OST
Ostschweizer
Fachhochschule

SAFRAN
Vectronix

BACHELORARBEIT

VERNETZUNG VON BEOBACHTUNGSSTATIONEN

Studierende:	Benjamin Koch, Patrick Good
Referent:	Prof. Dr. Carlo Bach
Korreferent:	Prof. René Pawlitzek
Industriepartner:	Safran Vectronix AG
Abgabetermin:	26.08.2022

Zusammenfassung

Die Bachelorarbeit «Vernetzung von Beobachtungsstationen» befasst sich mit dem Produkt MOSKITO TI der Firma Safran Vectronix AG. MOSKITO TI ist ein militärisch genutztes Fernglas, welches eine Distanz- und Winkelmessung durchführen kann und mit einem Thermal Imaging Chip in der Lage ist, Bilder in einem anderen Wellenbereich aufzunehmen. Zudem ist das Gerät mit einem Restlichtverstärker ausgestattet, mit welchem ebenfalls Bilder aufgenommen werden können.

Das Gerät bietet trotz eingebautem Bluetooth- und WLAN-Modul nicht die Möglichkeit des Datenaustausches mit anderen MOSKITO TIs. Die vorliegende Bachelorarbeit stellt Strukturen und Funktionen bereit, um den Datenaustausch unter den Geräten zu ermöglichen.

Der für die Vernetzung gewählte Lösungsansatz ist das Aufspannen eines eigenen Overlay-Netzwerkes über das vorhandene physische Netzwerk, mit dem mehrere MOSKITO TIs verbunden sind. Jedes MOSKITO TI generiert und verwaltet selbst eine Liste mit den verbundenen Geräten. Sobald ein Gerät hinzustösst oder sich aus dem physischen Netzwerk entfernt, wird dies automatisch durch die anderen verbundenen Geräte erkannt und im Overlay-Netzwerk bzw. in der eigenen Liste angepasst. Die An- und Abmeldung sowie die Aktualisierung der Geräte im Netzwerk erfolgt durch UDP Broadcast Pakete.

Wenn die Verbindung zwischen den Geräten über das Overlay-Netzwerk hergestellt und die Handover Funktion auf dem Gerät aktiviert ist, werden ab diesem Zeitpunkt sämtliche aufgenommen Messungen (Distanz, Azimut, Longitude, Altitude, Latitude usw.) in Form von Messpunkten auf alle im Overlay-Netzwerk liegenden, ausgewählten Geräte verteilt. Bilder, die aufgenommen werden, können ebenfalls über das Overlay-Netzwerk an sämtliche selektierte Geräte übermittelt werden. Weiter wird mit der Vernetzung der Beobachtungsstationen eine passive Distanzmessung mittels Triangulation ermöglicht. Diese Funktion basiert auf dem Austausch des Gerätewinkels und weiterer Standortdaten, welche zur Berechnung der Distanz benötigt werden. Die Kommunikation und das Versenden der Daten erfolgt mittels HTTP Protokoll über eine REST-Schnittstelle.

Die Aufgabe der Vernetzung von MOSKITO TIs wurde erfolgreich gelöst. Die Arbeit dient dem Industriepartner Safran Vectronix AG als Anwendungsbeispiel und soll aufzeigen, wie eine solche Vernetzung erstellt werden kann und welche Nutzeranwendungen sich daraus ergeben.

Abstract

The bachelor thesis "Networking of observation stations" deals with the product MOSKITO TI from the company Safran Vectronix AG. MOSKITO TI is a pair of binoculars used for military purposes, which can perform distance and angle measurement and, with a thermal imaging chip, is able to record images in a different waveband. In addition, the device is equipped with a residual light amplifier, which can also be used to capture images.

Despite the built-in Bluetooth and WLAN module, the device does not offer the possibility of data exchange with other MOSKITO TIs. This bachelor thesis provides structures and functions to enable data exchange between the devices.

The solution chosen for networking is to set up a separate overlay network over the existing physical network, to which several MOSKITO TIs are connected. Each MOSKITO TI generates and manages its own list of connected devices. As soon as a device joins or leaves the physical network, this gets automatically recognised by the other connected devices and adjusted in the overlay network or in its own list. The registration and deregistration as well as the update of the units in the network takes place by UDP broadcast packets.

If the connection between the devices is established via the overlay network and the handover function is activated on the device, all recorded measurements (distance, azimuth, longitude, altitude, latitude, etc.) are distributed to all selected devices in the overlay network in the form of measurement points from this time on. Images that are recorded can also be transmitted to all selected devices via the overlay network. Furthermore, the networking of the observation stations enables passive distance measurement by means of triangulation. This function is based on the exchange of the device angle and other location data needed to calculate the distance. The communication and sending of the data takes place using HTTP protocol via a REST interface.

The task of networking MOSKITO TIs was successfully solved. The work serves as an application example for the industry partner Safran Vectronix AG and is intended to show how such networking can be created and which user applications result from it.

Danksagung

An dieser Stelle möchten wir ein herzliches Dankeschön an alle richten, die uns im Rahmen der Erarbeitung unserer Bachelorarbeit «Vernetzung von Beobachtungsstationen» unterstützt haben.

Wir möchten unseren Dank an die Firma Safran Vectronix AG richten, die uns als Industriepartner die Durchführung unserer Bachelorarbeit überhaupt ermöglichte.

Namentlich möchten wir uns herzlich bei Michael Eichhorn, Mitarbeiter der Firma Safran Vectronix AG, bedanken. Er betreute uns während der ganzen Arbeit und sorgte dafür, dass wir sämtliches benötigtes Material zur Verfügung gestellt bekamen.

Ein besonderer Dank gilt unserem Referenten Prof. Dr. Carlo Bach, der unsere Bachelorarbeit betreut hat. Für die konstruktiven Anregungen und die Hilfsbereitschaft möchten wir uns herzlich bedanken.

Ausserdem bedanken wir uns bei unserem Korreferenten Prof. René Pawlitzek, der uns ebenfalls stets seine Expertise und seinen Rat zur Verfügung stellte.

Benjamin Koch und Patrick Good, August 2022

Inhaltsverzeichnis

1 Einleitung

Für das Verständnis der vorliegenden Arbeit folgen in diesem Kapitel einige wichtige Informationen über das verwendete Gerät MOSKITO TI.

Das Produkt MOSKITO TI der Firma Safran Vectronix AG ist ein optisches Gerät, welches die Funktionen eines Fernglases sowie eines Nachtsichtgerätes (durch Wärmebild und Restlichtverstärkung) in sich vereint. Im MOSKITO TI sind weitere Funktionen wie Distanzmessung, Kompass und Standortermittlung mit GPS enthalten.

Abbildung 1: Produktbilder des Herstellers Safran Vectronix AG
Quelle: https://safran-vectronix.com/product/moskito-ti-family/

Das Gerät bietet mit seinen 1.4 kg ein hohes Gewicht-Leistungs-Verhältnis. Es verfügt über externe Schnittstellenanbindungen wie Ethernet und/oder USB für den Datenaustausch sowie einen integrierten Laserdistanzmesser mit einer Reichweite von bis zu 10'000 m. Ein hoch präziser Magnetkompass ermöglicht eine exakte Messung des Höhenwinkels und des horizontalen Winkels. Die Nutzung eines Thermal Imaging Chip (deshalb MOSKITO «TI») ermöglicht eine Nutzung bei schwachem Licht und bei Dunkelheit. Mit Hilfe des Restlichtverstärkers kann auch bei sehr wenig Licht noch ein klares Bild erzeugt werden. Des Weiteren hat das Gerät ein eingebautes Display, das eine benutzerfreundliche Menüführung (siehe Abbildung 2) und Funktionsnutzung ermöglicht. (Safran Vectronix AG, 2020)

Abbildung 2: Menüführung im MOSKITO TI

Weitere Informationen über das Gerät können auf der Herstellerseite www.safran-vectronix.com im Produktflyer nachgelesen werden.

1.1 Ausgangslage

Das Produkt MOSKITO TI stellt seinem Nutzer diverse Funktionen zur Verfügung (siehe Einleitung, Seite 6). Auf dem Gerät fehlen jedoch Funktionen, um mit anderen Personen, welche ebenfalls ein MOSKITO TI nutzen, zu interagieren und Daten auszutauschen. In der vorliegenden Arbeit «Vernetzung von Beobachtungsstationen» wird die im Gerät bestehende Hardware genutzt. Das Gerät besitzt einen eingebauten Bluetooth/WLAN (Wireless Local Area Network) Chip sowie einen Spezialstecker, welcher auf einen RJ45 Anschluss (Ethernet) geht, um damit Daten auszutauschen. Das Gerät ist in der Lage, sich über WLAN oder über Ethernet mit einem Netzwerk zu verbinden. Die Software, welche auf dem MOSKITO TI läuft, basiert auf der Programmiersprache C++ und wurde im Qt-Framework erstellt. Im Softwarecode ist bereits eine REST (Representational State Transfer) -Schnittstelle für den Datenaustausch ausprogrammiert. Diese wird aktuell genutzt, um Daten über ein Endgerät (Laptop oder Mobiltelefon) auszulesen.

1.2 Zielsetzung

Die Zielsetzung wurde im Laufe des vorangegangenen Fachmoduls ausgearbeitet. Zusammengefasst können die Ziele folgendermassen definiert werden: Die Firma Safran Vectronix AG möchte eine Möglichkeit haben, um in einem lokalen Netzwerk (WLAN/LAN) Daten zwischen verschiedenen Geräten des Typs MOSKITO TI auszutauschen. Wenn möglich soll das Softwaredesign so ausgelegt werden, dass in einem nächsten Schritt weitere Gerätetypen hinzugefügt werden können. Der Fokus der Arbeit liegt auf dem Aufbau eines Overlay-Netzwerkes, welches idealerweise auf verschiedenen physikalischen Netzwerken aufbauen kann (WLAN/LAN). Das erstellte Overlay-Netzwerk soll durch Funktionen wie Bildaustausch, Punktübertragungen und passive Distanzmessungen genutzt werden. Sicherheitsaspekte wie Verschlüsselung, sichere Übertragung und Integritätsschutz können aus Zeitgründen vernachlässigt oder weggelassen werden. Die Softwareerweiterung soll wie die bestehende Software ebenfalls auf der Programmiersprache C++ sowie dem Qt-Framework basieren. Die originale Aufgabenstellung kann auf der nachfolgenden Seite gelesen werden.

1.3 Fachmodul

Im Fachmodul wurden verschiedene Übertragungsarten wie Bluetooth und WLAN miteinander verglichen. Des Weiteren wurden Übertragungsprotokolle wie TCP (Transmission Control Protocol), UDP (User Datagram Protocol) und HTTP (Hypertext Transfer Protocol) genauer betrachtet. Im Fachmodul wurde so eine gewisse technische Grundlage geschaffen. Für den Leser dieser Arbeit ist es nicht notwendig, das Fachmodul vorher gelesen zu haben. Sämtliche relevante Informationen werden dem Leser direkt in der vorliegenden Arbeit zur Verfügung gestellt. Um Fachbegriffe im Zusammenhang mit der Netzwerktechnik zu verstehen, wird empfohlen, das Buch «Computernetzwerke (Andrew S. Tannenbaum)» als Nachschlagewerk zu verwenden.

1.4 Aufgabenstellung für die Bachelorthesis

Offizieller Auszug der Aufgabenstellung

Thema: Vernetzung von mobilen Beobachtungsstationen über ein LAN/WLAN Netzwerk

Hintergrund: Durch die voranschreitende Vernetzung der Welt, gerade auch im IoT Umfeld, sollten die Beobachtungsstationen für die Zukunft aufgerüstet werden. Die grundlegenden Ressourcen bringen die Beobachtungsstationen bereits mit. Dies sind die Interfaces für LAN, WLAN und Bluetooth.

Aufgabenstellung: Die Beobachtungsstationen sollen sich innerhalb eines bestehenden Netzwerkes selbständig finden können. Dabei soll ein allgemeines Protokoll verwendet werden, welches ein übergelagertes Netzwerk generiert.

Für die Kommunikation zwischen den Geräten sollte ein generischer Ansatz gewählt werden, um mögliche zukünftige Szenarien abzudecken. So können in zukünftigen Projekten verschiedene Geräte dem Netzwerk hinzugefügt werden, welche unterschiedliche Daten und Funktionalitäten zur Verfügung stellen.

Um die Vernetzung nutzen zu können sollen neue Usecases ausgearbeitet werden, welche die zuvor beschriebene Kommunikationsmethode verwendet.

Ziel

Netzwerkebene: Auf Netzwerkebene soll ein austauschbarer Layer selbständig nach Geräten im bestehenden Netzwerk suchen und die gefundenen Geräte in eine Liste eintragen. Dabei soll die Liste Geräte, welche nicht mehr im Netzwerk existieren, selbständig entfernen. Dies soll innerhalb eines bestimmten Zeitintervalls geschehen. Der Layer soll selbständig funktionieren und nur wenig Abhängigkeiten zu dem bestehenden Projekt haben, sodass ein Austausch jederzeit möglich ist.

Kommunikation: Die Kommunikation soll über ein Applikation Programming Interface (API) implementiert werden. Dabei soll jedes Gerät ein solches API zur Verfügung stellen, so dass über einen entsprechenden Client darauf zugegriffen werden kann. Die API soll auf einer REST-Schnittstelle aufbauen und generisch sein. Dies wird durch einen End Point erreicht, welcher die Daten sowie die jeweiligen Funktionen, welche das Gerät bereitstellt, publiziert.

Use Case 1: Direkte Übertragung eines aufgenommenen Bildes zu einem anderen Gerät, oder das Anfordern eines bereits existierenden Bildes auf einem anderen Gerät.

Use Case 2: Ist die gleiche Funktion wie beim ersten Use Case, jedoch mit einem Point of Interest (POI).

Use Case 3: Ist eine passive Distanzmessung mittels Triangulation. Dabei müssen zwei Fälle unterschieden werden: GPS Signal vorhanden und GPS Signal nicht vorhanden. Die zwei Fälle unterschieden sich hauptsächlich darin, dass ohne GPS die Position des zweiten Gerätes zuerst aktiv bestimmt werden muss.

2 Das Overlay-Netzwerk

Das Kernstück und die Basis der Arbeit bestehen in einem sich selbst verwaltenden Overlay-Netzwerk. Ein Overlay-Netzwerk ist ein eigenes, zweites Netzwerk, welches über ein bestehendes Netzwerk gelegt wird. Das Generieren und Erstellen des physischen Netzwerkes ist nicht Teil dieser Arbeit, es muss aber für das Erstellen des Overlay-Netzwerkes vorhanden sein. Während der Implementierung und Ausarbeitung der Bachelorarbeit wurde nur mit einem physischen Netz über Ethernet (LAN) gearbeitet. Dies hatte den Vorteil, dass mit einer statischen IP (Internet Protocol) -Adresse gearbeitet werden konnte. Diese vereinfachte den Zugriff auf die Geräte und deren Nutzung über die REST-Schnittstelle. Gegen Ende der Programmierung wurden die Generierung des Overlay-Netzwerkes sowie die Nutzung sämtlicher Funktionen über ein WLAN-Netzwerk getestet. Der finale Test beinhaltete acht Geräte, davon waren sechs via WLAN und zwei via Ethernet mit dem Netzwerk verbunden (mehr zum Test in Kapitel 3.3).

2.1 Visualisierung

Um die Struktur des Netzwerkes sowie des darüber liegenden Overlay-Netzwerkes besser zu verstehen, wird es nachfolgend an einem Beispiel mit vier MOSKITO TIs und einem Laptop beschrieben. In Abbildung 3 wird der Aufbau eines physischen Netzwerkes über einen WLAN-Router visualisiert.

