

Mobil eszköz alkalmazási lehetőségei fizikai kísérletekben OTDK dolgozat

Készítette:

Asztalos Gergő

Programtervező informatikus BSc

3. évfolyam

Témavezető:

Biró Csaba

Adjunktus

Tartalomjegyzék

1. Bevezetés	4
2. Tervezés	6
2.1. Specifikáció	7
3. A keretrendszer	8
3.1. Szerver oldal	8
3.2. TCP/IP kapcsolat	8
3.2.1. Adatok feldolgozása	9
3.2.2. Diagramok	10
3.2.3. Inga	11
3.2.4. Harmonikus rezgőmozgás	12
3.3. Kliens alkalmazás	13
3.4. Android Studio	16
3.5. Az alkalmazás	17
3.5.1. AsyncTask	17
3.5.2. Low Pass Filter	19
3.6. Szenzoradat mérés	20
3.6.1. Real Time adatátvitel	21
4. Fizikai kísérletek	22
4.1. Kalibráció	22
4.2. Exportálási lehetőségek	22
4.3. Egyenes vonalú, egyenletesen változó mozgás	23
4.4. Harmonikus rezgőmozgás	24
4.5. Fizikai inga	25
4.6. Körmozgás	26
5. További tervezek	28
6. Összefoglalás	29

Motiváció

„Hallom - elfelejtem.

Látom - emlékszem.

Csinálom - megértem.”

Confucius

Motivációm elsődlegesen általános és középiskolai tanulmányaimból gyökerezik. Az igazság az, hogy a természettudományos tárgyak (elsődlegesen a kémia és fizika) nem igazán nyerték el a tetszésem. Tanáraimról jó emlékeim vannak, bennem mégsem sikerült felkelteniük az érdeklődést. Távolodtam. Arra emlékszem, hogy meg kellett tanulni. Hogy miért? Azt akkoriban nem tudtam. Pedig kísérletezni jó, nagyon jó! Amikor középiskolába kerülttem, megtetszett az informatika és valamikor ilyenkor határoztam el, hogy ha még nem is tudom pontosan milyen, de informatikus lesz belőlem. A számítógép izgalmas, a programozás izgalmas, a programozás művészeti! A számítógépen megoldandó problémák felébresztették bennem a kísérletező kedvet és az érdeklődésem egyre inkább a természettudományos problémák (programozása, szemléltetése) felé irányult. Elkezdtem természettudományos cikkeket olvasni, és egyre több olyan alkalmazást készíteni, amelyeknek alapja valamilyen fizikai és/vagy matematikai modell. Innen már egyenes út vezetett ahhoz, hogy a dolgozatom témáját ebből a témakörből válasszak. Szerettem volna olyan alkalmazást fejleszteni, ami esetleg hasznos lehet a hozzám hasonló cipőben lévő, most általános és középiskolai éveit taposó diáktársaimnak.

1. fejezet

Bevezetés

A mai modern világban rohamléptekben nő az okostelefonok, táblagépek minden nap felhasználása. Egyre több azon alkalmazások száma, melyek segítik, esetleg szórakoztatják vagy épp tájékoztatják a felhasználót egy adott témaban. Szinte már mindenhez használjuk őket és kiaknázzuk az összes lehetőséget ahol újabb applikációval ismerkedhetünk meg. Tanuláshoz, munkához is használjuk ezeket, bár nem lenne elengedhetetlen számunkra. Sőt, annak ellenére, hogy az utolsó memóriát is lefoglaljuk ilyen programokkal az eszközeinkben, sok esetben meg sem fordul a fejünkben, hogy miből áll össze, milyen szoftveres vagy épp hardveres komponensek könnyítik meg az életünket. Én szerencsés vagyok, mert nem csupán eszembe jutott hanem választ is kaptam ezekre a "kérdésekre" az elmúlt három évben.

Amióta a főiskolára kerülttem és programozni tanulok, egyre inkább foglalkoztatott, hogy hogyan is működik egy okostelefon, hogyan működnak a gesztusok, a tájolások, milyen eszköz segítségével tudja a telefon, hogy éppen hol vagyok. Miért képes egy relatíve kis eszköz megmutatni azt amit a hétköznapokban sokkal hatalmasabb eszközök szoktak mérni. Hogy képes több eseményt egy azon eszköz kezelni... Később, tanulmányaim során megtudtam, hogy ezeket különböző szenzorok segítségével tudjuk elérni. Ezek a gondolatok megmozgatták a fantáziámat. Végül elkezdtem beleásni magam a téma: hogyan is lehetne ezeket még inkább beleszóni a minden napjainkba. Hogy tudnánk használni ezeket a szenzorokat olyan célokra, melyek különböznek az átlagostól, esetleg még inkább segítik a munkánkat. Mi lenne, ha ezeket nem csak szórakozás, de tanítás céljából is fel tudnánk használni, elősegíteni a diákok fejlődését egy adott téma körben. Ezen célok eléréséhez a telefon egy kiváló segédeszköz lehet.

Társadalmunk fejlődését alapvetően befolyásolja az, hogy milyen mértékben vagyunk képesek átadni a felhalmozódott tapasztalatokat a jövő nemzedékei számára. Az iskola az az intézmény, ahol ez szervezett keretek között céltudatos formát ölt. A természettudományos kísérletezés során a tanulóknak lehetőségük nyílik egy adott természetben lezajló jelenség, folyamat bemutatására, összefüggések feltárására, törvény-szerűségek megértésére. A kísérletezés, a próbálhatás az egyik legizgalmasabb emberi

tevékenység. Bemutatunk egy olyan oktatási segédeszközt, amely a kor kihívásainak megfelelve segítséget nyújthat, a természeti folyamatok meghatározó törvényszerűségeinek megismerésére, tanulmányozására. Az alkalmazás specifikációjának elkészítésekor kiemelt figyelmet fordítottunk arra, hogy az elkészült szoftver azon túl, hogy az egyes folyamatokat szemlélteti, lehetőséget biztosítson majd a tanulóknak, elméleti következtetések, hipotézisek felállítására, továbbá ellenőrzésére. Jelen alkalmazásunkkal elsősorban a 10-18 éves korosztályt céloztuk meg.

2. fejezet

Tervezés

Amikor egy projektről beszélünk, az első, és az egyik legfontosabb lépés az, hogy megfelelően megtervezzük a programunkat. Ezen folyamat során meg kell beszélnünk, hogy milyen lesz a program felépítése, struktúrája, dizájnja. Fontos ezeket még a tervezési fázisban megtenni, hisz egy programnál bármilyen alapvető funkciókat érintő változtatás utólagosan nehéz feladat. Célszerű a tervezési fázisban megbeszélteket feljegyezni valamilyen formában. Ilyenkor sokan a rajzoláshoz, íráshoz folyamodnak és ezzel időt, és energiát spórolnak maguknak. Ezt tettem én is! Az első pár megbeszélésen az ötleteleshöz semmi másra nem volt szükség csak papírra és ceruzára.

Azonban mielőtt hozzákezdtünk volna a tervezéshez, megvizsgáltuk, hogy ezen a területen milyen eredmények születtek már az elmúlt időszakban. A kezdetektől fogva egy olyan alkalmazást szerettünk volna készíteni, ami egyedi. Egy olyan a későbbiekbén könnyedén bővíthető valós idejű kommunikációt biztosító keretrendszer, amely segítségevel fizikai kísérletek széles skálája reprezentálható. Számos tanulmány született az elmúlt pár évben ezen a területen, melyekben azonban kizárolag egy-egy adott kísérlet bemutatására és a szenzorhibák elemzésére fektették a hangsúlyt. Az eredmények szemléltetése általában késleltetve, csak a mért adatok feldolgozása és korrigálása után történik meg. Természetesen találhatóak valós időben működő alkalmazások is, de ezek kizárolag csak a telefonon szemléltetik azt, ami valójában történik [1, 2, 3, 4, 5]. A célként megfogalmazott, valós időben működő, fizikai kísérletek szemléltetésére alkalmas keretrendszer nem találtunk. Folytathattuk tovább a tervezést!

Elsőként felépítettük a számunkra megfelelő struktúrát. Tudtuk, hogy nem csak egy alkalmazás lesz, hiszen főbb céljaink között szerepelt a számítógép és okostelefon közötti Real Time ¹ adatátviteli kapcsolat kialakítása. Megterveztük, hogy a telefonon és a számítógépen lévő programoknak milyen legyen a kinézete, milyen oldalak, ablakok kövessék egymást.

