Universität zu Lübeck
Bachelorprojekt
Fahrrad Sturzerkennung mittels IMU

17.04.2021
Max-Ole von Waldow
Julian Denker

Dokumentation Bachelorprojekt

Projekt verfügbar unter: https://github.com/JD1694/BachelorProject-BicycleCrasher

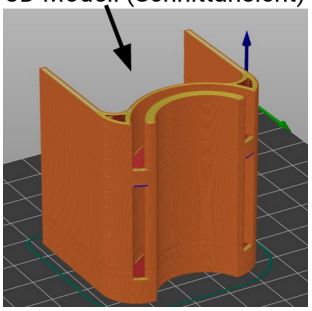
Zielsetzung

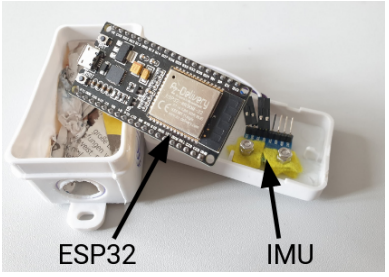
Es sollen Fahrradstürze mittels einer an einem Fahrrad angebrachten IMU erkannt werden. Diese Detektion bildet ein Grundgerüst, das um verschiedene Module zur Sicherheit des Radfahrers erweitert werden kann. Diese Erweiterungen könnten beispielsweise einen Airbag oder Notruf aktivieren, ohne eigene Sensorik zu benötigen.

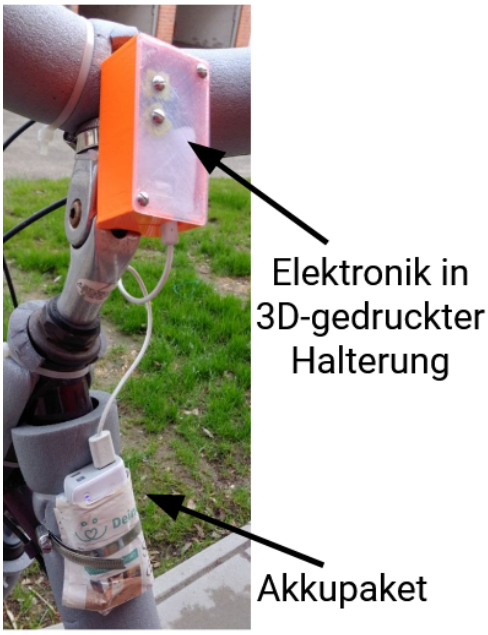
Vorbereitung der Hardware

Verwendet wird eine ESP32-WROOM als Mikrocontroller und als Sensor eine GY-521 MPU6050 die über das I2C-Protokoll untereinander kommunizieren. Für den ESP32 haben wir uns vor allem aus zwei Gründen entschieden: seine sehr hohe Taktrate und eine auf der ALU implementierte FPU (Floating Point Unit) zur schnelleren Berechnung von Fließkommaoperationen. Mithilfe eines eingebauten DMP (Dynamic Motion Processor) werden die Sensordaten bereits auf der IMU mithilfe eines Kalman- und Komplementärfilters bereinigt um u.A. den Drift des Gyrosensors auszugleichen, Rauschen zu minimieren und Yaw, Pitch und Roll zu berechnen. Da insbesondere der Kalman-Filter nur unter hohem Rechenaufwand in Software realisierbar ist und für die Erkennung von Stürzen möglichst viele Abfragen pro Sekunde benötigt werden, entlastet die DMP den ESP32.
Die gemessenen Sensordaten werden über einen in den Mikrocontroller integrierten WiFi-Chip an externe Geräte übermittelt. Die Elektronik wird in einer gepolsterten Abzweigdose am Lenker eines Fahrrads befestigt. Im Laufe des Projekts wurde die Abzweigdose durch eine 3D-gedruckte Halterung ersetzt, die leichter am Lenker befestigt werden kann und das Kabel zur Stromversorgung besser fixiert, welche durch eine am Fahrradrahmen befestigte USB-Powerbank gewährleistet wird.

