AUDIOVISUAL SYSTEMS ENGINEERING

Degree Thesis:

3D Pose Estimation of Visual Markers

A Degree Thesis Submitted to ETSETB (UPC), and fulfilled at TU Wien.

Submitted by:
Antonio Badal Regàs

Supervised by:
Markus Bader (TU Wien)
Albert Oliveras (UPC)
Abstract

This document is the final thesis of my degree in audiovisual systems engineering. It has been fulfilled at TU Wien, Vienna, Austria.

The aim of this project is to understand, analyze, but also improve a computer vision software based on ARToolKitPlus. With a camera, this software looks for a visual marker (ARTag) and locates it in the space, calculating the position and orientation of this marker in the three-dimensional space relative to the camera, from the two-dimensional space image. But when the marker is standing in front of the camera, in some positions, an apparently random flickering appears. This work presents the analyzes of that flickering and approaches an overcome of this problem. This thesis will describe how the program detects the marker and computes its position, dissecting every part of it. It will show that the problem appears when computing the space transformation, because the program applies Newton’s iterative method for solving a system of simultaneous, nonlinear equations until it finds the minimum error. The point is that there is another local minimum that is not the right solution, which corresponds to the position of the marker with central symmetry with the camera. The proposed solution is a combination of different analyzed solutions. This proposal firstly reduces the iteration space to avoid finding the second fake minimum, and secondly chooses the minimum with lower error rate, dismissing it if the pose is zero.
Contents

1 Introduction 4
   1.1 Project Context and Related Works 4
      1.1.1 Marker-Based Vision Systems 4
      1.1.2 Space Transformation 5
   1.2 Goals and Document Structure 6

2 Problem Presentation 8

3 Software description and possible issues 11
   3.1 Software description 11
      3.1.1 Camera Calibration 11
      3.1.2 Marker Detection 12
      3.1.3 Pose Transformation 14
      3.1.4 Pose Visualization 17
   3.2 Possible issues 17
      3.2.1 Camera Calibration 17
      3.2.2 Marker Detection 17
      3.2.3 Pose Transformation 17
      3.2.4 Pose visualization 19

4 Solution to the Problem 20
   4.1 Proceedings 20
   4.2 Proposed Solutions 20
      4.2.1 Change Sign 20
      4.2.2 Rotation 20
      4.2.3 Initialization in Front of the Camera 22
      4.2.4 No Error Thresholding 22
      4.2.5 Reducing Iteration Space 23
      4.2.6 Combination of Solutions 24
   4.3 Solution Implementation 24
      4.3.1 No Error Thresholding 25
      4.3.2 Reducing Iteration Space 26
      4.3.3 Execution of the Code 27

5 Time Plan 28

6 Budget 30

7 Conclusions 31
List of Figures

1.1 Transformation from the three-dimensional space to two-dimensional. 5
2.1 Input and output of the first frame. 8
2.2 Input and output of the second frame. 8
2.3 Position and cumulated number of pose jumps. 9
2.4 Zoom of zone where pose jump happens. 10
2.5 Example of central symmetry. 10
3.1 Marker detection program’s diagram. 11
3.2 Labeling procedure. 12
3.3 Contour detection procedure. 13
3.4 Vertex location procedure. 14
3.5 Different solutions that can return an iteration algorithm. 19
4.1 Position and cumulated number of pose jumps. 21
4.2 Marker’s position with Euler rotation. 21
4.3 Output image of the marker initializing the iteration in front of the camera. 22
4.4 Marker position without error threshold. 23
4.5 The iteration space is reduced from 150° to 75°. 23
4.6 Marker position reducing the iteration space. 24
4.7 Marker position without error threshold and reducing the iteration space. 25
3.1 Project plan’s Gantt Chart. 28
3.2 Critical review’s Gantt Chart. 28
5.1 Project costs. 30
5.2 Final Gantt Chart. Task 3 (Implement the Solution) and 4 (Implement the Solution) are longer than project plan’s and critical review’s prevision. 29

List of Tables

2.1 Pose output. 9
3.1 Get information input and output. 14
3.2 Output of the marker detection between two frames where the pose jump happens. 18
4.1 Position and error for the two initializations for the two frames where the pose jump happens. 22
6.1 Project costs. 30
1 Introduction

This project examines minutely how a software detects visual markers through a camera, and how it extracts its location in the three-dimensional space from the two-dimensional image captured by the camera. Before breaking down how is this done, some concepts need to be introduced, and a contextualization of the work needs to be presented. This first section give an introduction of these, but also will talk about the document structure and the main goals of this thesis.

1.1 Project Context and Related Works

1.1.1 Marker-Based Vision Systems.

There is a wide variety of techniques to detect and identify markers in computer vision. The most important properties of Marker-Based Vision (MBV) systems are explained by Andrew C. Rice, Robert K. Harle, Alastair R. Beresford in [1]. Here they present different simulations with some MBV systems and markers, analyzing which one works better for each application. To do it they use Cantag, an useful open source toolkit. Our system works on ARToolKitPlus, a MBV system which best point is, as explained in this paper, the positioning accuracy due to good thresholding and labeling. This is the main reason why our system is based on ARToolKitPlus.

