Tudományos Diákköri Konferencia - Dolgozat

Valós idejű automatizált mérésfeldolgozás

Moka Dániel

Budapesti Műszaki és Gazdaságtudományi Egyetem
Építőmérnöki Kar, Általános- és Felsőgeodézia Tanszék
Építőipari geodéziai szakirány

Tartalomjegyzék

1. Bevezetés

A 21. században az építőmérnöki tevékenység igen sokrétűvé vált, sokkal összetettebb és bonyolultabb szerkezeteket létesítenek nap, mint nap, melyek építésénél és üzemeltetésénél előforduló monitoring vizsgálat komolyabb feladat elé állítja szakembereket. Az ilyen komplikált szerkezetek megkövetelik, hogy a hagyományos technológiák helyett, modernebb, esetenként újabb módszereket alkalmazzanak az egyes építésirányítási és mozgásvizsgálati feladatokhoz. A mérnöki szakmában az elmúlt 10-15 év alatt bekövetkezett paradigmaváltás megköveteli, hogy az egyes építőmérnöki szerkezetekkel kapcsolatos mérések feldolgozása valós időben történjen. Ennek a megvalósítására alkalmaznak automatizált mérésfeldolgozó mozgásvizsgálati rendszereket, melyek hatékonyságára számos hazai és külföldi példa szolgál szerte a világon. Ezen rendszereknek köszönhetően a vagyon és az ember-élet megóvása sokkal nagyobb mértékben biztosított, mint korábban, így kialakításuknak a jelentősége és fontossága az egyes mérnöki beruházásoknál szinte már kötelező jellegűvé vált.

Az különböző mérésfeldolgozó rendszerek mögött „megbújó” rendszerkezelő felületek kialakítására számos módszer adott, azonban a rohamosan fejlődő információs technológia világában az egyes alkalmazások és szoftverek kialakítása körül ún. „szoftverkrízis” alakult ki, miszerint a hagyományos szekvenciális programozási módszerek már nem képesek az igényeknek megfelelő, minőségi alkalmazás illetve szoftver előállítására. Ennek következtében alakult ki az objektum-orientált programozás (OOP - object-oriented programming), melynek segítségével a valós világ modellezését lehetővé tevő absztrakciós folyamat sokkal egzaktabb módon véghez vihető. Az OOP programozás középpontjában nem az egyes programozási műveletek megalkotása szerepel, hanem az egymástól függő, egymással kapcsolatban álló programegységek hierarchiájának a megtervezése áll. Első főbb alapelve az egységbezárás (encapsulation), melynek keretén belül az egyes adatokat és a hozzájuk tartozó eljárásokat egyetlen egységben, az objektum-osztályban kezeljük, melyben a valós világgal az egyes osztálymetódusok kommunikálnak. Az OOP másik főbb módszertani „ereje” az öröklésben (interhitance) rejlik, miszerint egy adott objektum-osztály továbbfejlesztése rugalmasan megoldható. Ennek során a származtatott gyerek-osztály örökli az ősétől azok tulajdonságait és eljárásait, valamint ezek mellett újabb adattagokkal és metódusokkal lehet bővíteni az objektumot, illetve az örökölt metódusok felülbírálhatók. És végül, de nem utolsó sorban az objektumorientált nyelvek harmadik alappillére a többértelműség (polymorphism), mely kimondja, hogy egy függvényt a neve és a paraméterei azonosítják. A több azonos nevű függvény közül a rendszer a híváskori paraméterek alapján választja ki a végrehajtandót.

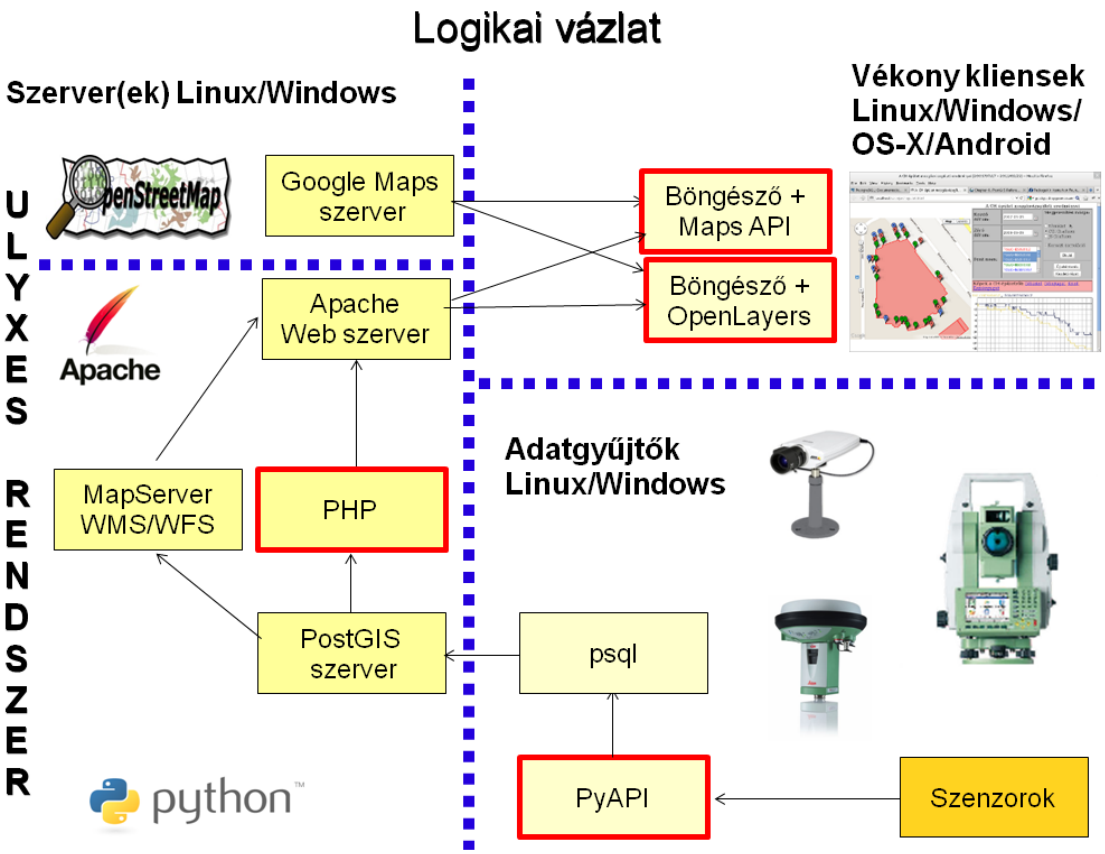
A TDK dolgozatom célja egy már meglévő automatizált mozgásvizsgálati rendszer szenzor-vezérlő alkalmazás programozó felületének (API - Application Programming Interface) logikai modelljének az újratervezése és megvalósítása objektum-orientált eszközök segítségével. A rendszer által kezelt szenzorok köre bővítésre került, kezelésük rugalmasabbá vált az új logikai modell megvalósításán keresztül. A megújított rendszert az Hárosi M0-s Duna híd próbaterhelésén tesztelésre került, melynek eredményei a dolgozat végén ismertetésre kerülnek.

2. Az Ulyxes rendszer

2.1. A rendszer jellemzése

Az Ulyxes (Odüsszeusz) nevű projekt egy nyílt forráskódú mozgásvizsgálati rendszer, melynek a szerzője Dr. Siki Zoltán. A projekt elsődleges célja, hogy egy olyan keretrendszert hozzunk létre, melynek a használatával képesek leszünk különböző típusú és gyártójú geodéziai szenzorok számítógépről történő vezérlésére, valamint a mért adatok interneten történő publikálására. A rendszer kialakításához számos nyílt forráskódú projektet használtak fel, melyek a szerves részeit képezik az Ulyxes-nek. A dolgozatomban bemutatott fejlesztés eredményeként a Python objektum-orientált programnyelv ezek egyikévé vált, melynek köszönhetően a szenzorok vezérélését lehetővé tevő alkalmazás programozói felület kialakítása és fejlesztése válik lehetővé és kiválthatja a TclAPI-t.

A másik nyílt forráskódú alkalmazás a PostgreSQL/PostGIS, mely hozzájárul a különböző adatok strukturált tárolásához, valamint az adatbázis működéséhez. A PostGIS a PostgreSQL egy bővítménye, melynek segítségével térinformatikai adatokat tárolhatunk az adatbázisban. A PostGIS a rendszeren belül a különböző szenzorokból származó adatok tárolására szolgál. Az interneten történő térképi publikálást a MapServer nyílt forráskódú, platform független alkalmazás teszi lehetővé, mely az egyes térképeket dinamikus térbeli adatok alapján képes előállítani. A MapServer egy igen sokoldalú alkalmazás, mely szinte az összes raszter, vektor és adatbázis formátumot támogatja, ezzel számos lehetőséget kínálva a felhasználó számára. A különféle adatbázis lekérdezések PHP szkriptek segítségével végzi a rendszer, a kiszolgáló oldalon pedig egy JavaScript program dolgozza fel a felhasználó utasításait. A 2.1.-es ábrán látható a rendszer három meghatározó része, melyeknél saját programokkal bővítettük ki a kész nyílt forráskódú szoftvereket.

