HÖGSKOLAN
I GÄVLE

INSTITUTIONEN FÖR TEKNIK OCH BYGGD MILJÖ

Positionering av markörer för luftrörelser

- *handhållen längdmätare kontra laserskanning*

Positioning of markers for air movements

- *handheld distance meter vs laser scanning*

Daniel Nordström

September 2008

Examensarbetet på D-nivå, 15 hp

Geomatik

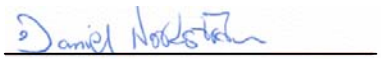
Geomatikprogrammet
Examinator: Stig-Göran Mårtensson
Handledare: Mats Sandberg, Högskolan i Gävle

Förord

Det här examensarbetet avslutar mitt magisterår på Geomatikprogrammet där studierna har varit inriktade mot geodetisk mätning. Chansen att skriva examensarbetet i ett ämne som innefattar flera olika mätningstekniker inklusive laserskanning fick jag via högskolan.

Jag vill tacka Mats Sandberg vid institutionen för teknik och byggd miljö, HiG för utmärkt handledning i för mig outforskade ämnen. Jag vill rikta ett stort tack till Stig-Göran Mårtensson som vetenskaplig handledare och examinator från HiG. Tack även till personalen i högskolans lokaler på Brynäs för stöd och nyttjandet deras lokalerna under studiens praktiska del.

Örebro, september 2008



Daniel Nordström

Sammanfattning

Luftens tillstånd (t.ex. temperatur och fuktighet) och rörelser i större lokaler kan mätas med hjälp av ett klimatmätsystem. För att luftens temperatur och fuktighet samt hastighet och riktning ska kunna simuleras i 3D måste sensorerna i klimatmätsystemet positioneras. I denna studie har två metoder som kan användas för positionering av objekt jämförts och utvärderats. En ena är enkel i sitt utförande och använder sig av en handhållen längdmätare. Objekts position beräknas genom triangulering av tre mätta längder till punkter med kända koordinater. Den andra metoden använder en terrester laserskanner och resultatet från den metoden kommer att anses som korrekt och utgöra studiens referens.

Syftet med studien är att undersöka om det med den enkla metoden går att uppnå centimeternoggrannhet vid positionering av objekt samt vilken markörstorlek som är lämplig att använda. Studien ska även innefatta utvecklingen av den mjukvara för att hantera mätdata och utföra positionsberäkningarna för en enkla metoden.

Fem kända punkterna mättes in med en Leica TPS1200 totalstation. 16 klotformade markörer med diameter från 20 till 50 mm monterades i 4 markörgrupper på stativ placerades ut i lokalen. Markörerna skannades med en Leica HDS3000 laserskanner och markörernas punktmoln modellerades för att erhålla deras positioner. Samtliga markörer mättes därefter in från de kända punkterna med en längdmätare Leica Disto Plus. Avstånden från de kända punkterna till respektive markör sparades i programvaran DistoPos som när längdmätningarna var gjorda även beräknade markörernas positioner. De båda metodernas resultat jämfördes, genom att beräkna den enkla metodens radiella noggrannhet, med resultatet från laserskanningen som referens. Den bästa noggrannheten uppnåddes när den enkla metodens positioner beräknades med avseende på avståndet till de kända punktera. När de tre kortaste avstånden användes för positionsberäkningen halverades onoggrannheten. Detta beror främst på att geometrin hos de i beräkningen ingående kända punkterna förändrats till det bättre. Detta understryker vikten av god geometri hos de kända punkterna som ligger till grund för positionsberäkningarna. Resultatet visar att det kommer att bli svårt att uppnå centimeternoggrannhet med metodens förutsättningar.

Markörernas diameter bör väljas till mellan 30 och 40 mm. Positionsberäkningarna för den enkla metoden skulle kunna förbättras ytterligare genom att inkludera någon typ av utjämning samt rutiner för att hitta och korrigera grova fel.

Abstract

The state of air (temperature and humidity) and movements in larger premises can be measured by a climate measuring system. In order to simulate air humidity, temperature, speed and direction in 3D the sensors in the climate measuring system needs to be positioned. In this study, two methods that can be used for positioning of objects are compared and evaluated. The first one is simple and uses a hand-held distance meter. The objects position are calculated by triangulation of three measured distances to points with known coordinates. The second method uses a terrestrial laser scanner and the result of this method will be considered as correct and constitute the study reference.

The purpose of the study is to examine whether it with the simple method is possible to achieve centimetre accuracy when positioning objects and determine what marker size that is appropriate to use. The study should also include the development of a software to manage measured data and perform the position calculations for the simple method.

Five known points were surveyed with a Leica TPS1200 total station. 16 spherical markers with the diameters from 20 to 50 mm were assembled in 4 marker groups and placed on stands. These markers were scanned with a Leica HDS3000 laser scanner and markers point cloud was model to obtain positions. All markers were then surveyed from the known points with a distance meter Leica Disto Plus. The distances from the known points to each marker were stored in the software DistoPos where the makers positions also were calculated.

The two methods result were compared by calculating the simple method radial accuracy with the results of the laser scanning as a reference. The best accuracy was reached when the simple method positions were calculated in terms of the distance to the known points. When the three shortest distances were used for positioning the unaccuracy was divide into halves. This is mainly because of the geometry of the known points included in calculations was changed to the better. This underlines the importance of good geometry of the known points included in the calculations. The accuracy of individual markers ranged between 3 and 36 mm. The result shows that it will be difficult to achieve centimetre accuracy with the methods prerequisites.

The position calculations for the simple method would be further enhanced by the inclusion of some kind of adjustment, and procedures to identify and correct gross errors.

Innehållsförteckning

1 Introduktion

1.1 Bakgrund

Det faktum att luften är osynlig gör att dess tillstånd (temperatur och fuktighet) och strömningar är svåra att analysera. Många studier har gjorts i ämnet. En vanlig teknik för att studera luftens rörelser är att sprida olika typer av spårbara partiklar i luften och registrera hur mycket de har flyttat sig inom ett visst tidsintervall (Settles, 1997). Sandberg, Lundström, Nilsson och Stymne (2008) och Sandberg (2007) beskriver tillgängliga metoder för att mäta luftens strömningar i sin helhet. Samtliga metoder är bildtekniker där sensorer som är känsliga för olika våglängder registrerar partiklarnas positioner efter olika tidsintervall. Dessa tekniker fungerar bara för begränsade områden så för större lokaler behövs andra metoder där mätningarna utförs punktvis. Vid punktvisa mätningar får man helheten genom att sätta mätdatat från punkterna i relation till varandra. En metod är att sprida ut en spårgas från en punkt och därefter mäta koncentrationen av gasen på andra punkter efter olika tidsintervall (Fischer et al., 2001). När olika metoder används tillsammans bildas ett nätverk av sensorer. Nätverket kallas klimatmätsystem och mäter luftens fuktighet och temperatur samt hastighet och riktning. Figur 1.1 visar en logger för temperatur och luftfuktighet. För att närmare studera luftens egenskaper och rörelser vill man vid *Avdelningen för teknik och byggd miljö, Högskolan i Gävle* installera klimatmätsystem i kyrkor. Detta för att kunna förändra kyrkors uppvärmnings- och ventilationssystem för att om möjligt sänka uppvärmningskostnaderna utan att i sin tur riskera att konditionen på kyrkans konstföremål försämras (Sandberg, 2008).

Figur 1.1. *Logger för temperatur och luftfuktighet.*

För att kunna göra en bra simulering behöver sensorernas positioner vara kända. I tidigare studier har man som standard använt sig av måttband, en metod om både är tidskrävande och ger dålig noggrannhet. I enklare studier har man till och med nöjt sig med att referera till något närbeläget föremål, till exempel en kristallkrona. I takt med att möjligheterna till simulering i 3D blir bättre höjs även kraven på noggrannheten i sensorernas positioner. Därför måste nya metoder för positionering av sensorer utarbetas (Sandberg, 2008).

I denna studie har två metoder som kan användas för positionering av objekt jämförts och utvärderats. Den ena är enkel i sitt utförande och använder sig av en handhållen laserbaserad längdmätare. Objekts position beräknas genom triangulering av tre uppmätta längder till punkter med kända koordinater. Den andra metoden använder avancerad utrusning i form av en terrester laserskanner. Resultatet från den senare metoden kommer att anses som korrekt och utgöra studiens referens. Studien utfördes i Högskolans mäthall belägen i stadsdelen Brynäs i Gävle, juni 2008. I lokalen finns det sedan tidigare ett byggplatsnät med känd noggrannhet etablerat.

1.2 Syfte

Syftet med studien är att utvärdera metoden där en handhållen längdmätare används för att positionerna objekt. Frågeställningar:

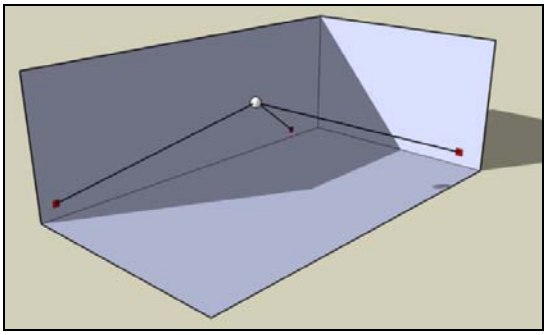
- Kan metoden positionera samtliga objekt med centimeternoggrannhet?
- Vad är en lämplig storlek på en markör för att sensorerna ska positioneras så bra som möjligt? Bör storleken varieras på olika avstånd?
- Är metoden praktiskt genomförbar i sensornätverk med 100 sensorer?
- Kommer metoden att kunna användas i framtida simuleringar? Finns det även andra användningsområden där den skulle kunna användas?

Studien kommer även att innefatta utvecklingen av en programvara som ska förenkla mätningsproceduren genom att lagra mätdata och beräkna objektens positioner samt längdmätningarnas precision.

2 Metod

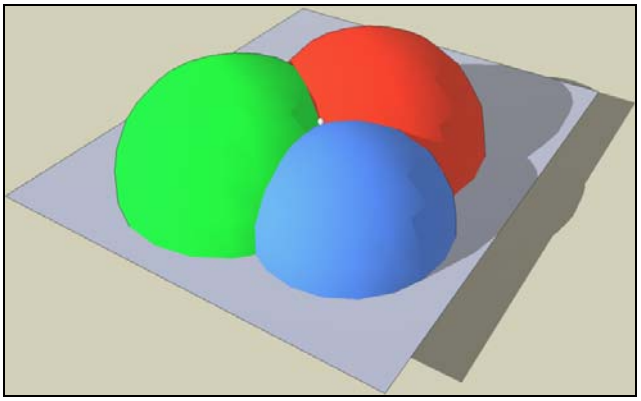
2.1 Principlösning

Principlösningen för att beräkna ett objekts koordinater i tre dimensioner med hjälp av avståndsmätningar är att mäta avståndet mellan objektet och tre punkter med redan kända koordinater. En grafisk illustration visas i figur 2.1. Punkternas koordinater blir kända genom att ett lokalt koordinatsystem upprättas på den plats där mätningarna ska utföras. Om de utförs inomhus kan ett hörn väljas till koordinatsystemets origo och koordinaterna i horisontalplanet refereras till det med hjälp av exempelvis ett måttband. Golvet i lokalen kan utgöra referens till objektets position i höjd. När de kända punkterna är positionsbestämda kan mätningarna påbörjas med att mätutrustningen placeras på en av punkterna. När avståndet till objektet har mätts och antecknats flyttas utrustningen till nästa kända punkt. När objektet är inmätt från tre kända punkter kan dess koordinater beräknas.



Figur 2.1. *Grafisk illustration av principlösningen för positionsbestämning av objekt med hjälp av avstånd till tre punkter med kända koordinater.*

2.2 Positionsberäkning



Figur 2.2. *Tre klot med olika radie som skär varandra. Vita pricken i mitten representerar objektet som ska positionsbestämmas.*

Positionsberäkningarna baseras på enkel geometri. Matematiskt kan problemet beskrivas som ”punkten där tre klot skär varandra” (se figur 2.2). De kända punkterna är lokaliserade i klotens centrum och klotens radier är avstånden till objektet.

Ett klot kan beskrivas enligt ekvation 1.

$$r^2 = x^2 + y^2 + z^2 \quad (1)$$

Ekvationer för tre klot med en gemensam skärningspunkt formar följande ekvationssystem (ekvation 2-4).

$$r_1^2 = (x-x_1)^2 + (y-y_1)^2 + (z-z_1)^2 \quad (2)$$
$$r_2^2 = (x-x_2)^2 + (y-y_2)^2 + (z-z_2)^2 \quad (3)$$
$$r_3^2 = (x-x_3)^2 + (y-y_3)^2 + (z-z_3)^2 \quad (4)$$

där centrum för de tre kloten $p_1(x_1, y_1, z_1)$, $p_2(x_2, y_2, z_2)$ och $p_3(x_3, y_3, z_3)$ samt avståndet till objektet (klotens radier) r_1, r_2 och r_3 är kända parametrar (The Math Forum, 2008a).
En av ekvationerna kan uteslutas genom att subtrahera den andra och tredje ekvationen från den första. Resultatet efter förenkling är ekvationerna 5 och 6.

$$r_2^2 - r_1^2 = 2(x_1 - x_2)x + x_2^2 - x_1^2 + 2(y_1 - y_2)y + y_2^2 - y_1^2 + 2(z_1 - z_2)z + z_2^2 - z_1^2 \quad (5)$$
$$r_3^2 - r_1^2 = 2(x_1 - x_3)x + x_3^2 - x_1^2 + 2(y_1 - y_3)y + y_3^2 - y_1^2 + 2(z_1 - z_3)z + z_3^2 - z_1^2 \quad (6)$$

Genom att multiplicera den första ekvationen med y_1-y_3 och den andra med y_1-y_2 och sen subtrahera de två erhålls ett uttryck utan y som beskriver x som en funktion av z. Samma sak gäller för att få fram y som en funktion av z.
Genom att substituera uttrycken för x(z) och y(z) i en av de första ekvationerna erhålls en andragradsekvation där endast kända parametrar beskriver z. När z är känd kan x och y lösas genom att gå tillbaka till det linjära ekvationssystemet. Programmeringsmässigt löses problemet genom att bryta ut koefficienter från den kvadratiska ekvationen som beskriver z (The Math Forum 2008b).

2.3 Kända punkter

Eftersom de kända punkterna utgör grunden för de båda metodernas mätningar inleddes studien med att mäta in dem. Till det användes en totalstation[1], Leica TPS1200. Mätpunkterna i lokalens byggplatsnät bestående av prismor och reflextejp användes för att etablera en fri station, där totalstationens placering är godtycklig eftersom stationens egen position är oviktig i sammanhanget. Byggplatsnätets koordinatsystem är ett så kallat 1000-1000-system som ofta används när man etablerar koordinatsystem över ett begränsat område t.ex. en lokal. De fem kända punkterna (A-E), markerade med körnarslag i golvet mättes därefter in med hjälp av ett miniprisma från Leica (se figur 2.3). Koordinaterna för de inmätta punkterna erhölls i byggplatsnätets koordinat-system och redovisas i bilaga A. En skiss över punkternas position finns i bilaga B.