ATag

Johannes Köhler, Alain Pagani, and Didier Stricker explain in [2] some other detection systems and the differences between the different markers. This paper introduces ARTag, a bitonal square markers used by our system. It explains that ARTag based systems are not the best performers regarding marker occlusion and minimal detectable pattern size, and it refers to circular markers as a good solution to his problem. The advantages and disadvantages of these type of markers are analyzed precisely in [3]. Also Mark Fiala analyzes ARTag markers and ARTag systems in [6]. These fiducial marker systems are tested and compared to other systems, mainly ARToolKit. Mark Fiala concludes that ARTag manage to achieve a low false negative but having a vastly lower false positives rates and a very low inter-marker confusion rate. This makes stronger the decision of taking ARToolKitPlus based on ARTag in our system.

ARToolKitPlus.

An analysis of the principles of ARToolKit has been done by Martin Hirzer in [4]. ARToolKit is a computer tracking library for creation of strong augmented reality applications that overlay virtual imagery on the real world. ARToolKitPlus is an improvement of this software. The most important improvements are the automatic thresholding and labeling commented before and a robuster...
planar pose algorithm that reduces jitter while tracking the pose of the marker.

Figure 1.1: Transformation from the three-dimensional space to two-dimensional. The point \((x, y, z)\) in the three-dimensional space is a two-dimensional point \((u, v)\) in the image of the camera.

1.1.2 Space Transformation

As this paper will show, there's another important point for the MBV systems, the space transformation. The input image is a two-dimensional space, while this systems usually need to know the three-dimensional position of the marker. Richard Hartley and Andrew Zisserman wrote a book ([7]) that gives a detailed explanation on different fields about space and projective geometry, from one or multiple points of view. The transformation of a single two-dimensional point of view to a three-dimensional space (Figure 1.1) is explained in the first chapters. Also Chien-Ping Lu analyzes this transformation in [8]. This transformation has two components, a rotation and a translation. Thus, a point \(p\) in scene coordinates is transformed to camera coordinates \(v\) by:

\[
v_i = \mathcal{R}p_i + T_i,
\]

with,

\[
\mathcal{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}, \quad T = \begin{pmatrix} T_x \\ T_y \\ T_z \end{pmatrix}
\]

where \(\mathcal{R}\) is the rotation matrix with rotation vectors \(r_i = (r_{i1}, r_{i2}, r_{i3})\), and \(T\) is the translation vector. To optimize this transformation it is necessary to apply an iterative algorithm.

Gerald Schweighofer and Alex Pinz analyze in their paper [9] the problems
of this transformation. In many tracking applications pose jumps can be observed. The iterative algorithms look for a minimum error rate. As it is shown in the mentioned paper, a fake local minimum appears. It is said that it appears because of the three point projective transformation. Some solutions can be applied to solve this problem, and they propose to find a way to improve the initialization of the algorithm, solving the flickering problem: First the program finds a solution (we can not know if it is the good one), and then, as we know the projection, we can find the other solution. We compare both solutions and the one that has the minimum error rate is assumed to be the good one.

**Quaternion transformation.**
Our system has another particularity that has to be analyzed. The rotation is done with a quaternion transformation. The quaternion space is an extension of imaginary space, where the vectors have this form:

\[ Q = q_1 + iq_2 + jq_3 + kq_4 \]  

(1.3)

with

\[ i^2 = -1, j^2 = -1, k^2 = -1 \]  

(1.4)

Where \( q_2, q_3 \) and \( q_4 \) define the rotation axis, and \( q_1 \) the rotation angle. The advantages of using quaternion rather to homogeneous rotation are that quaternion has a smaller size and it is less ambiguous, but the same rotations can be done. The unique problem of the quaternion transformation is that its multiplication is non-commutative:

\[ ij = k, ji = -k \]  

(1.5a)

\[ jk = i, kj = -i \]  

(1.5b)

\[ ki = j, ik = -j \]  

(1.5c)

More specific properties of quaternion transformation can be found in [10].

1.2 Goals and Document Structure

**Goals.**
The main goals of this project are:

- Be able to obtain useful and applicable information about a new topic, in particular, the operation of how to identify a marker through the camera.

- Examine the problems and bugs of the software, analyze a problem, find the best way to solve it and implement a reasonable and effective solution.

- Test, criticize and improve this solution.

- Learn to write a good technical paper in a foreigner language.
Document structure.
From section 1 to section 6 the structure of this document goes from general to specific concepts. First of all, in section 1.1 the frame of the project is illustrated, introducing some important points of this undertaking. There, some of the research that has been done is announced and summarized, presenting some related works. Then, in section 2 the pose jump problem is presented, characterized, and dissected. In section 3 a precise dismemberment about how the software works is done, analyzing, describing and elucidating all the stages of the marker detection. This section also includes possible causes of the pose jump problem in each phrase of the software, explaining if these are the real reasons of the pose jump problem or not, and finally exposing where and why did the pose jump appear. In section 4 a solution for the problem will be proposed, and the implementation of it will be exposed in section 5. The first time plan will be shown and compared with the final time plan followed in section 6, and an approximate budged will be done in section 7. Finally, in section 8 some conclusions will be presented, and also some ways to continue the work in the future.
2 Problem Presentation

In this section the pose jump problem will be presented. To examine it minutely, a video with some jumps has been recorded. Then, two frames where the pose jump happens have been stored.

![Figure 2.1: Input and output of the first frame.](image1)

![Figure 2.2: Input and output of the second frame.](image2)

In Figure 2.1 and Figure 2.2 the pose jump can be clearly seen. It is observable that with two frames that are almost the same (a), the program obtains two marker’s positions that are too different (b).