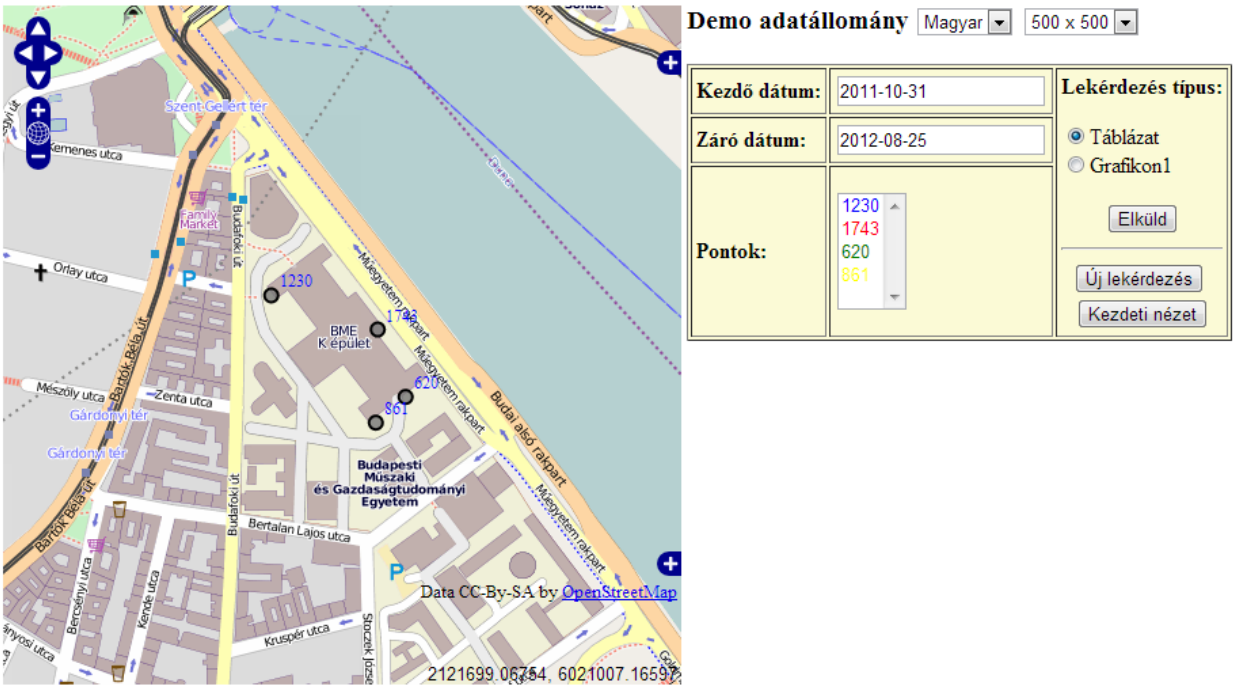


2-1 Ulyxes rendszer - logikai vázlat

2.2. A rendszer részei

2.2.1. Megjelenítést lehetővé tevő felület

Az Ulyxes rendszer első részét a megjelenítő felület képzi, melynek használatával a mérési eredményeinket és az ehhez kapcsolódó elemzéseinket publikálhatjuk táblázatok és grafikonok formájában az ezekhez tartozó térinformatikai megjelenítéssel együtt. A felület működtetéséhez a felhasználónak szüksége van internetkapcsolatra, valamint egy böngészőre is, mely támogatja a JavaScript-ek futtatását. Operációs rendszer szempontjából platform független az alkalmazás. A térképi megjelenítéshez használt térinformatikai rendszer hátterét az Open Street Map (OSM) szolgáltatja. Az OSM egy közösségi adatokon alapuló, bárki számára szabadon felhasználható utcatérkép, mely az interneten megtalálható. Az eredmények publikálásához, grafikonok megjelenítéséhez szükséges adatok tárolását viszont egy web szerver teszi lehetővé, melynek pontos részletezése a következő alfejezetben található.



2-2 Megjelenítést lehetővé tevő felület OSM használatával

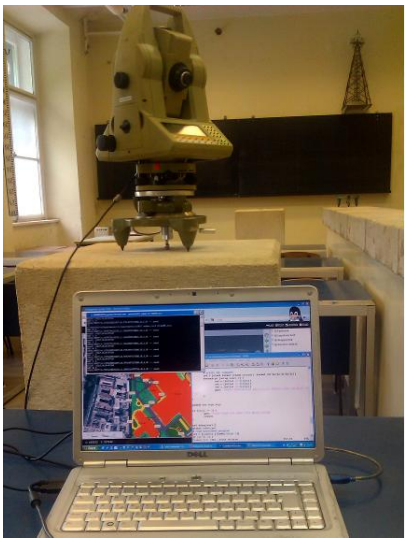
2.2.2. Szerverek

A rendszer összetett működéséhez számos nyílt forráskódú projekt illetve szerver hozzájárul. Annak érdekében, hogy különböző szenzorokból származó adatokat egy adatbázisba kerüljenek, a PostsgreSQL adatbázis kezelő rendszer PostGIS kiegészítőjét használja fel az Ulyxes. A PostGIS kiegészítő jelentősége abban rejlik, hogy segítségével a munkák során bemért téradatok tárolása válik lehetővé az adatbázisokban, melyek a PostGIS szerverén helyezkednek el. Az adatbázisból az adatok áramlását a PHP programnyelv segítségével valósítja meg a rendszer a web szerver felé. Az Ulyxes projekt során felhasznált web szervert a szintén szabadon használható Apache Web Server szolgáltatja. Az itt említett szerverek használata csak egy alternatíva az Ulyxes-hez kapcsolódóan. Ezeken kívül még kialakítható olyan megjelenítési módszer is mely a Google Maps API-t használja a térképi megjelenítéshez, valamint olyan lehetőség is létezik, melynél egy Map Server (pl.: WMS) biztosítja az adatok áramlását az adatbázis szerveréről a web szerver felé, azonban ezek a megvalósítási módok még nem készültek el teljes körűen az Ulyxes-t illetően.

2.2.3. Szenzorok kezelését lehetővé tevő alkalmazás programozói felület

A rendszer rendelkezik egy TclAPI-nak nevezett, Tcl programnyelvben íródott, magas szintű alkalmazás-programozói felülettel. Használatával a felhasználónak lehetősége nyílik viszonylag

alacsony programozói tudással, vezérlési műveletek és automatizált mérések végrehajtására a különböző típusú robot mérőállomásokon. A BSc diplomatervemben az Ulyxes részét képező TclAPI használatával és továbbfejlesztésével foglalkoztam, melynek keretén belül számos új programmal bővült a rendszer, melyek mind az automatizált mérések elvégzésével kapcsolatosak. Ezek mellett számos hibajavításra sor került, és végül, de nem utolsó sorban, a további felhasználók számára egy könnyen áttekinthető, strukturált formában lévő felhasználó dokumentáció is készült az API-hoz. A diplomamunka befejező fejezete a rendszerrel kapcsolatos fejlesztési lehetőségekről szólt, melyek közül az legnagyobb volumenű ötlet a TclAPI teljes újragondolása, objektumorientált programozási nyelvek és eszközök segítségével.



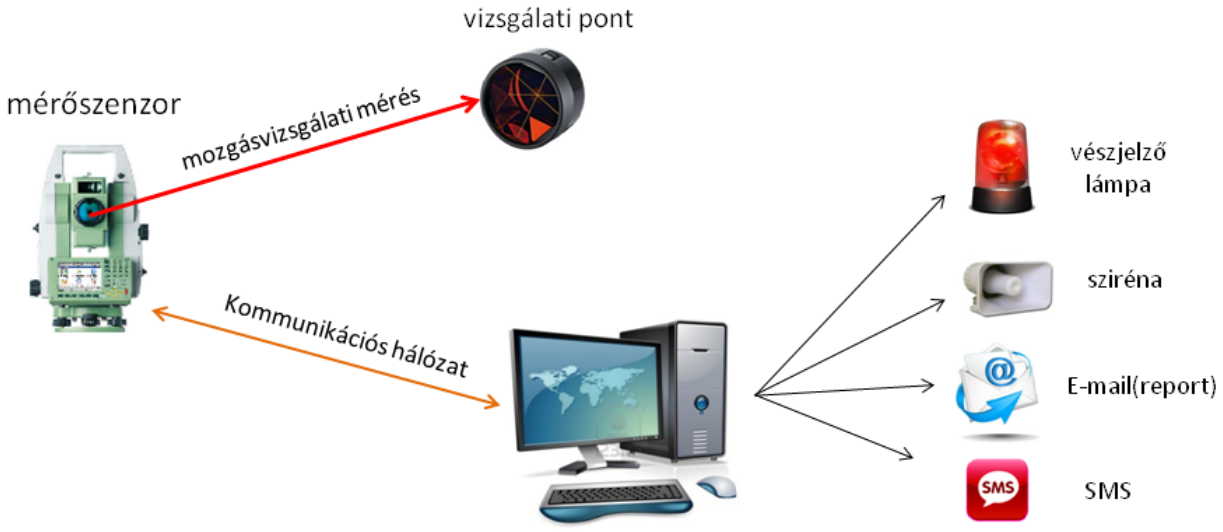
2-3 A szenzorok kezelését lehetővé tevő API tesztelése laboratóriumi körülmények között

2.3. A szenzorok kezelését lehetővé tevő API modelljének az újragondolása és megalkotása

Az API újragondolását és új alapokra helyezését megfogalmazó teória előmozdításához számos koncepció és ötlet is hozzájárult. Elsőként is a TclAPI csak a robot mérőállomások kezelésére és vezérlésére volt képes. A robot mérőállomások jelentősége a mérnökgeodézia munkálatokban természetesen nem vitatott, azonban több mozgásvizsgálati feladatnál megfelelő vagy akár jobb alternatívája lehet a mérőállomásnak a műholdas helymeghatározáson alapuló GPS technológia, például egy munkagép mozgásvizsgálatánál és vezérlésénél. Különböző, főként hosszabb ideig tartó valós idejű monitoring feladatoknál a légkör változásai nem elhanyagolható hatással van a méréseinkre, így az egyes meteorológiai elemekből számított korrekciókkal szükségszerű ellátni az egyes méréseinket. A különféle légköri adatok meghatározására meteorológiai szenzorokat használnak. Egy valós idejű mozgásvizsgálati rendszernél az egyes korrekciós adatok - azok változásának bekövetkezését követően - manuális bevitele mondjuk egy mérőállomásba

kényelmesen és gördülékenyen nem megoldható, azonban ennek elvégzését valamint az egyes műszerekkel való kommunikációt egy objektum-hierarchiát megvalósító rendszerben már könnyebb feladat. Általánosságban elmondható, hogy egy monitoring rendszer használhatósági és minőségi szintjéhez nagyban hozzájárul az egyes, monitoring alá vont vizsgálati objektumok kamerán ill. web-kamerán történő láttatása, így ezen eszközök kezelését is biztosítani kell. Természetesen a GPS-ek, kamerák és egyéb szenzorok integrálását egy alacsonyabb szintű API-ba is megoldható lenne, azonban egy objektum orientált rendszerbe az adott feladat rugalmasabban és átláthatóbb módon elvégezhető, és a rendszer újabb szenzorokkal történő bővítése is könnyebben végrehajtható.

