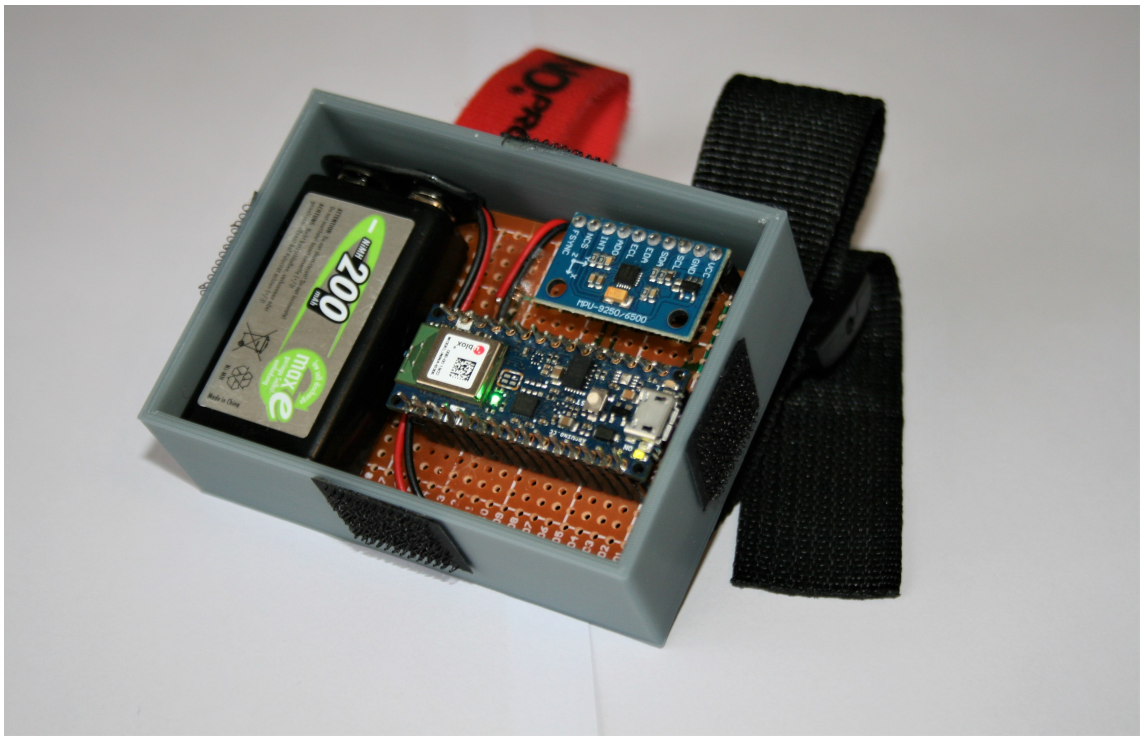


Archer-Tracking

Bewegungstracking für Sportler

Antonio Rehwinkel

12. Januar 2022

Unterstützende Lehrer: Herr Czernohous
m.czernohous@schiller-og.de
Herr Dierle
s.dierle@schiller-og.de

und Professoren: Hochschule1
Hochschule2
Emails?

Inhaltsverzeichnis

1 Kurzfassung

Mithilfe eines Bewegungssensors kann man viele Bewegungen erforschen und vermessen. Mit meinem System sollen mehrere Beschleunigungssensoren dazu eingesetzt werden, Bewegungsabläufe aufzunehmen, miteinander zu vergleichen und zu erkennen.

Die Daten werden über Bluetooth-Low-Energy an ein Handy geschickt, wo Sie sowohl gespeichert als auch ausgewertet werden können. Als Beispiel gilt hier für mich das Bogenschießen, bei dem selbst kleine Bewegungen immer wieder auf gleiche Weise ausgeführt werden müssen. Mit meinem Sensor sollen hier teure Kamerasysteme abgeschafft werden und es so jedem[1] ermöglichen, selbst ohne Bogen oder Trainer bei sich Zuhause den Bewegungsablauf zu trainieren.

1.1 Einleitung

1.1.1 Problemlage

Als Bogenschütze bekommt man schnell mit, auf welch hohem Niveau andere Schützen es schaffen, immer wieder dasselbe zu tun. Dies bezieht sich nicht nur auf den allgemeinen Aufbau, bei dem diese Eigenschaft sogar von Nöten ist, sondern auch auf Fehler im Aufbau.

Da ich selbst schieße und die Möglichkeit hatte in zwei unterschiedlichen Vereinen zu trainieren, ergab sich schnell das Problem, dass meine Schussform drohte schlechter zu werden. Ohne Trainer, der meinen persönlichen Aufbau kannte, konnte ich nicht feststellen, wann ich Fehler wiederholte oder sogar neue einbaute. Eine Lösung musste her, die es mir erlaubte meinen Schussablauf selbst nachzuverfolgen und Unterschiede oder Fehler selbst zu finden.

Bei Profi-Schützen wird schon seit langem Ähnliches getan. Mit Hilfe von mindestens zwei Hochgeschwindigkeitskameras und Punkten am Schützen, ist man in der Lage, den Bewegungsablauf eines Schützen zu analysieren und dreidimensional darzustellen.

