Report Title:

Intrinsic Camera Calibration and Camera – Radar Extrinsic Calibration

Report Objectives:

- Explore the technical background of camera intrinsic calibration, including the pinhole model and projective/planar transformations.
- Explore the technical background of radar–camera extrinsic calibration.
- Detail the methodology for obtaining intrinsic and extrinsic parameters.
- Present and analyze the calibration results, including reprojection errors, in the context of object-level fusion.

Task Description and Executive Summary

This report discusses the theory behind calibration and its role in radar–camera data fusion. Intrinsic calibration was performed using MATLAB's CameraCalibrate toolbox to obtain the camera parameters (f_x, f_y, $c_x, c_y, and\ skew$) and radial/tangential distortion coefficients, achieving a reprojection error of 0.21 pixels. Extrinsic parameters were obtained via the EPnP algorithm applied to correspondences between radar measurements and camera pixel points, with Gauss–Newton optimization. The resulting reprojection error was 54.365 pixels, which is acceptable given the image size ([4056 × 3040]) and that object-level fusion is implmented.

Work Contribution:

This report is prepared by Shayma Alteneiji

Table of Contents

Why is This Step Necessary for Data Fusion?

In data fusion algorithms, the predictions or measurements from multiple sensors are combined to produce a more reliable output with richer information. In radar–camera fusion, the algorithm confirms that both sensors detect the same object at the same position, reducing the chance of false detections.

But this introduces a key challenge: each sensor has a different point of view and uses a different coordinate frame. The camera uses a front view (FV) and treats its optical centre as the origin, with its axes defined as z-forward, y-down, and x-right. The radar, on the other hand, uses a bird's-eye view (BEV), with the radar at the origin and axes defined as z-up, y-forward, and x-right.

Hence, even if the radar and camera are placed as close as possible to maximize the overlap between their fields of view (FoV), there will always be some translational and rotational offset between the sensors' coordinate frames, and the same targets viewed by each sensor would appear at different positions.

The mechanism required to solve this issue should, in a way, rotate and translate the coordinate frames of one sensor such that the origin points and viewpoint are aligned. Then, cross-validation between the detected objects' positions by both sensors can be reliably performed. This is where calibration comes in.

The intrinsic camera calibration allows us to map 3D points (xc, yc, zc) in the camera's frame to their respective 2D pixel coordinates (u, v). The extrinsic calibration between the radar and camera enables us to project the radar 3D coordinate points into the camera's frame, or vice versa, using inverse projection.

Hence, once the calibration parameters are applied, the coordinates can be unified to either the camera's FV or the radar's BEV, and object cross-validation can be performed. In this report, we obtain these parameters offline. They will be used in real-time to continuously project the generated set of one sensor's data points into the chosen reference frame.

The figure below (Figure 1) shows how this step fits into the data fusion algorithm. In the next section, we discuss the projective transformation, a key tool in performing the coordinate transformation.

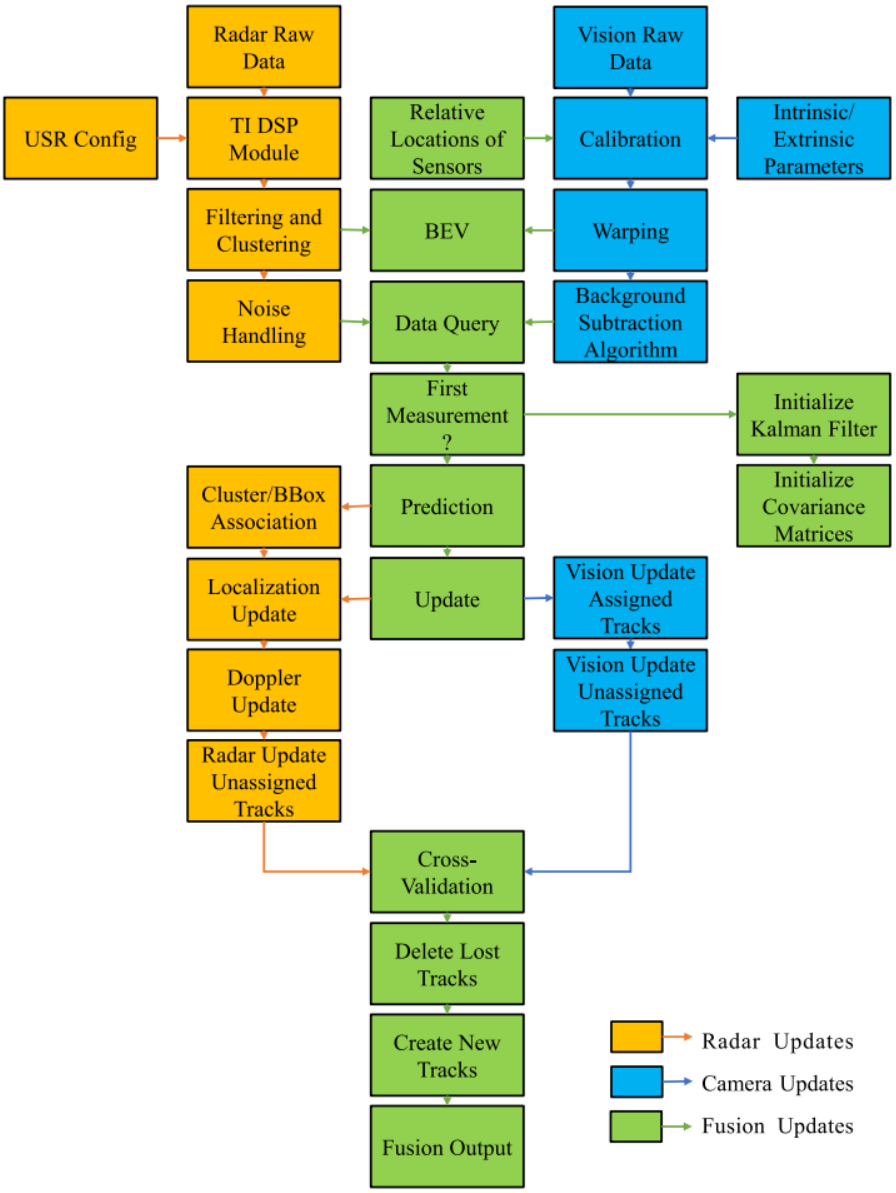


Figure 1. Radar – camera object level fusion algorithm using the Kalman filter [1].

