Exercise set: 3 **Student:** Tommaso Carraro **Student number:** 015147592

Exercise set 3 Computer Vision

Question 3.1

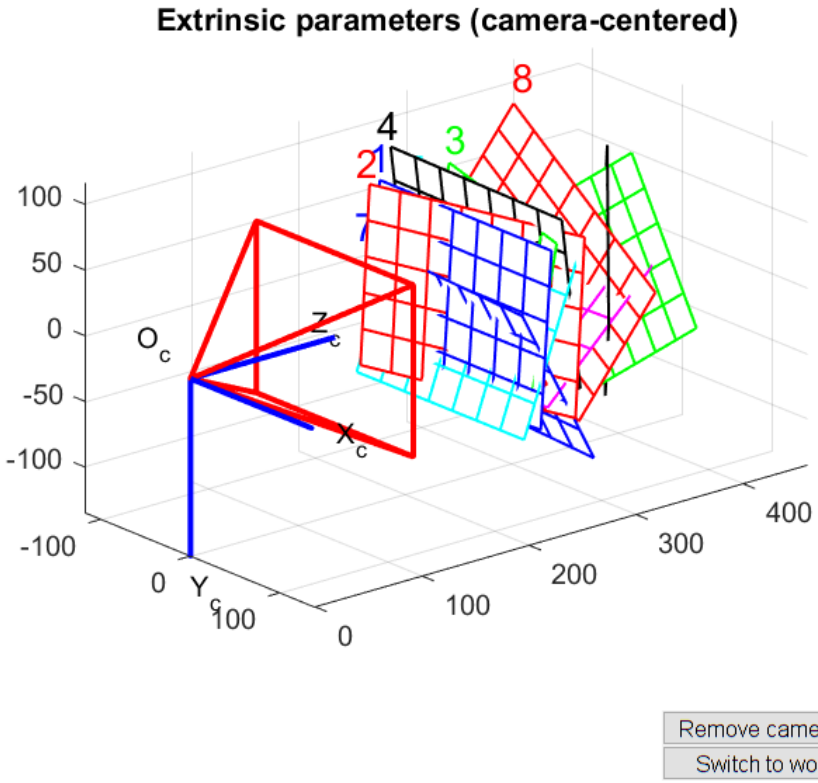
After following the instructions for doing the calibration of the camera I obtained the following calibration parameters:

