Data Visualization and Simulation Software for Testing Calibration Algorithms

Doha Djellit
Eötvös Loránd University
Computer Science for Autonomous Systems
Supervisor: **Tófalvi Tamás**

January 27, 2025

1 Overview

In recent years, the rapid advancement of autonomous systems, robotics, and computer vision has increased the demand for accurate and reliable sensor simulation tools. These tools are essential for developing and testing algorithms in a controlled environment before deploying them in real-world scenarios. This project focuses on the development of a Unity-based application capable of simulating 3D LiDAR and camera systems, providing users with a flexible platform to configure sensor parameters, place calibration targets, and save simulated data for further analysis.

The application is designed to simulate the behavior of LiDAR and camera sensors, including their intrinsic and extrinsic parameters. Users can interactively adjust the position, orientation, and properties of these sensors, as well as place common calibration targets such as planes, chessboards, cylinders, and spheres. The simulated data generated by the application can be exported and used to validate calibration algorithms or train machine learning models.

To ensure the accuracy of the simulated sensor data, the intrinsic and extrinsic parameters provided by the application were rigorously tested using MATLAB.

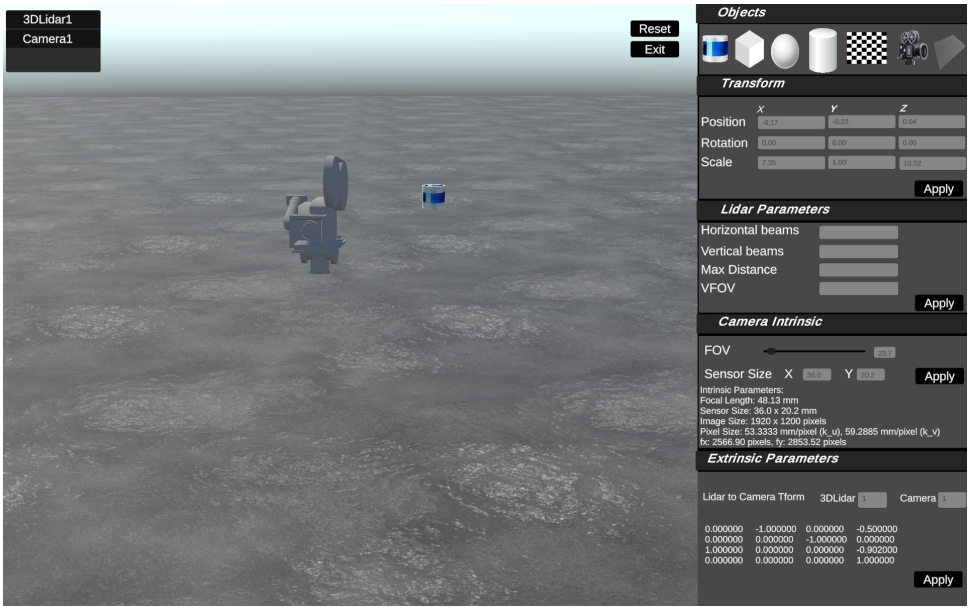

Figure 1: The application UI.

Figure 1 provides an overview of the application interface, showing the simulated environment, sensor placement, and calibration targets. The application is built using Unity, a powerful game engine that supports 3D rendering and physics simulation, making it an ideal choice for this project.

2 Project Description

2.1 Application features

The application provides an intuitive and interactive environment for simulating 3D LiDAR and camera systems. The key functionalities include:

- **Object Placement**: Users can spawn objects such as calibration targets (e.g., planes, chessboards, cylinders, and spheres) and Sensors (3D Lidar and cameras) into the scene by simply clicking on the desired object. This feature enables quick setup of calibration scenarios.
- **Object Transformation**: When an object or sensor is selected, its transformation parameters (position, rotation in degrees, and scale in meters) are displayed. Users can modify these parameters directly and click "Apply" to update the object's placement in the world space.
- **LiDAR Intrinsic Parameters**: Users can configure the intrinsic parameters of the LiDAR sensor, including: Horizontal beams ,Vertical beams, Maximum distance and the Vertical field of view (VFOV)

  Changes are applied by clicking the "Apply" button.
- **Camera Intrinsic Parameters**: Users can configure the intrinsic parameters of the camera sensor, including:
  - Field of view (FOV)
  - Sensor size

  Changes are applied by clicking the "Apply" button.
- **Extrinsic Parameters**: The application provides a feature to display the extrinsic parameters, specifically the transformation matrix between the LiDAR and camera. Users can select the desired sensors by entering their assigned numbers and clicking "Apply" to view the transformation matrix.