Figur 2.3. *Leica miniprisma.*

2.4 Markörer

Figur 2.4. *En markörgrupp monterad på en bräda som i sin tur är monterade på ett stativ med stativstjärna.*

I en förstudie med längdmätaren monterad på ett kamerastativ gjordes ett antal inprickningsmätningar mot markörer av olika storlekar för att fastställa vilka storlekar som var lämpliga att ha med i studien. En anledning till att välja en så liten markör som möjligt är att vissa sensorer ska påverkas av objekt i sin närhet i så liten utsträckning som möjligt (Sandberg, 2008). Förstudien visade att markörer med diametern 20, 30, 40 och 50 mm var lämpliga att ha med i studien. Klotformade markörer valdes eftersom man då kan beräkna längden till markörens centrum från vilket håll som helst genom att lägga till klotets radie som en konstant. Markörerna, i form av träkulor monterades bredvid varandra på en bräda med ett mellanrum på ca 80 mm och på lite olika höjd enligt figur 2.4. Markörernas diameter varierade något i förhållande till den nominella diametern, därför mättes alla

[1] Geodetiskt mätinstrument som med hög noggrannhet mäter vertikalvinkel, horisontalvinkel och lutande längd mot ett optiskt reflekterande prisma. Instrumentet använder informationen för att beräkna prismats tredimensionella position.

markörer så att de kommande beräkningarna kunde baseras på faktisk diameter för respektive markör. Markörgrupperna monterades på stativ och placerades ut på olika avstånd i förhållande till punkt A, vilken skulle utgöra första mätpunkt i under längdmätningarna. Fyra grupper om fyra markörer i varje placerades på ca 1.5, 9, 16 och 22 m avstånd från punkt A. 1.5 m representerar den kortaste längden som kommer att behöva mätas och 22 m den längsta längden som lokalen tillät. En skiss över hur markörgrupperna var placerade under mätningarna finns i bilaga B.

2.5 Laserskanning

För att få en referens till den enkla metodens mätningar skannades sedan markörerna med en laserskanner[2], Leica HDS3000 (se figur 2.5). HDS3000 är en pulserande s.k. terrester laserskanner som har en noggrannhet på 6 och 4 mm i position respektive längd. Instrumentets styrka ligger i det höga antalet mätpunkter, något som vid modellering av objekt ger en noggrannhet på 2 mm för längder mellan 1 – 50 m (Leica Geosystems, 2008c). Ju fler mätpunkter ett objekt har desto högre noggrannhet. Laserskannern placerades på en godtycklig plats där det som skulle skannas kunde ses. Skannerns position i rummet blev känd genom att skanna sfäriska mål med kända koordinater. Målen placerades på stativ och centreras med hjälp av trefotens[3] optiska lod över de fyra kända punkterna A-D. Därefter skannades målen i hög upplösning (1 x 1 mm) och modellerades till klot där centrum för klotet tilldelades koordinater i plan och höjd. Höjden är den kända punktens höjd adderat med signalhöjden för respektive mål. Signalhöjden mättes med måttband. Den till laserskannern medföljande programvaran Cyclone 5.8 användes för att georeferera[4] målen till de kända koordinaterna. Inpassningen resulterade i ett *Root Mean Square* (RMS) på 3 mm för

Figur 2.5. *Laserskanner Leica HDS3000.*

[2] Instrument som med hjälp av laser kan mäta avståndet till reflekterande ytor. Avståndet beräknas utifrån tiden det tar för den reflekterade laserstrålen att återvända, s.k. time of flight (TOF). HDS3000 kan mäta avståndet till tusentals mätpunkter per sekund. Tillsammans skapar de många mätpunkterna en typ av 3D-bild över det skannade området.

[3] Adapter för att fästa instrument eller prisma på ett stativ. Med hjälp av doslibellen, det optiskta lodet, de tre fotskruvarna och stativets ben hotisonteras och centreras intrumentet eller prismat över punkten.

[4] Laserskannerns interna koordinatsystem transformeras till ett annat koordinatsystem, i detta fall lokalens byggplatsnät. Ett samband upprättas genom t.ex. en Helmert-transformation med hjälp av minst tre för de båda koordinatsystemen gemensamma punkter. När sambandet är upprättat kan övriga koordinater transformeras.

punkterna A och D, och 1 mm för punkt B och C. Hur georeferering går till beskrivs i Reshetyuk (2007). När skannerns position i rummet var känd kunde markörerna skannas. Samtliga markörer skannades i hög upplösning (1 x 1 mm) och modellerads i Cyclone till klot. Koordinater för klotens centrum samt klotens diameter antecknades inför den kommande jämförelsen. Skärmdumpar som visar punktmolnen från laserskanningen visas i bilaga E.

2.6 Längdmätare

I den enkla metoden användes en längdmätare, Leica Disto Plus. Längdmätaren är en klass 2 laserprodukt och kan mäta avstånd upp till 30 meter utan speciella måltavlor. Den har enligt tillverkaren en noggrannhet på 1.5 mm. Dessa värden verifierades av ett EDM-test (Electronic Distance Measurement). Under testet mättes längder på 10, 20 och 30 meter och visade att det konstanta felets medelfel hos längdmätarens är 0.5 mm. Hur EDM-testet utfördes redovisas i bilaga D. Längdmätarens laser är röd till färgen med en våglängd på 620-690 nm och en maxstyrka på 0.95 mW (Leica Geosystems, 2008b). Avståndet (R_d) till objektet beräknas enligt ekvation 7,

$$R_d = c \times t/2 \qquad (7)$$

där c är ljusets hastighet och t är tiden det tar för laserstrålen att återvända, *time of flight* (TOF).

Via standardinfästningen på undersidan monterades längdmätaren på ett vanligt kamerastativ. Eftersom stativets kulled är placerad under själva infästningen konstruerades en adapter som monterades mellan längdmätaren och stativet enligt figur 2.6. Adapterns höjd sänker längdmätaren med 69 mm så att avståndsmätningen görs i linje med stativets kulled. Adaptern förkortar också det uppmätta avståndet med adapterns längd, något som måste tas hänsyn till i beräkningarna. Adapterns längd, avståndet från längdmätarens bakersta ände (längdmätningarnas utgångspunkt) till centrum på stativets kulled, mättes med skjutmått till 60 mm. Figur 2.10 visar hur adapterns längd (instrumentkonstanten)

Figur 2.6. *Ländmätaren Leica Disto Plus monterad med adaptern på kamerastativet.*

används i positionsberäkningarna. Stativets tre ben fästes på en stativstjärna[5] för att ge önskad stabilitet.

2.7 Handdator och DistoPos

För att hantera mätdata och sköta beräkningen användes en handdator eller *Personal Digital Assistant* (PDA), Fujitsu Siemens Loox N560 (se figur 2.7). Både handdatorn och längdmätaren har stöd för kommunikation via Bluetooth vilket utnyttjades för att skicka längdinformation under mätningarna.

Figur 2.7. *Handdator Fujitsu Siemens Loox N560.*

Programvaran som kördes i handdatorn är utvecklad som en del i studien och specialskriven för ändamålet. Programvaran har namnet DistoPos och har funktionalitet för att lagra de kända punkternas koordinater, ta emot och hantera relationen mellan känd punkt och objekt samt utföra beräkningen av objektens positioner och längdmätningarnas precision. En närmare beskrivning av programmet DistoPos återfinns i bilaga H. Programkoden redovisas i bilaga X.

2.8 Längdmätningar

Längdmätaren inklusive adapter monterades på kamerastativet och placerades på punkt A där stativet horisonterades med hjälp av en doslibell och centrerades med ett snörlod enligt figur 2.8.

Figur 2.8. *Doslibell och snörlod monterat på kamerastativets mittstång.*

Innan mätningarna påbörjades skrevs de kända punkteras koordinater in i handdatorn. Markörerna numrerades efter markörgruppens ordningsföljd och markörens storlek: tiotalssiffrorna (10-40) anger gruppens nummer och entalssiffrorna markörens diameter i centimeter (2-5). Avståndet till markörerna mättes därefter in med start från 20 mm-markören i markörgrupp 1 (markör 12), gruppen närmast punkt A. Även markörernas signalkonstant (markörens radie), mättes och registrerades i handdatorn. Avstånden till

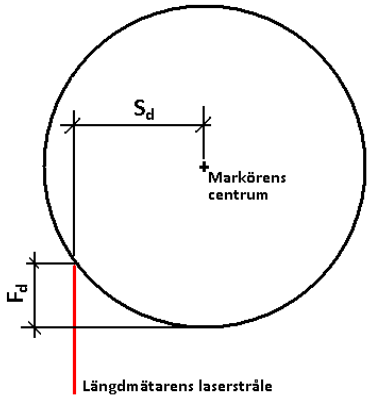
[5] Utrustning för att fixera stativets ben på plana hårda ytor där det inte är möjligt att trycka ner stativbenens spetsar i marken.

varje markör mättes 3-5 gånger och det uppskattade medelvärdet sparades i handdatorn samt antecknades i protokoll. När samtliga markörer var inmätta från punkt A flyttades utrustningen till punkt B där proceduren upprepades med skillnaden att informationen om de kända punkterna inte behövde matas in igen. Inte heller markörernas signalkonstant behövde matas in igen. Efter B mättes längderna från punkt C, D, och E. På varje punkt antecknades instrumenthöjden; avståndet mellan golvet vid den aktuella punkten och centrum på stativets kulled. Även den informationen sparades i handdatorn. Sambandet mellan avstånd, höjder och konstanter beskrivs i avsnittet *beräkningar* nedan. Mätdata från längdmätningarna redovisas i bilaga C.

Eftersom markörerna är klotformade riskerar längdmätningarna att bli längre om inte markören träffas på mitten av laserstrålen. Risken ökar med markörens radie men felet växer inte linjärt från mitten till markörens ytterkant utan enligt ekvation 8,

$$F_d = r - \sqrt{(r^2 - S_d^2)} \qquad (8)$$

där F_d är felet i längdmätningen, r är markörens radie och S_d är det mot längdmätningens riktning vinkelräta avståndet från mitten av markören till punkten dit avståndet är mätt. En grafisk illustration visas i figur 2.9.



Figur 2.9. *Grafisk illustration över parametrarna i ekvation 8.*

2.9 Beräkning

Innan beräkningen av markörernas position kan påbörjas behöver några variabler, konstanter och dess samband definieras:

K_h = Den kända punktens höjd

I_h = Instrumenthöjden

P_h = Höjden varifrån längdmätningen utgår

I_c = Instrumentkonstanten, adapterns längd (60 mm)

M_d = Den uppmätta längden till markören

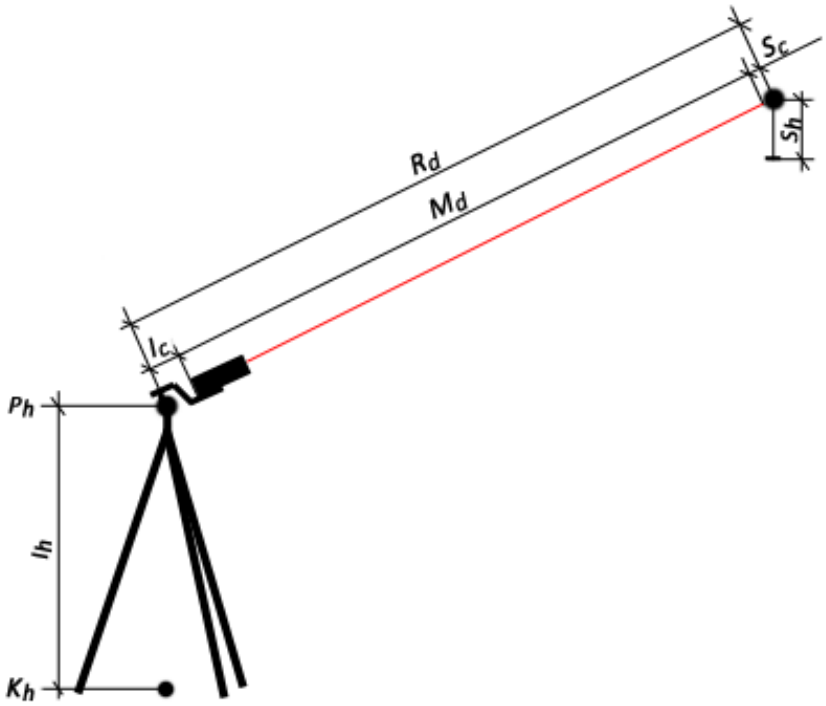
S_c = Signalkonstanten, markörens radie (10-25 mm)

R_d = Avståndet mellan stativets kulled och markörens centrum

Variabler och konstanter har samband enligt ekvation 9 och 10. Sambandet visas även grafiskt i figur 2.10.

$$P_h = K_h + I_h \quad (9)$$

$$R_d = I_c + M_d + S_c \quad (10)$$

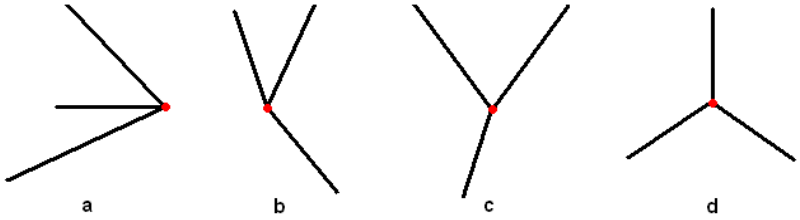


Figur 2.10. *Grafisk redovisning av sambandet mellan variabler och konstanter som ingår i positionsberäkningarna för den enkla metoden.*

Programmet DistoPos ger även användaren möjligheten att ange en signalhöjd (S_h) för varje markör för att markören ska kunna placeras över eller under sensorn som ska positioneras. Signalhöjden adderas med sitt tecken till z-koordinaten efter att beräkningen är klar. I den här studien sattes signalhöjden till 0 eftersom det var markörernas position som skulle beräknas. Markörernas positioner beräknades i handdatorn enligt metoden beskriven i avsnitt 2.2. Positionerna exporterades till en semikolonseparerad textfil som lästes in i Excel för vidare analys. Exempel på en exportfil visas i bilaga I.

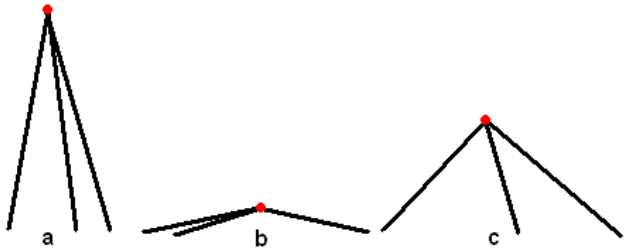
2.10 Geometri

Nätets geometri är av största vikt när det kommer till den enklare metodens noggrannhet. Som vid geodetisk inbindning ska längdmätningarna vara gjorda från så vitt skilda håll som möjligt för att uppnå maximal noggrannhet. Ju sämre geometri de kända punkterna har i relation till objektet som ska positioneras, desto mer påverkar enstaka längdmätningarna resultatet av positionsberäkningen. Figur 2.11 visar exempel på bra och dålig horisontell geometri. En tumregel är att inte låta vinkeln mellan två kända punkter bli större än 180° eller 200 gon.



Figur 2.11. *Exempel på bra och dålig horisontalgeometri. (a) dålig, (b) mindre bra, (c) bra och (d) mycket bra.*

Även nätets vertikala geometri har inverkan på metodens noggrannhet. Nätet som bildas av de kända punkterna och objektet bör inte vara ”flackt” men inte heller ”spetsigt”. Figur 2.12 visar exempel på bra och dålig vertikalgeometri.



