

# **Verbesserung eines Bluetooth-basierten Warnsystems für den Straßenverkehr durch Datenlogging und innovative Visualisierungstechniken**

## **Studienarbeit T3200**

Studiengang Elektrotechnik

Studienrichtung Fahrzeugelektronik

Duale Hochschule Baden-Württemberg Ravensburg, Campus Friedrichshafen

von

Luka Tadic

Abgabedatum:	14.07.2025
Bearbeitungszeitraum:	07.04.2025 - 14.07.2025
Matrikelnummer:	5726700
Kurs:	TFE22-1
Dualer Partner:	
Betreuerin / Betreuer:	Prof. Dr. Ing. Tobias Frank
Gutachterin / Gutachter:	Prof. Dr. Ing. Tobias Frank

## Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit T3200 mit dem Thema:

*Verbesserung eines Bluetooth-basierten Warnsystems für den  
Straßenverkehr durch Datenlogging und innovative Visualisie-  
rungstechniken*

selbstständig verfasst und keine anderen als die angegebenen Quellen  
und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte  
elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Friedrichshafen, den 8. Juli 2025

---

Luka Tadic

# Kurzfassung

## Abstract

# Hilfsmittel

## Abbildungsverzeichnis

1	Beispiel Rechtsabbiegen LKW . . . . .	1
2	XPLR-AOA Explorer Kit . . . . .	4
3	Datenlogger Cube (Avisaro 2.0) zur Speicherung von Sensor- und Prozessdaten . . . . .	6
4	Beispiel Triangulation . . . . .	11

## Abkürzungsverzeichnis

**AoA** Angle-of-Arrival

**API** Application Programming Interface

**BLE** Bluetooth Low Energy

**LKW** Lastkraftwagen

**SDK** Software Development Kit

**CSV** Comma-seperated values

**JSON** JavaScript Object Notation

**UI** User Interface

**JSONL** JavaScript Object Notation Lines

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Zielsetzung . . . . .	2
<b>2</b>	<b>Rückblick auf das mobile Warnsystem</b>	<b>3</b>
2.1	Konzept und Zielsetzung . . . . .	3
2.2	Bluetooth AoA und technische Grundlagen . . . . .	3
2.3	Herausforderungen und Gründe für die Projektpause . . . . .	4
<b>3</b>	<b>Grundlagen Datenlogger</b>	<b>6</b>
3.1	Zweck und Funktionalität eines Datenloggers . . . . .	6
3.2	Anwendungsbereiche im Kontext der Studienarbeit . . . . .	7
3.3	Vorhandene Herausforderungen und Lösungsansätze . . . . .	7
<b>4</b>	<b>Entwicklung des Datenloggers</b>	<b>9</b>
4.1	Anforderungen an die Software . . . . .	9
4.2	Auswahl geeigneter Technologien und Tools . . . . .	9
4.3	Implementierung und Integration in bestehende Systeme . . . . .	10
4.4	Ergebnisse und Evaluation . . . . .	14
<b>5</b>	<b>Grundlagen der Visualisierung</b>	<b>15</b>
5.1	Anforderungen an eine effektive Visualisierung . . . . .	15
5.2	Darstellungsmöglichkeiten und -technologien . . . . .	16
<b>6</b>	<b>Entwicklung der neuen Visualisierung</b>	<b>19</b>
6.1	Analyse der bestehenden Visualisierung . . . . .	19
6.2	Konzept einer verbesserten Visualisierung . . . . .	20
6.3	Implementierung der verbesserten Visualisierung . . . . .	20
6.4	Ergebnisse und Benutzerfreundlichkeit . . . . .	20
<b>7</b>	<b>Test und Validierung</b>	<b>20</b>
7.1	Testmethodik und -umgebung . . . . .	20
7.2	Ergebnisse der Tests . . . . .	20
7.3	Diskussion der Testergebnisse und Optimierungspotentiale . . . . .	20
<b>8</b>	<b>Kritische Bewertung und Ausblick</b>	<b>20</b>
8.1	Reflexion der erreichten Ergebnisse . . . . .	20
8.2	Grenzen der aktuellen Umsetzung . . . . .	20
8.3	Vorschläge für zukünftige Weiterentwicklungen . . . . .	20

# 1 Einleitung

## 1.1 Motivation

In den letzten Jahren ist die Zahl der tödlichen Verkehrsunfälle bedauerlicherweise gestiegen, was auf eine Vielzahl von Faktoren wie die steigende Verkehrsdichte sowie mangelnde Aufmerksamkeit und Sorgfalt im Straßenverkehr zurückzuführen sein kann. Um dieser negativen Entwicklung entgegenzuwirken, wurden unterschiedliche Maßnahmen ergriffen. Neben strengeren Sicherheitsgesetzen haben sich vor allem technologische Innovationen wie Fahrzeugkameras, Sensorik und verschiedene Fahrerassistenzsysteme als wesentliche Instrumente zur Unfallprävention herauskristallisiert.



