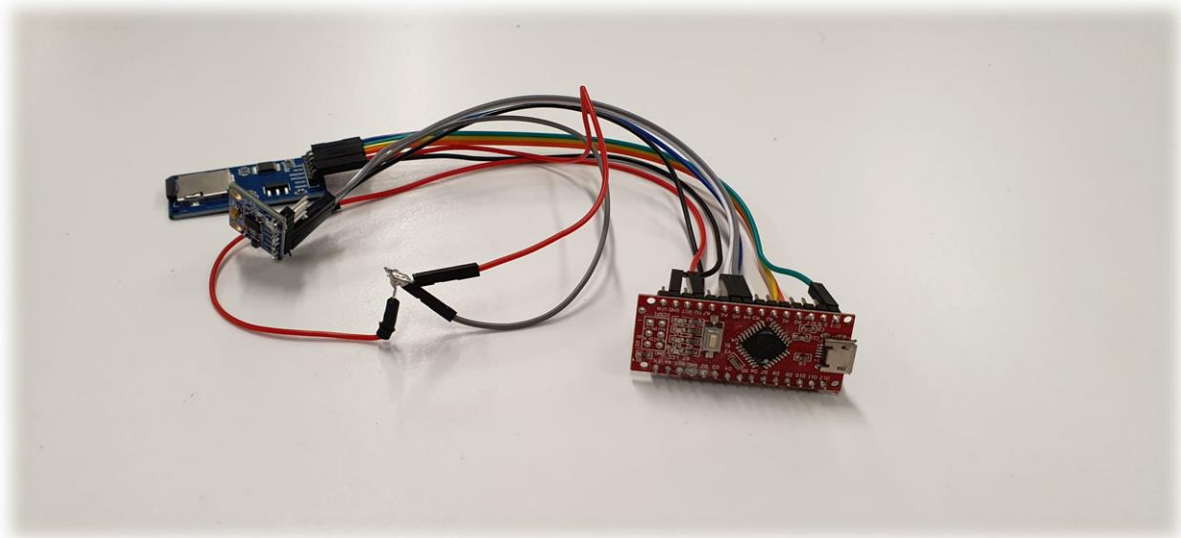


# Projekttitle: Entwicklung von Sensorik zur Analyse von Raketenflugbahnen



Teilnehmerin/Teilnehmer (mit Alter): Schöneberg Philipp (19), Gustke Phil (19), HAIDAR Hassan (19)

Erarbeitungsort: Gymnasium Athenaeum Stade

Projektbetreuer: Carmesin Hans-Otto

Fachgebiet: Technik

Wettbewerbssparte: Jugend forscht

Bundesland: Niedersachsen

Wettbewerbsjahr: 2024

## Inhaltsverzeichnis

Inhaltsverzeichnis .....	2
Fachliche Kurzfassung.....	3
Einleitung .....	3
Vorgehensweise, Materialien und Methoden .....	3
Materialien und Aufbau .....	3
Programmcode Sensoren.....	7
Experimente.....	11
Überprüfung der Materialien .....	11
Rekonstruktion .....	12
Zeitexperimente .....	12
Genauigkeitsexperiment .....	13
Geplantes Raketenexperiment .....	13
Rekonstruktion .....	13
Wahl der Technologie .....	13
Mathematische Umsetzung .....	14
Programmtechnische Umsetzung.....	14
Ergebnisse .....	16
Ergebnisdiskussion .....	17
Fazit und Ausblick .....	17
Quellen- und Literaturverzeichnis .....	18
Unterstützungsleistungen .....	18

## Fachliche Kurzfassung

Bereits seit über 100 Jahren arbeiten Wissenschaftler rund um die Welt an der Entwicklung von Raketen. Diese sollen nicht nur die Menschen zum Mond und Mars bringen, sondern auch langfristig wiederverwendbar werden. Hierzu und zur Analyse von fehlerhaften Starts und Manövern ist eine funktionierende Sensorik zur Sammlung von Daten unabdingbar. In unserem Projekt wollen wir eine funktionsfähige Sensorik entwickeln, diese soll der Rekonstruktion und Analyse von Raketenflugbahnen dienen. Des Weiteren wollen wir langfristig unsere Ergebnisse an einer auf Wasser- und Luftdruck basierenden Flaschenrakete testen und an dieser eine vollständige Nachverfolgung und Simulation der Flugbahn durchführen.

## Einleitung

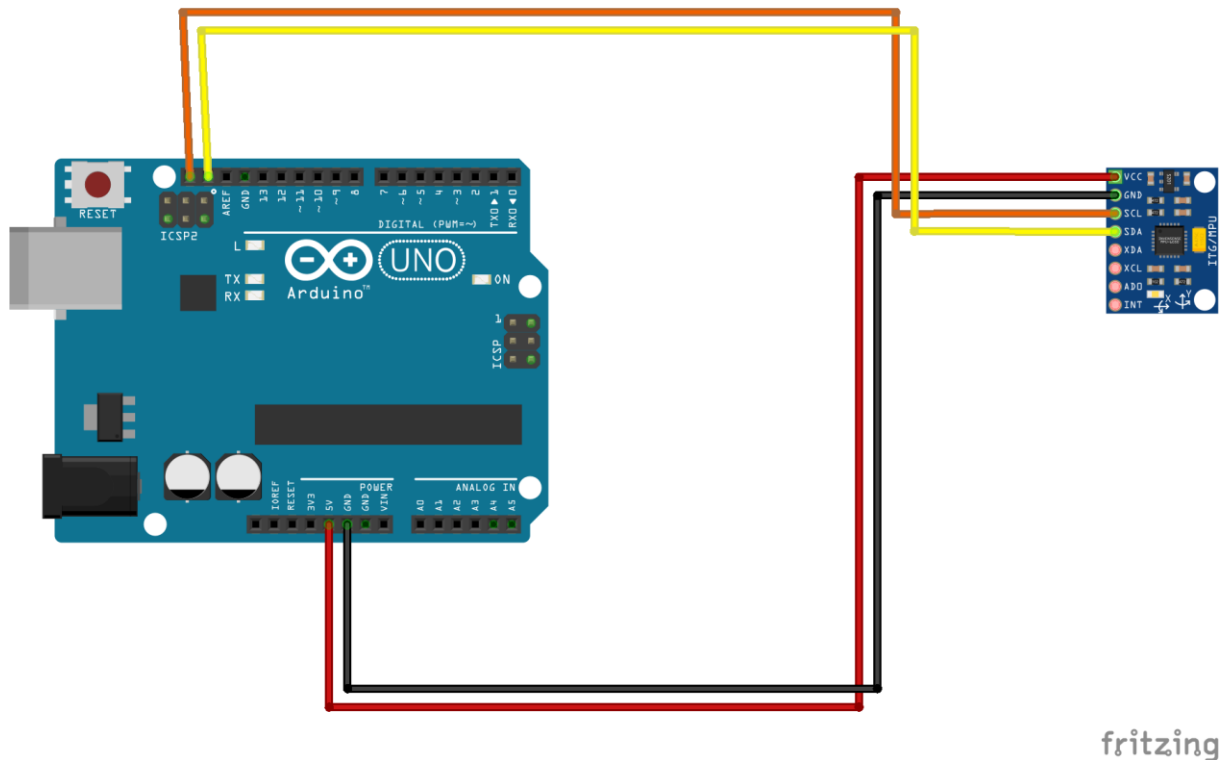
Der Wettlauf zum Mond, die Betonung der eigenen Fähigkeiten und die Vorherrschaft in der Raumfahrt, genau diese Aspekte waren es, die besonders in den USA und der Sowjetunion die Menschen und Gesellschaft Mitte des 20. Jahrhunderts prägten. Hierzu arbeiteten Menschen rund um die Welt an der Entwicklung und Optimierung von Raketen und Landesphären. Diese mussten mehreren Voraussetzungen gerecht werden. So mussten sie zum Beispiel durch geschickte Steuerung der Lebenserhaltungssysteme und Manövrieren der Raketen als Ganzes eine bemannte Mission zum Mond ermöglichen. Grundlage hierfür waren Sensoren, die die betroffenen Systeme mit den nötigen Daten versorgten, auf dessen Basis Anpassungen getroffen werden konnten. Heutzutage ist der Sensorik einer Rakete jedoch noch eine deutlich höhere Bedeutung zuzuschreiben, da neu entwickelte autonome und wiederverwendbare Raketen bei jeder Bewegung ihre Umgebung vollständig analysieren können müssen, um eine sichere Landung zu gewährleisten. An unserer Schule arbeitet die Jugend forscht AG gemeinsam mit der Astronomie AG bereits seit langer Zeit an einer vollständigen Raketenabschussbahn inklusive einer Konstruktion zur Befüllung einer Flasche mit Wasser und Druckluft, einem Sicherheitsturm sowie der Optimierung der Form der Rakete. Dies soll der Verbesserung der Flugbahn als auch der Maximierung der erreichten Höhe dienen. Zur Analyse dieser wurden bisher allerdings immer Kameraaufnahmen analysiert. Diese sind allerdings in ihrem Sichtfeld sowie der Fülle der erhaltenen Informationen stark eingeschränkt. In unserem Projekt wollen wir die Rakete durch Sensorik bezüglich der Beschleunigung als auch Ausrichtung ergänzen. Des Weiteren, planen wir für eine verbesserte Analyse der Raketenflugbahn eine Software zu entwickeln, welche die Bewegungen der Rakete exakt simuliert und somit eine Untersuchung und Optimierung dieser ermöglicht.