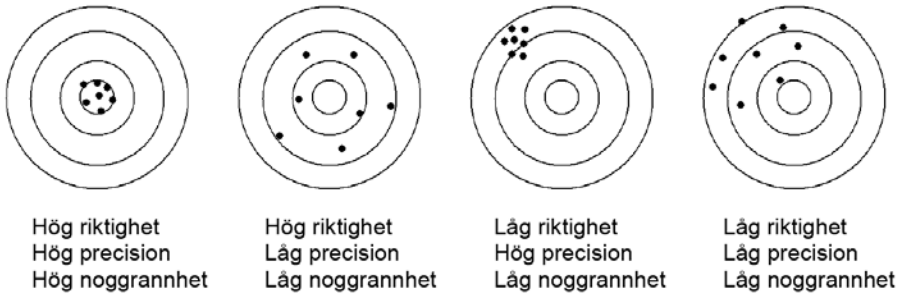
Figur 2.12. *Exempel på bra och dålig vertikalgeometri. (a) dålig,(b) dålig, (c) bra.*

2.11 Noggrannhet, precision och riktighet

Tre begrepp som används inom mätningen är noggrannhet, precision och riktighet. Noggrannhet är mätningarnas medelfel och anger hur nära ett mätvärde är från sitt sanna värde. Noggrannheten kan beräknas radiellt enligt ekvation 11 och kallas även *Root Mean Square* (RMS).

$$s_{XYZ} = \sqrt{(s_X^2 + s_Y^2 + s_Y^2)} \qquad (11)$$

där s_X, s_Y och s_Z är avvikelsen i respektive led. Om noggrannheten är ett mått på mätningarnas avvikelse från sitt *sanna värde* anger precisionen mätningarnas spridning runt sitt *medelvärde*. När grova och systematiska fel är eliminerade så visar precisionen de slumpmässiga felens spridning som vid upprepade mätningar är normalfördelad. Mätningarnas riktighet avgörs av både noggrannheten och precisionen och anger mätvärdenas genomsnittliga överensstämmelse med det sanna värdet. För att uppnå hög riktighet krävs att både noggrannheten och precisionen är hög (se figur 2.13).



Figur 2.13*. Sambandet mellan noggrannhet, precision och riktighet (Lilje, 2008).*

3 Resultat

3.1 Laserskanning

Resultatet från inpassningen i samband med att laserskannern georefererades visas i tabell 3.1.

Tabell 3.1. *Inpassningens fel hos mål A-D i respektive led samt inpassningens radiella noggrannhet.*

mål	Δx	Δy	Δz	noggrannh.
A	0.000	0.003	0.000	0.003
B	0.001	0.000	0.000	0.001
C	-0.001	0.000	0.000	0.001
D	0.000	-0.003	0.000	0.003

De skannade markörernas punktmoln modellerades till klot vars koordinater för origo presenteras i tabell 3.2. Där visas även de modellerade klotens diameter och markörernas mätta diameter samt skillnaden mellan dessa.

Tabell 3.2. *Markörernas positioner erhållna från laserskanningens modellerade punktmoln. Även modellernas diameter och markörernas uppmätta diameter samt skillnaden mellan dem visas i kolumnerna till höger.*

markör	x	y	z	skan.diam. (m)	mätt diam. (m)	Δdiameter (m)
12	1026.428	1120.043	1.250	0.023	0.020	0.003
13	1026.418	1120.121	1.295	0.032	0.029	0.003
14	1026.403	1120.204	1.300	0.040	0.039	0.001
15	1026.393	1120.283	1.352	0.049	0.048	0.001
22	1033.752	1121.008	1.360	0.022	0.019	0.003
23	1033.763	1121.086	1.384	0.032	0.028	0.004
24	1033.779	1121.166	1.410	0.042	0.040	0.002
25	1033.787	1121.249	1.457	0.052	0.048	0.004
32	1040.856	1121.957	1.297	0.022	0.020	0.002
33	1040.845	1122.030	1.340	0.032	0.029	0.003
34	1040.834	1122.108	1.342	0.040	0.039	0.001
35	1040.822	1122.194	1.380	0.050	0.048	0.002
42	1046.850	1123.007	1.292	0.021	0.019	0.002
43	1046.837	1123.085	1.317	0.031	0.028	0.003
44	1046.827	1123.169	1.342	0.041	0.040	0.001
45	1046.816	1123.247	1.389	0.051	0.048	0.003

3.2 Den enkla metoden

De båda metodernas resultat har jämförts genom att titta på skillnaden i x-, y- och z-led. Den radiella noggrannheten som visar avståndet mellan de båda metodernas positioner beräknades. Markörernas positioner beräknades först baserat på punkterna A, B och C. Punkt D användes för att beräkna längdmätningarnas precision. Resultatet visas i tabell 3.3.

Tabell 3.3. *Markörernas positioner från beräkningar baserade på de kända punkterna A, B och C. Punkt D användes för att beräkna längdmätningarnas precision. I kolumnerna till höger visas skillnaden i respektive led mot laserskanningens koordinater och den enkla metodens radiella noggrannhet.*

markör	x	y	z	prec. (m)	Δx (m)	Δy (m)	Δz (m)	noggrannh. (m)
12	1026.432	1120.046	1.245	0.003	-0.004	-0.003	0.005	0.007
13	1026.422	1120.120	1.285	0.003	-0.004	0.001	0.010	0.011
14	1026.407	1120.201	1.304	0.001	-0.004	0.003	-0.004	0.006
15	1026.399	1120.279	1.349	0.002	-0.006	0.004	0.003	0.008
22	1033.755	1121.009	1.347	0.004	-0.003	-0.001	0.013	0.013
23	1033.768	1121.089	1.331	0.006	-0.005	-0.003	0.053	0.053
24	1033.782	1121.168	1.369	0.005	-0.003	-0.002	0.041	0.041
25	1033.791	1121.251	1.426	0.005	-0.004	-0.002	0.031	0.031
32	1040.857	1121.952	1.309	0.005	-0.001	0.005	-0.012	0.013
33	1040.846	1122.027	1.319	0.002	-0.001	0.003	0.021	0.021
34	1040.836	1122.096	1.431	0.028	-0.002	0.012	-0.089	0.090
35	1040.824	1122.186	1.412	0.011	-0.002	0.008	-0.032	0.033
42	1046.853	1123.001	1.308	0.004	-0.003	0.006	-0.016	0.017
43	1046.838	1123.077	1.336	0.005	-0.001	0.008	-0.019	0.021
44	1046.828	1123.160	1.365	0.003	-0.001	0.009	-0.023	0.025
45	1046.816	1123.241	1.401	0.003	0.000	0.006	-0.012	0.013

När positionsberäkningen baserad på punkterna A, B och C inte gav en noggrannhet på under 10 mm gjordes fler beräkningar som baserades på andra kända punkter. Det bästa resultatet uppnåddes när de kända punkterna valdes efter avstånd till respektive markör. I tabell 3.4 visas resultatet för beräkningarna baserade på de tre närmaste punkterna för respektive markör.

Tabell 3.4. *Markörernas positioner från beräkningar baserade på de kortaste längdmätningarna till respektive markör. Längdmätningarnas precision beräknas på den fjärde kortaste längdmätningen och redovisas efter plustecknet i kolumnen punkter. I kolumnerna till höger visas skillnaden i respektive led mot laserskanningens koordinater och den enkla metodens radiella noggrannhet.*

markör	x	y	z	prec. (m)	punkter	Δx (m)	Δy (m)	Δz (m)	noggrannh. (m)
12	1026.433	1120.047	1.247	0.004	AEB+D	-0.005	-0.004	0.003	0.007
13	1026.423	1120.123	1.287	0.004	AEB+D	-0.005	-0.002	0.008	0.010
14	1026.410	1120.211	1.310	0.006	AEB+D	-0.007	-0.007	-0.010	0.014
15	1026.401	1120.285	1.351	0.006	AEB+D	-0.008	-0.002	0.001	0.008
22	1033.756	1121.004	1.396	0.003	BED+A	-0.004	0.004	-0.036	0.036
23	1033.768	1121.082	1.402	0.004	BED+A	-0.005	0.004	-0.018	0.019
24	1033.781	1121.163	1.410	0.001	BED+A	-0.002	0.003	0.000	0.004
25	1033.792	1121.244	1.484	0.004	BED+A	-0.005	0.005	-0.027	0.028
32	1040.857	1121.954	1.295	0.000	DCB+E	-0.001	0.003	0.002	0.004
33	1040.847	1122.026	1.326	0.000	DCB+E	-0.002	0.004	0.014	0.015
34	1040.831	1122.108	1.367	0.002	DCB+E	0.003	0.000	-0.025	0.025
35	1040.822	1122.191	1.387	0.000	DCB+E	0.000	0.003	-0.007	0.008
42	1046.850	1123.009	1.287	0.002	CDB+A	0.000	-0.002	0.005	0.005
43	1046.835	1123.087	1.308	0.002	CDB+E	0.002	-0.002	0.009	0.009
44	1046.825	1123.166	1.347	0.001	CDB+E	0.002	0.003	-0.005	0.006
45	1046.814	1123.246	1.387	0.001	CDB+E	0.002	0.001	0.002	0.003

För att göra studien mer realistisk gjordes även beräkningar där de kända punkternas höjd sattes till 0. Detta för att simulera hur resultatet påverkas om golvet förutsätts vara helt plant, något som krävs utan inmätning med hjälp av totalstation. Resultatet visas i tabell 3.5.

Tabell 3.5. *Markörernas positioner från beräkningar baserade på de kortaste längdmätningarna till respektive markör men med de kända punkternas höjder satta till noll. Längdmätningarnas precision beräknas på den fjärde kortaste längdmätningen och skrivs efter plustecknet i kolumnen punkter. I kolumnerna till höger visas skillnaden i respektive led mot laserskanningens koordinater och den enkla metodens radiella noggrannhet.*

markör	x	y	z	prec. (m)	punkter	Δx (m)	Δy (m)	Δz (m)	noggrannh. (m)
12	1026.433	1120.047	1.251	0.004	AEB+D	-0.005	-0.004	-0.001	0.006
13	1026.423	1120.122	1.291	0.004	AEB+D	-0.005	-0.001	0.004	0.006
14	1026.410	1120.210	1.314	0.006	AEB+D	-0.007	-0.006	-0.014	0.017
15	1026.401	1120.285	1.355	0.006	AEB+D	-0.008	-0.002	-0.003	0.009
22	1033.756	1121.003	1.399	0.003	BED+A	-0.004	0.005	-0.039	0.040
23	1033.769	1121.081	1.405	0.004	BED+A	-0.006	0.005	-0.021	0.022
24	1033.782	1121.162	1.414	0.001	BED+A	-0.003	0.004	-0.004	0.006
25	1033.792	1121.243	1.488	0.004	BED+A	-0.005	0.006	-0.031	0.032
32	1040.857	1121.954	1.299	0.000	DCB+E	-0.001	0.003	-0.002	0.004
33	1040.847	1122.025	1.330	0.000	DCB+E	-0.002	0.005	0.010	0.011
34	1040.831	1122.107	1.371	0.002	DCB+E	0.003	0.001	-0.029	0.029
35	1040.822	1122.190	1.392	0.000	DCB+E	0.000	0.004	-0.012	0.013
42	1046.850	1123.008	1.294	0.002	CDB+A	0.000	-0.001	-0.002	0.002
43	1046.835	1123.086	1.315	0.002	CDB+E	0.002	-0.001	0.002	0.003
44	1046.825	1123.165	1.354	0.001	CDB+E	0.002	0.004	-0.012	0.013
45	1046.814	1123.246	1.395	0.001	CDB+E	0.002	0.001	-0.006	0.006

3.3 Arbetssätt

Arbetssättet att utföra längdmätningar mot samtliga markörer från en känd punkt i taget fungerade över förväntan. Längdmätningarna från den första punkten tog längre tid eftersom respektive markörs radie då ska matas in i handdatorn. Från och med den andra punkten tog uppställningen på känd punkt och längdmätningarna mot de 16 markörerna ca 20 minuter.

Momenten i den enkla metoden bör utföras enligt ordningen i tabell 3.6. En utrustningslista för metoden redovisas i bilaga F.

Tabell 3.6. *Momenten i den enkla metoden.*

Moment	Aktivitet	Kommentar
1.	Utplacering av sensorer och markörer	
2.	Namngivning av markörer	Extra viktigt i klimatmätsystem med många sensorer.
3.	Planering och utplacering av kända punkter	Samtliga markörer ska kunna mätas in från minst tre kända punkter som tillsammans bildar god geometri.
4.	Etablera koordinatsystem	1000-1000 system över området/lokalen där mätningarna ska utföras.
5.	Inmätning av kända punkter	Jämför uppmätt och beräknad diagonal till origo med hjälp av Pythagoras sats.
6.	Längdmätningar från kända punkter	Horisontera och centrera stativet över den kända punkten. Anteckna instrumenthöjden. Mät avstånden till mitten av markörerna. Spara mätdatat i handdatorn. Upprepa för samtliga kända punkter.
7.	Positionsberäkningar	Utför positionsberäkningarna. Kontrollera längdmätningarnas precision.
8.	Eventuella kompletterade längdmätningar	Utför kompletterande längdmätningar för de objekt vars position inte kunde beräknas eller där längdmätningarnas precision blev sämre än önskat.
9.	Exportera koordinaterna till textfil	För vidare analys och simulering i annan mjukvara.

3.4 Storlek på markör

För att svara på frågan om vad som är en lämplig storlek på en markör gjordes en serie inriktningsförsök med längdmätaren mot markörerna. För att uppskatta hur lång tid det tog beroende på markörernas storlek och avstånd, riktades längdmätaren in mot samtliga markörer med hjälp av stativets panoreringshandtag. Nästan alla markörer siktades in på under 10 sekunder. Försöket visade att varken markörernas storlek eller avstånd hade någon större inverkan på hur svåra de var att träffa. Bara mot 20- och 30 mm-markörerna i grupp 4, ca 22 m bort, blev inriktningstiden något längre.

4 Diskussion

4.1 Noggrannhet

Noggrannheten hos de kända punkternas koordinater är avgörande för noggrannheten hos objektets beräknade koordinater; hög noggrannhet hos utgångspunkterna ger bättre förutsättningar för en hög noggrannhet hos nypunkterna. För att verifiera en känd punkts koordinater i plan kan mätningarna från den punkten inledas med en horisontell avståndsmätning till ett hörn i lokalen. Det uppmätta avståndet kan jämföras med teoretiska som beräknas med hjälp av Pythagoras sats.

Punkterna som användes som kända punkter i den enkla metoden kan anses ha millimeternoggrannhet. Det baseras på de omfattande mätningarna som ligger till grund för byggplatsnätets utjämning samt utrustningens noggrannhet (Leica Geosystems, 2008a).

Som studiens referens är noggrannhetskraven på laserskanningen höga. Dålig noggrannhet i referensmätningarna hade gjort studiens enkla metod svår, för att inte säga omöjlig att analysera. Ett mått på laserskanningens noggrannhet är felet som uppstår när de skannade punkterna georefereras till kända koordinater. Detta görs i programvaran Cyclone och visar på hur de skannade punkterna och de kända koordinaterna passar ihop. Cyclone försöker passa in de skannade målens centrum till givna koordinater och då uppstår alltid ett visst fel. I den här studien skannades fyra sfäriska mål, ett mer än vad som är tillräckligt för en inpassning. Resultatet av georefereringen visas i tabell 3.1. Möjliga felkällor kan vara att målen inte varit tillräckligt noggrant centrerade över punkten samt eventuell onoggrannhet i de punktmoln som modelleringen av målen grundas på. Inpassningen anses som mycket bra, särskilt i z-led.

Ett annat mått på laserskannerns noggrannhet är hur bra de modellerade markörernas diametrar överensstämmer med de verkliga. Jämförelsen visas i tabell 3.2 och skillnaden är som störst 4 mm för markörerna 23 och 25. Att samtliga modeller har större diameter än sin respektive markör, beror på att pulser som träffar markörens kant endast reflekteras till viss del. Pulsens centrum kan missa målet men det är ändå där träffen kommer att registeras. Kantträffarna gör därför att modellen blir något större. Hur bra modellerna överensstämmer med verkligheten avgörs även av antalet punkter som respektive modell baseras på, ju fler träffar desto bättre eftersom varje enskild punkt då får mindre betydelse. Detta är en anledning till att modellerna av de större markörerna borde ha bättre noggrannhet. Tabell 3.3 visar att även den radiella noggranheten är som lägst för

markörgrupp 2 vilket tyder på att laserskanningens noggrannhet är som lägst för denna grupp. En möjlig anledning till detta är att markörgrupp 2 stod närmast skannern.
Felkällorna i den enkla metoden är både fler och, i jämförelse med den andra metoden, större eftersom utrustningen som användes från början inte är ämnad för precisionsmätningar. Kamerastativet som användes var från början instabilt men det åtgärdades genom att benen fästes på en stativstjärna med buntband. Snörlodet som användes vid centreringen över punkten är inte lika noggrant som ett optiskt lod, men kombinationen snörlod och doslibell uppskattas klara av att hålla eventuellt centreringsfel under 3 mm. Eftersom centrum på stativets kulled inte var utmärkt utan fick uppskattas ger instrumentkonstanten (avståndet mellan centrum på stativets kulled och längdmätarens bakersta ände) utrymme för ett eventuellt fel som uppskattas till maximalt 2 mm.