Abbildung 1: Beispiel Rechtsabbiegen LKW

Eine besonders bedeutende Neuerung im Bereich der Lastkraftwagen (LKW)-Sicherheit stellen Abbiegeassistenten dar. Diese Systeme helfen, den toten Winkel zu reduzieren und somit Unfälle – insbesondere beim Rechtsabbiegen – zu vermeiden. Trotz dieser technischen Fortschritte besteht weiterhin Potenzial für weitere Verbesserungen. Eine umfassende Forschung und Entwicklung im Bereich von Fahrerassistenzsystemen könnte zukünftig dazu beitragen, das allgemeine Verkehrsrisiko weiter zu senken und die Verkehrssicherheit signifikant zu erhöhen. [1]



## 1.2 Zielsetzung

Zielsetzung dieser Studienarbeit ist es, einen leistungsfähigen Datenlogger zu entwickeln und die Visualisierung zu verbessern, um die Funktionalität des Fahrerassistenzsystems zu erhöhen und dadurch dessen Entwicklung zu erleichtern. Durch die Implementierung eines leistungsfähigen Datenloggers sollen Messwerte systematisch und zuverlässig erfasst werden können, was eine effektivere Zusammenarbeit im Entwicklungsteam ermöglicht und die Notwendigkeit einer dauerhaften Hardwareverbindung reduziert. Die Optimierung der Visualisierung dient dazu, Messergebnisse klarer und übersichtlicher darzustellen, um künftige Analysen und Tests zu vereinfachen und somit die Qualität und Genauigkeit der Positionsbestimmung zu erhöhen. Diese Verbesserungen sollen letztlich zu einer höheren Effizienz im Entwicklungsprozess führen und somit einen wesentlichen Beitrag zur Erhöhung der Verkehrssicherheit leisten.

## 2 Rückblick auf das mobile Warnsystem

### 2.1 Konzept und Zielsetzung

Die vorherige Studienarbeit verfolgte das Ziel, ein mobiles Warnsystem zur Minimierung von Abbiegeunfällen zwischen LKW und ungeschützten Verkehrsteilnehmenden wie Fußgängerinnen und Radfahrerinnen zu entwickeln. Zentrales Element dieses Systems war eine mobile Applikation, die als aktiver Bluetooth-Sender fungieren sollte. In Kombination mit einem am LKW montierten Empfängerboard (basierend auf dem u-blox XPLR-AOA-1 Kit) sollte mithilfe der Angle-of-Arrival (AoA)-Technologie die Position des Smartphones lokalisiert und bei drohender Gefahr eine Warnung ausgegeben werden. Das System versprach einen kostengünstigen und einfach zugänglichen Ansatz zur Verbesserung der Verkehrssicherheit im städtischen Raum.

### 2.2 Bluetooth AoA und technische Grundlagen

Die AoA-Technologie ist Teil des Bluetooth 5.1 Standards und ermöglicht die Positionsbestimmung eines Senders durch Messung des Einfallswinkels der Funksignale an mehreren Antennen eines Empfängers. Voraussetzung hierfür ist jedoch ein exakter Zugang zu den Bluetooth-Sendeparametern sowie eine Antennenkonfiguration mit bekannten geometrischen Abständen. Das u-blox XPLR-AOA-1 Explorer Kit stellt hierzu eine geeignete Hardwarelösung dar, da es mit einem AoA-fähigen Empfängerboard und einem sogenannten Tag (Sender) ausgestattet ist. Ziel der Arbeit war es, das Smartphone funktional durch diesen Tag zu ersetzen. [1][2][3]

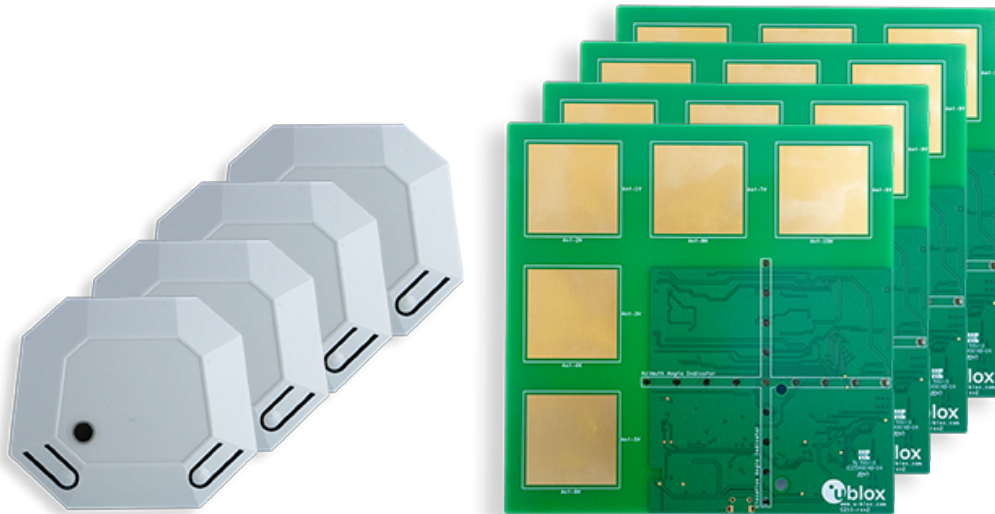


Abbildung 2: XPLR-AOA Explorer Kit

## 2.3 Herausforderungen und Gründe für die Projektpause

Im praktischen Verlauf der Umsetzung traten mehrere schwerwiegende technische Limitierungen auf, die die Fortführung des Projekts verhinderten:

- Aktuelle Smartphone-Firmware (insbesondere auf Android-Basis) verhindert in der Regel den vollständigen Zugriff auf die Bluetooth-Sendeparameter. Ein gezieltes Emittieren von Bluetooth Low Energy (BLE)-Signalen, wie es zur AoA-Ortung erforderlich ist, ist mit Standard-Software Development Kit (SDK) nicht möglich.
- Nur wenige aktuelle Mobilgeräte unterstützen Bluetooth 5.1 vollständig, was die Reichweite und Genauigkeit der Ortung stark einschränkt.

