Computer Vision Final Project Image Inpainting With Aruco Markers

Tiago Madeira 76321 Pedro Silva 72645

Abstract—This document contains a description of the process of developing a project for the subject of Visão por Computador, which is part of the fifth year of University of Aveiro's course Engenharia de Computadores e Telemática, and falls within the scope of Computer Vision. It showcases the work done, what was considered most important, the main goals, along with the problems and solutions found in order to develop this tool. Suggested and accepted as a final assessment item for the subject, this tool was created with the intention that it would allow for the inpainting of ArUco markers and use of the created texture in corresponding clouds of points.
Through an approach that is intended to be simple and organised, this report starts by setting the context and motivation that back this project up. The features, ideas, problems and solutions are then presented, following roughly the real chronological order in which they came about throughout the development of the project. Finally, the main conclusions are discussed, going over the acquired knowledge, the main goals and their completion, and the enriching characteristics of the project, technical-wise and social-wise.

Index Terms—Computer Vision, Inpainting, ArUco, Digital Images, Point Clouds, Projection of 3D points

1 Introduction

In the context of developing the final project for Visão por Computador, a subject that is part of the fifth year of University of Aveiro's course Engenharia de Computadores e Telemática, this idea of creating a tool that would allow for the inpainting of ArUco markers and projection of the created texture to corresponding clouds of points was suggested and accepted as a final assessment item. It is interesting to create tools that have real use, particularly in the scope of a bigger project. In this case, the work shall prove of particular interest in the context of the master's thesis of one of the authors, focusing on geometric optimisation for 3D reconstruction of environments.

This document is made up of 4 sections. After this introduction, the motivation behind this project is explained in section 2, contextualising the work carried out. Then, in section 2, the core of the tool is exposed, discussing some of the problems found and solutions developed in order to obtain the results showcased. Afterwards, in section 4, the visualisation of colour in 3D is presented, explaining in some detail how pixel colour in pictures was used to create coloured point clouds and how this is the bridge to utilising this project in the context of optimisation for 3D reconstruction. Finally, in section 5, some conclusions about the development of this project are presented.

2 The motivation

In the scope of the master's thesis of one of the authors, which focuses on geometric optimisation for 3D reconstruction of indoor environments, the desire to remove ArUco markers from the textures captured came about. The idea is that, using a series of pictures of the scene, through a technique called Bundle Adjustment it is possible to simultaneously refine the 3D coordinates describing the scene's geometry and the optical characteristics of the cameras. This technique may involve the use of ArUco markers to make a first guess, indicated by the blue dot in figure 1, and to define a ground truth, indicated by a red square in

figure 1. Needing the ArUcos in the scene, but not wanting them to show up in the texture of the optimised final version of the 3D model created this idea of removing them, inpainting them from the image. Furthermore, in this context of geometric optimisation, the points clouds were considered the best way to try to visually understand and measure the effect of the optimisation and its success. As such, in order to visualise the colour in 3D, associating colour to point clouds by extracting the corresponding rgb value from the corresponding picture seemed like the way to go.

The tools created in this project were built in a way that would best suit their use in the context of the master's thesis of the student Tiago Madeira. The datasets, tools for loading information and its internal structure are in harmony with the work already developed with professor Paulo Dias and Miguel Oliveira in that scope.

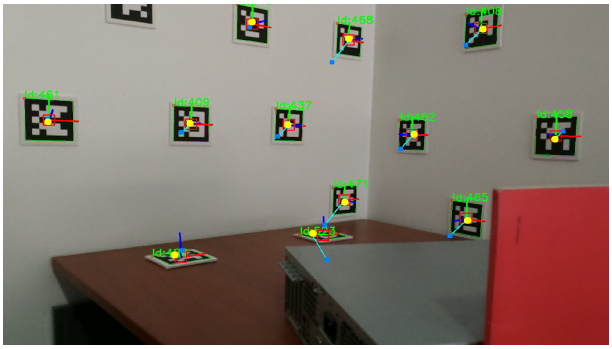

Fig. 1: Optimization example

3 Inpainting

All the masks showcased in this section were blended with images to better showcase where they fit and what regions are the target.

