ESC204 - INDIVIDUAL ASSIGNMENT 3.2
3D VISION WITH 2D CAMERAS

Soroush Khoubyarian - 1005112101

April 7, 2020

This report is supplemented with a video available on: Click here
The code for this project is available on GitHub: Click here
ReadMe.md contains an overview of the files in the repository.
The rest of this document is the 5 page report, with an additional page containing the used references.

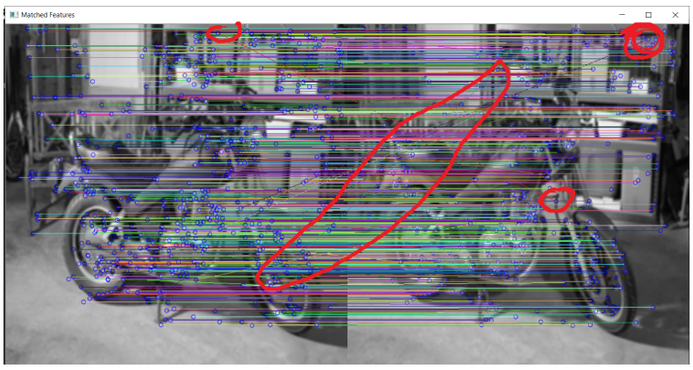


Figure 1: The performance of the SIFT algorithm on matching feature points on the Motorcycle database. Since the cameras are horizontally aligned, it is expected for all the matching lines to be horizontal; this is while multiple lines encircled in the picture are not horizontal which showcases the possiblity of computational erros in this algorithm. Such errors are can be avoided as explained in section 2.

1 Overview

The aim of this assignment is to explain the process of constructing a stereo vision point cloud based on two pictures taken from an object from two cameras spaced at some given distance in python. It is assumed that the focal lengths of the two cameras are identical and the cameras are perfectly aligned along the horizon.

To summarize, the algorithm uses the SIFT algorithm to find matching features[1] in the two pictures. (See section 2) The relative position of the matching points to the centre of each picture is used to calculate the disparity, and thus the depth[2] of each feature point can be computed using the method described in [1].(See section 3) Thereafter, the depth of feature points are used to extrapolate the depth of nearby pixels. (See section 4) This allows the extraction of a detailed map of the depth of each pixel in the input images, enabling the programmer to construct a stereo vision point cloud.

2 Feature Matching

The purpose of this component of the program is to detect and match features in the pictures taken by the left and right cameras. The output will be a set of pixel coordinates in both pictures, denoting the matching features.

To achieve this goal, the OpenCV in-built SIFT algorithm is used in Python. The SIFT algorithm (Scale-Invariant Feature Transform) is a feature detection algorithm that detects and matches features of two images in the presence of different scales and rotation. (see figure 1)

While this algorithm is useful in detecting numerous features, it sometimes matches two pixels while the pixels are not truely matching features. (as shown in figure 1) In order to avoid such computational errors, one could use the fact that the two cameras are horizontally aligned. So it is expected for two matching features to have more or less the same y component. Hence, if the difference between the y components of two matching features exceeded some error threshold ϵ, the matched features can be omitted in the calculations to gain more accurate results. (In this assignment, ϵ was set to 1% of the image height)

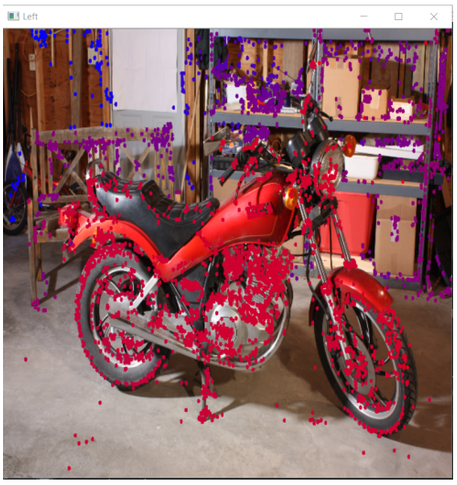
An alternative for defining an error threshold is to split the picture into thin horizontal blocks and run the SIFT algorithm on each block separately; this forces SIFT to look for matching features along the same y-component. However, this method decreases the number of detected features as shown in figure 2. This could be because the algorithm fails to detect matching points along the borders of these rectangular blocks decreases. Hence, the former approach was used for the final implementation.

