Instituto Superior Técnico

Mestrado Integrado em Engenharia Mecânica

Computational Vision

Work number 2 – Problem 16

Monocular Visual Odometry

Afonso Valador, 87142

José Trigueiro, 87225

Professors: José Azinheira and Alexandra Moutinho

2020 – 2021

20th of November 2020

TÉCNICO LISBOA

Contents

1. Introduction

According to [1], “Visual odometry is the process of determining the location and orientation of a camera by analysing a sequence of images. Visual odometry is used in a variety of applications, such as mobile robots, self-driving cars, and unmanned aerial vehicles.” This is the subject of the present work, where the example in [1] will be analysed, followed by a brief discussion of the mathematical theory behind the implemented algorithm will be discussed.

Two experiments were performed to test if the example could be applied in a real-life scenario, where estimate data for the magnitude of camera displacement was available from an accelerometer or similar sensor. Experiment 1 (“Kitchen Scene”) deals with movement in the direction the camera points to, while Experiment 2 (“Room Scene”) models a scenario where the camera is moving perpendicular to the direction it points to. Estimate trajectory and tracking error are calculated and analysed, and a sensitivity analysis to the variation of the algorithm’s hyperparameters is performed to try to improve the results.

2. MATLAB Example

The given example is based on an image sequence from the “New Tsukuba Stereo Dataset,” from Tsukuba University’s CVLAB [2] [3]. This image dataset was generated from a computer graphics (CG) rendering of an office (“Head and Lamp” scene). The advantage of using a CG simulation is that camera ground truth position and rotation may be generated inside the program and provided along with the image data as a way of evaluating performance of any camera tracking algorithm that is applied to the sequence. While, according to [2], the original dataset provides multiple stereo pairs and lighting conditions, in this case only a sequence representing a single camera moving around the room will be used, since monocular visual odometry, as the name indicates, deals only with trajectory estimates using a single camera view. Images from the start and end of the camera trajectory are shown in Figure 1.

Figure 1 – Start (left) and ending (right) of the camera trajectory in the dataset. Image from [1].

Camera intrinsics (focal length, principal point, and image size) are also given in the dataset. In this case, the virtual simulation did not add any lens distortion or skew to the image, so both parameters were assumed to be 0. Since the image is digital no conversion factors for the number of pixels per unit length need to be considered. The camera's intrinsic matrix, using the values provided, is then given by:

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 615 & 0 & 320 \\ 0 & 615 & 240 \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

Note that in MATLAB this matrix is transposed from what is seen in most references, since in the software it is assumed that point coordinates are line vectors as opposed to column vectors in the literature.

The example algorithm is divided in three steps: **estimating the pose of the second view relative to the first view**, **bootstrapping estimating camera trajectory using global bundle adjustment** and **estimating remaining camera trajectory using windowed bundle adjustment.**

- **Estimating the pose of the second view relative to the first view**

In a first step of the algorithm, matching interest points will be found between the first and second images of the sequence, by converting the images to grayscale and then applying the SURF algorithm to both. An example of the resulting matching points between the two first frames of the sequence is shown in Figure 2.

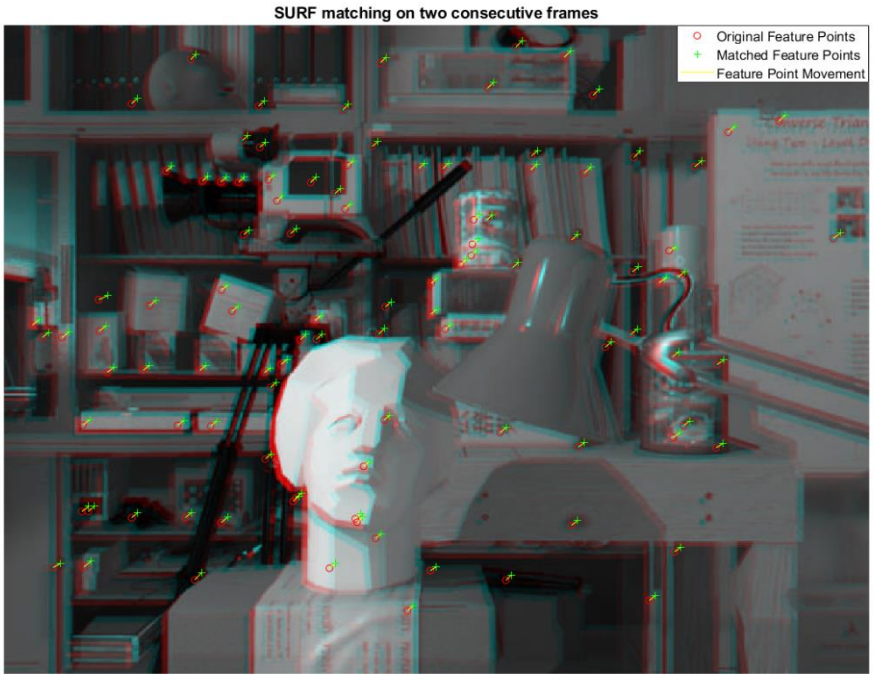

Figure 2 – Matching SURF features between two consecutive frames

With matching points in the first two consecutive frames available, the relative position of the camera between these two frames may be estimated. This may be done through an approximate computation of the essential matrix by solving a five-point relative pose problem [4] [5] using MSAC [6], a more computationally efficient variation of the RANSAC algorithm, and then choosing the corresponding physically realizable solution. Finally, outliers are excluded from the set of matching points

Note that the previous process is scale ambiguous, so the ground truth data for the location of the cameras is then used to normalize the estimated location, by multiplying it by the mean ratio between real and estimated positions. This simulates the effect in the system of adding an external sensor that gives additional data about the position of the camera.

- **Bootstrapping estimating camera trajectory using global bundle adjustment**

The previous calculations serve as initialization for the next step of the trajectory estimation process. For the following images in the sequence, the 3D position of matching SURF feature point in an image and the previous two is found and then used to find the corresponding 2D points in the current image, after eliminating outliers. The camera pose in the world coordinate system is then solved through a perspective-three-point problem [7] solved again by MSAC.

For a final refinement of the results of any reconstruction algorithm, as recommended by [8], bundle adjustment should be applied. In the first 15 frames of the sequence, global bundle adjustment is applied at every frame, using all matching points from all previous

frames. Since global bundle adjustment optimizes both 3D point positions and the camera matrices, normalization needs to be called again to ensure the scale is coherent with the ground truth data and that the first camera is placed at the origin of the world coordinate system pointing along the Z axis. Bundle adjustment is fundamental to the correct estimation of the trajectory, or else the drift effect resulting from the perspective-n-point problem would lead to a large, accumulated estimation error.

- **Estimating remaining camera trajectory using windowed bundle adjustment**

For the remaining frames, to save computational power and make the algorithm faster, the process of separating outlier points is skipped. Windowed bundle adjustment is used instead for a window of the 15 previous views, being repeated every seventh view instead of every view, since the growing number of optimization parameters would make the computation prohibitively expensive.

Note that Since after this only partial bundle adjustment is used, the initial positions remain fixed and no further normalization is needed.

After sequentially applying all three steps of the process, the resulting real and estimated camera trajectory may be plotted, with the results being shown in Figure 3.

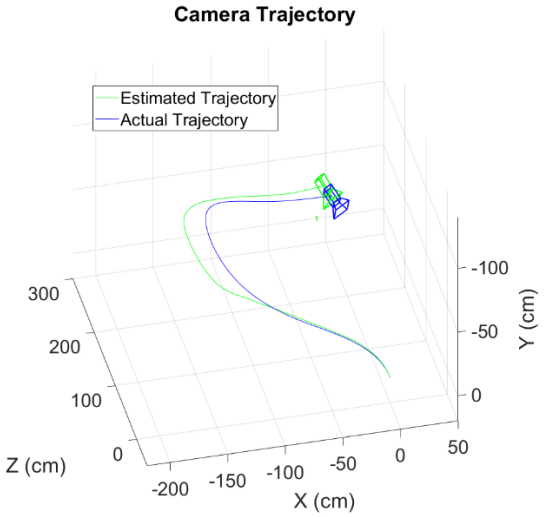

Figure 3 – Real and estimated trajectory of the camera

3. Mathematical foundation

In the previous section, multiple Computer Vision algorithms related to finding both 3D location of points, finding camera location and angle (extrinsics) and matching features in

sequential images were used. This section aims to briefly describe the principles behind these algorithms. A summary of the most important procedures

Table 1 - Main algorithms used in the example

Algorithm	*Usage*
SURF	Matching feature points between consecutive images
Five-point relative pose problem	Estimate Essential matrix and find relative camera poses
Triangulation – Hartley [8]	Find 3D (world referential) location of 2D points
Perspective-three-point Problem	Find the orientation and location of a camera in world coordinates
Bundle adjustment	Global optimization of 3D point coordinates and multiple view camera matrices

3.1. SURF algorithm

As expressed in section 2, SURF was used to find common points of interest between two consecutive frames. As known, SURF is a scale and rotation invariant algorithm, based on image convolution of integral images [9]. There are three main steps in this method. First there is a search for interesting points in each frame, at distinct locations (blobs, T-junctions, and corners, for example). Secondly, the neighbourhood of every interesting point is represented by a feature vector and, finally, the feature vectors are matched between consecutive frames based on, for example, Euclidian distance.

