# Radar-Driven ADAS Validation Framework with Cloud-Connected Dashboard

**- CHARITHA ARMOOR**

**Aim:**Build a radar-based ADAS object validation system with simulated CAN messages and real-time dashboard telemetry.

This project aims to **validate radar-based object detection** using **Python and ROS**, simulate **vehicle communication over CAN**, and **send** critical **ADAS data** to a **real-time dashboard.** It demonstrates system integration skills relevant to automotive validation, embedded systems, and telematics engineering roles.

**Learning Goals:**

1. Build the core radar data processing module using Python
2. Simulate CAN communication and integrate with radar processing.
3. Send data to the cloud and visualize in real time.
4. Pull all parts into a working prototype and document it.

**Project Structure Setup**

Create your local folder (or GitHub repo) with this structure:

**radar-adas-validation**/

│

├── radar\_processing/ # Python scripts for radar data handling

├── can\_simulation/ # CAN message generator & log

├── telemetry/ # Cloud uploader (MQTT or Firebase)

├── dashboard/ # Dashboard code (Dash/HTML)

├── data/ # Radar logs, CAN logs

├── docs/ # Diagrams, test plan

├── main.py # Main integration script

└── README.md

You can use README.md to:

* Describe the project in 3–5 lines
* Add the tools you plan to use
* List the 4 modules (radar, CAN, telemetry, dashboard)
* Set your milestones by week

**Why This Project?**

* Aligns with your radar system experience (CARISSMA, Master’s thesis).
* Uses your Python, ROS, CAN, and HIL tools.
* Adds cloud & dashboard components to cover telematics and connected car skills.
* Boosts your system-level validation portfolio — exactly what German OEMs and Tier-1 suppliers want.

**Project Outline**

1. **Real or Simulated Radar Data Pipeline**

* Use recorded or simulated radar data from CARLA or existing datasets:
* Use Python + ROS to process object detection data
* Perform target tracking, height estimation, and false positive filtering
* You already did this at CARISSMA — reuse your past methods,

but now:

* Add automated test case validation scripts (Python + Pandas + Matplotlib)
* Create a configurable pipeline for testing new radar sensors or functions

1. **CAN Bus Integration**

Simulate vehicle integration:

* Use CANoe or Python-CAN to simulate CAN messages for radar, speed, and brake commands
* Add fault injection (e.g., drop packets, delay radar updates)
* This shows your grasp of vehicle communication robustness testing

1. **Cloud-Connected Telematics Layer**

Build a lightweight cloud interface:

* Use Raspberry Pi or laptop to send radar data via MQTT or HTTP to a Firebase or AWS dashboard
* Store vehicle position, object info, and test status

Optional: Use Grafana or a web dashboard to visualize results remotely.

1. **Visualization Dashboard**

* Use Python (Dash/Plotly or Flask) or Firebase + HTML:
* Show object map
* Alert for missed detections
* Show radar object logs and timestamps

**Skills You Will Demonstrate**

|  |  |
| --- | --- |
| **Skillset** | **Evidence in Project** |
| Radar validation | Reuse and expand your radar object detection/  height estimation work |
| ROS, Python, CAN | Core system integration with logging and bus simulation |
| Data automation | Automated result validation pipeline |
| Telematics | Real-time cloud upload and dashboard |
| Embedded systems | CAN simulation and cloud communication with low-cost hardware |
| ADAS/AV testing | End-to-end HIL/SIL simulation with analysis of detection accuracy |

**Add to Resume Like This:**

Personal Project – Radar-Based ADAS Validation & Telematics System

* Built a radar object validation framework using Python and ROS, simulating CAN-based communication and edge-case scenarios.
* Integrated cloud telemetry via MQTT for real-time data logging and visualization of object detection results.
* Developed a custom dashboard to monitor ADAS test results, radar logs, and detection metrics.

**Project Plan**

**Week 1: Setup & Radar Data Pipeline:**

**Goal**: Build the core radar data processing module using Python.

**Tasks**:

* Set up your local project folder and GitHub repo
* Define project scope in a README.md with diagram of the system
* Download or simulate sample radar data (CSV, ROSBag, or CARLA)
* Write a script in Python to:
* Load radar data
* Detect objects
* Visualize using Matplotlib (range, azimuth, velocity)
* Begin simple object filtering or height estimation logic
* **Tools:**
* Python, Pandas, Matplotlib
* Jupyter or PyCharm
* (Optional) ROS + CARLA if simulating

