



Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Projektdokumentation

Drahtlose Netzwerke B211

GPS-Spoofing via LimeSDR mini mit einem JavaFX-GUI

Duncan Wittner [592190]
SoSe25

Abgabe: 07.07.2025

Inhaltsverzeichnis

1. Einleitung.....	2
1.1. Zielsetzung.....	2
2. Stand des Projekts.....	2
2.1. Vorarbeiten.....	2
2.2. Ausgangslage zu Projektstart.....	3
3. Aufbau.....	3
3.1. Hardware.....	3
3.1.1. LimeSDR mini.....	3
3.1.2. Antenne.....	4
3.1.3. Rechner.....	5
3.2. Software.....	5
3.2.1. gps-sdr-sim.....	5
3.2.2. Ephemeriden.....	5
3.2.3. JavaFX-GUI.....	6
4. Durchführung.....	7
4.1. Projektablauf.....	7
4.2. Eigene Änderungen und Ergänzungen.....	9
5. Probleme und Lösungen.....	11
6. Fazit und Entwicklungsmöglichkeiten.....	12
7. Literaturverzeichnis.....	12

1. Einleitung

Das Projekt beschäftigt sich mit der Analyse und Erweiterung eines GNSS-Spoofing-Systems(Global Navigation Satellite System[1]) auf Basis eines LimeSDR mini sowie der Open-Source-Software gps-sdr-sim. Mithilfe dieses Systems haben wir die Möglichkeit, simulierte GPS-Signale zu erzeugen, um GNSS-Empfängern wie einem Smartphone frei definierbare Positionen vorzutäuschen.

Das Projekt wird im Rahmen des Kurses Drahtlose Netzwerke an der HTW Berlin durchgeführt. Das Projekt basiert auf der Bachelorarbeit und den dazugehörigen Tools von Ben Sauerländer.

1.1. Zielsetzung

Ziel des Projekts ist es, das vorhandene Spoofing System zunächst stabil starten zu können und seine grundlegende Funktionsweise zu überprüfen. Aufbauend darauf sollen gezielte Erweiterungen entwickelt werden. Insbesondere im Hinblick auf die Benutzerfreundlichkeit der grafischen Oberfläche als auch die Verbesserung und Erweiterung des Debug- und Systemtest-Moduls. Außerdem soll das System mit handelsüblichen GNSS-Empfängern wie einem Smartphone getestet werden, um eine praxisnahe Erfahrung des Spoofings zu erhalten und diese protokollieren zu können.

2. Stand des Projekts

2.1. Vorarbeiten

Vor Beginn dieses Projekts existierte ein lauffähiger Aufbau zur GPS-Simulation. Die zentrale Basis bildete ein LimeSDR mini als Sendekomponente sowie das Repository gps-sdr-sim[2] zur Erzeugung von GPS-L1-Signalen. Zusätzlich war eine auf JavaFX aufbauende Benutzeroberfläche vorhanden, die grundlegende Funktionen zur Steuerung und zum Starten des Spoofing-Mechanismus bereitstellte.

Dieses Gesamtsystem wurde im Rahmen einer vorhergehenden Bachelorarbeit implementiert. Die technische Architektur ermöglichte eine Speicherung von Szenarien, mit denen es möglich war, simulierte GNSS-Signale zu erzeugen und auszusenden. Durch diese Arbeit lag eine solide Basis vor, auf der das beschriebene Projekt aufbauen konnte.

2.2. Ausgangslage zu Projektstart

Zu Beginn dieses Projekts konnte die grafische Benutzeroberfläche gestartet werden und die technischen Voraussetzungen für die GNSS-Signalgenerierung waren implementiert und funktionstüchtig.

Allerdings war die vorliegende Konfiguration veraltet oder nicht direkt für die Nutzung an den Hochschul-PCs ausgelegt. Konkret fehlte ein gültiger Earthdata-Token für den automatisierten Download von Ephemeris Daten und die vorkonfigurierten Verzeichnispfade waren nicht mit der Verzeichnisstruktur der Hochschulrechner kompatibel. Ohne diese Änderungen konnte das Tool nicht wie vorgesehen genutzt werden.