3.1.1. Features Extraction

a) Integral Images

Integral images are used to allow a fast computation of box type convolution filters. It represents the sum of all values in each image I (in this case, a grayscale one), within a rectangular region formed by the origin and the coordinate x, and it is calculated:

$$I_{\Sigma}(x) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i,j) \tag{2}$$

b) Fast Hessian Matrix

This approach uses a basic Hessian matrix, commonly called as 'Fast Hessian matrix' due to its reliable performance and low computation time. Given a point $\mathrm{X} = (x, y)$ at scale σ the Hessian matrix is defined as:

$$\mathcal{H}(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{yx}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \quad (3)$$

where $L_{ij}(x,\sigma)$ is the convolution of the Gaussian second order derivative with the image I in point X. To select the location and the scale, SURF uses the determinant of this matrix to achieve this goal, which can be calculated by applying, first, convolution with Gaussian Kernel, followed by the second-order derivative. Box filters are then used to push the approximation even further, allowing it to be evaluated at a very low computational cost with the usage of integral images.

In the figure above it is possible to see the 9x9 box filters

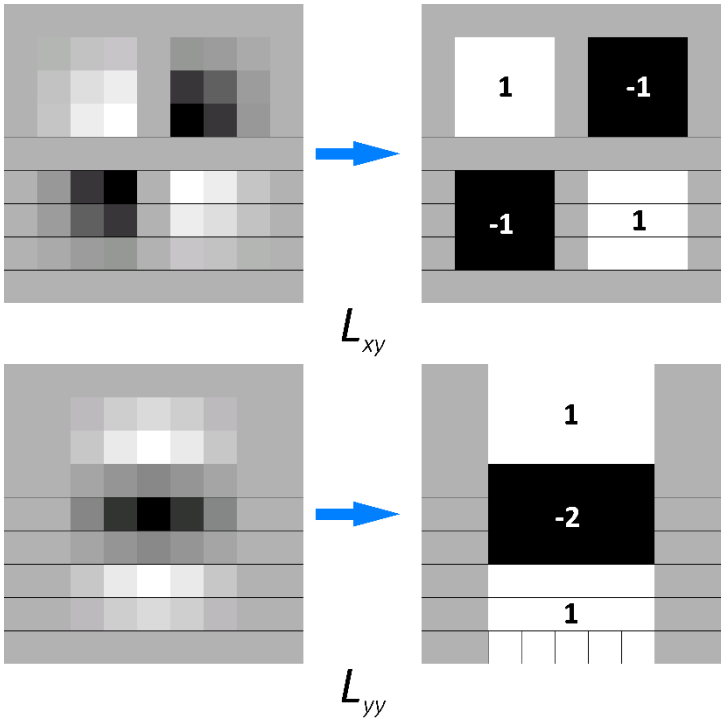

Figure 4 - Gaussian partial derivative in xy (up) and y (bottom). Image adapted from [9]

In the figure above it is possible to see, for example, a 9x9 box filter which represents the referred approximations (Gaussian second order derivatives).

The determinant of the Hessian – or 'Fast Hessian,' to be more precise – can now be computed:

$$det(\mathcal{H}_{approx}) = D_{xx}D_{yy} - (\omega D_{xy})^2 \quad (4)$$

with $\omega = 0.9$ [9].

c) Scale Spacing

Scale spaces are represented as image pyramids, as Figure 5 suggests:

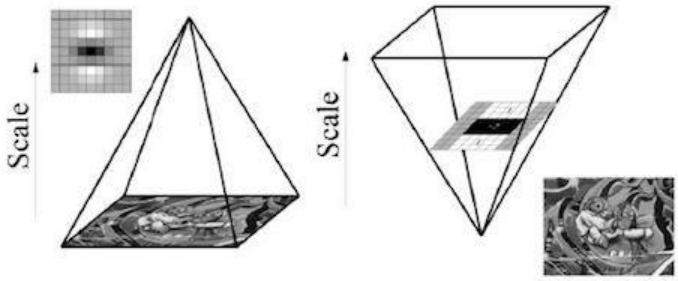

Figure 5 – Scales Spaces in SURF. Image from [9]

Figure 5 also allows to a better understanding of scale space in SURF. Due to the use of box filters and integral images, scale space is analysed by up scaling the filter size, rather than iteratively reducing the image size.

In the example given above, the 9x9 box filters represent the initial scale layer, or scale $s = \sigma$ and the following ones are up scaled (in this case, $15 \times 15, 21 \times 21, 27 \times 27$ etc.). For each new octave, the size of the filter is doubled (from 6 to 12 and 24). The scale of each layer is proportionally computed, i.e.:

- $15 \times 15,\ \sigma' = 1 \times \sigma$
- $21 \times 21,\ \sigma' = 2 \times \sigma$
- $27 \times 27,\ \sigma' = 3 \times \sigma$
- etc.

3.1.2. Features Description

a) Orientation Assignment

First, SURF calculates the Haar-wavelet responses in x and y directions and in a circular neighbourhood of radius $r = 6s$ around the current key point, s being the scale of that key point.

Second, an analysis is made in a range of $\frac{\pi}{3}$ radians across all circles. The section with the greatest number of wavelet responses is the section that contains the orientation vector, as Figure 6 suggests:

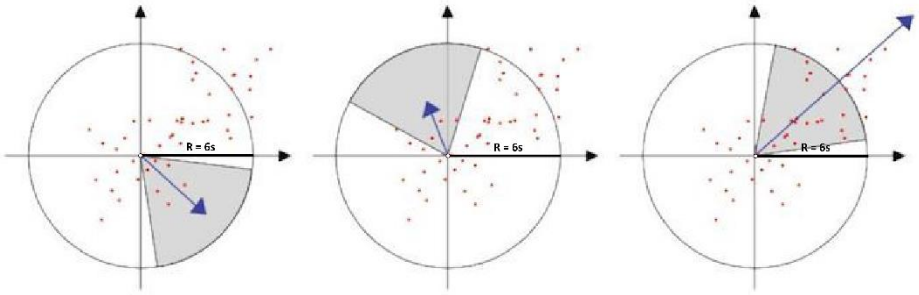

Figure 6 - Orientation Assignment in SURF. Image from [9]

b) Descriptor Components

It is now time to extract the descriptor, first by building a square around the current key point and with the orientation estimated in a). This square is then divided into a 4×4 sub-region. The Wavelets with horizontal direction are now named dx and the vertical ones dy. It is also important to extract the absolute value of dx and dy, so that It is now possible to form the descriptor vector $v = (\sum dx, \sum dy, \sum |dx|, \sum |dy|)$

3.2. Five-point relative pose problem

The solution to this problem consists in iteratively calling two steps: the estimation of the essential matrix and the computation of the views' relative pose. When reaching a high enough percentage (80%) of placement of 3D points in front of the camera by the model, the relative pose is considered correct and the process is stopped. The two steps of the algorithm are described below.