3D Modell (Schnittansicht)







Programme: Übersicht

Daten aufnehmen und speichern:

- ESP32_ClientSendDMPData.ino
    - Misst Bewegungen am Fahrrad und sendet diese Sensordaten an “DataCollection_Server.py”
    - Fügt Timestamps mit aktueller Zeit hinzu
- DataCollection_Server.py
    - Läuft auf dem Computer als Server, mit dem sich “ESP32_ClientSendDMPData.ino” verbinden kann
    - Empfängt Sensordaten und speichert sie in einer Log-Datei
    - Nimmt während einer Testfahrt bei manuellem Tastendruck den Sturzzeitpunkt auf

Analysetools und Visualisierung:

- PlotCrash.py
    - Visualisiert bereits aufgenommene Sensordaten und die resultierenden Sturzwahrscheinlichkeiten
- LogUpdater.py
    - Aufwerten von timestamps in Log-Dateien
    - Unterstützt das Verschieben des timestamps eines Sturzes

Erkennungssoftware:

- finalBikeCrasherServer.ino
    - Läuft auf dem Mikrocontroller und bietet eine Website mit der Sturzwahrscheinlichkeit an
    - Sensordaten werden gefiltert und mit festgelegten Grenzwerten verglichen
- Bikecrasher.cpp/Bikecrasher_singleParam.cpp
    - Eine Version der Sturzerkennungs-Software, die zu Simulationszwecken auf dem Computer laufen kann
    - Läuft auf bereits aufgenommenen Testdaten und ohne Netzwerkkommunikation
    - Kann mit verschiedenen Parametern gestartet werden und gibt die Resultate der Erkennung zurück

Simulation und Optimierer mit Simulationsaufruf:

- Simulation_SingleParam.py
    - Startet und bewertet mehrere Durchläufe von “Bikecrasher.cpp” über verschiedene Log-Dateien mit spezifizierten Grenzwerten
    - Anhand der Bewertung werden die Grenzwerten angepasst und optimiert

Programme: Benutzung und Funktionsweise

ClientSendData.ino & DataCollection_Server.py

“ClientSendData.ino” läuft auf dem Mikrocontroller, liest per I2C-Protokoll Sensordaten von der IMU aus und schickt sie per WiFi an das Hostprogramm “DataCollection_Server”, welches auf einem Laptop läuft. In den Daten enthalten sind die Orientierung im Raum als YPR-Winkel, die Beschleunigung einmal mit und einmal bereinigt von der Schwerkraft und die Winkelgeschwindigkeit für die drei orthogonalen Achsen x, y, z. Außerdem angefügt sind auf Millisekunden genaue Timestamps, damit die Sensordaten unabhängig der Latenz durch die Übertragung zeitlich richtig interpretiert werden können. Die aktuelle Zeit wird mit dem Network Time Protocol über das Internet aktualisiert.
“DataCollection_Server” kann wiederholt gestartet werden und nimmt dann eine Verbindung von dem durchgehen laufenden “ClientSendData.ino” an. Die empfangenen Daten werden im CSV Format in eine Log-Datei geschrieben. Per Click auf die Leertaste wird der Timestamp eines Sturzes in einer Datei, die mit “_timestamps.txt” endet, festgehalten. Dieser Zeitpunkt wird zum bewerten in der Simulation späterer Sturzdetektionen benötigt.
Exemplarische Zeile aus der Log-Datei: ”exampleLog.txt”

2020-12-18_18-00-31-316000, ypr, -25.95, 2.88, 0.29, areal, 1056, 435, -696, accel, 1468, 477, 7484, gyro, 0, -9, -5
2020-12-18_18-00-31-339000, ypr, -25.79, 3.29, 0.38, areal, 1129, 403, -714, accel, 1599, 458, 7463, gyro, 1, -10, -7
2020-12-18_18-00-31-364000, ypr, -25.47, 3.95, 0.55, areal, 1053, 364, -904, accel, 1618, 443, 7267, gyro, 2, -13, -11

Exemplarischer Timestamp eines Sturzes: ”exampleLog_timestamps.txt”

2020-12-18_18-00-36-761432

PlotCrash.py

Dieses Tool kann den Namen einer Log-Datei als Kommandozeilenargument entgegennehmen oder nach einer Datei fragen, sollte keine spezifiziert worden sein. Die Sensordaten aus der Log-Datei werden geplottet und es wird der zugehörige Zeitpunkt eines Sturzes im Diagramm markiert.
Auf Wunsch wird “Bikecrasher.cpp” gestartet und zu jedem Zeitpunkt die Wahrscheinlichkeit eines Sturzes geplottet.
Die letzte Grafik unter dem Punkt Resultate wurde mit diesem Tool erstellt.