## Vorgehensweise, Materialien und Methoden

### Materialien und Aufbau

Grundlegend für die Entwicklung der Sensorik war für uns die Untersuchung der Bewegung der Rakete. Daher überlegten wir zu Beginn, welche Messdaten wir sammeln mussten, um eine Rekonstruktion der Raketenflugbahn durchführen zu können. Da hierfür die Ortsinformation unerlässlich, allerdings auch nicht direkt per Sensor auslesbar ist, entschieden wir uns für die Messung der Beschleunigung. Diese gibt uns sowohl Aufschluss über die Beschleunigung als auch die daraus berechenbare Geschwindigkeit sowie zurückgelegte Strecke der Rakete in Abhängigkeit von der Sensorausrichtung und somit eine Fülle an Informationen. Wichtig als Referenzpunkt der Beschleunigung, ist also zusätzlich die Ausrichtung des Sensors und daraus abzuleiten die, der gesamten Rakete. Die Ausrichtung kann uns auch auf einem auf Unstimmigkeiten in der Form der Rakete hinweisen und bietet somit weitere hilfreiche Informationen. Daher entschieden wir uns schlussendlich für einen Beschleunigungs- sowie einen Gyrosensor. Bei der Recherche nach geeigneten Sensoren stießen wir dann auf den MPU-6050, welcher sowohl als Beschleunigungssensor als auch Gyroskop dient. Das war für unsere geplante Sensorik optimal, da zwei Sensoren auf einer kleinen Platine verbaut waren und somit wenig Platz benötigten und Gewicht hinzufügten, was essenziell ist, um die Rakete so wenig wie möglich zu beeinflussen. Zur Steuerung des

Sensors entschieden wir uns zu Beginn für einen Arduino UNO, da wir bereits Erfahrung mit dessen Programmierung hatten und von der Fülle der möglichen Ergänzungsmodulen überzeugt waren. Zur Stromversorgung unserer Sensorik verwendeten wir eine 9V Blockbatterie. Als nächstes bestellten wir die benötigten Teile und schlossen sie an (siehe Abb. 1).



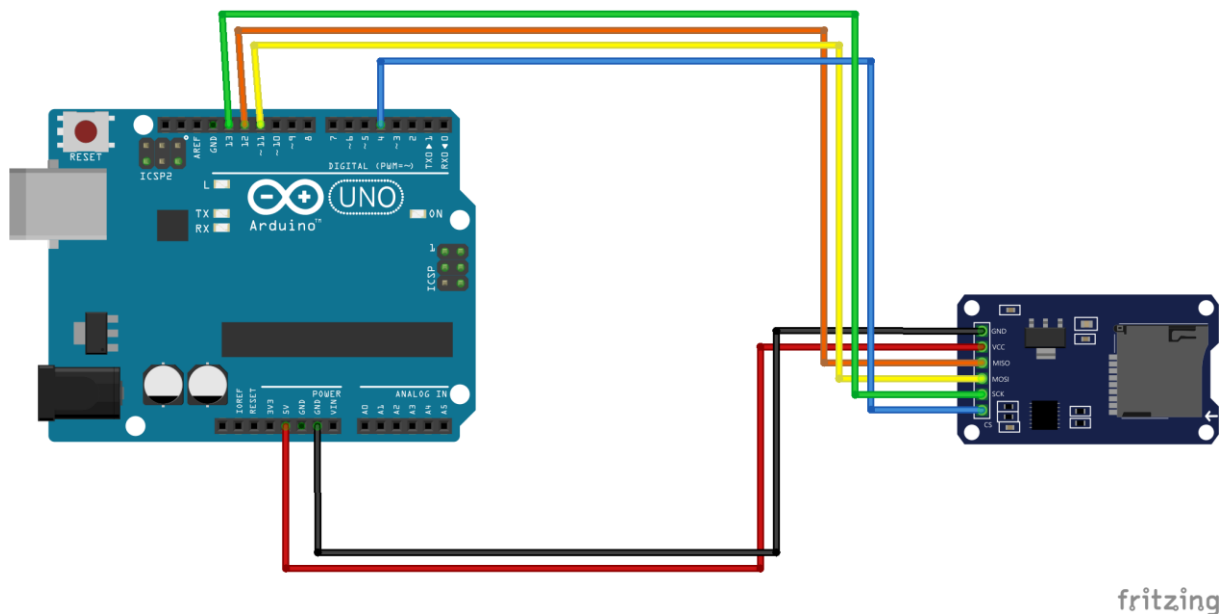
**Abb. 1:** Schaltplan der Verbindung zwischen dem Arduino UNO und dem MPU-6050 Sensor.

Unser nächster Schritt bestand nun in der Speicherung der ausgelesenen Daten. Zu Beginn dachten wir über eine kabellose Verbindung zwischen dem Mikrochip mit angeschlossenem Sensor sowie dem Speichermedium, unserem Computer nach. Schlussendlich entschieden wir uns jedoch dagegen, da es besonders bei der anfänglichen sehr hohen Beschleunigung und der erreichten Höhe zu Blackouts und sonstigen Übertragungsfehlern kommen kann. Deshalb planten wir nun unsere Sensorik, um ein lokales Speichermedium zu ergänzen. Insbesondere entschieden wir uns für ein Mikro-SD-Karten Modul, da Mikro-SD-Karten besonders klein und leicht sind. Des Weiteren ist Ihre Auslesung mittels SD-Karten-Adapter (siehe Abb. 2) sehr effizient und die Verbindung des Moduls mit dem Arduino problemlos möglich.



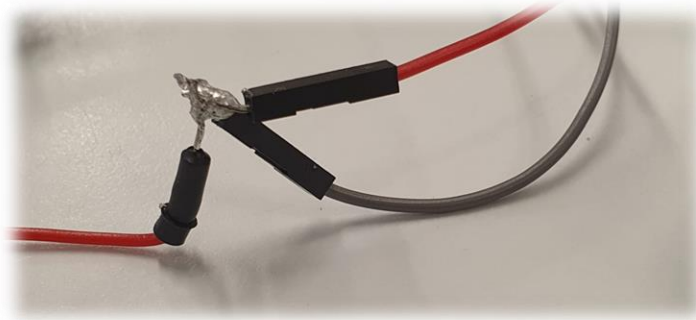
**Abb. 2:** Adapter von Mikro-SD-Karte zu SD-Karte.

Diese nehmen wir nach der Lieferung der Materialien als Nächstes vor (siehe Abb. 3).



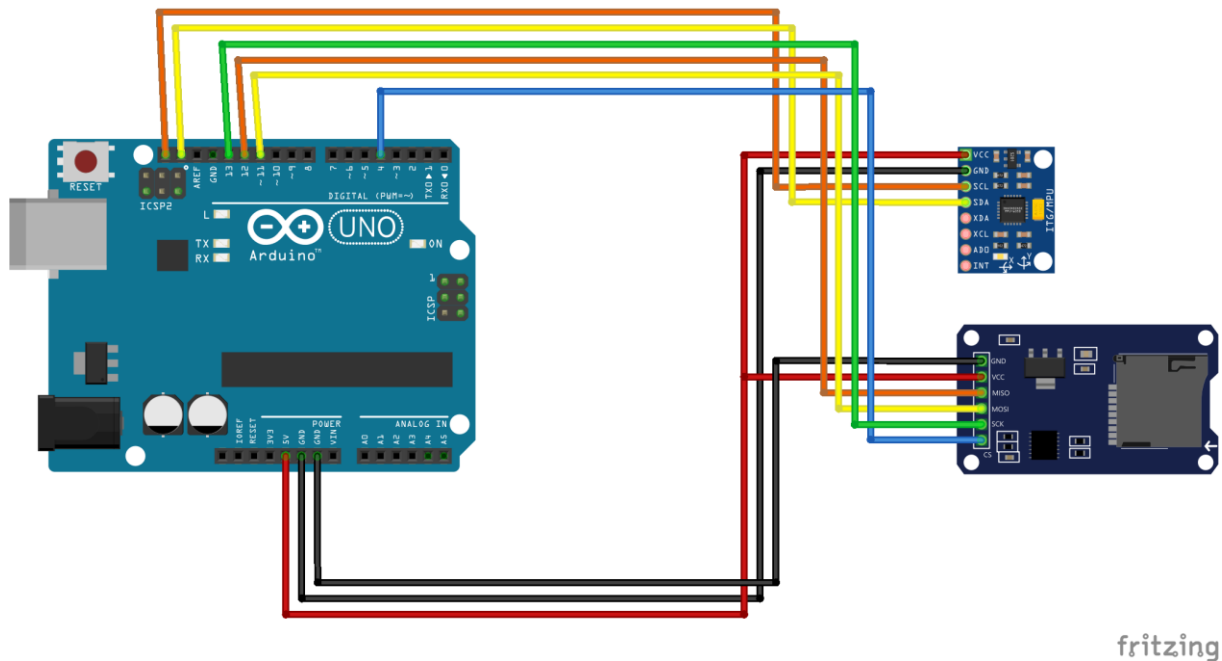
**Abb. 3:** Schaltplan der Verbindung zwischen dem Arduino UNO und dem SD-Karten Modul.

