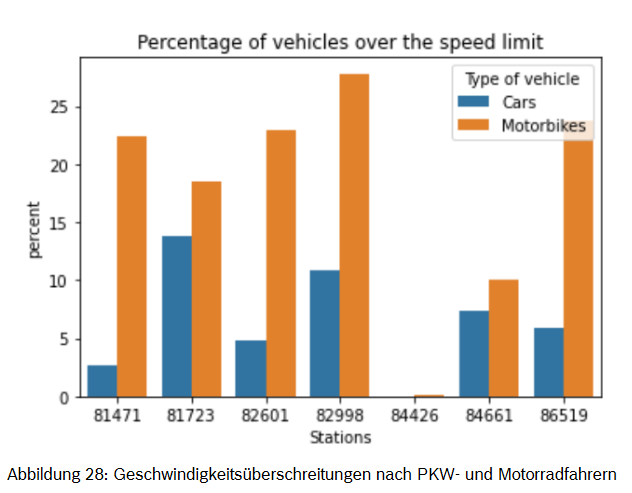
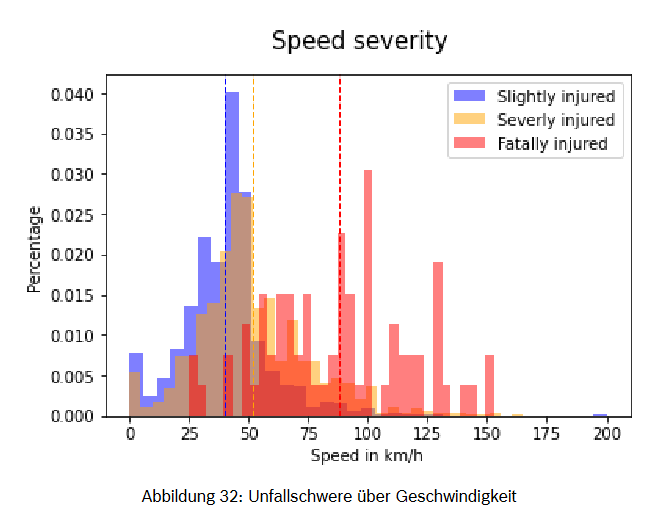
# Grundlagen

Die Unfallerkennung ist wichtig wegen folgenden Gründen.

* Geschwindigkeitsüberschreitung bei Motorrädern viel häufiger als bei Autos



(1, kein Datum)

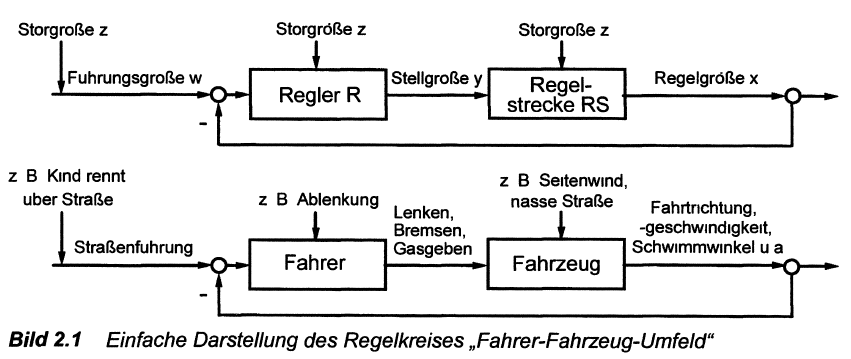


(1, kein Datum)

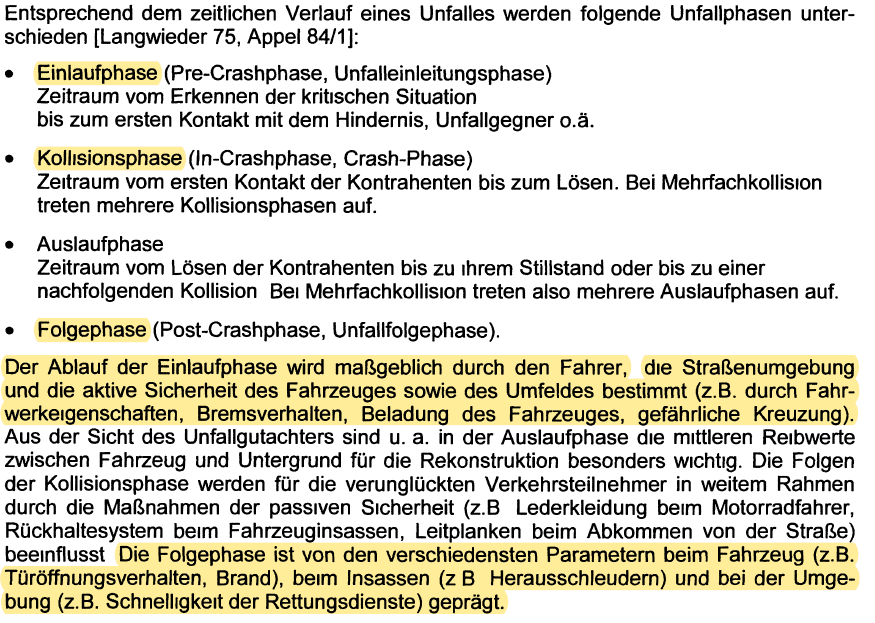
## Unfall

### DEFINITIONEN: UNFALL

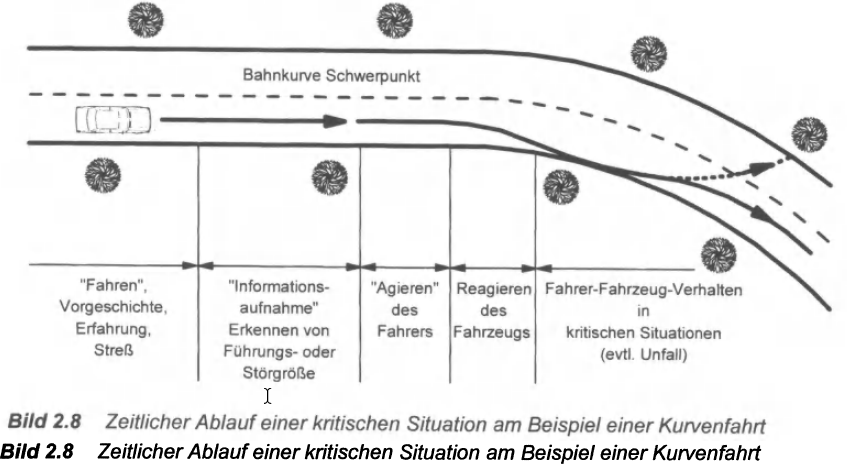
Unfälle Im Straßenverkehr lassen sich In Ihrem Ablauf im Allgemeinen nur unter Betrachtung des geschlossenen Regelkreises "Fahrer-Fahrzeug-Umfeld" erklären, analysieren und beurteilen. Denn sowohl die Ursachen als auch die Auswirkungen eines Unfalls sind fast niemals allein auf eine Komponente des Regelkreises zurückzuführen, sondern sind das Ergebnis eines interaktiven Zusammenwirkens der drei Komponenten. Fast immer sind daher Unfälle in Ihrem Entstehen auf die Verkettung mehrerer Ursachen (z.B. Blendung durch entgegenkommenden Verkehr und dunkle Kleidung eines Fußgängers) und in ihren Auswirkungen auf das wechselseitige Verhalten von mehreren Umständen (z.B. Tragen von Schutzhelmen, Airbag-Auslösung, Aufprall auf Leitplanken oder Baum) zurückzuführen. (2, kein Datum)



### ZEITLICHE PHASEN DES UNFALLS



Zur Einlaufphase zeigt das Bild 2.8 am Beispiel einer Kurvenfahrt vereinfacht den Ablauf einer kritischen Situation, die zum Unfall und eventuell zur Kollision führen kann, aber nicht muss. Der Fahrer erkennt zu einem bestimmten Zeitpunkt eine kritische Situation, unabhängig davon, ob es zu einem Unfall oder nur zu einem Beinahe-Unfall kommt. Nach dem Erkennen der kritischen Situation wird der Fahrer entscheiden, welche Maßnahmen er einleiten wird, um diese abzuwenden. Dabei wird er auf vorliegende Erfahrungen zurückgreifen und eine zur Abwehr der kritischen Situation geeignete Handlung einleiten. Das Fahrzeug reagiert auf Aktionen des Fahrers, so dass es zu einer Interaktion von Fahrer und Fahrzeug kommt, die zu einem Unfall führen kann.



# App-Entwicklung:

Die Entwicklung mobiler Apps ist der Prozess zur Erstellung von Software für Smartphones und digitale Assistenten, am häufigsten für Android und iOS. Die Software kann auf dem Gerät vorinstalliert, aus einem mobilen App Store heruntergeladen oder über einen mobilen Webbrowser aufgerufen werden. Eine der bekannten Sprachen in der App-Entwicklung ist C.

C ist eine leistungsstarke Programmiersprache, mit der Anwendungen in mehreren Bereichen erstellt werden können, von einfachen Taschenrechnern und Apps bis hin zu Videospielen. C ist eine Sprache auf niedriger Ebene, dies bietet Geschwindigkeit und eine weitaus bessere Möglichkeit zur Speicherverwaltung.

Für die Generierung eines C-Codes aus Simulink-Modellen wird der in Matlab/Simulink integrierte C/C++-Coder verwendet.

## C/C++-Coder:

Hier handelt es sich um Simulink Coder, der zur Generierung von C/C++-Code aus Simulink-Modellen verwendet wird. Der C/C++-Code-Generator wird für Rapid Prototyping, Hardware-in-the-Loop-Tests, Simulationsbeschleunigung oder einfach als ausführbare Datei zur Ausführung außerhalb von MATLAB und Simulink verwendet.

