Camera Auto-Calibration in a Pipe-less Plant

1. Overview:

Calibrating the camera, which means finding the intrinsic and extrinsic parameters, can be achieved using a chessboard and the calibration functions of OpenCV library that are wrapped in Emgu library for .Net languages.

To obtain these parameters, one must capture two or more pictures of a chessboard of known size from different angles. Usually, this can be achieved by placing the chessboard in a known position within the workspace then taking a photo. Later on, the chessboard should be moved to a different known position and another photo can be taken. These two captured images can be used later to estimate the camera parameters by mapping the coordinates of the chessboard's corners in the image to the coordinates of the real plant and then running an optimizing algorithm to minimize the error.

In this implementation, we used a slightly different approach. Instead of manually moving the chessboard, five identical chessboards were fixed permanently on the plant in different locations (four in the corners and one in the center). By performing simple masking, one can obtain 5 different photos of the chessboard from different angles.

In addition to the camera calibration, the chessboards' coordinates can be used to locate the stations and the PLC, which are glimmering objects, and mask them out.

In what follows, the modifications in the original code and the results will be discussed.

2. Code Modifications:

Most of the work was done in `CameraCalibration_1` inside `CameraModule`. When an instance of that class is created during the launch of the application, the constructor `CameraCalibration_1()` will be called. The first task of the constructor is to initialize the values of coordinates of the upper left corners of the five chessboards. In the current implementation, the origin is assumed to be in the center of the plant. After that, the constructor will try to load the calibration images that are assumed to be in the path `CalibImagePath` using the function `LoadCalibrationImages()`. If it succeeds to load the images, the next step will be to locate the chessboards' corners by calling `FindChessboards()`. If the chessboards are found, then the function `CalibrateCamera()`, which is responsible for the actual claibraiton, can be called.

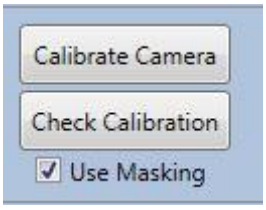

Figure 1: GUI modifications

In case that the calibration images cannot be found or the camera was repositioned, the user will have to manually invoke `AutoCalibration()` function by clicking on 'Calibrate Camera' button (Figure 1). This function will generate the calibration images by taking a photo of the plant and then using predefined masks to create five calibration images (Figure 2).

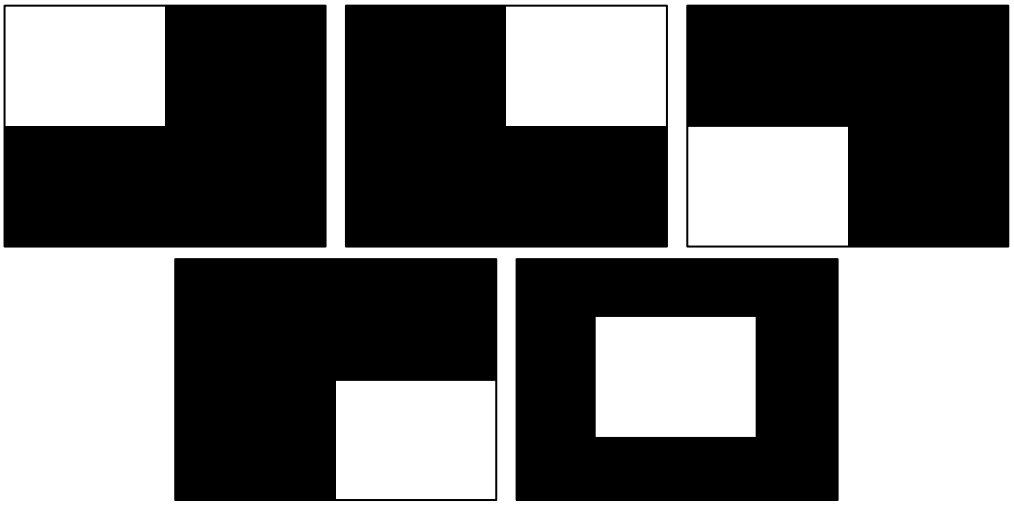

Figure 2: Predefined masks

Figure 3: Generated calibration images

After creating the calibration images, the same procedures of the constructor will be followed.