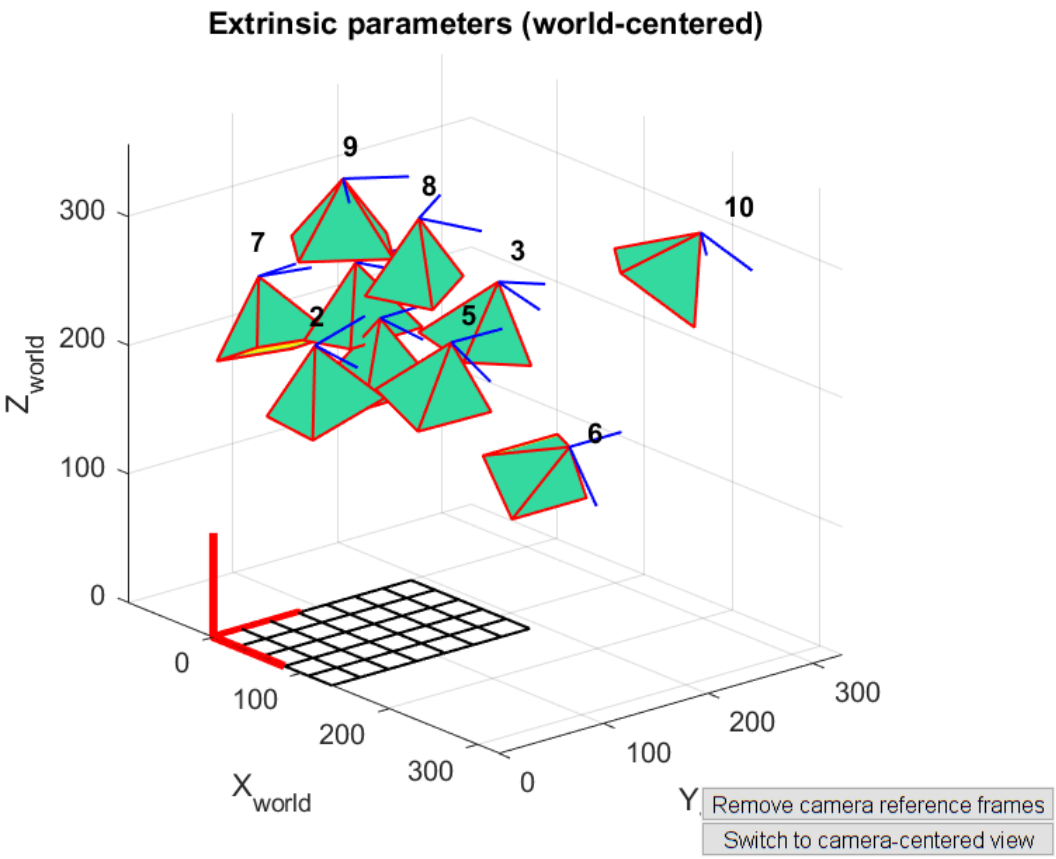
1. Focal length: [3560.26781 3568.47068] +/- [27.21252 27.44694];
2. Principal point: [2310.49774 1692.41203] +/- [25.93407 21.74466];
3. Skew: [0.00000] +/- [0.00000];
4. Distortion: [0.00542 -0.13376 0.00214 0.00120 0.00000] +/- [0.02285 0.08811 0.00179 0.00222 0.00000];
5. Pixel error: [2.27696 2.21013].

With these parameters it is possible to compute the K matrix of my camera:

$$K = \begin{bmatrix} 3560,27 & 0 & 2310,5 \\ 0 & 3568,47 & 1692,41 \\ 0 & 0 & 1 \end{bmatrix}.$$

In the two following images it is possible to see other interesting results (extrinsic parameters) computed by the tool.





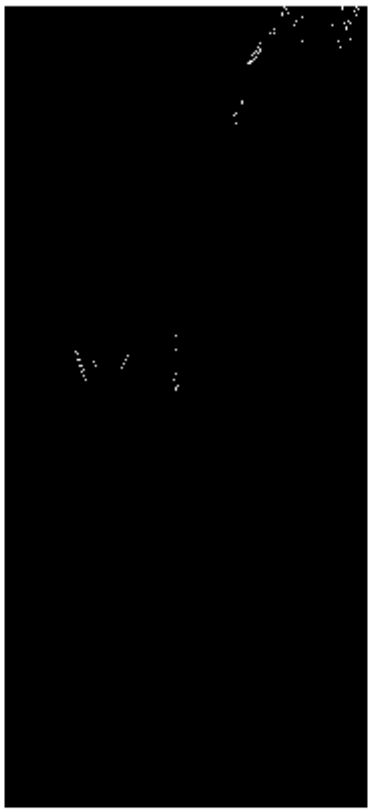
Question 3.2.1

This is my MATLAB implementation of the Hough transform.

```
img = rgb2gray(imread('Corridor1.jpg'));
edges1 = edge(img,'canny');
imshow(edges1);
[x_max, y_max] = size(edges1);
rho_range = floor(hypot(x_max, y_max));
hough_space = zeros(rho_range, 181);
% compute hough transform
for x=1:x_max
    for y=1:y_max
        if edges1(x, y) ~= 0
            for theta=1:181
                rho = x*cosd(theta) + y*sind(theta);
                if abs(floor(rho)) ~= 0
                    hough_space(abs(floor(rho)), theta) =
                    hough_space(abs(floor(rho)), theta) + 1;
                end
            end
        end
    end
end
```

In the following image I show the non thresholded hough space for the first image provided. It is possible to see that there are many sinusoids cause we have many edge points in the processed image.

