Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Automatizálási és Alkalmazott Informatikai Tanszék

Sebesség meghatározási lehetőségek vizsgálata kerékpáron mobil támogatással

Készítette
Olasz-Szabó Bence

Konzulens
Dr. Ekler Péter

2014. október 22.

Tartalomjegyzék

1. Bevezetés

Napjainkban sokan töltik szabadidejüket kerékpározással, ezért egyre nagyobb az igény a kerékpáron megtett út adatainak mérésére és megjelenítésére. A hagyományos kerékpár kilométerórák azonban csak néhány adat mérésére képesek, továbbá korlátozott kijelzőméretük miatt gyakran a mért adatok megjelenítése is kevésbé áttekinthető formában történik. Ezért a kerékpáron megtett út adatainak mérésére és megjelenítésére célszerű okostelefonokat alkalmazni.

Az okostelefonok kijelzőjének mérete lehetővé teszi a mért adatok áttekinthető megjelenítését. A mai készülékek különféle hardveres és szoftveres lehetőségei teret biztosítanak kreatív megoldásoknak is. Ilyen lehet például egy hangbemenetre kötött kilométeróra mágnes. Továbbá a modern mobil készülékek különféle szenzorokkal vannak felszerelve (GPS, gyorsulásérzékelő), amik kiegészítő adatforrásként szolgálhatnak a megtett út adatainak mérésekor. Mindezek mellett a modern okostelefonok számítási kapacitása lehetővé teszi a mért adatok feldolgozását is.

1.1. A feladat áttekintése

A feladatom egy hardver és mobil szoftver alapú megoldás kidolgozása pontos offline kerékpár sebesség mérésre. Ehhez egy egyedi hardveres megoldást és egy hozzá illeszkedő kerékpáros fedélzeti számítógép alkalmazást terveztem és valósítottam meg Android platformon. Az általam tervezett és megvalósított egyedi megoldás lényege, hogy kihasználja, hogy a mobiltelefon mikrofon bemenetére csatlakoztatott jeleket programozottan lehetőség van vizsgálni. A megoldást összehasonlítottam a mobil telefonon elérhető egyéb sebesség mérésre szolgáló lehetőségekkel. Mérések és vizsgálat segítségével bebizonyítottam, hogy az általam kidolgozott megoldás pontos mind sebesség, mind pedig távolság mérésre. A dolgozatban bemutatom az elkészített megoldásom hardveres és szoftveres komponenseit. Ezt követően kifejtem az elvégzett részletes méréseket, illetve azok eredményét, továbbá igazolom a megoldás helyességét.

1.2. A dolgozat felépítése

A 2. fejezetben kerékpáros sebesség-meghatározásra képes eszközöket és alkalmazásokat, valamint a dolgozat témájához kapcsolódó korábbi kutatási eredményeket mutatok be.

A 3. fejezetben bemutatom a munkám során felhasznált kerékpáros sebesség-meghatározásra alkalmas technológiák alapjait.

A 4. fejezetben a mérések elvégzéséhez készített Android alkalmazás felépítését és működési elvét ismertetem.

Az 5. fejezetben a mérési eredmények elemzésével mutatom be a vizsgált kerékpáros sebesség-meghatározási módszerek előnyeit és hátrányait és igazolom az általam tervezett megoldás helyességét.

A 6. fejezetben összefoglalom a mérési eredmények alapján levont következtetéseket a vizsgált módszerekről.

A 7. fejezetben további kerékpáros sebesség-meghatározási lehetőségeket említek meg, továbbá bemutatom a mérések elvégzéséhez készített Android alkalmazás továbbfejlesztési lehetőségeit.

2. Irodalomkutatás

Munkám során elemeztem a jelenleg elterjedt kerékpáros sebesség-meghatározásra képes eszközök tulajdonságait, továbbá tájékozódtam a dolgozat témájához kapcsolódó korábbi kutatási eredményekről.

2.1. Hagyományos kerékpár kilométerórák

Napjainkban a kerékpározók jelentős része még mindig a hagyományos kerékpár kilométerórákat (*cyclocomputers[1]*) használja a kerékpáron megtett út adatainak mérésére. Ezek az eszközök az adatokat többnyire kerékfordulat-érzékelőről gyűjtik, bár más adatforrásokat (GPS, pedálfordulat-érzékelő) is felhasználó eszközök is elérhetőek a magasabb árkategóriában.

A hagyományos kerékpár kilométerórák előnyei az alábbiak:

- A pillanatnyi sebességet egészen pontosan tudják mérni a kerékfordulat-érzékelőnek köszönhetően.
- Eleve kerékpározásra tervezték őket, ezért ellenállóak a fizikai hatásokkal (erőhatások, víz) szemben.
- Felépítésük egyszerű, ezért kevés hibalehetőséget tartalmaznak, továbbá akár otthon is elkészíthetőek[2].

Az előnyök mellett azonban a hagyományos kerékpár kilométeróráknak jelentős hiányosságaik vannak:

- Korlátozott kijelzőméretük miatt gyakran a mért adatok megjelenítése kevésbé áttekinthető formában történik.
- Kezelésük, beállításuk nehézkes, gyakran bonyolult menürendszerrel rendelkeznek.
- Általában csak néhány adat mérésére képesek.

Ezért a kerékpáron megtett út adatainak mérésére és megjelenítésére érdemes korszerűbb eszközöket alkalmazni.

2.2. Kerékpáros fedélzeti számítógép alkalmazások okostelefonokra

A hagyományos kerékpár kilométerórák hiányosságaira megoldást nyújthat az okostelefonok kerékpáros fedélzeti számítógépként történő felhasználása.

A módszer előnyei az alábbiak:

- Az okostelefonok kijelzőjének mérete lehetővé teszi a mért adatok áttekinthető megjelenítését.
- A modern mobil készülékek különféle szenzorokkal vannak felszerelve (GPS, gyorsulásérzékelő), amik kiegészítő adatforrásként szolgálhatnak a megtett út adatainak mérésekor.
- A modern okostelefonok számítási kapacitása lehetővé teszi a mért adatok feldolgozását.

- A mért adatok könnyen feltölthetőek a számítógépünkre vagy megoszthatóak ismerőseinkkel az interneten.

Ugyanakkor ennek a módszernek is léteznek hátrányai:

- Az adatok folyamatos megjelenítéséhez az okostelefon képernyőjének folyamatosan bekapcsolva kell lennie, ami sok energiát fogyaszt[3], így egy hosszabb túra alatt akár teljesen lemerülhet a készülék akkumulátora.
- A legtöbb mobil készülék fizikai hatásokra érzékeny, könnyen kár keletkezhet bennük kerékpározáskor.

A jelenlegi kerékpáros fedélzeti számítógép alkalmazások többsége az alábbi két kategóriába sorolható:

- Az okostelefonok beépített szenzorait használó alkalmazások.
- Külső mérő hardverek adatait használó alkalmazások.

2.2.1. Az okostelefonok beépített szenzorait használó alkalmazások

Kerékpáros sebesség-meghatározásra a modern okostelefonok beépített szenzoraiból kinyerhető adatok önmagukban is egészen jól felhasználhatóak. A módszer előnye, hogy nem igényel speciális hardvert, ezért a kerékpározás mellett más tevékenységek (például futás) során megtett út adatainak mérésére is alkalmas. Ilyen alkalmazás például az *„AllSport GPS FREE”[4]* vagy a *„SpeedView: GPS Speedometer”[5]*.

A legelterjedtebb megoldás a beépített GPS szenzor felhasználása. A GPS szenzor segítségével a pillanatnyi pozíció mellett többek között meghatározható a pillanatnyi sebesség, a megtett út és az aktuális tengerszint feletti magasság is. Azonban a GPS adatok pontosságát a környezeti tényezők jelentősen képesek befolyásolni, ahogy azt a *„SpeedView: GPS Speedometer”* alkalmazás leírásában meg is említik[1]. Ezért azok az alkalmazások, amik kizárólag a GPS adatokat használják fel, nem elég megbízhatóak.

2.2.2. Külső mérő hardverek adatait használó alkalmazások

Léteznek olyan mobil alkalmazások is, amik az okostelefonok beépített szenzorain kívül külső mérő hardverektől érkező mérési adatokat is felhasználnak. A módszer előnye, hogy pontos mérési adatok állnak rendelkezésre a kerékpáron elhelyezett szenzoroknak köszönhetően. Ugyanakkor a mérési adatokat gyűjtő speciális hardver eszközöket a felhasználónak külön meg kell vennie a rendszer használatához.

Ezek az eszközök tipikusan Bluetooth kapcsolaton keresztül kommunikálnak az okostelefonnal. Ilyen eszköz például a *„Mobile Action CS-20 Speed / Cadence Bike Sensor”[6]*.

[1] „Please note that the accuracy of GPS measurements is affected by a number of factors including atmospheric conditions, obstructions and the visibility of satellites.” - https://play.google.com/store/apps/details?id=com.codesector.speedview.free

2.3. A dolgozat témájához kapcsolódó korábbi kutatási eredmények

A modern mobil eszközök beépített érzékelőiben rejlő lehetőségeket már számos tudományos munka során igénybe vették. Az *„An Automatic Recognition of Bicycle Riding State by Using a Smartphone”[7]* című dokumentum szerzői olyan algoritmust fejlesztettek, ami képes az okostelefon gyorsulásmérő szenzorának segítségével meghatározni, hogy a kerékpár emelkedőn vagy lejtőn halad, vagy megállt. Szintén az okostelefon gyorsulásmérő szenzora által biztosított adatokat használják fel az *„Accelerometer-Based Transportation Mode Detection on Smartphones”[8]* című munka szerzői, ezúttal a felhasználó által igénybe vett közlekedési eszköz felismerésére.

Az *„Accurate Caloric Expenditure of Bicyclists using Cellphones”[9]* című publikáció szerzői a gyorsulásmérő szenzor mellett a beépített GPS szenzort is használják munkájuk során. Ezek mellett speciális mérő hardvereket is alkalmaznak, amik Bluetooth kapcsolaton keresztül kommunikálnak az okostelefonnal. Érdekesség, hogy a tengerszint feletti magasságok különbségét légnyomásmérő szenzor által mért adatok alapján is kiszámítják, továbbá képesek a pedálfordulat érzékelésére a kerékpározó nadrágzsebében található okostelefon gyorsulásmérő szenzorának segítségével.

A *„SocialSensing - Social sensing application for Smartphone”[10]* című dokumentum szerzője a kerékpározók által megtett útvonalak adatainak közösségi megosztását is támogató alkalmazást fejlesztett Android platformon. A *„Method for Reading Sensors and Controlling Actuators Using Audio Interfaces of Mobile Devices”[11]* című publikáció a mobil eszközök audio interfészének az ipari automatizálás területén történő felhasználására mutat példát.

2.4. Konklúzió

A hagyományos kerékpár kilométerórák legnagyobb előnye, hogy a pillanatnyi sebességet képesek pontosan meghatározni a kerékfordulat-érzékelőnek köszönhetően. Ugyanakkor korlátozott kijelzőméretük miatt nem képesek a mért adatok áttekinthető megjelenítésére. Erre a problémára megoldást jelenthet az okostelefonok kerékpáros fedélzeti számítógépként történő alkalmazása. Azonban a jelenleg elterjedt okostelefon alapú kerékpár kilométeróra megoldások csak speciális, bonyolult hardver segítségével képesek a pillanatnyi sebesség pontos meghatározása, ennek hiányában többnyire csak a GPS szenzor által szolgáltatott adatok alapján képesek megbecsülni a pillanatnyi sebességet.

Az általam kidolgozott okostelefon alapú kerékpáros fedélzeti számítógép megoldás egyesíti a hagyományos kilométerórák és az okostelefonok előnyeit. A mért adatok megjelenítése a mobil készülék kijelzőjén, áttekinthető formában történik. A pillanatnyi sebesség meghatározására kerékfordulat-érzékelőt használok, azonban bonyolult, speciális hardver helyett csupán egy 4 pólusú Jack csatlakozó segítségével csatlakoztatom a kerékszenzort az okostelefonhoz.

3. Felhasznált technológiák

Ebben a fejezetben bemutatom a munkám során felhasznált kerékpáros sebesség-meghatározásra alkalmas technológiák alapjait.

3.1. GPS

A *Global Positioning System* (GPS, Globális Helymeghatározó Rendszer) az Amerikai Egyesült Államok Védelmi Minisztériuma (*Department of Defense*) által (elsődlegesen katonai célokra) kifejlesztett és üzemeltetett – a Föld bármely pontján, a nap 24 órájában működő – műholdas helymeghatározó rendszer.[12]

A rendszer kulcselemei a földfelszín felett 20 000 km magasságban keringő helymeghatározó műholdak. Ezek tulajdonképpen nagyon pontos órák, amik a pozíciójukat és az általuk mért idő pontos értékét rádióhullámok segítségével folyamatosan sugározzák a Föld felé.

A műholdak által közölt adatok alapján a GPS vevő készülékek a saját pozíciójukat az alábbi módszer szerint határozzák meg:

- A vevők a műholdak üzeneteiből ismerik az elérhető műholdak aktuális pozícióját, valamint az egyes műholdaktól érkező legutóbbi üzenet kibocsájtásának pontos idejét.
- A vevők az egyes üzenetek érkezési és küldési időpontjainak különbségéből és a rádióhullámok (feltételezett) terjedési sebességéből kiszámítják az egyes műholdaktól számított távolságukat.
- Az egyes műholdaktól számított aktuális távolságok ismeretében 3 műhold segítségével már meghatározható az aktuális pozíció a földfelszínen. Azonban ez az érték még pontatlan, mivel a vevő órája általában nincs szinkronban a műholdakéval. Ezért a pozíció pontosításának érdekében szükség van egy 4. műholdra is.

A módszer pontosságát többek között az korlátozza, hogy a rádióhullámok sebessége a légkörbe érve előre nem kiszámítható mértékben lelassul, ezzel torzítva az egyes műholdak és a vevő között számított távolságok értékét. A módszer további hátránya, hogy csak nyílt terepen alkalmazható, mikor a vevőnek elegendő műholdra van közvetlen rálátása. Ez a feltétel épületek vagy hegyek között nem mindig teljesül. Ilyenkor a GPS vevő által meghatározott pozíció és sebesség adatok pontatlanok vagy nem elérhetőek.