Diese Codegenerierung ist der Prozess der Generierung von Low-Level-Code direkt aus einer High-Level-Programmiersprache oder Modellierungsumgebung.

# Aktueller Algo:

### GroundHit-Analyse

*AccEnergyStXYintern = AccImpulseSTX^2+ AccImpulseSTY^2*

*GH\_ScalexGHEnergyThreshold* = *GroundHitEnergyThreshold* \* *GroundHitScaleFactor*

Wobei *GroundHitEnergyThreshold* bei hohen *TO\_curAngle\_deg* sinkt und *GroundHitScaleFactor* bei hohen *GyrVarSum* sinkt.

GH wird erkannt (GroundHitFire = 1), wenn:

* (RideFire = 2 || RideFire = 0) &&

AccEnergyStXYintern >= ScalexGHEnergyThreshold &&

TO\_curAngle\_deg >=45°

* RideFire = 1 &&

AccEnergyStXYintern >= ScalexGHEnergyThreshold &&

TO\_curAngle\_deg >=40°

Beispiele

* Kein GroundHit:

Daten aus: C:\Users\dao1wa2\Bosch Group\Connected Life - [Pocketmode@HC\Messdaten\20220811\_Hosentasch\_FussAnAmpelRunter\IMU\_logFile\_mb\_android\_samsung-SM-G973F\_lib1.1.2\_crcl4.0003\_mcal1.2005\_2022-08-11T16-16-05.665+0200\_v13.csv](mailto:Pocketmode@HC\Messdaten\20220811_Hosentasch_FussAnAmpelRunter\IMU_logFile_mb_android_samsung-SM-G973F_lib1.1.2_crcl4.0003_mcal1.2005_2022-08-11T16-16-05.665+0200_v13.csv)

Die richtige Kalibrierung nicht vergessen:

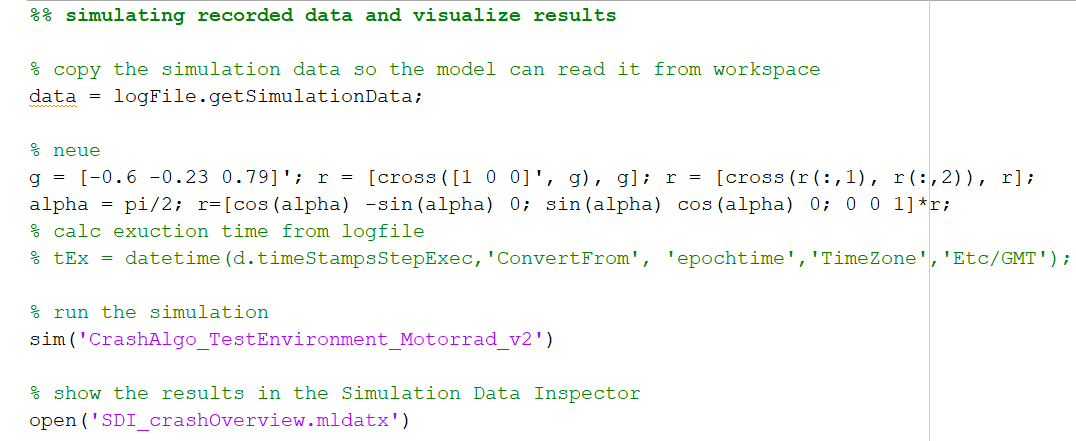


Abbildung 12: Eine manuelle Kalibrierung wird manuel mitgegeben.

Um eine (falsche) Nachkalibrierung zu verhindern, wurde das im Modul deaktiviert.

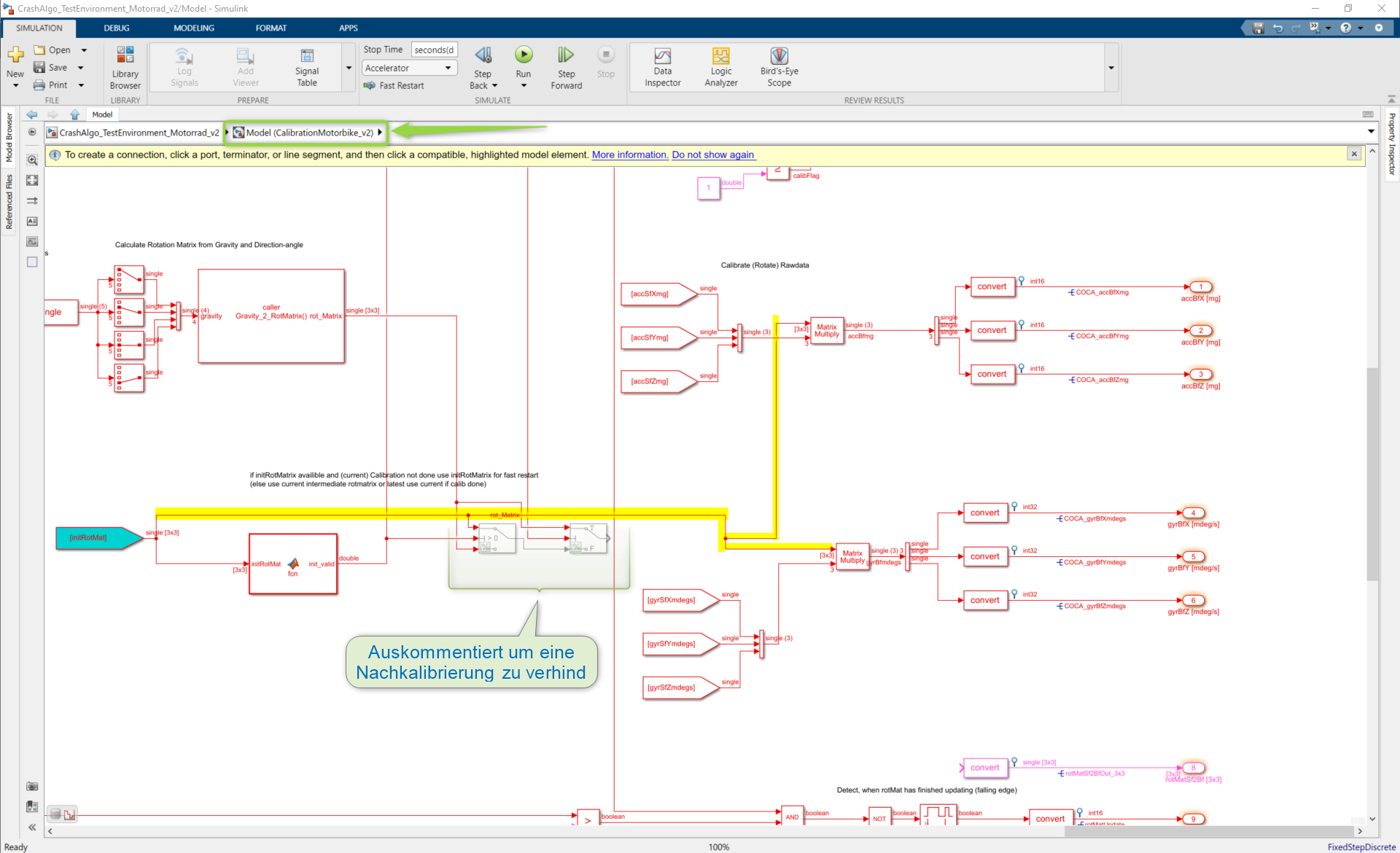


Abbildung 13: Simulink - Nachkalibrierung deaktiviert

In den Abbildungen (14, 15) ist der Fall vorgestellt, in dem kein Groundhit erkannt wurde. Während dieser Fahrt hat kein Unfall stattgefunden. Es wurde allerdings mehrere Szenarien getestet, die im Pocketmode vorkommen können. Z.B. laufen mit einem aktivierten Unfallerkennung oder beim Fahren an der Ampel bremsen und den Fuß auf dem Boden runterstellen mit dem Smartphone in der Hosentasche.