4.2 Analys

När positionsberäkningarna baserade på punkterna A, B och C inte resulterade i önskad noggrannhet valdes de kända punkterna istället med avseende på avståndet till respektive markör. När beräkningarna baserades på de tre kända punktera med de kortaste avstånden från respektive markör, mer än halverades den radiella onoggrannheten. Den radiella noggrannheten för enskilda markörer varierade mellan 3 och 36 mm men bara fyra värden var högre än 15 mm. Resultatet visas i tabell 3.3. För att ta reda på anledningen till förbättringen analyserades vilka punkter som beräkningarna baserats på. Analysen visade att horisontalgeometrin var den bästa möjliga för samtliga markörgrupper, som om punkterna hade valts för hand. Förbättringen berodde alltså inte på avstånden till de kända punkterna hade blivit kortare utan snarare på att geometrin förändrats till det bättre.
Beräkningarna för den enkla metoden kunde förändras tack vare möjligheten att jämföra resultatet för metoderna, något som inte hade varit möjligt utan en referens. Ett sätt att utvärdera resultatet från den enkla metoden utan referens skulle kunna vara att titta på längdmätningarnas precision.
Det bästa resultatet skulle antagligen uppnås genom att göra en utjämning baserad på samtliga längdmätningar till respektive markör. Navidi et al. (1998) presenterar ett antal statistiska metoder som är lämpliga att använda vid positionsbestämning med hjälp av triangulering.
Försöket att göra studien mer realistisk genom att sätta de kända punkternas höjd till 0 försämrade inte noggrannheten nämnvärt, för sex av markörerna blev noggrannheten till och med bättre. Det gör att den enkla metoden har goda förutsättningar att fungera även när inmätningen av de kända punkterna görs utan geodetiska instrument.

5 Slutsatser

Studien visar att det är svårt att uppnå centimeternoggrannhet med beskrivna metod, särskilt om mätningarna utförs av personer med ringa mätningsteknisk kompetens. Metoden att basera beräkningarna på de tre närmaste punkterna förbättrar inte noggrannheten utan understryker snarare vikten av god geometri hos de kända punkterna, något som bör väljas med omsorg och planeras noga innan mätningarna påbörjas. En noggrannhet under 30 mm är mer realistiskt med metodens förutsättningar.
Markörernas diameter bör väljas till mellan 30 och 40 mm därför att mindre markörer blir svåra att träffa på avstånd längre än 20 m. Med större markörer ökar risken för att längdmätningarna ska bli felaktiga på grund av att lasern inte siktas mot markörens mitt vid mättillfället.
Arbetssättet och DistoPos användarvänliga gränssnitt ger goda förutsättningarna för att metoden ska kunna användas i nätverk med ett hundratal sensorer. För att kunna hålla ordning på vilken sensor som står på tur att mätas in bör någon form av namnstandard och tydlig märkning av sensorerna och dess markörer tillämpas.

5.1 Framtiden

Potentiella förbättringspunkter för det enkla metoden:

- Noggrannheten i resultatet skulle kunna förbättras genom att utveckla positionsberäkningen för att inkludera någon typ av utjämning samt rutiner för att hitta och korrigera grova fel.
- Beräkningsproceduren kan vidareutvecklas för att även beräkna vilka kända punkter som skapar bäst geometri för respektive objekt och använda dem för att beräkna objektets position.
- Metoden skulle även behöva innehålla instruktioner hur man etablerar ett koordinatsystem för kända punkterna utan att använda sig av geodetiska instrument.
- Vid positionsbestämning av hastighetsgivare behöver även givarens orientering bestämmas. Detta skulle kunna göras med en riktningsmarkör som positionsbestäms på samma sätt som andra markörer men med skillnaden att DistoPos även beräknar givarens orientering med hjälp av hur markören och dess riktningsmarkör relaterar till varandra.

Under studiens gång kom det fram nya uppslag på sammanhang då metoden skulle kunna användas. Metoden skulle kunna användas till att beräkna tjockleken på föremål där direkta mätmetoder inte fungerar. Ett exempel är murar där tjockleken ofta varierar med höjden över marken.

Referenser

FIG (1994) The International Federation of Surveyors (FIG); *Recommended Procedures for Routine Checks of Electro-Optical Distance Meters (EDM)*. Publication no. 9, A technical monograph. Australian Government Publishing Service.

Fischer, M . L., Price, P. N., Thatcher, T. L., Schwalbe, C. A., Craig, M. J., Wood, E. E., Sextro, R. G. & Gadgil, A. J. (2001) Rapid measurements and mapping of tracer gas concentrations in a large indoor space, *Atmospheric Environment*, vol. 35, s. 2837-2844.

Leica Geosystems (2008a) *Leica TPS1200+ Series – Technical Data*. Tillgänglig på: http://www.leica-geosystems.com/common/shared/downloads/inc/downloader.asp?id=2655 (hämtad 2008-06-26).

Leica Geosystems (2008b) *Leica Disto Plus – User manual*. Tillgänglig på: http://www.leica-geosystems.com/cpd/en/support/lgs_6598.htm?cid=3985 (hämtad: 2008-06-26).

Leica Geosystems (2008c) *Leica HDS3000 – Product Specifications*. Tillgänglig på: http://www.leica-geosystems.com/hds/en/Leica_HDS3000.pdf (hämtad: 2008-06-26).

Lilje, M. (2008) *Vad är rätt och vad är fel?* Tillgänglig på: http://www.lantmateriet.se/upload/filer/kartor/geodesi_gps_och_detaljmatning/Rapporter-Publikationer/Publikationer/Felteori_artikel_SINUS_2004_nr_3.pdf (hämtad 2008-08-19).

Mizrahi, A. (2003) *Intersection of three spheres.* Tillgänglig på: http://groups.google.com/group/comp.graphics.api.opengl/browse_thread/thread/b3a5f208f27c5b93/0c14a607676cc1c4?hl=sv&lnk=st&q=intersection+of+three+spheres#0c14a607676cc1c4 (besökt 2008-06-26)

Navidi, W., Murphy, W. S. Jr. & Hereman, W. (1998) Statistical methods in surveying by trilateration, *Computational Statistics & Data Analysis*, vol. 27 , s. 209-227.

Reshetyuk, Y. (2007) *Introduction to terrestrial laser scanning*. Kurskompendium för kursen ’Laserskanning från manken och luften’ vid Högskolan i Gävle.

Sandberg, M. (2007) Whole-Field Measuring Methods in Ventilated Rooms, *HVAC&R Research*, vol. 13, s. 951-970.

Sandberg, M., Lundström, H., Nilsson, H.O., Stymne, H. (2008) Experimental methods in Ventilation. *Advances in Buildig Energy Research*, vol. 2, s. 159-210.

Sandberg, M. (2008) Samtal med författaren.

Settles, G. S. (1997) Visualizing Full-Scale Ventilation Airflows, *ASHRAE Journal*, s. 19-27.

The Math Forum (2008a) *Intersection of Three Spheres*. Tillgänglig på: http://mathforum.org/library/drmath/view/63138.html (besökt 2008-06-26).

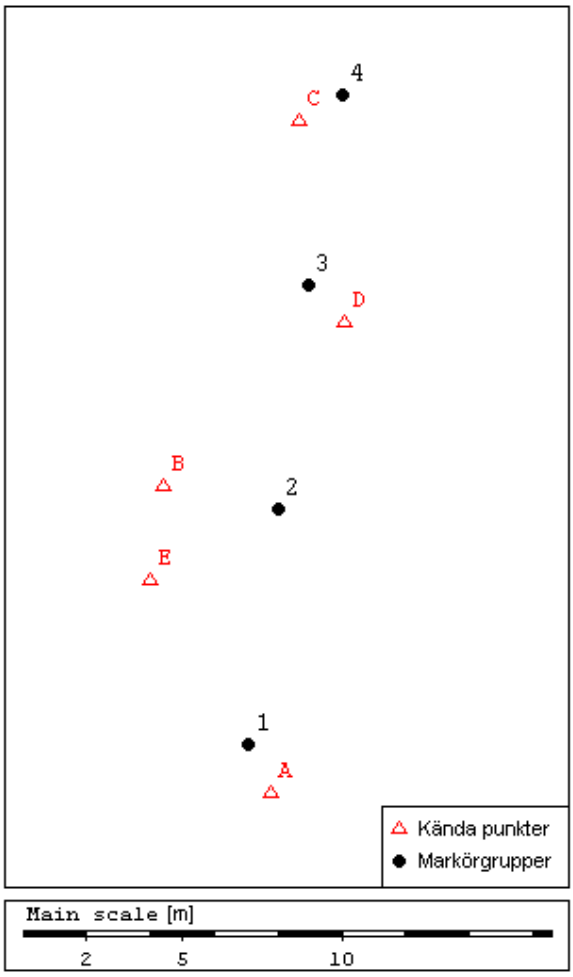
The Math Forum (2008b) E-postkonversation med Doctor Vogler.

Bilaga A – Mätdata inmätning av kända punkter

Tabell A1. *Mätdata från inmätningen av de kända punkterna (A-E). Punkterna 811, 817 och 820 är punkter i byggplatsnätet som användes vid etableringen av totalstationen (fri station).*

mätpunkt	x	y	z
811	1 081.849	1 126.325	5.267
817	1 022.316	1 119.508	7.181
820	1 065.673	1 113.473	6.765
A	1 024.899	1 120.920	-0.006
B	1 034.561	1 117.573	0.003
C	1 046.052	1 121.838	-0.005
D	1 039.723	1 123.254	-0.006
E	1 031.593	1 117.126	0.003

Bilaga B – Punktskiss



Figur B1. *Skiss över hur de kända punkterna (A-E) och markörgrupperna (1-4) var placerade i plan under studien.*

Bilaga C – Mätdata längdmätningar

Tabell C1. *Mätdata från den enkla metodens längdmätningar. Avstånden i meter (m) är mätta från de kända punkterna A-E till respektive markör.*

markör	A	B	C	D	E
12	1.714	8.430	19.633	13.608	5.865
13	1.671	8.458	19.632	13.596	5.908
14	1.620	8.493	19.636	13.586	5.960
15	1.587	8.522	19.634	13.575	6.004
22	8.794	3.477	12.260	6.320	4.389
23	8.803	3.546	12.238	6.278	4.459
24	8.815	3.617	12.214	6.233	4.531
25	8.825	3.698	12.199	6.198	4.610
32	15.925	7.606	5.136	1.682	10.381
33	15.914	7.635	5.145	1.624	10.401
34	15.906	7.667	5.162	1.559	10.424
35	15.897	7.706	5.173	1.497	10.451
42	21.985	13.371	1.379	7.068	
43	21.974	13.385	1.435	7.047	16.297
44	21.966	13.404	1.499	7.032	16.311
45	21.959	13.424	1.567	7.018	16.327

Bilaga D – EDM-test

Bakgrund

EDM står för *Electronic Distance Measurment* och är ett samlingsnamn för elektroniska mätinstrument som använder sig av ljus för att mäta avstånd. Med ett EDM-test kan man beräkna precisionen för dessa instrument (FIG, 2004). Metoden kan även användas för att beräkna prismakonstanter.

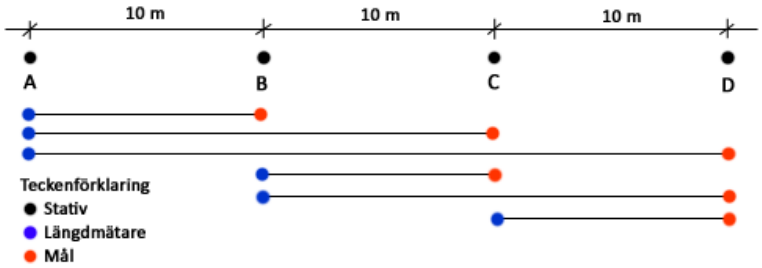
Metod

Testet går till så att fyra punkter markeras på en linje med ett jämnt avstånd mellan punkterna. I EDM-testet som utfördes på längdmätaren som användes i studien valdes avstånden mellan punkterna till 10 m. Avstånden mättes med måttband. Över punkterna placeras stativ på vilka trefötter monterades och horisonterades med hjälp av den inbyggda doslibellen. Längdmätaren och ett mål, i det här fallet en vridbar halv-sfär, monterades på var sin trefot enligt figur D1 och D2.

Figur D1. *Längdmätaren monterad en trefot.*

Figur D2. *Mål för EDM-testets längdmätningarna (vridbar halv-sfär)*

Avstånden mellan punkterna mättes enligt schemat beskrivet i figur D3.



Figur D3. *Mätschema för EDM-testets avståndsmätningar.*

Längdmätaren och målet flyttades mellan stativen via trefotens snabbkoppling. Trefötterna lämnades kvar på stativet och horisonterades med så små justeringar som möjligt innan varje längdmätning.

Beräkningar

Mätdata

Längdmätningarna som utfördes enligt mätschemat i figur D3 gav följande resultat:

$$d_{AB} = 10.036 \text{ m}$$
$$d_{AC} = 20.037 \text{ m}$$
$$d_{AD} = 30.055 \text{ m}$$
$$d_{BC} = 10.003 \text{ m}$$
$$d_{BD} = 20.020 \text{ m}$$
$$d_{CD} = 10.018 \text{ m}$$

där d är avstånden mellan de indexerade punkterna.

Medeltalsberäkning

Det enklaste sättet att beräkna noggrannheten hos instrumentet är genom att göra en medeltalsberäkning med ekvationerna 1-3

$$d_{AB} + d_{BC} + d_{CD} - d_{AD} = -2a \quad (1)$$
$$d_{AB} + d_{BD} - d_{AD} = -a \quad (2)$$
$$d_{AC} + d_{CD} - d_{AD} = -a \quad (3)$$

där d är avståndet mellan de indexerade punkterna. Medelvädet av a, det konstanta felet hos längdmätaren, beräknades till 0.001 m.

Minsta kvadratmetoden

En metod för att beräkna det konstanta felets medelfel för längdmätaren är att använda minsta kvadratmetoden enligt ekvation 4-7.

$$\mathbb{X} = (\mathbb{A}^T\mathbb{A})^{-1} \times (\mathbb{A}^T\mathbb{L}) \quad (4)$$

$$\mathbb{V} = \mathbb{A}\mathbb{X} - \mathbb{L} \quad (5)$$

$$\sigma_0 = \sqrt{((\mathbb{V}^T\mathbb{V})/(n-u))} \quad (6)$$

$$\mathbb{C} = \sigma_0^2 (\mathbb{A}^T\mathbb{A})^{-1} \quad (7)$$

där de ingående matriserna är:

$$\mathbb{A} = \begin{matrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{matrix} \qquad \mathbb{L} = \begin{matrix} 10.036 \\ 20.037 \\ 30.055 \\ 10.003 \\ 20.020 \\ 10.018 \end{matrix}$$

$\mathbb{A}$ beskriver vilka distanser som ingår i motsvarande rad i $\mathbb{L}$. Rad 2 betyder t.ex. att avståndet mellan A och B samt B och C tillsammans blir 20.037 m. n och u är antalet observationer och antal obestämda vilket ger n = 6 och u = 4.