Table 2.1 helps to compare the output pose for both frames. The problem seems to be that for some reason, the algorithm changes the pose’s sign, because the absolute values of the position for frame0008.jpg and frame0009.jpg are almost the same, but the signs are all opposite. Different values for the pose cause that orientation values are also so different, in this case they are crossed.
Annex 3D Pose Estimation of Visual Markers

Table 2.1: Table showing the output pose between two frames where the pose jump happens.

<table>
<thead>
<tr>
<th>ARMarker</th>
</tr>
</thead>
<tbody>
<tr>
<td>u10</td>
</tr>
<tr>
<td>Position</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td>Z</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Orientation</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td>Z</td>
</tr>
<tr>
<td>W</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ARMarker</th>
</tr>
</thead>
<tbody>
<tr>
<td>u20</td>
</tr>
<tr>
<td>Position</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td>Z</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Orientation</td>
</tr>
<tr>
<td>X</td>
</tr>
<tr>
<td>Y</td>
</tr>
<tr>
<td>Z</td>
</tr>
<tr>
<td>W</td>
</tr>
</tbody>
</table>

A scene of around 300 frames has been recorded to analyze the different situations with the same input. The position of the marker in this situation and the cumulated error due to the pose jump can be observed in Figure [2.3]. This graph shows quiet good the situation. In some positions the values of the 3D coordinates of the marker are inverted.

Figure 2.3: This graph shows the cumulated number of pose jump, and also the 3D position of the marker. It shows clearly where the pose jump happens, around frames 50, 225, 240 and between 270 and 280.

If we zoom the position around the marker 50 (Figure [2.4]), we can observe that there is two kind of problems. First of all, at frame 39 the pose is (0, 0, 0). In the other jumps it is observable that the pose changes it’s sign. This means
that we have a central symmetry with origin at \( O(0,0,0) \) (Figure 2.5), that is the position of the camera.

Figure 2.4: Zoom of zone where pose jump happens.

Figure 2.5: Example of central symmetry, where \( V_i \) is the original view and \( V'_i \) is the symmetric image.
3 Software description and possible issues

This section will cut up the functioning of the software, trying to specify possible causes of the pose jump.

3.1 Software description

The software can be split in three stages:

- Camera calibration.
- Marker detection.
- Pose transformation.
- Pose visualization.

This process can be seen in this diagram:

![Diagram of marker detection program's diagram](http://wiki.ros.org/)

Figure 3.1: Marker detection program’s diagram.

3.1.1 Camera Calibration

The first thing that needs to be done is to set up the camera information parameters. To do it we calibrate the camera following the tutorial that can be found at ROS webpage[^1]. As we can see in the block diagram, the calibration only needs to be done once, and the camera values are stored and used every time that the program is executed.

[^1]: http://wiki.ros.org/
3.1.2 Marker Detection

This stage just analyzes the input image and tries to extract some information of the markers, like the area, the position of the vertexes, and others. It follows these steps to find out where the marker is:

- Thresholding.
- Labeling.
- Contour detection.
- Vertex location.
- Get information and verification.

Thresholding
All the markers used are black and white, so we don’t need the color information. For each pixel, we apply \( r + g + b < \text{threshold} \times 3 \). All the pixels that satisfy this rule are suitable for the next stage while the rest are discarded.

Labeling
In this stage the program looks for groups of connected pixels. The Figure 3.2 can help to understand this procedure. For each pixel that suited the thresh-

![Figure 3.2: Position of the eight closest pixels relative to the center one.](image)

olding, the program follows the next steps, starting at the top left pixel of the image, and going from left to right, and from top to bottom. The upcoming conditions followed in this order; if one of them is true, the program applies it stopping the procedure for this pixel and jumping to the next pixel. If not, it jumps to the next condition.

1. First it looks at the top pixel. If it is labeled, the same label is applied to the current pixel
2. If the top right pixel is labeled, then we check if left or top left pixels are labeled. If one of them is labeled, then both labels are merged.
3. Then the top left pixel is checked and if it is labeled, the same label is applied to the current pixel.

4. Finally we check the left pixel and if it is labeled, the same label is applied to the current pixel.

If all these conditions are false, the program applies a new label to the pixel and it starts the same procedure with the right pixel. In the contours of the image, the pixels that are outside the image are considered as non-labeled pixels.

**Contour detection**

To detect the contour of each zone it takes the pixel at the top of the current zone to start (Figure 3.3 (a)). This pixel is considered as the first point of the contour, and the program follows the next steps:

1. It checks all the pixels starting from the one above, going in clockwise direction (Figure 3.3 (b)).

2. When it finds another pixel of the zone, it is stored like the next pixel of the contour.

3. The program goes back to point 1 but centering this new pixel and starting from the previous one (Figure 3.3 (c)). If this new pixel is the first one, the program stops, having found all the contour of the label.

![Figure 3.3: This three images show the contour detection procedure.](image)

**Vertex location**

The first vertex \(v_1\) is assumed to be the starting point of contour detection stage, and the second vertex \(v_2\) is the furthest contour point from the first vertex. The other two pixels are more difficult to find. Lets define \(R_1\) as the line between \(v_1\) and \(v_2\). Then \(v_3\) would be the furthest contour point from \(R_1\) on one side of the zone, and \(v_4\) is the furthest point from \(R_1\) on the other side (Figure 3.4 (a)). If \(R_1\) is on the contour, then \(R_2\) is defined as the line between \(v_1\) and \(v_3\), and \(v_4\) is the furthest contour point from \(R_2\) (Figure 3.4 (b)).