¹ Valós idejű adatfeldolgozás

Tervezési fázisban felmerülő egyéb a kommunikációra és szenzorokra vonatkozó kérdések:

- Milyen módon fog kommunikálni az a két eszköz?
- Milyen szenzorral dolgozzak?
- Hogyan vigyük át az adatot úgy, hogy megközelítőleg valós idejű legyen?
- Milyen eljárásokkal lehet majd a kiugró szenzorhibákat kezelni?

A kérdések megválaszolásához, számos technikai dokumentációt cikket kellett fel-dolgozni. Előfordult azonban az is, hogy néhány kérdésre csak akkor kaptunk választ, amikor már elkezdtük magát a programozást.

2.1. Specifikáció

Célunk egy olyan többrétegű kliens-szerver alapú alkalmazás fejlesztése, amely képes az okostelefonról érkező szenzoradatokat valós időben feldolgozni és megjeleníti.

Módszertani követelmények

Természetesen tudjuk azt, hogy ebben nagyon nagy a kockázat, ugyanis egy rosszul kivitelezett és megjelenített fizikai kísérlettel akár alá is aknázhatjuk az adott megértési folyamatot. Az előzetesen feldolgozott irodalmakból jól látható, hogy olykor jelentős mértékű szenzorhibákkal is meg kell majd várhatóan küzdeni. Ezért az alkalmazásunkból semmiképp nem maradhat ki majd az un. „fizika szertár” funkció. Ez tulajdonképpen nem jelent mást, minthogy minden egyes kísérlet elvégzése előtt egy kalibrációt kell majd elvégezni. A kalibráció lehet egy vagy több próba kísérlet, amely alatt a tanár ismerteti az adott kísérletet. A próbák kísérletek adatait el kell tárolnunk és valamilyen módon elemezni kell majd. Ezen un. referenciaértékeket a későbbiekben a valós idejű megjelenítésnél (kísérlet alatt) az esetleges szenzorhibák (kiugróan magas és alacsony értékek) elsimítására, elnyelésére lehet majd felhasználni. A grafikus felület tervezésekor kiemelt figyelmet kell majd arra fordítani, hogy a mért adatok (skalárok, vektorok) valóságnak megfelelő értékeket és irányokat reprezentálják.

3. fejezet

A keretrendszer

3.1. Szerver oldal

A tervezési fázis után elkészítettük a keretrendszer drótváz modelljét.

A drótváz modell elkészítése után következett egy rendkívül nehéz feladat. Meg kellett határozni, hogy milyen programozási környezeteket fogunk majd használni. Számos környezetet megvizsgáltunk (java, python, delphi), de végül a dotNet és a C# programozási nyelv mellett döntöttünk. A fejlesztői környezet tekintetében pedig a Visual Studio-ra esett a választás.

3.2. TCP/IP kapcsolat

A Transmission Control Protocol, azaz TCP, az internet egyik legfőbb részét, az úgymond "gerincét" alkotó TCP/IP protokollcsalád egyik fő protokollja. Ezen protokollcsalád két legfontosabb transzport protokollja a TCP és az UDP¹. A TCP egy kapcsolatorientált, megbízható protokoll. A kommunikáció megkezdése előtt ki kell építenünk a kapcsolatot, majd ezután megkezdhetjük az adatátvitelt. Amennyiben hiba történik, mint például elveszik egy csomag, vagy meghibásodik, esetleg már hibásan érkezik, a TCP saját maga újraindítást kér. Az átvitel úgy történik, hogy az adat, amit szeretnénk átküldeni egy úgynévezett byte-folyam, amit a TCP szétszeparál csomagokra és elküldi. A kapcsolat tehát full-duplex², továbbá rendelkezik egy olyan szinkronizációs mechanizmussal, ami megakadályozza, hogy az adó elárassza a vevőt. Emellett a TCP figyeli a kapcsolatot és megróbálja megtippelni a sávszélességét a torlódásokból, válaszidőből stb., amit később felhasznál az adatsebesség beállításakor.

Alkalmasunkban az okostelefon minősül az adónak, és a számítógép pedig a vevőnek. Értelemszerű, hiszen a szenzoradat a telefonról érkezik és azt kell továbbítanunk

¹ User Datagram Protocol

² Megengedi a kétirányú kapcsolatot, szimultán módon

a vevőnek. Ez úgy történik, hogy a telefon egy IP cím és egy Port szám segítségével kérést küld a szerveroldal felé a kapcsolódásra, amit a már várakozó szerver később fogad. Ezután meg is kezdődik az adatátvitel. Esetünkben, mivel az egész program a Real Time adatátvitelen alapszik, ezért a kommunikáció nyitott, mindaddig, míg azt "erőszakkal" be nem zárjuk.

3.2.1. Adatok feldolgozása

Az élő kapcsolat fenntartása annyit jelent, hogy a beérkező adatot azonnal feldolgozzuk és a célunknak megfelelően fel is használjuk, adatról-adatra. A hangsúly azon van, hogy megpróbáljuk abban a pillanatban feldolgozni szerver oldalon az adatot, amikor az előállítódik a kliens oldalon. Természetesen száz százalékosan ez nem teljesíthető, hiszen ez a folyamat elég erőforrás igényes.

A kapcsolat beérkezése után meg is kezdődik az adatok feldolgozása. Ez a feldolgozási folyamat természetesen attól függ, mit szeretnénk csinálni majd az adatunkkal.

```
int port = 8888;
label5.Invoke(new Action(() => label5.Text =
    GetLocalIPAddress()));
TcpListener myListener = new
    TcpListener(IPAddress.Parse(GetLocalIPAddress()), port);
myListener.Start();
label1.Invoke(new Action(() => label1.Text = "Varakozas
    kliensekre a " + port + " porton."));
TcpClient connectedClient = myListener.AcceptTcpClient();
label1.Invoke(new Action(() => label1.Text = "Kliens
    csatlakozott"));
```

A csatlakozást követően szükségünk van egy StreamReader-re, ahhoz, hogy a kapcsolódott eszköz által küldött információt meg is tudjuk jeleníteni, vizsgálni. Az adatok, azaz a 3 tengely értéke 1 adott sorként érkezik át a kapcsolat folyamán. Ha ezekkel az adatokkal dolgozni is szeretnénk (márpédig szeretnénk), szükségünk lesz arra, hogy azt az 1 sort, szétbontsuk 3, különálló értékre. Így megkapjuk az x , y és z tengelyek értékét.

–0,10534488 9,787497 – 1,091756

A szétbontásuk pedig a következőképp történik:

```
string s = r.ReadLine();
string[] str_array = s.Split(' ');
x = double.Parse(str_array[0],
    CultureInfo.InvariantCulture);
```

```

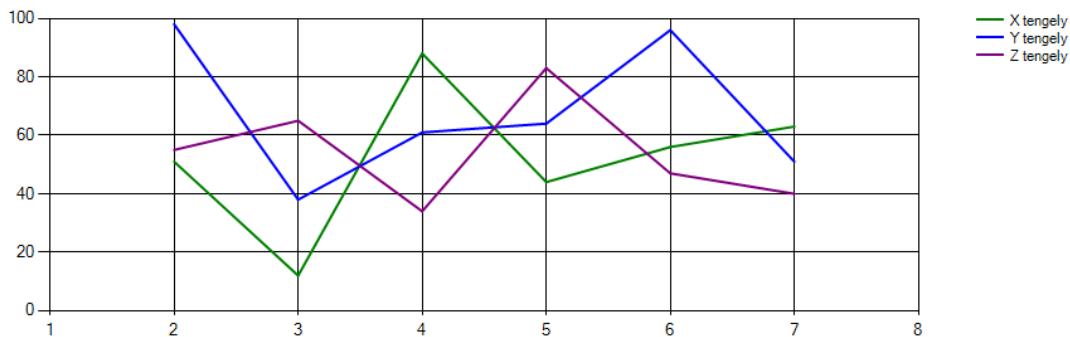
y = double.Parse(str_array[1],
    CultureInfo.InvariantCulture);
z = double.Parse(str_array[2],
    CultureInfo.InvariantCulture);

```

Ebben az esetben a(z) *s* változónk tárolja az átküldött sort, a(z) *str_array* pedig annak a szébtöntött változatát. Így meg is kaptuk a szövegként átküldött sorunkat, változóra szébtöntva és megkezdhetjük a feldolgozásukat, beillesztésüket a feladatunkba.

