Bachelorprojekt

SensorBike
aka BikeCrasher

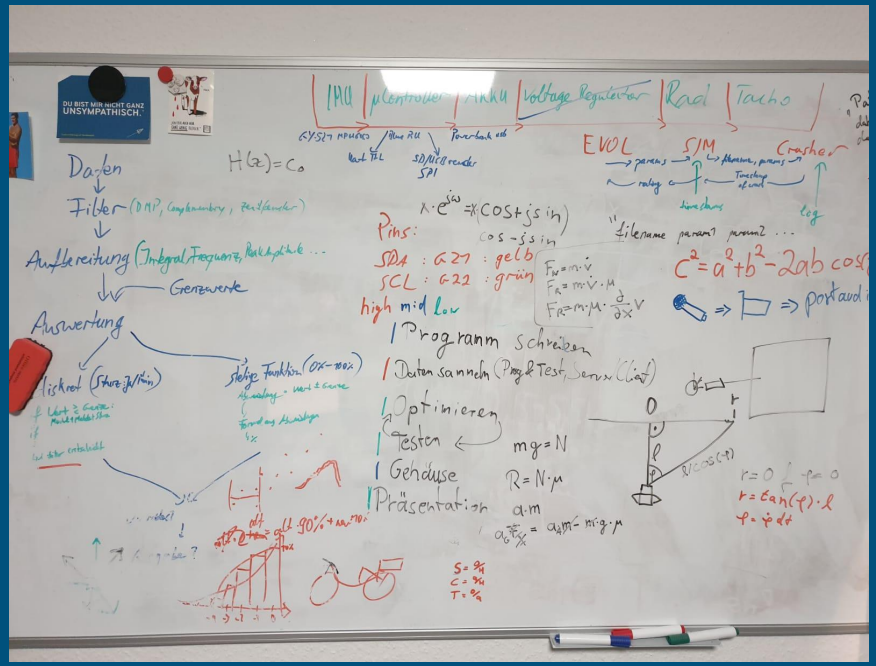
Zielsetzung

- Fahrradstürze mit IMU vorhersagen
- Um Module erweiterbare Plattform bilden

Vorgehensplan

- Treffen jeden Freitag
- Absprache über Whatsapp und Discord
- GitHub als Repository

https://github.com/JD1694/BachelorProject-BicycleCrasher



Methoden

Daten sammeln-> ESP32-WROOM-32 mit Modulo MPU-6050 ausstatten, Clientprogramm zum Auslesen/Aufbereiten der Sensorwerte/Timestamps für Sturz in Arduino/C++, Hostprogramm empfängt Daten und schreibt in CSV in Python---->Gesammelte Sensorwerte

Auswerten der Daten-> Umschreiben des Programms zum Sammeln der Sensorwerte, sodass mithilfe der CSV eine “Simulation” ablaufen kann in C++, Sensorwerte werden aufbereitet und mit verschiedenen Thresholds verglichen, um festzustellen, ob momentan ein Sturz stattfindet. Parameter erhält das Programm von einem evolutionären Algorithmus (verschiedene Ansätze, final bruteforce) in Python, der unterschiedliche Parameter anhand der Abweichung des detektierten Sturzes verglichen mit dem aufgenommenen Timestamp bewertet---->Parameter zum erkennen eines Sturzes

Erkennen eines Sturzes->Umschreiben des Arduino/C++ Codes von Client zu Hostanwendung, wertet mit Hilfe von aufbereiteten Sensordaten und bestimmten Parametern Situation aus und sendet Informationen über Sturzverhalten/Sensordaten an Client---->Sturz erkannt

Zwischenschritte->Vorbereiten des Fahrrads (Polstern empfindlicher Stellen, abbauen des Sattels/Lichts, Aufhängung für Dummy(verworfen)), Vorbereiten des ESP(Anbringen von MPU, Einbau in gepolsterte Abzweigdose als Gehäuse)

Benötigt: Foto präpariertes Fahrrad, ESP32+MPU6050 (in Abzweigdose), Videomaterial, Codesnippets (eval, main-loop 1,2,3 Vergleich, genetic algo), Fotos von Webseite auf Handy, Plots (genAlg, Vergleich Timestamp ergebnis), Schaubilder für Vorgehen (“Datenpfad”, Optimierungskette,)

