Annotation of Calibration Patterns for RGB-LiDAR Evaluations using Segmentation Models

Bruno Silva
PhD Student
Department of Mechanical Engineering
University of Aveiro
Aveiro, Portugal

Gonçalo Ribeiro
PhD Student
Department of Mechanical Engineering
University of Aveiro
Aveiro, Portugal

***Abstract*—Extrinsic calibration is a crucial step in sensor fusion. This work presents a deep-learning-based augmentation to ATOM, a multi-sensor and multi-modal calibration framework, for automating the annotation of the outer borders of calibration patterns, eliminating the need for manual labeling. Instead of predicting the outer corners directly with a CNN/FC combo, which proved infeasible due to variable output sizes, we trained segmentation models to detect the pattern and later extracted the edges using classical computer vision techniques. After testing various architectures, a U-Net model with a pretrained ResNet50 backbone delivered the best results.**

***Index Terms*—Extrinsic Calibration, Robot Calibration, Calibration Pattern, Machine Learning**

I. Introduction

Extrinsic calibration is a fundamental process in robotics vision that involves determining the relative pose (position and orientation) between different sensors, known as *sensor to sensor calibration*, or between a sensor and a known reference frame, which is known as *sensor to coordinate frame*. Extrinsic calibration is crucial because it allows for the accurate integration of data from multiple sensors, enabling sensor fusion. For instance, in an autonomous vehicle, the visual information of the camera needs to be accurately aligned with the distance measurements of the LiDAR to build a coherent understanding of the surroundings. Similarly, in robot arms, the position of the camera position relative to the end-effector must be precisely known to perform tasks like object manipulation [1].

Usually, iterative approaches are used. These rely on a cost function specific to a sensor modality but usually suffer from ambiguities in some way or another. An typical cost function for an RGB camera relies on computing the difference between the projection of the detection in the 2D image of some key points into a coordinate frame where these key points are precisely known. The objects that contain these precisely known points are called *Calibration patterns*. The most common types are chessboards and *ChArUcos*. *ChArUcos* are chessboards with unique identifiable symbols embedded on each square that allow computer vision algorithms to decipher if the pattern is upside down or if the framing of the image cuts part of the calibration object off. The issue with the aforementioned RGB cost function is that they have multiple local minima and not always converge. A simple example is to picture an RGB camera fixed on the end of a prismatic joint with the calibration pattern in front of the sensor, perfectly perpendicular to it. Both moving the offset of the joint or moving the RGB sensor on the mount can lead to the same relative distance between the sensor and the pattern, leading to equal detections of the key points, thus leading to ambiguity [1].

Despite the shortcomings mentioned, these cost functions have the advantage of generally performing more accurately in larger systems, as there is much more variety of data, and can still be used effectively in most simpler systems, as their requirements are only the existence of a sensor and a pattern. However, this creates the need for true evaluation procedures to assess the quality of the calibration results and to make them comparable in order to be publishable [1].

This project tackles an improvement to a previously fully manual and cumbersome evaluation method between RGB and LiDAR sensors, integrated on ATOM [1], a well established multi-sensor multi-modal calibration framework in the scientific community. The working principle is that by knowing the physical outer limits of the pattern in the 2D image, these points can be projected into the coordinate frame of LiDAR sensor, provided that the camera intrinsics are known. Afterward, the 3D points resulting from the projection can be directly compared with the 3D points of the outer border of the pattern. This comparison is not ambiguous as the projected points only line up in the 3D frame if the geometric transformations required for the projection are indeed correct. In ATOM, the 3D border points are already priorly labeled as they are a requirement for the cost function that optimizes the pose of the 3D LiDAR sensors. However, the 2D points of the border are not required by the RGB cost function. The current solution is a manual border labeling method, as a simple automatic method would struggle with detecting orientation and deal with edge cases. [2] This project aims to develop an automatic method using deep learning to simplify the evaluation pipeline to the user. Figure 1 and Figure 2 represent an example input and a desired output. On the output, each color encodes the border of one of the sides of the calibration pattern.

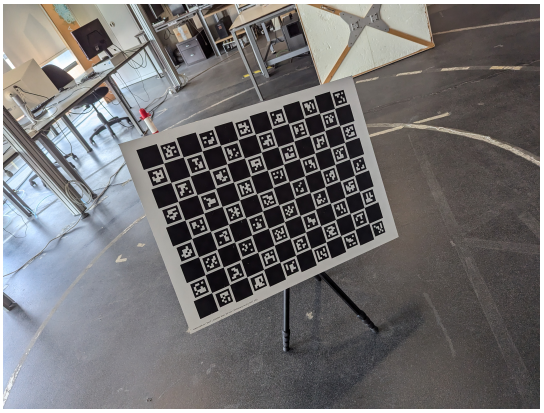

Fig. 1: Example of an input image for the automatic labeler we aim to develop.

