Automatizált mérőrendszer alkalmazása és továbbfejlesztése

Dr. Siki Zoltán
Holéczy Ernő
konzulens

Király Tamás
hallgató

Áttekintő

Szakdolgozatomban a geodéziai műszerek számítógépen keresztül történő vezérlésével, valamint az így nyert mérési adatok internetes megjelenítésével foglalkoztam. A dolgozat első felében ennek háttere kerül bemutatásra: a mozgásvizsgálatok áttekintése, a világszerte megvalósult különböző típusú munkák a svájci monitoring rendszertől kezdve az alaszkai vulkánkutatásig, valamint a későbbi munkánk során alkalmazott szoftverek, programnyelvek és műszerek kerülnek előtérbe. Szakdolgozatom második felében a kidolgozott alkalmazások, programok kerülnek bemutatásra, elsőként a GPS segítségével történő földmunkagép irányítása magassági értelemben, majd egy általunk kiépített monitoring rendszer és az ehhez kapcsolódó internetes oldal. Ennél a munkánál nem csak a már létrejött szenzor oldali programok kerültek fejlesztésre, hanem a hozzá kapcsolódó internetes adatfeldolgozást is teljes egészében aktualizáltam a jelenlegi feladathoz, ahol is 3D-s deformációk voltak a vizsgálat alanyai.

Summary

In my thesis I pay attention to the control of surveying instrument by a computer such as total station and GPS. Besides that we developed a web page, where we can analyze these measurements. In the first part we look through the professional background, then we represent a few worldwide realized jobs from a monitoring system from Switzerland to Alaska, where we can find a volcano monitoring system, finally we talk about the later used software, programming languages and instruments. In the second part we present our work in this subject. First of all a GPS system which is serve to guide a floating equipment by a DTM model. Finally we pay attention to the Ulyxes monitroing system. Here we don't just develope the server side programs to be ready to work in live operation, but the connected web page too. Here we had a website, which was perfect to analyze height deformation, but now we developed that to provide information for the user about 3D movement.

Tartalom

1. Bevezetés

A 2000-es évektől kezdődően – csak úgy, mint sok más területen – a geodéziában is egyre nagyobb teret kezdett hódítani magának az informatika és a digitális világ. Az adatok számítógépes feldolgozása több évtizedre nyúlik vissza, illetve műszereinknek mára már csak kisebb hányadát teszik ki a mechanika elvűek, ezeket többnyire mind felváltották a mindenapi geodéziában digitális elven működők (szintezők, mérőállomások). E műszereken többnyire a gyártó által készített rendszerek, programok futnak. Ezek fejlettségi szintjüktől függően, vagy csak az adatok tárolására alkalmasak vagy a legújabb generációk egyedei ilyen téren már bővíthetők, új a felhasználót segítő programok vásárolhatók meg a gyártótól, melyek egyre komplexebb feladatok megoldását képesek elvégezni (vonalszámítások, hálózati kiegyenlítések).

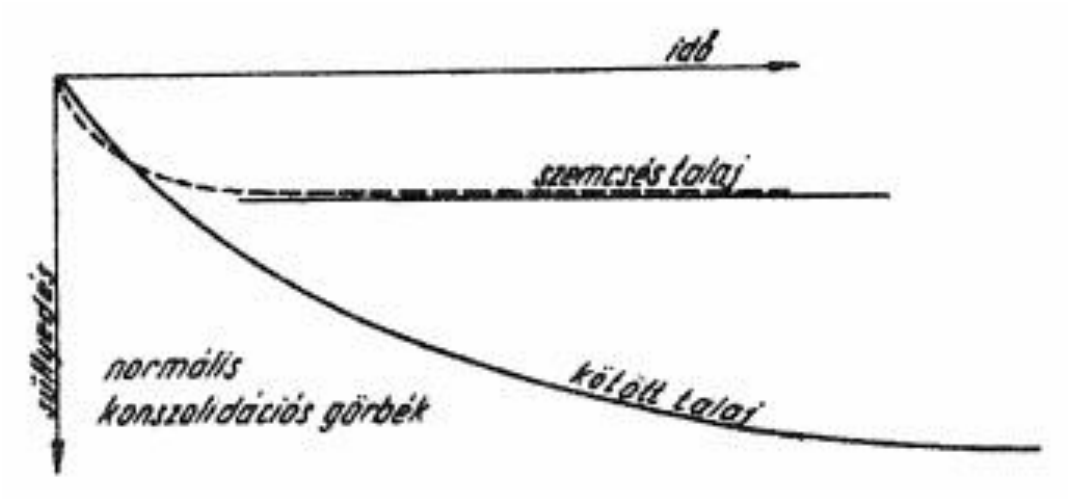
Napjainkban azonban egy új trend lép életbe a szabad forráskódú programok vezényletével. Mint az informatika oly sok területén, természetesen a geodéziában is megjelent arra az igény, hogy mindenki a maga számára legmegfelelőbb alkalmazásokat telepíthesse műszereire. Ez nem csak az anyagi vonzatokra vezethető vissza, ugyanakkor ez is nagy szerepet játszik, de fontos előnye még, hogy a mérnök legideálisabb szoftvert állíthatja elő a maga számára (vagy akár az interneten közzé teheti), illetve azokat az adott igények szerint fejlesztheti is. Ezek lehetnek kész programok, melyeket továbbfejleszthet a felhasználó, vagy teljességében önálló alkalmazások. Egyre nagyobb körben alakult ki az igény a geodéták között is, hogy saját maguk által készített/fejlesztett alkalmazásokat használhassanak nagyrészt mérőállomásaikon kisebb részt egyéb műszereiken vagy műszer csoportok összességén. Ezt a törekvést a műszergyártók is támogatják, azaz műszereikre lehetőség van új programok feltöltésére, vagy valamilyen számítógépes kapcsolaton (soros port-on, bluetooth-on) keresztül azok közeli vagy távirányítására. Az ilyen egyedi programok által olyan feladatok is sikeresen elvégezhetők, melyek korábban csak nagy anyagi és emberi ráfordítással voltak lehetségesek. Ezek közé sorolhatók a gyors lefutású mozgásvizsgálatok, munkagépek irányítása (fúrópajzs, finisher, földmunkagép, mezőgazdasági gépek), monitoring rendszerek, stb. Természetesen a feladathoz nem csak az adatok, mérési eredmények begyűjtése tartozik hozzá, hanem azok feldolgozása is fontos szerepet játszik, azokból a megfelelő információk kinyerése. Mozgásvizsgálatok során a különböző deformációk vizsgálata, azok közötti összefüggések kimutatása (auto- és keresztkorrelációk), riasztási rendszer kialakítása túlzott mértékű süllyedések esetére; fúrópajzsok, munkagépek

irányítása esetén a pillanatnyi pozíció és irány alapján megfelelő korrekciók számítása, a vezetőt támogató instrukciók közlése. Diplomamunkámban ezen lehetőségekről nyújtok áttekintést, felhasználási lehetőségeket. Dolgozatom részeként bemutatásra kerül egy egyszerűbb GPS alkalmazás, mely földmunkagép magassági értelmű irányítását szolgálhatja, illetve a BME Általános és Felsőgeodézia tanszéken Dr. Siki Zoltán által létrehozott Ulyxes mozgásvizsgálati monitoring rendszer továbbfejlesztése. Ennek keretében a már elkészült programrészeket kiegészítettük, átalakítottuk, illetve újakat is létrehoztunk azzal a céllal, hogy a szenzor és a szerver oldali rendszeregyüttes üzemszerűen alkalmas legyen 3 koordinátatengely irányú deformációk mérésére és az így nyert adatok internetes közzétételére.

2. Helyzetkép bemutatása

2.1. Mozgásvizsgálatok áttekintése

Az építőiparban, a legtöbb esetben *(Detrekői és Ódor, 1998)* magas építmények, lakóházak, hidak, töltések, alagutak, gátak, bányák, erőművek stb. esetén szükséges mozgásvizsgálatot végezni. Általában az építkezés befejeztével kezdődnek meg a mérések és attól kezdve akár a szerkezet teljes élettartama alatt zajlanak (pl.: veszélyes üzemek). Az építmény típusától és az altalaj minőségétől függően különböző hosszúságú mérési időtartamra lehet számítani. Egy családiház építésénél általában nem merül fel a süllyedésvizsgálat kérdése, míg több 10 emeletes házak vagy hidak esetén a konszolidációs időszakban szükséges méréseket végezni. Egy atomerőmű esetén a teljes üzemidő alatt folyamatosan bizonyos időközönként vizsgálni kell az esetleges süllyedéseket. Fontos tényező a mozgásvizsgálatok során a talajtípus. Egy kötött típusú altalaj esetén *(Farkas, 2004)* a tömörödés időtartama sokkal elnyújtottabb, míg szemcsés talajoknál a részecskék átrendeződése pár nap vagy hét alatt végbemegy (1. ábra), ezért már az építkezés során jelentős mozgások lehetnek.



1. ábra: Konszolidációs görbék (Farkas, 2004)

Az előzőek alapján is látható, hogy sok esetben szükséges mozgásvizsgálati méréseket végrehajtani. Ezek különböző okokra vezethetők vissza *(Farkas, 2004)*:

- Az elmozdulások előrejelzése: Ez a leggyakrabban előforduló cél, az esetleges balesetek, katasztrófák megelőzése hidak, gátak, stb. esetén.

- Már bekövetkezett mozgások esetén is szükség lehet mérésekre a süllyedés okainak kiderítésére.
- Különböző épületek tervezését is elősegíthetik a mérések. Elkészült szerkezeteken végzett mérések alapján az új szerkezeti elemek szilárdságtani paraméterei jobban meghatározhatók.
- Műszaki használatbavétel miatt is végeznek ilyen méréseket, pl. hidak próbaterhelésénél.
- Tudományos kutatások.
- Igazságügyi szakértés.

Különböző feltételezésekkel, megkötésekkel (modell alkalmazása, alappontok mozdulatlansága, pontosság) kell élnünk vizsgálataink során (*Krauter, 2007), (Detrekői és Ódor, 1998)*. Itt is, mint minden más geodézia mérés során a szerkezetet valamilyen modellel helyettesítjük. Fontos, hogy ez a modell ne térjen el nagymértékben a valóságtól, de legyen annyira egyszerű, hogy a számításainkat könnyen végre tudjuk hajtani. Ahhoz, hogy az objektum mozgását jól tudjuk követni, fontos, hogy megfelelő helyen helyezzünk el a vizsgálati pontokat az építményen. Ezekre a pontokra végzünk méréseket, tehát az épület egészét le kell fedniük és az esetleges mozgásokat vagy a szerkezet deformációit jól kell jellemezniük.

Méréseinkhez elengedhetetlen, hogy mozdulatlan alappontok álljanak a rendelkezésünkre. Pontatlan következtetéseket vonhatnánk le számításainkból, ha az alappontok mozgása is terhelné a vizsgálati pontok mozgását. Ennek a kiszűrésére több lehetőség van: A mozgási zónán kívül helyezzük el az alappontokat valamilyen mozdulatlan kőzetben, illetve egy helyre több pontot is állandósítunk és bizonyos időközönként egymáshoz viszonyítva ellenőrizzük azok mozdulatlanságát.

A várható elmozdulások alapján szükséges a mérési pontosság előzetes meghatározása. Természetesen figyelembe kell venni a gazdaságosságot is. Ezen tényezők vizsgálata alapján kell a megfelelő mérési technológiát is kiválasztani. A különböző mérési időpontok is megválaszthatók a mozgás várható lefolyása alapján, például optimálisan igazítva a süllyedések várható jelleggörbéjéhez.

A vizsgálat során feltételezzük még, hogy egy-egy mérési időtartam alatt az objektum mozdulatlan, tehát mérési időpontokról beszélünk, azonban ez nem teljesül minden esetben, ilyenkor érdemes folyamatos mérési módszereket alkalmazni.

Ezek alapján a mozgásvizsgálatok általános sémája a következő: különböző időpontokban a megfelelő technológiával méréseket végeznek az objektumon elhelyezett vizsgálati

pontokra, és meghatározzuk azok bizonyos paramétereit, például a koordinátáit. Ha a különböző időpontokban mért adatokból eltérő koordinátákat kapunk, akkor valószínűség számítási módszerekkel meghatározható, hogy a koordinátaeltéréseket ténylegesen mozgás/deformáció okozza vagy csak mérési hiba.

Mint a geodézia legtöbb területén, itt is nagyrészt szétválasztjuk a vízszintes és magassági értelmű mozgásokat. A mozgásokat azok térbeli iránya alapján csoportosíthatjuk *(Farkas, 2004)*. Az építkezések túlnyomó részén magassági értelmű mozgások következnek be, ezeket nevezzük süllyedésnek. Természetesen lehetséges negatív előjelű süllyedés is, azaz emelkedés, pl. a víz felhajtóerejéből következően. Fontos megjegyezni, hogy az épületkárok többsége nem a túlzott mértékű süllyedés miatt következik be (abszolút süllyedés), hanem az egyenlőtlen süllyedések miatt (relatív süllyedés). Ekkor az épület egyik része nagyobb mértékben süllyed, ennek következtében feszültség keletkezik az egyes szerkezeti elemekben és repedések, törések jelennek meg az épületen.

A vízszintes irányú elmozdulásoknál hasonlóan beszélhetünk abszolút és relatív mozgásról. Elcsúszásról beszélünk, ha az adott objektum minden pontja ugyan olyan mértékben mozdul el egy irányba. Ha az épület egyik része nagyobb mértékben mozdul el, mint a másik, vagy a mozgás iránya különbözik, akkor elfordulásról illetve deformációról beszélhetünk.

Megemlítendő még az összenyomódás és a nyúlás is külön csoportként. Ez a nem tökéletesen merev testekre jellemző deformáció. Ekkor a szerkezeteknek valamilyen mérete, nem pedig helyzete változik meg. Detektálása leginkább az igénybevételek meghatározása miatt fontos a statikusok számára, mivel azok közvetlenül nem mérhetőek, csak a méretváltozásokból következtethetünk rájuk a Hook törvény alapján:

$$\sigma=E*\varepsilon \qquad (1.1)$$

ahol σ a szerkezetben keletkező feszültség, E a rugalmassági modulus (adott szerkezetre jellemző állandó), ε pedig a nyúlás mértéke.

Egy tökéletesen merev objektum mozgása összesen 6 paraméterrel jellemezhető: 3 elmozdulás vektor (süllyedés és síkban való eltolódás x, y tengely mentén) és 3 koordinátatengely körüli elfordulás. Nem tökéletesen merev test esetén hasonlóan jelentkeznek ezek a mozgásvektorok az egész testre vonatkozóan, de megjelenik a szerkezeti elemek deformációja is. Ezeket a méretváltozásokat, ha csak egy irány mentén történik

deformáció, a nyúlás/összenyomódás mértékének megadásával jellemezhetjük, komplexebb torzulások esetén az objektum egyes pontjainak relatív elmozdulásaival adhatjuk meg.

A méréseket a teljesség igénye nélkül végezhetjük egyenesre méréssel, háromszögeléssel, vetítéssel, sokszögeléssel, GNSS eljárással vagy folyamatos méréssekkel, melyek közé sorolhatók az induktív adó, helyzetérzékelő dióda, nyúlásmérő bélyeg, inga, elektromos libella. A folyamatos mérések módszere abban különbözik a hagyományos metódusoktól, hogy itt nem különíthetők el különböző mérési időpontok, az észlelés folyamatosan történik. A robot mérőállomásokkal történő mérés valahol a hagyományos és a folyamatos mérési technológiák között helyezkedik el, mivel nem képes olyan szintű folyamatos mérésekre, mint egy nyúlásmérő bélyeg, ellenben jóval gyorsabb mint például az egyenesre mérés (Körülbelül 1 Hz-es mérési sebesség érhető el).

Süllyedések, elmozdulások különböző okok miatt jelentkezhetnek az építményeken *(Farkas 2004)*:

- statikus terhek
- dinamikus terhek és egyéb hatások (pl. szél)
- talajban lévő víz hatása, ingadozása
- felszíni víztest mozgása
- aláüregelődés (visszavezethető a víz hatására)
- talajcsúszás
- kémiai átalakulások (duzzadás, kioldás)
- hőmérsékleti hatások

Ezen okok mellett az altalaj állapota is nagymértékben befolyásolja a mozgások lefolyását. Egy szemcsés talajra helyezett statikus teher gyorsan konszolidálódik, míg egy kötött agyagos talaj jelleggörbéje sokkal elnyújtottabb. Minden talajban másképp viselkedik a víz is, e miatt rengeteg épületkár történik (pl. a lösz víz hatására nagymértékben roskad)

A fő épületkárok az egyenlőtlen mozgásokból adódnak. Ennek fő okai:

a. egyenlőtlen talajrétegződés
b. egyenlőtlen terhelés
c. különböző alapozási mód
d. feszültséghalmozódás
e. talaj oldalkitérése
f. meglévő épület egyensúlyának megbontása.

A vízszintes irányú elmozdulások leginkább a víz hatására vezethetők vissza. A talajvízszint változás hatására megbomlik a talaj szerkezete és emiatt csúszásveszély alakul ki. Folyóknak is komoly hatása lehet támfalakra, a folyamatos dinamikus erőhatás miatt meggyengülhet a szerkezet. Erre egy igen szemléletes példa az alant látható kulcsi löszfal csuszamlás (2. ábra).

2. ábra: Kulcsi löszfal
(http://www.dunaharasztiapro.hu/news.php?extend.305.8)

2.2. A legújabb típusú monitoring rendszerek

2.2.1. Vasúti híd monitoring rendszere (Kirchtobel - Svájc)

A St. Gallen és Arth-Goldau között fejlesztett vasúti hálózat építése során merült fel az igény 2011 végén a Leica cég által fejlesztett monitoring rendszer alkalmazására. Az újonnan épülő vasúti híd egy immáron 100 éve álló hasonló funkcióval bíró híd (3. ábra) közvetlen közelébe került megépítésre, mely természetes kőből épült és pillérei mára már nem tekinthetők kellő mértékben stabilnak. Ennek következtében az építkezés során nem várt elmozdulások is felléphettek, melynek előrejelzéséhez egy mozgásvizsgálati rendszer került kiépítésre. A feladatot nagyban nehezítette, hogy a hídon állandó vonatforgalom haladt át mindkét irányból, így nem csak magának a szerkezetnek a mozgását kellett figyelemmel követni, hanem a rajta fekvő vasúti sínpárokét is, hiszen azok deformációja balesethez vezethetett volna.

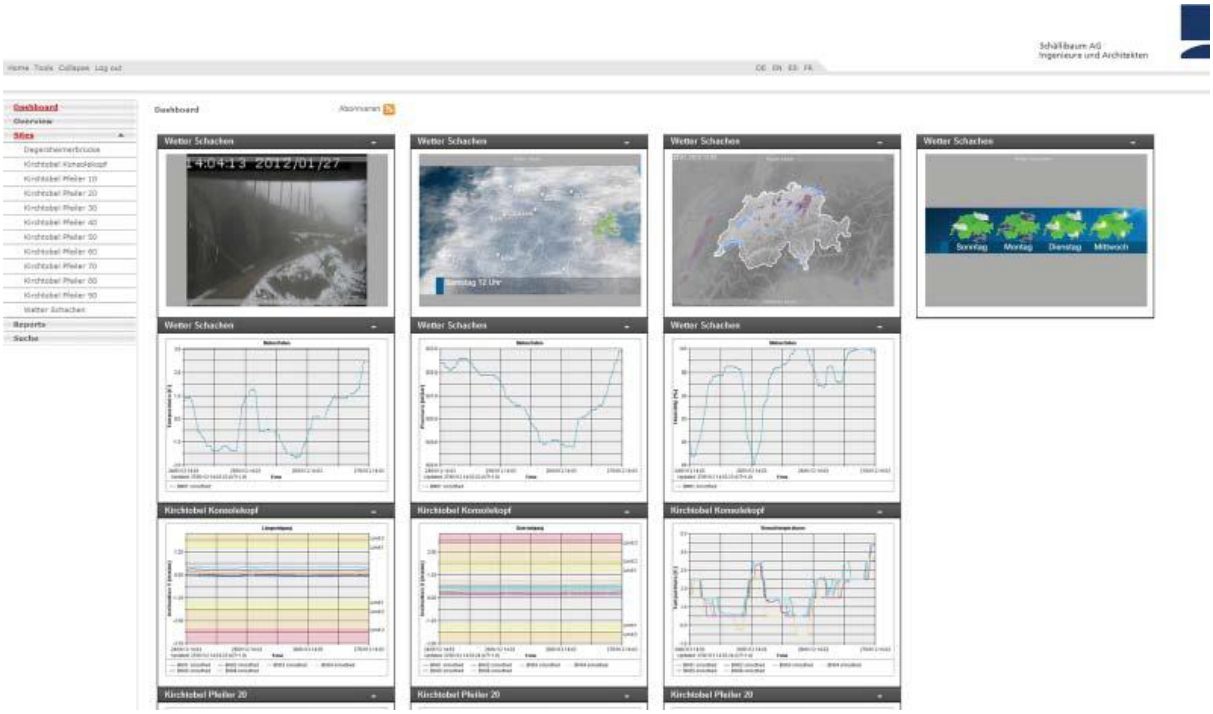
3. ábra: Kirchtobel-i híd, (Leica-Kirchtobel, 2011)

A feladat megoldásához egy Leica TC2003 és egy TC1800-as műszert alkalmaztak a híd egy-egy végén, melyek távolság mérési pontossága $\pm$ 1+1,5 ppm. A viadukton összesen 81 prizmát helyeztek el, mely pontokkal a hidat magát és annak mozgását is jól modellezhették, valamint a kihelyezett mérőállomások mozdulatlanságának ellenőrzéséhez, pozícióik esetleges korrekciójához még 12 mozdulatlan beton alapzaton elhelyezett alappontot is létesítettek. Magukhoz az álláspontokhoz is speciális pilléreket készítettek, melyek fagy és

egyéb külső hatás elleni védelmét egy védőburkolat képezte. A mérőállomások ezen pillérek magjához lettek rögzítve (4. ábra). Mivel a rossz körülmények, mint a hideg, téli köd, maga az új híd építési munkálatai, zavaróan hathatnak az optikai mérések pontosságára illetve magára az összeláthatóságra is, így összesen 6 darab dőlésmérőt (Leica Nivel220) is elhelyeztek a híd végein, így figyelemmel követve a hosszirányú és kereszt irányú dőléseket is. Ezek segítségével fölös méréseket is előállítottak a mérőállomások mérései mellé. A mérések javításához továbbá meteorológiai szenzorokat is kihelyeztek, így lehetőséget teremtve a távolságmérések korrekciójához, valamint ez által az új híd kivitelezéséhez is használható információkat állítottak elő (a betonozás egyes fázisaihoz).