![Figure 3.4: This three images show the vertex location procedure.](image)
Get information and verification
This stage has two purposes. First of all, to transform the variables as we can see in Table 3.1. This stage has another important role. This transformation also makes some verifications to ensure that the zone is a marker; it checks if the zone is a quadrilateral, if there is any zone is inside another one, if the pattern inside it corresponds to an ARTag marker\(^3\) and other verifications.

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>area</td>
<td>area</td>
</tr>
<tr>
<td>position</td>
<td>id</td>
</tr>
<tr>
<td>coordinate number</td>
<td>dir</td>
</tr>
<tr>
<td>coordinate X</td>
<td>cf</td>
</tr>
<tr>
<td>coordinate Y</td>
<td>position</td>
</tr>
<tr>
<td>vertex</td>
<td>vertex</td>
</tr>
<tr>
<td>lines</td>
<td>lines</td>
</tr>
</tbody>
</table>

Table 3.1: Transformation of variables, input compared with output

3.1.3 Pose Transformation
In this stage, a transformation is applied to the position to extract the three-dimensional position from the two-dimensional data that the previous stage has extracted from the input image. It will be shown that the program follows an iteration process to get to the three-dimensional position of the marker.

As was said in the introduction (1.1), the transformation of a 3D point \( \mathbf{P} \),

\(^3\)Each ARTag has a different pattern. The output variable ID stores which marker is it. If the zone does not correspond to a marker, then ID=-1.
Annex

3D Pose Estimation of Visual Markers

from a model point \( \mathbf{P}_i^m \) is defined as:

\[
\mathbf{P}_i = \mathbf{R} \mathbf{P}_i^m + \mathbf{T},
\]

where \( \mathbf{R} \) is the rotation of entries \( r_{ij} \) and \( \mathbf{T} = [T_x, T_y, T_z]^T \) is the translation. To calculate the transformation we define the 2D point as:

\[
[x_i, y_i] = [fX_i, fY_i] \quad (3.7)
\]

If we plug equations (3.6) and (3.7), we get these two equations:

\[
x_i = f \frac{r_{11}X_i^m + r_{12}Y_i^m + r_{13}Z_i^m + T_1}{r_{31}X_i^m + r_{32}Y_i^m + r_{33}Z_i^m + T_3}
\]

\[
y_i = f \frac{r_{21}X_i^m + r_{22}Y_i^m + r_{23}Z_i^m + T_2}{r_{31}X_i^m + r_{32}Y_i^m + r_{33}Z_i^m + T_3}
\]

This system (3.8 and 3.9) has six unknowns, as \( \mathbf{R} \) depends only on three angles, \( \alpha, \beta \) and \( \gamma \). In our case, this rotation is defined as:

\[
r_{11} = \cos\alpha\cos\beta \cos\gamma + \sin\alpha\sin\beta \sin\gamma - \sin\alpha\cos\beta \sin\gamma
\]

\[
r_{12} = -\cos\alpha\cos\beta \sin\gamma - \sin\alpha\sin\beta \sin\gamma + \sin\alpha\cos\beta \cos\gamma
\]

\[
r_{13} = \cos\alpha \sin\beta
\]

\[
r_{21} = \sin\alpha\cos\beta \cos\gamma - \sin\alpha\sin\beta \sin\gamma + \cos\alpha\cos\beta \sin\gamma
\]

\[
r_{22} = -\sin\alpha\cos\beta \sin\gamma + \sin\alpha\sin\beta \cos\gamma + \sin\alpha\cos\beta \cos\gamma
\]

\[
r_{23} = \sin\alpha \sin\beta
\]

\[
r_{31} = -\cos\alpha \sin\beta \cos\gamma - \sin\alpha \sin\beta \sin\gamma
\]

\[
r_{32} = \cos\alpha \sin\beta \sin\gamma - \sin\alpha \sin\beta \cos\gamma
\]

\[
r_{33} = \cos\beta
\]

To apply this rotation is equivalent to apply two different rotations, one after the other:

\[
\mathbf{R} = \mathbf{R}_{\alpha\beta} \mathbf{R}_{\gamma},
\]

where

\[
\mathbf{R}_{\gamma} = \begin{pmatrix}
\cos\gamma & \sin\gamma & 0 \\
-\sin\gamma & \cos\gamma & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

and

\[
\mathbf{R}_{\alpha\beta} = \begin{pmatrix}
\cos^2\alpha\cos\beta + \sin^2\alpha \sin\gamma & \sin\alpha\cos\beta - \sin\alpha\cos\gamma & \cos\alpha \sin\beta \\
\sin\alpha\cos\beta - \sin\alpha\cos\gamma & \sin^2\alpha\cos\beta + \cos^2\alpha \sin\gamma & \sin\alpha \cos\beta \\
-\cos\alpha \sin\beta & -\sin\alpha \sin\beta & \cos\beta
\end{pmatrix}
\]

This means that this rotation can be understood as two rotations. First \( \mathbf{R}_{\gamma} \) is applied, and then \( \mathbf{R}_{\alpha\beta} \). \( \mathbf{R}_\gamma \) is a rotation of \( \gamma \) through \( Z \) axis. In the other hand, \( \mathbf{R}_{\alpha\beta} \) is a rotation that depends on both angles \( (\alpha \) and \( \beta \)) at the same
time. This rotation can not be split in two, but it can be understand as two rotations around Y and Z axis. The rotation is not separable because α rotation depends on β and vice versa.

