Stappenteller, Tussentijds Rapport 4, Groep E

Arthur Saelens, Staf Vanhauwaert, Maxim Ponomariov, Linde Roggeman

Begeleiders: Vincent Bracke, Maarten de Mildt

1. Inleiding

Nadat we vorige les kennis hebben gemaakt met java mogen we onze net geleerde begrippen direct gaan toepassen. Deze les kregen we de opdracht om al onze java kennis te combineren met onze filter die we de eerste twee lessen hebben gecreëerd om zo tot onze werkende stappenteller te komen. Maar snel realiseerden we dat dit net iets ingewikkelder is dan het klinkt. De problemen die we tegen kwamen zal ik bespreken.

1. Offline analyse

Eerst en vooral zal ik moeten zeggen dat deze code niet door ons is geschreven maar door onze geliefde proffen. De functie van deze code is dat we kunnen experimenteren met filters op data die niet live is. Dit is praktisch omdat we een filter op een csv bestandje kunnen uitvoeren. De filter word constant op dezelfde waarden toegepast en zo kunnen we zeker zijn als onze filter daadwerkelijk werkt of niet. We hebben ook nog niet het probleem dat we java live data moeten laten inlezen en we kunnen kiezen wanneer onze code wordt uitgevoerd. We hebben er wel al gebrainstormd over hoe we live data bruikbaar zouden maken. Een lijst die zich oneindig lang blijft uitbreiden is natuurlijk niet makkelijk leesbaar of verwerkbaar voor java. Om die reden hadden we het idee om met een circulaire lijst te gaan werken die ongeveer het dubbel aantal metingen bevat dan we nodig zouden hebben en iedere keer als een meting bij komt, wordt de oudste meting vergeten.

1. Dummy-detector

Het csv bestandje dat gebruikt werd, moest een correcte naam hebben met namelijk de naam van de persoon die het experiment uitvoerde, de plaats waar de gsm zich bevond tijdens het experiment en het aantal stappen bv. Stap-broekzak-10.csv. Deze dummy-detector telde het aantal metingen en we konden hier gebruik van maken door hem te vertellen wat hij als meting moet meetellen. In het begin telde hij een 600-tal stappen omdat we nog geen voorwaarden hadden toegekend. Om voorwaarden toe te maken hebben we eerst argumenten nodig. Zo maakten we de integers aan namelijk *mSteps* = 0 en *vorige* = 0. Daarna maakten we de functie addAccelerationData() aan om iedere stap te detecteren. DetectorLog is daar een argument van om extra data te rapporteren.

1. filteren van data

In het csv bestandje staan er vier kolommen met de tijdseenheid en de waarden op de drie assen. Deze moeten gefilterd worden zodat ze bruikbaar zijn. Dit deden we volledig hetzelfde als in practicum 2 door de vierkantswortel te nemen van de som van de kwadraten van de assen en natuurlijk trekken we de valversnelling ervan af. Dit konden we doen door de waarden in de kolommen van de assen toe te kennen aan de functie addAccelerationData() als argumenten. Nu konden we beginnen aan de voorwaarden op te stellen om een stap te detecteren.

1. Opgelegde voorwaarden

Om een stap te hebben zal onze gefilterde data een bepaalde drempel moeten overschrijden en we willen dat deze eenmalig geregistreerd wordt zolang dat hij er niet meer onder gaat. Dit vertalen we voor java om in twee voorwaarden. Als de waarde onder onze drempel zit, is er zeker geen stap en als deze erboven gaat, is er enkel een stap als de vorige waarde onder de drempel lag. Dit doen we door onze gedefinieerde waarde *vorige* één in waarde te doen stijgen zodra de drempel overschreden is als *vorige* nog nul was. Zodra de waarde terug onder de drempel gaat, trekken we één af bij *vorige* en staat deze terug op nul. Zo laten we java eenvoudig berekenen of er een stap is.

1. Drempelwaarde

Onze drempel waarde hebben we voorlopig enkel nog maar experimenteel kunnen afleiden. Er is veel gebrainstormd geweest over hoe we deze kunnen berekenen en afhankelijk kunnen maken van de stappen zelf door bijvoorbeeld een interval te nemen van ongeveer 300 metingen en de drempel waarde gelijk te stellen aan de waarde die groter is dan 80% van de waarden in dat interval. Dit is nog net iets te moeilijk om uit te voeren. De drempelwaarde die bewijzen heeft te kloppen in verschillende experimenten is 1,5. Dit is redelijk logisch want een drempelwaarde van minder zou zeer gevoelig zijn aan ruis en iets meer zal stappen over het hoofd zien.

1. Onze stappenteller

Het was effectief nodig om de data te filteren maar dit hadden we verwacht en dit hebben we gedaan zoals in practicum twee. De volledige filter van practicum twee overnemen hebben we nog niet kunnen uitvoeren maar er is al een idee van hoe we dit zouden doen, namelijk met een circulaire lijst. We zijn ervan overtuigd dat onze filter die we toen bekomen zijn, zal werken in alle omstandigheden. Alle assen zijn noodzakelijk voor de stappenteller te doen werken. Doordat we niet zeker zijn hoe de gsm georiënteerd is te allen tijde hebben we een duidelijke gefilterde waarde nodig die alle assen vertegenwoordigd. Anders zou de stappenteller niet functioneren als de gsm vastgehouden wordt in de hand in plaats van de broekzak. De parameters die we tot nu toe al hebben moeten kiezen bestaat enkel uit de drempelwaarde die experimenteel is afgeleid. We zijn ervan bewust dat dit wetenschappelijk niet de juiste manier is en daarom zijn we aan het werken aan een manier om deze te berekenen.

1. Pseudocode

De psuedocode werkt als volgt: vier kolommen worden opgehaald uit de meting en deze kennen we toe aan de functie addAccelerationData() . We gebruiken de x, y en z kolommen om een gefilterde waarde te bekomen waarop we dan onze detector gaan uitvoeren. De detector gaat alle waarden af en telt enkel een stap als de waarde boven een drempel waarde gaat en de vorige er niet boven was. Voor de reële stappenteller gaan we zorgen dat deze detector aangepast wordt zodanig dat we niet enkel de waarden van een bepaald tijdstip kunnen lezen maar ook gewogen gemiddelde kunnen gaan nemen. Dit gaan we doen door een circulaire lijst aan te maken en te gaan rekenen met de elementen van die lijsten. Het resultaat van de detector als we wandelen komt overeen met het aantal stappen die we genomen hebben. Hieronder kunt u zien hoe onze pseudocode eruit ziet.

Code 1: pseudocode

private int mSteps = 0;  
private int vorige = 0;  
  
public void addAccelerationData(long timestamp, float x, float y, float z, DetectorLog log) {  
 double a = java.lang.Math.*sqrt*(x\*x + y\*y + z\*z) - 9.81;  
 if (vorige == 0) {  
 if (a >= 1.5) {  
 mSteps += 1;  
 vorige = 1;  
 }  
 }  
 else {  
 if (a <= 1.5) {  
 vorige = 0;  
 }  
 }

1. Conclusie

De detector die we nu hebben bekomen werkt, maar nog niet volledig op de manier dat we willen dat het werkt. We zullen in de komende lessen dus focussen op het live kunnen ontvangen van data en deze in lijsten krijgen zodat we de elementen ervan kunnen gebruiken met uiteindelijk als doel de filter uit practicum twee te kunnen toepassen. Daarnaast willen we ook gaan onderzoeken of het niet interessanter en correcter zou zijn om een variërende drempelwaarde te nemen in plaats van een vaste omdat de drempelwaarde verandert naargelang de snelheid van het wandelen.