3D-Visualisierung

Informatik-Projekt WS 2021/2022
Gruppe 2

Frankfurt University of Applied Sciences
Fachbereich 2, Informatik (B.Sc.)

Dozenten:	Markus Kraußе	
	David Merkl	
Autoren:	Dominik Hansen	1231124
	Gregor Napierski	1219667
	Lucas Röder	1093414
	Martin Komarnicki	1260603
	Pascal Kautzmann	1320404
	Philipp Bao-Phuc Pham	1321843

Abgabetermin: 14.02.2022

1 Einleitung

1.1 Projektbeschreibung

Die an die Gruppe zugewiesene Aufgabe erfordert die Auseinandersetzung mit den BNO055 9-Achsensensor von BOSCH und die Realisierung eines Anwendungsfalls für diesen Sensor. Dieser besteht aus einem Beschleunigungssensor, einem Gyroskop und einem Magnetometer. Im Konkreten sollen die gemessenen Daten des Sensors erfasst und mittels ZigBee-Funkmodule an einen Computer zur Weiterverarbeitung übertragen werden. Dazu wird der Aufbau eines ZigBee-Funknetzwerk benötigt. Anhand der Sensordaten soll dann die Lage der Funkmodule im Raum bestimmt und grafisch visualisiert werden. Zum einen soll die grafische Visualisierung durch die aus den Sensordaten berechneten Eulerwinkel und zum anderen durch die vom Sensor zur Verfügung gestellten Quaternionen erfolgen. Zudem wird die Speicherung der erfassten Daten mit zusätzlichen Zeitstempel in eine Datenbank verlangt, womit eine spätere Betrachtung und Auswertung der Daten ermöglicht wird. Schließlich soll noch ein konkreter Anwendungsfall für den Sensor realisiert werden. In dieser Projektarbeit wurde entschieden, ein Spiel als Anwendungsfall für den Sensor zu implementieren. Im Wesentlichen handelt es sich dabei um ein Raumschiff-Spiel, bei dem der Spielende ein Raumschiff mit Hilfe des Sensors steuert und dabei versucht verschiedenen Hindernissen auszuweichen.

1.2 Motivation

Für einen 9-Achsensensor, bestehend aus Beschleunigungssensor, Gyroskop und Magnetometer, gibt es im Allgemeinen sehr viele unterschiedliche Anwendungsgebiete. So kann dieser z.B. bei der Bestimmung und Stabilisierung der Lage von Flugobjekten wie Drohnen oder auch die Bestimmung der Ausrichtung eines Handydisplays verwendet werden. Darüber hinaus kann der Sensor auch zur Realisierung von Spielen, wo die Steuerung z.B. durch die räumliche Lage eines Controllers erfolgt, eingesetzt werden. Eben daran knüpft sich der zu realisierende Anwendungsfall dieser Projektarbeit. Die Zielsetzung war daher, die Daten des Sensors richtig zu erfassen, zu verstehen und zu interpretieren.

2 Konzeptualisierung

2.1 Anforderungsanalyse

Basierend auf der beabsichtigten Anwendung und des oben beschriebenen Projektauftrages, sind folgende Anforderungen an das Projekt notwendig.

2.1.1 Funktionale Anforderungen

Hardware

- Aufbau des ZigBee-Funknetzwerks
- Ansteuerung des Bosch 9-Achsensensors BNO055
- Batteriespannung des Funkmoduls überwachen
- Übermittlung der Daten an ein am PC angeschlossenes Koordinator Funkmodul

Software

- Auslesen der Daten durch eine Applikation
- Berechnung der Lagewinkel (Beschleunigungssensor/Gyroskop und Quaternionen)
- Grafische 3D Visualisierung von mindestens zwei Funkmodulen
- Speichern der Sensorwerte mit Zeitstempel
- Visualisierung der Sensorwerte über ein Koordinatensystem
- Realisierung eines Anwendungsfalles

2.2 Stack

Die Auseinandersetzung mit den BNO0055 und die daraus resultierende Entwicklung einer Software erfordert die Nutzung von verschiedenen Technologien. Alle verwendeten Technologien zusammengefasst bezeichnet man als Solution-Stack. Nachfolgend sind die zur Lösung des Projektauftrages verwendeten Technologien aufgelistet.

BitCloud-Software-Stack
Das Projekt verwendet ZigBee-Funkmodule als Hardware. Zur Entwicklung mit der ZigBee Hardware wird der BitCloud-Software-Stack verwendet. BitCloud ist eine voll eingebettete Software-Entwicklungsplattform von Atmel [1]. Sie bietet ein Rahmen für die Entwicklung von ZigBee-Hardware.

Python
Für die Bearbeitung der vom Sensor an den PC ermittelten Daten, wurden Python-Anwendungen entwickelt. Python ist eine interpretierte, höhere Programmiersprache und bietet zahlreichen Module/Bibliotheken an, mit denen verschiedene Anwendungen entwickelt werden können. Unter anderem gibt es z.B. Bibliotheken zur 3D-Modellierung (VPython) oder zur Spieleentwicklung (Ursina), womit die an das Projekt gestellten Anforderungen realisiert werden können.

Git
Das Projekt ist ein Gruppenprojekt, bei dem die Mitglieder idealerweise unabhängig voneinander an geplanten Funktionen und Anforderungen arbeiten, um gemeinsam entwickelte Designkonzepte zu realisieren. Dies bedeutet, dass Code an mehreren Enden gepflegt und erweitert werden muss. Ein Werkzeug, das dabei behilflich sein kann, ist die Versionsverwaltungssoftware Git. Git unterstützt die Nachverfolgung von Änderungen an der Anwendung. Vor allem hilft es bei der Koordination der Arbeit an einem einzigen Projekt mit mehreren Entwicklern.

2.3 Organisation und Workflow

Gute Kommunikation zwischen den Projektmitgliedern, ein gut organisierter Projektplan und eine gerechte Aufgabenteilung sind die Schlüssel für eine erfolgreiche Lösung des Projektauftrages. Es wurden daher verschiedene Anwendungen wie etwa *Zoom* und *WhatsApp* verwendet, um eine einfache und schnelle Kommunikation und Organisation zu ermöglichen. Über Zoom wurden wöchentlich Meetings abgehalten, in denen jedes Projektmitglied den anderen Mitgliedern z.B. ihren wöchentlichen Fortschritt oder auch Veränderungen am Code erklären und zeigen konnten. Des Weiteren erwies sich Zoom in der Entwicklungsphase als recht vorteilhaft, da Mitglieder ihren Desktop-Bildschirm mit ihren Kollegen teilen konnten, sodass diese den den Fortschritt ihrer Arbeit sehen und codebasierte Fragen stellen konnten. Speziell bei der Entwicklung der Spiels als Anwendungsfall wurde zusätzlich das Organisationstool *Trello* verwendet. Dieses schaffte ein eine klare Übersicht, welches Mitglied an welchen Anforderungen des Spiels arbeitet, und verhinderte, dass ein Feature gleichzeitig von zwei Mitgliedern bearbeitet wurde. Ein Trello-Board besteht aus einer To-Do-Liste, einer Liste aus im Moment bearbeiteten Aufgaben und einer Liste aus fertig implementierten Features.

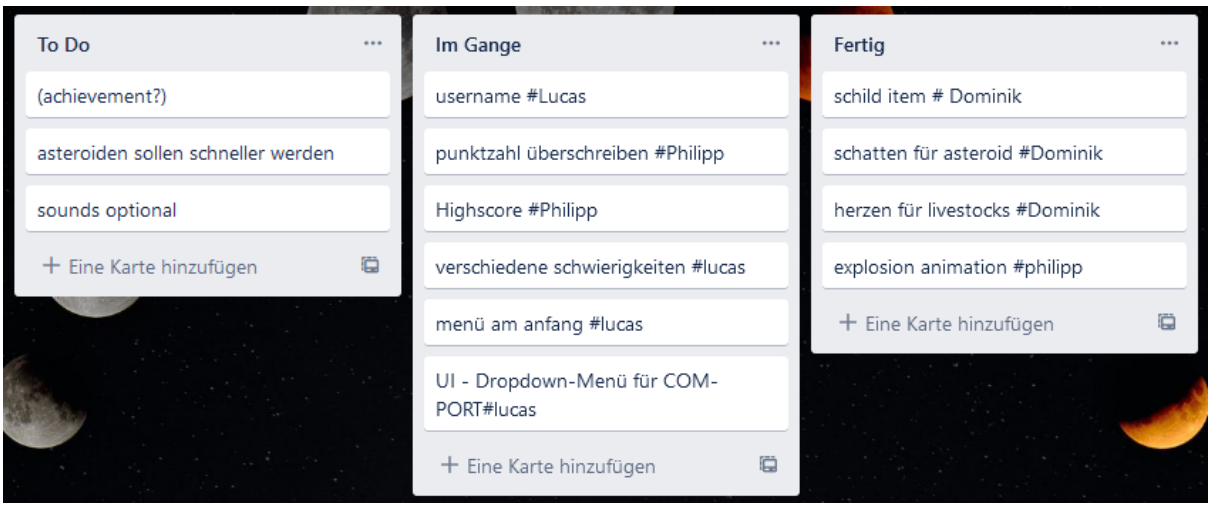

Abb. 1 Trello-Board

Für die Arbeit am Anwendungscode wurde, wie bereits in 2.2. erklärt, die Versionsverwaltungssoftware Git verwendet. Mit Git hat jedes Projektmitglied ein Repository, das eine lokale Version der Codebasis ist. Das Repository wird zentral auf GitHub gehostet. GitHub ist ein webbasierter Dienst, der versionskontrollierten Code hostet. Das Konzept besteht darin, dass jeder Projektentwickler einen eigenen Zweig aus der zentralen Entwicklungsversion erstellen und darauf lokal arbeiten kann. Sobald z.B. ein Feature fertig implementiert wurde, kann dieser von den anderen Projektentwicklern überprüft werden, bevor es in die zentrale Entwicklungsversion aufgenommen wird.

2.3.1 Aufgabenverteilung

Hardware Team:

Gregor Napierski	Sensor auslesen/ZigBee-Netzwerk/Bitcloud Integration
Martin Komarnicki	Bitcloud Integration/Sensor auslesen/Coordinator
Pascal Kautzmann	serielle Schnittstelle/Coordinator

Software Team:

Dominik Hansen	Weltraumspiel/Blender
Lucas Röder	Weltraumspiel/Datenbank/GUI
Philipp Bao-Phuc Pham	Weltraumspiel/Visualisierung

3 Hardware

Bei der Entwicklung für das Hardware Programm, wurden uns drei ZigBee Module welche mit einer 2,4GHz-Antenne ausgerüstet waren, zwei BNO055 Sensoren mit Adaptern verwendet, sowie eine UART Bridge. Darüber hinaus ein JTAG-Programmer zum Flashen und zwei Batterie Packs. Als Software wurde Atmel Studio 6.2 verwendet. Für das Programm wurde mit dem Bitcloud stack gearbeitet und es wurde die von Bosch zur Verfügung gestellte Library zum Arbeiten mit dem BNO055 verwendet.