- Die Emulation des u-blox Tags durch das Smartphone scheiterte daher an mangelnden Systemrechten, fehlender Low-Level-Application Programming Interface (API)-Zugriffe und nicht ausreichender Hardwareunterstützung.

Diese Faktoren führten dazu, dass das Projekt in der geplanten Form nicht abgeschlossen werden konnte. Die zugrundeliegende Idee bleibt jedoch vielversprechend und kann in Zukunft wieder aufgegriffen werden, sobald sich die technischen Rahmenbedingungen verbessert haben.[4][5][6][7]

## 3 Grundlagen Datenlogger

### 3.1 Zweck und Funktionalität eines Datenloggers

Ein Datenlogger ist ein autonom arbeitendes Gerät oder Softwaremodul, das physikalische oder digitale Messwerte über einen definierten Zeitraum hinweg aufzeichnet. Ziel ist es, Veränderungen systematisch und zuverlässig zu erfassen, um sie später analysieren, auswerten oder dokumentieren zu können. Typische erfasste Daten umfassen beispielsweise Temperatur, Spannung, Zeitstempel, Lage- oder Positionsdaten.



Abbildung 3: Datenlogger Cube (Avisaro 2.0) zur Speicherung von Sensor- und Prozessdaten

Im Gegensatz zu interaktiven Messsystemen arbeitet ein Datenlogger in der Regel unabhängig und ohne dauerhafte Verbindung zu einem Steuergerät. Dies macht ihn besonders geeignet für mobile oder schwer zugängliche Anwendungen. Durch kontinuierliche Aufzeichnung lassen sich Messwerte nachvollziehbar dokumentieren, was vor allem in der Entwicklung, Fehlersuche und Validierung technischer Systeme von Bedeutung ist.

Wichtige Eigenschaften eines Datenloggers sind unter anderem eine ausreichende Speicherkapazität, Energieeffizienz und die Kompatibilität zu verschiedenen Datenformaten wie Comma-separated values (CSV) oder

JavaScript Object Notation (JSON). Moderne Logger bieten darüber hinaus Schnittstellen für drahtlose Datenübertragung oder automatisierte Synchronisation mit Analysetools.

### **3.2 Anwendungsbereiche im Kontext der Studienarbeit**

Im Rahmen dieser Studienarbeit wird der Datenlogger gezielt eingesetzt, um die Entwicklungsarbeit am Bluetooth-AoA-basierten Lokalisierungssystem effizienter und flexibler zu gestalten. Ziel ist es, die Abhängigkeit von einer aktiven Verbindung zur Hardware während der Test- und Anpassungsphasen zu minimieren. Personen, die an der Parametrierung oder Feinjustierung des Systems arbeiten, sollen in die Lage versetzt werden, auch ohne direkte Verbindung zur u-blox-Hardware auf reale Messdaten zuzugreifen.

Der Datenlogger dient dabei nicht nur der reinen Aufzeichnung, sondern ermöglicht die Speicherung von tatsächlich erfassten, originalgetreuen Messwerten. Diese Werte können anschließend in separate Programme oder Auswertungswerkzeuge importiert werden, um dort Berechnungen, Analysen und grafische Darstellungen durchzuführen. Änderungen an Parametern oder Auswertealgorithmen können somit auf Basis gespeicherter Daten vorgenommen und getestet werden, ohne dass dafür eine erneute Datenerfassung mit der realen Sensorik notwendig ist.

Darüber hinaus erlaubt der Datenlogger eine „Offline-Simulation“ oder ein Daten-Replay mit echten Werten. Diese Vorgehensweise bietet den Vorteil, dass Optimierungen zunächst im Testsystem geprüft werden können. Erst wenn ein Ergebnis zufriedenstellend ist, wird es auf das reale System übertragen. Damit leistet der Datenlogger einen wesentlichen Beitrag zur Entkopplung von Entwicklung und Hardwareverfügbarkeit, was vor allem in Teamkonstellationen und bei paralleler Arbeit an mehreren Teilaspekten des Systems von großem Nutzen ist.

### **3.3 Vorhandene Herausforderungen und Lösungsansätze**

Bei der Entwicklung eines zuverlässigen Datenloggers im Kontext des Bluetooth AoA-Systems treten verschiedene Herausforderungen auf. Eine der zentralen Schwierigkeiten liegt in der exakten zeitlichen Zuordnung der erfassten Daten. Für eine sinnvolle Auswertung müssen Zeitstempel, Empfangswinkel und weitere Messgrößen synchronisiert und konsistent gespeichert werden.

Ein weiteres Problem stellt die effiziente und verlustfreie Speicherung

großer Datenmengen dar. Hierbei müssen sowohl die Speicherkapazität als auch die Lesbarkeit und Weiterverwendbarkeit berücksichtigt werden. Gerade im Hinblick auf die spätere Analyse mit gängigen Tools wie Python oder MATLAB ist ein gut strukturiertes Datenformat – etwa CSV oder JSON – von hoher Bedeutung.

Auch die Integration in bestehende Entwicklungsprozesse erfordert eine durchdachte Schnittstellengestaltung. Der Datenlogger muss flexibel und modular aufgebaut sein, damit er problemlos in verschiedene Testumgebungen eingebunden werden kann. Gleichzeitig soll der Energieverbrauch möglichst gering gehalten werden, um auch den mobilen Einsatz über längere Zeiträume hinweg zu ermöglichen.