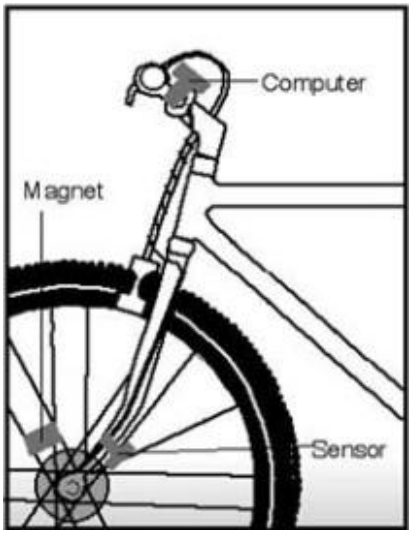
3.2. Kerékfordulat érzékelő

Munkám során egy kerékfordulat-érzékelő hardveres megoldást és a hozzá illeszkedő mobil szoftvert készítettem el, mely a kutatás alapjául szolgált. Ennek rövid bemutatását az alábbi szakasz tartalmazza.

3.2.1. Működési elv

A kerékpáros sebesség-meghatározás leggyakrabban kerékfordulat érzékelő segítségével történik (3.1. ábra). A módszer alapötlete a következő: a kerékpár (tipikusan első) kerekére egy kisméretű mágnest rögzítünk, a villára (vagy vázra, hátsó kerék esetén) pedig egy

érzékelőt (tipikusan *reed kapcsolót[13]*, esetleg *Hall szenzort[14]*), aminek bizonyos elektromos tulajdonságai megváltoznak mágneses tér hatására. Ezáltal képesek vagyunk érzékelni a kerék által megtett fordulatokat. Amennyiben tudjuk a kerék kerületét, az előbbiekből már kiszámítható a kerékpár által megtett út. Ha a kerékfordulatok között eltelt időt is képesek vagyunk elfogadható pontossággal mérni, akkor már a legutóbbi kerékfordulathoz tartozó átlagsebességet is meg tudjuk határozni, ami az esetek többségében jó közelítést ad a pillanatnyi sebességre.

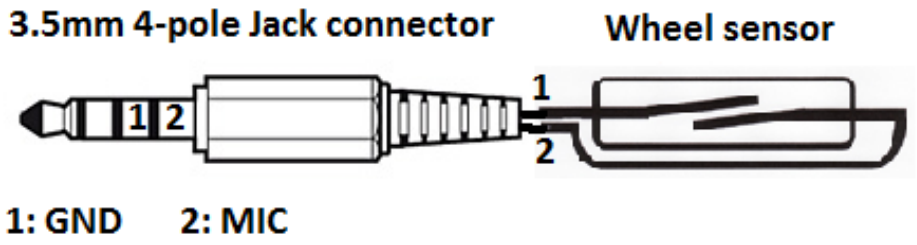


3.1. ábra.[15] Kerékfordulat érzékelése.

3.2.2. Összekapcsolás mobil eszközökkel

A kerékfordulat érzékelőt mobil eszközökkel legegyszerűbben a készülék audio interfészének segítségével lehet összekapcsolni. A módszer előnye, hogy csupán minimális számú, olcsó alkatrészt igényel, szemben a mérésre speciális hardvert használó, a mobil készülékkel Bluetooth kapcsolaton vagy más digitális interfészen keresztül kommunikáló megoldásokkal.

A csatlakozó tipikusan 3,5mm-es 4 pólusú Jack csatlakozó. Fontos, hogy 4 pólusú legyen, ugyanis a mobil eszközök többségére ezzel a módszerrel lehet mikrofon bemenetként külső eszközt csatlakoztatni, mivel nincs külön be- és kimeneti audio csatlakozójuk (ellentétben például a laptopokkal). Az összekapcsolás elvét a 3.2. ábra szemlélteti.



3.2. ábra.[16][17] Kerékszenzor csatlakoztatása mobil eszközökhöz.

Megjegyzés: Ahhoz, hogy bizonyos mobil eszközök biztosan felismerjék a kerékszenzort külső mikrofonként, szükség lehet még néhány passzív elektronikai alkatrészre (ellenállásra, kondenzátorra) is. Azonban saját tesztjeim során ezek nélkül is működött a rendszer, ezért az ábrán ezeket nem tüntettem fel.

4. A mérésekhez szükséges Android alkalmazás

A mobil támogatással elérhető kerékpáros sebesség-meghatározási módszerek vizsgálatához készítettem egy alkalmazást Android platformon.

4.1. Követelmények

Az alkalmazásnak teljesítenie kell az alábbi követelményeket:

1. Képesnek kell lennie a kerékfordulat érzékelő által az audio bemeneten szolgáltatott jelből felismernie a kerékfordulatot jelentő impulzusokat.
2. A kerékfordulatok között eltelt időt legalább ms-os pontossággal kell tudni mérni a pillanatnyi sebesség elfogadható pontosságú meghatározásához.
3. Az alkalmazásnak meg kell valósítania a hagyományos kerékpár kilométerórák szokásos funkcióit.
4. A beépített GPS szenzor segítségével is meg kell határozni a pillanatnyi sebességet és a megtett távolságot.
5. Az alkalmazásnak a mérési adatokat abban az esetben is gyűjtenie kell, mikor más alkalmazás fut előtérben a készüléken.
6. A mérési eredményeket CSV formátumú log fájlokba kell menteni a későbbi feldolgozás érdekében elegendően magas mintavételi frekvenciával ahhoz, hogy a mért adatok többsége rögzítve legyen akkor is, ha a kerékfordulat érzékelőről 200ms-onként érkeznek az impulzusok.
7. A készülék kijelzőjén meg kell jeleníteni a felhasználó által kiválasztott adatsorok aktuális értékeit.
8. A mérések elvégzése nem blokkolódhat illetve nem lassulhat le jelentősen a mért adatok megjelenítése vagy mentése miatt.

A 4. fejezet további részeiben az alkalmazás felépítését és működési elvét ismertetem.

4.2. Szoftver architektúra

A fenti követelményeknek megfelelő alkalmazás elkészítésének első lépése a szoftver architektúra megtervezése volt. Először a szoftver komponenseit, majd azok kapcsolatait definiáltam.

4.2.1. Szoftver komponensek

MenuActivity

Az alkalmazás indításakor megjelenő főmenü. Ez a komponens indítja el a többi alkalmazás komponenst adott események bekövetkezésekor (indítás vagy felhasználói interakció).

MeasurementService

Az 5. követelmény alapján a mérési adatokat akkor is gyűjteni kell, ha az alkalmazás éppen nincs előtérben. Erre a feladatra Android platformon *Service[18]* (szolgáltatás) komponenst célszerű használni. Azonban az Android operációs rendszere a háttérben futó szolgáltatásokat leállíthatja, ha éppen kevés a memória. Ezért a mérések elvégzéséhez érdemes *Foreground Service* komponenst használni. A *Foreground Service* olyan szolgáltatás, aminek a futásáról a felhasználó tud, ezért az Android rendszer csak extrém memóriahiány esetén állíthatja le.

MeasurementSources

A mérési adatok számításához szükséges alapadatokat szolgáltató komponensek.

Ilyen komponens például az audio interfészen érkező impulzusok keresését végző magas prioritású szál és a földrajzi helyadatokat szolgáltató osztályok.

DataStore

A mérési adatok számítását és tárolását végző osztály.

DataRefresher

A mérési adatok frissítését ezen az interfészen keresztül lehet kezdeményezni.

DataGetter

A mérési adatokat ezen az interfészen keresztül lehet lekérdezni.

DataListener

A mérési adatok változásáról a *DataListener* interfészt megvalósító osztályok értesülhetnek.

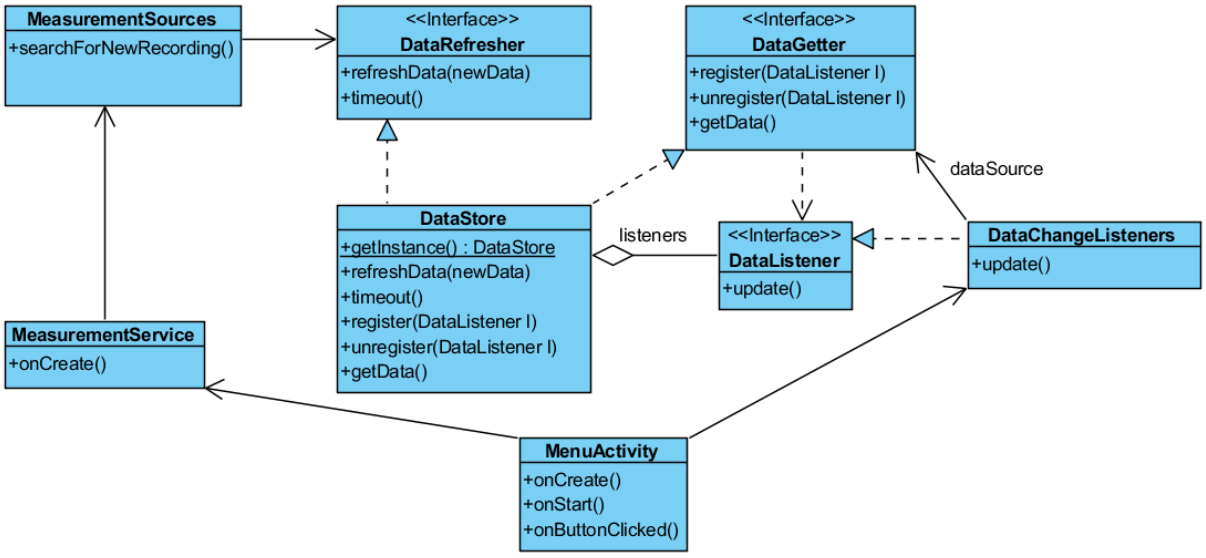
DataChangeListeners

A mérési eredmények változására feliratkozott komponensek. Megvalósítják a *DataListener* interfészt.

Ilyen komponens például az aktuális mérési eredményeket megjelenítő *CurrentActivity*.

4.2.2. Architektúra vázlat

Az előbbiekben ismertetett szoftver komponensek kapcsolatait a 4.1. ábra szemlélteti.



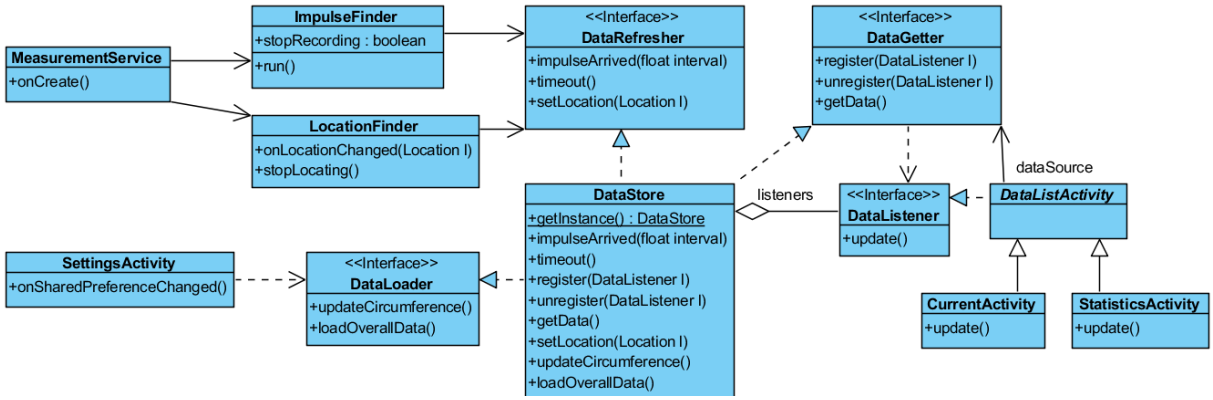
4.1. ábra.[31] Szoftver komponensek kapcsolatai

A komponensek alapvető együttműködése az alábbiak szerint foglalható össze:

- A *MenuActivity onStart()* metódusában elindítja a *MeasurementService* komponenst, *onButtonClicked()* metódusai pedig a megfelelő *DataChangeListener*-eket indítják el. A *MeasurementService* elindítja a különböző *MeasurementSources* komponenseket.
- A *MeasurementSources* komponensek a *DataRefresher* interfészen keresztül értesítik a *DataStore*-t új alapadatok érkezésekor. Ennek hatására a *DataStore* kiszámítja az aktuális mérési adatokat.
- Az aktuális mérési adatok az *Observer* tervezési minta alapján kerülnek a megfelelő komponensekhez: a *DataStore* értesíti a hozzá beregisztrált *DataListener*-eket, akik a *DataGetter* interfészen keresztül lekérik a szükséges adatokat.

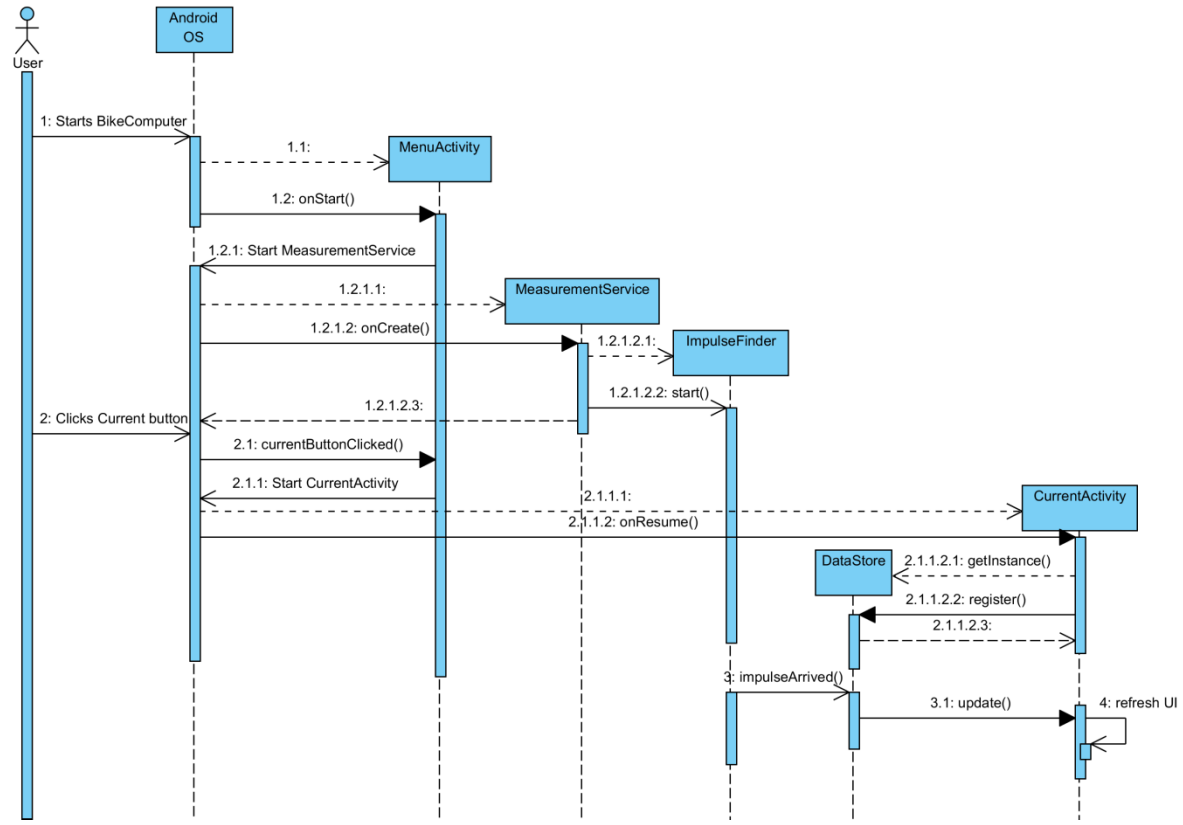
4.2.3. Részletes tervek

A részletes terveket az architektúra vázlat pontosításával dolgoztam ki. A 4.1. ábra és a 4.2. ábra összevetésével megfigyelhető, hogy a *DataChangeListener* és *MeasurementSource* absztrakt fogalmak helyett milyen konkrét osztályok szerepelnek az alkalmazásban.