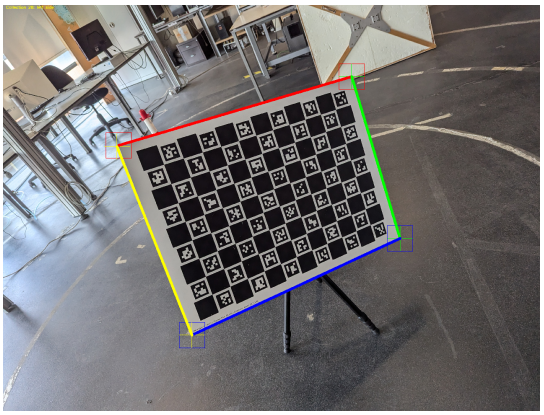

Fig. 2: Expected output from the automatic labeler, where the top border is marked in red, the right border in green, the bottom border in blue, and the left border in yellow.

II. Proposed Approach

A. Data preparation

We initially had a dataset consisting of images with labeled outer corners of a ChArUco board, specifically the four corners that define the physical boundary of the pattern. To improve dataset diversity, we captured additional images featuring different types of calibration patterns.

To facilitate data annotation, we developed a program to label these outer corners bypassing the usual calibration pipeline necessary to obtain these annotations. Additionally, we created a program to convert the labeled corners into segmentation masks, which were later used for model training.

B. Why use segmentation models

Using a simpler CNN/FC combination to directly predict the coordinates of the outer corners of the ChArUco board was not feasible because the number of output points is not constant. Designing a neural network with dynamic outputs would introduce significant complexity.

In many cases, at least one of the board's outer corners is clipped in the image. When this happens, five points are needed instead of four to properly define the pattern's edges. If two adjacent corners are clipped, only three sides remain visible.

To address this challenge, we opted to predict the segmentation mask of the pattern instead. This allowed us to later extract the edges using classical computer vision techniques.

C. Model selection and training setup

To select the models, we prioritized the availability of source code and pre-trained weights. The PyTorch Vision framework currently provides three models for semantic segmentation: DeepLabV3 [3], FCN [4], and LRASPP [5]. Among these, we selected DeepLabV3, as the authors reported the best results.

In addition to the PyTorch models, we sought a simple and well-established model to serve as a baseline. For this purpose, we chose U-Net [6].

Since this problem involves only one class of interest—the checkerboard—with all other pixels considered background, it can be framed as a binary classification task at the pixel level. Consequently, we selected a loss function designed for binary classification: binary cross-entropy loss.

For optimization, we experimented with the Adam optimizer due to its fast convergence and adaptive learning rates. Additionally, we evaluated Adam with Weight Decay to mitigate potential overfitting.

For batch size, we used the maximum value that the GPU could accommodate.

We attempted to implement an automatic stopping criterion based on loss and accuracy. Training would stop if the loss did not decrease for five consecutive epochs or if the accuracy did not improve within the same period. However, this approach led to premature stopping due to the instability of training, as discussed in the next section. Consequently, we opted to train for a fixed 200 epochs and retain the model with the highest accuracy.

III. Results

As discussed in subsection II-C, we utilized the DeepLabV3 and U-Net models. For DeepLabV3, PyTorch offers three backbone options: MobileNetV3-Large, ResNet-101, and ResNet-50. We selected ResNet-50 due to its smaller size. For U-Net, we initially trained a model from scratch and later experimented with a version using a pre-trained ResNet-50 as the backbone.

A. Transfer Learning with DeepLabV3 and Resnet50 Backbone

Using transfer learning with a pretrained DeepLabv3 model was the first option considered. DeepLabV3 is a complex model designed for image segmentation. Including the Resnet50 backbone, which is a CNN, the model has in total around 41M parameters. Upon freezing the backbone, we were left with 17M trainable parameters. The pretrained model came from PyTorch built-in models and was originally trained on a subset of known COCO dataset, using only the 20 categories

that are present in the Pascal VOC dataset, mainly consisting of big every objects like cars, planes, sofas, bicycle...