Lösungsansätze bestehen in der Verwendung eines einfachen, standardisierten Speicherformats, einem modularen Softwaredesign mit klar dokumentierten Schnittstellen sowie in der Implementierung grundlegender Fehlererkennungs- und Synchronisationsmechanismen. Diese Maßnahmen sollen sicherstellen, dass der Datenlogger zuverlässig, energieeffizient und nutzerfreundlich eingesetzt werden kann.

## 4 Entwicklung des Datenloggers

### 4.1 Anforderungen an die Software

Die bestehende Softwarebasis setzt bereits auf eine performante C++-Schicht für die zeitkritische Kommunikation mit der u-blox-Hardware und auf ergänzende Python-Skripte für Tests und Auswertungen. Dieses zweistufige Konzept bleibt erhalten: Der neue Datenlogger wird vollständig in Python implementiert und greift über schlanke Schnittstellen auf die vorhandenen C++-Funktionen zu. So können Erweiterungen schnell umgesetzt werden, ohne den stabilen Kerncode anzutasten.

Als Speicherformat dient JSON. Es ermöglicht, Messwerte – etwa Zeitstempel, AoA-Winkel, Signalstärke und relevante Metadaten – in einer klar strukturierten, menschenlesbaren Datei abzulegen. Die Dateien lassen sich bei Bedarf komprimieren und ohne Konvertierung in gängige Analyse- oder Visualisierungsprogramme importieren.

Diese Kombination aus C++ (für hardwarenahen Durchsatz) und Python (für flexible Datenverarbeitung) erfüllt die Anforderungen an Zuverlässigkeit, Erweiterbarkeit und unkomplizierte Offline-Analyse. Sie stellt zudem sicher, dass Teammitglieder auch ohne direkten Hardwarezugriff reale Datensätze einspielen, parametrieren und visualisieren können.

[8][9]

### 4.2 Auswahl geeigneter Technologien und Tools

Für die Erweiterung des bestehenden Prototyps wurde bewusst an der zweigeteilten Sprachebene festgehalten: zeitkritische Kommunikation und Treiberfunktionen laufen weiterhin in C++, während der neue Datenlogger in Python implementiert wird. So können die vorhandenen C++-Routinen unverändert genutzt werden, während Python durch seine hohe Entwicklungsgeschwindigkeit schnelle Anpassungen und Auswertungen ermöglicht.

Als Speicherformat wurde JSON gewählt. Es erlaubt, Messwerte – etwa Zeitstempel, Winkel und Signalstärke – zusammen mit Metadaten in einer klaren, hierarchischen Struktur abzulegen. Die Dateien bleiben damit menschenlesbar und lassen sich ohne Umwege in gängige Analyse- und Visualisierungswerkzeuge importieren. Bei Bedarf kann eine Kompression eingesetzt werden, um Speicherplatz zu sparen.

Mit dieser Kombination werden die in Abschnitt 4.1 genannten Anforderungen erfüllt: Die Lösung ist leicht erweiterbar, plattformunabhängig und stellt die Daten in einem Format bereit, das direkt für Offline-Analysen genutzt werden kann.



## 4.3 Implementierung und Integration in bestehende Systeme

Die Implementierung des Datenloggers erfolgte im Rahmen eines bestehenden Lokalisierungssystems, das auf der Bluetooth-AoA-Technologie basiert. Ziel war es, die Erfassung, Speicherung und spätere Analyse von Messwerten so zu gestalten, dass diese unabhängig von einer Live-Hardwareverbindung möglich ist und flexibel in den Entwicklungsprozess integriert werden kann.

**Messung – Entstehung der Rohdaten** Im Zentrum der Messung stehen zwei Hauptkomponenten: das sogenannte *Tag*, das als zu verfolgendes Objekt fungiert und kontinuierlich Signale sendet, sowie mindestens zwei *Anker* (Anchors), die als fest installierte Empfangseinheiten dienen. Die Anker, ausgestattet mit mehreren Antennen, empfangen das Signal des Tags und bestimmen den Einfallswinkel (Angle-of-Arrival (AoA)), unter dem das Signal ankommt. Zusätzlich werden häufig weitere Sensordaten erfasst, beispielsweise Geschwindigkeit oder Risikoabschätzungen.

**Datenerfassung und -aufbereitung** Die Kommunikation zwischen Ankern und Rechner erfolgt typischerweise über eine serielle Schnittstelle oder BLE. Ein Python-Skript (z. B. `getAnchorData.py`) liest die Rohdaten kontinuierlich aus. Für jede einzelne Messung werden strukturierte Informationen erfasst, darunter:

- `anchors`: Positionen der Anker, z. B. `[[3, 0], [0, 0]]`
- `angles`: Die gemessenen Winkel, z. B. `[-31, 24]`
- `sensor_values`: Eine Liste weiterer Sensordaten:

```
[
    {"theta": 90, "val": -31, "pos": [3, 0], "
      speed_along_line": 0, "risk_level": 1},
    {"theta": 90, "val": 24, "pos": [0, 0], "
      speed_along_line": 0, "risk_level": 1}
]
```

Die Daten werden zeilenweise im JSON Lines-Format (`.jsonl`) gespeichert. Jede Zeile enthält einen Zeitstempel und die zugehörigen Sensordaten. Ein Beispiel für eine gespeicherte Messung ist:

```
{"timestamp": "2025-06-03T06:32:50.140342",
```

```
"raw_sensor_data": {
  "anchors": [[3, 0], [0, 0]],
  "angles": [-31, 24],
  "sensor_values": [...]
}}
```

