Alma Mater Studiorum · University of Bologna

department of computer science and engineering

Degree Programme - Artificial Intelligence

Stereo Robot Navigation

Class Project for
Image Processing and Computer Vision

Presented by:
Daniela Baraccani
Mattia Gagliardi
Asif Khan

Academic Year 2021 - 2022

Overview

This paper describes the realization of the Stereo Robot Navigation project for the class of Image Processing and Computer Vision for UNIBO's Artificial Intelligence's degree programme.
The aim of the project is to simulate the movement and behaviour of a robot that must dodge obstacles, by recognizing their distance.
We are given a video taken by a stereo camera mounted on a moving vehicle, and the system must output an 'Alarm' in case the distance from the object, in this video's case a chessboard, is lower than a given threashold, which is in this case 0.8m.
In the next chapters we will narrate in detail our approach to solving the problem and satisfying the given requirements.
We start by illustrating which are the necessary python libraries that we used, the process of reading the frames, rectifying the images in order to be able to compare the left and right views, calculating the disparity map and the main disparity in order to get the distance.
Finally we also calculate the real dimentions of the chessboard by using OpenCV functions to calculate its corners.

Contents

List of Figures

Chapter 1

Project Requirements

This chapter is the direct transcription of the requirements specified by the professor for this project.

1.1 Objective

Given a video sequence taken by a stereo camera mounted on a moving vehicle, project's objective is to sense information concerning the space in front of the vehicle which may be deployed by the vehicle navigation system to automatically avoid obstacles.

1.2 Data Set

The input data consist of a pair of synchronized videos taken by a stereo camera (robotL.avi, robotR.avi), with one video concerning the left view (robotL.avi), the other the right view (robotR.avi). Moreover, the parameters required to estimate distances from stereo images are provided below:

- focale f = 567.2 pixel
- baseline b = 92.226 mm

1.3 Functional Specifications

Sensing of 3D information related to the obstacles in front of the vehicle should rely on the stereo vision principle. Purposely, students

should develop an area-based stereo matching algorithm capable of producing a dense disparity map for each pair of synchronized frames and based on the SAD (Sum of Absolute Differences) dissimilarity measure.
For each pair of candidate corresponding points, the basic stereo matching algorithm consists in comparing the intensities belonging to two squared windows centred at the points. Such a comparison involves computation of either a dissimilarity (e.g. SAD, SSD) or similarity (e.g. NCC, ZNCC) measure between the two windows. As the matching process is carried out on rectified images, once a reference image is chosen (e.g. the left view), the candidates associated with a given point need to be sought for along the same row in the other image (right view) only and, usually, within a certain disparity range which depends on the depth range one wishes to sense. Accordingly, given a point in the reference image, the corresponding one in the other image is selected as the candidate minimizing (maximizing) the chosen dissimilarity (similarity) measure between the windows. As such, the parameters of the basic stereo matching algorithm consist in the size of the window and the disparity range. In this project, students should choose the former properly, while the latter is fixed to the interval [0, 128].

The main task of the project requires the following steps:

1. Computing 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 should it keep a straight trajectory.

2. 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.

3. 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:
   z(mm) = (b(mm) * f(pixel)) / dmain(pixel)

4. Generate a suitable output to convey to the user, in each pair of frame, 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.

5. Compute the real dimensions in mm (W,H) of the chessboard pattern present in the scene. Purposely, the OpenCV functions cvFindChessboardCorners and cvDrawChessboardCorners may be deployed to, respectively, find and display the pixel coordinates of the internal corners of the chessboard. Then, assuming the chessboard pattern to be parallel to the image plane of the stereo sensor, the real dimensions of the pattern can be obtained from their pixel dimensions (w,h) by the following formulas:
W(mm) = (z(mm) * w(pixel)) / f(pixel)
H(mm) = (z(mm) * h(pixel)) / f(pixel)
Moreover, students should compare the estimated real dimensions to the known ones *(125 mm x 178 mm)* during the first approach manoeuvre of the vehicle to the pattern, so to verify that accuracy becomes higher as the vehicle gets closer to the pattern. Students should also comment on why accuracy turns out worse during the second approach manoeuvre.