If we put a threshold on the values put inside the accumulator we obtain the maximums that correspond to the straight lines we want to detect in the image. In this image I show the thresholded hough space for the first image provided.

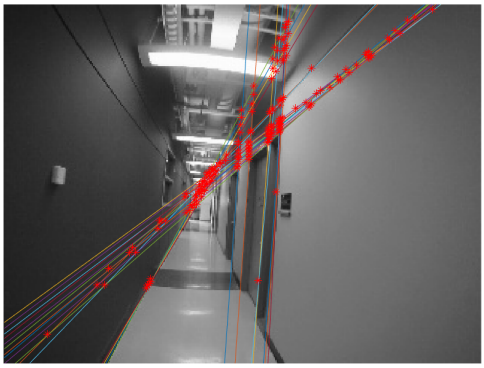


After performing the hough transform in the two images it is possible to show the detected lines. Since the lines in the images are really too much I will show only few lines as an example. In this image I show only few lines detected after performing the hough transform on the first image.

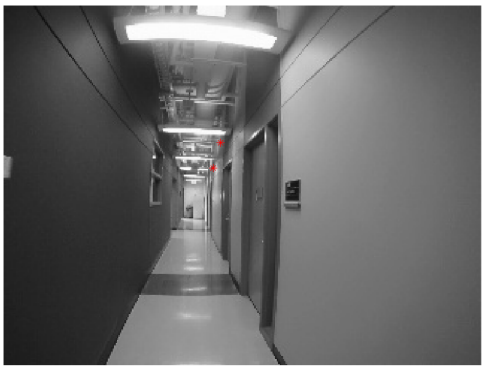
Question 3.2.2

To compute the vanishing points on the provided images we have firstly to compute all the intersections between the lines we have found in these images. In the following image I show the intersection points found by my algorithm in the first image provided. You have to take into account that in the example image I don't show the intersection points of all the possible lines

but just of the few lines of the previous image. This because if I plot all the possible intersection points these will cover the entire image.

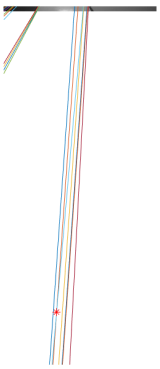


After computing the all possible intersection points it is possible to find the vanishing points. These points correspond to the intersection points where the majority of the straight lines intersect. To perform this computation I used the procedure explained in the source attached to this week exercises. In the following two images I show the central vanishing points detected by my algorithm for the two images provided.



I've chosen the one of the two that's nearest to the center of the image as central vanishing point. In the images there are two points because these points have the same maximum number of intersections.

An important thing to take into account is that the vertical direction vanishing points fall outside of the image. In the following images I show the vertical direction vanishing points detected by my algorithm in the images provided.



The coordinates of the vanishing points detected are:

1. Central vanishing point in Corridor1.jpg: $(141, 134)$;
2. Central vanishing point in Corridor2.jpg: $(141, 109)$;
3. Vertical direction vanishing point in Corridor1.jpg: $(111, 870)$;
4. Vertical direction vanishing point in Corridor2.jpg: $(157, 634)$.

From the images displayed it is possible to see that the central vanishing point isn't detected by my algorithm in the right position, in fact it is a little bit moved to the right of the image. This is a problem related to the images provided, in fact they present a radial distortion due to the camera lens of the GOPRO. Every straight line becomes a curve in the image plane due to this distortion, so when we perform the hough transform it is really difficult to find only one line per edge. It seems that my algorithm finds all the lines that are tangent to the distorted edge (it has become a curve due to distortion). It is possible to see that in the images I showed about the lines detected by my algorithm. Other problems could be the noise in the images and the small resolution of the images. It is possible to reduce the noise trying different parameters for the Gaussian kernel of the Canny edge detector.