4. ábra: A mérőállomások pillérei, (Leica-Kirchtobel, 2011)

A központi és az egyes mérő szenzorok között a kapcsolatot vezetékesen illetve a nélkül oldották meg. A híd elhelyezkedése miatt az energiaellátást üzemanyagcellákon és akkumulátorokon keresztül kellett megoldani.



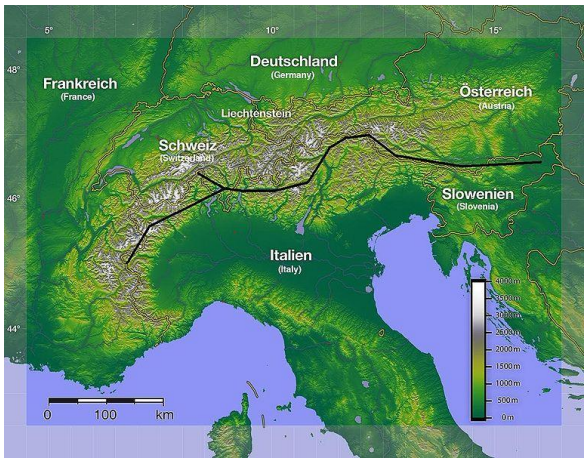
5. ábra: Leica GeoMoS adatsorok, (Leica-Kirchtobel, 2011)

Az építkezés megkezdése előtt szükség volt egy kiindulási nulla állapot meghatározására is, amely mérések lehetőséget nyújtottak a monitoring rendszer finomhangolásához, illetve az így kapott adatok információt nyújtottak a még mozdulatlannak tekinthető híd viselkedéséről.

A mért adatokat egy központi állomáson (Leica ComBox 20) kerültek összegyűjtésre ahonnan mobil interneten keresztül kerültek továbbításra a kontroll központ (Leica GeoMoS) felé. Ez a központ végezte el a szükséges számításokat és a meghatározott küszöbértékek meghaladása esetén riasztást küldött SMS-ben és e-mailben a megfelelő szakemberek számára. A Leica GeoMoS Web rendszere (5. ábra) lehetőséget nyújtott a mérési adatsorok interneten keresztüli vizsgálatához/elemzéséhez is. (Leica-Kirchtobel, 2011)

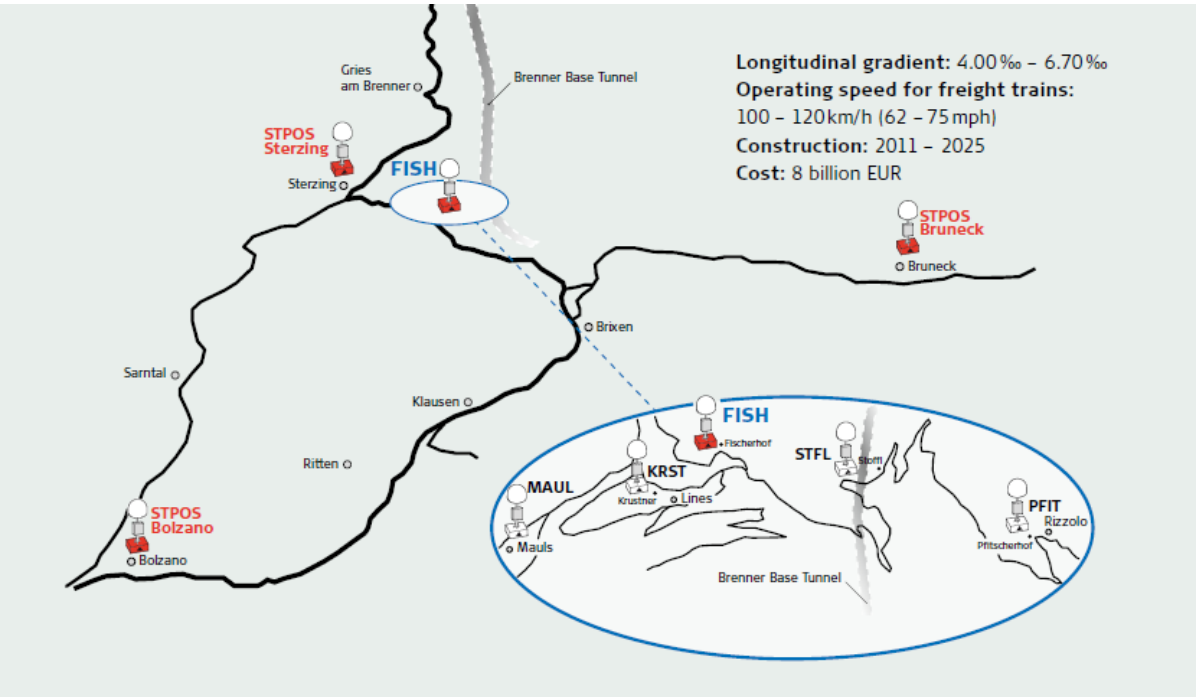
2.2.2. GNSS monitoring rendszer a Brenner-hágón

A Brenner alatt húzódó 55 km hosszúságú alagút, mely a Berlin-München-Verona-Bolona-Palermo között húzódó nagysebességű vasútvonalnak a része, mintegy 13 éven belül (2025-re) készül el teljesen. A Leica cég műszerei nem csak az alagút fúrás irányításában vesznek részt, hanem a hegység instabil zónájának mozgásvizsgálatában is. Az alagúthoz tartozó több feltáró járat már építés alatt áll, ezek később a fő alagútjáratok alatt fognak futni biztonsági menekülő járatokként. Az alagút egy része egy geológiailag aktív területen halad át (6. ábra), ahol a Déli-Alpok lemeze találkozik az ausztriai Közép-keleti Alpok lemezével (Periadriatic Seam).



6. ábra: Periadriatic Seam, (http://en.wikipedia.org/wiki/File:Alps-Relief_02de%2BPeriadriatic.jpg)

Az építkezés ezen része az egész alagút kivitelezése során nagy körültekintéssel volt kezelve, hiszen nagyobb mozgásokra lehetett itt számítani. A fő kérdést az jelentette, hogy a fúrás során a felszínen tapasztalhatók-e deformációk, és ha igen, azokról azonnali riasztás lehessen küldhető. A mozgási zóna 2 km^2 területű, tehát csak GNSS megoldás jöhetett szóba ilyen pontossági elvárások mellett. Ezen a területen belül helyeztek el 5 állomást, melyek közül 1 referencia állomásként volt tekinthető (FISH). A pontok elhelyezése során fontosnak tekintették a rövid bázisvonalak alkalmazását, tudatában annak, hogy ezzel potenciálisan a referencia vevőt is a deformációs zónán belül létesítik. E hibaforrás kiküszöbölésének céljából



7. ábra: A kiépített hálózat, (Brenner, 2012)

létesítettek még három állomást a referencia állomástól 10-40 km-es távolságokban, ezzel egy hierarchikus elrendezést kialakítva (7. ábra).

4 állomásnál sikerült megoldani a 230V-os tápellátást az 5-nél (Stoffl) akkumulátoros megoldást kellett alkalmazni, fotovoltaikus napelemes megoldással és tartalék akkumulátorokkal. A bázisvonalakat egy 48 órás mérési periódus során határozták meg, ahol ugyanaz a műhold konstelláció többször is előfordult. A mérésekhez 1 db Leica GMX902GG és 4 db Leica GMX901 típusú (8. ábra) vevőt alkalmaztak, az adatáramlás GPRS/UMTS-en keresztül történt. A méréseket egy hónappal később megismételték.

8. ábra: Az egyik kitelepített vevő, (Brenner, 2012)

A területen található Maulus településen a GNSS vevőkkel összhangban dolgozik egy mérőállomás is megfelelő számú prizmával, mely a felszíni mozgásokat figyeli a faluban. A rendszer 2012-től számítva várhatóan 3

éven keresztül működik, amíg az építkezés továbbhalad a veszélyeztetett zónából. (Brenner, 2012)

2.2.3. Lézerszkennerrel a bányák biztonságáért

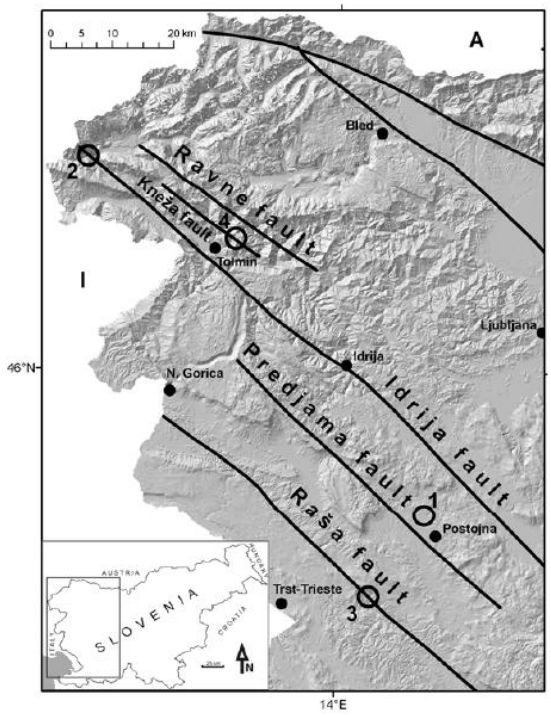
Egy igazán izgalmasnak tűnő alkalmazási terület illetve műszer párosát mutatja a Kongói demokratikus köztársaságban egy nyílt színi bányában az ENRC által integrált monitoring rendszer a megfelelő biztonság előteremtése érdekében. Az ENRC egy világszinten 70.000 embert foglalkoztató vállalat, mely bányákat, feldolgozó üzemeket, energia ipari létesítményeket telepített Kazahsztánban, Oroszországban, Brazíliában és Afrikában.

A Kongóban kitelepített 3D-s laser scanner segítségével biztosítható a bánya falainak mozgása esetén annak detektálása és így a szükséges intézkedések meghozhatók. Ezzel a megoldással nem csak a deformáció vizsgálat végezhető, hanem a bányákban szükséges geodéziai felmérések is költséghatékonyabbá tehetők e rendszerrel valamint még produktívabb kitermelés is elérhető munkaerő-megtakarítás mellett. A monitoring rendszer várhatóan főként esős időben fog hatékonynak bizonyulni, amikor is a vízerek földcsuszamlásokhoz vezethetnek.

9. ábra: RIEGL VZ-1000 laser scanner, (LaserScanner, 2013)

A rendszer egy RIEGL VZ-1000-es laser scanner (9. ábra) és SiteMonitor szoftver házasításából alakult ki. A VZ-1000-es egy nagy sebességű laser scanner, széles látószöggel. A műszer egy egyedi echo-s digitalizálással és online hullámforma elemzéssel képes elérni az 5 mm-es pontosságot egészen 1.400 m-es távolságig akár kedvezőtlen időjárási körülmények között is mintegy 122.000 pontot megmérve egy másodperc alatt 100x360°-os tartományban. A SiteMonitor monitoring rendszert kifejezetten nyílt színi bányákban alkalmazzák állékonyság vizsgálatra. (LaserScanner, 2013)

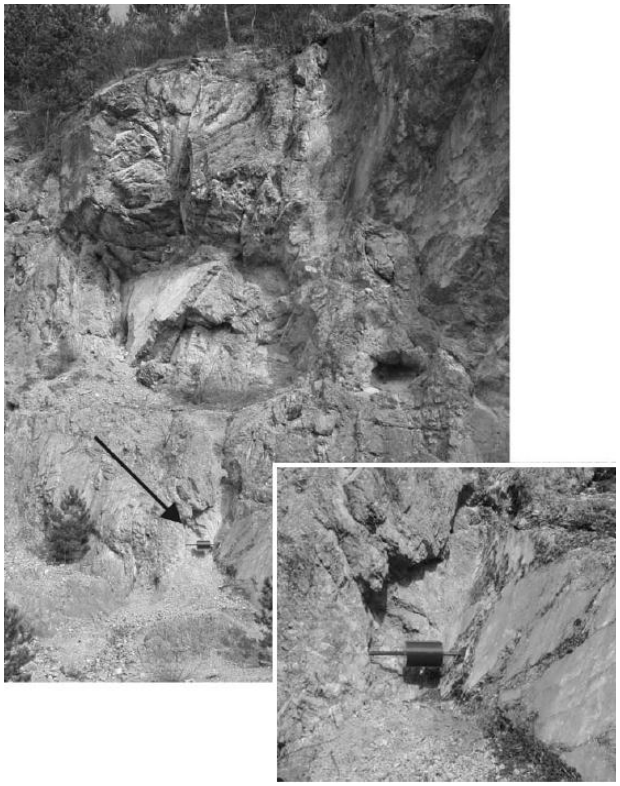
2.2.4. Mikro-deformációs monitoring rendszer Szlovéniában



10. ábra: A vetődések és a kihelyezett mérőeszközök, (Andrej Gosar et al. 2007)

Egy kisebb és viszonylag szerényebb kivitelezésű rendszer épült ki Szlovéniában. Az ország három tektonikusan aktív lemez határán fekszik: az Alpok, a Dinári-hegység és a Panon-medence határán, így földrengésekben mérsékelten veszélyeztetett területnek tekinthető. A legnagyobb észlelt földrengés 1511-ban M=6.8-as erősségű volt.

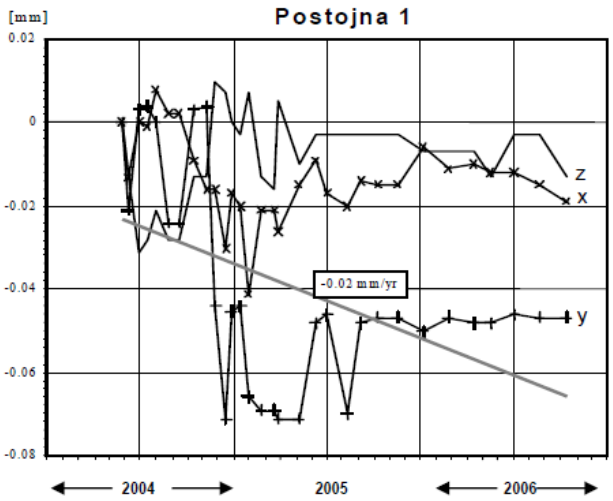
A méréseket TM 71 típusú (10. ábra) nyúlásmérőkkel végezték, kettőt a Postojna barlangban, egyet-egyet a Raša illetve a Idrija vetődésben valamint egyet a Kneža vetődésben. A TM 71-es egy mechanikus nyúlásmérő, melyet keskeny repedések közti távolság mérésére terveztek. A műszer a mechanikus interferencia elvén működik, tehát az elmozdulások interferencia mintákként kerülnek rögzítésre 3D-ben 0,05-0,0125 mm-es pontossági szinten. Nagy előnye a műszernek, hogy mechanikus elven működik, így nem szükséges hozzá elektromos ellátás, így rossz körülmények között is stabilan működik. A változó hőmérséklet kis mértékben hatással lehet a mérésekre, de például a barlangokban elhelyezett méréseknél ez nem lép fel, hiszen azok hőmérséklete stabil. A kutatók észlelték, hogy a mérésekre hatással van a barlangokban tapasztalható radon kisugárzás is, így ezek szintjét is mérni kezdték a



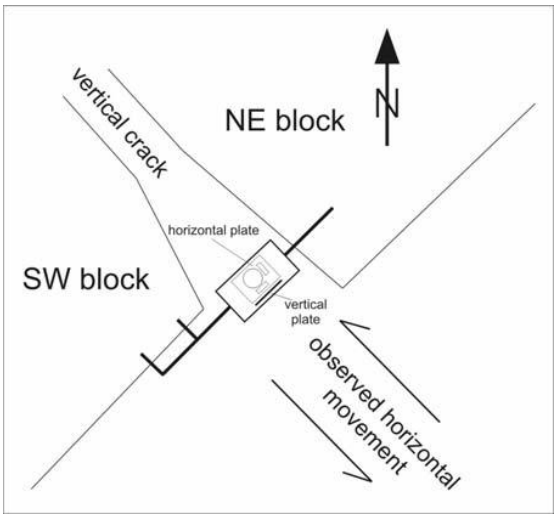
11. ábra: TM 71 nyúlásmérő, (Andrej Gosar et al. 2007)

mérések javításához.

A mérések alapján a Postojna barlangban évente 0,02 mm-es, az Idrija vetődésnél évente 0,54 mm-es, míg a Raša vetődésnél 0,3 mm-es eltolódások voltak kimutathatók. A műszerek



12. ábra: Mérési eredmények, (Andrej Gosar et al. 2007)



13. ábra: A nyúlásmérők rögzítésére egy példa, (Andrej Gosar et al. 2007)

hatékonyságát mutatja, hogy a Postojna barlangban elhelyezett mindkét műszer egyértelműen észlelt egy tőlük hetven kilométerre kipattant földrengést.

Fontos megjegyezni, hogy ilyen mértékű elmozdulások GNSS eljárással nem észlelhetők. A megoldás hibájaként vethető fel, hogy a műszerek, mint az a képeken is látható (11. ábra, 13. ábra), csak a vetődések felszíni kibukkanásainál rögzíthetők. A kapott végeredményekről egy példa látható a 12. ábrán. (Andrej Gosar et al. 2007)

2.2.5. A modern vulkánkutatás

A Földünk belső mozgásait jól tükrözi az 1538-ban Olaszország egy falujában tapasztaltak, melyek szerint a talajszint 48 óra alatt 4 méterrel emelkedett meg, melynek következtében a partvonal 400 m-el vonult visszább. A korábbi évszázadok során folyamatosan tapasztalható volt a terület felpúpozódása, mely végül ebben az évben vezetett kitöréshez. Hasonló mozgások tapasztalhatók világszerte, például a Galapagos szigeteken vagy Pápua Új Guineában (14. ábra) még napjainkban is.

Az új technológiai eszközök a vulkán kutatók számára is új lehetőségeket nyújtanak a kitörések előtt, közben és után is információk gyűjtéséhez. A vulkánok űrből történő mozgásvizsgálata, a magma mozgásának követése és a kitörések előrejelzése nagy fejlődésen

ment át az elmúlt években. A kommunikációs lehetőségek ugrásszerű fejlődésével manapság sokkal gyorsabb a riasztások kiadása, illetve a megfelelő mentőszervezetek is értesíthetők. A megfelelő információk kiküldéséhez azonban a megfelelő monitoring rendszer kiépítésére van szükség. Ehhez a szeizmológia, a geodézia és a geokémia hármasának kell együttműködnie.

14. ábra: Pápua Új Guinea, a Rabaul Caldera az űrből (Daniel Dzurisin, 2007)

A szeizmológiai megoldások viszonylag könnyen elérhetők bárki számára. 6-8 rövidtávú szeizmométer a vizsgálni kívánt terület körül kihelyezve, illetve a várható epicentrum közvetlen közelében igen jó képet nyújthat a szakemberek számára a föld mélyén zajló történésekről. Ezekkel még az igen kis magnitúdójú ($M < 1$) rengések is érzékelhetők néhány kilométeres körzetben. Miközben a vulkán belsejében akár naponta több méteres változások következnek be, 2-3 km-es távolságban ezek már nem mutathatók ki geodéziai eszközökkel, ellenben az apró földrengések mind érzékelhetők ezekkel a szenzorokkal.

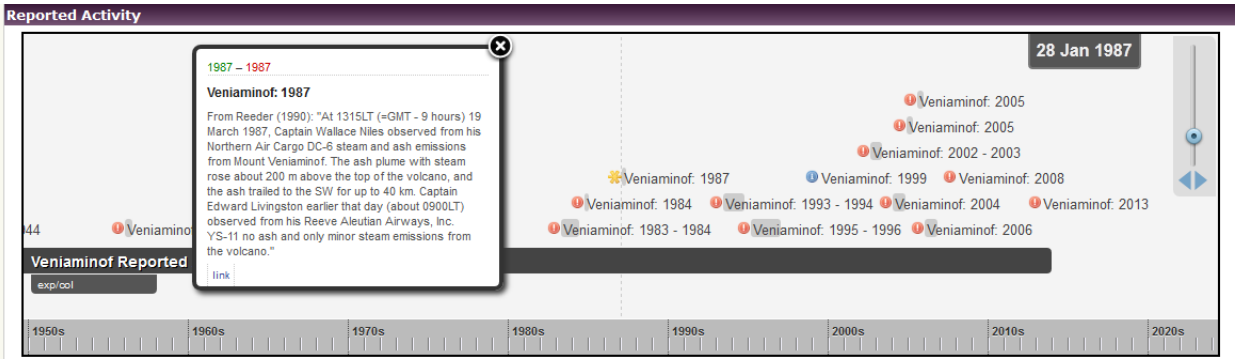
Geokémiai műszerekkel főként a vulkánok közelében a levegő, a vizek tartalmát a kitörő gázokat vizsgálják, hogy mely a vulkánkitörések előtt tapasztalható elemek, molekulák jelennek meg illetve halmozódnak fel nagymértékben.

Kutatások végezhetők graviméterekkel is, melyekkel a föld alatt található bármilyen változást kimutathatunk. A fő gondot természetesen az jelenti, hogy az egyes változásokat, mint a talajvízszint emelkedés, a tektonikus lemezek mozgása, illetve egyéb a magma mozgásából következő mozgások annak akár hőmérsékletének megváltozása is szétválaszthatók legyenek.

SP (Elektromos ön-potenciál) méréssel is beszerezhetők szükséges információk. E mérés keretében két egymáshoz és a földhöz is csatlakoztatott elektródát helyeznek el különböző pozíciókban. A hely változtatásának függvényében időnként kisebb mértékű potenciál-változás érzékelhető a talajban illetve a talajvízben található eltérő töltések miatt.