Through applying this algorithm, one can extract the exact coordinates of the matched features and these coordintes can be used to calculate the depth of each feature point as explained in section 3.

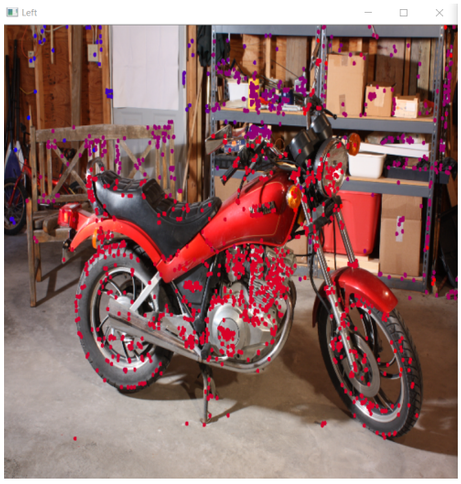
[1]Features are defined as points at which there is a sharp change in color tone or exists a corner. The exact definition of feature points varies for different feature-detection algorithms. For the purposes of this assignment, feature points are defined based on the standards of the SIFT algorithm.

[2]Depth of a point in this context means the distance of that point to the camera plane

(a) To avoid outliers, the algorithm is run on thin segments of the picture so that the matching features are forced to have the same y component.



(b) Detected feature points after running SIFT on the entirety of the picture.



(c) Features generated by running SIFT on thin segments of the picture. It has visibly less feature points than in figure 2b

Figure 2: A demonstration that analyzing the pictures in detail as in figure 2a decreases the number of feature points.

3 Feature Depth Extraction

According to [1], one could calculate the depth of each point provided the location of the point on the image plane is known with respect to both lenses. A short review of the method is provided in this section.

As seen in figure 3, the point P has coordinates x_1 and x_2 as seen by the cameras at C_1 and C_2 respectively. From section 2, the two values are known. As well, it is assumed that the distance between the two cameras $C_1C_2 = XY = b$ is a given and the focal lengths of the cameras is $f = f_1 = f_2$. Using this information, one could calculate the depth of point P, namely its distance from the camera plane PO.

Using the similarity between the triangles P_1XC_1 and P_1PZ, it can be inferred that:

$$\frac{-x_1}{f} = \frac{C_1O}{PO} \tag{1}$$

Where the negative sign behind x_1 is because P_1 is to the left of C_1. Likewise, due to the similarity between the triangles P_2YC_2 and P_2PZ, it can be shown that:

$$\frac{x_2}{f} = \frac{C_2O}{PO} = \frac{b - C_1O}{PO} \tag{2}$$

By adding the equations (1) and (2) the depth of the point P, namely PZ can be written as:

$$\frac{x_2 - x_1}{f} = \frac{b}{PO}$$

Resulting in the following equation.

$$\boxed{PO = \frac{bf}{x_2 - x_1}} \tag{3}$$

Where PO is by definition the depth of the point P, b is the distance between the cameras, f is their focal length and $x_2 - x_1$ is labeled as disparity. Notice that the disparity is readily known for each feature point from section 2.

This algorithm can be used to calculate the depth of feature points, as shown in figure 2b. Notice that in this picture, red represents points closer to the camera and blue poitns represent points further from the camera.

