**Computer Vision & Image Processing M Report of the final project**

***“Robot Navigation Project”***

***A.A. 2019/2020***

***Castellucci Davide (Artificial Intelligence)***

***Stefani Gianluca (Automation Engineering)***

The project we developed deals with the use the stereo vision technology in a robotic application for obstacle avoidance purposes. The starting point is given by 2 synchronized videos representing the environment seen by a mobile robot during the motion inside a room filled by obstacles, particularly, a chessboard useful to get information about the quality of the computer vision procedures we ran.

Moreover, we are given the parameters of the employed stereo camera mounted on the robot:

* Baseline **b**: 92,226 mm
* Focal length **f**: 567,2 pixels

The main goals of the algorithm we designed are:

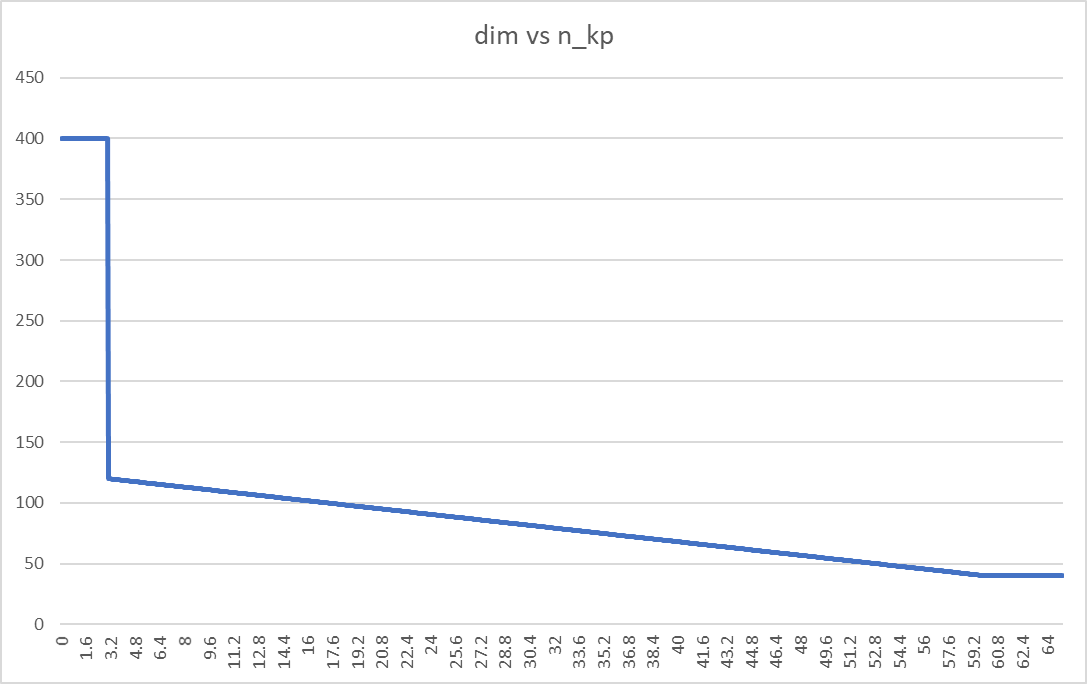
1. Compute the disparity map in a central area of the reference frame (e.g. a squared area of size 60x60, 80x80 o 100x100 pixels), so to sense distances in the portion of the environment which would be travelled by the vehicle by keeping a straight trajectory.

1. Estimate a *main disparity* (**dmain**) for the frontal (wrt the camera) portion of the environment based on the disparity map of the central area of the reference frame computed in the previous step, e.g. by choosing the average disparity or the most frequent disparity within the map.
2. Determine the distance (*z*, in *mm*) of the obstacle wrt to the moving vehicle based on the *main disparities* (in pixel) estimated from each pair of frames:
3. Generate a suitable output to convey to the user, in each pair of frames, the information related to the distance (converted in meters) from the camera to the obstacle. Moreover, an alarm should be generated whenever the distance turns out below 0.8 meters.
4. Compute the real dimensions in *mm* (**W, H**) of the chessboard pattern present in the scene.