3.1 Creation of the masks

In order to use the OpenCV *inpaint* function, which is intended to restore a selected region in an image using the region neighbourhood, one must create a mask in order to define this region. The plan was to remove the ArUco markers from the scene, and as such, their detection was used to create these masks. This can be observed in figure 2.

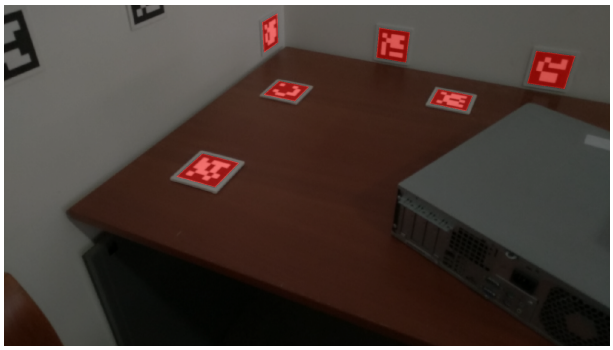

Fig. 2: Simple mask created from aruco detection

The problem with this is that the square detected only includes symbol itself, not accounting for the margin around it, creating a mask that would not allow for the inpainting of the whole item, leaving blank cards scattered through the scene. The first attempt at solving this problem was to dilate the masks, using OpenCV's *dilate* function, which can be observed in figure 3. This didn't work very well because it is a 2D operation. Markers more at an angle and closer to the camera need a bigger mask dilation and possibly in different directions, for example.

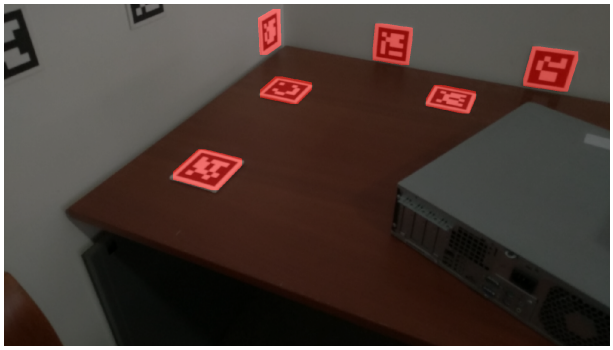

Fig. 3: Mask after dilation was applied

The solution found for this was to create a 3D object with the same measurements as the markers, including thickness, so the markers can be masked even when the angle is so big that it's possible to see the sides of it. This object is made up of only the vertices. It must be transformed from each detected ArUco's reference frame to the camera's reference frame and then projected to a 2D image. This is accomplished by using the OpenCV function

projectPoints with the translation and rotation vectors associated with each ArUco (and the intrinsic parameters and distortion coefficients associated with the camera). Then, taking advantage of the created function *drawMask*, it's possible to draw the vertices and fill in the figure in white over a black image. These masks are showcased in figure 4.

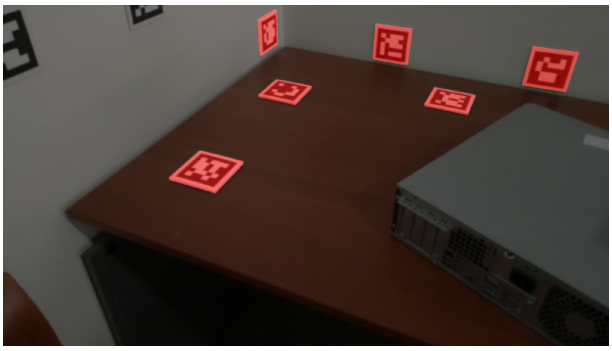

Fig. 4: Mask created from 3D object projection with thickness

3.2 Simply inpainting

Even with the masks now covering the markers fully, see figure 5, the results of the inpainting were underwhelming, as shown in figure 6. Inpainting in this manner is usually done in smaller areas, while the size of an ArUco marker in the scene is substantial.