Now that the transformation has been defined, the iteration procedure can be explained. Following Newton’s method, this iteration starts off with an initialization for $R$ and $T$ (lets call them $\bar{R}$ and $\bar{T}$). For each of the four vertexes $p_i$, a model vertex point is defined as $P^{m}_{i} = (center_x \pm width, center_y \pm width, 0)^T$.

We can define the residuals as:

$$\delta x_i = x_i(\bar{R}, \bar{T}) - \bar{x}_i \quad (3.10a)$$
$$\delta y_i = y_i(\bar{R}, \bar{T}) - \bar{y}_i \quad (3.10b)$$

Calculating the partial derivatives of $x_i$ and $y_i$ the following pair of linear equations can be found, where $\phi_1 = \alpha$, $\phi_2 = \beta$, $\phi_3 = \gamma$, and $\Delta T_j$ and $\Delta \phi_j$ are the unknown corrections of the translation and the rotation.

$$\sum_{j=1}^{3} \frac{\partial x_i}{\partial T_j} \Delta T_j + \frac{\partial x_i}{\partial \phi_j} \Delta \phi_j = \delta x_i \quad (3.11a)$$
$$\sum_{j=1}^{3} \frac{\partial y_i}{\partial T_j} \Delta T_j + \frac{\partial y_i}{\partial \phi_j} \Delta \phi_j = \delta y_i \quad (3.11b)$$

After defining the previous equations, the iteration algorithm can be defined. It follows these steps:

1. Use $\bar{R}$ and $\bar{T}$ to compute $P_i$ with (3.6) from $P^{m}_{i}$.
2. Apply (3.7) to $P_i$ to project it to the image plane.
3. Compute the residuals for each vertex with (3.10).
4. Solve the system (3.11) for the four vertexes and update the translation vector and the rotation matrix with the corrections $\Delta T_j$ and $\Delta \phi_j$.
5. If the residuals are sufficiently small the iteration ends. If not, it goes back to step 1 with the current estimations of $\bar{R}$ and $\bar{T}$.

For precisely information about this algorithm check [11].

The program starts with two different initializations. The first one uses the last values of $\bar{R}$ and $\bar{T}$, and the second one takes them from some values defined automatically during the calibration of the camera. In this case $\bar{R}$ and $\bar{T}$

---

$\text{width} = 0.5$, and $(center_x, center_y)$ is the position of the marker in 2D.

$\text{Updating the translation is straightforward: } T_{j+1} = T_j + \Delta T_j$, while the rotation matrix is updated computing the multiplication of the matrices: $R_{j+1} = R_j R_{\phi_1 \phi_2 \phi_3}$.
are defined according to the characteristics of the camera. If the first error is under a threshold, then it is assumed to be the good solution. If not, then the error is calculated with the second initialization and the one which has lower error rate is assumed to be the good one.

### 3.1.4 Pose Visualization

This is the last stage of the program. The program used to visualize the solution is RViz. This program helps to visualize the output of the program in a graphical way. As we can see at Figures [2.1](b) and [2.2](b), it shows the position and orientation of the marker in a three-dimensional space.

### 3.2 Possible issues

#### 3.2.1 Camera Calibration

Examining critically some papers about the camera calibration procedure, it came of that some issues appeared. By the way, these issues have been solved, so we have assumed that the pose jump problem did not occur here.

#### 3.2.2 Marker Detection

To check if the flickering problem happens here, the outputs of this stage for two frames where the pose jump appears have been analyzed. All the information of the markers extracted by the marker detection is stored in the variable called `marker_info`, which keeps all the information showed in Table [3.2](b) for each marker detected in the image.

We can observe in Table [3.2](b) that the difference between both frames is insignificant. The biggest difference value is 11.76%. If we calculate the average error is 2.24%, and the standard deviation is 3.36%. As said before, this stage just obtains some information about the marker from the image. Therefore, we can ensure that the pose jump does not happen in this stage, because this information about the marker in both frames is so similar.

#### 3.2.3 Pose Transformation

The pose transformation has two possible causes to the pose jump problem, the quaternion transformation or the iteration algorithm:

**Quaternion transformation.**

As was said at the introduction, our system uses quaternion rotation. This rotation has a property that could cause the change of sign of the pose; it is not
Table 3.2: This table shows the output of the marker detection stage between two frames where the pose jump happens. To simplify, only marker 20 values are exposed. It is observable that the outputs are almost the same for the two frames where the pose jump happens.

\[
\begin{align*}
ij &= k, ji = -k \\
jk &= i, kj = -i \\
ki &= j, ik = -j
\end{align*}
\]  

But after analyzing the outputs of the two frames where the pose jump happens, we can ensure that the quaternion rotation does not cause it, because the problem appeared before the quaternion transformation was applied.

**Iteration algorithm.**

The iteration method explained before could cause the pose jump. The iteration starts from a value for the transformations, and looks for an improve of it. It
was shown that from the starting point, it searches for the minimum error rate using derivatives direction. So, if the iteration finds a local minimum which is sufficiently small the algorithm would keep it as the solution, but it is not always the true. Figure 3.5 helps to understand why.