Resultatet av ekvation 4, 5 och 6 blir:

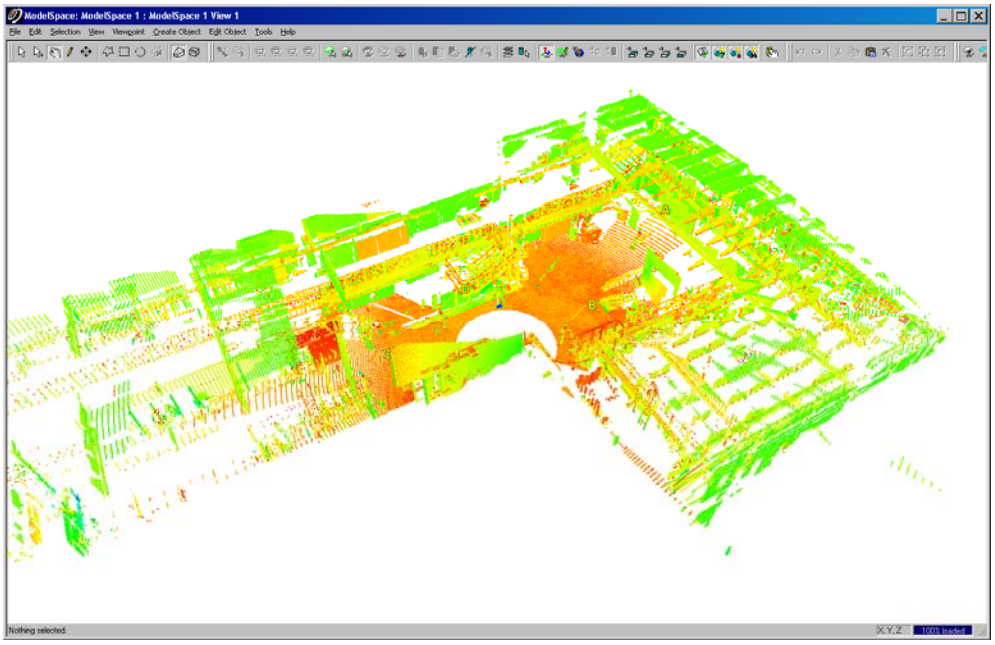
$$\mathbb{X} = \begin{matrix} 10.035 \\ 10.002 \\ 10.017 \\ 0.001 \end{matrix} \qquad \mathbb{V} = \begin{matrix} -0.00025 \\ 0.00050 \\ -0.00025 \\ -0.00025 \\ 0.00000 \\ 0.00025 \end{matrix} \qquad \sigma_0 = 0.0005$$

Resultatet av ekvation 7 blir:

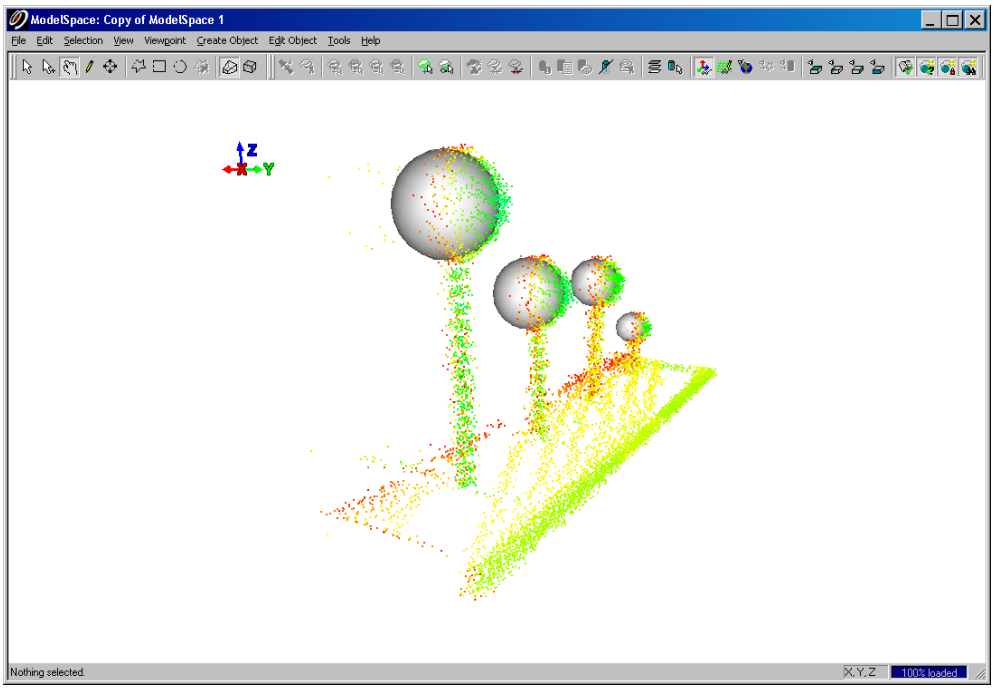
$$\mathbb{C} = \begin{matrix} 1.875\times10^{-07} & -1.850\times10^{-23} & 6.250\times10^{-08} & -1.250\times10^{-07} \\ 0 & 1.875\times10^{-07} & -4.163\times10^{-23} & -1.250\times10^{-07} \\ 6.250\times10^{-08} & -2.776\times10^{-23} & 1.875\times10^{-07} & -1.250\times10^{-07} \\ -1.250\times10^{-07} & -1.250\times10^{-07} & -1.250\times10^{-07} & \mathbf{2.500\times10^{-07}} \end{matrix}$$

där det konstanta felets medelfelet för längdmätaren beräknades till 0.5 mm genom att dra roten ur cell (4,4).

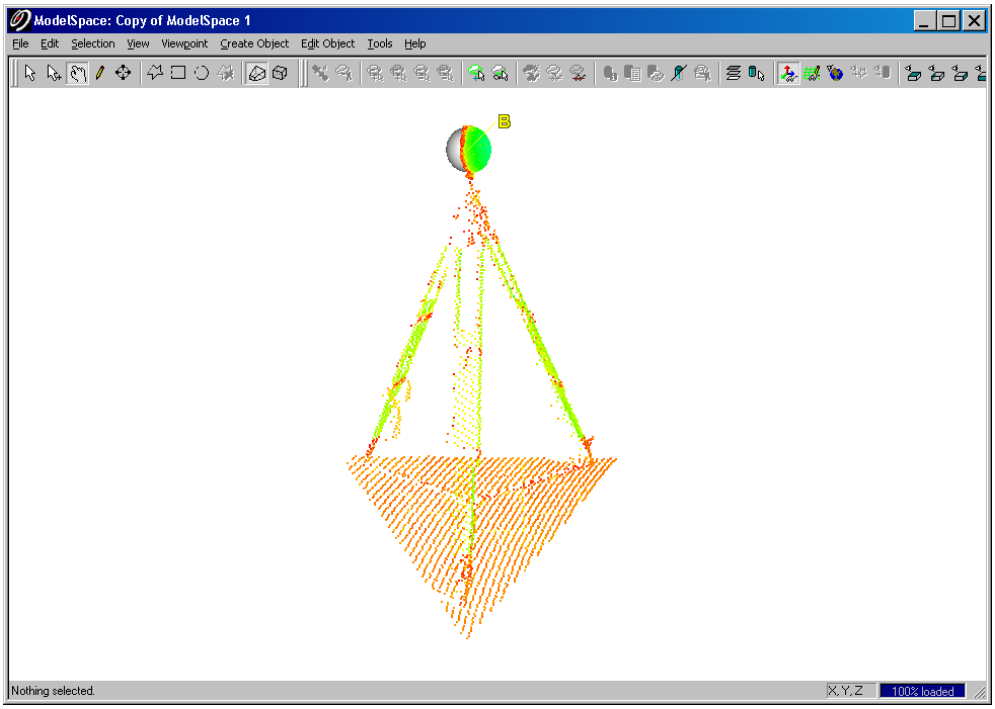
Bilaga E – Punktmoln från laserskanningen



Figur E1. *Punktmolnet från skanningen av hela mäthallen i låg upplösning. Instrumentet var placerat i centrum av den vita cirkeln i figurens mitt. Området saknar därför mätpunkter.*



Figur E2. *Punktmolnet från markörgrupp 1 samt de modellerade markörerna.*



Figur E3. *Punktmolnet från det sfäriska målet placerat över punkt B.*

Bilaga F – Utrustningslista

För att utföra den enkla metoden behövs följande utrustning:

Etableringoch markering av kända punkter

- Måttband (för inmätning i koordinatsystem)
- Tejp
- Märkpenna

Mätutrustning

- Längdmätare (med stöd för bluetooth)
- Adapter (tillverkad och anpassad för ovanstående längdmätaren)
- Standardfäste för att fästa kamera på stativ
- Kamerastativ med panoreringshandtag
- Stativstjärna
- Buntband (för att fästa kamerastativet på stativstjärnan)
- Doslibell (för horisontering av kamerastativ)
- Snörlod (för centrering av kamerastativ över känd punkt)

Handdator

- Handdator med Windows Mobile 5 (med stöd för bluetooth)
- Docka till handdatorn för installation av DistoPos samt export av textfil med koordinater
- Applikationen DistoPos inklusive databasfil

Markörer

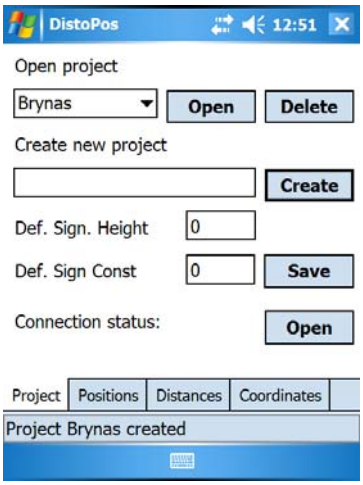
- Markörer, klotformade (ej genomskinliga)
- Klisterlappar (för namnmärkning)
- Linjal/skjutmått för att mäta signalhöjden (avståndet mellan markörens centrum och sensorn)
- Skjutmått (för att eventuellt mäta markörernas diametrar)

Bilaga G – Figure captions

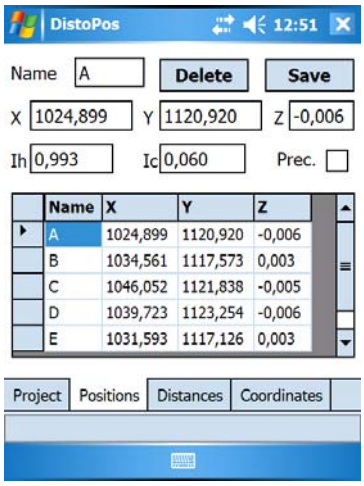
Bilaga H – DistoPos

Användargränssnitt

Användargränssnittet i DistoPos består av fyra formulär som användaren navigerar mellan med hjälp av flikar. Det första formuläret (se figur E1) hanterar projekten och dess parametrar. Här kan användaren välja att hämta ett befintligt projekt eller skapa ett nytt. Parametrarna som kan uppdateras är *Default Signal Height* (standard signalhöjd) och *Default Signal Constant* (standard signalkonstant). Ifyllda värden sparas genom ett klicka på knappen *Save*. Dessa parametrar laddas automatiskt när ett objekt ska mätas in för första gången på formuläret som hanterar längdmätningarna. Knappen *Delete* tar bort ett projekt samt all informationen som är relaterad till det projektet. Projektformuläret hanterar även bluetooth-kommunikationen med längdmätaren. Användaren kan starta och stänga av kommunikationen via en knapp. En text visar statusen för kommunikationen.



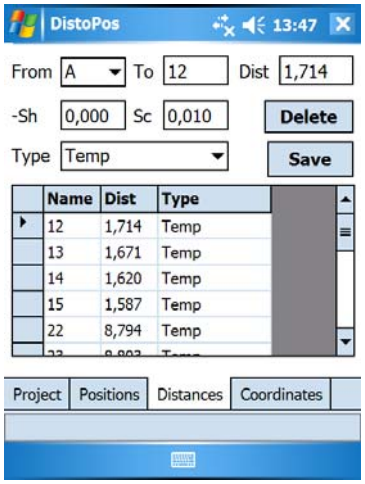
Figur H1. *Projekt-formuläret i DistoPos.*



Figur H2. *Formuläret för att hantera de kända punkterna.*

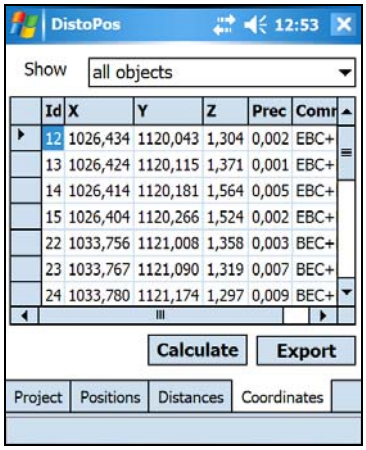
Det andra formuläret hanterar de kända punkterna och dess koordinater (se figur E2). Här får användaren namnge de kända punkterna och mata in koordinater för respektive punkt. Här ska även instrumenthöjden och instrumentkonstanten anges. Eftersom instrumenthöjden ännu inte är känd för samtliga punkter måste användaren gå tillbaka och fylla i den i samband med att mätningarna flyttas till en ny punkt. Instrumentkonstanten som anger avståndet mellan centrum på stativets kulled och bakänden på längdmätaren borde vara samma för samtliga punkter. Kryssrutan för precision ger möjligheten att tala om en viss känd punkt i första hand ska användas till beräkningen av längdmätningarnas precision. Ifyllda värden sparas genom att klicka på knappen *Save*. Knappen *Delete* tar bort den kända punkten samt längdmätningarna från densamma.

Det tredje formuläret hanterar längdmätningarna (se figur E3). *From* anger från vilken känd punkt mätningarna utförs och är en rullgardinslista som innehåller de kända punkterna sparade på formulär två. När användaren väljer en annan punkt i *From* laddas listan med objekt (på nedre halvan av formuläret) om och visar objekten inmätta från den valda punkten. *To* visar namnet på objektet och *Dist* avståndet till detsamma. Signalhöjden (Sh) anger avståndet i z-led mellan markören och sensorn som ska positionsbestämmas. Signalkonstanten (Sc) är markörens radie. Användaren har även möjligheten att ange sensortypen via rullgardinslistan *Type*. Ifyllda värden sparas genom ett klicka på knappen *Save*. Knappen *Delete* ger användaren möjlighet att ta bort längdmätningar eller hela objekt inklusive längdmätningar.



Figur H3. *Formuläret för att hantera längdmätningarna.*

På det sista formuläret utförs positionsberäkningarna (se figur E4). Med rullgardinslistan *Show* kan användaren välja vilka objekt som ska visas i objektlistan. Det finns möjlighet att visa alla objekt, de med koordinater, de utan koordinater samt de utan beräknad precision. Ett klick på knappen *Calculate* beräknar positionerna för objekten i listan och uppdaterar därefter listan. I kolumnen *Comment* visas vilka kända punkter som positionsberäkningarna baserats på. Punkterna väljs beroende på avståndet till objekten. Positionsberäkningen baseras på avstånden till de tre närmsta kända punkterna. Punkterna efter plustecknet i *Comment* ligger till grund för beräkningen av längdmätningarnas precision. Om det är fler än en beräknas ett medelvärde. Om ett objekts position inte kan beräknas på grund av för få avstånd till kända punkter kan användaren komplettera mätningarna från någon av de kända punkterna för att sedan utföra beräkningarna på nytt. Via ett klick på knappen *Export* kan koordinaterna exporteras till textfil för vidare import i andra mjukvaror.



