

    // Convert your image "img" to gray (store it in "gray_img") with the function cvCvtColor(), using the parameters called cv::COLOR_BGR2GRAY
    cvCvtColor(img,gray_img,CV_RGB2GRAY,1);

    // Smooth the gray image by using "cvSmooth" function and using gaussian method (CV_GAUSSIAN).
    // Make sure to create a new image called "smooth_gray_img" first (cvCreateImage) where you can store the smoothed image
    smooth_gray_img = cvCreateImage(cvGetSize(img), 8, 1);
    cvSmooth(gray_img,smooth_gray_img,CV_GAUSSIAN);

    // Create another new image called "edge_detect" which we will use for edge detection from the converted gray image
    edge_detect = cvCreateImage(cvGetSize(img), 8, 1);

    // Detect edges using canny edge detection by using "cvCanny" function with the parameter of thresh to define the range. Use "thresh" for the
    // first threshold for the hysteresis procedure and "thresh*2" for the second threshold for the hysteresis procedure and an aperture size of 3.
    // refer to https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=canny#cv.Canny
    cvCanny(smooth_gray_img,edge_detect,thresh,thresh*2,3);

    // Create variables to store contours (already done)
    CvMemStorage *mem;
    mem = cvCreateMemStorage(0);
    CvSeq *contours = 0;

    // Find contours in the canny output using the cvFindContours() function
    cvFindContours(edge_detect, mem, contours);

    // Create a new image called "edge_img" for storing edge tracking image from gray image. Use 3 channels.
    // you can use cvSet to set the whole image to a specific color (background)
    edge_img = cvCreateImage(cvGetSize(img), 8, 3);
    cvSet(edge_img, (0, 0, 0));

    // define the color of the contour using cvScalar (make sure it's consistent with the number of channels)
    cvScalar cont_color(255, 255, 255);

    //define the color of contours using cvScalar()
    // | same as previous point | //

    while (contours != 0)
    {
        // draw  contours by using the cvDrawContours() function
        // set maxLevel = 0
        cvDrawContours(edge_img, contours, cont_color, cont_color, 0, 1, 8);

        // pointer to the next sequence
        contours = contours->h_next;
    }

    // Display images by using cvShowImage() function
    cvNamedWindow("image", CV_WINDOW_AUTOSIZE);      cvShowImage("image",img);
    cvNamedWindow("gray", CV_WINDOW_AUTOSIZE);       cvShowImage("image",gray_img);
    cvNamedWindow("edge", CV_WINDOW_AUTOSIZE);       cvShowImage("edge",edge_img);


    // Save Images (already done, just uncomment)

    if (frcounter%30 == 0)
    {
        char filename[50];
        sprintf(filename,"Contour_frame%d.jpg",frcounter);
        SvImage(edge_img,filename);
    }



    //release the used images by using cvReleaseImage() function. Pass the address of your image pointers to cvReleaseImage
    cvReleaseImage(&img);
    cvReleaseImage(&gray_img);
    cvReleaseImage(&smooth_gray_img);
    cvReleaseImage(&edge_detect);
    cvReleaseImage(&edge_img);

    return 0;
}
```