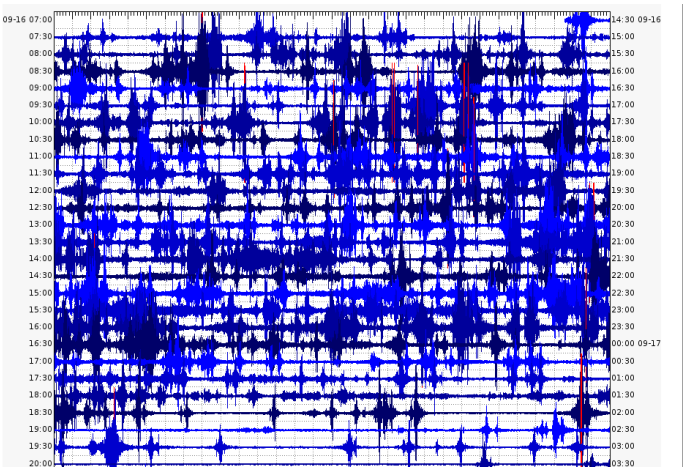
15. ábra: Alaszkai vulkánok, (AlaskaVolcano, 2013)

A kutatók a hagyományos geodéziai technológiák mellett alkalmaznak folyamatos szenzoros méréseket is, mint a dőlésmérők és nyúlásmérők, GPS rendszereket, InSAR műholdas megfigyeléseket valamint fotogrammetriai megoldásokat. Természetesen a hálózatok megfelelő kiépítése itt is nagy körültekintést igényel. (Daniel Dzurisin, 2007)



16. ábra: A vulkánról rendelkezésre álló idősor, (AlaskaVolcano, 2013)

Egy szemléletes példa a vulkánoknál alkalmazott monitoring rendszerekről az Alaszkában az USGS által üzemeltetett, mely bárki számára elérhető az interneten keresztül. A térképen (15. ábra) kiválasztva a számunkra érdekes vulkánt arról rengeteg adatot kaphatunk meg, mint például általános leírást, képeket, térképeket,



17. ábra: Egy szenzor adatsora, (AlaskaVolcano, 2013)

korábbi aktivitásokról leírásokat (még a történelmi időkből is) (16. ábra) illetve a körülötte elhelyezett mérő szenzorok idősorait is megtekinthetjük (17. ábra).

Hasonló rendszerek találhatók még Kaliforniában, Hawaii-in, A Cascade-hegységben és a Yellowstone nemzeti parkban is mindenki számára szabadon hozzáférhetően. (AlaskaVolcano, 2013)

2.3. Szoftverkörnyezet

Munkánk során – mely a 3. fejezettől kezdődően kerül bemutatásra – különböző programozási nyelveket és szoftvereket használtunk illetve az azok közötti kapcsolatot is meg kellett teremtenünk, a következőkben ezek kerülnek bemutatásra.

2.3.1. GeoCom

A GeoCom a Leica mérőállomások által alkalmazott RPC (Remote Procedure Call), illetve ASCII protokoll alapú vezérlő nyelv. RPC tehát egy szerver és egy kliens között teremt kapcsolatot, ASCII protokoll azaz soronként tagolt parancsokat tud értelmezni.

Általánosságban megkülönböztetünk meglehetősen kezdetleges ASCII protokollokat illetve magas szintű függvényeket (high-level function call interface). Az egyszerű ASCII protokoll kérésekből és válaszokból áll össze, tehát az alkalmazás elküld egy parancsot, melyre aztán választ vár és azt dekódolja; az utóbbi függvényekkel operál. Fontos megjegyezni, hogy egy kérés/válasz párost nem lehet megszakítani egy másikkal.

A következőkben látható, hogy hogy is néz ki egy ASCII alapú kérés:

```
[<LF>]%R1Q,<RPC>[,<TrId>]:[<P0>][,<P1>,...]<Term>
```

ahol is	<LF>	törli a fogadó bufferét
	%R1Q	GeoCom kérési típus 1-es
	<RPC>	Az RPC kód azonosító száma 0 és 65535 között
	<TrId>	Szabadon választható ID: 1-7-ig növekszik, a válasznál is ugyan az szerepel
	:	Elválasztó jel a bevezető protokoll és az azt követő paraméterek között
	<P0>, <P1>	A paraméterek
	<Term>	Lezárás (alapesetben CR/LF)

Valamint a kapott válasz:

```
%R1P,<RC_COM>[,<TrId>]:<RC>[,<P0>,<P1>, ...]<Term>
```

ahol is	%R1P	GeoCom válasz típus 1-es
	<RC_COM>	A sikeres kommunikációt jelöli (0 = sikeres)
	<TrId>	Ugyan az, mint a kérésnél
	:	Elválasztó jel a bevezető protokoll és az azt követő paraméterek között
	<RC>	A sikeres válaszadást jelöli (0 = sikeres)
	<P0>, <P1>	A paraméterek
	<Term>	Lezárás (alapesetben CR/LF)

Az ASCII protokoll különböző adattípusokat különböztet meg, mint a logikai, szöveg, különböző számtípusok (byte 0→255, long -2^{31}→2^{31}-1, stb.) stb, valamint hozzájuk a megfelelő SI mértékegységek is hozzárendelhetők. (GeoCom Reference Manual)

Esetünkben a kéréseket mi magunk adjuk ki számítógépünkről és a szenzorunktól várjuk a rá adott választ. A későbbiekben bemutatásra kerülő Ulyxes rendszerben a GeoCom csak a „háttérben" jelenik meg, azaz a rendszerhez kapcsolódó TclAPI parancsainak hátterében ilyen utasításokat küldünk a mérőállomás felé és a kapott válaszokat értelmezzük.

2.3.2. Tcl (Tool Command Language)

A Tcl egy nyílt forráskódú dinamikus programozási nyelv. Igen széles felhasználói körrel rendelkezik a hackerektől a kutatókon át a diákokig. Maga a programnyelv gyakran alkalmazott megoldás webes alkalmazásoknál. Tesztelésre is kiválóan alkalmas, mivel könnyen összekapcsolható szoftverekkel, hardverekkel. Minden általános adatbázistípust képes használni, mint a MySQL, Oracle, Sybase, PostgreSQL, stb. Sok eszközben használják beágyazott alkalmazásként például a Tivo-ban és a GeoEasy-ben is. (Tclwebpage)

Több platformot is támogat: Windows, Macintosh és számos Unix/Linux. Magas szintű API-k állnak rendelkezésünkre, melyek biztosítják az egyes operációs rendszerek közötti különbségek áthidalását. Használatát nagyban megkönnyíti a hozzá kapcsolódó Tk GUI (grafikus megjelenítés) a kevésbé programozó beállítottságú felhasználóknak.

Fontos megjegyezni, hogy a Tcl egy dinamikus programozási nyelv (dynamic programming language), hasonlóan mint a Perl, a PHP vagy a Python, és nem egy rendszer

programozási nyelv (system programming language), mint a C++ vagy a Java. A két típus egymás kiegészítője, általában különböző típusú problémák megoldására használják az egyiket vagy a másikat. A rendszer programozási nyelvekkel alkothatunk komplex adat hálózatokat, algoritmusokat, stb.. A dinamikus programozási nyelvekkel ellenben könnyebb az adatok kezelése, felhasználói felületek illetve eszközök programozása. Gyakran nevezik a dinamikus programozási nyelveket magasabb szintűeknek is, mivel összetettebb parancsok állnak rendelkezésünkre és így egy adott programot sokkal kevesebb, akár 5-10-szer kevesebb gépelt sorban megalkothatunk. Egy rendszer programozási nyelvben viszont sokkal mélyrehatóbban kezelhetjük számítógépünket.

A következőkben egy egyszerűbb példa látható a Tcl programozási nyelvről a TclAPI-ból kiemelve:

```
1. proc Bearing{ea na eb nb} {
2.       global PI2
3.       set de [expr {$eb - $ea}]
4.       set dn [expr {$nb - $na}]
5.       set w [expr {atan2($de, $dn)}]
6.       while {$w < 0} { set w [expr {$w + $PI2}]}
7.       while {$w >= $PI2} { set w [expr {$w - $PI2}]}
8.       return $w
9. }
```

A nyelv maga parancsokból áll össze, melyeket sortörés vagy pontosvessző választ el egymástól. Az előző példában egy függvény látható az irányszög kiszámításához. Két részből áll össze: a paraméter lista ({ea na eb nb}) valamint maga az elvégzendő feladat. Esetünkben a bekért változók sorra az álláspontunk koordinátái valamint a referencia pont koordinátái. A feladatban látható elsőként egy már korábbiakban definiált változó meghívása (2. sor), majd magának az irányszögnek a kiszámítása (3-5 sor). A 6,7. sorban egy-egy while ciklusban kerül ellenőrzésre, hogy a kapott szögérték kisebb-e mint 0 vagy nagyobb-e mint 360° (esetünkben radiánban megadva 2π). Az eredmény mindig 0 és 2 π radián közé esik (0-360 fok).

A különböző parancsok tárháza és így a megoldható feladatok köre is igen széles és egyre bűvölő tendenciát mutat, főként mivel egy nyílt forráskódú nyelvről beszélünk. (Tclwebpage)

A diplomamunkám második felében a GPS-el vezérelt földmunkagép példáján keresztül részletesebben is bemutatásra kerül maga a programozási nyelv is, de ez kerül felhasználásra minden az Ulyxes rendszer keretében működő feladatnál, ez az a nyelv mellyel a TclAPI vezérelhető. De erről részletesebben az Ulyxes rendszerről szóló fejezetben írok.

2.3.3. JavaScript

A JavaScript egy objektumalapú szkriptnyelv, melyet leginkább weblapok készítése során alkalmaznak (mint esetünkben is). A program jellemzően egy tetszőleges böngészőben kerül futtatásra, melyben az integrált JavaScript-motor értelmezi az adott sorokat. Három alap adattípust különböztetünk meg: szöveg, szám és logikai. A JavaScriptben egy változó külön deklarálható a var paranccsal (pl.: var a), majd inicializálható (pl.: a = 5), illetve megtehető mindez egy lépésben is (var a = 5). Az egyes változók deklarálhatók mint tömb, definiálatlan érték, NaN (Not a Number), 0, stb.

Alkalmazhatók speciális parancsok az egyes weblapok egészére vonatkozó eseményekhez, mint például load (az oldal minden objektumának betöltése), scroll (görgetés), stb.. Az egér általi vezérlés eseménykezelésen keresztül oldható meg, például: Click, MouseOver (amikor az adott elem fölé kerül a kurzor), MouseMove (amikor az adott elem fölött mozog a kurzor), stb.. Úgynevezett formokra vonatkozó események is kezelhetők: Focus (amikor az adott elem aktívvá válik), Blur (amikor az adott elem inaktívvá válik), stb. (JavaScript wiki)

```
1. var tal = 0;
2. var maxl = tabla.survey[0].data.length;
3. for (var i=1; i < tabla.survey.length; i++) {
4.    if (maxl < tabla.survey[i].data.length) {
5.       maxl = tabla.survey[i].data.length;
6.       tal = i;
7.    }
8. }
```

A fenti példa egy az ulyxes program keretében alkalmazott eljárás részlete. Az itt látható ciklus lényege, hogy a tabla.survey JSON objektumban eltárolt listák közül kikeresi a leghosszabbat. A JSON egy szöveg alapú objektumtípus. A kereséshez egy ciklust használunk, mely a listák darabszámát veszi alapul (tabla.survey.length). Ezek után az kerül vizsgálatra, hogy a maxl változóban elhelyezett első lista hossza az i-edik darabhoz képest mekkora. Ha nagyobb, akkor a maxl változó felülíródik, történik mindez amíg van nem vizsgált lista.

Diplomamunkámban a JavaScirpt is igen jelentős szerepet kap egy külön munkarész során, amikor is a monitoring rendszer által gyűjtött adatok webes megjelenítése kerül az előtérbe.

2.3.4. PHP

A PHP szintén egy nyílt forráskódú programnyelv, leginkább web szervereken futtatott szkriptekben jelenik meg, segítségével hoznak létre dinamikus weblapokat. A webszerverek több mint háromnegyedén működik valamilyen PHP szkript. Magát a kódot nem kell egy külön fájlba elhelyezni mivel az a HTML oldalba ágyazható, a kódot a web szerver értelmezi. A nyelv képes akár nagyméretű webes adatbázis alapú alkalmazások kezelésére is. A hagyományos HTML oldalakkal ellentétben a PHP parancsok nem kerülnek elküldésre a kliensnek, hanem azokat a szerver oldalon a PHP-értelmező modul feldolgozza.

Mint oly sok nyelvnél itt is rengeteg lehetőségünk van: kapcsolatot létesíthetünk távoli szerverekkel, fájlokat importálhatunk, exportálhatunk vagy adatbázis kezelői teendőket végezhetünk. Nagyon leegyszerűsítve a PHP egy bemeneti file-ból készít valamilyen a böngésző által megjeleníthető fájlt. A PHP nyelv szorosan összefügg a HTML-el, szervesen kiegészíti azt. Az ilyen nyelven írt oldalaknál a kiszolgáló feldogozza a PHP parancsokat és ez alapján készít egy HTML kimenetet, melyet elküld a kliensnek. Egy PHP kódot könnyű elkülöníteni a HTML résztől, mivel minden szkriptet <?php karaktersorral kezdünk és ?> karaktersorral zárunk. (PHPwiki)

Az alábbiakban látható egy az Ulyxes rendszerben alkalmazott PHP szkript (sensor2server.php):

```
1. <?php
2. include_once("config.php"); //external configuration file
3. if (! isset($_REQUEST["pn"]) || ! isset($_REQUEST["epoch"])) {
4.             die("1"); }
5. $pn  = $_REQUEST["pn"];
6. $t = $_REQUEST["epoch"];
7. if (! isset($_REQUEST["e"]) || ! isset($_REQUEST["n"]) || !
         isset($_REQUEST["z"])) {
8.             die("2"); }
9. $e = $_REQUEST["e"]; $n = $_REQUEST["n"]; $z = $_REQUEST["z"];
10. $conn = pg_connect ("host=$dbhost dbname=$dbname user=$user
        password=$pass");
11. if (! $conn)
12.      die ("3");
13.$sql="INSERT INTO results (point_name,epoch,geom)
         VALUES ('$pn','$t',ST_GeomFromText('POINT($e $n $z)',0));";
14. echo $sql;
15. $resource = pg_query ($sql);
16. if (! $resource) {
17.      die(pg_result_error($resource)); }
18. echo "0";
19. ?>
```

Az itt látható program célja, hogy a szenzor által mért adatokat egy adatbázisba tárolja. Ehhez egy beérkező kérés esetén ($_REQUEST) ellenőrzi, hogy az adatbázis számára megfelelő adattípusok a rendelkezésére állnak-e, azaz pontazonosító, időpont és három koordináta. A beérkező adatokat egy SQL parancsba rendezi (13. sor) és ezt elküldi az adatbáziskezelőnek és ellenőrzi az adatbáziskezelő válaszát.

Diplomamunkámban a GeoCom-hoz hasonlóan a PHP is inkább csak a „háttérben" jelenik meg, azaz az Ulyxes program keretében, ahol is a PostGIS szerver és a web szerver között teremt kapcsolatot. Hogy mi is az a PostGIS szerver arról a következő fejezetben lesz szó.

2.3.5. PostgreSQL/PostGIS

A PostgreSQL egy nyílt forráskódú adatbázis kezelő rendszer. Főbb jellemzői: objektum-relációs, minden jelentősebb operációs rendszeren fut, támogatja az idegen kulcsokat, a nézeteket, a triggereket és a tárolt eljárásokat. Támogatja még továbbá a főbb adattípusokat, mint az egész számok, numerikus adatok, logikai, szöveg, dátum, stb., valamint bináris objektumokat is (kép, hang, videó). Különböző programozási felületek kapcsolódnak hozzá: C/C++, Java, Tcl, Python, stb.

Olyan lehetőségek is kapcsolódnak hozzá, mint aszinkron replikáció, beágyazott tranzakciók, online biztonsági mentések, lekérdezés tervező/optimalizáló. Korlátlan méretű adatbázisok hozhatók létre, akár maximális 32 terabyte-os táblaméretekkel. Az adat integrációs lehetőségekkel, elsődleges és idegenkulcsok alkalmazhatók, ellenőrzési, egyedi illetve nem null korlátok adhatók meg. Használhatók még továbbá automatikus növekményű mezők, összetett, egyedi, részleges, és funkcionális indexek. Újabb táblákat az öröklés funkcióval egy már létező tábla alapján hozhatunk létre. A szabály rendszer segítségével a felhasználó szabályokat/műveleteket kreálhat, melyek az egyes táblákhoz, nézetekhez kapcsolódnak és azokat dinamikusan átalakítják, ha futtatásra kerülnek. Az események rendszer segítségével üzenteket továbbíthatunk az egyes kliensek között, valamint lehetőségünk van az adatbázisunk frissítéseinek, új eseményeinek, törléseknek a monitorozásához. (PostgreSQLwebsite)

Egy rövid SQL szkript példa PostgreSQL-hez, mely szintén az ulyxes rendszerhez készült:

```
1. DROP TABLE IF EXISTS results CASCADE;
2. DROP TABLE IF EXISTS points CASCADE;
3. CREATE TABLE points (
4.       gid serial PRIMARY KEY,
5.       point_name varchar(14) UNIQUE
6. );
7. SELECT AddGeometryColumn('','points','geom','0','POINT',3);
8. CREATE TABLE results (
9.       point_name varchar(14) REFERENCES points(point_name),
10.      epoch timestamp NOT NULL,
11.      PRIMARY KEY (point_name, epoch)
12. );
13. SELECT AddGeometryColumn('','results','geom','0','POINT',3);
14. CREATE INDEX date_indx ON results(epoch);
15. CREATE INDEX name_indx ON results(point_name);
```

Az itt látható szkript a már létező adatbázisban új táblákat hoz létre, előtte azonban ellenőrzi azok létezését és ennek fennállása esetén törli azokat. Ezek után a megfelelő attribútumokkal tölti fel a táblákat, illetve elsődleges kulcsokat is rendel hozzájuk (4, 11. sor) valamint a két tábla közötti kapcsolat is megteremtődik (9. sor). A létrehozott táblákhoz indexeket is hozzárendeltünk, melyek nagy adatmennyiség esetén megkönnyítik az adatok közti keresést.

A PostgreSQL-hez szorosan kapcsolódik a PostGIS. Ez egy kiegészítő, mely térinformatikai adatok kezelését teszi lehetővé, a térképi elem geometriáját egy speciális ’geometry’ típusú mezőben tárolja. Ezek kezeléséhez szabványos függvény biztosított (terület/kerület számítás, övezetek generálása, stb.). Több ezer vetületi rendszert tartalmaz a csomag, köztük egyetlen magyarként az EOV-t (EPSG azonosítója 23700), a térképi keresések felgyorsítására térbeli indexelési lehetőségek biztosítottak, mely nagy állományok esetén nagy mértékben gyorsíthatja a munkát. Főleg ott alkalmazható kiválóan, ahol egyszerre több kliens szeretné az adatbázis tartalmát módosítani vagy az adatok elérését jogosultsághoz szeretnénk kötni. A térképi elemek hozzárendelése az adott táblához a fenti példában a 7. és 13. sorban történik az AddGeomertyColumn paranccsal.

Ezzel egy oszlop kerül beillesztésre a táblába, mely pont ('POINT') típusú geometriai adatokat tartalmaz ('geom'). Továbbá még háromdimenziós koordinátákról beszélhetünk, melyet a „3”-es szám mutat, a 0-ás EPSG kód jelen esetben a vetületi nélküli helyi rendszert jelöli.

A geometriai adatok kezelésére egy jó példa egy ilyen beillesztése a táblába, a geometriai adatok kezelése, az attribútum adatokhoz hasonlóan SQL utasításokkal történhet.

```
INSERT INTO points (point_name,geom) VALUES
      ('608', ST_GeometryFromText('POINT(119.19 130.03 120.37)', 0));
```

Az így kapott eredményekkel már rengeteg alapszintű műveletet el tudunk végezni hasonló parancsokkal, minden féle grafikai kezelőfelület nélkül. (PostGISwiki)

Láthatóan az Ulyxes rendszer is egy PostGIS adatbázist alkalmaz, mely a szenzorokból jövő adatokat a számunkra megfelelő táblákba szervezi, majd a webes megjelenítésnél és alkalmazásoknál a kliens oldali kérések egy része innen kerül kiolvasásra, de erről a következő fejezetekben lesz szó részletesebben.

2.3.6. MapServer

A MapServer egy nyílt forráskódú webes térinformatikai programcsomag. Segítségével vektoros vagy raszteres állományokból is előállíthatunk térképeket weblapunkra és azon különböző lekérdezéseket hajthatunk végre. A MapServer lelke a Mapfile, mely arra vonatkozóan tartalmaz adatokat (név, méret, típus, adatelérési útvonal, stb.), hogy az egyes adatforrások, miként jelenjenek meg. Az elkészült térképek valamilyen interaktív megjelenítő felületen kerülhet ki az internetre, például mint esetünkben is az OpenLayers segítségével.

Természetesen több művelet is végezhető az egyes térképeken. Ilyenek az elemek osztályozása valamilyen attribútum alapján, attribútum adatok kiíratása, diagramok készítése, vonalas elemek rajzoltatása, méretarányfüggő megjelenítés, jelkulcsok kezelése. (MapServer webpage)

2.3.7. OpenLayers

Az OpenLayers egy JavaScript alapú úgynevezett függvénykönyvtár, mely a különböző térképi adatokat képes interaktívan megjeleníteni weboldalakon. Az előző mondatból fontos kiemelni, hogy az adatokat csak megjeleníteni képes, ezen kívül valamilyen egyéb „adattároló egységre” van szükségünk, például a korábban taglalt MapServer-re vagy a Google Maps Serverre, mely kettőt az Ulyxes rendszer alkalmazza, de kezelhetők még az OpenStreetMap, Bing Maps vagy a Yahoo! Maps térképei is.

A kódokat kétféleképpen érhetjük el: letöltjük a szükséges fájlokat a szerverünkre vagy az openlayers.org linkjére hivatkozunk a weboldalunkon. Ennek hátránya, hogy lassabban töltődik be weboldalunk, vagy egyáltalán nem – ha az openlayers.org nem elérhető – ellenben mindig a legfrissebb változatot alkalmazhatjuk weblapunkon.