The train was conducted using the Adam optimizer and cross entropy loss, batch size of 4 and 10 epochs. Our dataset had 80 images. We found the loss to stay constant and high, meaning our model was not learning anything from the training. We figured that the dataset that the model was originally trained on was too different from ours and the model could not learn within a feasible number of epochs with only 80 images. In addition to that, we latter found out that this model has an auxiliary output that must be used in the training phase. We couldn't figure out how to deal with this auxiliary output, so we decided to attempt using simpler segmentation model, the U-Net.

B. U-net trained from scratch

After encountering difficulties with DeepLabV3, we shifted our approach to training a U-Net model from scratch. Given the complexity of implementing U-Net directly from its original paper with our current expertise, we opted instead to find an existing implementation. We adapted one from an online tutorial[1]. to suit our needs.

The chosen U-Net model comprised 31 million trainable parameters, utilizing a batch size of 20 and processing images at a resolution of 512x512 pixels. While the training process was notably unstable, as evidenced by Figures 3 and 4, which illustrate the loss and DICE curves, it was far better than DeepLabV3. The model achieved a peak validation DICE score of 78% at epoch 157.

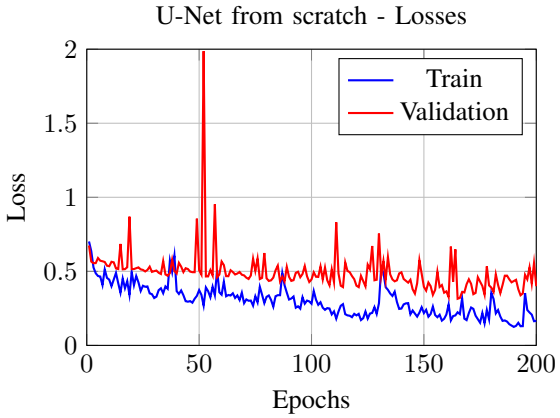

Fig. 3: Training and validation loss curves of the U-Net model trained from scratch for 200 epochs.

For enhanced clarity in visualizing the training dynamics, Figures 5 and 6 present smoothed versions of the loss and DICE curves up to epoch 157. The smoothing was achieved using a Savitzky-Golay filter, which helps in highlighting trends over noise.

[1]https://medium.com/@fernandopalominocobo/mastering-u-net-a-step-by-step-guide-to-segmentation-from-scratch-with-pytorch-6a17c5916114

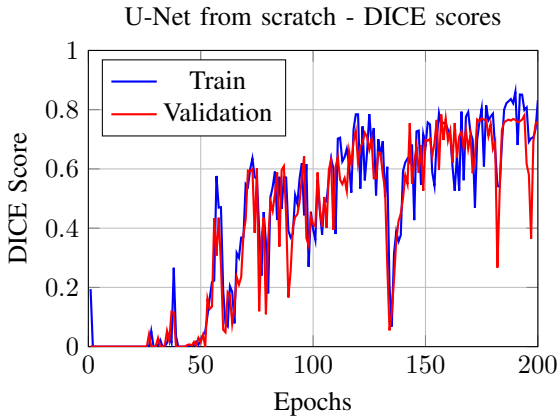

Fig. 4: Training and validation DICE score curves of the U-Net model trained from scratch for 200 epochs.

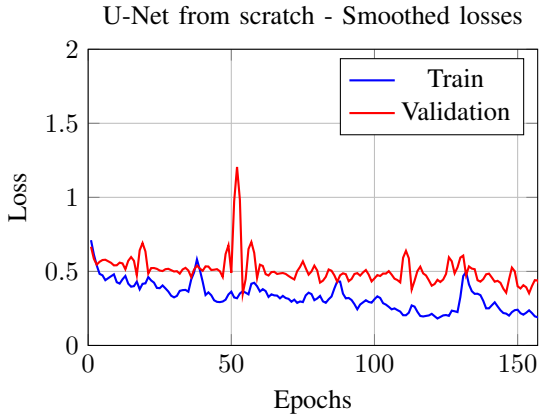

Fig. 5: Training and validation loss curves of the U-Net model trained from scratch until reaching the maximum DICE score (157 epochs).