3.1 BNO055

Der BNO055 ist ein System-In-Package und besteht aus drei Sensoren. Einem dreiachsigen 14-bit Beschleunigungsmesser, einem dreiachsigen 16-bit Gyroskop, einem dreiachsigen geometrischen Sensor. Zudem besitzt der BNO055 einen 32-bit cortex M0+ Microcontroller, welcher Bosch Sensortec sensor fusion Software betreibt.

Power Modes

Es besteht die Möglichkeit den BNO055 in drei verschiedene Power Modes zu setzen:

- Normal Mode: Die Sensoren, welche durch den Operationsmodus gebraucht werden, sind durchgehend angeschaltet.
- Low Power Mode
- Suspend Mode: Alle Sensoren sind im Schlafmodus

Für das Projekt wurden alle Sensoren im Normal Mode betrieben, weil sich alle Module nah beieinander befinden und jederzeit zugänglich sind. Es ist daher nicht nötig, Energie zu sparen.

Operation Modes

Der BN0055 besitzt eine Vielzahl an Operationsmodi, welche jeweils unterschiedliche Daten auslesen und speichern. Diese sind grundsätzlich in Non-Fusion Modes und Fusion Modes zu unterteilen. Die Fusion Modes nehmen Gebrauch von dem 32-bit Microcontroller, um beispielsweise aus Beschleunigung, Magnetfeld und Drehbewegung die Quaternionen auszurechnen. Nach einer Stromzufuhr befindet sich der BNO055 im CONFIGMODE. In diesem Operationsmodus werden keine Werte gemessen. Um den Operationsmodus zu wechseln, muss in das Register 0x3D hineingeschrieben werden. Aus dem 8-Bit

Register bestimmen jedoch nur die letzten 4-bits den Operationsmodus. Die restlichen Bits sind reserviert.

Beide BNO055 Sensoren laufen im NDOF (Nine degrees of freedom), da in diesem Modus alle Daten gespeichert werden, welche zur Realisierung dieses Projektes nötig sind. [2]

3.2 Netzwerk

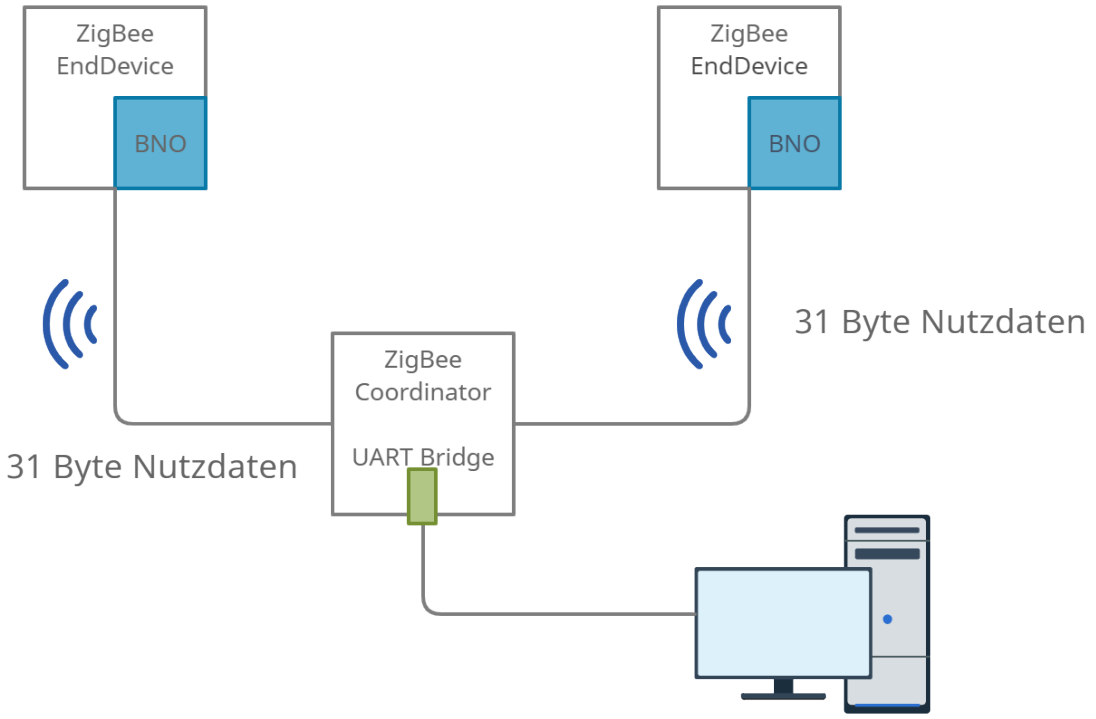

Abb. 2 Sensornetzwerk

Das Netzwerk beinhaltet zwei ZigBee Module, an denen jeweils ein BNO055 über einen Adapter angeschlossen ist. Diese dienen als Endgeräte. Ein drittes ZigBee Modul dient als Koordinator. An diesem ist eine UART Bridge angeschlossen. Ein Endgerät misst die Quaternionen. Das andere Modul misst die Beschleunigungs-, Magnetfeld- und Rotationsdaten. Die Daten werden über ein Paket mit 31 Byte Nutzdaten versendet. Der Aufbau der Nutzdaten ist genauestens definiert. So ist beispielsweise das 14 bis 19 Byte vorgesehen für die Magnetfeld Daten. Der Koordinator empfängt die Daten und gibt sie über die UART-Schnittstelle weiter.

Für den Aufbau des Netzwerkes wurde eine zusätzliche Variable in configuration.h eingeführt. DEV_ID beschreibt welche Rolle das Modul annimmt und welche CS_UID dem Modul zugeschrieben wird.

0 = Coordinator

1/2 = Endgerät

Die callback Funktion welche beim erhalten eines Datenpakets aufgerufen wird, wechselt den Zustand der rote LED und ruft die Funktion *rawDataToUart(data)* oder *quatDataToUart(data)* auf.

3.3 I2C

Die Kommunikation zwischen ZigBee und BNO055 findet über den seriellen Datenbus statt. Zuerst wird der I2C initialisiert. Darauf folgt der Start, dabei wird die Slave Adresse mit einem read oder write Bit. Ein komplettes Byte wird dafür übergeben. Wichtig dabei, ist das Shiften der Slave Adresse um 1 Bit nach links. Das read oder write bit, welches den Wert 0 für write, hat wird drangehängt.

Für die write Operation wird zudem die Registeradresse übergeben, in welche hineingeschrieben werden soll und die hineinzuschreiben den Daten.

Die read Operation beginnt wie die write Operation. Die Slave Adresse wird mit einem write bit übergeben und dem abzulesen Register. Darauf folgt ein wiederholter Start der I2C Schnittstelle mit der übergabe der Slave Adresse und dem read bit. Daraufhin werden die Daten aus dem zuvor festgelegten Register gelesen.

3.4 Funktionen

3.4.1 Daten Auslesen

Der erste Ansatz für das Auslesen und Reinschreiben bestimmter Register, waren selbstgeschriebene I2C Funktionen. Die Entscheidung fiel aber auf die von Bosch zur Verfügung gestellte Library für das Arbeiten mit dem BNO055. Da beide BNO055 Sensoren im NDOF Modus laufen, ist es möglich, die Quaternionen und Beschleunigungs-, Magnetfeld- und Rotationsdaten auszulesen. Die Daten sind eingeteilt in LSB(Least Significant Bit) und MSB (Most Significant Bit). Die Register, in denen die Daten gespeichert werden, liegen hintereinander, sodass es möglich ist, alle Daten mit einer Readbefehl auszulesen. Zum Auslesen werden die Funktionen *bno055_read_accel_xyz_raw*, *bno055_read_gyro_xyz_raw*, *bno055_read_mag_xyz_raw* und *bno055_read_quaternion_wxyz_raw* verwendet. Diese Funktionen laufen wie folgt ab: Der Status der Page ID wird überprüft. Sollte die Page ID nicht 0 sein, wird diese auf null gesetzt. Darauf folgt eine read Operation durch die Funktion *_bno055_i2c_bus_read*. Diese Operation unterscheidet sich in den jeweiligen Funktionen durch das Startregister, welches ausgelesen werden soll und

die Anzahl der auszulesenden Bytes. Bei den Quaternionen werden 8 Bytes ausgelesen und bei den Anderen 6 Bytes. Die Ausgelesenen Daten werden zurückgegeben.

3.4.2 External Crystal

Bosch empfiehlt die Verwendung eines Externen Crystals, weil der externe Crystal eine höhere Genauigkeit hat als der interne. Die Bestimmung eines externen Crystals wird in der Funktion *set_external_crystal()*. Dabei wird das Register 0x3F zuerst ausgelesen und das Bit, welches sich ganz links befindet, nimmt den Wert 1 an. Daraufhin wird das veränderte Byte wieder in das Register geschrieben.

3.4.3 Reset

Beim Reset wird der BNO055 zurückgesetzt. Dies geschieht durch das Überprüfen der Page ID. Sollte diese nicht 0 sein, wird die Page ID auf Null gesetzt. Daraufhin folgt das Auslesen des Registers 0x3F. Sollte dieses den Wert 0 haben, wird der übergebene Wert 1 um fünf stellen nach links geshiftet und in das Register geschrieben. Es wird ein Timer gestartet, welcher als Delay agiert. Wie aus dem Datenblatt zu entnehmen ist, muss nach einem Reset ein Delay erfolgen.

3.4.4 Sensor Initialisierung

Zuerst wird der Power Mode gesetzt. In die Funktion *bno055_set_power_mode* wird der Wert 0 übergeben. Der Wert 0 ist dafür zuständig, dass der BNO055 in den Normal Mode gesetzt wird. Es wird überprüft, ob sich der BNO055 im Configmode befindet. Sollte dies nicht der Fall sein, wird der Sensor in den Configmode gesetzt. Durch das Auslesen des aktuellen Power Modes

```
v_data_u8r = (v_data_u8r & ~0X03) | (v_power_mode_u8 & 0X03);
```

wird gewährleistet, dass nur die Bits, welche sich am weitesten rechts befinden verändert werden, da die anderen Bits reserviert sind.

Es wird danach ein externer Crystal gesetzt. Daraufhin wird der Operation Modus durch *bno055_set_operation_mode* gesetzt. Beide Sensoren laufen im NDOF Modus. Dies funktioniert nach dem gleichen Prinzip wie die Veränderung des Power Modes.

Die grüne LED wird zum Leuchten gebracht. Und zuletzt werden die Werte für den *readSensorTimer* festgelegt und der Timer wird gestartet. Der Timer arbeitet im Repeat Modus und wird alle 50ms aufgerufen. Die Callback Funktion ändert

lediglich den Status in APP_READ_SENSOR, sollte dieser vorher im Modus APP_READ_BAT sein.