**Our project proposal considers the following steps:**

* The analysed window (ROI) dimensions are modified in a proportional fashion wrt to the number of found key-points in order to counteract the decreasing of the number of the latter. The number of found key-points is mapped in the size of the window with a linear function plus saturation. The corresponding law is as follows:

Where m and q are the parameters of a line:



The actual value of dimension given to the algorithm for the next step is computed over a five-step window in order to filter out high frequency variations of the number of found key-points and keep only significative variations which need to be compensated. The saturation value of 400 for the range n\_kp ϵ [0, 3] is used to rapidly increase the dimensions of the searching window whenever the number of found key-points tends to zero. Thus, we avoid having large periods of time with an empty disparity map.

* Each couple of frames is subjected to three image processing algorithms in order to improve their quality. In particular, the following functions are used:
* *“cv2.cvtColor()”:* to convert the frame into a gray-scale one
* *“cv2.equalizeHist()”:* to flatten the histogram of each frame
* *“cv2.medianBlur()”:* to apply a median filter to each frame with a window of 5, the one giving the highest number of found key-points.
* The key-points are indeed found in each couple of frames by using the SIFT algorithm (function: “*cv2.sift.detectAndCompute()”*) which returns a set of key-points and their corresponding “*descriptors*”.
* The ROI is divided into **n** horizontal stripes of a given height, in our case 2, and **n** is obtained by: . Since we deal with a standard stereo geometry, we assume that matching key-points have, approximately the same y coordinate.
* Afterwards, the Euclidian distance in the descriptors space is computed considering points belonging to corresponding stripes (i.e. with the same range of pixel in the vertical direction) by using “*numpy methods”* (*np.sqrt(), np.sum(), np.newaxis()*)*.*
* Couples of points having distance lower than a certain threshold are signed as “matching” and stored in a structure divided by stripes.
* If 2 or more points of a side frame corresponds to a single point of the other side frame, then, the lowest distance one is maintained while the others are removed.
* The disparity map is filled by computing the disparity of matching key-points.
* Key-points which are out of the considered disparity range or have negative values are considered as outliers. We, selected as disparity range the following interval:

Because we want it to be centred in dmain and larger when we are closer to the obstacle.

* The standard deviation of the disparity map is computed in order to understand whether the disparity values are spread or not. If the standard deviation is sufficiently high (wrt a certain threshold), it is reasonable to assume that disparities belong to different objects. Thus, in this case we compute 2 different distances: “*dist\_ob”* and “*dist\_bg”*. As before, the threshold is increased as the robot get closer to the obstacle since the disparity range is larger (0,5/dist\_ob, which is inversely proportional to the distance).
* However, it is possible to expect false positive matching which lead to big discrepancies in the estimation of the current distance, so, variations of the latter greater than a certain value are filtered out.
* Furthermore, a low pass filter (FIR filter) is applied to the 2 obtained measures in order to reduce the ripple. Coefficients of the filter are computed by using a Matlab script by choosing a proper cut-off frequency. The best trade-off between a “good” quality of the measure and the increased delay is pursued.
* The 2 functions “*cv2.findChessBoardCorners()”* and *“cv2.drawChessBoardCorners()”* are embedded in a procedure that allow to find internal corners of the chessboard glued on the obstacle, draw them and estimate the dimensions W and H to obtain a quantitative measure of the performances of the algorithm in terms of percentage error wrt actual values.

Considering the actual implementation of the computer vision algorithm, it has been decided to define 2 classes: “*Robot*” and “*MyFilter*”.

The first class *“Robot”* is initialized with the focal length, the baseline and the height of the stripes, considering these parameters as the parameters of the stereo device mounted in the mobile robot.