3.2.2. Diagramok

Most már megvannak az adataink, fel is tudjuk hát dolgozni őket. Első és legfontosabb dolgunk az, hogy létrehozzunk egy diagramot, ami segítségével meg tudjuk jeleníteni (tengelyenként) a telefonunk gyorsulását. Ez már önmagában véve is egy olyan szemléltetés, melyet használhatnánk a fizika tanórákon. Szerencsére van erre egy beépített Tool³, melynek értékeket átadva már el is készíti a megfelelő diagramot. Ezen Tool neve: Chart. Dolgunk nehezedik annyiban, hogy ennek a diagramnak előre definiálnunk kell bizonyos pontokat a megjelenítéshez, viszont bármikor tudunk hozzáadni új pontokat. Ezzel csak annyi a probléma, hogy az új pontokat beszúrja, a többi pont mellé, így viszont bizonyos idő után feltorlódnak az adatok és a vonaldiagramunk elveszti varázsát.



3.1. ábra. Gyorsulás grafikon

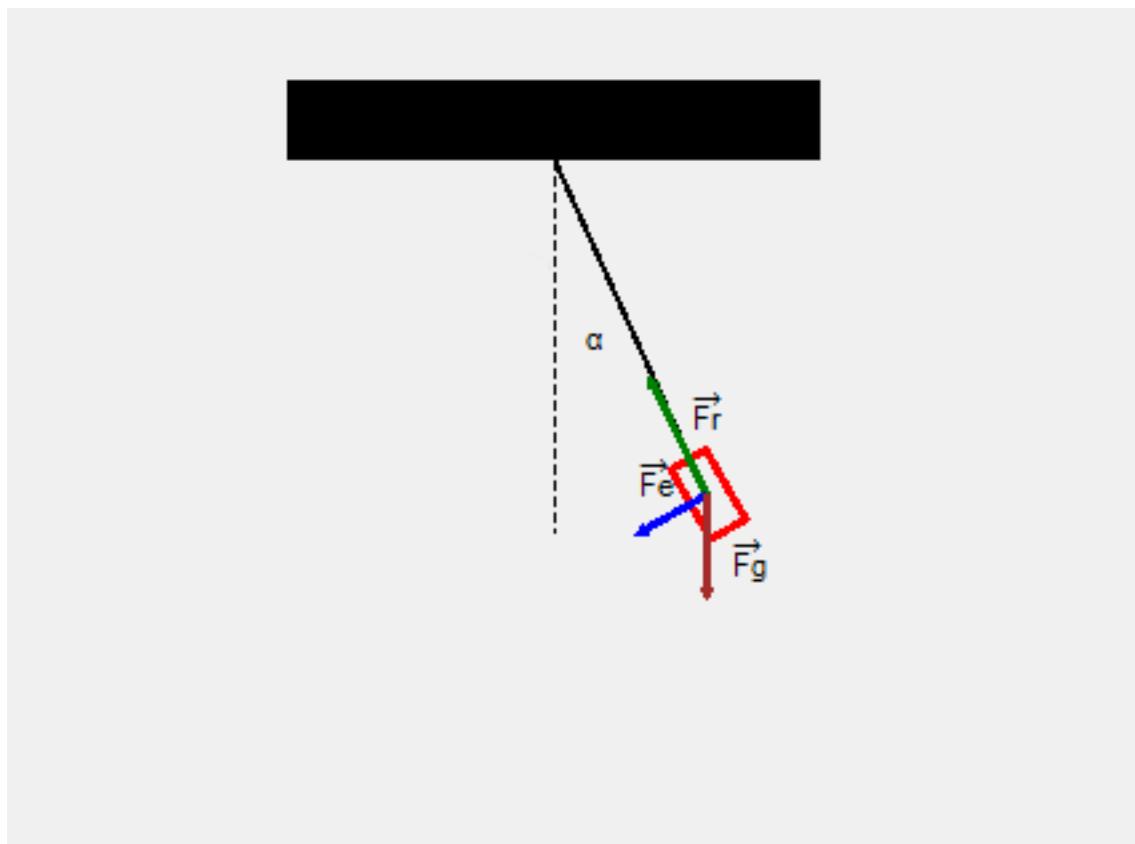
Erre beépített megoldásunk sajnos nincs, magunknak kell megírnunk a függvényünket úgy, hogy a bent lévő adatok egy része megmaradjon, viszont a régi adatokat kitöröljük. Erre egy olyan megoldást eszközöktem, hogy amennyiben a diagramunk pontjainak száma elér egy előre megadott mennyiséget, a pontokat elmentjük egy listába, a listát megfelezzük, az első részét töröljük a diagram összes elemével együtt, majd a lista második része lesz a diagramunkra rállesztve. Ezzel azt érjük el, hogy a diagram törlődik, az elemek fele elvész, a másik fele pedig újból beíródik, így a diagramunknál nem lesz torlódás, mégis folyamatosnak látszik a megjelenítés.

³Visual Studio-ban a felhelyezhető gombokat, mezőket egy ToolBox-ban találjuk

3.2.3. Inga

Sokszor elgondolkoltam azon, hogy ha már ennyi mindenre képesek a telefonjaink a mai világban, miért ne használnánk őket a Chatelésen és a fénykép készítésen kívül valami másra is, valami hasznosabbra esetleg. Természetesen a telefon legfőbb célja még mindig a telefonálás, de mi lenne, ha például egy inga lemodellezésére tudnánk használni, esetleg fizikai tanórákon?

Számtalan internetes fórumot és hasonló oldalt átnéztem, de sehol nem találtam olyan megoldást, amelyben a telefon valamelyik szenzorát úgy használják ki, hogy az valós időben monitorozva legyen esetleg egy számítógépen is, kirajzolva azon az egyes fizikai adatait az ingamozgásnak, vagy ha esetleg volt is, nem valós idejű. Elkezdtem gondolkodni, hogyan is lehetne megoldani, ezt a fajta monitorozást. Az első dolog ami eszembe jutott, hogy megvizsgáljam, milyen adatokat kapunk, egy-egy mérés során? Nos, igaz hogy a gyorsulásmérőt nem feltétlenül erre találták ki, de az adatok olyan biztatóak voltak, hogy nyugodtan lehetett velük tervezni. Ezután már csak az jött, hogy hogyan használjam fel ezeket az adatokat?



3.2. ábra. Inga

A megoldás az lett, hogy figyelembe véve az ablak méretét (magasság/szélesség), amire rajzolni szeretnénk (tehát megjeleníteni az ingánkat) és az adatokat összevetve, arra jutottam, hogy a telefon mért tengelyadatait, a képernyőre vetítve akár még meg-

felelő megoldást is kaphatok. A szenzoradatok viszont nem egész értékek, hanem valós számok, szemben a képernyő szélesség vagy magasságával, hiszen azok pixelszámok, amik minden esetben egészek. Át kellett tehát konvertálni a szenzoradatunkat egész számmá, kerekítve azt. Ennek eredménye pedig az lett, hogy a telefon inga-mozgása már lekövethető lett a képernyőn is.

```
float xPoint = Convert.ToSingle(Nx) + 500;
float yPoint = Convert.ToSingle(Ny) + 50;
// "Fonal"
g.DrawLine(pendPen, 515, 40, xPoint+15, yPoint);
// Thetajelzo vonal
g.DrawLine(new
Pen(Color.Red), 515, 150, ((float)Nx+515, (float)Ny+65));
// A telefont helyettesítő test
g.FillEllipse(new SolidBrush(Color.Red), ((float)Nx+500),
((float)Ny + 50), 30, 30);
```

Tehát lényegében annyi történik, hogy minden egyes szenzoradat mérésnél behelyettesítjük a telefon új pozícióját a rajzoló függvényünkbe. Ennek hatására egy folyamatosnak tűnő mozgást érünk el. Ha rezgőmozgásról beszélünk, két fajta rezgőmozgást különböztetünk meg:

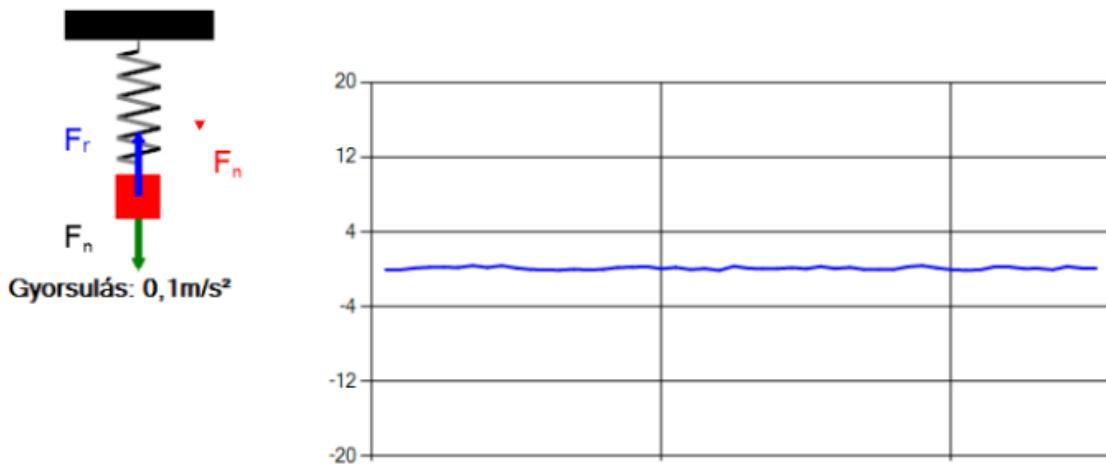
3.2.4. Harmonikus rezgőmozgás

A harmonikus rezgőmozgásról akkor beszélünk, ha egy testet az egyensúlyi helyzetéből kimozdítunk, és ennek köszönhetően a test két szélső helyzet között periodikus mozgást végez.

1. Csillapítatlan rezgőmozgás: Időben állandó a maximális kitérés (idealizált eset).
2. Csillapított rezgőmozgás: A maximális kitérés időben csökken.

Ha egy kicsit jobban bele szeretnénk mélyülni a fizikába, megvizsgálhatjuk az idő szinuszos függvényét is. Ebben az esetben, ha egy rugóra függgesztett test rezgését vizsgáljuk, és azonos időközönként megfigyeljük a kitérést, akkor azt tapasztaljuk, hogy a kitérés az idő szinuszos függvénye lesz.

A tervezési szakaszban, az ingán gondolkodva, eszembe juttatott még egy kísérletet, ami hasonló elven működik és szemléltetésnek se a legrosszabb dolog, ez pedig a Harmonikus Rezgőmozgás. A gyorsuló mozgást minden irányból, minden tengelyen tudjuk mérni. Rezgő mozgásnál azonban csak egy adott tengelyre van szükségünk, mégpedig az Y tengelyre. Ennél a kísérletnél biztos hogy mindig ezt a tengelyt fogjuk figyelni, hiszen biztos, hogy a telefon állított pozícióban lesz és az is biztos, hogy mi csak a le-fel



3.3. ábra. Harmonikus rezgőmozgás

mozgását fogjuk figyelembe venni. Tehát az olyan rezgőmozgást, ahol teljesül, hogy a kitérés az idő szinuszos függvénye harmonikus rezgőmozgásnak nevezzük.

```
g.DrawLine(515, 40, 515, Convert.ToSingle(Ny)+100);
```

A rajzolás pozicionálásához természetesen szükségünk van olyan konstans értékekre, amik csak azt adják meg, hogy nagyjából hova rajzoljunk, hiszen ha csak a szenzoradat-ra támaszkodnánk, akkor nem feltétlenül azzal az eredménnyel szembesülnénk, amire vártunk.

Szükségesnek találtam még azt is, hogy feltüntessem a kísérlethez megfelelő vektorokat. Esetünkben, ha egy kísérlet szemléltetéséről van szó, akkor a lehető legtöbb információt megjelenítjük, segítve ezzel a felhasználót, hogy jobban megértse miről is van szó az adott ábrán. A mozgásnál fellépő, illetve az testre ható erők szemléltetése épp olyan fontos, mint maga az inka kirajzolása.

Mellékeltem továbbá a rezgőmozgás ábrája mellé az egy tengelyre ható gyorsulás grafikont is, hisz mozgás közben ennek is fontos szerepe van, ha szemléltetésről beszélünk. Azonnal érthetőbb lesz a kísérletünk, bármely felhasználó számára, kisebb kiegészítésekkel.

3.3. Kliens alkalmazás

Mivel a szenzoradatok feldolgozását esetben fizikai kísérletek végrehajtásához használnám fel, figyelembe kellett vennem, hogy ne legyen túlságosan hardverigényes a programom. Ezért is választottam a gyorsulásmérőt, mint szenzort, mivel az manapság szinte minden Androidos eszközben megtalálható, így nem kell abba időt, pénzt és energiát fektetnünk, hogy megfelelő eszközt találunk, szinte minden eszköz megfelel számunkra. Az okostelefon tájolását tekintve nem kell nagy figyelmet fordítanom arra,

hogy milyen irányban is van a telefon, (amennyiben a programról beszélünk). Ellenben fontos, hogy hogy milyen kísérletet szeretnénk ábrázolni, megjeleníteni. Mégpedig azért fontos, mert a tengelyek, amiket használunk fix pozícióban vannak, viszont ha telefont elfordítjuk, akkor a megfigyelt tengelyünk horizontális/vertikális állásból ellenkező állásba kerül. Így az adatok is más hogyan fognak megjeleníteni.

A tervezésnél kialakítottam 2 Activity-t⁴, az egyik a kapcsolódásra szolgál, a másik pedig a szenzoradatokat jeleníti meg, továbbá lehetőséget ad a szenzor mérési gyorsaságának beállítására és a filterezés beállítására. Igaz, a megjelenített oldalunkon nem lesz sok adat feltüntetve, de a háttér munka, a számítások, pontosítások amiket a telefon végez, fontosabbak, mint az, hogy kinézetre milyen az alkalmazásunk.

Tengely:	Szenzor adat
X:	-0,10534488
Y:	9,787497
Z:	-1,091756

Szükségünk lesz továbbá arra, hogy a mért szenzoradatokat némely esetben pontossuk, filterezzük. Erre csak pár kísérletnél van szükség, viszont ez éppen elég indok ahhoz, hogy lehetőséget biztosítsunk a felhasználónak a választásra a tiszta szenzoradatok és a pontosítottak között.

Természetesen nem csak ezt az egy szenzort lehet kihasználni ilyen kísérletek végrehajtásához, szemléltetéséhez. Jó pár szenzor van még a telefonban mint például a Gyroscope, vagy akár a mikrofon, de még a kamerát is kihasználhatjuk valamilyen kísérlet bemutatásához, akár szabadesést szeretnénk szimulálni, vagy a hang terjedését kívánjuk szemléltetni.

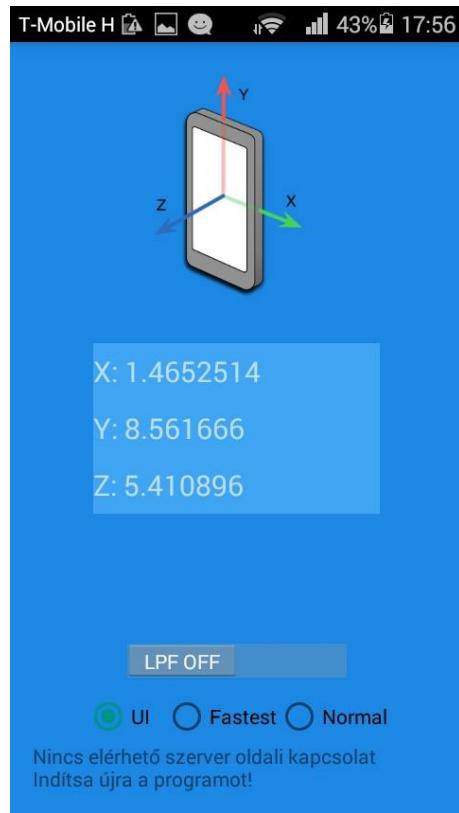
Az okostelefon szenzorokat is két külön csoportra bonthatjuk. Hardveres szenzor illetve Szoftveres szenzor. Ezek nevükön adódóan olyan szenzorok, melyek vagy be vannak építve a telefonba, vagy egy másik szenzort és egy kis szoftveres ügyeskedést felhasználva, szimuláljuk azt.

Szenzor:	Típus:
Accelerometer / gyorsulásmérő	Hardver
Környezeti hőmérséklet szenzor	Hardver
Gravitációs szenzor	Szoftver vagy Hardver
Gyroscope	Hardver
Fény szenzor	Hardver
Orientation / Tájolás	Szoftver

⁴ Activity, más néven Form, azaz eg yoldalt jelent.