A monitoring rendszerek legfőbb célja egy adott objektumban bekövetkező deformáció detektálása illetve az ebből származó információ továbbítása az illetékes személyeknek, mérnököknek. Egy mozgásvizsgálati pont helyzetével kapcsolatos információkat a megjelenítést lehetővé tevő felületen kívül egyéb módon is biztosítani kell, mivel felléphetnek olyan esetek is, amikor egy adott létesítmény mozgásvizsgálatánál bekövetkező deformáció veszélyeztetheti az objektumhoz kapcsolódó vagyoni értéket, valamint gondolni kell a létesítmény területén tartózkodó emberek életének a megóvására is. Ennek a biztosítására szolgálnak a különféle vészjelző rendszerek, melynek a logikai kialakítását a következő ábra illusztrálja:

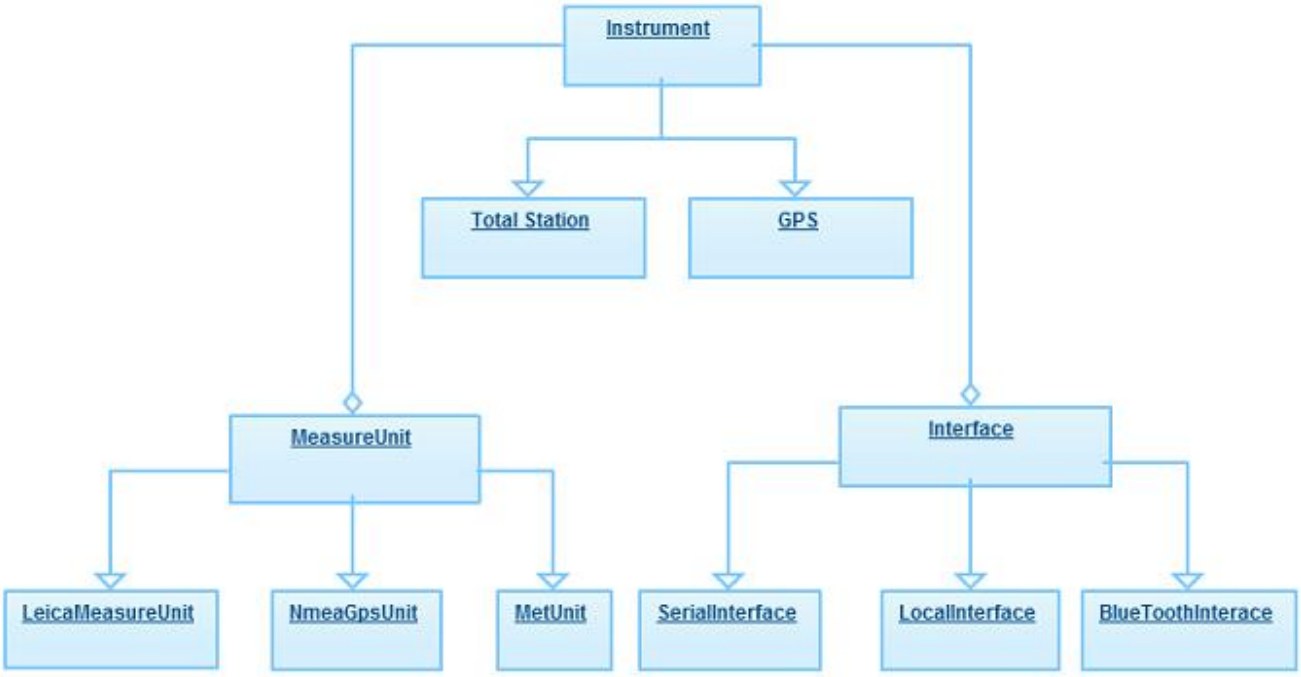


2-4 Vészjelző rendszerek kialakításának egyes alternatívái

A TclAPI Tcl programnyelvének a fejlesztése a szoftvermérnökök tekintetében az utóbbi időben háttérbe szorult, ami főként az újabbnál újabb objektum-orientált nyelvek megjelenése ad okot. Mind a Tcl, mind a többi hagyományos szekvenciális programozási nyelv a társadalom egyre nagyobb mértékben fejlődő információs technológiai és alkalmazás technikai igényeinek a

kielégítésére az alacsony absztrakciós szintjük következtében már nem képesek. A külvilág problémáit modellező minőségi alkalmazások és szoftverek létrehozását és fejlesztését tehát objektum-orientált nyelvek és eszközök segítségével érdemes megoldani napjainkban. A mi esetünkben az egyes mérnöki szenzorok modellezését és az ezt körülölelő szenzorvezérlést lehetővé tevő API megalkotását a Python objektum-orientált programnyelv segítségével hajtottuk végre. A Python egy általános célú, magas szintű programozási nyelv, mely egy nagyon gyors fejlesztési és tesztelési eszközt nyújt a felhasználó számára. Jelentősége a gyors fejlesztési képessége mellett abban rejlik, hogy számtalan kiegészítő könyvtárral (library) rendelkezik, így rendkívül széles körben alkalmazzák. Említésre méltó a térinformatikában betöltött szerepe, miszerint számos, téradatokat kezelő alkalmazás választotta fejlesztőeszközének, pl.: QGIS, ArcGIS.

A TclAPI Python programnyelvben megalkotott, PyAPI névre keresztelt változatának a logikai modelljének a kialakítása arra irányult, hogy egy olyan alkalmazás programozói felületet hozzunk létre, melyben az egyes mérnöki szenzorok között szülő-gyermek egyedkapcsolat alakul ki, ezzel egy könnyen átlátható és kezelhető rendszert kialakítva. Ezen modell kialakítás már könnyebben bővíthető újabb szenzorokkal. Az objektumtípusokat és az objektumok közötti asszociációs és tartalmazási viszonyokat leíró hierarchikus adatmodellt a következő ábra szemlélteti:



2-5 PyAPI hierarchikus adatmodell

A hierarchia modell tetején helyezkedik el az Instrument(Műszer) osztály, mely magába foglalja a rendszer által támogatott szenzorok közös tulajdonságait és metódusait. Ennek két „gyerek” osztálya van a Total Station(Mérőállomás) és a GPS osztály. Ezeken az osztályokon keresztül történik az egyes mérőegységek által fogadott üzenetek kiválasztása és továbbítása az adott kommunikációs csatornán. Az Instrument osztály tartalmaz egy MeasureUnit(Mérőegység) és egy Interface(Kommunikációs felület) osztályt, melyekből származnak az egyes mérőszenzorok és kommunikációs csatornákat megalkotó osztályok.

A MeasureUnit-ból származtatott osztályok már a konkrét mérőszenzorok megvalósított modelljei, melyek tartalmazzák az adott szenzorhoz vonatkozó eredmények kezelését, tárolását és a továbbítását illetve az egyes üzenetek deklarálása is itt történik. A MeasureUnit gyerek-osztályai:

- LeicaMeasureUnit: A Leica típusú műszerek szülő-osztálya
- NmeaGpsUnit: Az Nmea üzeneteket küldő GPS mérőműszer osztálya
- MetUnit: A meteorológiai szenzort megvalósító osztály

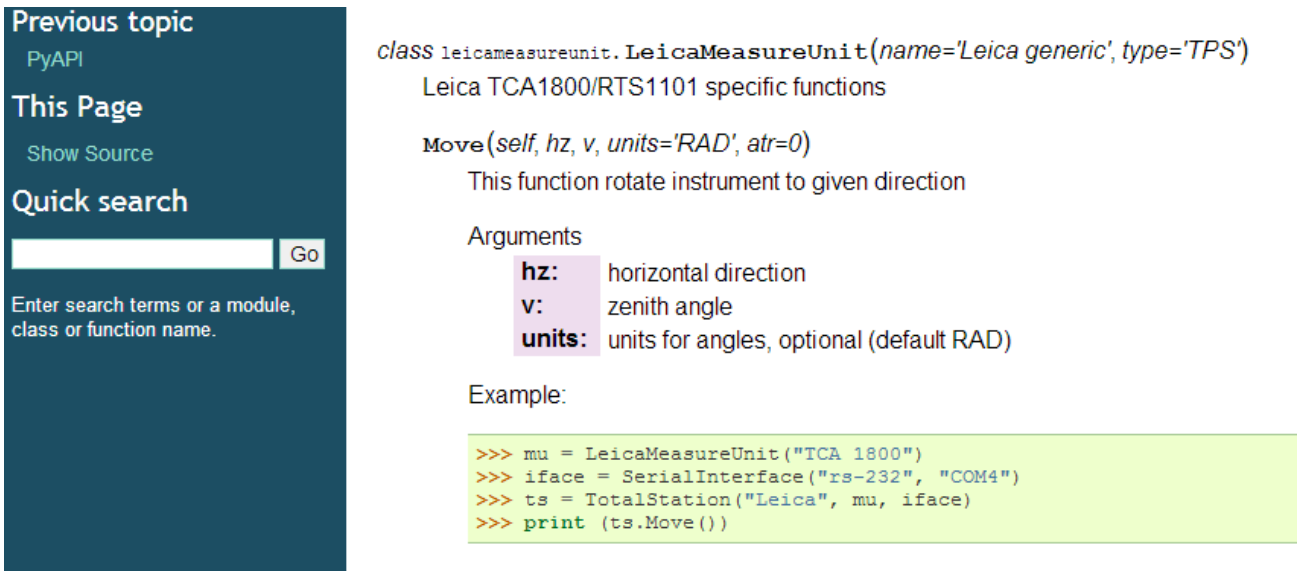
A LeicaMeasreUnit osztály tehát a MeasureUnit osztály tekintetében „gyerek” szerepet tölt be, azonban „szülő” osztályként is szerepelhet a további, újabb Leica műszerek számára. A rendszer által támogatott Leica műszerek közül a TCA1800 valamint a TPS1103-as robot mérőállomás tartozik ebben az osztályba. A Leica 1201-es robot mérőműszer vezérlési üzenetei kis százalékában eltérés mutatkozik az előbb említett két régebbi típusú mérőállomás üzeneteitől, ezért annak érdekében, hogy a rendszerbe beintegráljuk az új műszert, szükség van egy új gyere-osztály kialakítására az objektum-orientált programozás öröklés tulajdonságát felhasználva. Ennek következtében az új osztályban csak az őstől eltérő helyzeteket kell kezelni. A rendszerbe további szenzorok gyors integrálása az így említett módon valósítható meg.

Az Ulyxes logikai modelljének az újragondolása azért is volt szükség, hogy igény szerint, könnyedén és rugalmasan lehessen az egyes kommunikációs csatornák között válogatni. Bár a lista még nem teljes, és csak részben kialakított, a PyAPI Interface a következő, kommunikációs csatornák használatát lehetővé tevő gyerek-osztályokkal rendelkezik:

- SerialInterface: soros vonali kommunikációs kapcsolat
- BlueToothInterface: bluetooth-on történő kommunikációs kapcsolat

- LocalInterface: a rendszer fejlesztését lehetővé tevő helyi kommunikációs kapcsolatkialakítás, segítségével műszer nélkül is tesztelhetjük a rendszert

Annak érdekében, hogy a PyAPI-ban lévő osztályokat, függvényeket és rutinokat az egyes felhasználók könnyen áttekinthető és strukturált formában lássák, egy fejlesztői dokumentációt készítettem, melynek neve: **Ulyxes PyAPI documentation**. Felhasználói dokumentációra akkor van szükség, amikor az adott programot (mi esetünkben alkalmazás-programozói felületet) a készítőn kívül más felhasználók is használni fogják. Számukra biztosítani kell egy olyan, interneten elérhető dokumentációt illetve leírást, melynek használatával egyértelműen világossá válik a programban lévő osztályok, metódusok működése valamint a program tevékenységi köre. A PyAPI felhasználói dokumentáció a Sphinx dokumentációgeneráló segítségével lett készítve. A Sphinx egy web-designer által készített, a hagyományos dokumentáció készítőknél sokkal több lehetőséggel bíró dokumentációgeneráló. A HTML kimeneti struktúra mellet rendelkezik pdf kiterjesztésű generálóval, valamint az egyes szintaxisok ábrázolásához szintaxiskiemelő-t használ, így egy könnyebben átlátható pogrom-struktúra megjelenítést biztosítva (pl.: példa adása egy függvény használathoz (2.6. ábra)). Ezek mellett a Sphinx által készített, HTML alapú dokumentációk rendelkeznek egy integrált, JavaScript alapú keresőmotorral is, melynek köszönhetően a dokumentációban történő keresés nagyságrendekkel gyorsabb.