LogUpdater.py

Dieses Tool nimmt über die Kommandozeile eine oder mehrere Log-Dateien entgegen oder geht alle Dateien in dem Log-Ordner durch, falls keine bestimmte spezifiziert wurde.
Es aktualisiert die Timestamps in einer Log-Datei, falls “ESP32_ClientSendDMPData.ino” die Zeit nicht aus dem Internet beziehen konnte. In dem Fall liegen die Timestamps der Sensordaten im Jahr 1970, dem Beginn der Unix Zeit. In den Timestamps ist jedoch noch der korrekte zeitliche Abstand relativ zu den anderen Messungen enthalten. Sie können also einheitlich zu dem Erstellzeitpunkt der Log-Datei verschoben werden.
Wegen der langsamen, menschlichen Reaktionszeit beim initialen Speichern des Sturzzeitpunkts, liegen diese oft daneben. Dadurch, dass die interne Uhr des Mikrocontrollers um wenige Sekunden von der des Computers, auf dem “LogUpdater.py” läuft, abweichen kann, kommt es zu einer weiteren möglichen Ungenauigkeit des Timestamps.
Durch Aufruf von “PlotCrash.py” wird die Log-Datei geplottet und dabei hat der Benutzer die Möglichkeit den bei Aufnahme des Logs gespeicherten Sturzzeitpunkt zu korrigieren.

Bikecrasher.cpp/Bikecrasher_singleParam.cpp

Das Programm parst durch die aufgezeichneten Sensordaten und vergleicht diese mit beim Aufruf übergebenen Parametern um die Wahrscheinlichkeit eines Crashes zu ermitteln. Wird ein Crash erkannt, gibt das Programm den Zeitpunkt des Crashes, sowie einen Wahrscheinlichkeitswert zurück, der sich aus der Anzahl der überschrittenen Grenzwerte geteilt durch die Anzahl von Tests zusammensetzt. Auf diese Weise können in wenigen Sekunden eine Vielzahl von Fahrsituationen mit unterschiedlichen Grenzwerten simuliert werden, um die optimalen Parameter zu bestimmen. Das Programm wird dazu von Simulation.py aufgerufen. In der neueren Version “Bikecrasher_singleParam“ werden nur noch einzelne Parameter unabhängig voneinander getestet.

finalBikeCrasherServer.ino

“finalBikeCrashServer” wird wieder auf dem ESP32-WROOM ausgeführt. Das Programm verwendet die zuvor bestimmten Grenz- und Sensorwerte der IMU um festzustellen, ob ein Sturz stattfindet.
Gleichzeitig funktioniert der Mikrocontroller als WiFi-Hotspot, der an einen Client auf Anfrage eine Webseite verschickt, die mit Hilfe von JavaScript alle 50ms aktualisiert wird. Übertragen werden u.A. die Sturzwahrscheinlichkeit, die wie im “Bikecrasher” beschrieben berechnet wird, und der Auslöser der Detektion.

Die Tests:

In einer Methode “prepareData()” werden die Sensordaten wie im folgenden einzeln beschrieben aufbereitet, sobald sie von der IMU ausgelesen werden. Sie werden entrauscht und zum Beispiel integriert, um vergleichbare Indizien für einen Sturz zu erzeugen. Die abstrahierten Rohdaten können nun gut mit Grenzwerten verglichen werden. Je mehr Grenzwerte überschritten werden, desto höher ist die Wahrscheinlichkeit, dass ein Sturz stattfindet.

Pitch/Roll:

Die Sensorwerte für die Winkelgeschwindigkeit werden vom DMP integriert, um einen Winkel zu erhalten. Die so berechneten Werte werden auf einen Offset addiert, der die Achsen des MPU-Koordinatensystems auf die des Fahrrads rotiert. Der Winkel wird dann von Rad zu Gradient umgerechnet und mit einem Grenzwert verglichen. Yaw wird dabei nicht berücksichtigt, da eine Rotation um diese Achse bei normalem Fahrverhalten schon mehr als 360° betragen kann.