Projective Transformation and Homogeneous Coordinates System

In the previous section, we touched on how extrinsic parameters allow us to transform perspectives from one sensor's point of view to another, as illustrated by Figure 2. We also discussed how intrinsic parameters enable the mapping from 3D camera coordinates to 2D image plane coordinates, as illustrated by Figure 3. These two operations together define what is known as the projective transformation [2, 3].

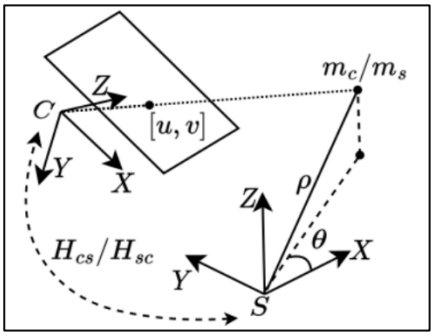


Figure 2. Extrinsic parameters used to transform from 3D camera coordinate frame to 2D image plane [3].

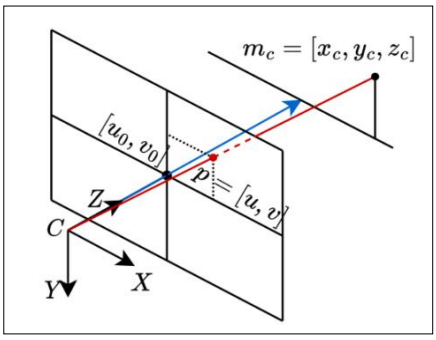


Figure 3. Intrinsic parameters used to transform from 3D camera coordinate frame to 2D image plane [3].

The general expression for this transformation in the case of perspective projection onto the image plane, is given below, where (X, Y, Z) is a real-world coordinate point [2, 3]:

$$k\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P_{3\times4}\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

When the world coordinate points all lie on the same plane (Z = 0), we have a special case of this transformation called homography or plane-to-plane transformation [2, 3], expressed as:

$$k\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = H_{3\times3}\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Matrix multiplication performs a linear coordinate transformation; where the point of origin remains unchanged, while the coordinate system's axes and basis vectors only change. Therefore, by multiplying the coordinates (X, Y, Z) by the appropriate matrix ($P_{3\times4}$), we can transform the view from the perspective of the real-world coordinate (specifically, with respect to some chosen origin point) to that of the camera's perspective in (u, v). The point (u, v) is measured with respect to the optical centre or the centre of the image plane, given by (c_x, c_y).

Matrix multiplication, however, does not allow us to also shift the origin point. This is resolved through the use of homogeneous coordinate system. It allows full perspective transformation, whether it is translation, rotation, or scaling using a single matrix multiplication [2].

In homogeneous coordinate systems, we represent the Euclidean (real-world) coordinate points (whether 2D or 3D) by appending an extra element, typically a 1, extending the Euclidean space R^n into a projective space $P^n = R^{n+1}$ [2]. Moreover, two points that are scalar multiples of each other, are considered equivalent, $(x, y, 1) \equiv (kx, ky, k)$ [2]. Dividing by k allows us to easily go back to the original form, where k is a non-zero scalar. With this property, homogenous coordinates allow us to model "parallel lines meeting at infinity" by defining the special case when k = 0 [2].

It is important to note that projective transformations do not preserve geometric shapes or parallelism. A circle may become deformed into an ellipse, and parallel lines can appear to intersect at a finite point. This occurs because, in projective transformation, all 3D points, even those with a zero as the last coordinate, are treated equally and get mapped to some set of finite points. This results in the loss of parallelism, as there is no explicit case to define a line at infinity in 2D projective space or a plane at infinity in 3D when k = 0 [2].

In our application, we are primarily interested in the camera's intrinsic parameters, rather than the full projective transformation matrix which encodes both the intrinsic and extrinsic parameters. Since the intrinsic parameter matrix K is an upper triangular matrix, and the extrinsic parameter matrix is orthonormal, we can extract both from the full $P_{3\times4}$ matrix using QR factorization [4]. In the next section, we delve deeper into the derivation of the intrinsic matrix K, specifically for the pinhole camera model.

Camera Model

In data fusion, the camera is used to produce bounding boxes around the targets using the chosen object detection method. In this section, we delve into the concepts of the camera module. A camera model describes the mapping from 3D world coordinates into 2D pixels coordinates [2]. While multiple models exist, each applies depending on the field view of the used camera. When the field of view is below 95°, the pinhole model is typically sufficient [5]. The full expression of the model is given below. The K matrix includes the intrinsic parameters, while [R | t], describe rigid body transformation or the extrinsic parameters matrix.

$$s\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = K_{3\times4}[R|t]_{4\times4}\begin{bmatrix} X \\ y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}\begin{bmatrix} X \\ y \\ Z \\ 1 \end{bmatrix}$$

To derive the elements of matrix K, we consider the following. The model assumes the camera has no lenses but instead a simple aperture, where rays from 3D points converge and pass through to form an inverted 2D image. The point at which the rays converge is called the centre of projection [6].

