INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC 1/SC 29/WG 4
MPEG VIDEO CODING

ISO/IEC JTC 1/SC 29/WG 4 m66989
April 2024, Rennes

Title **Update on the Manual of Extrinsic Camera Parameters Calibration Software**

Source **PUT, ETRI**

Authors **Błażej Szydełko, Dawid Mieloch, Adrian Dziembowski, Jakub Stankowski, Dominika Klóska, Jun Young Jeong, Gwangsoon Lee**

Abstract

This document presents an update to the Extrinsic Camera Parameters Calibration Software with reference to [M64164]. The changes include, among others, simplification of the software building process, reduction of the processing time and updated evaluation of results. The presented version of software is now also available with the paper published in SoftwareX Journal [Szydelko24].

1 Software changes

- Simplified user operations, number of processing steps reduced, software building process automated
- Creating initial parameters includes copying intrinsic camera parameters, fixed bug in using real camera rotation angles in initial calibration parameters, currently all cameras are facing the same direction
- Total processing time reduced down to approx. 30 min:
  - Number of cameras in Blender scene halved
  - Number of rendered frames reduced
  - Default depth estimation parameters changed (IVDE)
- Time of calibration process reduced
- Conditional use of Numba depending on the Python version and availability
- Rescaling parameters for evaluation purposes
- Added script for extracting true marker positions in each view
- Changes to evaluation results presentation

2 Software description

Overview - Figure 1.

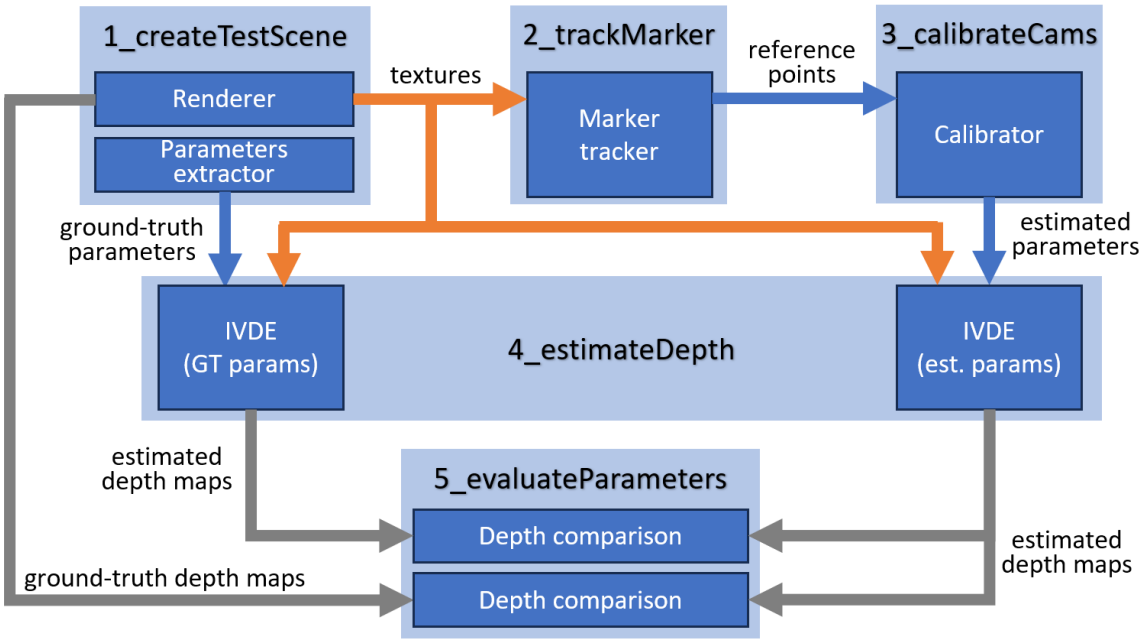

Figure 1. The general scheme of the ECPC software.

2.1 Blender project for generating of multiview calibration sequence

To calibrate multicamera systems and assess estimated camera parameters accurately, reliable calibration and ground-truth reference data are needed. Blender's virtual environment offers controlled conditions for calibration and evaluation. The project provides a scene (Figure 2.) captured by a multicamera setup and scripts to output multiview calibration sequences in MVD format. The scene includes 10 virtual cameras arranged on an arc, typical for immersive video applications. Users can customize camera arrangements as needed. The calibration algorithm estimates camera positions based on tracked reference points, represented by an orange ball.

Figure 2. The Blender project contains a test scene with 10 cameras, several static objects, and a moving calibration marker.

2.2 Marker tracking

Marker tracking module analyzes calibration sequence for marker positions in each frame and view. Output file includes horizontal and vertical coordinates with a visibility flag indicating marker visibility. Algorithm detects orange pixels (Cb ∈ [50, 70], Cr ∈ [155, 195]) and segments them into clusters. Largest cluster with high circularity in the first frame is selected and tracked in subsequent frames for position calculation (Figure 3).