Nach dem Testen der einzelnen Module planen wir sowohl den Sensor als auch das SD-Karten Modul gleichzeitig mit dem Arduino UNO zu verbinden. Hierbei fiel uns auf, dass beide der Sensoren über eine Betriebsspannung von 5 Volt liefern. Da der Arduino jedoch nur über eine Spannungsquelle dieser Höhe verfügt, löteten wir die beiden Kabel für die Stromversorgung, welche jeweils über die Pins 5V zu VCC liefern, zusammen (siehe Abb. 4).



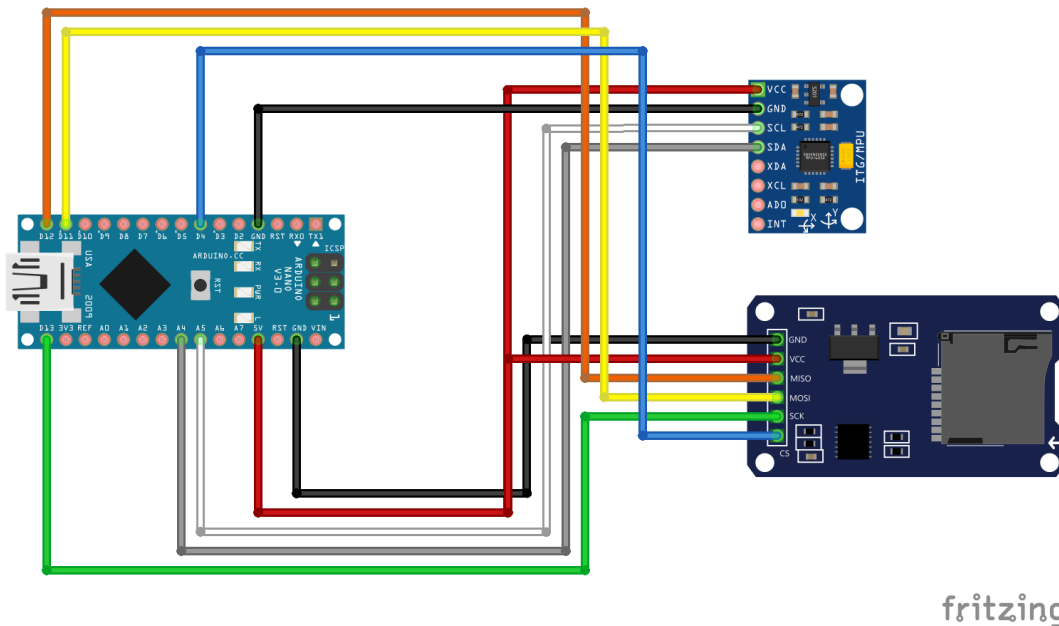
**Abb. 4:** Verlötete Kabel zur Stromversorgung des Sensors und des SD-Karten Moduls.

So konstruierten wir eine Parallelschaltung, in welcher alle Komponenten über die gleiche Spannung miteinander verbunden sind (siehe Abb. 5).



**Abb. 5:** Schaltplan der Verbindung zwischen dem Arduino UNO, dem MPU-6050 Sensor sowie dem SD-Karten Modul.

Nun wollten wir den gesamten Versuch noch kompakter und leichter gestalten, um die Rakete weniger zu beeinflussen und den Anbau der Sensorik an diese zu vereinfachen. Dafür betrachteten wir all unsere Komponenten und überprüften, ob es kleinere und leichtere Alternativen gibt. Dabei stießen wir auf den Arduino Nano, welcher über die meisten Funktionen des Arduino UNO's verfügt, allerdings deutlich kleiner ist. Da dessen Funktionen für unseren Aufbau genügten, bestellten wir einen solchen und ersetzten den Arduino UNO damit (siehe Abb. 6).



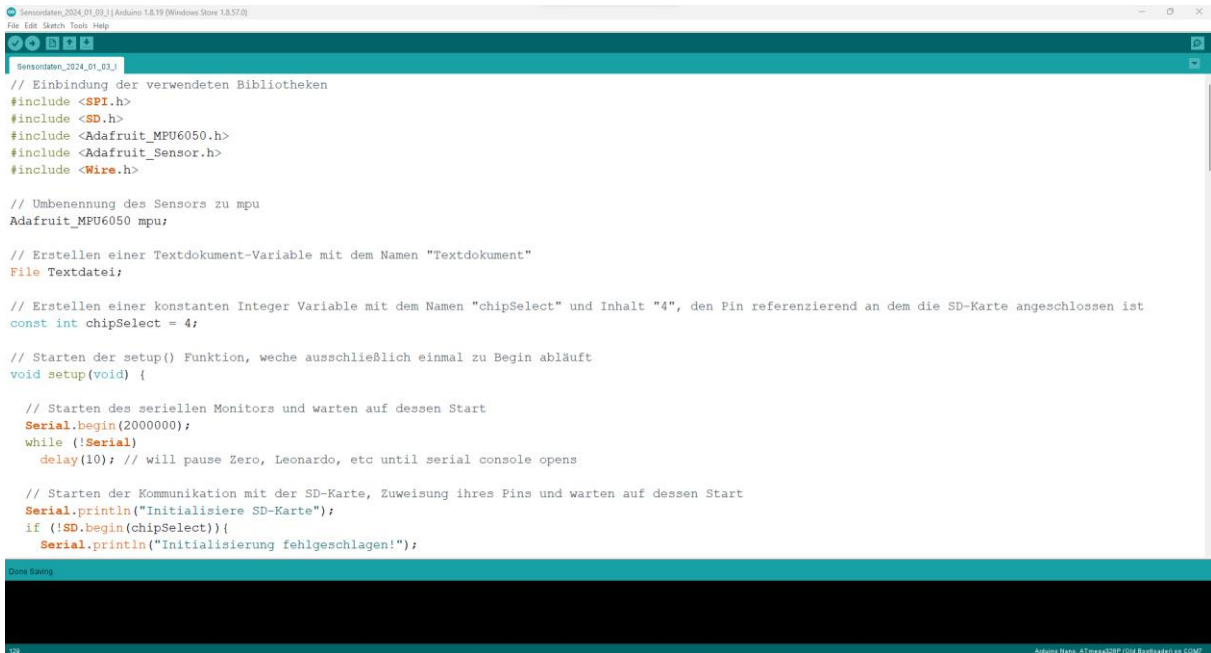
**Abb. 6:** Schaltplan der Verbindung zwischen dem Arduino Nano, dem MPU-6050 Sensor sowie dem SD-Karten Modul.

Des Weiteren planen wir in der Zukunft eine Platine sonderanfertigen zu lassen, welche als Ersatz für die Kabel direkt am Arduino Nano angeschlossen wird und exakt die passenden Steckplätze für den Sensor als auch das SD-Karten Modul bereitstellt.

## Programmcode Sensoren

Bevor wir die nun aus Arduino UNO, dem MPU-6050 Sensor sowie dem SD-Karten Modul bestehende Sensorik programmieren konnten, wollten wir die einzelnen Bestandteile einzeln testen. So können wir spätere Fehler im Gesamtprogramm besser optimieren und gewisse Fehlerquellen von Beginn an ausschließen. Des Weiteren steht uns so jederzeit ein Programm zum Testen der einzelnen Komponenten zur Verfügung, um mögliche, auftretende Defekte der Hardware schnell erkennen zu können. Wir begannen bei unseren Tests bei der Programmierung der SD-Karte und nutzten hierbei als Startpunkt das Beispiel „ReadWrite“ der verwendeten Bibliothek „SD.h“. (siehe Quellenverzeichnis, GitHub, SDKartenSpeicherung.ino). und passten dieses leicht an. Durch dieses Programm wird eine Verbindung zur SD-Karte aufgebaut und eine Textdatei erstellt. Diese lässt sich anschließend beschreiben und abspeichern. Alle essenziellen Schritte werden hierbei über eine Ausgabe bestätigt oder als fehlerhaft identifiziert. Zu Beginn konnten wir hierbei keine Erfolge erzielen und erhielten die Ausgaben „Initialisiere SD-Karte“, „Initialisierung abgeschlossen“ sowie „Textdatei konnte nicht erstellt oder geöffnet werden“. Durch diese wussten wir bereits, dass der Arduino zwar das SD-Karten Modul erkennt und wir die Verkabelung erfolgreich vorgenommen hatten, allerdings keine Möglichkeit hat die SD-Karte zu beschreiben. Unsere erste Idee war, dass diese schreibgeschützt sein könnte. Deshalb überprüften wir zuerst den physikalischen Schreibschutz des Adapters. Als dies erfolglos blieb, haben wir die Software „EaseUS Partition Master“ heruntergeladen und versuchten hierüber einen möglichen internen Schreibschutz zu deaktivieren. Jedoch konnten wir hierbei erkennen, dass dieser bereits deaktiviert war. Bei weiterer Recherche fanden wir dann heraus, dass die zu beschreibende SD-Karte im Format „FAT32“ formatiert sein muss. Als wir unsere SD-Karte überprüften, fiel auf, dass diese anders formatiert war, jedoch war eine Anpassung der Formatierung, aufgrund der Größe der SD-Karte von 64 GB nicht über den Windows-Manager möglich. Bei der Suche nach einer Software, welche uns die entsprechende Formatierung ermöglicht, stießen wir auf „HPUSBDisk“ und konnten diese nun vornehmen. Die neue Version unseres Programmcodes und der Test des SD-Karten Moduls waren somit erfolgreich.