The model defines the mapped (u, v) pixel point to be collinear with the centre of projection and the corresponding 3D point (X_c, Y_c, Z_c) from the camera coordinate frame [2, 6]. In computer vision, it is more convenient to place the image plane in front of the projection centre to avoid dealing with a negative focal length. This pinhole model geometry is shown in Figure 4.

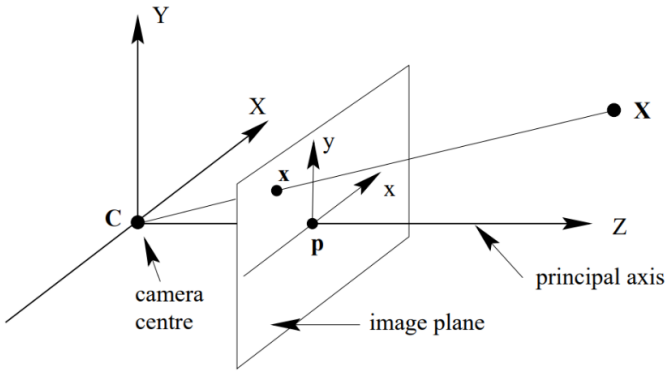


Figure 4. Pinhole model geometry with the image plane before the centre of projection point (C). p is the principal point. [2]

We can derive the coordinates of the (u, v) pixel corresponding to a given (X_c, Y_c, Z_c) point in terms of f using Figure 5. The figure illustrates how we can exploit the properties of similar triangles to obtain the expression. We can express u, and v as shown below. D is a scaling factor used to convert the focal length metric units into pixels [7]:

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \frac{D \times f}{Z_c} \begin{bmatrix} X_{i,c} \\ Y_{i,c} \end{bmatrix}$$

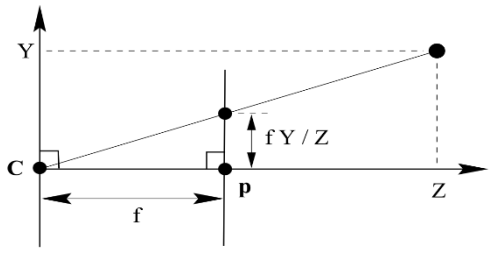


Figure 5. Similar tringles showing how derive the expression for the (u, v) pixel coordinates. [2]

The equation above presumes that the centre of the image plan is at the principal point **p** [2, 7] (Figure 5), in practice however this is not the case. It is actually at the upper left corner of the image plane [7].

Hence, to shift the origin to be the principal point, we shift the u coordinates of all points by c_x and the v points by c_y as shown below [7].

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \frac{D \times f}{Z_c} \begin{bmatrix} X_{i,c} \\ Y_{i,c} \end{bmatrix} + \begin{bmatrix} c_x \\ c_y \end{bmatrix}$$

It was stated above that homogenous coordinates allow us to perform multiple linear transformations simultaneously using a single matrix multiplication. The following equation illustrates the resulting matrix K. The reader may attempt to prove the equivalence of the two expressions (note that w_i is the scaler number k).

$$\begin{bmatrix} u_i w_i \\ v_i w_i \\ w_i \end{bmatrix} = \begin{bmatrix} D \times f & 0 & c_x & 0 \\ 0 & D \times f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} X_{i,c} \\ Y_{i,c} \\ Z_{i,c} \\ 1 \end{bmatrix}$$

In practice, there is a small difference between the focal length in the x and y directions due to lens effects. Hence, instead we define f_x and f_y separately. Also, element (1, 2) of the intrinsic matrix K can be non-zero and is used to model the skew between the camera x and y axes [7]. The final form of the intrinsic matrix K then becomes:

$$K = \begin{bmatrix} D \times f_x & skew & c_x & 0 \\ 0 & D \times f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The resulting model is referred to as the linear pinhole model [7]. The model is relatively simple, but it is not sufficient when high accuracy is required [7]. For that, we take into consideration (1) the radial distortions which are due to the bending of light near the edges more so than at the centre. (2) We also account for the tangential distortions which occur when the camera lens and the image 2D plane are not parallel [6, 7].

Typically, three parameters $(k_1, k_2, and\ k_3)$ are sufficient to model the radial distortion, while two parameters $(p_1,\ and\ p_2)$ model the tangential distortion [6]. The resulting model then becomes as expressed below. In the following subsection, we discuss the mathematical formulation of the extrinsic or rigid body transformation between the radar – and camera, aligning the sensor modules coordinate frame.

$$\begin{bmatrix} u_i w_i \\ v_i w_i \\ w_i \end{bmatrix} = \begin{bmatrix} D \times (f_x + \delta u_i^{(r)} + \delta u_i^{(t)}) & skew & c_x & 0 \\ 0 & D \times (f_y + \delta v_i^{(r)} + \delta v_i^{(t)}) & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} X_{i,c} \\ Y_{i,c} \\ Z_{i,c} \\ 1 \end{bmatrix}$$

The direct linear transformation (DLT), and the Levenberg-Marquardt (LM) Methods used for obtaining the 12 parameters of the camera model. In paper [7], the authors discuss the DLT method,

and the linear and non-linear methods used to estimate the parameters. They also delve into more advanced techniques for more accurate estimation, including solving the asymmetric image projection problem. The MATLAB toolbox CameraCalibrator, which we use for the intrinsic and distortion parameters estimation, was designed based on this published paper.