Die wohl bekanntesten Aufnahmen werden derzeit vorrangig zum Perfektionieren der mechanischen Ausrüstung und als Lehrmaterial verwendet. Da Profi-Schützen häufig einen nahezu perfekten Aufbau haben, werden die Daten selten zum Korrigieren von Aufbaufehlern eingesetzt.

Zur Auswertung der Daten ist eine weitere Person von Nöten, die speziell darauf geschult wurde, die erzeugten Daten auszuwerten.

Die Hochgeschwindigkeitskameras, welche bei den ersten Aufnahmen von Bogenschützen benutzt wurden, sind in der Lage 6000 bis 8000 Fotos in der Sekunde zu schießen. Kameras dieser Qualität kosten selbst heute noch über 100 000 €.

Der Raum, den diese Messmethode in Beschlag nimmt, ist ebenfalls nicht zu unterschätzen.

[1] Aus Gründen der Lesbarkeit wird das Gendern weggelassen

1.1.2 Anforderungen

Wenn man es nun schafft, den Kostenpunkt zu drücken und weniger Platz zu benötigen, ergeben sich völlig neue Möglichkeiten. Dies würde vor allem den kleinen Vereinen zu Gunsten kommen. Sie wären endlich in der Lage, den Schussaufbau aller Schützen auf kleinste Fehler zu prüfen und eintrainierte Fehler einfach festzustellen.

Die Anforderungen an mein Projekt wären damit, dass eine kostengünstige Alternative geschaffen wird, die wenig Platz benötigt und schnelle Ergebnisse liefert. Dabei müssen diese Ergebnisse genau und für jeden verständlich sein.
Desweiteren darf der Schütze auf keinen Fall gestört werden. Dies wirkte sich bei meiner Idee vor allem auf die Größe, das Gewicht und die Datenübertragung aus.
Entwickelt und gebaut wurde dieses Projekt im Schuljahr 2020/21 von Zuhause aus und wurde im Schuljahr 2021/22 weitergeführt.

1.2 Vorgehensweise

1.2.1 Möglichkeiten

Auf meiner Suche hatte ich die Idee, einen Ultraschallsensor am Schützen zu befestigen. Der Ultraschallsensor hätte am Boden befestigt werden können, um die Höhe der einzelnen Punkte des Schützen bestimmen. Ebenfalls wäre eine Kabelführung für schnellere Datenübertragung und damit ein günstigerer Preis möglich gewesen. Dieses statische System hätte allerdings auf einen sich bewegenden Schützen eingestellt werden müssen. Ein großer Widerspruch, lösbar nur druch weitere Technik, die zu kaufen gewesen wäre. Auch die Verwendung einer günstigeren Kamera wäre möglich gewesen. Dabei vereint man allerdings alle Nachteile, die das Kamera-Tracking hat. Man braucht viel Platz und trotz sehr günstiger Kameras treibt man die Kosten in die Höhe. Meine finale Idee war, einen Beschleunigungssensor und einen BLE-Chip zu kombinieren.

1.2.2 Materialsuche

Es galt nun, zu den genannten Kriterien die passende Hardware zu finden. Um die Bewegungen des Schützen nachzuverfolgen, muss ich wissen, wo die einzelnen wichtigen Punkte des Aufbaus sind. Dies betrifft beide Arme und die Schultern. Da sich die Arme viel bewegen, schien es mir möglich, mithilfe eines günstigen Beschleunigungssensors die Änderungen festzustellen. Bei hoher Genauigkeit könnte dieser vielleicht sogar die Bewegungen der Schultern messen. Um den Schützen nicht zu behindern, ist eine kabellose Verbindung von Vorteil.

2 Hardware

2.1 Arduino Nano 33 BLE

Der Arduino Nano 33 BLE ist ein Prozessor aus dem Hause Arduino der Nano Reihe. Was diesen von der normalen Nano-Reihe unterscheidet ist der BLE-Chip NINA-b3(nRF52840) auf seinem Rücken. Dieser Chip ermöglicht es dem Arduino über Bluetooth 5.0, auch Bluetooth-Low-Energy genannt, mit allen anderen Bluetooth-Geräten ab der Bluetooth Version 4.0 kabellos zu kommunizieren.

Eigenschaft	Daten
Memory	1MB Flash — 256 KB SRAM
Interfaces	I^2C, SPI, ...
Volt	Input: 4,5 - 21 V — Output: 3,3 V

Der Arduino ist aufgrund seines BLE-Chips, der I²C-Verbindungsmöglichkeit und der Output-Volt Zahl von 3,3 Volt der richtige Prozessor für dieses Projekt. Auch die kleinen Maße und das geringe Gewicht bringen ihm nur Pluspunkte ein.

2.2 MPU9250

Für eine genaue Datenlage sorgt in meinem Projekt der Multi-Chip MPU9250. Dieser ist mit einem 3-Achsen Beschleunigungssensor, einem 3-Achsen Gyroskop Sensor und einem 3-Achsen Magnetometer ausgerüstet. Damit bietet er neun Freihtsgrade (9 Degrees of Freedom). Alle Sensoren erhielzen eine Kalibrierung innerhalb der Firma und können einen Selbsttest bei Benutzung vollziehen. Der Sensor benötigt nur 2,4 bis 3,6 Volt während des Betriebs. In der Tabelle sind die Datenpins und die Genauigkeit der Sensoren notiert.