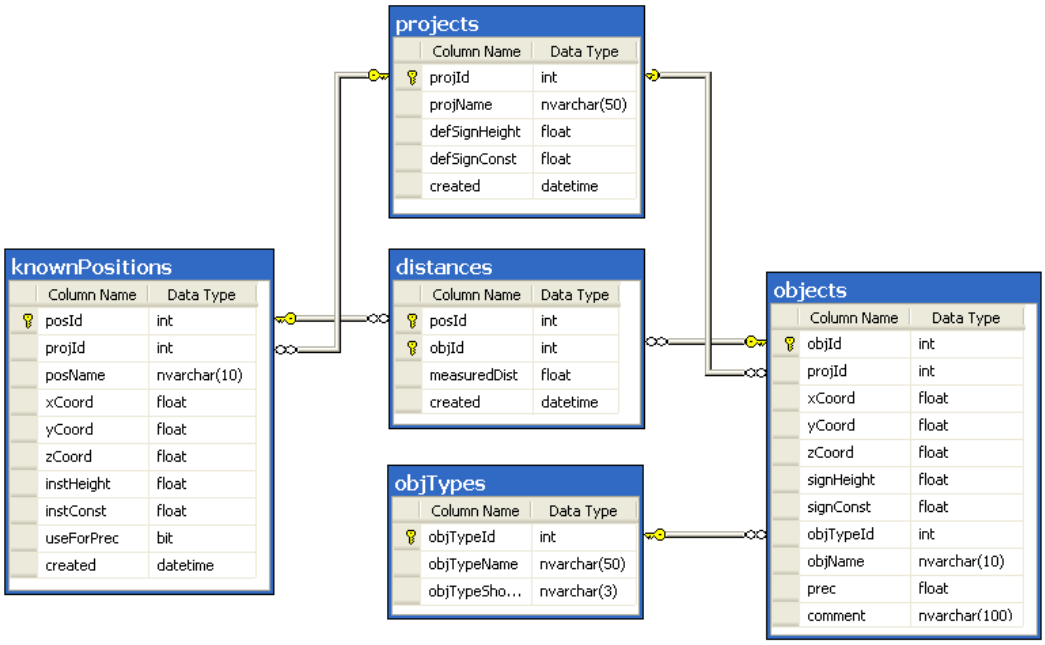
Figur H4. *Formuläret där positionsberäkningarna utförs.*

Utvecklingsverktyg

DistoPos är utvecklat i C#.NET som ingår i Microsoft Visual Studio 2005. Compact Framework 2.0 och Active Sync 3.5 behövdes för utvecklingen av applikationer för handhållna enheter. Operativsystemet i handdatorn är Windows Mobile 5. Programkoden för DistoPos redovisas i bilaga X.

Databas

För att lagra informationen i handdatorn användes databasen Microsoft SQL Server Compact Edition. Figur E5 visar en databasmodell för hur de ingående tabellerna relaterar till varandra. Fälten med nycklar är tabellens primärnyckel vilken måste vara unik i respektive tabell. Primärnycklarna är i de flesta fall räknare som automatiskt hålls unika av databasen.



Figur H5. *Databasmodell som visar hur tabellerna i DistoPos databas relaterar till varandra.*

Bilaga I – Exportfil från DistoPos

```
Export from DistoPos
Project: Brynas
Date: 2008-07-15 10:41

Nr;X;Y;Z;Prec;Type;Comment
12;1026,434;1120,043;1,304;0,002;Temp;EBC+D
13;1026,424;1120,115;1,371;0,001;Temp;EBC+D
14;1026,414;1120,181;1,564;0,005;Temp;EBC+D
15;1026,404;1120,266;1,524;0,002;Temp;EBC+D
22;1033,756;1121,008;1,358;0,003;Temp;BEC+D
23;1033,767;1121,090;1,319;0,007;Temp;BEC+D
24;1033,780;1121,174;1,297;0,009;Temp;BEC+D
25;1033,791;1121,250;1,436;0,005;Temp;BEC+D
32;1040,858;1121,951;1,314;0,006;Temp;CBE+D
33;1040,847;1122,026;1,325;0,000;Temp;CBE+D
34;1040,840;1122,083;1,491;0,057;Temp;CBE+D
35;1040,821;1122,195;1,365;0,009;Temp;CBE+D
42;1046,850;1123,009;1,287;0,002;Temp;CDB+A
43;1046,854;1123,036;1,430;0,027;Temp;CBE+D
44;1046,835;1123,140;1,414;0,014;Temp;CBE+D
45;1046,825;1123,216;1,463;0,016;Temp;CBE+D
```

Bilaga J – Programkod DistoPos

frmDistoPos.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlServerCe;
using System.IO;
using System.Reflection;


namespace DistoPos
{
    public partial class frmDistoPos : Form
    {
        #region common variables
        private DataAccess da; // class to handle the connection
to the database
        public SqlCeDataAdapter daPos;
        public DataSet ds;

        public static int projId; // store the loaded projectid
        public static string projName; // store the name of the
loaded project
        public double defSignHeight; // the default signal height
for the loaded project
        public double defSignConst; // the default signal contant
for the loaded project
        public static string loadedPosName; // store the selected
position in the positions tab
        public int selectedPosId; // store the selected position
on the distances tab
        #endregion

        #region common functions
        public frmDistoPos()
        {
            InitializeComponent();
        }
        private void frmDistoPos_Load(object sender, EventArgs e)
        {
            da = new DataAccess();
            string connString;


            // Init and check the databas. continue only if
everything is OK
            if (da.InitDB(out connString) == false || ValidateDB()
== false)
            {
                MessageBox.Show("Could not connect to database at
'" + connString +
```

```
                                "' or database not valid.\nClosing
application.", "Databas",
                                MessageBoxButtons.OK,
MessageBoxIcon.Exclamation,
                                MessageBoxDefaultButton.Button1);

                Application.Exit();
            }
            else
            {
                // Fill the project combobox
                FillProjCombo(-1);

                updateConnStatusLbl();

                timerNewDist.Enabled = true;
            }
        }
        // handle clicks on tabs
        private void tabControl1_SelectedIndexChanged(object
sender, EventArgs e)
        {
            // Projects tab has been clicked
            if (tabControl1.SelectedIndex == 0)
            {
                // Update status label
                updateConnStatusLbl();
            }
            // Positions tab has been clicked
            else if (tabControl1.SelectedIndex == 1)
            {
                // fill grid
                FillKnownPosGrid();

                // Modify columns
                CreatePosTableStyle();
            }
            // Distances tab has been clicked
            else if (tabControl1.SelectedIndex == 2)
            {
                // fill knownpos combobox
                if (FillKnownPosCombo())
                {
                    // set comboitem end fill grid
                    SetComboItem(cboPos, selPosId);
                    FillDistGrid();
                }
                // fill objecttypes combobox
                FillObjTypeCombo();
            }
            // Calculations tab has been clicked
            else if (tabControl1.SelectedIndex == 3)
            {
                // fill show-combobox and coordinate grid
                FillCoordCombo();
                FillCoordGrid();
                CreateCoordTableStyle();
            }
            statusBar.Text = "";
        }
        // class to handle comboItems in comboBoxes
```

```
        private class comboItem
        {
            public string Name; // store the text to be shown
            public int Value;   // store the id of the item

            public comboItem(string name, int value)
            {
                Name = name; Value = value;
            }

            public override string ToString()
            {
                // Generates the text shown in the combo box
                return Name;
            }
        }
        // gets the itemvalue from the selected item in a certain
combobox
        private int GetComboItemValue(ComboBox box)
        {
            try
            {
                comboItem item =
(comboItem)box.Items[box.SelectedIndex];
                return item.Value;
            }
            catch
            {
                // if something goes wrong, return -1
                return -1;
            }
        }
        // gets the itemname from the selected item in a certain
combobox
        private string GetComboItemName(ComboBox box)
        {
            try
            {
                comboItem item =
(comboItem)box.Items[box.SelectedIndex];
                return item.Name;
            }
            catch
            {
                // if something goes wrong, return ""
                return "";
            }
        }
        // sets the selection in a combobox
        private void SetComboItem(ComboBox box, int selItemValue)
        {
            int selItem = 0;
            box.SelectedIndex = 0;

            try
            {
                // loop throw the items one by one
                foreach (comboItem item in box.Items)
                {
                    if (item.Value == selItemValue)
                    {
```

```
                    box.SelectedIndex = selItem;
                    return;
                }
                selItem++;
            }
        }
        catch { }
    }
    // function to replace ',' with '.' before database insert
or update
    private string FixDbString(string dirtyString)
    {
        return dirtyString.Replace(",", ".").Trim();
    }
    // validates that the connection to the database is
working
    private bool ValidateDB()
    {
        SqlCeConnection conn = new SqlCeConnection();
        SqlCeCommand command = new SqlCeCommand();

        try
        {
            conn = da.GetDbConnection();
            command = conn.CreateCommand();

            // check that all needed tables exists in the
database
            command.CommandText = "SELECT COUNT(projId) FROM
projects";
            command.ExecuteScalar();
            command.CommandText = "SELECT COUNT(posId) FROM
knownPositions";
            command.ExecuteScalar();
            command.CommandText = "SELECT COUNT(objId) FROM
objects";
            command.ExecuteScalar();
            command.CommandText = "SELECT COUNT(posId) FROM
distances";
            command.ExecuteScalar();
            command.CommandText = "SELECT COUNT(objTypeId)
FROM objTypes";
            command.ExecuteScalar();

            conn.Close();

            return true;
        }
        catch
        {
            return false;
        }
    }
    #endregion

    #region Project Tab
    // fill the project combobox with all projects in the
database
    private void FillProjCombo(int selProjId)
    {
        int selProjIndex = -1;
```

```
            SqlCeConnection conn = new SqlCeConnection();

            try
            {
                conn = da.GetDbConnection();

                SqlCeCommand command = conn.CreateCommand();
                command.CommandText = "SELECT projId, projName
FROM projects";
                SqlCeDataReader dr = command.ExecuteReader();

                // Empty combo
                cboProj.Items.Clear();
                int indexCounter = 0;

                while (dr.Read())
                {
                    int projId = dr.GetInt32(0);
                    string projName = dr.GetString(1);

                    // Add item to combo
                    cboProj.Items.Add(new comboItem(projName,
projId));

                    if (selProjId == projId)
                        selProjIndex = indexCounter;

                    indexCounter++;
                }
                dr.Close();

                if (selProjIndex != -1)
                    cboProj.SelectedIndex = selProjIndex;

                txtNewProj.Text = "";

            }
            catch (Exception e)
            {
                Console.Write(e.ToString());
            }
            finally
            {
                conn.Close();
            }
        }
        // handle the open project event
        private void btnOpenProj_Click(object sender, EventArgs e)
        {
            // get selected project from the projects combobox
            projId = GetComboItemValue(cboProj);
            projName = GetComboItemName(cboProj);
            FillDefSignTextBoxes();

            // update statusbar
            statusBar.Text = "Project '" + projName + "' loaded";
        }
        // handle the create project event
        private void btnCreateProj_Click(object sender, EventArgs
e)
        {
```

```
            double defSignHeight, defSignConst;
            int newProjId;
            string newProjName;

            SqlCeConnection conn = new SqlCeConnection();
            SqlCeCommand command = new SqlCeCommand();

            // check the project name
            newProjName = txtNewProj.Text.Trim();
            if (newProjName.Length == 0)
            {
                MessageBox.Show("The project name can not be of
zero length");
                return;
            }

            // check default signal height and contant
            try
            {
                defSignHeight =
Convert.ToDouble(txtDefSignHeight.Text);
                defSignConst =
Convert.ToDouble(txtDefSignConst.Text);
            }
            catch
            {
                MessageBox.Show("Incorrect input");
                return;
            }

            try
            {
                conn = da.GetDbConnection();


                command.CommandText = "projects";
                command.CommandType =
System.Data.CommandType.TableDirect;
                command.Connection = conn;

                SqlCeResultSet rs =
command.ExecuteResultSet(ResultSetOptions.Updatable |

ResultSetOptions.Sensitive);
                // create a new row
                SqlCeUpdatableRecord rec = rs.CreateRecord();

                // fill the row to be inserted
                rec.SetString(1, newProjName);
                rec.SetDecimal(2,
Convert.ToDecimal(defSignHeight));
                rec.SetDecimal(3,
Convert.ToDecimal(defSignConst));
                rec.SetDateTime(4, DateTime.Now);

                // make the insert
                rs.Insert(rec,
DbInsertOptions.PositionOnInsertedRow);

                // @@identity of the inserted row
                newProjId = rs.GetInt32(0);
```

```
            }
            catch (Exception ex)
            {
                Console.Write(ex.ToString());
                newProjId = -1;
            }
            finally
            {
                conn.Close();
            }

            projId = newProjId;
            // update the project combobox and select the new
project
            FillProjCombo(projId);

            statusBar.Text = "Project " + newProjName + "
created";

        }
        // gets the default signal height and contant for the
        // selected project and updated its textboxes
        private void FillDefSignTextBoxes()
        {
            SqlCeConnection conn = new SqlCeConnection();
            SqlCeCommand command = new SqlCeCommand();
            SqlCeDataReader dr;

            try
            {
                conn = da.GetDbConnection();
                command = conn.CreateCommand();
                command.CommandText = "SELECT * FROM projects
WHERE projId = " +
                                      projId.ToString();
                dr = command.ExecuteReader();

                while (dr.Read())
                {
                    defSignHeight =
Convert.ToDouble(dr["defSignHeight"]);
                    defSignConst =
Convert.ToDouble(dr["defSignConst"]);
                }
                dr.Close();
            }
            catch (Exception e)
            {
                defSignHeight = 0;
                defSignConst = 0;
                Console.Write(e.ToString());
            }
            finally
            {
                conn.Close();
            }

            txtDefSignHeight.Text = defSignHeight.ToString();
            txtDefSignConst.Text = defSignConst.ToString();

        }
```

```
        // handles the save default height and constant event
        private void btnSaveDefSign_Click(object sender, EventArgs
e)

        {
            double defSignHeight, defSignConst;

            SqlCeConnection conn = new SqlCeConnection();
            SqlCeCommand command = new SqlCeCommand();

            try
            {
                // check the values in the textboxes
                defSignHeight =
Convert.ToDouble(txtDefSignHeight.Text);
                defSignConst =
Convert.ToDouble(txtDefSignConst.Text);
            }
            catch
            {
                MessageBox.Show("Incorrect input");
                return;
            }

            try
            {
                conn = da.GetDbConnection();
                command = conn.CreateCommand();

                command.CommandText = "UPDATE projects SET
defSignHeight = " +

FixDbString(defSignHeight.ToString()) +
                                        ", defSignConst = " +

FixDbString(defSignConst.ToString()) +
                                        " WHERE projId = " +
projId.ToString();

                // make the update
                command.ExecuteScalar();
            }
            catch (Exception ex)
            {
                Console.Write(ex.ToString());
            }
            finally
            {
                conn.Close();
            }

            statusBar.Text = "Project updated";
        }
        // handel the delete project event
        private void btnDelProj_Click(object sender, EventArgs e)
        {
            // get the selected project
            int projId = GetComboItemValue(cboProj);
            string projName = GetComboItemName(cboProj);

            if (projId != -1)
```

```
            {
                // ask for confirmation
                if (MessageBox.Show("Are you sure you want to
delete project '" +
                    projName + "' ?",
                    "Delete", MessageBoxButtons.OKCancel,
MessageBoxIcon.Question,
                    MessageBoxDefaultButton.Button1) ==
DialogResult.OK)
                {
                    // Delete the project
                    DeleteProject(projId, projName);
                    txtNewProj.Text = "";
                    // update the project combobox
                    FillProjCombo(-1);
                }
            }
            else
            {
                MessageBox.Show("No project selected.", "Delete");
            }
        }
        // deletes a project and all its positions, objects and
distances
        private void DeleteProject(int projId, string projName)
        {
            SqlCeConnection conn = new SqlCeConnection();
            SqlCeCommand command = new SqlCeCommand();

            try
            {
                conn = da.GetDbConnection();
                command = conn.CreateCommand();