Hardware

Mikrocontroller: ESP32

- integriertes Wi-Fi als Schnittstelle
- 160-240MHz
- FPU

IMU: GY521-MPU

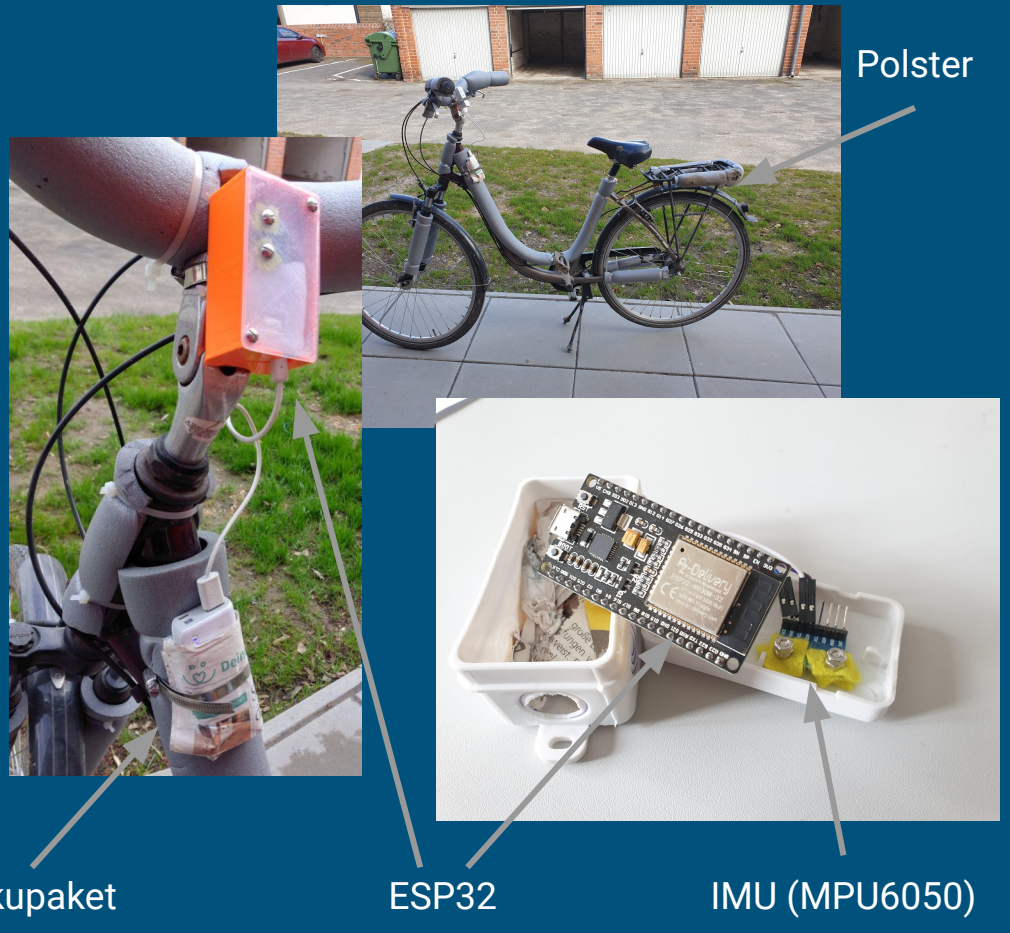
- 3-Achsen Gyroskop und Accelerometer
- DMP

Aufbau

Fahrrad gepolstert, Zerbrechliches abmontiert

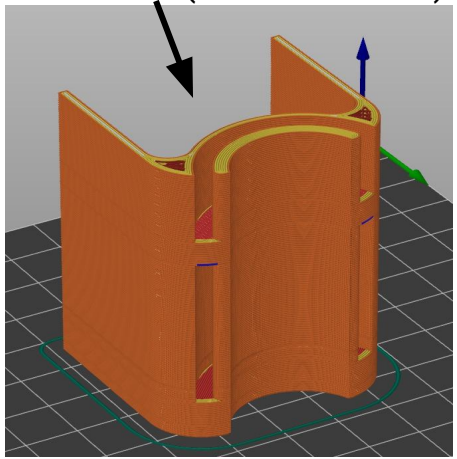
Elektronik in 3D-gedruckter Halterung montiert und gepolstert

Elektronik und Akkupaket an Lenker angebracht



TOREMOVE - Meine Online Bildbearbeitung

3D Modell (Schnittansicht)



Elektronik in 3D-gedruckter Halterung

Akkupaket

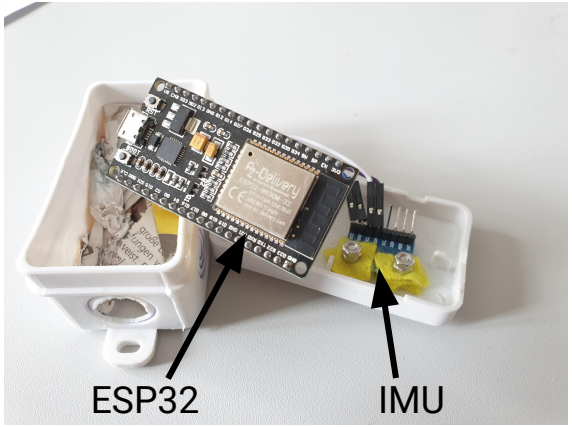
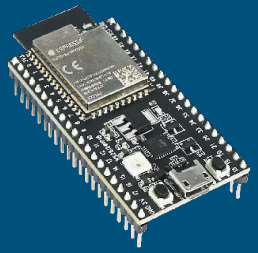