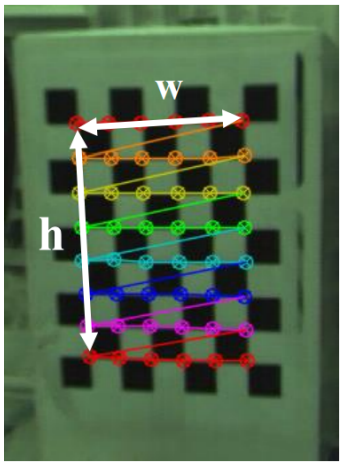


Figure 1.1: OpenCV function cvDrawChessboardCorners

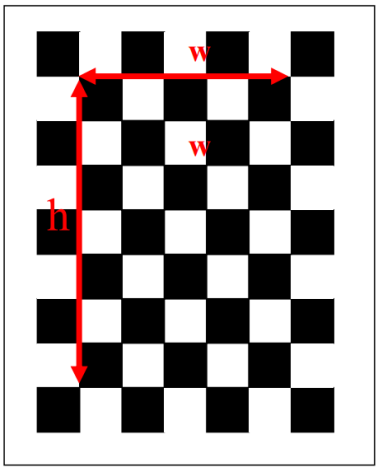


Figure 1.2: Calculate the real W and H dimensions of the chessboard

Chapter 2

The Code

In this chapter we will review in detail the development of the code. Here is the explenation of the libraries used and the main cycle whith its components.

2.1 Libraries

The python libraries needed are the main ones: OpenCv, Numpy and Matplotlib. And from IPython we imported clear_output, which we need to display the video.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Import additional library to properply play videos on jupyter notebook
from IPython.display import clear_output
```

Figure 2.1: Python Libraries imported

2.2 Inside the Reading Cycle

After loading both the 'Right' and the 'Left' stereo videos (see figure 2.2) we enter the reading cycle.
We start off by loading the right and left frames, we check if they are loaded correctly, in case something goes wrong or we reach the end of the video we break from the cycle. Otherwise the program goes on to convert the frames in grayscale, as shown in the image 2.3

After that the program gets the stereo and calculates the disparity, Once the disparity is defined, it uses the average disparity to get the distance, which determines the displaying of an 'Alarm' if the value requires it. Finally, if the distance conditions are given the program calculates the real dimensions of the chessboard.

```
#Loading Right and Left stereo videos
capR = cv2.VideoCapture("robotR.avi")
capL = cv2.VideoCapture("robotL.avi")
```

Figure 2.2: Loading videos

```
while (capR.isOpened() and capL.isOpened()):
    Alarm=False
    #Loading the frame/image from the videos
    retL, frameL = capL.read()
    retR, frameR = capR.read()
    if not retL or frameL is None:
        break
    if not retR or frameR is None:
        break
    imgR1= cv2.cvtColor(frameR, cv2.COLOR_BGR2GRAY)
    imgL1= cv2.cvtColor(frameL, cv2.COLOR_BGR2GRAY)
```

Figure 2.3: Loading frames

2.3 The Disparity Map

The image 2.4 shows the part of the cycle in which the disparity is produced.
First we use the OpenCV ***cv2.StereoSGBM_create()*** function, specifying the disparity values between 0 and 128. Then we get the disparity by calling the ***stereo.compute()*** function and we calculate the average disparity.