Sensoren	Datenübertragung	Empfindlichkeit
Gyroskop	3 * 16bit ADCs	$\pm 250/\,\text{sec}$, $\pm 500\,\text{sec}$, $\pm 1000/\,\text{sec}$, $\pm 2000/\,\text{sec}$
Beschleunigungssensor	3 * 16bit ADCs	$\pm 2g$, $\pm 4g$, $\pm 8g$, $\pm 16g$
Magnetometer	3 * 16bit ADCs	full-scale range of $\pm 4800\,\text{mT}\,\text{mT}$
Übertragung	I^2C, SPI, ...	

Die Daten werden über den I²C-Bus vom Arduino abgefragt. Die Abtastrate beträgt hierbei mögliche 400kHz. Dabei werden alle Sensoren abgefragt und die Daten versendet. Durch die geringe Größe des Chips (150mm*250mm), dem geringem Energieverbrauch (3,5mA wenn alle Sensoren ausgelesen werden), der hohen Genauigkeit und der Geschwindigkeit der Datenübertragung ist dieser Sensor perfekt für dieses Projekt. Der verbaute DMP (Digital Motion Processor) wird ebenfalls verwendet und filtert die Daten mit einem Low-Pass-Filter.

2.3 Benötigte Leistung

Der Stromverbrauch wurde mit einem Multimeter am Batterieanschluss in verschiedenen Modi gemessen. Die Ergebnisse stehen in der Tabelle:

Modul	An/Aus	Verbrauch(in mA)
BLE	aus	5
BLE	an	10.2
BLE	an und verbunden	13
MPU9250	an	5

Der Stromverbrauch lässt sich so berechnen und erleichtert die korrekte Batterie-Wahl. Die verwendeten Formeln:

P = U / I
Ah / V = Wh
Wh / W = t → (U*I*t) / U*I = t

(2.1)

So hat der Arduino eine Leistungaufnahme von:

0,005 A * 7,4 V = 0,037 W

(2.2)

Die verschiedenen Batterien-Typen stehen in der folgenden Tabelle:

Typ	Laufzeit(in Stunden)	Gewicht	Cut-Off-Spannung	Differenz
9V	40	50g	7.2V	$9 - 7.2 = 2,8$
CR2025	12	2,5g	2V	$3 - 2 = 1$
2 * CR2025	48,6	5g	2V	$6 - 2 = 4$

Um eine Stromversorgung des Arduinos sicher zu stellen, benötigt man 2 Knopfzellen des Typs CR2025. Dennoch hat die Knopfzelle CR2025 nicht nur eine bessere Laufzeit sondern ebenfalls weniger Gewicht, braucht weniger Platz und eine bessere Differenz zwischen Cut-Off-Spannung zu angebotener Spannung.
Gegen das Umrüsten auf die Knopfbatterie spricht einzig der Umweltschutz. Denn im Gegensatz zu 9-Volt-Batterien gibt es keine Akkus für Knopfzellen.

3 Bluetooth-Low-Energy

Mit Bluetooth 5.0 wurde eine neue Übertragunsweise zu Bluetooth hinzugefügt. Diese nennt sich Bluetooth-Low-Energy und zeichnet sich durch einen geringen Stromverbrauch und dennoch einem höherem Datendurchsatz aus.

Bluetooth sendet Daten in Paketen. Hierbei ist bei Bluetooth-Low-Energy (zukünftig BLE) der Sender als Server ausgewiesen und der Empfänger als Client.

Die Server bieten *Services* an die mit *Characteristics* befüllt sind. So bietet mein Arduino den Service *MPU9250* an mit dem Characteristis *Accl*, *Gyro* und *Mag*.
Der Nachteil dieser Verteilung der einzelnen Daten besteht hierbei in der Zeit die für die Abfrage gebraucht wird. Jede *Characteristic* muss einzeln abgefragt werden, hierbei kann ein Großteil

des Datensatzes des MPU9250 verloren gehen.

Laut Dokumentation beträgt der maximale Datensatz von BLE 244 Bytes pro Paket bei aktiviertem DLE. Diese Funktion ließ ich ausgeschaltet, wodurch ich maximal 27 Bytes pro Paket versenden kann. Dieses Problem erklärt ebenfalls weshalb die Sensor-Daten auf verschiedene *Characteristics* aufgeteilt werden. Alle Daten passen nicht in ein einzelnes zu versendendes Paket.

3.1 Datengröse

Die Daten werden als String versendet, diese werden von Arduino mit einer Null terminiert.

Die Größer der Sensordaten beträgt:
Vorkommastellen (3) + Komma (1) + Dezimalstellen (2) + Terminierung (1) = 7 Char

1 Char entspricht 1 Byte, somit gilt:
*9 Sensoren * 7 Byte = 63 Byte*
63 Byte / 27 Byte = 2,3 Datenpakete pro alle Sensoren

Somit brauche ich für das Senden aller Sensoren mindestens drei *Characteristics*.