2-6 A LeicaMeasureUnit.py Move(mozgatás) függvényének a dokumentációja

3. A rendszer használata a Hárosi híd próbaterhelésénél

A feladat célja az Ulyxes rendszer műszervezérlő egységének a hatékonyságának a szemléltetése az M0 Hárosi Duna-híd használatba vételét megelőző terhelési próba során. A híd teherbíró képességének a vizsgálatai több teherállásból állt, vizsgálva az egyes statikus és dinamikus hatásokat. A statikus próbaterhelési sorozat során 12 darab statikus teherállás hoztak létre járművek segítségével, melyek a híd hossztengelyére szimmetrikusan helyezkedtek el. A felmérés során a híd alakváltozását felsőrendű automata szintezőműszerekkel vizsgálták az egyes teherállások által bekövetkezett lehajlásokat, valamint ezzel egyidejűleg az Ulyxes rendszer használatával, Leica1201-es típusú robot mérőállomással vizsgáltuk a lehajlásokat.

3-1 Leica TPS 1201 típusú robot mérőállomások mérés közben

A mozgásvizsgálati méréseinket 14 darab közel egyenlő távolságra helyezett prizmára végeztük, melyeket a híd kifolyási oldalán pillanatszorítók segítségével rögzítettük. A prizmák automatizált méréseinek a megvalósítása érdekében, előzetesen készíteni kellett egy programot, mely az egyes prizmákat adott sorrendben leméri, a mérési eredményének pedig tárolja. A prizmákra teherállásonként 3 (4-es és 11-es teherállásnál 4 darab) mérési sorozat került rögzítésre, minden egyes mérésnél az időpontokat feljegyezve.

Elsőként a különböző teherállásokban, kísérleti feladat jelleggel megvizsgálva került, hogy egy teherálláson belül mozognak e a vizsgálati pontok, tehát a mérések elkezdésekor beállt-e a híd adott teherállást jellemző nyugalmi állapotra. A 2. teherállásnál az egyes pontok a második méréssorozatban az elsőhöz képest mozognak 1-2cm-t, tehát a híd még nem állt be a teherállásnak megfelelő nyugalmi állapotába. A harmadik méréssorozat értékeit a másodikéval összehasonlítva szintén mozgás tapasztalható, azonban itt a legnagyobb függőleges irányú mozgás 4-5mm volt. A többi teherállást megvizsgálva elmondható, hogy a vizsgálati pontok nem mozogtak az egyes teherállások méréssorozatai között.

Az egyes lehajlások értékeinek a kiszámítását és a nyílt forráskódú Octave-ban hajtottam végre. Az Octave egy nyílt forráskódú programkörnyezet, melynek használatával különféle numerikus számításokat, vizuális megjelenítéseket és programozási megoldásokat vihetünk véghez. Az itt kiszámított lehajlások maximum értékei összehasonlításra kerültek az ANSYS véges elemes programrendszer híd-terhelés modelljéből számított előzetes értékeivel[DUNA]. Az összehasonlítás eredményeképpen elmondható, hogy az Ulyxes rendszer által mért lehajlások értékei 5-10mm-rel tértek el az ANSYS program szimulációjával kapott értékektől. Ebből következtetésképpen mind az ANSYS program használhatósága, mind pedig az Ulyxes rendszer által alkalmazott automatizált mérési módszer gyakorlati működése igazolható.

Max. elmozdulások [mm]	ANSYS	Ulyxes	Δ
2. teherállás	-79,4	-71,8	7,6
4. teherállás	-119,5	-117,9	1,6
6. teherállás	-71,3	-72,7	1,4
7. teherállás	-71,3	-69,6	1,7
9. teherállás	-47,8	-37,4	10,4
10. teherállás	-47,8	-40	7,8
11. teherállás	-47,1	-48,5	1,4

3-2 Az ANSYS-ban tervezett és az Ulyxes rendszerrel mért maximális lehajlások összehasonlítása

A próbaterhelés során az árvíz miatt a tervezett műszerálláspont helyett új helyre kellett felállnunk a műszerekkel, mely a legközelebbi vizsgálati ponttól 200 méter, a legtávolabbi prizmától közel 540m-re esett. Ez a meghatározott lehajlások középhibáját megnövelte, ezért a magasságkülönbség középhibáját meghatároztuk a hibaterjedés törvényének alkalmazásával:

$$M = t_{ferde} * \cos Z$$

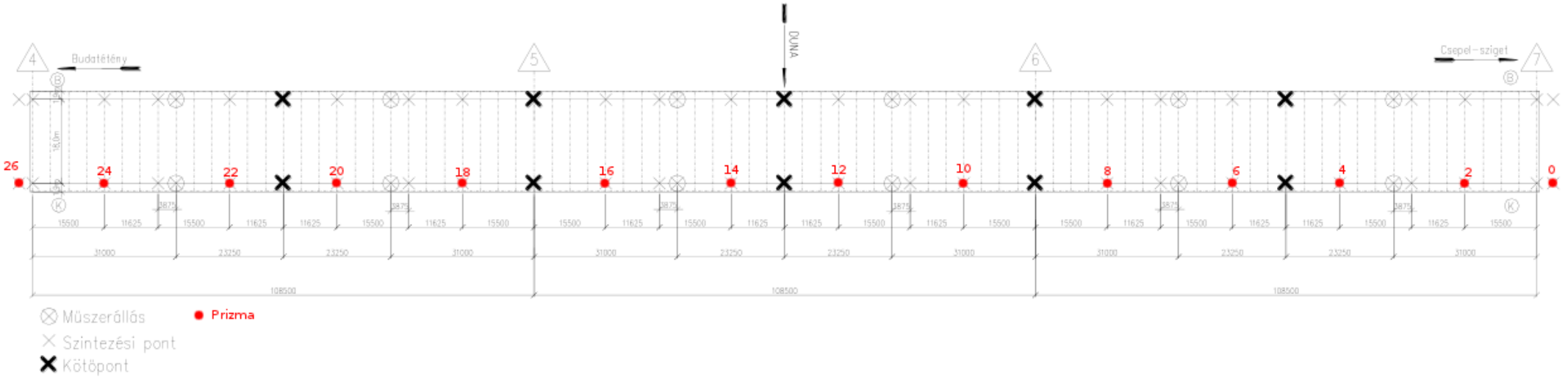
$$mM = \sqrt{\left(\frac{\partial h}{\partial\, \mathrm{t_{ferde}}}\right)^2 * m_{tf+}^2 \left(\frac{\partial h}{\partial \mathrm{Z}}\right)^2 * \frac{m_z^2}{\rho^2} + \left(\frac{\partial h}{\partial \delta}\right)^2 * \frac{m_\delta^2}{\rho^2}}$$

3-3. A magasságkülönbség középhibájának a kiszámítása a hibaterjedés törvényéből

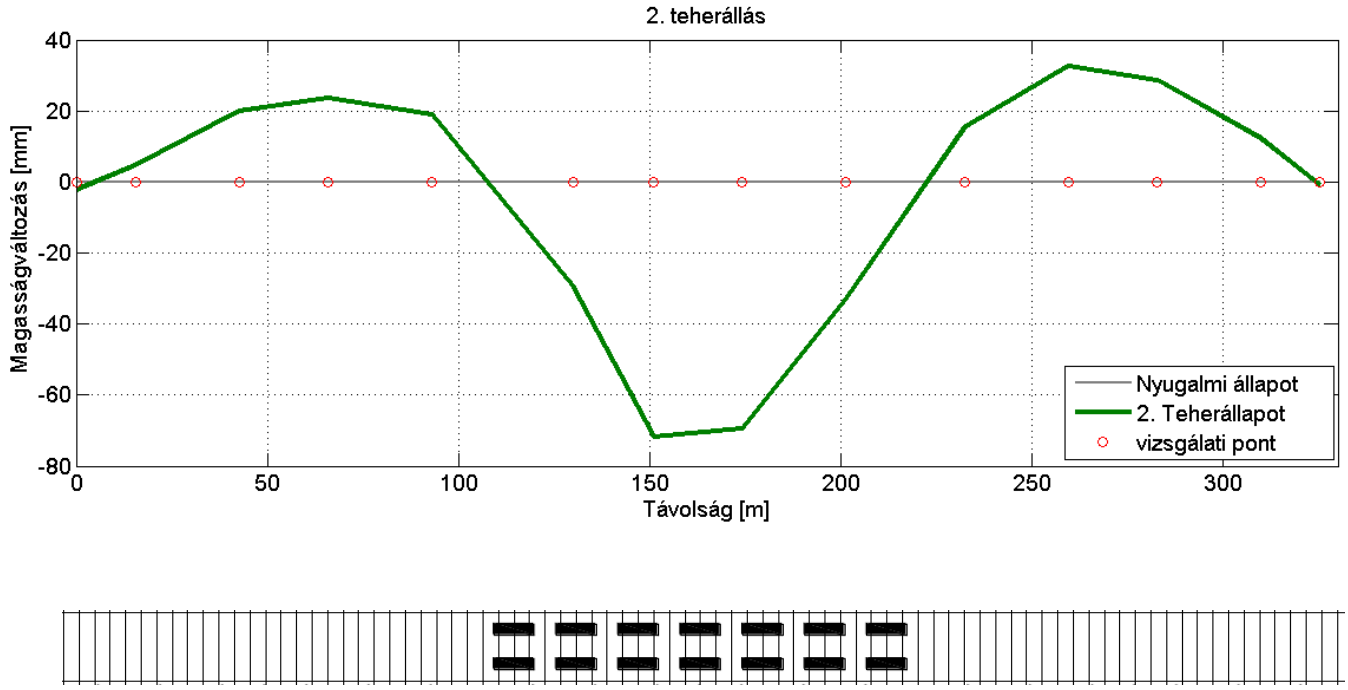
A magasságkülönbség(M) középhibája 540m-es távolságnál 2.6mm, 325m-es távolságnál pedig 1.6mm-re adódott. Az itt kapott számérték valamint az egyes lehajlások nagyságrendjét összevetve megállapítható, hogy mérési eredményeink valós információkkal rendelkező számértékek, viszont az árvíz miatti műszerállás helyének a változása 1 mm-el megnövelte lehajlásokat jellemző középhiba értékét.