4.2. ábra. Osztálydiagram

Az alkalmazás alapvető működését a 4.3. ábra szemlélteti.



4.3. ábra. Szekvencia diagram

Megjegyzés:

A Service komponens alapból az alkalmazás main szálában fut. Ezért a *MeasurementService* komponens indítása után új szálat indít az audio interfészen érkező impulzusok keresésére (*ImpulseFinder*), hogy a mérések elvégzése és az adatok megjelenítése ne akadályozzák egymást, ahogy azt a 8. követelmény előírja.

4.3. Mérésadatgyűjtés

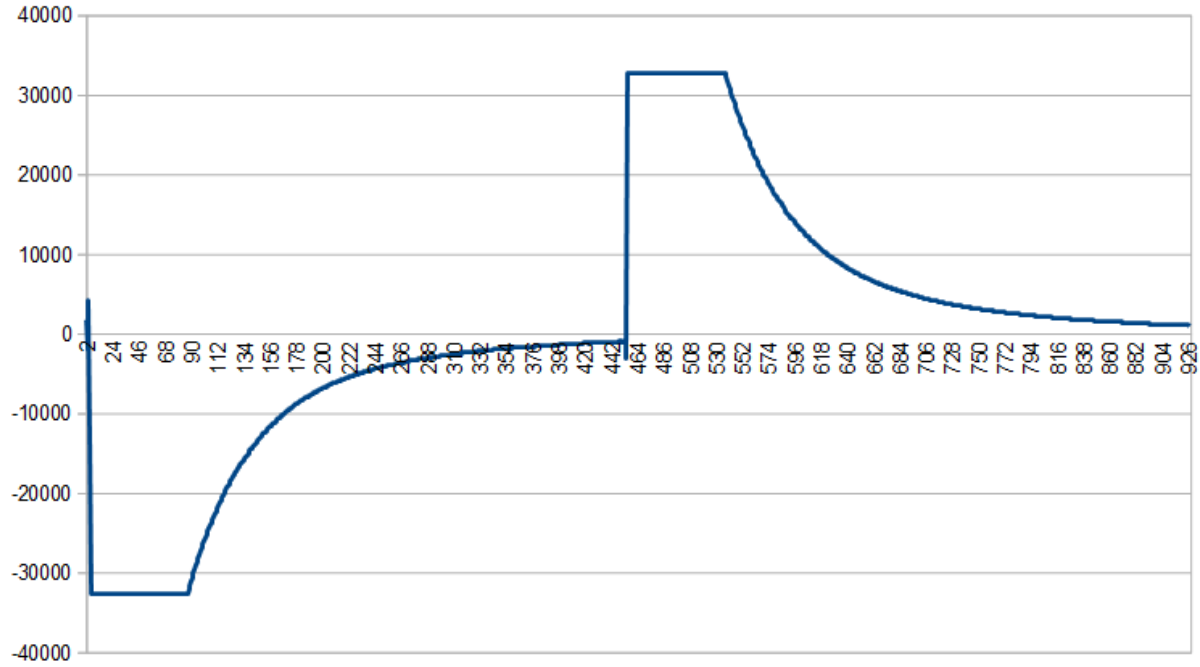
4.3.1. A kerékfordulat érzékelőről érkező jel feldolgozása

A készülék audio bemenetén keresztül érkező jel feldolgozása az alábbiak szerint történik:

Az audio bemenetet a készülék adott frekvenciával mintavételezi. A mintavételi frekvencia értéke ugyan megadható, de a 44100Hz az egyetlen olyan érték, ami biztosan támogatott minden készüléken, ezért ezt választottam. Hasonló a helyzet az egyes minták kódolásával is: a 16 bites PCM kód az egyetlen garantáltan támogatott formátum.

A készülék a felvett mintákat automatikusan egy bufferbe helyezi. Az alkalmazás ebből a bufferből olvashatja ki a felvett mintákat az *AudioRecord[19]* osztály segítségével.

Alapesetben a felvett minták abszolút értéke alacsony. Kerékfordulat esetén azonban a felvett minták a 4.4. ábra látható jelalak szerint alakulnak.



4.4. ábra. Kerékfordulat impulzus. X: minták sorszáma. Y: minták értéke

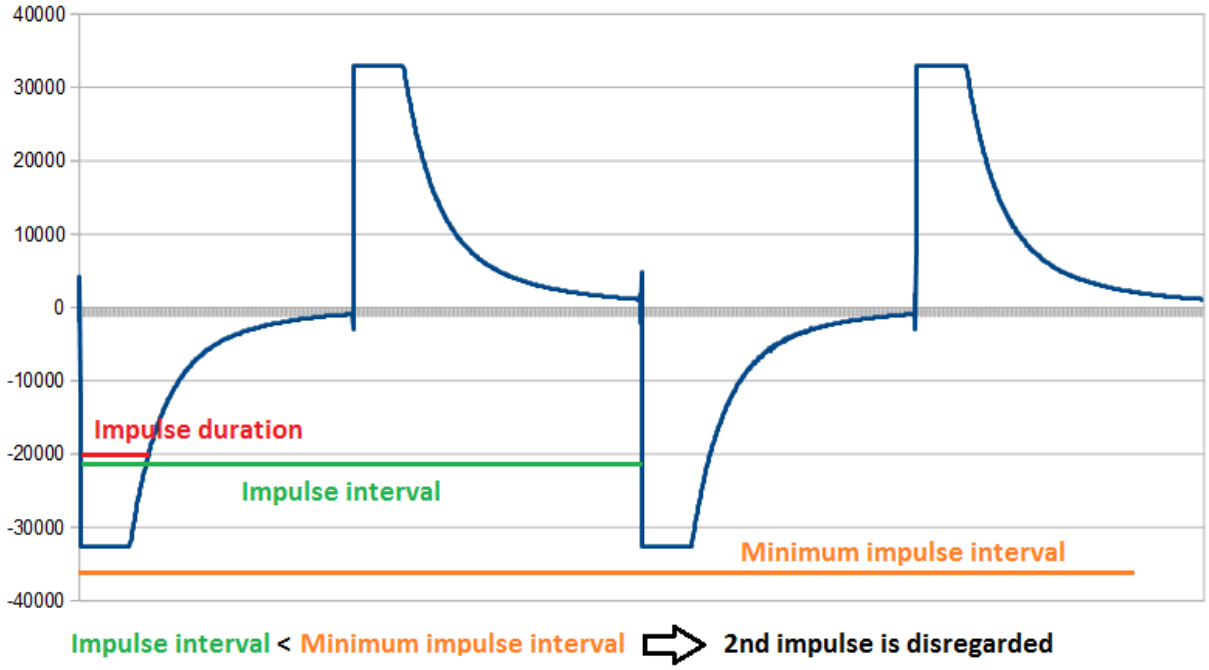
Tehát kerékfordulatonként érkezik egy negatív és egy pozitív értékű impulzus. Az impulzus abszolút értéke néhány mintavétel alatt eléri a 16 bites számábrázolási tartomány megfelelő szélső értékét, majd kis idő múlva exponenciálisan csökkenni kezd.

4.3.1.1. Kerékfordulatok érzékelése

Az 1. követelmény alapján az alkalmazásnak képesnek kell lennie a kerékfordulatot jelentő impulzusok felismerésére. A fenti jelalakból a kerékfordulatok érzékelését az alábbi feltétel szerint végzem:

Ha az aktuális minta értéke egy bizonyos érték (*impulseLevel*, jelenleg -20000) alatti, és a legutóbbi impulzus kezdete óta legalább *minImpulseInterval* idő eltelt, akkor egy új impulzus

érkezett. A második feltétel azért szükséges, mert adott impulzus esetén több egymást követő minta értéke is az *impulseLevel* szint alatt marad. A módszer előnye a klasszikus lefutóél-detekcióval szemben az, hogy minimális pergésmentesítésre is alkalmas: ha a *minImpulseInterval* ideje alatt újabb impulzus érkezik az előbbihez túl közel (pergés miatt), akkor ezt nem érzékeljük új impulzusnak (4.5. ábra).

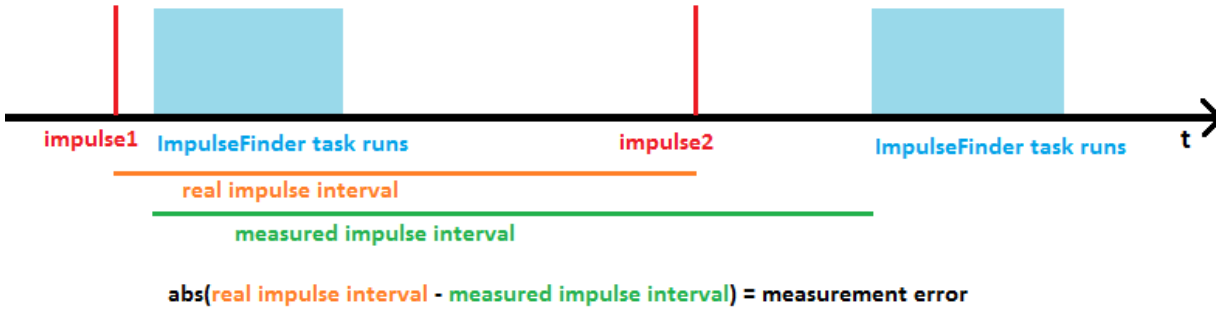


4.5. ábra. Figyelmen kívül hagyott impulzus

Azonban a *minImpulseInterval* értéke nem lehet akármilyen nagy, ugyanis a közvetlenül egymás után érzékelt impulzusok között eltelt idő fordítottan arányos a kerékpár egy kerékfordulat alatti átlagsebességével. Tehát, ha a kerékpár gyorsan halad, akkor rövid idő alatt tesz meg egy fordulatot a kereke, és ha ez az idő kisebb a választott *minImpulseInterval* értéknél, akkor nem érzékeltünk egy értékes impulzust.

4.3.1.2. Kerékfordulatok között eltelt idő mérése

A 2. követelmény alapján a kerékpár egy kerékfordulat alatti átlagsebességének elfogadható pontosságú meghatározásához a kerékfordulatok között eltelt időt legalább ms-os pontossággal kell tudni mérni. Azonban az Android operációs rendszer nem valósidejű[20], ami jelen esetben problémát jelent, mivel nem garantálható az impulzusokat kereső szál számára, hogy a *System timer* értékét az impulzus beérkezése után adott időn belül ki tudja olvasni. A problémát a 4.6. ábra szemlélteti.



4.6. ábra. Időmérési hiba

Ezért **az impulzusok között eltelt időt a két impulzus között beérkező minták számlálásával mérem.** Ebből, és a mintavételi frekvenciából kiszámítható az impulzusok között eltelt idő:

```
float impulseIntervalInSeconds = ((float)impulseIntervalCounter) / ((float)sampleRate);
```

Mivel a mintavételi frekvencia 44100Hz, ezért az előbbi módszerrel az impulzusok között eltelt időt 1/44100 másodperc pontossággal lehet mérni, ami eleget tesz a követelményeknek.

4.3.1.3. DataStore értesítése

Új impulzus érkezésekor az *ImpulseFinder* aszinkron módon értesíti a *DataStore* osztályt, paraméterül átadva az utolsó két impulzus között eltelt idő értékét. Az aszinkron hívás azért szükséges, mert az audio interfészen érkező jel feldolgozása nem állhat le a mérési adatok számítása alatt.

```
// Start calculations on another thread
(new Thread(){
	public void run(){
		DataRefresher ds = DataStore.getInstance();
		ds.impulseArrived(impulseIntervalInSeconds);
	}
}).start();
```

4.3.1.4. Érvénytelen impulzusok kiszűrése

Az alkalmazás első tesztjei során azt tapasztaltam, hogy mikor a kerékpár alacsony sebességgel halad és a mágnes éppen közel van az érzékelőhöz, akkor egy impulzus helyett gyakran két impulzus érkezik egymás után. A jelenséget az előbbiekben ismertetett pergésmentesítés mérsékelte ugyan, de nem szüntette meg teljesen. Ezért az érvénytelen impulzusok kiszűrésére az alábbi algoritmust dolgoztam ki:

Új impulzus érkezésekor:

1. Kiszámítom az utolsó néhány (jelenleg 5) impulzus intervallum átlagát.
2. Ha az új intervallum jelentősen kisebb az előbb kiszámított átlagtól, akkor az új impulzus érvénytelen.

Fontos, hogy az átlagot az érvénytelen impulzusokhoz tartozó intervallumok figyelembe vételével számítom ki. Ez azért szükséges, mert ennek hiányában hirtelen gyorsítás esetén

előfordulhatna, hogy a gyorsítás után bizonyos sebesség felett minden új impulzus érvénytelennek számítana. A módszer hátránya így viszont az, hogy ha több valóban érvénytelen impulzus érkezik közvetlenül egymás után, azok kellően lecsökkentik az átlagintervallumot ahhoz, hogy ezután akár egy valóban érvénytelen intervallumot érvényesnek ítél meg az algoritmus. A tesztek során azonban ez a probléma nem jelentkezett, az algoritmus helyesen működött.