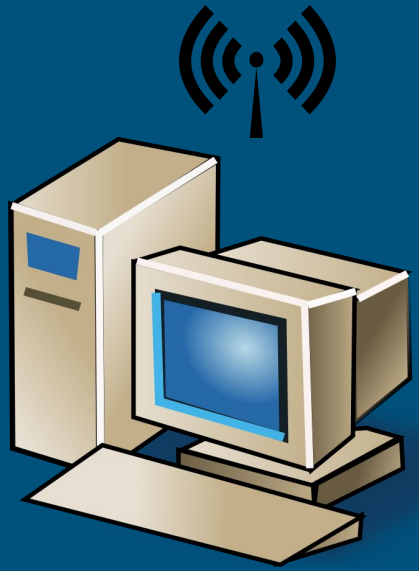
Schaubild: Daten sammeln

Crash-Tests & Normalfahrten



ESP32_ClientSendDMPData.ino

Fahrtdaten

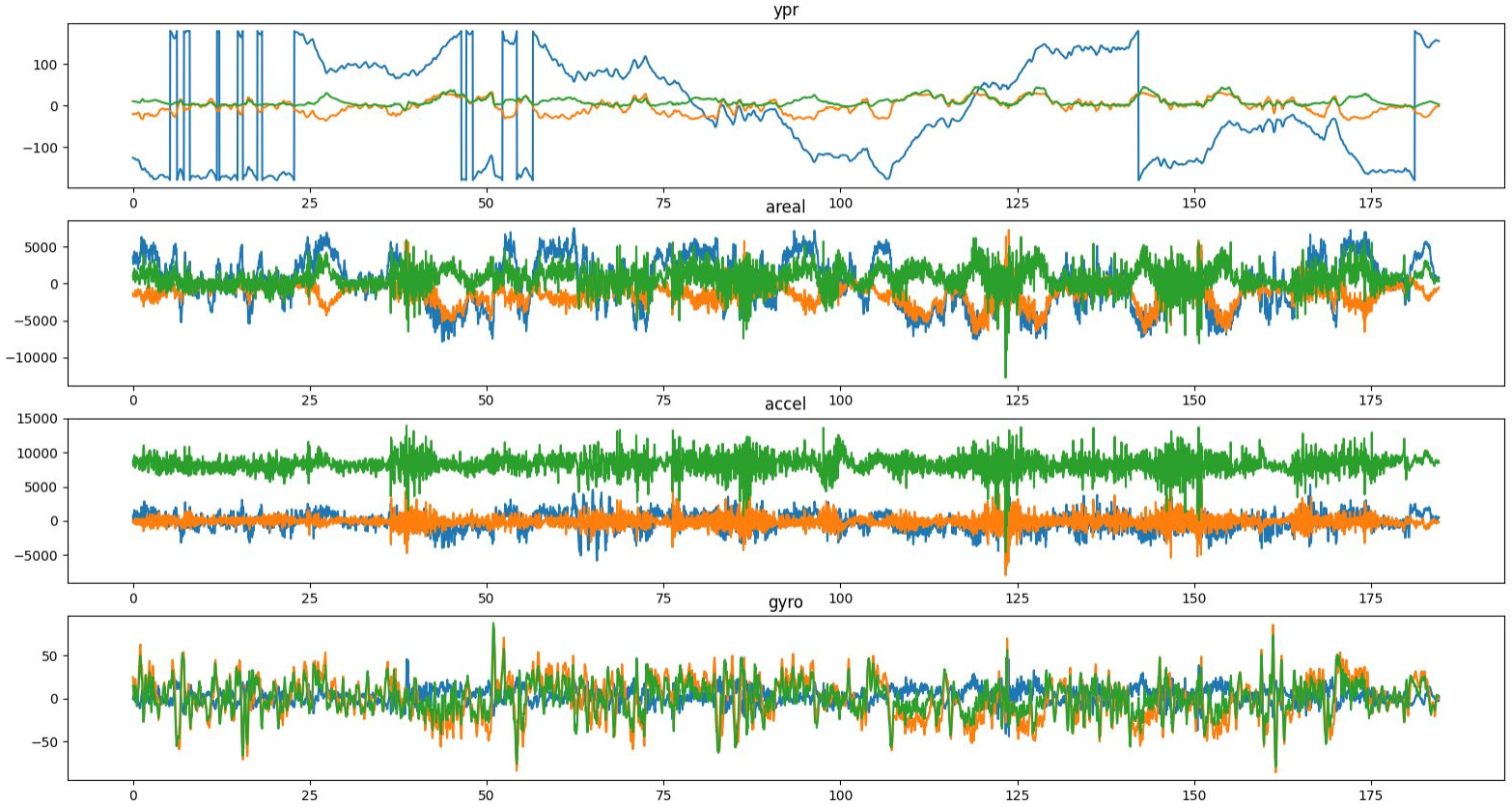


DataCollection_Server.py

Fahrtdaten

Timestamps

Logdatei: Normales Fahren

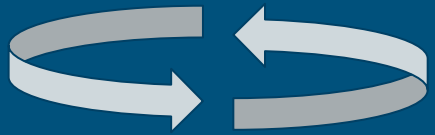


Vergleich mit bestimmten Grenzwerten

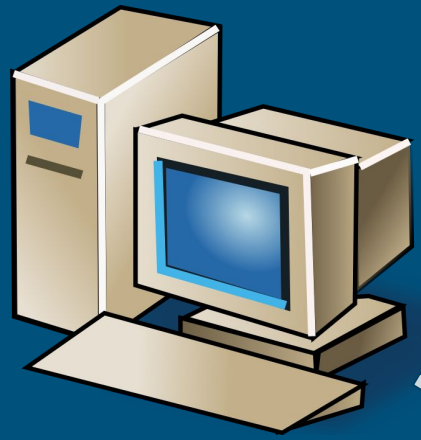
```
double evalDiscreteSteps() {
  /* Evaluate inputs into discrete Steps according to the number of thresholds triggered. Result is converted to Percent */
  crashCause = "";
  stepCount = 0
              // Absolute Angle
              + 1 * checkThreshold(abs(ypr [1] * 180 / M_PI), THRESHOLD_PITCH, "THRESHOLD_PITCH")
              + 1 * checkThreshold(abs(ypr[2] * 180 / M_PI), THRESHOLD_ROLL, "THRESHOLD_ROLL")

              // Rate of rotation
              + 1 * checkThreshold(abs(gyroSmoothend.x), THRESHOLD_SMOOTH_GYRO_X , "THRESHOLD_SMOOTH_GYRO_X")
              + 1 * checkThreshold(abs(gyroSmoothend.y), THRESHOLD_SMOOTH_GYRO_Y , "THRESHOLD_SMOOTH_GYRO_Y")
              + 1 * checkThreshold(abs(gyroSmoothend.z), THRESHOLD_SMOOTH_GYRO_Z , "THRESHOLD_SMOOTH_GYRO_Z")

              // Acceleration
              + 1 * checkThreshold(abs(accelIntegral.x), THRESHOLD_INT_ACCEL_X , "THRESHOLD_INT_ACCEL_X")
              + 1 * checkThreshold(abs(accelIntegral.y), THRESHOLD_INT_ACCEL_Y , "THRESHOLD_INT_ACCEL_Y")
              + 1 * checkThreshold(abs(accelIntegral.z), THRESHOLD_INT_ACCEL_Z , "THRESHOLD_INT_ACCEL_Z")

              // Direction of Gravity and Momentum
              + 1 * checkThreshold(abs(accelGravityIntegral.x), THRESHOLD_INT_GRAVITY_X , "THRESHOLD_INT_GRAVITY_X")
              + 1 * checkThreshold(abs(accelGravityIntegral.y), THRESHOLD_INT_GRAVITY_Y , "THRESHOLD_INT_GRAVITY_Y")

              // Change in Acceleration
              + 1 * checkThreshold(abs(accel_delta), THRESHOLD_ACCEL_DELTA , "THRESHOLD_ACCEL_DELTA")
              ;
  return stepCount / 11;
}
```