2.1.1 Das physische Netz (LAN/WLAN)

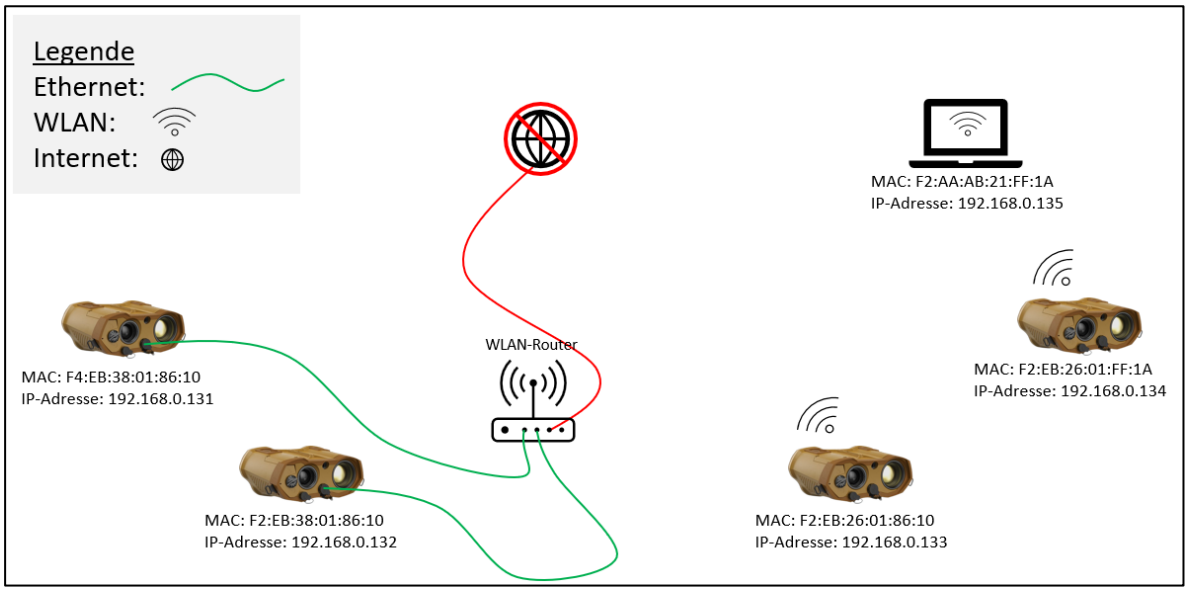


Abbildung 3: Netzwerk mit verschiedenen Geräten, verbunden über Ethernet und WLAN

Ein Router (in der Mitte von Abbildung 3) erkennt sämtliche angeschlossenen Geräte anhand ihrer einzigartigen MAC (Media Access Control) -Adresse. Alle Geräte, die am Router angeschlossen sind, erhalten von diesem eine im lokalen Netzwerk einzigartige IPv4-Adresse. Über den Router können sämtliche Geräte zusätzlich mit dem Internet verbunden werden. Dies könnte für die Vernetzung in Zukunft spannende Anwendungen mit sich bringen, wird jedoch in dieser Arbeit nicht genutzt. Die Verbindung zum Router kann entweder durch ein Ethernet Kabel oder durch die Nutzung des WLAN-Chips im MOSKITO TI erreicht werden. Im Gegensatz zum Ethernet mode, in welchem dem Gerät

in den Geräteeinstellungen eine statische IP-Adresse zugewiesen werden kann, ist es mit der integrierten WLAN-Funktion nur möglich, die IP-Adresse via DHCP (Dynamic Host Configuration Protocol) vom Router zu erhalten. Da eine statische IP-Adresse den Umgang mit den Geräten stark vereinfacht, wurde der Router so konfiguriert, dass die Adresse, welche die verwendeten Geräte erhalten, aufgrund ihrer MAC-Adresse statisch zugewiesen wurde. Es besteht die Möglichkeit, Geräte auch ohne Router zu verbinden. Dabei wird ein MOSKITO TI als AP (Access Point) konfiguriert und die weiteren Geräte holen sich die IP-Adressen bei diesem Gerät. Das Gerät übernimmt sozusagen die Funktion des Routers. Dies wurde während der Bachelorarbeit nur zu Testzwecken angewendet, da ein richtiger Router mehr Funktionen bietet und deshalb bevorzugt verwendet wurde.

2.1.2 Das Overlay-Netzwerk

Das Overlay-Netzwerk baut auf Abbildung 3 auf und wird mit einem neuen, darübergelegten Netzwerk erweitert. Wenn auf einem MOSKITO TI die Funktion ConnectionTi aktiviert ist (siehe Abbildung 4), hört es automatisch auf andere MOSKITO TIs im selben physischen Netzwerk und spannt so ein zweites Netzwerk auf (siehe rote Linien in Abbildung 5).

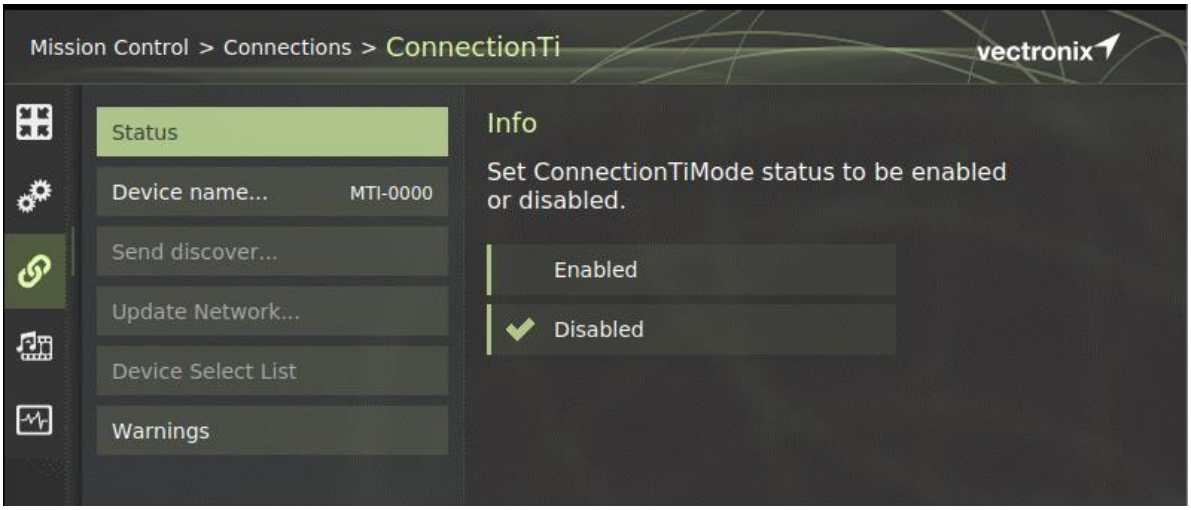


Abbildung 4: Menüführung MOSKITO TI (Einschalten ConnectionTi)

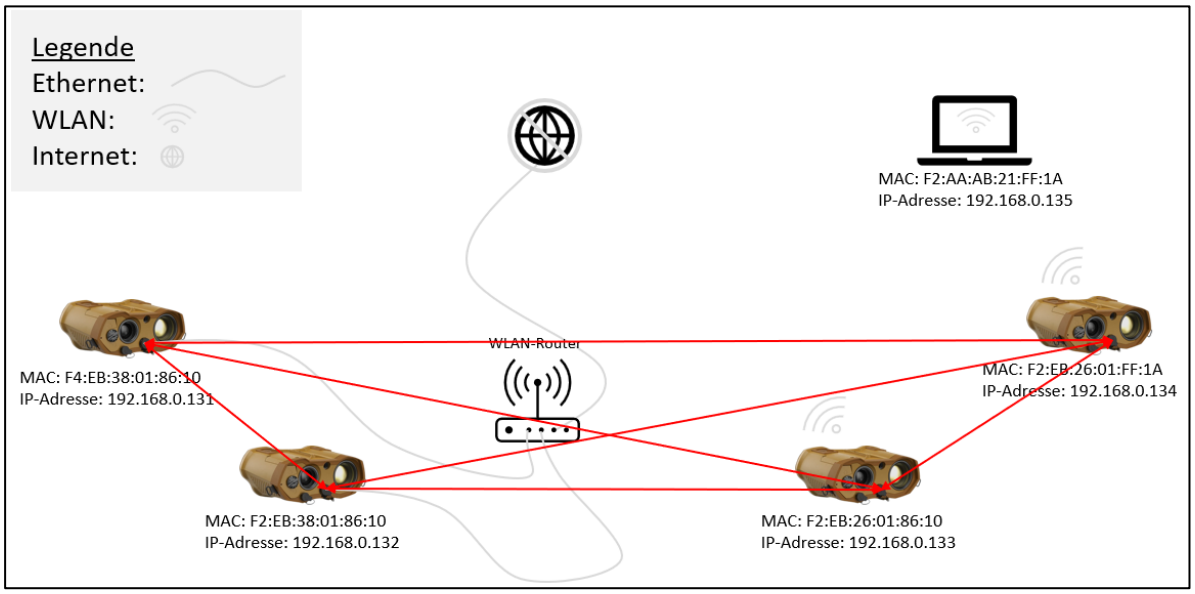


Abbildung 5: Overlay-Netzwerk auf bestehendem, physischem Netzwerk

2.2 Funktionsweise

Es gibt verschiedene Möglichkeiten, ein Overlay-Netzwerk zu generieren und es anschliessend zu verwalten. Eine Möglichkeit wäre das Master-Slave Prinzip. Ein Gerät wird zum Master ernannt und ist anschliessend für den Aufbau und den Unterhalt des Netzwerkes zuständig. Bei diesem System besteht die Schwierigkeit, dass es immer nur einen Master gibt. Wenn dieser unerwartet aus dem Netzwerk scheidet (durch Signalverlust, Kabelbruch oder plötzliches Abschalten des Gerätes), kann das gesamte Netzwerk zusammenbrechen. Ein weiterer Faktor ist die Fehlerproblematik. Ein Master-Slave Prinzip muss sehr gut und detailliert ausgearbeitet werden, sodass sämtliche Fehlerquellen eliminiert werden. Beispielsweise was geschieht, wenn zwei Geräte die Masterfunktion aktiviert haben oder wenn aus unvorhergesehenen Gründen plötzlich zwei Master zur Verfügung stehen. Aus diesen Gründen wurde eine andere Struktur für die Verwaltung des Overlay-Netzwerkes gewählt.

Das Overlay-Netzwerk in dieser Arbeit ist einfacher gehalten. Geräte legen eine Liste an, in welcher sie andere Geräte abspeichern. Durch regelmässige UDP Broadcastmessages der Geräte ins physische Netzwerk können die Listen auf den Geräten aktualisiert werden. Die genaue Beschreibung der verschiedenen Paketvarianten wird in Kapitel 2.3.4 beschrieben.

Wenn der ConnectionTi Modus auf einem Gerät eingeschaltet wird, legt es auf dem eigenen Stack eine neue, persönliche Geräteliste an. Des Weiteren sendet das Gerät eine UDP Broadcastmessage im gesamten physischen Netzwerk mit seinen eigenen Daten/Informationen. Andere MOSKITO TIs im physischen Netzwerk hören auf dem Broadcastkanal zu und erkennen, wenn ein neues Gerät eine Nachricht sendet. Sobald ein Gerät im Netzwerk die Broadcastmessage eines neuen Gerätes empfängt, speichert dieses den Namen sowie die IP-Adresse des neuen Gerätes in seine persönliche Liste. Weiter wird im empfangenden Gerät eine Funktion ausgelöst, welche über die REST-Schnittstelle Informationen über das neu hinzugefügte Gerät abfragt und ebenfalls abspeichert. Beim Gerät, welches neu im Netzwerk ist, wird, sobald ein neues Gerät in seiner persönlichen Liste erscheint, ebenfalls über die REST-Schnittstelle abgefragt, welche Services dieses zur Verfügung stellt.

Sobald das Overlay-Netzwerk steht, senden sämtliche Teilnehmer in regelmässigen Abständen eine UDP Broadcastmessage, um zu zeigen, dass sie immer noch anwesend sind. Die Nachrichten können als eine Art Herzschlag betrachtet werden. Mit jeder gesendeten Nachricht (mit jedem Schlag) erkennen die anderen Geräte, dass das Gerät, von dem die Nachricht stammt, noch am Leben ist, beziehungsweise sich noch im physischen Netzwerk befindet. Die Broadcastmessages sind sehr klein. Gemäss Wireshark liegt die Grösse eines gesendeten Paketes bei einem Gerätenamen mit 10 Grossbuchstaben und der Verwendung von IPv4 bei rund 60 Bytes. Es gilt zu erwähnen, dass in diesem Beispiel nur UTF-8 (8-Bit UCS Transformation Format, UCS – Universal Coded Character Set) kodierte Buchstaben verwendet wurden, die im deutschen/englischen Sprachraum verwendet werden. Bei der Verwendung von Sonderzeichen oder anderen Alphabeten, wie zum Beispiel dem kyrillischen Alphabet, kann die Paketgrösse zunehmen. Die Verwendung eines längeren Namens könnte die Paketgrösse ebenfalls ansteigen lassen. Mit der gemessenen Paketgrösse wird der Traffic im Netzwerk durch die UDP Nachrichten klein gehalten (siehe Abbildung 6). Die Grösse der Pakete ist so gering, dass sie in Bezug auf den gesamten möglichen Netzwerkverkehr vernachlässigt werden können.

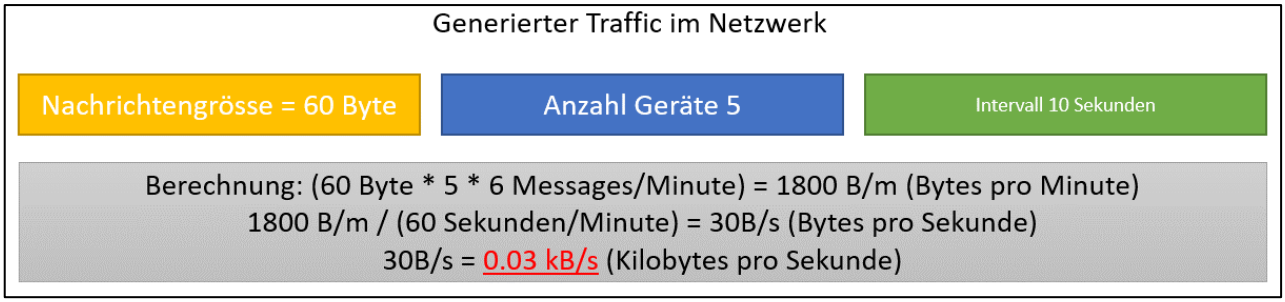


Abbildung 6: Durch UDP Herzschläge generierter Traffic im Netzwerk

Erhält ein Gerät eine etwas längere Zeit keine UDP Message (Herzschlag) mehr von einem Gerät, welches sich in seiner Liste befindet, wird das Gerät automatisch aus der Liste entfernt. Die Zeitintervalle können im Config-Headerfile (mehr dazu in Kapitel 2.3.1) angepasst werden.

Der Hauptvorteil des gewählten Overlay-Netzwerk Verfahrens gegenüber einem Master-Slave Verfahren liegt in der Einfachheit dieses Systems. Es gibt nur sehr wenige Fehlerquellen, was zu einer sehr hohen Zuverlässigkeit führt. Ebenfalls können Geräte beliebig hinzukommen oder gehen, ohne dass Komplikationen mit einem Master entstehen. Für den Anwender ist diese Art von Overlay-Netzwerk ebenfalls einfacher anzuwenden, da in den Einstellungen nicht festgelegt werden muss, ob das Gerät ein Master oder ein Slave ist. Trotz des gewählten Systems kann das Overlay-Netzwerk zusammenbrechen. Dies geschieht, sobald das darunterliegende physische Netzwerk Probleme hat. Ist das Netzwerk überlastet, zum Beispiel durch das Senden mehrerer Fotos, ist es möglich, dass UDP Pakete verloren gehen. Dies kann dazu führen, dass Geräte aus der Liste gelöscht werden, die eigentlich noch mit dem Netzwerk verbunden sind. Das entfernte Gerät würde jedoch mit dem nächsten empfangenen Herzschlag wieder in die Liste eingefügt werden.

Im Menü (Mission Control) des MOSKITO TI unter Connections > ConnectionTi gibt es verschiedene Menüpunkte (siehe Abbildung 7). Im obersten Menü «Status» kann die ConnectionTi Verbindung ein- und ausgeschaltet werden.

Unter «Device name…» kann der ConnectionTi Name, welcher bei anderen Geräten angezeigt wird, geändert werden.