The function `ExtractCorners()` is used to extract the corner of each chessboard that is closest to the real corner of the plant. These corners will be useful later if the user chooses to use the 'Masking' function to get rid of the reflection caused by the stations or the PLC.

As can be seen in figure 1, the user has the option whether to use stations' mask or not. Also, he can test the current calibration of the camera.

3. Final Results:

3.1 Camera Auto-Calibration:

Though there are five chessboards, only chessboards number 4 and 5 are used for the 'actual' calibration of the camera (estimating the camera intrinsic parameters). The other three chessboards are only used to acquire the coordinates needed to mask out the stations and the PLC. (Figure 4)

Two factors were used to evaluate the implemented approach:

1. The reprojection error returned by the method CalibrateCamera().
2. How the captured frame looks like after remapping it using the estimated parameters.

In the old implementation, the reprojection error was **0.65 [pixel]**. Now, using the suggested approach, that error is **0.26 [pixel]** (around 0.7[mm]). Additionally, the remapped image is fairly good. It can be seen in figure 5.

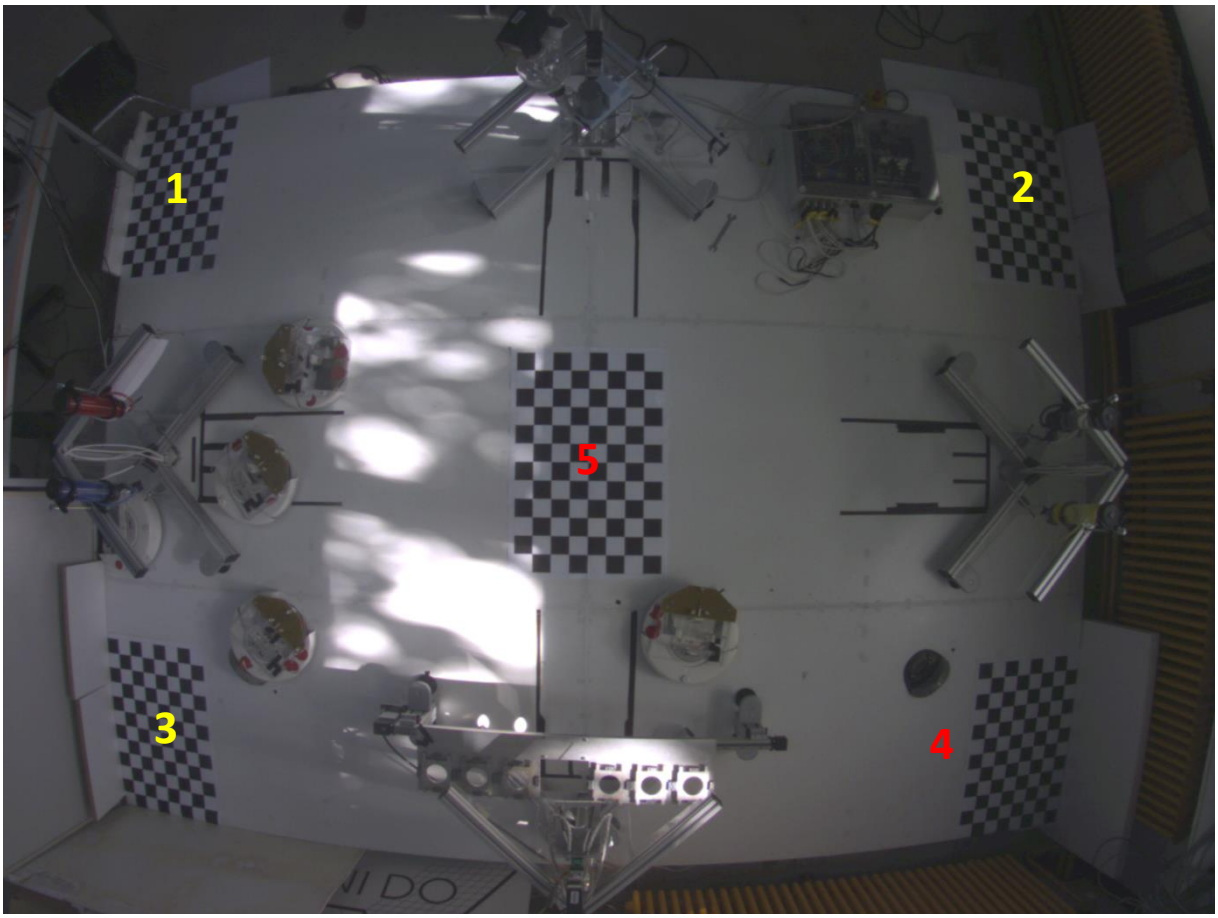

Figure 4: Chessboards used for camera calibration

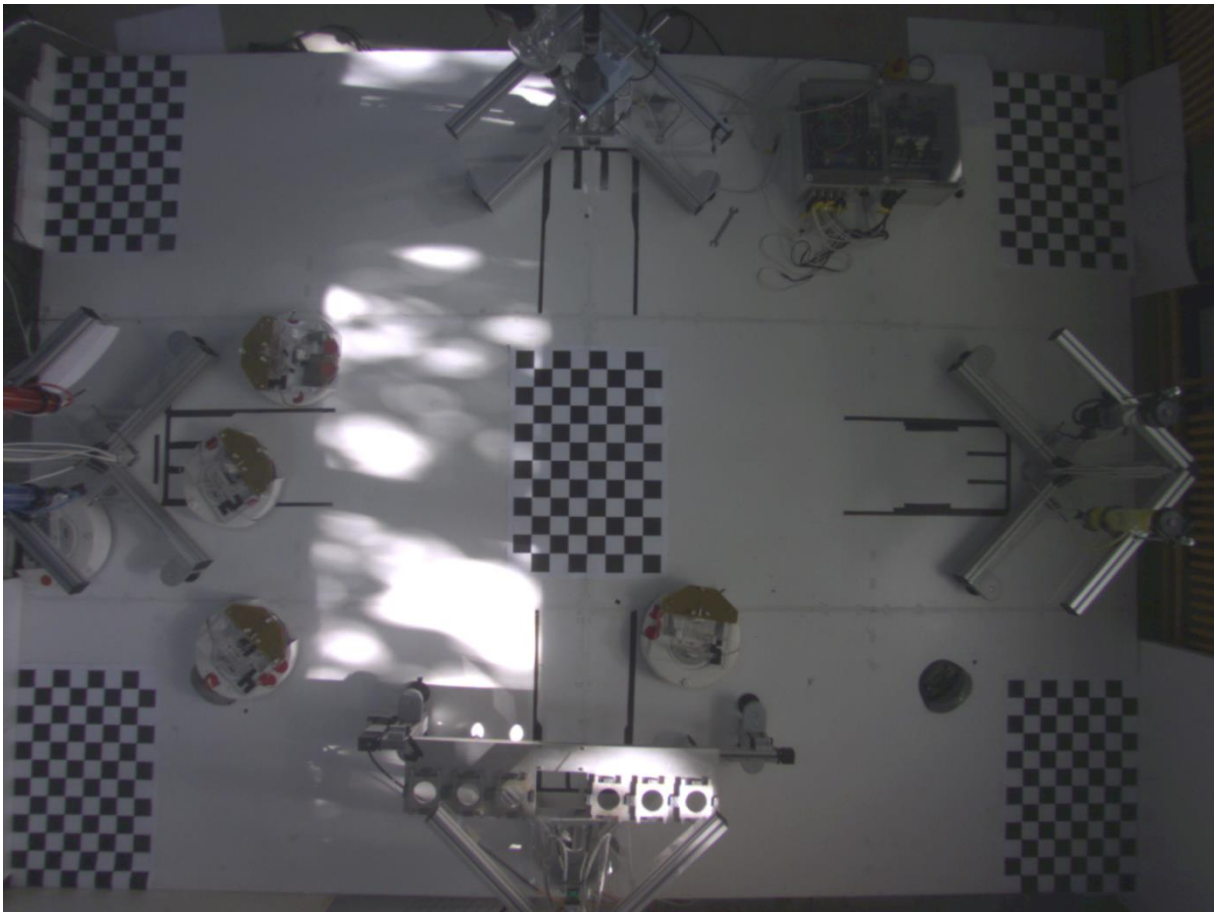

Figure 5: The remapped image using chessboards no. 4 and 5 for calibration

When using all chessboards in the corners (i.e. 1,2,3,4), the reprojection error was **0.66 [pixel]** and the remapped image did not look good. (Figure 6)

A similar result was obtained when using all five chessboards for estimating camera intrinsic parameters. The projection error was **0.61 [pixel]** and the remapped image can be seen in figure 7.

3.2 Masking-out Stations and PLC:

After calibrating the camera, an attempt to create a mask to hide the stations and the PLC will be performed. This mask will be created dynamically based on the detected chessboards AFTER remapping the captured image that was used for calibration. Therefore, if the camera was not positioned in such a way that it can capture the plant with enough margins, some of the chessboards will be cropped after the image is remapped and the process of creating and using the mask will be skipped. Furthermore, it was assumed that the positions of the stations are fixed. In case they are moved, the parameters inside the method createStationsMask() will need to be modified. Figure 8 illustrates how a dynamically created mask would look like. The result of applying this mask over the remapped captured image can be seen in figure 9. The time needed to apply this mask for each frame is 2[millisecond].