Als nächstes testeten wir unseren MPU-6050 Sensor mit dem Beispiel „basic\_readings“ der verwendeten Bibliothek „Adafruit\_MPU6050.h“ (siehe Quellenverzeichnis, GitHub, SensordatenAuslesung.ino). Hierbei erhielten wir direkt erfolgreich ausgelesene Sensorwerte und konnten Fortfahren mit der Zusammenführung sowie Anpassung und Optimierung unseres Programmcodes (siehe Quellenverzeichnis, GitHub, Sensordaten\_2024\_01\_03\_I.ino) (siehe Abb. 7 - 13).



```

// Sensordaten_2024_01_03_I.ino
// Einbindung der verwendeten Bibliotheken
#include <SPI.h>
#include <SD.h>
#include <Adafruit_MPU6050.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>

// Umbenennung des Sensors zu mpu
Adafruit_MPU6050 mpu;

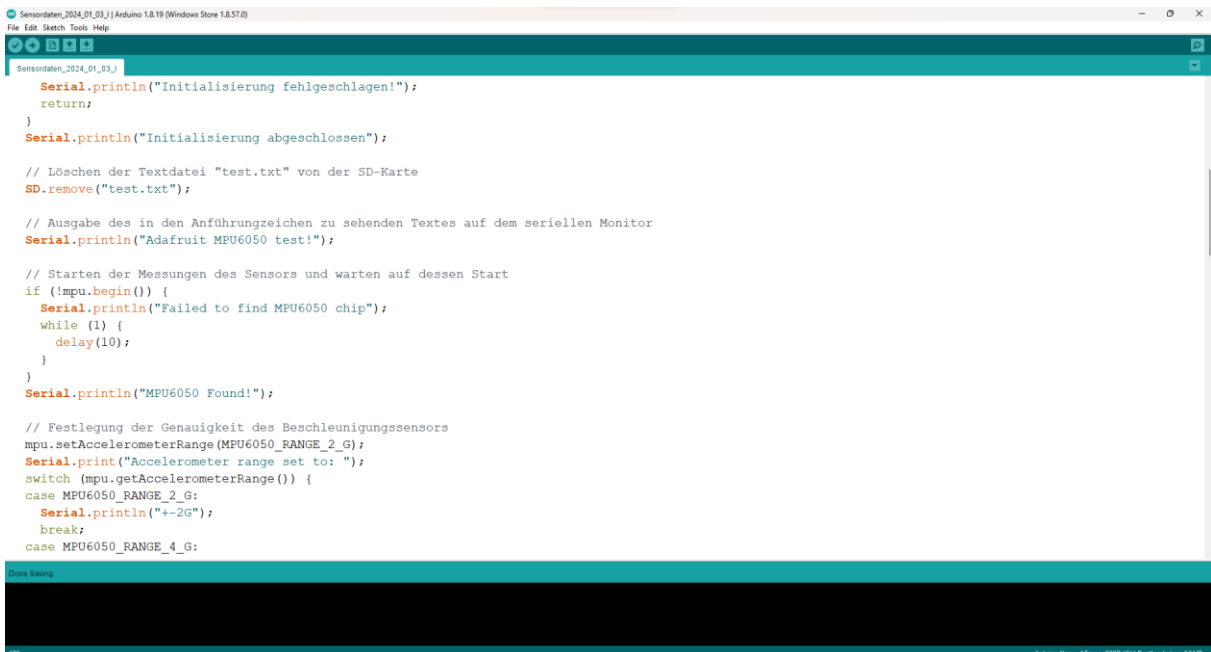
// Erstellen einer Textdokument-Variable mit dem Namen "Textdokument"
File Textdatei;

// Erstellen einer konstanten Integer Variable mit dem Namen "chipSelect" und Inhalt "4", den Pin referenzierend an dem die SD-Karte angeschlossen ist
const int chipSelect = 4;

// Starten der setup() Funktion, welche ausschließlich einmal zu Beginn abläuft
void setup(void) {
    // Starten des seriellen Monitors und warten auf dessen Start
    Serial.begin(2000000);
    while (!Serial)
        delay(10); // Will pause Zero, Leonardo, etc until serial console opens

    // Starten der Kommunikation mit der SD-Karte, Zuweisung ihres Pins und warten auf dessen Start
    Serial.println("Initialisiere SD-Karte");
    if (!SD.begin(chipSelect)){
        Serial.println("Initialisierung fehlgeschlagen!");
    }
}
    
```

**Abb. 7:** Teil des zusammengeführten, angepassten Programmes der gesamten Sensorik



```

        Serial.println("Initialisierung fehlgeschlagen!");
        return;
    }
    Serial.println("Initialisierung abgeschlossen");

    // Löschen der Textdatei "test.txt" von der SD-Karte
    SD.remove("test.txt");

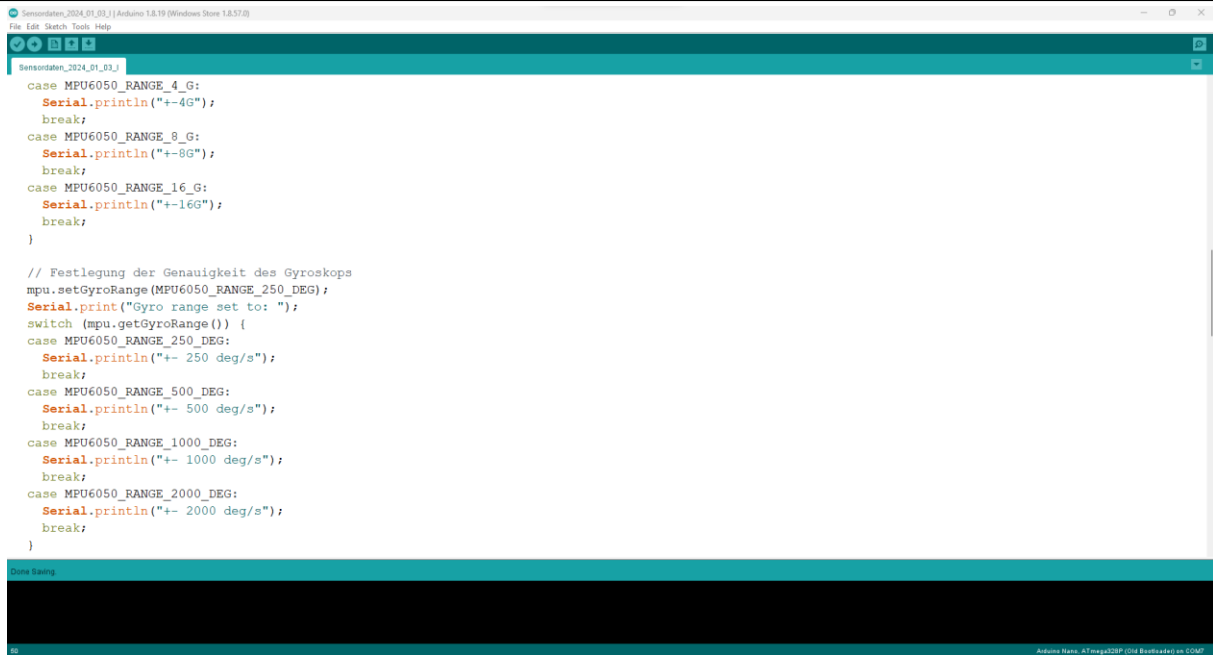
    // Ausgabe des in den Anführungszeichen zu sehenden Textes auf dem seriellen Monitor
    Serial.println("Adafruit MPU6050 test!");

    // Starten der Messungen des Sensors und warten auf dessen Start
    if (!mpu.begin()) {
        Serial.println("Failed to find MPU6050 chip");
        while (1) {
            delay(10);
        }
    }
    Serial.println("MPU6050 Found!");

    // Festlegung der Genauigkeit des Beschleunigungssensors
    mpu.setAccelerometerRange(MPU6050_RANGE_2_G);
    Serial.print("Accelerometer range set to: ");
    switch (mpu.getAccelerometerRange()) {
        case MPU6050_RANGE_2_G:
            Serial.println("+/-2G");
            break;
        case MPU6050_RANGE_4_G:
    }
}
    
```