Diese Punkte waren in der vorliegenden Dokumentation nicht ausreichend beschrieben, wodurch selbst einfache Anpassungen ans System mit hohem zeitlichen Aufwand verbunden waren.

3. Aufbau

3.1. Hardware

3.1.1. LimeSDR mini

Als Hauptkomponente zum senden von Signalen benutzen wir ein LimeSDR mini. Dieses Software-Defined Radio deckt einen Frequenzbereich von 10 MHz bis 3,5GHz[3] ab und ermöglicht Bandbreiten in Höhe von 30 MHz. Für unser Projekt nutzen wir das GPS-L1-Band, welches auf 1575,42 MHz[4] gesendet wird.

Der LimeSDR mini verfügt über einen Sendeanschluss(TX) und einen Empfangsanschluss (RX). In unserem Versuch benutzen wir nur den TX-Port. Außerdem spielt das Gerät I/Q-Daten ab, die aus gespooften GNSS-Signalen bestehen und zuvor von gps-sdr-sim erzeugt wurden.

Diese I/Q-Daten sind schon moduliert und haben alle notwendigen Signalwerte. Das SDR wandelt die I/Q-Daten in ein hochfrequentes analoges Signal um und überträgt dieses an die Antenne[5].

Zur optischen Kontrolle besitzt der LimeSDR mini eine mehrfarbige LED-Anzeige. Während des Sendens leuchtet diese grün auf, während sie im Ruhezustand oder bei Fehlern rot-orange leuchtet.

Das in diesem Projekt genutzte SDR wurde in einem 3D-gedruckten Gehäuse verbaut, um so mehr Schutz, Handhabung und Stabilisierung zu gewährleisten.



Abbildung 1: LimeSDR mini mit Gehäuse

3.1.2. Antenne

Für die Signalübertragung wird eine Stabantenne verwendet, die den Frequenzbereich um 1575 MHz abdeckt. Wichtig ist dabei, die Antenne am TX-Anschluss des LimeSDR mini anzuschließen, da nur dort das simulierte Sendesignal ausgegeben wird. Der RX Anschluss dient ausschließlich dem Empfang und würde für den Versuch nichts bringen.



Abbildung 2: LimeSDR mini mit Antenne

3.1.3. Rechner

Für den Versuch benutzen wir einen Computer, auf dem die Ubuntu LTS 24.04 als Betriebssystem installiert ist. Dieser Rechner beinhaltet die Softwarekomponenten gps-sdr-sim, LimeSuite sowie die JavaFX-basierte grafische Oberfläche. Das LimeSDR ist mit Antenne über einen USB 3.0 Port mit dem Rechner verbunden, wobei ein USB 2.0 auch ausreichen würde.

3.2. Software

3.2.1. gps-sdr-sim

gps-sdr-sim ist eine Open-Source Software zur Erzeugung von simulierten GPS-Signalen. Sie erzeugt digitale I/Q-Daten, die alle erforderlichen Parameter inklusive Modulation und Satellitendaten beinhalten. Diese I/Q-Daten können anschließend direkt über den LimeSDR mini abgestrahlt werden. Die Parameter können über die Konsole im Bereich der Startzeit, Geschwindigkeit, Dauer und weiteren Optionen angepasst werden.

Die Software unterstützt zwei Einstellungen:

- Static Mode, in dem eine feste Position über einen bestimmten Zeitraum simuliert wird
- Dynamic Mode, in dem eine Bewegungsbahn basierend auf mindestens zwei Punkten nachgebildet wird

Unabhängig vom gewählten Modus benötigt gps-sdr-sim aktuelle Ephemerisdaten, um realistische Satellitenpositionen und Zeitinformationen zu berechnen.