4.3.2. Mérési adatok számítása

A *DataStore* kerékfordulatonként megkapja az utolsó 2 impulzus között eltelt időt (`interval`). Az utolsó kerékfordulat alatti átlagsebesség számításához emellett még szükség van a kerék kerületére (`deltaDistanceInKilometers`), ami az alkalmazás menüjében állítható be. Ezekből az adatokból már kiszámíthatóak a hagyományos kilométeróra funkciók megvalósításához szükséges mérési adatok (3. követelmény), az alábbiak szerint:

Kerékfordulatok száma másodpercenként:

```
impulsesPerSec = 1.0 / interval;
```

Megtett út:

```
tripDistanceInKiloMeters += deltaDistanceInKilometers;
overallDistanceInKiloMeters += deltaDistanceInKilometers;
```

Időtartamok:

```
// If start
if(tripStartTimestampInMillis == 0){
	// Set start timestamp
	tripStartTimestampInMillis = SystemClock.elapsedRealtime();
}
else{
	tripTimeInSeconds = ((double)(SystemClock.elapsedRealtime() -
tripStartTimestampInMillis)) / 1000.0;
	tripRideTimeInSeconds += interval;
	overallRideTimeInSeconds += interval;
}
tripStandTimeInSeconds = tripTimeInSeconds - tripRideTimeInSeconds;
```

Pillanatnyi sebesség (valójában kerékfordulat alatti átlagsebesség):

```
instantaneusSpeedInKmhs = deltaDistanceInKilometers * impulsesPerSec * 3600.0;
```

Gyorsulás:

```
// Calculate acceleration
// Now instSpeed is current speed, combined speed is last speed
accelerationInMeterPerSecSquare = ((instantaneusSpeedInKmhs - combinedSpeedInKmhs)
* impulsesPerSec) / 3.6;
```

Átlagsebességek:

```
if(tripRideTimeInSeconds > 0){
	tripAvgSpeedInKmhs = (tripDistanceInKiloMeters / tripRideTimeInSeconds) *
3600.0;
}
if(tripTimeInSeconds > 0){
	tripAvgSpeedIncludingStandTime = (tripDistanceInKiloMeters /
tripTimeInSeconds) * 3600.0;
}
if(overallRideTimeInSeconds > 0){
	overallAvgSpeedInKmhs = (overallDistanceInKiloMeters /
overallRideTimeInSeconds) * 3600.0;
}
```

Maximális sebességek:

```
if(instantaneusSpeedInKmhs > tripMaxSpeedInKmhs){
	tripMaxSpeedInKmhs = instantaneusSpeedInKmhs;
}
if(instantaneusSpeedInKmhs > overallMaxSpeedInKmhs){
	overallMaxSpeedInKmhs = instantaneusSpeedInKmhs;
}
```

Az aktuális átlagsebesség alatt/felett megtett út és eltöltött idő:

```
// If speed is below instantaneous avg
if(instantaneusSpeedInKmhs < tripAvgSpeedInKmhs){
	dstBelowInstAvg += deltaDistanceInKilometers;
	timeBelowInstAvg += impulsePeriodInSeconds;
}
// If speed is above (or equal to) instantaneous avg
else{
	dstAboveInstAvg += deltaDistanceInKilometers;
	timeAboveInstAvg += impulsePeriodInSeconds;
}
```

4.3.3. GPS adatok felhasználása

A 4. követelmény szerint a beépített GPS szenzor segítségével is meg kell határozni a pillanatnyi sebességet és a megtett távolságot. A GPS adatok lekérését az Android rendszertől a *LocationFinder* osztály végzi. Ez az osztály megvalósítja a *LocationListener* interfészt, ezért beregisztrálható a GPS adatok figyelésére a *LocationManager[21]* segítségével:

```
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, this);
```

A 2. (*minTime*) és 3. (*minDistance*) paraméterekkel az elvárt frissítési gyakoriság állítható be. A rendszer képességeinek teszteléséhez ezeket a paramétereket nullára állítottam be, hogy az elérhető legmagasabb gyakorisággal kapjak új adatokat. Azonban az esetek többségében a *minTime* paraméter értékének érdemes nagyobb értéket választani a fogyasztás csökkentésének érdekében.

Új GPS adat érkezésekor egy *Location[22]* objektumot kapunk. A *LocationListener* a kapott Location objektumot továbbadja a *DataStore*-nak. A *DataStore* az új és az előző Location objektumok felhasználásával kiszámítja a két mérés közötti távolságot:

```
// Get distance between last 2 locations
double lastPassageInKiloMeters = (double) lastLocation.distanceTo(newLocation) /
1000.0;
```

Ezeket összegezve kiszámítható a GPS alapján megtett út:

```
// Increment trip distance
this.GPStripDistanceInKiloMeters += lastPassageInKiloMeters;
```

A Location objektumból lekérdezhető az aktuális tengerszint feletti magasság is, amennyiben azt a GPS vevő képes meghatározni. A magasság adatok alapján szintkülönbségeket is számítok. A Location objektumból továbbá lekérdezhetőek az aktuális földrajzi szélesség és hosszúság adatok, valamint ezek hozzávetőleges pontossága is.

A GPS adatok alapján a pillanatnyi sebességet 2 módszerrel is meghatároztam:

- **GPScurrentSpeedInKmhs**: A pillanatnyi sebességet a Location objektumból közvetlenül lekérdezve.
- **speed2**: A pillanatnyi sebességet a legutóbbi 2 helyadat közötti távolság és a 2 mérés között eltelt idő hányadosával becsülve. A módszer hátránya, hogy a Location objektum *API Level 17* alatt nem tartalmaz pontos időbélyeget, ezért a tesztek során ez a sebesség várhatóan pontatlan lesz.

4.3.4. Fused Location Provider API[23]

Android platformon a Location adatok a *LocationManager* mellett a *Fused Location Provider API* segítségével is lekérdezhetőek, amennyiben a készüléken telepítve van a *Google Play Services[24]*. A *Fused Location Provider* az aktuális pozíció meghatározását több különböző *Location Provider* (például GPS vagy *Network Provider*) együttes használatával végzi, ezáltal magasabb szintű hozzáférést nyújt a helyadatokhoz.

Az API használata hasonló a *LocationManager* használatához. A kívánt frissítési gyakoriság itt is beállítható, azonban a helyadatokat szolgáltató Provider nem, mivel ez az API által nyújtott absztrakciós szint alatt van. Helyette néhány konstans közül kiválasztható, hogy inkább pontos helyadatokra vagy alacsony fogyasztásra van-e szükség.

Inicializálás:

```
// Set current class to handle callbacks
client = new GoogleApiClient.Builder(context)
        .addApi(LocationServices.API)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .build();
client.connect();

// Set location request parameters
request = new LocationRequest();
// Set refresh rate to fastest available
request.setInterval(0);
request.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
```

Frissítések kérése:

```
LocationServices.FusedLocationApi.requestLocationUpdates(client, request, this);
```

A *LocationManager* által szolgáltatott adatokkal való összehasonlítás érdekében a pillanatnyi sebesség, megtett út és pontosság adatokat a *Fused Location Provider API* felhasználásával is meghatároztam. Mivel szabadtéren az okostelefonnal jelenleg elérhető helymeghatározási technológiák közül a GPS a legpontosabb, ezért a *Fused Location Provider API* várhatóan a *GPS Provider* által biztosított helyadatokat fogja a tesztek során az alkalmazás számára biztosítani.

4.3.5. Mért adatok mentése

4.3.5.1. Mérési eredmények mentése

A mérések elvégzéséhez az adatokat a 6. követelmény alapján CSV fájlokba kell menteni. Ezt a feladatot a *DataLogger* osztály végzi. Ez az osztály a mért adatokból 100ms-onként készít egy rekordot, amit hozzáfűz az aktuális log fájl végéhez. Ehhez definiálni kellett egy szálat, ami elvégzi a feladatot, majd be kellett állítani a *ScheduledExecutorService[25]* osztály segítségével, hogy a szál kódja 100ms-onként lefusson. A log írását végző szál prioritása alacsony, mivel a 8. követelmény alapján a log írása nem akadályozhatja a mérési adatok gyűjtését, továbbá a felhasználói felület se lassulhat le emiatt.

A log rekordok írását végző szál forráskódja:

```
private final Runnable loggerTask = new Runnable(){
      public void run() {
      android.os.Process.setThreadPriority(android.os.Process.THREAD_PRIORITY_BACK
GROUND);
            createRecord();
            writeRecord();
      }
};
```

A *loggerTask* ütemezése:

```
// Run logger task periodically after 100ms
loggerTaskHandler = scheduler.scheduleAtFixedRate(loggerTask, 100, 100,
TimeUnit.MILLISECONDS);
```

4.3.5.2. Statisztikák mentése

A mérésekhez szükséges log fájlok készítése mellett szükség van a hosszabb távon értelmezhető számlálók aktuális értékének mentésére és betöltésére is. Ilyen számláló például az összes megtett út értékét tároló változó. Ezen adatok mentését és betöltését a *MeasurementService* komponens kezdeményezi az *onDestroy()* és *onCreate()* metódusaiban.

4.4. Adatmegjelenítés

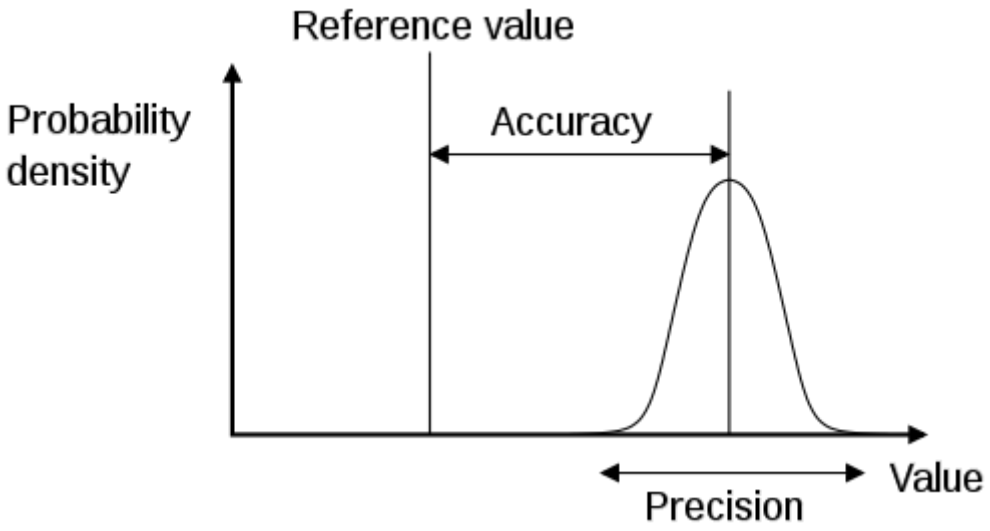
A 7. követelmény alapján a mérések aktuális értéket meg kell tudni jeleníteni a felhasználói felületen. A mért adatok áttekinthető megjelenítésére az okostelefonok kijelzője lehetőséget biztosít. Például a *CurrentActivity currentSpeedTV* elemének színe az alapján változik, hogy a pillanatnyi sebesség az átlagsebesség alatt vagy felett van-e, ezzel könnyen értelmezhető formában jeleníti meg az említett két sebesség értékének pillanatnyi különbségét.

5. Mérési eredmények

Az előbbiekben ismertetett kerékpáros sebesség-meghatározási módszerek vizsgálatához 3 mérést végeztem el. A mérések során a kerékpár sík terepen, emelkedőn és lejtőn is haladt, így lehetőség adódott az egyes módszerek összehasonlítására alacsony-, magas-, közel egyenletes- és változó sebességek esetén is.

Az egyes mérések elvégzése után a mért eredményeket elemeztem. Ehhez felhasználtam a *FreeMat[26]* programot. Az egyes mérések eredményeinek vizsgálata új ötleteket adott az Android alkalmazás továbbfejlesztésére, amit a következő mérés előtt elvégeztem, majd a következő mérésen teszteltem az új megoldások működését.

A vizsgált módszerek pontosságának meghatározásához fontos tisztázni a pontosság fogalmát, ugyanis a magyar pontosság szó utalhat az angol *accuracy* és *precision* szavakra is. A két fogalom közötti különbséget az 5.1. ábra szemlélteti.



5.1. ábra. Accuracy and Precision[30]

A továbbiakban pontosság alatt a *„precision”* fogalmat értem, az *„accuracy”* fogalom kifejezésére pedig a hitelesség szót használom.

A fejezet további részeiben a mérési eredmények elemzésével mutatom be a vizsgált sebesség-meghatározási módszerek jellemzőit.

5.1. 1. mérés

Dátum: 2014.09.30.