GyroSmoothend:

$Value_n = Value_{n-1} * (1 - MovingAverageDecline) + Value_{sensorInput} * MovingAverageDecline$

wobei $0 < MovingAverageDecline < 1$

Mit einem exponentiell gleitenden Mittelwert werden die Werte des Gyrosensors verrechnet, um diese zu glätten. Die so erhaltenen Werte $Value_n$ werden mit einem Grenzwert verglichen.

accelIntegral / accelGravityIntegral:

Ähnlich wie beim GyroSmoothend wird ein gleitender Mittelwert gebildet, in diesem Fall aber über die Daten des Accelerometers, einmal mit den Rohdaten und einmal mit Werten in denen der DMP die Erdbeschleunigung entfernt. Die Beschleunigung in z-Richtung wird nur für die bereinigten Daten berücksichtigt.

accel_delta:

Dieser Test geht davon aus, dass ein Unfall als eine plötzliche Veränderung in der Kraft, die auf das Fahrrad wirkt zu interpretieren ist. Für diesen Test wird die Differenz in der Beschleunigung in alle Richtungen betrachtet. Dazu subtrahieren wir die neuen Werte für die Beschleunigung von den vorherigen um die Veränderung zu bestimmen und bilden daraus den Betrag um eine reine Veränderung der Beschleunigung ohne Richtungssinn zu erhalten. Wenn die Differenz einen bestimmten Grenzwert überschreitet, interpretieren wir das als Zeichen für einen Sturz.

Simulation_SingleParam.py

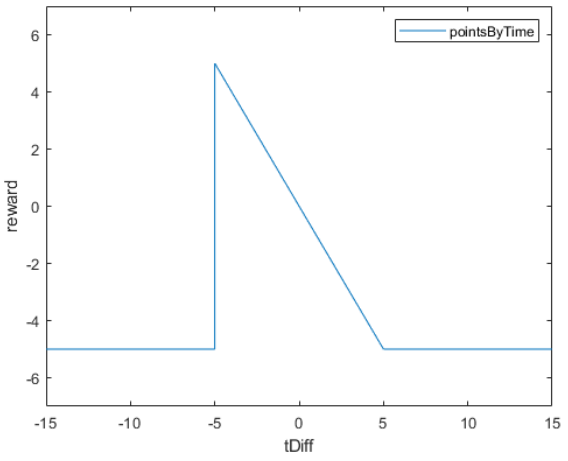
Abhängig von den beim Aufruf übergebenen Parametern hat dieses Programm drei Modi:

1. Ein Aufruf mit dem Namen eines bestimmten Grenzwerts (z.B.: “Python Simulation_SingleParam.py THRESHOLD_SMOOTH_GYRO_X”) optimiert nur diesen einen.
2. Ein Aufruf ohne weitere Parameter startet die sequentielle Optimierung aller Grenzwerte.
3. Ein Aufruf mit mehreren Parametern startet einen einzigen Durchlauf der Simulation ohne Optimierung, aber mit ausführlicher Ausgabe.

Die Methode “runSimulation(genome)” bekommt eine Sequenz von Grenzwerten, das genome, übergeben, die es zu bewerten gilt.
Ein einzelner Simulationsaufruf startet “Bikecrasher_singleParam.cpp” mehrmals, jeweils mit einer anderen Log-Datei als Übergabeparameter. Zudem wird die Sequenz aus Grenzwerten weitergegeben. Der Aufruf von “Bikecrasher_singleParam.cpp” liefert Sturzwahrscheinlichkeiten zu verschiedenen Zeitpunkten, die mit dem erwarteten Zeitpunkt des Sturzes aus der “_timestamps”-Datei verglichen werden. Die Ergebnisse werden mit einem Punktesystem bewertet, um die Güte vom genome zu bestimmen. So können verschiedene Versionen der genomes miteinander verglichen und insgesamt ein optimales gefunden werden. Zur Bewertung werden 2 Reward-Functions herangezogen:

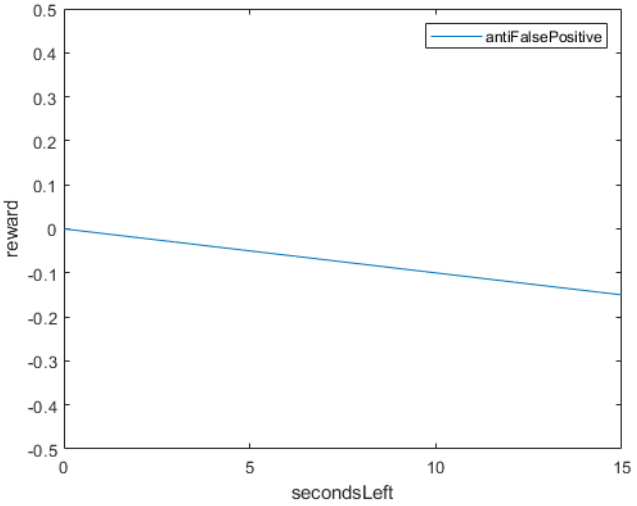
1. pointsByTime(tDiff):

Dieser Funktion wird die Differenz aus dem erwartetem Timestamp eines Sturzes und dem erkannten Timestamp in Sekunden übergeben. Ist diese Differenz zu groß (in der Abbildung rechts eine Differenz von über 5 Sekunden), so handelt es sich wahrscheinlich um eine fehlerhafte Erkennung und es gibt einen Punktabzug. Ansonsten, in einem Fenster um den gewünschten Zeitpunkt (Differenz=0), steigt die Punktzahl je früher ein Sturz erkannt wird.



2. antiFalsePositive(secondsLeft):

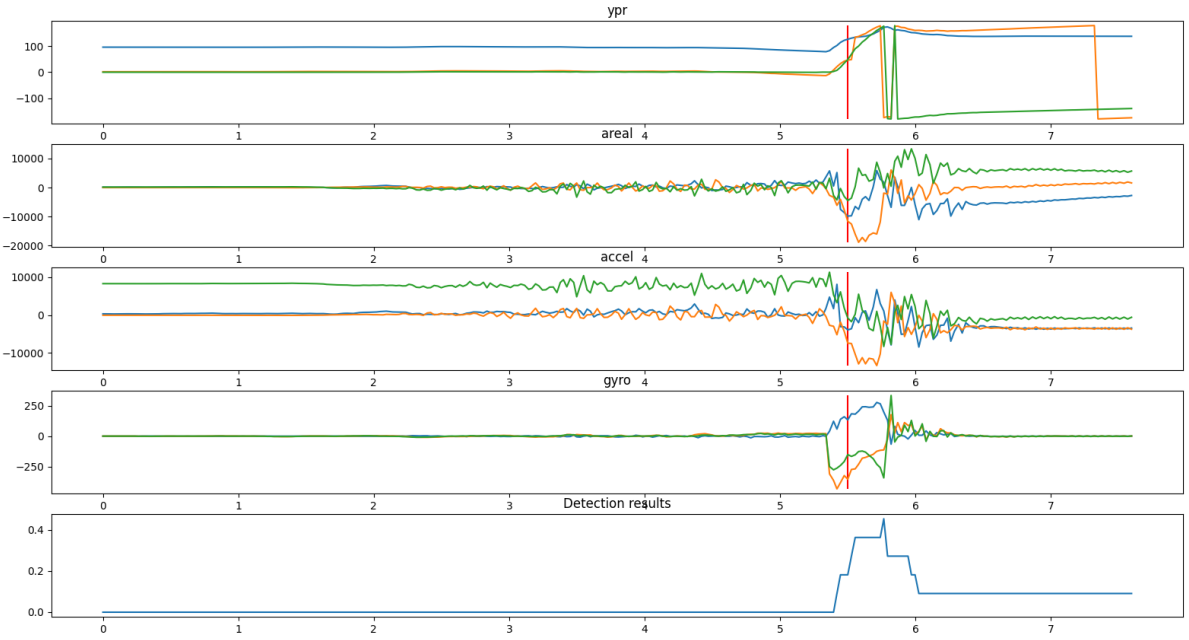
Falls kein Sturz in der Log-Datei verzeichnet ist, aber einer erkannt wird, wird diese Funktion statt “pointsByTime” aufgerufen.
Es wird die übrige Sekundenanzahl vom erkannten Sturz bis zum Ende der Log-Datei übergeben.
Ein solcher false-positive wird immer durch einen Punktabzug negativ bewertet, der größer wird, je früher der false-positive auftritt, um einen klaren Gradienten zur Verbesserung zu erzeugen.