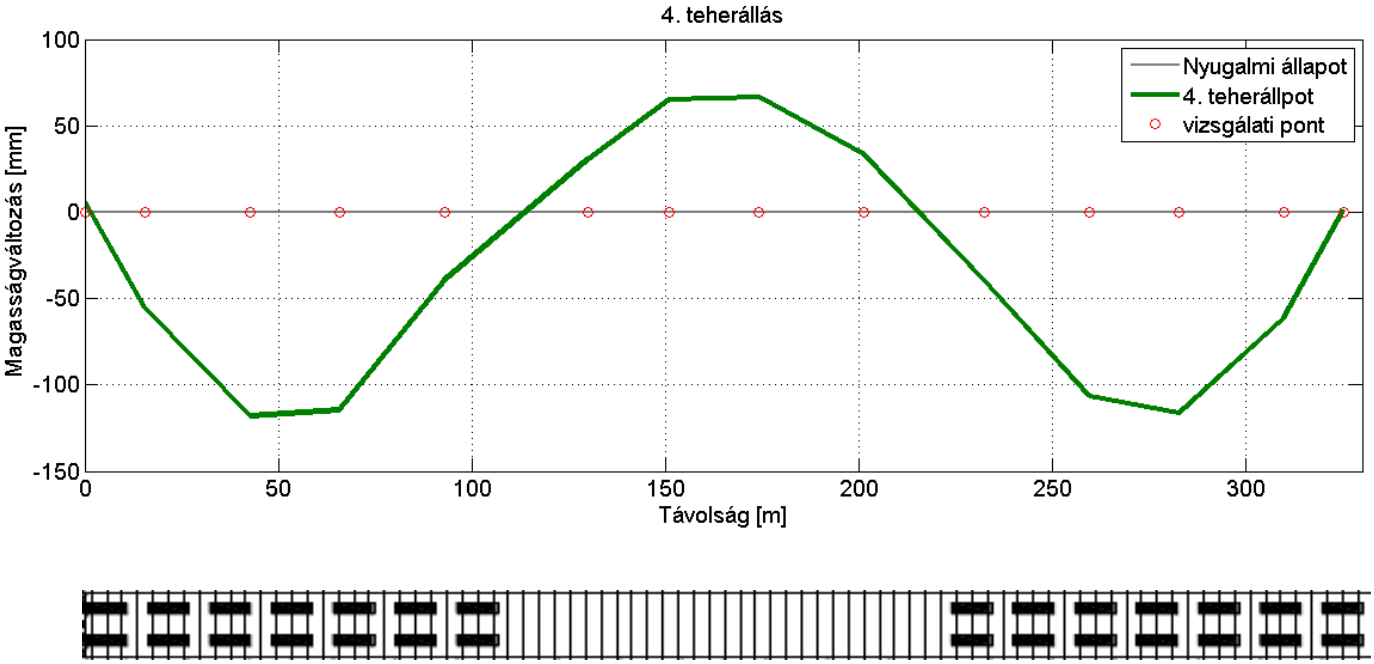
3-4 A statikus terhelek(járművek) elhelyezkedése a Hárosi Duna-hídon [BMEH]

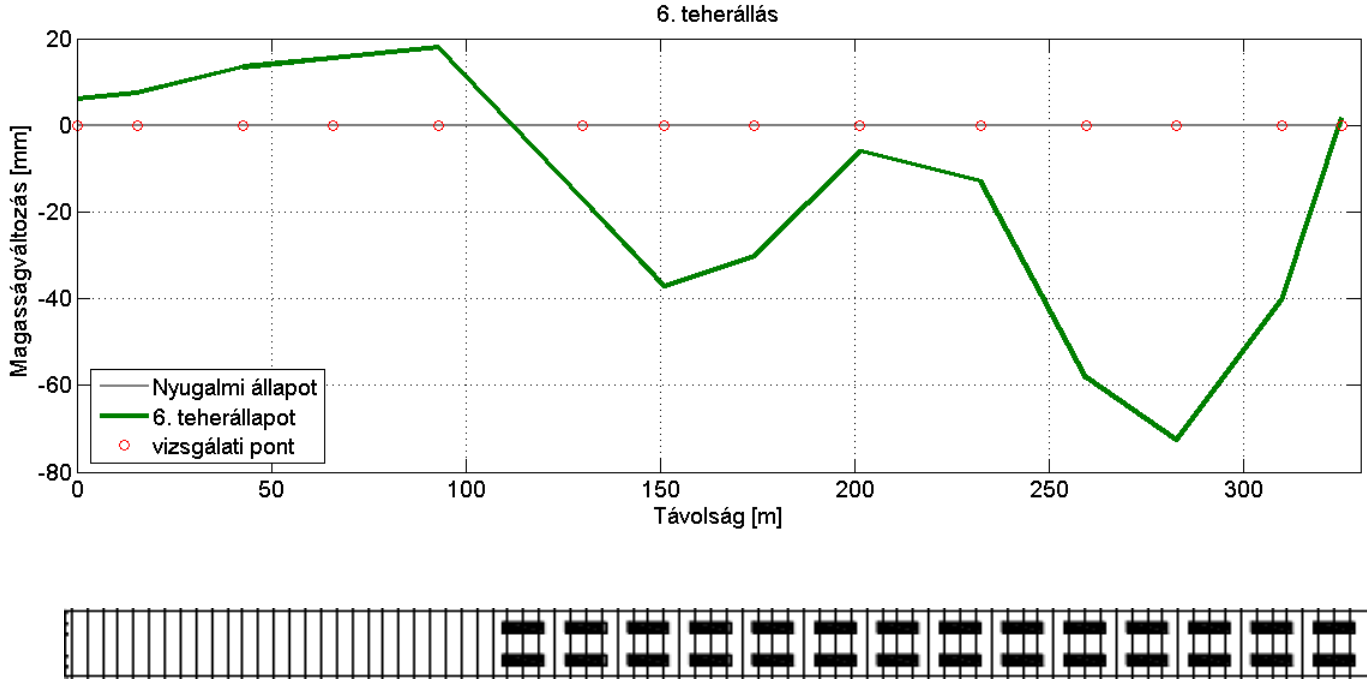
A vizsgált prizmák elhelyezkedése Hárosi Duna-híd oldal[DUNA]