3.2 Datendurchsatz per BLE

Das Sendeprotokoll von Bluetooth schreibt vor, das ein Datenpaket von leeren Datenpaketen eingepackt wird, ebenso ist eine kurze Wartezeit vorgeschrieben. Diese beträgt 150 Mikrosekunden und wird abgekürt mit *IFS*. Der Arduino Nano unterstützt *2Mbps* bei der BLE-Übertragung, dies ist also die Datenrate. Des weiteren wird nicht auf eine Antwort des *Clients* gewartet, was die Übertragungsgeschwindigkeit weiter erhöht.

Somit beträgt die optimale Sendezeit pro Datenpaket:
Zeit = Sendedauer[Leer] + IFS + Sendedauer[Voll] + IFS
Sendedauer[Leer] = Leerespacket / Datarate

Für mich heißt das:
*Leerespacket = 2 + 4 + 2 + 3 = 11 Bytes * 8 = 88 bit*

und die Sendezeit für das leere Paket beträgt damit:
Sendedauer[Leer] = 88 bit / 2Mpbs = 44 Mikro-Sekunden

Für ein volles Datenpaket brauche ich:
*2+4+2+4+27+3 = 42 Byte * 8 = 336 bit*
Sendedauer[Voll] = 336 bit / 2Mbps = 168 Mikro-Sekunden

Für ein gesamtes Datenpaket brauche ich somit mindestens:
Zeit = 44 + 150 + 168 + 150 = 512 Mikro-Sekunden

beziehungsweise 0,512 Millisekunden. Die maximal erreichbare Datenübetragungs-Frequenz liegt bei 512Hz.

3.3 Tatsächliche Übetragunsgeschwindigkeit

Die ausgerechnete Datenrate kann in der Praxis kaum erreicht werden, weshalb ein Test zur tatsächlichen Datenrate Pflicht ist. Gemessen wurde die Wartezeit am Handy zwischen zwei Datenpaketen. Diese beträgt 20 Millisekunden, was wie erwartet deutlich weniger ist als die Theorie zulässt. Die Übertragung läuft so auf 50 Hz.

4 Bewegungen, Analyse und bekannte Systeme

4.1 Analyse

Da der Schussablauf eines Bogenschützen viele Stationen mit verschiedenen Bewegungen beinhaltet fällt es häufig sogar den Trainern schwer, zwischen einem technisch guten oder schlechtem Schuss zu unterscheiden.
Somit galt es einen Punkt zu finden bei dem Fehler auffällig sind, so dass ich die Daten auch in der Praxis nachvollziehen kann.

Der MPU9250 bietet 9 Freiheitsgrade (Degrees Of Freedom - DOF). Aus welchen diese bestehen, wird im Kapitel zum MPU9250 erläutert. Diese Freiheitsgrade lassen es zu, die Orientierung des Sensors und damit die Orientierung des Körpers an der er befestigt ist, genau zu messen. Sogar die Distanz die der Sensor sich bewegt ist in der Theorie messbar.

So ist eine Darstellung als 3D-Modell möglich, an der der Schütze seine Fehler sieht und Unterschiede zu vorangegangen Schüssen hervorgehoben werden.
Denkbar wäre eine Klassifizierung der einzelnen Schüsse, um, wie in einem Videospiel, die Genauigkeit der Wiederholung in Prozent anzugeben. Interessant wird es, sobald mehrere Schützen Datensätze ihrer Schüsse vergleichen. Unterschiede werden ersichtlich, genau wie Gemeinsamkeiten.

4.2 3D-Darstellung (Theorie)

Die benötigten Daten für ein 3D-Modell variiern je nach gewollter Genauigkeit der Darstellung. Mithilfe von Euler-Winkeln ist es möglich, aus Beschleunigungsdaten die Orientierung des Sensors zu berechnen. Somit müsste man nur noch einen Datensatz übertragen. Allerdings verliert die Darstellung so einen Freiheitsgrad durch den sogennanten *Gimbal-Lock*. Die Euler Winkel drehen das Objekt um gedachte Achsen herum, wenn dabei eine Achse um 180° gedreht wird würde es zu einer zweimaligen Drehung um die selbe Achse kommen. Ebenfalls gestalltet sich die Berechnung der Gier-Achse schwer, da hier keine Veränderung in eine der gemessenen Achsen für Beschleunigung erkennbar ist.
Um dieses Problem zu lösen, muss man das Gyroskop mit einbeziehen, dieses kann die Beschleunigung in Winkeln bemessen. Damit könnte man die Orientierung eines Objekt bestimmen, allerdings ohne Bezug zur Orientierung in der echten Welt. Man wüsste nicht wie das Objekt steht, man weiß nur, dass es sich um einen bestimmten Winkel in diese oder die andere Richtung gedreht hat.
Deshalb muss man die Daten des Beschleunigungsensors und des Gyroskops fusionieren. So kann man die Drehung in jeder Richtung erfassen. Der *Gimbal-Lock* bleibt jedoch bestehen.