3.4.5 Kalibrierungsstatus auslesen

Die Kalibrierung des BNO055 kann für jeden Sensor separat ausgelesen werden: *bno055_get_gyro_calib_stat, bno055_get_accel_calib_stat* und *bno055_get_mag_calib_stat*. Diese Werte gehen von 0 bis 3, wobei 3 bedeutet, dass der Sensor optimal kalibriert ist. Wenn der Wert nicht gleich 3 ist, ist die Kalibrierung nicht perfekt. Um diesen Status im Projektbetrieb zu veranschaulichen, werden die am ZigBee Modul angebrachten LEDs benutzt.

Der Kalibrierungsstatus des Gyroskops wird an der gelben LED angezeigt, wobei die LED bei einem Wert von 3 ausgeschaltet ist und bei jedem anderen Wert an ist.

Accell und Mag Sensor werden aus LED Knappheit beide durch die roten LED angezeigt. Blinkendes LED bedeutet, der Mag Sensor ist nicht kalibriert (!= 3) und durchgehendes Leuchten heißt, Accell ist nicht kalibriert. Wenn beide korrekt kalibriert sind ist die LED aus.

3.5 State Diagram

Der Ansatz für die Erstellung des State Diagrams, war es ein logisches und überschaubares Diagramm zu erstellen. Das Ergebnis waren 9 Zustände.

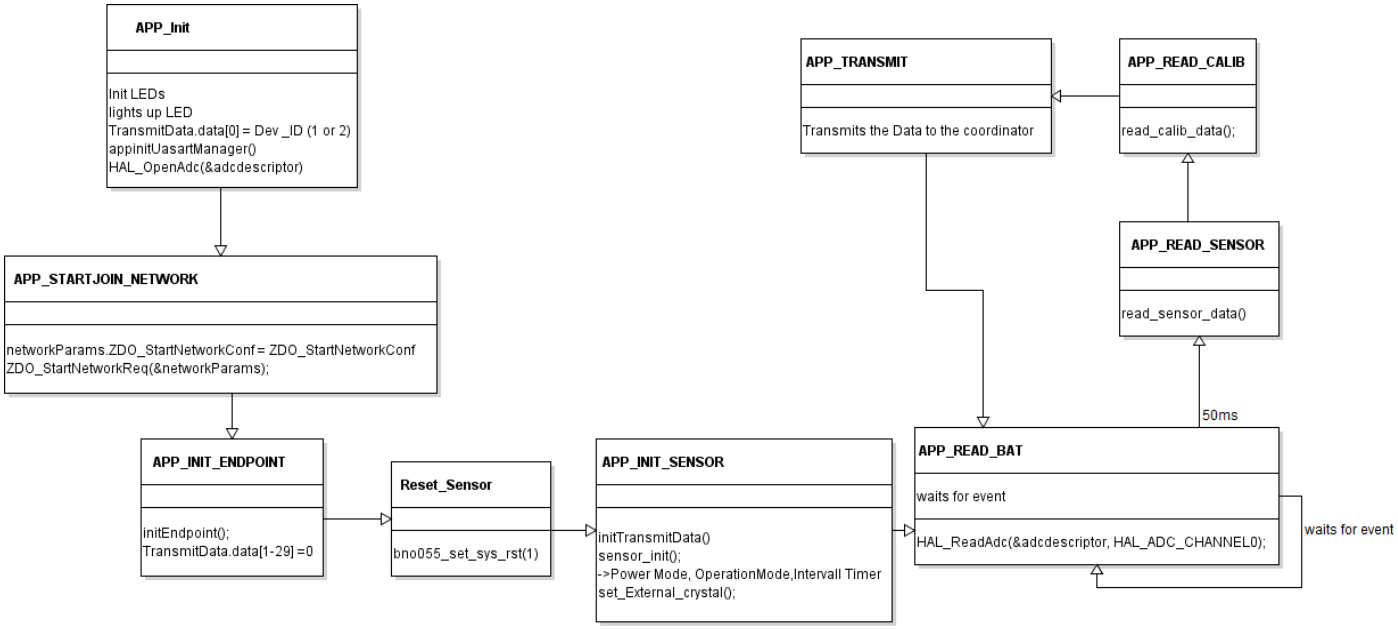

Abb. 3 Zustandsdiagramm

Die Abfrage der Zustände befindet sich in der Funktion *APL_TaskHandler()* und diese wird mit Switch Cases abgefragt. Hierfür wird die Variable *appstate* verwendet, welche in jedem State verändert wird.

APP_Init: Hier werden die Ports des Mikrocontrollers als Ausgänge definiert. Dadurch ist es möglich, die drei LEDs des ZigBee Module zum Leuchten zu bringen. Die gelbe und rote LED wird zum Leuchten gebracht. Die erste Stelle des zu übertragenen Datenpakets wird der Wert 1 oder 2 zugeschrieben. Dies ist abhängig vom jeweiligen Endgerät. Dies ist im späteren Verlauf wichtig. Durch das erste Byte bestimmt der Koordinator wie das Datenpaket zu verarbeiten ist. Mit der zu Verfügung gestellten Funktion *appinitUsartManager()* wird der UART-Manager initialisiert. Zudem wird der Analog-Digital-Converter Port mit der Funktion *HAL_OpenAdc (&adcdescriptor)* geöffnet. Darauf folgt die Veränderung des Status.

APP_STARTJOIN_Network: In diesem Zustand wird das Netzwerk gestartet.

APP_INIT_ENDPOINT: Der Endpunkt des zuvor aufgebauten Netzwerkes wird definiert. Zudem werden die Bytes 1-29 auf null gesetzt, damit der Datenaustausch funktioniert.

APP_Reset_Sensor: Es findet ein Reset durch die Funktion *bno055_set_sys_rst(1)* statt.

APP_INIT_SENSOR: Hier wird durch die Funktion *initTransmitData()* die Adressierungsart, die Endpunkt ID, Zieladresse und die Sender Endpunkt ID bestimmt. Der Sensor wird initialisiert. Dabei wird der Power Mode, ein externer Crystal und der Operation Mode in dieser Reihenfolge gesetzt und der Timer wird zum wiederholten Aufruf des Zustandes APP_READ_SENSOR gestartet. Der Timer wird alle 50ms aufgerufen und setzt den appstate auf APP_READ_SENSOR, sofern er vorher den Wert APP_READ_BAT hatte. Sollte dies nicht der Fall sein wird, der Zustand nicht geändert.

APP_READ_BAT: Dieser Zustand ist der „Wartezustand“. Der Appstate wird nicht ohne Aufruf des Timers verändert. Es wird außerdem noch der Batteriezustand abgelesen.

APP_READ_SENSOR: Hier wird in der Funktion *read_sensor_data()* anhand der DEV_ID entschieden, welche Daten ausgelesen werden sollen. Bei DEV_ID = 1 werden acc, gyro, mag ausgelesen, sonst werden die Quaternionen ausgelesen. Die ausgelesenen Daten werden daraufhin in das Datenpaket geschrieben.

APP_READ_CALIB: Es wird der aktuelle Status der Kalibrierung durch die Funktion *read_calibration_data()*; gelesen

APP_TRANSMIT: Hier wird das gesamte Datenpaket verschickt.

APP_NOTHING: Dieser Zustand ist nicht im Diagramm aufgeführt. Der Koordinator kommt in diesen Zustand nach der Initialisierung des Endpunktes. In diesem Zustand geschieht nichts. Jedoch bei jeder Ankunft eines Datenpakets wird die Callback Funktion aufgerufen, welche für die Umrechnung und Ausgabe sorgt.

3.6 Coordinator

Der Coordinator dient als Schnittstelle zwischen dem Funknetzwerk und dem Computer. Um diese Funktion zu erfüllen, muss zuerst mit dem Befehl "*appInitUsartManager()*" die Schnittstelle zum Computer geöffnet werden. Danach wird das Netzwerk durch den Befehl "*ZDO_StartNetworkReq(&networkParams)*" gestartet. Der Coordinator wartet nun auf eingehende Nachrichten, um diese an den Computer weiterzuleiten.

Da die Daten als Byte beim Coordinator ankommen, müssen sie zuerst noch umgeformt werden, damit diese von Python weiterverarbeitet werden können.

Umformen
In Python wird ein bestimmtes Format der Daten erwartet. Es sieht wie folgt aus:
Für die Rohdaten:
0,accx,accy,accz,gyrox,gyroy,gyroz,magx,magy,magz,batterie
Für die Quaternionen:
1,quat0,quat1,quat2,quat3,batterie
Zur Unterscheidung zwischen den Achsen und den Quaternionen steht an erster stelle eine 0 oder eine 1. Danach folgen die entsprechenden Werte. Diese werden immer von einem Komma getrennt. Zum Schluss folgt der Batteriestand des Moduls. Aus dem Sensor wird für jeden Wert immer ein LSB und ein MSB ausgelesen, um diese in float zu ändern, müssen sie erst zusammengesetzt werden. Dies funktioniert in C mit einer shift operation: "*(int16_t)(data[1]|data[2]<<8)*" .
Um die Werte nun in der richtigen Einheit zu erhalten, müssen sie noch weiter geändert werden. Dafür müssen sie noch durch einen bestimmten Wert geteilt werden. Diese sind im Datenblatt des Sensors zu finden: Für den Accelerometer 100, den Magnetometer und das Gyroskop 16 und für die Quaternionen 2^14.
Nachdem man die richtigen Werte hat, müssen sie noch in einen String geschrieben werden. Dafür muss der Teil vor dem Komma von den Nachkommastellen der Zahl getrennt werden.

Dies erfolgt in drei Schritten: Zuerst wird nur der Integer Teil der Zahl genommen und in eine Variable gespeichert "*intpart = ((int)f)*", danach wird dieser von der ganzen Zahl subtrahiert "*decpart = (f) - (intpart)*". Da diese Zahl jetzt nur noch aus einem Nachkomma Teil besteht, kann man sie mit 1000 multiplizieren, um 3 Nachkommastellen als Integer zu erhalten "*dec = (int)(decpart*1000)*".
Da die Zahlen sowohl positiv als auch negativ sein können, müssen, bevor man diese in ein Array schreiben kann, unterschieden werden, damit die Zahlen richtig übertragen werden können.

4 Software

Auf Grundlage der gemessenen und vom Computer erhaltene Sensordaten gibt es drei hauptsächliche Anforderungen für die zu realisierende Software. Die Implementation der Software besteht daher aus der grafischen Echtzeit 3D-Visualisierung der Sensoren, aus der Speicherung der Daten in eine Datenbank und aus der Realisierung eines Spiels, bei der der Sensor als Steuerungselement gilt.