Fig. 5: Mask used for inpainting

Fig. 6: Inpainting with *inpaint* method from OpenCV

3.3 Improving the inpainting

3.3.1 First stage

In order to improve the result of the inpainting, blurring the area where the inpainting took place seemed like a good idea. To accomplish this, the image where the inpainting was applied was copied and blurred using the OpenCV function *medianBlur* with a high aperture linear size (ksize=201). Then, the mask used to perform the inpainting was used to replace the pixels in those areas of the original image, with the pixels of the blurred image, see figure 8. The result is showcased in figure 9.

Fig. 7: Auxiliar image 1 - Median blur of figure 6 image with ksize=201

Fig. 8: Mask for the pixels which will be replaced by the Auxiliar image 1

Fig. 9: First stage result

3.3.2 Second stage

The overall colour of the ”first stage” looks a lot better than the original simple inpaint. However, the limits of where the texture (taken from the blurred image) was applied are quite clear. In order to fix this, a new set of masks was created. This takes advantage of the method described in subsection 3.1, transforming and projecting a 3D object with the shape of the marker, yet making the area of the square somewhat larger, see figure 11. The image obtained in the first stage is copied and blurred using a median blur again, however this time with a smaller aperture linear size (ksize=51), resulting in figure 10. This new mask is, much like before, used to copy the corresponding regions from the blurred image to the original ”first stage” result, creating figure 12.

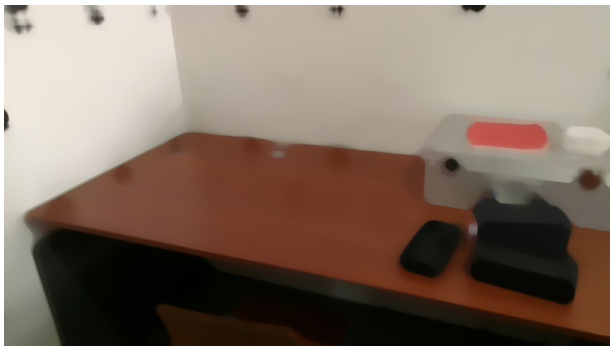

Fig. 10: Auxiliar image 2 - Median blur of the first stage result with ksize=51

Fig. 11: Mask for the pixels which will be replaced by the Auxiliar image 2

Fig. 12: Second stage result

3.4 Final Results

After the ”second stage”, the improvement on the inpainting seems considerable. However, there’s still an effect of a mist in the areas of inpainting and blurring. This is addressed in the end by applying a Bilateral Filtering function (OpenCV’s *bilateralFilter*) with relatively small values for the `pixel neighbourhood`, `sigmaColor` and `sigmaSpace` (9, 75, 75). This allows for the preservation of the edges in

the image and avoids the mixing of colours, while making the surfaces a little smoother, making the grainy pattern of the walls a little less noticeable for example, which makes the inpainted areas blend in a little better. The final result of the whole inpainting process is showcased in figure 14

Fig. 13: Original image - OfficeDemoV1 dataset (00000008.jpg)

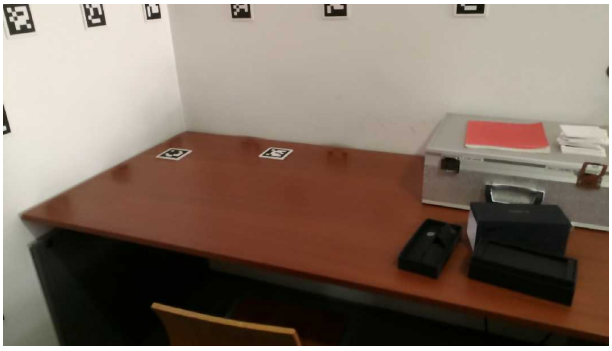

Fig. 14: Inpainted image - OfficeDemoV1 dataset (00000008.jpg)

3.5 Cross-Inpainting

More often than not there's ArUco markers in the pictures of a dataset that aren't detected. This can be because the marker is not fully in the field of vision of the camera, because the camera was in motion and the picture taken was blurred, or simply because the marker is at a very awkward angle. Whatever the case, some examples can be seen in this document. Because the transformation from ArUco markers to cameras and from the cameras to the world are known, it was theorised that it should be possible to know if an ArUco can be found within an image, even if it wasn't detected in it.