3.2.1. Essential matrix estimation

The five-point relative pose problem is used to find the relative pose of camera P_2 with respect to camera P_1. The relation between the projections of corresponding 3D points X in cameras 1 and 2 (x_1 and x_2, respectively) are given by:

$$x_2^T F x_1 = 0 \quad (5)$$

In the above equation F is the fundamental matrix. This matrix only allows for the extraction of the camera poses up to a projective ambiguity, so the essential matrix E should be used instead. This matrix leads to camera matrices that are only fourfold ambiguous (four

possible solutions) plus a scale ambiguity, solved in the example by the ground truth data. For the use of this matrix, normalized coordinates $\hat{x}_1$ and $\hat{x}_2$ must be used:

$$\hat{x}_2^T E \hat{x}_1 = 0 \tag{6}$$

For camera intrinsic matrices K_1 and K_2, and setting the referential to the first camera, the camera matrices are given by:

$$P_1 = K_1[I|0] \tag{7}$$

$$P_2 = K_2[R|t] \tag{8}$$

In this case, since the camera is the same, the camera intrinsic matrix K is the same for both images, meaning the new relations are easily calculated:

$$\hat{P}_1 = K^{-1}P_1 = [I|0] \tag{9}$$

$$\hat{P}_2 = K^{-1}P_2 = [R|T] \tag{10}$$

$$\hat{x}_1 = K^{-1}x_1 \tag{11}$$

$$\hat{x}_2 = K^{-1}x_2 \tag{12}$$

The five-point algorithm proposed in [5] takes advantage of a fundamental property of essential matrices: as stated in [8], “a 3x3 matrix is an essential matrix if and only if two of its singular values are equal, and the third is zero. In [5] the property is shown to lead to the following two equations:

$$\det(E) = 0 \tag{13}$$

$$EE^TE - \frac{1}{2}trace(EE^T)E = 0 \tag{14}$$

The essential matrix constraint for the projections of the points in each image may be written as:

$$\tilde{q}^T \tilde{E} = 0 \tag{15}$$

$$\tilde{q} \equiv [\hat{x}_1\hat{x}_2 \;\; \hat{y}_1\hat{x}_2 \;\; \hat{z}_1\hat{x}_2 \;\; \hat{x}_1\hat{y}_2 \;\; \hat{y}_1\hat{y}_2 \;\; \hat{z}_1\hat{y}_2 \;\; \hat{x}_1\hat{z}_2 \;\; \hat{y}_1\hat{z}_2 \;\; \hat{z}_1\hat{z}_2] \tag{16}$$

$$\tilde{E} \equiv [E_{11}\; E_{12}\; E_{13}\; E_{21}\; E_{22}\; E_{23}\; E_{31}\; E_{32}\; E_{33}] \tag{17}$$

Calculating $\tilde{q}$ for five points leads to a 5x9 matrix, meaning the solution for E will be made up, by Single Value Decomposition of:

$$E = xX + yY + zZ + wW \tag{18}$$

In the above equation X, Y, Z, and W are 3x3 matrices and x, y, z, and w are unknown constants. Since the scalars are only defined up to a scale factor, w is assumed to be 1, while the others must be determined by substitution in the two equations associated with the definition of the essential matrix, leading to ten third order polynomial equations in three variables and 20 monomials, which may then be written in matrix form, where M is the 10x20 matrix of the polynomial coefficients and X is a vector with all possible monomials with three cubic variables:

$$MX = 0 \tag{19}$$

Solving this equation is done in MATLAB through a technique called Polynomial Eigenvalues, leading finally to the computation of the essential matrix.

Note that for robustness to noise and inaccuracies, more than 5 points are used, and the essential matrix is computed using MSAC, where random 5-point samples are taken from the matched points in two images, and models are iteratively computed to minimize a cost function, in this case the Sampson distance, as defined in [8]:

$$\sum_i \frac{\left(x_{2,i}^T F x_{1,i}\right)^2}{(Fx_{1,i})_1^2 + (Fx_{1,i})_2^2 + (Fx_{2,i})_1^2 + (Fx_{2,i})_2^2} \tag{20}$$

3.2.2. Relative pose estimation

Having computed the essential matrix, the problem remains of extracting the relative pose of the cameras from it. As said before, the essential matrix allows for the estimation of relative position of the cameras up to a fourfold ambiguity plus the scale. This part of the algorithm will then be responsible for computing the four possible solutions and choosing the one that corresponds to the real position of the cameras and points.

Following the deduction in [8], it is possible to decompose the essential matrix E into $E = [t]_\times R = SR$, where t is the translation vector and R is the rotation matrix between the two cameras. Assuming the Single Value Decomposition of E is $Udiag(1,1,0)V^T$, there are two possible factorizations: $S = UZU^T$ and $R = UWV^T$ or $R = UW^TV^T$, where:

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad Z = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{21}$$

Since the sign of E, and consequently the sign of the translation vector t is also indetermined, the two different factorizations yield four different solutions, depending on the sign added to the last column of U:

$$P_2 = [UWV^T|+u_3] \; or \; [UWV^T|-u_3] \; or \; [UW^TV^T|+u_3] \; or \; [UW^TV^T|-u_3] \quad (22)$$

These four solutions have a simple geometric interpretation, shown in Figure 7, which also gives a hint for the procedure to choose the realizable solution: only in a) the point reconstruction is in front of both cameras.

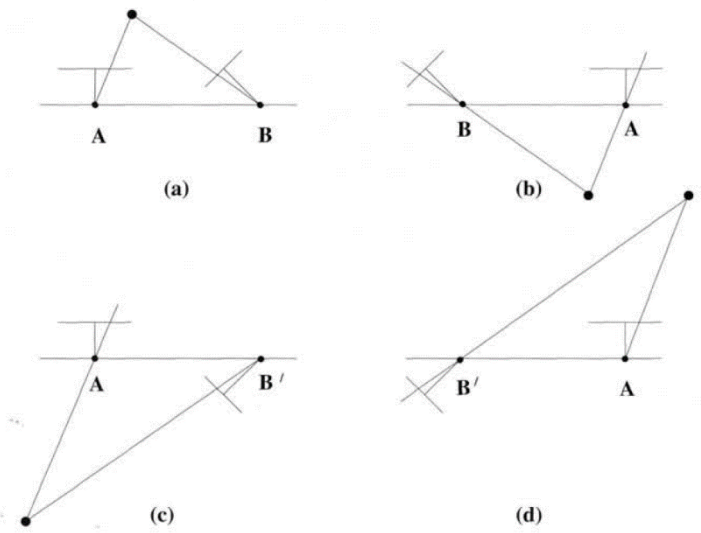

Figure 7 – Geometric interpretation of the four solutions of the relative camera poses problem. Image from [8]

Note that these relationships give the coordinate transformation between view 1 and view 2. To find view 2's pose, the relation must be inverted:

$$R_{1\to2} = R^{-1} = R^T \quad (23)$$

$$t_{1\to2} = -t * R^{-1} = -tR_{1\to2} \quad (24)$$

To choose the right solution, the triangulation algorithm described in [10] is used, using the estimated camera matrix from before. It consists in finding the midpoint of the common perpendicular between the two rays corresponding to every pair of matched points, by minimization of the squared distance between the two rays. The authors note that this algorithm is easy to compute but also not projective or affine invariant, since perpendicularity is not an affine concept and midpoint not a projective one. However, since only a rough estimate of the point locations is necessary to choose between solutions, this does not pose a problem. The best solution is the one that places the minimum number of points behind the cameras.

3.3. Triangulation

Triangulation is used to find the 3D location of points from their projection in a camera's image, up to a scale ambiguity. In MATLAB, an algorithm proposed by [8] is used, based on the Direct Linear Transformation (DLT) algorithm. As defined before, projections on two views of the same point are given by $x = PX$ and $x' = P'X$. Taking the cross product $x \times (PX)$ and $x' \times (P'X)$, taking p^{iT} to be row *i* of P, a system of linear equations is defined:

$$AX = 0 \Leftrightarrow \begin{bmatrix} xp^{3T} - p^{1T} \\ yp^{3T} - p^{2T} \\ x'p'^{3T} - p'^{1T} \\ y'p'^{3T} - p'^{2T} \end{bmatrix} X = 0 \tag{25}$$

For multiple matching points, the system is over-determined and may be solved by setting an extra condition. In this case the norm $\|AX\|$. It is shown that the solution to this problem is the unit eigenvector corresponding to the smallest eigenvalue of $A^T A$, which when performing Singular Value Decomposition as $A = UDV^T$, X is the last column of V if the entries of the diagonal of D are arranged in descending order.

3.4. Perspective-three-point Problem

The perspective-three-point Problem deals with the calculation of camera pose knowing the 3D world coordinates of three points in an object. The geometry of the problem is shown in Figure 8.