**Week 2: CAN Simulation + Data Logging:**

**Goal:** Simulate CAN communication and integrate with radar processing

**Tasks:**

* Simulate CAN messages (e.g., object ID, speed, brake status) using:
* Python-CAN or
* Vector CANoe (if licensed)
* Write a module that:
* Sends and logs messages from radar output to CAN
* Injects edge cases (e.g., delay, noise)
* Combine CAN data + radar results into one log file
* Create test cases for different driving scenarios

**Tools:**

* Python-CAN, CSV logs
* CANoe (optional

**Week 3: Cloud Telemetry + Dashboard**

**Goal:** Send data to the cloud and visualize in real time.

**Tasks:**

* Choose a cloud backend (Firebase Realtime DB, MQTT with Mosquitto, or AWS IoT)
* Send key radar/CAN data to cloud (timestamp, object ID, position, speed)
* Create a simple dashboard (using Dash/Plotly or Firebase + HTML)

Table of object data

Graph of object distance vs. time

(Optional) Map with GPS-like view

**Tools:**

* Python + Dash (or Flask), Firebase (or MQTT), Plotly

**Week 4: Integration + Finalization**

**Goal:** Pull all parts into a working prototype and document it.

**Tasks:**

* Integrate:
* Radar processor
* CAN simulator
* Cloud telemetry sender
* Dashboard visualizer
* Record a demo run (screen recording or screenshots)
* Polish GitHub repo:
* README with system diagram and usage instructions
* Sample data and logs
* Write a short LinkedIn post or PDF summary

**Deliverables by Day 30:**

✅ Working codebase (GitHub)

✅ Dashboard with radar object display

✅ Project documentation (README, diagrams)

✅ Demo (video/screenshots)

✅ Resume bullet & LinkedIn post

**Planning Day Tasks (Day 0)**

**1. Define Project Goal**

"Build a radar-based ADAS object validation system with simulated CAN messages and real-time dashboard telemetry."

**2. Project Structure Setup**

Create your local folder (or GitHub repo) with this structure:

radar-adas-validation/

│

├── radar\_processing/ # Python scripts for radar data handling

├── can\_simulation/ # CAN message generator & log

├── telemetry/ # Cloud uploader (MQTT or Firebase)

├── dashboard/ # Dashboard code (Dash/HTML)

├── data/ # Radar logs, CAN logs

├── docs/ # Diagrams, test plan

├── main.py # Main integration script

└── README.md

You can use README.md to:

* Describe the project in 3–5 lines
* Add the tools you plan to use
* List the 4 modules (radar, CAN, telemetry, dashboard)
* Set your milestones by week

**3. Tools to Install Today**

Tool Use

✅ Python 3.10+ Main language

✅ Jupyter Notebook or PyCharm Development environment

✅ Pandas, Matplotlib Radar data analysis

✅ Python-CAN or CANoe (if available) CAN simulation

✅ Dash or Flask Web dashboard

✅ Firebase or Mosquitto MQTT Cloud telemetry (you can choose this later)

**4.Data to Prepare**

Decide whether you'll:

* Use real radar logs (e.g., from your CARISSMA project or open datasets)
* Simulate radar data (e.g., random object positions in CSV)
* Use CARLA simulator later (optional for Week 2+)

I can help generate or mock up radar data if needed.

**5. Milestone Checkpoint Setup**

Make a simple log file or Trello board with these 4 columns:

To Do (Week-by-week tasks)

Doing

Done

Notes / Ideas

**6.Optional: Write Your Project Statement**

To stay focused:

This project aims to validate radar-based object detection using Python and ROS, simulate vehicle communication over CAN, and send critical ADAS data to a real-time dashboard. It demonstrates system integration skills relevant to automotive validation, embedded systems, and telematics engineering roles.

✅ Planning Checklist (for today)

* Create project folder or GitHub repo ⬜
* Install Python tools (Pandas, Dash, MQTT, etc.) ⬜
* Decide on radar data source (real, mock, CARLA) ⬜
* Set up Trello or task log (optional) ⬜
* Write README.md with modules + goals ⬜

Once you check these off today, you’ll be fully ready to start coding tomorrow (Day 1).

* Want me to generate:
* A starter README.md
* A sample radar dataset

Or a Trello template link?

### 🛠️ How to Create the Diagram

#### Option 1: Use ****draw.io (diagrams.net)****

Easiest and free.

Create boxes and arrows based on the layout above.

Export as PNG or SVG.

Save the file as system\_diagram.png in your repo.

Embed it in your README.md like:

markdown

CopyEdit

## System Architecture

![System Diagram](system\_diagram.png)

#### Option 2: ASCII Diagram (Temporary)

You can also paste the diagram above directly into your README as text for now if you’re not ready to design a visual version.