3.2.2. Ephemeriden

Die Ephemerisdaten beinhalten die Bahnen der GNSS-Satelliten. Diese Daten werden im sogenannten RINEX-Format bereitgestellt und vom gps-sdr-sim ausgelesen. Um Ephemerisdaten herunterzuladen, muss man sich einen Account auf der Plattform earthdata.nasa.gov erstellen und kann sich dann einen persönlichen Token für den automatisierten Download erstellen oder manuell die Daten herunterladen[6].

3.2.3. JavaFX-GUI

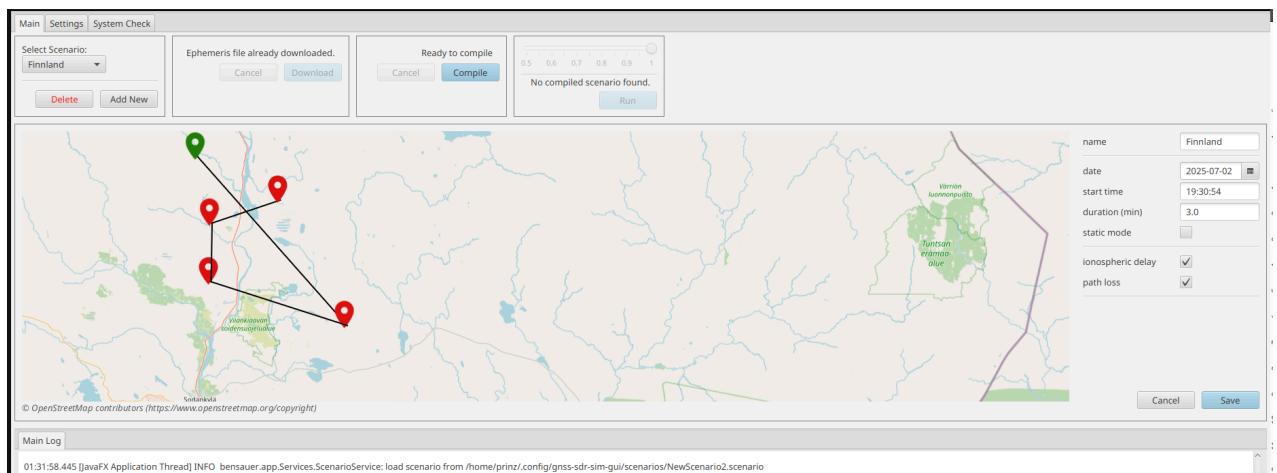


Abbildung 3: Hauptansicht von JavaFX-GUI

Die Hauptansicht des GUI auf Basis von JavaFX ist in Abbildung 3 zu erkennen. Diese ermöglicht es, Szenarien visuell zu erstellen, Parameter anzupassen und die Signal Simulierung zu starten, ohne in der Konsole arbeiten zu müssen. Die Hauptansicht des GUIs ist die Kartenansicht mit den dazugehörigen Scenarios und Parametern. In der Karte können Punkte markiert werden und für diese werden dann die GPS-Daten simuliert. Im unteren Abteil der Hauptansicht ist der Debugging Bereich, welcher Informationen über die Anwendung als auch Errors anzeigt.