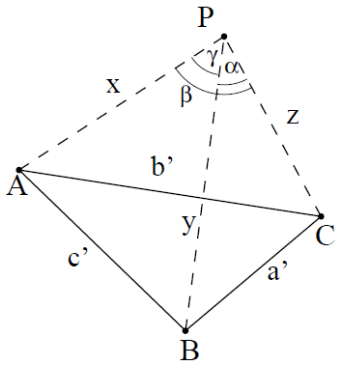

Figure 8 – Geometry of the P3P problem. Image from [7].

According to [7], the P3P problem may be reduced to finding the zeros of a quartic polynomial, with up to four unique solutions for the camera pose. From Figure 8, using the geometry of triangles PBC, PAC and PAB, where P is the centre of perspective of the camera and A, B and C are the control points in world coordinates, the system of equations that allows for the solution may be defined:

$$\begin{cases} Y^2 + Z^2 - 2YZcos(\alpha) - a'^2 = 0 \\ Z^2 + X^2 - 2XZcos(\beta) - b'^2 = 0 \\ X^2 + Y^2 - 2XYco(\gamma) - c'^2 = 0 \end{cases} \quad (26)$$

Following some reality constraints, the solutions for the world points in the camera referential may be obtained, with the camera centre position then being calculated via a rigid transformation between the two coordinate systems.

Due to noise, outlier points and missing matches, and due to the multiple possible solutions, this algorithm is applied using MSAC, taking sets of four points from all matching points, finding the solution for the P3P problem that best approximates the fourth points position, with the correct solution being the model that best describes the maximum number of four-point subsets of the matching points.

3.5. Bundle adjustment

As described by [8], bundle adjustment is an optimization algorithm that iteratively calculates a set of camera matrices P_i and 3D point coordinates X_j, given the projections of point *j* in camera *i* x_{ij}. This method provides a Maximum Likelihood estimator (assuming gaussian noise) that is general and tolerant of missing data (for example, missing a point's matching image in a certain view). Bundle adjustment has, however, some significant problem's, which the example's authors tried to minimize, mainly the lack of guarantee of convergence to an optimal solution given any initial estimate for the iterative process and the slowness of the algorithm given many matching points and images (since each camera has 11 and each point 3 degrees of freedom for *p* matching points observed in *n* views, *3p + 11n* optimization para meters will be needed. The first problem is minimized by the application of all algorithms described in this work in conjunction with bundle adjustment, giving both a starting set of camera matrix estimates and a triangulated position for 3D world points. The second problem is mitigated by reducing the number of views used simultaneously (a maximum of 15 in this case) through windowed bundle adjustment, by only applying the algorithm every seven views, and by taking advantage of appropriate sparse matrix algorithms, since the matrices used for bundle adjustment are often highly sparse.

In this example the algorithm used for bundle adjustment is the one described by [11], using a sparse variation of the Levenberg-Marquardt algorithm to minimize the Euclidean distance between the observed projected points on each image and the projection calculated using iteratively estimated world coordinate positions of the points and camera matrices:

$$\min_{\hat{P}_i, \hat{X}_j} \sum_{ij} d\left(\hat{P}_i \hat{X}_j, x_{ij}\right)^2 \quad (27)$$

3.6. RANSAC and MSAC

Since the matching process between images is based only on proximity and similarity, the occurrence of mismatches is usual [6]. Consequently, the RANSAC (*Random Sample Consensus*) algorithm is commonly used for robust fitting of models in the presence of a considerable number of outliers in the input data. In this case, outliers are just pairs of matching points that are not consistent with the epipolar geometry that characterizes inliers.

The fundamental matrix *Z* can be estimated from random sets of seven data points, resulting in:

$$Z = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_7x_7 & x'_7y_7 & x'_7 & y'_7x_7 & y'_7y_7 & y'_7 & x_7 & y_7 & 1 \end{bmatrix} \quad (28)$$

Because the dimension of Z is $[7 \times 9]$, the solution *F* can be obtained from the two-dimensional null space of Z as demonstrated in the next steps.

Starting by calling f_1 and f_2 the two right-hand singular vectors of *Z* with singular values of zero, then they form an orthogonal basis of the null space of *Z*. Let U_1 and U_2 be the $[3 \times 3]$ corresponding to f_1 and f_2, then the fundamental matrices F_l, *l* = 1, 2, 3, are can be obtained from $F_l = \alpha U_1 + (1-\alpha)U_2$, subject to scaling and $det(F_l) = 0$ (which gives a cubic in α from which one or three real solutions are obtained).

In the initial match set, the number of "matches" with the error (Equation 29) below a threshold *T* is designated the support of the fundamental matrix.

$$e_i^2 = \sum_{j=1,2} \left(\hat{x}_i^j - x_i^j\right) + \left(\hat{y}_i^j - y_i^j\right) \quad (29)$$

Each solution F is tested for support and the full process is repeated along all the random sets defined. The fundamental matrix with the highest support is accepted and the output is a set of inliers and outliers well separated.

At this point one problem remains: reaching the ideal number of samples (*m)* to use. The expression below is calculated to give a probability γ a value of 95% or higher, so the parameter *m* should be tuned higher enough for this purpose.

$$\gamma = 1 - (1 - (1-\epsilon)^p)^m \quad (30)$$

Where ϵ is the fraction of contaminated data and *p* the number of features in each sample.

It is better to take more samples than needed simply because, as is usual, some of them may be degenerate. More than this, this algorithm may require fewer repetitions than the number of outliers, as it is only related to the proportion of the latter.

The MSAC (*M – Estimator Sample Consensus)* algorithm is a reformulation of the previous one that sets the threshold parameter T as $T = 1.96\sigma$, where σ is a standard deviation, so that the Gaussian inliers are only reject 5% of the time. This methodology is proven to be computationally better than the RANSAC and is preferable to use.

4. Experimental Procedures

The example is, for the most part, ready for testing in a real scenario, with two exceptions: perfect ground truth data for camera location and orientation is not available in a real scenario and camera intrinsic matrix is unknown until a calibration is performed. To deal with these two problems, an algorithm must be developed for scaling the captured images in the first steps of the algorithm, in the normalization phase, as well as to calibrate the camera

4.1. Camera Description

Due to time constraints and COVID restrictions of number of students in the lab, the datasets for the implementation were recovered at home. Due to this, the camera used was the rear facing camera of smartphone Xiaomi Poco X3 NFC. This camera only takes photos in 4k resolution (3840 x 2160 pixels), which has the disadvantage of increasing file sizes and computational times (for example, the first two steps as described in section 2 , on the same computer, take 20 seconds to run as opposed to 7 in the original example) as opposed to the images of the original examples. It was decided to keep the images at this resolution as opposed to running them through a compression software on advice of the lab teacher, since this type of program could affect the geometry of the scene if implemented incorrectly and calibration could also be compromised.

4.2. Camera Calibration

The camera intrinsic matrix can easily be extracted by a calibration process with the help of the MATLAB app “Camera Calibrator.” Since Monocular Visual Odometry deals with only a single camera, the relative position of a stereo pair of cameras was not needed or possible to use for the calibration.

Some pictures of the MATLAB default checkerboard were used in various positions and rotations to perform the calibration, as illustrated below[1] in Figure 9.

[1] The images in presented in figure 8 are just some of the whole set of images that were used.