**Abb. 8:** Teil des zusammengeführten, angepassten Programmes der gesamten Sensorik



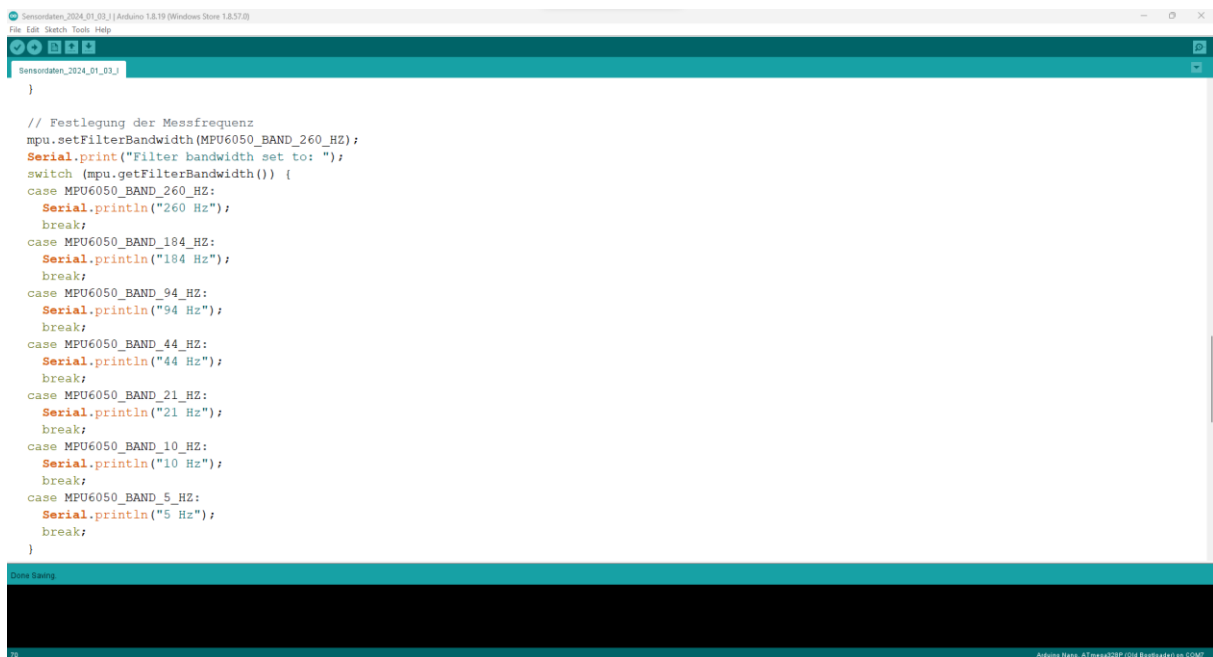


```

case MPU6050_RANGE_4_G:
  Serial.println("+4G");
  break;
case MPU6050_RANGE_8_G:
  Serial.println("+8G");
  break;
case MPU6050_RANGE_16_G:
  Serial.println("+16G");
  break;
}

// Festlegung der Genauigkeit des Gyroskops
mpu.setGyroRange(MPU6050_RANGE_250_DEG);
Serial.print("Gyro range set to: ");
switch (mpu.getGyroRange()) {
case MPU6050_RANGE_250_DEG:
  Serial.println("+ 250 deg/s");
  break;
case MPU6050_RANGE_500_DEG:
  Serial.println("+ 500 deg/s");
  break;
case MPU6050_RANGE_1000_DEG:
  Serial.println("+ 1000 deg/s");
  break;
case MPU6050_RANGE_2000_DEG:
  Serial.println("+ 2000 deg/s");
  break;
}
  
```

**Abb. 9:** Teil des zusammengeführten, angepassten Programmes der gesamten Sensorik



```

// Festlegung der Messfrequenz
mpu.setFilterBandwidth(MPU6050_BAND_260_HZ);
Serial.print("Filter bandwidth set to: ");
switch (mpu.getFilterBandwidth()) {
case MPU6050_BAND_260_HZ:
  Serial.println("260 Hz");
  break;
case MPU6050_BAND_184_HZ:
  Serial.println("184 Hz");
  break;
case MPU6050_BAND_94_HZ:
  Serial.println("94 Hz");
  break;
case MPU6050_BAND_44_HZ:
  Serial.println("44 Hz");
  break;
case MPU6050_BAND_21_HZ:
  Serial.println("21 Hz");
  break;
case MPU6050_BAND_10_HZ:
  Serial.println("10 Hz");
  break;
case MPU6050_BAND_5_HZ:
  Serial.println("5 Hz");
  break;
}
  
```

**Abb. 10:** Teil des zusammengeführten, angepassten Programmes der gesamten Sensorik

```

Sensordaten_2024_01_03_1 | Arduino 1.8.19 (Windows Store 1.8.57.0)
File Edit Sketch Tools Help

Sensordaten_2024_01_03_1
}
Serial.println("");

// Anweisung zum hundertmillisekündigen warten des Programs vor dem weiteren ablaufen
delay(100);
}

// Starten der loop() Funktion, welche sich nach ablaufen der setup() Funktion dauerhaft wiederholt
void loop() {

// Erstellen und öffnen der Textdatei "test.txt" von der SD-Karte sowie Speicherung dieser in der Textdokument-Variable "Textdokument"
Textdatei = SD.open("test.txt", FILE_WRITE);

// Initialisierung und Benennung der Messungen
sensors_event_t a, g, temp;
mpu.getEvent(&a, &g, &temp);

// Abfrage der Verbindung zum Textdokument "test.txt"
if (Textdatei)
{
// Ausgabe des in den Anführungszeichen zu sehenden Textes und der dazugehörigen Sensorwerte auf dem seriellen Monitor bei erfolgreicher Verbindung
Serial.println(millis());
Serial.print(String(a.acceleration.x) + ";" + String(a.acceleration.y) + ";");
Serial.println(String(a.acceleration.z));
Serial.print(String(g.gyro.x) + ";" + String(g.gyro.y) + ";");
Serial.println(String(g.gyro.z));

// Beschreiben des Textdokument "test.txt" mit dem in Anführungszeichen zu sehenden Textes und der dazugehörigen Sensorwerte bei erfolgreicher Verbindung
}
else
{
// Ausgabe des in den Anführungszeichen zu sehenden Textes auf dem seriellen Monitor bei fehlgeschlagener Verbindung
Serial.println("Textdatei konnte nicht erstellt oder geöffnet werden");
}

// Schließen der Textdatei "test.txt"
Textdatei.close();
}
}

```

**Abb. 11:** Teil des zusammengeführten, angepassten Programmes der gesamten Sensorik

```

Sensordaten_2024_01_03_1 | Arduino 1.8.19 (Windows Store 1.8.57.0)
File Edit Sketch Tools Help

Sensordaten_2024_01_03_1
mpu.getEvent(&a, &g, &temp);


// Abfrage der Verbindung zum Textdokument "test.txt"
if (Textdatei)
{
// Ausgabe des in den Anführungszeichen zu sehenden Textes und der dazugehörigen Sensorwerte auf dem seriellen Monitor bei erfolgreicher Verbindung
Serial.println(millis());
Serial.print(String(a.acceleration.x) + ";" + String(a.acceleration.y) + ";");
Serial.println(String(a.acceleration.z));
Serial.print(String(g.gyro.x) + ";" + String(g.gyro.y) + ";");
Serial.println(String(g.gyro.z));

// Beschreiben des Textdokument "test.txt" mit dem in Anführungszeichen zu sehenden Textes und der dazugehörigen Sensorwerte bei erfolgreicher Verbindung
Textdatei.println(millis());
Textdatei.print(String(a.acceleration.x) + ";" + String(a.acceleration.y) + ";");
Textdatei.println(String(a.acceleration.z));
Textdatei.print(String(g.gyro.x) + ";" + String(g.gyro.y) + ";");
Textdatei.println(String(g.gyro.z));
}
else
{
// Ausgabe des in den Anführungszeichen zu sehenden Textes auf dem seriellen Monitor bei fehlgeschlagener Verbindung
Serial.println("Textdatei konnte nicht erstellt oder geöffnet werden");
}

// Schließen der Textdatei "test.txt"
Textdatei.close();
}
}

```

**Abb. 12:** Teil des zusammengeführten, angepassten Programmes der gesamten Sensorik



```

// Starten der setup() Funktion, welche ausschließlich einmal zu Beginn abläuft
void setup() {

    // Starten des seriellen Monitors
    Serial.begin(115200);

}

// Starten der loop() Funktion, welche sich nach Ablauf der setup() Funktion dauerhaft wiederholt
void loop() {

    // Ausgabe der millis() Funktion auf dem seriellen Monitor
    Serial.println(millis());

    // Anweisung zum einsekündigen Warten des Programms vor dem Weiteren ablaufen
    delay(1000);

    // Erstellen einer String Variable mit dem Namen "ka" und Inhalt "Hallo "
    String ka = "Hallo ";

    // Erstellen einer Integer Variable mit dem Namen "a" und Inhalt "800"
    int a = 800;

    // Ausgabe der Kombination aus dem String "ka" und der Integer a, welche zuvor in einen String konvertiert wird, auf dem seriellen Monitor
    Serial.println(ka + String(a));

}

```

**Abb. 13:** Teil des zusammengeführten, angepassten Programmes der gesamten Sensorik

Hierbei dokumentieren wir die erlesenen Sensorwerte der Beschleunigung B und Winkelgeschwindigkeit W nach dem folgenden Schema im Textdokument:

B1 in X Richtung; B1 in Y Richtung; B1 in Z Richtung  
 W1 in X Richtung; W1 in Y Richtung; W1 in Z Richtung  
 B2 in X Richtung; B2 in Y Richtung; B2 in Z Richtung  
 W2 in X Richtung; W2 in Y Richtung; W2 in Z Richtung  
 ...