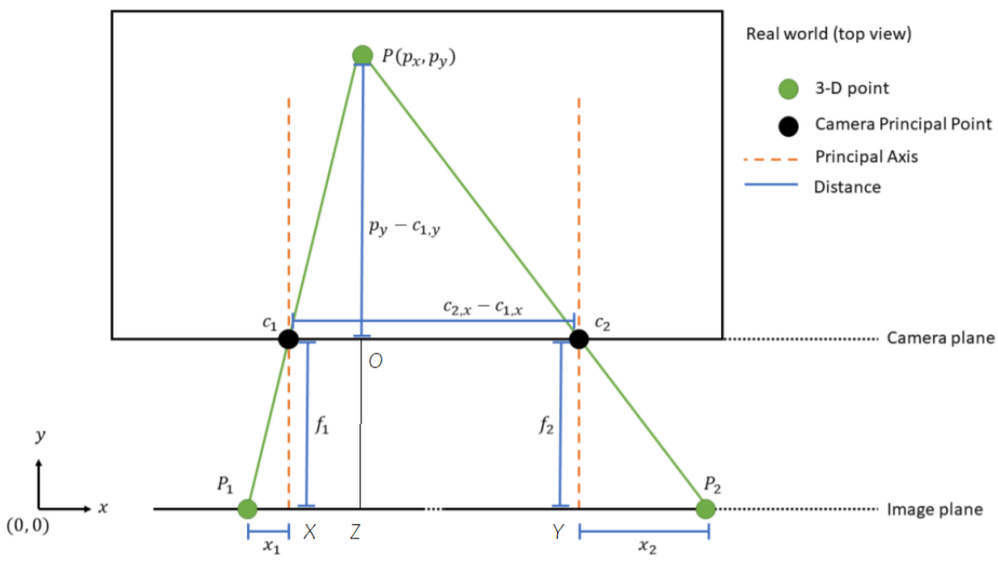


Figure 3: A schematic of how a point is viewed in the image plane by two cameras with the same focal length. It is assumed the two cameras are both along the horizon.(Taken from the assignment, slightly altered)

However, as observed in the figure, a large portion of the picture is consisted of non-feature pixels which means their depth needs to be approximated based on the calculated depths for the feature points. This is discussed in section 4.

4 Non-Feature Depth Extrapolation

The purpose of this section is to use the feature point depths in the preceding section to extrapolate the depth of the other pixles in the image.

The heuristic algorithm used for calculating the depth for any point P in the image was the following.

1. Find all feature points at a distance of less than or equal to R(an arbitrary value) to P. This ensures that only feature points close to the point P are involved in estimating its depth. For this assignment the value of R was set to 5% of the total width of the picture.

2. Calculate the distance of P with the nearby feature points. (With distance less than or equal to R)

3. Take the weighted average of the nearby feature points where the weight of each point is the reciprocal of its distance from P.

Mathematically, the aforementioned algorithm can be outlined as follows.

$$\boxed{\text{depth}(P) = \frac{\sum_{x \in A_P} \frac{\text{depth(x)}}{\text{dist}(x,P)}}{\sum_{x \in A_P} \frac{1}{\text{dist}(x,P)}}} \tag{4}$$

Where A_P is defined as $A_P = \{X \mid \text{dist}(X,P) \leq R\}$, and $\text{dist}(x,P)$ is the cartesian distance between the two points x and P.

For points P such that $A_P = \emptyset$, the depth of P cannot be calculated using this extrapolation method and an alternative algorithm is required. For the purposes of this assignment, in the code it was assumed that the depth of such points is equal to the highest depth for points X where $A_X \neq \emptyset$. In other words:

$$\text{if } A_P = \emptyset \implies \text{depth}(P) = \max\{\text{depth}(x) \mid A_x \neq \emptyset\} \tag{5}$$

Note that the higher the number of feature points, the more accurate this depth extrapolatoin becomes.

Since the precision of this step depends on the density of the feature points, the higher the number of feature points the more accurate the final results will be. This is why in section 2 max-

imizing the number of matching features was a preference.

In figure 4 the result of extrapolating the depths from figure 2b is given.

5 Stereo Vision Point Cloud Construction

All that remains, is to construct a 3D map of the analyzed images in the preceding sections. Since Python uses an interpreter, it is significantly slower than other programming languages like C++. Since in this section it is needed to draw and rotate the location of every pixel as the user rotates the point cloud, the point cloud construction was done in C++ instead of Python.

For drawing the point cloud the SDL library in C++ was used and for rotating the point cloud, the coordinates of the pixels were multiplied by the rotation matrix defined below. For this assignment only rotations about the x and y were implemented.

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix} \tag{6}$$

$$R_y = \begin{pmatrix} -\sin(\theta) & 0 & \cos(\theta) \\ 0 & 1 & 0 \\ \cos(\theta) & 0 & \sin(\theta) \end{pmatrix} \tag{7}$$

6 Rectification of Stereo Images

The analysis in this report was all under the assumption that the image planes of the two cameras are parallel to the camera plane; in other words, it was assumed the cameras are parallen and along the horizon.

In mobile robots, it is possible for the cameras to have any relative orientation with respect to each other, making the assumption made in this assignment invalid. This is because the equation derived in section 2 fails. (Equation (3))

However, if the relative pose of the two cameras are known (The translation vector between them $\vec{T}$ and the rotation matrix representing their relative orientation $\boldsymbol{R}$), one could use a linear mapping to map the two pictures captured by the two cameras onto the same plane. Afterwards, the process explained in this report can be followed to generate a stereo vision point cloud.