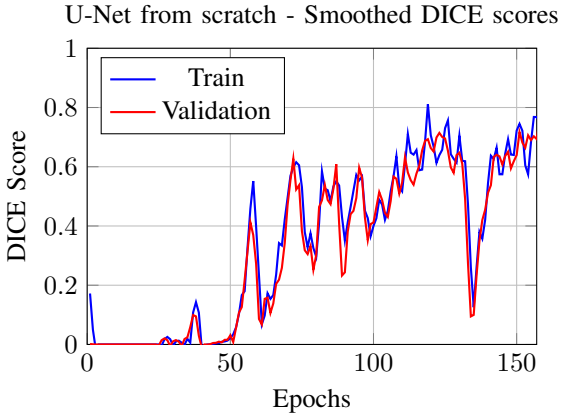

Fig. 6: Training and validation DICE score curves of the U-Net model trained from scratch until reaching the maximum DICE score (157 epochs).

C. U-net with pretrained Resnet50 Backbone

After experimenting with training a U-Net model from scratch, we explored modifying its backbone architecture by incorporating a pretrained ResNet50. Similar to our initial approach, we sought an existing implementation of this configuration and adapted one found on GitHub[2].

However, there was an unexpected discrepancy in the parameter count. The original U-Net model from scratch had 31 million parameters, but adding a ResNet50 backbone—which should have contributed approximately 50 million parameters—resulted in a total of 148 million parameters instead of the expected 80 million. This suggests that either this or the previous implementation may have contained an error.

Despite this issue, we began by training the model with the ResNet50 backbone's parameters frozen, leaving us with 124 million trainable parameters. Ideally, only the 50 million backbone parameters should have been frozen, but our approach was imperfect. Nevertheless, we proceeded with the training using a batch size of 19 and processing images at a resolution of 512x512 pixels.

The resulting training process was significantly more stable compared to the U-Net trained from scratch, as demonstrated in Figures 7 and 8.

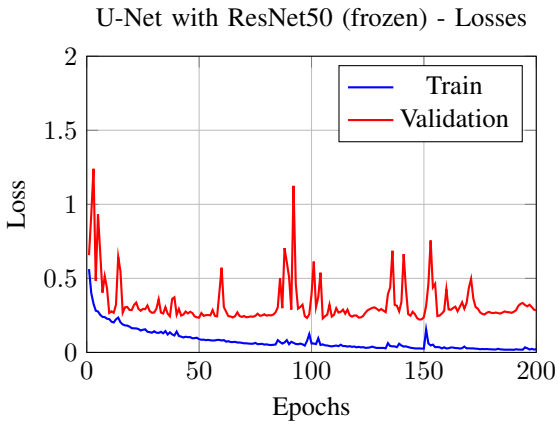

Fig. 7: Training and validation loss curves of the U-Net model with the pre-trained ResNet50 as backbone (frozen) trained for 200 epochs.

The model achieved a peak validation DICE score of 85% at epoch 142. Similarly to the U-Net from scratch, for enhanced clarity in visualizing the training dynamics, Figures 9 and 10 present smoothed versions of the loss and DICE curves up to epoch 142.

[2]https://github.com/rawmarshmellows/pytorch-unet-resnet-50-encoder

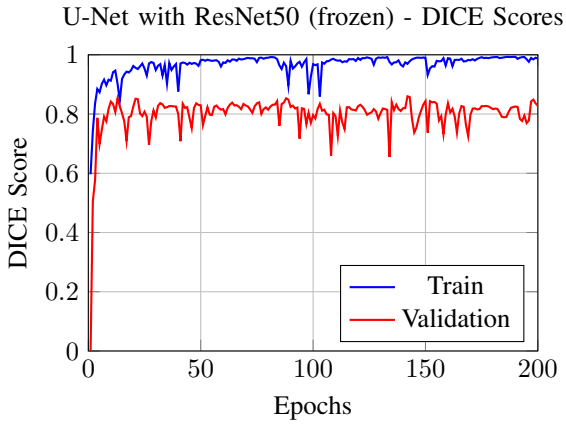

Fig. 8: Training and validation DICE score curves of the U-Net model with the pre-trained ResNet50 as backbone (frozen) trained for 200 epochs.

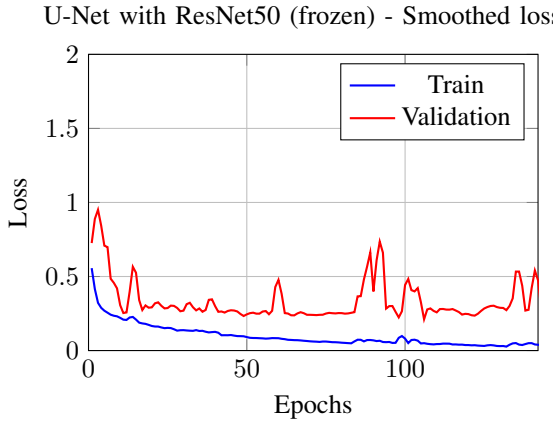

Fig. 9: Training and validation loss curves of the U-Net model with the pre-trained ResNet50 as backbone (frozen) until reaching the maximum DICE score (142 epochs).