The code that implements the vanishing points detection is provided below.

```
% finding and computing all intersections of detected lines
pointsx = [];
pointsy = [];
incr = [];
cont = zeros(x_max, y_max);
for rho=1:rho_range
    for theta=1:91
        for rho2=1:rho_range
            for theta2=1:91
                if hough_space(rho, theta) > 75 &&
                hough_space(rho2, theta2) > 75 &&
                rho ~= rho2 && theta ~= theta2
                    % building two straight lines
                    x = 1:0.1:x_max;
                    y1 = -x * (sind(theta)/cosd(theta)) + (rho/cosd(theta));
                    y2 = -x * (sind(theta2)/cosd(theta2)) + (rho2/cosd(theta2));
                    x11 = 1;
                    x12 = 2;
                    x21 = 3;
                    x22 = 4;
                    y11 = -x11 * (sind(theta)/cosd(theta)) + (rho/cosd(theta));
                    y12 = -x12 * (sind(theta)/cosd(theta)) + (rho/cosd(theta));
                    y21 = -x21 * (sind(theta2)/cosd(theta2)) +
                    (rho2/cosd(theta2));
                    y22 = -x22 * (sind(theta2)/cosd(theta2)) +
                    (rho2/cosd(theta2));
                    point = linlinintersect([x11 y11; x12 y12; x21 y21; x22
y22]);
                    found = false;
                    [s1, s2] = size(pointsx);
                    for ix=1:s2
                        if floor(pointsx(ix)) == floor(point(1)) &&
                        floor(pointsy(ix)) == floor(point(2))
                            % number of straigh lines that intersect in
                            % the same point
                            incr(ix) = incr(ix) + 1;
                            found = true;
                        end
                    end
                    if found == false
                        pointsx(end+1) = point(1);
                        pointsy(end+1) = point(2);
                        incr(end+1) = 0;
                    end
                    % finding coordinates of intersection points outside of
                    % the image
                    if abs(floor(point(1))) > x_max ||
                    abs(floor(point(2))) > y_max
                        disp(floor(point(1)));
                        disp(floor(point(2)));
                    end
                    plot(floor(point(1)), floor(point(2)), 'r*');
                end
            end
        end
```

```
    end
end

% finding coordinates of the vanishing points on the image
[s1, s2] = size(incr);
for i=1:s2
    % show only points with a minimum number of intersections
    if incr(i) > 54 % threshold to show only the desidered point
        plot(floor(pointsx(i)), floor(pointsy(i)), 'r*');
    end
end

% function that compute the intersection between two given lines
function point = linlinintersect(lines)
    x = lines(:,1);
    y = lines(:,2);
    % Calculation
    denominator = (x(1)-x(2))*(y(3)-y(4))-(y(1)-y(2))*(x(3)-x(4));
    point = [((x(1)*y(2)-y(1)*x(2))*(x(3)-x(4))-(x(1)-x(2))*(x(3)*y(4)-
y(3)*x(4)))/denominator ...
        ,((x(1)*y(2)-y(1)*x(2))*(y(3)-y(4))-(y(1)-y(2))*(x(3)*y(4)-
y(3)*x(4)))/denominator];
end
```

Question 3.2.3

To compute how much the camera has rotated between the two images I've used the formulas presented in the source attached to this week exercises. I've computed only the heading and the pitch of the camera on the two images using the central vanishing points detected by my algorithm. To compute the roll of the camera it is possible to use the vertical direction vanishing points together with the central one. The formulas to compute the heading and the pitch of the camera given the focal length (fx, fy), the principal point (u, v) and the central vanishing point (x, y) are:

1. $\theta = arcsin((x - u)/fx)$;
2. $\varphi = arcsin((y - v)/(-fy * cos(\theta)))$.

The results I've obtained are shown below:

1. Heading of the camera in the first image: -1.1068;
2. Heading of the camera in the second image: -1.1068;
3. Pitch of the camera in the first image: 0.2820;
4. Pitch of the camera in the second image: 0.3020.

It is possible to observe that the camera hasn't changed in the heading angle but it has changed a little bit in the pitch angle and it is possible to see that also from the images.