**Berechnung der Tag-Position – Triangulation** Aus den gemessenen Winkeln und bekannten Positionen der Anker wird die Position des Tags durch Triangulation berechnet. Jeder Anker definiert eine Gerade, auf der sich der Tag befinden könnte. Die Schnittpunktberechnung dieser Geraden liefert eine Positionsschätzung. Die Richtung jeder Linie ergibt sich aus dem gemessenen Winkel mittels:

$$dx = \sin(\theta), \quad dy = \cos(\theta)$$

Der Schnittpunkt wird mithilfe der folgenden Gleichung bestimmt:

$$t_1 = \frac{(x_2 - x_1) \cdot dy_2 - (y_2 - y_1) \cdot dx_2}{dx_1 \cdot dy_2 - dy_1 \cdot dx_2}$$

und

$$x = x_1 + t_1 \cdot dx_1, \quad y = y_1 + t_1 \cdot dy_1$$

Diese Berechnung kann sowohl live während der Datenerfassung als auch im Nachgang im sogenannten Replay-Modus erfolgen.

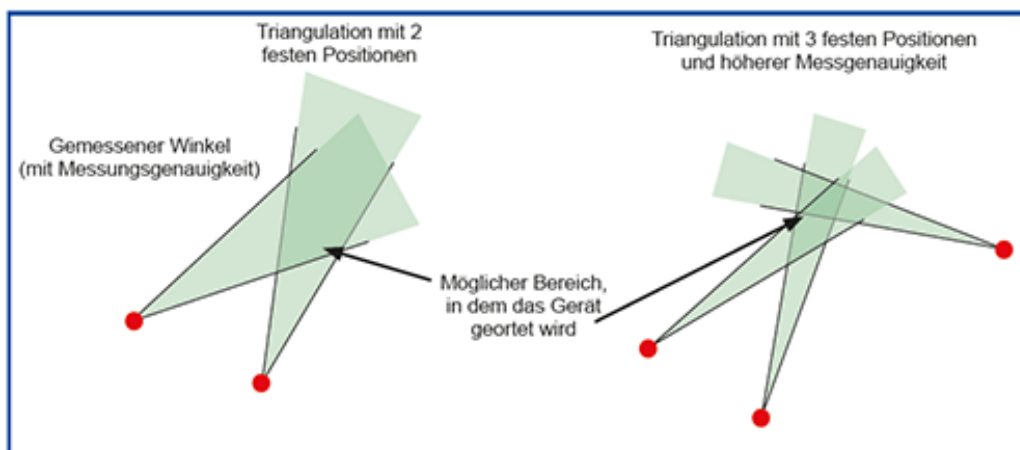


Abbildung 4: Beispiel Triangulation

**Speicherung und Weiterverarbeitung** Sämtliche berechneten Werte – Winkel, Position, Sensordaten – werden chronologisch gespeichert. Das Python-Skript `replayLogger.py` ermöglicht das Einlesen dieser Datei und berechnet bei Bedarf die Position erneut. Dies erlaubt eine erneute Analyse oder die Anwendung überarbeiteter Algorithmen.

**Nachträgliche Optimierung der Rechenlogik** Während der frühen Entwicklungsphase bestand ein grundlegendes Problem darin, dass beim Replay die gespeicherten Daten lediglich wiedergegeben wurden, wie sie während der ursprünglichen Messung aufgezeichnet worden waren. Das bedeutete, dass keine Neuberechnung der Tag-Position erfolgte – Anpassungen an Parametern oder Algorithmen blieben somit wirkungslos.

Im weiteren Verlauf wurde dieser Fehler behoben. Das Replay-Modul (`replayLogger.py`) wurde so erweitert, dass beim Einlesen der Logdaten die gespeicherten Rohwerte wie Winkel und Ankerpositionen erneut verarbeitet werden. Auf diese Weise erfolgt bei jedem Replay eine aktuelle Berechnung der Position mittels Triangulation. Dadurch können Änderungen in der Positionsberechnung zunächst rein softwareseitig überprüft und validiert werden – ganz ohne Hardwareeinsatz. Nach erfolgreicher Evaluierung lassen sich die neuen Parameter und Berechnungsverfahren schließlich in das Echtzeit-Tracking-System übernehmen.

**Visualisierung der Messdaten** Der Datenlogger stellt über eine TCP-Verbindung eine Schnittstelle zur grafischen Visualisierung her (z. B. durch das Skript `ui.py`). Jede Messung wird als JSON-Objekt an die Benutzeroberfläche übertragen und enthält neben der berechneten Position auch zugehörige Sensordaten. Die Benutzeroberfläche stellt folgende Informationen dar:

- Die aktuelle Position des Tags
- Visualisierte Sensorwerte (z. B. durch Farben, Pfeile oder Symbole)
- Zusatzinformationen wie Bewegung oder Risikobewertung

**Zusammenfassung des Datenflusses** Der vollständige Ablauf lässt sich wie folgt beschreiben:

1. **Messung:** Anker empfangen Signale und bestimmen Winkel sowie Sensordaten.

2. **Datenerfassung:** Python-Skripte lesen die Daten aus und speichern sie als JSONL.
3. **Speicherung:** Die Daten werden zeilenweise mit Zeitstempel archiviert.
4. **Replay:** Ein weiteres Skript liest die Daten, berechnet ggf. die Position und sendet sie weiter.
5. **Visualisierung:** Eine Benutzeroberfläche stellt Position und Sensordaten visuell dar.