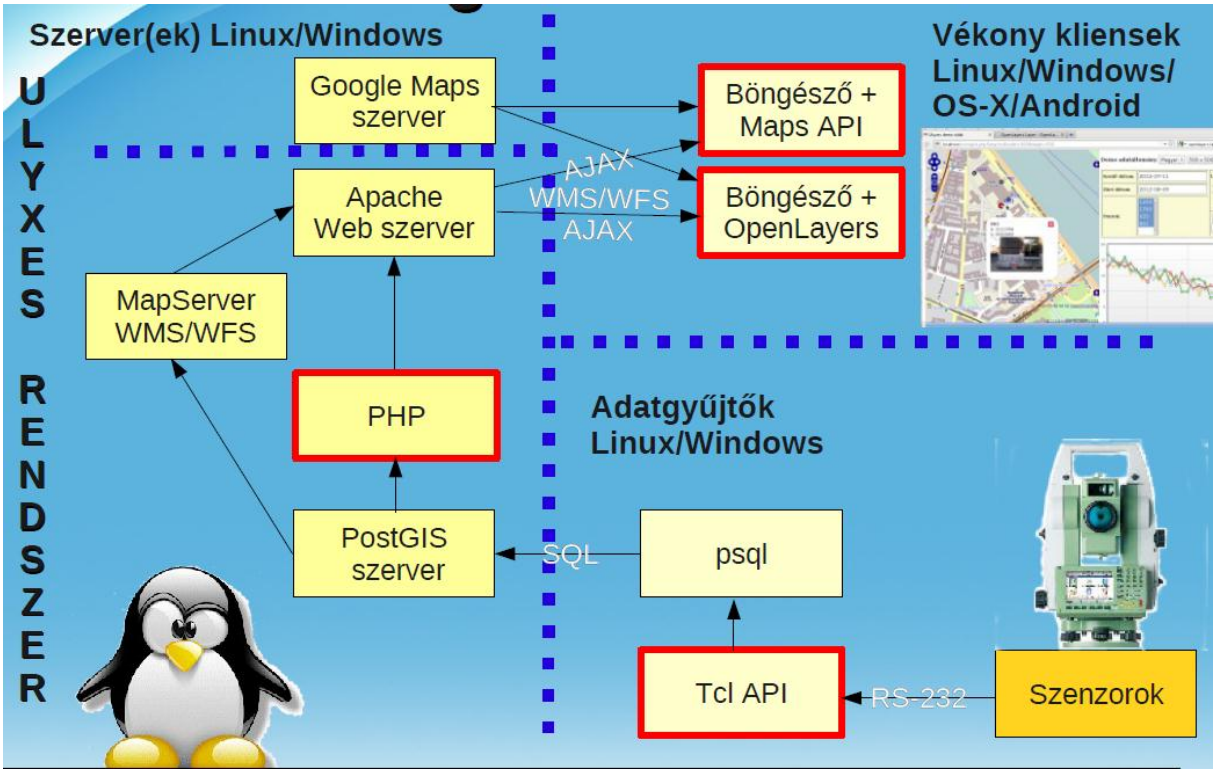
Az OpenLayers segítségével különböző műveleteket végezhetünk: több réteg alkalmazása egy térképen, eseménykezelés (pl.: egér események), különböző réteg típusok (Marker, Vector, Image layer) vetületi rendszerek kezelése. (OpenLayers webpage)

2.3.8. jQuery

A jQuery az előzőekhez hasonlóan szintén nyílt forráskódú. Ez egy keretrendszer a JavaScripthez kapcsolódóan, melynek fő célja, hogy a HTML kódról leválassza azt és e kettős különböző eseményvezérlőkön, azonosítókon keresztül kommunikáljon egymással. Ez által a kliens oldali szkripteket leegyszerűsödnek, valamint a jQueryUI alkalmazásával sokkal dinamikusabb weboldalak hozhatók létre. (jQuery wiki)

2.3.9. Az Ulyxes rendszer

„Az Ulyxes egy nyílt forráskódú projekt a robot mérőállomások és egyéb szenzorok vezérlésére valamint a mérési eredmények internetes térkép alapú publikálására.” A rendszer fő célja egy keretrendszer kialakítása az egyes szenzorok számítógépes vezérléséhez. Ez a mérőállomások kezelésével kezdődött majd szintezőműszerekkel és GPS-sekkel bővült. Az



18. ábra: Az Ulyxes rendszer felépítése

előbbiekben bemutatott valamennyi szoftvert felhasználjuk a rendszerben. A rendszer használható Linux valamint Windows operációs rendszeren is.

A 18. ábra mutatja az Ulyxes rendszer elvi felépítését. A folyamat a szenzorok vezérlésével indul, mely a TclAPI-val lehetsége. A TclAPI a felhasználók számára igen egyszerűen alkalmazható, célja az volt, hogy elrejtse a különbséget az egyes szenzorok között. Egy ilyen rövid, a TclAPI felhasználásával kiadható parancs egy mérőállomás számára például a

% GetAngles

Ez a parancs a háttérben átalakítja kérésünket az adott mérőállomás számára érthető parancsra, Leica mérőállomások esetén egy GeoCOM-os parancsra, majd a kapott választ visszaalakítja számunkra könnyen kezelhetővé és olvashatóvá. Az előbbi parancs mögött az alábbi script rejtőzik:

```
1. Sub GetAngles {} {
2. global buf
3.       if {[set res [::Send "%R1Q,2003:0"]] != 0} { return $res}
4.       set buflist [split $buf ":,"]
5.       set res {}
6.       lappend res [list 7 [lindex $buflist 4]]
7.       lappend res [list 8 [lindex $buflist 5]]
8.       return $res
9. }
```

Az igazán lényeges mozzanat a 2. sorban látható ahol is a szögméréshez tartozó GeoCOM parancs látható (%R1Q,2003:0). A szkript lényegében egy feltétellel megvizsgálja, hogy a kiadott értékre nem nulla választ kap-e és ha ez teljesül, akkor a kapott válaszból, ismerve annak felépítését, kiolvassa a 4. és 5. helyről a számunkra fontos szögértékeket, az egyéb adatokat pedig elhagyja. Ezek után egy {{7 hz} {8 v}} alakú listát hoz létre, melyet a hívó program megkap.

Hasonló parancsok léteznek a felhasználók számára például a távolságméréshez, a koordináták kiolvasásához, az ATR bekapcsolásához, az EDM mód változtatásához, a műszer különböző irányú mozgatásához, stb.

A TclAPI fontos eleme még a COM fájl, melyben a kommunikációs paraméterek kerülnek beállításra, mint a sebesség, port (a használni kívánt Com port száma), paritás, stb.

Visszatérve az Ulyxes rendszer felépítéséhez, a kinyert adatok feldolgozása első lépésként egy PostGIS szerveren kerülnek eltárolásra a megfelelő rendszerezésükkel. Az adatfolyam első része az alfanumerikus adatoké. Ekkor a JavaScript egy Ajax híváson keresztül egy PHP programot indít el, amely a szükséges adatokat kiolvassa a PostGIS adatbázisból és egy JSON formátumban visszaküldi a JavaScriptnek, amely a további adatfeldolgozást is végezi. A térképi megjelenítés az OpenLayers és a MapServer segítségével történik, ahol is különböző rétegek definiálhatók és jeleníthetők így meg. Alaptérképek nem csak az általunk a MapServer-re feltöltöttek lehetnek, de a Google Maps Server-ről is behívhatunk megfelelő alapot adataink megjelenítéséhez. (Ulyxes website)

2.4. A felhasznált műszerek, eszközök

A következőkben bemutatásra kerülnek azok a műszerek, melyeket az egyes feladatrészek során alkalmaztunk.

2.4.1. Leica GPS System 500

A földmunkagép irányításához kapcsolódó GNSS méréseket egy Leica GPS System 500-as (19. ábra) műszerrel végeztük. Ez a típusú vevő 24 csatornás, két frekvenciás. Statikus méréseknél 3mm+0.5ppm pontosságot biztosít, RTK mérések esetén cm-es pontosságot akár nagy hatótávon is. A mérésekhez a bázis és a rover vevő közti kommunikáció rádió kapcsolaton keresztül biztosítható, a GNSSnet.hu által biztosított korrekciókhoz is alkalmas.

19. ábra: Leica GPS System 500 (http://www.wild-heerbrugg.com/catalog/images/SNAG-1120.jpg)

A kommunikáció biztosított különböző (nem csak Leica) vevőtípusok között is. Főbb jellemzői még: csökkenti a többutas terjedés hatását, alacsony műholdszám esetén is stabilan szolgáltat koordinátákat, a vevőhöz közvetlenül csatlakoztatható a terminál, amellyel a különböző beállítások elvégezhetők (pl.: az adatok bevitele), Két akkumulátorról működik, így a cseréjük megoldható jelvesztés nélkül. (GPS System 500 Reference Manual)

2.4.2. Leica TCRA 1201+, Leica TCA 1800

A Leica TCRA 1201+ (20. ábra) egy igen korszerűnek tekinthető mérőállomás, mely alkalmas többféle geodéziai munka elvégzésre. Szögmérési pontossága 1” szögmásodperc, távolságmérési hibája 1mm+1.5ppm. Képes méréseket végezni prizmára és bármely egyéb objektumra is lézeres irányzással prizma nélküli üzemmódban. Szervo motorja teszi számunkra lehetővé, hogy fizikai kontaktus nélkül egy külön számítógépről vezérelhessük a Leica által bevezetett már említett GeoCOM parancsok révén. A beépített ATR segítségével a prizmák durva irányzása esetén automatikusan megirányozza azokat, esetünkben ez teszi lehetővé azok mozgása esetén az irányzásukat. A Power Search funkció révén gyors prizmakeresését is végezhetünk melynek a későbbiekben még jelentősége lesz fejlesztéseink során. Monitoring rendszerekről lévén szó fontos beszélni még az áramellátásról is, mely esetünkben egy darab akkumulátorral történik, de megoldható még a tápellátás y kábelen keresztül külső akkumulátorról vagy számítógépre való csatlakoztatással is. (LeicaTPS1200+)

20. ábra: Leica TCRA 1201+

A tesztelés nagy részét az előbb bemutatott mérőállomással végeztük, de használtunk továbbá még egy Leica TCA1800-as műszert is. Ennek szögmérési pontossága szintén 1” szögmásodperc, távolságmérési pontossága 1mm+1.5ppm. Hasonlóan vezérelhető GeoCOM

parancsok révén ellenben, az előzőekben bemutatott műszertől annyiban eltér korszerűtlensége miatt, hogy nem tartalmazza a Power Search keresésre vonatkozó utasításokat. A későbbiekben bemutatásra kerülő vakon tájékozást végző, ilyen irányba továbbfejlesztett program egyes parancsait így nem tudja értelmezni, tehát csak egy korábbi lassabb módszer volt alkalmazható ezzel a mérőállomással. Nem található benne továbbá még prizma nélküli mérési mód, mely esetünkben inkább pozitívumot jelent (lásd később vakon tájékozás), viszont ATR van a műszerben, így mozgásvizsgálat során a pontok kisebb elmozdulása során is feltalálja azokat. (TCA1800)

2.4.3. Acer Aspire 5750G, szoftverek

A földmunkagép irányításához kapcsolódó feladat elvégzéshez, valamint a monitoring rendszer teszteléséhez és a nyert adatok feldolgozásához is szükség volt egy számítógépre is, mely a terepen mért adatokat a helyszínen képes feldolgozni. Ezt egy Acer Aspire 5750G típusú laptoppal oldottuk meg. A munka során felmerült számítási igények nem tekinthetők jelentősnek, így ilyen szempontból a gép adottságai nem fontosak, ellenben az egyes szoftverek telepítése során figyelembe kellett venni, hogy Windows 7 64-bites verziója került telepítésre.

A földmunkagép irányításához telepítésre került a Tcl (64 bites verzió), a TclAPI, valamint a GeoEasy program. A monitoring rendszer teszteléséhez kapcsolódóan ugyan ezekre volt szükségünk, ellenben a mért adatok internetes feldolgozása jelentősebb szoftverkörnyezetet igényelt: OpenLayers, jQuery, PostgreSQL, PostGIS, OSGeo4W mely tartalmazza az Apapche webserver-t, PHP-t és a Mapserver-t is. Ezen szoftverek mindegyik szabadon hozzáférhető bárki számára.

3. A megoldott feladatok részletezése

3.1. GPS domborzat

3.1.1. A megoldandó feladat részletezése

Ugyan a monitoring rendszerekhez nem szorosan kapcsolódó alkalmazásról van szó, mégis a különböző munkagépek irányítása napjainkban egyre gyakrabban történik valamilyen GNSS megoldással. Ilyen lehet mezőgazdasági kombájn, aratógép irányítása, valamint a földmunkagépek irányítása is előre megtervezett domborzat kialakításához. Ilyen rendszerek már szép számmal találhatók a piacon különböző kivitelezésben. Vannak melyek nem is egy antennát alkalmaznak, így megoldva a munkagép lapjának 3D-s helyzetének precíz meghatározását. Ezzel sokkal változatosabb terepfelszín alakítható ki, mint az egy antennás megoldás esetén, amikor is azt vízszintesnek kell tekintenünk modellünkben. Természetesen több antenna alkalmazása esetén, a bonyolultabb modellből adódóan, szoftverünknek is összetettebbnek kell lennie, hogy ezt a plusz tényezőt kezelni tudja.

A feladat típusából adódóan a magasságnak és a vízszintes pozíciónak a kapcsolata igen sajátságos. A felhasználónak fontos tisztában lennie azzal, hogy milyen igényei vannak e két paramétert tekintve és az ehhez legjobban megfelelő rendszert beszereznie. Elképzelhető olyan feladat, ahol a pontos pozícióhoz tartozó kevésbé pontos magasság is megengedhető, és ennek ellenkezője is lehetséges. Fontos megjegyezni, hogy a GNSS eljárásoknál a magassági értelmű pontosság mindig elmarad a vízszintestől, valamint esetlegesen szükséges megemlíteni a mérések történhetnek kódmérés vagy fázismérés elvén is. A két módszer különböző műszer típusokat követel meg magának és így természetesen különböző árkategóriába és pontossági kategóriába tartoznak. A szoftverek szemszögéből nézve ezek a témák érdektelenek, hiszen a pontatlan adatokból a rendszer ugyan úgy képes a számításokat elvégezni, azonban a felhasználónak ezzel szükséges tisztában lennie.

Diplomamunkám keretében egy ilyen GNSS alapú megoldást alakítottunk ki. Mivel nem állt rendelkezésünkre semmilyen földmunkagép így a tesztelésre nem került sor élesben, csak a műszer kézi mozgatása révén hoztunk létre a szoftver teszteléséhez szükséges adatokat. Az alkalmazott műszer egy, a korábbiakban bemutatott Leica GPS System 500-as volt, a modellünkben a földmunkagép lapjának állását nem vettük figyelembe, azt vízszintesnek

tekintettük. A TclAPI-t és a hozzá készült szkriptet az említett laptopon, Windows operációs rendszer alatt futattuk. Az adatkapcsolatot soros porton keresztül valósítottuk meg.

3.1.2. Az elkészült program bemutatása

Elsődlegesen egy domborzat modell megalkotására volt szükség, azaz egy viszonyítási rendszerre, melytől a műszerünk aktuális magassági eltérését számítani fogjuk. Vonatkoztatási rendszerként az EOV rendszert választottuk, ellenben a későbbiekben részletezett NMEA üzenetek csak WGS84-es rendszerben értelmezett szélességi és hosszúsági koordinátákat tartalmazhatnak, tehát a két rendszer közötti transzformációt is meg kellett oldanunk. A tesztüzemre a legalkalmasabb hely a Duna rakpart volt, ahol tiszta kilátás nyílt az égboltra és kellő méretű tér a szükséges méretű DTM létrehozásához. A teszt területen 15 pontot mértem meg, egy közelítően négyszögletű terület sarokpontjain és azon belül egyenletesen elszórva a maradék pontokat.

A méréseket GeoEasy-ben dolgoztuk fel, ahová manuális úton töltöttük fel a mért koordinátákat. A bevitt koordinátákból a program segítségével egy TIN modellt hoztunk létre, melyből egy rácsot generálva kimentettük a domborzatmodellt egy úgynevezett ESRI ASCII GRID formátumban. A kapott eredmény részletét a 21. ábra mutatja. Mint látható a felépítése igen egyszerű, egy header részben tárolja a kiinduló koordinátákat, a lépésközt, a nincs adat értéket és a sorok oszlopok számát. Lényeges néhány szót szólni a kimentés során általunk megválasztható lépésközről is. Természetesen elég kicsinek kell lennie, hogy kellő számú pontot generáljunk a munkaterületen, melyek viszonylag jól visszaadják a terep változatosságát, de nem túl kicsinek, hogy a kapott koordináták száma túlságosan megnövekedjen és nehézkesen kezelhetővé váljanak. Egy TIN modell alkalmazásával maguk a terepviszonyok jobban visszaadhatók, ellenben a későbbiekben a számítást megnehezítette volna, főként annak

```
ncols 22
nrows 35
xllcorner 183970
yllcorner 646380
cellsize 1
nodata_value -9999
-9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999
-9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999
-9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999 102.09
-9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999 101.96 101.8
-9999 -9999 -9999 -9999 -9999 -9999 -9999 -9999 101.95 101.83 101.71 101
-9999 -9999 -9999 -9999 -9999 101.95 101.82 101.70 101.58 101.45 1
-9999 -9999 -9999 -9999 101.95 101.76 101.58 101.45 101.32 101.20
-9999 -9999 -9999 -9999 101.88 101.70 101.51 101.33 101.14 100.96
-9999 -9999 -9999 -9999 101.81 101.63 101.44 101.26 101.08 100.89
-9999 -9999 -9999 -9999 101.75 101.56 101.38 101.19 101.01 100.85
-9999 -9999 -9999 -9999 101.68 101.50 101.31 101.13 100.94 100.86
-9999 -9999 -9999 -9999 101.61 101.43 101.24 101.06 100.96 100.87
-9999 -9999 -9999 101.73 101.55 101.36 101.18 101.06 100.97 100.89
-9999 -9999 -9999 101.66 101.48 101.30 101.15 101.07 100.99 100.90
-9999 -9999 -9999 101.60 101.41 101.25 101.17 101.08 101.00 100.92
-9999 -9999 -9999 101.53 101.35 101.27 101.18 101.10 101.01 100.95
-9999 -9999 -9999 101.46 101.36 101.28 101.20 101.11 101.03 100.99
```

21. ábra: ESRI ASCII GRID

megkeresése, hogy adott pozíciónk mely háromszögbe esik. Ez egy négyszögháló esetén jóval kisebb számítást követel meg, mint azt a későbbiekben láthatjuk.

Az ESRI ASCII GRID fájl maradék része négyszöghálóban tartalmazza a magassági adatokat, melyekhez a pozíció a header részben található adatok alapján kiszámítható.

Fontos megjegyezni azt is, hogy erősen ajánlott a DTM elkészítésekor a felhasználni kívánt területnél nagyobb területet felmérni. Ennek fő oka, hogy az interpoláció során a DTM határán mért pozíciók esetén könnyen hibára futhat a programunk, mivel nem talál olyan négyzethálót amibe beleesne pozíciónk, illetve maga az ESRI ASCII GRID kimentés is csak pontok közötti interpolációt végez, két szélső pont közötti egyenesen kívülre nem extrapolál magasságot, így sok helyen úgynevezett nincs adat értékek kerülnek a táblázatba, melyek ”-9999”-es karaktersorral szerepelnek.

Következőekben tekintsük át ennek a DTM modellnek az importálási lehetőségét további számításokhoz. Ehhez a TclAPI-ban egy kész script állt rendelkezésünkre, mely egy listába importálja a header rész adatait. Ennek meghívása a következő sorokban történik

```
% source dtm.tcl
% global dtm
% set xxx [dtm "foldmunka1.asc"]
```

mely a következő választ adja számunkra:

```
22 35 475175.5 190702.5 0.5 -9999
```

Ezek a header részben megtalálható adatok és immár a xxx változóból kinyerhetők, illetve egy másik változóból (dtm) a magassági adatok. Ezek a megfelelő index-el hívhatók elő. Pl.:

```
% puts $dtm(5,7)
101.70
```

illetve egy érdekesebb:

```
% puts $dtm(0,0)
-9999
```

mely az első sor első elemét adja vissza, tehát az indexelés nem 1-től hanem 0-tól indul a Tcl nyelvben, ez természetesen igaz a header listájára is. Így a későbbiekben már könnyedén hivatkozhatunk bármely magassági adatra.

Érdekességként megjegyzendő, hogy az indexek szinte bármilyen karakter sorok lehetnek és az adatok vektorként kerülnek eltárolásra ezekkel összepárosítva. Az előbbiekből mégis

úgy tűnhet, hogy egy mátrix sorát és oszlopát hívtuk meg, azonban esetünkben a 101.70-es magassághoz tartozó index az „5,7” karaktersor. Az „5, 7” (a vessző után szünet karakter) sor már mást jelölne. Attól függetlenül azonban, hogy az indexek karaktersorok, mégis a szám típusú karakterekre hivatkozhatunk valamilyen más változón keresztül,

```
% set x 1
%puts $dtm(5,[expr {8-$x}])
101.70
```

Ez a magasabb szintű programoknál, melyek ugyan tudják kezelni a mátrixokat és így az előbbi feladat is könnyen megoldható, nem lenne elképzelhető. Matlab-nál vagy Octave-nál a string típusú változók nem adhatók meg ilyen módokon.

Visszatérve a konkrét feladatunkhoz, a tesztelés második lépésében a GPS-ünktől folyamatos NMEA üzeneteket kellett kapnunk. Ennek beállítása ennél a műszernél a Config/Interfaces/NMEA menüben található ahol is kiválasztható az általunk használni kívánt port, a használni kívánt és támogatott eszköz, a mérések közötti időköz (esetünkben másodpercenként), hogy azonnal vagy pontos epochában (hozzá késleltetés is kapcsolható az üzenet átviteli sebességét figyelembe véve) legyen-e kiküldve az üzenet. Ez igen hasznos lehet, ha több műszert akarunk összekapcsolni, így a mért adatok könnyebben összefűzhetők. Több kimeneteli adattípus közül is választhatunk (GGA, GLL, GNS, VTG, ZDA, LLK, LLQ). Ezek közül néhány részletesen az (GPS System 500 Reference Manual) alapján:

GPGGA (Global positioning Fix Data):

```
$GPGGA,131533.00,4728.8840876.N,01903.3889650,E,1,09,1.3,104.284,M,
         43.55,,*62
```

ahol sorban vesszőkkel elválasztva az egyes adatok: Fejléc, UTC idő, szélesség, N/S, hosszúság, E/W, GPS minőség (1-NAV fix, 2-DGPS fix, 3-RTK fix), műholdak száma, HDOP, antenna magasság tengerszinthez képest, ennek mértékegysége, geoid unduláció, ennek mértékegysége, DGPS adat kora (0-ha nem használt), Referencia állomás azonosítója, Checksum. Ez az üzenet típus kiválóan alkalmas lehet a koordináták kinyeréséhez, illetve az azokhoz tartozó pontossági információk eléréshez.