![Figure 3.5](image)

Figure 3.5: This image shows clearly that two different initializations for an iterative algorithm can return different solutions. In this case, initialization A returns a solution with lower error rate, while initialization B has another solution that is also under the minimum error threshold.

In this case it is obvious that the symmetric marker would have a local minimum, because it would have the same projection in the two-dimensional space but inverted.

### 3.2.4 Pose visualization.

It was assumed that the problem was not caused in the pose visualization, because this application only helps to visualize the position in a three-dimensional space, taking the output of the previous stages. Furthermore, the output shown at Table 2.1, where the pose jump can be observed, was printed without the help of RViz, so it’s not possible that the problem is caused in this stage.
4 Solution to the Problem

4.1 Proceedings

The procedure followed to find out a solution has been:

1. First, a proposal of a solution is done, discussing and examining minutely all the pros and cons of it.

2. Then, this solution is implemented and adapted to the program.

3. Finally the results of this solution are analyzed and criticized. If the results of this solution are not convincing enough and it needs to be discarded, then we go back to point 1. Also the implementation could be improved. In this situation we would go back to point.

It has to be said that some lines had to be added to the code to extract the marker’s position. The problem needs to be run at real time, and other kind of executions like debugging can return non-consistent solutions. This is an important point to have in mind.

4.2 Proposed Solutions

In section 3 was shown that the pose jump problem was caused because of the iteration algorithm. In this section a solution to the presented problem will be proposed. In this section the pose output of some proposed solutions will be compared with figure 2.3 that can be seen here in figure 4.1

4.2.1 Change Sign

The first proposed solution is simple but efficient. It is to check the sign of the Z component of the three-dimensional point. If $Z < 0$ it means that we have a pose jump, so the signs of $r_{ij}$ (for $i = 1, 2, 3$ and $j = 1, 2, 6$) and all the components of the translation are inverted. Although this solution works, it will be dismissed because it is a patch, and it is always better to get to the root of the problem.

4.2.2 Rotation

After analyzing the rotation $\mathcal{R} = \mathcal{R}_\alpha \mathcal{R}_\beta \mathcal{R}_\gamma$, it was proposed to change it for a separable rotation with euler angles:

$$\mathcal{R} = \mathcal{R}_\alpha \mathcal{R}_\beta \mathcal{R}_\gamma$$

Where $\mathcal{R}_\gamma$ is a rotation of $\gamma$ radians around $Z$ axis, $\mathcal{R}_\beta$ is a rotation of $\beta$ radians around $Y$ axis, and $\mathcal{R}_\alpha$ is a rotation of $\alpha$ radians around $Z$ axis. If we change the rotation for this one we get the output of the Figure 4.2. Although the

\[^6\text{Looking at Figure 2.5 is observable that invert the X and Y components of the rotation vectors } r, \text{ we get the symmetric object. If } Z \text{ is also inverted, the new object is not symmetric.}\]
Figure 4.1: This graph shows the cumulated number of pose jump. This is figure 2.3 copied here to facility the comparison of the reader.

Figure 4.2: Markers’s position with Euler rotation. As can be seen in the graph, this rotation returns a lot of noise.

number of pose jumps is lower than the case with the other rotation, we can not keep this solution because a lot of noise appeared, as can be seen in Figure 4.2. This is something that could cause a lot of trouble depending on the application.
of the problem, so this solution is discarded.

### 4.2.3 Initialization in Front of the Camera

Another possible solution to avoid finding the minimum that seems to be at the back of the camera is to initialize the iteration in front of the camera. Figure 4.3 shows that this is not a good solution. If we initialize 1 unit in front of the camera the pose solution is always 1 unit behind the marker.

![Initialization in Front of the Camera](frame0008.jpg)

Figure 4.3: Output image of the marker initializing the iteration in front of the camera. It is observable that the pose solution is behind the marker.

### 4.2.4 No Error Thresholding

As was said before, if the error of the iteration starting from the previous values of \( \mathbf{R} \) and \( \mathbf{T} \) is under a threshold, the solution is assumed to be correct.

<table>
<thead>
<tr>
<th>Frame 0008</th>
<th>Initialization</th>
<th>Position</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>(-0.076536, 0.123477, -0.815684)</td>
<td>1.00455</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>(0.0774665, -0.124878, 0.825043)</td>
<td>0.558534</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frame 0009</th>
<th>Initialization</th>
<th>Position</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>(-0.0661108, 0.119712, -0.814319)</td>
<td>0.81968</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>(0.0685828, -0.124256, 0.842974)</td>
<td>0.233577</td>
</tr>
</tbody>
</table>

Table 4.1: Position and error for the two initializations for the two frames where the pose jump happens.

The defined threshold is 1, so if error 1 < 1 the second initialization is not tried. This happens in frame0009.jpg (error 1 = 0.81968 < 1). As can be seen in Table 4.1 if this threshold is not applied the program should get the good position value. If we compute the marker position we get Figure 4.4.

It can be seen that a lot of pose jumps have disappeared, but not all.
4.2.5 Reducing Iteration Space

Another way to avoid the algorithm getting the marker pose at the back of the camera is to reduce the iteration space. In this case the rotation angles during the iteration move 10°, and the program iterates a maximum of 15 times up and 15 times down; so it searches for the minimum from $-150°$ to $150°$ (Figure 4.5). It was said before that $\alpha$ and $\gamma$ are rotation angles around $Z$, so total rotation around $Z$ can be much more than $360°$, and that is not necessary but unproductive.