Dieses Problem kann man mithilfe von Quaternionen als Orientierung lösen. Da ich jedoch keine Winkel von 180° erwarte sind die Euler-Winkel meine Problemlösung.

4.3 Bewegung

Eine interessante Bewegung stellt vor allem der Zugarm des Schützen dar. Der Auszug verläuft nahezu linear, der häufigere Fehler an dieser Stelle versteckt sich in der Höhe des Zugarms. Um diese zu messen, muss man die Erdanziehungskraft der Z-Achse herausrechnen.
Möglich ist ebenso, die Neigung des Armes über die Euler-Winkel zu berechnen. Sollte der Arm zu hoch sein, wird sich die Ausrichtung des Sensors stark verändern.

Um den Schützen möglichst genau zu tracken, sind mehrere Sensoren an eine, Schützen denkbar und wünschenswert. So könnten Fehler der einzelnen MPUs herausgerechnet werden. Auch die Klassifizierung der Schüsse liefe so auf einem höherem Standard.

5 Software

Die Anzeige der Daten erfolgt am Handy, hier habe ich genug Rechenpower nicht nur Nachkommastellen genau zu berechnen, sondern auch um visuelle Darstellungen erzeugen. Die Daten werden hierfür über das bereits erklärte BLE an das Handy gesendet. Somit brauchte ich sowohl eine Android-App als auch ein Programm für den Arduino.

5.1 Datenübertragung

Die Daten werden, wie in Kapitel zur BLE-Datenübetragung beschrieben, in maximal drei verschiedenen *Charakteristiken* gesendet und empfangen.
Eine Instanz muss hierbei sicherstellen, dass Daten nicht doppelt gesendet oder empfangen werden.
Dies wird beim Arduino durch die Abfrage einer bibliothekseigenen Funktion sichergestellt, die erst auslöst, wenn neue Daten des MPU9250 erzeugt wurden. Dies bewirkt das Bereitstellen der Daten in den Characteristics.
Sobald diese Daten gesendet wurden, bekommt das Handy ein Signal und liest daraufhin die neuen Daten.

5.2 Arduino Firmware

Das Programm auf dem Arduino Nano 33 BLE stellt zu Beginn eine I^2C Verbindung mit dem MPU9250 her und startet das BLE-Modul.
Sobald ein Gerät sich mit dem Arduino verbindet, beginnt dieser mit der Datenübertragung.
Hierbei wurden die Filter vom Digital-Motion-Processor des MPU9250 schon mit einem Low-Pass-Filter verarbeitet.

5.3 Android-App

Zur Erstellung der Android-App wurde AppInventor[1] und die BLE-Extension verwendet.
Geschrieben wurde das Programm von mir. Ideen und Anregungen wurden in verschiedensten Foren gefunden.

Beim Start der App wird man aufgefordert Bluetooth und GPS anzuschalten, das GPS ist nach Android-Richtlinien zu aktivieren. Danach kann man nach verschiedenen Geräten scannen und sich mit diesen zu verbinden. Erfolgt die Verbindung mit einem falschen Gerät, schließt sich die App. Nach der Verbindung wird sofort die Übertragung gestartet.
Das empfangene Datenpaket muss vor der Verarbeitung in die einzelnen Daten aufgespalten und von String zu Float-Werten, besser Double-Werten, gepaarst werden. Diese werden in ein Array gespeichert welches später die einzelnen aktuellen Werte ausgeben kann.

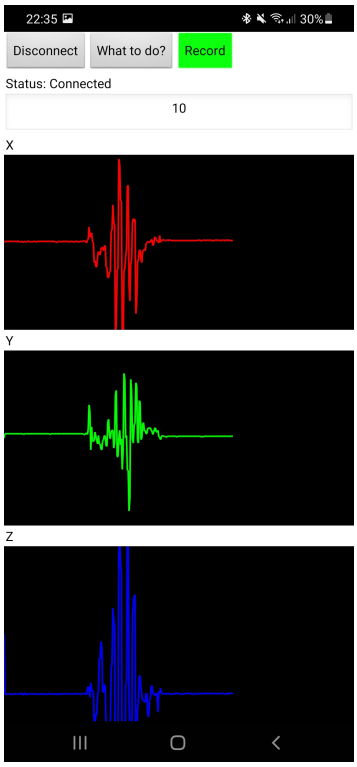
Es gibt insgesamt drei Möglichkeiten die Daten anzeigen zu lassen.

[1] https://appinventor.mit.edu/

5.3.1 Rohdaten

Die gelesenen Daten werden direkt im Textformat auf dem Bildschirm ausgegeben. Der Zeitunterschied zwischen Datenpaketen wird in Millisekunden auf dem Bildschirm angezeigt. Es ist möglich die Daten gleichzeitig aufzuzeichnen.