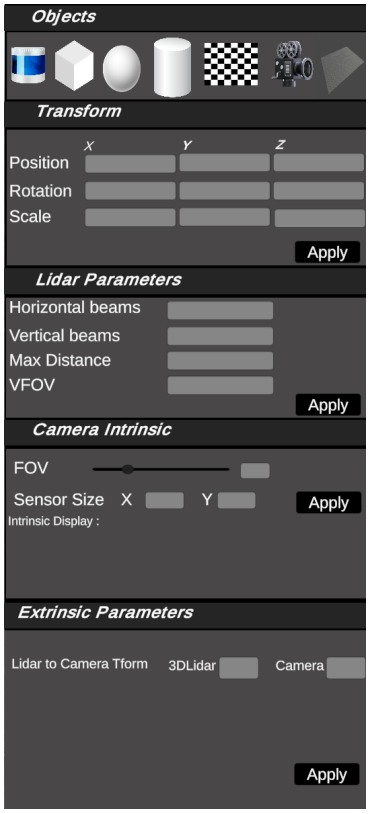

Figure 2: User interface for object placement, transformation, and sensor parameter configuration.

Figure 2 illustrates the user interface, showcasing the Inspector and its functionalities. This modular and user-friendly design ensures that users can easily set up and modify their simulation environment.

- **Sensor Tracking Panel** To help users manage multiple sensors in the scene, the application includes a **Sensor Tracking Panel**. Whenever a new sensor (e.g., Camera1, 3DLidar1) is spawned, it is automatically added to the panel. Users can click on a sensor's tag in the panel to highlight it in the scene with a visible marker, making it easy to identify. Sensors can also be removed by selecting their tag and pressing the "Delete" key.

Figure 3: Sensor tracking panel showing the list of sensors in the scene.

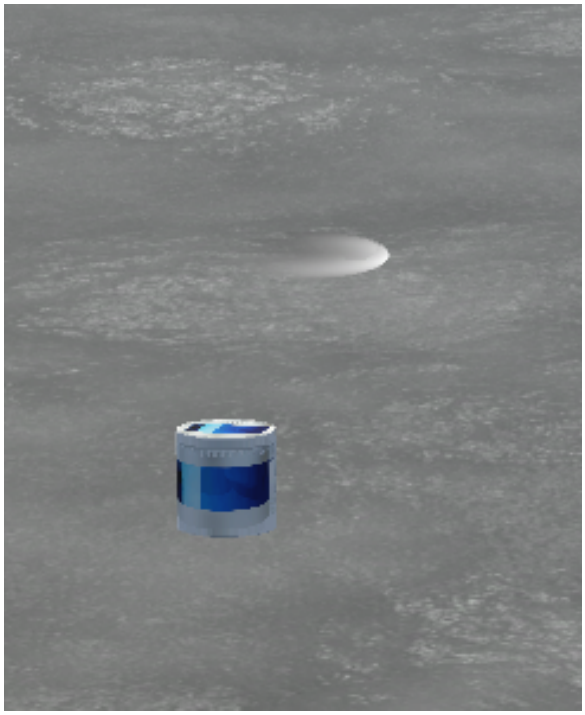

Figure 4: A tracked sensor highlighted with a visible marker in the scene.

- **Sensor Data Save**: The application allows users to save sensor data using the following keybindings:
  - **Save All Sensors**: Pressing the "S" key saves data from all sensors currently in the scene.
  - **Save Selected LiDAR**: Users can select a specific LiDAR sensor and press the "L" key to save its data.
  - **Save Camera Image**: By double-clicking a camera, users enter its view. Pressing the "C" key captures and saves the current scene from that camera's perspective. Users can return to the main scene by pressing the "Esc" key.
- Additionally, the application includes a **Reset Button** that allows users to reset the simulation entirely. This feature clears all objects and sensors from the scene, providing a clean slate for setting up new simulations.

2.2 Sensor's Intrinsic Parameters

2.2.1 3D LiDAR Intrinsic Parameters

The LiDAR simulation generates a point cloud by emitting laser beams in a spherical pattern and calculating their intersections with objects in the scene. The key intrinsic parameters include:

- **Vertical FOV**: The angular range covered vertically (e.g., 45°).
- **Vertical Beams**: The number of laser beams emitted vertically (e.g., 128).

- **Horizontal Beams**: The number of beams emitted horizontally (e.g., 1000 for 360° coverage).
- **Max Distance**: The maximum detection range (e.g., 20 meters).