This class has the following methods:

* *“video\_reading()”:* used to load the current synchronized frame couple at each step;
* *“image\_initialization()”:* which contains all the image processing procedures applied to the frames, in order to improve the quality (as described above);
* *“keypoints\_division()”: this method get the output of “sift.detectAndCompute()”* (the set of found key-points), and sort it by stripes; this is done by generating 2 structures, one for key-points coordinates and one for corresponding descriptors;
* *“disparity\_map\_calculation()”:* used to compute the distance (in the descriptor space) among all the possible combinations of key-points belonging to corresponding stripes and inserting matchings into the disparity maps (obstacle and background), excluding those considered as outliers wrt rules described above;
* *“distance\_calculation()”:* disparity maps are analysed in terms of dimensions, standard deviations and disparity ranges; then, the main disparities are computed (“*d\_main\_ob”* and *“d\_main\_bg”*) and, subsequently, the distances are obtained by applying the following relation:
* “*write\_on\_image()*”*:* the obtained information are collected and reported on the window; this function takes care of reporting to the user if dangerous conditions are reached (in this case when the distance is below 0.8m); the plotted data are computed distances, disparities, estimation of W and H and their estimation error;
* “*square\_dimension()*”*:* used to adapt, at each step, the size of the ROI by using the mapping law described above.

The second class is *“MyFilter”*, initialized with the frame rate of the video, which represents the sampling frequency of our measures, the cut-off frequency, initial condition, the order of the filter and the coefficients computed in Matlab. It has only one method but, a future development could be the implementation of new type of filters:

* *“filtering”:* this method contains functions which filter data on a sliding window containing data of past samples plus the current one.

In the main loop, all these functions are called according to the presented algorithm and, at the end of the video, by pressing ‘Q’, results of the measurement are reported in a plot distance vs frames either in filtered and not filtered shape. Moreover, error plots are shown. There is the possibility to stop the simulation at any instant (again by pressing ‘Q’) or to pause it (by pressing ‘A’). After issuing the pause command, a zoom of the currently analysed frame is returned. To resume the simulation, it is enough to press “esc”.

Now, we can see some results of the simulation.

***Immagine che contiene testo, mappa

Descrizione generata automaticamentePlot of the estimated distances***

***Immagine che contiene testo, mappa

Descrizione generata automaticamenteRange of estimation error for W and H***

It is possible to see that for the most part of the simulation, the estimation error is below the 10%. When we are closer to the obstacle the error is even lower, below 5%.

We observe, in the second part of the simulation, when we proceed towards the obstacle again, that the distribution of the error is more spread. This fact can be explained considering that the second trajectory of the robot is less centred wrt the obstacle, therefore the hypothesis we made about being perpendicular to the latter is less accurate.

***Further development***

It can be noticed, by looking at the error plot, that the random distribution of the error function has a positive mean value. This fact is quite counterintuitive since, the estimated dimensions of the chessboard should have been lower than the original one because of the relative rotation (about the vertical axis) between the chessboard itself and the camera reference frame, and the vertical offset between the optical axis and the chessboard centre. The perspective tells us that the apparent lengths we see in the image should be lower than the one we would get from a chessboard parallel to the image plane. Therefore, we assume that the mean value of the error should be negative and there must exists a systematic error in the estimation. As a solution to this problem, we suppose having a parametric error in the model we adopted, and we proceed by compensating it through a constant multiplication factor.

Moreover, we assume that the estimation of the coordinates of the inner chessboard corners returned from the function “*cv2.findChessBoardCorners()”*, is unbiased, so the resulting random distributions have a null mean.

The computation of the constant scaling factor we introduced is based on the mean of the error distributions without that correction. The mean values are:

* 5,6 % for the width W
* 6,4 % for the height H.

So, the constant factor is set to 0.065 and it is multiplied to distance computation formula as follows:

Immagine che contiene mappa, testo

Descrizione generata automaticamenteThe obtained distance plot and error distribution are the following:

Immagine che contiene testo, mappa

Descrizione generata automaticamente