Schaubild: Trainieren

Aufgezeichnete Daten



Simulation und Evaluation

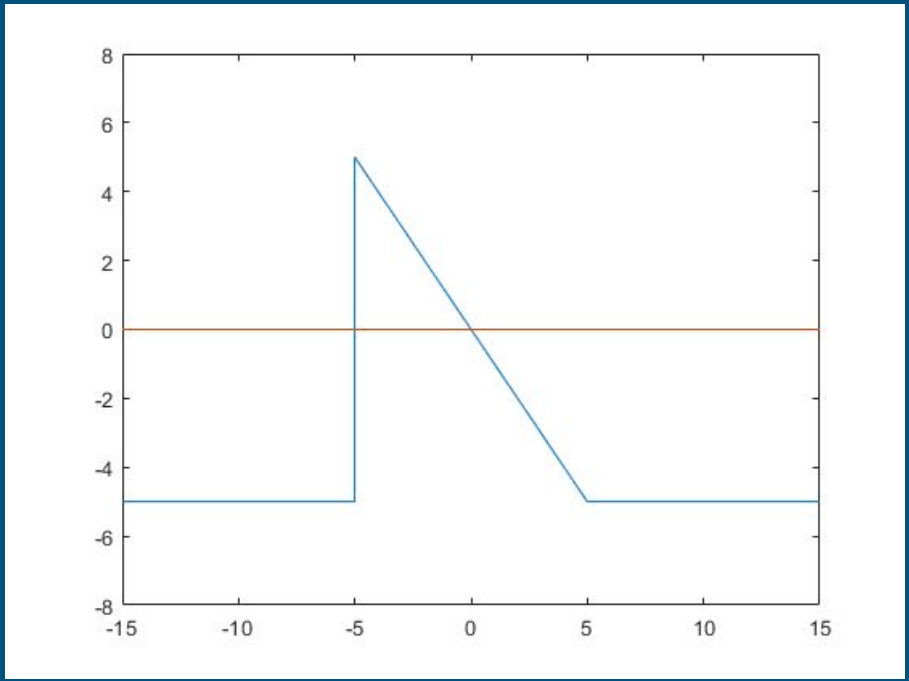


Bikecrasher.exe
Simulation_SingleParam.py

Parameter

Simulation Bewertung

Punkte abhg. von Abstand zu erwartetem Zeitpunkt



Abzug für false positives

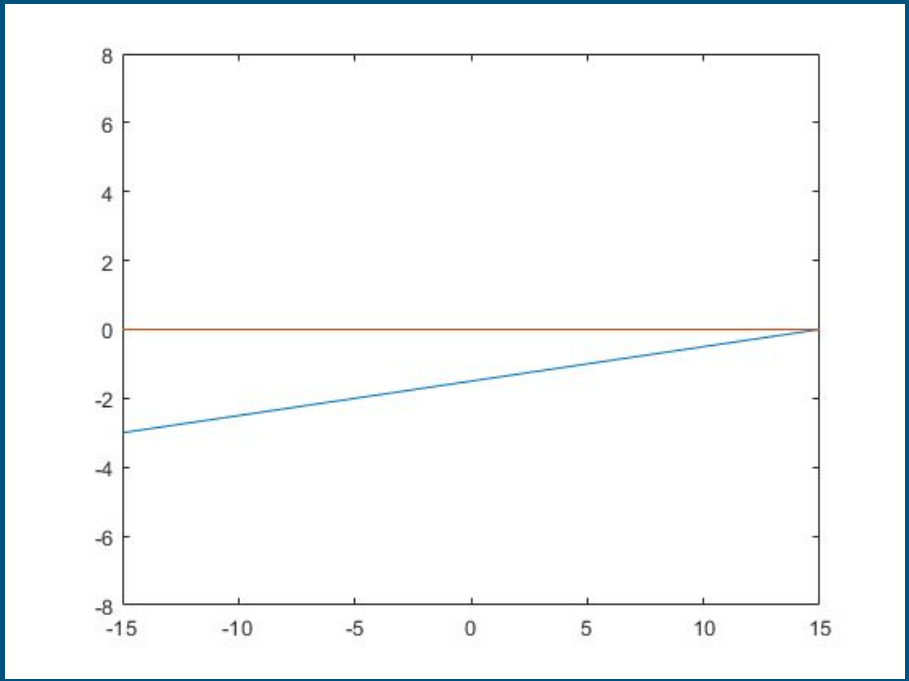
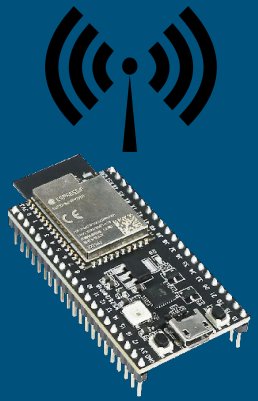