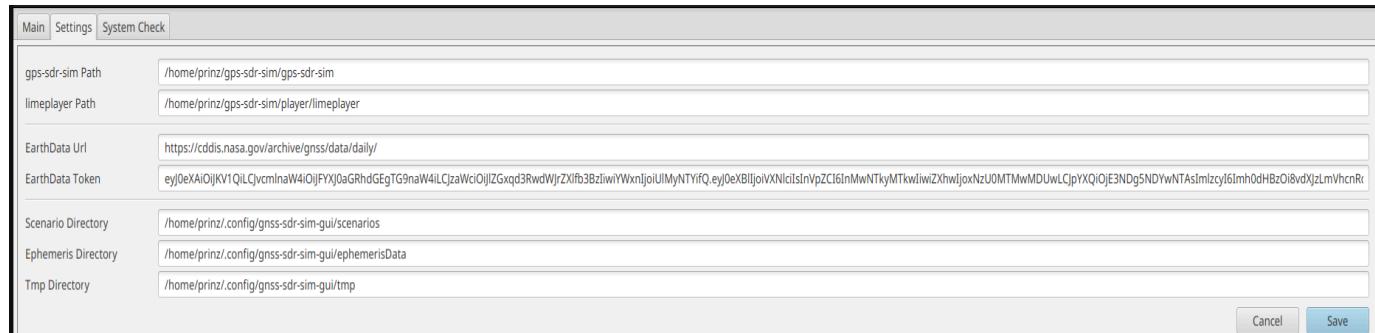


Abbildung 4: Settingsansicht von JavaFX-GUI

In dieser Ansicht kann man die Pfade für die Softwarekomponente als auch Verzeichnisse festgelegt werden. Dazu gehören unter anderem der Pfad zu gps-sdr-sim, die Earthdata URL und der zugehörige Token für den automatischen Zugriff auf aktuelle Ephemerisdaten. Ebenso werden hier die Ordnerstrukturen definiert. Mithilfe dieser Settings kann man die Software an unterschiedlichen Rechnern anpassen.

Die System-Check-Ansicht dient sowohl der Überprüfung der Pfade als auch der benötigten Komponenten und ob diese korrekt eingerichtet sind.

4. Durchführung

Alle im Folgenden beschriebenen Befehle, Installationsschritte und Konfigurationen sind zusätzlich in der README-Datei dokumentiert und können dort bei Bedarf im Detail begutachtet und verwendet werden.

4.1. Projektablauf

4.1.1. Vorbereitung

Im ersten Schritt wurden die benötigten Software-Abhängigkeiten auf dem Ubuntu-Rechner installiert. Um mit dem LimeSDR[7] mini zu kommunizieren war die Installation von LimeSuite erforderlich, welche die benötigten Treiber bereitstellt. Darüber hinaus mussten grundlegende Entwicklungswerzeuge wie GIt, CMake, Build-Essentials sowie die Bibliothek libusb für die USB-Kommunikation eingerichtet werden. Zusätzlich wurde das OpenJDK installiert um mit JavaFX arbeiten zu können.

Eine Überprüfung, ob der LimeSDR mini korrekt erkannt wurde, erfolgte mit dem Command LimeUtil, das eine Übersicht aller Lime-Geräte liefert und die Funktionsfähigkeit bestätigt.

4.1.2. Klonen

Für die Weiterentwicklung und den späteren Build-Prozess wurde das benötigte Repository gps-sdr-sim von Github heruntergeladen. Dies enthält sowohl den Build für das simulierte GNSS-Signal als auch den Limeplayer, der für die spätere Aussendung der erzeugten Daten über den LimeSDR mini zuständig ist.

4.1.3. Kompilieren

Zum Kompilieren wurde zunächst im Hauptverzeichnis von gps-sdr-sim der Kompilierungsprozess gestartet, um die Anwendung zu erstellen. Anschließend erfolgte eine separate Kompilierung im Unterordner limeplayer, der dazu führte, dass das Sendeprogramm für den LimeSDR mini gebaut wurde.

Wenn beide Kompilierungsschritte fehlerfrei verliefen, sollten zwei lauffähige Binaries sowohl für gps-sdr-sim als auch für den Limeplayer zur Verfügung stehen.

4.1.4. Funktionstest

Um die Funktionsfähigkeit vom LimeSDR zu testen, muss man ihn senden lassen. Am einfachsten geht das mit dem LimeQuickTest. Wird dieser als bestanden gemeldet, ist das LimeSDR LED grün blinkend und es ist funktionsbereit.

Zusätzlich kann mit dem Befehl “LimeUtil –find” sichergestellt werden, dass der LimeSDR mini vom System korrekt erkannt wird.

Erst nach einem erfolgreichen Testlauf sollte der weitere Durchlauf mit gps-sdr-sim und dem Limeplayer erfolgen.