Data Generation Logic The simulation uses raycasting to detect intersections between laser beams and objects. Each beam's direction is calculated using spherical coordinates based on the configured FOV and angular resolution. The intersection points are transformed into the LiDAR's local coordinate system and stored as a point cloud.

Key Formulas

- **Vertical Angular Resolution**:

$$\text{Vertical Step} = \frac{\text{Vertical FOV}}{\text{Vertical Beams}}$$

- **Horizontal Angular Resolution**:

$$\text{Horizontal Step} = \frac{360^\circ}{\text{Horizontal Beams}}$$

- **Ray Direction**:

$$\text{Direction} = \text{Quaternion.Euler}(\text{Vertical Angle}, \text{Horizontal Angle}, 0) \times \text{Forward Vector}$$

2.2.2 Camera Intrinsic Parameters

The intrinsic parameters of the camera define its internal characteristics, such as focal length, sensor size, and image resolution. These parameters are crucial for accurately simulating the behavior of a camera system.

Key Parameters The camera's intrinsic parameters include:

- **Focal Length (f)**: The distance between the camera's lens and the image sensor (in millimeters).
- **Sensor Size**: The physical dimensions of the camera's sensor (in millimeters).
- **Image Resolution**: The dimensions of the captured image (in pixels).
- **Pixel Size (k_u, k_v)**: The size of each pixel on the sensor (in millimeters per pixel).
- **Focal Length in Pixels (f_x, f_y)**: The focal length expressed in pixel units.

Computation of Intrinsic Parameters The intrinsic parameters are computed as follows:

- **Pixel Size**:

$$k_u = \frac{\text{Image Width}}{\text{Sensor Width}}, \quad k_v = \frac{\text{Image Height}}{\text{Sensor Height}}$$

- **Focal Length in Pixels**:

$$f_x = f \times k_u, \quad f_y = f \times k_v$$

Like the 3DLidar parameters, The Camera Intrinsic parameters can be adjusted as well by changing either the FOV or the snesor size of the camera, allowing for flexible and customizable testing.

3 Application Accuracy Test with MATLAB

To validate the accuracy of the simulated sensor data, the application was tested using MATLAB. Two key calibration processes were performed:

- **Camera Calibration**: To verify the intrinsic parameters of the simulated camera.
- **Camera-LiDAR Calibration**: To validate the extrinsic parameters (transformation matrix) between the LiDAR and camera.

3.1 Coordinate System Challenges

One of the primary challenges in this validation process was the difference in coordinate systems between Unity and MATLAB. Unity uses a **left-handed coordinate system**, while MATLAB and most other simulation engines assume a **right-handed coordinate system**. This discrepancy required careful transformation of the data to ensure consistency and accuracy.

Transformation Steps To align the coordinate systems, the following transformations were applied:

– **LiDAR** : The LiDAR point cloud was transformed from Unity's coordinate system to MATLAB's coordinate system by swapping and negating axes:

$$(x_{\text{Unity}}, y_{\text{Unity}}, z_{\text{Unity}}) \rightarrow (z_{\text{MATLAB}}, -x_{\text{MATLAB}}, y_{\text{MATLAB}})$$

A rotation matrix was applied to align the orientation of the LiDAR data with MATLAB's expectations. The rotation matrix $R_{\text{MatlabLidar}}$ is defined as:

$$R_{\text{MatlabLidar}} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \times R_{\text{UnityLidar}}$$

This matrix ensures that the LiDAR's orientation in Unity is correctly transformed to MATLAB's coordinate system.

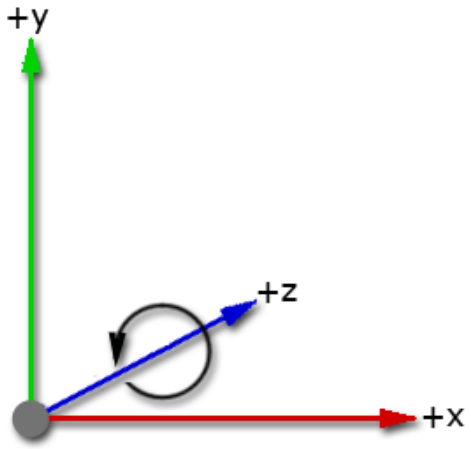

Figure 5: Unity's left-handed coordinate system.