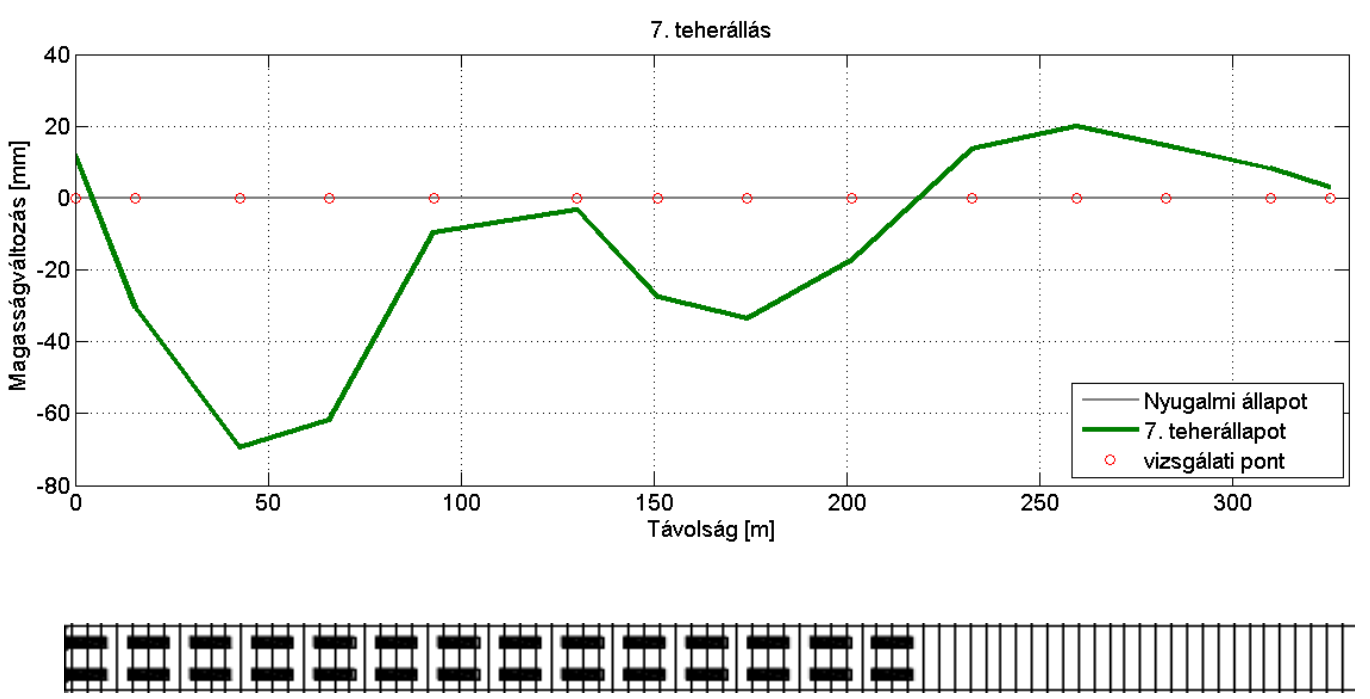


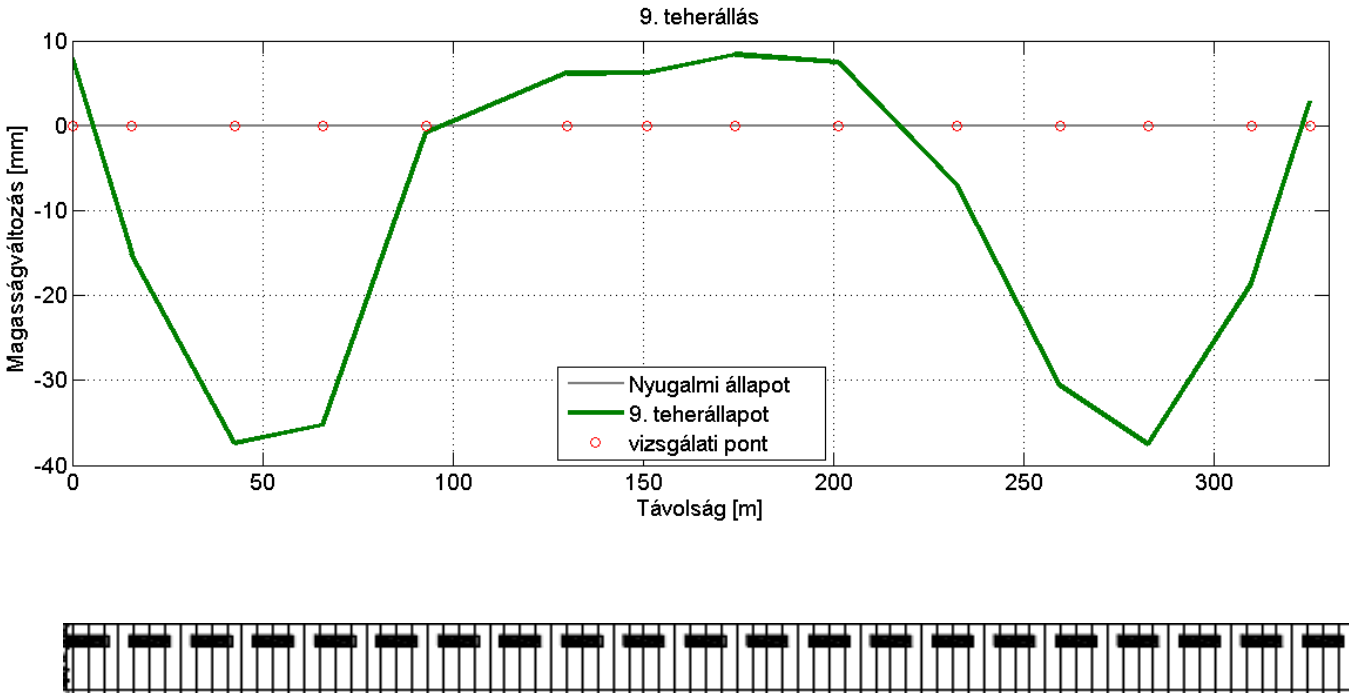
A következő ábrák az egyes teherállásokhoz kapcsolódó lehajlások értékeit szemléltetik, alattuk feltüntetve a statikus terhek elhelyezkedését. Az grafikonok 0 abszcisszája megegyezik a híd budai oldali első mederpillérével:

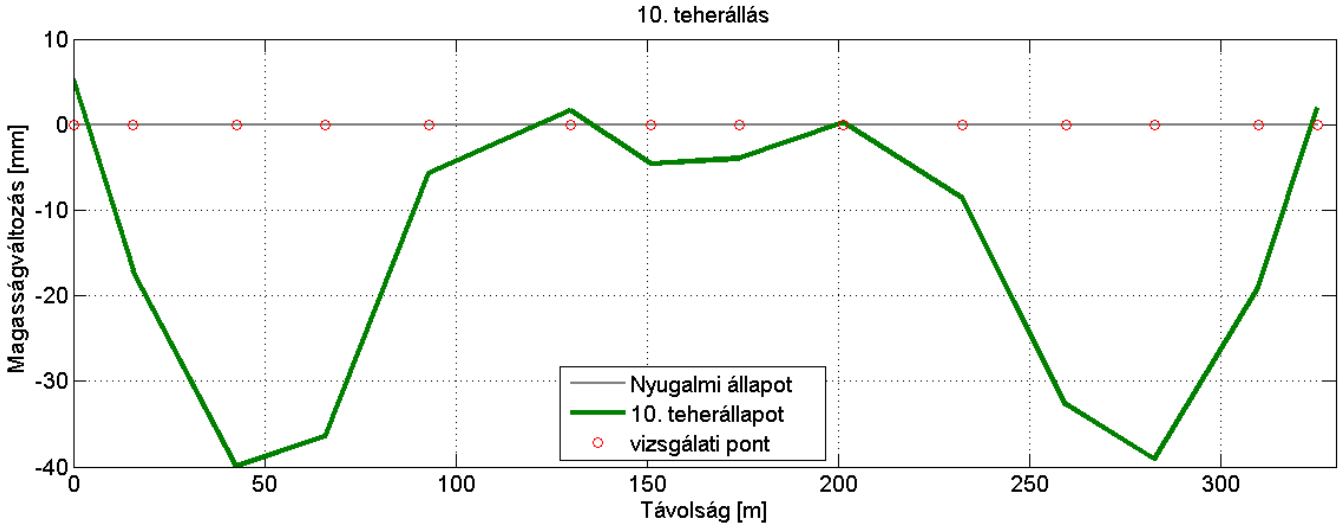


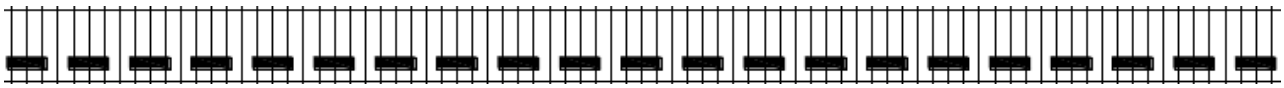


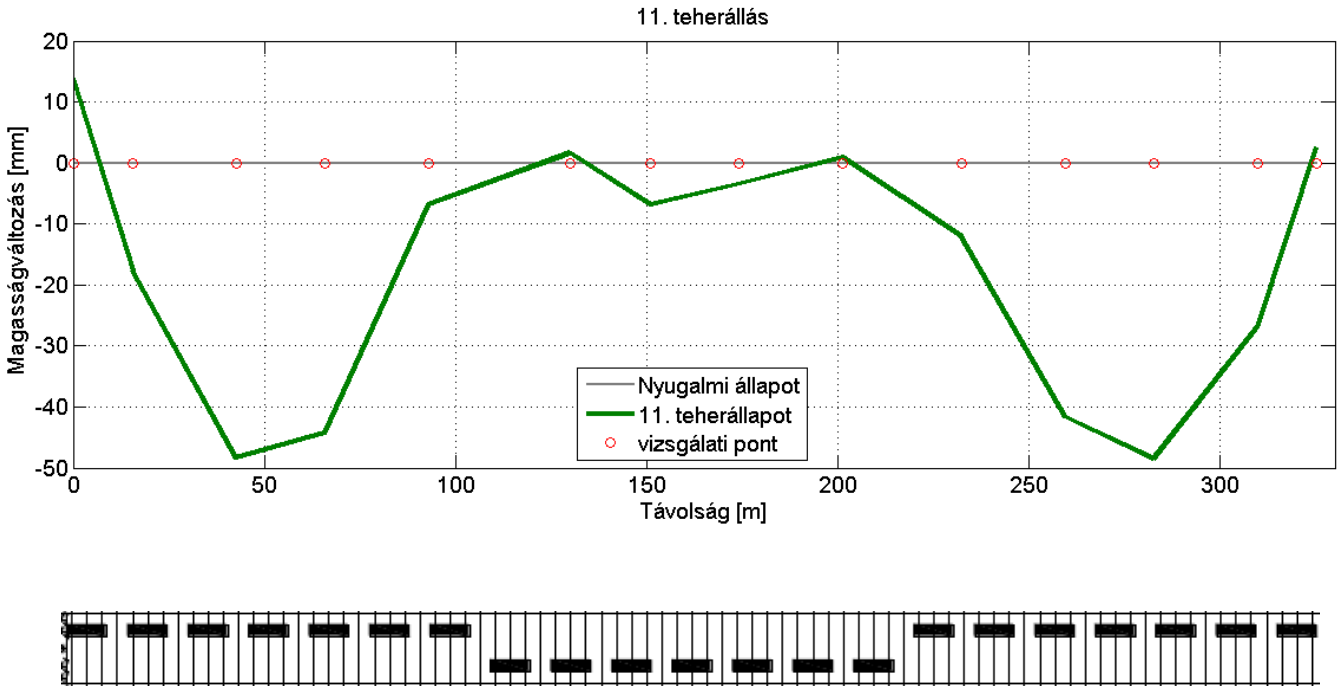

6. teherállás
Magasságváltozás [mm]
Távolság [m]
Nyugalmi állapot
6. teherállapot
vizsgálati pont


7. teherállás
Magasságváltozás [mm]
Távolság [m]
Nyugalmi állapot
7. teherállapot
vizsgálati pont


9. teherállás
Magasságváltozás [mm]
Távolság [m]
Nyugalmi állapot
9. teherállapot
vizsgálati pont


10. teherállás
Magasságváltozás [mm]
Távolság [m]
Nyugalmi állapot
10. teherállapot
vizsgálati pont





A eredményeket tekintve a mérési módszer hatékonyságát illetően megállapítható, hogy az Ulyxes rendszer segítségével létrejött automatizált mozgásvizsgálati rendszer a vele párhuzamosan történő felsőrendű szintezéshez képest szignifikánsan gyorsabb. A felsőrendű szintezés végrehajtása több időt vesz igénybe, valamint jóval több ember(10+) bevonását kívánja meg, ezzel szemben a mi technológiánk egy személy által elvégezhető, továbbá a hagyományos felsőrendű szintezés analóg adatgyűjtésével szemben digitális, automatizált adatgyűjtési eljárást alkalmaz. A műszer és a prizma közötti távolságok csökkentésével a mérőállomással elérhető középhiba csökkenthető.

4. A rendszer jövője

Az Ulyxes projekt a diplomamunkám befejezése után számos fejlesztéssel bővült, melyek mind hozzájárultak a rendszer hatékonyságához és sokrétűségéhez. Ilyen például a GPS műszerek integrálása a rendszerbe, valamint a dolgozatban tárgyalt, szenzorvezérlő egység logikai modelljének az újragondolása és megalkotása. A PyAPI azonban még nem teljes, ezért még szükség van a további fejlesztésére. Ezen kívül a rendszert számos egyéb új funkcióval lehetne még bővíteni és továbbfejleszteni. Ezek közül néhányat megemlítve:

- Szenzorok vezérlése okos telefonon keresztül;
- Újabb szenzorok integrálása a rendszerbe (pl.: Trimble RTS-ek, web-kamerák);
- Különböző kommunikációs csatornák alkalmazása a műszervezérléshez;

- A rendszer alapegységei (szenzorvezérlő felület, megjelenítést lehetővé tevő felület, szerverek) működésének az összehangolása, ezzel kialakítva egy teljes körű, automatizált monitoring rendszert.