Radar – Camera Extrinsic Calibration

This section discusses the rigid body transformation required to align the radar and camera coordinate frames. The radar used in this project is the TI IWR6843, which outputs 3D data with the axes defined as: x – right, y – forward, and z – up. As stated above, the camera typically follows the order as x – right, y – down, and z – forward. We model the transformation from the radar to the camera coordinate frame using the equation given below.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = R \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} + t$$

Together, R and t form 12 parameters, expressed as given below.

$$[R|t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To estimate these parameters which model the camera pose with respect to the radar, we use the Efficient Perspective-n-Point (EPnP) algorithm. The algorithm requires at least 6 known correspondences between 3D radar data and 2D camera pixel coordinates. We obtain these correspondences by recording the image pixel points corresponding to a known target detected by the radar and repeat the process by changing the target's position in each iteration. The EPnP estimation results are refined using the Gauss–Newton optimization algorithm, implementation steps will be detailed in the methodology section.

FV versus BEV

We previously discussed that FV is the default point of view (PoV) of the camera sensor module, while BEV, or top-down view, is the default PoV for the radar sensor. We also discussed why it is essential to unify the PoV to a single reference frame. In this section, we discuss the rationale for selecting the fusion PoV, as well as its mathematical formulation.

In FV, the 3D radar measurements are projected into the camera image plane using the projective transformation matrix. In BEV, however, the 2D image pixel coordinates are projected into the radar 2D coordinate plane using the inverse projective mapping, or the inverse of the Homography matrix, a

plane-to-plane transformation. While FV may seem superior to BEV because it preserves the depth information of the detected objects when projected onto the image plane, contrast to BEV, which assumes all detections lie on the same radar Z plane, losing true depth information and introducing distortion.

In target tracking applications, however, BEV is preferred, as it is more robust against the crucial challenge of occlusion [1]. Moreover, in the literature on radar-camera data fusion, implementations tend to unify the coordinate frame to BEV. Therefore, we conclude to use BEV to project the YOLO-detected bounding box centroids into the radar plane $Z = z_{\text{projection}}$. The Homography matrix is derived from the projective transformation matrix as shown below:

$$k\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = P_{3\times4}\begin{bmatrix} X \\ Y \\ z_{projection} \\ 1 \end{bmatrix}$$

$$k\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = [col_1(P_{3\times4}) \quad col_2(P_{3\times4}) \quad col_3(P_{3\times4}) \times z_{projection} + col_4(P_{3\times4})]\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$H_{3\times3} = [col_1(P_{3\times4}) \quad col_2(P_{3\times4}) \quad col_3(P_{3\times4}) \times z_{projection} + col_4(P_{3\times4})]$$

The projected radar coordinate points obtained from the YOLO pixel coordinates are then calculated as follows. The k term is the scaling factor; dividing by k produces the normalized true (X, Y) coordinates:

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \frac{1}{k} H_{3\times3}^{-1}\begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

In the next section we discuss the hardware model considerations and implementation.

Hardware Model, Experimental Setup, and Equipment

To ensure reliable calibration between the radar and the camera, it is essential to keep their positions fixed throughout the process and throughout their usage in data fusion. Any shift in camera focus, camera position, or radar alignment would require performing the calibration. Therefore, we created a model that mounts the radar and camera tightly.

We placed the two sensors as close as possible to maximize the overlap between their fields of view. The model also accommodates for the processing unit (the Raspberry Pi) and the power bank used to power it. The model is shown in Figure 6. The positioning of the two sensors in the setup is crucial; the radar antenna is mounted on its narrower side, which is the side kept closer to the camera.

The radar mounter was part of the TI IWR6843 package; the camera mounter was 3D designed and printed as shown in Figure 7. The printed model was fully solid (100%) to ensure its strength in holding the camera. The equipment and components needed for the calibration step is listed below.

1. 35 cm x 40 cm wood mounting board
2. Main system components and connections (Sensor modules and the processing unit)
3. (2) Sensor mounts
4. Printed checkboard picture
5. Triangular radar reflector
6. Plastic plugs
7. 18 mm nails
8. 5 mm nails

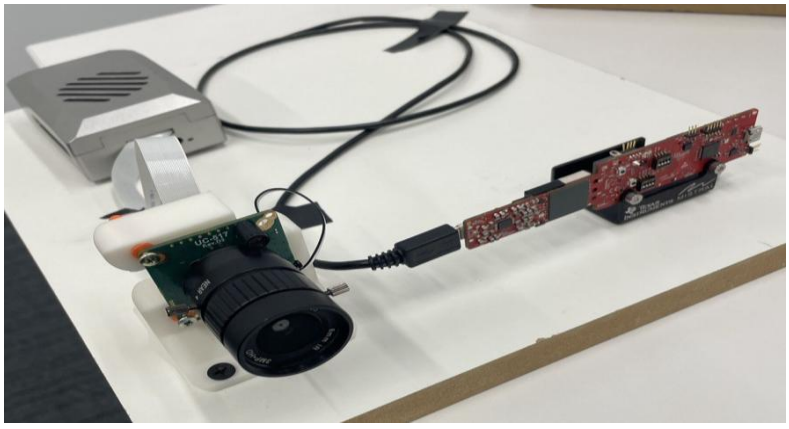


Figure 6. Hardware model.

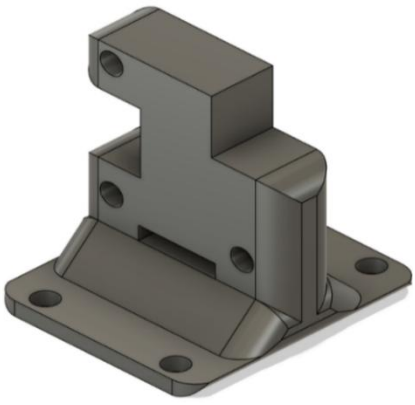


Figure 7. Camera stand 3D model.