5.3.2 Graph

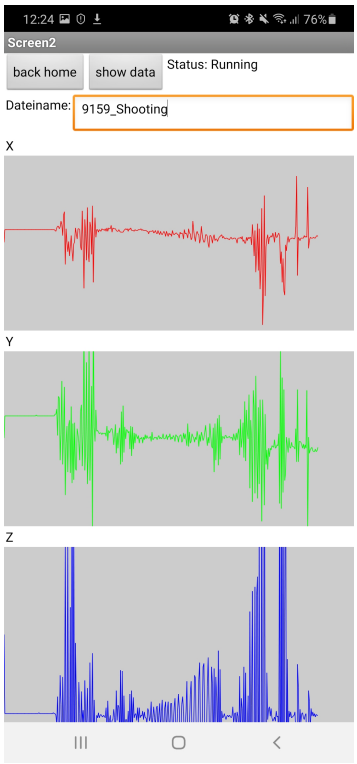


Die Beschleunigungs-Daten werden in einem Graph dargestellt, hierzu werden sie zuerst in ein Array geschrieben. Aus diesem Array erzeugt das Programm in einem vorgegebene Bereich die Datenpunkte, die aufgrund ihrer Masse wie ein Liniendiagramm aussehen.
Neue Daten werden rechts geschrieben, während die alten Daten nach Links aus dem Bildschirm verschwinden.
Der Zeitunterschied zwischen Datenpaketen wird in Millisekunden auf dem Bildschirm angezeigt. Es ist möglich die Daten gleichzeitig aufzuzeichnen.

5.3.3 Animation

Hier wird die Distanz aus den Beschleunigungsdaten berechnet und mithilfe eines Punktes visualisiert. Benutzt werden hierfür die X- und Y-Achse. Die Rechnung ist in Kapitel *5.4.1 Gleichmäßige Formel* beschrieben.

5.3.4 Aufgezeichnete Daten



Die von den anderen Funktionen aufgezeichneten Funktionen können hier ausgegeben werden. Hierzu benötigt der Schütze den Namen der Datei. Die Dateien werden seit kurzem unter Android in einem App-eigenem Ordner gespeichert. Diesen muss der Schütze momentan auslesen, um den zufälligen Namen der neuen Datei zu kennen.
Die Daten werden als Graph dargestellt. Die Beschleunigungsdaten werden außerdem in Rohform über dem Graphen ausgegeben.

5.4 Gleichmäßige und ungleichmäßige Beschleunigung

Um die Distanz aus der gemessenen Beschleunigung zu berechnen, fand ich zwei verschiedene Formeln. Die wohl bekannteste Umrechnung benutzt Integrale, die zweite Formel ist die der gleichmäßigen Beschleunigung.

Die Integration wird von allen mir bekannten Forschungen verwendet. Man muss eine Doppelintegration ausführen um von Beschleunigung auf Distanz zu kommen, hierbei verwandelt sich das Rauschen des Sensors in Drift und so einen exponentiell steigenden Fehler.
Die gleichmäßige Formel kann im Gegensatz zum Integrall, nur positive Beschleunigungen verwerten, hat in den folgenden Tests allerdings deutlich genauere Werte und einen geringeren Fehler bei Stillstand aufgezeigt. So wird in diesem Projekt die gleichmäßige Beschleunigungsformel verwendet.

Als Zeit wird die Frequenz, mit der der Sensor Daten misst, genommen. Hierfür wird die Frequenz in Zeitabschnitte umgerechnet, mit der die Formeln letzendlich arbeiten.

5.4.1 Gleichmäßige Formel

Die angepasste Formel für gleichmäßige Beschleunigung berechnete die Distanz in meinen Versuchen mit einer Genauigkeit von ± 8 cm auf 30cm Teststrecke.
Die physikalisch richtige Formel lautet $s + v * t + a * t^2 * 0.5$, allerdings wurden mit der Formel $s + a * t^2 * 0.5$ bessere Testergebnisse erzielt.
Da die negative Beschleunigung den Wert der Distanz wieder nullierte, wurden zu dieser Formel nur positive Distanzmessungen zugelassen.
Hier der Code-Ausschnitt:

```
if (acc > 0) {
t = (freq / 1000); //hz is not time but frequenzy
```

```
distance = (distance) + (acc * (t * t) * 0.5);
velocity = acc * t;
}
```

Für die Tests wurde dieser Code statt der BLE-Übetragung auf dem Arduino ausgeführt.

5.4.2 Integral

Die Berechnung der Distanz über Integrale ist der Standard in der Wissenschaft. Da etwaige Messfehler des Sensors in Drift verwandelt werden, steigt die Fehlerrate pro Integral enorm.
Die Testergebnisse ergaben einen Fehler von ±15cm auf einer Strecke von 30cm. Ebenso war das Ergebnis sofort in Zentimetern, statt wie erwartet in Metern. Wurde der Sensor zurückbewegt an seinen Startpunkt, sank der Wert jedoch wieder auf Null ab. Somit kann man schließen, dass das Ergebniss nur falsch skaliert ist.
Dieser Fehler wurde noch nicht behoben.
Ein klarer Vorteil dieser Rechnung zeigt sich schon beim Test, die Formel funktioniert auch für negative Beschleunigungen. Der Wert sinkt am Ende der Teststrecke nicht ab, sondern bleibt auf seinem hohen Wert.
Folgend der Code-Ausschnitt der Integral-Rechnung:

```
t = (freq / 1000); // hz to time

velocity = t * ((acc + accOld) / 2) + velocity;
accOld = acc;
distance = t * ((velocity + velocityOld) / 2) + distance;
velocityOld = velocity;
```

6 Tests

6.1 Distanz - Testaufbau

Der Sensor wurde auf einem Breadboard mit einem Arduino Uno verbunden und übertrug die Werte via USB-Kabel an den Seriellen Monitor der Arduino IDE. Um eine mögliche Schieflage auf dem Breadboard abzufedern wurden aus 200 Messungen der Durchschnitt berechnet und von den Werten vor der Verarbeitung subtrahiert. So liegt der Sensor mathematisch absolut flach auf.