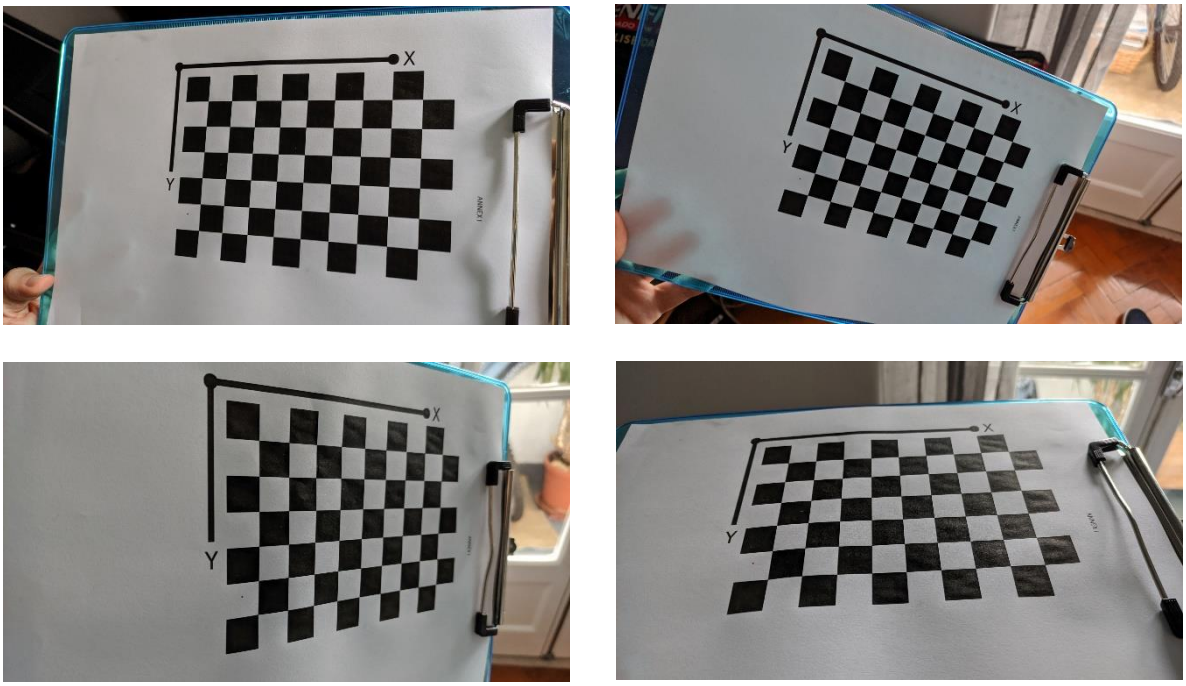

Figure 9 - Image samples used in calibration

Having inserted all the images in the app, before the calibration procedure itself, the size of the checkerboard square, in millimetres, is needed to proceed. After this, the camera is ready to be calibrated and the results are expressed in Figure 10.

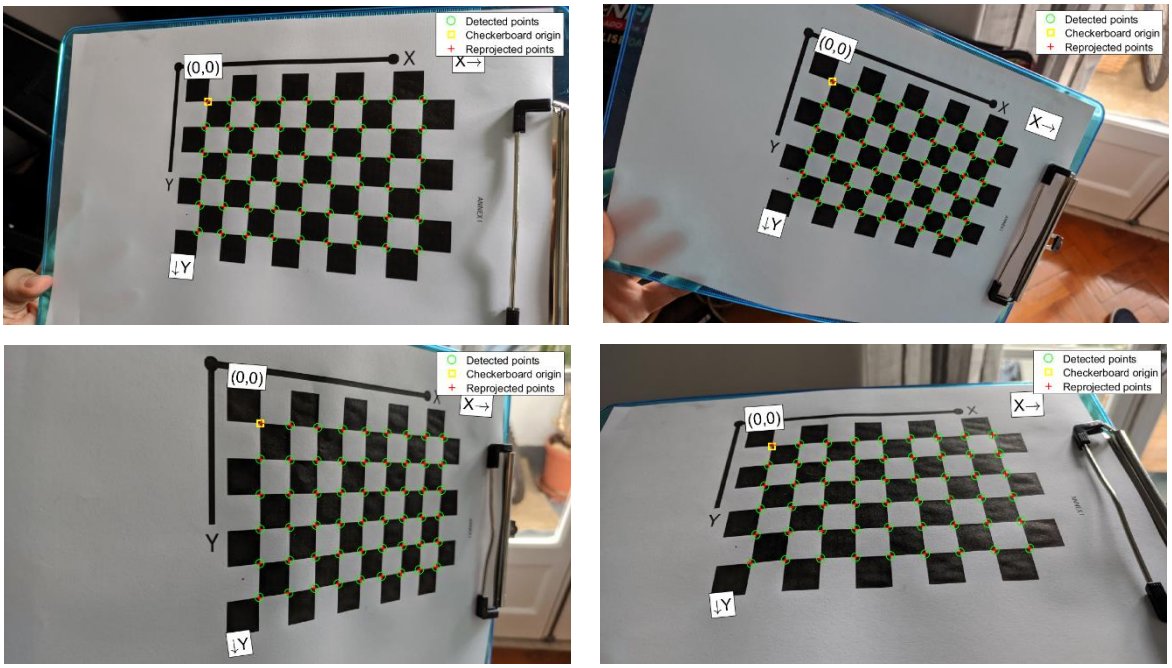

Figure 10 – Points detected and reprojected points of the images of figure 9

The extrinsic parameters (relative position and rotation from camera to checkerboards) can now be viewed in Figure 11, either in a pattern-centric way (left) or in a camera-centric way (right). From these 2 images, an estimation of the accuracy of the calibration can be made by checking if the pattern is too far or too close from the camera or even by verifying if the patterns appear behind the camera, for example.

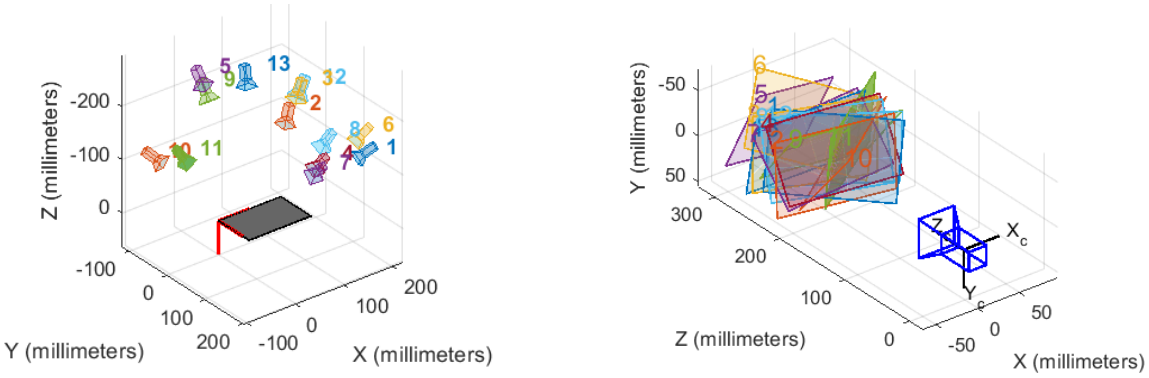

Figure 11 – Extrinsic parameters visualization: Patter centric (left) and Camera centric (right)

It is also possible to evaluate the accuracy of the calibration in a quantitative way, by estimating the reprojection errors. A reprojection error is the distance between a pattern key point detected in a calibration image, and the corresponding world point projected into the same image by the projection matrix of the camera [12]. The following figure provides a useful comparison between the reprojection error across all the images used for calibration and the overall mean error. If this error is too high one should consider excluding the images with error above a certain threshold and recalibrating. Reprojection error for the calibration image set used is shown in Figure 12.

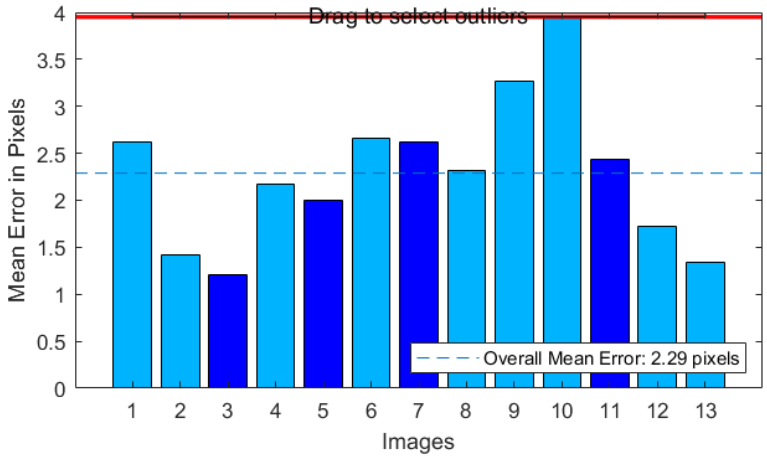

Figure 12 - Reprojection error in all the images used for calibration

Because the value of the type of error is expressed in pixels, its evaluation is a task that is related with the resolution/quality of the camera. Since we are dealing with a high-resolution camera (3840 pixels in length and 2160 pixels in height), the overall mean error, as well as the maximum error, are numerically insignificant compared to the size of the image.