Készülék: Samsung GT-S5570, Android 2.3.3 operációs rendszer

Mért mennyiségek: Timestamp,Wheel rotation period,Trip distance,Instantaneous speed,Trip time,Trip ride time,Trip stand time,Trip average speed,Trip avg speed with stand time,GPS current speed 2,GPS current speed,GPS trip distance,GPS trip time,GPS trip avg speed with stand time,Current altitude

5.1.1. Sebességek összehasonlítása

Vizsgált adatsorok:

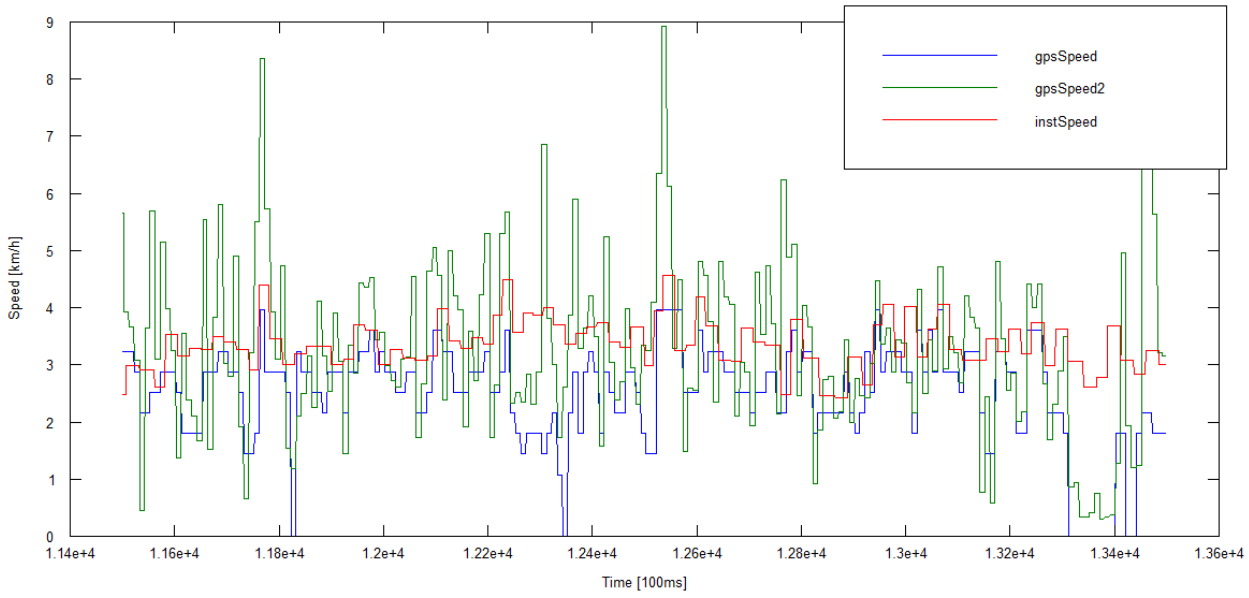
- **instSpeed:** a kerékszenzor által mért pillanatnyi sebesség
- **gpsSpeed:** a GPS szenzor által mért pillanatnyi sebesség
- **gpsSpeed2:** a GPS szenzor által az utolsó 2 mérési pont között mért távolság és a közben eltelt idő hányadosa

5.1.1.1. Alacsony, közel egyenletes sebességgel megtett szakasz

Az egyes sebességmérési módszerek pontosságát először egy közel egyenletes sebességgel megtett szakaszon mért adatok (5.2. ábra) alapján hasonlítottam össze, az egyes adatsorok szórásának vizsgálatával. Mivel az adott szakaszon közel egyenletes sebességgel haladt a kerékpár, ezért a nagyobb szórás nagyobb mérési pontatlanságot (*less precision[27]*) jelent.

Korrigált tapasztalati szórás:

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$ [28]



5.2. ábra. Alacsony, közel egyenletes sebességgel megtett szakasz

Szórások:

instSpeed: 0.4322

gpsSpeed: 0.8946

gpsSpeed2: 1.5130

Látható, hogy a *gpsSpeed2* adatsor szórása a legnagyobb.

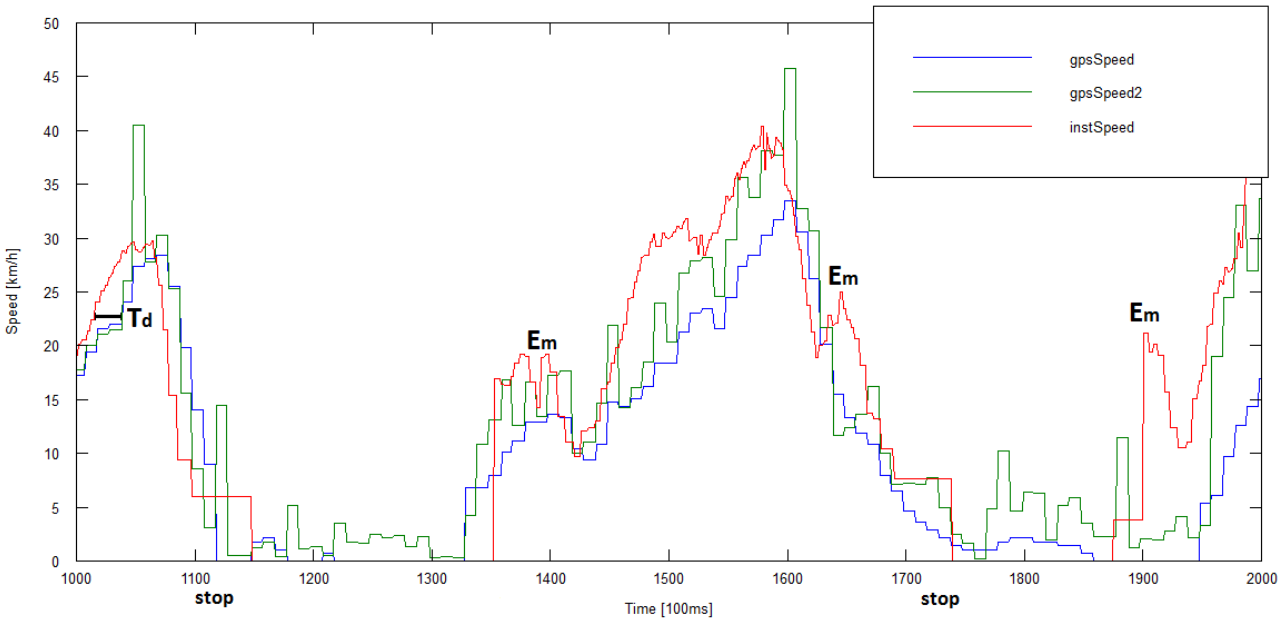
Ezt a *gpsSpeed2* sebesség számításához felhasznált időalap pontatlansága vagy a helyadatok bizonytalansága okozhatja.

Az elemzéshez szükséges kódrészlet:

```
%Choose file
file = csvread('D:/bme/7/bikeComputer/logs/csv_comma/log_2014-09-30_10-51-
06.csv');
%Choose records to display
index = [11500:13500];
instSpeed = file(index,4);
gpsSpeed = file(index,11);
gpsSpeed2 = file(index,10);

%Plot to a common figure
plot(index,gpsSpeed,'b',index,gpsSpeed2,'g',index,instSpeed,'r'),
xlabel('Time [100ms]'),
ylabel('Speed [km/h]'),
legend('gpsSpeed','gpsSpeed2','instSpeed');

%Count standard deviations
%http://freemat.sourceforge.net/help/elementary_std.html[29]
std(instSpeed)
std(gpsSpeed)
std(gpsSpeed2)
```

5.1.1.2. Változó sebességgel megtett szakasz

A sebességmérési módszereket változó sebességgel megtett szakaszon mért adatok alapján is vizsgáltam. Változó sebesség esetén az adatok szórása nem használható a pontosság meghatározására. Ezért az adatsorokból ábrázolt grafikon (5.3. ábra) alapján következtettem a vizsgált módszerek további jellemzőire.



5.3. ábra. Változó sebességgel megtett szakasz

Nagyobb sebességeknél a kerék gyakrabban fordul körbe, ezért gyakrabban érkezik új impulzus a kerékszenzorról. Ezért a kerékszenzor által mért adatsor időbeli felbontása ilyenkor nagyobb, mint a GPS szenzor által mért adatsorok felbontása, mivel a GPS szenzorról maximális frissítési gyakoriság mellett is csak 1 másodpercenként érkezik adat.

Nagyobb sebességek esetén a kerékszenzor hamarabb érzékeli a sebesség változását. Ez abból látszik, hogy a *gpsSpeed* többnyire néhány másodperc késéssel (**Td**) követi az *instSpeed*-et.

A *gpsSpeed* gyakran nem érzékeli a rövid idő alatt bekövetkező sebesség változásokat. Ez a grafikonon az **Em** (*Event Missed*) jelzésű helyeken figyelhető meg. Ezeken a helyeken a *gpsSpeed* lemaradt az *instSpeed* grafikonján megfigyelhető lokális minimum/maximum sebességekről. Ezek a lokális minimumok és maximumok valódiak (nem az *instSpeed* mérési hibái), mivel több egymás utáni minta erősíti meg a létezésüket, nem 1-1 kiugró értékről van szó (mint pl ahogy az a *gpsSpeed2* esetén több helyen megfigyelhető).

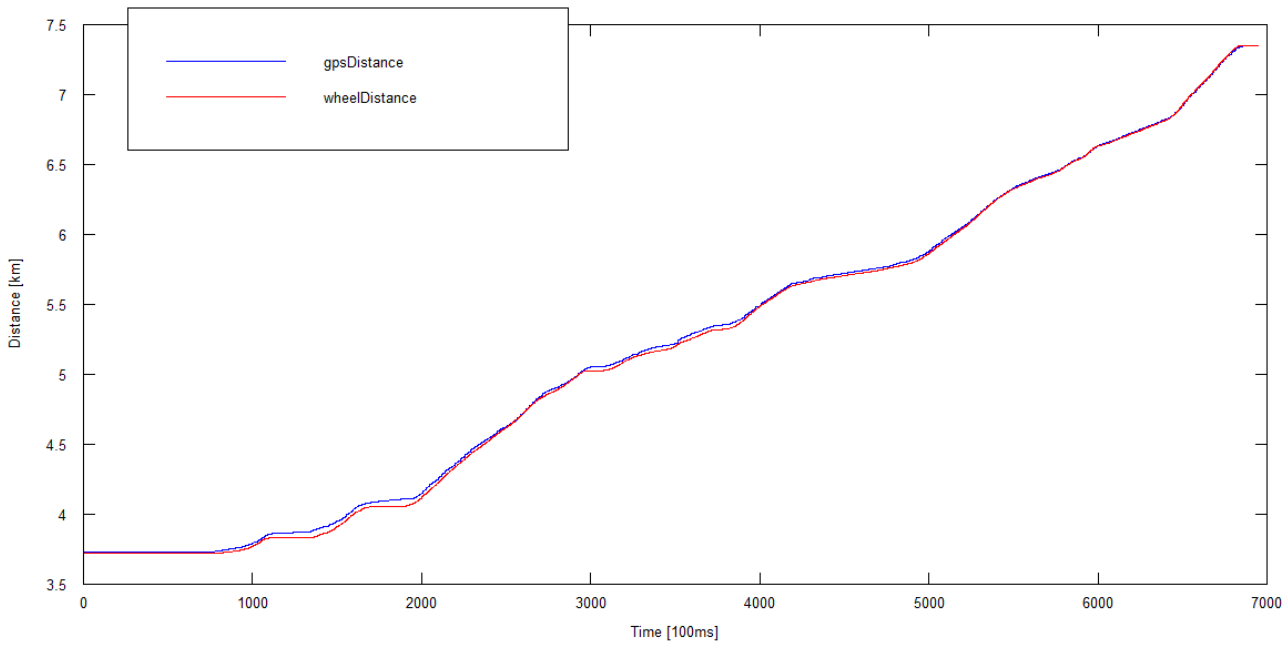
Kis sebességeknél a kerék ritkábban fordul körbe, ezért ritkábban érkezik új impulzus a kerékszenzorról. Ez elsősorban a megállások detektálásánál okoz gondot: bizonyos idő eltelte után tudjuk csak elfogadható valószínűséggel meghatározni, hogy a kerékpár lassan halad vagy megállt-e. A megállások detektálásához a *gpsSpeed* nyújthat segítséget, ugyanis az mindig 1 másodpercenként frissül. (**stop**)

Az elemzéshez szükséges kódrészlet:

```
%Choose file
file = csvread('D:/bme/7/bikeComputer/logs/csv_comma/log_2014-09-30_11-41-
23.csv');

%Choose records to display
index = [1000:2000];

instSpeed = file(index,4);
gpsSpeed = file(index,11);
gpsSpeed2 = file(index,10);

%Plot to a common figure
plot(index,gpsSpeed,'b',index,gpsSpeed2,'g',index,instSpeed,'r'),
xlabel('Time [100ms]'),
ylabel('Speed [km/h]'),
legend('gpsSpeed','gpsSpeed2','instSpeed');
```

5.1.2. Távolságok összehasonlítása

A sebességek mellett a GPS és a kerékszenzor által mért távolságokat is összehasonlítottam. Erre azért volt szükség, mert az előzőekben csak a mérések pontosságát vizsgáltam, hitelességét (*accuracy*) nem.

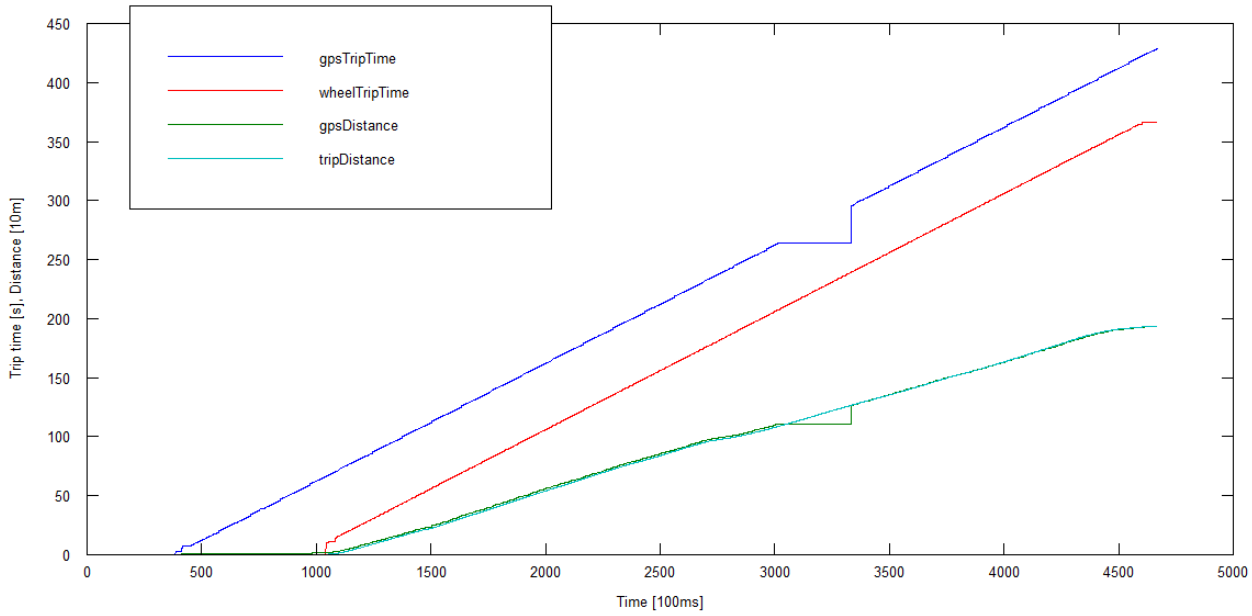


5.4. ábra. Távolságok

Az 5.4. ábra látható, hogy a kerékszenzor és a GPS szenzor által mért távolságok hosszú távon megegyeznek. Ebből arra lehet következtetni, hogy a két különböző módszer szerint mért távolságok hitelessége (*accuracy*) megegyezik, azaz a valójában megtett távolságtól hosszú távon azonos mértékben térnek el. Következtetés: a kerék kerület érték helyesen van beállítva, a különböző módszerekkel mért sebességekben tapasztalható eltéréseket nem ez okozza.