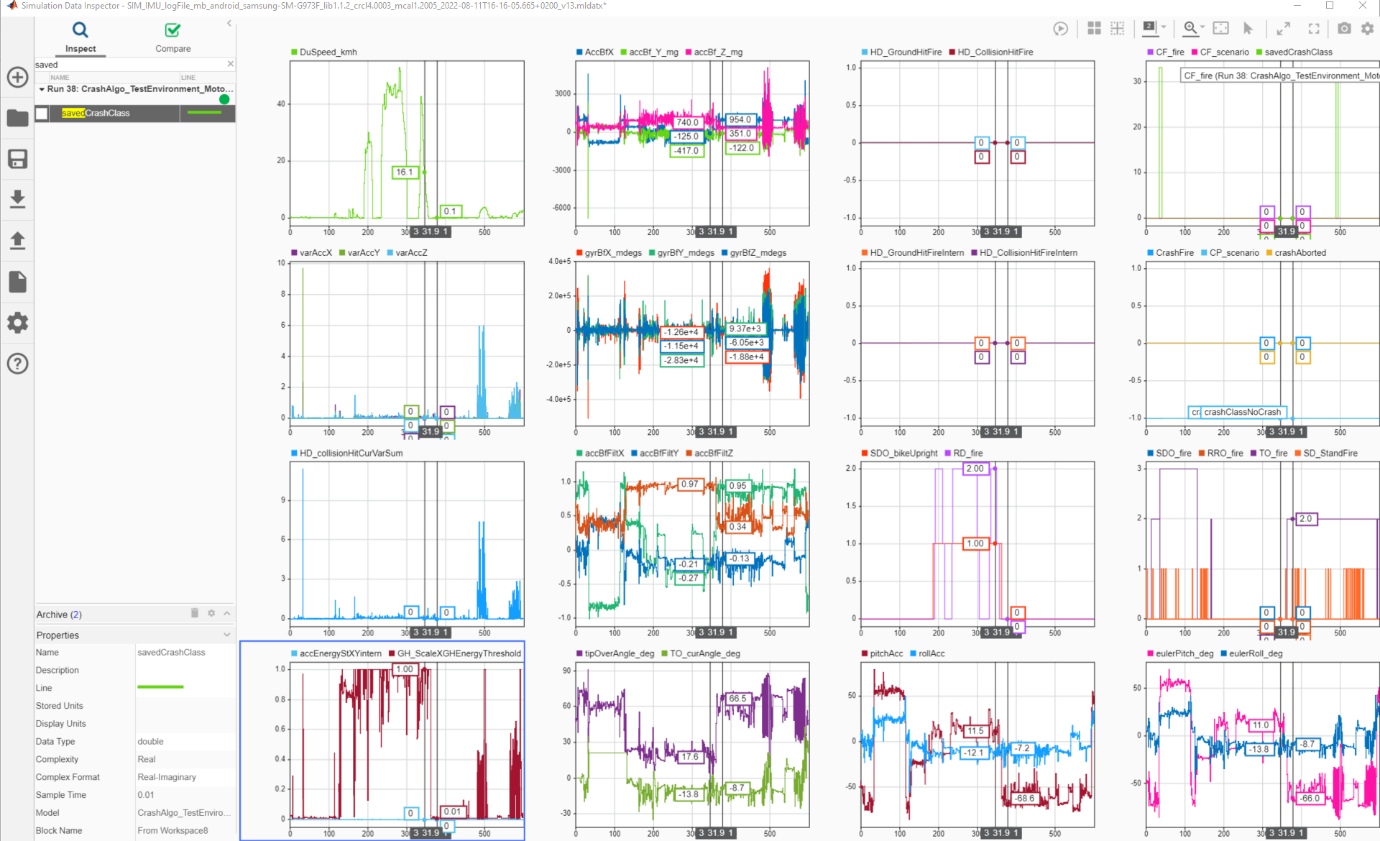


Abbildung 14: Testfahrt - no GroundHit - Full view

In der Abbildung 14 sind die simulierten Daten der Testfahrt dargestellt. Es ist zu bemerken, dass die GH\_SaleXGHEnergythreshold von dem Winkel (tipOverAngle\_deg) abhängig ist. Bei einem hohen Winkel sinkt die SaleXGHEnergythreshold (siehe die linke unterste Darstellung der Abbildung 14). Zum Zeitpunkt 346 ist die tipOverAngle\_deg = 17,6 und SaleXGHEnergythreshold = 1. Zum Zeitpunkt 391 ist die tipOverAngle\_deg = 66,5 und SaleXGHEnergythreshold ist auf 0.01 gesunken ist.

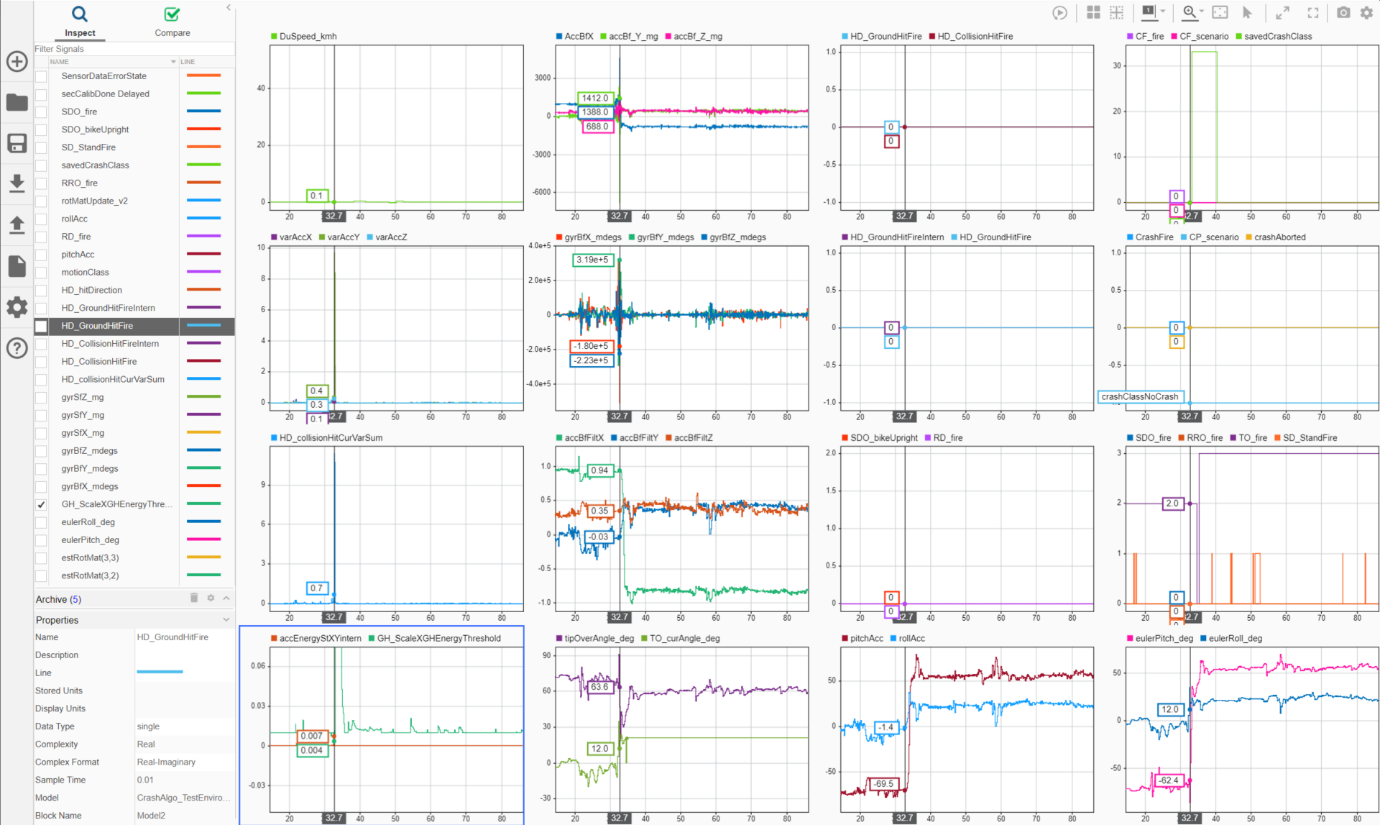


Abbildung 15: Testfahrt - no GroundHit - auf die EnergyThreshold und die tatsächliche GroundHitEnergy gezoomt

In der Abbildung 15 ist zum Zeitpunkt 32,7 s Folgendes abzulesen:

* RideFire = 0, keine Fahrt
* accEnergyStXYintern = 0,007
* GH\_ScalexGHEnergythreshold = 0,004
* TO\_curAngle\_deg = 12°
* eulerPitch\_deg springt von ca. -60° auf ca. +60° (Das handy wurde wahrscheinlich um 180° gedreht. Das könnte auch durch die Acc-Daten bestätigt werden)

An der Stelle ist accEnergyStXYintern > GH\_ScalexGHEnergythreshold, was eigentlich einen GroundHit-Alarm auslösen soll. (Während des Testens hat das Smartphone an der Stelle einen falsch-postiven Alarm erkannt) Grund dafür ist, dass die Kalibrierung nicht 100% richtig war.

In den simulierten Daten wurde keinen Unfall erkannt, da die Kalibrierung richtig war, und der TO-Winkel (TO\_curAngle\_deg) nicht über 45° ist.

* Mit GroundHit:

DB-ID: 2806. realCrash:

Die Abbildungen (16, 17)



Abbildung 16: Realcrash ID: 2806 - Full view - Mit GroundHit

In der Abbildung 16 sind die simulierten Daten einer echtfahrt mit einem Unfall dargestellt. Es ist zu bemerken, dass die GH\_SaleXGHEnergythreshold von dem Winkel (tipOverAngle\_deg) abhängig ist. Bei einem hohen Winkel sinkt die SaleXGHEnergythreshold (siehe die linke unterste Darstellung der Abbildung 15).



Abbildung 17: Realcrash ID: 2806 - auf die EnergyThreshold und die tatsächliche GroundHitEnergy gezoomt - Mit GroundHit

In der Abbildung 17 sind die Daten eines Unfalls (ID: 2806) dargestellt und auf die Unfallphase gezoomt. In der Abbildung ist zu erkennen, dass die GH\_SaleXGHEnergythreshold bei einem hohen Winkel (tipOverAngle\_deg) sinkt. An der Stelle wo die accEnergyStXYintern > GH\_SaleXGHEnergythreshold ist (ca. 591,35 s), ist der Winkel fas gleich 45° -> dadurch löst ein Alarm der GroundHit aus. Das entspricht die Erwartungen.