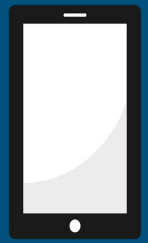
Schaubild: Final



finalBikeCrasherServer.ino

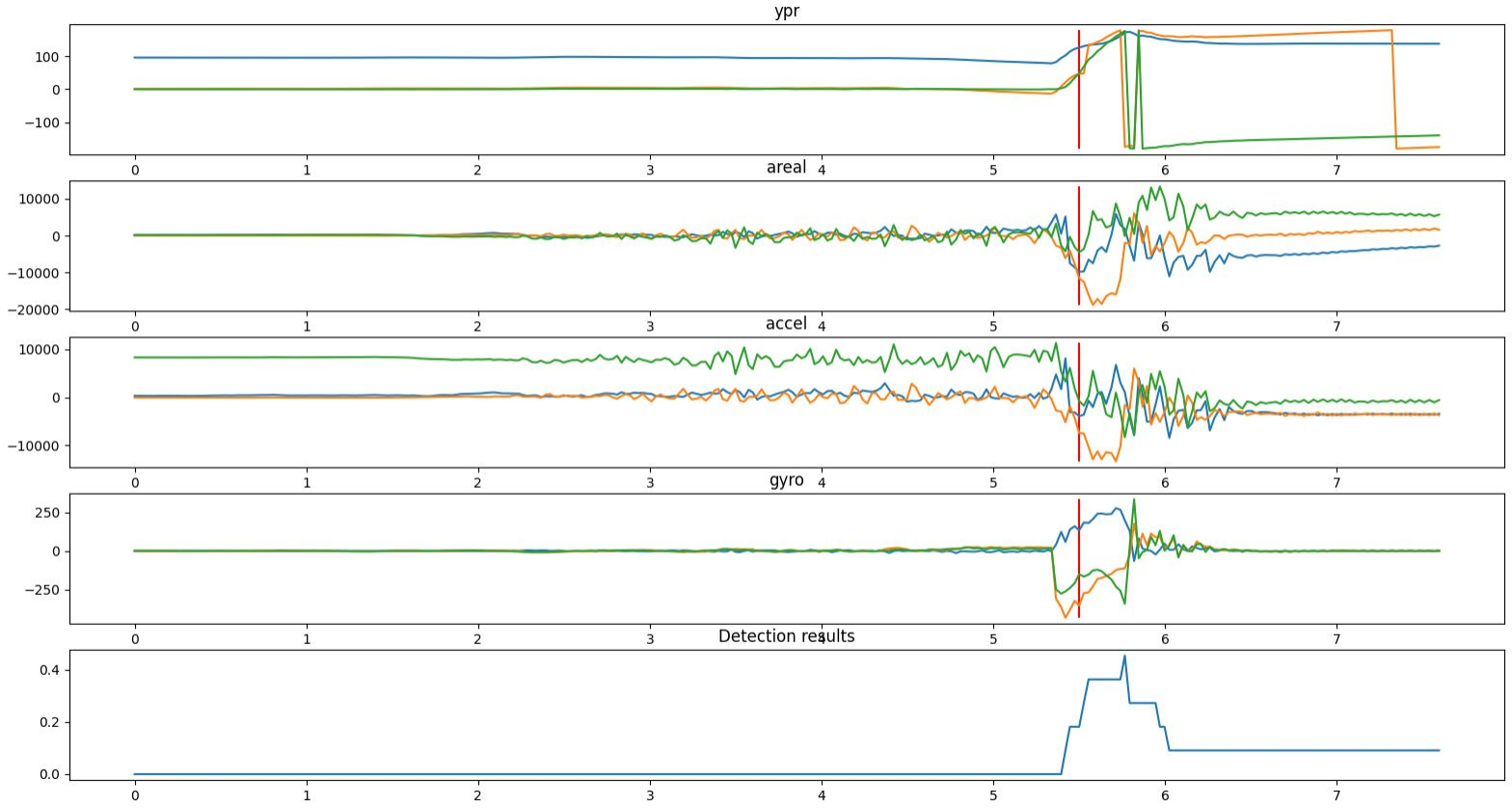
Crash?
Wahrscheinlichkeit & Auslöser

Erweiterungen
z.B.: Airbag, Aktive Handschuhe, ...



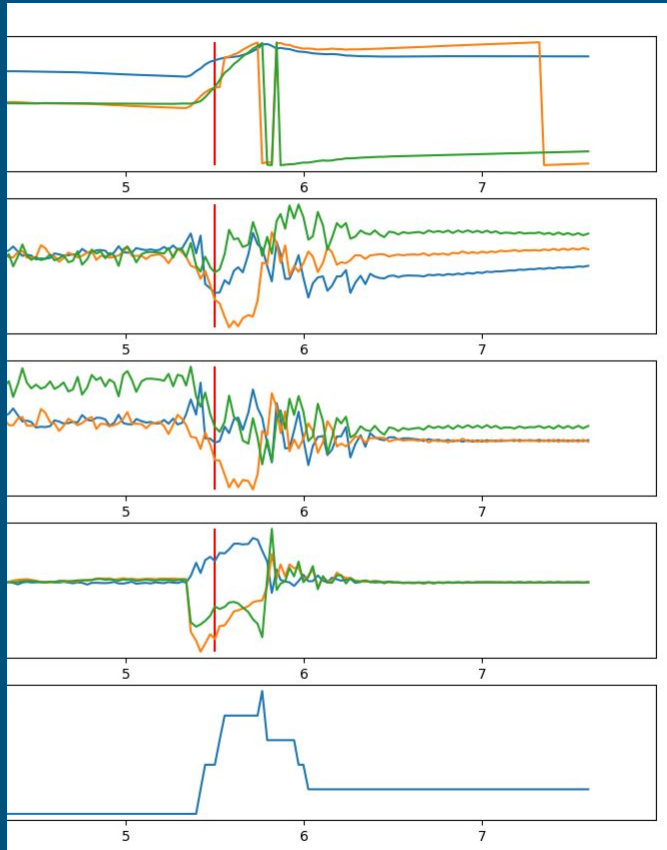
Website

Crash Erkennung