4.1 Dateistruktur

Um eine Lesbarkeit der Daten und eine Navigation in den Dateien zu erleichtern, ist eine klare Dateistruktur notwendig. UI-bezogene Dateien befinden sich in den Ordner *images, icons, font*s. Für das Spiel verwendete Texturen und Audiodateien befinden sind in den Ordnern *models_compressed* und *audio* lokalisiert. Allgemein wurde für jeden Teilbereich der Software (z.B. 3D-Visualisierung, Spiel, Berechnung Winkel, usw.) ein Python-Skript erstellt.

```
\---3visu_zigbee
    +---audio
    +---fonts
    +---icons
    +---images
    +---model_compressed
    +---computation.py
    +---database.py
    +---diagrams.py
    +---GUI.py
    +---main.py
    +---visualization.ps
```

Abb. 4 Dateistruktur der Software

4.2 Lesen der seriellen Schnittstelle

Das Empfangen der Sensordaten, das vom ZigBee-Koordinator über die UART-Bridge an den Computer gesendet wird, ist die wesentliche Grundlage für die zu entwickelnden Software Anwendungen des Projektes. Das Lesen der seriellen

Schnittstelle, über die die Daten übertragen werden, ist daher eine zwingender Bestandteil der 3D-Visualisierung und des Spiels. In Python gibt es dafür das Modul "*pySerial*".
PySerial vereinfacht den Zugriff auf die serielle Schnittstelle und bietet die Möglichkeit auf den gängigen Betriebssystemen wie Windows, OSX, Linux usw. diese einfach auszulesen.
Um *pySerial* zu benutzen, muss es, nachdem es installiert worden ist, zuerst mit "*import Serial*" importiert werden. Danach kann ein Objekt mit "*serial.Serial*" erstellt werden. Im Konstruktor kann direkt der Portname und die Baudrate angeben werden. Als Baudrate werden die Standardwerte von 50 bis 115200 auf allen Plattformen unterstützt. Höhere Werte können aber je nach Gerät auch funktionieren. Da beim ZigBee-Modul die Baudrate 38400 benutzt wird und auch zum Testen über den Arduino Nano die Baudrate mit 115200 in den Standardwerten liegt, kann dieses Modul ohne Bedenken eingesetzt werden.

Da wir dem Nutzer die Möglichkeit geben beim Starten des Programms, den Port anzugeben, wird dem Konstruktor eine Variable, in der der Port Name geschrieben wird, und die Baudrate des ZigBee-Moduls übergeben: "*serial.Serial(com, 38400)*" .

Wenn dem Konstruktor direkt Werte übergeben werden, wird das Programm versuchen die Schnittstelle zu öffnen und somit ist es möglich sofort Daten auszulesen.

Bevor versucht wird Daten auszulesen, wird überprüft, ob Daten im Input Buffer vorhanden sind, das funktioniert über den Befehl *"in_waiting*". Dieser gibt die Anzahl der Bytes im Buffer zurück. Solange keine Daten vorhanden sind wird nichts weiter auf dem Port gemacht. Wenn Daten vorhanden sind, hat das Modul mehrere Möglichkeiten Daten auszulesen. Man kann einzelne Bytes oder ganze Zeilen auslesen. Um eine Zeile auszulesen wird der Befehl "*readline()*" benutzt. Mit diesem Befehl wird die nächste Zeile, bis sie von „\n" beendet wird, eingelesen.[3]

Nachdem das Datenpaket eingelesen wurde, muss es noch so umformatiert werden, sodass es von anderen Funktionen benutzt werden kann. Dafür wird als Erstes das Datenpaket mit hilfe von "*str(dataPacket, 'utf-8')*" in einen String umgewandelt. Danach wird mit "*strip()*" die Daten entfernt, die nicht gebraucht werden, wie zum Beispiel „\n". Nachdem das erledigt ist, wird durch "*split(",")*" der String in ein Array unterteilt. Somit ist es für andere Funktionen möglich die benötigten Daten aus dem Array auszulesen.

4.3 Graphische Benutzeroberfläche

Die ursprüngliche Idee einer graphischen Benutzeroberfläche (GUI) entstand, um den Zweck eines Spielmenüs zu erfüllen, von dem aus das Spiel gestartet werden kann. Im Laufe des Projekts kamen stetig mehr Ideen bezüglich der Funktionalität der GUI hinzu, bis schließlich beschlossen wurde, dass die GUI nicht nur als Spielmenü dienen solle. Im aktuellen Stand lassen sich durch die GUI, neben den eigentlichen Einstellungen für das Spiel, unter anderem die 3D-Visualisierung der Sensoren darstellen, sowie der COM-Port auswählen, an welcher der Sensor angeschlossen ist. Des Weiteren hat die GUI auch Zugriff auf die Datenbank, damit lassen sich Nutzernamen eintragen als auch Spielstände abrufen.

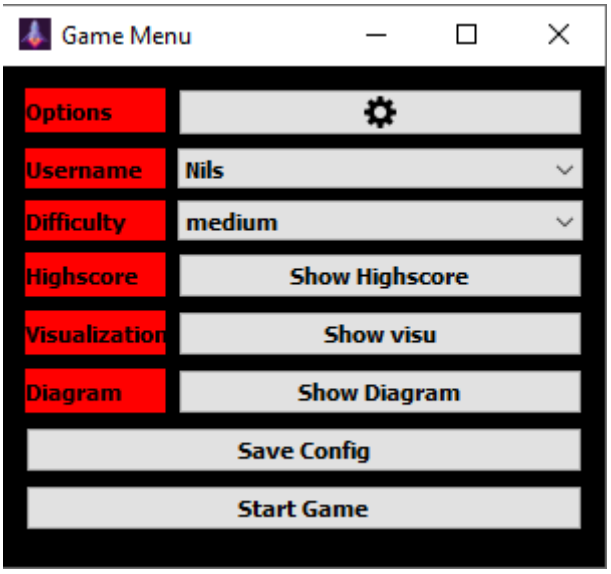

Abb. 5 Grafische Benutzeroberfläche

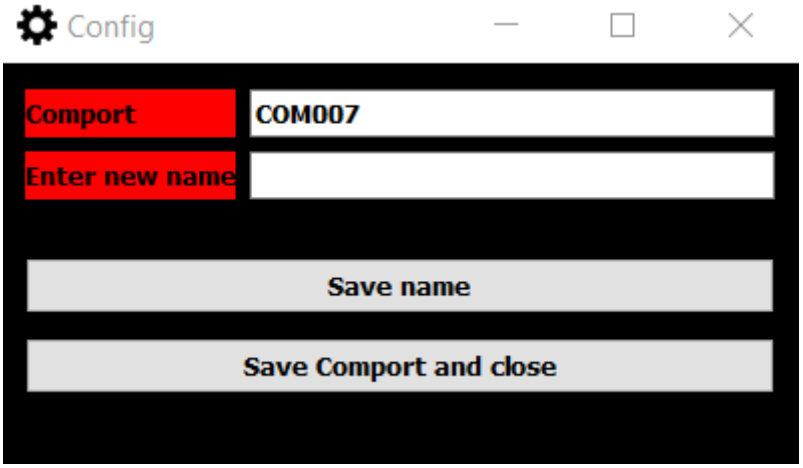

Abb. 6 Config Fenster

Die auf der Abbildung zu sehenden Fenster, sind mit Hilfe des QT-Frameworks entstanden bzw. den aktuellen* Python Ableger davon, PyQT5. Das Entwerfen der Formen sowie der Umgang mit den einzelnen Komponenten funktioniert dank der

ausführlichen Dokumentation von PyQT intuitiv und es treten nur sehr selten Probleme auf, die sich nicht durch eine kurze Recherche beheben lassen. Hinter jeder Funktionalität der grafischen Benutzeroberfläche steckt jedoch eine Herausforderung, die es zu lösen gilt. Die auf den ersten Blick erkennbare Herausforderung war es, die zwei Fenster zu initialisieren und miteinander arbeiten zu lassen. Im Options/Config Fenster lassen sich Namen in die Datenbank eintragen und im Idealfall sind die eingetragenen Namen auch unmittelbar danach im Hauptfenster der GUI zu finden. Diese holt sich aber die Namen beim Öffnen aus der Datenbank. Um zu gewährleisten, dass die Namen nach Eintragung in die Datenbank in der Liste erscheinen, wurden sogenannte Signale verwendet, die es ermöglichen Daten zwischen den einzelnen Fenster zu emittieren. Das Abrufen und Beschreiben der Daten in die Datenbank und in die config.ini stellte sich als weniger herausfordernd dar. Werte aus den einzelnen Komponenten, auch Widgets genannt, herauszulesen und hineinzuschreiben, lässt sich sehr einfach durch das Lesen der Dokumentation realisieren. Das Importieren von anderen Python Dateien sowie die Erstellung von Datenbank Queries war zu diesem Zeitpunkt kein Neuland mehr, ebenso war der Umgang mit der config.ini Datei schnell angeeignet.

*PyQT6 ist bereits erschienen, jedoch nicht vollständig ausgereift

4.4 3D-Visualisierung des Sensors

Ein wesentlicher Teil des Projektes besteht aus der Interpretation der erhaltenen Sensordaten und die daraus sich ergebende 3D-Visualisierung der Sensoren. Genauer gesagt, erfolgt die Visualisierung des einen Sensor über selbst berechnete Eulerwinkel und des anderen Sensor über die vom Sensor zur Verfügung gestellten Quaternionen.

4.4.1 VPython

Eine unabdingbare Voraussetzung für die 3D-Visualisierung, ist die Nutzung einer Bibliothek oder eines Frameworks, welches es, ermöglicht dreidimensionale Objekte wie Quader oder Kugeln zu erstellen und grafisch darzustellen. Die Python-Library “VPython” (für Visual Python) beinhaltet diese Eigenschaften und schien geeignet für die 3D-Visualisierung der Sensoren [10]. Der komplette Visualisierungsprozess befindet sich in der Datei *visualization.py*.

Erstellung 3D-Modell des Sensors

Der erste Schritt der 3D-Visualisierung, ist die Erstellung eines ungefähren 3D-Modells des Funkmoduls, auf dem sich der Sensor befindet. Dieser Aufbau geschieht in den Funktionen *sensorVis()* und *axisVis()* erreicht. In der folgenden Abbildung sieht man ein ungefähres Modell des Sensors mit den raumfesten und körperfesten Koordinatenachsen, das anhand der von VPython bereitgestellten Funktionalitäten geschaffen wurde.