DB-ID: 2546. realCrash:

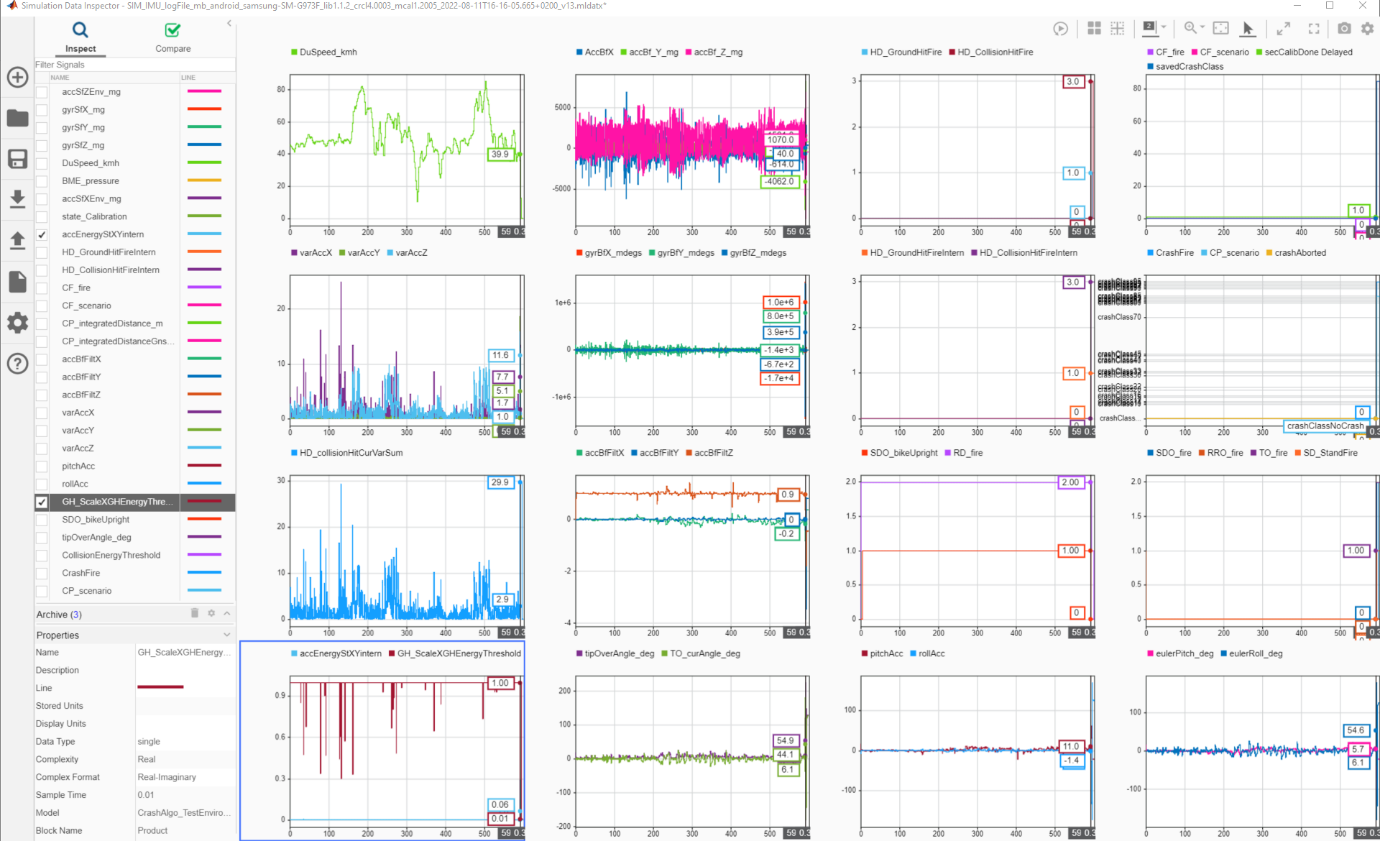


Abbildung 18: Realcrash ID: 2546 - Full view - Mit GroundHit

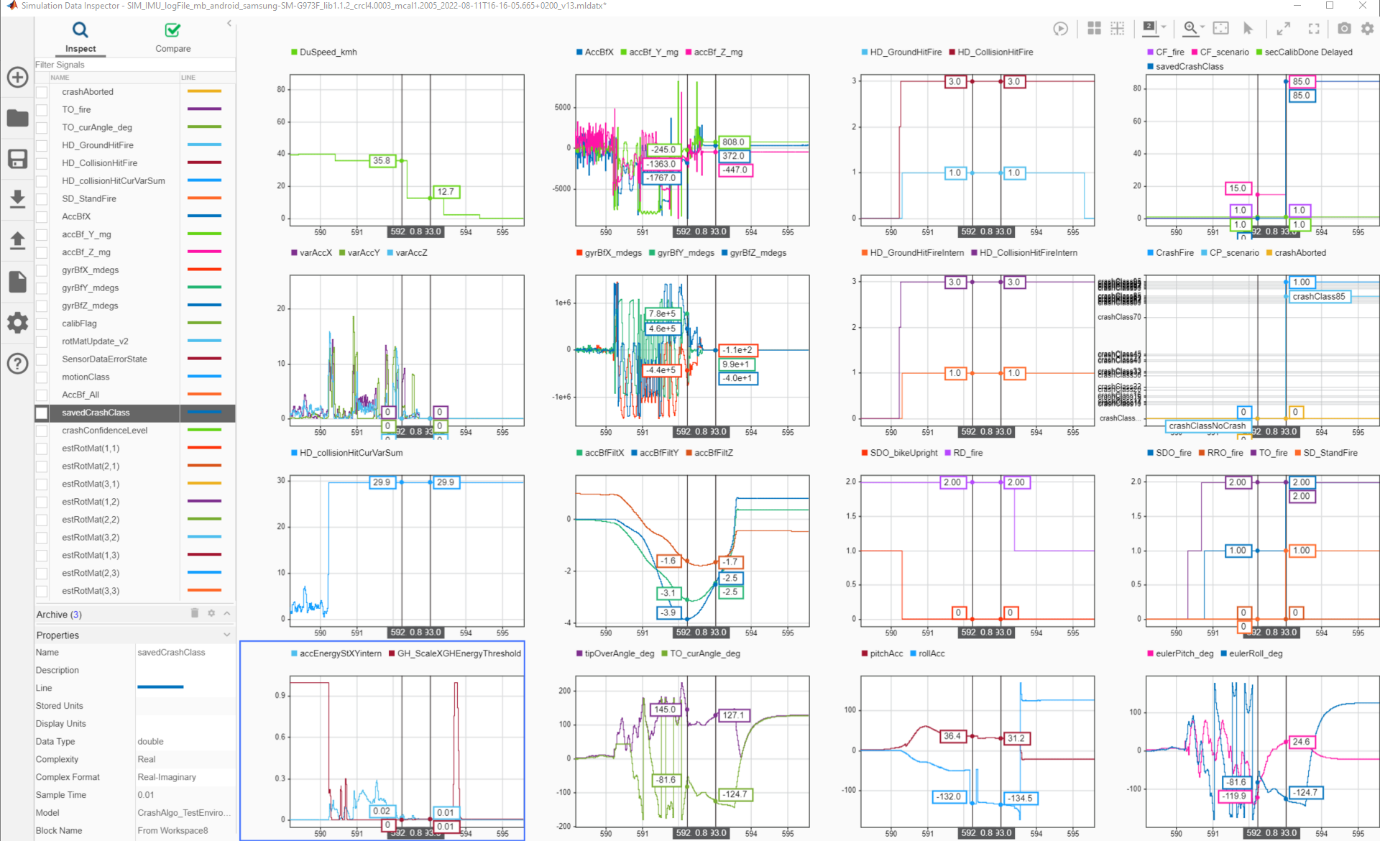


Abbildung 19: Realcrash ID: 2546 - auf die EnergyThreshold und die tatsächliche GroundHitEnergy gezoomt - Mit GroundHit

### CollisionHit-Analyse:

Analog zum GroundHit-Analyse!

# Haupt

Ein Modell erstellen, um das Laufen zu erkennen:

Das hilft, viele Falsche Auslösungen zu vermeiden.

### 1.Idee: Alle Peaks aufzählen

Einen Zähler zu implementieren. Das soll alle Peaks aufzählen mit der Hoffnung, dass es Unterschied zwischen Laufen und Fahren zu erkennen ist.

Das LaufFrequenz ist zwischen 0.5-4 Hz. Das Fahren ist über 20 Hz.

Es wurde einen Testbeispiel mit einem Sinussignal gebaut und das Prinzip getestet.