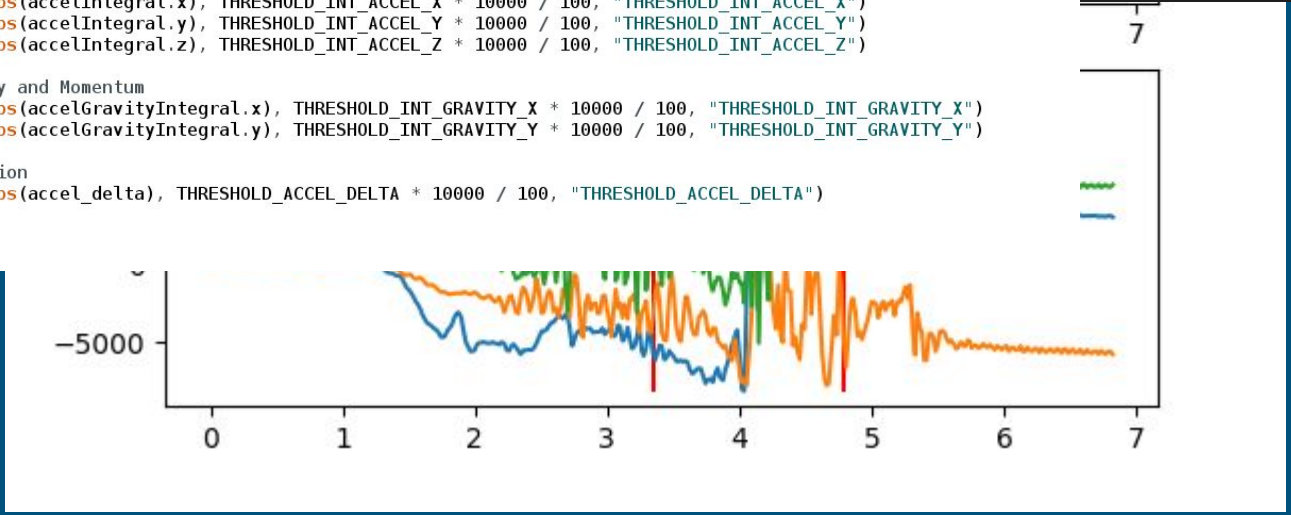
Resultate

- Robuste Detektion
- Zeitnahe Detektion
- Noch Potential für schnellere Detektion



```
for (auto  sensorData:sensorDataList){
  // Get sensor readings
  getSensorReadings(sensorData);

  // Prepare data, filter and convert to get more measurable
```