Zu Beginn gaben wir am Anfang des Dokumentes immer unser eingestelltes „delay()“ als konstante Zeit zwischen den Messwerten aus und später die „millis()“ Funktion vor jedem Set an Messwerten.

## Experimente

### Überprüfung der Materialien

Wir haben unsere verbundenen Materialien am Rechner angeschlossen und mit leichten Bewegungen überprüft, ob der Sensor eine Bewegung ausliest und gleichzeitig eine Speicherung der ausgelesenen Daten auf der SD-Karte folgt. Wir haben auf dem Tisch geprüft, ob die Gravitationskraft ausgelesen und auch erfolgreich gespeichert wird. Infolgedessen haben wir die gesamte Sensorik mit angeschlossenem Rechner in verschiedenen Richtungen bewegt und unmittelbar geprüft, ob diese Richtungsänderungen erfolgreich ausgelesen werden. Dies haben wir jedes Mal nach Änderung des Programmes und Hochladen auf dem Arduino Uno geprüft und auf dem Rechner wiedergeben lassen. Im nächsten Schritt haben wir nach Entnehmen der SD-Karte überprüft, ob diese Daten auch genauso gespeichert werden. Der nächste Schritt ist die Überprüfung der Achsen unseres Sensors und dem Rekonstruktionsprogramm Unity. Wir mussten wissen, ob die Definition der Achsen des Sensors mit der von Unity übereinstimmt. Hierfür haben wir den Sensor vom Tisch jeweils auf jeder Achse einzeln bewegt und überprüft, ob die Bewegung des Sensors mit der Richtung der dargestellten Rekonstruktion übereinstimmen. Daraus erschloss sich, dass die y- und z-Achse vertauscht werden müssen, damit die Richtungen des Sensors mit dem des Rekonstruktionsprogrammes übereinstimmen.

## Rekonstruktion

Nach dem Programmieren des MPU-6050 Sensors mithilfe des Arduino UNO und der Fertigstellung der Verbindungen zwischen Arduino UNO, MPU-6050 und des SD-Karten Moduls, konnten die ersten richtigen Experimente des Projektes beginnen. Bevor wir den Sensor mit der Rakete losschicken können, mussten wir das Zusammenspiel zwischen Sensorik und Rekonstruktion anhand unterschiedlicher, simpler Bewegungen überprüfen. Daher nahmen wir den Arduino, den Sensor und die SD-Karte in die Hand und bewegten uns mit unterschiedlicher Richtung und Geschwindigkeit durch den Raum. Als Stromquelle haben wir eine 9V Batterie benutzt, welche an den Arduino angeschlossen wurde. Anschließend haben wir die Daten aus dem Versuch genutzt und versucht eine Rekonstruktion in einem Programm zu erstellen. Auch diesen Durchgang haben wir wiederholt durchgeführt und immer wieder optimiert. Damit wollten wir die Genauigkeit des gesamten Systems prüfen. Dieses Experiment haben wir häufiger und mit leichten Änderungen im Programm des Sensors durchgeführt, um möglichst genaue und viele Daten auslesen zu können. Beispielsweise führten wir mit dem Sensor in der Hand eine kreisförmige Bewegung durch, während wir einmal um den Tisch liefen und überprüften, ob dessen Rekonstruktion ebenfalls eine ähnliche bis zu identischer Laufbahn anzeigt. Hierbei zeigte sich, dass die reale und simulierte Bewegung nicht übereinstimmte.

## Zeitexperimente

Hier haben wir das Problem aus dem Ergebnis des vorherigen Experimentes überprüft und wollten herausfinden, wo der Fehler der Speicherung liegt. Zuerst haben wir uns das Programm des Arduino UNO angeschaut. Uns war bewusst, dass der Sensor eine maximale Wiederholungsrate von 4ms hat. Dementsprechend haben wir in das Programm ein Delay von 500ms für jede Datenmessung eingefügt, sodass wir nach jeder Sekunde der Messung 2 Datensätze erhalten müssten. Um ein möglichst gutes Ergebnis zu bekommen haben wir uns für eine relativ lange Laufzeit von 20 Sekunden entschieden. Danach haben wir die Anzahl der Messergebnisse, welche in der Theorie erscheinen müssten, mit der Anzahl der realen Datensätze verglichen. Hierbei stellten wir fest, dass es leichte Abweichungen von der erwarteten Menge an Daten gab. Daher haben wir als nächstes das gleiche Vorgehen mit einem Delay von 10ms durchgeführt, sodass normalerweise nach einer Sekunde 100 Datensätze erreicht werden müssten. Hierbei entdeckten wir große Abweichungen und überlegten uns wie wir dieses Problem genauer untersuchen könnten. Unsere erste Überlegung war, dass das SD-Karten Modul Zeit zum Abspeichern der Daten benötigen könnte. Nach einiger Recherche stellten wir fest, dass das Modul tatsächlich eine Speicherzeit von 3ms benötigt. Daher addierten wir für alle Speicheranweisungen im Programm, 3ms auf das, in der Textdatei notierte, „delay()“. Nun führten wir einen weiteren Vergleich der theoretischen und tatsächlichen Anzahl an Messwerten durch und stellten erneut große Abweichungen fest. Also suchten wir nach weiteren Lösungsansätzen. So war unsere nächste Idee, dass das geschriebene Programm auch Zeit benötigen könnte, um einmal durchzulaufen und somit eine weitere Verzögerung hervorzurufen könnte. Unser Plan war nun diese Zeit auszulesen und anschließend mit dem „delay()“, wie bei der SD-Karten Speicherung, zu verrechnen und somit als Fehlerquelle zu entfernen. Hierzu fügten wir zu Beginn des sich wiederholenden Teils des Programmes, der „loop()“-Schleife eine Ausgabe mittels der Anweisung „Serial.println()“ hinzu. Während der Arduino an dem Computer angeschlossen ist, lassen sich Daten auf dem Seriellen Monitor ausgeben. Dieser hat auch Zugriff auf die Uhrzeit des Computers. Das machten wir uns zu Nutze, indem wir die sich wiederholende Ausgabe zusammen mit der Uhrzeit ausgeben ließen. So konnten wir die Zeiten, die das Programm für jeden Durchlauf benötigt, ablesen. Wir stellten fest, dass diese nicht nur teilweise mehrere dutzende Millisekunden lang, sondern zusätzlich variabel sind. Das stellte ein großes Problem dar, da wir daher diese Zeiten nicht herausrechnen konnten. Wir mussten also eine Möglichkeit finden die zu den Messungen zugehörige Zeit mit auszugeben, sodass die Zeitdifferenz zwischen den Messungen separat und richtig bestimmt werden kann. Unsere erste Idee, um dies umzusetzen war ein Echtzeitcompiler, welcher im Prinzip keine Variablen Zeiten benötigt. Uns ist schnell aufgefallen, dass dieser Compiler nicht für den Arduino verfügbar ist. Unsere zweite Idee war eine Uhr anzubringen, welche jedoch unnötige Umstände verursachen würde. Denn uns ist aufgefallen, dass im Arduino Programm die Anweisung „Millis“ die Lösung unserer Probleme ist. Hierbei wird ab Beginn des Sensors die Zeit in Millisekunden dauerhaft angegeben. Das Besondere an dieser Anweisung ist die parallele Laufzeit zum Code, ohne dass dieser beeinflusst wird. Dies haben wir getestet, indem wir uns die „millis()“ Funktion vor und nach einem „delay()“

haben ausgegeben lassen. Hierbei führte das entsprechende „delay()“ zu keiner Einschränkung der Fortzählung der Millisekunden (siehe Quellenverzeichnis, GitHub, TestMillis.ino). Daher gaben wir von nun an die vor jeder Messung abgelaufenen Millisekunden aus, um die variable Zeit zwischen den Messungen bei der Rekonstruktion berücksichtigen zu können. Jedoch stellten wir fest, dass die realen und simulierten Bewegungen immer noch nicht übereinstimmten.