GPVTG (Course Over Ground and Ground Speed):

```
$GPVTG.69.7010.T,69.7010,M0.007,N,0.014,K,A*21
```

ahol az egyes adatok sorban: fejléc, irány (0.0°-359.9°), tényleges (T), irány (0.0°-359.9°), mágneses (M), sebesség, csomóban (N), sebesség, km/h-ban (K), Checksum. Ez az üzenettípus inkább a navigációs alkalmazásoknál használható.

GPZDA (Time and Date):

```
$GPZDA,131533.00,30,09,2013,-01,00*46
```

ahol az egyes adatok sorban: fejléc, UTC idő, UTC nap (01-31), UTC hónap, (01-12), UTC év (1997-), helyi időzóna eltérés órára vonatkozóan (-13 - 13), helyi időzóna eltérés percekre vonatkozóan, Checksum.

GPLLK (Leica Local Position and GDOP): Jelentősége, hogy a GDOP értékeket is tartalmazza az adatsor 9. helyén (0-tól kezdve a számolást).

Mi a GGA üzenetekből nyertük ki a számunkra szükséges szélességi és hosszúsági koordinátákat, az LLK üzenetekből pedig GDOP értékeket. Ennek pontos módjáról a következőekben még szó lesz.

A műszer vezérléséhez szükséges volt még néhány parancs kiadására a számítógépünkön.

```
%source global.tcl
%source common.tcl
%source nmea.tcl
%source nmea_com.tcl
```

Ezekkel a TclAPI egyes már elkészült scriptjei kerülnek meghívásra. Részleteikben:

global.tcl: Általános paraméterek kerülnek beállításra, mint a π megadása, maximális próbálkozások száma míg a szenzor üzenetet nem küld, debuglevel stb.

common.tcl: Több hasznos funkciót tartalmaz, mint például a műszer relatív mozgatása, radián és fok közötti átszámítás, pontok közötti távolság számítása, stb. Ami a számunkra ezek közül szükséges volt az a GetVal parancs, mely egy listából kiveszi a kért elemeket. Jó tudni, hogy a TclAPI-ban a 37, 38, 39 es számú elemek az x, y, z koordinátákat tartalmazzák.

nmea_com.tcl: Külön a GPS-es mérésekhez tartozó modul. A port megnyitásához és záráshoz szükséges parancsokat, valamint a GetLine parancsot tartalmazza, mely a szenzorból folyamatosan érkező adatokat visszaadja.

nmea.tcl: Ez a rész is külön a GPS-es mérésekhez készült, mely a GGA típusú NMEA üzenetekből kinyeri a koordinátákat és a megfelelő listába illeszti a már említett 37, 38, 39 es pozíciókba.

Mindezek után a műszer csatlakoztatható a soros porton keresztül a számítógéphez. Szükséges a használni kívánt port számát a TclAPI-hoz tartozó leica.com file-ben is beállítani, hiszen a kapcsolat csak így jöhet létre. A kapcsolatot a port megnyitásával (OpenCom, 0 válasz esetén sikeres) tesszük teljessé, illetve tesztelhetjük azt a már említett

%puts [GetLine]

parancs kiadásával.

Így már futatásra kerülhetett az újonnan elkészült irányítást vezérlő szkrpit, mely a mellékletek között megtalálható (1. melléklet)

A programsor elején a már ismertetett DTM importálás látható, és annak header részének egyes változókra bontása a későbbi egyszerűbb felhasználás végett (14-24. sor). Ezek után egy for ciklussal egy periódust indítunk, melynek hossza a program indításakor megadott paraméterek között kell szerepelnie. A program indításához három paraméter megadása szükséges: DTM modell neve, elvárt pontosság (GDOP értékként), valamint a mérések száma. A pontosság rögtön a for ciklus elején ellenőrzésre kerül, nem megfelelő érték esetén megszakítja azt.

```
1. if {[GetVal 100 [Coords]] < $GDOP} {
      …
2. } else {
3.    puts Accuracy isn't enough
4. }
```

A következőkben a koordináták kerülnek lekérésre (31-41. sor), valamint rögtön formázásra is kerülnek. A GGA üzenetekben érkező koordináták tizedesvessző előtti része fokban értendő a tizedesvessző utáni része pedig perc mértékegységű. Ezeket figyelembe véve kerül átalakításra a beérkező koordináta, hogy tisztán fok mértékegységű legyen.

A kapott koordinátákat ezek után áttranszformáltuk EOV rendszerbe. A számítások elvégzéséhez rendelkezésemre állt egy wgseov.tcl fájl mely ezeket a számításokat egy polinomiális egyenlet alapján elvégzi deciméteres pontossággal a két rendszer között oda és vissza. Az egyenleteket Laky Piroska dolgozta ki (Laky Piroska wgs-eov). Ennek meghívása is a source paranccsal történhet. Mivel ez a program a koordinátákat fokban és annak

tizedeseiként képes kezelni, ezért volt szükséges az nmea üzenetekben érkező fok, perc és annak tizedesit is ilyen alakúvá átalakítani. Ezek után a további számítások már mind elvégezhetők voltak EOV koordinátákkal.

Az adatok beszerzése után, gondolok itt a DTM modellre és a pozíciók kinyerésére, kezdődhetett ezek feldolgozása, mely a pozíciónk DTM modellben való elhelyezésével indul, azaz annak meghatározásával, hogy mely rácshálóba esik aktuális helyzetünk (49-78. sor). Mindez azzal kezdődik, hogy pozíciónk egyáltalán ténylegesen a DTM területére esik, ha ez nem teljesül, akkor a megfelelő hibaüzenet közlésével a program leáll.

```
1.  if {$x < $xllcorner || $x > [expr {$xllcorner + $ncols *
            $cellsize}] || $y < $yllcorner || $y > [expr {$yllcorner
            + $nrows * $cellsize}]} {
2.     puts "GPS out of field of action"
3.     break
4.  }
```

Ha megfelelő a pozíciónk, akkor egy-egy while ciklussal kikeressük a befoglaló téglalapot és az így kapott négy ponthoz tartozó magasságokat is.

A kikeresett négy pont magassága is ellenőrzésre kerül, hogy értékük nem-e egyenlő a nincs adat érékkel. Ha ez teljesül, az azt jelenti, hogy pozíciónk a ténylegesen felmért terepen kívül esik. Ebben az esetben a ciklus szintén megszakad a megfelelő hibaüzenet közlése után.

```
1. if {$z1 == $nodatavalue || $z2 == $nodatavalue ||
            $z3 == $nodatavalue || $z4 == $nodatavalue} {
2.     puts "Not valid height"
3.     break
4. }
```

Az adatok feldolgozásának második részében a kapott koordináta négyessel, illetve a pozíciónk vízszintes koordinátáival egy terület arányos magassági interpolációt végzünk. A sarokpontokhoz tartozó magasságokból kiindulva terület arányosan kerül meghatározásra a pozíciónk szükséges magassága bilineáris interpolációval (34-59). A program végső soron csak a tényleges és a szükséges magasság közti különbséget írja ki a számunkra előjelhelyesen (negatív érték esetén az antennát lefelé szükséges mozgatni a kívánt szint eléréséhez).

Programunk tehát a következő paranccsal indítható

```
% nmea_foldmunkagep.tcl DTM 0.4 10
```

ahol az egyes elemek sorban: a program neve, DTM modellünk neve (.asc kiterjesztésűnek kell lennie), szükséges GDOP érték, mérések száma.

Az elkészült programon kívül a mellékletek között megtalálható a módosított nmea.tcl program egy részlete is, mely a GDOP értékek kinyerését mutatja.

3.1.3. Fejlesztési lehetőségek

Több további fejlesztési lehetőség is felmerül az előbbiekben részletezett rendszerrel kapcsolatban.

Első helyen említendő az egy időben több antenna használata. Jelen helyzetünkben ennek megvalósításának fő nehézsége abban rejlett, hogy a TclAPI-ban egy parancssorban egyszerre csak egy szenzor vezérelhető. Természetesen megoldható két szenzor vezérlése is két parancssorból, ellenben az így kapott két adatsor összekapcsolása már bonyolultabb feladatot jelent. Ennek megoldása történhet például egy fájl import és export segítségével, azaz az egyik parancssorból az adatokat kiírjuk egy fájlba, majd azt a másik parancssorba behívjuk és már is rendelkezésünkre állnak mindkét szenzor adatai a szükséges számításokhoz.

Még jobb megoldást jelentene az előbbi szenzoregyüttes kezeléséhez, ha a mérési eredményeinket egy adatbázisba helyeznénk el és onnan kinyerve dolgoznánk fel őket. Így az is biztosított lenne, hogy esetleges későbbi számításokhoz is megmaradjanak mérésink és ne vesszenek el, mint az előbbi megoldás esetén.

Az így nyert kettős adatsorral már megoldható annak kérdése is, hogy a munkagép lapja milyen helyzetben is áll két szögelfordulást tekintve (a két antenna által kijelölt tengely körüli elfordulás még nem detektálható.). A három szög illetve a három különböző irányú elmozdulás méréséhez 3 műszer nem egy egyenesben való elhelyezésére van szükség.

Mivel jelenleg zajlik az Ulyxes program Tcl nyelvről Python-ra való átalakítása, mely képes lesz egy parancssoron belül kezelni több szenzort is objektum orientáltsága révén, így ez a feladat még könnyebben megoldhatóvá válik.

Másik megoldási lehetőség lehet, ha nem egy vagy két újabb GPS antenna alkalmazásában gondolkodunk, hanem esetleg egy másik típusú szenzor integrálásán, példának okáért egy giroszkópot. Ezzel megoldható a munkagép lapjának egy előre beállított vízszintestől való eltérésnek mérése.

Másik fejlesztési lehetőségként felmerül a DTM modell hatékonyabb kialakítása, azaz valamilyen másik adatcsere formátumra való áttérés az ESRI ASCII GRID-ről, amely jobb adatstruktúrával rendelkezik. Esetünkben például nem kerül tárolásra semmi olyan információ, ami a terepmodellünk pontosságára vonatkozna az egyes pontokban, adott esetben viszont erre is szükség lehet.

Komplex megoldásként felmerülhet, hogy a szoftver a pillanatnyi meghatározáson túl akár továbbfejleszthető előrejelző funkcióval is. Mint említettük a GPVTG mérési adatokból kinyerhetők a haladási irányunkra és sebességünkre vonatkozó információk is. Ezen adatokból a vezetőnek akár előre jelezhető, hogy ha továbbra is ilyen sebességgel halad az adott irányba, akkor esetleg túlhalad a munkaterületen vagy a tervezett magassági szintnek aláás.

Végül, de nem utolsó sorban megemlítendő még az is, hogy jelenleg mérési adataink mind elvesznek azok megjelenítését követően, pedig nagy szükség lehet azok tárolására is. A későbbiekben bemutatásra kerülő rendszerhez hasonlóan, ehhez kapcsolódóan is létre lehetne hozni egy adatbázis struktúrát, ahova az adott pozíciók feltölthetők lennének és későbbi felhasználás véget kinyerhetők. Ilyen felhasználás lehet például a megtett útvonal elemzése, optimalizálása, a munka megszakítás esetén, hogy honnan szükséges folytatni azt, valamint egy szemléletes térkép is létrehozható arra vonatkozóan, hogy hol milyen mértékben kell még a terepet rendezni és hol nincs szükség további tevékenységre.

A fejlesztési lehetőségek száma és iránya ezek alapján még igen számos lehet.

3.2. Monitoring rendszer tesztelése

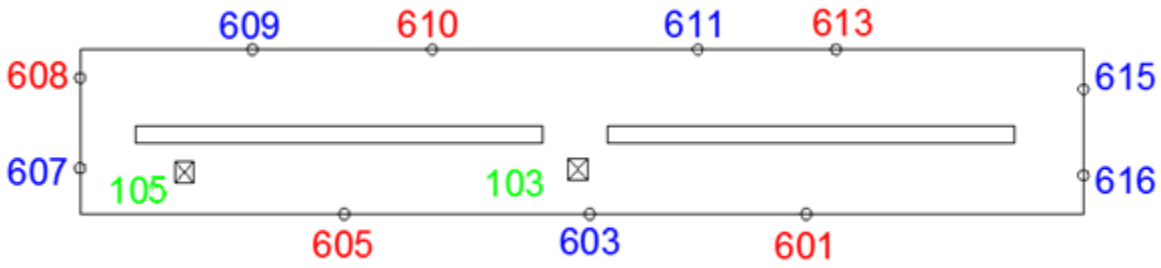
A korábbiakban már röviden szó esett az Ulyxes programról és annak működéséről, a következőkben részletesen is bemutatásra kerül ez a rendszer.

3.2.1. A monitoring rendszer bemutatása

Diplomamunkám keretében az egyik fő feladatot az Ulyxes rendszer fejlesztése jelentette, ennek egyik szeletében a hozzá kapcsloódó szenzor oldali programok átdolgozása. Ennek színhelyéül az Általános és Felsőgeodézia tanszék komparátor termét választottuk, ahol több alkalommal felállítottuk rendszerünket annak tökéletesítése érdekében. Kiindulásként egy helyi koordináta rendszerben helyeztük el műszerünket. A terület sematikus vázát a 22. ábra

mutatja. Műszerünket a 103-as ponton helyeztük el, tájékozást pedig a 105-ös pontra végeztünk, hogy az adott körülmények bármikor újra előállíthatók legyenek.

A falakon elhelyezett prizmákat két osztályba soroltuk: Az első csoport a referencia prizmáké, melyeket mozdulatlannak tekintettünk méréseink során. Egy üzemszerű munka során ezek a prizmák jelentik számunkra azokat az alappontokat, melyekre mozdulatlanságunk ellenőrzése végett méréseket végezhetünk, illetve újra meghatározhatjuk műszerünk pozícióját. A második csoportba azokat a prizmákat soroltuk, melyek a mozgásokat hivatottak modellezni. Az ábrán az előbbiek pirossal utóbbiak kékkel jelöltek.



22. ábra: A monitoring rendszer teszt területe

A rendszer elindítása különböző fázisokból áll össze, ezeket a következőekben részletesen bemutatjuk. Természetesen az egyes elemek teljes körű működéséhez itt is szükséges az egyes beépített modulok behívása, mint a global.tcl, common.tcl, leica.tcl.

Első fázisként a fileMaker.tcl programmal a két csoportra osztott prizmákat egy-egy körben végigmérjük és meghatározzuk koordinátáikat az általunk kijelölt koordinátarendszerben. Ehhez a program csak az álláspontunk azonosítóját és 3 koordinátáját kéri be, majd az egyes prizmák azonosítóit. A műszerünket a folyamat előtt tájékoznunk szükséges. Az így elvégzett két manuális mérési fordulóval két-két fájlt hoz létre a program, a referencia és a mozgó prizmákra vonatkozó méréseket (.geo) és koordinátákat (.coo) későbbi felhasználásra. Ezek a fájlok a későbbiekben az egyes munkafolyamatok során újra behívásra kerülnek, tehát csak a műszer kitelepítésekor kell egyszer végrehajtani a műveletet. A műszer magára hagyása után bármikor elindítható az automatikus mérési ciklus.

Első lépésként a tájékozást kell ellenőriznünk. Külső okokból származóan előfordulhat, hogy műszerünk a referencia prizmákat nem találja meg a rájuk vonatkozó koordináták alapján. Ennek oka a hibás tájékozottságban keresendő, esetlegesen valamilyen zavaró objektum miatt. Utóbbi lehet kitakarás, ez esetben az adott pont kihagyása szükséges, de lehet valamilyen zavaró fényvisszaverő tárgy is, esetünkben például előfordult, hogy a műszer egy

prizma közelében elhelyezett fóliát talált meg, ellenben méréseket nem tudott rá végezni ezért a program hibára futott.

Tájékozásunk elromlása esetén a blindOrientation.tcl programmal tudunk prizmákat kerestetni a műszerrel. Ehhez először egy minimális és egy maximális magassági szög kerül kiszámításra, melyek között a prizmák elhelyezkedhetnek a műszerünkhöz viszonyítva. Ehhez az álláspontunk és a referencia prizmáink közti magasság eltérés kerül kiszámításra és az adott tartományhoz fél-fél méter kerül hozzáadásra alul és felül. A kapott tartomány két szélének magassági szöge kerül meghatározásra. Ezek után a műszerünk akkurátusan végigpásztázza az adott vízszintes szögnél ezt a magassági tartományt bizonyos lépésközönként (alapbeállításként 3°) spirális mozgással, majd vízszintesen is tovább fordul ugyan ekkora lépésközzel. Egy prizma megtalálása esetén az arra vonatkozó távolságmérés alapján dönti el, hogy mely prizmát találhatta meg (0,5 m-es eltérés engedélyezett). Ebből a prizmából kiindulóan a program ismét letájékozza limbuszkörét és elindítható a szabad álláspont meghatározásához kapcsolódó szkript (lásd később), mely tovább pontosítja ezt a „durva” tájékozást (Mivel itt csak egy pont alapján végezzük el a számításokat).

Látható, hogy ez az eljárás igen hosszadalmasra nyúlhat, például ha a műszerünk néhány fokkal jobbra elcsavarodott és az ATR már nem találja meg a prizmát, akkor a blindOrientation.tcl program alkalmazásával a műszer jobbra forogva kezdi el keresni a prizmát, így már csak a következő pontjelet találhatja fel. Előfordulhat az is, hogy az esetlegesen túl nagy lépésköz miatt a műszer túlhalad egy prizmán. Ilyen esetben a forráskódban ennek átírása szükséges, de az általunk végzett tesztek során a lépésközből fakadóan nem történt hiba, csak a tévesen megtalált fóliák miatt. Ennek a hibának a kiküszöbölése még nem oldódott meg, de éles helyzetben a prizmák gondos elhelyezése megszünteti ezt a hibaforrást.

A keresés felgyorsítása végett alkalmazható az újabb generációs műszereknél egy sokkal gyorsabb metódus is. Ez az úgynevezett Power Search eljárás, mely szintén egy GeoCom paranccsal elindítható és így beépíthető a TclAPI-ba is. Ennek használati lehetőségei a (TPS1200 GeoCOM Manual)-ból kikereshetők. Először csak az úgynevezett keresési ablak kerül megadásra egy középponttal valamint a vízszintes és a magassági értelmű tartománnyal:

```
%R1Q,9043:dCenterHz,dCenterV,dRangeHz,dRangeV,bEnabled
```

Vízszintes értelemben célszerű a teljes síkot letapogatni azaz 360°-ot beállítani, a középpont magassági értelemben és a magassági tartomány a prizmák elhelyezkedésének függvényében kell, hogy kiszámításra kerüljön. Ezek után elindítható a keresés.

```
%R1Q,9052:
```

A műszer ekkor a beállított sávban bocsát ki magából jelet és ebből a tartományból bárhonnan visszaérkező jelet képes detektálni és így a prizmát megtalálni. Ez a keresés kevesebb mint 10 másodperc alatt lezajlik.

A keresés végeredményeként különböző üzeneteket kaphatunk: 0, 27, 8720, 8710. Ezek jelentése sorban: sikereses keresés, érvénytelen GeoCom license kulcs, nem definiált keresési ablak, keresés sikertelen.

Ezt az opciót kihasználva a blindOrientation.tcl programnak elkészült egy olyan verziója is, mely a kiszámított magassági tartományban alkalmazza ezt a lehetőséget, és a megtalált prizmát ugyan úgy leellenőrzi távolságmérés alapján. Ezzel a tájékozás elvesztése esetén annak visszanyerése töredék idő alatt lezajlik; természetesen csak a megfelelő műszer alkalmazása esetén működik. Az elkészült módosított blindOrinentation2.tcl forráskódja szintén megtalálható a mellékletek között (3. melléklet).

Mivel a tapasztaltak alapján, ha egy ilyen Power search egy prizmáról kiindulva kerül elindításra, akkor valamely okból kifolyólag időnként azt nem találja meg. Ennek kiküszöbölésére először egy ellenőrző mérést végez a műszer, hogy nincs-e ott prizma, és ha igen akkor az a számára megfelelő-e (71-84. sor). A keresési mező beállítása 85-93. sorig látható, a prizma keresése a 94-113. sorig látható. Érdemes lehet még megemlíteni, hogy egy általunk érdektelen prizma megtalálása esetén a keresési mező kiindulási pozícióját újra meghatározzuk, hiszen ha ugyan abból az állásból indulna, akkor ismét a már megtalált prizmát keresné fel.

A következő fázis során álláspontunk mozdulatlansága kerül ellenőrzésre. Ehhez a freestation.tcl programra van szükségünk, mely egy fordulóban a referencia prizmák irányai alapján végigméri azokat és szabad álláspontként meghatározza saját pozícióját. Minden esetben az új álláspont koordinátákkal folytatódik a mérési ciklus. A szabad álláspont meghatározásához több fölös mérés is rendelkezésünkre áll, így a pontosság növelése érdekében kiegyenlítést alkalmazhatunk. Ezek a számítások nem kerültek megírásra a TclAPI-ban ellenben a GNU Gama program az ilyen számításokat képes elvégezni. Hogy a

két programot összekapcsoljuk a TclAPI a kapott eredményeket XML sorokba rendezi és ezeket egy XML fájlban helyezi el. Ezeket a GNU Gama képes beolvasni, a szükséges számításokat elvégezni és számunkra az új koordinátákat szolgáltatni. Ezek ismét egy XML fájlba kerülnek kiírásra. A kapott koordinátákat beállítjuk műszerünk álláspontjaként valamint a tájékozásunkat is pontosítjuk.

Ha tájékozásunk rendben van és álláspontunk is meghatározott, akkor a robot.tcl program futtatásával mérhetjük be a mozgónak tekintett prizmákat. Ez a program végeredményként a pontok koordinátáit számítja ki. Fejlesztésként most annyiban módosult, hogy egy HTTP GET kérés kerül kiküldésre a kapott koordinátákkal és a mérési epochával, mely üzenetet egy szerver oldali php szkript (sensor2server.php) értelmez, majd a kapott adatokat az adatbázisunkban elhelyezi. Itt csak az újonnan elkészült néhány sor kerül bemutatásra:

```
1. set token [http::geturl"http://localhost/server_scripts/
      sensor2server.php?pn=$pn&epoch=$t&e=$e&n=$n&z=$z"]
2. upvar #0 $token state
3. if {$state(body) != "0"} {
4.    puts "Error: $state(body)"
5. }
6. puts "$pn;$e;$n;$z;$t"
```

Az első sorban látható, hogy az alkalmazni kívánt HTTP GET URL parancsba kerülnek beolvasásra a meghatározott pn (azonosító), t (időpont), e, n, z (koordináták) tagok. Látható, hogy az url-ben meghívásra kerül a már korábbiakban bemutatott sensor2server.php szkript, mely végső soron az adatbázis számára elküldi a szükséges parancsokat az adatok tárolásához.