The 5 mm nails and plastic plugs were used to attach the camera to the stand, while the 18 mm nails were used to anchor the two sensor mounts to the mounting board. The triangular radar reflector was used as the known target that can be reliably detected by the radar during the extrinsic calibration. For

the intrinsic calibration, the checkerboard pattern was printed and glued on thick cardboard to ensure its flatness. The used checkboard image is given in Appendix A. In the next section, we discuss the details of the calibration procedure, detailing how intrinsic and extrinsic parameters are obtained.

Methodology

Multiple calibration approaches were reviewed [8-11]. The selected method was chosen based on radar capabilities and practical deployability. More advanced techniques may be explored in future work. The adopted procedure is based on [8], and the overall intrinsic–extrinsic flow is shown in Figure 8. The following sections summarize the intrinsic calibration implementation steps and results.

Intrnsic Calibration

The intrinsic parameters were determined using MATLAB's cameraCalibrator toolbox. Images were captured with the HQ–IR Cut camera and uploaded into the toolbox. The initial reprojection error was 0.29 px (Figure 9). After removing outlier images, the error dropped to 0.21 px (Figure 10).

MATLAB's tools also allow visualization of the capture angles (Figures 11–12) and detection vs. reprojection points (Figure 13), showing minimal displacemen

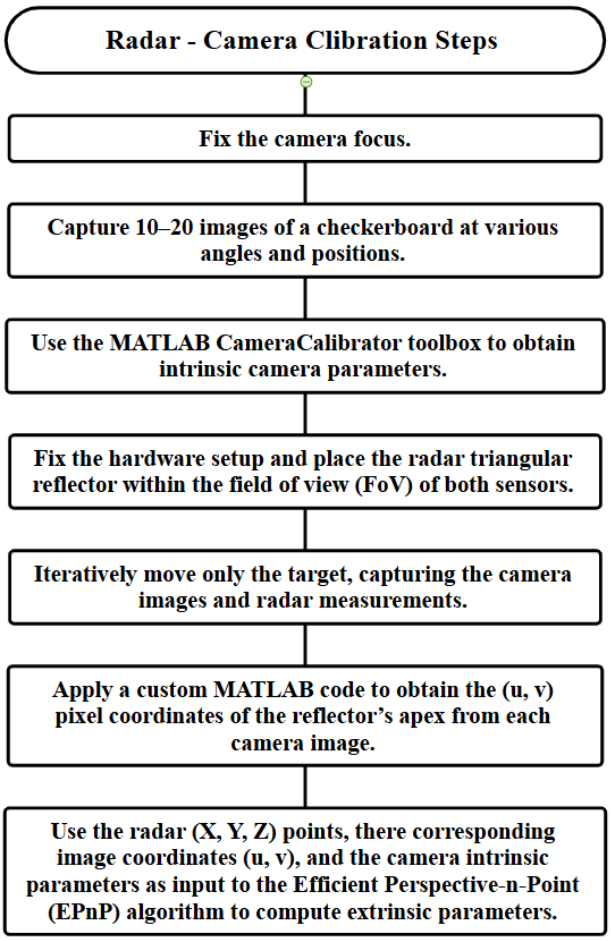


Figure 8. Full Radar-Camera calibration steps.

The resulting intrinsic matrix and distortion coefficients are:

$$K = \begin{bmatrix} 4053.6085 & -12.6845 & 1979.8909 \\ 0 & 4046.2944 & 1577.1950 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{k} = [\ -0.4916362112 \quad 0.2651989268 \quad 0\ ]$$

$$\boldsymbol{p} = [-0.0029367783 \quad 0.0040783507\ ]$$

Extrinsic Radar-Camera Calibration

To obtain the extrinsic parameters, we follow steps 4–7 of the flowcharts presented in Figure 8. A minimum of six images, along with their corresponding radar measurements of the radar reflector, must be collected. It is important to ensure a relatively significant change in the reflector's position for each iteration. Figures 14, and 15 illustrate examples of the gathered radar and camera measurements.

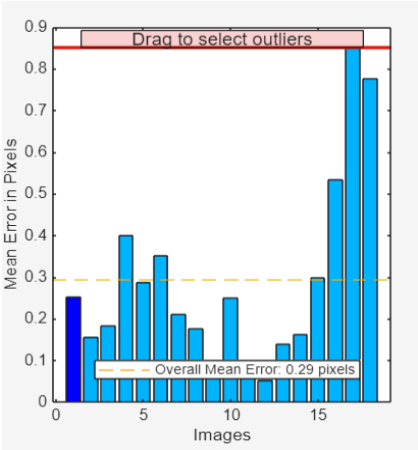


Figure 9. Initial mean reprojection error (in pixels) versus images chart

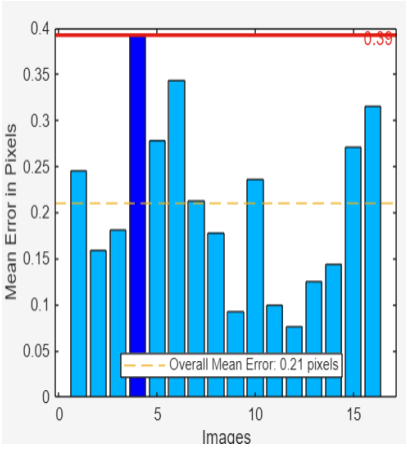


Figure 10. Final mean reprojection error (in pixels) versus images chart after adding more images and removing outliers.