### Beispiel einer vollständigen Messung

- Anker 1: Winkel =  $-31^\circ$ , Position =  $[3, 0]$
- Anker 2: Winkel =  $24^\circ$ , Position =  $[0, 0]$

#### Speicherung:

```
{"timestamp": "...",  
  "raw_sensor_data": {  
    "anchors": [[3, 0], [0, 0]],  
    "angles": [-31, 24],  
    "sensor_values": [...]  
  }  
}
```

**Replay:** Der Datenlogger berechnet erneut die Position und sendet z. B.:

```
{"point": {"position": [x, y], "Uncertainty": 0},  
  "sensor_values": [...]}
```

**Visualisierung:** Die Benutzeroberfläche zeigt Position, Bewegung und Risikobewertung an.

**Fazit** Durch die Trennung von Messung, Speicherung, Wiederverwendung und Visualisierung wurde ein System geschaffen, das sowohl im Live-Betrieb als auch für Offline-Analysen geeignet ist. Die Nutzung des JavaScript Object Notation Lines (JSONL)-Formats gewährleistet einfache Nachverarbeitung und hohe Transparenz im Entwicklungsprozess.

## 4.4 Ergebnisse und Evaluation

Die Implementierung des Datenloggers konnte erfolgreich in die bestehende Systemumgebung integriert werden. Die Erweiterung erfolgte durch ein einzelnes, in Python geschriebenes Modul, das mit nur wenigen Anpassungen in den vorhandenen Code eingebunden wurde. Diese modulare Herangehensweise ermöglichte eine einfache Integration, ohne die Stabilität der bestehenden C++- und Python-Strukturen zu beeinträchtigen.

Der Datenlogger ist in der Lage, strukturierte Messdaten zuverlässig zu erfassen, im JSONL-Format zu speichern und diese in der `replayLogger.py` zur erneuten Verarbeitung und Visualisierung bereitzustellen. Durch die gezielte Nachberechnung der gespeicherten Parameter kann die Parametrisierung und Optimierung der Tag-Ortung nun auch ohne aktiven Hardwarezugang erfolgen. Dies stellt einen wesentlichen Fortschritt gegenüber früheren Ansätzen dar, bei denen für jede Änderung eine Live-Verbindung zur u-blox-Hardware erforderlich war.

Zur weiteren Erleichterung der Nutzung wurde eine Batch-Datei (BAT-Datei) erstellt, die den Datenlogger automatisch startet und eine benutzerfreundliche Auswahl der abzuspielenden Daten ermöglicht. Dadurch können mehrere aufgezeichnete Datensätze parallel gespeichert, umbenannt und bei Bedarf erneut abgespielt werden. Dies erleichtert sowohl die Organisation als auch die Wiederverwendbarkeit der Messdaten erheblich.

Die Kommunikation zwischen Logger und Visualisierung über TCP funktioniert stabil, sodass eine Echtzeit- oder zeitnahe Darstellung der Messdaten in der grafischen Oberfläche möglich ist. Die Darstellung der Ergebnisse im UI ist erfolgreich umgesetzt worden; die gespeicherten Positions- und Sensordaten lassen sich korrekt laden und werden visuell ansprechend aufbereitet.

Ein weiterer Vorteil liegt in der verbesserten Teamfähigkeit des Systems. Da echte Messwerte gespeichert und wiederverwendet werden können, ist eine parallele Bearbeitung durch mehrere Teammitglieder möglich – unabhängig von der Verfügbarkeit der physischen Hardware.

Aktuell wurden nur kurze bis mittellange Messreihen (mehrere Minuten) getestet, die zuverlässig aufgezeichnet und verarbeitet werden konnten. Für den produktiven Einsatz müssen jedoch noch weiterführende Tests mit längeren Aufzeichnungen durchgeführt werden, um Speicherstabilität und Verarbeitungsgeschwindigkeit auch bei größeren Datenmengen zu validieren.

Insgesamt bildet der Datenlogger eine stabile Grundlage für künftige Entwicklungs- und Testarbeiten und ermöglicht sowohl effizientere als auch hardwareunabhängige Validierungsprozesse.

## 5 Grundlagen der Visualisierung

### 5.1 Anforderungen an eine effektive Visualisierung

Die visuelle Darstellung von Mess- und Sensordaten spielt eine zentrale Rolle bei der Entwicklung technischer Systeme. Im Kontext dieser Studienarbeit soll die Visualisierung nicht nur als Ausgabeinstrument dienen, sondern als aktives Analysewerkzeug die Entwicklung des Fahrerassistenzsystems unterstützen. Daraus ergeben sich spezifische Anforderungen an Struktur, Funktionalität und Performance der Visualisierung.

Ein wesentliches Kriterium ist die **Klarheit und Verständlichkeit** der Darstellung. Die Position des zu verfolgenden Objekts (Tag), die Winkelmessungen sowie weitere Sensordaten wie Geschwindigkeit oder Risikoindikatoren müssen für die Benutzer klar erkennbar und intuitiv interpretierbar sein. Die gewählte Darstellungsform darf keine zusätzliche Komplexität einführen, sondern soll das Verständnis der Systemdynamik fördern.

Zudem ist **Reaktionsfähigkeit** von hoher Bedeutung. Insbesondere bei der Wiedergabe von Logdaten im Replay-Modus ist sicherzustellen, dass Positionsdaten und Zustandsänderungen nahezu in Echtzeit visualisiert werden, ohne spürbare Verzögerung oder Datenverluste. Dies ist entscheidend für die genaue Nachverfolgung zeitkritischer Situationen und zur Validierung der Algorithmen.