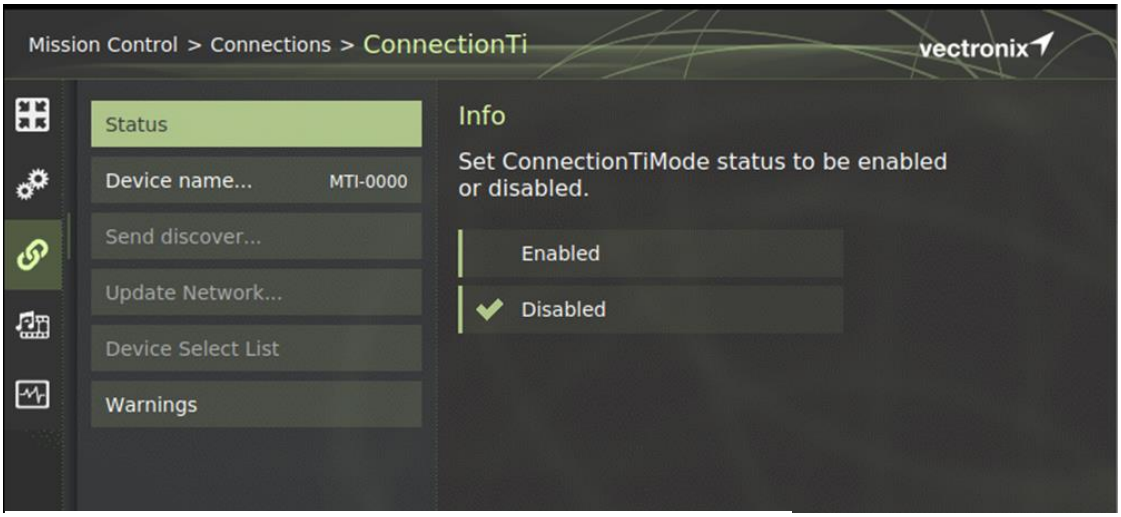


Abbildung 7: Connections > ConnectionTi

Wird unter «Status» die Funktion ConnectionTi eingeschaltet, werden die Felder «Send discover…», «Update Network…» und «Device Select List» selektierbar. Sie bieten dem Nutzer Funktionen, um das Overlay-Netzwerk zu generieren. Das Netzwerk wird grundsätzlich automatisch generiert, aber mit der Funktion «Send discover…» kann manuell nach Geräten im Netzwerk gesucht werden.

Mit der Funktion «Update Network…» kann der Nutzer bei einer Änderung in den Geräteeinstellungen nach allen Netzwerkverbindungen suchen. Dies wird benötigt, wenn zuerst die ConnectionTi Funktion eingeschaltet und erst im Nachhinein die Verbindung mit dem Netzwerk hergestellt wird.

Im Untermenü «Device Select List» werden alle Geräte angezeigt, welche mit dem Overlay-Netzwerk verbunden sind. Mit dem Element «Select all» können alle Geräte in der Liste an- oder abgewählt werden. Bei mehreren Geräten müssen so nicht alle Geräte einzeln angewählt werden. Wird auf ein Gerätename navigiert, so sieht der Nutzer dessen IP-Adresse, welche im Backend in der Liste eingetragen ist und zur Kommunikation verwendet wird (siehe Abbildung 8).

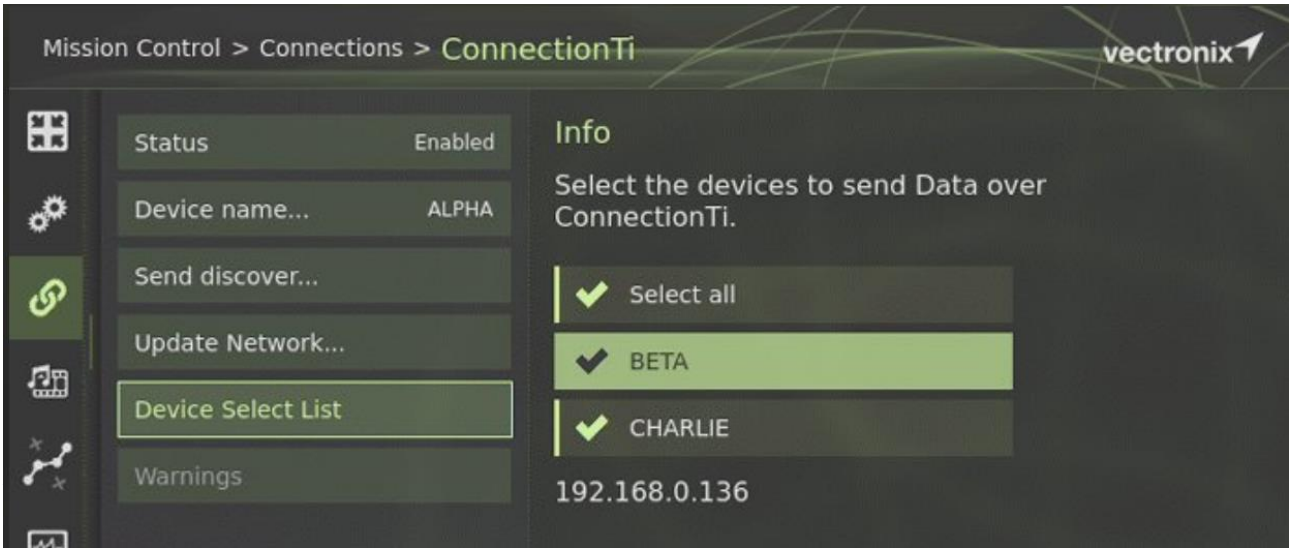


Abbildung 8: Device Select List mit angewähltem Gerät

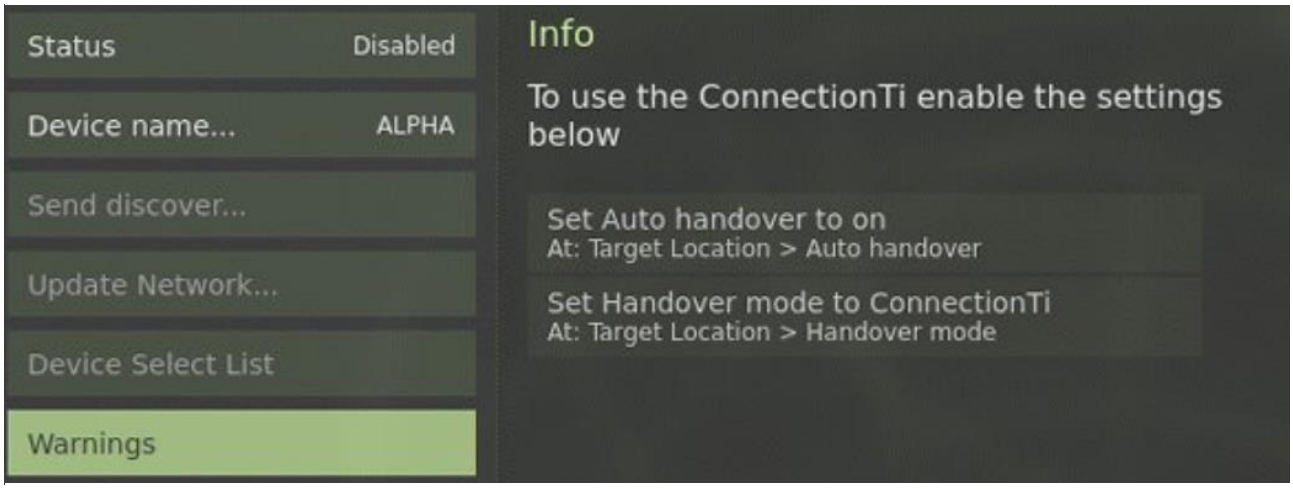


Abbildung 9: ConnectionTi > Warnings

Unter «Warnings» wird dem Nutzer angezeigt, wenn der «Handover mode» in den Einstellungen nicht auf «ConnectionTi» eingestellt ist (siehe Abbildung 9). Die «Auto handover» Funktion muss zudem auf «On» gesetzt sein. Diese beiden Punkte müssen eingestellt werden, um die verschiedenen Use Cases korrekt zu benutzen.

Die hier beschriebenen Funktionalitäten zeigen vor allem die Nutzerfunktionen zum Aufbau und Unterhalt des Overlay-Netzwerkes. Sie beschreiben das Overlay-Netzwerk jedoch nur oberflächlich. Um einen detaillierten Einblick zu erhalten, werden in den nachfolgenden Kapiteln die einzelnen Funktionen bzw. Klassen des Overlay-Netzwerkes beschrieben.

2.3 Die Klassen des Overlay-Netzwerkes

Aufgrund der verschiedenen Bereiche und Funktionen im Overlay-Netzwerk wurden verschiedene Klassen programmiert und im Softwarecode des MOSKITO TI eingebettet. Eine Übersicht wird im Ausschnitt des Klassendiagramms (siehe Abbildung 10) dargestellt. Das vollständige Diagramm ist im Anhang unter V.V abgelegt. Sämtliche Klassen sind mit der Programmiersprache C++ implementiert, zur Programmierung wurde QT Creator 5.9.8 verwendet.

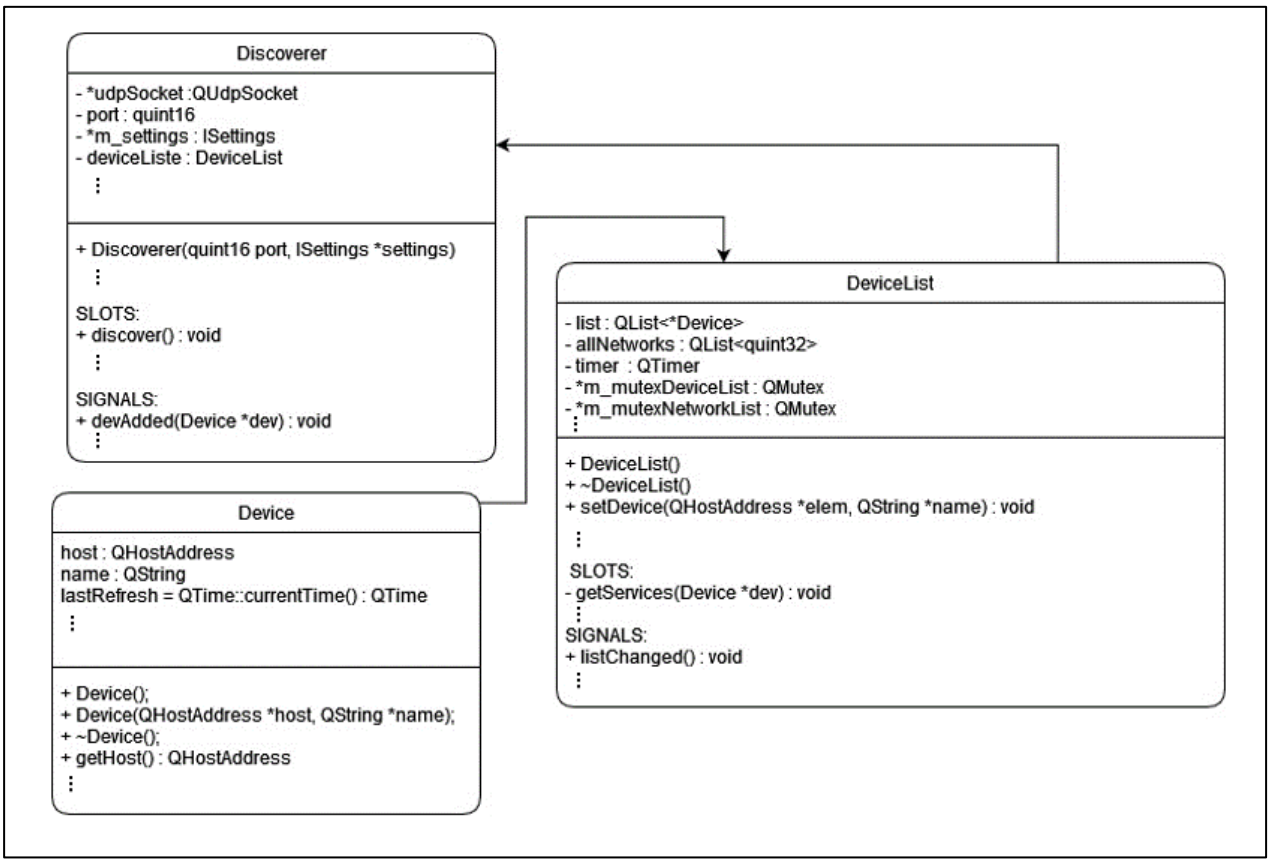


Abbildung 10: Ausschnitt des Klassendiagramms des Overlay-Netzwerkes

2.3.1 Config

Im Config Headerfile sind verschiedene Werte gespeichert, die zum Unterhalt und zur Aktualisierung des Overlay-Netzwerkes benötigt werden. Die Zeitintervalle für das Senden eines Herzschlags sind im Config festgelegt. Sie beträgt aktuell 10 Sekunden. Des Weiteren ist die maximale Namenlänge für das Gerät auf 20 Zeichen begrenzt.

Mit der Variablen «Delta device age» wird festgelegt, wie lange ein Gerät mindestens in der Liste erhalten bleibt, bevor es aufgrund von fehlenden Herzschlägen entfernt wird. «Check age device» ist das Zeitintervall, welches definiert, mit welchem zeitlichen Abstand durch die Liste iteriert wird und die virtuellen Geräte auf ihr Alter überprüft werden. Geräte, welche zu alt sind, werden aus der Liste entfernt.

Die Zeitwerte in der Config Datei müssen so gewählt werden, dass nicht unnötig oft Daten über das Netzwerk gesendet werden. Trotzdem müssen genügend Daten gesendet werden, damit die Liste auf

dem Laufenden bleibt und Geräte nicht zu lange in der Liste bleiben, nachdem sie aus dem physischen Netzwerk gefallen sind.

2.3.2 Device

Die Klasse «Device» spiegelt ein physisches Device wider. In ihr sind alle Informationen, welche für die Nutzung und Anwendung der Use Cases und für den Aufbau des Overlay-Netzwerkes benötigt werden, gespeichert. Dazu gehören folgende Punkte:

- Host Adresse
- Name des Devices (ConnectionTi Name)
- Information, ob das Device in der «Device Select List» selektiert ist oder nicht
- Einen Zeitstempel, wann ein virtuelles Device zuletzt von seinem physischen Device einen Herzschlag empfangen hat
- Den Root-Path für die REST-Schnittstelle
- Vier Listen mit den Diensten der REST-Schnittstelle (GET, POST, DELETE und PUT)

Des Weiteren sind folgende Funktionalitäten in der Klasse Device vorhanden:

- Die Informationsbeschaffung für das Füllen der vier Listen über die REST-Schnittstelle. Dies wird mit separaten Threads erreicht. Durch die Verwendung von unterschiedlichen Locks (Read und Write) können die Threads gleichzeitig lesen, aber nur ein Thread kann jeweils schreiben. Das Ganze ist somit Threadsave.
- Eine Funktion, um zu überprüfen, ob ein Path für ein bestimmtes Device vorhanden ist oder nicht. Dies wird genutzt, um zu vermeiden, dass Anfragen gemacht werden, welche durch den Fehlercode «404 Resource Not Found» beantwortet werden und somit unnötige Rechenleistung und Netzwerk Traffic erzeugen.
- Eine Funktion, welche direkt die IP-Adresse als String zurückgibt, indem sie die IP-Adresse aus der HostAddress extrahiert und anschliessend richtig formatiert.

2.3.3 Devicelist

Die Aufgabe der Klasse «Devicelist» ist die Verwaltung aller virtuellen Devices, welche im Netzwerk gefunden wurden. Die virtuellen Devices sind Abbildungen der physischen Devices. Die Klasse Devicelist ist nicht für das Finden der physischen Devices zuständig. Dies übernimmt die Klasse «Discoverer», welche nachfolgend beschrieben wird.

Die Devicelist enthält eine Liste von virtuellen Devices. Diese bestehen aus Objekten der Klasse «Device». Die Grösse der Liste ist abhängig von der Anzahl an gefundenen physischen Devices im Netzwerk. Die Hauptaufgabe ist das Verwalten der Listen. Dies wird durch folgende Funktionen erreicht:

Device hinzufügen:	Siehe Pseudocode im Anhang, Kapitel V.I.
Device entfernen:	Wenn ein physisches Device eine Nachricht sendet, dass es sich aus dem Netzwerk entfernt, wird dieses aus der Liste gelöscht. Dabei wird das Gerät über die IP-Adresse identifiziert und so aus der Liste entfernt.

Alterskontrolle: Die periodische Kontrolle des Alters der Devices sowie das Löschen aus der Geräteliste, wenn es zu alt ist. Die Parameter für den zeitlichen Abstand der Kontrollen und des maximal erlaubten Alters können mit dem «config.h» File bestimmt werden.

Namenswechsel: Ändert ein physisches Device den Gerätenamen, wird das virtuelle Device aufgefunden und der alte Name wird durch den neuen Namen ersetzt.