4.1.5. Konfiguration

Nach fertiger Einrichtung ruft man die grafische Oberfläche auf und passt innerhalb der Settingsansicht den “limeplayer Path” und “gps-sdr-sim Path” an. An den Rechnern in der HTW Berlin ist dieser Pfad schon richtig eingestellt.

Zusätzlich musste der persönliche Earthdata-Token hinterlegt werden, um den automatisierten per Knopfdruck Download der aktuellen EphemerisDaten zu gewährleisten. Diese Einstellung kann auch in der Settingsansicht vorgenommen werden.

Diese Konfiguration stellt sicher, dass alle hinzukommenden Szenarien ohne manuelle Anpassungen verarbeitet werden können.

4.1.6. Erster Testszenarienlauf

Als Testszenario wurde über die JavaFX-GUI ein dynamisches Szenario mit mehreren Wegpunkten erstellt, die eine realistische Route bilden sollen. Im vorliegenden Beispiel wird eine Route in Finnland simuliert.

In der grafischen Oberfläche wurden dazu Parameter wie Datum, Startzeit, Gesamtdauer sowie zusätzliche Optionen eingestellt. Die Gesamtdauer wurde bewusst auf 3 Minuten gestellt, damit das verwendete Smartphone ausreichend Zeit hat, sich mit dem Spoofing-Signal zu synchronisieren. Das Szenario konnte anschließend gespeichert und als ausführbare .bin-Datei kompiliert werden, um es mit dem Limeplayer-Befehl auszusenden.

Nach dem Start der Simulation wurde mit einem Android Handy und der App “Locus Map” überprüft, ob die Signale korrekt empfangen und interpretiert werden. Dabei fiel auf, dass das Gerät während der Simulation die vorgegebenen Positionen erkannt hat, was die erfolgreiche Funktionsweise bestätigt.

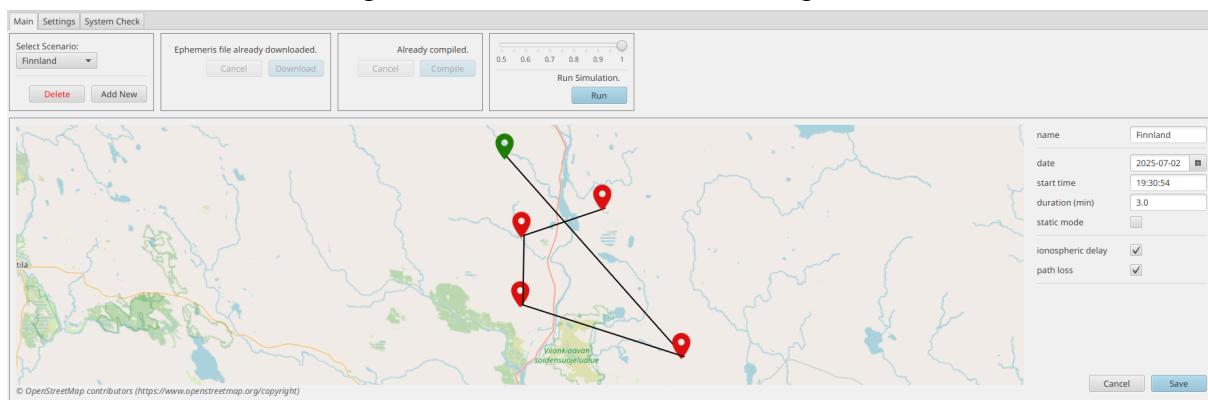


Abbildung 5: Testszenario in Finnland

Die Abbildung zeigt ein dynamisches Szenario, was man an dem fehlenden Haken bei "static mode" erkennt. Die roten Markierungen stehen für die definierten Zwischenziele, die während der Simulation durchlaufen werden. Der grüne Marker kennzeichnet den Startpunkt.

4.2. Eigene Änderungen und Ergänzungen

Im Rahmen des Projektziels wurden mehrere Anpassungen und Erweiterungen am bestehenden Code vorgenommen, um die Bedienbarkeit als auch die Weiterentwicklung des Systems zu vereinfachen.