## Genauigkeitsexperiment

Als weitere mögliche Fehlerquelle analysierten wir als nächstes die Genauigkeit des Sensors. Hierfür haben wir den Arduino mit verbundenen MPU-6050 Sensor und SD-Karten Modul mit dem Rechner verbunden. Wir haben den Sensor auf dem Tisch liegen gelassen, ohne diesen zu Bewegen. Nach Einschalten der Auslesung sollte nur eine Beschleunigung von  $9,81 \text{ m/s}^2$ , welche die Gravitationskraft widerspiegelt, ausgelesen werden (siehe Quellenverzeichnis, GitHub, Messgenauigkeit.ino). Da der Sensor nur schwer perfekt gerade auszurichten ist, haben wir hier den Betrag des Beschleunigungsvektors berechnet, um die Erdbeschleunigung genauer auslesen zu können. Anschließend berechneten wir das dazugehörige arithmetische Mittel sowie dessen Standardabweichung.

$$\sigma = \sqrt{\sum(x-\bar{x})^2 / N} = \sqrt{0.01907163284} = 0.1381000827$$

$$\bar{x} = \sum x / N = 7856.90 / 776 = 10.12487113$$

Durch die Division  $10,12/9,81$  sind wir auf eine Abweichung von 3% gekommen. 10,12 stellt hierbei unser arithmetisches Mittel, aller gemessenen Erdbeschleunigungen und 9,81 den Literaturwert dar. Aufgrund dieser Ungenauigkeit, welche sich durch die hohe Anzahl an Messwerten verstärkt, und der hohen Standardabweichung konnten wir die Abweichungen zwischen der Simulation und der realen Bewegung mit der Genauigkeit Sensor erklären.

## Geplantes Raketenexperiment

Auch wenn die Durchführung der Messung eines Raketenstarts aufgrund der Ungenauigkeiten des Sensors keinen Zweck für uns hatte, wollten wir es dennoch planen, um es bei Einbau eines genaueren Sensors durchführen zu können. Hierbei würden wir beginnen die gesamte Sensorik an die Rakete anzubringen und diese in die Luft steigen zu lassen. Wichtig wäre, dass der Sensor fest an der Rakete befestigt wird, sodass die genauen Sensordaten der Rakete ausgelesen werden und es nicht zu Verfälschungen der Daten kommt. Außerdem sollte der Aufbau und der Start der Rakete mithilfe einer Spiegelreflexkamera dokumentiert werden. Diese Aufnahmen würden der Kontrolle unserer Simulation sowie der Veranschaulichung dienen. Zuerst wollten wir die Rakete so platzieren, dass diese einen standhaften Grund hat, um einen sicheren Start zu gewährleisten und die potenzielle Gefahr der herabfallenden Rakete zu minimieren. Im nächsten Schritt sollte die Kamera an einen dreibeinigen Ständer befestigt und aufgestellt werden, um auch dieser einen möglichst guten Halt zu geben. Nun müssten wir nur noch die Rakete starten. Mit den gewonnenen Daten hätten wir nun eine Rekonstruktion der Flugbahn erstellt und wären an unser Ziel angekommen.

## Rekonstruktion

### Wahl der Technologie

Nach dem Auslesen und Abspeichern der Daten galt es nun, die Daten in einem weiteren Programm sinnvoll zu verarbeiten und damit die Raketenflugbahn zu rekonstruieren. Da wir hier mit dreidimensionalen Strukturen arbeiten, schien eine Engine eine offensichtliche Lösung zu sein, um die Flugbahn auf möglichst unkomplizierte Art und Weise dreidimensional darstellen zu können. Die 3D Game Engine „Unity“ war für uns eine offensichtliche Wahl, nicht nur weil damit anschauliche Grafiken und Flugbahnen einfach umsetzbar sind und die eingebauten Methoden und Klassen das Rechnen mit Vektoren erleichtern, sondern auch, weil einige von uns schon privat Erfahrung mit dem Programm gesammelt hatten und die Umsetzung somit intuitiver und interessanter erschien.

## Mathematische Umsetzung

Bevor wir uns jedoch mit dem Programmieren auseinandersetzen konnten, galt es zunächst die Berechnung der Flugbahn mathematisch zu lösen. Hierfür sind bei der gegebenen Funktion der Beschleunigung  $\vec{a}$  und der Winkelgeschwindigkeit  $\vec{\omega}$  mehrere Integrale nötig, um den zeitlichen Verlauf der Geschwindigkeit  $\vec{v}$  und damit des Ortes  $\vec{r}$  sowie den Verlauf der Rotation  $\vec{\alpha}$  nachzuverfolgen:

$$\vec{v}(t) = \vec{v}_0 + \int_0^t \vec{a}(t) dt$$

$$\vec{r}(t) = \vec{r}_0 + \int_0^t \vec{v}(t) dt$$

$$\vec{\alpha}(t) = \vec{\alpha}_0 + \int_0^t \vec{\omega}(t) dt$$

Leider haben wir über die Messdaten die Funktionen nicht direkt gegeben, da sich die Achsen der angezeigten Beschleunigung durch die Rotation verändern. Deshalb haben wir uns dazu entschieden, die Integrale kumulativ auszurechnen und die Ausrichtung der Rakete vor jedem Messvorgang der Beschleunigung zunächst anzupassen, um dann die Beschleunigung in der richtigen Richtung, in der sie auch gemessen wurde, anzuwenden. Zu dieser Berechnungsweise der Flugbahn haben wir einen Geschwindigkeitsvektor, zu welchem in jedem Durchlauf der Beschleunigungsvektor in korrigierter Richtung und mit der Zeit  $\Delta t$  multipliziert, addiert wird. Die Zeit beschreibt hierbei den Zeitabstand seit der letzten Messung und kann aus den Messdaten durch Subtraktion der aufgezeichneten Zeitpunkte berechnet werden.

## Programmtechnische Umsetzung

Für das Programm geht es zunächst darum, die Daten aus dem txt-File auszulesen und zu einer bzw. mehrere Listen an Vektoren und floats zu konvertieren, was mit folgenden Methoden umgesetzt wurde (siehe Abb. 14).

```
1 reference
private void ReadFile(string path)
{
    stringVectors = File.ReadAllLines(path);
    delays.Add(float.Parse(stringVectors[0]));

    for(int i = 1; i < stringVectors.Length; i+=3)
    {
        AddStringVector(i);
        AddStringVector(i+1);
        delays.Add(float.Parse(stringVectors[i+2]));
    }
}

2 references
private void AddStringVector(int index)
{
    string[] stringVector = stringVectors[index].Split(new string[] { ";" }, StringSplitOptions.None);

    if (stringVector.Length != 3)
    {
        print("Error");
    }

    data.Add(new Vector3(float.Parse(stringVector[0]), float.Parse(stringVector[2]), float.Parse(stringVector[1])));
}
```

**Abb. 14:** Auslesen und Konvertieren der txt-Datei

Die Vektoren für Geschwindigkeit und Position wurden in der Unity eigenen Klasse "Transform" gespeichert, um Unitys volle Kapazität an Funktionen verwenden zu können. Anschließend wurden die Berechnungen in einzelnen Methoden umgesetzt und teilweise noch einige Einheiten passend konvertiert, sodass Unity die Werte nutzen kann (siehe Abb. 15)



```

1 reference
private void CalculateRotation(int index)
{
    Vector3 rotationAngle = data[index] * currentDelay * Mathf.Rad2Deg;
    rocketTransform.Rotate(rotationAngle);
    velocityObject.Rotate(rotationAngle);
}

1 reference
private void CalculateVelocity(int index)
{
    if(applyGravity)
        ApplyGravity();

    velocityObject.Translate(data[index] * currentDelay);

    if(lockHeight)
        velocityObject.position = new Vector3(velocityObject.position.x, 0f, velocityObject.position.z);
}

1 reference
private void CalculatePosition()
{
    rocketTransform.Translate(velocityObject.position * currentDelay, Space.World);
}

```

**Abb. 15:** Anwendung der Berechnung in jeweiligen Methoden für die einzelnen Schritte

So könnten jetzt die Werte in einer einfachen For-Schleife durchgerechnet werden. Um das ganze jedoch in Echtzeit nachsimulieren zu können, sodass der Flug der Rakete anschaulicher wird, wurde das Ganze mit dem jeweiligen Delay zwischen den Messschritten in der Update Funktion durchgeführt (siehe Abb. 16).

```

//Simulate
Unity Message | 0 references
private void Update()
{
    //End
    if(finished) return;
    if (i >= data.Count)
    {
        finished = true;
        Debug.Log("Final Height: " + rocketTransform.position.y.ToString());
        return;
    }

    //Timer
    if (!running) return;
    timer += Time.deltaTime;
    if(timer >= currentDelay)
    {
        running = false;
        currentDelay = ConvertDelayToSeconds(delays[(i / 2)+1] - delays[(i / 2)]);

        CalculateRotation(i+1);
        CalculateVelocity(i);
        CalculatePosition();

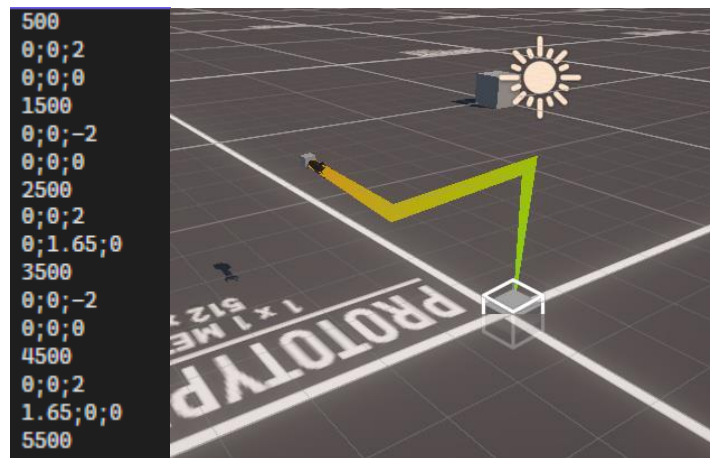
        i += 2;
        timer = 0;
        Debug.Log("rotation: " + velocityObject.localRotation.eulerAngles.ToString());
        Debug.Log("forward: " + velocityObject.forward.ToString());
        Debug.Log(currentDelay.ToString() + " seconds passed");
        running = true;
    }
}

```

**Abb. 16:** Update Funktion mit Endbedingung, Delay und Berechnung für alle Daten

Leider ließ sich der Code nicht einfach debuggen, da das einfache Einsetzen und Testen der Messwerte zu viele mögliche Fehlerquellen haben kann, die auch auf der Seite des Ausleseprogrammes oder des Sensors selbst liegen könnten. Des Weiteren sind die gemessenen Daten nicht intuitiv

verständlich und führen nicht zu einer klar erwartbaren Flugbahn, dessen Verlauf man überprüfen könnte. Deshalb haben wir, nachdem wir die Übereinstimmung der Achsen von Unity und dem Sensor mit einigen Experimenten bestätigt hatten, eigene Messdaten entworfen, um das Programm zu testen (siehe Abb. 17).



