Detector

De gsm geeft eigenlijk live metingen aan ons die wij moeten gaan lezen en gaan omzetten in bruikbare data. Deze bruikbare data moet dan ook nog eens correct worden geinterpreteerd.

Onze aanpak was zeer eenvoudig, we zijn eerst aan de slag geweest in een offline detector die met een eindig aantal metingen werkt om een code te schrijven die de data correct omzet en interpreteerd. Nadien hebben we een originele manier gevonden om deze live data bruikbaar te maken. Daarna hebben we alles samengebracht en was dit de detector van de stappenteller.

Formule van een gemiddelde

Het omzetten van de data gebeurt in één eenvoudige formule. Deze formule hoe ik hem nu uitleg is natuurlijk vertaald in een hoop if else-statements en een while-lus zodat java deze begrijpt. We nemen een gewogen gemiddelde met coefficienten 0.1,0.2,0.4,0.2,0.1 per as, dus in de x-, y- en z-richting. Deze zetten we eenvoudig om tot een 1D-signaal met de formule r = sqrt( x^2+y^2+z^2) FORMULE. Deze formule is afkomstig van de stelling van Pythagoras omdat alle assen loodrecht op elkaar staan maar daar zullen we niet dieper op ingaan. Met deze 1D-signalen gaan we opnieuw aan de slag en nemen we nog eens een gewogen gemiddelde. Deze keer doen we dit met andere coefficienten, namelijk 0.1,0.1,0.3,0.3,0.2. Deze methode om tot een waarde te komen heeft ons door de grafiek overtuigd. Aan de grafiek kan je zien dat we zonder probleem een drempel kunnen instellen en dat alle stappen zouden gedetecteerd worden. GRAFIEK Dit geld voor alle beweegtoestanden zoals lopen en wandelen en op verschillende ondergronden. Natuurlijk zou de drempel anders zijn en dit zullen we oplossen door een varierende drempelwaarden te gebruiken maar daar kom ik later op terug.

Drempel

Een drempel instellen in Java is niet zo makkelijk als het klinkt. We moeten dit gaan op delen in verschillende delen zodat java ons zou begrijpen. Onze aanpak was aan de hand van if else-statements. We definierden eerst twee waarden, de stappen die we bij iedere stap met één gaan verhogen en een waarde, we noemen deze ‘*vorige’*, die 0 of 1 kan zijn. We definieren ook een vaste drempel. Als de waarde die berekent wordt door de formule onder de drempel zit, krijgt *vorige* de waarde nul tot dat het boven de drempel gaat. Als de waarde boven de drempel gaat, tellen we één stap bij en wordt *vorige* 1. Door *vorige* gaan we niet bij iedere waarde boven de drempel een stap bij tellen. GRAFIEK

Live data

Iedere meting krijgt de gsm drie waarden van de accellero meter, één in de x-, y- en z-richting. Java kan de formule daar niet op toepassen want de formule heeft waarden nodig op meerdere tijdstippen en momenteel hebben we er maar één per meting. De metingen gaan dus moeten opgeslagen worden. Ook willen we dat we oude data die we niet meer gebruiken verwijderen want na een paar uur zou er al heel veel onnodige data bestaan die gewoon opslag ruimte en batterij verbruikt. Om deze redenen werden er drie begrensde nul lijsten van lengte 200 aangemaakt. Deze worden ingevuld naargelange dat er data ontvangen wordt van de accelero meter. Op de plaats die bepaald wordt door index die we per meting één verhogen en waarvan de modulo 200 genomen van wordt, zodat we binnen de lijst blijven. Op deze lijsten kan de formule wel toegepast worden. We verwijzen naar de elementen van de lijst om de waarden te gebruiken. De uitkomst van deze live data formule wordt dat toegepast op onze drempel formule om zo de stappen te bepalen.

Complexe drempelwaarde

Momenteel wordt er gewerkt met een vaste drempelwaarde. Dit is oké tot dat de gebruiken zijn bewegingstoestand veranderd en we plots geen stappen meer zouden waarnemen. De drempelwaarde zou eigenlijk afhankelijk moeten zijn van de waarden die we binnenkrijgen. Er zal dus een extra lijst moeten zijn voor onze waarden die we krijgen uit de formule om later hiervan een maximum van te bepalen. We hebben experimentieel ondervonden dat als de drempelwaarde op 70% van de maximum waarde leggen die we in een interval hebben, we een correct beeld krijgen van de genomen stappen. Natuurlijk hebben we complicaties want de code start en heeft enkel nul waarden, doordat we nullijsten aanmaakten. De max van een lijst die amper elementen bevat zal natuurlijk een fout beeld geven. Daarom dat we ook voor de eerste 200 metingen met een vaste drempelwaarde beginnen, namelijk 1.25\*9.81. Iedere keer dat een waarde wordt ingevuld op het laatste element van de nullijsten zal de drempelwaarde aangepast worden. Er wordt gewerkt met de meetsnelheid GAME en 200 metingen is gelijkwaardig met ongeveer 25 seconden. Er kan dus enige afwijking op zitten door de eerste metingen maar deze zal minimaal zijn.

Intents

In één van onze practica hebben we kennis leren maken met intents en hoe deze functioneren. Er bestonden al een start, stop en reset intent maar deze zijn allemaal handmatig. Als een gebruiker een stappenteller wil gebruiken, zal hij toch niet de moeite er willen insteken om iedere dag handmatig zijn stappenteller te doen starten, stoppen en resetten. We hebben dus een intent proberen schrijven die het aantal stappen reset, door op nul te zetten, iedere dag om 12 uur ’s nachts. Dit kon aan de hand van een calender import. De tijd die java opmeet hebben we ook moeten omzetten in ‘real time’ zodat java zou weten wanneer het 12 uur zou zijn. Dit is jammer genoeg niet gelukt om te voltooien. We kregen een ‘intent error’ die we niet hebben kunnen oplossen.

We hadden het zelfde idee met de start intent. Er zou een if-statement kunnen aangemaakt worden dat als de gsm niet uitstaat en de detector nog geen waarden aan het verwerken is dat hij dat automatisch start. In plaats van eerst de app te moeten openen en dat de stappenteller dan pas van start gaat, hoe het nu is.