A bemutatott három programrész (freestation.tcl, blindorientation2.tcl, robot.tcl), együttes alkalmazásához készült egy monitoring.bat fájl, mely a három programot a szükséges sorrendben futtatja (4. melléklet). Három bemeneti paramétert kell megadnunk számára: az álláspontunk számát, a referencia pontokra és a mozgó pontokra vonatkozó fájlok nevét (kiterjesztés nélkül; a koordinátalistának és a mérési listának azonos nevűnek kell lennie egy pontcsoportra vonatkozóan). A szkript első lépésben a blindOrientation2.tcl segítségével elvégzi a tájékozást, majd a freestation.tcl-el szabad álláspontként meghatározza pozícióját, majd futtattatásra kerül a robot.tcl, mely elküldi a mért adatokat az adatbázisunkba. Ha a robot.tcl valamely okból kifolyólag hibára fut (ennél a pontnál feltehetően valami a pozíciónkkal kapcsolatban romolhat el), akkor a vakon tájékozáshoz tér vissza a folyamat majd a szabad álláspont meghatározáshoz.

Hogy a programot ne kelljen manuálisan újra és újra elindítani valahányszor egy új mérési ciklust szeretnénk, érdemes a monitoring.bat fájlhoz ütemezést rendelni. Itt különböző programokhoz rendelhetünk műveleteket, esetünkben annak elindítását, megfelelő időpontokhoz kapcsolva. Tesztüzemünk során 10 percenként végeztünk egy-egy mérési ciklust 2 különböző napon. Az első napon a TPS1200+-os, a második napon a TCA1800-as műszert alkalmaztuk. Utóbbi esetén nem tudtuk a Power Search módszert alkalmazni, mivel ez egy korábbi műszertípus, így ott a tájékozást először kézzel végeztük el, majd egy esetleges tájékozásvesztés esetén a korábbi verzióval kerestettünk volna prizmát a műszerrel, de erre nem került sor.

3.2.2. A tesztüzem kiértékelése

A több órás méréseink során szembetűnő jelenség volt, hogy mind az álláspont meghatározása mind a mozgó prizmák bemérése során a műszer nem a legoptimálisabb mozgást végzi. Ez arra vezethető vissza, hogy az egyes pontlistákon növekvő sorrendben haladunk végig és nem vesszük figyelembe a műszer pillanatnyi állását, ezért olyan jelenségek voltak megfigyelhetők, hogy a blindorientation.tcl sikeres lefutása után a freestation.tcl nem a megtalált ponttól kezdte el a méréseit, hanem a pontlistában szereplő legkisebb irányszöggel rendelkező ponttól. Ez ugyan nem jelent jelentős mértékű időveszteséget, de igen gyors mozgások során felmerülhet arra vonatkozó igény, hogy ezt optimalizáljuk, tehát a mérendő irányok alapján futassuk le egyes programrészeinket.

Egy másik kérdéses probléma még továbbá, hogy jelenleg a robot.tcl program a mérések elvégzése után minden egyes ponthoz egységesen hozzárendeli ugyan azt az epochát. Természetesen ezek a mérések nem egyszerre történtek és adott esetben akár több mint egy perc is eltelhet két pont bemérése között. Itt szintén előfordulhatnak olyan igények, melyek ennek javítását igénylik, ellenben ez később az internetes adatfeldolgozásnál problémákat vethet fel. Mint látni fogjuk a táblázatos és a grafikus megjelenítésnél is ezek az eltérő epochák mind belekerülnek az egyes listákba "-" értékekkel, de több pontra vonatkozó mérés esetén ez igen zavaróan hat a megjelenítésre. Keresztkorreláció számítása során például egyáltalán nem lehet így kimutatni összefüggést az egyes pontok között az általunk kialakított rendszer segítségével, mivel nem egy időpontra vonatkoznának a mérések.

Egy másik fontos fejlesztési lehetőség a két távcsőállásban való mérés. Ezzel a módszerrel a hagyományos műszereknél rengeteg hibát kiszűrhettünk, jelenleg ez a módszer az ipari

geodéziából kezd kiszorulni, mivel a mérőállomások ezen hibái kalibrálásuk során meghatározzásra kerülnek, melyeket ezek után a műszer figyelembe vesz. Mégis, nagy pontosságú mérések eléréséhez adott esetben szükség lehet a két távcsőállásban való mérésre, mely a freestation.tcl programba kiegészítésként könnyedén beilleszthető: az első távcsőállásban kapott szögleolvasások alapján a prizmák újra felkutatása második távcsőállásban. A robot.tcl program képes több távcsőállásban méréseket végezni.

Másik fejlesztésként megemlítendő még, hogy az adatbázisba jelenleg csak a kapott koordináták kerülnek feltöltésre, de szükség lehet a nyers mérési adatok feltöltésére is, valamint a kiegyenlítés során kapott eredményekre, például a középhiba értékére. Itt fontos jelezni azt is, hogy jelenleg a durvahibaszűréshez szükséges statisztikákat a GNU Gama ugyan kiszámítja, de nem kerülnek kiértékelésre. A szabad álláspont meghatározása során viszont nem lenne szabad feltételeznünk, hogy referencia prizmáink elmozdíthatatlanok, tehát szükséges lenne ilyen jellegű szűrés. Hibaforrást jelent az is, hogy arra vonatkozóan nem végzünk ellenőrzést, vajon az újonnan meghatározott szabad álláspontunk nem-e tér el az eddigiektől hibahatáron fellül. Az ilyen számítások elvgézéséhez szükség lehet a szerver oldali adatbázisra is, ahonnan az alap koordinátákat kinyerheti a program és így a szükséges számításokat elvégezheti.

3.3. Adatfeldolgozás és web-es megjelenítés

Az előző részben részletesen bemutatásra került a monitoring rendszer adatgyűjtési metódusa. A következőkben áttekintjük az adatok feldolgozását, azok internetes publikálását.

3.3.1. Az adat áramlási rendszer bemutatása, webes megjelenítés

Mint már a korábbiakban említettük a mért koordinátákat, a pontszámot és a mérés időpontját a robot.tcl program elküldi a PostGIS adatbázisunkba a sensor2sever.php programon keresztül. Természetesen a megfelelő adatbázis struktúrát már előre létre kellett hoznunk ehhez. Ez esetünkben két táblából épül fel. Az első tábla (points) áll egy egyedi azonosítóból (gid), pont névből (point_name) és a hozzá tartozó geometriai attribútumból (geom). Ez a már korábban is említett geometry típusú mező, mely a megfelelő koordináta-hármast tartalmazza. Ez a tábla a webes megjelenítés miatt tárol egy ponthoz kapcsolódóan egy koordináta-hármast. A másik tábla (results) ezen kívül kiegészül még egy epoch oszloppal is, mely egy mérési időpontot is tartalmaz az adott koordinátákra vonatkozóan.

Ebbe a táblába kerülnek feltöltésre a mért koordináták, majd később további számítások véget továbbküldésre.

Az internetes megjelenítéshez megfelelő térkép biztosítása is szükséges, esetünkben a komparátorteremről. Ehhez rendelkezésemre állt egy vázlatos térkép a laborról, mely georeferálva megfelel a pontok sematikus megjelenítéséhez. A georeferálást QGIS-ben végeztem el, ahova a létrehozott adatbázisból a points2d nézetet (points táblából létrehozott nézet, mely 2D-s koordinátákat tartalmaz csak) PostGIS réteg hozzáadásaként betöltöttem. Ehhez a két program közti kapcsolatot megteremtése volt szükséges, mely a PostGIS adatbázis nevének, helyének és portjának megadásával történhet a QGIS-ben. A behívott pontokat megfeleltettem az ábra megfelelő pontjaival, ezek után a program elmentette azt georeferált tif képként.

Ahhoz, hogy ez a kép egy tetszőleges böngészőben megjelenjen a MapServer-t kellett alkalmaznunk. Ehhez egy map fájl állt rendelkezésemre, melyben egy új rétegként definiáltuk a létrehozott georeferált képet:

```
1. LAYER
2.    NAME 'labor'
3.    DATA 'Laborrajz.tif'
4.    TYPE RASTER
5.    STATUS ON
6.    CLASSITEM "[pixel]"
7.    CLASS
8.       EXPRESSION "0"
9.       STYLE
10.         COLOR 0 0 0
11.      END
12.   END
13.   CLASS
14.      EXPRESSION ([pixel] >= 1 AND [pixel] < 255)
15.      STYLE
16.         COLOR 0 0 0
17.      END
18.   END
19.   END
```

Hasonló módon a pontjainkat is el kellett helyeznünk a térképen, ehhez az importált points2d nézetet elmentettük shapfile-ként és az előbbiekhez hasonló módon egy másik réteget hoztunk létre, ennek típusa azonban nem raszter lett, hanem points. Ez által a későbbiekben a pontokat a térképen egyesével kezelhetjük. Ezek a rétgek csak a vizuális megjelenítést szolgálják, mindenfajta mérési eredmény az adatbázisból kerül kiolvasásra az egyes lekérdezések során.

Hogy a rétegeteket megfelelően hoztuk-e létre a map fájlunkban, ellenőrizhettük annak a böngészőből történő meghívásával a következő url megadásával:

```
http://localhost/cgi-bin/mapserv.exe?map=/OSGeo4W/apache/htdocs/
   server_scripts/ol.map&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap&
   LAYERS=labor,points&STYLES=&SRS=EPSG:3857&BBOX=105,110,130,150&
   WIDTH=600&HEIGHT=700&FORMAT=image/png
```

Itt a MapServeren keresztül kerül meghívásra a map fájlunk, melybe a két említett réteg elhelyezésre került, és így a GetMap paranccsal a georeferált tif fájlunk és a points2d shapefile-unk egy png kiterjesztésű képben egyesítve kerül a böngészőnkben megjelenítésre.

Ezek után a szintén rendelkezésemre álló getdata.php aktualizálást kellett elvégeznünk az adott adatbázis struktúrához. A points és a results táblához kapcsolódóan is létrehoztunk egy-egy lekérdezést, mely az adatbázisból a points tábla esetén lekérdezi a pont azonosítóját, 3 koordinátáját, valamint a hozzá kapcsolódó results táblában található mérések számát:

```
SELECT points.point_name, ST_X(points.geom) as lng, ST_Y(points.geom)
   as lat, ST_Z(points.geom) as height, count(results.point_name) as n
   FROM points LEFT JOIN results ON points.point_name =
   results.point_name
   GROUP BY points.point_name, points.geom
   ORDER BY points.point_name
```

A results tábla esetén az egyes mérésekre vonatkozó adatokat (pont azonosító, epocha, 3 koordináta) kérdezzük le:

```
SELECT point_name, epoch, ST_X(geom) as x, ST_Y(geom) as y,
   ST_Z(geom) as z FROM results WHERE point_name=…
```

Az így lekérdezett adatokat egy a Javascript által értelmezhető JSON objektumba rendezzük, ha az ol.php kódból erre vonatkozó utasítás érkezik. Az ol.php fájl szintén a rendelkezésemre állt és az előzőekhez hasonlóan az adott körülményekhez kellett aktualizálnunk. Ebben különböző eljárások találhatók, mint a GetPoints eljárás, mely az előbb említett points táblára vonatkozó php utasítást hívja meg a getdata.php-ben és a válaszként érkező JSON objektumot egy változóban helyezi el. Így az adatok már további feldolgozásra JavaScriptben is elérhetők.

Az internetes adatfeldolgozás megjelenítéséért felelős html programkód valamint az ehhez szorosan kapcsolódó JavaScript forráskód is az ol.php fájlban találhatók. Ebben a fájlban első módosításként egy új WMS réteget hoztunk létre mely az alaptérképet tartalmazza. Ehhez lényegében csak az elérési útvonalat kellett megadnunk:

```
1. map.addLayer(new OpenLayers.Layer.WMS('labor',
2.    'http://' + ulyxesHost + '/cgi-bin/mapserv.exe?
           map=/OSGeo4W/apache/htdocs/server_scripts/ol.map',
3.    {layers: 'labor'}
4. ));
```

Hasonló módon, ellenben WFS rétegként, került definiálásra a pontokat tartalmazó réteg is.

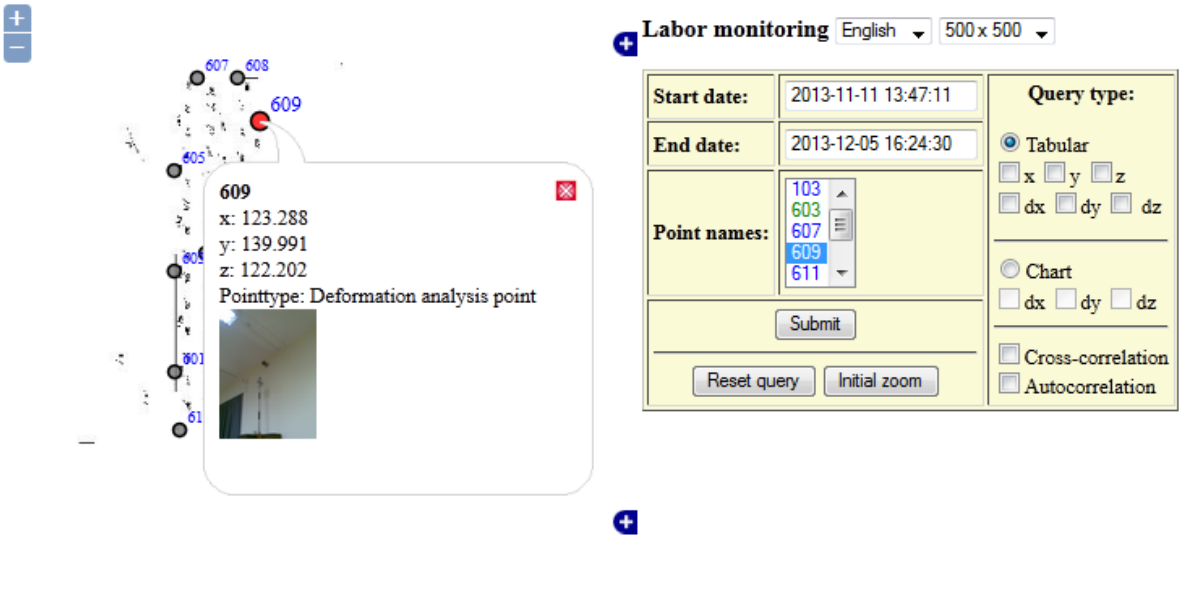
A PointSelected eljárás a térképen így megjelenő pontok közüli kiválasztás esetén az adott pontra vonatkozó adatokat egy felugró buborékban helyezi el. Ehhez kiegészítésként elkészült egy FindPoint eljárás, mely a kinyert adatok közül egy megadott pontazonosító alapján, egy újabb változóban helyezi az arra vonatkozó adatokat:

```
1. function FindPoint(pn) {
2.    for (i=0 ; i < pts.xy.length ; i++) {
3.      if (pts.xy[i].point_name == pn) {
4.          return pts.xy[i]
5.      }
6.    }
7. }
```

A PointSelected eljárásban ez az eljárás kerül meghívásra és a kért adatok a következőképpen kerülnek kiíratásra egy a pontról készült képpel kiegészítve:

```
1. function PointSelected(event) {
2.   popupFeature = event.feature;
3.   var pp = FindPoint(popupFeature.data.point_name);
4.   if (pp.n > 0) {
5.     ptype = getMsg("movp")
6.   } else {
7.     ptype = getMsg("refp")
8.   }
9.   var attr = '<div><b>' + popupFeature.data.point_name +
           '</b><br />' +
10.    'x: ' + new Number(pp.lng).toFixed(3) + '<br />' +
11.    'y: ' + new Number(pp.lat).toFixed(3) + '<br />' +
12.    'z: ' + new Number(pp.height).toFixed(3) + '<br />' +
13.    getMsg("ptype") + ': ' + ptype + '<br />' +
14.    '<a href="' + popupFeature.data.point_name +
           '.jpg" target="_blank"><imgsrc="' +
           popupFeature.data.point_name +
           '.jpg" height="100"></a></div>';
15.  var popup = new OpenLayers.Popup.FramedCloud("attrPopup",
16.    popupFeature.geometry.getBounds().getCenterLonLat(),
17.    null, attr, null, true, onPopupClose);
18.  popupFeature.popup = popup;
19.  map.addPopup(popup);
20.  // Select clicked point in point list
21.  $("#pn").find('option:[value='+event.feature.attributes.
           point_name+']').attr('selected', true);
22.   }
```

Itt az általam elvégzett fő módosítás abban rejlik, hogy még a getdata.php-ben lekérésre kerültek a results mappában megtalálható az adott pontra vonatkozó mérések száma is. Így itt a 4-8. sorig láthatóan vizsgálatra kerül ez a szám. Ha nincs az adott pontra vonatkozó mérés, az azt jelenti, hogy az nem mozgásvizsgálati pont, hanem referenciapont. Ez az adat is kiíratásra kerül a felugró buborékban (13.sor). Láthatóan nem a konkrét szöveg olvasható a forráskódban, hanem egy változó kerül kiíratásra ("refp", "movp", "ptype"). Ezek a változók a msg_en.php és az msg_hu.php fájlokban vannak eltárolva és attól függően kerül az egyik vagy a másik fájlból meghívásra az adott változóban tárolt szöveg, hogy melyik nyelv lett beállítva a böngészőnkben. Az alábbiakban látható a fenti munkánk eredménye és a későbbiekben bemutatásra kerülő lekérdezést vezérlő kezelőfelület (23. ábra):



23. ábra: Internetes megjelenítés

További módosítások történtek még a html kódban is, ahol is eddig az adatok kinyerése során olyan választási lehetőségeink voltak csak, hogy táblázatosan vagy grafikonon kérjük az adatokat ábrázolni. Mivel eddig csak magassági értelmű adatok kerültek az adatbázisba így ez elegendő volt, most azonban három koordináta került tárolásra és az ezekhez kapcsolódó eredeti pozíciótól való eltérés is kinyerhető. Ezek kezeléséhez az eddig alkalmazott rádiógombokon kívül bekerültek a menübe újabb úgynevezett checkbox-ok is. Ezek csak akkor válnak aktívvá, ha a hozzájuk tartozó rádiógomb került kiválasztásra. További fejlesztésként bekerült a rendszerbe a kereszt- és az autokorreláció számítása is, melyek szintén chekboxokkal érhetők el, egyszerre azonban mindig csak az egyik választható ki. A táblázat és grafikon megjelenítésben annyi változtatás történ még, hogy a táblázat a

kezelőfelület alatt jelenik meg, a grafikon pedig a kezelőfelülettől jobbra, így egyszerre látható mindkét lekérdezési típus.

A következőkben bemutatjuk a rádiógombokhoz tartozó egyes lekérdezési típusokat, elsőként a táblázatot. Ez az AddTable eljárással történik, mely több részből tevődik össze.

Első lépésként az adatokat kérjük le a már említett módon a getdata.php-n keresztül és tároljuk el őket a "tabla" nevű változóban. Ezek után a táblánk fejlécét hozzuk létre egybekötve azzal, hogy ellenőrizzük, mely koordináták kerüljenek bele. Természetesen a kiválasztott checkbox-ok alapján történik mindez, ha egy sem került kiválasztásra, akkor a megfelelő nyelvű hibaüzenet kerül kiküldésre egy felugró ablakban. Egy ilyen ellenőrzés a következőképpen néz ki:

```
1. if ($("#tabx").attr('checked')) {
2.     p += "<th>" + tabla.survey[j].point_name + " x [m]</th>";
3.     sum = sum+1;
4. }
```

A "#tabx" a html kódban egy elem azonosítója, melyen keresztül az adott elem (checkbox) állapota kerül kiolvasásra. Hasonlóan kerül ellenőrzésre az y, z, dx, dy, dz táblatípus is.