Eine **strukturierte und modulare Darstellung** ermöglicht es, verschiedene Datenebenen simultan darzustellen. Dazu gehören beispielsweise die grafische Repräsentation von Ankerpositionen, Trajektorien, Messwinkeln sowie Risikoabschätzungen in Form von Farbcodierungen, Symbolen oder Vektorpfeilen. Die visuelle Trennung dieser Elemente soll die Lesbarkeit erhöhen und gleichzeitig Korrelationen zwischen den Daten sichtbar machen.

Weiterhin ist **Flexibilität** in der Konfiguration erforderlich. Die Visualisierung soll es erlauben, verschiedene Parameter interaktiv zu ändern, Messbereiche einzugrenzen oder unterschiedliche Datensätze zu vergleichen. Funktionen wie Zoom, Zeitfilterung oder Layer-Kontrolle erleichtern die gezielte Analyse.

Die Software muss zudem eine **hohe Stabilität und Performance** aufweisen. Auch bei großen Datenmengen oder längeren Wiedergaben dürfen keine Speicherprobleme oder Verzögerungen auftreten. Gleichzeitig soll der Ressourcenverbrauch moderat bleiben, um eine Nutzung auf Standardhardware zu gewährleisten.

Darüber hinaus ist die **Schnittstellenkompatibilität** entscheidend. Die Visualisierung kommuniziert über eine TCP-Verbindung mit dem Datenlogger und verarbeitet strukturierte JSON-Daten. Sie muss in der Lage sein,

kontinuierlich neue Informationen entgegenzunehmen, korrekt zu interpretieren und grafisch darzustellen.

Schließlich sollte die Visualisierung eine **benutzerfreundliche Interaktion** bieten. Die Bedienoberfläche muss auch für neue Projektmitglieder oder externe Tester leicht verständlich sein. Ergänzende Funktionen wie Legenden, Tooltips oder Echtzeit-Feedback erhöhen die Zugänglichkeit und erleichtern die Nutzung im Entwicklungsteam.

## 5.2 Darstellungsmöglichkeiten und -technologien

Die grafische Darstellung von Messergebnissen ist ein zentrales Element zur Analyse, Entwicklung und Validierung des Fahrerassistenzsystems. Ziel der Visualisierung ist es, komplexe Daten wie Winkelmessungen, berechnete Positionen und Sensordaten so aufzubereiten, dass sie schnell erfasst und zielgerichtet interpretiert werden können.

**Darstellungsformen** Für die Anforderungen dieses Projekts wurde eine zweidimensionale Darstellung gewählt. In einem kartesischen Koordinatensystem werden sowohl die festen Positionen der Anker als auch die berechnete Position des Tags eingezeichnet. Die Bewegungsrichtung kann über eine Spurhistorie (Trajektorie) oder durch Richtungspfeile visualisiert werden. Zur Darstellung zusätzlicher Informationen, wie beispielsweise der Risikobewertung, kommt eine Farbcodierung zum Einsatz: Je nach Risiko- oder Geschwindigkeitslevel kann der Datenpunkt beispielsweise in Grün, Gelb oder Rot eingefärbt werden.

Auch die Darstellung der Sensordaten erfolgt visuell, etwa durch Richtungslinien oder Symbole, die Informationen wie Bewegungsrichtung, Geschwindigkeit oder potenzielle Kollisionsgefahr übermitteln. Ein Beispiel wäre die Anzeige der „Speed along line“-Werte als Pfeillänge oder die Kodierung der Risikostufe über ein Symbol oder eine Farbfläche.

Zur besseren Nachvollziehbarkeit der Messabläufe werden zusätzlich Zeitreihen oder Replays implementiert, in denen frühere Positionen nacheinander dargestellt werden. Diese dynamische Komponente erleichtert die Bewertung der Bewegungsmuster und Reaktionen des Systems über den Zeitverlauf hinweg.

**Technologische Umsetzung** Für die technische Realisierung wurde Python als Basissprache verwendet, da es bereits im bestehenden Projektumfeld (z. B. Datenlogger, TCP-Client) genutzt wird und eine Vielzahl an Visualisierungsbibliotheken bietet. Die grafische Benutzeroberfläche basiert auf einem

einfachen Zeichenfenster, das Datenpunkte und Zusatzinformationen visuell aufbereitet.

Zur Anzeige der Daten kommen vor allem folgende Bibliotheken und Technologien zum Einsatz:

- **Matplotlib:** Für einfache 2D-Visualisierung der Anker, Tags und Bewegungslinien.
- **Pygame (alternativ):** Bei Bedarf für performantere, interaktive Darstellungen mit benutzerdefinierter Steuerung.
- **TCP-Kommunikation:** Die Visualisierung empfängt die Daten über eine TCP-Verbindung, die vom Datenlogger bereitgestellt wird (z. B. Skript `ui.py`).
- **JSON-Verarbeitung:** Die Logdateien und Echtzeitdaten liegen im JSON- bzw. JSONL-Format vor und werden zeilenweise eingelesen.

Der Aufbau der Visualisierung ist modular gehalten, sodass zukünftige Änderungen – etwa am Format der Sensordaten oder an der Positionsrechnung – ohne tiefgreifende Eingriffe umgesetzt werden können.