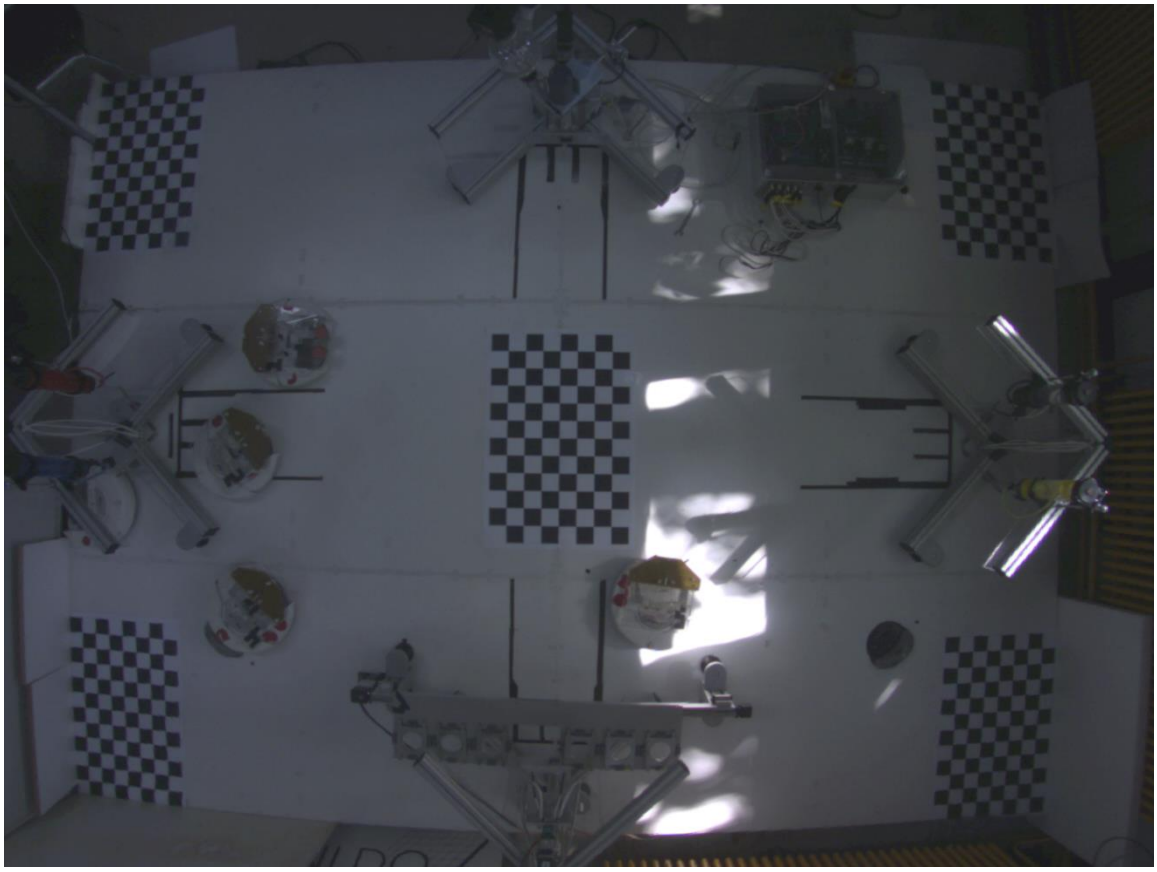

Figure 6: The remapped image using chessboards 1,2,3,4 for calibration