The solution implemented for what was called "Cross-Inpainting" takes advantage of transformations from the ArUco to the world. Every time a new ArUco marker is detected in one of the images, the transformation of the ArUco to the world is calculated and saved, through the use of the transformations of the ArUco to the camera and from that camera to the world. Whenever one image is being inpainted, it's possible to access this transformation for every ArUco found in every image and check whether each ArUco can be projected within the image. This is done by taking the marker 3D object, transforming it from the ArUco reference frame to the world, then transforming it from the world to the camera (the one that took the image currently being inpainted) reference frame, and finally projecting it to the image. What is obtained from these projections can be seen in figure 15.

The algorithm seems to work, however because the optimisation is not done yet, the estimated positions of the cameras in relation to the world and the ArUcos don't match, and the inpainting cannot succeed. However, it is expected that simply by performing a good optimisation this "cross-inpainting" will work fully and allow for the removal of undetected markers, the ones at the edges of the images being of particular interest since they are harder to avoid.

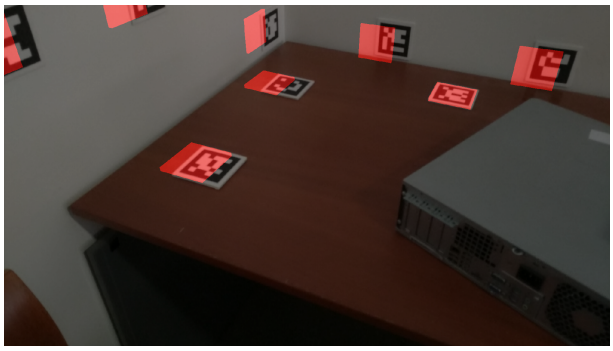

Fig. 15: Cross inpainting mask demo

4 3D COLOUR VISUALISATION

As mentioned before, point clouds will prove instrumental to visually measure the effects of

the geometric optimisation. In order to better visualise these point clouds it would be interesting to have them coloured, particularly with the inpainted images. In order to achieve this one has to read the information from the .ply extension files found in the dataset, project the points to the image and extract the rbg value of the corresponding pixel. The value for the 3d points is found in the world reference frame, however it is stored in an OpenGL coordinate system. Thus, the points must be read from the file, converted from OpenGL to OpenCV coordinate system, transformed from the world to the camera's reference frame and projected to the image. The rgb value can then be extracted. Finally a new .ply extension file is created and the information of colour is stored along with the original vertices. This format has a set of properties/labels on its header that define the context of the file, such as the format(ASCII), the number of elements present in each vertex, a set of properties that define each element, etc...

When all of this is done, and the `.PLY` file is saved correctly, the content of this file can be visualised using Meshlab or some `CloudViewer` class, the `PCL` library for instance or a simple viewer provided by the `PPTK` library, and the output should be something similar to figure 16 image. For comparison purposes, in figure 17 a point cloud coloured with the original image can be observed.

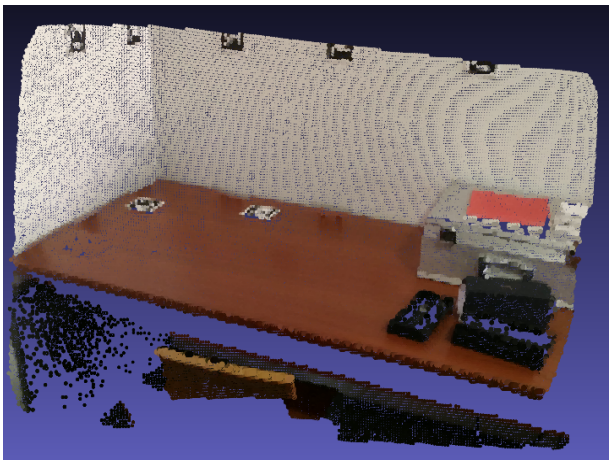

Fig. 16: Point cloud with colour obtained from 2D inpainted picture (meshlab with points enlarged