6.1.1 Distanzmessungen

Insgesamt wurden 10 Tests pro Formel gemacht, die Ergebnisse können Sie unten einsehen.

6.1.2 Fehler

Die zwei weiteren Graphen zeigen die Fehler bei 10 sekündigem Stillstand. Der exponentielle Drift bei der Integralen Formel ist typisch.

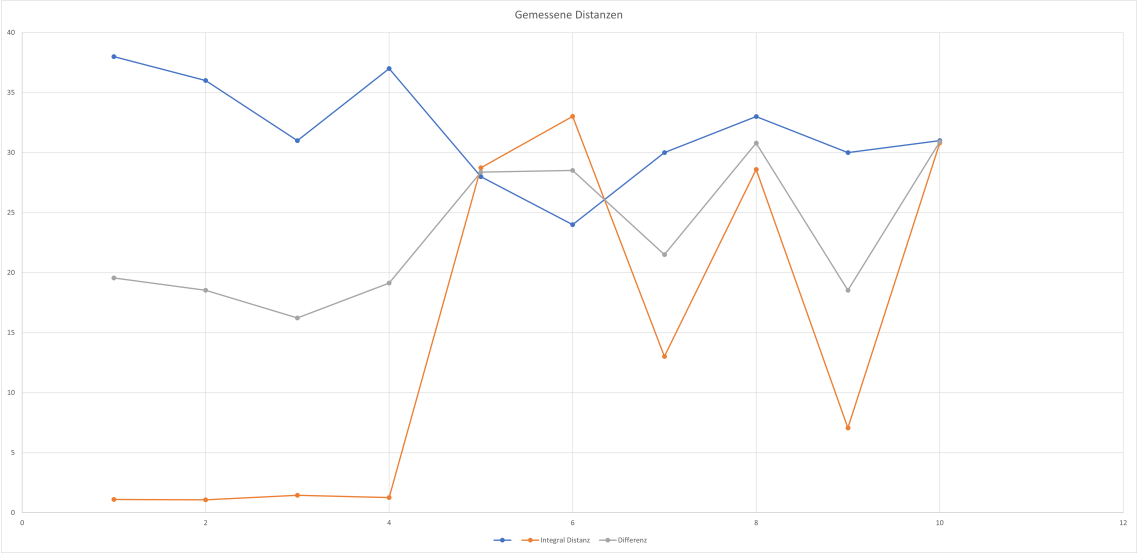


Abbildung 6.1: Distanzen — Alle Tests

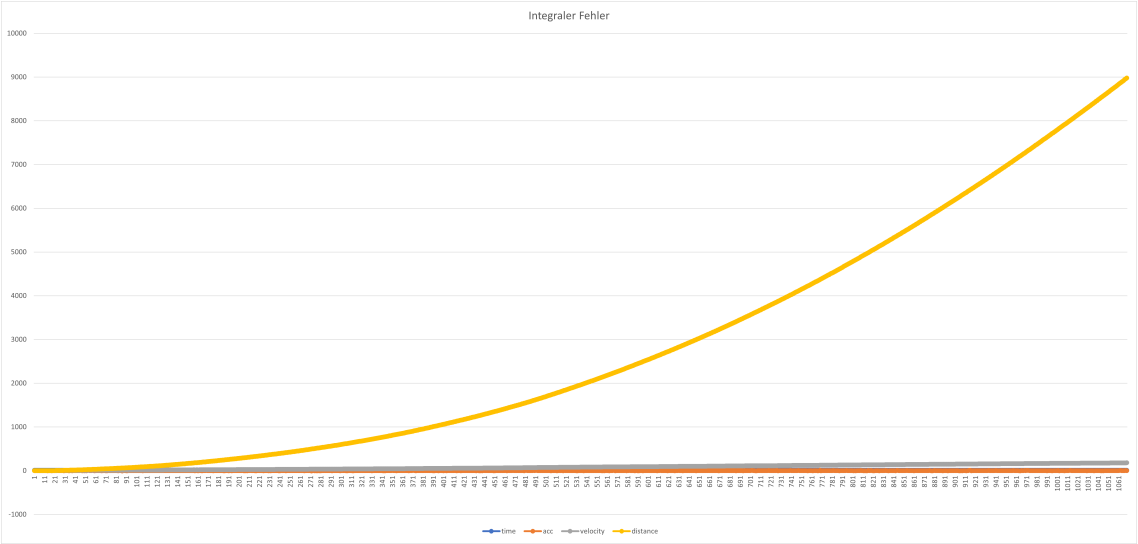


Abbildung 6.2: Integrale Formel — Drift

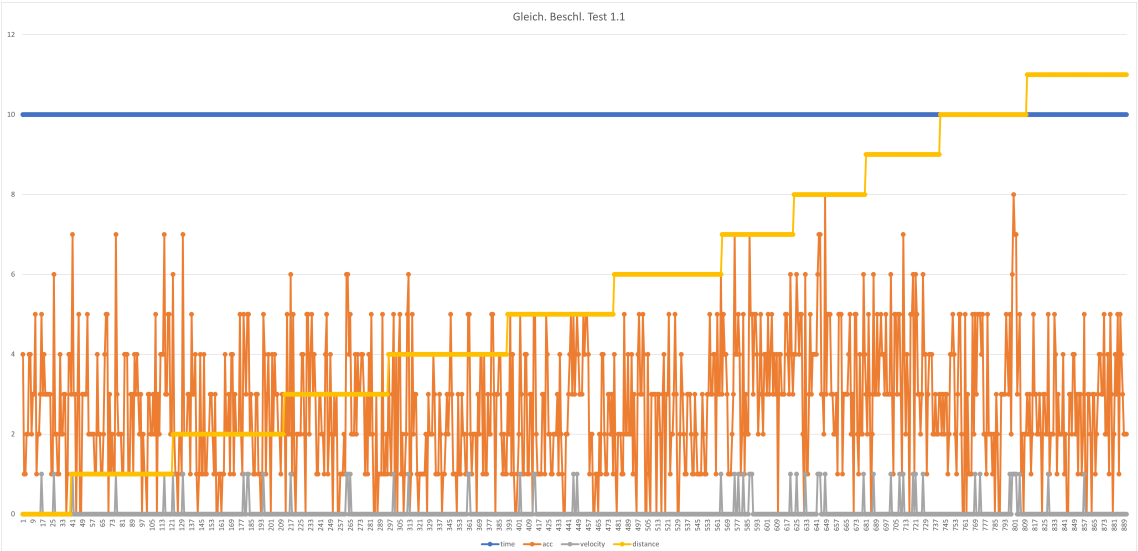


Abbildung 6.3: Gleichförmige Beschl. Formel — Drift

7 Fazit

Das fertige Modell ist 900mm*200mm*750mm groß und wiegt mit den neuen Batterietypen ungefähr 30 Gramm. Die Größe und das Gewicht erfüllen die erste Anforderung. Man kann die Box überall am Schützen befestigen ohne das dieser durch zusätzliches Gewicht stark beeinflusst würde.
Die zweite Anforderung, der geringe Preis, wurde erfüllt, die Kosten belaufen sich auf 25 € pro Sensor. Die letze Anforderung, die leichte Handhabung nähert sich an, es ist mir nun möglich, Distanzen und Winkel in eine bestimmte Richtung zu berechnen und darzustellen. Die Auswertung der Graphen ist nach wie vor möglich.

Die BLE-Übertragung funktioniert gut, das Problem der überlaufenden Characteristics wurde behoben. Somit kann man alle Daten an das Smartphone senden und empfangen, diese Option muss im Code nur aktiviert werden. Je nach Fehlerwerten der einzelnen Sensoren muss dies jedoch nicht getan werden.