                // delete distances
                command.CommandText = "DELETE FROM distances WHERE
posId IN " +
                                      "(SELECT posId FROM
knownPositions WHERE projId = " + projId.ToString() + ")";
                command.ExecuteScalar();

                // delete objects
                command.CommandText = "DELETE FROM objects WHERE
projId = " + projId.ToString();
                command.ExecuteScalar();

                // delete knownPositions
                command.CommandText = "DELETE FROM knownPositions
WHERE projId = " + projId.ToString();
                command.ExecuteScalar();

                // delete project
                command.CommandText = "DELETE FROM projects WHERE
projId = " + projId.ToString();
                command.ExecuteScalar();

                statusBar.Text = "Project deleted";
            }
            catch
            {
```

```
                MessageBox.Show(projName + " couldn't be
deleted.", "Delete");
            }
            finally
            {
                conn.Close();
            }
        }
        #endregion

        #region Position tab
        // fills the grid with known positions
        private void FillKnownPosGrid()

        {
            SqlCeConnection conn = new SqlCeConnection();
            ClearPosTextBoxes();

            try
            {
                conn = da.GetDbConnection();

                // select all positions in the project
                string sSQL = "SELECT * FROM knownPositions WHERE
projId = " +
                              projId.ToString() + " ORDER BY
posId";

                ds = new DataSet("knownPos");
                SqlCeDataAdapter dap = new SqlCeDataAdapter(sSQL,
conn);
                SqlCeCommandBuilder cb = new
SqlCeCommandBuilder(dap);
                // fill the datasat
                dap.Fill(ds, "knownPos");
                // .. and bind it to the grid
                dgPos.DataSource = ds.Tables["knownPos"];
            }
            catch (Exception e)
            {
                Console.Write(e.ToString());
            }
            finally
            {
                conn.Close();
            }
        }
        // handle the cell change event for the position grid
        private void dgPos_CurrentCellChanged(object sender,
EventArgs e)
        {
            FillKnownPosTextBoxes();
        }
        // fills the textboxes with information from one position
        private void FillKnownPosTextBoxes()
        {
            try
            {
                // get selected row
                DataRow row =
ds.Tables["knownPos"].Rows[dgPos.CurrentRowIndex];
```

```
            txtPosName.Text = row["posName"].ToString();
            loadedPosName = txtPosName.Text;

            txtPosX.Text = row["xCoord"].ToString();
            txtPosY.Text = row["yCoord"].ToString();
            txtPosZ.Text = row["zCoord"].ToString();
            txtPosIh.Text = row["instHeight"].ToString();
            txtPosIc.Text = row["instConst"].ToString();

            bool useForPrec;

            if (Convert.IsDBNull(row["useForPrec"]))
                useForPrec = false;
            else if ((bool)row["useForPrec"] == true)
                useForPrec = true;
            else
                useForPrec = false;

            cbPosUseForPrecition.Checked = useForPrec;
            statusBar.Text = "";
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.ToString());
        }

    }
    // clear textboxes
    private void ClearPosTextBoxes()
    {
        txtPosName.Text = "";
        txtPosX.Text = "";
        txtPosY.Text = "";
        txtPosZ.Text = "";
        txtPosIh.Text = "";
        txtPosIc.Text = "";
        cbPosUseForPrecition.Checked = false;
    }
    // handle save click
    private void btnSavePos_Click(object sender, EventArgs e)
    {
        string useForPrec;
        if (cbPosUseForPrecition.Checked == true)
            useForPrec = "1";
        else
            useForPrec = "0";

        SqlCeConnection conn = new SqlCeConnection();
        SqlCeCommand command = new SqlCeCommand();

        int posId;
        double xCoord, yCoord, zCoord, instHeight, instConst;
        string posName, posNameToUpdate = "";


        if (txtPosZ.Text.Trim() == "")
            txtPosZ.Text = "0";
        if (txtPosIh.Text.Trim() == "")
            txtPosIh.Text = "0";
        if (txtPosIc.Text.Trim() == "")
```

```
                txtPosIc.Text = "0";

            // check values
            try
            {
                posName = txtPosName.Text.Trim();
                xCoord = Convert.ToDouble(txtPosX.Text);
                yCoord = Convert.ToDouble(txtPosY.Text);
                zCoord = Convert.ToDouble(txtPosZ.Text);
                instHeight = Convert.ToDouble(txtPosIh.Text);
                instConst = Convert.ToDouble(txtPosIc.Text);
            }
            catch
            {
                MessageBox.Show("Incorrect input");
                return;
            }

            // check name
            if (posName.Length == 0)
            {
                MessageBox.Show("The name of the position can not
be of zero length");
                return;
            }
            if (loadedPosName != posName)
            {
                // ask for confirmation to update
                DialogResult result = MessageBox.Show("The
positions name is changed." +
                                      " Update old position?",
"Update",

MessageBoxButtons.YesNoCancel,
                                      MessageBoxIcon.Question,

MessageBoxDefaultButton.Button1);

                // yes : update old position
                if (result == DialogResult.Yes)
                    posNameToUpdate = loadedPosName;
                // no : insert new position
                else if (result == DialogResult.No)
                    posNameToUpdate = posName;
                // cancel : abort update
                else if (result == DialogResult.Cancel)
                    return;
            }

            try
            {
                conn = da.GetDbConnection();
                command = conn.CreateCommand();
                command.CommandText = "SELECT posId FROM
knownPositions " +
                                      "WHERE projId = " +
projId.ToString() +
                                      " AND posName = '" +
posNameToUpdate + "'";

                try
```

```
                {
                    // get the right position id
                    posId =
Convert.ToInt32(command.ExecuteScalar());
                }
                catch
                {
                    posId = 0;
                }

                if (posId == 0)
                {
                    // insert if its a new one
                    command.CommandText = "INSERT INTO
knownPositions (projId, posName, " +
                                          "xCoord, yCoord, zCoord,
instHeight, instConst, " +
                                          "useForPrec, created)
VALUES (" +
                                          projId.ToString() + ",
'" +
                                          posName + "', " +

FixDbString(xCoord.ToString()) + ", " +

FixDbString(yCoord.ToString()) + ", " +

FixDbString(zCoord.ToString()) + ", " +

FixDbString(instHeight.ToString()) + ", " +

FixDbString(instConst.ToString()) + ", " +
                                          useForPrec + ", " +
                                          "getdate())";
                }
                else
                {
                    // update the old one
                    command.CommandText = "UPDATE knownPositions
SET posName = '" + txtPosName.Text + "', " +
                                          "xCoord = " +
FixDbString(txtPosX.Text) + ", " +
                                          "yCoord = " +
FixDbString(txtPosY.Text) + ", " +
                                          "zCoord = " +
FixDbString(txtPosZ.Text) + ", " +
                                          "instHeight = " +
FixDbString(txtPosIh.Text) + ", " +
                                          "instConst = " +
FixDbString(txtPosIc.Text) + ", " +
                                          "useForPrec = " +
useForPrec +
                                          " WHERE posId = " +
posId.ToString();
                }
                // execute command
                command.ExecuteScalar();
            }
            catch (Exception ex)
            {
                Console.Write(ex.ToString());
```

```
            }
            finally
            {
                conn.Close();
            }

            // update datagrid
            FillKnownPosGrid();
            // update statusbar
            statusBar.Text = "Position saved";

        }
        // take care of the grid layout
        private void CreatePosTableStyle()
        {
            DataGridTableStyle posTable = new
DataGridTableStyle();
            posTable.MappingName = "knownPos";

            DataGridTextBoxColumn idColumn = new
DataGridTextBoxColumn();
            idColumn.MappingName = "posId";
            idColumn.HeaderText = "Id";
            idColumn.Width = 0;

            DataGridTextBoxColumn nameColumn = new
DataGridTextBoxColumn();
            nameColumn.MappingName = "posName";
            nameColumn.HeaderText = "Name";
            nameColumn.Width = 75;

            DataGridTextBoxColumn xColumn = new
DataGridTextBoxColumn();
            xColumn.MappingName = "xCoord";
            xColumn.HeaderText = "X";
            xColumn.Width = 100;

            DataGridTextBoxColumn yColumn = new
DataGridTextBoxColumn();
            yColumn.MappingName = "yCoord";
            yColumn.HeaderText = "Y";
            yColumn.Width = 100;

            DataGridTextBoxColumn zColumn = new
DataGridTextBoxColumn();
            zColumn.MappingName = "zCoord";
            zColumn.HeaderText = "Z";
            zColumn.Width = 75;

            //DataGridTextBoxColumn precColumn = new
DataGridTextBoxColumn();
            //precColumn.MappingName = "useForPrec";
            //precColumn.HeaderText = "Prec";
            //precColumn.Width = 75;

            // Adds the column styles to the grid table style.
            posTable.GridColumnStyles.Add(idColumn);
            posTable.GridColumnStyles.Add(nameColumn);
            posTable.GridColumnStyles.Add(xColumn);
            posTable.GridColumnStyles.Add(yColumn);
            posTable.GridColumnStyles.Add(zColumn);
```

```
            //posTable.GridColumnStyles.Add(precColumn);

            // Add the table style to the collection, but clear
the
            // collection first.
            dgPos.TableStyles.Clear();
            dgPos.TableStyles.Add(posTable);
        }
        // handle the delete position click
        private void btnDelPos_Click(object sender, EventArgs e)
        {
            SqlCeConnection conn = new SqlCeConnection();
            SqlCeCommand command = new SqlCeCommand();

            int posId;
            string posName = txtPosName.Text.Trim();

            conn = da.GetDbConnection();
            command = conn.CreateCommand();

            // get the right positionid
            command.CommandText = "SELECT posId FROM
knownPositions " +
                                  "WHERE projId = " +
projId.ToString() +
                                  " AND posName = '" + posName +
"'";
            try
            {
                posId = Convert.ToInt32(command.ExecuteScalar());
            }
            catch
            {
                posId = 0;
            }

            if (posId != 0)
            {
                // ask for confirmation before deletion
                if (MessageBox.Show("Are you sure you want to
delete position '" +
                    posName + "' ?",
                    "Delete", MessageBoxButtons.OKCancel,
MessageBoxIcon.Question,
                    MessageBoxDefaultButton.Button1) ==
DialogResult.OK)
                {
                    // delete position
                    DeletePosition(posId, posName);
                    // clear textboxes
                    ClearPosTextBoxes();
                    // update grid
                    FillKnownPosGrid();
                }
            }
            else
            {
                MessageBox.Show("Could not find position '" +
posName + "'", "Delete");
            }
        }
```

```
        // delete position
        private void DeletePosition(int posId, string posName)
        {
            SqlCeConnection conn = new SqlCeConnection();
            SqlCeCommand command = new SqlCeCommand();

            try
            {
                conn = da.GetDbConnection();
                command = conn.CreateCommand();

                // delete distances
                command.CommandText = "DELETE FROM distances WHERE
posId IN " +
                                      "(SELECT posId FROM
knownPositions WHERE posId = " + posId.ToString() + ")";
                command.ExecuteScalar();

                // delete knownPositions
                command.CommandText = "DELETE FROM knownPositions
WHERE posId = " + posId.ToString();
                command.ExecuteScalar();

                // update statusbar
                statusBar.Text = "Position deleted";
                // update peoject-combo, no preselection
                FillKnownPosCombo();
            }
            catch
            {
                MessageBox.Show(posName + " couldn't be deleted.",
"Delete");
            }
            finally
            {
                conn.Close();
            }
        }
        #endregion

        #region Distance Tab
        private int selPosId;
        private int selDistId;
        private bool txtObjChanged;

        // fill distances grid
        private void FillDistGrid()
        {
            SqlCeConnection conn = new SqlCeConnection();

            // clear textboxes
            ClearDistTextBoxes();

            try
            {
                conn = da.GetDbConnection();

                // select distances for the first position in the
position combobox
                string sSQL = "SELECT D.posId " +
                              "    , D.objId " +
```

```
                                "    , O.objName " +
                                "    , D.measuredDist " +
                                "    , O.signHeight " +
                                "   , O.signConst " +
                                "    , OT.* " +
                                "FROM distances D " +
                                "    JOIN objects O ON O.objId =
D.objId " +
                                "    JOIN knownPositions P ON
P.posId = D.posId " +
                                "    LEFT OUTER JOIN objTypes OT
ON O.objTypeId = OT.objTypeId " +
                                "WHERE P.projId = " +
projId.ToString() +
                                "    AND D.posId = " +
selPosId.ToString() +
                                " ORDER BY D.objId";


                ds = new DataSet("Dist");
                SqlCeDataAdapter dap = new SqlCeDataAdapter(sSQL,
conn);
                SqlCeCommandBuilder cb = new
SqlCeCommandBuilder(dap);
                // fill the dataset
                dap.Fill(ds, "Dist");
                // bind it to the grid
                dgDist.DataSource = ds.Tables["Dist"];
                // take care of the layout of the grid
                CreateDistTableStyle();
            }
            catch (Exception e)
            {
                Console.Write(e.ToString());
            }
            finally
            {
                conn.Close();
            }
        }
        // fill the position combobox
        private bool FillKnownPosCombo()
        {
            try
            {
                // return false while no positions has been loaded
                bool returnValue = false;

                SqlCeConnection conn = da.GetDbConnection();

                SqlCeCommand command = conn.CreateCommand();
                // get all positions
                command.CommandText = "SELECT posId, posName FROM
knownPositions " +
                                      "WHERE projId = " +
projId.ToString();
                SqlCeDataReader dr = command.ExecuteReader();

                // Empty combo
                cboPos.Items.Clear();
```

```
                // loop positions
                while (dr.Read())
                {
                    int posId = dr.GetInt32(0);
                    string posName = dr.GetString(1);

                    // Add item to combo
                    cboPos.Items.Add(new comboItem(posName,
posId));

                    // return true if at least one position is
loaded into the combo
                    returnValue = true;
                }
                // clear up
                dr.Close();
                conn.Close();

                return returnValue;
            }
            catch (Exception e)
            {
                Console.Write(e.ToString());
                return false;
            }
        }
        // fill the object type combobox
        private void FillObjTypeCombo()
        {
            try
            {
                SqlCeConnection conn = da.GetDbConnection();
                SqlCeCommand command = conn.CreateCommand();
                // select all types
                command.CommandText = "SELECT * FROM objTypes";
                SqlCeDataReader dr = command.ExecuteReader();

                // Empty combo
                cboObjType.Items.Clear();
                cboObjType.Items.Add(new comboItem("none", -1));

                // loop types
                while (dr.Read())
                {
                    int objTypeId = dr.GetInt32(0);
                    string objTypeName = dr.GetString(1);

                    // Add item to combo
                    cboObjType.Items.Add(new
comboItem(objTypeName, objTypeId));

                }
                // clear up
                dr.Close();
                conn.Close();
            }
            catch (Exception e)
            {
                Console.Write(e.ToString());
            }
        }
```

```
        // handle change event in the position combobox
        private void cboPos_SelectedIndexChanged(object sender,
EventArgs e)
        {
            // get the selected item
            int posId = GetComboItemValue(cboPos);
            // check if its changed
            if (selPosId != posId)
            {
                selPosId = posId;
                // update grid
                FillDistGrid();
            }
        }
        // clear textboxes
        private void ClearDistTextBoxes()
        {
            try
            {
                txtObj.Text = "";
                txtObjDist.Text = "";
                txtObjSc.Text = defSignConst.ToString();
                txtObjSh.Text = defSignHeight.ToString();
                cboObjType.SelectedIndex = 0;
            }
            catch { }
        }
        // take care of the gridlayout
        private void CreateDistTableStyle()
        {
            DataGridTableStyle distTable = new
DataGridTableStyle();
            distTable.MappingName = "Dist";

            DataGridTextBoxColumn nameColumn = new
DataGridTextBoxColumn();
            nameColumn.MappingName = "objName";
            nameColumn.HeaderText = "Name";
            nameColumn.Width = 75;