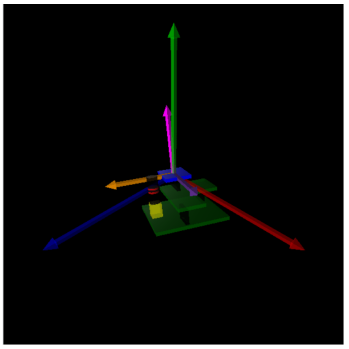

Abb. 7 3D-Modell des Sensors

4.4.2 Eulerwinkel und Quaternionen

3D-Visualisierung durch Eulerwinkel

Im nächsten Schritt des Visualisierungsprozesses, müssen die Eulerwinkel des einen Sensors aus den an den Computer übermittelten Sensordaten berechnet werden, um die Echtzeit-Visualisierung dieses Sensor zu gewährleisten. Als Eulerwinkel bezeichnet man drei Winkel, die die Lage eines festen Objektes im dreidimensionalen Raum beschreiben können. Die Lage eines Objektes im Raum wird erzeugt, indem das Objekt aus seiner Ursprungslage heraus nacheinander um die drei Eulerwinkel um Koordinatenachsen gedreht wird [11]. Dabei spielt das Prinzip von “Roll-Pitch-Yaw” eine wichtige Rolle. Die Roll-/Pitch-/Yaw-Winkel sind spezielle Eulerwinkel, mit der man insbesondere die Ausrichtung von Fahrzeugen beschreiben kann [12].

Berechnung der Neigungswinkel aus den Beschleunigungsmesser

Mithilfe des Beschleunigungssensors des BNO0055, der die Beschleunigungen in x-, y- und z-Richtung misst, ist es möglich, die Neigungswinkel Pitch und Roll zu berechnen. Mit einfacher Trigonometrie lässt sich z.B. der Pitch-Winkel folgendermaßen berechnen [13]:

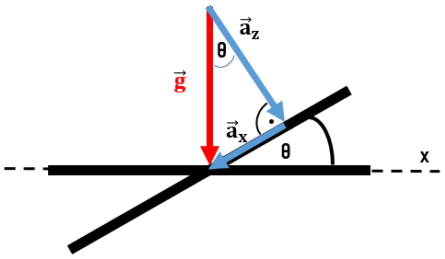

Abb. 8 Neigungswinkel

$$\theta = \arctan\left(\frac{\vec{a}_x}{\vec{a}_z}\right)$$

θ steht für den Pitch-Winkel,
$\mathbf{a_x}$ für Beschleunigungsvektor in x-Richtung
$\mathbf{a_z}$ für Beschleunigungsvektor in z-Richtung

Der Nachteil dieser Herangehensweise ist, dass die Berechnung anfällig gegenüber kurze, schnelle Bewegungen des Sensors (Vibrationen) ist, womit die Berechnung fehlerhaft wird.

Berechnung der Neigungswinkel aus den Gyroskop

Der Gyroskop des BNO055 misst die Rotations-/Winkelgeschwindigkeit des Sensors an den jeweiligen x-,y- und z-Achse. Ebenso wie beim Beschleunigungssensor kann man anhand der gemessenen Gyroskopwerte die Neigungswinkel Pitch und Roll des Sensors folgendermaßen berechnen[14]:

$$\theta_G = \theta_{old} + \omega_y \cdot dt$$

$\boldsymbol{\theta_G}$ steht für den Pitch-Winkel,
$\boldsymbol{\theta_{old}}$ für den alten Pitch-Winkel der letzten Berechnung
$\boldsymbol{\omega_y}$ für die gemessene Winkelgeschwindigkeit um die y-Achse,
dt für die verstrichene Zeit zwischen zwei nacheinander folgender Messungen

Jedoch hat diese Berechnung das Problem, dass langfristig ein Drift der Gyroskopswerte entsteht. Drift bezeichnet das langsame Auswandern der Drehachse des Gyroskops aus seiner ursprünglichen Lage [15].

Complementary Filter

Die oben genannten Nachteile aus der einzelnen Berechnung des Neigungswinkel aus Gyroskop oder Beschleunigungssensor erfordern, dass diese Berechnung stabil gegen Vibrationen sein muss und langfristig nicht driften darf. Dies kann durch die Fusion der Sensoren (Gyroskop und Beschleunigungsmesser) erreicht werden:

$$\theta = (\theta_{old} + \omega_y) \cdot 0.95 + (\theta_{Ax}) \cdot 0.05$$

θ steht für den Pitch-Winkel des gesamten System,
$\boldsymbol{\theta_{old}}$ für den alten Pitch-Winkel der letzten Berechnung,

$\boldsymbol{\theta_{Ax}}$ für den berechneten Pitch-Winkel aus den Beschleunigungssensor
$\boldsymbol{\omega_y}$ für die gemessene Winkelgeschwindigkeit um die y-Achse,

Diese Art der Berechnung der Winkel wird "**Complementary Filter**" bezeichnet [16]. Im Prinzip sagt es hier folgendes aus: Bei ruckartigen Bewegungen des Systems wird der Wert des Gyroskops stärker vertraut, da dieser stabil gegen derartige Bewegungen ist(deshalb der Faktor 0.95). Hingegen wird bei Stillstand des Systems die Werte des Beschleunigungsmesser stärker vertraut (deshalb Faktor 0.05).

Berechnung Yaw-Winkel

Zur Berechnung des Yaw-Winkel wird im Gegensatz zu Roll und Pitch noch der im BNO055 enthaltene Magnetometer miteinbezogen. Prinzipiell kann der Yaw-Winkel nicht vom Beschleunigungssensor gemessen werden, da die bisherige Berechnung mit dem Beschleunigungssensor auf die gemessene Gravitationskraft beruht und die Yaw-Achse senkrecht zur Erdgravitation steht [17]. Stattdessen werden nun die vorher ausgerechneten Roll- und Pitch-Winkel zusammen mit den Werten des Magnetometer zur Berechnung des Yaw-Winkel verwendet. Die Formel lautet wie folgt[18]:

$$\psi = \arctan\left(\frac{magx \cdot cos(pitch) - magy \cdot sin(roll) \cdot sin(pitch) + magz \cdot cos(roll) \cdot sin(pitch)}{magy \cdot cos(roll) + magz \cdot sin(roll)}\right)$$

ψ steht für den Yaw-Winkel,
magx, magy, magz für die gemessene Erdmagnetfeldstärke in x-,y- und z-Richtung,
roll für den vorher berechneten Roll-Winkel aus Complementary-Filter,
pitch für den vorher berechneten Pitch-Winkel aus Complementary-Filter

3D-Visualisierung durch Quaternionen

Die Visualisierung des zweiten Sensor erfolgt über die sogenannten Quaternionen, dessen Werte selbst vom BNO055 bereitgestellt und direkt an den PC übermittelt wird. Quaternionen, bestehend aus vier Komponenten, sind grob gesagt eine Zahlenbereichserweiterung der reellen Zahlen, mit der man Rotationen im 3-dimensionalen Raum elegant darstellen kann [19].

Transformation Quaternionen zu Eulerwinkel

Um jedoch in VPython Quaternionen darzustellen müssen diese ebenfalls in Quaternionen umgewandelt werden. Dazu gibt es verschiedene Umwandlungsformeln [20], die entsprechend in das Programm implementiert wurden.

Implementation in Python

Die Berechnung der Eulerwinkel aus den Sensorwerten nach den oben genannten Lösungswegen und die Transformation von Quaternionen in Eulerwinkel wurden in der Datei *computation.py* jeweils als Funktionen (*computeRollAngle(), computePitchAngle(), computeYawAngle(), transformQuatEuler())* implementiert. Diese werden von der *visualization.py* aufgerufen, sobald Sensordaten an den PC über einen seriellen Port übermittelt wurden.

Darstellung von Rotationen in VPython

Die Darstellung von Rotationen des dreidimensionalen Bild des Sensors in VPython wird durch die Rotation des körperfesten Koordinatensystem des Sensors realisiert, d.h entsprechend der ermittelten Eulerwinkel wird das körperfeste Koordinatensystem gedreht, womit man schließlich das dreidimensionale Bild des Sensors rotieren kann.

Für die Berechnung der Koordinatenachsen/-vektoren gibt es ebenfalls Formeln [21], die auf einfacher Trigonometrie basieren. Ein Problem, das aber dabei auftaucht, ist die Darstellung der Roll-Bewegung des Sensors. Die Erzeugung des körperfesten Koordinatensystem basiert auf die Berechnung eines Referenzvektor, aus der dann die zwei restlichen Achsen mithilfe von Kreuzprodukten berechnet werden kann. Da aber bei einer rollenden Bewegung des Objekts sich die Richtung des Referenzvektor nicht ändert, wird die Berechung der beiden anderen Achsen im Bezug auf die Roll-Bewegung fehlerhaft. Dieses Problem kann durch die **Rodrigues-Rotations-Formel** [22] gelöst werden, mit der man die Ausrichtung der restlichen Achsen des körperfesten Koordinatensystems korrigieren kann.

Ablauf der 3D-Visualisierung

Insgesamt besteht also die 3D-Visualisierung des Sensors über VPython zum einen aus dem Einlesen der Daten, dem Berechnen der Eulerwinkel nach dem erwähnten Wegen, dem Berechnen des körperfesten Koordinatensystem und dem Ausrichten des 3D-Objekts entsprechend des körperfesten Koordinatensystem. Bei der Echtzeit-Visualisierung geschieht das innerhalb einer Endlosschleife. Das nachfolgenden Sequenzdiagramm verdeutlicht noch mal den Ablauf.

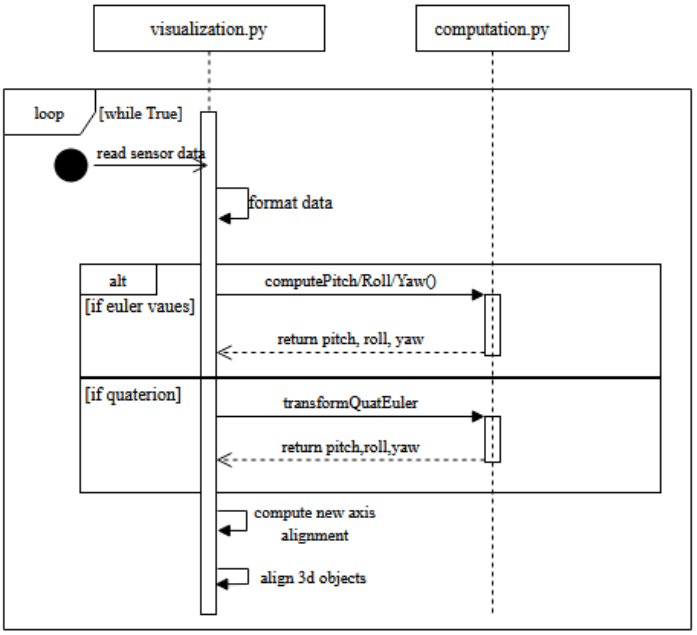

Abb. 9 Sequenzdiagramm der 3D-Visualisierung

Unterschied 3D-Visualisierung Eulerwinkel und Quaternionen

Startet man die *visualization.py*, erkennt man, dass die Visualisierung mit den selbst errechneten Eulerwinkel weitaus "zittriger"/"empfindlicher" ist als die Visualisierung mit den Quaternionen. Dies liegt unter anderem an den bereits oben genannten Störfaktoren (z.B. Drift der Gyroskopwerte oder Vibrationen). Ebenso können auch metallische Gegenstände in der Nähe des Sensors die Werte des Magnetometers beeinflussen. Insgesamt decken aber beide Visualisierungen ziemlich genau die echten Bewegungen des Sensors.