Az alkalmazást telefonos részről már akkor is tudjuk használni, ha még nem csatlakoztunk rá a szerverre. Ekkor persze csak a szenzoradatokat tudjuk megtekinteni a telefonon, de már ez is egy felhasználható doleg, valamely szempontból. mindenek előtt ahhoz, hogy teljes mértékben használni tudjuk az alkalmazást, szükséges, hogy a klienssel rákapcsolódjunk a szerverre. Ilyenkor először is szerver oldalon le kell kérnünk a saját IP címünket, és meg kell adnunk egy Portot, amin keresztül majd csatlakozni szeretnénk. Ekkor csak annyi a dolgunk, hogy megnyomjuk a kapcsolódás gombot az alkalmazáson, és az automatikusan lekéri az IP címet. Ez után elő kell vennünk a kliens alkalmazást. A kliens layout-on szereplő EditText-ekbe be kell írnunk a megfelelő adatokat. Itt szintén egy IP címet és egy Port számot várunk el, mégpedig azokat, amiket a szerver oldal már lekért magának. Szükséges továbbá az, hogy a kliens és szerver is egyazon WiFi-re legyen rákapcsolódva.

Abban az esetben, ha kliens oldalon esetleg elgépeljük az IP címet vagy Port számot, az alkalmazás még továbbenged, hogy megtekinthessük a szenzoradatokat, viszont az alkalmazás figyelmeztet, hogy "Nincs kapcsolat", tehát vagy nem megfelelő helyre kapcsolódtunk, vagy nincs internetelérés.



3.4. ábra. Nincs elérhető kapcsolat

Ezt az Activity alján egy egyszerű Label-be írjuk ki, hogy figyelmeztessük a felhasználót. A kapcsolódás gombra kattintva (a kliens oldalon), az alkalmazás leellenőrzi hogy képes-e kapcsolódni, van-e olyan "elérési út", azaz IP cím amire kapcsolódni sze-

retnénk. Ezt a folyamatot a felhasználó nem látja, csak a későbbiek folyamán kap egy visszajelzést, hogy sikerült-e kapcsolódni vagy sem.

Abban az esetben, ha a szerver oldalon a kapcsolódást anélkül kíséreljük meg, hogy nincs elérhető internet kapcsolat, az alkalmazás erről is figyelmeztetést küld.

A kapcsolódást követően megkezdődik a kommunikáció. Ilyenkor a kliens elkezdi küldeni a szenzoradatokat a szerver felé. Ilyenkor már használhatóvá válnak a form-ra felhelyezett eszközök. Megkezdhetünk egy kísérletet. A kliens oldalon elérhető Switch segítségével az adatokat tudjuk megváltoztatni. Vannak esetek, amikor szükségünk van szűrésre, de vannak olyanok, amikor nem. Ez a lehetőség erre ad választási módöt. Ezen felül szerepel még három RadioButton. Ezek arra hivatottak hogy a szenzor gyorsaságát tudjuk beállítani. Ez azért jó, mert ha például az ingáról beszélünk, fontos, hogy minél több adatot kapunk meg, amiből dolgozni tud a szerver oldal. Minél kevesebb ez az adat, annál pontatlanabb eredményt kapunk.

A szerver oldalra visszatérve, a lehetőségeink tárháza elég nagy. A gyorsulás diagramon lehetőségünk van választani, hogy mely tengelyek gyorsulását szeretnénk látni. E mellett arra is van lehetőségünk, hogy a mobiltelefon aktuális pozícióját lementsük. Ez olyan kísérleteknél jöhet jól, ahol a kezdőállapottól függ a kísérlet végkimenetele.

A szenzoradatok mellett a klienstől érkezik egy negyedik adat is, ez pedig az idő. Az idő, olyan értelemben, hogy az első küldött adattól számítva mennyi idő telik el. Ez fontos lehet például abban az esetben ha sebességet, vagy hosszt szeretnénk kiszámítani.

Lehetőségünk van még továbbá arra is, hogy egy adott kísérlet során lementsük az adatokat. Ez a későbbi munkánkat tudja elősegíteni. Esetleg ha szeretnénk a későbbiekben felhasználni a kísérlet adatait, akkor ezek az adatok eltárolódnak egy TXT file-ba és a későbbiekben fel tudjuk azt használni.

3.4. Android Studio

A szerver oldalt követően a mobil alkalmazás fejlesztését is el kellett kezdenem. Ehhez én az Android Studio-t⁵ választottam. Itt alakítottam ki a kapcsolódáshoz szükséges képernyőt, a szenzoradat megjelenítő képernyőt és a fontosabb részeket. De még mielőtt belefolyánánk ezen fejlesztői környezet szerepét, tisztáznunk kell pár lényegesebb dolgot, ami nem feltétlenül egyértelmű mindenki számára.

Mi az hogy Android? Az Android napjaink egyik legsikeresebb mobil operációsrendszer. Bizonyos statisztikák szerint több száz millió Android-alapú készülék van már a piacon és ez a szám egyre csak nő és nő. Naponta több száz-ezer készüléket aktiválnak. A rendszernek régen külön verziója létezett telefonkészülékekre és táblagépekre, ám 2012-ben megjelent a 4.0-s verzió, amely már egyesíti ezt a két külön ágazó útvonalat, így manapság már minden eszközön egyazon verzió fut.

⁵ Fejlesztői eszköz amit a Google adott ki

A platform népszerűségét nagyon sok tényezőnek köszönheti, melyek között szerepel a kiemelkedően látványos felhasználói felület, a könnyű használhatóság, a nyíltság és a magas fokú kompatibilitás. Ám nem csak ezeknek köszönheti népszerűségét. Fontos szerepe van az operációs rendszert futtató készülékek hardverképességeinek, a gyors processzornak és a nagy méretű memóriának.

Az androidon egyrészt magát a mobil operációsrendszert értjük, másrészt pedig a futtató eszközt. A Google 2005-ben felvásárolta az Android Incorporated nevű vállalatot és ezután saját maga kezdte el a fejlesztést, így részben az Android a Google fejlesztése.

Az Android fejlesztését tekintve elég barátságos, ugyanis ingyenes és nyílt forrás-kódú az operációsrendszer. Hivatalosan az Open Handset Alliance konzorcium fejleszti és a Google a vezetője.

Számos fejlesztői környezet adódik, az ismeretlenektől kezdve az ismertekig, ám én mégis az Android Studio-t választottam. Ennek legfőbb oka az lehet, hogy a barátságos megjelenését, emberbarát fájl struktúráját és azt a tényt, hogy már fejlesztettem benne, tartottam szem előtt.

Fejlesztői környezet	Előnye	Hátránya
Android Studio	Ingyenes, nyílt forráskódok	Lassú buildelés ⁶
Eclipse	"helyetted írja a kódot"	Nagy memória felhasználás
Visual Studio	Erőforrás mérés	Nem Androidra fejlesztették

3.5. Az alkalmazás

Mint már említettem, a kliens alkalmazásom csak 2 képernyőt tartalmaz. Az elsődleges a kapcsolódáshoz szükséges adatokat kéri be (IP cím, Port), a második pedig a szenzor feldolgozást hajtja végre. Ez előbbi úgy történik, hogy felcsatlakozunk a WiFi-re a szerver alkalmazással, majd az általa kapott IP-címre rácsatlakozunk a kliens alkalmazással. Triviális tehát hogy elsőként a szerver alkalmazásnak kell futnia, csak utána tudunk csatlakozni a klienssel. Tehát miután beírtuk a megfelelő címet, melyre kapcsolódni szeretnénk, az alkalmazásunk továbbmegy a következő ablakra. A tényleges kapcsolódás itt történik, az előzőnél még csak elkértük a címet. Ez a lépés azért fontos, mert így a felhasználó anélkül is használhatja az alkalmazást, hogy lenne internet elérése. Ilyenkor az alkalmazás csak egy üzenetet küld, miszerint nem éri el a szerver oldalt, funkcióiban pedig közel ugyanazt nyújtja.