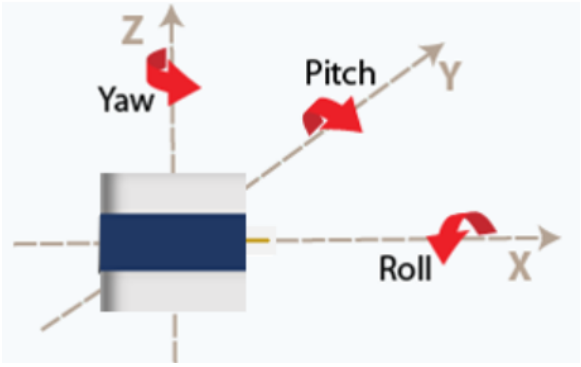

Figure 6: MATLAB's expected right-handed coordinate system for LiDAR data.

- **Camera**: The camera's coordinate system in Unity was aligned with the physical perspective camera model used in MATLAB. The transformation involved only a reflection on the y-axis to account for the differences in coordinate systems. The transformation matrix $T_{\text{perspectiveC}\rightarrow\text{W}}$ is defined as:

$$T_{\text{perspectiveC}\rightarrow\text{W}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \times T_{\text{UnityC}\rightarrow\text{World}}$$

Here, $T_{\text{UnityC}\rightarrow\text{World}}$ represents the transformation matrix of the camera in Unity's coordinate system.

3.2 Camera Calibration

The MATLAB Camera Calibrator app was used to calibrate the simulated camera. The app estimates the intrinsic parameters, including focal length, principal point, and distortion coefficients, based on a set of calibration images. Figure 7 shows the MATLAB Camera Calibrator interface used for this process.

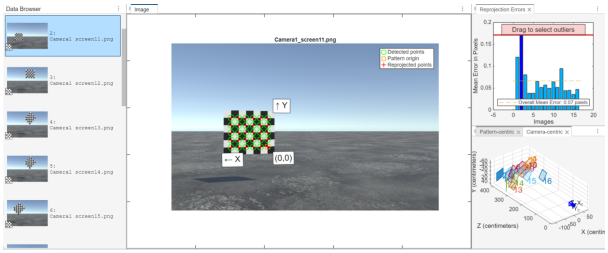

Figure 7: MATLAB Camera Calibrator app used for camera calibration.

3.3 Intrinsic Parameters Comparison

The intrinsic parameters obtained from MATLAB were compared with those generated by the Unity-based application. The results are summarized below:

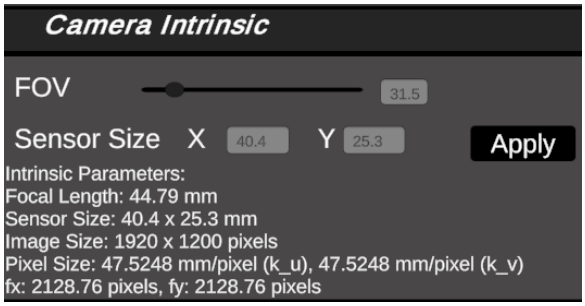

Figure 8: Intrinsic parameters from the Unity-based application.

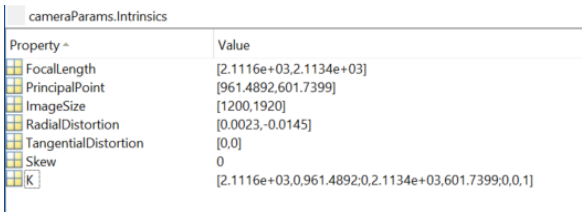

Figure 9: Intrinsic parameters from MATLAB calibration.

Key Observations

- **Focal Length**: The focal length values from Unity ($f_x = 2128.76$ pixels, $f_y = 2128.76$ pixels) closely match those from MATLAB ($f_x = 2111.6$ pixels, $f_y = 2113.4$ pixels).
- **Principal Point**: The principal point from Unity $(960, 600)$ is consistent with MATLAB's results $(961.4892, 601.7399)$.
- **Distortion**: The distortion coefficients from MATLAB (Radial $= [0.0023, -0.0145]$, Tangential $= [0, 0]$) indicate minimal distortion, which aligns with the Unity simulation's assumption of an ideal camera model.

These results demonstrate that the Unity-based application accurately simulates the intrinsic parameters of a camera, as validated by MATLAB's calibration process.

3.4 Camera-LiDAR Calibration

The extrinsic parameters between the LiDAR and camera were validated by projecting the LiDAR point cloud onto the camera image and comparing the transformation matrices obtained from MATLAB and the Unity-based application.