Figure 4.5: The iteration space is reduced from 150° to 75°.
The iteration space has been reduced to 15 steps of 5° each. Now the iteration space goes from $-75^\circ$ to $75^\circ$, that is enough to locate the markers in the space of the camera. Figure 4.6 shows the output position for a reduced iteration space. This solution works better than the normal but it is not perfect.

![Cumulated error graph](image1)

![Marker position graph](image2)

**Figure 4.6:** Marker position reducing the iteration space. The number of pose jumps is lower than in figure 4.1 but there are too much jumps.

### 4.2.6 Combination of Solutions.

The best solution is to combine both solutions. In this case, the 3D marker’s position can be seen in Figure 4.7. It can be beheld that in this case the output position has not any pose jump or noise. This solution can be considered as the good one. It was also necessary to add another condition. Sometimes the iteration returns a marker position in the origin $O(0,0,0)$, and if this happens the program takes the solution from the other initialization.

### 4.3 Solution Implementation

In the previous section a theoretical solution was proposed. This section will show the implementation of this solution. All the graphs shown before were done with the help of MATLAB, but the entire project is developed with C++. MATLAB has been used because its easiness doing plots, but the datum has been
extracted directly from the C++ program. Here, some of the most important functions of the program and the solution will be presented.

4.3.1 No Error Thresholding

The function where the transformation is done in `arGetTransMatCont2(...)`. The code of it has been changed to give to the user the option of executing the old program where the pose jump happens or execute the solution exposed in this project:

```c
AR_TEMPL_TRACKER :: arGetTransMatCont2 ( ARMarkerInfo * marker_info ,
    ARFloat center[2], ARFloat width, ARFloat conv[3][4])
{
    if( CHECK_POSE_JUMP ){
        return arGetTransMatCont ( marker_info , conv , center , width , conv );
    }
    else {
        return arGetTransMatContCheck ( marker_info , conv , center , width , conv );
    }
}
```

Where `arGetTransMatCont` is the function where the flickering problem appeared. Now, `CHECK_POSE_JUMP = 1` by default, so that the program automatically runs `arGetTransMatContCheck`, the implementation of our solution.

```c
AR_TEMPL_TRACKER :: arGetTransMatContCheck ( ARMarkerInfo * marker_info ,
    ARFloat prev_conv[3][4], ARFloat center[2], ARFloat width ,
    ARFloat conv[3][4])
{
```

Figure 4.7: Marker’s position without error threshold and reducing the iteration space. In this case, the pose of the marker has no noise or jumps.
In lines 3 and 4 the two errors with both different initializations are calculated, and in line 5 both errors are calculated. The initialization which has lower error rate is considered as the solution.

There is also an option (deactivated by default) to change the sign of the rotation if the marker is behind the camera\footnote{A proposed solution exposed in section 4.1 Change Sign.} the variable FLICKERING\_CHANGE\_SIGN (Line 13).

### 4.3.2 Reducing Iteration Space

The iteration is done in the function \texttt{arModifyMatrix(...)}. The whole function can be found in the annex. Here, the most important part of it will be shown:

```
AR_TEMPL_TRACKER::arModifyMatrix(ARFloat rot[3][3], ARFloat trans[3], ARFloat cpara[3][4], ARFloat vertex[][3], ARFloat pos2d[][2], int num)
{
    ...
    factor = (ARFloat)(5.0*MD_PI/180.0);
    for (j = 0; j < 15; j++) {
        minerr = 1000000000.0;
        for(t1=-1; t1<=1; t1++) {
            for(t2=-1; t2<=1; t2++) {
                for(t3=-1; t3<=1; t3++) {
                    a1 = a2 + factor*t1;
                    b1 = b2 + factor*t2;
                    c1 = c2 + factor*t3;
                    ...
                }
            }
        }
    }

```
Annex 3D Pose Estimation of Visual Markers

At line 6 the variable factor is calculated as 5°. This is the iteration space reduction explained in the section 4.5. After this, the residuals are calculated with (3.10) and the total error is returned.

Also in the file opencv_cam_node.cpp some changes were needed to execute the program with an image from a file. The way to do it is quite simple, it just needs to add some lines to function publishCamera(...):

```c
void OpenCvCam::publishCamera()
{
    ... char text[0xff];
    sprintf(text, "%s", file);
    if(isStatic) {
        memcpy(&cameraImage_.data[0], cv_image_static_.data,
            cameraImage_.data.size());
    } else {
        memcpy(&cameraImage_.data[0], img_.data, cameraImage_.data.size());
    }
    cameraPublisher_.publish(cameraImage_, cameraInfo_);
}
```

In this function the string file has the route of the file, and isStatic is true if the image comes from a file and false if the image comes from the camera.

### 4.3.3 Execution of the Code

The rest of the code of ARToolKitPlus can be found at GIT: [https://github.com/v4r-tuwien/v4r_ros.git](https://github.com/v4r-tuwien/v4r_ros.git). Although, the most important files are in the annex of this project:

- First of all the camera image needs to taken. To do it it is necessary to execute v4r_opencv_cam, a function in opencv_cam_node.cpp.
- When the program is executed, the main function in v4r_artoolkitplus.cpp is executed, helped by other functions in v4r_artoolkitplus.cpp.
- The marker detection is done in files arDetectMarker.cxx and arDetectMarker2.cxx.
- The space transformation is done in files arGetTransMat.cxx, arGetTransMat2.cxx, arGetTransMat3.cxx, arGetTransMatCont.cxx.