Services hinzufügen: Beim Hinzufügen eines neuen virtuellen Devices wird ein HTTP-GET an das physische Device mit dem HTTP-Pfad «…/api/info/services» durchgeführt. Der GET liefert einen JSON (JavaScript Object Notation) -String mit allen POST-, PUT-, DELETE- und GET-Services, welche das physische Device anbietet, zurück. Dieser JSON-String wird interpretiert und anschliessend in die jeweiligen Datenfelder des virtuellen Devices geschrieben. Diese Funktionalität ist in einen separaten Thread ausgelagert, um die Applikation nicht zu blockieren, wenn auf die Antwort der HTTP-Anfrage gewartet wird.

Die Klasse Devicelist kommuniziert mittels Signalen mit der Aussenwelt. Diese werden gesendet, wenn ein virtuelles Device hinzugefügt oder entfernt wird, ein neuer Name gesetzt wird, oder es eine Änderung an der Liste gibt. Bei den drei Signalen «virtuelles Device hinzugefügt», «virtuelles Device entfernt» oder «Name gewechselt» wird das betroffene Device als Parameter übergeben. Bei einer Änderung der Liste wird kein Parameter mitgegeben.

2.3.4 Discoverer

Die Hauptaufgabe der Klasse «Discoverer» ist das Versenden, Empfangen, Erstellen und Interpretieren der UDP Datenpakete. Des Weiteren beinhaltet die Klasse Funktionen, um auf die virtuellen Devices zuzugreifen. Dies ist nötig, da die Klasse Devicelist nicht direkt zu erreichen ist.

Die Nutzlast (Payload) eines Paketes kann eine variable Grösse von 1-81Byte aufweisen. Sie besteht aus zwei Segmenten:

Segment 1: Beinhaltet den Beschrieb der Nachricht und ist ein Byte lang. Wird in Tabelle 1 beschrieben.

Segment 2: Beinhaltet den Namen des Devices, welches das Paket gesendet hat. Das Segment kann auch leer übermittelt werden, wenn kein Name unter dem Menüpunkt «Connection > ConnectionTi > Device name…» vorhanden ist. Der Name hat eine maximale Länge von 20 Zeichen und ist UTF-8 encodiert. Da bei UTF-8 ein Zeichen eine Länge von bis zu vier Bytes erreichen kann, ist die maximale Länge des Namens 80 Byte.

Folgende sechs Möglichkeiten werden im ersten Segment unterschieden:

Ziffer	Kurzbeschreibung	Beschreibung
0x00	Empty	Unbenützt, jedoch vorbereitet für leere Pakete
0x01	Normal	Wird verwendet für Pakete, welche in einem bestimmten Intervall immer wieder gesendet werden (Herzschlag). Wird benötigt, um den Zeitstempel in virtuellen Devices zu erneuern.
0x02	Discover	Löst eine sofortige Antwort aus mit dem Case 0x01. Wird verwendet, um ein Netzwerk nach anderen Geräten zu durchsuchen.
0x03	Name Change	Wird gesendet, wenn sich der Name des Devices in den Einstellungen geändert hat. Damit wird der Name im virtuellen Device aktualisiert.
0x04	Shutdown	Löscht das Device aus allen Listen. Wird versendet, wenn der Discoverer bzw. ConnectionTi mode deaktiviert wird.
0xff	Error	Unbenützt

Tabelle 1: Erstes Byte der Payload der UDP Broadcastnachrichten

Der Discoverer leitet alle Signale, welche die Devicelist aussenden kann, weiter. Zusätzlich stellt die Klasse zwei Funktionen zur Verfügung, um auf ein virtuelles Device zuzugreifen.

Um die lokale Broadcastadresse zu ermitteln, werden zuerst alle aktiven Netzwerk Interfaces abgefragt und aus jedem die Broadcastadresse geholt. Diese werden gefiltert, um den Lokalhost nicht mit Daten zu belasten.

Die Logik, wie ein UDP-Packet empfangen und verarbeitet wird, ist in zwei Pseudocodes unterteilt und im Anhang abgebildet. Der erste beinhaltet den Konstruktor und ist im Anhang unter Kapitel V.II zu finden. Der zweite Pseudocode mit der Funktion, wie ein Paket verarbeitet wird, wenn es ankommt, ist im Anhang unter Kapitel V.III zu finden.

2.3.5 WrapperDiscover

Die Klasse WrapperDiscover implementiert das Singleton-Pattern (Eilebrecht & Starke, 2018), um so einen globalen Zugriff auf den Discoverer zu erhalten. Der Discoverer wird im WrapperDiscover initialisiert und gelöscht, wenn die Funktion ConnectionTi aktiviert oder deaktiviert wird.

3 Anwendung

Die Bachelorarbeit «Vernetzung von Beobachtungsstationen» hat das Ziel, MOSKITO TIs untereinander zu verbinden, mit der Absicht, dass diese Verbindung künftig von Anwendern genutzt wird. Dazu wurden mit der Firma Safran Vectronix AG drei Use Cases besprochen, welche das Overlay-Netzwerk nutzen sollen. Sie dienen zur Veranschaulichung und sollen aufzeigen, was möglich ist und wie das Overlay-Netzwerk sinnvoll genutzt werden kann. Die drei Use Cases, im Folgetext auch als Anwenderfunktion bezeichnet, werden nachfolgend detailliert beschrieben.

Alle drei Anwenderfunktionen nutzen das HTTP Protokoll sowie eine REST-Schnittstelle zur Datenübertragung. Die REST-Schnittstelle war bereits im Softwarecode des MOSKITO TI implementiert, wurde aber im Laufe der Arbeit erweitert und angepasst.

3.1 Aufbau der API / REST-Schnittstelle

Die REST-Schnittstelle, auf welche die Funktionen dieser Bachelorarbeit zurückgreifen, wurde aus Übersichtsgründen auf einem neuen Branch «'Individuelle IPv4 Adresse'/api/» angelegt. Der PORT der REST-Schnittstelle liegt unverändert auf dem Standardport des HTTP Protokolls (PORT 80). Der neue Branch beinhaltet alle in Abbildung 11 gezeigten Nodes mit den in Tabelle 2 und Tabelle 3 beschriebenen Services.

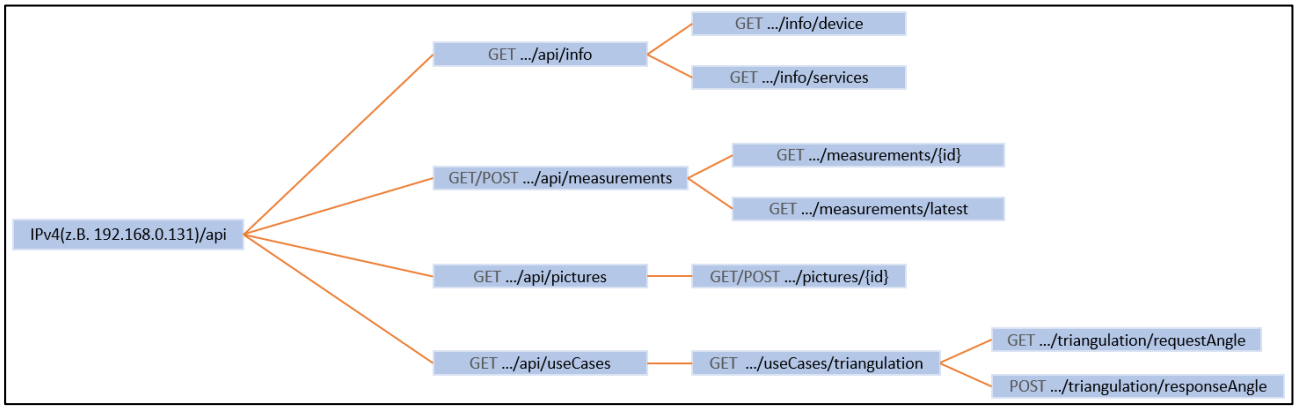


Abbildung 11: REST-Schnittstelle mit allen verfügbaren Nodes

Die REST-Schnittstelle wird über das HTTP Protokoll angesprochen. HTTP verwendet TCP Pakete, um seine Daten zu übermitteln. Die individuelle IPv4 Adresse, welche benötigt wird, um den Branch des jeweiligen Gerätes zu erreichen, wird auf jedem MOSKITO TI in der Geräteliste gespeichert. Dementsprechend weiss jedes Gerät, an welche IP-Adresse zu kommunizieren ist und kann so immer die richtige URL (Uniform Resource Locator) des jeweiligen Gerätes erreichen.

Der Node «…/api/info/device» sowie der Node «…/api/info/services» spielen für die Anwenderfunktionen eine zentrale Rolle. Das Overlay-Netzwerk soll generisch sein, so dass in einer späteren Phase durch den Industriepartner weitere Geräte eines anderen Typs hinzugefügt werden können. Mit dem ersten genannten Node kann abgefragt werden, um welchen Gerätetypen es sich handelt. Mit dem zweiten Node kann abgefragt werden, welche Services zur Verfügung stehen. Ein Beispiel der Services des MOSKITO TI ist in Abbildung 12 zu sehen.

3.1.1 REST-Schnittstelle GET-Services

Der implementierte Branch der REST-Schnittstelle bietet diverse GET-Services an. Mit GET-Services können Informationen über die REST-Schnittstelle abgerufen werden. Folgende GET-Services wurden für diese Arbeit implementiert:

Node	Funktion
…/info/device	Der Node /info/device gibt dem Nutzer im JSON-Format verschiedene Informationen über das Gerät zurück. Beispielsweise ob Bluetooth oder der ConnectionTi mode eingeschaltet ist.
…/info/services	Der Node /info/services gibt dem Nutzer sämtliche zur Verfügung stehenden Nodes zurück. (Siehe Abbildung 12)
…/measurements	Auf dem Node /measurements erhält der Nutzer ein JSON-File mit sämtlichen auf dem Gerät gespeicherten Messungen und deren Ids.
…/measurements/{id}	Auf dem Node /measurements/{id} kann eine Messung anhand ihrer Id aufgerufen werden. Diese wird ebenfalls im JSON Format zurückgegeben und enthält Informationen wie beispielsweise Standort des Gerätes, Distanz, Azimut, Uhrzeit etc.
…/measurements/latest	Auf dem Node /measurements/latest wird die letzte auf dem Gerät gespeicherte Messung im gleichen Format wie bei /measurements/{id} ausgegeben.
…/pictures	Der Node /pictures liefert im JSON Format eine Liste aller auf dem Gerät gespeicherten Fotos zurück. Zu jedem Foto gibt es eine Id.
…/pictures/{id}	Mit /pictures/{id} können einzelne Fotos aus dem Gerät ausgelesen werden.
…/useCases	Der Node /useCases dient als Dummy-Node. Er liefert selbst nur einen String zurück. In einem späteren Schritt könnten alle Use Cases zurückgegeben werden.

…/useCases/triangulation/requestangle	Der Node /useCases/triangulation/requestangle ruft die Userinterface-Interaktion auf und gibt anschliessend die durch die Messung erhaltenen Daten zurück.

Tabelle 2: REST-Schnittstelle GET-Services


```
{
    "delete": [
    ],
    "get": [
        "measurments",
        "measurments/{id}",
        "measurments/latest",
        "pictures",
        "pictures/{id}",
        "useCases",
        "useCases/triangulation/requestangle"
    ],
    "post": [
        "measurements",
        "pictures/{id}",
        "useCases/triangulation/responseangle"
    ],
    "put": [
    ],
    "root-path": "/api"
}
```


Abbildung 12: Node …/api/info/services

Abbildung 12 zeigt die Ausgabe nach einem GET auf den Node «api/info/services» und zeigt sämtliche Funktionen, welche auf dem Gerät zur Verfügung stehen. Diese Funktion wurde für die spätere Nutzung des Overlay-Netzwerkes erstellt. Sie wird benötigt sobald sich verschiedene Gerätetypen im selben Overlay-Netzwerk befinden. Obwohl in dieser Arbeit nur mit dem Gerät MOSKITO TI gearbeitet wurde, wurden die Services als Vorbereitung für später bereits abgefragt. Weitere Beispielausgaben der REST-Schnittstelle können im Anhang, Kapitel V.IV angeschaut werden.

Neben den GET-Services gibt es auch die sogenannten POST-Services, bei denen Daten über die REST-Schnittstelle auf das Gerät geladen werden können. Diese werden im folgenden Abschnitt genauer betrachtet.

3.1.2 REST-Schnittstelle POST-Services

Die POST-Services werden dazu benutzt, um Daten von einem Gerät auf ein anderes Gerät zu übermitteln.

Folgende Post-Services stehen zur Verfügung:

Node	Funktion
…/measurements	Mit dem Node /measurements kann eine Messung im JSON Format gepostet und so im Gerät gespeichert und nutzbar gemacht werden.
…/pictures/{id}	Der Node /pictures/{id} ermöglicht es, ein Foto zu posten, welches dann verarbeitet und in die Dateien gespeichert wird.
…/useCases/triangulation/responseangle	Mit dem Node /useCases/triangulation/responseangle kann dem Gerät die Distanz zum Ziel übermittelt werden.

Tabelle 3: REST-Schnittstelle Post-Services

3.2 Use Cases

Das in der vorliegenden Arbeit generierte Overlay-Netzwerk soll durch Use Cases, welche dem Nutzer des MOSKITO TI einen praktischen Nutzen bieten, erweitert werden. Dazu wurden im Fachmodul, welches der Bachelorarbeit vorausging, drei Use Cases bzw. Anwenderfunktionen finalisiert. Die Funktionen, welche während der Arbeit ausprogrammiert wurden, nutzen das Overlay-Netzwerk und die REST-Schnittstelle als Basis. Sie funktionieren über LAN und WLAN.

3.2.1 Bildübermittlung

Die Bildübermittlung ist die wichtigste Anwendung, welche in dieser Arbeit implementiert wurde. Sie bietet dem Nutzer den grössten funktionellen Nutzen. Ein konkretes Beispiel für einen Anwendungsfall könnte folgendes sein: Zwei Personen, ausgerüstet mit einem MOSKITO TI, beobachten in der Nacht ein Gebäude. Person 1 sieht das Gebäude von vorne, während Person 2 so positioniert ist, dass sie den seitlichen Teil und die Rückseite des Gebäudes sieht, jedoch nicht die Vorderseite. Beide Personen sind über Sprachfunk miteinander verbunden. Sieht nun Person 1 oder Person 2 etwas, das für die andere Person relevant ist, kann mit Hilfe des ConnectionTi mode ein Foto davon aufgenommen und mit der anderen Person geteilt werden.

Bereits vor dieser Arbeit verfügte das MOSKITO TI über die Funktion, Bilder aufzunehmen. Im Rahmen der vorliegenden Arbeit wurde eine neue Funktion implementiert, mit welcher die Bilder an andere Geräte übermittelt werden können.

3.2.1.1 ANWENDUNG DER FUNKTION

Um die Funktion der Bildübermittlung nutzen zu können, müssen folgende Punkte erfüllt werden:

Zu Beginn muss der ConnectionTi mode eingeschaltet werden (siehe Abbildung 4). Wenn dies gemacht wurde, können im Menüpunkt unter Connections > ConnectionTi > Device Select List (siehe Abbildung 8) alle im Overlay-Netzwerk verfügbaren Geräte an- oder abgewählt werden. Anschliessend muss im Mission Control (Menü) unter «Target Location» der «Handover mode» «ConnectionTi» ausgewählt werden sowie die «Auto handover» Funktion auf «On» gesetzt werden (siehe Abbildung 13). Der ConnectionTi mode funktioniert nur dann, wenn das Gerät mit einem lokalen Netzwerk verbunden ist, in welchem sich andere MOSKITO TIs befinden. Sollte dies nicht der Fall sein, werden unter ConnectionTi > Device Select List keine Geräte angezeigt.

Abbildung 13: Mission Control Handover mode

Wenn der Handover mode ConnectionTi und die Auto handover Funktion eingeschaltet sind, kann im Hauptbildschirm (im Wärmebild Modus oder im Low-Light Modus) die Image Recording Funktion aufgerufen werden. Dazu wird mit den Tasten auf dem Gerät nach links gedrückt und es erscheint in der Fusszeile ein Menü (siehe Abbildung 14). Das Menü und die Funktionen würden auch erscheinen, wenn der ConnectionTi mode ausgeschaltet ist, jedoch würde dann kein Foto übermittelt werden.

