Competition Statement - Stereovision

Here we are ! the robot Pioneer p3dx is at your disposal, equipped with two cameras which constitute its stereovision system. Load the scene that is in ***(Tools/ Parcours_robot_avec_simulateur / cuisine_Explore.ttt)*** to discover your robot and Romain's house.

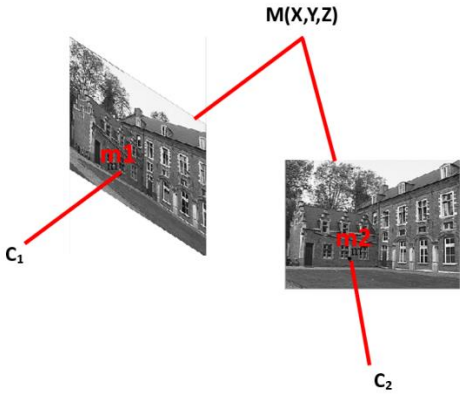

Figure 01: Stereovision principle. The 3D point M(X, Y, Z) of the scene has two 2D projections: m_1, m_2 in the left and right images respectively.

STEP 1 – CALIBRATION (1 point)

To allow 3D reconstruction, we need to calibrate our stereovision system. The calibration consist in calculating the intrinsic parameters of each camera, as well as the relative position of the right camera to the left camera (extrinsic parameters). Let's put that into equations!

M_L = Left camera calibration matrix.

M_R = Right camera calibration matrix.

$$M_L = K_L\,[R_L, t_L]$$

$$M_L = \begin{bmatrix} \alpha_u & 0 & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Intrinsic Extrinsic

$$M_D = K_D\ [R_D, t_D]$$

$$M_D = \begin{bmatrix} \alpha_u & 0 & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Intrinsic Extrinsic

Note: You will notice that the extrinsic parameters of left camera are R = Identity matrix, and t = $[0, 0, 0]^t$. This is explained by the fact that the left camera is taken as center of world coordinates system for 3D reconstruction. The coordinates framed by red rectangle are the homogeneous coordinates, used for matrix calculations. You do not need to create them, opencv functions will automatically include them later.

Your objective: create the calibration matrices in your code so that you can use them later. The matrices to forme are: **K_D, K_G, [R_D, t_D].** Indications are hidden in Romain's house to recover the values of the matrices. You can either, manipulate the scene with your mouse and search for their location. Otherwise, use your robot that knows where to find them by running the simulation. You have two simulations: cuisine_Explore.ttt et salon_Explore.ttt. Launch each simulation, then watch what is displayed on the two windows that represent the robot's eyes (its two cameras). Do not hesitate to pause in the simulation, then zoom in on what seems to be an indication.

STEP 2 – 3D Reconstruction

Now let's start the most interesting part of stereovision. We are going to implement a dense reconstruction method which allows us to find the 3D model of a scene from only two images of this scene. Here, are the 3D reconstruction steps:

- Rectification of the two stereo images in order to make the epipolar lines parallel.
- Points matching, which consists in finding in the left and right images the homologous points, i.e., the points which are the projections of the same entity of the scene.
- Triangulation of points, which is the calculation of the 3D coordinates of each point using the 2D coordinates of its two projections on the left and right image.

1. **Rectification of images (1 point)**

The rectification of a pair of stereoscopic images makes it possible to be reduced to a simple epipolar geometry, where the epipolar lines are parallel to the lines of images. The purpose behind this change is to facilitate matching. Indeed, each point belonging to the image on the right will find its counterpart in the image on the left on the same line of pixels. We will use this constraint to drastically reduce the computation time of the mapping.

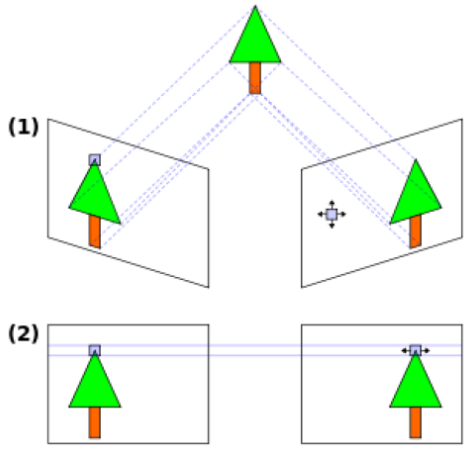

Figure 02: Illustration of stereoscopic images: (1) before rectification (2) after rectification.

You have at your disposal a code to read two stereo images, and to display them. **Your objective** is to rectify these images using the functions provided by opencv: stereoRectify(), initUndistortRectifyMap(), remap(). Then to display the rectified images.