5. Összefoglalás

A dolgozatom célja, hogy az olvasó számára betekintést nyújtsak a mozgásvizsgálati rendszerek, ezeken belül is a szenzorvezérlő felület objektum-orientált nyelv segítségével történő programozásának a lehetőségeire és hatékonyságára. Az Ulyxes egy olyan monitoring rendszer megvalósítás, mely kizárólag nyílt forráskódú eszközök használatával valósul meg, és melynek a fejlesztése a mai napig tart a készítő, szakdolgozatokat író és végül, de nem utolsó sorban az önkéntes mérnökök által. Az Ulyxes elemi részét képező PyAPI egy Python programnyelvben megalkotott alkalmazás programozói felület, melynek segítségével képesek lehetünk különféle geodéziai szenzorok vezérlésére, irányítására illetve automatizált mérések elvégzésére és feldolgozására. A PyAPI logikai modell kialakításának fő szempontjai a mérőegységek rugalmas kezelése és a rendszerbe történő új szenzorok integrálásának a megkönnyítése, kihasználva az objektum-orientált programozás három alappillérét, az egységbezárást, öröklést és a polimorfizmust.

A rendszerrel kapcsolatban számos lehetőség nyílik azok számára, akik rendelkeznek némi tudással mind platform és mind pedig web-programozás terén, valamint igényt tartanak automatizált monitoring feladatok elvégzésére és az ezzel kapcsolatos előnyök kihasználására.

Irodalomjegyzék

[MARK] *Mark Pilgrim:* **Dive into python** (322 oldal, Apress Kiadó, 2004)

[BRED] *Bred D. McLaughlin:* ***Head First Object-Oriented Analysis and Design*** (636 oldal, Oreilly, 2006)

[DUNA] *Dunai László* **Hárosi híd próbaterhelési terv - Mederhíd.pdf** (25 oldal, BME, 2012)

[ULYX] **Az Ulyxes rendszer honlapja:** http://www.agt.bme.hu/ulyxes/

[BMEH] *BME.hu cikk:* **„Műegyetemi szakértelem az új Duna-Híd szolgálatában”:** http://www.bme.hu/hirek/20130606/muegyetemi_szakertelem_az_uj_Duna-hid_szolgalataban

Melléklet

A PyAPI moduljainak forráskódjai:

angle.py:

```
#!/usr/bin/env python
import math
"""
This file contains common instrument, angle conversion functions
    <p>Ulyxes - an open source project to drive total stations and
           publish observation results</p>
    <p>GPL v2.0 license</p>
    <p>Copyright (C) 2010-2013 Zoltan Siki <siki@agt.bme.hu></p>
    @author Zoltan Siki
    @author Daniel Moka
    @version 1.1
"""
RO = 180 * 60 * 60 / math.pi

class Angle(object):
    def __init__(self, value=0, unit='RAD'):
        self.SetAngle(value, unit)

    def GetAngle(self, out):
        if out == 'RAD':
            output = self.value
        elif out == 'DMS':
            output = self.__DMS()
        elif out == 'DEG':
            output = self.__Rad2Deg()
        elif out == 'GON':
            output = self.__Rad2Gon()
        else:
            # to do : write error in log file
            print('The output format is unknown: try RAD,DEG,DMS,GON')
        return output

    def SetAngle(self, value, unit='RAD'):
        if unit == 'RAD':
            self.value = value
        elif unit == 'DMS':
            self.value = self.__DMS2Rad(value)
        elif unit == 'DEG':
            self.value = self.__Deg2Rad(value)
        elif unit == 'GON':
            self.value = self.__Gon2Rad(value)
        elif unit == "NMEA":
            self.value = self.__NMEA2Rad(value)
        else:
            # unknown unit
            self.value = 0

    def __Deg2Rad(self, angle):
        return math.radians(angle)

    def __Gon2Rad(self, angle):
```

```
        return (angle / 200.0 * math.pi)

    def __DMS2Rad(self, dms):
        #degrees    =    numpy.sum(numpy.fromstring(dms,sep='-')*[1.0,    1/60.0,
1/3600.0])
        items = [float(item) for item in dms.split('-')]
        return math.radians(items[0] + items[1] / 60.0 + items[2] / 3600.0)

    def __Rad2Gon(self):
        return (self.value / math.pi * 200.0)

    def __Rad2Sec(self):
        return self.value * RO

    def __Rad2Deg(self):
        return math.degrees(self.value)

    def __DMS(self):
        secs = round(self.__Rad2Sec())
        min, sec = divmod(secs, 60)
        deg, min = divmod(min, 60)
        deg = int(deg)
        #dms = '{}-{}-{}'.format(deg,round(min),round(sec))
        dms = "%d-%02d-%02d" % (deg, min, sec)
        return dms

    def __NMEA2Rad(self, v):
        pass

if __name__ == "__main__":
    a = Angle('359-59-59', 'DMS')
    print (a.GetAngle('RAD'))
    print (a.GetAngle('DMS'))
    print (a.GetAngle('DEG'))
    print (a.GetAngle('GON'))
    print (Angle(a.GetAngle('RAD'), 'RAD').GetAngle('DMS'))
    print (Angle(a.GetAngle('DMS'), 'DMS').GetAngle('DMS'))
    print (Angle(a.GetAngle('DEG'), 'DEG').GetAngle('DMS'))
    print (Angle(a.GetAngle('GON'), 'GON').GetAngle('DMS'))
```

instrument.py:

```
#!/usr/bin/env python
"""
    <p>Ulyxes - an open source project to drive total stations and
           publish observation results</p>
    <p>GPL v2.0 license</p>
    <p>Copyright (C) 2010-2013 Zoltan Siki <siki@agt.bme.hu></p>
    @author Zoltan Siki
    @author Daniel Moka
    @version 1.1
"""
from interface import *
from measureunit import *
import re

class Instrument(object):
    "base class for different instruments"
    def __init__(self, name, measureUnit, measureInterf):
        self.name = name
        self.measureUnit = measureUnit
        self.measureInterf = measureInterf

    def GetInterface(self):
        return self.measureInterf

    def GetMeasureUnit(self):
        return self.measureUnit

    def GetName(self):
        return self.name

if __name__ == "__main__":
    mu = MeasureUnit('Test', 'Proba')
    iface = Interface('interf')
    print (mu.GetName())
    print (mu.GetType())
    print (iface.GetName())
    print (iface.GetState())
    a = Instrument("test instrument", mu, iface)
```

leicameasureunit.py:

```
#!/usr/bin/env python
"""
    Leica TCA1800/RTS1200 specific functions
    <p>Ulyxes - an open source project to drive total stations and
           publish observation results</p>
    <p>GPL v2.0 license</p>
    <p>Copyright (C) 2010-2013 Zoltan Siki <siki@agt.bme.hu></p>
    @author Zoltan Siki
    @author Daniel Moka
    @version 1.1
"""

from measureunit import *
from angle import *
import re

class LeicaMeasureUnit(MeasureUnit):
    def __init__(self, name = 'Leica generic', type = 'TPS'):
        # call super class init
        super(LeicaMeasureUnit, self).__init__(name, type)

    def Result(self, msg, ans):
    # get command id form message
        msgBufflist = re.split(':|,',msg)
        commandID = msgBufflist[1]
    # get error code from answer
        ansBufflist = re.split(':|,',ans.decode('utf-8'))
        errCode = ansBufflist[3]
        if errCode != '0':
            # ??? TODO ?Logging?
            return {'errorCode': errCode}
        else:
            #Measure()
            if commandID == '2108':
                hz = Angle(float(ansBufflist[4]))
                v = Angle(float(ansBufflist[5]))
                dist = ansBufflist[6]
                return  {'errorCode':  0,  'hz':  hz.GetAngle('DMS'),  'v':
v.GetAngle('DMS'), 'distance': dist}
            #MeasureDistAng
            if commandID == '17017':
                hz = Angle(float(ansBufflist[4]))
                v = Angle(float(ansBufflist[5]))
                dist = ansBufflist[6]
                return  {'errorCode':  0,  'hz':hz.GetAngle('DMS')  ,  'v':
v.GetAngle('DMS'), 'distance': dist}
            #GetATR()
            elif commandID == '9019':
                atrStat = ansBufflist[4]
                return {'errorCode': 0, 'atrStatus': atrStat}
            #GetLockStatus()
            elif commandID == '9021':
                lockStat = ansBufflist[4]
                return {'errorCode': 0, 'lockStatus': lockStat}
            #GetAtmCorr()
            elif commandID == '2029':
                valueOfLambda = ansBufflist[4]
                pressure = ansBufflist[5]
                dryTemp = ansBufflist[6]
```

```
                wetTemp = ansBufflist[7]
                return {'errorCode': 0, 'lambda': valueOfLambda, 'pressure':
pressure, 'dryTemp': dryTemp, 'wetTemp': wetTemp}
            #GetRefCorr()
            elif commandID == '2031':
                status = ansBufflist[4]
                earthRadius = ansBufflist[5]
                refracticeScale = ansBufflist[6]
                return {'errodCode': 0, 'Status': status, 'earthRadius':
earthRadius, 'refracticeScale': refracticeScale }
            #GetStation()
            elif commandID == '2009':
                east = ansBufflist[4]
                north = ansBufflist[5]
                elevation = ansBufflist[6]
                return {'errorCode': 0, 'easting': east, 'northing': north,
'elevation': elevation}
            # GetEDMMode()
            #PASTE the hashed part TO 1200 UNIT
            elif commandID == '2021':
                edmMode = ansBufflist[4]
                #edmModeMap={'0': 'IR Std', '1': 'IR Fast', '2': 'LO Std',
'3': 'RL Std', '4': 'IR Trk', '6': 'RL Trk', '7': 'IR Avg', '8': 'LO Avg',
'9': 'RL Avg'}
                return {'errorCode':0, 'edmMode':edmMode}
            #Coords()
            elif commandID == '2082':
                yCoord = ansBufflist[4]
                xCoord = ansBufflist[5]
                zCoord = ansBufflist[6]
                return {'errorCode': 0, 'y': yCoord, 'x': xCoord, 'z':zCoord}
            #GetAngles()
            elif commandID == '2003':
                hz = Angle(float(ansBufflist[4]))
                v = Angle(float(ansBufflist[5]))
                return {'errorCode' : 0, 'hz': hz.GetAngle('DMS'), 'v':
v.GetAngle('DMS')}

        return {'errorCode' : 0}

    def SetATRMsg(self, atr):
        return '%%R1Q,9018:%d' % (atr)

    def GetATRMsg(self):
        return '%R1Q,9019:'

    def SetLockMsg(self, lock):
        return '%%R1Q,9020:%d' % (lock)

    def GetLockMsg(self):
        return '%R1Q,9021:'

    def SetAtmCorrMsg(self, valueOfLambda, pres, dry, wet):
        return '%%R1Q,2028:%f,%f,%f,%f' % (valueOfLambda, pres, dry, wet)

    def GetAtmCorrMsg(self):
        return '%R1Q,2029:'

    def SetRefCorrMsg(self, status, earthRadius, refracticeScale):
        return '%%R1Q,2030:%d,%f,%f' % (status, earthRadius, refracticeScale)
```

```
def GetRefCorrMsg(self):
    return '%R1Q,2031:'

def SetStationMsg(self, e, n, z):
    return '%%R1Q,2010:%f,%f,%f' % (e, n, z)

def GetStationMsg(self):
    return '%R1Q,2009:'

def SetEDMModeMsg(self, mode):
    #2 = IR
    #5 = Rl
    return '%%R1Q,2020:%d' % (mode)

def GetEDMModeMsg(self):
    return '%R1Q,2021:'

def SetOriMsg(self, ori, units ='RAD'):
    ori_rad = Angle(ori, units).GetAngle('RAD')
    return '%%R1Q,2113:%f' % (ori_rad)

def SetRCSMsg(self, rcs):
    return '%%R1Q,18009:%f' % (rcs)

def MoveMsg(self, hz, v, units='RAD', atr=0):
    # change angles to radian
    hz_rad = Angle(hz, units).GetAngle('RAD')
    v_rad = Angle(v, units).GetAngle('RAD')
    return '%%R1Q,9027:%f,%f,0,%d,0' % (hz_rad, v_rad, atr)

def MeasureMsg(self, prg = 2, wait = 12000, incl = 0):
    return '%%R1Q,2008:%d,%d|%%R1Q,2108:%d,%d' % (prg, incl, wait, incl)

def MeasureDistAngMsg(self):
    return '%R1Q,17017:2'

def CoordsMsg (self, wait = 1000, incl = 0):
    return '%%R1Q,2082:%d,%d' % (wait, incl)

def GetAnglesMsg(self):
    return '%R1Q,2003:0'

def ClearDistanceMsg(self):
    return '%R1Q,2082:1000,1'
    #return '%R1Q,2082:1000,1'

def ChangeFaceMsg(self):
    return '%R1Q,9028:'
```