5.1.3. Hiányzó GPS adatok

A GPS és kerékszenzor távolságokat több szakaszon mért adatok alapján is összehasonlítottam. Az egyik szakaszon az alábbi jelenség figyelhető meg:



5.5. ábra. Hiányzó GPS adatok

Látható, hogy a GPS távolság adatok a 3000-es időbélyegtől kezdve közelítőleg 30 másodperc ideig nem frissültek, miközben a kerékszenzor szerint a kerékpár haladt. Ez idő alatt a *GPS Time* adat se frissült, tehát az eltérés oka az, hogy a GPS szenzorról éppen nem érkezett új adat.

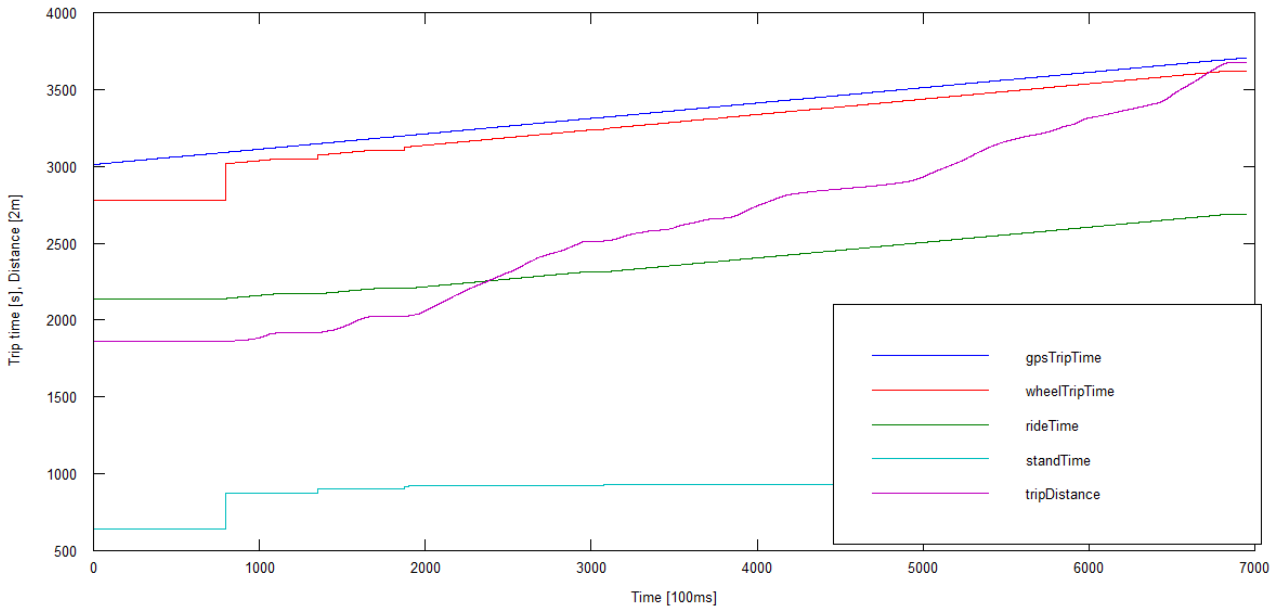
Az 5.5. ábra megfigyelhető továbbá a GPS és a kerékszenzor idők közötti konstans eltérés. Ennek oka az, hogy a GPS az időt az első GPS adat érkezésétől számolja, a kerékszenzor pedig az első érvényes impulzus beérkezésétől (első indulástól).

Az elemzéshez szükséges kódrészlet:

```
%Choose file
file = csvread('D:/bme/7/bikeComputer/logs/csv_comma/log_2014-09-30_10-40-
18.csv');

%Choose records to display
%All records
index = [2:size(file,1)];

tripDst = file(index,3);
wheelTripTime = file(index,5);
gpsDst = file(index,12);
gpsTripTime = file(index,13);

%Plot to a common figure
plot(index,gpsTripTime,'b',index,wheelTripTime,'r',index,gpsDst*100,'g',index,trip
Dst*100,'c'),
xlabel('Time [100ms]'),
ylabel('Trip time [s], Distance [10m]'),
legend('gpsTripTime','wheelTripTime','gpsDistance','tripDistance','location','nort
hwest');
```

5.1.4. Eltelt időt mérő változók frissítése

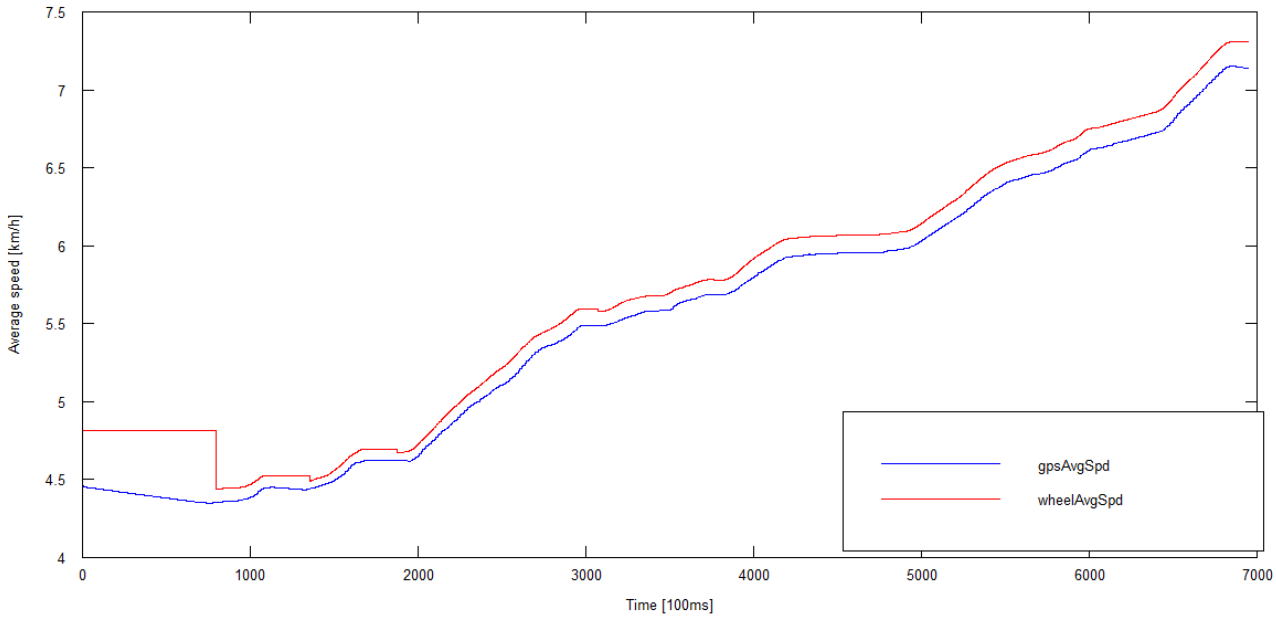
Az 5.6. ábra a mért idő és távolság adatsorok együttes vizsgálatával az is megfigyelhető, hogy a kerékszenzor által mért idő impulzusonként frissül, ezért nem változik az értéke, mikor a kerékpár áll.



5.6. ábra. Eltelt időt mérő változók frissítése

5.1.5. Átlagsebességek

Az előző pontban említett jelenség az átlagsebességek vizsgálatával is megfigyelhető (5.7. ábra).



5.7. ábra. Átlagsebességek

Az átlagsebességek eltérése az időmérés kezdeti időpontjának eltéréséből adódik (mivel a távolságok megegyeznek).

5.2. 2. mérés

Dátum: 2014.10.11.

Készülék: Samsung GT-S5570, Android 2.3.3 operációs rendszer

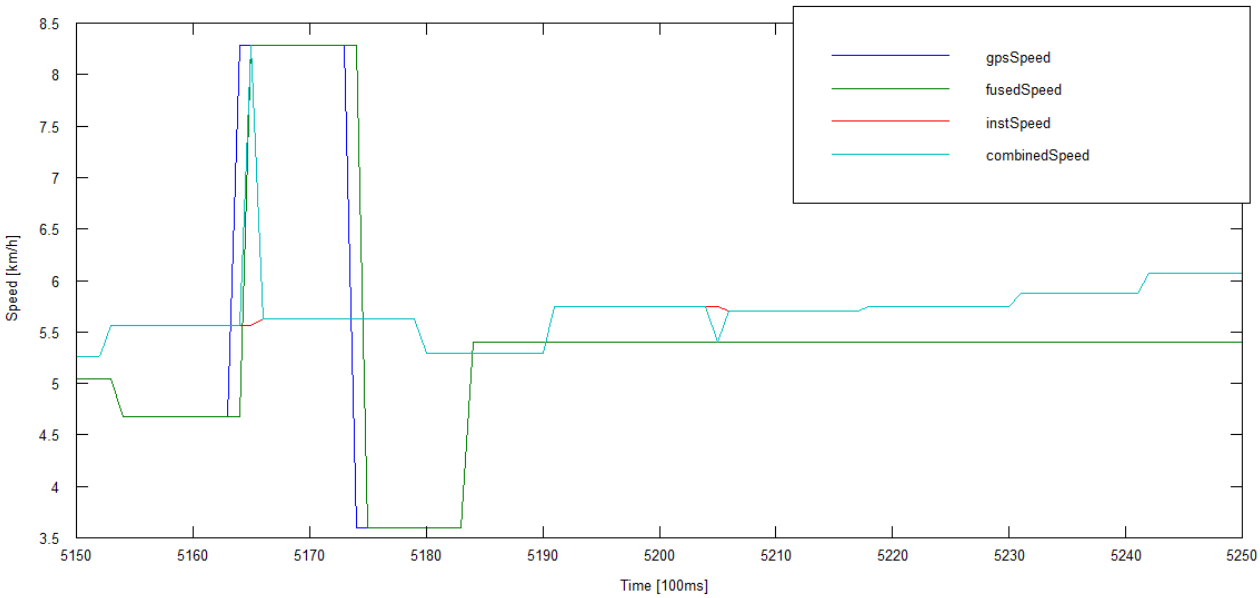
Változtatások az alkalmazáson az 1. mérést végző verzióhoz képest:

- *GPS Accuracy* érték mérése
- Gyorsulás számítása kerékszenzor adatok alapján
- *Altitude* érték korrigálása konstanssal (*altitude* érték hitelességének javítása)
- Pozíció adatok mérése
- *CombinedSpeed* bevezetése: ha a kerékszenzorról nem érkezik új mérési eredmény bizonyos ideig, akkor az aktuális sebesség meghatározásába besegít a GPS is
- *FusedLocation API* felhasználása

Mért mennyiségek: Timestamp,Wheel rotation period,Trip distance,Current speed,Combined speed,Acceleration,Trip time,Trip ride time,Trip stand time,Trip average speed,Trip avg speed with stand time,GPS current speed,GPS current speed 2,GPS trip distance,GPS trip time,GPS trip avg speed with stand time,Current altitude,Latitude,Longitude,GPS accuracy,Satellites,Trip uphill altitude,Trip downhill altitude,Fused current speed,Fused trip distance,Fused accuracy

5.2.1. Kombinált sebesség adatok (Combined speed)

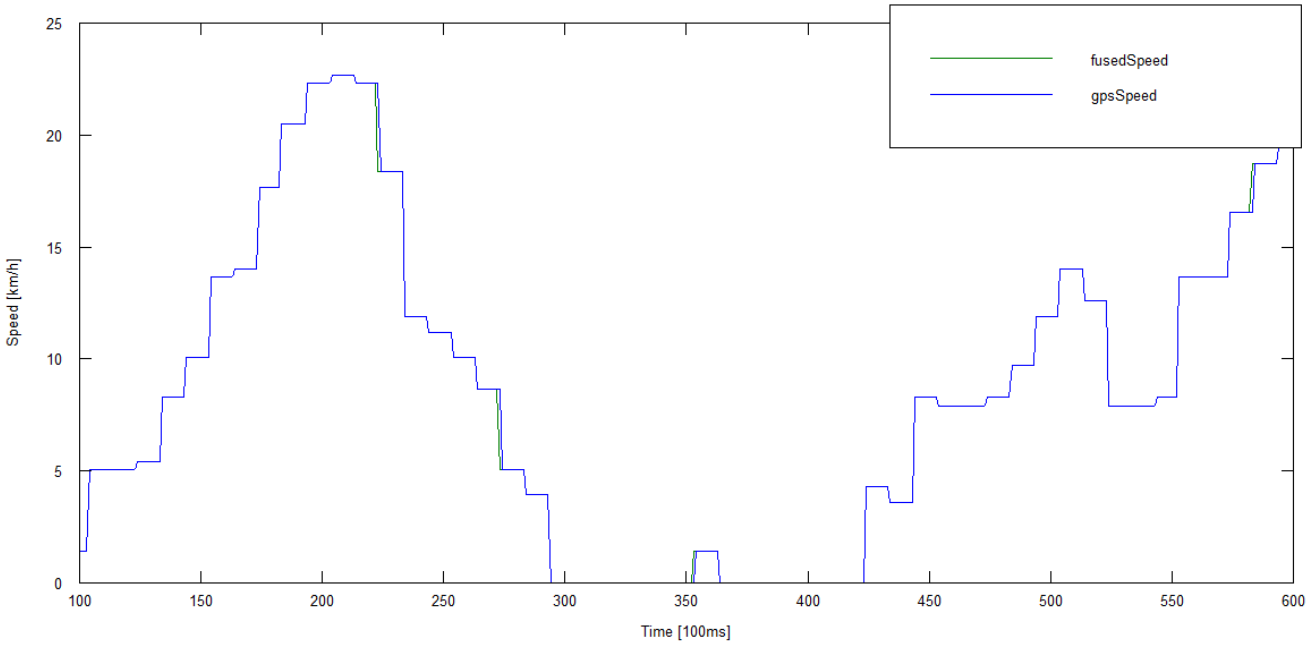
Az 1. mérés eredményei alapján látható, hogy míg nagy sebességeknél a kerékszenzor segítségével származtatott adatok frissülnek gyakrabban, addig alacsony sebességeknél a GPS adatok 1 Hz-es frissítési frekvenciája meghaladja a kerékfordulatok másodpercenkénti számát. Ezért az alkalmazást kibővítettem egy olyan sebesség-meghatározási módszerrel (*Combined speed*), ami a GPS és a kerékszenzor adatait egyaránt figyelembe véve határozza meg a pillanatnyi sebességet. Ezt úgy implementáltam, hogy ha a kerékszenzorról nem érkezik új mérési eredmény bizonyos ideig, akkor a *Combined speed* értékét a GPS adatok határozzák meg. Ezzel a módszerrel az a probléma, hogy mikor a GPS sebesség jelentősen eltér a kerékszenzor által mért sebességtől, akkor a *Combined speed* értéke „ugrál”, ahogy ez az 5.8. ábra is megfigyelhető.