4.2.1. Anpassung der Standardpfade

Die Pfadkonfiguration der Anwendung wurde angepasst, denn zwar waren bereits Standardpfade im Code hinterlegt, diese bezogen sich jedoch nicht auf die Verzeichnisstruktur der aktuellen HTW-Computer. Daher wurden gezielt die Pfade für die gps-sdr-sim Anwendung sowie den Limeplayer auf das neue Verzeichnis angepasst, um die Nutzung ohne zusätzliche Konfiguration zu ermöglichen.

Diese Anpassung wurde innerhalb der Methode loadDefaults() gemacht, damit die Einstellungen bei jedem Programmstart neu geladen werden.

```
public static void loadDefaults() {
    logger.info("Load default settings");
    change(
        "/home/student/gps-sdr-sim/gps-sdr-sim", //Change to your path - Standard is ready to go for the HTW Berlin computers
        "/home/student/gps-sdr-sim/player/limeplayer", //Change to your path - Standard is ready to go for the HTW Berlin computers
        "https://cddis.nasa.gov/archive/gnss/data/daily/",
        "",
        settingsPath.getParent().resolve("scenarios").toString(),
        settingsPath.getParent().resolve("ephemerisData").toString(),
        settingsPath.getParent().resolve("tmp").toString());
    onChangedListners.forEach(Runnable::run);
}
```

Abbildung 6: Ändern der Standardpfade im Code

4.2.2. Verbesserungen im LogViewController

Eine Schwäche der bisherigen GUI war, dass die Log-Ausgaben weder markiert noch kopiert werden konnten. Um diese Einschränkung zu beheben, wurde die Log-Anzeige überarbeitet. Statt der bisherigen Kombination aus ScrollPane und TextFlow wird nun ein TextArea [8] verwendet, in dem alle Logeinträge direkt angezeigt werden.

Das TextArea ist so konfiguriert, dass es nicht editierbar ist, aber die Einträge markier- und kopierbar bleiben. Dafür wurden die Methoden createTab() und addLogEntry() im LogViewController angepasst. Diese Änderungen erleichtern vor allem das Debugging und die Fehlersuche während der Anwendung

```

// add the logEntry to the TextArea of the tab with given id
private void addLogEntry(final String context, final String message, final int id) {
    String logEntry = "";

    if (context != null && !context.isEmpty()) {
        String c = context.trim();
        if (c.endsWith(":")) {
            logEntry += c + " ";
        } else {
            logEntry += c + ": ";
        }
    }

    logEntry += message + "\n";

    TextArea textArea = logViews.get(id);
    textArea.appendText(logEntry);

    // scrolls to bottom
    textArea.setScrollTop(Double.MAX_VALUE);
}

// creates a new tab with a TextArea for logEntries
private Tab createTab(final String name) {
    TextArea textArea = new TextArea();
    textArea.setEditable(false);
    textArea.setWrapText(true);

    textArea.setStyle("-fx-padding: 2 5 2 5; -fx-line-spacing: 0.8em;");

    logViews.add(textArea);
    Tab tab = new Tab(name);
    tab.setContent(textArea);

    return tab;
}

```

Abbildung 7: Codeverbesserung im LogViewerController

4.2.3. Erweiterung der System-Check-Funktion

Die System-Check-Ansicht wurde um einen Button erweitert, welcher die Prüfung der vorausgesetzten installierten Programme erleichtert. Dieser Button prüft ob die hinterlegten Pfade tatsächlich gültige ausführbare Dateien für gps-sdr-sim und den Limeplayer enthalten. Mithilfe dieses Buttons lässt sich vor einem Testszenario schon sicherstellen, ob alle benötigten Komponenten vorhanden sind.