```
 #Matched block size. It must be an odd number >=1 . Normally, it should be somewhere in the 3..11 range.
window_size = 15
min_disp = 0
num_disp = 128 - min_disp
stereo = cv2.StereoSGBM_create(minDisparity=min_disp,
                               numDisparities=num_disp,
                               blockSize=16,
                               P1=8 * 3 * window_size ** 2,
                               P2=32 * 3 * window_size ** 2,
                               disp12MaxDiff=1,
                               uniquenessRatio=10,
                               speckleWindowSize=150,
                               speckleRange=32
                               )

# print('computing disparity...')
disparity_SGBM = stereo.compute(imgL1, imgR1).astype(np.float32) / 16.0



disparity=disparity_SGBM[disparity_SGBM.shape[0]//2-50:disparity_SGBM.shape[0]//2+50,
                         disparity_SGBM.shape[1]//2-50:disparity_SGBM.shape[1]//2+50]
```

Figure 2.4: Code specifying how we obtained the disparity map

2.4 Calculating the Distance

Inside the cycle, this part of the code explains how the program finds the current distance from the chessboard **z**.
After finding the disparity map, we reduce the image obtained to focus only on the central pixels, or the central values of disparity, since we are interested on where our 'robot' is going and we are not really interested on his peripheral vision.
Having the disparity values of interest, we are finally able to obtain **z** by dividing the product of the baseline **b** and the focale **f**, by the average disparity **avg** *(np.average(disparity))*. We then display the distance in the output video frame (see figure 2.5).
If **z** is less than 0.8m, the string 'Alarm' is also displayed in the output video frame, as shown in the figure 2.6.

```
#calculation of depth

    avg = np.average(disparity)
    print("average of disprity", avg)
    b=92.226
    f=567.2
    numerator= b*f
    z= (numerator/(avg))*0.001 #DEPTH IN METERS

    cv2.putText(frameR,'Distance '+ str(z), (10,50), cv2.FONT_HERSHEY_SIMPLEX,
                fontScale = 0.5, color = (0, 255, 0), thickness = 1)

    print("depth",z)
```

Figure 2.5: Calculating the distance

```
#Setting Alarm!!

    if(z<0.8):
        Alarm=True
    if(Alarm==True):
        cv2.putText(frameR,'Alarm!', (550,50), cv2.FONT_HERSHEY_SIMPLEX, fontScale = 0.5, color = (0, 0, 255), thickness = 2)

    cv2.imshow("video",frameR)
    cv2.waitKey(10)
```

Figure 2.6: Extract from the code, representing the part in which the program displays the Alarm, following the event of approaching the chessboard, getting closer than the 0.8m threshold

2.5 Finding the Real Dimensions

The final part of the cycle focuses on calculating the real width and heigth of the chessboard. (see figure 2.7
The program executes this part of the code only once the robot is close enough to the chessboard to 'see' it clearly (when **z¡**0.8m). Here we use the OpenCV functions *cv2.findChessboardCorners()* and *cv2.drawChessboardCorners()* to find and display the pixel coordinates of the internal corners of the chessboards.
Once found the coordinates we get the real W and H by following the formulas mentioned in the previous chapter 1.3:
$\boldsymbol{W(mm) = (\ z(mm)\ *\ w(pixel)\ )\ /\ f(pixel)}$ and
$\boldsymbol{H(mm) = (\ z(mm)\ *\ h(pixel)\ )\ /\ f(pixel)}$.

2.6 Exiting the Cycle and Releasing the Videos

When the input videos end, we exit the cycle, release de video captures and destroy all the windows the program outputed (figure 2.8.

```
#.....Chess board pattern calculation!.....

#setting the pattern size
pattern_size = (8,6)

#calculating the chessboard measurements only when the robot gets closer to the target

if z<0.8:
    z=z*1000 #converting z from meter to mm

    #Corners Detector

    found,corners = cv2.findChessboardCorners(imgL1,pattern_size)
    print('Found:',found)
    print('corners coordinates:', corners)

    #Refinment step

    if found:
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_COUNT, 30, 1.0)
        cv2.cornerSubPix(imgL1, corners, (5,5), (-1,-1), criteria)

    # visualization of the pattern

    draw = cv2.cvtColor(imgL1,cv2.COLOR_GRAY2BGR)
    cv2.drawChessboardCorners(draw,pattern_size,corners,found)
    plt.imshow(cv2.cvtColor(draw,cv2.COLOR_BGR2RGB))
    plt.show()