4.5 2D-Diagramme

Eine weitere Möglichkeit den qualitativen Unterschied zwischen den selbst errechneten Eulerwinkel und den Quaternionen aufzuzeigen, ist die Visualisierung der Winkel bezüglich der Zeit in 2D-Diagramme. Anfangs wurde versucht, die Diagramme mithilfe des Python-Moduls "*matplotlib*" zu erstellen. Jedoch fiel beim Testen auf, dass zwar die die Daten richtig geplottet wurden, es aber dennoch eine starke zeitliche Verzögerungen zwischen den echten Bewegungen des Sensors und den dargestellten Datenpunkten im Diagramm gab. Zudem war die Frame-Per-Second-Rate sehr niedrig. Aus Recherche ergab sich das "matplotlib" eher nicht für Echtzeit-Plotting geeignet ist. Stattdessen wurde nun das Python-Modul "*PyQtGraph*" verwendet, um die Echtzeit-2D-Diagramme zu erstellen. Dieses Modul ist wesentlich schneller als "*matplotlib*" und basiert auf das GUI-Modul "*PyQt*" [23], welches, wie bereits in 4.3 erwähnt, ebenfalls für die graphische Benutzeroberfläche dieses Projektes benutzt wird.

Im Nachfolgenden sind die zwei Diagramme für Eulerwinkel und Quaternionen bezüglich der Zeit zu sehen, wobei die gelbe Linie den Yaw-Winkel, die grüne Linie den Roll-Winkel und die rote Linie den Pitch-Winkel darstellen. Die Diagramme wurden geschaffen, indem Listen für die jeweiligen Winkel und Zeitstempel erstellt und neue Werte an das Ende der Liste hinzugefügt wurden. Diese Listen kann man anhand der PyQtGraph-Funktionalitäten leicht plotten. Überschreitet die Liste eine bestimmte Größe, so wird das erste Element der Liste entfernt. Somit werden die Listen nicht “unendlich” größer und eine scheinbare Animation der Daten findet statt.

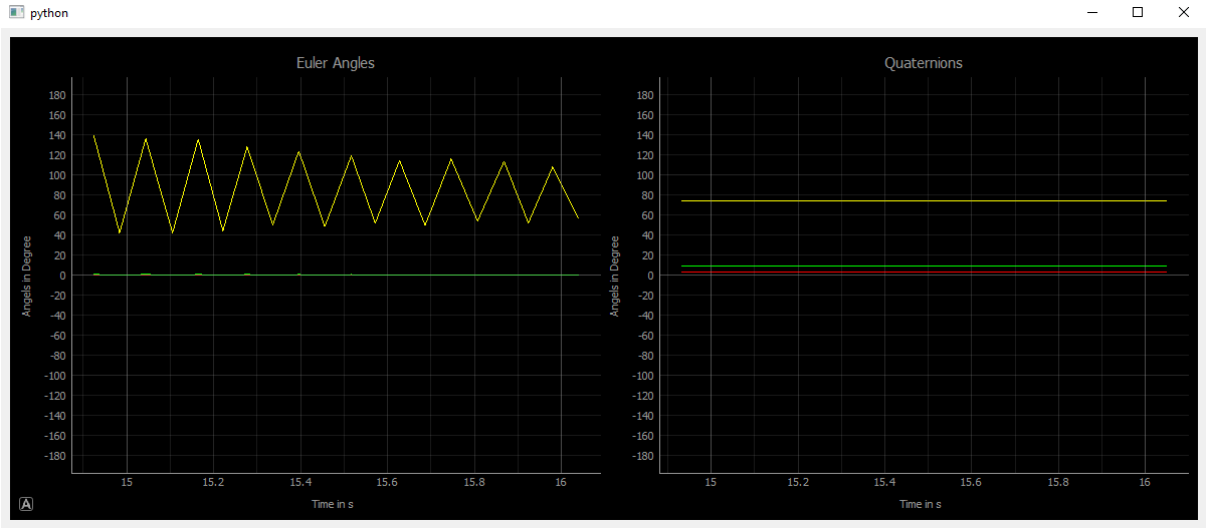

Abb. 10 links selbst errechnete Roll-, Pitch-, Yaw bzgl. Zeit;
rechts Roll-, Pitch-, Yaw aus Quaternionen bzgl. Zeit

4.5 Datenbank

Die Vorgabe des Projekts, "die Werte der Sensoren mit einem Zeitstempel zu speichern" wird mit Hilfe einer Datenbank realisiert. Benutzt wird die SQLite Bibliothek, welche standardmäßig in Python implementiert ist und keine zusätzliche Installation benötigt. Wie der Name der Bibliothek bereits suggeriert, ist SQLite für kleine Projekte wie dieses sehr gut geeignet. SQLite verfügt über alle nötigen Funktionen, um die Daten aus einem Anwendungsfall zu speichern. Des Weiteren läuft SQLite komplett lokal und benötigt keinen Server.

4.5.1 Aufbau der Datenbank

Wie eben erwähnt, erfüllt die Datenbank des Projektes den Zweck, die empfangenen Sensordaten zu speichern, demnach wurde für jeden relevanten Wert, eine Tabelle angelegt. Des Weiteren ist die Datenbank für die Verwaltung der Nutzernamen,

sowie deren Spielstände verantwortlich, deshalb gibt es für diesen Zweck eine zusätzliche Tabelle. Den Aufbau der Tabellen und damit die Datenbankstruktur lässt sich auf der Abbildung erkennen. Für jeden relevanten Sensorwert gibt es einen Platz, der mit einem Zeitstempel versehen ist. Da die einzelnen Tabellen keine Verbindungen untereinander haben, gibt es kein klassisches Entity-Relationship-Modell.

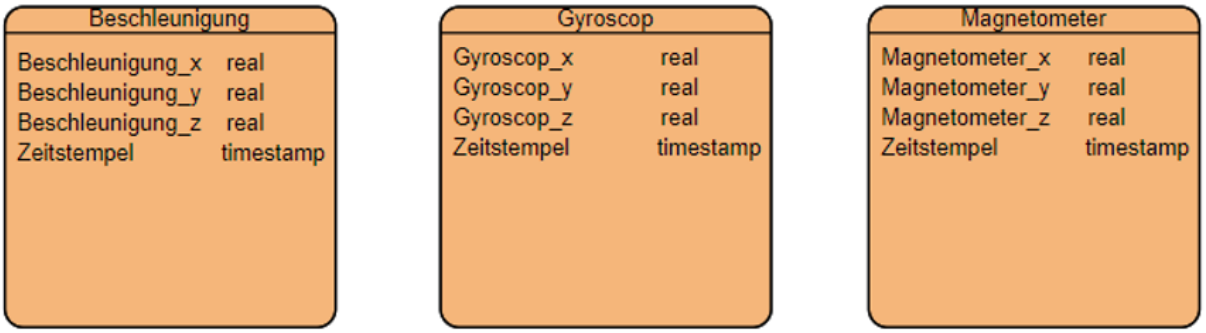

Abb. 11 Aufbau der Datenbankstruktur der Sensorwerte

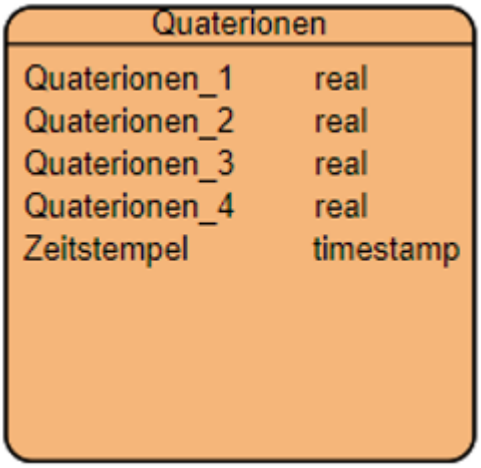

Abb. 12 Quaterionentabelle

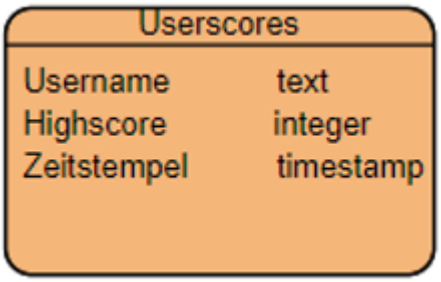

Abb. 13 Benutzertabelle

Die Benutzernamen werden separat in der Tabelle “Userscores” gespeichert, ebenfalls mit einem Zeitstempel versehen, welcher dazu dient, das Datum festzuhalten, an dem der Nutzer erstellt wurde, ebenso wie das Datum zu einer erzielten Punktzahl. Beim Erstellen eines Nutzers, wird dieser in die entsprechende Tabelle geschrieben und dabei die Punktzahl “0” mitgespeichert, diese Punktzahl lässt sich im normalen Spielverlauf nicht erzielen und dient dazu, das Datum herauszufinden, an welchem der Nutzer erstellt wurde.
Von den empfangenen Daten werden nicht alle gespeichert, so zum Beispiel der Batteriestatus, diesen permanent festzuhalten, erschien nicht sinnvoll. Für die Speicherung, Auslesung, beziehungsweise CRUD* Optionen, benutzt SQLite die Standard SQL Query Sprache. Diese Sprache wirkt auf den ersten Blick einschüchternd, da die Syntax sehr ungewöhnlich ist und es treten oft Fehler auf,

die in anderen Programmiersprachen nicht vorkommen. Zum Aneignen der Sprache bietet sich ein Blick in die Dokumentation an, welche ausführlich und leicht verständlich geschrieben ist, bei Fehlern hilft eine Suche im Internet, diese resultiert in der Regel eine passende Lösung. Die Querys werden von Python als String erkannt und für den Einsatz der Sprache für das Projekt, ist es von Vorteil das Python eigene String Formatting zu benutzen. Beim String Formatting wird ein String vorgegeben, in dem sich Platzhalter für Variablen befinden. Für den Ablauf des Projekts war es erforderlich einige diese String zu benutzen, um nicht jeden einzelnen Query hardcoden zu müssen.

```
cursor.execute(f"SELECT * FROM userscores WHERE username = '{name}'")
```

Abb. 14 ein Query, der die Punkte von einem Nutzer queried

*Create, Read, Update, Delete

4.6 Anwendungsfall - Raumschiffspiel

4.6.1 Ursina

Ursina ist eine für Python entwickelte Game Engine, welche “Open Source” ist und kostenlos zur Verfügung steht. Es wird damit geworben, dass diese Game Engine das Gestalten und Entwickeln von Spielen und anderer Software vereinfachen soll. Nach dem Import des Packages stehen dem Benutzer einige Hilfsmittel zur Verfügung, die das Erstellen eines Spiels vereinfachen. Ein entscheidender Punkt hierbei, ist die Klasse Entity. Sie ermöglicht es dem Benutzer verschiedene Objekte zu erstellen und diesen unterschiedliche eigenschaften zu übergeben. Diese Objekte können Positionen, Rotationen und unterschiedliche Größen besitzen. Ebenfalls ist es möglich verschiedene Models, Texturen oder Farben zu definieren. Der Collider ist ebenfalls in der Klasse Entity enthalten und ermöglicht es Objekte mit einem Kollisionsbereich zu versehen. Dadurch ist es möglich, eine Abfrage zu starten, ob ein Objekt mit einem anderen in Berührung kommt.[24] Ein anfängliches Problem war es, diesen Kollisionsbereich zu verstehen und damit die Spielmechanik umzusetzen. Dinge die beachtet werden mussten waren z.B., dass ein Asteroid nicht weiterfliegen darf, sobald er das Raumschiff berührt hatte, da sonst mehrere Kollisionen berechnet werden würden. Die Lösung dafür war, dass der Asteroid an eine neue Stelle gesetzt wird sobald eine Kollision ermittelt wurde. Das Koordinatensystem von Ursina ist für Gestaltung dieses Spiels von großer Bedeutung gewesen.

