Live-Darstellung aller Luftfahrtsysteme in näherer Umgebung auf einer digitalen Übersichtskarte

Ingo Weinmann

Technische Hochschule Wildau, Hochschulring 1, 15745 Wildau

Abstract

Im Rahmen des Forschungsprojektes ALARM[1] wurde ein Gesamtsystem bestehend aus mehreren unbemannten Luftfahrtsystemen, Sensorik, Datenerfassung und Datenübertragung, Luftraumüberwachung und Flugleitung entwickelt, welches vorrangig zur Luftunterstützung bei Rettungs- und Katastropheneinsätzen ausgelegt ist.

Ein Bestandteil des Projektes war die Implementierung einer Software, welche primär dazu dient, Position und Telemetriedaten aller Luftfahrtsysteme in der näheren Umgebung auf einer Übersichtskarte darzustellen. Hierdurch können beispielsweise Kollisionen zwischen bemannten und unbemannten Luftfahrzeugen vermieden werden.

Daneben ermöglicht die entwickelte Software unter anderem auch die Anzeige eines Live-Videostreams einer Beobachtungsdrohne.

Es wurde eine webbasierte Anwendung entwickelt, damit diese auf allen Betriebssystemen lauffähig ist.

Um Positionen und Telemetriedaten aller Luftfahrzeuge zu empfangen, werden Funkempfänger für das Frequenzband 1090 MHz (ADS-B) verwendet.

Die ADS-B Daten werden mithilfe von Software-Bibliotheken dekodiert und an einen zentralen MQTT-Broker gesendet. Hierdurch können beliebige Clientanwendungen im selben Netzwerk diese Daten empfangen und nahezu in Echtzeit verarbeiten und anzeigen.

In meinem Beitrag plane ich, auf Software-Architektur und verwendete Softwarebibliotheken einzugehen, die vom Empfang der Positions- und Telemetriedaten bis zu deren Anzeige im Webbrowser nötig sind.

1. Motivation

Beim Einsatz von Drohnen zur Überwachung von Naturkatastrophen wie etwa Waldbränden muss stets sichergestellt werden, dass die eingesetzten Drohnen den nötigen

[1] https://www.th-wildau.de/forschung-transfer/luftfahrttechnik/forschungsaktivitaeten/alarm/

Sicherheitsabstand zu anderen Luftverkehrsteilnehmern, z.B. Rettungshelikoptern einhalten.

Mittels eines auf ADS-B (Automatic Dependent Surveillance – Broadcast) basierenden Kollisionswarnsystems und zugehöriger Software können Daten wie bspw. Flugkennung, Position und Kurs von allen mit ADS-B Sendern ausgestatteten Luftfahrtsystemen empfangen werden.

Es wird eine Software implementiert, die über Funk empfangene ADS-B Signale von Luftverkehrsteilnehmern in der Umgebung dekodiert und auf einer digitalen Übersichtskarte darstellt.

2. Umsetzung

2.1 Hardware

Zum Betrieb der Software wird der Einplatinencomputer Raspberry Pi 4 verwendet. Durch seinen geringen Platz- und Strombedarf ist dieser Computer sehr flexibel und mobil einsetzbar – ideal für die Bedingungen des Forschungsprojektes ALARM.

Daneben wird ein RTL-SDR (Software Defined Radio) Empfänger zum Empfang der ADS-B Signale benötigt. Dieser wird per USB am Einplatinencomputer angeschlossen. ADS-B Signale werden auf einer Frequenz von 1090 MHz gesendet. Daher sollten der ADS-B-Empfänger und die daran angeschlossene Antenne idealerweise auf diese Frequenz optimiert sein.

2.2 Betriebssystem

Als Betriebssystem wird Raspberry Pi OS[2] verwendet, welches auf der Linux Distribution Debian 11, Codename „Bullseye“[3] basiert.

2.3 Empfang und Dekodierung von ADS-B Signalen

Zur softwareseitigen Dekodierung der ADS-B Daten kommt die Bibliothek dump1090[4] zum Einsatz, in einer von der Firma FlightAware[5] angepassten Version dump1090-fa. Die Software kann in einer aktuellen Version von Github bezogen werden:

https://github.com/flightaware/dump1090

Nach der Installation des Debian-Paketes wird dump1090 als systemd[6]-Hintergrunddienst gestartet. Dieser Dienst liest alle unterstützten RTL-SDR Empfänger periodisch aus und stellt die dekodierten Daten für andere Programme zur Verfügung.