    #Calculating Dimension

    H = corners[7][0][1] - corners[0][0][1]
    W = corners[0][0][0]-corners[40][0][0]
    print(W,"WIDTH AND;","HEIGHT:",H)
    Wr=(z*W)/f
    Hr=(z*H)/f

    print('DIMENSION : (',Wr,',',Hr,')')
```

Figure 2.7: Extract from the code, detailing the process of finding the real dimensions of the chessboard

```
capR.release()
capL.release()
cv2.destroyAllWindows()
```

Figure 2.8: Extract from the code, where the program, after exiting the cycle when the input video reaches its end, releases the video captures and closes the windows outputed

Chapter 3

Analyzing the Output

3.1 The Disparity

The average disparity follows the expected course, growing with the decrease of the distance between the robot and the chessboard.
The outputs follow almost in a monotonic way the proceeding of the video, following the approach (respectively decreasing and growing) and the distancing (respectively growing and decreasing), which would be the best results that we can expect taking into account the possible errors in the reading of the frames.
The outputs are shown in the following images 3.1, 3.2, 3.3, 3.4.

3.2 The Distance

As seen in the prior subchapter, the values of the distance follow the course of the video in a satisfying way.
The distance values are outputed not only in the console but also in the video, and when the value is under 0.8m the string 'Alarm' is also outputed on the video frames, as shown in the figures 3.5, 3.6 and 3.7

3.3 The Chessboard's Dimensions

When in range (**z ¡ 0.8m**) the real dimensions of the chessboard are calculated, but really they are estimated, because we never truly get the real dimensions as specified in the requisites.
In the first approach, the best estimate is found at a distance of **z =**

0.7893881386245246m. Here the program gets **w = 147.32206546192387mm** and **h = 210.31861265502633mm**, as shown in the figure 3.8

In the second approach, at a slightly closer distance than the previous example, in this case with **z = 0.7779858720795291m** w and h are less accurate: **w is 161.68116981006187mm** and **h is 236.05540453566638mm**.
(See figure 3.9).

The best result is gotten in the second approach (figure 3.10), at **z = 0.5038398187382954m** . Here, although the approach is not frontal, this doesn't matter because Harris corner detection is rotation inveriant, and the w and h estimations are closer to the reality of the chessboard's dimentions, with **w = 134.24644553499647mm** and **h = 194.2283064187051mm**.

```
average of disprity 19.515043
depth 2.6805263255959173
average of disprity 19.534437
depth 2.677865080992506
average of disprity 19.552675
depth 2.6753672599104523
average of disprity 19.538393
depth 2.6773229069948155
average of disprity 19.573694
depth 2.6724943481624943
average of disprity 19.564468
depth 2.67375459296119
average of disprity 19.58275
depth 2.671258446542822
average of disprity 19.519968
depth 2.679850044426405
average of disprity 19.541368
depth 2.6769152447793014
average of disprity 19.533674
depth 2.677969672115268
average of disprity 19.513338
depth 2.680760562925785
average of disprity 19.511063
depth 2.6810732051481336
average of disprity 19.555374
depth 2.674998024111781
average of disprity 19.621038
depth 2.666045804265411
average of disprity 19.652563
depth 2.6617692026676103
average of disprity 19.7293
depth 2.6514163404495146
average of disprity 19.779543
depth 2.644681295402538
average of disprity 19.839075
depth 2.6367452598795795
```

Figure 3.1: The figure reports the values of the disparity and the distance (depth). This are the first values outputed, here the first distance is the maximum distance and it gradually decreases, while the disparity is the minimum disparity and it goes gradually increases reaching its maximum on the second apporach.