4.3. Experiment 1: the "Kitchen Scene dataset"

The first experiment to be performed was done with the setup shown in Figure 13. The smartphone was mounted in a set of supports to keep its rotation and sideways movement fixed. The camera was then moved parallel to a measuring tape (seen in yellow) and photos were taken in increments of 10 mm per view. A set of objects were placed in front of the camera to provide a scene richer in SURF features for the algorithm to generate matching points. As in the example, the world referential has its origin at the camera's initial position, with the camera facing along the z axis. Special precaution was taken to not click the focus button on the phone to make the acquired calibration matrix as accurate as possible.

Figure 13 – Experiment 1 setup

Some example images (first and last) from this dataset are shown in Figure 14, undistorted and with SURF points overlaid. Note that a considerable number of SURF features moves out of view of the camera, but as mentioned before the bundle adjustment algorithm is robust enough to missing data to allow for this to happen.

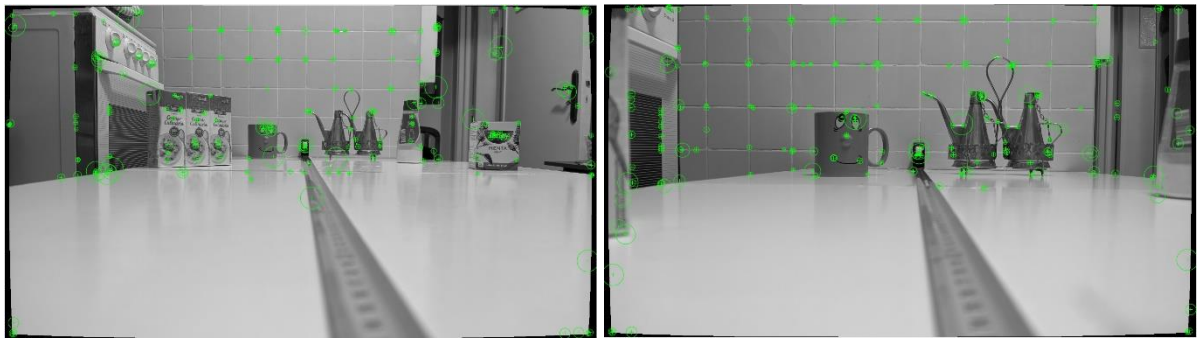

Figure 14 – Initial and final image of the dataset for experiment 1

Running the code from the example with only the camera intrinsic matrix, the image set, and the ground truth being changed, the result for the tracking is shown in Figure 15.

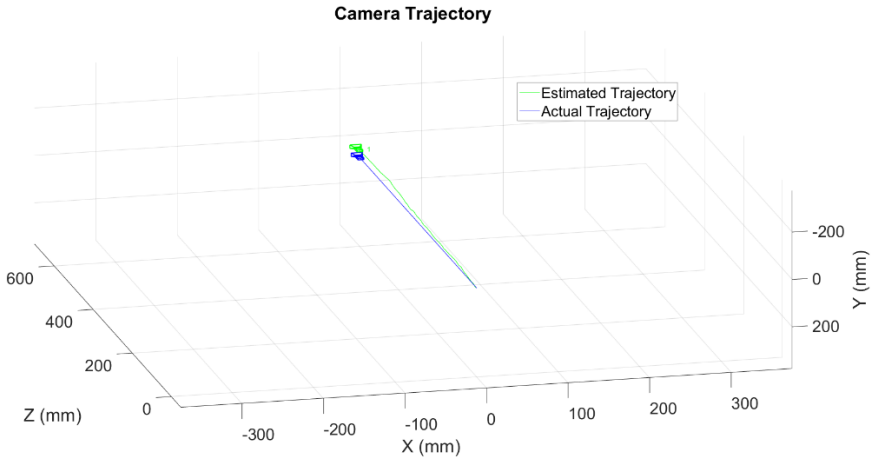

Figure 15 – Real and estimated camera trajectory for experiment 1

To make analysis of the accuracy of the estimation easier, the estimation errors on each coordinate were calculated, as well as the Euclidean distance between estimated and real position at each view. An example these results is shown in Figure 16.

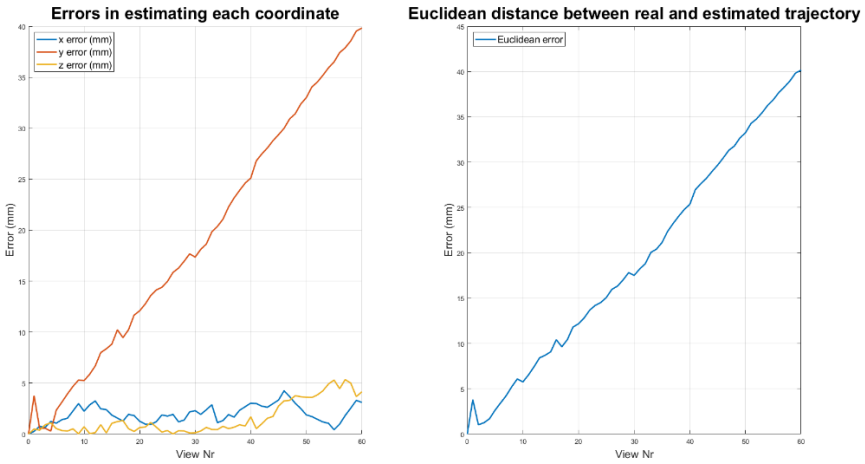

Figure 16 – Estimation errors in each coordinate and Euclidean distance for experiment 1

The Euclidian distance between real and estimated trajectory in Figure 16 is a measure of the difference between the two trajectories and, therefore, can be interpreted as an error or the norm of the difference described. This value is, roughly, equal to 40 mm and the relative error is 6.67%, since the motion of the camera is linear and the final distance equal to 600 mm.

The camera can approximate the real x and z coordinates to a high precision, with an estimation error of less than 5 pixels along the whole translation. However, a notable drift from the real trajectory is seen in the y coordinate, meaning the algorithm estimates that the camera moves up during the sequence. This could be caused by a combination of multiple factors: error in the calibration, due to an autofocus feature on the smartphone, a slight inclination in the table section used for the movement (the objects and the camera were on different sections of an articulated table, as illustrated in **Error! Reference source not found.**), human error causing involuntary rotation of the camera upwards during the movement and finally the estimation error inherent to the algorithm. If the main cause of error is the table, the estimated trajectory may be the more accurate one, and the error may be in the ground truth data.

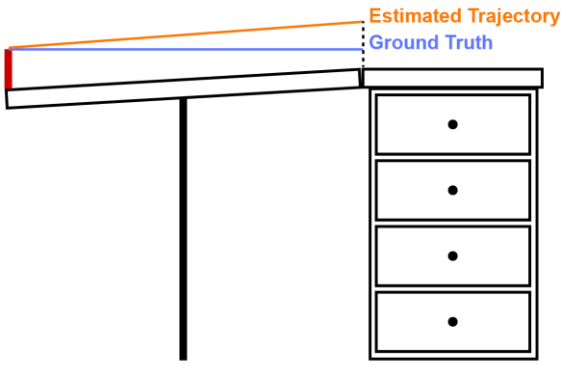

Figure 17 - Illustration of the table used for the experimentation

Some key hyperparameters were identified to be changed to try and improve the algorithm:

- Number of SURF points used for matching.
- Threshold for detecting SURF points.
- Number of views used in the second step (global bundle adjustment).
- Number of views between windowed bundle adjustments in the third step.
- Size of the window for windowed bundle adjustment.

In the MATLAB code, the user can either change these parameters to their liking or choose default values.

After this identification, these values were changed to evaluate if there is relation between the values of each parameter and the error. After some tests, it was possible to verify that, except for the SURF Threshold, all the other parameters remained approximately the same. The threshold for detecting SURF works better in a range of 200 to 600.

One problem noted with the application of the algorithm is the variability of the results every time the trajectory is estimated, which is caused by the randomness associated with the selection of SURF points and generation of the camera model using MSAC, which may lead to convergence to a suboptimal solution in the bundle adjustment optimization procedure.

4.4. Experiment 2: the “Room Scene dataset”

The second experiment was executed mostly for two reasons: the first was to test if the algorithm was robust to a change of movement direction, lighting conditions and background and the second to validate that the largest cause of error in experiment 1 was the slope of the table. This time, the measuring tape was placed perpendicular to the to the direction the camera points in (positive z axis), and the phone was moved to the right (in the positive x direction) in increments of 1 cm, with x varying from 0 cm to 50 cm, for a total of 51 captured images. The initial and final views of the dataset are shown in

Figure 18 – Initial and final views of the second experiment

Once again running the code to perform the second experiment, the estimated trajectory can be obtained, as well as the errors and the difference of Euclidian distance.