Beispielhaft kann dump1090-fa mit folgenden Parametern gestartet werden:

```
/usr/bin/dump1090-fa --quiet --device-type rtlsdr --device-index 0 --gain 60 --adaptive-range --fix --max-range 360 --net-ro-port 30002 --net-sbs-port 30003 --net-bi-
```

[2] https://www.raspberrypi.com/software/

[3] https://www.debian.org/releases/bullseye/

[4] https://github.com/flightaware/dump1090

[5] https://flightaware.com/

[6] https://systemd.io/

```
port 30004,30104 --net-bo-port 30005 --json-location-accuracy 2 --write-json
/run/dump1090-fa
```

Das Debian-Paket von dump1090-fa stellt zusätzlich einen lighttpd[7]-basierten HTTP Dienst bereit. Über diesen Dienst können aktuelle ADS-B Informationen per HTTP Request abgefragt werden.

Die HTTP Response enthält Daten über die aktuell empfangbaren Luftverkehrsteilnehmer, kodiert im JSON Format.

Wichtige im JSON enthaltene Attribute sind: Flugkennung (hex), barometrische Höhe (alt_baro), Geometrische Höhe (alt_geom) und Flugrichtung relativ zum Boden (track). Hier ein Beispieldatensatz:

```
{
  "now":1620734968.3,
  "messages":19654,
  "aircraft":[
    {
      "hex":"4d2133",
      "alt_baro":45000,
      "alt_geom":45825,
      "track":255.6,
      ...
    }
  ]
}
```

Um die vom lighttpd-Dienst abgefragten ADS-B Daten weiterzuleiten wurde ein Python Skript entwickelt. Dieses fragt den HTTP Dienst periodisch ab und dekodiert das JSON-Format für die weitere Verarbeitung.

2.4 Weiterleitung an MQTT Broker

Um die dekodierten ADS-B Daten für eine Vielzahl an Clients zur Verfügung zu stellen bietet sich die Verwendung eines Message Brokers an. An diesen können alle ADS-B Daten weitergeleitet werden. Im späteren Verlauf ist es möglich, dass diese Daten auch von anderen Hosts im Netzwerk kommen.

Als Nachrichtenprotokoll wird MQTT[8] verwendet. MQTT ist ein standardisiertes Nachrichtenformat, das vorrangig für IoT-Anwendungen konzipiert wurde. MQTT eignet sich besonders, um Hardware-Ressourcen wie Netzwerkbandbreite, CPU-Auslastung und Arbeitsspeicher-Auslastung zu sparen. Clientbibliotheken für MQTT sind für alle namhaften Programmiersprachen verfügbar.

[7] https://www.lighttpd.net/
[8] https://mqtt.org/

Als MQTT Broker wird der im Open-Source Umfeld häufig verwendete Broker Eclipse Mosquitto[9] verwendet.

Damit auch Web Clients in der Lage sind, sich mit Eclipse Mosquitto zu verbinden, muss das Websocket Protokoll aktiviert werden. Eine Beispielkonfiguration (mosquitto.conf) sieht wie folgt aus:

```
listener 1883 0.0.0.0
listener 9001 0.0.0.0
protocol websockets
allow_anonymous true
log_type warning
```

Die empfangenen und dekodierten ADS-B-Signale werden mithilfe der Python Bibliothek paho-mqtt[10] an den MQTT Broker weitergeleitet.

Die folgende Abbildung 1 – Datenfluss der ADS-B Signale fasst die Abläufe aus den Abschnitten 2.3 bis 2.4 schematisch zusammen.