A) B)

C)

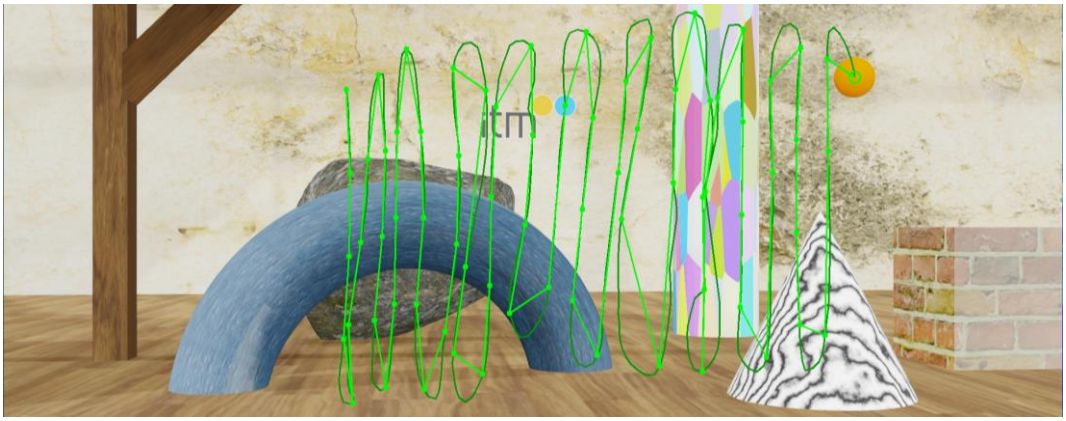

Figure 3. Tracked marker. Yellow lines: the predicted path of a marker; green lines: the actual path measured by the marker tracker algorithm; blue rectangle: area in which the marker is being searched for;

A: analysis of consecutive frames of a calibration sequence,

B: analysis of every 10th frame of a calibration sequence,

C: comparison of detected marker positions in A (dark green line) and B (light green line) – the curves overlap every 10th frame (light green dots).

2.3 Calibration

The calibrator module within the ECPC software is pivotal, responsible for deriving extrinsic parameters for multicamera systems. It achieves this by minimizing the distance between reference points and their corresponding epipolar lines, employing the L-BFGS algorithm renowned for handling problems with numerous variables. The calibration process unfolds across multiple global iterations, each iteration refining the model by excluding outliers. Points exhibiting an error magnitude ten times greater than the average are flagged as outliers, allowing for automatic detection and removal, even in cases of erroneous object detection caused by factors like partial occlusion or changes in lighting conditions. Finally, the calibrator tool produces a .json file containing sequence parameters essential for next step - depth estimation process.

2.4 Depth estimation

In the ECPC software, the accuracy of estimated parameters is evaluated by comparing depth reference depth maps with the ones estimated based on camera parameters received in the previous step.

To provide fast and reliable comparison, the ISO/IEC MPEG Video Coding reference software for depth estimation was used – IVDE (Immersive Video Depth Estimation), as it provides good quality depth maps (Figure 4) for multiview systems with arbitrary camera arrangements [1]

Figure 4. Depth maps for view 6: A) rendered by Blender, B) estimated using exact camera parameters, C) estimated using calibrated camera parameters. The contrast of images was increased to highlight differences. Note that the quality of B is worse than for A because of imperfection of a depth estimation algorithm which cannot perfectly estimate depth for some parts of the scene (especially for areas acquired by a single camera (e.g., left top part of the view).

2.5 Evaluation

In the final stage, we evaluate the accuracy of estimated camera parameters by comparing the estimated depth maps to ground truth. This evaluation focuses on identifying inaccurate depth values, defined as bad pixels.

Bad pixel calculation occurs through two methods:

1. Comparison between estimated depth maps and ground-truth depth maps rendered by Blender.
2. Comparison between estimated depth maps using calculated parameters and those using ground-truth parameters extracted from Blender.

By default, the number of bad pixels is calculated for several δ threshold values representing 1%, 2%, and 4% of the range represented by a 16-bit depth map (Tab. 1).

Tab. 1. Evaluation results: a percentage of incorrect depth map samples for the threshold values measured for each view.

	Percentage of incorrect samples in depth maps estimated using calibrated camera parameters in comparison to rendered depth maps			**Percentage of incorrect samples in depth maps estimated using calibrated camera parameters in comparison to depth maps estimated using exact camera parameters**		
Acceptable difference threshold [% of max depth value]	1%	2%	4%	1%	2%	4%
Average	***5.68***	***4.55***	***4.01***	***1.61***	***0.64***	***0.38***

3 User manual

To clone the repository to your computer:

```
git clone https://github.com/bszydelko/ecpc.git
```

Then, update submodules:

```
cd ecpc

git submodule init

git submodule update
```

The framework contains scripts to be run in a specific order, the scripts are numbered accordingly. The time required to run all stages (from rendering test sequence to the evaluation of camera parameters) is around 30 minutes.