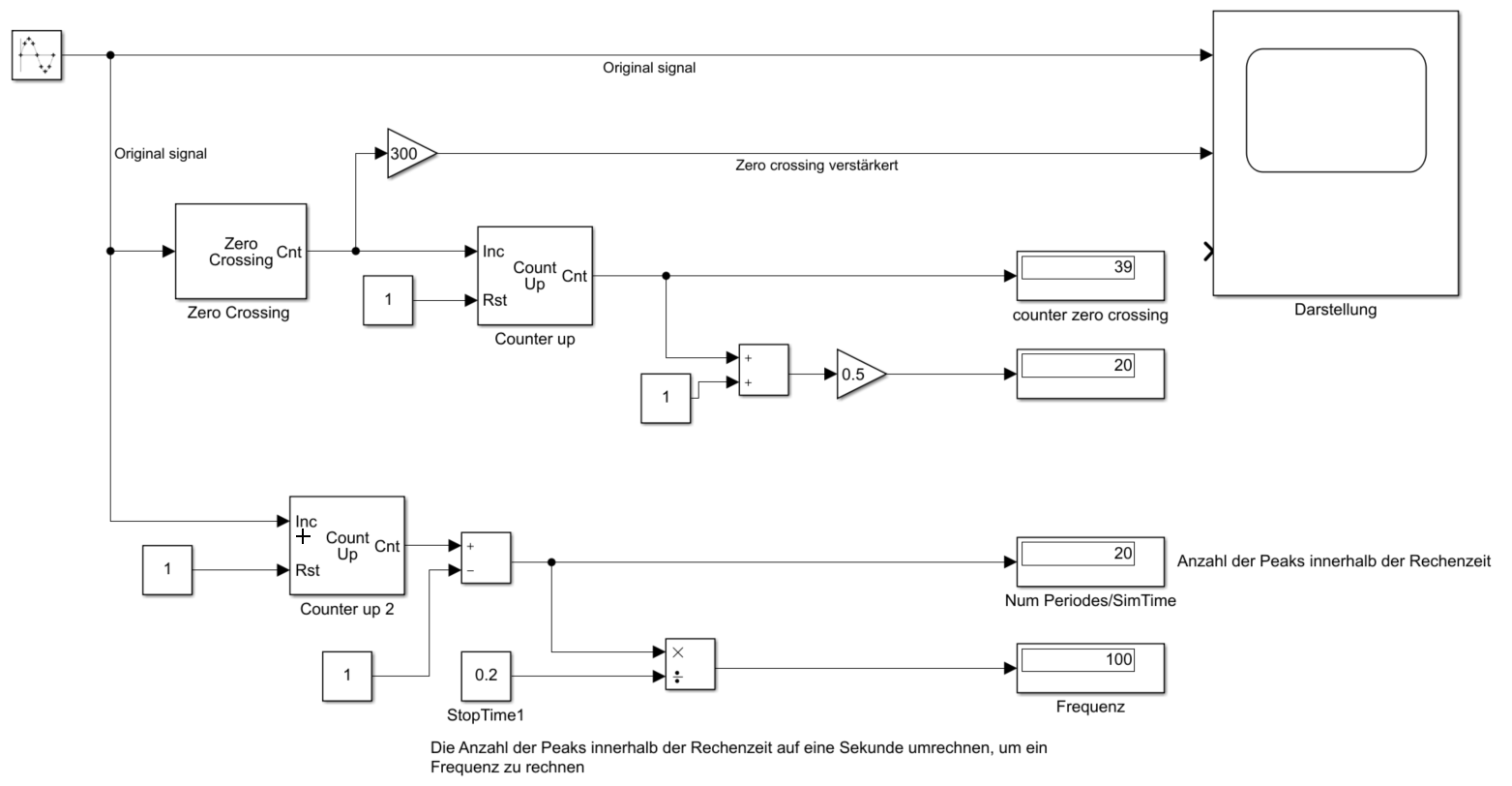


Abbildung 1 Das Testmodell

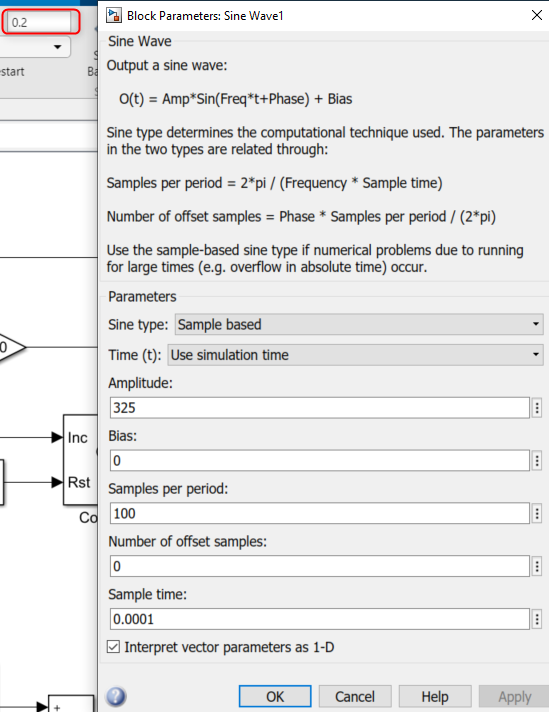


Abbildung 2 Die Einstellungen der Sinuskurve des Testmodulls

Eine zeitliche Frequenz wird folgendes berechnet:

Freq(Hz) = Freq(rad/sec) / (2 \* pi); pi = 3,14

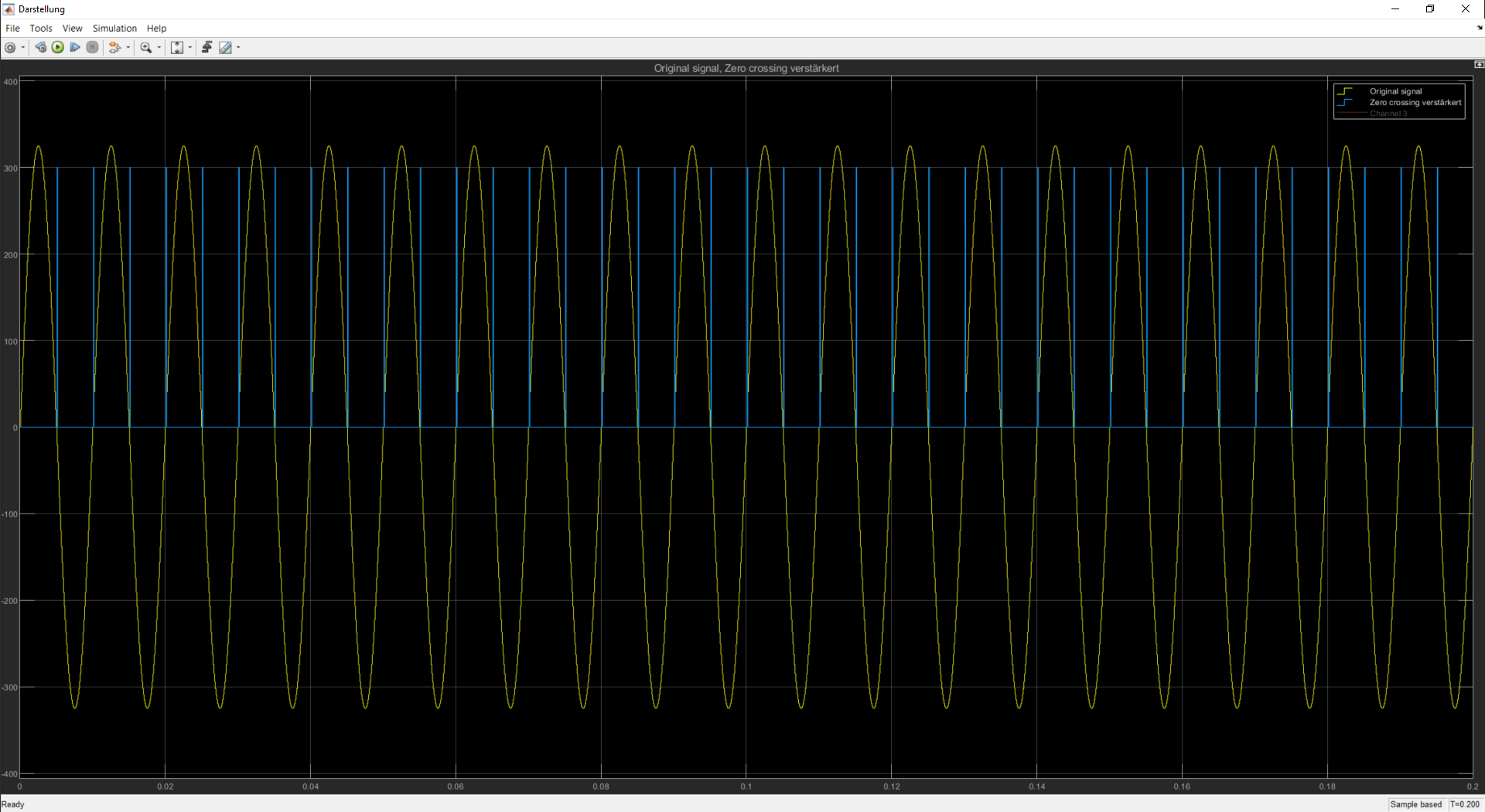


Abbildung 3 Das Ergenis des Modells (Scope)

In der Scope-Darstellung sind das originale Sinussignal und ein Zähler sichtbar. Der Zähler hat jeden ‚Zero-crossing‘ aufgezählt.

Das Testmodell hat die richtige (erwartete) Ergebnisse geliefert: 100 Hz als Frequenz.

Beim Einsetzen des gleichen Vorgehens auf das richtige Signal (AccBfX\_mg) (das kalibrierten Signal) wurde eine Frequenz von ca. 11 Hz beim Laufen und eine Frequenz zwischen 20-30 Hz ausgerechnet. Das hat die Erwartungen nicht entsprochen.

Daraus kann folgendes extrahiert werden:

Beide Signale (Szenarien) (Laufen und Fahren) haben die gleiche Menge von Störungen (Rauschen). Der Unterschied ist die Amplitude. Es kann keine Amplitude ‚Hartkodiert‘ entdeckt werden, da die Amplitude sich ständig ändern kann.

\*\*\*\*\*\*\*\*\*\*\*\* Grafiken von den zwei Signalen darstellen und die Art der Rauschen näher analysieren (was könnte die Ursache sein?); Welche mögliche Lösungen gäbe es dafür? \*\*\*\*\*\*\*\*\*\*\*\*\*

**Die Frequenz muss allerdings doch gesucht und schließlich verglichen werden.**

### 2. Idee: Frequenz ermitteln

zum Berechnen der Frequenz: FFT (Fast Fourier Transformation)

Nochmal das gleiche mit dem Testmodel durchgeführt.