A harmadik negyedik lépés egy közös while ciklusban történik, ahol elsőként a kiválasztott pontokhoz tartozó mérési listákból kikeressük a legkisebb dátumot, majd az ehhez kapcsolódó mérési eredményt elhelyezzük a táblázatban, minden egyes pontra vonatkozóan. Ezek után a mérési listát tovább léptetjük és a következő időponthoz tartozó eredmények kerülnek feltöltésre. Természetesen ellenőrizni kell, hogy milyen típusú adatokat kért le a felhasználó és az annak megfelelő adatokat töltjük csak be a táblába. A szükséges adatok kiszámításához egy külön eljárás készült, amely bemenő paraméterként egy koordinátát, valamint opcionálisan a tizedesjegyek számát (d), kiinduló koordinátát (k) és szorzótényezőt (sz) vár el.

```
1. function field(a, d, k, sz) {
2.     var m;
3.     k = k || 0;
4.     sz = sz || 1;
5.     d = d || 3;
6.     try {
7.         m = new Number((a - k) * sz).toFixed(d);
8.         if (m == "NaN") {m = "-"}
9.     } catch (err) {
10.        m = "-";
11.    }
12.    return "<td align=\"right\">" + m + "</td>";
13.    }
```

Egy paraméter megadása esetén az eljárás a megfelelő koordinátát adja vissza három tizedesjegy élességgel (alapbeállítás: 5. sor) a táblázatba való betöltéshez

```
p += field(temp.x); ahol temp = tabla.survey[j].data.shift();
```

a többi paraméter megadása esetén a koordinátaváltozás kerül kiszámításra miliméterben 1 tizedesjegy élességgel:

```
p += field(temp.x, 1, pp.lng, 1000);
```

A kapott végeredmény a következőképpen néz ki (24. ábra):

Date	603 x [m]	603 dy [mm]	607 x [m]	607 dy [mm]
2013-11-11 13:47:11	116.834	-1.2	NaN	NaN
2013-11-11 13:56:00	NaN	NaN	118.567	-1.0
2013-11-11 13:56:05	116.834	-1.1	NaN	NaN
2013-11-11 14:11:30	116.834	-1.4	118.568	-1.2
2013-11-11 14:17:20	116.834	-1.4	118.568	-1.0
2013-11-11 14:27:20	116.834	-1.4	118.568	-1.2
2013-11-11 14:37:19	116.834	-1.4	118.568	-1.3
2013-11-11 14:47:19	116.834	-1.4	118.568	-1.1
2013-11-11 14:57:18	116.834	-1.4	118.568	-1.3
2013-11-11 15:07:31	116.834	-1.4	118.568	-1.2
2013-11-11 15:17:19	116.834	-1.4	118.568	-1.2
2013-11-11 15:27:16	116.834	-1.4	118.568	-1.2
2013-11-11 15:37:18	116.834	-1.4	118.568	-1.3

24. ábra: A létrehozott táblázat

A grafikonos megjelenítések aktualizálása során hasonló módosításokat kellett elvégeznünk a forráskódban, mint a táblázatok kezelése esetén. Elsőként itt is az adatok kerülnek lekérésre, majd a pontok közül a leghosszabb mérési listával rendelkező kiválasztása történik meg, valamint ebből kiindulva az egyes listákban egyediként szereplő epochák többi listába való betöltése is. Egy ilyen hiányzó epocha keresése az alábbiakban látható:

```
1.  talt = 0;
2.  talalat = 0;
3.  for (var j=0; j < tabla.survey.length; j++) {
4.   for (var i=0; i < tabla.survey[j].data.length; i++) {
5.     for (var k=0; k < tabla.survey[tal].data.length; k++) {
6.       if (tabla.survey[j].data[i].dt !=
               tabla.survey[tal].data[k].dt) {
7.         talt = 0;
8.       } else {
9.         talt = 1;
10.        break
```

```
11.       }
12.     }
13.     if (talt == 0) {
14.       talalat = tabla.survey[j].data[i].dt;
15.       tabla.survey[tal].data.splice(i+1, 0, {dt:talalat,
                x:"-",y:"-",z:"-"});
16.     }
17.     talt = 0;
18.     talalat = 0;
19.   }
20. }
```

Ezzel a kereséssel betöltjük a leghosszabb elemű listába az abból hiányzó epochákat "-" koordinátaértékkel, valamint egy hasonlóval a többi listába is a csak a leghosszabb elemű listában szereplő epochákat. Az így kapott listákat még a dátumok alapján sorba kell rendeznünk további felhasználás végett:

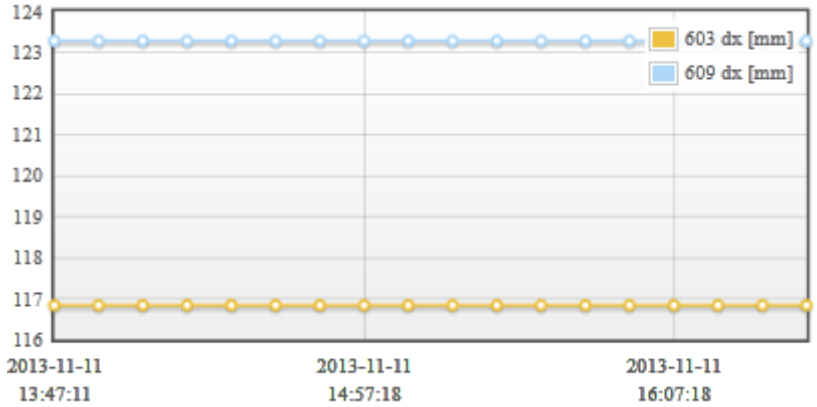
```
1.  for (var j=0; j < tabla.survey.length; j++) {
2.    var temp = [];
3.    for (var i=1; i < maxl; i++) {
4.      if(tabla.survey[j].data[i-1].dt > tabla.survey[j].data[i].dt) {
5.        temp = tabla.survey[j].data[i-1]
6.        tabla.survey[j].data.remove(i-1)
7.        tabla.survey[j].data[maxl-1] = temp;
8.        i=0;
9.      }
10.   }
11. }
```

A 6. sorban használt remove parancs nem egy általánosan alkalmazható utasítás, hanem szintén egy eljárás, mely egy tömb egy elemét hivatott eltávolítani. Ez az eljárás egy internetes fórumról került letöltésre (Arrayremove).

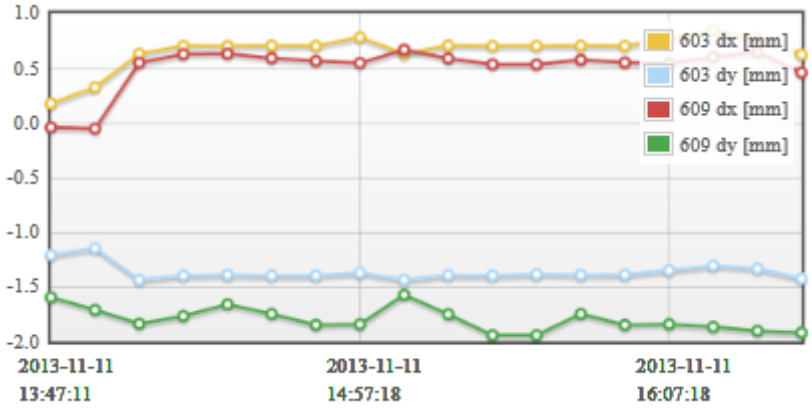
Mindezek után a szükséges ábrázolandó elem kerül kiszámításra. Ez a grafikonok esetén koordinátaváltozást jelent (a lekért adatok típusa itt is ellenőrzésre kerül, hogy a megfelelő koordinátatípusok kerüljenek a grafikonra):

```
m = [i, new String((new Number(tabla.survey[j].data[i].x) -
        new Number(pp.lng))*1000)];
```

Annak oka, hogy a nyers mérési eredmények itt nem kérdezhetők le, több adat egyszerre történő ábrázolásából fakad. Egy pont egyetlen koordinátája esetén lényegtelen lenne, hogy a koordinátaváltozást vagy a nyers mérést ábrázoljuk, mivel mindkét esetben jól kirajzolódnáknak a néhány miliméteres vagy néhány tized miiliméteres változások. Mivel az egyes koordináták több méteres eltérésűek, így a megfelelő ábrázoláshoz reálisan megválasztott lépték esetén a nyers koordináták grafikonon történő megjelenítése során ez a látványos ábrázolási mód egy egyszerű egyenessé redukálódna (26. ábra).



26. ábra: Alkalmatlan ábrázolási mód



25. ábra: Alkalmas ábrázolási mód

Az általunk választott ábrázolási mód a 26. ábrán látható:

A korábbiakhoz képest újításként bevezetésre került a kereszt- és az autokorreláció számítása is grafikus formában. Ezen adatok kinyeréséhez egy-egy checkbox áll rendelkezésünkre a kezelőfelületen, melyek kijelölésével az előbb létrehozott grafikon helyén, azt felülírva jelenik meg a végeredmény. A két függvény számításához szintén választanunk kell az egyes koordináttípusok közül.

A keresztkorreláció (3.1) és az autokorreláció (3.2) számításánál alkalmazott képletek rendre az alábbiakban láthatók:

$$r(d) = \frac{\sum_i^n [(x_i - m_x) * (y_{i-d} - m_y)]}{\sqrt{\sum_i^n (x_i - m_x)^2} * \sqrt{\sum_i^n (y_{i-d} - m_y)^2}} \qquad (3.1)$$

$$r(k) = \frac{\sum_i^n [(x_i - m_x) * (x_{i-k} - m_x)]}{\sqrt{\sum_i^n (x_i - m_x)^2} * \sqrt{\sum_i^n (x_{i-d} - m_x)^2}} \qquad (3.2)$$

A keresztkorreláció számításának célja, hogy bizonyos összefüggéseket mutasson ki két idősor között. Esetünkben ez két pont adott koordinátájának azonos irányba történő elmozdulásának időbeli eltérése, tehát egy pont elmozdulását mekkora idő elteltével követi a másik. Az autokorreláció vizsgálatával egy idősorban periódikusságot fedezhetünk fel.

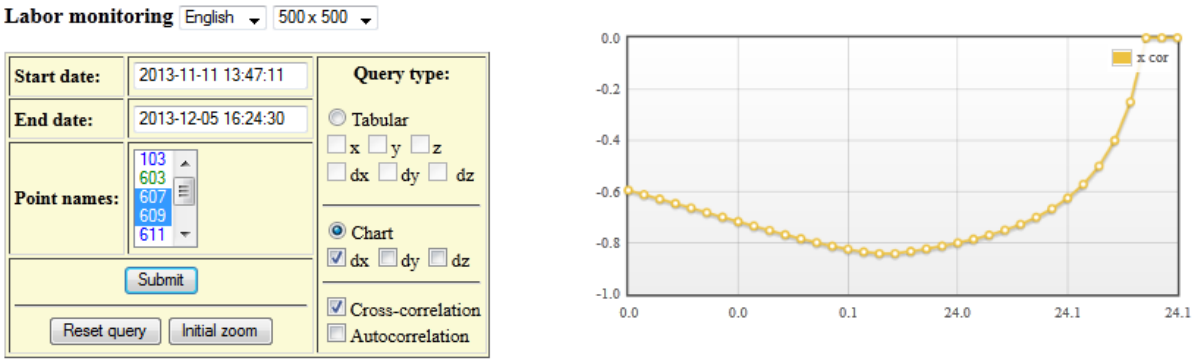
A kereszt- és az autokorreláció számítása is hasonlóan történt az eddigiekhez. Először vizsgálatra kerül, hogy az adott checkbox kijelölésre került-e, majd megfelelés esetén két cikluson keresztül (az első a késleltetés növelése, a második a mérési epochák végiggörgetése) számításra kerülnek egy-egy külön változóban a fenti képletekben szereplő számlálók és nevezők (két részletben). Ezekkel a ciklusokkal a szummázásokat helyettesítettük. A következőkben látható a keresztkorreláció számításánál alkalmazott forráskód a grafikus megjelenítéshez x koordinátára vonatkozóan:

```
1. if ($("#cgrax").attr('checked')) {
2.   var corr = [];
3.   for (d=0 ; d < maxl ; d++) {
4.     corrsz = 0;
5.     corrx1 = 0;
6.     corrx2 = 0;
7.     corre = 0;
8.     for (i=0; i < maxl-d ; i++) {
9.       if (xx[0, i][2] != "-" && xx[1, i+d][2] != "-" ) {
10.          if (tabla.survey[0].data[i].x != "-" &&
                   tabla.survey[1].data[i].x != "-")
11.            corrsz = corrsz + (new Number(xx[0, i][2] -
                 xatl[0][1]) * new Number(xx[1, i+d][2] -
                 xatl[1][1]));
12.            corrx1 = corrx1 + Math.pow(new Number(xx[0, i][2] -
                         xatl[0][1]), 2);
13.            corrx2 = corrx2 + Math.pow(new Number(xx[1, i+d][2] -
                         xatl[1][1]), 2);
14.            corre = corre + (corrsz/(Math.sqrt(corrx1) *
                         Math.sqrt(corrx2)))/(maxl-d);
15.          }
16.      }
17.    }
18.    m = [d, corre]
19.    corr.push(m);
20.    if (d % td === 0) {
21.      tic.push([d, new Number((Date.parse(tabla.survey[0].data[d]
           .dt.replace(" ","T")) - Date.parse(tabla.survey[0]
           .data[0].dt.replace(" ","T")))/86400000).toFixed(1)]);
22.    }
23.  }
24.  cor[k] = corr;
25.  k=k+1;
26.   }
```

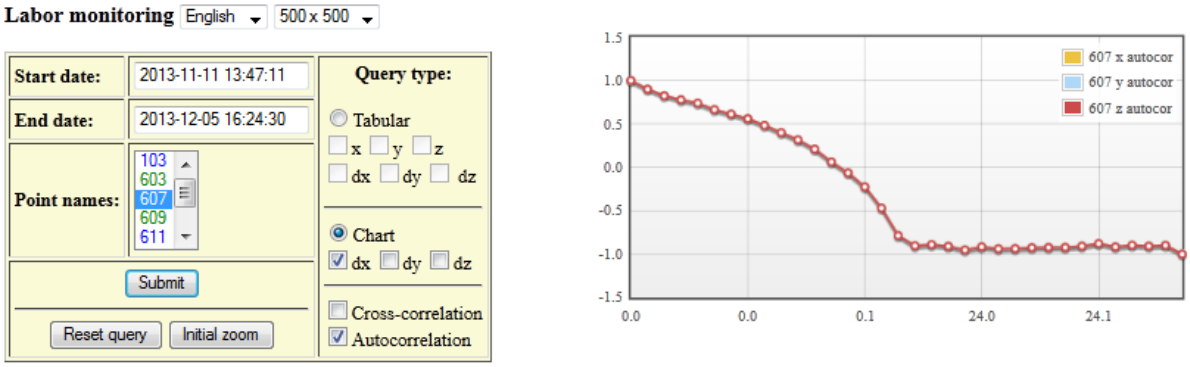
Az autokorreláció számítása hasonlóan történik azzal a különbséggel, hogy csak egy adatsorra vonatkozóan végezzük el a számításokat, tehát xx[1, i+d][2] helyett mindenütt

xx[0, i+d][2] szerepel. Ebben az xx változóban (többi koordinátatípus esetén yy, zz) még a deformáció számítása során kerültek elhelyezésre a koordináták valamint a koordináták átlagértékei is ott kerültek kiszámításra (xatl). A tic változóban a grafikon x tengelyének értékei kerülnek kiszámításra nap mértékegységben. A Date.parse függvénnyel két dátum különbségét tudjuk kiszámítani miliszekundumban.

A végeredmények az alábbi ábrákon láthatók, rendre a keresztkorreláció (27. ábra) és az autokorreláció (28.ábra):



27. ábra: Keresztkorreláció

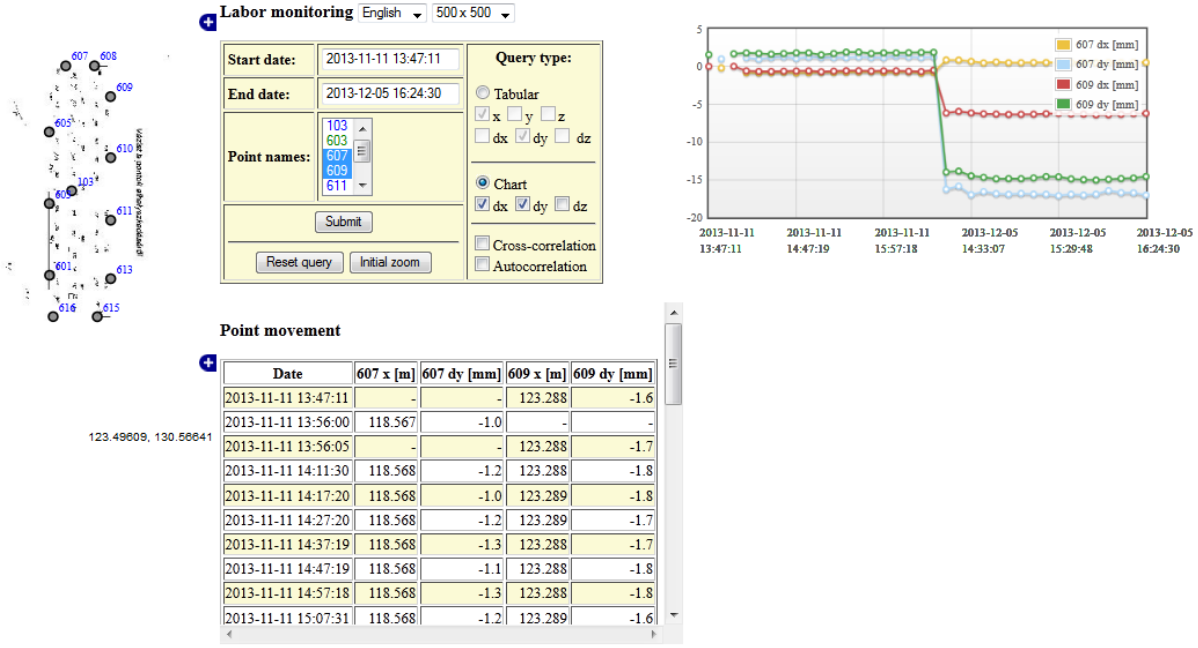


***28**. ábra: Autokorreláció*

Az egyes számítások csak kettő illetve egy pont kiválasztása esetén futnak le, mint az a fenti képeken is látható, egyéb esetben a chekboxok kiválasztása esetén a megfeleő hibaüzenet jelenik meg a képernyőn.

Az újonnan elkészült programrészek teljességükben megtalálhatók a mellékletek között (CD-n).

Álljon még itt egy kép a fejlesztett honlap egészéről is (29. ábra):



Date	607 x [m]	607 dy [mm]	609 x [m]	609 dy [mm]
2013-11-11 13:47:11	-	-	123.288	-1.6
2013-11-11 13:56:00	118.567	-1.0	-	-
2013-11-11 13:56:05	-	-	123.288	-1.7
2013-11-11 14:11:30	118.568	-1.2	123.288	-1.8
2013-11-11 14:17:20	118.568	-1.0	123.289	-1.8
2013-11-11 14:27:20	118.568	-1.2	123.289	-1.7
2013-11-11 14:37:19	118.568	-1.3	123.288	-1.7
2013-11-11 14:47:19	118.568	-1.1	123.288	-1.8
2013-11-11 14:57:18	118.568	-1.3	123.288	-1.8
2013-11-11 15:07:31	118.568	-1.2	123.289	-1.6

29. ábra: Az elkészült weblap

3.3.2. Fejlesztési lehetőségek

Az elkészült rendszeren a bemutatottakon kívül még számos fejlesztésre kerülhet sor. Ezek között elsőként a kezdődátum és a végdátum kezelését lehetne említeni, melyek változtatása egy avatatlan fél számára nehézkes lehet elsőre. Jelenleg az oldal felkeresése során ez a két érték feltöltésre kerül a legkisebb illetve a legnagyobb adatbázisban található elemmel (dátum és időpont is), változtatás esetén azonban csak egy dátumválasztó ugrik fel és onnan csak naptári nap választható ellenben időpont nem. Az óra perc másodperc érték megadása csak kézzel történhet és csakis az óó:pp:mm formátumban különben a lekérdezés nem fut le. A felugró dátumkezelő a datepicker paranccsal található a forráskódban a $("#sd") változóhoz hozzárendelve, hasonlóan az időpillanat megadása is történhetne a timepicker utasítással egy másik változóban. Ennek beiktatásához a forráskódban a kezdődátum és a végdátum is két-két változóként szerepelne és így minden lekérdezési típust ennek megfelelően kellene módosítani.

Másik fejlesztési lehetőségként említendő, hogy a grafikonos megjelenítés során a listákból hiányzó epochaértékek fetöltése egy kevésbé elegáns megoldással történik, itt ajánlatos lenne áttérni egy hasonló módszerre, mint amely a táblázatoknál is alkalmazásra

került, hiszen az több lépést is megold egy cikluson belül, míg itt az egyes lépések külön-külön ciklusokban történnek.

Ennek kapcsán említendő a következő hiányosságunk, mely az egyes táblázat illetve grafikon típusok állandó ellenőrzését érinti. Mindkét megjelenítési típus esetén többször ismétlődnek az arra vonatkozó ellenőrzések, hogy mely koordinátákra vonatkoznak a lekérdezések. Mivel az erre vonatkozó változókat a html kódból vesszük át így a foreach parancs csak közvetve alkalmazható, de ajánlott lenne alkalmazni. Ezzel a paranccsal egy adott tömb minden eleme elérhető sorrendben, hasonlóan egy for ciklushoz. Esetlegesen egy eljárással az egyes koordinátatípusokat 0,1,2… számértékkel feleltethetnénk meg, majd a későbbiekben ezek felhasználásval már alkalmazhatnánk egy foreach ciklust. Ez által a forráskód sokkal áttekinthetőbbé és hatékonyabbá tehető az ismétlődő sorok felszámolásával.

További lekérdezési típusok is alkalmazhatók lennének még a mérési eredmények elemzésére, például valamilyen valószínűségi elemzések beiktatása arra vonatkozóan, hogy a pontok mozgása tényleges mozgásnak tekinthető-e vagy csak mérési hiba. Durva hiba szűrésére is szükség lehet, mint például esetünkben, amikor is az első mérési ciklusunk során hibás műszermagasság maradt beállítva a műszerben, mely a kapott magassági értékeket 40 cm-el torzította.

Az adatbázis struktúrában is lehetene arra vonatkozó fejlesztéseket tenni, hogy ne csak a koordináták kerüljenek tárolásra, hanem a nyers mérési eredmények is a prizmákra vonatkozóan, mint korábban is említettük az álláspontunk meghatározására vonatkozó adatok nagy része jelenleg elveszik csak a kapott pozíció kerül eltárolásra az adatbázisban, ellenben érdemes lenne a kiegyenlítés bizonyos elemeit is eltárolni, hiszen ebből is rengeteg információt lehetne még kinyerni. Úgynevezett logfájlok készítése is előnyös lenne. A tesztüzemünk során a rendszert ugyan nem hagytuk magára, folyamatosan javítottuk az egyes felmerülő hibákat, azonban egy éles helyzetben a műszer kihelyezésre kerül az adott objektumhoz és teljes mértékben távirányításról beszélhetünk. Ilyen helyetbe egy esetlegesen felmerülő problémát a rendszer egy hibaüzenet formájában a logfájlban jelez a szakembereknek.

4. Összefoglalás

A korábbaikban bemutatottak jól példázzák, hogy napjainkban a szakmai irányok erősen az informatika irányába mutatnak a geodézia területén. Ez jól megfigyelhető a hétköznapokban is, ahol a geodézia és az informatika közös vívmányai egyre szélesebb körben jutnak el az egyszerű emberekhez. Gondolhatunk itt többek között a világméretű térképekre, mint a GoogleMap, BingMap, OpenStreetMap, stb. vagy a navigációs rendszerekre, melyek mára már szinte minden autóban megtalálhatók. Ezek és hasonló innovációk is mind az egyes szakterületek összefonódásából jöttek létre, valamint a szakemberek szakmában, és általában a tudományokban való jártasságuk révén.