Die Prüflogik wurde im SystemCheckController implementiert. Nach Betätigung des Buttons wird kontrolliert, ob die hinterlegten Dateien im angegebenen Verzeichnis existieren. Dabei wird nicht nur erkannt, ob ein Programm fehlt, sondern auch, ob der gesetzte Pfad in Einstellungen falsch eingetragen wurde. Das Ergebnis wird über zwei separate Status-Labels in der GUI angezeigt. Dadurch hat der Nutzer eine direkte Rückmeldung, ohne einen kompletten Testlauf zur manuellen Prüfung machen zu müssen.

```

<children>
    <Button fx:id="limeSdrConnectionButton" mnemonicParsing="false" text="LimeSDR connected?" />
    <Label fx:id="limeSdrConnectionResult" text="?" GridPane.columnIndex="1" />
    <Button fx:id="programsCheckButton" text="Programs installed?" onAction="#checkPrograms" GridPane.rowIndex="1"/>
    <Label fx:id="gpsCheckResult" text="?" GridPane.columnIndex="1" GridPane.rowIndex="1"/>
    <Label fx:id="limeCheckResult" text="?" GridPane.columnIndex="1" GridPane.rowIndex="2"/>

```

Abbildung 8: Erweiterung der Systemcheckview.fxml um den Button

```

@FXML
private void checkPrograms() {
    Path gpsPath = SettingsService.getGpsSdrSimPath();
    Path limePath = SettingsService.getLimeplayerPath();
    boolean gpsExists = Files.exists(gpsPath);
    boolean limeExists = Files.exists(limePath);

    gpsCheckResult.setText(gpsExists ? "gps-sdr-sim installed" : "gps-sdr-sim not installed");
    gpsCheckResult.setTextFill(gpsExists ? Color.GREEN : Color.RED);
    limeCheckResult.setText(limeExists ? "limeplayer installed" : "limeplayer not installed");
    limeCheckResult.setTextFill(limeExists ? Color.GREEN : Color.RED);

    logger.info("Checking if all required programs are installed");
}

```

Abbildung 9: Erstellung der Funktion checkPrograms()

5. Probleme und Lösungen

Während der Umsetzung des Projekts traten einige Probleme auf, die gezielt gelöst werden mussten. Eine essentielle Problematik bestand in der vorgegebenen Pfadkonfiguration innerhalb der grafischen Benutzeroberfläche. Diese Pfade waren ursprünglich auf eine andere Verzeichnisstruktur ausgelegt und haben daher an den jetzigen Hochschul-Computern nicht ohne Anpassung funktioniert. Als Folge konnten Programme wie gps-sdr-sim und Limeplayer nicht gefunden oder gestartet werden, trotz „offiziellen“ bestandenen Tests und Prüfungen außerhalb der GUI. Dies hat den gesamten Arbeitsablauf erheblich behindert. Besonders ärgerlich war es, dass da dieser vergleichsweise triviale Fehler einen erheblichen Zeitaufwand verursachte, obwohl die notwendigen Pfadangaben in der bestehenden Dokumentation nur unzureichend beschrieben waren.

Zur Behebung wurde der Standardpfad im Quellcode angepasst, sodass die Programme nun im voreingestellten Verzeichnis erkannt werden. Das Hauptziel dahinter war, eine fehleranfällige Konfiguration beim ersten Start der Anwendung zu verhindern.

Zusätzlich ergaben sich Schwierigkeiten durch die teilweise unvollständige Dokumentation im Hinblick auf den Limeplayer. Nach dem erfolgreichen Kompilieren von gps-sdr-sim musste der Limeplayer separat über einen eigenen Make-Prozess gebaut werden, was in der Anleitung nicht eindeutig beschrieben war. Diese fehlende Information führte zu zusätzlichen Fehlversuchen und Zeitverlust. Um diesen Fehler nicht repetitiv wiederholen zu lassen, habe ich in der README-Datei eine explizite Anleitung und auf diesen potenziellen Fehler extra hingewiesen.