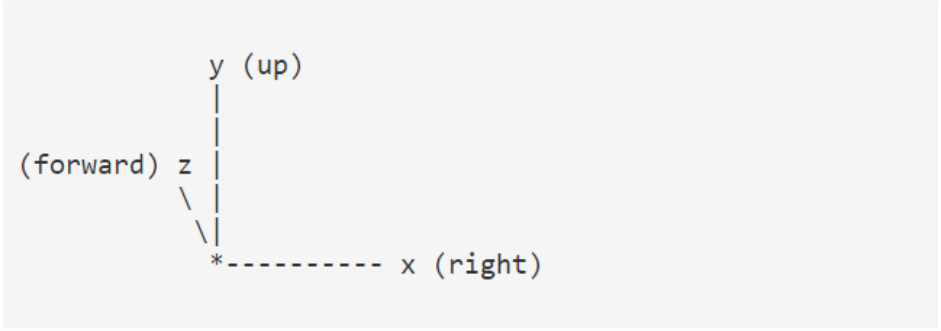

Abb. 15 Koordinatensystem

Bei allen Objekten musste die Position und die Bewegung an das Koordinatensystem angepasst werden. Die Asteroiden bewegen sich auf der z-Achse vom Positiven ins Negative[25]. Eine weitere Herausforderung war die Positionierung der Kamera. Diese sollte einen guten Überblick auf das Spielfeld geben, aber nicht von den Objekten verdeckt werden. Das Problem wurde gelöst, indem das Spielfeld etwas weiter vor die Kamera gesetzt wurde und somit sind keine Asteroiden mehr vor die Kamera geflogen.

Für das Importieren von Blender Dateien ist die Ursina Game Engine ebenfalls gut geeignet, da nach dem ersten Ausführen die .blend Dateien in .obj umgewandelt werden und somit ohne Probleme verwendet werden können. In der Dokumentation wurden keine für diese Applikation passende Weise erläutert, wie das Hintergrundbild darzustellen wäre. Deshalb wurde ein Objekt von der Klasse Entity erzeugt, welches die Textur des gewünschten Bildes hat. Dieses Objekt hat die Form eines Rechtecks bekommen und wird weit hinter dem eigentlichen Spielfeld als Hintergrund angezeigt.

4.6.2 Klassendiagramm

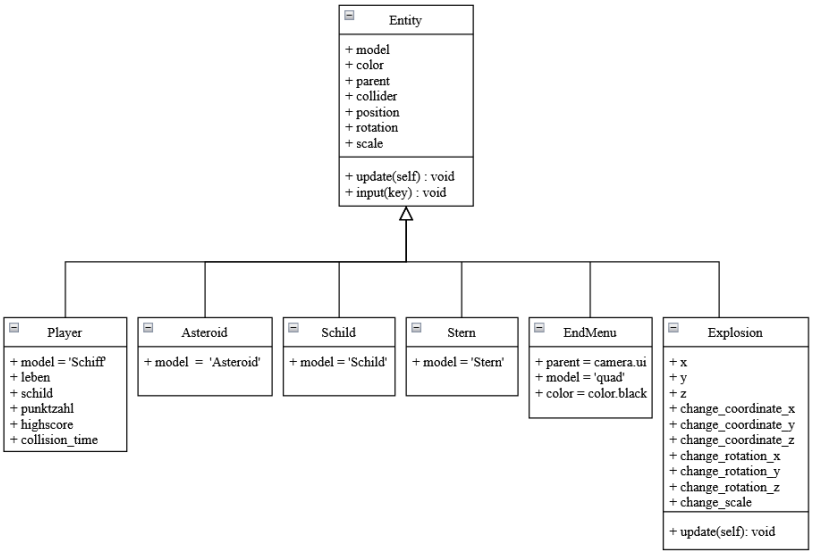

Abb. 16 Klassendiagramm des Spiels

Das Klassendiagramm zeigt die unterschiedlichen Klassen die von der Klasse Entity erben. Dies ermöglicht die Unterscheidung der einzelnen Klassen, um z.B. festzustellen welches Objekt mit dem Raumschiff zusammen stößt und somit unterschiedliche Aktionen auszuführen. Zudem können durch den Einsatz von Klassen Werte wie Punktzahl oder Highscore gespeichert werden, ohne sie jeweils als globale Variable zu deklarieren.

4.6.3 Ablauf Spiel

Wenn das Spiel startet, erscheint ein Fenster, in dem das Spiel dargestellt wird. In der Mitte dieses Fensters ist das Raumschiff zusehen, welches sich mit dem Sensor oder den Tasten WASD steuern lässt. Das Raumschiff fliegt optisch von dem Betrachter weg und somit entgegengesetzt der Asteroiden. Die Asteroiden fliegen auf das Raumschiff zu, bewegen sich also negativ auf der z-Achse. In der rechten oberen Ecke befinden sich sowohl die Punkteanzeige als auch die Anzeige des aktuellen Highscores. Ebenfalls werden dort die aktuellen Frames per Second (FPS) angezeigt. In der rechten unteren Ecke werden Herzen abgebildet, welche das aktuelle Leben des Spieler darstellen. Bei jedem Spiel startet der Spieler mit 3 Herzen und verliert ein Herz, sobald er einen Asteroiden berührt. Mit diesem Wissen kann man nun das Ziel des Spiel erläutern. Das Ziel ist es möglichst viele Punkte zu erreichen und somit einen hohen Highscore aufzustellen. Man bekommt für jeden Asteroiden, den man ausgewichen ist einen Punkt und für einen eingesammelten Stern werden 20 Punkte der aktuellen Punktzahl gutgeschrieben. Ebenfalls erscheinen gelegentlich Schilder, die dem Spieler einen Schutz vor dem nächsten Zusammenprall mit einem Asteroiden bieten. Jedoch kann immer nur ein Schild gleichzeitig eingesammelt werden. Falls der Spieler ein Schild eingesammelt hat wird dieses rechts unten neben den Herzen dargestellt. Sollte der Spieler kein Leben mehr besitzen, erscheint eine Nachricht mit den erreichten Punkten und den Optionen ein neues Spiel zu starten oder das Spiel zu beenden.

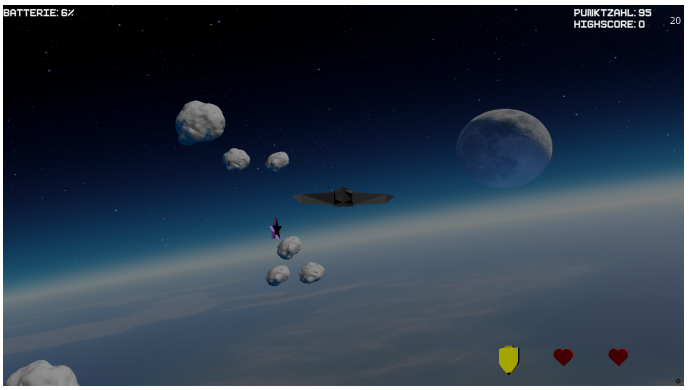

Abb. 17 Spielanwendung

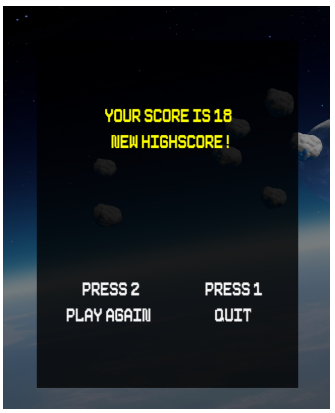

Abb. 18 Endmenü

4.6.4 Blender

Blender ist eine freie 3D-Grafiksoftware. Diese Grafik Software ist mit der General Public Licence (GPL) lizenziert und bietet somit die Möglichkeit die Software frei und ohne Kosten zu nutzen. Blender eignet sich zum Erstellen von Körpern, welche man modellieren, texturieren und animieren kann. Für dieses Projekt wurden alle Objekte, die in dem Spiel vorhanden sind mit Hilfe von Blender designed.
In Blender gibt es sechs verschiedene Modi mit denen man Objekte verändern kann.

Objekt Mode:
Veränderung der Szenen und grobes bearbeiten der Objekte, z.B. die Ausrichtung
Edit Mode:
Bearbeitung der Punkte, Kanten und Flächen eines Objektes. Mit diesem Modus wurden viele der Objekte dieses Projektes gestaltet.
Sculpt Mode:
Dieser Modus ist für die Bearbeitung mit der freien Hand. Es gibt verschiedene Pinsel, welche das Objekt verändern.
Vertex Paint:
Erstellung von Vertex Gruppen.
Weight Paint:
Mit diesem Modus kann man die erstellten Vertex Gruppen einfärben und somit die Wirkung der Farbe festlegen.
Texture Paint:
In diesem Modus können die Objekte texturiert oder mit einem Pinsel angemalt werden.

Die Objekte, die mit Blender designed wurden, konnten daraufhin als .blend Datei in das Python-Projekt importiert werden. Das Problem, welches anfangs auftrat war,

dass die Objekte in einer .blend Datei nur mit dem installierten Programm Blender darzustellen waren. Die Lösung dafür war, dass man in Python die Objekte zu .obj Dateien umwandeln kann, welche auch ohne das Programm Blender darstellbar sind. Im Folgenden sind Bilder der verwendeten Objekte.