3d.png

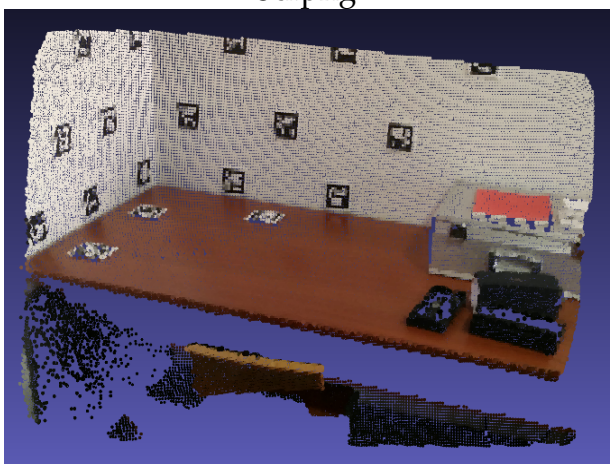

Fig. 17: Point cloud with colour obtained from 2D original picture (meshlab with points enlarged

4.1 Combination of clouds

Just to give a sense of what the combination of point clouds might look like once an optimisation is done, it was decided that the point clouds should be aligned, combined and visualised for demonstration purposes.

In order to align and combine the clouds calculated in the previous section, the `PCL` library provided by `OpenCV` was used. The idea was to use the functions provided by the `PCL` library in python, however there was a problem installing the `python-pcl` package, and so, as this was a sort of demonstration add-on, the `PCL` module for `C++`, which had been used in the `Lecture 7 - 3D Vision` class, was utilised.

In order to read the point cloud information saved in `PLY` format, the `PCL` function *loadPLYFile* was used, assigning the data to a `PointCloud<PointXYZRGB>::Ptr` object. To combine and to align the clouds, an alignment `ICP Alignment` using the `pcl::IterativeClosestPoint` class is performed. This class implements an algorithm based on `Singular Value Decomposition(SVD)` which aligns the given down sampled cloud points based on two `<PointXYZRGB>` points (a source point and a target point). Afterwards, a set of point transformations on the `ICP` point are applied,

using a series of methods provided by the `ICP` class and the information of the point clouds referred on the previous section. One cloud as the source (to be aligned) and the other cloud as the target cloud (the reference for the alignment).
In this context, multiple point clouds are used, therefore some strategy must be chosen in order to perform `ICP Alignment` of everything. The route chosen was to align each cloud based on the information of the alignment of the previous two clouds, performing a sort of `"Chain" Alignment` process. In order to visualise the output of the alignment the *spin* method of the `PCLVisualizer` class of the `PCL` library was used, concatenating each point cloud obtained from the alignments into a `PCLVisualizer` object through the *addPointCloud* method. The output result can be seen in figure 18.

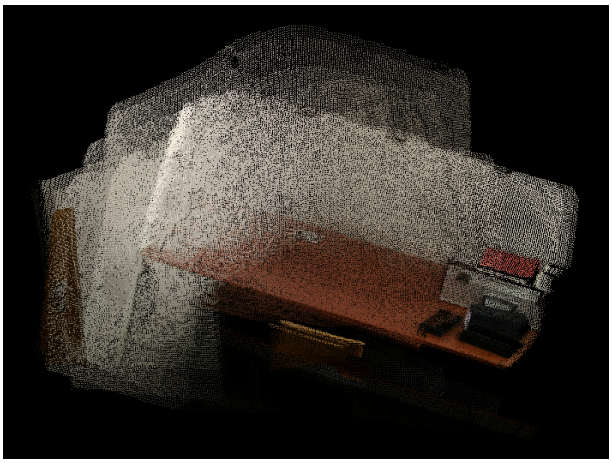

Fig. 18: Point cloud with colour combined through ICP

5 Conclusions

Throughout the development of this project, it was possible to learn how a multitude of techniques work and match them together in a greater scale than merely lessons and tutorials can provide, with a greater motivation backing it up. It was all planned with a target use in mind, but the features developed can be used in other contexts. ArUco marker use is quite popular and the ability to hide the markers from sight in the final texture is something interesting, for example in Augmented Reality, when these are used to project 3D objects on the scene. Taking point clouds in .ply format, extracting colour for their points from images and writing a new set of .ply files containing colour information is another example of a module that was interesting to create and may be useful in the context of other projects.