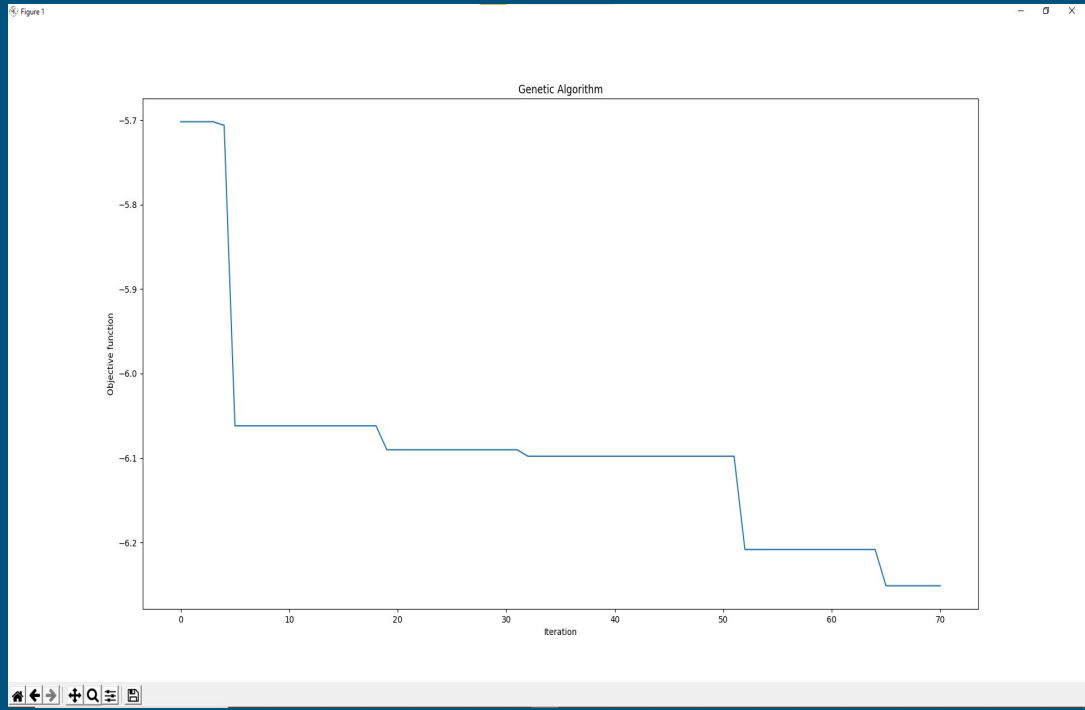
```
        ;*/
        // Absolute Angle
        + 1 * checkThreshold(abs(ypr_rot[1] * 180 / M_PI), THRESHOLD_PITCH * 45 / 100, "THRESHOLD_PITCH")///// change to rad for mor
        + 1 * checkThreshold(abs(ypr_rot[2] * 180 / M_PI), THRESHOLD_ROLL * 50 / 100, "THRESHOLD_ROLL")

        // Rate of rotation
        + 1 * checkThreshold(abs(gyroSmoothend.x), THRESHOLD_SMOOTH_GYRO_X * 200 / 100, "THRESHOLD_SMOOTH_GYRO_X")
        + 1 * checkThreshold(abs(gyroSmoothend.y), THRESHOLD_SMOOTH_GYRO_Y * 200 / 100, "THRESHOLD_SMOOTH_GYRO_Y")
        + 1 * checkThreshold(abs(gyroSmoothend.z), THRESHOLD_SMOOTH_GYRO_Z * 200 / 100, "THRESHOLD_SMOOTH_GYRO_Z")

        // Acceleration
        + 1 * checkThreshold(abs(accelIntegral.x), THRESHOLD_INT_ACCEL_X * 10000 / 100, "THRESHOLD_INT_ACCEL_X")
        + 1 * checkThreshold(abs(accelIntegral.y), THRESHOLD_INT_ACCEL_Y * 10000 / 100, "THRESHOLD_INT_ACCEL_Y")
        + 1 * checkThreshold(abs(accelIntegral.z), THRESHOLD_INT_ACCEL_Z * 10000 / 100, "THRESHOLD_INT_ACCEL_Z")

        // Direction of Gravity and Momentum
        + 1 * checkThreshold(abs(accelGravityIntegral.x), THRESHOLD_INT_GRAVITY_X * 10000 / 100, "THRESHOLD_INT_GRAVITY_X")
        + 1 * checkThreshold(abs(accelGravityIntegral.y), THRESHOLD_INT_GRAVITY_Y * 10000 / 100, "THRESHOLD_INT_GRAVITY_Y")

        // Change in Acceleration
        + 1 * checkThreshold(abs(accel_delta), THRESHOLD_ACCEL_DELTA * 10000 / 100, "THRESHOLD_ACCEL_DELTA")
        ;

return stepCount / 11;
```