5.8. ábra. Combined speed

5.2.2. Fused Location API által mért sebesség

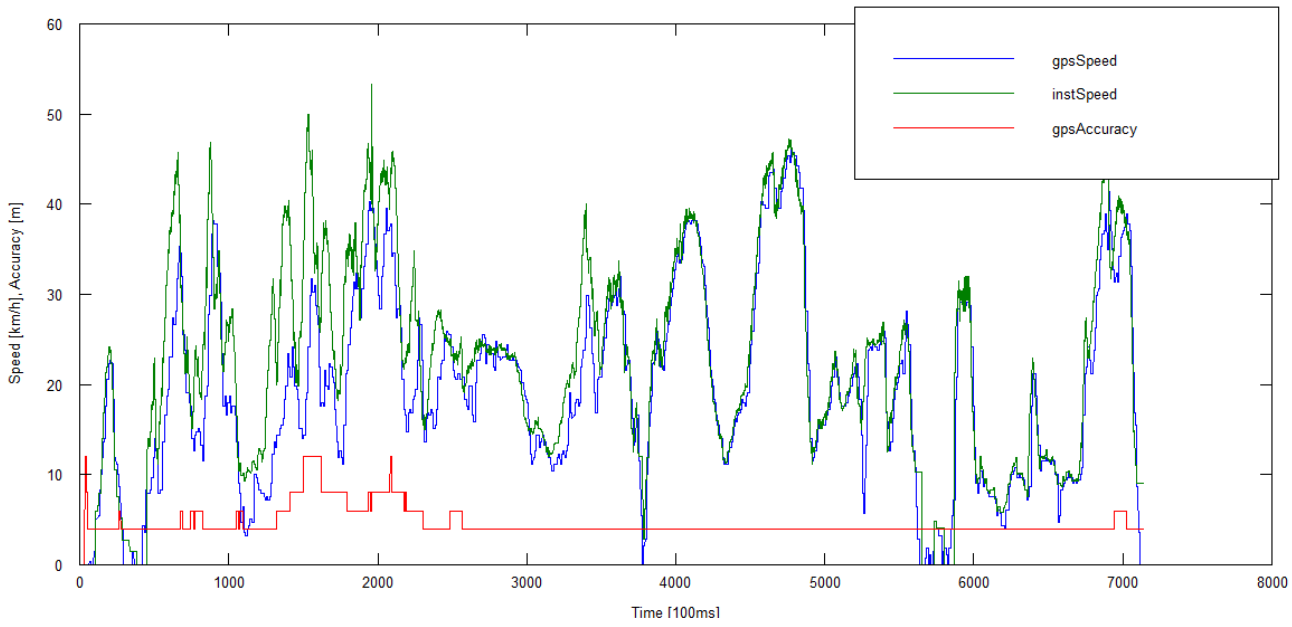
Az 5.9. ábra megfigyelhető, hogy a *LocationManager* és a *Fused Location Provider API* segítségével lekérdezett Location objektumokból kinyerhető sebesség adatok a mérések során megegyeztek. Ebből arra lehet következtetni, hogy *a Fused Location API* kültéren pontos helymeghatározásra GPS adatokat használ.



5.9. ábra. Fused speed

5.2.3. GPS sebesség és pontosság

A *Location* osztály *getAccuracy()* metódusának dokumentációja[22] alapján a metódus által visszaadott *accuracy* érték a *latitude* és *longitude* adatokat jellemzi, a sebesség adatokat nem. Ugyanakkor az 5.10. ábra alapján összefüggés figyelhető meg a kerékszenzor és a GPS szenzor által mért sebességek eltérése (különbsége) és az *accuracy* érték nagysága között.



5.10. ábra. GPS speed és accuracy

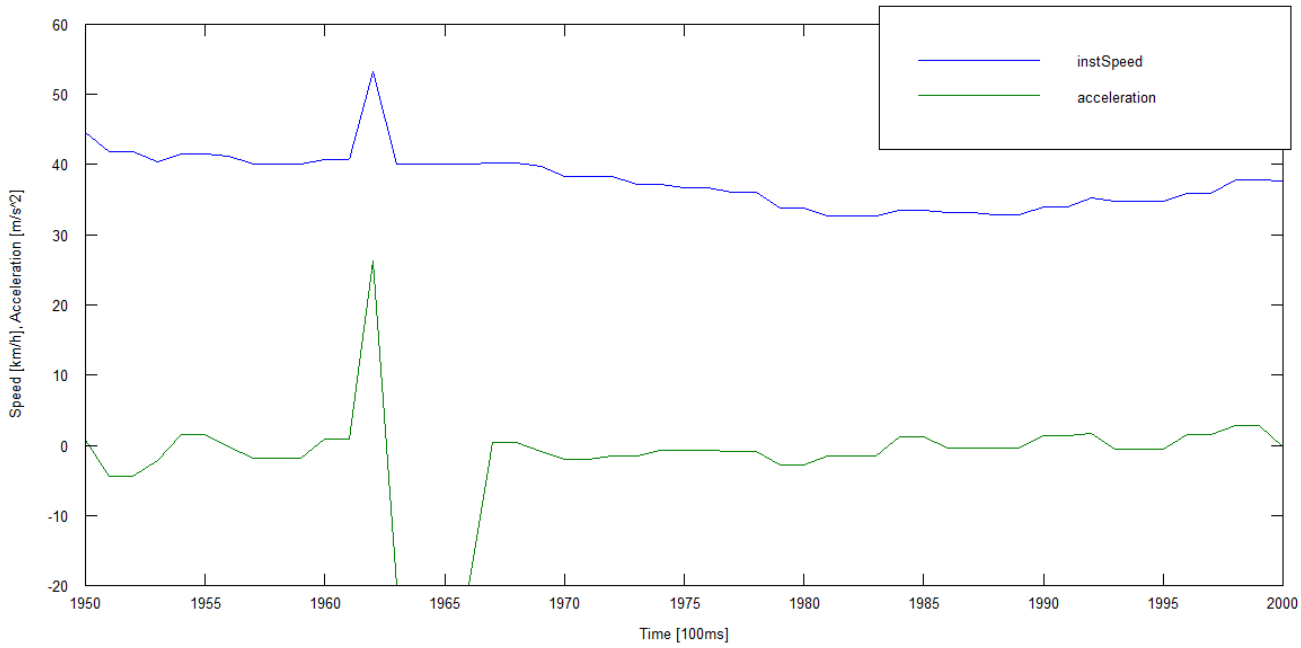
Látható, hogy ahol a GPS kevésbé pontosan tudja meghatározni az aktuális pozíciót (nagyobb *accuracy* érték), ott nagyobb az eltérés a kerékszenzor és a GPS által meghatározott sebességek között, tehát nagyobb *accuracy* érték esetén a GPS a sebességet is kevésbé pontosan képes meghatározni.

Az elemzéshez szükséges kódrészlet:

```
%Choose file
file     =     csvread('D:/bme/7/bikeComputer/logs/csv_comma/log_2014-10-11_16-55-
07.csv');

%Choose records to display
%All records
index = [2:size(file,1)];

instSpeed = file(index,4);
gpsSpeed = file(index,12);
accuracy = file(index,20);

%Plot to a common figure
plot(index,gpsSpeed,'b',index,instSpeed,'g',index,accuracy,'r')
xlabel('Time [100ms]')
ylabel('Speed [km/h], Accuracy [m]')
legend('gpsSpeed','instSpeed','gpsAccuracy');
```

5.2.4. Kerékszenzor mérési hiba

Bár a kerékszenzor által mért sebesség adatok pontossága felülmúlta a GPS adatok pontosságát, ez a módszer sem hibátlan. Az 5.11. ábra a kerékszenzor által mért sebességben egy mérési hiba fedezhető fel. A hiba észlelését a kerékszenzor adatok alapján számított gyorsulás érték segítette (a gyorsuláson a mérési hibák könnyebben észlelhetőek).



5.11. ábra. Kerékszenzor mérési hiba

5.3. 3. mérés

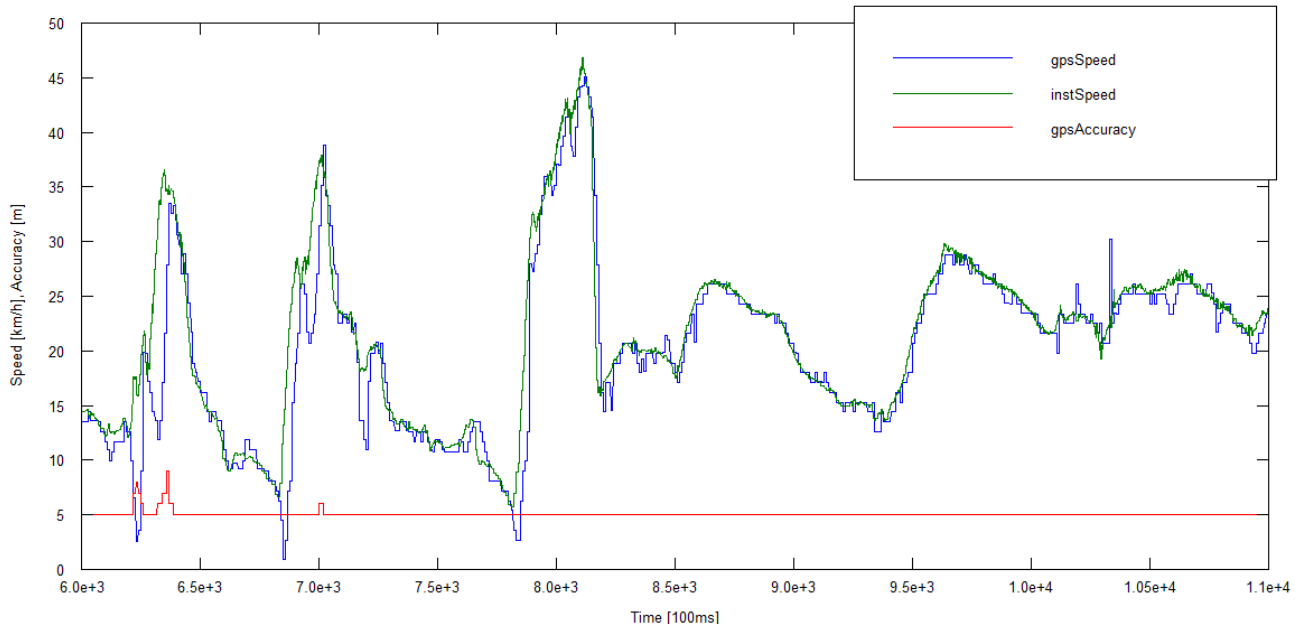
Dátum: 2014.10.13.

Készülék: Samsung GT-S7560, Android 4.0 operációs rendszer

Mért mennyiségek: megegyeznek a 2. mérés során mért mennyiségekkel

5.3.1. GPS pontosság

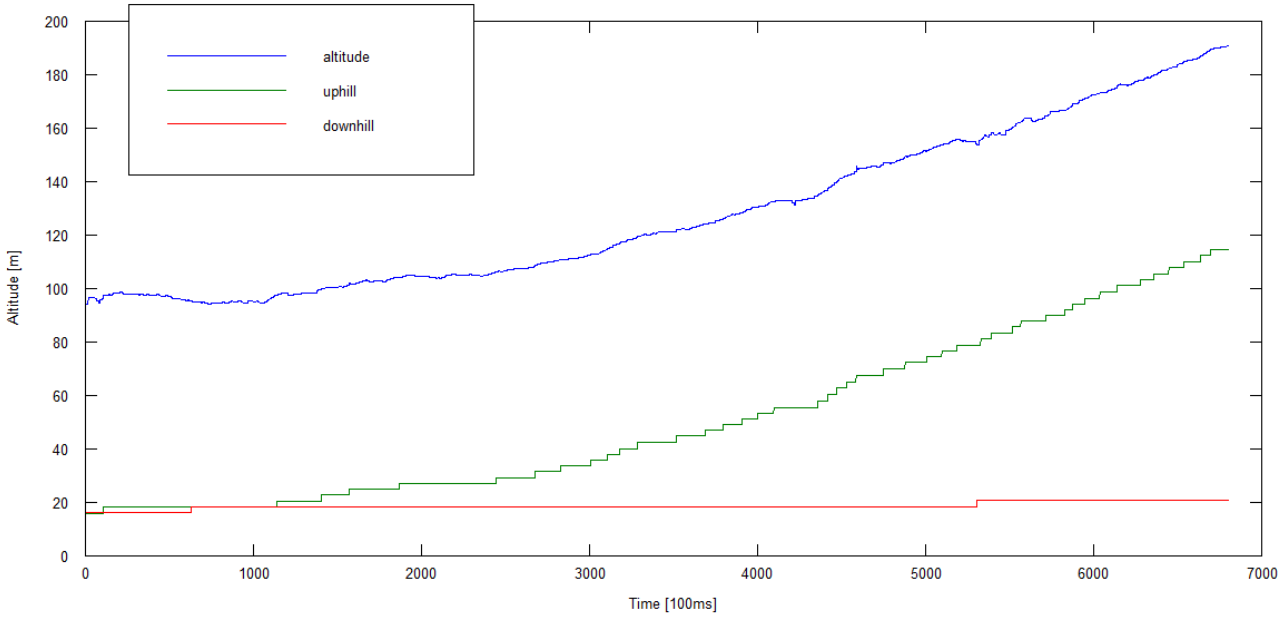
A 3. mérést az első két méréstől eltérő mobil készülékkel végeztem, hogy teszteljem a vizsgált sebesség-meghatározási módszerek esetleges hardverfüggését. Számottevő eltérést nem tapasztaltam, a sebesség-idő grafikonok jellege az előbbiekhez hasonló, ahogy a GPS *accuracy* értéke is (5.12. ábra).