Abbildung 14: Fusszeilenmenü, welches erscheint, nachdem der Nutzer nach links gedrückt hat

In diesem Menü kann nun das Feld «Image Recording» ausgewählt werden. Wird mit der «OK» Taste bestätigt, wird der Nutzer in ein weiteres Menü geführt (siehe Abbildung 15).

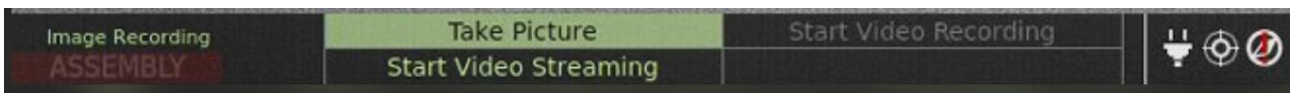


Abbildung 15: Image Recording Menü

In diesem Menü muss nun das Feld «Take Picture» ausgewählt und mit «OK» bestätigt werden. Wenn der Nutzer «OK» drückt, wird automatisch ein Foto aufgenommen und dieses in die eigene Mediengalerie gespeichert und an jene Geräte gesendet, welche in der «Device Select List» selektiert wurden. Anschliessend erscheint auf dem Bildschirm die Meldung, dass ein Foto aufgenommen wurde (siehe Abbildung 16).

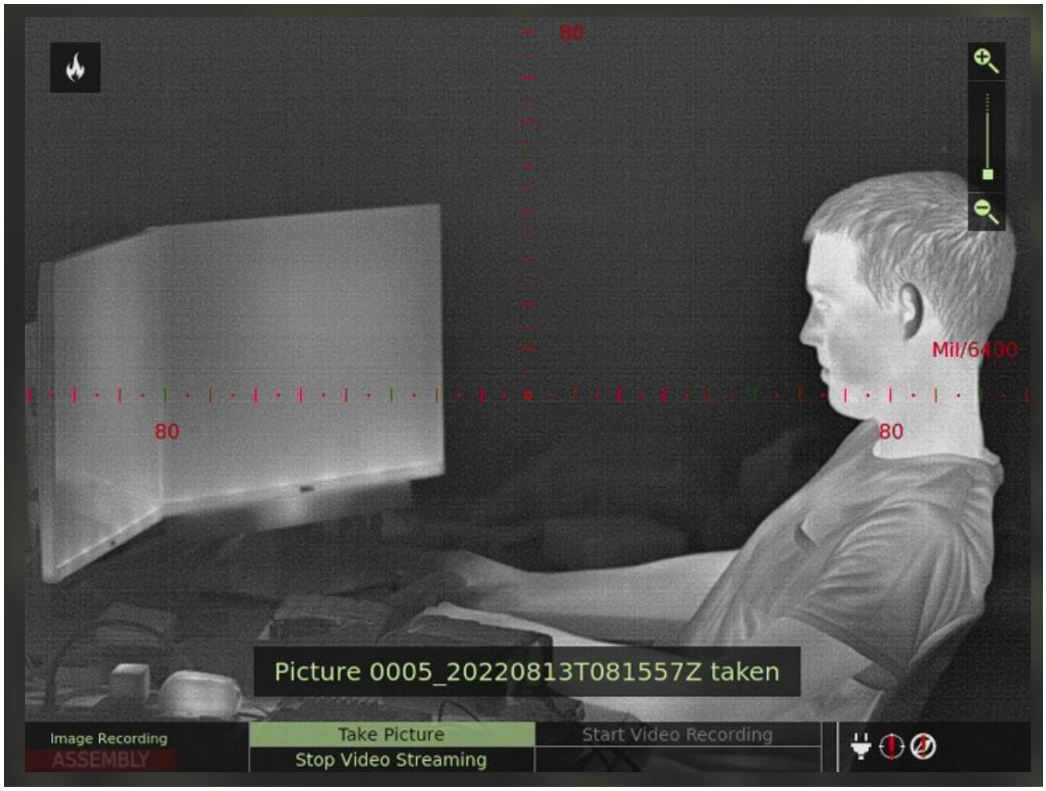


Abbildung 16: Bestätigung, dass ein Foto aufgenommen wurde

Im Mission Control unter dem Register Multimedia > Browse… ist nun auf allen Geräten, die zum Zeitpunkt der Aufnahme unter ConnectionTi > Device Select List selektiert waren, das aufgenommene Foto zu sehen (siehe Abbildung 17).

Abbildung 17: Mission Control > Multimedia > Browse… Menü

3.2.1.2 FUNKTIONSBESCHRIEB

Um die Funktionalität der Bildübertragung zu ermöglichen, wurden diverse Klassen des bestehenden MOSKITO TI Codes angepasst und erweitert. Die Funktion der Bildübertragung wird im P13MenuIController > m_takePictureMenuItem aufgerufen. Darin liegt ein Connect, welcher beim Auslösen einen multimediaSavePhotoRequested aufruft. Dieser überprüft, ob genügend Speicherplatz für ein Foto zur Verfügung steht und ruft anschliessend die Funktion takePicture auf. In der Funktion takePicture wird am Ende die Funktion handlePictureTakingFinished aufgerufen. Es wird nicht näher auf die beschriebenen Funktionen eingegangen, da dies nicht relevant für diese Arbeit ist und die Funktionen bereits vorhanden waren.

Relevant für die vorliegende Arbeit ist, dass zu einem bestimmten Zeitpunkt die Funktion handlePictureSavingFinished aufgerufen wird. Wenn diese Funktion aufgerufen wird, wird die Verlinkung zwischen dem bestehenden und dem neu erstellten Code hergestellt.

Konkret wird in einer Instanz des Handover mode die Funktion sendFileWithHandoverCases aufgerufen und als Parameter wird der Filepath des eben gespeicherten Bildes mitgegeben. Wenn in den Einstellungen des Gerätes der Handover mode ConnectionTi aktiviert ist, wird das File im Hintergrund im «Read Only» Modus geöffnet, die Daten werden ausgelesen und anschliessend das File wieder geschlossen. Danach wird über die in den Geräten gespeicherte Liste der im Overlay-Netzwerk verbundenen Geräte iteriert. Ist das Gerät in der Liste selektiert und der Service ist auf dem selektierten Gerät verfügbar, wird ein neuer Thread geöffnet und der Sendevorgang wird gestartet. Befindet sich ein Gerät in der Liste, welches nicht selektiert ist, wird es ausgelassen. Das asynchrone Senden bringt den Vorteil, dass das Gerät weiterhin nutzbar ist und das graphische Userinterface nicht blockiert ist.

Das Senden erfolgt über die REST-Schnittstelle. Zuerst wird die URL-Adresse zusammengebaut, um den richtigen Node der REST-Schnittstelle zu erreichen. Anschliessend wird der Filename beziehungsweise die Id des zu sendenden Fotos ausgelesen und ebenfalls in den URL Path eingefügt. Mit Hilfe des QNetworkAccesManager und des QNetwokRequest wird auf die REST-Schnittstelle der empfangenden Geräte ein POST durchgeführt. Dieser löst im empfangenden Gerät die Funktion aus, sich das Foto mittels GET zu beschaffen. Zur Veranschaulichung wird in Abbildung 18 der Prozess visualisiert.

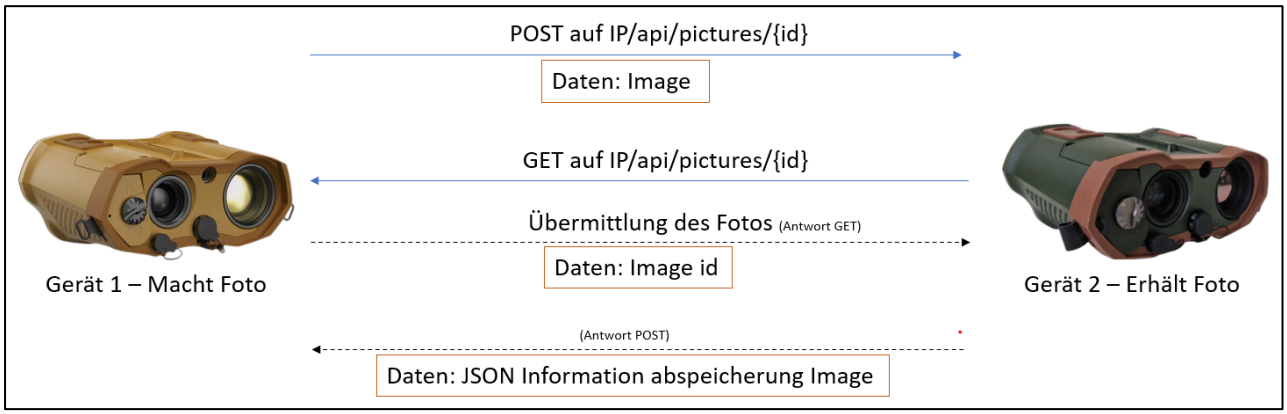


Abbildung 18: Übermittlung eines aufgenommenen Fotos

Die REST-Schnittstelle wurde ursprünglich so entwickelt, dass die Übermittlung eines Fotos direkt mit nur einem POST durchgeführt werden kann. Dies ist aber aktuell nicht möglich aufgrund der bestehenden Implementierung der REST-Schnittstelle. Werden grössere Daten über das HTTP Protokoll an die REST-Schnittstelle gepostet, wurden zwei verschiedene Verhaltensweisen beobachtet: Wenn die Daten kleiner als ~60kB sind, werden die ersten ~10kB wiederholt, bis die Dateigrösse erreicht ist. Wenn die Daten grösser als ~60kB sind, wird die Datei bei ~60kB abgeschnitten oder es werden gar keine Daten über den REST-Stack weitergegeben. Dieses Fehlverhalten wurde der Firma Safran Vectronix AG umgehend mitgeteilt. Leider war es aber nicht möglich, das Problem in nützlicher Frist zu beheben, weshalb diese Methode im Rahmen der vorliegenden Arbeit nicht weiter behandelt wurde. Da die Bildübermittlung aber nicht nur theoretisch, sondern auch praktisch funktionieren soll, wurde entschieden, einen Umweg über einen POST zu programmieren, der dann einen GET auslöst.

Das übermittelte Foto wird in der Mediengalerie des Gerätes gespeichert und kann dort angesehen werden. Wird mit der API auf die Multimedia Galerie zugegriffen, sieht der Anwender nur die Id des Fotos, nicht jedoch eine Vorschau (siehe Abbildung 19).

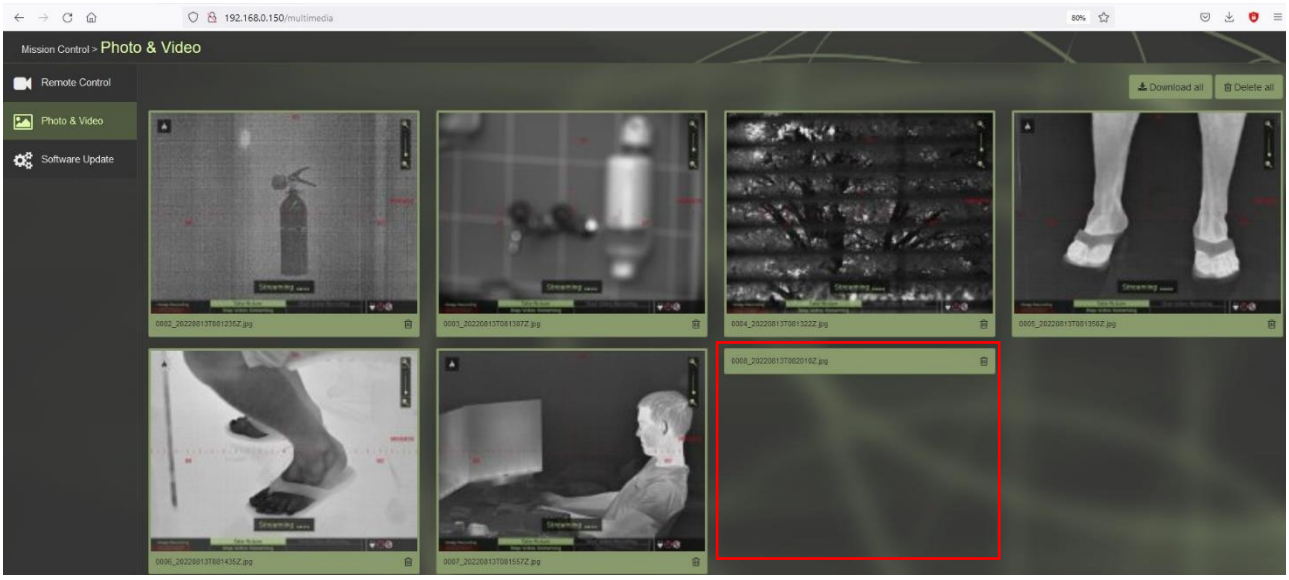


Abbildung 19: Mediengalerie via Browser direkt nach Erhalt eines Fotos

Wird die Galerie direkt im Gerät geöffnet, wird kein Unterschied zwischen einem selbst aufgenommenen und einem empfangenen Foto erkannt. Das Foto kann normal mit dem Browser und direkt auf dem Gerät geöffnet werden. Grund für die fehlende Vorschau ist das Nichtvorhandensein eines Thumbnails. Dieses wird erstellt, sobald im Gerät selbst die Mediengalerie geöffnet wird. Wird die Galerie geöffnet und anschliessend der Browser aktualisiert, sieht es folgendermassen aus:

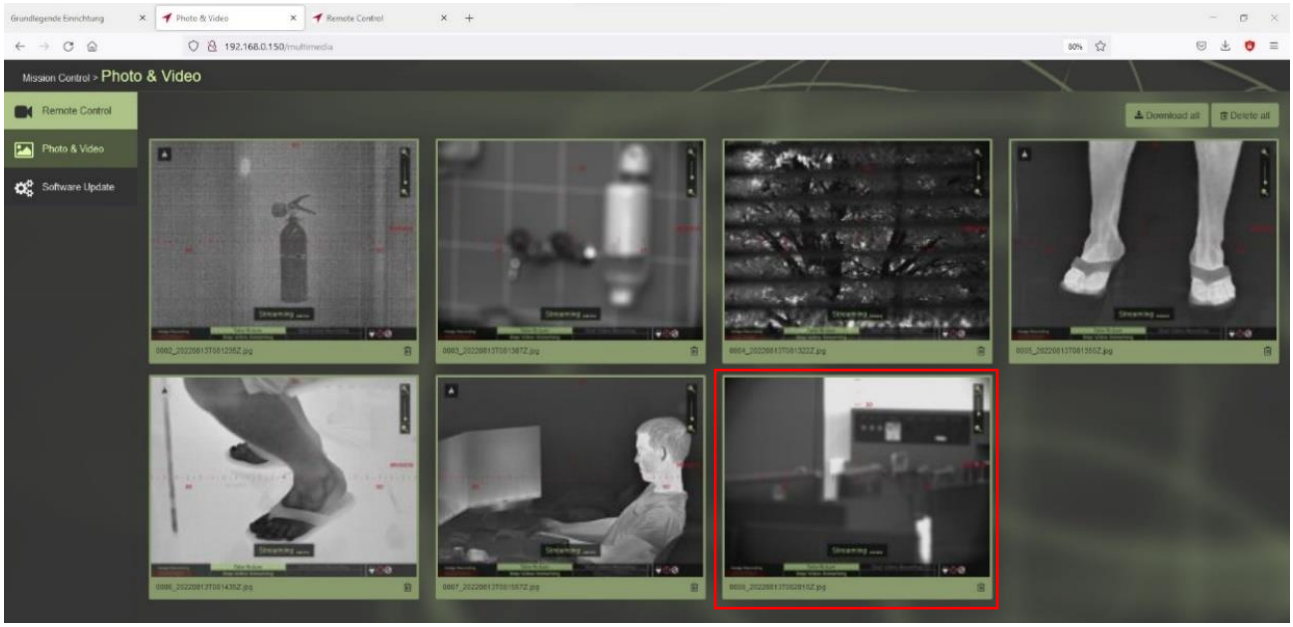


Abbildung 20: Mediengalerie nach Öffnen im Gerät und Aktualisieren des Browsers

Was in Abbildung 19 und Abbildung 20 nicht erkannt werden kann, ist, dass das neu in die Galerie eingefügte Foto nicht an der vordersten Stelle angezeigt wird. Dies liegt an der alphanumerischen Anzeigereihenfolge. Das Foto erhält die Id vom Aufnahmegerät. Wird es versendet, wird es dort eingeordnet, wo es im neuen Gerät von der Id aus hingehört.