Az előzőekben lépésről lépésre bemutattuk azt a helyzetképet, mely az ipari geodéziában dolgozó szakemberektől is egyre nagyobb jártasságot követel meg az informatika nem csak felhasználói, de programozói szintű ismeretében is. Mindezt jól példázzák a bemutatott világszerte megvalósult egyes rendszerek, melyeken kersztül láthattuk ennek a témának változattoságát: A már szinte ”hagyományosnak” mondható mérőállomás-programozáson kívül láthattuk a GNSS rendszerek új típusú alkalmazását, valamint a lézerszkennerek innovatív felhasználási módjait is, illetve az ipari geodéziától kissé eltávolodva bemutattuk a monitoring rendszerek alkalmazási lehetőségeit a techtonikus lemezmozgások és a vulkánkutatás területén is.

A dolgozat második felében az is bemutatásra került két példán keresztül, hogy ezek az új típusú megoldások, milyen lépéseken keresztül állnak össze egy teljes programmá, programcsomaggá.

Az első programban, mely egy földmunkagép GPS-el történő irányítását modellezte, egy egyszerűbb, de könnyebben áttekinthető feladatot mutattunk be, mely jó példája annak, hogy egy geodéta kezdő programozói tudással is képes a szakmájában könnyen, de jól alkalmazható programokat írni. A feladat nagy hiányosságának mondható, hogy a mérési adatok nem kerülnek tárolásra, ellenben ebben rejlik a megoldás egyszerűsége. Nem kell egyszerre 5-6 programozási nyelvet ismernünk és az ezek közötti átjárhatóságokat, amely – mint azt a moniroring rendszer esetében láthattuk – ahhoz szükséges, hogy a méréseket egy adatbázisban tároljuk, majd onnan kinyerjük további számításokhoz. Egy egyszerűbb, de nem túl elegáns módon, megoldható lett volna ennél a feladatnál és más hasonlónál is, hogy a mérési eredményeket egy egyszerű szövegfájlba helyezzük el, és az adatokat később félig

manuális módon rendszerezzük. Úgy gondolom, hogy a csekély programozói ismeretekkel rendelkező, ellenben az új, automatizált megoldások felé nyitott geodéták számára, az ilyen félig automatizált-félig kézileg vezérelt megoldások jelentik a fejlődés irányát.

A második részben bemutatott monitoring rendszer szenzor és szerver oldali programjai és az egész rendszer felépítése már jóval bonyolultabb képet mutat. Mint látható volt egy már kész programcsomag került átdolgozásra a jelenleg alkalmazni kívánt területre. Fontos – főleg napjainkban, amikor is az interneten keresztül rengeteg szabad forráskódú és ingyenesen elérhető alkalmazás áll a szakemberek rendelkezésére – hogy előbb tájékozódjunk, milyen megoldások lelhetők fel az interneten és azok között nem találunk-e olyat, amely tovább fejlesztve rengeteg programozásba ölt órától menthet meg minket.

A bemutatott rendszer programozási szempontból főleg azért egy komplexebb feladat, mert egyszerre több programozási nyelv és szoftver ismeretét igényli meg. Ez ugyan elsőre riasztólag hathat, ellenben szélesebb körű jártasságunk révén egy adott feladatot így sokkal könnyebben oldhatunk meg, mint egy nyelv alkalmazásával. Itt elsősorban olyan esetekre kell gondolnunk, mint esetünkben a szabad álláspont meghatározása kiegyenlítéssel. Itt ismét visszautalhatunk az előző bekezdésben írtakra: ezt a problémát többek között a GNU Gama program készítői is megoldották, így ajánlatosabb volt a mérési eredményeinket XML parancsokon keresztül automatizált módon a programba beolvastatni és azzal elvégeztetni a számításokat, mint saját magunknak leprogramozni egy ilyen egészen bonyolult feladatot Tcl-ben.

Általánosságban a dinamikus weblapok hasonló komplexitást mutatnak az alkalmazott programozási nyelvek terén. Láthattuk, hogy esetünkben a szerver oldalon megjelent az SQL, PHP, HTML, JavaScirpt és jQueryUI nyelvek és egyéb szoftverek, mint az Apache Web Server, MapServer, OpenLayers. Ezek mindegyike a maga területén igen jól alkalmazható, de egy dinamikusan és jól működő oldal elkészítéséhez mindegyik alkalmazására szükség van, sőt adott esetben még akár többre is. Az ezek közötti átjárhatóság és összefüggés sok esetben nehéznek tűnik és nehezen fedezhető fel, de ezek megismerése révén szakmailag sokkal nagyobb tudásra tehetünk (tehettem) szert, és egyre bonyolultabb problémákat oldhatunk meg egyre könnyebben.

A bemutatott ismeretanyag a laikusok számára is egy érdekes és színes képet mutathat a geodézia világáról annak fejlődési irányáról, valamint a szakmai berkeken bellül is egy

áttekintést nyújthat arról, hogy milyen megoldások is jönnek létre világszerte ebben a témában és azok körülbelül hogyan épülhetnek fel az általunk megoldott két feladat tükrében. Mégis úgy gondolom, hogy a dolgozat általánosságban azt mutatja be – főként az utolsó részben – hogy milyen fontos a különböző ismeretanyagok és szakterületek fúziója, hogy ez által minnél kreatívabb és elegánsabb módon oldhassuk meg az elénk kerülő problémákat.

Köszönet Dr. Siki Zoltánnak a szakdolgozatomban nyújtott rengeteg segítségért.

Irodalomjegyzék

Internetes hivatkozások:

AlaskaVolcano (2013): http://www.avo.alaska.edu/activity/

Arrayremove: http://stackoverflow.com/questions/3954438/remove-item-from-array-by-value

Brenner (2012): http://www.leica-geosystems.com/downloads123/zz/general/general/TruStories/GNSS_Monitoring_for_Safety_on_the_Brenner_TRU_en.pdf

GeoCom Reference Manual: http://www.surveyequipment.com/PDFs/FlexLine_GeoCOM_Manual_en.pdf

GPS System 500 Reference Manual: http://www.virginiadot.org/business/resources/locdes/survey-leica-gps-techref.pdf

Laky Piroska-wgseov: http://www.agt.bme.hu/staff_h/zaletnyik/Atszamitas.html

LaserScanner (2013): http://www.amerisurv.com/content/view/11686/2/

Leica-Kirchtobel (2011): http://www.leica-geosystems.com/downloads123/zz/general/general/TruStories/Railway_Bridge_Monitoring_Kirchtobel_TRU_en.pdf

LeicaTPS1200+: http://www.leica-geosystems.com/downloads123/zz/tps/tps1200/brochures/Leica_TPS1200+_brochure_en.pdf

JavaScript wiki: http://hu.wikipedia.org/wiki/JavaScript

jQuery wiki: http://hu.wikipedia.org/wiki/JQuery

MapServer webpage: http://mapserver.org/

OpenLayers webpage: http://openlayers.org/

Periadriatic Seam: http://en.wikipedia.org/wiki/File:Alps-Relief_02de%2BPeriadriatic.jpg

PHPwiki: http://hu.wikipedia.org/wiki/PHP

PostGISwiki: http://wiki.hup.hu/index.php/PostGIS

PostgreSQLwebsite: http://www.postgresql.org/

TCA1800: http://hds.leica-geosystems.com/en/System-2000_5253.htm

Tclwebpage: http://www.tcl.tk/about/

TPS1200 GeoCOM Manual: http://www.leica-geosystems.com/en/page_catalog.htm?cid=2944

Ulyxes website: http://www.agt.bme.hu/ulyxes/index_hu.html

VolcanoMonitoring (2009): http://volcanoes.usgs.gov/activity/methods/deformation/

Cikkek, könyvek:

Andrej Gosar, Stanka Šebela, Blahoslav Košták, Josef Stemberk (2007): Micro-defromation monitoring of tectonic structures in W Slovenia. Acta Geodyn. Geomater Vol. 4, No. 1 (145), 87-98

Daniel Dzurisin (2007): Volcano Deformation – Geodetic Monitoring Techniques. Springer Praxis Books / Geophysical Sciences

Detrekői Ákos, Ódor Károly (1998): Ipari geodézia I-II, Műegyetemi Kiadó

Farkas József (2004): Alapozás – előadások rövidített jegyzete (Bsc. képzés)

Krauter András (2007): Geodézia. Műegyetemi Kiadó.

Mellékletek

1. Földmunkagép irányítás

```
1. # This program calculate the difference between a DTM model and
   the current position in height
2. # author Király Tamás
3. # @param DTM
4. # @param accuracy
5. # @param number of measurement
6. proc foldmunka {filenev, GDOP, meresszam} {
7. source global.tcl
8. source common.tcl
9. source leica.tcl
10.   source dtm.tcl
11.   source wgseov.tcl
12.   # load DTM model
13.   global dtm
14.   set xxx [dtm $filenev]
15.   #define DTM model parameters
16.   set ncols [lindex $xxx 0]
17.   set nrows [lindex $xxx 1]
18.   set xllcorner [lindex $xxx 2]
19.   set yllcorner [lindex $xxx 3]
20.   set cellsize [lindex $xxx 4]
21.   set nodatavalue [lindex $xxx 5]
22.   for {set i 1} {$i < $meresszam} {incr i} {
23.    # checking the accuracy
24.    if {[GetVal 100 [Coords]] < $GDOP} {
25.        # take out fi coordinate from list
26.        set fi [expr [GetVal 37 [Coords]]]
27.        set fif [expr {round($fi / 100)}]
28.        set fip [expr {$fi - $fif * 100}]
29.        set fi [expr {$fif + $fip / 60}]
30.        # take out lambda coordinate from list
31.        set lambda [expr [GetVal 38 [Coords]]]
32.        regsub -- "0" $lambda "" lambda
33.        set lambdaf [expr {round($lambda / 100)}]
34.        set lambdap [expr {$lambda - $lambdaf * 100}]
35.        set lambda [expr {$lambdaf + $lambdap / 60}]
36.        # transformation from WGS84 to EOV
37.        set x [eovx $fi $lambda]
38.        set y [eovy $fi $lambda]
39.        set z [GetVal 39 [Coords]]
40.        # checking out of field
41.        if {$x < $xllcorner || $x > [expr {$xllcorner + $ncols *
                $cellsize}] || $y < $yllcorner || $y >
                [expr {$yllcorner + $nrows * $cellsize}]} {
42.           puts "GPS out of field of action"
43.           break
44.        }
45.        # searching bounding box
46.        set j 1
47.        while {$x>[expr {$xllcorner+($j-1)*$cellsize}]} {
48.           set j [expr {$j+1}]
49.        }
50.        set x1 [expr {$xllcorner+($j-2)*$cellsize}]
```

```
51.         set x2 [expr {$xllcorner+($j-2)*$cellsize}]
52.         set x3 [expr {$xllcorner+($j-1)*$cellsize}]
53.         set x4 [expr {$xllcorner+($j-1)*$cellsize}]
54.         set k 1
55.         while {$y>[expr {$yllcorner+($k-1)*$cellsize}]} {
56.           set k [expr {$k+1}]
57.         }
58.         set y1 [expr {$yllcorner+($k-2)*$cellsize}]
59.         set y2 [expr {$yllcorner+($k-1)*$cellsize}]
60.         set y3 [expr {$yllcorner+($k-1)*$cellsize}]
61.         set y4 [expr {$yllcorner+($k-2)*$cellsize}]
62.         set z1 $dtm([expr {$j-2}],[expr {$ncols-($k-2)}])
63.         set z2 $dtm([expr {$j-2}],[expr {$ncols-($k-1)}])
64.         set z3 $dtm([expr {$j-1}],[expr {$ncols-($k-1)}])
65.         set z4 $dtm([expr {$j-1}],[expr {$ncols-($k-2)}])
66.         # checking out of valid DTM happening
67.         if {$z1 == $nodatavalue || $z2 == $nodatavalue ||
                $z3 == $nodatavalue || $z4 == $nodatavalue} {
68.           puts "Not valid height"
69.           break
70.         }
71.         #  calculate  distances  between  box  points  and  current
            position
72.         set t1 [Distance {$x1 $y1 $x $y}]
73.         set t2 [Distance {$x2 $y2 $x $y}]
74.         set t3 [Distance {$x3 $y3 $x $y}]
75.         set t4 [Distance {$x4 $y4 $x $y}]
76.         set t12 [Distance {$x1 $y1 $x2 $y2}]
77.         set t23 [Distance {$x3 $y3 $x2 $y2}]
78.         set t34 [Distance {$x4 $y4 $x3 $y3}]
79.         set t14 [Distance {$x4 $y4 $x1 $y1}]
80.         # calculate the half circumference
81.         set s12 [expr {($t1+$t2+$t12)/2}]
82.         set s23 [expr {($t2+$t3+$t23)/2}]
83.         set s34 [expr {($t3+$t4+$t34)/2}]
84.         set s14 [expr {($t1+$t4+$t14)/2}]
85.         # calculate areas
86.         set T12 [expr {sqrt($s12*($s12-$t1)*($s12-$t2)*
                ($s12-$t12))}]
87.         set T23 [expr {sqrt($s23*($s23-$t2)*($s23-$t3)*
                ($s23-$t23))}]
88.         set T34 [expr {sqrt($s34*($s34-$t3)*($s34-$t4)*
                ($s34-$t34))}]
89.         set T14 [expr {sqrt($s14*($s14-$t1)*($s14-$t4)*
                ($s14-$t14))}]
90.         set T [expr {$t12*$t23}]
91.         # calculate triangle heights
92.         set m12 [expr {($T12*2)/$t12}]
93.         set m23 [expr {($T23*2)/$t23}]
94.         set m34 [expr {($T34*2)/$t34}]
95.         set m14 [expr {($T14*2)/$t14}]
96.         set suly1 [expr {$m23*$m34}]
97.         set suly2 [expr {$m34*$m14}]
98.         set suly3 [expr {$m12*$m14}]
99.         set suly4 [expr {$m12*$m23}]
100.        # calculate height difference
101.        set magassagkell [expr {(($z1*$suly1)+($z2*$suly2)+
                ($z3*$suly3)+($z4*$suly4))/$T}]
102.        set kulonbseg [expr {$magassagkell-$z}]
103.        puts $kulonbseg
104.    }
```

```
105.   else {
106.       puts Accuracy isn't enough
107.   }
108.  }
109.  }
```

2. Nmea.tcl

```
1. #//#
2. #  MNEA GPS handler
3. #  <p></p>
4. #  <p>Ulyxes - an open source project to drive total stations and
5. #              publish observation results</p>
6. #  <p>GPL v2.0 license</p>
7. #  <p>Copyright (C) 2010-2012 Zoltan Siki <siki@agt.bme.hu></p>
8. #  @author Zoltan Siki
9. #  @author Daniel Moka (TclDoc comments)
10.   #     @author Tamás Király
11.   #     @version 1.1
12.   #//#
13.   proc Coords {{timeout 5}} {
14.   set start [clock seconds]
15.   while {[expr {[clock seconds] - $start}] < $timeout} {
16.     set rec [GetLine]
17.     if {[regexp "^.GPGGA," $rec]} {
18.       set reclist [split $rec ",*"]
19.       set quality [lindex $reclist 6]
20.       if {[string first $quality "12345"] != -1} {
21.           set lat [DM2Rad [string trimleft [lindex $reclist 2]
                   "0"]]
22.           set lon [DM2Rad [string trimleft [lindex $reclist 4]
                   "0"]]
23.           set height [lindex $reclist 9]
24.           return [list [list 37 $lat] [list 38 $lon]
                  [list 39 $height]]
25.       }
26.     }
27.   }
28.   }
29.   proc GDOP {{timeout 5}} {
30.   set start [clock seconds]
31.   while {[expr {[clock seconds] - $start}] < $timeout} {
32.     set rec [GetLine]
33.     if {[regexp "^.GPLLK," $rec]} {
34.       set reclist [split $rec ",*"]
35.       set quality [lindex $reclist 6]
36.       if {[string first $quality "12345"] != -1} {
37.           set GDOP [lindex $reclist 9]
38.           return [list [list 100 $GDOP]]
39.       }
40.     }
41.   }
42.   }
```

3. blindOrientation2.tcl

```
1. #!/bin/sh
2. # the next line restarts using tclsh \
3. exec tclsh "$0" "$@"
4. #//#
5. #  This program search for a prism and set up orientation using
   Power Search
6. #  @param station station name in coordinate list
7. #  @param coo_file coordinate list in GeoEasy form
8. #
9. #  <p>Ulyses - an open source project to drive total stations and
10.   #                  publish observation results</p>
11.   #     <p>GPL v2.0 license</p>
12.   #     <p>Copyright (C) 2010-2013 Zoltan Siki
                      <siki@agt.bme.hu></p>
13.   #     @author Daniel Moka
14.   #     @author Zoltan Siki
15.   #     @author Tamas Kiraly
16.   #     @version 2.0
17.   source global.tcl
18.   source common.tcl
19.   source leica.tcl
20.   set debuglevel 1
21.   global argv argc
22.   global coo
23.   global PI2
24.   global PI
25.   set usage "Usage blindOrientation.tcl station coo_file
            <com_file> <debuglevel>"
26.   if {$argc < 2 } {
27.   puts $usage
28.   return
29.   }
30.   if {[LoadCoo [lindex $argv 1]]} {
31.   puts "Error in input file, line: $w"
32.   exit 1
33.   }
34.   set stat [lindex $argv 0]
35.   if {[info exists coo($stat)] == 0} {
36.   puts "Station not found in coordinate file"
37.   exit 1
38.   }
39.   # open and set communication port
40.   set compar "leica.com"
41.   if {$argc > 2} {
42.   set compar [lindex $argv 2]
43.   }
44.   OpenCom $compar
45.   set debuglevel 0
46.   if {$argc > 3} {
47.   set debuglevel [lindex $argv 3]
48.   }
49.   set dm 0.5  ;# Height torelance for search (+-)
50.   set dd 0.1  ;# Distance tolerance for search
51.   # set EDM mode & ATR
52.   SetEDMMode 2
53.   SetATR 1
54.   set dhz 0
55.   set vmin_limit [expr {30.0 / 180.0 * $PI}]
```

```
56.   set vmax_limit [expr {125.0 / 180.0 * $PI}]
57.   set vma $vmin_limit
58.   set vmi $vmax_limit
59.   # calculate distances and vertical directions
60.   foreach pn [array names coo] {
61.   if {$stat == $pn} { continue }
62.   set dist($pn) [Distance [GetVal 37 $coo($stat)]
            [GetVal 38 $coo($stat)] [GetVal 37 $coo($pn)]
            [GetVal 38 $coo($pn)]]
63.   set dh [expr {[GetVal 39 $coo($pn)] - [GetVal 39 $coo($stat)]}]
64.   set v_min($pn) [expr {atan2($dist($pn), $dh + $dm)}]
65.   set v_max($pn) [expr {atan2($dist($pn), $dh - $dm)}]
66.   if {$v_min($pn) < $vmi} { set vmi $v_min($pn) }
67.   if {$v_max($pn) > $vma} { set vma $v_max($pn) }
68.   }
69.   if {$vmi < $vmin_limit} { set vmi $vmin_limit }
70.   if {$vma > $vmax_limit} { set vma $vmax_limit }
71.   # First Measurement
72.   set found 0
73.   set lp [::Measure]
74.   if {[llength $lp] > 1} {
75.   set v [::GetVal 8 $lp]
76.   set hd [expr {[::GetVal 9 $lp] * sin($v)}]
77.   foreach pn [array names dist] {
78.     if {[expr {abs($dist($pn) - $hd)}] < $dd && $v >
           $v_min($pn) && $v < $v_max($pn)} {
79.        if {$debuglevel} { puts "Prism found at $hz $v $dist" }
80.        set found 1
81.        break
82.     }
83.   }
84.   }
85.   # Set Search Area
86.   set mer [::GetAngles]
87.   set dCenterHz [expr [GetVal 7 $mer] + $PI]
88.   while {$dCenterHz >= $PI2} { set dCenterHz [expr {$dCenterHz -
           $PI2}] }
89.   set dCenterV [expr {($vma+$vmi)/2.0}]
90.   set dRangeHz [expr {$PI2-0.001}]
91.   set dRangeV [expr {($vma-$vmi)}]
92.   set bEnabled 1
93.   if {[set res [::Send "%R1Q,9043:$dCenterHz,$dCenterV,$dRangeHz,
           $dRangeV,$bEnabled"]] != 0} {return $res}
94.   # search prism
95.   while {$found < 1} {
96.   if {[set res [::Send "%R1Q,9052:"]] != 0} {return $res}
97.   set lp [::Measure]
98.   set v [::GetVal 8 $lp]
99.   set hd [expr {[::GetVal 9 $lp] * sin($v)}]
100.  # which prism has found?
101.  foreach pn [array names dist] {
102.    if {[expr {abs($dist($pn) - $hd)}] < $dd && $v > $v_min($pn)
           && $v < $v_max($pn)} {
103.    if {$debuglevel} { puts "Prism found at $hz $v $dist" }
104.      set found 1
105.      break
106.    }
107.  }
108.  if {$found == 1} {break}
109.  # wrong prism found
110.  MoveRel 0.3 0
```

```
111.  set mer [::GetAngles]
112.  # Set search area again
113.  set dCenterHz [expr [GetVal 7 $mer] + $PI]
114.  while {$dCenterHz >= $PI2} { set dCenterHz [expr {$dCenterHz -
            $PI2}] }
115.  if {[set res [::Send "%R1Q,9043:$dCenterHz,$dCenterV,
           $dRangeHz,$dRangeV,$bEnabled"]] != 0} {return $res}
116.  }
117.  # Orientation
118.  if {$found} {
119.  if {$debuglevel} {
120.    puts "Orientation set"
121.  }
122.  set bearAn [Bearing [GetVal 38 $coo($stat)]
        [GetVal 37 $coo($stat)] [GetVal 38 $coo($pn)]
        [GetVal 37 $coo($pn)]]
123.  SetOri $bearAn
124.  }
125.  CloseCom
```

4. monitoring.bat

```
1. SET TCL=tclsh83
2. SET UMSG=Usage: station name, referencia, meres
3. SET STN=%1
4. SET REF=%2
5. SET MER=%3
6. :blind
7. %TCL% blindOrientation2.tcl %STN% %REF%.coo
8. IF NOT ERRORLEVEL == 0 GOTO usage
9. %TCL% freestation.tcl %REF%.geo
10.   IF NOT ERRORLEVEL == 0 GOTO usage
11.   :robot
12.   %TCL% robot.tcl %MER%.geo
13.   IF NOT ERRORLEVEL == 0 GOTO blind
14.   GOTO fine
15.   :usage
16.   ECHO %UMSG%
17.   GOTO fine
18.   :fine
```