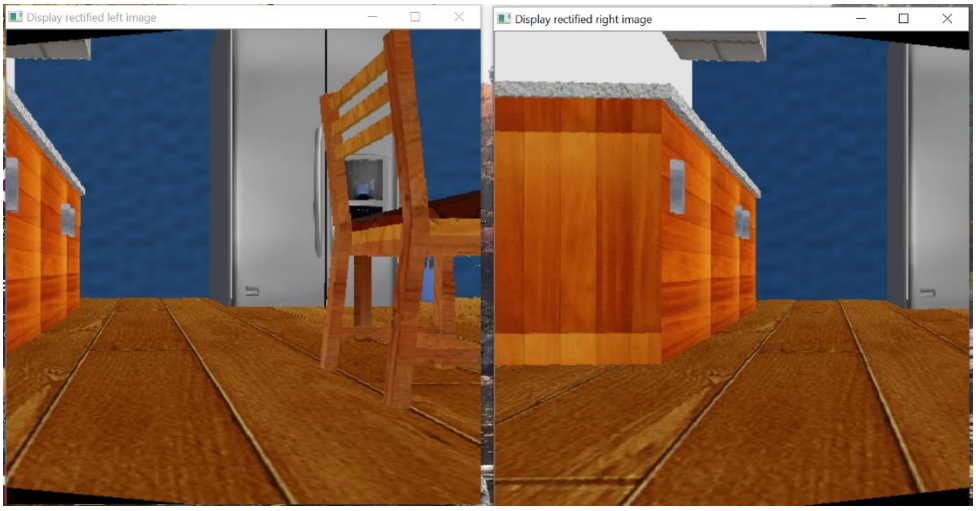

Figure 03: Expected result after rectification.

2. Matching (2 points)

The matching is the most critical step in our approach. Indeed, we must find for each pixel in the left image its homologue in the right image. Knowing that some pixels have very similar intensities, not to mention the number of pixels an image can have (example 1024x1024). It is certain that constraints must be used in order to reduce the field of research. The basic method we propose is a simple correlation.

- ***How correlation works***

We use a fixed correlation window F1 in the left image centered on the point to be paired, as well as a sliding window F2 in the right image which will browse the search area. For each position of F2, a correlation score C (x, y) is calculated between F1 and F2. Examples of correlation measurement: the sum of the quadratic SSD deviations, the sum of the absolute SAD deviations, the centered and normalized cross correlation ZNCC, etc. After calculating for each pair of pixels its likeness score. We reduce the number of candidates through a threshold. Then, we select the best.

Your objective is to implement the correlation matching method using opencv function: matchTemplate(). But be careful, don't forget to introduce the epipolar constraint or your code will take more than 20 minutes to execute! Another important point, matchTemplate () function can be applied only to grayscale images, don't forget to transform your color images into grayscale images.

3. Disparity map (2 points)

The disparity map displays the displacement that a single pixel will have from the left image to the right image, assuming that the two images are superimposed. We will need the pairs of matched pixels obtained in the correlation step. Since the homologous pixels are found on the same line thanks to the rectification of the images. The disparity is calculated as follows:

$$\text{disparity}(x, y) = y\_left - y\_right$$

In order to enhance the disparity image contrast, here is the formula to make uniform the distribution of intensities in the gray level range [0 - 255] :

$$\text{ImgOut}(x,y) = 255 \left(\frac{\text{ImgIn}(x,y) - c}{d - c} \right)$$

With : c and d = min and max values of the image.

Your objective is to generate the disparity map and display it on screen. This map already allows you to observe the depth of the scene. The closer the object is to camera, the brighter its intensity will be, and conversely, the objects at the back of the scene will appear darker.

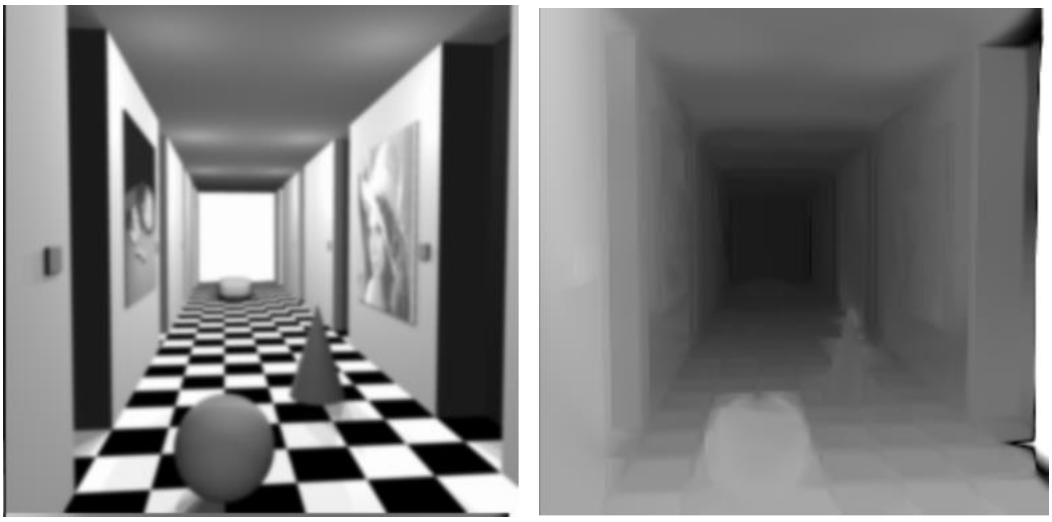

Figure 04 : To the left original image, to the right disparity map.

4. Triangulation (1 point)

Triangulation calculates the 3D coordinates of each pixel. Use the function: triangulatePoints(). The function takes as inputs: the two rectified projection matrices as well as the matrices of the matched points.