Was in Abbildung 19 und Abbildung 20 auffällt ist die Tatsache, dass es nicht möglich ist zu erkennen, ob das Foto selbst aufgenommen wurde oder ob es von einem verbundenen MOSKITO TI gesendet wurde. Da jedoch davon ausgegangen wird, dass Sender und Empfänger über Funk miteinander verbunden sind, kann dieser Aspekt in der vorliegenden Arbeit vernachlässigt werden. Für die Zukunft wäre dies jedoch ein Punkt, der noch erweitert werden könnte.

3.2.2 Punktübermittlung

Die Punktübermittlung ist die zweite Anwenderfunktion, die implementiert wurde. Der Hauptnutzen der Punktübermittlung liegt im Austausch eines Point of Interests.

Wird auf einem MOSKITO TI eine Messung durchgeführt, werden die Daten im Arbeitsspeicher des Gerätes gespeichert. Dieser Punkt kann nur vom jeweiligen Gerät ausgelesen werden.

Sind nun mehrere MOSKITO TIs über das Overlay-Netzwerk miteinander verbunden, werden die Messungen mit allen Daten auf die anderen Geräte übertragen. Von diesen Geräten können die Messungen jederzeit über die REST-Schnittstelle abgefragt werden (Siehe Tabelle 2).

3.2.2.1 ANWENDUNG DER FUNKTION

Die Funktion der Punktübermittlung ist einfach anzuwenden. Wie bei der Übertragung von Fotos muss in den Einstellungen des MOSKITO TI der ConnectionTi mode eingeschaltet sein. Des Weiteren muss unter «Target Location» der Handover mode «ConnectionTi» ausgewählt sein und die «Auto handover» Funktion auf «On» gesetzt werden. Sind diese Bedingungen erfüllt, kann im optischen Modus sowie im Thermal Modus eine Messung durchgeführt werden. Die Messung kann auch ohne Einschalten des ConnectionTi mode oder Handover mode durchgeführt werden, jedoch wird sie dann am Schluss nicht übermittelt. Um die Messung zu starten, wird das MOSKITO TI auf das zu messende Ziel ausgerichtet und der Knopf «Messung» gedrückt (siehe Abbildung 21).

Abbildung 21: Knopf auf dem MOSKITO TI, um eine Messung zu starten

Sobald der Knopf gedrückt wird, erscheint am unteren Bildschirmrand ein «Measuring» Feld.

Abbildung 22: Messung wird durchgeführt

Sobald die Messung erfolgreich war, wird die Distanz, Azimut und Neigung (Inclination) angezeigt.

Abbildung 23: Messung wurde erfolgreich durchgeführt

Die Messung wird im RAM (Random Access Memory) des Gerätes gespeichert. Es kann vorkommen, dass die Distanz nicht ermittelt werden konnte, zum Beispiel aufgrund einer zu geringen oder zu grossen Distanz. Dies ändert jedoch nichts daran, dass die Messung gespeichert wird.

Sobald die Messung durchgeführt wurde, werden die Daten der Messung an alle in der ConnectionTi > Device Select List ausgewählten Geräte gesendet und können sofort über die REST-Schnittstelle ausgelesen werden.

3.2.2.2 FUNKTIONSBESCHRIEB

Die Übermittlung wird wie das Senden eines Fotos im DataHandoverService gesteuert. Eine übergeordnete Funktion «sendTargetingResult» wird aufgerufen, wenn eine Messung durchgeführt wird. In dieser Funktion gibt es einen Case, der aufgerufen wird, wenn der ConnectionTi mode aktiviert ist. Es wird eine Messung erstellt, Daten wie UUID (Universally Unique Identifier) werden der Messung angefügt und ein JSON-Document wird erstellt. Anschliessend wird die Messung mit Hilfe eines QNetworkAccessManager an die zuvor zusammengefügte REST-Schnittstellen-URL gesendet. Dabei wird wie beim Senden eines Fotos über die gesamte Liste der Geräte im Overlay-Netzwerk iteriert und sämtliche Geräte, welche selektiert sind, werden berücksichtigt.

Im Gegensatz zur Bildübermittlung muss hier nicht ein POST gesendet werden, welcher das empfangende Gerät dazu auffordert, sich die Informationen mit einem GET zu besorgen. Hier können die Daten direkt mit einem POST übermittelt werden. Der Grund, weshalb dies im vorliegenden Fall direkt funktioniert und bei der Bildübermittlung nicht, liegt an der Grösse der Daten. Diese sind beim Übermitteln einer Messung deutlich kleiner als beim Senden eines Fotos.

3.2.3 Passive Distanzmessung

Die passive Distanzmessung, welche mittels Triangulation erreicht wird, ist der dritte und letzte Use Case, der als Teil dieser Arbeit implementiert wurde. Er bietet die Möglichkeit, eine Distanz zu einem Ziel zu messen, ohne dass ein aktiver Laserstrahl ausgesendet werden muss. Die passive Messung bietet zwei Hauptvorteile gegenüber der herkömmlichen, aktiven Messung:

Wird ein Ziel (Gebäude, feindliche Stellung etc.) mit einer aktiven Messung angepeilt, könnte diese Messung im Ziel vom Feind detektiert werden und würde so die Anwesenheit des Anwenders enttarnen. Dies könnte zu strategischen Nachteilen führen. Mit einer passiven Distanzmessung wird das Ziel nicht mit einem aktiven Laserstrahl anvisiert, sondern das Gerät bestimmt die Distanz mit Hilfe des Kompasses sowie der Distanz und Position zu einem zweiten MOSKITO TI.

Der zweite Vorteil besteht darin, dass auch Objekte, welche mit einem Laserstrahl nicht erfasst werden können, detektierbar sind. Als Beispiel wird eine Boje auf dem Meer genommen: Die Boje bietet kaum genügend Platz, um eine aktive Messung durchzuführen. Das Ganze kann zusätzlich durch ein schaukelndes Boot erschwert werden. Sind nun zwei MOSKITO TIs miteinander verbunden, kann eine passive Distanzmessung durchgeführt und die Distanz so ermittelt werden.

3.2.3.1 ANWENDUNG DER FUNKTION

Die passive Distanzmessung unterscheidet zwei Anwendungsfälle:

Anwendungsfall 1: Das Gerät verfügt über GPS und ist mit mehreren Satelliten verbunden.

Anwendungsfall 2: Das Gerät hat kein GPS oder ist nicht mit Satelliten verbunden.

Der Nutzer kann selbst entscheiden, welcher Fall ausgewählt wird. Es ist also trotz GPS-Signal möglich, eine Messung ohne GPS durchzuführen. Je nachdem, welcher Fall gewählt wird, läuft die Messung etwas anders ab. Dies wird mit Hilfe eines Ablaufdiagramms im Anhang, Kapitel V.VI beschrieben.

Wenn im MOSKITO TI in den Einstellungen der ConnectionTi mode aktiviert ist, kann die passive Distanzmessung folgendermassen aufgerufen werden: Der Anwender befindet sich im Thermal-Modus oder im Low-Light-Modus und drückt mit den Tasten nach links. Am unteren Bildschirmrand öffnet sich ein Menü (siehe Abbildung 24).

Abbildung 24: Menü beim Drücken der «Links navigieren Taste»

Nun kann auf das Register «Triangulation» navigiert und mit der «OK» Taste der Menüpunkt geöffnet werden. Es erscheint ein neues Menü Fenster am unteren Bildschirmrand, auf welchem ausgewählt werden kann, ob Anwendungsfall 1 mit GPS oder Anwendungsfall 2 ohne GPS durchgeführt werden soll (siehe Abbildung 25).

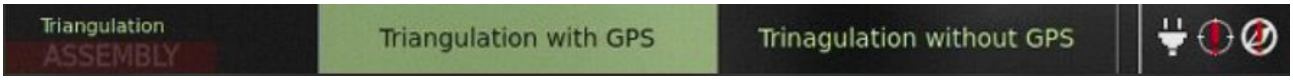


Abbildung 25: Auswahl zwischen «mit GPS» und «ohne GPS»

Aus zeitlichen Gründen und weil der Fokus in der Bachelorarbeit auf der Vernetzung der Geräte liegt, wurde die mathematische Berechnung der passiven Distanzmessung mittels GPS weggelassen. Der programmtechnische Ablauf sowie die Übermittlung der Daten wurde implementiert und getestet. Die Funktion mit GPS dient der Illustration und zeigt auf, was im weiteren Vorgehen erarbeitet werden könnte. Die Abfolge der Anweisungen ist gleich wie bei der Abfolge ohne GPS, mit der Ausnahme, dass die Distanz zum zweiten Gerät nicht ermittelt werden muss. Es entfällt die Anforderung zur Messung des zweiten Device (siehe Abbildung 29). Bei den aufgezeichneten Daten sind die Unterschiede grösser. Mit GPS werden zusätzlich zu den Winkeln, welche die Richtung des Ziels angeben, auch die beiden Winkel Longitude (Längengrad) und Latitude (Breitengrad) sowie die Höhe über Meer (in Meter) aufgezeichnet. Zusätzlich werden zur Qualitätsbestimmung des Signales noch die GPS-Qualitätsinfos aufgezeichnet.

Wird die Messung «ohne GPS» gewählt, werden am unteren Bildschirmrand Nachrichten angezeigt, welche dem Nutzer mitteilen, was zu tun ist. Als erstes wird der Nutzer dazu aufgefordert, das zu messende Ziel anzuvisieren und auf «OK» zu drücken (siehe Abbildung 26).

Abbildung 26: Gerät 1, Anweisung nach Klick auf «Triangulation without GPS»

Zeitgleich erhält das erste Gerät, welches im Backend in der Liste mit verbundenen Geräten steht, die Aufforderung, das Ziel anzuvisieren und auf «OK» zu drücken (siehe Abbildung 27).

Abbildung 27: Gerät 2, erhält Anweisung zum Messen des Ziels

Wenn das zweite Gerät die Messung getätigt hat, wird ein Schriftzug eingeblendet, welcher den Nutzer zum Warten auffordert. Diese Meldung verschwindet, sobald das erste Device mit den Messungen und Berechnungen fertig ist (siehe Abbildung 28).

Abbildung 28: Gerät 2, Warteanzeige

Anschliessend an die erste Messung des ersten Gerätes erscheint auf dem ersten Gerät die Aufforderung, das zweite Gerät, welches für die Messung verwendet wird, anzuvisieren und «OK» zu drücken (siehe Abbildung 29). Die Distanz zum Partnergerät wird aktiv ermittelt.

Abbildung 29: Gerät 1, Aufforderung Distanz zum zweiten Gerät zu messen

Wenn die Messungen auf dem ersten Gerät getätigt wurden, aber die Messung auf dem zweiten Gerät noch nicht erledigt ist, taucht eine Warteanzeige auf (siehe Abbildung 30). Diese verschwindet, sobald die Messung auf dem zweiten Device getätigt und übermittelt wurde.

Abbildung 30: Anzeige warten auf zweites Device

Nach Abschluss der Berechnung wird das Resultat «P2 → Ziel» mittels eines HTTP-POST an das zweite Device übermittelt und dieses sogleich angezeigt. Zeitgleich wird das Resultat «P1 → Ziel» für das erste Device ebenfalls visualisiert (siehe Abbildung 31).

Abbildung 31: Distanz zum gemessenen Ziel

Die Distanzangaben sind mit Vorsicht zu geniessen. Wie bei den Tests festgestellt wurde, gibt es bezüglich der Genauigkeit des Kompasses von Gerät zu Gerät einige Abweichungen. Die Messung basiert auf der Genauigkeit zweier Kompasse. Wenn ein Kompass ungenau ist oder sie voneinander abweichen, kann die Messung stark verfälscht werden.

3.2.3.2 FUNKTIONSBESCHRIEB

Der Funktionsbeschrieb im Hintergrund des Gerätes ist in verschiedene Teilbereiche aufgesplittet. Ein Bereich ist die Menü-Führung am unteren Bildschirmrand. Diesem Menü unterliegt ein grosser Zustandsautomat. Dieser regelt, wann welcher Kontroller eingesetzt werden soll und ob von einem Kontroller zu einem anderen gewechselt werden darf. Für diese Arbeit wurden weitere Zustände hinzugefügt sowie das Verhalten des Zustandsautomaten angepasst, um die Funktionalitäten für die Triangulation einzubauen. Für das Verhalten während der Triangulation wurde ein eigener Kontroller realisiert, welcher den Zustandsautomaten für den Ablauf der Triangulation enthält. Dieser Kontroller kommt zur Anwendung, wenn der Menüpunkt «Triangulation > withGPS» oder «Triangulation > withoutGPS» angewählt wird. Der Ablauf des Zustandsautomaten ist in der Abbildung 32 beschrieben.

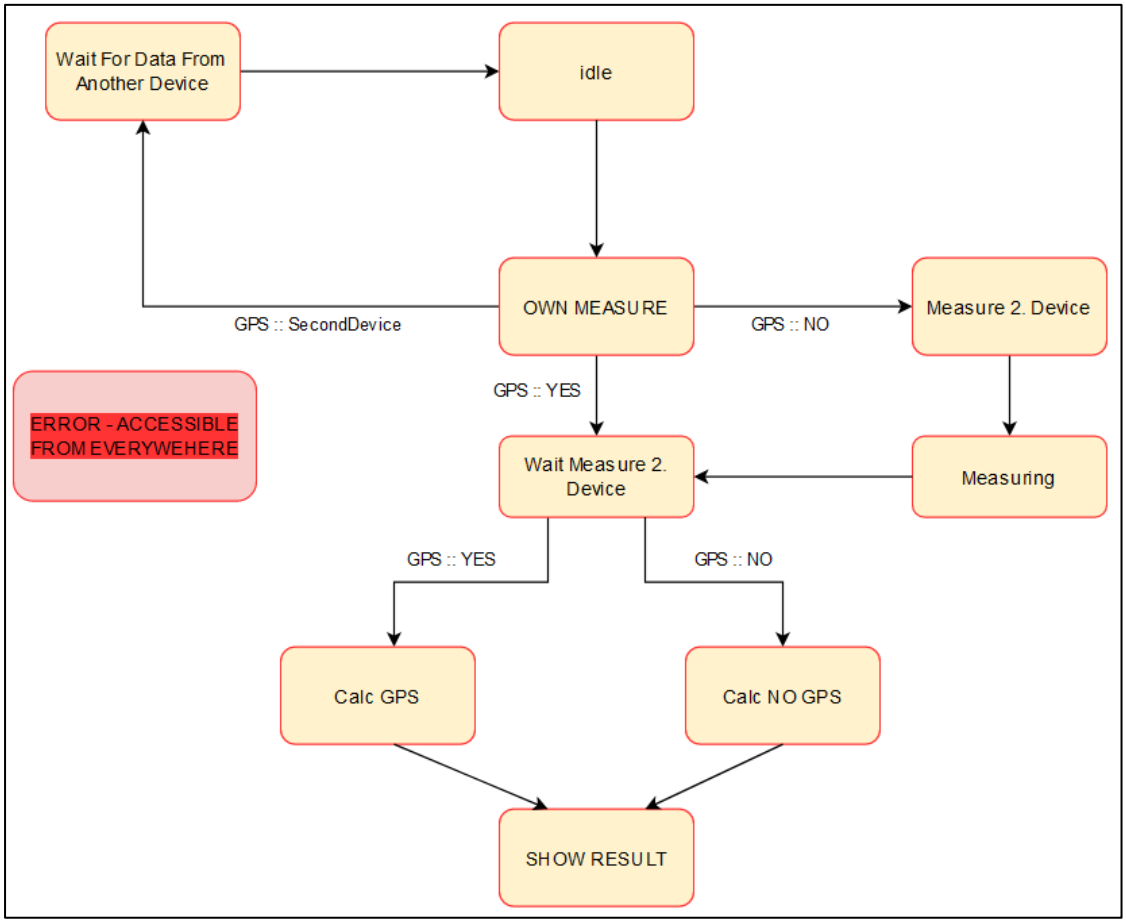


Abbildung 32: Zustandsautomat für die passive Distanzmessung

Es muss noch erwähnt werden, dass von jedem Zustand jederzeit in den Zustand «idle» oder «Error» gewechselt werden kann.