3.5.1. AsyncTask

Az AsyncTask egy olyan metódus, mely segítségével a háttérben futó időigényes feladatokat tudjuk megoldani. Ezt a programunk fő szálától elkülönítve tudjuk megtenni

a segítségével, ugyanis a fő szálat (User Interface) nem állíthatjuk le hosszú időre, ilyenkor ugyanis semmiféle felhasználói utasításra nem reagál. Éppen ezért, ezeket a feladatokat a háttérben, egy background szálon kell futtatnunk. Tehát lényegében a felhasználói szál és a háttérben dolgozó szál egyszerre fut. Természetesen előfordul, hogy ezek időnként kommunikálnak.

Éppen ezért, a kapcsolat kezelését én egy AsyncTask metódus segítségével oldottam meg. Kicsit nehezítette a feladatot az is, hogy miközben a kapcsolat él, megállás nélkül kommunikálunk a szerverrel, így aztán a felhasználói felületnek néha bele-bele kell nyúlnia a másik szálba.

```
protected Void doInBackground(Void... arg0) {
try {
    socket = new Socket(ipAddress, portN);
    ByteArrayOutputStream byteArrayOutputStream =
        new ByteArrayOutputStream(1024);
    byte[] buffer = new byte[1024];
    int bytesRead;
    InputStream inputStream = socket.getInputStream();
    while ((bytesRead = inputStream.read(buffer)) != -1){
        byteArrayOutputStream.write(buffer, 0, bytesRead);
        response += byteArrayOutputStream.toString("UTF-8");
    }
} catch (UnknownHostException e) {
    e.printStackTrace();
    response = "UnknownHostException: " + e.toString();
} catch (IOException e) {
    e.printStackTrace();
    response = "IOException: " + e.toString();
}finally{
    if(socket != null){
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
return null;
}
```

Az AsyncTask-nak is vannak beépített metódusai. A doInBackGround is egy ilyen metódus, de ezen felül is vannak hasonlóak. Ilyen például az onPostExecute, mely akkor hajtódik végre, ha például esetleg a kapcsolat megszakadt.

3.5.2. Low Pass Filter

A Low Pass Filter egy olyan filterezési megoldás, amely csak az olyan frekvenciájú jeleket engedi át, melyek kisebbek mint a bizonyos határfrekvenciák és csillapítja azokat a jeleket. Ezt más néven zajszűrésnek is nevezzük. Ez a csillapítás nagyban függ a filter használati módjától. A Low Pass Filter a High Pass Filter ellentettje (mint az egyértelmű is), továbbá létezik olyan hogy Band Pass Filter, ami a 2 filter kombinációja. Low Pass Filter-t a mi esetünkben opcionális választásként tüntettem fel. (Bizonyos esetben szükségünk van erre a pontosításra.)

De hogy miből is áll ez pontosan? Androidos okostelefonoknál beszélünk bizonyos $9.6m/s^2$ gravitációs erőről. Ennek konyhanyelven a lényege, hogy ez az erő minden "ránehézkedik" a telefonra, így, a telefon pozíciójától függően, szétoszlik a tengelyeken. Van olyan eset amikor ez jó nekünk. Ilyen például az, amikor a telefont szeretnénk bepozicionálni gyorsulásmérővel. Ha ráengedjük a filtert, akkor tisztán a telefon gyorsulását kapjuk vissza, ami annyit jelent, hogy bárholgy fordítom el a telefont, ha az nincs mozgásban, minden tengelyére 0-t kapunk. Viszont, ha ezt a filtert nem engedjük rá a szenzorunkra, a gravitáció szétoszlik a tengelyeken, a telefon pozíciójától függően, és már meg tudjuk állapítani, hogyan helyezkedik el a telefon.

Mi történik akkor, ha a pozíció is és a telefon mozgása is kell nekünk? Abban az esetben, ha a telefont például egy ingaként szeretnénk használni, de csak a gyorsulásmérő szenzort kihasználva, ez egy picit nehezít a dolgunkat. Ugyanis igaz, hogy a gravitációval könnyebb pozicionálnunk, viszont amint megmozdul a telefon, függően attól hogy milyen gyorsan mozdult el, már más adatot kapunk. Abban az esetben viszont, ha lassú mozgásról van szó, szépen lekövethető így is. Mi van akkor, ha a mozgás gyors, mégis szeretnénk lekövetni? Ebben az esetben kell elővennünk a Low Pass Filter-t.

```
alpha = 0.8F;  
//Megnezzük, mennyi a gravitacio az  
//adott tengelyen es elmentjuk azt  
gravity[0] = alpha * gravity[0] + (1 - alpha) *  
    event.values[0];  
gravity[1] = alpha * gravity[1] + (1 - alpha) *  
    event.values[1];  
gravity[2] = alpha * gravity[2] + (1 - alpha) *  
    event.values[2];  
//Levonjuk a gravitaciót az alap szenzoradatokból
```

```

linearAcceleration[0] = event.values[0] - gravity[0];
linearAcceleration[1] = event.values[1] - gravity[1];
linearAcceleration[2] = event.values[2] - gravity[2];

```

A fentebbi kód egy alapvető használata a Low Pass Filter-nek. A mi esetünkben ez kicsit módosul, hiszen nekünk babrálni kell a tengelyeken lévő adatokkal, ha megfelelően szeretnénk szemléltetni, mondjuk egy ingamozgást. Természetesen ha szeretnénk, babrálhatunk a filterrel önállóan is. Itt például konstansként vettük az α -t, de vannak olyan szituációk, amikor egy TimeStamp⁷-et alkalmaznak, annak a régi és új értékét, így meghatározva azt.

Abban az esetben, ha a kísérletünkhez nincs szükségünk erre a filterre, a felhasználó ki tudja azt kapcsolni az alkalmazásban egy Switch⁸ segítségével, ez pedig nem igényel semmiféle kapcsolatbontást vagy újra kalibrálást.

3.6. Szenzoradat mérés

Android alkalmazásunkban a szenzoradatok mérése egy ciklikus, beépített függvény segítségével történik. mindenek előtt be kell állítanunk az alkalmazás vagy Activity elindulásakor, hogy milyen szenzort szeretnénk használni adott esetben. Ehhez egy Listenerben be kell állítanunk hogy milyen szenzortípust szeretnénk használni. A mi esetünkben ez a: Sensor.TYPE_ACCELEROMETER paraméterrel érhető el. E mellé a Listenerünkben meg kell még adnunk azt is, hogy mekkora legyen a szenzor úgymond "gyorsasága". Gyorsaság alatt itt azt értjük, hogy a szennorunk milyen időközzel mérjen adatokat.

Sensor Delay	Jelentése
FAATEST	A lehető leggyorsabban megkapjuk a szenzor adatot
GAME	Játékokhoz legalkalmasabb rátá
NORMAL	Alapértelmezett, tájoláshoz a legalkalmasabb (képernyő)
UI	A felhasználói interfészhez legalkalmasabb

Ezeket a felhasználó, saját kényelmének megfelelően manuálisan is tudja állítani időközben, attól függően, milyen sebességgel szeretné mérni az adatokat. Miután a szenzoradatokat lekértük, egy SensorEvent típusú tömbből tudjuk azokat kinyerni mégpedig úgy, hogy a tömb elemei lesznek az egyes tengelyek értékei.

X tengely	<i>SensorEvent.values[0]</i>
Y tengely	<i>SensorEvent.values[1]</i>
Z tengely	<i>SensorEvent.values[2]</i>

⁷ Időbényeg, milyen időpillanatban mértük adatot, esetünkben

⁸ 2 állású kapcsoló

Ezek után már meg is kaptuk a gyönyörű, nyers adatainkat. Innentől kezdve már csak rajtunk áll, mit csinálunk vele.

3.6.1. Real Time adatátvitel

A RealTime adatátvitel lényege tehát az lenne a mi esetünkben, hogy abban a pillanatban, hogy lemértük az adatot, már küldjük is tovább. Ehhez természetesen szükségünk van azokra a lépésekre, amelyeket az előbbiekben megtettünk, tehát megszerezni a szenzoradatot, megnyitni a kapcsolatot a szerver felé, és felkészülni az adatátvitelre. Esetben az, hogy élő legyen a kapcsolat, úgy oldódik meg, hogy ki-használom a szenzor mérés adta lehetőséget, vagyis azt, hogy a beépített függvényünk ciklikusan működik. Röviden: bármilyen adatváltozásnál újra lefut a függvény, természetesen ezred pontossággal. A megoldásom tehát az lett, hogy ebben a függvényben nem csak az adatmentést, pontosítást, filterezést végezzük, hanem egyben a szerverre való "írást" is. Tehát mi helyst lemérünk egy adatot a gyorsulásmérőből, PrintWriter⁹ segítségével már küldjük is a szerver felé, ahol már úgy dolgozzuk fel ahogy szeretnénk.