**Abb. 17:** txt-Datei mit selbstgeschriebenen Messdaten und zugehörige Simulation

Zuletzt ging es darum verschiedene Möglichkeiten hinzuzufügen, um die Schwerkraft dazuzurechnen oder die Höhe insgesamt statisch zu lassen, um das Programm überprüfen zu können und akkurate Flugbahnen zu erhalten.

Das voll funktionsfähige Programm inklusive einiger Testdateien finden sie in unserem öffentlichen Github Repository (siehe Quellenverzeichnis).

Wir haben außerdem geplant, den genauen Abstand vom Sensor zum Masseschwerpunkt der Rakete zu bestimmen, um die Flugbahn des Masseschwerpunktes, anstatt die des Sensors zu verfolgen, was in Unity unproblematisch umzusetzen ist.

Denn leider haben, obwohl wir die Funktion des Programms bestätigen konnten, die simulierten Flugbahnen, vor allem am Ende ihres Verlaufs, dennoch nicht mit den beobachteten Flugbahnen übereingestimmt. Wir gehen deshalb davon aus, dass sich die kleinen, durch das Genauigkeitsexperiment nachgewiesenen Ungenauigkeiten bei Messungen im Sensor mit der Zeit addieren und die Flugbahn verfälschen.

## Ergebnisse

Die ersten Ergebnisse, die wir erreicht haben, entsprachen unserer Vorstellung der Experimente. Die Verbindungen sowohl zwischen den einzelnen Materialien als auch zwischen allen drei Materialien haben wir ohne große Schwierigkeiten erstellen können. Durch Ablaufen der Daten konnten wir herausstellen, dass in der Programmierung teilweise Fehler vorhanden waren, welche wir nach Umschreiben des Programmes an einzelnen Stellen schnell beheben konnten. Nun hatten wir ein fertiges Programm für die Auslesung der Daten.

Weitere Ergebnisse erreichten wir bei dem ersten richtigen Experiment mit vollständiger und funktionsfähiger Verbindung. Während der Durchführung unseres Experimentes "Speicherung der Daten" wurden die Daten auf dem ersten Blick erfolgreich ausgelesen und gespeichert. Bei der Auswertung dieser Daten und Erstellung der Rekonstruktion erschloss sich jedoch der erste Durchlauf des Experimentes als fehlgeschlagen. Die Daten wurden in viel zu große Abstände ausgelesen und waren nicht für eine präzise Rekonstruktion geeignet. Beim Überprüfen des Fehlers kamen wir zum Entschluss, dass in erster Hinsicht nicht die Programmierung der Fehler war, sondern die Speicherung der Daten auf der SD-Karte. Zwischen jeder Speicherung von Daten benötigt die SD-Karte eine gewisse Zeit, bis der nächste Datenwert gespeichert werden kann. Auch wenn sich dieser Prozess im Millisekunden-Bereich abspielt, wirkt es sich auf den Zeitverlauf und dementsprechend auf die Rekonstruktion aus. Aus diesem Ergebnis folgend haben wir weitere Versuche



durchgeführt und wichtige Teilergebnisse entnehmen können. Zum Beispiel konnten wir nach erneuter Überprüfung herausfinden, dass nicht nur die SD-Karte Zeit benötigte, sondern auch der Compiler, also der Arduino und Sensor zum Ausführen ihrer Anweisungen. Hier konnten wir die Funktion "millis()" verwenden, um die Ausgabe der variablen Zeiten zu gewährleisten.

Die Funktion des Rekonstruktionsprogramms konnte mit eigenen Datensätzen bestätigt werden, jedoch zeigten die Rekonstruktionen von gemessenen Bewegungen starke Abweichungen von unseren Beobachtungen. Im Einklang damit hat unser Genauigkeitsexperiment eine Abweichung von ca. 3% vom Idealwert ergeben. Die vollständige Rekonstruktion der Flugbahn der Rakete, haben wir deshalb noch nicht getestet.

## Ergebnisdiskussion

Das Projekt insgesamt ist sehr gut gelaufen und wir konnten, trotz einiger Rückschläge und Probleme die Fehlerquellen zügig ausfindig machen, um schnell weiterzuarbeiten. Vor allem das Erwerben und Zusammenschließen der Materialien lief reibungslos ab. Auf der Softwareseite in Bezug auf das Auslesen der Daten konnten wir viele Probleme nach einigen Experimenten und Nachforschungen beheben. So haben wir das Problem der Formatierung der Daten auf der SD-Karte durch genauere Recherche lösen können. Dadurch, dass wir die Komponenten zuerst ausgiebig untersucht haben, konnten wir viele Probleme auf Softwareseite vorbeugen und beheben, wie es beispielsweise mit einer Verwechslung der Achsen der Fall gewesen ist. Wir konnten somit erfolgreich eine Kommunikation zwischen mehreren Programmen herstellen und die entsprechenden Daten übertragen. Auch das Problem der variablen Laufzeit des Auslesevorgangs, welche uns zunächst nicht bewusst gewesen ist, konnten wir schnell erkennen, durch Experimente bestätigen und schließlich durch das Ausgeben der Zeitpunkte und Berechnen der variablen Zeitintervalle lösen. Auch das Programm für die Rekonstruktion konnten wir aufgrund unserer mathematischen Schulbildung und Programmierfähigkeiten problemlos konstruieren und umsetzen. Die Schwierigkeit des Testens konnten wir durch selbstentworfenen Datensätze beseitigen. Dass die rekonstruierte Flugbahn dennoch nicht akkurat ist, konnten wir kumulierten Messfehlern zuschreiben, welche wir auch durch ein Experiment bestätigen konnten. Aus diesen Gründen haben wir jedoch auf weitere Experimente, wie beispielsweise das Integrieren der Sensorik in eine Wasser-Luftdruckrakete, verzichtet. Unsere Ergebnisse stimmen insofern mit den wissenschaftlichen Erwartungen überein, dass bei einem Sensor im Wert von ca. 3€ nicht davon ausgegangen werden kann, dass er für hohe Geschwindigkeiten und exakte Rekonstruktionen die Messdaten liefert, die der nötigen Genauigkeit entsprechen. Unser größter Fehler war somit vermutlich, zu viel Geld beim Kauf eines Sensors sparen zu wollen. Für die Zukunft planen wir deshalb, dass wir entweder weitere ähnliche Sensoren kaufen, diese gleichzeitig einbauen und den Durchschnitt der Messungen nehmen, um so die Abweichung zu minimieren oder im besten Fall mit besseren finanziellen Mitteln einen genaueren Sensor zu erwerben.

## Fazit und Ausblick

Um nochmal einen abgerundeten Überblick unseres Projektes darzustellen, kommen wir nun zu unserm Fazit. Vorab lässt sich sagen, dass wir leider nicht mehr das finale Experiment durchführen konnten und dementsprechend keine Ergebnisse bezüglich dieses Experimentes sammeln konnten. Jedoch konnten wir mit anderen erfolgreichen, aber auch fehlgeschlagenen Experimenten sinnvolle und zielführende Ergebnisse erbringen. Mit diesen Ergebnissen konnten wir unser Projekt positiv verbessern und kamen unser Ziel immer näher.

## Quellen- und Literaturverzeichnis

GitHub, Das MPU-6050 Modul: RaketenSensorik, <https://github.com/Feehl/RaketenSensorik>, besucht am 14.01.2024.

Funduino GMBH, Nr. 28 – Das SD-Karten Modul, <https://funduino.de/nr-28-das-sd-karten-modul>, besucht am 10.01.2024.

Funduino GMBH, Das MPU-6050 Modul: Gyroskop und Beschleunigungssensor, <https://funduinoshop.com/das-mpu-6050-modul-gyroskop-und-beschleunigungssensor>, besucht am 10.01.2024.

## Unterstützungsleistungen

Dr. Hans-Otto Carmesin, Lehrer, Gymnasium Athenaeum Stade, Stade, hat uns bei unserem Projekt, durch das zur Verfügung stellen der Materialien und eines Ortes zum Arbeiten unterstützt.