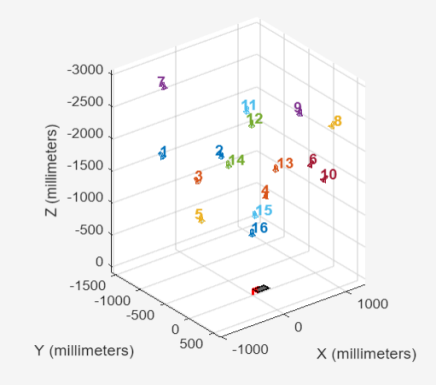


Figure 11. Extrinsic calibration illustration, showing the various angles at which the checkboard was captured.

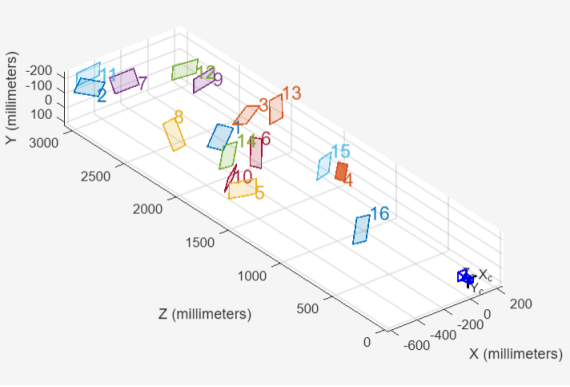


Figure 12. Extrinsic calibration illustration, showing the orientation of the checkboard with respect to the camera.

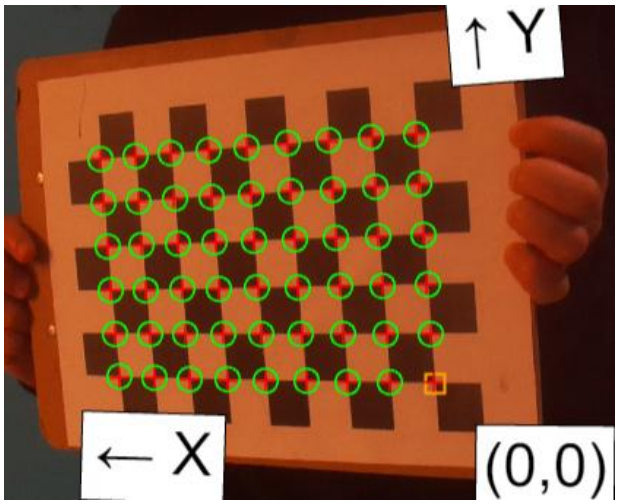


Figure 13. captured image with reprojected crosses plotted on the actual detected corners. This figure shows only a tiny reprojection error in pixels.

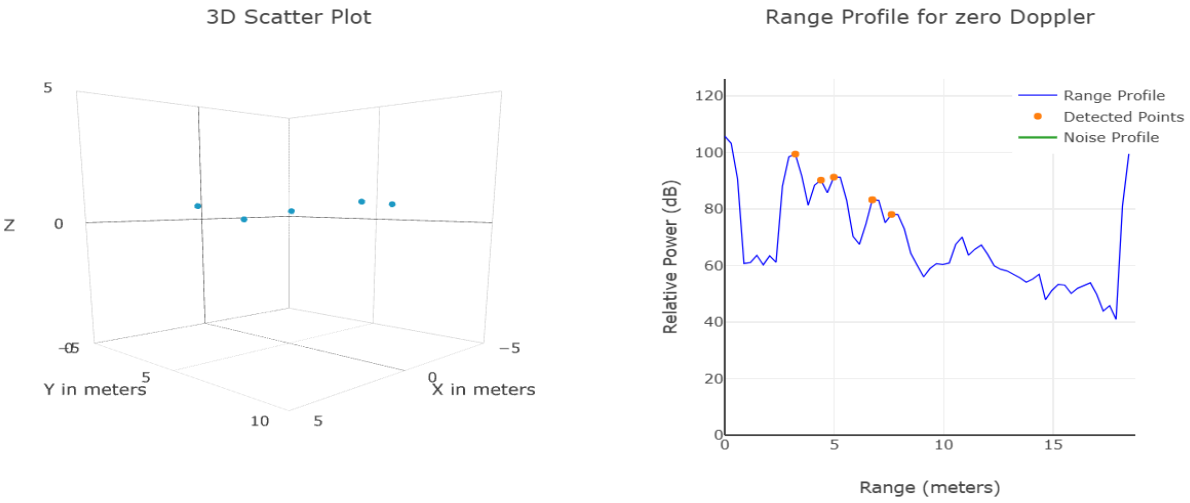


Figure 14. Radar measurement for a reflector in an extrinsic calibration setup on mmWave_Demo_Visualizer webpage

Figure 15. Reflector in Extrinsic Calibration Setup

Before proceeding further, it is necessary to confirm that the collected points are not planar, as this could result in degenerate solutions requiring additional points to be gathered. By visual inspection of the scatter plot, shown in Figure 16, we can verify that sufficient variation exists along all three axes.

Following this, the next step involves obtaining the (u,v) pixel coordinates corresponding to the apex of the radar reflector in each image. This is accomplished using the MATLAB script detailed in Appendicx B. The corresponding pixel coordinates are displayed on the image, as shown in Figure 17, and also printed to the command window.