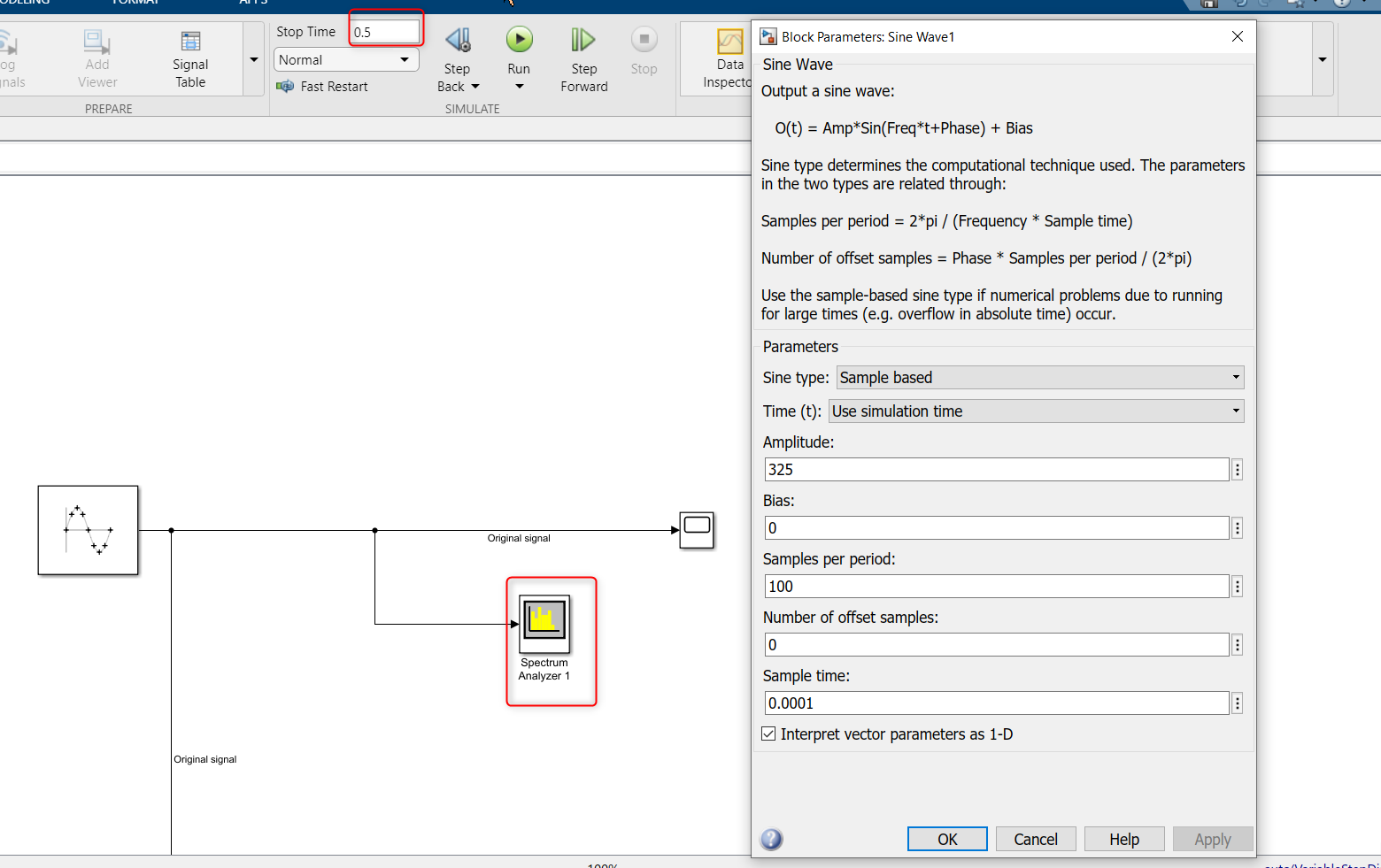


Abbildung 4 Testmodell mit dem Spectrum Analyzer

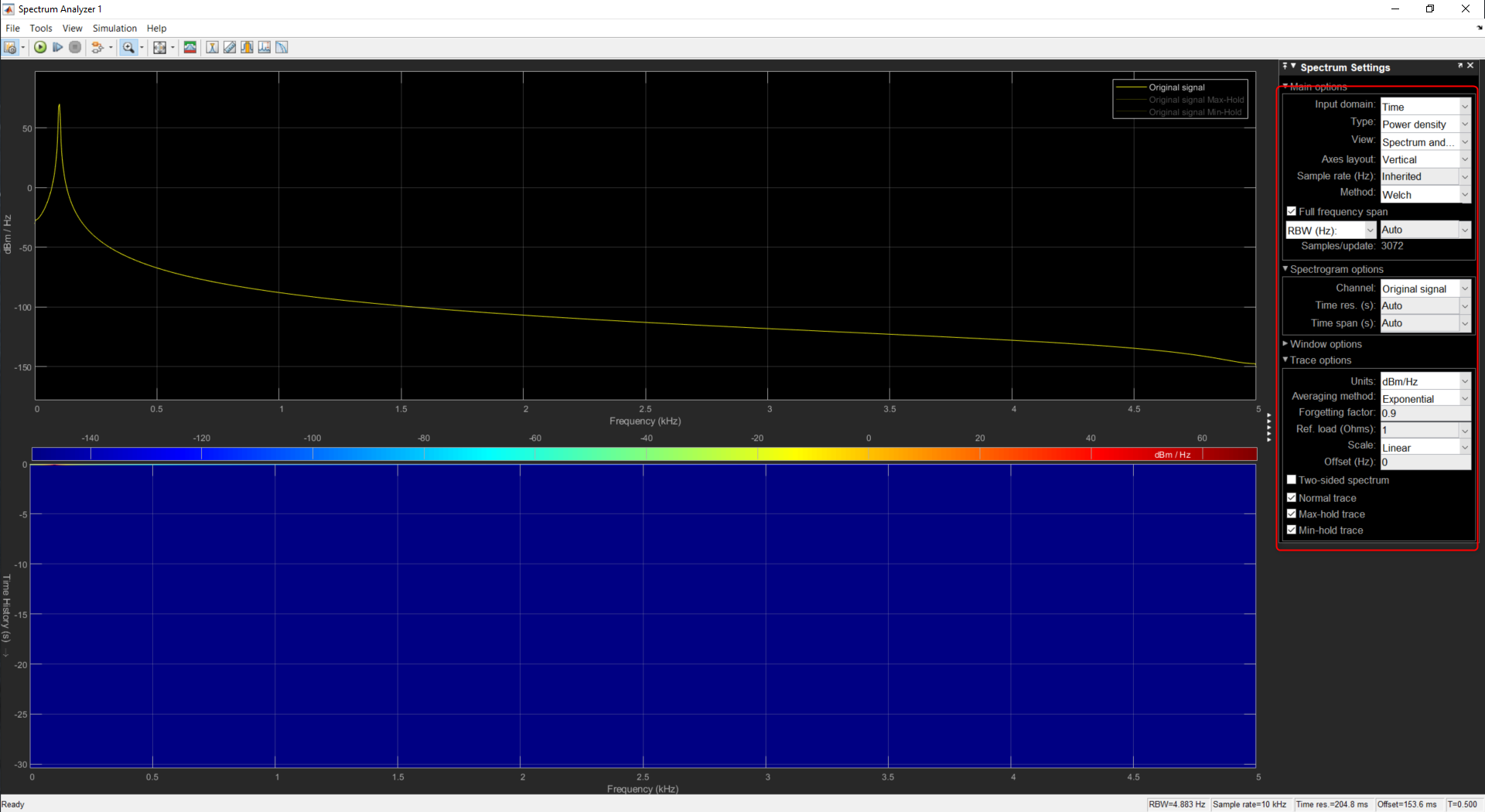


Abbildung 5 Spectrum Analyzer vom Testmodul 1 \*\*\* Zeit

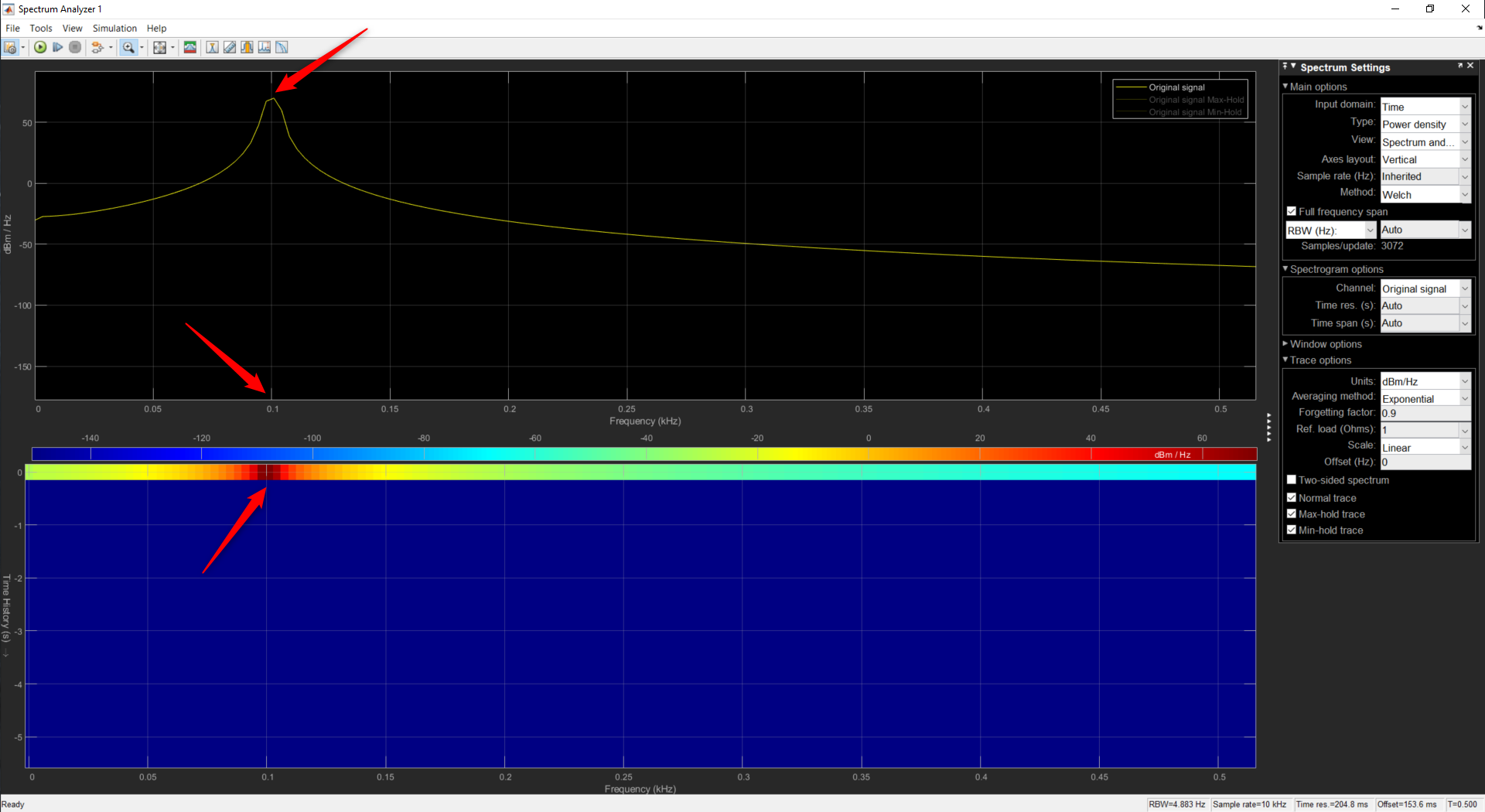


Abbildung 6 Spectrum Analyzer vom Testmodul 2 (zoom in)