Verworfene Ideen

Genetischer Algorithmus:

- Optimiert Kombination aus Parametern
- Parameter voneinander unabhg.
- GA fixiert auf einzelnen Threshold



Verworfene Ideen

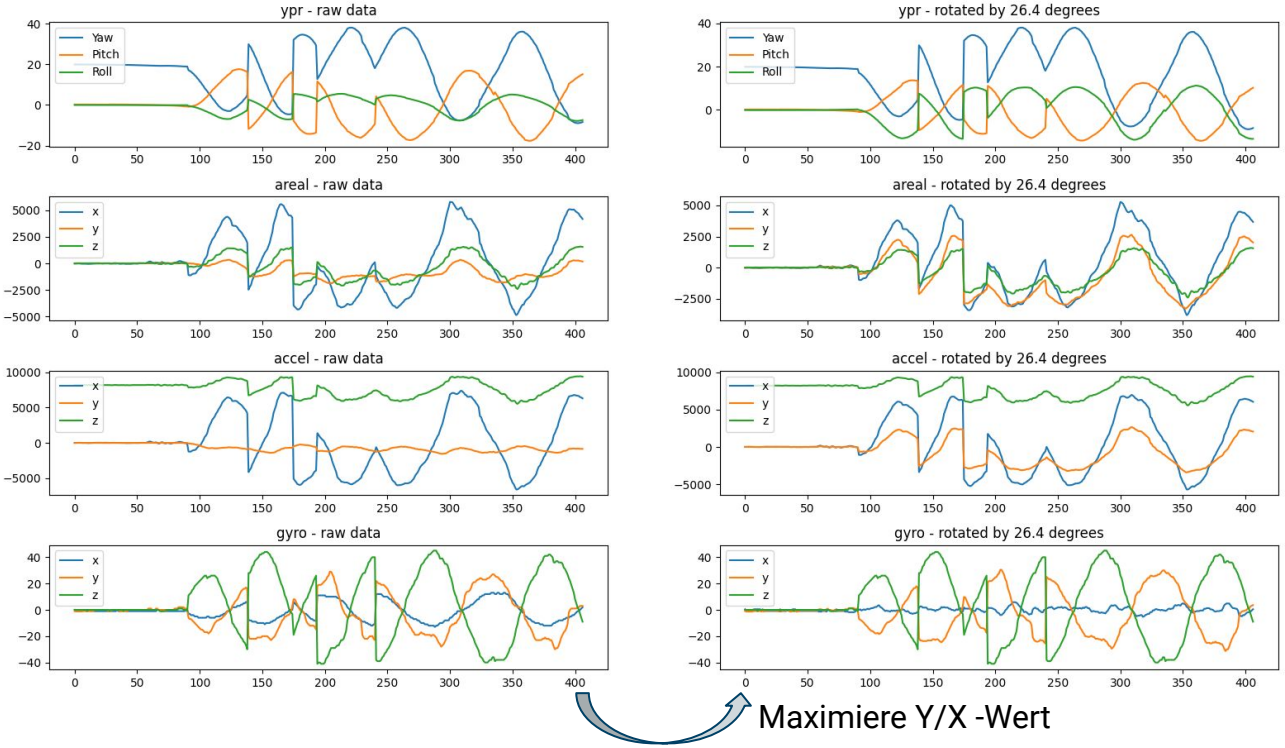
Automatische Kalibrierung:

- Annahme: Z-Achse per onboard tool kalibriert
- → Bringe Y-Achse mit Fahrtrichtung in Übereinstimmung

1. Rad kippen um gewünschte Achse (hier Y)
2. Calibration_RotateCoordSys.py findet optimalen Winkel (Maximiert Y/X -Gyro-Wert)
3. Rotation aller Sensordaten um Z-Achse.

Verworfene Ideen

Bekannte Bewegung: Rollen



Maximiere Y/X -Wert

Aussicht

- Automatische Kalibrierung des Koordinatensystems
- Bessere Hardware (genauere Sensorik, mehr Rechenleistung)
- Bessere Filter (Unterdrückung von Störfrequenzen)
- Modellbasierte Detektion
- Erweiterungsmodule

Bildquellen (vom 24. März 2021)

PC Symbol:
https://openclipart.org/detail/17668/rgtaylor_csc_net_computer

Wlan Symbol:
https://openclipart.org/image/800px/294681

ESP32 Symbol:
https://www.antratek.de/esp32-s2-saola-1r-development-board

Handy Symbol:
https://lh3.googleusercontent.com/proxy/vMZ1PiInd5acaG9ijB2wiPms7eHR_iZPyCzsDmdI722xxvKpAT4dcUcjayGucv1MrRh_qFnxJSEfcPoqiRynX-tlRHMsuBfKhOO7-ZY0d8ANMa3N4fdIwWqI0QdHlNTL-8zzpL_s=w1920-h1005

CSV Datei Symbol:
https://lh6.googleusercontent.com/proxy/ti1-LIsHvf6kXou6rNmTxxt8GNjIud2KZ4V4aDtOjSlRAGDR9T6UI4LZAk7X-EYay166G0FYGhQa98OM4alVMh5yPy7UFG077C5NHlGwNfMC1xehFpg9Lg=w1920-h1006