Assuming the cameras are placed rigidly on the robot (Their relative pose does not change), one could use a formulation of the hande-eye callibration problem to find this relative pose. In order to do so, two IMUs can be placed on the cameras. These two sensors measure the relative pose of the cameras with respect to their original orientation as the robot moves. The output of the IMU sensors is repeatedly analyzed to produce the transformation matrix X which describes the relative pose of the two cameras. The way X in this context is defined is as follows.

$$X = \left[\begin{array}{c|c} \boldsymbol{R} & T \\ \hline 0^{1\times 3} & 1 \end{array}\right]$$

The outputs of the two gyroscopes (Labeled A_i and B_i for camera 1 and camera 2 respectively) can now be used to reduce this problem into a special case of the hand-eye callibration problem as follows:

$$A_i X = X B_i$$

The justification for this equation as well as one way in which this problem can be solved, is shown in [2].

So using two IMUs and an implementation of the hand-eye callibration problem, one could calculate the relative pose of the two cameras, making it possible to derive the appropriate disparity equation to calculate the depth of the feature points. Notice that the SIFT algorithm mentioned in section 2 still applies, as SIFT is rotation invariant.

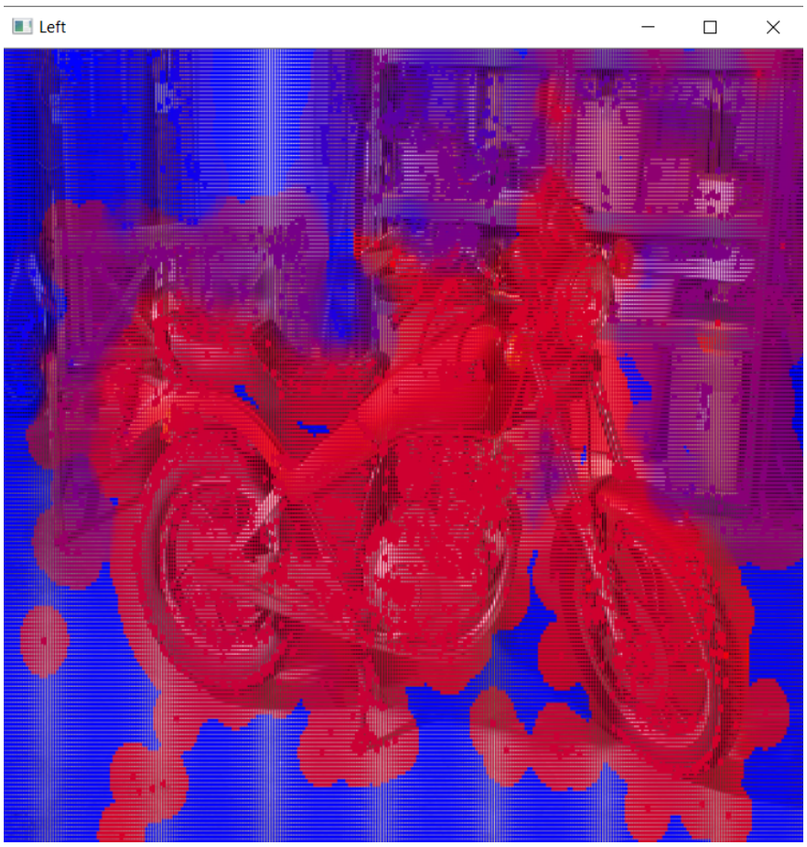


Figure 4: The result of extrapolating the depths based on the depth of the feature points. The red to blue spectrum represents depth, red being low depth and blue being high depth points. As observed, the algorithm successfully signifies the motorcycle in red shade, the chair on the top left and the bookcase on the top right of the image in purple shade. (So the chair and the book case are behind the motorcycle)

References

[1] Chapter 11 depth, individual assignment recommended resource. https://www.cse.usf.edu/~r1k/MachineVisionBook/MachineVision.files/MachineVision_Chapter11.pdf, [Accessed on 6/4/2020].

[2] Jan Heller et al. Structure-from-motion based hand-eye calibration using l-infinity minimization. https://www.researchgate.net/publication/221362714_Structure-from-motion_based_hand-eye_calibration_using_L_minimization, [Accessed on 7/4/2020].