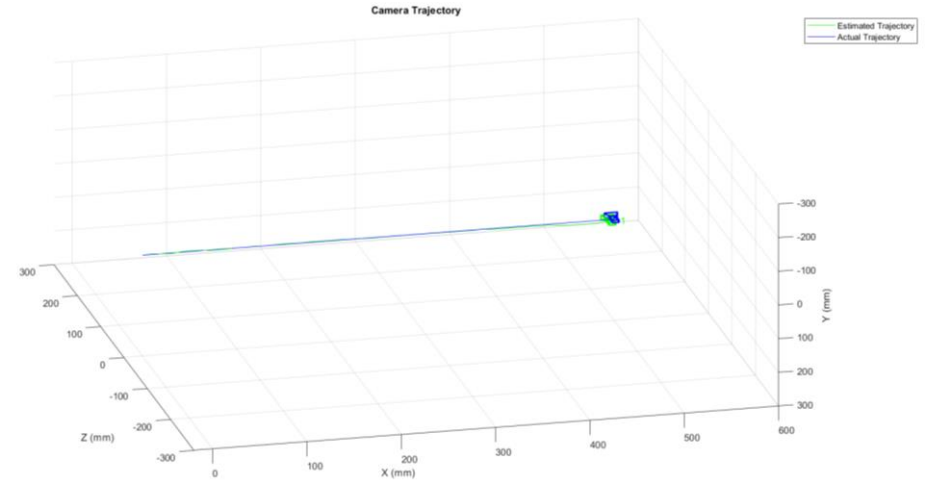

Figure 19 - Real and estimated camera trajectory for experiment 2

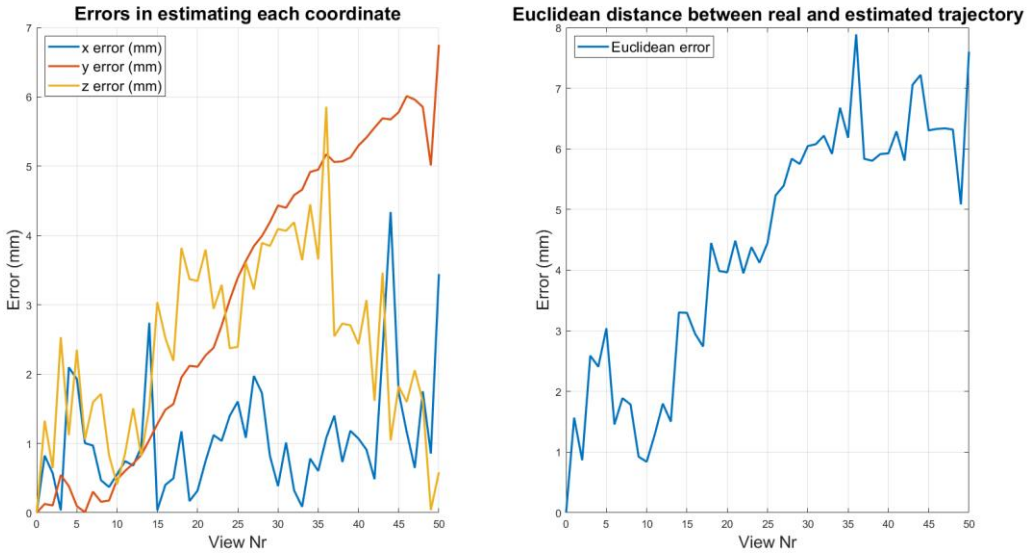

Figure 20 - Estimation errors in each coordinate and Euclidean distance for experiment 2

From Figure 20 it is clear that the algorithm performs better, with the absolute errors at each one of the axes being less than 7 mm along the whole path. The Euclidian distance between real and estimated trajectory is now, at the final point, equal to 7.60 mm which results in a relative error of 1.52%, lower than experiment 1, recalling that the total distance in direction x supposed to be made was 500 mm.

In this experiment, particular care was taken to guarantee that the motion of the camera could be made in a flat surface without any slope (approximately). Observing figure 19 and 20 it is possible to conclude that, in fact, the slope of the table was one of the causes of the main error in experiment 1, since the error associated with the translation is y, at the final point, is much lower (around 80%). Even though this is true, the absolute error in this coordinate is still the greatest one, but the causes are also the same as described in 4.2.

5. Program Description and Tutorial

5.1. Program description and guide

The algorithms used for the experimental procedure were written in MATLAB 2020b. Two sets of images representing one image sequence each are available for download, since they could not be included in the delivery folder due to size constraints. Folders “kitchenscene” and “roomscene” used for experiments 1 and 2, respectively, are available in the following link:

https://drive.google.com/drive/folders/1g_0ykAX_EsZJTUuElsp2mnfoykFf3fH3?usp=sharing

After downloading, the folders should be transferred to the MATLAB working directory. No renaming of folders or images is necessary or recommended since the program is pre-configured to be able to detect them automatically.

The script that should be called to access the programs functionalities is in file “MainScript.m.” Here is implemented a command line menu that allows for selection of the various algorithms for image segmentation and processing.

The menu presents the user with 5 options:

- Option A – Opens the default MATLAB example about Monocular Visual Odometry, using the “New Tsukuba Stereo Dataset.”
- Option B – Fine-tune the hyperparameters of the algorithm.
- Option C – Apply an adapted version of the algorithm to experiment 1, using the “Kitchen Scene” dataset.
- Option D – Apply an adapted version of the algorithm to experiment 2, using the “Room Scene” dataset.
- Option E – Exit the program.

5.2. Program flowchart

Figure 21 shows the flowchart of the main menu, with all the options that the user may select. The user may manually select various hyperparameters and choose what scene to load and apply the Monocular Visual Odometry.

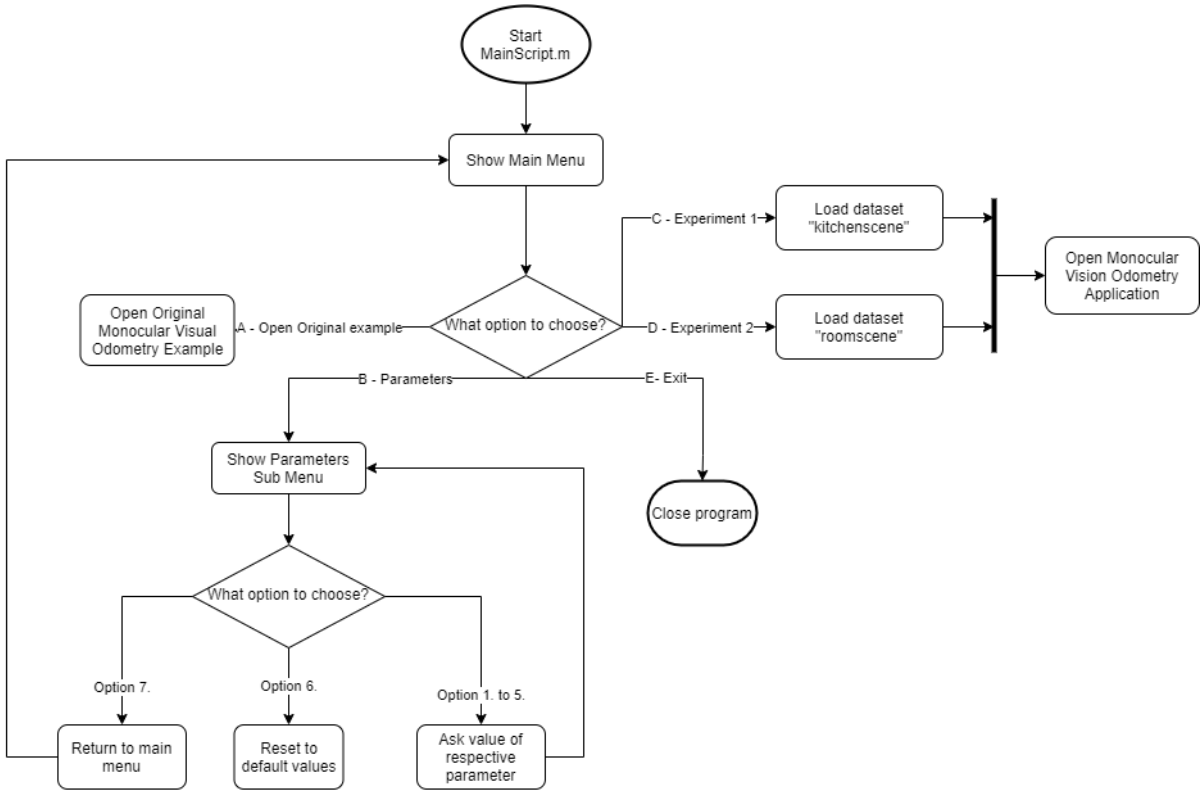

Figure 21 - Flowchart of the main script of the program

5.3. Monocular Visual Odometry Application flowchart

Figure 22 contains a flowchart describing the adaptation of the provided Monocular Visual Odometry Example used to perform experiments 1 and 2.