```
depth 0.9948534372015474
average of disprity 54.855038
depth 0.9536150079118527
average of disprity 55.963776
depth 0.9347222664423628
average of disprity 56.876724
depth 0.919718705605433
average of disprity 59.630455
depth 0.8772461519035534
average of disprity 61.081757
depth 0.8564027971491767
average of disprity 60.66037
depth 0.8623519327277146
average of disprity 62.559837
depth 0.836168849266797
average of disprity 64.407166
depth 0.8121858301277333
average of disprity 66.26726
depth 0.7893881386245246
average of disprity 65.56856
depth 0.7977998754833084
average of disprity 62.58546
depth 0.8358265105981366
average of disprity 64.46477
depth 0.8114601085881504
average of disprity 62.52047
depth 0.8366953652116135
average of disprity 60.801186
depth 0.8603547229709688
average of disprity 60.754425
depth 0.8610169079529295
average of disprity 58.59604
depth 0.8927324825174017
average of disprity 55.918095
depth 0.9354858662735777
average of disprity 54.50712
depth 0.959701941753246 7
average of disprity 52.251755
depth 1.0011259418851521
```

Figure 3.2: The figure reports the values of the disparity and the distance (depth). Here the distance reaches a local minimum of 0.7893881386245246m while the average disparity reaches a local maximum of 66.26726.

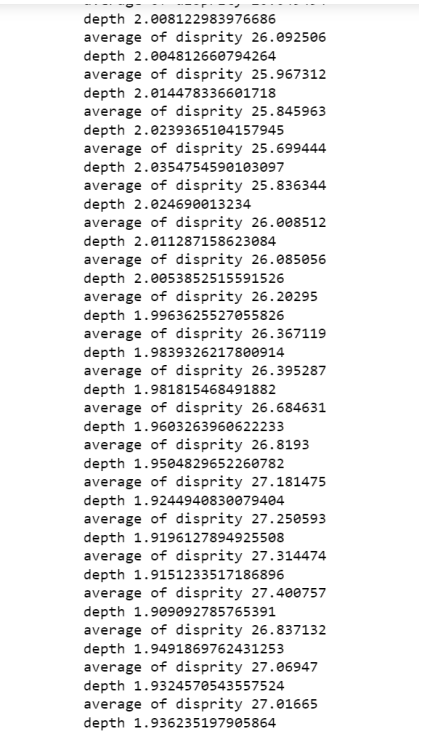


Figure 3.3: The figure reports the values of the disparity and the distance (depth). This values describe the beginning of the second approach, here the first distance is high and it gradually decreases, while the disparity is low and it gradually increases.

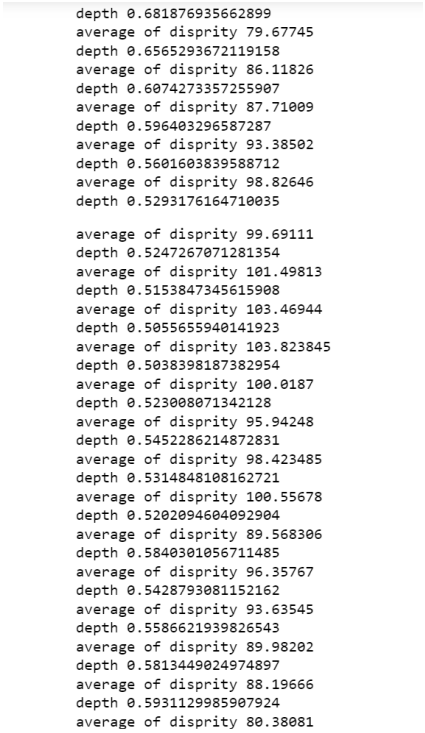
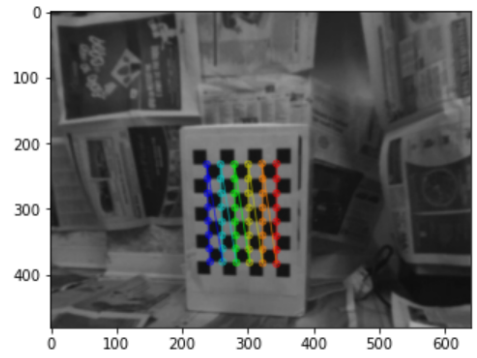


Figure 3.4: The figure reports the values of the disparity and the distance (depth). Here the distance reaches an absolute minimum of 0.5038398187382954m while the average disparity reaches an absolute maximum of 103.823845.

Figure 3.5: This is a capture of the output video showing the robot's first approach to the chessboard. Here the distance between the robot and the chessboard is 0.8603547229709688m and since it is more than 0.8m the 'Alarm' is not being printed.

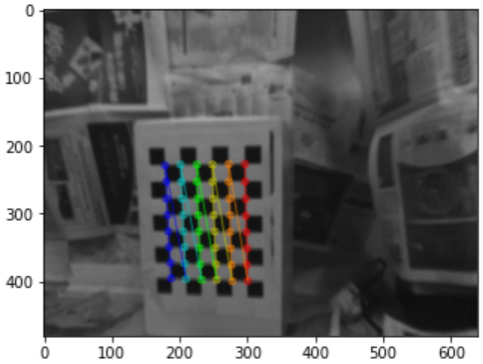
Figure 3.6: This is a capture of the output video showing a high distance between the robot and the chessboard. Here the distance is 2.3554614589577927m and since it is more than 0.8m the 'Alarm' is not being printed.

Figure 3.7: This is a capture of the output video showing the robot's second approach to the chessboard. Here the distance between the robot and the chessboard is 0.681876935662899m and since it is less than 0.8m the 'Alarm' string is printed on the top right corner of the frame.