3.2.3.3 BERECHNUNG OHNE GPS

Als Ausgangslage sind von den folgenden drei Messungen Daten aufgezeichnet worden:

P1 → Ziel: Azimut und Elevation

P2 → Ziel: Azimut und Elevation

P1 → P2: Azimut, Elevation und Distanz

Der Azimut ist jeweils der horizontale Winkel, welcher von 0 bis 2π (Bogenmass) geht. Dieser beginnt bei null in Richtung des magnetischen Nordpoles und vergrössert sich im Uhrzeigersinn bis zu einer vollen Umdrehung (2π).

Die Elevation ist der Höhenwinkel und hat den Bereich von -π/2 bis π/2. Auf Höhe des Horizontes beträgt der Winkel 0 π. Das untere Limit ist der Nadir, das obere Limit ist der Zenit.

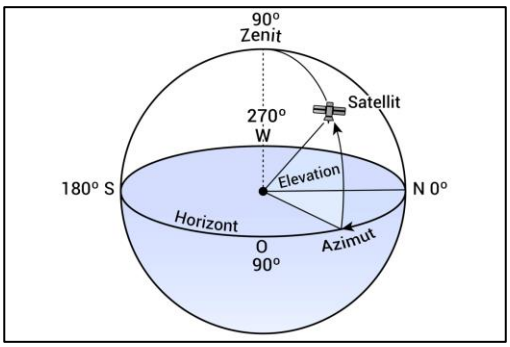


Abbildung 33: Darstellung Winkel
Quelle: www.bootspruefung.de/

Die Distanz (P1 → P2) wird mittels einer aktiven Lasermessung ermittelt und in Meter angegeben.

Als Erstes wird mittels der gemessenen Winkel ein Vektor berechnet, der in die Richtung zeigt, welche durch die drei gemessenen Winkel vorgeben wird. Dies wird mittels der folgenden Formel erreicht:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} distanz * \cos(Elevation) * \sin(Azimut) \\ distanz * \cos(Elevation) * \cos(Azimut) \\ distanz * \sin(Elevation) \end{pmatrix}$$

Wenn dies mit allen drei Messungen durchgeführt wurde, sind drei Richtungsvektoren bekannt. Dabei bilden zwei einen Einheitsvektor (P1 → Ziel, P2 → Ziel) ab und der dritte Richtungsvektor (P1 → P2) beinhaltet zusätzlich die Distanz zwischen den beiden Devices.

Um die Koordinaten zum Ziel zu ermitteln, muss der Schnittpunkt der beiden Geraden gefunden werden. Dies ist im 3D-Raum jedoch nicht so einfach (überbestimmtes Gleichungssystem), da sich die beiden Geraden nicht zwingend schneiden müssen. Mit der Erweiterung durch eine Gerade, welche immer rechtwinklig auf den beiden Hauptgeraden steht, wird ein bestimmtes Gleichungssystem erstellt. Mit dieser weiteren Geraden wird so der kürzeste Abstand zwischen den beiden Hauptgeraden bestimmt. In der Mitte dieser zusätzlichen Geraden befindet sich unser Ziel bzw. dieser Punkt kommt dem Ziel wahrscheinlich am nächsten.

Das Problem des komplexen Gleichungssystems wurde stark vereinfacht, indem die Berechnung auf zwei Dimensionen reduziert wurde, um ein nicht überbestimmtes Gleichungssystem zu erhalten. Das ist in diesem speziellen Fall möglich, da:

- Die Messungen der Winkel eine Toleranz besitzen.
- Die Winkelverzerrung, welche durch die Kugelform unseres Planeten entsteht, ebenfalls ignoriert wurde. Dieser Fehler wird grösser, je näher man sich den magnetischen Polen nähert.
- Davon ausgegangen wird, dass der Winkel Elevation keine grösseren Werte annimmt (> π/4 und <π/4)

Die Lösung des mathematischen Problems wurde folgendermassen erreicht:

Geg: $\overrightarrow{R1} := Vektor\ zwischen\ Device\ 1\ und\ dem\ Ziel$

$\overrightarrow{R2} := Vektor\ zwischen\ Device\ 2\ und\ dem\ Ziel$

$\overrightarrow{R3} := Vektor\ zwischen\ Device\ 1\ und\ Device\ 2$

Ges: Distanz zwischen P1 zum Ziel und P2 zum Ziel

Lsg:

1. Setze Device 1 in den Nullpunkt des Koordinatensystems: $\overrightarrow{D1} := \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$
2. Berechne die Position des Device 2: $\overrightarrow{D2} := \overrightarrow{D1} + \overrightarrow{R3}$
3. Die Position mittels der Vektoren für das Ziel ist von Device 1: $\overrightarrow{T1} = \overrightarrow{D1} + r * \overrightarrow{R1}$ und aus Sicht von Device 2: $\overrightarrow{T2} = \overrightarrow{D2} + s * \overrightarrow{R2}$
4. Daraus ergibt sich folgende Gleichung, indem die zwei vorherigen Gleichungen gleichgesetzt werden: $\overrightarrow{D1} + r * \overrightarrow{R1} = \overrightarrow{D2} + s * \overrightarrow{R2}$. Dies ist möglich, da im 2D-Raum gilt: $\overrightarrow{T1} = \overrightarrow{T2}$.
5. Nun wird aus den oben genannten Gründen vereinfacht und dabei die Z-Komponente ignoriert (Skizze im Anhang, Kapitel V.VII). Dadurch wird ein bestimmtes Gleichungssystem erreicht. Dieses wird anschliessend nach s und r aufgelöst. Dabei entstehen folgende Formeln:
$$s = \frac{-(D1_y * R1_x + R1_y * (D2_x - D1_x) - R1_x * D2_y)}{R1_y * R2_x - R1_x * R2_y}$$
$$r = \frac{D2_x + s * R2_x - D1_x}{R1_x}$$
6. Nun kann die Position der beiden Punkte: $\overrightarrow{T1}$ und: $\overrightarrow{T2}$ berechnet werden, indem die beiden Formeln unter Punkt 3 angewendet werden.
7. Anschliessend wird der Distanzvektor, welcher sich zwischen den beiden Punkten: $\overrightarrow{T1}$ und $\overrightarrow{T2}$ befindet, mittels der Gleichung: $\overrightarrow{R4} = -\overrightarrow{T1} + \overrightarrow{T2}$ berechnet. Dieser Vektor wird benötigt, da sich das Ziel genau zwischen den Punkten $\overrightarrow{T1}$ und $\overrightarrow{T2}$ befindet.
8. Jetzt kann die Position des Ziels berechnet werden durch: $\vec{T} = \overrightarrow{T1} + \frac{\overrightarrow{R4}}{2}$
9. Letztlich können die Distanzen zwischen dem Device 1, dem Device 2 und dem Zielpunkt berechnet werden. Dies wird mittels folgender Formeln erreicht:
$$Distanz\ P1\ zum\ Ziel = \left|-\overrightarrow{D1} + \vec{T}\right|$$
$$Distanz\ P2\ zum\ Ziel = \left|-\overrightarrow{D2} + \vec{T}\right|$$

Anmerkung zur Berechnung:

Die physikalischen Einheiten der Vektoren sind während den Berechnungen stets in der metrischen Einheit «Meter» anzugeben. Denn die Einheit, mit der die Distanzen mittels der Winkel in die Vektoren umgerechnet werden, sind in Meter vorhanden.

3.2.3.4 BESCHREIBUNG FEHLERMÖGLICHKEITEN

Da auf eine Netzwerkverbindung zurückgegriffen wird, können diverse Fehler entstehen. Um diese abzufangen, wurde ein Fehlerzustand in den Zustandsautomaten implementiert. Dieser Fehlerzustand zeigt dem Nutzer die Fehlerbeschreibung auf, welche in der Datenstruktur hinterlegt ist, bevor in den Fehlerzustand gewechselt wurde. Es können folgende Fehler auftreten:

Fehleranzeige	Fehlerbeschreibung	Fehler-Auftretungsort
GPS signal invalid	Wenn Triangulation mit GPS ausgewählt wurde, aber das GPS-Modul kein gültiges GPS-Signal bekommt, da z.B. kein Satellit sichtbar ist.	Beim Starten der Messung und nach der ersten Messung
Please, enable ConnectionTi first	Wenn der ConnectionTi mode in den Einstellungen nicht aktiviert ist.	Beim Erstellen der HTTP-Anfrage
Please, select a device	Wenn kein Device in der Device Select List ausgewählt ist oder wenn kein Device in der Liste vorhanden ist.	
Device {xyz} does not support this function	Wenn das ausgewählte Device die Funktion Triangulation nicht unterstützt.	
Internal error	Tritt auf, wenn das empfangene JSON vom zweiten Device einen Formatierungsfehler erhält und dadurch nicht geparst werden kann.	Beim Auswerten der Antwort des HTTP requests
Response Error	Der MIME-Type der Antwort enthält kein JSON.	
Network Error	Tritt auf, wenn z.B. die Netzwerkverbindung abbricht oder zu lange keine Antwort des Gegenübers erfolgt.	Während die HTTP Verbindung offen ist oder beim Aufbau der Verbindung

Tabelle 4: Fehlermöglichkeiten passive Distanzmessung

3.3 Tests

Wird die Software im MOSKITO TI erweitert, ist es wichtig, diese zu testen. Aus zeitlichen Gründen und weil der bestehende Testaufbau des Gerätes sehr komplex ist, wurde auf eine Anpassung des Testframeworks mit dem in dieser Arbeit generierten Code verzichtet. Dennoch wurden im Laufe der Arbeit verschiedene Tests durchgeführt. Dabei wird zwischen Softwaretests und Feldtests unterschieden.

3.3.1 Softwaretests

Automatisierte Softwaretests, in denen zum Beispiel sämtliche Codefragmente abgearbeitet werden, um zu sehen, ob alles ausgeführt wurde, wurden nicht durchgeführt. In dieser Arbeit bestanden Softwaretests darin, mit der Ausgabe von Logs zu überprüfen, ob die Software sämtliche Teilstücke durchführt und die richtigen Ergebnisse hervorbringt. Teilweise wurden separate Anwendungen mit einem Teil des Codes programmiert, um zu sehen, ob dieser funktioniert. Erst danach wurde das Teilstück in den Hauptcode implementiert.

3.3.2 Feldtests

Die Funktionen der Arbeit sowie das darüberliegende Overlay-Netzwerk wurden immer wieder mit Geräten getestet. Der Haupttest fand mit acht Geräten statt, wovon sechs über WLAN und zwei über Ethernet verbunden waren. Sämtliche Geräte wurden auf das neuste Image aktualisiert und mit dem WLAN-Router verbunden. Der Router wurde so konfiguriert, dass jedes Gerät über seine MAC-Adresse, trotz DHCP, eine feste IP-Adresse zugewiesen bekommt. Die Geräte wurden mit dem NATO Buchstabieralphabet (*NATO Buchstabieralphabet*, o. D.) benannt und erhielten aufsteigende IP-Adressen. Das Gerät «ALPHA» erhielt die IP-Adresse 192.168.0.131, das Gerät «BETA» die Adresse 192.168.0.132 usw. Die Benennung und IP-Vergabe wurden so vorgenommen, dass sämtliche REST-Schnittstellen überprüft werden konnten.

Als das physische Netzwerk aufgebaut war, wurde der ConnectionTi mode in den Einstellungen aktiviert, der Handover mode ConnectionTi selektiert und die Funktion Auto handover aktiviert. Im

Abbildung 34: Screenshot mit sämtlichen Geräten in der Liste

Untermenü ConnectionTi > Device Select List wurden automatisch sämtliche neuen Geräte sofort hinzugefügt. Durch Ein- und Ausschalten des ConnectionTi mode wurde beobachtet, ob die Liste aktuell gehalten wird oder nicht. Es stellte sich heraus, dass die Zeiten etwas geändert werden müssen, da teilweise Geräte aus der Liste verschwanden und nach wenigen Sekunden wieder in der Liste auftauchten. Was ebenfalls auffiel war die sich nicht alphabetisch ordnende Liste. So musste stets überprüft werden, ob wirklich alle Geräte angezeigt wurden, da dies nicht auf einen Blick ersichtlich war. Dies könnte in Zukunft allenfalls noch angepasst werden. In Abbildung 34 ist die Liste mit sämtlichen Geräten zu finden.

Getestet wurden auch die Funktionen. Das Overlay-Netzwerk war stabil und so wurden Fotos auf Geräten aufgenommen und bei allen anderen Geräten überprüft, ob diese in der Mediengalerie zu finden waren. Dies funktionierte einwandfrei. Anschliessend wurden in der «Device Select List» einige Geräte abgewählt und wieder Fotos gemacht. Diese Fotos wurden nur an die selektierten Geräte gesendet.

Dasselbe wurde mit den Messungen durchgeführt. Dabei stellte sich heraus, dass die neuste Software-Version noch einen Fehler enthielt und die Messpunkte deshalb nicht gesendet wurden. Der Fehler konnte im Code lokalisiert und behoben werden und im anschliessenden Test wurden die Messungen fehlerfrei übertragen.

Zum Schluss wurde die Funktion passive Distanzmessung mittels Triangulation getestet. Dazu wurde in der «Device Select List» nur ein Gerät ausgewählt, sodass erkannt wurde, welches Gerät als Messungspartner agierte. Die Messung konnte durchgeführt werden, jedoch nur, wenn sich das Gerät des Partners in einem freien Zustand befand. Frei bedeutet, dass sich der Gerätenutzer nicht im Menü (MissionControl) befand oder anderweitige Funktionen geöffnet hatte. Danach konnten sämtliche Teilschritte durchgeführt werden und das Resultat erschien. Die Genauigkeit des Resultats wurde nicht ermittelt und ist deshalb nicht aussagekräftig. In dieser Arbeit stand die Vernetzung im Zentrum, die Triangulation dient lediglich der Veranschaulichung und zeigt Nutzungsmöglichkeiten auf. Die mathematische Implementierung wurde nicht durch eine Fachperson auf ihre Richtigkeit geprüft und müsste vor einer Anwendung auf einem Gerät überprüft und allenfalls verbessert werden.

4 Fazit

Die vorliegende Bachelorarbeit «Vernetzung von Beobachtungsstationen» behandelt die Vernetzung verschiedener MOSKITO TIs. Diese Vernetzung wurde durch die Implementierung eines Overlay-Netzwerkes erreicht und mit drei verschiedenen Use Cases bzw. Anwendungen ergänzt. Das Overlay-Netzwerk ist stabil und auch beim Wegfallen eines oder mehrerer Geräte immer noch intakt. Die Übermittlung von Bildern und Messungen funktioniert einwandfrei. Sie können an alle oder nur an selektierte Geräte im Overlay-Netzwerk gesendet werden. Die Funktion der passiven Distanzmessung mittels Triangulation kann ebenfalls aufgerufen werden. Die Ergebnisse der Tests zeigten jedoch, dass die Resultate nicht sehr akkurat sind. Dies liegt an der Genauigkeit des eingebauten Kompasses. Sobald die Richtung des Kompasses von einem Gerät zum anderen abweicht, beispielsweise durch lokale Störfaktoren wie ein Neodym Magnet in der Brille des Anwenders, das Stahlkonstrukt eines Gebäudes oder fehlende Eichung, weicht das Resultat stark vom richtigen Wert ab.

Alle Anforderungen und Ziele des Industriepartners Safran Vectronix AG und der Aufgabenstellung der Bachelorarbeit wurden somit erreicht und erfüllt.

5 Reflexion

Das Erarbeiten dieser Bachelorarbeit war eine sehr interessante und lehrreiche Erfahrung. Sie bot uns einen Einblick in einen Bereich, den wir sonst wahrscheinlich nie angetroffen hätten. Das Arbeiten mit dem MOSKITO TI war sehr faszinierend. Die Einarbeitung in einen derart grossen und komplexen Code wie der des MOSKITO TI war herausfordernd. Die Erstellung einer für uns geeigneten Programmierumgebung dauerte sehr lange und war mit diversen Problemen verbunden. Die Arbeit im Team funktionierte sehr gut. Arbeiten konnten aufgeteilt werden und es war möglich, sich gegenseitig zu unterstützen, zu motivieren und gemeinsam ein solides Endergebnis zu erzielen.