**Schnittstellen und Erweiterbarkeit** Die Benutzeroberfläche ist so gestaltet, dass verschiedene Datenquellen unterstützt werden: Neben Live-Daten aus dem Logger ist auch ein Replay-Modus möglich, in dem ältere Messungen abgespielt werden. Dies wird durch einen einfachen Startbefehl (z. B. über eine Batch-Datei oder Kommandozeilenparameter) gesteuert.

Die Datenstruktur aus der JSONL-Datei ermöglicht eine verlustfreie Weiterverarbeitung der gespeicherten Informationen. Durch die Trennung von Backend (Logger/Replay) und Frontend (Visualisierung) ist das System erweiterbar. Eine spätere Portierung auf moderne GUI-Frameworks (z. B. **Tkinter**, **PyQt**, oder webbasierte Tools wie **Plotly Dash**) ist ebenso denkbar wie eine Erweiterung auf 3D-Darstellungen mithilfe von OpenGL oder Unity.

**Optionale Abbildungsempfehlung** Eine illustrative Abbildung könnte den schematischen Ablauf der Datenvisualisierung zeigen – von der Messung über die Speicherung bis zur Anzeige. Beispielhaft könnte sie folgende Elemente enthalten:

- Position der Anker (als fixe Punkte)



- Messwinkel (als Linien vom Anker ausgehend)
- Position des Tags (als Punkt mit Zeitstempel)
- Risikobewertung als Farbring oder Symbol
- Spurverlauf der Tag-Position (z. B. gestrichelte Linie)

Eine solche schematische Darstellung erleichtert das Verständnis des Visualisierungskonzepts auch für fachfremde Leserinnen und Leser.

**Fazit** Die gewählten Technologien und Darstellungsformen ermöglichen eine performante, verständliche und flexible Visualisierung der Messergebnisse. Sie bilden damit eine wichtige Grundlage für die weiterführende Analyse, Parametrisierung und Validierung des Fahrerassistenzsystems.

## 6 Entwicklung der neuen Visualisierung

### 6.1 Analyse der bestehenden Visualisierung

Die bisherige grafische Benutzeroberfläche (UI) wurde in Python entwickelt und diente der Echtzeitdarstellung der aus dem Bluetooth-AoA-System ermittelten Tag-Position. Als zentrales Element wurde ein zweidimensionales kartesisches Koordinatensystem verwendet, innerhalb dessen ein einzelner Punkt die Bewegung des Tags repräsentierte. Die Position dieses Punktes wurde entweder durch eine direkte Live-Messung mit der Hardware oder durch eine vereinfachte Simulation erzeugt – eine Möglichkeit zur Wiederverwendung gespeicherter Daten durch einen Datenlogger war zu diesem Zeitpunkt noch nicht implementiert.

Zur besseren Nachvollziehbarkeit der geometrischen Zusammenhänge wurden auch die fest installierten Empfangseinheiten (Anker) im Koordinatensystem visualisiert. Von deren Positionen ausgehend wurden Linien gezeichnet, welche die gemessenen Einfallswinkel (Angle of Arrival, AoA) darstellten. Die Idee dahinter war, die Triangulation – also die Schnittpunktberechnung zweier Richtungslinien – visuell darzustellen und die damit berechnete Tag-Position zu validieren.

Allerdings wies diese Darstellung mehrere funktionale und darstellerische Einschränkungen auf. Eine grundlegende Schwäche bestand darin, dass der dargestellte Schnittpunkt der Winkel-Linien oft nicht exakt mit dem angezeigten Punkt des Tags übereinstimmte. Dies führte zu Inkonsistenzen, die insbesondere bei der Bewertung der Lokalisierungsgenauigkeit zu Verwirrung führen konnten. Die Ursache lag möglicherweise in Rundungsfehlern, unsauberen geometrischen Annahmen oder fehlender Synchronisierung zwischen Rechenmodul und Visualisierung.

Darüber hinaus fehlten wichtige Interaktionsmöglichkeiten und Darstellungsmodi: Sensorwerte, Bewegungsgeschwindigkeit oder Risikobewertungen konnten nicht separat angezeigt oder analysiert werden. Eine Zoom-Funktion, eine Zeitschrittsteuerung oder ein Replay-Modus waren nicht vorhanden. Da der Datenlogger zum damaligen Zeitpunkt noch nicht existierte, beschränkte sich die Funktionalität der Oberfläche auf Live-Datenströme oder manuell eingespeiste Simulationsdaten.

Zusammenfassend war die ursprüngliche Visualisierung funktional auf das Wesentliche reduziert, ermöglichte aber weder eine tiefere Analyse noch eine Nachbearbeitung historischer Messdaten. Die Einführung des Datenloggers und eine verbesserte Visualisierung sollen diese Defizite beheben und eine fundierte, nachvollziehbare sowie interaktive Darstellung aller relevanten Systemparameter ermöglichen.

## **6.2 Konzept einer verbesserten Visualisierung**

## **6.3 Implementierung der verbesserten Visualisierung**

## **6.4 Ergebnisse und Benutzerfreundlichkeit**

# **7 Test und Validierung**

## **7.1 Testmethodik und -umgebung**

## **7.2 Ergebnisse der Tests**

## **7.3 Diskussion der Testergebnisse und Optimierungspotentiale**

# **8 Kritische Bewertung und Ausblick**

## **8.1 Reflexion der erreichten Ergebnisse**

## **8.2 Grenzen der aktuellen Umsetzung**

## **8.3 Vorschläge für zukünftige Weiterentwicklungen**