A PrintWriter-el való átküldés azonban nem olyan egyszerű, mint gondolnánk. Ugyanis az a probléma, hogy egyszerre kell elküldeniünk mind a három adatot, mi helyst az lemérésre került, ellenkező esetben nem úgy fog alakulni az adatok érkezése mint vártuk, ugyanis ha minden egyes függvénybe lépéskor szeretnénk külön az X-et, Y-t és Z-t elküldeni, akkor lehetséges hogy még az előző adat megy át, vagy éppen a következő. Erre egy megoldás, hogy egy nagy darab String-ként, össze koncatenálva¹⁰ küldjük át a tengelyadatokat.

```
printWriter.println(linearAcceleration[0] + " " +
    linearAcceleration[1] + " "+ linearAcceleration[2]);
```

Ebben az esetben, az adatok szerverre való megérkezésekor hasonló lesz a dolgunk, mint akkor, amikor az eszközön mért adatot szeretnénk kiszedni. Ezek után az eszközünknek már nincs más dolga, mint ismételgetni önmagát, addig amíg a kapcsolat meg nem szűnik, az alkalmazást be nem zárjuk, vagy valami hiba nem következik.

⁹ Az általunk használt kommunikációhoz szükséges eszköz

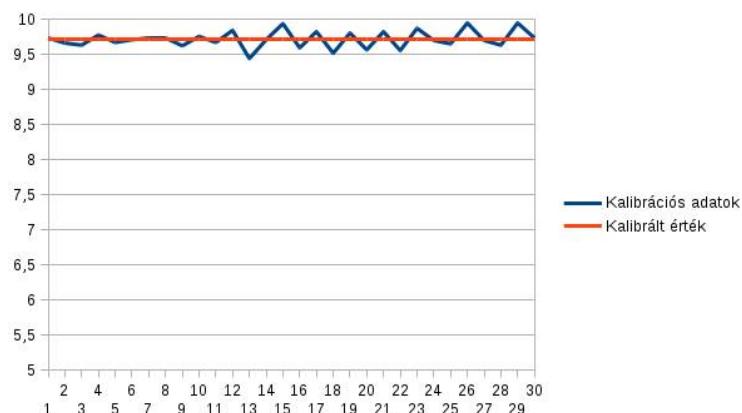
¹⁰ Összefűzve

4. fejezet

Fizikai kísérletek

4.1. Kalibráció

A kalibrációs modul az egyik legfontosabb komponens. Az alkalmazás jelenlegi verziójában minden egyes kísérlet előtt kalibrálni kell az eszközöket. A kalibrálást nem kötelező közvetlenül a kísérletezés előtt elvégezni, az alkalmazás képes a kalibrációs adatok mentésére. A kalibrálási eljárás jelenleg mozgástípusonként különbözőképpen történik, de általánosan elmondható, hogy a szemléltetés előtt legalább kétszer kell a szemléltetendő kísérletet a kalibrációs folyamat részeként elvégezni. A két kalibráció által visszaadatott értékeket egyesével és a két vagy több kalibrálási mérést összességében is képes elemezi a szoftver.



4.1. ábra. Kalibráció

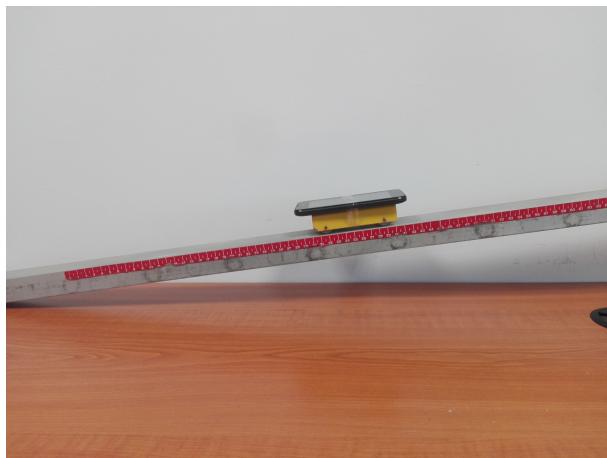
4.2. Exportálási lehetőségek

Az alkalmazás lehetőséget biztosít a kliensről érkező adatok mentésére. Az adatok menthetőek txt, csv, xls ésxlsx fájlformátumokba. Ezzel lehetőség adódik egy külső alkalmazás (pl. MS excel) által a mért adatok további részletes analizálására.

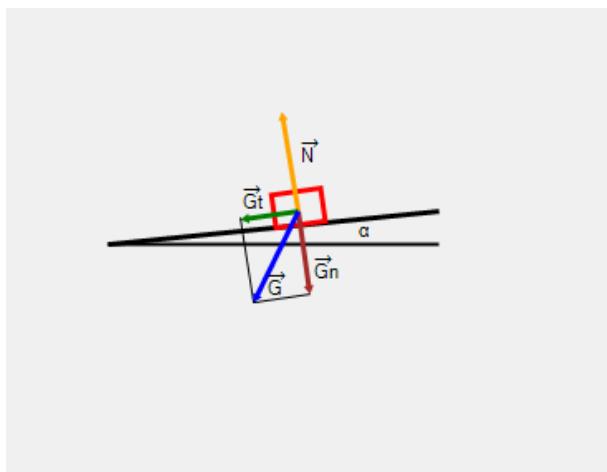
4.3. Egyenes vonalú, egyenletesen változó mozgás

Ennél a mozgásnál az alábbi értékek és összefüggések vizsgálatára van lehetőség.

- lejtő dőlésszöge
- pillanatnyi sebesség és gyorsulás
- megtett út
- eltelt idő
- út-idő függvény
- sebesség-idő függvény
- gyorsulás-idő függvény



4.2. ábra. Egyenes vonalú, egyenletesen változó mozgás



4.3. ábra. Egyenes vonalú, egyenletesen változó mozgás - futás közben

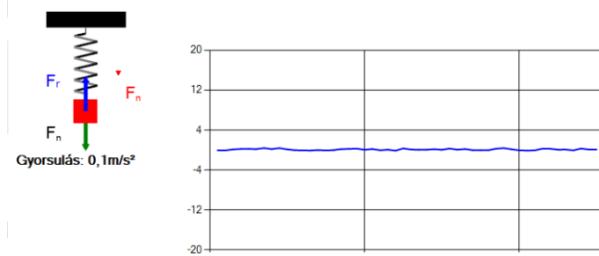
4.4. Harmonikus rezgőmozgás

Ennél a mozgásnál az alábbi értékek és összefüggések vizsgálatára van lehetőség.

- periódusidő
- amplitúdó
- rezgésszám
- frekvencia
- körfrekvencia
- kitérés-idő függvény
- sebesség-idő függvény
- gyorsulás-idő függvény
- csillapítási tényező
- rúgóállandó statikus és dinamikus meghatározása



4.4. ábra. Harmonikus rezgőmozgás



4.5. ábra. Harmonikus rezgőmozgás-futás közben

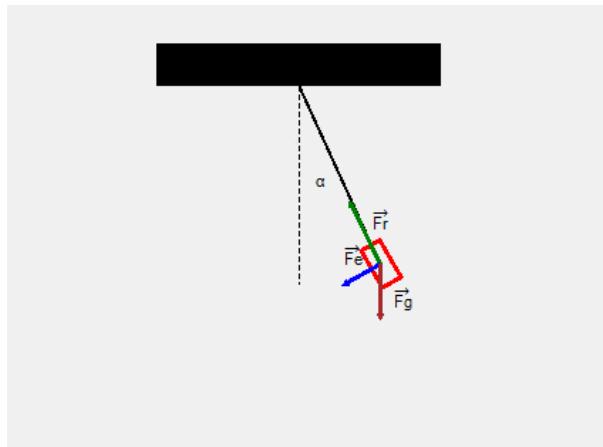
4.5. Fizikai inga

Ennél a mozgásnál az alábbi értékek és összefüggések vizsgálatára van lehetőség.