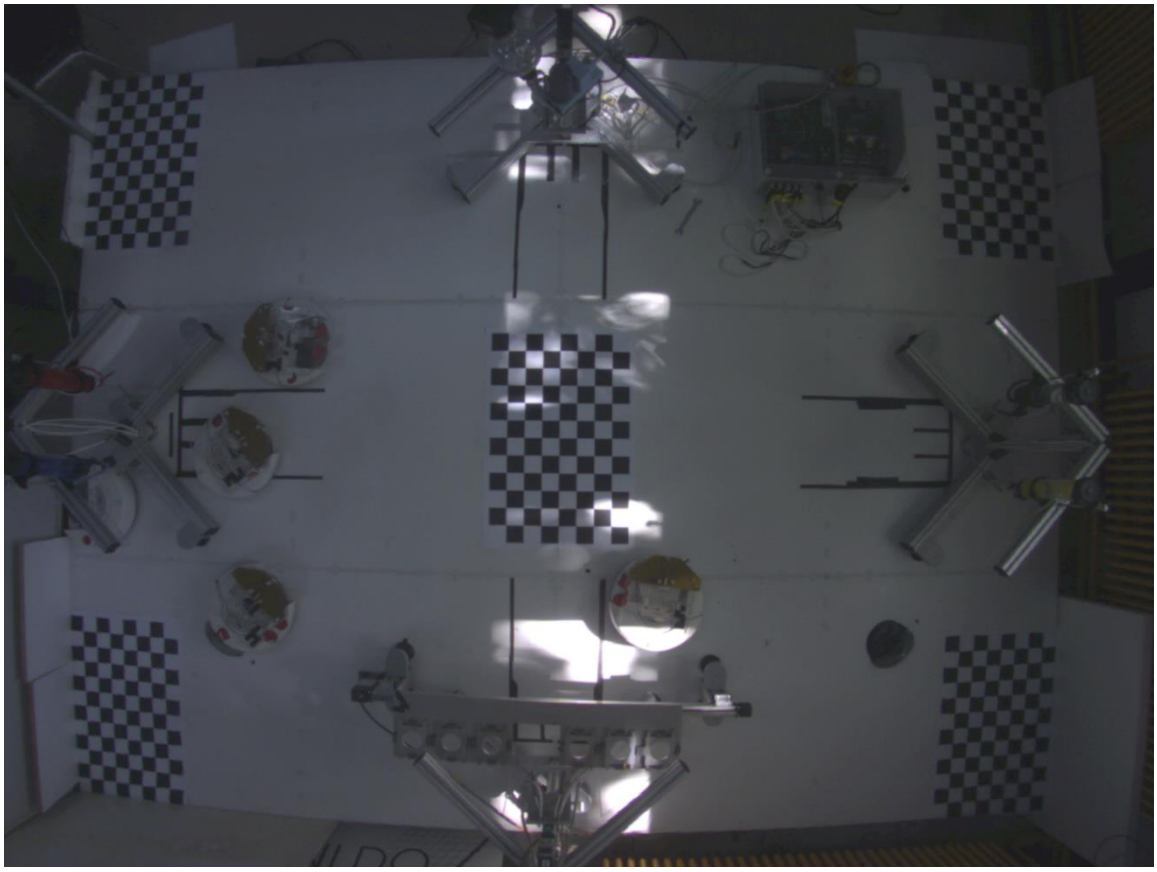

Figure 7: The remapped image using all five