Step 0 - Prepare environment, build and install

Prerequisites

Windows users having OpenCV installed using vcpkg are required to provide location of OpenCV toolchain file as a environmental variable:

```
SET
CMAKE_TOOLCHAIN_FILE=[PATH_TO_VCPKG]vcpkg/scripts/buildsystems/vcpkg.cmake
```

where **PATH_TO_VCPKG** points to location of vcpkg root directory.
Exemplary, if vcpkg is installed in C:/Tools/ the command typed into the command line will be:

```
SET
CMAKE_TOOLCHAIN_FILE=C:/Tools/vcpkg/scripts/buildsystems/vcpkg.cmake
```

Please note that environment variables are only available for the current console session and will not persist after it is closed.

Execute `python 0_buildSoftware.py`

This script is responsible for building tools used whithin this framework. Compiled projects are located in **DepthEstimation/ivde/build** and **MarkerTracking/build**. In current state only Visual Studio solution is supported.

Step 1 - Test scene creation

Prerequisites

Windows users should add path to Blender and ffmpeg to *PATH* environmental variable. The example for default installation folder of Blender and ffmpeg is provided below. If Blender or ffmpeg is installed in another location - the command should be modified accordingly.

```
SET PATH=%PATH%;"C:/Program Files/Blender Foundation/Blender
4.0";"C:/ffmpeg"
```

Execute `python 1_createTestScene.py`

This script renders multiple viewpoints of scene created in Blender. Views are outputted in MVD format (multiview plus depth), ground-truth camera parameters are also saved. To provide compatibility with MPEG Immersive Video environment, views and depth maps are converted into “yuv” format using ffmpeg. Files will be created in `1_renderedScene` directory.

Note: the script requires approximately 5-10 minutes to render views, the progress can be seen in command line window (Blender window will be blank during rendering).

Step 2 - Calibration points generation

Execute `python 2_trackMarker.py`

This script starts MarkerTracker and extracts the marker position from rendered scenes. Files will be created in `2_trackedMarker` directory.

Note: See Appendix B for ideal calibration points.

Step 3 - Calibration

Execute `python 3_calibrateCameras.py`

The calibration script generates an output files estimatedParams_itN.json (where N is the number of a global iteration) containing calibrated camera parameters. Files will be created in `3_calibration` directory.

Step 4 - Depth reconstruction

Execute `python 4_estimateDepth.py`

Further steps of the provided example evaluate the accuracy of calibrated camera parameters through the quality of depth maps estimated on their basis. This script executes IVDE and calculates two sets of depth maps: using reference parameters and using calibrated camera parameters. The estimation of depth maps takes ~4 minutes for each set (on Ryzen 5900X CPU). Files will be created in `4_depthMaps` directory.

How to change framework parameters

Framework parameters can be set in `globalSetup.py` file:

```
#starting parameters:
start_frame = 0
end_frame = 2000
step_frame = 10
#closest and furthest depth value:
znear = 10
zfar = 200
#resolution of the calibration sequence:
width = 1920
height = 1080
#number of cameras used to captrure the calibration sequence:
num_of_cams = 10
```

Appendix A - Installing OpenCV via vcpkg

Clone vcpkg repository. Run bootstrapping process:

```
./vcpkg/bootstrap-vpkg.bat
```

Install OpenCV:

```
./vcpkg/vcpkg install opencv
```

Appendix B - Ideal calibration points

ECPC provides script for extracting ideal marker coordinates in scene. In order to obtain it, execute:

```
blender -b TestScene/scene.blend --python scripts/extractMarker.py
```

4 Software

MPEG Git Repository: https://git.mpeg.expert/MPEG/Explorations/6DoF/ecpc

Public access: https://github.com/bszydelko/ecpc

Software coordinator: Błażej Szydełko, blazej.szydelko@put.poznan.pl

5 Usage and citation

Please cite reference [Szydelko24] when using ECPC.

6 References

[M64164] D. Mieloch, B. Szydełko, A. Dziembowski, D. Klóska, J. Jeong, G. Lee, “[MIV] Versatile multicamera system calibration framework for immersive video applications”, ISO/IEC JTC1/SC29/WG04 MPEG143 M64164, July 2023, Geneva, Switzerland.

[Szydelko24] B. Szydełko, D. Mieloch, A. Dziembowski, J. Stankowski, D. Klóska, J. Jeong, G. Lee, “ECPC – versatile multicamera system calibration framework for immersive video applications”, SoftwareX, Vol. 26, No. 101669, 2024, DOI: 10.1016/j.softx.2024.101669.

7 Acknowledgement

This work was supported by Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-00207, Immersive Media Research Laboratory).