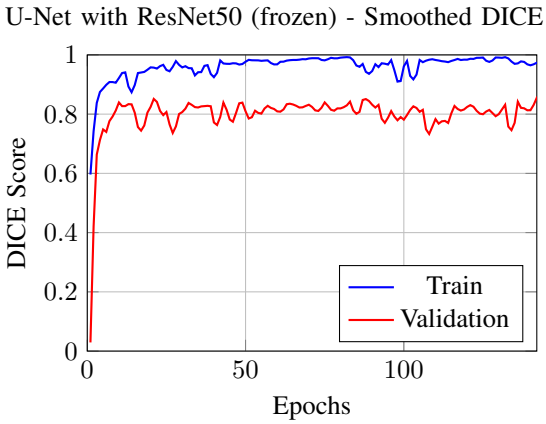

Fig. 10: Training and validation DICE score curves of the U-Net model with the pre-trained ResNet50 as backbone (frozen) until reaching the maximum DICE score (142 epochs).

In the final experiment, we trained all 148 million param-

eters without freezing, using a batch size of 14 and 512x512 pixel images. The outcomes are shown in Figures 11 and 12.

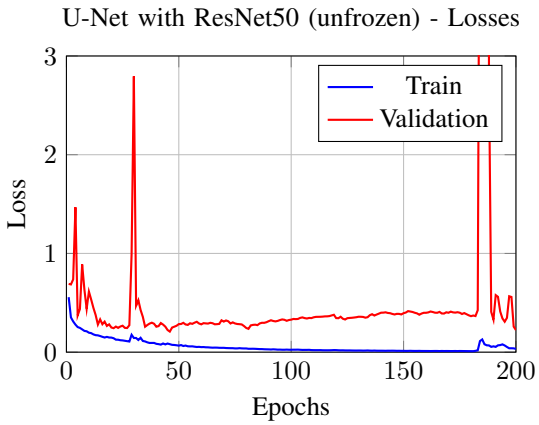

Fig. 11: Training and validation loss curves of the U-Net model with the pre-trained ResNet50 as backbone (unfrozen) trained for 200 epochs.

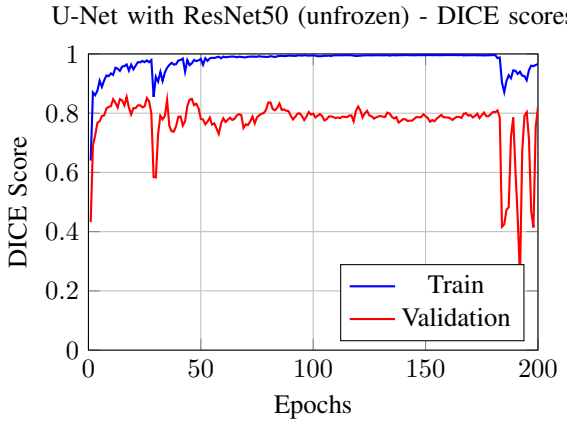

Fig. 12: Training and validation DICE score curves of the U-Net model with the pre-trained ResNet50 as backbone (unfrozen) trained for 200 epochs.

The model achieved a peak validation DICE score of 85% at epoch 17. Surprisingly, this peak was reached much earlier than in the other experiments. When examining the smoothed results up to epoch 17, as shown in Figures 13 and 14, the training appears significantly more stable.

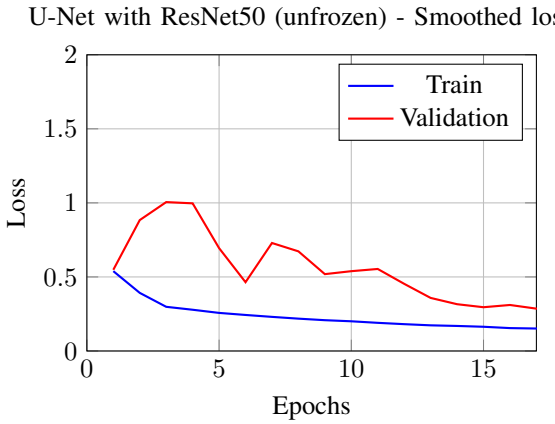

Fig. 13: Training and validation loss curves of the U-Net model with the pre-trained ResNet50 as backbone (unfrozen) until reaching the maximum DICE score (17 epochs).