Die Darstellungen der Daten funktioniert weiterhin, seit Android 11 müssen die gespeicherten Schüsse manuell in den Gerätespeicher verschoben werden. Android richtete eine Datenschutzmaßnahme ein, nach der Apps nicht mehr in den Gerätespeicher, sondern in einen appeigenen Unterordner schreiben müssen. Das Lesen stellt jedoch kein Problem dar.

Mit den neuen mathematischen Formeln stellt sich nun ein weiteres Problem, die kabellose Übertragung. Je mehr Nachkommastellen übertragen werden, desto genauer erfolgt die Berechnung der Winkel und vor allem der Distanz. Der Multi-Chip liefert 16 Nachkommastellen, was pro Achse eine Characteristic benötigen würde.
Momentan sind die berechneten Ergebnisse genau genug mit nur 2 Nachkommastellen, weshalb ich dieses Problem vorerst nicht weiter bearbeitete.

Die berechneten Winkel und Distanzen haben einen Fehler-Wert, der im Vergleich zu momentan benutzten Hochgeschwindigkeitskameras enorm ist, jedoch hat die angesprochene Kundschaft andere Wünsche. Es ist möglich einen Schützen mit den Rohdaten auf $\pm$10cm genau zu tracken, vielen Schützen wird diese Genauigkeit genügen. Für eine höhere Genauigkeit können zu diesem Zeitpunkt sowohl der Code als auch die Rechnungen verbessert werden. Es ist noch viel Luft nach oben.

Geplant ist die vollständige Darstellung eines Schützen in der Android-App und die Klassifizierung der Schüsse via Machine-Learning. Hierzu müssen die Rechnungen verfeinert werden und eine Datenbasis geschaffen werden.

8 Danksagung