Es ist schön zu bemerken, dass die Frequenz (100 Hz) sehr gut sichtbar ist. Der obere Teil zeigt die Häufigkeit einer Frequenz (Spektrum) unabhängig von der Zeit. Der untere Teil zeigt die Häufigkeit einer Frequenz mit der Zeit (Spektrogramm).

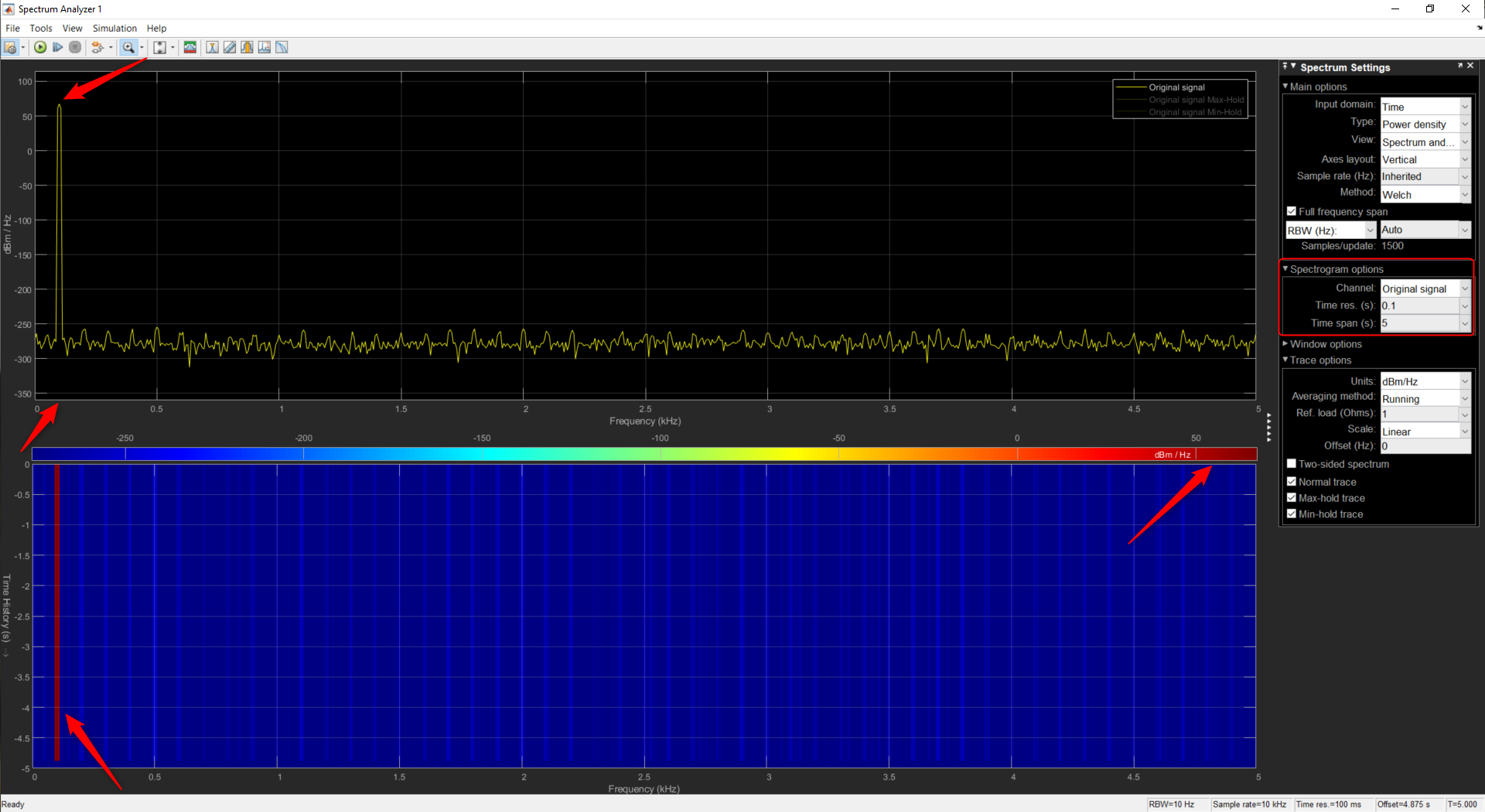


Abbildung 7 Spectrum Analyzer andere Einstellungen! bessere sichtbare Ergebnisse

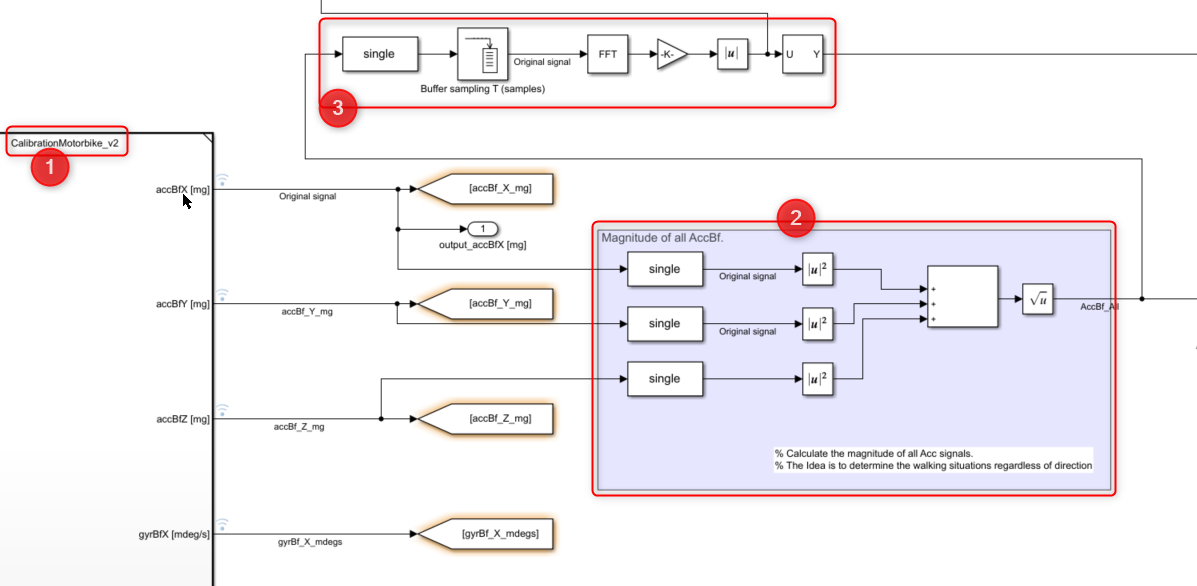


Abbildung 8: Teil des Modells, wo den Betrag von allen AccBf berechnet wird und anschließend eine FFT zur Frequenzermittlung programmiert wurde

Die AccBfX, AccBfY und AccBfZ sind die Ausgänge des Modells ‚CalibrationsMotorbike\_V2‘ und sie sind die Kalibrierte Signale.

Diese Werte werden für die Berechnung der Betrag mit der Formel

verwendet (Abbildung 8- Nummer 2). Das Ziel ist die Frequenzermittlung richtungsunabhängig zu stellen.

Danach wurde eine FFT an der Variablen durchgeführt, um die Frequenzen zu ermitteln (Abbildung 8- Nummer 2).