This list just shows the most important files that take part in each stage of the program. All the other files can be found at GIT repository.
5 Time Plan

A the project plan was made at the beginning of the project, but it has changed during the fulfillment of the project. This section will show how this time plan has changed, from the first one (Figure 6.1), to an edited one in the critical review (Figure 6.2), and finally the followed plan (Figure 6.3).

![Project plan’s Gantt Chart](image1)

**Figure 5.1: Project plan’s Gantt Chart.**

![Critical review’s Gantt Chart](image2)

**Figure 5.2: Critical review’s Gantt Chart. Task 3 (Implement the Solution) is longer than project plan’s prevision.**

The three Gantt Charts are similar, but not the same. Some little issues appeared during the realization of the project, but the project plan was almost followed.

The second chart was made during December. There is only one small difference between the 2 first time plans (Figures 5.1 and 5.2), the internal tasks of
Figure 5.3: Final Gantt Chart. Task 3 (Implement the Solution) and 4 (Implement the Solution) are longer than project plan’s and critical review’s prevision.

*Implement the Solution* were changed because *Analyze the Problem* was longer and harder than expected.

Finally, comparing the followed time plan (Figure 5.3) to the other ones, can be concluded that *Implement the Solution* was much longer than expected. It was first planned to take 32 days, but finally took around 53 days, three weeks more than expected. Also *Test the solution* took more time than expected.

This delay was due to the complex solution. It was more difficult than expected to find the root of the problem. Also some proposed solutions where dismissed because they were wrong, that’s why *Implement the solution* and *Test the solution* were partially done at the same time.
6 Budget

In this section is calculated an approximate price of this project. All the prices will be considered in a table at the end of this section.

This project has no costs in terms of or software. All the software used (Ubuntu and ARToolKitPlus) was open source, so everyone could get it for free. But, in the other hand, the hardware used have to be considered. The costs come from buying or renting a computer during four months (600 €), and buying a Kinect camera (50 €).

This project took around 4 months of work, from October to February, that means 15 working weeks. It was spent approximately 24 hours per week, that means a total of 360 hours. The salary of a junior engineer is 8 €/per hour. This project maybe could been done by an experienced engineer with less time, but his salary would increased a lot.

<table>
<thead>
<tr>
<th>Hardware</th>
<th>650 €</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal computer</td>
<td>600 €</td>
</tr>
<tr>
<td>Kinect camera</td>
<td>50 €</td>
</tr>
<tr>
<td>Software</td>
<td>0 €</td>
</tr>
<tr>
<td>Ubuntu</td>
<td>0 €</td>
</tr>
<tr>
<td>ARToolKitPlus</td>
<td>0 €</td>
</tr>
<tr>
<td>Incomes</td>
<td>2880 €</td>
</tr>
<tr>
<td>Junior engineer salary</td>
<td>8 €/hour</td>
</tr>
<tr>
<td>Junior engineer hours</td>
<td>360 hours</td>
</tr>
<tr>
<td>Junior engineer</td>
<td>2880 €</td>
</tr>
<tr>
<td>TOTAL</td>
<td>3530 €</td>
</tr>
</tbody>
</table>

Table 6.1: This table summarizes the costs of the entire project.

So, as we can see in the Table 6.1 the total cost of this project would be hypothetically around 3530 €. By the way, this project had no costs, because it was not necessary to buy a new computer or a new camera, and the junior engineer was working on this thesis so he had no salary.
7 Conclusions

This program is mainly applied to robotics and augmented reality. That’s why it’s so necessary to achieve the best precision possible. An improvement that could be applied is to change the type of visual markers. In this case square markers are used, but as was said in the introduction, these are not good markers regarding occlusion and minimal detectable pattern size. The circular markers are robust in these aspects, so a new work could be to use these type of markers. The problem would be that a lot of things would need to be changed; the entire marker detection stage and part of the transformation.

After the fulfillment of the projects, the goals need to be evaluated (section 1.2). It can be said that all four goals have been accomplished. Particularly, the goal examine the problems and bugs of the software, analyze a problem, find the best way to solve it and implement a reasonable and effective solution was the hardest one. It took more time than expected, as is reflected in the Time Plan.

The software analyzed in this work is a small part of a big project in robotics. The whole project is a development for robotics of a technique called SLAM (Simultaneous Localization And Mapping). This technique tries to construct a map of an unknown environment while simultaneously keeping track of its location in the physical environment. In this case it will be a robot with a camera mounted on it, and the software will try to identify visual markers placed on certain locations within the room. Then the SLAM algorithm should be used to map the location of the markers and place the robot in the space.

In a more personal opinion, it has to be said that the project was more difficult than expected, because of the lack of background in this complex software and the many-faceted environment and tools of computer vision. But the experience of working in a weak field is always positive, because it gives a lot of knew knowledge and it enlarges one’s mind. It also has been great to work in a foreign university like TU Wien, where the department of visual computing and robotics is one of the best ones in Europe dedicated to educational purposes.
Acknowledgments

I would like to thank both of my supervisors. In TU Wien Markus gave me the idea for the project and helped me and guided me during the elaboration of the thesis, giving me the opportunity to experience in a new exciting field. Also Albert Oliveras helped me to organize my time and to carry out all the documents needed in my home university.
References


[9] Robust Pose Estimation from a Planar Target, by Gerald Schweighofer and Alex Pinz.