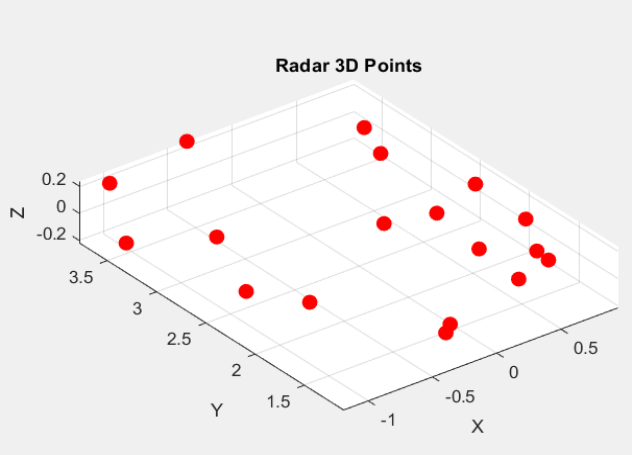


Figure 16. Radar measurement scatter plot used to verify whether there is sufficient variation along all three axes.

Figure 17. Image showing the pixel point corresponding to the apex of the corner reflector.

With the radar points, their corresponding pixel coordinates, and the intrinsic camera parameters available, the EPnP algorithm can be executed. A link to the GitHub repository containing the EPnP code is provided here [12]. Morover, the rotation (R) and translation (t) matrices obtained are given below.

$$R_{radar\rightarrow camera} = \begin{bmatrix} 0.9993975151 & -0.0000758999 & -0.0347073626 \\ -0.0347069625 & -0.0074618466 & -0.9993696751 \\ -0.0001831289 & 0.9999721571 & -0.0074599852 \end{bmatrix}$$

$$t_{radar\rightarrow camera} = [-0.1556986662 \quad 0.0378039639 \quad 0.0776697348]$$

The achieved reprojection error was 54.365 px (Figure 18). Although this may appear high, the image resolution is 4056×3040, and the fusion operates at the object level. Normalizing relative to the diagonal of the image gives: $\frac{54.365}{\sqrt{4056^2+3034^2}} \approx 0.0107$ approximatly 1% error, which is acceptable. In the nest section, we discuss the fusion algorithm implementation.

Iteration	Func-count	Resnorm	First-order optimality	Lambda	Norm of step
0	7	65550	4.38e+05	0.01	
1	14	59580.4	1.08e+04	0.001	0.0319013
2	21	59577.2	15.9	0.0001	0.000527034
3	28	59577.2	0.18	1e-05	4.96677e-06

Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to its initial value is less than the value of the function tolerance.

<stopping criteria details>
error EPnP_Gauss_Newton: 54.365

Figure 18. EPnP algorithm output results.

References

[1]. R. Zhang and S. Cao, "Extending Reliability of mmWave Radar Tracking and Detection via Fusion With Camera," *IEEE Access*, vol. 7, pp. 137065–137079, 2019, doi: https://doi.org/10.1109/access.2019.2942382.

[2]. R. Hartley, A. Zisserman, S. Wei, and Quanbing Zhang, *计算机视觉中的多视图几何 = Multiple view geometry in computer vision / Ji suan ji shi jue zhong de duo shi tu ji he = Multiple view geometry in computer vision*. 机械工业出版社, Beijing: Ji Xie Gong Ye Chu Ban She, 2020.

[3]. Mahdi Chamseddine, J. Rambach, and D. Stricker, "CaRaCTO: Robust Camera-Radar Extrinsic Calibration with Triple Constraint Optimization," pp. 534–545, Jan. 2024, doi: https://doi.org/10.5220/0012369700003654.

[4]. B. Fisher, "Projective Transformations," *homepages.inf.ed.ac.uk*, Nov. 07, 1997. https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/BEARDSLEY/node3.html

[5]. First Principles of Computer Vision, "Intrinsic and Extrinsic Matrices | Camera Calibration," *YouTube*, Apr. 18, 2021. https://www.youtube.com/watch?v=2XM2Rb2pfyQ (accessed Apr. 26, 2025).

[6]. "Fisheye Calibration Basics - MATLAB & Simulink," *www.mathworks.com*. https://www.mathworks.com/help/vision/ug/fisheye-calibration-basics.html (accessed Apr. 26, 2025).

[7]. "What Is Camera Calibration?," *Mathworks.com*, 2019. https://www.mathworks.com/help/vision/ug/camera-calibration.html (accessed Apr. 26, 2025).

[8]. J. Heikkila and O. Silven, "A four-step camera calibration procedure with implicit image correction," *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, doi: https://doi.org/10.1109/cvpr.1997.609468.

[9]. J. WEN, J. YANG, M. FU, J. ZHANG, and W. YANG, "A Calibration Algorithm for Maximum Likelihood Estimation of Extrinsic Parameters of a Camera and a 3D Laser Radar," *ROBOT*, vol. 33, no. 1, pp. 102–106, Aug. 2011, doi: https://doi.org/10.3724/sp.j.1218.2011.00102.

[10]. E. Wise, Juraj Peršić, C. Grebe, I. Petrović, and J. Kelly, "A Continuous-Time Approach for 3D Radar-to-Camera Extrinsic Calibration," *arXiv (Cornell University)*, pp. 13164–13170, May 2021, doi: https://doi.org/10.1109/icra48506.2021.9561938.