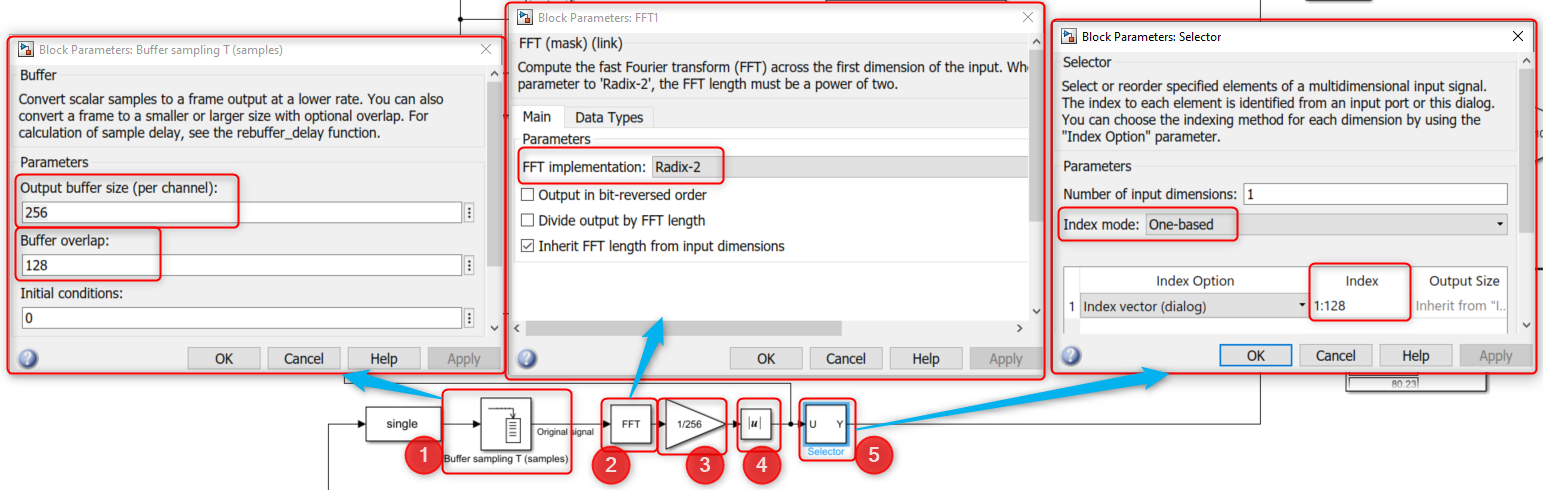


Abbildung 9: Teil des Modells, wo eine FFT durchgefürt wird

In der Abbildung 9 wird eine FFT durchgeführt. Die Einstellparameter jedes Element ist ersichtlich.

1. In der Buffer sind 256 Samples zu betrachten (d.h. ca. 2,5 Sekunden des Signals, da die Abtastrate der Sensor 100 Hz ist). Eine Überlappung von ca. 1 Sekunde wurde auch eingestellt, damit die Zwischen Frequenzen nicht übersehen werden.
2. Der FFT-Typ ist eine Radix-2.
3. Die Ausgabenwerte der FFT durch 265 dividieren. (warum?)
4. Betrag des FFT-Ausgangs bilden (warum?)
5. Da FFT ein gespiegelter Ausgang liefert wird nur die Hälfte der Matrix angenommen (1:128)

Mögliche Konflikte: ID = 2488; CrashNoPSAP;

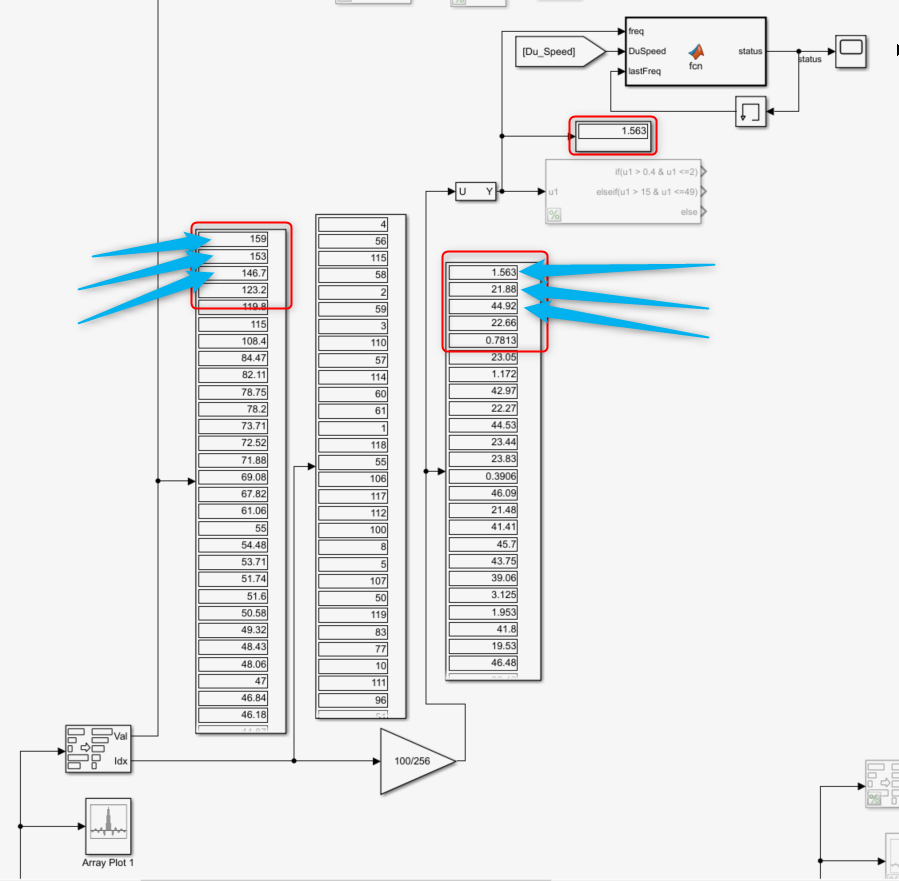


Abbildung : Frequenzerkennung - Simulink

Da hier die maximale Intensität der Frequenz 1,5, wird diese als das Maximum übernommen und weiterbearbeitet. Die nächste größte Intensität liegt sehr nah dazu und hat die Frequenz 21,88 Hz, was eigentlich richtiger ist.

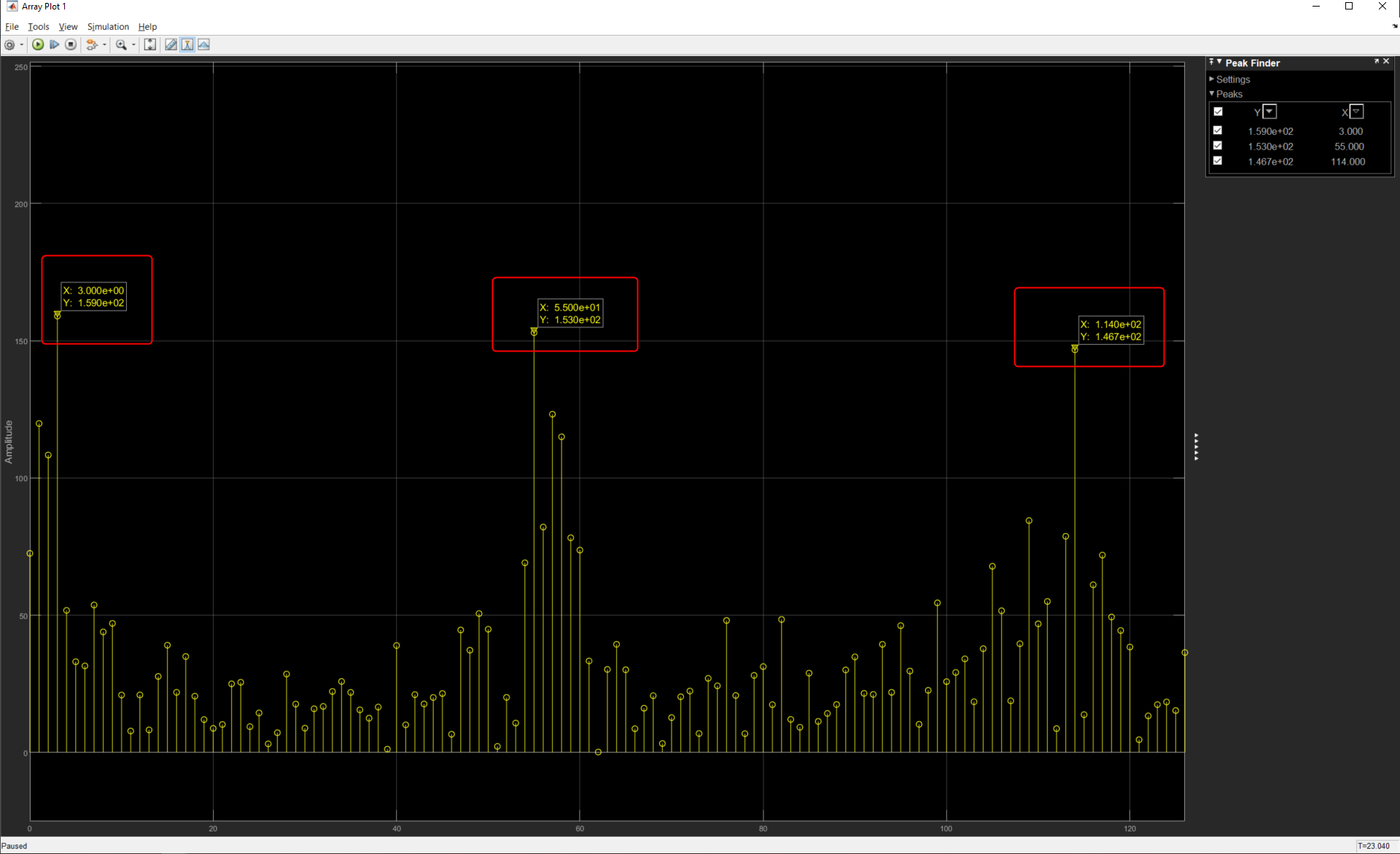


Abbildung : Scope mit den Frequenzen (nicht umgerechnet)

# Verweise

1, B. (kein Datum). Von C:\Users\dao1wa2\Desktop\Thesis\Ausarbeitung\Quellen\Analysis of traffic safety of powered two-wheelers for predictive assessment of normal driving critical situations and accidents.pdf abgerufen

*Unfallforschung*. (kein Datum). Von Unfallforschung, Unfallmechanik und Unfallrekonstruktion: C:\Users\dao1wa2\Desktop\Thesis\Ausarbeitung\Quellen\vdoc.pub\_unfallforschung-unfallmechanik-und-unfallrekonstruktion.pdf abgerufen