chessboards for calibration

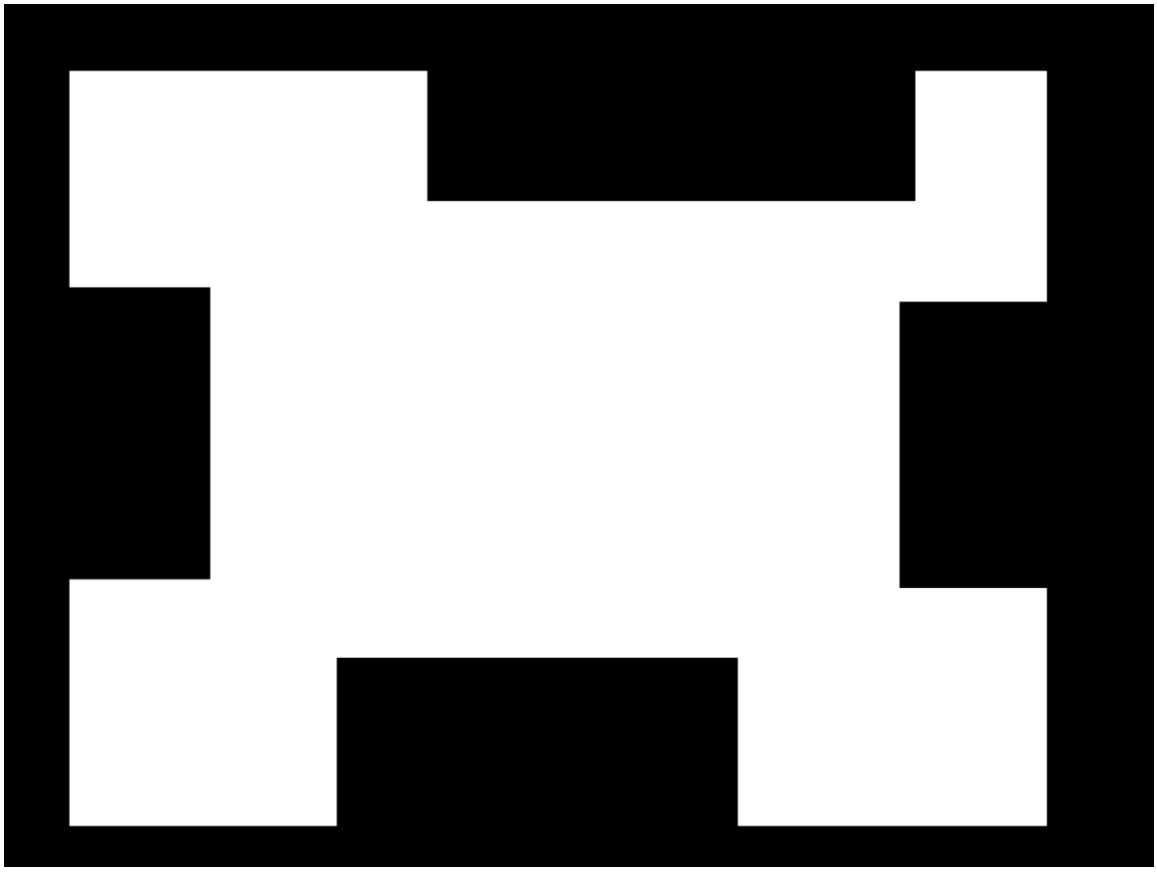

Figure 8: A dynamically created mask