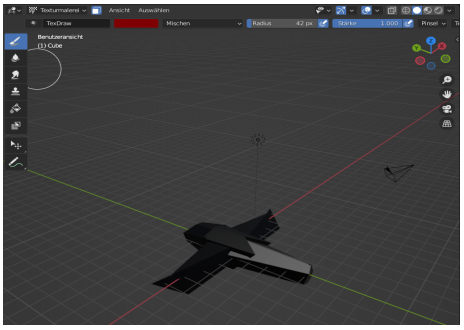

Abb. 19 Raumschiff

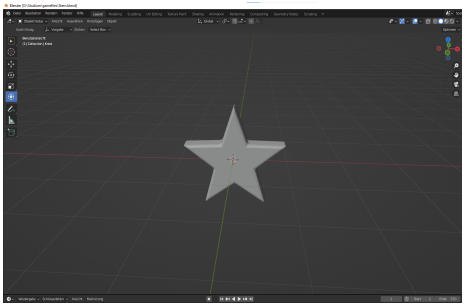

Abb. 20 Stern

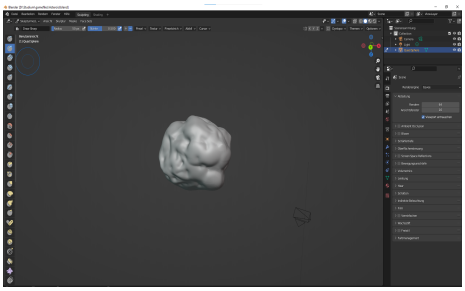

Abb. 21 Asteroid

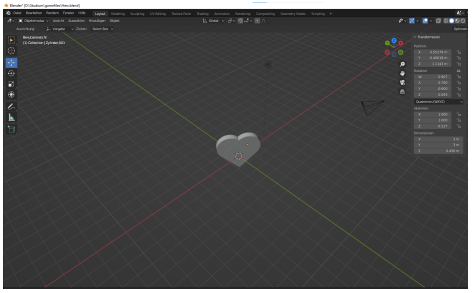

Abb. 22 Herz

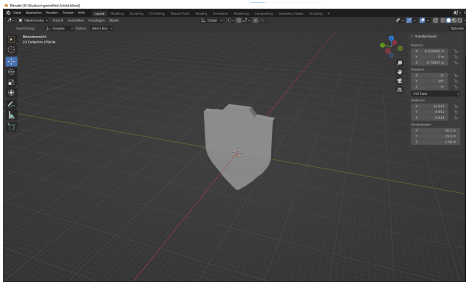

Abb. 23 Schild

4.6.5 Spielmechanik

Animationen:

Die Asteroiden sind so animiert, dass sie zufällig in eine bestimmte Richtung rotieren. Die Stern und Schild Objekte rotieren nur auf der Y-Achse. Wenn ein Asteroid mit dem Raumschiff zusammen stößt, wird ebenfalls eine kleine Animation ausgelöst. Diese besteht darin, dass der Asteroid in mehrere kleine Asteroiden zerspringt. Sollte der Spieler keine Leben mehr haben, erscheint eine

Explosionsanimation. Bei dieser Animation erscheinen sowohl Asteroiden Teile als auch rote Explosions-Kreise. Die Performance wurde nach dem Hinzufügen der unterschiedlichen Animationen immer schlechter, deswegen wurde auf weitere Animationen verzichtet.

Steuerung:
Da nicht alle Gruppenmitglieder einen Sensor zur Verfügung hatten, kann das Raumschiff sowohl über die Tasten WASD gesteuert werden als auch mit Hilfe des Sensors. Es ist folgendes festgelegt: W →oben / A → links / S → unten / D → rechts. Die Daten, die für die Steuerung des Spiels genutzt werden, sind nur die Quaternionen Daten des Sensors. Die Eulerdaten hatten zu viele Störungen in der Darstellung, wie man gut bei der Visualisierung sehen konnte. Mit diesen Daten wäre keine reibungslose Steuerung für das Spiel möglich gewesen. Beim Nach-Vorne-Neigen des Sensors fliegt das Raumschiff nach unten und beim nach hinten neigen fliegt das Raumschiff nach oben. Wenn der Sensor nach links gekippt wird, fliegt das Raumschiff nach links, wenn es nach rechts gekippt wird, fliegt das Raumschiff nach rechts. Es können ebenfalls kombinierte Steuerungen durchgeführt werden indem der Sensor gleichzeitig gekippt und geneigt wird, dabei fliegt das Raumschiff dann diagonal.

Audio:
Die Audiodaten aus dem Projekt sind von der Internetseite “Mixkit.co”[26]. Auf dieser Seite werden verschiedene Audiodateien zur Verfügung gestellt. Für das Projekt wurden unterschiedliche Audiodateien importiert, die bei den passenden Events abgespielt werden. Ebenfalls wird im Hintergrund eine durchgängige Melodie gespielt die eine Weltraum-Atmosphäre erzeugen soll.

5 Fazit

Die ersten Wochen des Projektes waren ein Auf und Ab. Nachdem wir mit unserer Auswahl alle zufrieden waren, kam direkt die erste große Aufgabe, das Pflichtenheft. Die Wenigsten von uns hatten eine Vorstellung, um was es sich dabei handelt. Durch Internetquellen konnten wir uns eine grobe Übersicht über die Anforderungen machen. Dennoch war unsere erste Version des Pflichtenhefts mangelhaft und wir erstellten eine zweite Version. Die folgenden Wochen arbeiteten sich zunächst alle in das Thema ein. Wir hatten die Gruppe in zwei Untergruppen unterteilt damit die Aufgaben besser verteilt werden konnten. Jedoch fiel es uns anfangs schwer, trotz regelmäßiger treffen, die gesammelten Ergebnisse zusammenzuführen. Somit hatten wir nach der Winterpause viele Schritte doppelt abgeschlossen. Nach der Winterpause wurde die Kommunikation immer besser und

die Fortschritte beider Teilgruppen immer greifbarer. Sowohl das Hardware Team, welches sich mit dem Sensor und dem ZigBee-Netzwerk befasste, konnte viele Fortschritte verzeichnen, wie auch das Software Team, welches die ersten Versionen der Datenbank, der Visualisierung und der Applikation darstellen konnte. Während das Hardware Team weiter die erlernten Fähigkeiten aus den Übungsblättern anwandte und verbesserte, musste das Software Team sich auf die Programmierung mit Python fokussieren. Dies führte dazu, dass das erlernte Wissen nicht ganz übereinstimmt, aber dennoch konnten beide Teams eine große Menge an neu gewonnenem Wissen erringen. Nach viel Arbeit konnte das Ziel des Projektes ebenfalls erreicht werden. Die Visualisierung der ausgelesenen Daten findet statt und das Weltraumspiel ist mit dem BNO über das ZigBee-Netzwerk steuerbar.

5.1 Erweiterungsmöglichkeiten der Anwendung

Das Raumschiffspiel als Anwendungsfall erfüllt aktuell alle grundlegende Anforderungen. Sowohl die Spielmechanik als auch die Steuerung über den Sensor funktionieren. Um das Spiel weiter interessant und nicht langweilig zu gestalten, können später eventuell mehrer Levels, verschiedene Flugobjekte oder auch ein Multiplayer-Modus hinzugefügt werden. Allgemein bieten sich durch die Steuerung mit den Sensor vielfältige Möglichkeiten zur Entwicklung von Spielen.

6 Abbildungsverzeichnis

7 Literaturverzeichnis

[1] Atmel, AVR2052: BitCloud SDK Quick Start Guide [14.02.2022]
[2]Datasheet BNO055 Intelligent 9-axis absolut orientation sensor
https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bno055-ds000.pdf (Release Oktober 2021)
[3]Chris. Liechti, *pySerial Documentation Release 3.4,* 5.12.2021
https://buildmedia.readthedocs.org/media/pdf/pyserial/latest/pyserial.pdf
[4] QT Framework
https://www.qt.io/ [05.02.2022]
[5] Python String Formatting
https://realpython.com/python-string-formatting/#3-string-interpolation-f-strings-python-36 [11.02.2022]
[6] PyQt
https://pypi.org/project/PyQt5/ [12.02.2022]
[7] SQLite [10.01.2022]
https://sqlite.org/index.html [11.02.2022]
[8] SQL Language
https://de.wikipedia.org/wiki/SQL [14.02.2022]
[9] Entity Relationship Modell
https://www.datenbank-grundlagen.de/entity-relationship-modell.html
[10] VPython,
https://www.glowscript.org/docs/VPythonDocs/index.html [12.02.2022]
[11] Wikipedia, Eulerwinkel
https://de.wikipedia.org/wiki/Eulersche_Winkel [11.02.2022]
[12] Wikipedia, Roll-Pitch-Yaw
https://de.wikipedia.org/wiki/Roll-Nick-Gier-Winkel [11.02.2022]
[13] Paul McWhorter, Determine Tilt From 9-Axis Accelerometer
https://toptechboy.com/9-axis-imu-lesson-6-determine-tilt-from-3-axis-accelerometer/ [12.02.2022]
[14] Paul McWhorter, Using Gyros For Measuring Rotational Velocity and Angle
https://toptechboy.com/9-axis-imu-lesson-8-using-gyros-for-measuring-rotational-velocity-and-angle/ [12.02.2022]
[15] Wikipedia, Kreiseldrift
https://de.wikipedia.org/wiki/Kreiseldrift [12.02.2022]
[16] W. T. Higgins (1975), “A comparison of complementary and Kalman filtering,” S.1, zugänglich:
https://ieeexplore.ieee.org/document/4101411/citations#citations [12.02.2022]
[17] Beginner’s Guide to IMU

https://students.iitk.ac.in/roboclub/2017/12/21/Beginners-Guide-to-IMU.html [12.02.2022]
[18] Paul McWhorter, Making A Tilt Compensated Compass https://toptechboy.com/9-axis-imu-lesson-10-making-a-tilt-compensated-compass-with-arduino/ [12.02.2022]
[19] Wikipedia, Quaternion https://de.wikipedia.org/wiki/Quaternion [12.02.2022]
[20] Blanco, Jose-Luis (2013). "A tutorial on SE(3) transformation parameterizations and on-manifold optimization". *University of Malaga, Tech. Rep.*, S.14, zugänglich: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.468.5407 [12.02.2022]
[21] Paul McWhorter, VPython Visualiszation Of Pitch and Yaw https://toptechboy.com/9-axis-imu-lesson-19-vpython-visualization-of-pitch-and-yaw/ [12.02.2022]
[22] Wikipedia, Rodriguez-Rotations-Formula https://en.wikipedia.org/wiki/Rodrigues%27_rotation_formula [12.02.2022]
[23] PyQtGraph, https://www.pyqtgraph.org/ [12.02.2022]
[24] Entity Basics https://www.ursinaengine.org/entity_basics.html [12.02.2022]
[25] Coordiante System https://www.ursinaengine.org/coordinate_system.html [12.02.2022]
[26] Audio Data, https://mixkit.co/ [13.02.2022]