6 Ausblick

Obwohl wir sehr zufrieden sind mit unserer Arbeit, gibt es aus unserer Sicht noch einige Punkte, die am MOSKITO TI verbessert oder erweitert werden könnten. Der Aufbau des Overlay-Netzwerkes funktioniert sehr gut. Die Liste, die im «Device Select List» erstellt wird, könnte jedoch noch etwas ansprechender geordnet werden. Allgemein ist die Benutzererfahrung noch verbesserungsfähig. Teilweise gibt es Funktionen, die aufgerufen werden können, welche dem Nutzer kein Feedback geben, sodass dieser sich unter Umständen fragt, ob im Hintergrund überhaupt etwas gemacht wurde. Auch bei der Übermittlung der Daten gäbe es noch Verbesserungspotenzial. So ist es aktuell nicht möglich, zu erkennen, welches Bild vom eigenen Gerät aufgenommen wurde und welches von einem anderen Gerät aus empfangen wurde. Die Funktion passive Distanzmessung mittels Triangulation müsste mit den Berechnungen für GPS erweitert und überprüft werden. Das Thema Sicherheit wurde in dieser Arbeit nicht berücksichtigt und müsste bei der definitiven Implementierung im Gerät ebenfalls nachgerüstet werden. Während der Implementierung vom POST eines Bildes in der REST-Schnittstelle wurde ein Fehler im bestehenden Code aufgedeckt. Dieser tritt auf, wenn grosse

Datenmengen empfangen werden. Wenn dieser behoben wird, müsste auf den POST eines Bildes nicht mehr ein GET folgen, um das Bild herunterzuladen.

I Abbildungsverzeichnis

II Tabellenverzeichnis

III Literaturverzeichnis

Eilebrecht, K. & Starke, G. (2018). *Patterns kompakt: Entwurfsmuster für effektive Softwareentwicklung (IT kompakt) (German Edition)* (5., Ak. und erw. Aufl. 2019 Aufl.). Springer Vieweg.

NATO Buchstabieralphabet. (o. D.). Buchstabieralphabet. Abgerufen am 19. August 2022, von https://www.buchstabieralphabet.org/cms/nato.php

Papula, L. (2017). *Mathematische Formelsammlung*. Springer Publishing.

Qt Company. (o. D.-b). *Qt Framework - One framework to rule all!* Abgerufen am 15. August 2022, von https://www.qt.io/product/framework

Safran Vectronix AG. (2020, 20. April). *MOSKITO TI Family*. Safran Vectronix. Abgerufen am 12. August 2022, von https://safran-vectronix.com/product/moskito-ti-family/

Tanenbaum, A. S. & Wetherall, D. (2011). *Computer Networks*. Prentice Hall.

IV Eidesstattliche Erklärung

Die Verfasser erklären eidesstattlich, dass wir die vorliegende Arbeit selbstständig, keine anderen als die angegebenen Hilfsmittel benutzt und alle aus ungedruckter, sowie gedruckter Quellen, Literatur oder Internet im Wortlaut oder im wesentlichen Inhalt übernommenen Formulierungen gemäss den Richtlinien wissenschaftlicher Arbeiten zitiert wurden. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Ort, Datum……

Unterschrift der Verfasser

Patrick Good

Benjamin Koch

V Anhang

V.I Pseudocode Devicelist::setDevice

```
void Devicelist::setDevice(HostAdress elem, string name) {
| // Überprüfen ob elem existiert
| sperre Deviceliste
| foreach(dev : jedesDevInListe) {
| | if(hat Device gleiche IP wie elem) {
| | | update Zeit im Device
| | | entsperre Deviceliste
| | | return
| | }
| }
| entsperre Deviceliste
|
| // überprüfe ob elem Localhost ist
| Sperre Netzwerkliste
| foreach(IpAlsInt : jedesDevInListe) {
| | if(IpAlsInt hat gleichen Wert wie elem zu IP) {
| | | entsperre Netzwerkliste
| | | return
| | }
| }
| Netzwerkliste
|
| Erstelle neues Device mit name und elem
| sperre Deviceliste
| Füge neues Device zur Liste hinzu
| entsperre Deviceliste
|
| sende Signal, dass sich die Liste geändert hat
| sende Signal, dass ein Device hinzugefügt wurde
}
```

V.II Pseudocode Discoverer::Discoverer

```
Discoverer::Discoverer(port, settings) {
|  Speichere port und settings in globalen Variabeln
|
|  Verbinde alle Signale von der Deviceliste mit den eigenen Signalen (Weiterleitung)
|  // insgesamt 3 Signale, aus Einfachheit weggelassen
|
|  Erstelle neuen UDP Socket
|  Binde den UDP Socket an den port und die Eigenschaft ShareAddress
|  Erstelle ein Host Objekt im Heap
|
|  Verbinde von dem Timer das Signal timeout mit der Funktion sendDatagramm
|  Verbinde von dem Socket das Signal lesebereit mit der Funktion processPendingDatagramm
|  Verbinde den ipSuchTimer das Signal timeout mit der korrekten Funktion
|  Verbinde von den settings das Signal NameChanced mit der Funktion nameChanced
|
|  Starte die beiden Timer mit den Zeiten, welche im config file hinterlegt sind.
}
```

V.III Pseudocode Devicelist::processPendingDatagramm

Wird ausgeführt, wenn ein Paket am UDP-Socket fertig empfangen wurde.

```
void Discoverer::processPendingDatagramm() {
|  while(wiederhole solange UDP-Socket Pakete hat) {
|  |  Grössenanpassung von dem Bytearray, um daten zu empfangen.
|  |  Lese Daten von UDP-Socket und schreibe sie in den ByteArray.
|  |  Hole UDP-Header von dem empfangenen Paket.
|  |
|  |  Erstelle die Datenfelder fall und deviceName.
|  |  Extrahiere die Daten vom Bytearray und schreibe sie in den Fall und deviceName.
|  |
|  |  switch(Fall) {
|  |  |  EMPTY: tu nichts
|  |  |  DISCOVER: gib Name und den UDP-Header an Devicelist weiter (setDevice)
|  |  |  NAME_CHANGE: gib Name und den UDP-Header an Devicelist weiter (nameChange)
|  |  |  SHUTDOWN: gib UDP-Header an Deviceliste weiter zum Entfernen des Devices
|  |  |  Default: tu nichts
|  |  }
|  }
}
```

V.IV Abbildungen der REST-Schnittstelle

1. Node …/api/info/device:


```
{
    "bluetooth": {
        "bluetooth-enbaled": false,
        "bluetooth-name": "MTI-2415"
    },
    "connection-ti-mode-name": "RUMPELSTILZLI",
    "device-id": "913090-002415",
    "ethernet": {
        "ethernet-ip-address": "192.168.0.104"
    }
}
```


2. Node …/api/measurements


```
[
    {
        "id": "f983d924-e29c-470b-b4a6-551b27233837",
        "timestamp": {
            "date-time-utc": "2022-08-15T10:21:54Z",
            "utc-offset": "+00:00"
        }
    },
    {
        "id": "302de561-42d6-4fd4-a069-3f9d3ba4ecd0",
        "timestamp": {
            "date-time-utc": "2022-08-15T10:21:50Z",
            "utc-offset": "+00:00"
        }
    },
    {
        "id": "69c73ce3-f41b-4295-b3c4-7a72e63e7eaa",
        "timestamp": {
            "date-time-utc": "2022-08-15T10:21:47Z",
            "utc-offset": "+00:00"
        }
    }
]
```

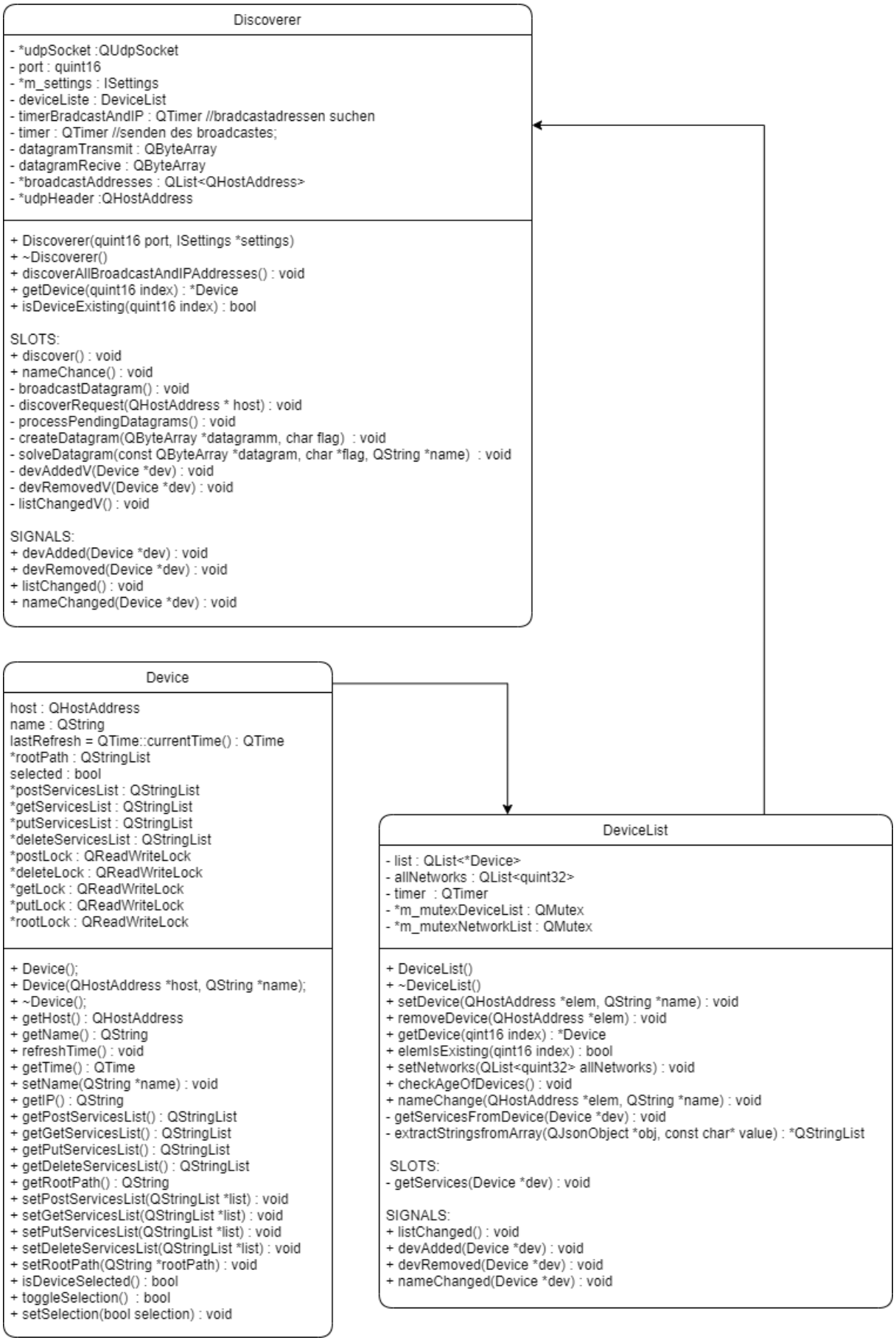
3. Node …/api/measurements/f983d924-e29c-470b-b4a6-551b27233837


```
{
    "device-id": "916362-000000",
    "distance": {
        "distance-m": 10.51,
        "status": "ok"
    },
    "id": "f983d924-e29c-470b-b4a6-551b27233837",
    "orientation": {
        "azimuth-magnetic-deg": 19.117,
        "azimuth-precision-deg": 1,
        "bank-deg": -2.747,
        "declination-mode": "auto",
        "inclination-deg": 9.328,
        "status": "ok"
    },
    "own-position": {
        "source": "internal-gps",
        "status": "no-communication"
    },
    "target-position": {
        "status": "invalid"
    },
    "timestamp": {
        "date-time-utc": "2022-08-15T10:21:54Z",
        "utc-offset": "+00:00"
    }
}
```

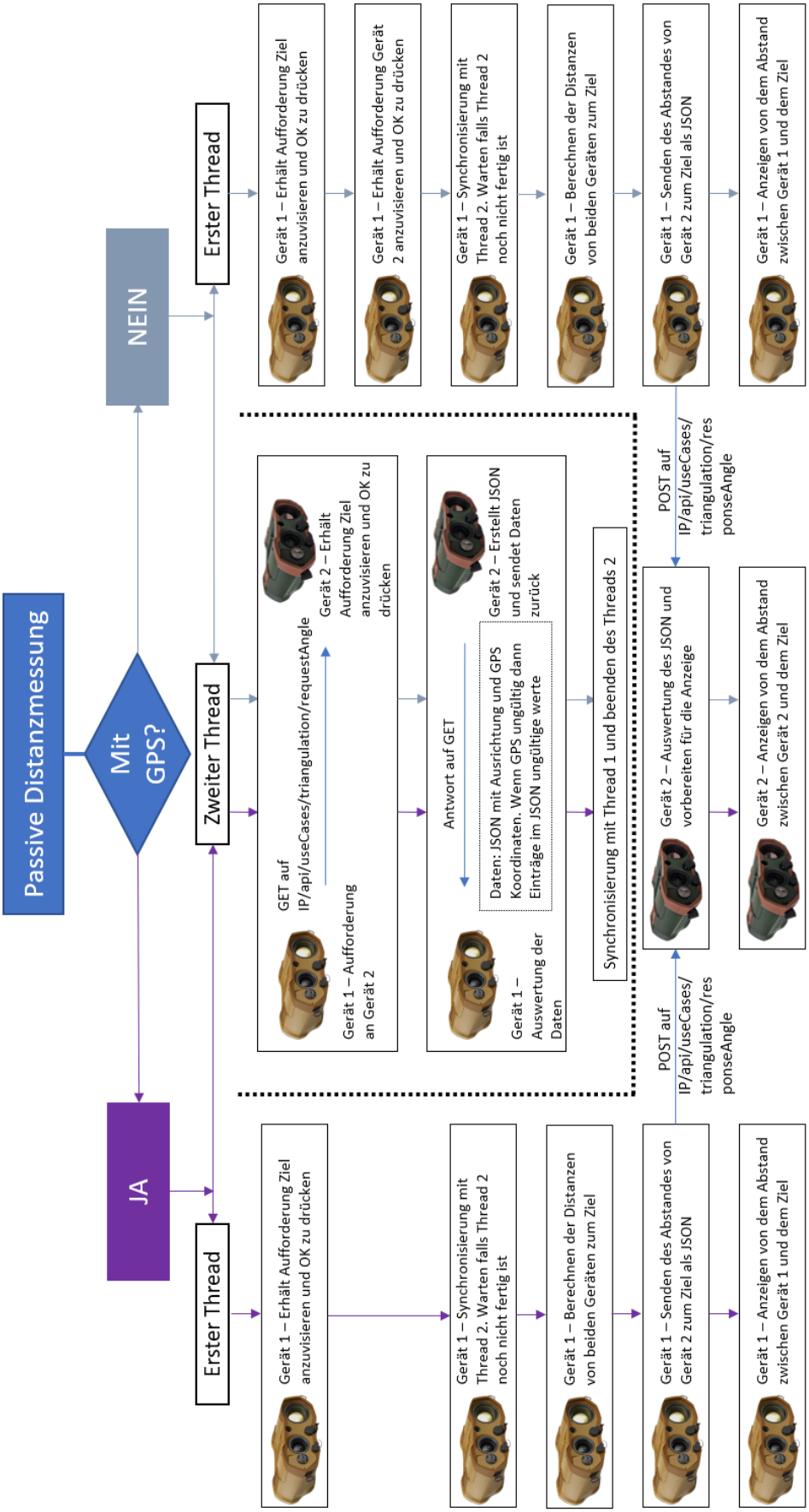

4. Node …/api/pictures


```
[
    {
        "id": "0006_20220813T082010Z.jpg",
        "media-type": "image/jpeg",
        "size-bytes": 119033,
        "timestamp": {
            "date-time-utc": "2022-08-13T08:20:11Z",
            "utc-offset": "+00:00"
        }
    },
    {
        "id": "0005_20220813T081557Z.jpg",
        "media-type": "image/jpeg",
        "size-bytes": 113118,
        "timestamp": {
            "date-time-utc": "2022-08-13T08:15:57Z",
            "utc-offset": "+00:00"
        }
    },
    {
        "id": "0004_20220813T081435Z.jpg",
        "media-type": "image/jpeg",
        "size-bytes": 120906,
        "timestamp": {
            "date-time-utc": "2022-08-13T08:14:35Z",
            "utc-offset": "+00:00"
        }
    }
]
```

V.V Klassendiagramm



V.VI Ablaufdiagramm Passive Distanzmessung



V.VII Skizze für mathematische Berechnung ohne GPS