Die Bewertung einer Sequenz von Grenzwerten durch den Simulationsaufruf wird genutzt, um die Grenzwerte zu optimieren. Dabei werden mit wachsender Auflösung verschiedene Werte getestet. Die Grenzwerte werden sequentiell und separat voneinander optimiert.
Eine vorherige Variante der Optimierung nutzte für die Optimierung einen evolutionären Algorithmus. Dieser erbrachte jedoch schlechte Ergebnisse, denn es kam zum overfitting und einzelne Grenzwerte wurden vernachlässigt, während andere übermäßig verfeinert wurden. Mehr Testdaten zu benutzten könnte ein overfitting reduzieren, jedoch steigt damit die Rechenzeit bei der Optimierung drastisch an. Die Grenzwerte tragen unabhängig voneinander zur Erkennung eines Sturzes bei. Ein evolutionärer Algorithmus betrachtet jedoch stets Kombinationen, weshalb der Algorithmus hier ungeeignet ist.

Resultate

Die Grafik zeigt die Vorhersage der Sturzwahrscheinlichkeit über den Verlauf der Zeit in Sekunden bei einem Lauf mit zuvor aufgezeichneten Daten. Der rote Strich markiert grob den Zeitpunkt, an dem von außen ein Sturz erkennbar war.



Plot eines Sturzes erstellt von Plotcrash.py

An den Sprüngen in der Wahrscheinlichkeit im untersten Graph zeigt sich, wie verschiedene Grenzwerte in schneller Abfolge überschritten werden. Man sieht, dass ein Sturz erfolgreich erkannt wurde. Außerdem geschieht die Detektion zeitnah, sodass für Erweiterungen die Möglichkeit besteht, vollständig auszulösen, bevor der Fahrer auf den Boden aufschlägt. Allerdings gibt es nur einen Bruchteil einer Sekunde Zeit für diese Maßnahmen, was die Optionen für Erweiterungsmodule einschränkt. Die Erkennung müsste eher geschehen, damit robuster reagiert werden kann.

Aussicht

Eine Kalibrierung für den Anbau des Sensors an ein neues Fahrrad ist nicht vorhanden und wäre eine sinnvolle Erweiterung. Bisher ist es bei Neuorientierung des Sensors erforderlich, neue Grenzwerte zu suchen, die eine abweichende Zuverlässigkeit haben könnten.
Bei Programmstart kalibriert “finalBikeCrashServer” die DMP, sodass die z-Achse mit der Gravitation ausgerichtet ist. Diese Kalibrierung gilt allerdings - anders als von uns angenommen - nur für die YPR-Werte. Damit schlug ein selbstgeschriebenes Kalibriertool “Calibration_RotateCoordSys.py” fehl, welches zusätzlich die Fahrtrichtung durch Rotation um die z-Achse mit einer Achse in Übereinstimmung bringen sollte.

Durch schnellere, genauere Hardware könnte ein Sturz besser erkannt werden. Ein fortgeschrittener Filter könnte außerdem die Vibrationen am Fahrrad (z.B.: Fahrt über Kopfsteinpflaster, hin und her kippen des Rades beim Fahren im Stehen) analysieren und diese unterdrücken. Durch das Herausrechnen störender Schwingungen könnte der $MovingAverageDecline$ des exponentiell gleitenden Mittelwert-Filter größer gewählt werden, was zu schnelleren Reaktionen führt.

Der durch den DMP bestimmte Winkel reagiert verglichen mit den anderen Sturzkriterien langsam auf Änderungen und dadurch schlagen die YPR-Grenzwerte später aus. Es ist möglich die Parameter des DMP anzupassen, um schneller die Winkel zu berechnen, allerdings taucht dann vermehrt Drift auf.

Auch die Kommunikation mit externen Erweiterungen, bei uns per HTTP-Requests realisiert, könnte schneller gestaltet werden. So könnte zum Beispiel UDP an Stelle von TCP als Netzwerkprotokoll verwendet werden oder komplett auf eine Verbindung mit Kabeln umgestiegen werden.