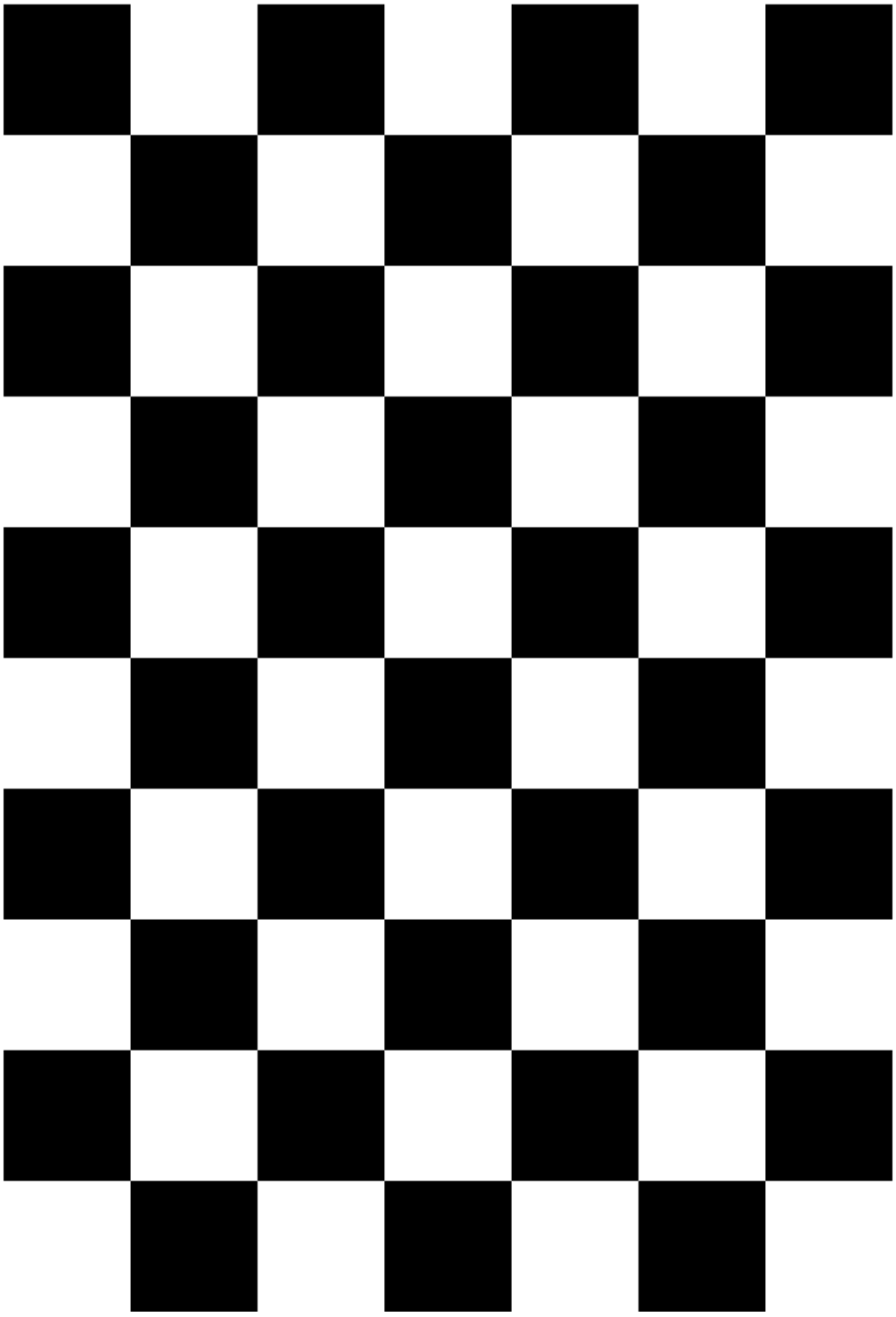
[11]. C. Song, G. Son, H. Kim, D. Gu, J.-H. Lee, and Y. Kim, "A Novel Method of Spatial Calibration for Camera and 2D Radar Based on Registration," Jul. 2017, doi: https://doi.org/10.1109/iiai-aai.2017.62.

[12]. D. Kim and S. Kim, “Extrinsic parameter calibration of 2D radar-camera using point matching and generative optimization,” pp. 99–103, Oct. 2019, doi: https://doi.org/10.23919/iccas47443.2019.8971568.

[13]. Adynathos, “EPnP: An Accurate O(n) Solution to the PnP Problem,” *Github.com*, 2025. https://github.com/cvlab-epfl/EPnP.

Appendices

Appendix A – Checkboard Image



Appendix B – MATLAB Code to Manually Select the (u, v) pixel points

```
% The Python code for UV pixel selection was mostly developed with assistance
from  %ChatGPT (OpenAI, 2023).

clc;clear;

image= "camn2.jpg";

% set the number of reflectors or targets in the image

num_reflector = input("Number of Reflectors: ");

while(num_reflector == [])

    % wait

end

imshow(image ); % displays the image

for k = 1:num_reflector

    [x, y] = ginput(1); % manually select the target
    x= round(x); y = round(y);
    hold on
    xi(k) = x;
    yi(k) = y;

    plot(xi(k), yi(k), 'x', 'MarkerSize', 10, 'LineWidth', 1); % plot an x at the
point

    str = sprintf('u = %d\nv = %d', xi(k), yi(k));

    % Adds a text box with the u, and v values
    text(xi(k) + 50, yi(k) -40, str, ...
        'Color', 'white', ...
        'FontSize', 10, ...
        'BackgroundColor', [3/255, 102/2555, 252/255], ...
        'EdgeColor', 'white', ...
        'Margin', 4, ...
        'FontWeight', 'bold');
    hold on
    if (k == 1)
    fprintf("The pixel value corresponding to the %d-st point is: (%d, %d) \n", k,
xi(k), yi(k))

    elseif (k ==2)
        fprintf("The pixel value corresponding to the %d-nd point is: (%d, %d)
\n", k, xi(k), yi(k))

    elseif (k == 3)
        fprintf("The pixel value corresponding to the %d-rd point is: (%d, %d)
\n", k, xi(k), yi(k))

    else
        fprintf("The pixel value corresponding to the %d-th point is : (%d3, %d)
\n", k, xi(k), yi(k))
    end
end
```