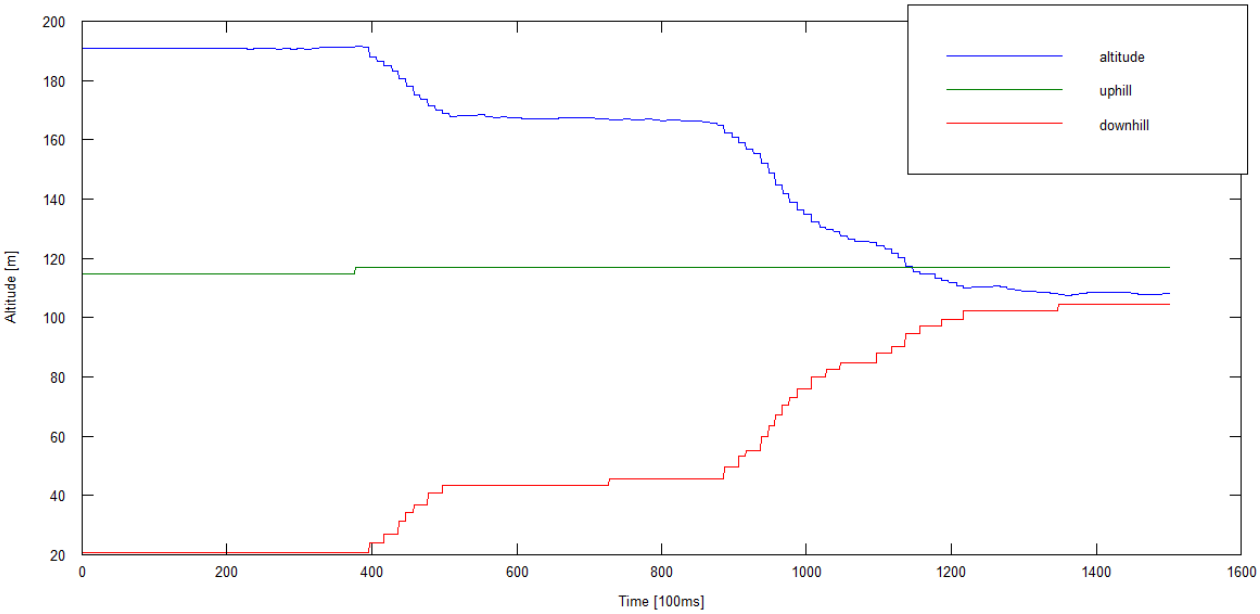
5.12. ábra

5.3.2. Szintkülönbségek

Az alkalmazás 2. méréshez használt verziójában a megtett szintkülönbséget számláló algoritmus hibásan működött. Ezt az újabb verzióban javítottam.



5.13. ábra. Emelkedő



5.14. ábra. Lejtő

Az 5.13. ábra és az 5.14. ábra alapján látható, hogy a javított szintkülönbség számláló algoritmus helyesen működik.

6. Összefoglalás

A dolgozatban mobil támogatással elérhető kerékpáros sebesség-meghatározási módszereket vizsgáltam. Ehhez egy egyedi hardveres megoldást és egy hozzá illeszkedő kerékpáros fedélzeti számítógép alkalmazást terveztem és valósítottam meg Android platformon. Méréseket végeztem a készülék mikrofon bemenetére csatlakoztatott kerékfordulat-érzékelő segítségével, valamint az okostelefon beépített GPS szenzora által szolgáltatott adatok felhasználásával is. Ezután a mérési eredmények kiértékelésével összehasonlítottam a vizsgált módszereket.

A vizsgált kerékpáros sebesség-meghatározási módszereknek az elvégzett mérési eredmények elemzésével megállapított jellemzőit a 6.1. táblázat foglalom össze.

Módszer	**Kerékszenzor segítségével meghatározott pillanatnyi sebesség**	**GPS szenzortól lekérdezhető pillanatnyi sebesség**	**GPS szenzor által mért távolság és idő alapján számolt pillanatnyi sebesség**	**Kerékszenzor és GPS adatokból kombinált sebesség**
Pontosság alacsony sebesség esetén	pontos	átlagos	alacsony	alacsony
Pontosság magas sebesség esetén	pontos	átlagos	alacsony	pontos
Frissítési gyakoriság alacsony sebesség esetén	ritka	átlagos	átlagos	gyakori
Frissítési gyakoriság magas sebesség esetén	gyakori	átlagos	átlagos	gyakori
Rendelkezésre állás	megfelelő	alacsony	alacsony	magas

6.1. táblázat. Vizsgált módszerek összehasonlítása

A legrosszabbul a *„GPS szenzor által mért távolság és idő alapján számolt pillanatnyi sebesség”* módszer teljesített, elsősorban alacsony pontossága miatt. A *„GPS szenzortól lekérdezhető pillanatnyi sebesség”* módszer a vizsgált módszerek között átlagosnak mondható. A *„Kerékszenzor és GPS adatokból kombinált sebesség”* módszer előnyeit alacsony sebességek esetén tapasztalható pontatlansága háttérbe szorítja. A vizsgált módszerek közül az eddigi mérések alapján a *„Kerékszenzor segítségével meghatározott pillanatnyi sebesség”* módszer tekinthető a legjobbnak, használhatóságát csak az alacsony sebességek esetén tapasztalható alacsony frissítési gyakoriság csökkenti minimális mértékben.

A mérési eredmények alapján látható, hogy az okostelefonok beépített GPS szenzora nem optimális megoldás a pillanatnyi sebesség pontos meghatározására. Ugyanakkor a GPS szenzor segítségével olyan adatok is rendelkezésre állnak, amiket egy kerékfordulat-érzékelő nem tud biztosítani. Ilyen adatok például a földrajzi pozíció- és magasság adatok. Tehát a GPS szenzor által biztosított adatok nagyszerűen felhasználhatóak okostelefonos kerékpár fedélzeti számítógép alkalmazás kiegészítő funkcióihoz, például a megtett útvonalat térképen megjelenítő alkalmazás részlethez. Azonban a kerékpár pillanatnyi sebességének pontos meghatározásához a GPS adatok önmagukban nem elegendőek.

7. Továbbfejlesztési lehetőségek

Dolgozatomat további kerékpáros sebesség-meghatározási lehetőségek felvetésével és a mérésekhez használt Android alapú kerékpáros fedélzeti számítógép alkalmazás továbbfejlesztési lehetőségeinek bemutatásával zárom.

7.1. További kerékpáros sebesség-meghatározási lehetőségek

A vizsgált kerékpáros sebesség-meghatározási módszerek további ötletek segítségével továbbfejleszthetőek, valamint a dolgozatban nem vizsgált lehetőségek is léteznek, az alábbiakban ezeket foglalom össze.

7.1.1. Több kerékmágnes használata

A mérések során a kerékfordulat-érzékelő segítségével történő sebesség-meghatározási módszer legnagyobb hátrányának az alacsony sebességek esetén tapasztalható alacsony frissítési gyakoriság bizonyult. Ezt a gyakoriságot például azzal lehet növelni, ha a kerékre több mágnest helyezünk el egyenletesen. Például 2 db kerékmágnes használata esetén a frissítési gyakoriság megduplázódna. Nem használható azonban tetszőlegesen sok kerékmágnes, mert ebben az esetben a kerékfordulat-érzékelőről érkező impulzusok között eltelt idő nagy sebességeknél minimálisra csökkenne, ami jelentősen megnehezítené az egyes impulzusok detektálását.

A módszer hátránya, hogy a kerékmágneseknek tökéletesen egyenletesen kell elhelyezkedniük a keréken, különben a pillanatnyi sebesség értéke ingadozik. Ezt azonban az átlagos kerékpárok küllőzési megoldása jelentősen megnehezíti. Az ötlet azonban nem megalapozatlan, megfelelő szoftveres támogatással használható lehet, ha másra nem, a kerékpár megállásának gyorsabb észlelésére biztosan.

7.1.2. Több szenzor által biztosított adat együttes felhasználása

Munkám során a kombinált sebesség meghatározásakor felhasználtam a kerékfordulat-érzékelő és a GPS adatokat is. Azonban a kombinált sebességet meghatározó algoritmus a két szenzor által biztosított adatok között tapasztalható eltérések miatt nem működött megfelelően. Az algoritmus továbbfejlesztésével ki lehetne dolgozni olyan megoldást, ami a két szenzor adatait megfelelően kombinálva képes megbízhatóan meghatározni a pillanatnyi sebességet alacsony sebességek esetén is.

A vizsgált szenzorok mellett a modern okostelefonok tartalmaznak még olyan szenzorokat, amik segíthetnek a pillanatnyi sebesség meghatározásában. Például a gyorsulásmérő szenzor segítségével elérhető a készülék pillanatnyi gyorsulása, amiből kiszámítható a készülék pillanatnyi sebessége, amennyiben képesek vagyunk az idő pontos mérésére. Ez utóbbi azonban Android operációs rendszer alatt nem egyszerű feladat. Továbbá problémát jelenthet az is, hogy a kerékpáros fedélzeti számítógépként használt mobil készüléknek a kerékpár kormányán célszerű elhelyezkednie, ott azonban a készülékre sok olyan erő hat, ami megnehezíti a kerékpár sebességének ezzel a módszerrel történő meghatározását.

7.2. Az Android alkalmazás továbbfejlesztési lehetőségei

A mérések elvégzéséhez készített Android alkalmazás továbbfejleszthető többfunkciós kerékpáros fedélzeti számítógép alkalmazássá. Az alkalmazás már jelenlegi állapotában is meghaladja a hagyományos kerékpár kilométeróráknál megszokott funkcionalitást, azonban még nem használja ki eléggé az okostelefonok által nyújtott lehetőségeket. Az alkalmazás bővíthető például a megtett útvonalat térképen megjelenítő funkcióval vagy a mért adatoknak a mobil készülék kijelzőjén grafikus formában történő megjelenítésével.

Hivatkozások

[1] Cyclocomputer: http://en.wikipedia.org/wiki/Cyclocomputer, 2014.10.16.

[2] Hagyományos kerékpár kilométeróra házilag: http://eet.etec.wwu.edu/gillets/project/docs/Cyclometer%20description%201%20pdf.pdf, 2014.10.16

[3] Braun Patrik János, *„Energiahasználat optimalizálási megoldások tervezése Android platformon”*, https://tdk.bme.hu/VIK/DownloadPaper/Energiahasznalat-optimalizalasi-megoldasok, 2014.10.18.

[4] „AllSport GPS FREE” Android alkalmazás: https://play.google.com/store/apps/details?id=com.trimble.allsportle, 2014.10.18.

[5] „SpeedView: GPS Speedometer” Android alkalmazás: https://play.google.com/store/apps/details?id=com.codesector.speedview.free, 2014.10.18.

[6] Mobile Action CS-20 Speed / Cadence Bike Sensor: http://global.mobileaction.com/product/product_i-gotU_CS20.jsp, 2014.10.18.

[7] Jeong-Jin Yeo, Yoon-Ho Lim, Mun-Ho Ryu and Yoon-Seok Yang, *„An Automatic Recognition of Bicycle Riding State by Using a Smartphone”*, http://onlinepresent.org/proceedings/vol6_2012/20.pdf, 2014.10.18.

[8] Samuli Hemminki, Petteri Nurmi, Sasu Tarkoma, *„Accelerometer-Based Transportation Mode Detection on Smartphones”*, http://www.cs.helsinki.fi/u/shemmink/Transportation/hemminki13transportation.pdf, 2014.10.18.

[9] Andong Zhan, Marcus Chang, Yin Chen, Andreas Terzis, *„Accurate Caloric Expenditure of Bicyclists using Cellphones”*, http://www.cs.jhu.edu/~andong/papers/sensys_108.pdf, 2014.10.18.

[10] Roberto Jos Gomes Silva, *„SocialSensing - Social sensing application for Smartphone”*, https://fenix.tecnico.ulisboa.pt/downloadFile/395145927077/ExtendedAbstract.pdf, 2014.10.18.

[11] Rafael V. Aroca, Aquiles F. Burlamaqui and Luiz M. G. Goncalves, *„Method for Reading Sensors and Controlling Actuators Using Audio Interfaces of Mobile Devices”*, http://www.mdpi.com/1424-8220/12/2/1572/htm, 2014.10.18.

[12] Global Positioning System: http://hu.wikipedia.org/wiki/Global_Positioning_System, 2014.10.18.

[13] Reed kapcsoló: http://en.wikipedia.org/wiki/Reed_switch, 2014.10.18.

[14] Hall szenzor: http://en.wikipedia.org/wiki/Hall_effect_sensor, 2014.10.18.

[15] Kilométeróra használati útmutató: http://sheldonbrown.com/cyclecomputers/atech-manual.jpg, 2014.10.18.

[16] Jack csatlakozó kép: http://colinbookman.com/content/images/2014/Mar/4poleminiJack.jpg, 2014.10.18.

[17] Reed kapcsoló kép: http://www.chicagosensor.com/images/HowItWorksReed.jpg, 2014.10.18.

[18] Service komponens: http://developer.android.com/guide/components/services.html, 2014.10.18.

[19] AudioRecord osztály: http://developer.android.com/reference/android/media/AudioRecord.html, 2014.10.19.

[20] Bhupinder S. Mongia, Vijay K. Madisetti, *„Reliable Real-Time Applications on Android OS”*, http://users.ece.gatech.edu/~vkm/Android_Real_Time.pdf, 2014.10.19.

[21] LocationManager osztály: http://developer.android.com/reference/android/location/LocationManager.html, 2014.10.19.

[22] Location osztály: http://developer.android.com/reference/android/location/Location.html, 2014.10.19.

[23] Fused Location Provider API: https://developer.android.com/reference/com/google/android/gms/location/FusedLocationProviderApi.html, 2014.10.19.

[24] Google Play Services: https://developer.android.com/google/play-services/index.html, 2014.10.19.

[25] ScheduledExecutorService osztály: http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ScheduledExecutorService.html, 2014.10.19.

[26] FreeMat program: http://freemat.sourceforge.net/, 2014.10.20.

[27] Accuracy and Precision: http://en.wikipedia.org/wiki/Accuracy_and_precision, 2014.10.20.

[28] Korrigált tapasztalati szórás: http://hu.wikipedia.org/wiki/Tapasztalati_sz%C3%B3r%C3%A1s, 2014.10.20.

[29] Standard deviation function: http://freemat.sourceforge.net/help/elementary_std.html, 2014.10.20.

[30] Accuracy and Precision ábra: http://en.wikipedia.org/wiki/File:Accuracy_and_precision.svg, 2014.10.20.

[31] Visual Paradigm modellező szoftver: http://www.visual-paradigm.com, 2014.10.21.