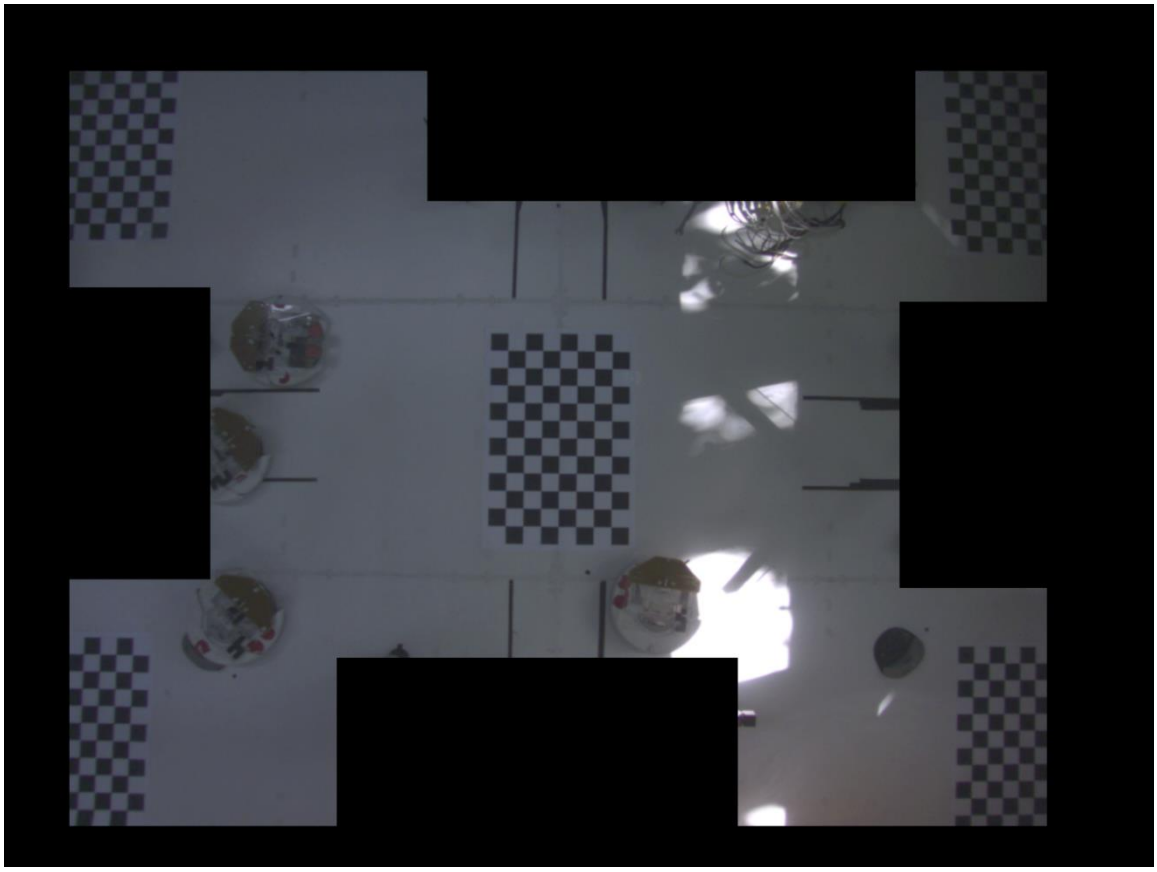

Figure 9: A remapped image after applying the mask