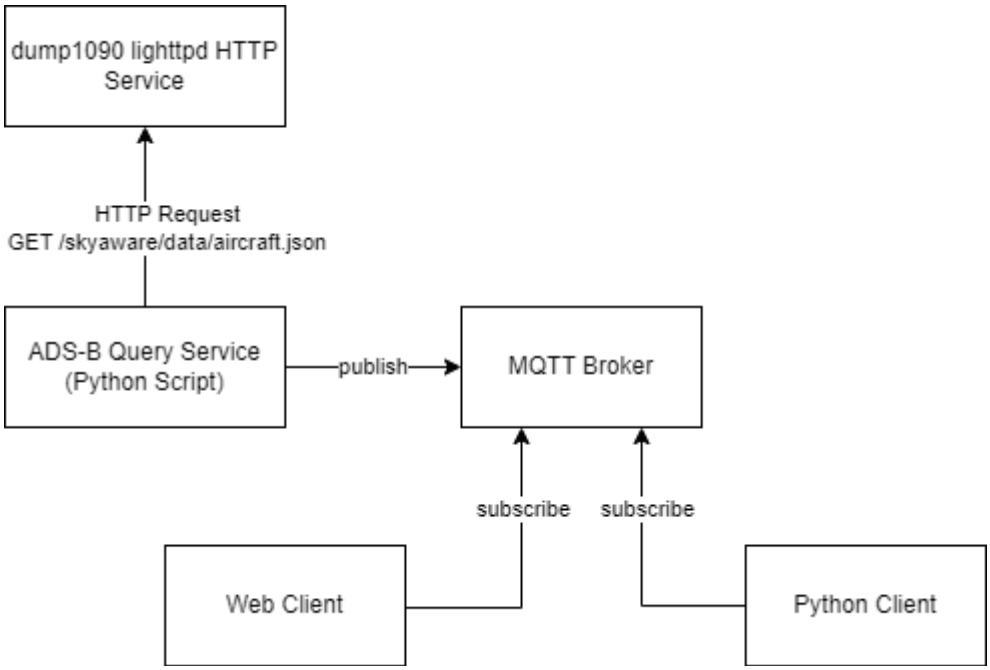

Abbildung 1 – Datenfluss der ADS-B Signale

2.4 Darstellung im Web Frontend

Zur Darstellung der Luftverkehrsteilnehmer auf einer Übersichtskarte wird eine Webanwendung implementiert, basierend auf der JavaScript Bibliothek React[11]. Durch die Verwendung von Web-Technologien ist die Anwendung unabhängig vom Betriebssystem

[9] https://mosquitto.org/
[10] https://pypi.org/project/paho-mqtt/
[11] https://react.dev/

und ohne gesonderte Installation auf beliebigen Endgeräten (z.B. Notebook, Smartphone, Tablet-PC) lauffähig.

Zur Darstellung einer digitalen Übersichtskarte wird die Bibliothek MapLibre[12] verwendet. Innerhalb der Webanwendung kann zwischen verschiedenen Kartendiensten umgeschaltet werden. Diese können über externe Kartendienste aus dem Internet bezogen werden. Sofern keine Internetverbindung vorhanden ist, ist es möglich, die Kartendaten auch über einen Proxy Server vorzuhalten. Hierfür wird der Dienst MapProxy[13] verwendet.

Abbildung 2 – Kartenstile

- Abbildung 2 (links): Digitale Orthophotos, für die Region Berlin Brandenburg sind diese über den Dienst Geobasis BB[14] verfügbar
- Abbildung 2 (Mitte): OpenStreetMap Kartenstil
- Abbildung 2 (rechts): Minimaler Kartenstil, bezogen über Maptiler[15]

Mittels der Bibliothek MQTT.js[16] verbindet sich die Webanwendung per WebSocket mit dem MQTT Broker und erhält die ADS-B Daten im JSON-Format.

Zur Darstellung der Luftverkehrsteilnehmer auf einer digitalen Karte wird die Bibliothek DECK.GL[17] verwendet. Diese erlaubt eine besonders performante Darstellung von großen, sich dynamisch ändernden Datensätzen beliebiger Art auf einer Karte.

Die finale Webanwendung ist in Abbildung 3 - Finale Webanwendung zu sehen.

[12] https://maplibre.org/
[13] https://mapproxy.org/
[14] https://geobasis-bb.de/lgb/de/
[15] https://www.maptiler.com/
[16] https://github.com/mqttjs/MQTT.js
[17] https://deck.gl/

Abbildung 3 - Finale Webanwendung