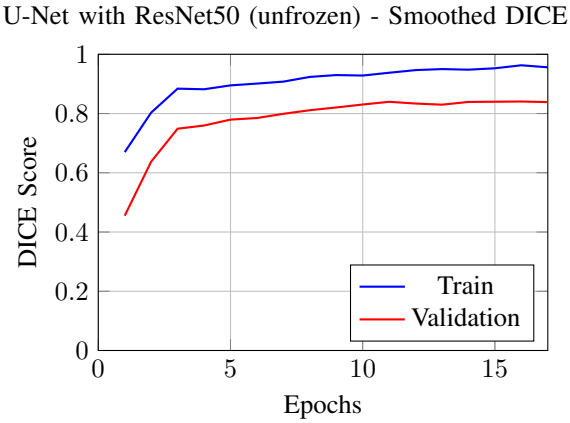

Fig. 14: Training and validation DICE score curves of the U-Net model with the pre-trained ResNet50 as backbone (unfrozen) until reaching the maximum DICE score (17 epochs).

IV. Conclusion

We presented an approach to automate the annotation of the outer borders of calibration patterns using deep learning. By leveraging segmentation models instead of direct CNN/FCN combinations, we addressed challenges related to pattern occlusions and variable output sizes.

While transfer learning with DeepLabV3 proved more challenging than anticipated and did not yield satisfactory results, we successfully employed U-Net models. All U-Net models exhibited overfitting, indicating that our dataset was neither sufficiently large nor diverse. However, employing a pre-trained ResNet50 backbone helped mitigate this issue, achieving an accuracy of 85% on the validation set. Figure 15 displays examples of the model's predictions.

This work served as a successful proof-of-concept, confirming that this approach holds promise for developing an automated labeling solution that reduces manual effort without compromising accuracy.

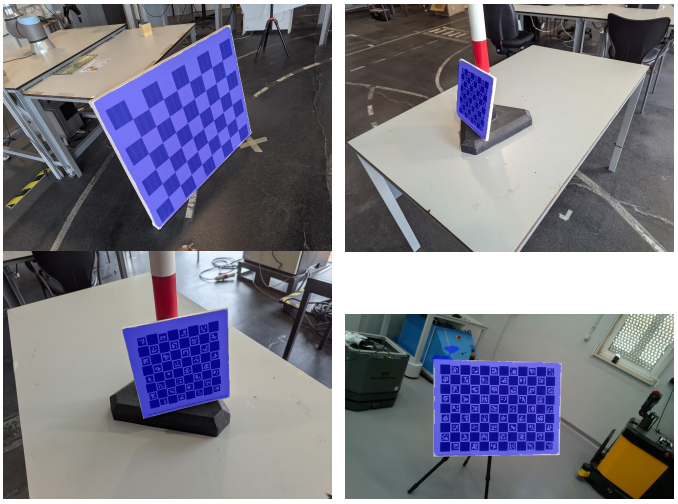

Fig. 15: Some examples of the predictions of the best model

Future work could focus on obtaining and annotating additional images, as well as implementing data augmentation techniques. If these measures do not resolve the overfitting issue, further optimizing or refining the model architecture could be considered.

V. References

[1] M. Oliveira *et al.*, "ATOM: A general calibration framework for multi-modal, multi-sensor systems," *Expert Systems with Applications*, vol. 207, p. 118 000, Nov. 30, 2022. DOI: 10.1016/j.eswa.2022.118000.

[2] A. S. Pinto de Aguiar, M. A. Riem de Oliveira, E. F. Pedrosa, and F. B. Neves dos Santos, "A camera to LiDAR calibration approach through the optimization of atomic transformations," *Expert Systems with Applications*, vol. 176, p. 114 894, Aug. 15, 2021. DOI: 10.1016/j.eswa.2021.114894.

[3] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, *Rethinking atrous convolution for semantic image segmentation*, Dec. 5, 2017. DOI: 10.48550/arXiv.1706.05587. arXiv: 1706.05587[cs].

[4] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 640–651, Apr. 2017. DOI: 10.1109/TPAMI.2016.2572683.

[5] A. Howard *et al.*, "Searching for MobileNetV3," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2019, pp. 1314–1324. DOI: 10.1109/ICCV.2019.00140.

[6] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds., Cham: Springer International Publishing, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28.