Projection of LiDAR Points Figure 10 shows the result of projecting the LiDAR points onto the camera image. The points are colorized based on the corresponding pixel values in the image, demonstrating accurate alignment between the LiDAR and camera data.

Transformation Matrices The transformation matrices obtained from MATLAB and the Unity-based application are shown below:

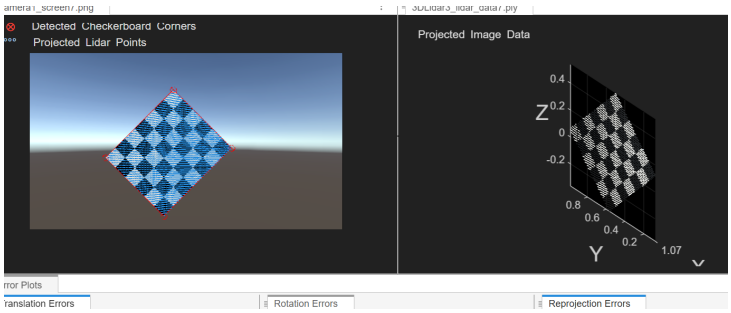

Figure 10: Projection of LiDAR points onto the camera image, colorized based on pixel values.

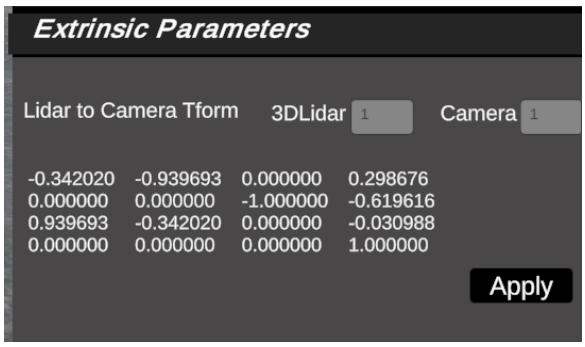

Figure 11: Transformation matrix from the Unity-based application.

tform.A

	1	2	3	4
1	-0.3420	-0.9397	-8.7880e-04	0.2966
2	0.0015	3.7912e-04	-1.0000	-0.6243
3	0.9397	-0.3420	0.0013	-0.0584
4	0	0	0	1

Figure 12: Transformation matrix from MATLAB calibration.

Rotation Error Calculation To quantify the accuracy of the extrinsic calibration, the rotation error between the two transformation matrices was calculated. Let R_1 be the rotation matrix from the Unity-based application and R_2 be the rotation matrix from MATLAB. The error in radians is given by:

$$\text{Error} = \arccos\left(\frac{\text{trace}(R_1^T \times R_2) - 1}{2}\right)$$

The calculated error was 0.0037417 radians, indicating a high level of agreement between the two calibration results.

Key Observations

- The projection of LiDAR points onto the camera image demonstrates accurate alignment, as shown in Figure 10.
- The transformation matrices from Unity and MATLAB are highly consistent, with a small rotation error of 0.0037417 radians.
- These results validate the accuracy of the extrinsic calibration between the LiDAR and camera in the Unity-based application.

4 Future Work

While the current application provides a robust simulation environment for LiDAR and camera systems, several enhancements can be made to improve its functionality and usability:

- **Multi-Sensor Testing**: Extend the application to support multiple sensors (e.g., radar, ultrasonic sensors) and test their interactions in complex scenarios.
- **Data Visualization**: Implement features to display the generated sensor data (e.g., LiDAR point clouds, camera images) directly within the application, enabling real-time analysis and debugging.

- **Integrated Calibration**: Develop an in-app calibration tool to perform and validate sensor calibration without relying on external tools like MATLAB. This would streamline the workflow and make the application more self-contained.
- **Advanced Calibration Targets**: Add support for more complex calibration targets and real world scenarios to improve the accuracy and versatility of the calibration process.

5 Conclusion

This project successfully developed a Unity-based application for simulating 3D LiDAR and camera systems, allowing users to configure sensor parameters, place calibration targets, and save simulated data. The application's accuracy was validated through MATLAB, with results demonstrating close alignment between the simulated and expected intrinsic and extrinsic parameters. Key achievements include:

- Accurate simulation of LiDAR and camera intrinsic parameters.
- Validation of extrinsic calibration between LiDAR and camera using MATLAB.
- A user-friendly interface for configuring sensors and visualizing results.

The application serves as a valuable tool for researchers and engineers working on sensor fusion, calibration, and autonomous systems. Future work will focus on expanding its capabilities, including multi-sensor testing, integrated calibration, and enhanced data visualization, to further improve its utility and versatility.