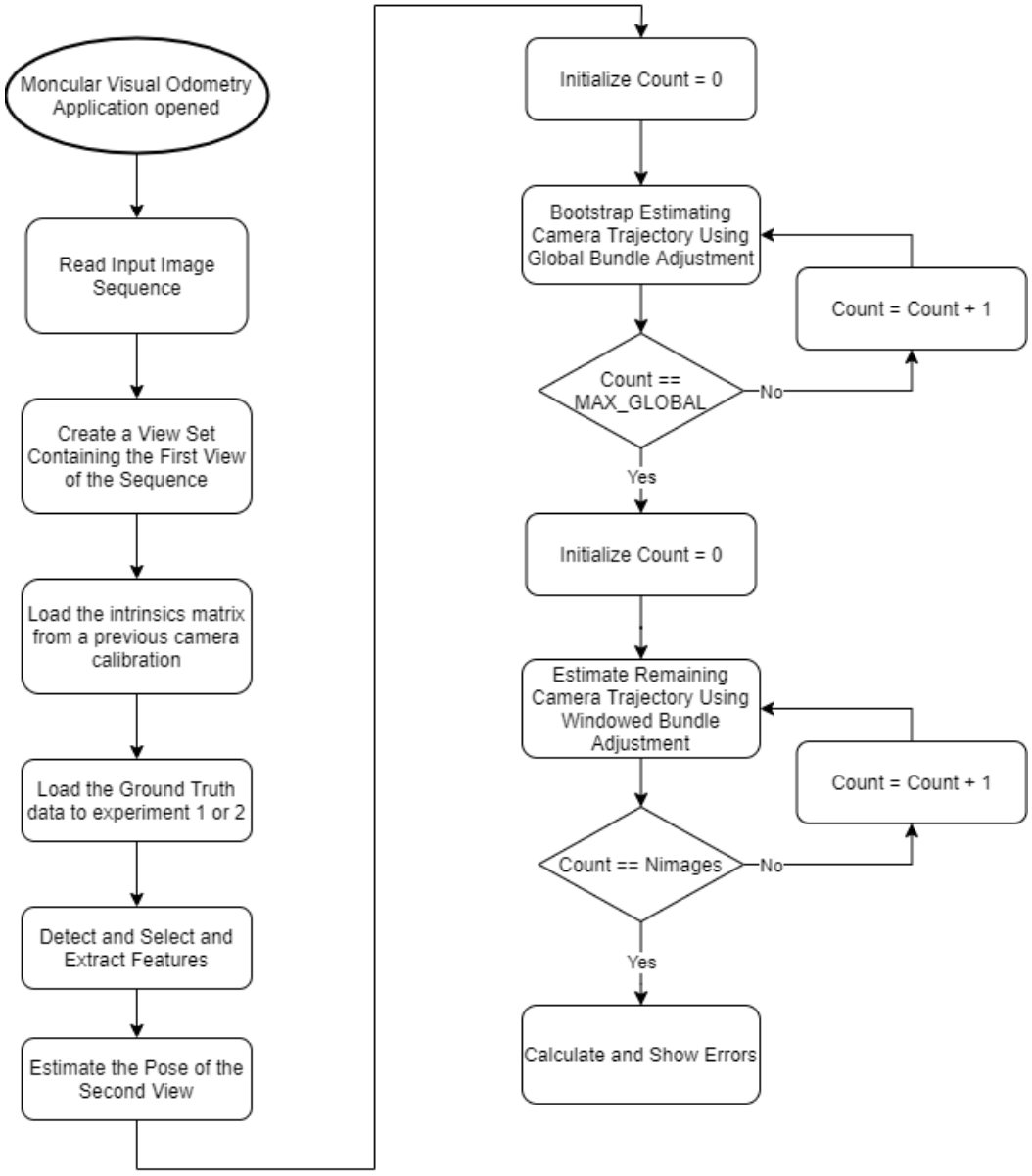

Figure 22 - Flowchart of the Monocular Visual Odometry Algorithm Application

6. Conclusion

Monocular Visual Odometry was able to correctly track the trajectory of the camera in both experiments, showing its usefulness for situations where only one camera is available, but there is location information from an external sensor. It even correctly identified an issue with the ground truth in experiment one, by recognizing the slope of the table causing the camera to move up.

One further challenge that was addressed by this group was trying to substitute the scale factor calculated using the tape measure by the dimension of a known object that is in view of the camera during the movement.

For this the following steps were implemented:

1. Binarize the image, separating the object from the remaining image.
2. Calculate a characteristic, known dimension, like the diameter of a circle and the side of a square (both were tried) in pixel units.
3. Calculate the scale factor as the quotient between the dimension in real metric units and the dimension in pixel units.

This method, however, did not work, making the optimization algorithm diverge and yield impossible values when tried. This is because the algorithm is expecting a scale factor between normalized (unitary) camera displacements and real displacements, and not just by a conversion unit. Applying the scale factor to the 3D point cloud also caused the algorithm to diverge, so further work would be necessary to develop an algorithm that can estimate the scale of the relative position between views without a need for a sensor that measures the magnitude of the displacement and corrects it.

7. References

[1] "Monocular Visual Odometry." .

[2] S. MARTULL, M. PERIS, and K. FUKUI, "Realistic CG Stereo Image Dataset with Ground Truth Disparity Maps," *Sci. Program.*, vol. 111, no. 431, pp. 117–118, 2012.

[3] M. Peris, S. Martull, A. Maki, Y. Ohkawa, and K. Fukui, "Towards a simulation driven stereo vision system," *Proc. - Int. Conf. Pattern Recognit.*, no. Icpr, pp. 1038–1042, 2012.

[4] Z. Kukelova, M. Bujnak, and T. Pajdla, "Polynomial eigenvalue solutions to the 5-pt and 6-pt relative pose problems," *BMVC 2008 - Proc. Br. Mach. Vis. Conf. 2008*, 2008, doi: 10.5244/C.22.56.

[5] D. Nistér, "An efficient solution to the five-point relative pose problem," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 6, pp. 756–770, 2004, doi: 10.1109/TPAMI.2004.17.

[6] P. H. S. Torr and A. Zisserman, "MLESAC: A new robust estimator with application to estimating image geometry," *Comput. Vis. Image Underst.*, vol. 78, no. 1, pp. 138–156, 2000, doi: 10.1006/cviu.1999.0832.

[7] X. S. Gao, X. R. Hou, J. Tang, and H. F. Cheng, "Complete solution classification for the perspective-three-point problem," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 8, pp. 930–943, 2003, doi: 10.1109/TPAMI.2003.1217599.

[8] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, vol. 16, no. 2. Cambridge University Press, 2004.

[9] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, Jun. 2008, doi: 10.1016/j.cviu.2007.09.014.

[10] R. I. Hartley and P. Sturm, "Triangulation," *Comput. Vis. Image Underst.*, vol. 68, no. 2, pp. 146–157, 1997, doi: 10.1006/cviu.1997.0547.

[11] M. I. A. Lourakis and A. A. Argyros, "SBA: A software package for generic sparse bundle adjustment," *ACM Trans. Math. Softw.*, vol. 36, no. 1, 2009, doi: 10.1145/1486525.1486527.

[12] "Evaluating the Accuracy of Single Camera Calibration - MATLAB & Simulink." https://www.mathworks.com/help/vision/ug/evaluating-the-accuracy-of-single-camera-calibration.html.