- lengésidő



4.6. ábra. Fizikai inga



4.7. ábra. Fizikai inga-futás közben

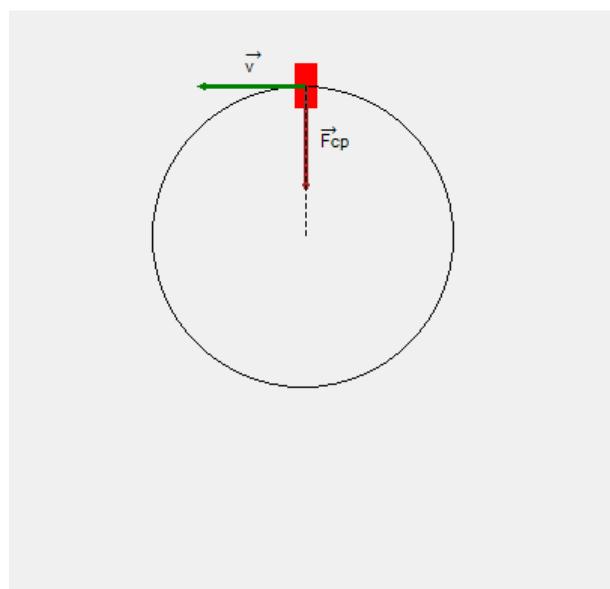
4.6. Körmozgás

Ennél a mozgásnál az alábbi értékek és összefüggések vizsgálatára van lehetőség.

- periódusidő
- szögsebesség
- kerületi sebesség
- frekvencia, fordulatszám
- centripetális gyorsulás



4.8. ábra. Körmozgás



4.9. ábra. Körmozgás

5. fejezet

További tervezek

További tervezek között szerepel az alkalmazás további kísérletekkel történő bővítése. Felmerül a kérdés, hogy miért jobb ezt az alkalmazást és a telefont használni segédeszközöként, mint pedig egy sima ingával tanítani továbbra is a hallgatókat, tanulókat? A válasz, amely bennem megfogalmazódott az, hogy amíg egy egyszerű kísérleti eszköz bemutatásával csak esetleg a mozgást tudjuk szemléltetni, addig az alkalmazással ugyan ezt végrehajtva olyan adatokhoz, információkhoz juthatunk, amelyekkel könnyebbé tehetjük a tanulást, tanítást. Jelenleg a számítógépen lévő alkalmazás Windows Form Application-ben íródott, ezt a jövőben szeretném lecserélni, még pedig Windows Presentation Foundation-re. Erre több szempontból is szükség lehet. Többek közt azért, mert igaz, hogy jelenleg még gyerekcipőben jár a projekt és a már megvalósított kísérleteket igen jól szemlélteti, de nincs meg animálva. Mivel a form-ra rajzolunk, minden egyes beérkezett adatnál, szükségünk van arra hogy mielőtt kirajzoljuk az új pozíciót, le is töröljük azt. Ennek az eredménye pedig az lesz, hogy enyhén "villogni" fog az ábránk. Ezt WPF¹-ben ki tudjuk küszöbölni, hiszen ott úgy rajzolunk a felületre, hogy animáljuk azt, így okkal szebb prezentációt kapunk a kísérletre, mint az elődjénél.

¹ Windows Presentation Foundation

6. fejezet

Összefoglalás

Befejezésképpen összegezve a korábban leírtakat. Ehhez elevenítsük fel, az eredetileg megfogalmazott célokot.

„...olyan Android alapú kliens applikáció, amely kapcsolatot tart egy szerveroldali alkalmazással. Ezeket a szenzoradatokat fizikai kísérletek megvalósítására, szemléltetésére használhatjuk fel, így a telefon egyfajta oktatási segédeszközé válik.”

Ennek megfelelően a Visual Studio segítségével elkészítettük a szerveroldalt, amely tartja a kapcsolatot a mobil klienssel. Személy szerint elégedett vagyok és sok tapasztalatot szereztem ezen fejlesztői környezet használatával, mivel könnyedén el tudtam készíteni a szenzoradat feldolgozó rendszert.

A szenzoradatok megértéséhez és azok megjelenítéséhez sokat kellett kutatni, és új dolgokat megtanulni.

Azon kívül, hogy a kész alkalmazás már használható arra a célra, amire ki lett találva, el kellett gondolkodni a további fejlesztési lehetőségen is. Ez a téma, amit az első oldal óta boncolgattam számtalan irányból megközelíthető a továbbfejlesztést tekintve.

Az első és legfontosabb dolog, ami a bővítést említve eszembe jut, az a fizikai kísérletekre vonatkozik. A keretrendszer "rugalmasságának" köszönhetően ezen kísérletek megvalósítása csak a fantáziánkon múlik. Lényegében ha a telefon egy szimpla segédeszköznek tekintenénk, mint (a példa kedvéért) akár egy labdát, akkor kapcsolatot létesítve le tudnánk szűrni az adatokat, amelyek a kísérlet során történnék anélkül, hogy a látottak alapján megtippelnénk azokat.

Megközelíthetjük a fejlesztés ezen szakaszát szenzor oldalról is. Jelenleg az applikáció egyetlen szenzort használ, a gyorsulásmérőt. Mint már mondtam, ezt azért választottam, mert ez a szenzor található meg a telefonokban a leggyakrabban. Másrészt viszont, van még pár szenzor, amellyel más irányból is megközelíthetnénk ezeket a

kísérletezéseket. Ilyen például a fényszenzor is. Tegyük fel, hogy valaki fénytani kísérleteket szeretne végrehajtani a telefon segítségével. Erre is van alkalmas szenzorunk, amellyel ezt akár meg is valósíthatjuk.

Végül de nem utolsó sorban megköszönöm a témavezető tanáromnak a bátorítást és a támogatást. Köszönöm mindenkinek, akik segítettek, ha kérdésem volt.

Irodalomjegyzék

- [1] J. A. SANS, F. J. MANJÓN, V. CUENCA-GOTOR, M. H. GIMÉNEZ-VALENTÍN, I. SALINAS, J. J. BARREIRO, J. A. MONSORIU, AND J. A. GOMEZ-TEJEDOR SMARTPHONE: A NEW DEVICE FOR TEACHING PHYSICS 1ST INTERNATIONAL CONFERENCE ON HIGHER EDUCATION ADVANCES, HEAD'15 UNIVERSITAT POLITÈCNICA DE VALÈNCIA, VALÈNCIA, 2015 DOI: <http://ocs.editorial.upv.es/index.php/HEAD/HEAD15/paper/viewFile/332/211>
- [2] SMARTPHONES IN THE CLASSROOM 2: TEACHING PHYSICS
<https://opensignal.com/blog/2015/05/05/smartphones-in-the-classroom--2-teaching-physics/>
- [3] PHYSICS EDUCATION USING A SMARTPHONE ACCELEROMETER RANDALL D. PETERS PHYSICS DEPARTMENT MERCER UNIVERSITY, MACON, GA
<http://arxiv.org/pdf/1012.3416.pdf>
- [4] L. MARTINEZ AND P. GARAIZAR LEARNING PHYSICS DOWN A SLIDE: A SET OF EXPERIMENTS TO MEASURE REALITY THROUGH SMARTPHONE SENSORS DEUSTO INSTITUTE OF TECHNOLOGY (DEUSTOTECH), UNIVERSITY OF DEUSTO, BILBAO, SPAIN.
<http://paginaspersonales.deusto.es/garaizar/papers/EDUCON2014-LM--PG.pdf>
- [5] JUAN C. CASTRO-PALACIO , LUISBERIS VELAZQUEZ , JOSÉ A. GÓMEZ-TEJEDOR , FRANCISCO J. MANJÓN , JUAN A. MONSORIU USING A SMARTPHONE ACCELERATION SENSOR TO STUDY UNIFORM AND UNIFORMLY ACCELERATED CIRCULAR MOTIONS REVISTA BRASILEIRA DE ENSINO DE FÍSICA, V. 36, N. 2, 2315 (2014)
<http://ocs.editorial.upv.es/index.php/HEAD/HEAD15/paper/view/332/211>
- [6] GYORSULÁSMÉRŐ:
<https://hu.wikipedia.org/wiki/Gyorsulás>
- [7] GYORSULÁSMÉRŐK A FIZIKÁBAN
<http://www.mogi.bme.hu/TAMOP/mikromechanika/math-ch05.html#ch-V.2>

- [8] HARMONIKUS REZGŐMOZGÁS
http://ecseri.puskas.hu/oktseged/mechanika/harmonikus_rezgomozgas.pdf
- [9] SZENZOROK
http://users.nik.uni-obuda.hu/malk/android/ea_2011_tavasz/08_-_Szenzorok.pdf
- [10] ASYNCTASK
<http://developer.android.com/reference/android/os/AsyncTask.html>