totalstation.py:

```
#!/usr/bin/env python
"""
    <p>Ulyxes - an open source project to drive total stations and
           publish observation results</p>
    <p>GPL v2.0 license</p>
    <p>Copyright (C) 2010-2013 Zoltan Siki <siki@agt.bme.hu></p>
    @author Zoltan Siki
    @author Daniel Moka
    @version 1.1
"""

from instrument import *
from angle import *

class TotalStation(Instrument):

    def __init__(self, name, measureUnit, measureInterf):
        # call super class init
        super(TotalStation, self).__init__(name, measureUnit, measureInterf)

    def SetATR(self, atr):
        msg = self.measureUnit.SetATRMsg(atr)
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def GetATR(self):
        msg = self.measureUnit.GetATRMsg()
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def SetLock(self,lock):
        msg = self.measureUnit.SetLockMsg(lock)
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def GetLock(self):
        msg = self.measureUnit.GetLockMsg()
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def SetAtmCorr(self,valueOfLambda, pres, dryTemp, wetTemp):
        msg  =  self.measureUnit.SetAtmCorrMsg(valueOfLambda,  pres,  dryTemp,
wetTemp)
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def GetAtmCorr(self):
        msg = self.measureUnit.GetAtmCorrMsg()
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def SetRefCorr(self, status, earthRadius, refracticeScale):
        msg     =     self.measureUnit.SetRefCorrMsg(status,     earthRadius,
refracticeScale)
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def GetRefCorr(self):
        msg = self.measureUnit.GetRefCorrMsg()
```

```
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def SetStation(self, easting, northing, zenith):
        msg = self.measureUnit.SetStationMsg(easting, northing, zenith)
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def GetStation(self):
        msg = self.measureUnit.GetStationMsg()
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def SetEDMMode(self, mode):
        msg = self.measureUnit.SetEDMModeMsg(mode)
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def GetEDMMode(self):
        msg = self.measureUnit.GetEDMModeMsg()
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def SetOri(self, ori):
        clMsg = self.measureUnit.ClearDistanceMsg()
        ans = self.measureInterf.Send(clMsg)
        errorCode = self.measureUnit.Result(clMsg, ans)
        if errorCode['errorCode'] != 0:
            return errorCode
        msg = self.measureUnit.SetOriMsg()
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def SetRCS(self, rcs):
        msg = self.measureUnit.SetRCSMsg(rcs)
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def Move(self, hz, v, units='RAD', atr=0):
        msg = self.measureUnit.MoveMsg(hz, v, units, atr)
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def Measure(self, prg = 1, wait = 12000, incl = 0):
        msg = self.measureUnit.MeasureMsg(prg, wait, incl)
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def MeasureDistAng(self):
        msg = self.measureUnit.MeasureDistAngMsg()
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def Coords(self, wait = 1000, incl = 0):
        clMsg = self.measureUnit.ClearDistanceMsg()
        ans = self.measureInterf.Send(clMsg)
        errorCode = self.measureUnit.Result(clMsg, ans)
        if errorCode['errorCode'] != 0:
            return errorCode
        msg = self.measureUnit.CoordsMsg(wait, incl)
        ans = self.measureInterf.Send(msg)
```

```
        return self.measureUnit.Result(msg, ans)

    def GetAngles(self):
        msg = self.measureUnit.GetAnglesMsg()
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def ClearDistance(self):
        msg = self.measureUnit.ClearDistanceMsg()
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def ChangeFace(self):
        msg = self.measureUnit.ChangeFaceMsg()
        ans = self.measureInterf.Send(msg)
        return self.measureUnit.Result(msg, ans)

    def MoveRel(hz_rel, v_rel, units='RAD', atr=0):
        #get the actual direction
        msg = self.measureUnit.GetAnglesMsg()
        ans = self.measureInterf.Send(msg)
        res = self.measureUnit.Result(msg,ans)
        hz = Angle(res['hz'],units)
        return 0



if __name__ == "__main__":
    from leicameasureunit import *
    from serialinterface import *
    mu = LeicaMeasureUnit("TCA 1800")
    iface = SerialInterface("rs-232", "COM4")
    ts = TotalStation("Leica", mu, iface)
    print (ts.Move(0,0))
```

Interface.py:

```
#!/usr/bin/env python
"""
    <p>Ulyxes - an open source project to drive total stations and
           publish observation results</p>
    <p>GPL v2.0 license</p>
    <p>Copyright (C) 2010-2013 Zoltan Siki <siki@agt.bme.hu></p>
    @author Zoltan Siki
    @author Daniel Moka
    @version 1.1
"""

class Interface(object):
    "base class for all interfaces"
    IF_OK = 0
    ERR_OPEN = 1
    ERR_WRITE = 2
    ERR_TIMEOUT = 3
    ERR_READ = 4
    def __init__(self, name = 'None'):
        self.name = name
        self.state = self.IF_OK

    def GetName(self):
        return self.name

    def GetState(self):
        return self.state

    def ClearState(self):
        self.state = self.IF_OK

    def Send(self, msg):
        pass

if __name__ == "__main__":
    a = Interface()
    print (a.GetName())
    print (a.GetState())
```

localinterface.py:

```
#!/usr/bin/env python

"""
only for testing purposes
using leica GeoCom commands
"""
from interface import *

class LocalInterface(Interface):
    def __init__(self, name = 'Local'):
        super(LocalInterface, self).__init__(name)
        self.atr = 0
    self.lock = 0
        self.edmmode = 0

    def Send(self, msg):
        id = int(msg.split(',')[1].split(':')[0])
        ans = '%R1P,0,0:0'
    if id == 9027:
            # move
            ans = '%R1P,0,0:0'
        elif id == 18005:
            # setatr
            self.atr = int(msg.split(':')[1])
            ans = '%R1P,0,0:0'
        elif id == 18006:
            # getatr
            ans = '%%R1P,0,0:0,%d' % self.atr
        elif id == 18007:
            # setlock
            self.lock = int(msg.split(':')[1])
            ans = '%R1P,0,0:0'
        elif id == 18008:
            # getlock
            ans = '%%R1P,0,0:0,%d' % self.lock
        elif id == 2020:
            # setedmmode
            self.edmmode = int(msg.split(':')[1])
            ans = '%R1P,0,0:0'
        elif id == 2021:
            # getedmmode
            ans = '%%R1P,0,0:0,%d' % self.edmmode
        return ans

if __name__ == "__main__":
    a = LocalInterface()
    print a.GetName()
    print a.GetState()
    print a.Send('%R1Q,9018:1')
```

serialinterface.py:

```
#!/usr/bin/env python
"""
    <p>Ulyxes - an open source project to drive total stations and
           publish observation results</p>
    <p>GPL v2.0 license</p>
    <p>Copyright (C) 2010-2013 Zoltan Siki <siki@agt.bme.hu></p>
    @author Zoltan Siki
    @author Daniel Moka
    @version 1.1
"""

from interface import *
import serial
import re

class SerialInterface(Interface):

    def __init__(self, name, port, baud=9600, byteSize=8,
parity=serial.PARITY_NONE, stop=1, timeout=12):
        self.state = self.IF_OK
        self.name = name
        # open serial port
        try:
            self.ser = serial.Serial(port, baud, byteSize, parity, stop,
timeout)
        except:
            self.state = self.ERR_OPEN

    def __del__(self):
        try:
            self.ser.close()
        except:
            pass

    def GetLine(self):
        # read answer till end of line
        ans = b''
        ch = b''
        while (ch != b'\n'):
            ch = b''
            try:
                ch = self.ser.read(1)
            except:
                self.state = self.ERR_READ
            if ch == b'':
                # timeout exit loop
                self.state = self.ERR_TIMEOUT
                break
            ans += ch
        # remove end of line
        ans = ans.strip(b'\r\n')
        return ans

    def PutLine(self, msg):
        ans = b''
    # do nothing if interface is in error state
        if self.state != self.IF_OK:
            return -1
        # add CR/LF to message end
```

```
        if (msg[-2:] != '\r\n'):
            msg += '\r\n'
        # remove special characters
        msg = msg.encode('ascii', 'ignore')
        # send message to serial interface
        try:
            self.ser.write(msg)
        except:
            self.state = self.ERR_WRITE
            return -1
        return 0

    def Send(self, msg):
        msglist = re.split("\|", msg)
        res = b""
        for m in msglist:
            if self.PutLine(m) == 0:
                res += self.GetLine() + b"|"
        if res.endswith(b"|"):
            res = res[:-1]
        return res

if __name__ == "__main__":
    a = SerialInterface('test', 'COM4')
    print (a.GetName())
    print (a.GetState())
    print (a.Send('%R1Q,2008:1,0'))
```