```
(48, 1, 2)
105.8555 WIDTH AND; HEIGHT: 151.12048
REAL DIMENSION : ( 147.32206546192387 , 210.31861265502633 )
```

Figure 3.8: This image shows the chessboard at its minimum distance in the first approach. Here the chessboard corners are drawn, and the program outputs the W and H values found, where w = 147.32206546192387mm and h = 210.31861265502633mm.




```
(48, 1, 2)
117.87561 WIDTH AND; HEIGHT: 172.09904
REAL DIMENSION : ( 161.68116981006187 , 236.05540453566638 )
```

Figure 3.9: This image shows the dimensions calculated during the seccond approach at a distance of z = 0.7779858720795291m, here w is 161.68116981006187mm and h is 236.05540453566638mm

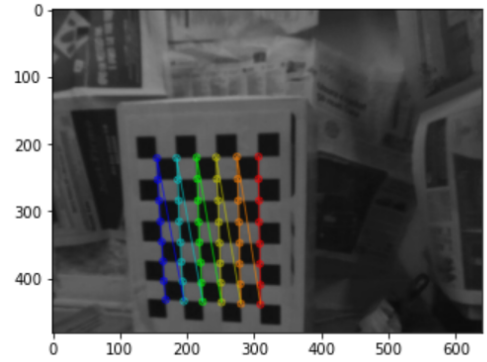


Figure 3.10: This image shows the chessboard at its minimum distance in the second approach. Here the chessboard corners are drawn, and the program outputs the W and H values found, where w = 134.24644553499647mm and h = 194.2283064187051mm.

Chapter 4

Conclusions

Based on the result that we have found we can notice that at the beginning of the maneuver the distance turns out to be inconsistent. This might be caused due to the fact that by applying the window (100 x 100), as long as we are far away from the chessboard, we are also considering, in our window, pixels of scene points that are behind the chessboard, in particular those that belong to the wall. After a while in fact, as soon as the window captures almost only the chessboard, the values start becoming consistent and the disparity increases whereas the distance decreases accordingly. The dimensions of the chessboard that we have calculated, based on the depth from the average of the disparity turn out to be not as accurate as we expected. It might be because we might not have properly tuned some parameters either that belong to the disparity function or the Harris corner detector as a result the obtained values of the chessboard are not close to that of the actual values. The first approach maneuver was supposed to be more accurate when compared with that of the second approach maneuver because of the fact that "Harris corner detector is not scale invariant" and in the second approach maneuver, the robot goes much closer to the chessboard than that of the first approach maneuver.