Having this said, the goals set for the project were achieved. There was success in the goal of creating a tool useful to improve the visual aspect of texture in a scenario where ArUco markers are present. As discussed before, some of the results obtained are just a demonstration, and should improve considerably after an optimisation has been performed, such is the case of ”cross-inpainting”15 and the visualisation of combined point clouds. Besides the inpainting of ArUco markers, this tool allows for the visualisation of the final colours in the point clouds. This may prove crucial in the visual verification of the effects of the optimisation. The success of the optimisation should be verifiable visually in the comparison of the point clouds, if all goes according to plan.

It is also of course important to go over the whole social aspect of working in a team, even if only a two-man one, soft-skills become imperative, cooperation and understanding is crucial and something to always be worked on granted the opportunity, which this project has.

Appendix A
Acknowledgments

The authors of this report would like to thank their professors of Visão por Computador António Neves and Paulo Dias, and also professor Miguel Oliveira for their time, patience, and insight.

REFERENCES

[1] Docs.opencv.org, "Camera Calibration and 3D Reconstruction" [Online].
Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[2] Docs.opencv.org, "Image Inpainting" [Online].
Available: https://docs.opencv.org/2.4/modules/photo/doc/inpainting.html
https://www.nvidia.com/research/inpainting/
https://en.wikipedia.org/wiki/Inpainting

[3] Docs.opencv.org, "Image Inpainting Masks" [Online].
Available: https://docs.opencv.org/3.4/df/d3d/tutorial_py_inpainting.html
https://www.researchgate.net/post/How_to_design_mask_of_an_image_for_inpaintings2
http://answers.opencv.org/question/205615/how-to-get-a-mask-of-an-image-so-that-i-can-use-it-in-the-inpainting-function/
https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_reconstruction_Inpainting_Interpolation.php

[4] Docs.opencv.org, "Simple Image Inpainting" [Online].
Available: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_photo/py_inpainting/py_inpainting.html
https://stackoverflow.com/questions/10340537/image-inpainting-cvrectangle
https://stackoverflow.com/questions/53838667/how-to-get-a-mask-of-an-image-so-that-i-can-use-it-in-the-inpainting-function
https://dsp.stackexchange.com/questions/36222/what-is-the-difference-between-image-inpainting-and-image-denoising

[5] Docs.opencv.org, "PCD (Point Cloud Data) 3D Format Images" [Online].
Available: http://pointclouds.org/documentation/tutorials/pcd_file_format.php
https://stackoverflow.com/questions/47103273/pcd-file-cannot-view-color-information-correctly
https://abhishek4273.com/2014/07/01/opencv-and-pclfiltering-a-pcd-file/
https://stackoverflow.com/questions/45783815/view-multiple-point-clouds-in-same-window-using-pcl-in-c?fbclid=IwAR1hIPdh1MTeLM9bDk2URmgiXhdF7zSn14SYnW9_35P-CzaJGJfbTKaFfH8
https://stackoverflow.com/questions/3796452/how-do-i-display-draw-a-ply-object-in-opengl
https://stackoverflow.com/questions/30764222/how-to-read-ply-file-using-pcl
https://stackoverflow.com/questions/51350493/convertion-of-ply-format-to-pcd-format
http://docs.pointclouds.org/trunk/structpcl_1_1_r_g_b.html

[6] Docs.opencv.org, "ICP Alignment with Point Clouds" [Online].
Available: http://docs.pointclouds.org/1.0.1/classpcl_1_1_iterative_closest_point.html#_details
https://docs.opencv.org/ref/master/d9/d25/group__surface__matching.html
https://www.mrpt.org/Iterative_Closest_Point_%28ICP%29_and_other_matching_algorithms