Ein weiteres Problem zeigte sich beim Testen des Static Mode von gps-sdr-sim. In diesem Modus wurde ein festes Ziel über eine längere Zeit simuliert. In der Praxis konnte jedoch kein genaues Spoofing-Signal beobachtet werden, da Smartphones diesen statischen GPS-Spoofers als unrealistisch markierten und die Position nur fehlerhaft im Breitengrad übernahmen. Eine Lösung bestand darin, ausschließlich dynamische Szenarien zu testen mit realistischen und mehreren Wegpunkten. Diese Vorgehensweise wurde bereits in der vorliegenden Bachelorarbeit von Ben Sauerländer dokumentiert und wurde daher als Lösung übernommen.

6. Fazit und Entwicklungsmöglichkeiten

Das Projekt konnte erfolgreich die Grundlage eines GNSS-Spoofing-Systems auf Basis von dem LimeSDR mini und gps-sdr-sim analysieren und gezielt erweitern. Der bestehende Aufbau wurde aufbereitet und an die aktuelle Rechnerumgebung angepasst und mit Tests validiert. Dabei wurde die GUI verbessert und eine höhere Fehlertoleranz ist die Folge.

Trotz einiger Probleme, wie der genannten fehlerhaften Pfadkonfiguration oder der unzureichenden Dokumentation zum Limeplayer, konnte die Funktionalität des Systems im Projekt sichergestellt werden. Diese Hindernisse führten allerdings, wie schon erwähnt, zu einem höheren Zeitaufwand, wodurch weniger Zusatzfunktionen umgesetzt werden konnten als ursprünglich erwartet. Insgesamt bin ich jedoch zufrieden mit dem Ergebnis, da die wesentlichen Ziele erreicht wurden.

Für die Zukunft stehen mehrere Entwicklungsmöglichkeiten offen. So wäre es sinnvoll, das System auf weitere GNSS-Standards auszuweiten, um noch realistische Szenarien zu erstellen. Andernfalls könnte man die Benutzeroberfläche noch weiter ausbauen und potenziell sogar gestalten.

Außerdem wären Presettings für bestimmte Routen oder Locations ein Denkansatz.

7. Literaturverzeichnis

- [1] „Global Navigation Satellite System (GNSS) and Satellite Navigation Explained“, Advanced Navigation. Zugegriffen: 6. Juli 2025. [Online]. Verfügbar unter:
<https://www.advancednavigation.com/tech-articles/global-navigation-satellite-system-gnss-and-satellite-navigation-explained/>
- [2] T. Ebinuma, *osqzss/gps-sdr-sim*. (6. Juli 2025). C. Zugegriffen: 6. Juli 2025. [Online].
Verfügbar unter: <https://github.com/osqzss/gps-sdr-sim>
- [3] admin, „LimeSDR Mini Unboxing and Initial Review“, rtl-sdr.com. Zugegriffen: 6. Juli 2025. [Online]. Verfügbar unter:
<https://www.rtl-sdr.com/limesdr-mini-unboxing-initial-review/>
- [4] „GPS Signal Plan - Navipedia“. Zugegriffen: 6. Juli 2025. [Online]. Verfügbar unter:
https://gssc.esa.int/navipedia/index.php/GPS_Signal_Plan
- [5] „IQ Sampling | PySDR: A Guide to SDR and DSP using Python“. Zugegriffen: 6. Juli 2025. [Online]. Verfügbar unter: <https://pysdr.org/content/sampling.html>
- [6] N. Earth Science Data Systems, „Your Gateway to NASA Earth Observation Data | NASA Earthdata“. Zugegriffen: 7. Juli 2025. [Online]. Verfügbar unter:
<https://www.earthdata.nasa.gov/>
- [7] T. Ebinuma, *osqzss/LimeGPS*. (5. Juli 2025). C. Zugegriffen: 7. Juli 2025. [Online].

Verfügbar unter: <https://github.com/osqzss/LimeGPS>

- [8] J. Jenkov, „JavaFX TextArea“. Zugegriffen: 7. Juli 2025. [Online]. Verfügbar unter: <http://jenkov.com/tutorials/javafx/textarea.html>