            DataGridTextBoxColumn distColumn = new
DataGridTextBoxColumn();
            distColumn.MappingName = "measuredDist";
            distColumn.HeaderText = "Dist";
            distColumn.Width = 75;

            DataGridTextBoxColumn typeColumn = new
DataGridTextBoxColumn();
            typeColumn.MappingName = "objTypeName";
            typeColumn.HeaderText = "Type";
            typeColumn.Width = 150;
            typeColumn.NullText = "";

            // Adds the column styles to the grid table style.
            distTable.GridColumnStyles.Add(nameColumn);
            distTable.GridColumnStyles.Add(distColumn);
            distTable.GridColumnStyles.Add(typeColumn);

            // Add the table style to the collection, but clear
the
            // collection first.
```

```
            dgDist.TableStyles.Clear();
            dgDist.TableStyles.Add(distTable);
        }
        // handle the cellchange event in the grid
        private void dgDist_CurrentCellChanged(object sender,
EventArgs e)
        {
            // get selected position
            try
            {
                selDistId =
Convert.ToInt32(dgDist.CurrentCell.RowNumber);
            }
            catch
            {
                MessageBox.Show("Select a row");
            }

            // fill textboxes
            FillDistTextBoxes(selDistId);
        }
        // fill textboxes with info from one distance
        private void FillDistTextBoxes(int distId)
        {
            try
            {
                DataRow row = ds.Tables["Dist"].Rows[distId];
                // update textboxes
                txtObj.Text = row["objName"].ToString();
                txtObjDist.Text = row["measuredDist"].ToString();
                txtObjSh.Text = row["signHeight"].ToString();
                txtObjSc.Text = row["signConst"].ToString();

                int objTypeId;
                try
                {
                    objTypeId = Convert.ToInt32(row["objTypeId"]);
                }
                catch
                {
                    objTypeId = -1;
                }

                // set the objecttype combobox selection
                SetComboItem(cboObjType, objTypeId);
                txtObjChanged = false;

                // update statusbar
                statusBar.Text = "";
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.ToString());
            }
        }
        // check if its a new object and if its so - make the
insert
        private int InsertNewObjectIfNew(int posId, string
objName,
                                         double signHeight, double
signConst,
```

```
                                        int objTypeId)
        {
            int objId;
            SqlCeConnection conn = new SqlCeConnection();
            SqlCeCommand command = new SqlCeCommand();

            try
            {
                conn = da.GetDbConnection();
                command = conn.CreateCommand();

                // try to find a object with the same name
                command.CommandText = "SELECT objId FROM objects
WHERE " +
                                      "projId = " +
projId.ToString() +
                                      " AND objName = '" +
objName + "'";

                objId = Convert.ToInt32(command.ExecuteScalar());

                // if none was found
                if (objId == 0)
                {
                    command.CommandText = "objects";
                    command.CommandType =
System.Data.CommandType.TableDirect;
                    command.Connection = conn;

                    SqlCeResultSet rs =
command.ExecuteResultSet(ResultSetOptions.Updatable |

ResultSetOptions.Sensitive);
                    SqlCeUpdatableRecord rec = rs.CreateRecord();

                    rec.SetInt32(1, projId);
                    rec.SetString(8, objName);
                    rec.SetDecimal(5,
Convert.ToDecimal(signHeight));
                    rec.SetDecimal(6,
Convert.ToDecimal(signConst));
                    rec.SetInt32(7, objTypeId);

                    // make the insert
                    rs.Insert(rec,
DbInsertOptions.PositionOnInsertedRow);

                    // @@identity of the new row
                    objId = rs.GetInt32(0);
                }
            }
            catch (Exception e)
            {
                objId = -1;
                Console.Write(e.ToString());
            }
            finally
            {
                conn.Close();
            }
```

```
            return objId;
        }
        // check if the distance allready exists
        private bool DistExists(int posId, int objId)
        {
            int res;

            SqlCeConnection conn = new SqlCeConnection();
            SqlCeCommand command = new SqlCeCommand();

            try
            {
                conn = da.GetDbConnection();
                command = conn.CreateCommand();

                // check if the distance between the position and
the object is measured
                command.CommandText = "SELECT COUNT(*) FROM
distances " +
                                      "WHERE posId = " +
posId.ToString() + " AND objId = " + objId.ToString();

                res = Convert.ToInt32(command.ExecuteScalar());

            }
            catch (Exception e)
            {
                Console.Write(e.ToString());
                res = 1;
            }
            finally
            {
                conn.Close();
            }

            if (res == 0)
                return false;
            else
                return true;

        }
        // handle save distance click
        private void btnSaveDist_Click(object sender, EventArgs e)
        {
            // selPosId
            string objName;
            double dist, signHeight, sigConst;
            int objTypeId;

            bool updateGrid = true;
            objName = txtObj.Text.Trim();
            objTypeId = GetComboItemValue(cboObjType);

            if (objName.Length == 0)
            {
                MessageBox.Show("The field 'To' can not be of zero
length");
                return;
            }
            if (selPosId == 0)
            {
```

```
                MessageBox.Show("Please select a position in the
'From' dropdown");
                return;
            }

            // check values
            try
            {
                dist = Convert.ToDouble(txtObjDist.Text);
                signHeight = Convert.ToDouble(txtObjSh.Text);
                sigConst = Convert.ToDouble(txtObjSc.Text);
            }
            catch
            {
                MessageBox.Show("Incorrect input");
                return;
            }

            // insert a new object if its the first distance
            int objId = InsertNewObjectIfNew(selPosId, objName,
signHeight, sigConst, objTypeId);
            SqlCeConnection conn = new SqlCeConnection();
            SqlCeCommand command = new SqlCeCommand();

            try
            {
                conn = da.GetDbConnection();
                command = conn.CreateCommand();

                if (DistExists(selPosId, objId))
                {
                    // check of the user has changed the value in
txtObj
                    if (txtObjChanged)
                    {
                        // ask for confirmation before update
                        if (MessageBox.Show("The distans between "
+ GetComboItemName(cboPos) +
                                            " and " + objName + "
has already been measured. Update?",
                                            "Update",
MessageBoxButtons.YesNo, MessageBoxIcon.Question,

MessageBoxDefaultButton.Button1) == DialogResult.No)
                            return;
                    }
                    // update distance
                    command.CommandText = "UPDATE distances SET
measuredDist = " + FixDbString(dist.ToString()) +
                                          ", realDist = 0 WHERE
posId = " + selPosId.ToString() +
                                          " AND objId = " +
objId.ToString();
                    command.ExecuteScalar();

                    // update object
                    command.CommandText = "UPDATE objects SET
signHeight = " + FixDbString(signHeight.ToString()) + ", " +
                                          "signConst = " +
FixDbString(sigConst.ToString()) + ", " +
```

```
                                                "objTypeId = " +
objTypeId.ToString() + " " +
                                                "WHERE projId = " +
projId.ToString() + " AND objId = " + objId.ToString();
                        command.ExecuteScalar();
                    }
                    else
                    {
                        // if the distance did not exits, make an
insert
                        command.CommandText = "INSERT INTO
distances(posId, objId, measuredDist, created) " +
                                                "VALUES(" +
selPosId.ToString() + ", " + objId.ToString() + ", " +

FixDbString(dist.ToString()) + ", getdate())";
                        command.ExecuteScalar();
                    }
                }
                catch (Exception ex)
                {
                    Console.Write(ex.ToString());
                }
                finally
                {
                    conn.Close();
                }

                // update datagrid
                if (updateGrid)
                    FillDistGrid();

                // update statusbar
                statusBar.Text = "Distance saved";
            }
            // handle textbox change event
            private void txtObj_TextChanged(object sender, EventArgs
e)
            {
                txtObjChanged = true;
            }
            // delete distance and/or object
            private void btnDelDist_Click(object sender, EventArgs e)
            {
                // get the selected position
                int posId = GetComboItemValue(cboPos);
                string posName = GetComboItemName(cboPos);

                // get the selected object
                int objId;
                string objName = txtObj.Text.Trim();

                SqlCeConnection conn = new SqlCeConnection();
                SqlCeCommand command = new SqlCeCommand();

                try
                {
                    conn = da.GetDbConnection();
                    command = conn.CreateCommand();

                    // get the object id
```

```
                command.CommandText = "SELECT objId FROM objects
WHERE " +
                                      "projId = " +
projId.ToString() +
                                      " AND objName = '" +
objName + "'";

                objId = Convert.ToInt32(command.ExecuteScalar());

                if (objId != 0)
                {
                    // ask for confirmation and what to delete -
the distance or the hole object
                    DialogResult result = MessageBox.Show("Do you
want to delete the object '" + objName +
                                                          "' and
all it's distances as well?'",

"Delete", MessageBoxButtons.YesNoCancel,

MessageBoxIcon.Question,

MessageBoxDefaultButton.Button2);
                    if (result == DialogResult.Yes)
                    {
                        // if yes, delete the object and all its
distances
                        DeleteObject(objId, objName);
                    }
                    else if (result == DialogResult.No)
                    {
                        // if no, delete only the one distance
                        DeleteDist(posId, objId, objName);
                    }
                    // clear textboxes
                    ClearDistTextBoxes();
                    // update grid
                    FillDistGrid();
                }
                else
                {
                    MessageBox.Show("Object not found.",
"Delete");
                }
            }
            catch
            {
            }
            finally
            {
                conn.Close();
            }
        }
        // delete object
        private void DeleteObject(int objId, string objName)
        {
            SqlCeConnection conn = new SqlCeConnection();
            SqlCeCommand command = new SqlCeCommand();

            try
            {
```

```
                conn = da.GetDbConnection();
                command = conn.CreateCommand();

                // delete distances
                command.CommandText = "DELETE FROM distances WHERE
objId IN " +
                                      "(SELECT objId FROM objects
WHERE objId = " + objId.ToString() + ")";
                command.ExecuteScalar();

                // delete objects
                command.CommandText = "DELETE FROM objects WHERE
objId = " + objId.ToString();
                command.ExecuteScalar();

                // update statusbar
                statusBar.Text = "Object deleted";
            }
            catch
            {
                MessageBox.Show("Object " + objName + " couldn't
be deleted.", "Delete");
            }
            finally
            {
                conn.Close();
            }
        }
        // delete the distance
        private void DeleteDist(int posId, int objId, string
objName)
        {
            SqlCeConnection conn = new SqlCeConnection();
            SqlCeCommand command = new SqlCeCommand();

            try
            {
                conn = da.GetDbConnection();
                command = conn.CreateCommand();

                // delete distances
                command.CommandText = "DELETE FROM distances WHERE
posId = " + posId.ToString() +
                                      " AND objId = " +
objId.ToString();
                command.ExecuteScalar();

                // update statusbar
                statusBar.Text = "Distance to '" + objName + "'
deleted";
            }
            catch
            {
                MessageBox.Show("Distance to '" + objName + "'
couldn't be deleted.", "Delete");
            }
            finally
            {
                conn.Close();
            }
        }
```

```
        #endregion

        #region Coordinate Tab
        // fill show combobox
        private void FillCoordCombo()
        {
            // Empty combo
            cboShowCoord.Items.Clear();

            // Add item to combo
            cboShowCoord.Items.Add(new comboItem("all objects",
1));
            cboShowCoord.Items.Add(new comboItem("objects with
coords", 2));
            cboShowCoord.Items.Add(new comboItem("objects without
coords", 3));
            cboShowCoord.Items.Add(new comboItem("objects without
prec ", 4));
        }
        // generate where statments depending on the selected
value of the show combobox
        private string GenerateWhereStatment()
        {
            string statment;

            switch (GetComboItemValue(cboShowCoord))
            {
                case 1: // All objects
                    statment = "";
                    break;
                case 2: // Objects with coords
                    statment = " AND (O.xCoord IS NOT NULL) AND
(O.yCoord IS NOT NULL) " +
                               "AND (O.zCoord IS NOT NULL)";
                    break;
                case 3: // Objects without coords
                    statment = " AND (O.xCoord IS NULL OR O.yCoord
IS NULL OR O.zCoord IS NULL)";
                    break;
                case 4: // Objects without prec
                    statment = " AND O.prec IS NULL";
                    break;
                default:
                    statment = "";
                    break;

            }
            return statment;
        }
        // fill the coordinate grid
        private void FillCoordGrid()
        {
            string whereStatement = GenerateWhereStatment();
            SqlCeConnection conn = new SqlCeConnection();

            try
            {
                conn = da.GetDbConnection();

                // select all the object in the project
```

```
                string sSQL = "SELECT O.objName, O.xCoord,
O.yCoord, O.zCoord" +
                              "   , O.prec, O.comment,
OT.objTypeShortName " +
                              "FROM objects O " +
                                  "   LEFT OUTER JOIN objTypes
OT ON O.objTypeId = OT.objTypeId " +
                              "WHERE O.projId = " +
projId.ToString() +
                              whereStatement +
                              "   ORDER BY O.objName";

                ds = new DataSet("Coord");
                SqlCeDataAdapter dap = new SqlCeDataAdapter(sSQL,
conn);
                SqlCeCommandBuilder cb = new
SqlCeCommandBuilder(dap);
                // fill the dataset
                dap.Fill(ds, "Coord");
                // .. and bind it ti the grid
                dgCoord.DataSource = ds.Tables["Coord"];
            }
            catch (Exception e)
            {
                Console.Write(e.ToString());
            }
            finally
            {
                conn.Close();
            }
        }
        // take care of the gridlayout
        private void CreateCoordTableStyle()
        {
            DataGridTableStyle coordTable = new
DataGridTableStyle();
            coordTable.MappingName = "Coord";

            DataGridTextBoxColumn nameColumn = new
DataGridTextBoxColumn();
            nameColumn.MappingName = "objName";
            nameColumn.HeaderText = "Id";
            nameColumn.Width = 40;

            DataGridTextBoxColumn xColumn = new
DataGridTextBoxColumn();
            xColumn.MappingName = "xCoord";
            xColumn.HeaderText = "X";
            xColumn.Width = 95;
            xColumn.NullText = "";

            DataGridTextBoxColumn yColumn = new
DataGridTextBoxColumn();
            yColumn.MappingName = "yCoord";
            yColumn.HeaderText = "Y";
            yColumn.Width = 95;
            yColumn.NullText = "";

            DataGridTextBoxColumn zColumn = new
DataGridTextBoxColumn();
            zColumn.MappingName = "zCoord";
```

```
            zColumn.HeaderText = "Z";
            zColumn.Width = 60;
            zColumn.NullText = "";

            DataGridTextBoxColumn precColumn = new
DataGridTextBoxColumn();
            precColumn.MappingName = "prec";
            precColumn.HeaderText = "Prec";
            precColumn.Width = 75;
            precColumn.NullText = "";

            DataGridTextBoxColumn typeColumn = new
DataGridTextBoxColumn();
            typeColumn.MappingName = "objTypeShortName";
            typeColumn.HeaderText = "Tp";
            typeColumn.Width = 50;
            typeColumn.NullText = "";

            DataGridTextBoxColumn comColumn = new
DataGridTextBoxColumn();
            comColumn.MappingName = "comment";
            comColumn.HeaderText = "Comment";
            comColumn.Width = 100;
            comColumn.NullText = "";

            // Adds the column styles to the grid table style.
            coordTable.GridColumnStyles.Add(nameColumn);
            coordTable.GridColumnStyles.Add(xColumn);
            coordTable.GridColumnStyles.Add(yColumn);
            coordTable.GridColumnStyles.Add(zColumn);
            coordTable.GridColumnStyles.Add(precColumn);
            coordTable.GridColumnStyles.Add(comColumn);

            // Add the table style to the collection, but clear
the
            // collection first.
            dgCoord.TableStyles.Clear();
            dgCoord.TableStyles.Add(coordTable);
        }
        // handle changes in the show combobox
        private void cboShowCoord_SelectedIndexChanged(object
sender, EventArgs e)
        {
            // fill the grid depending on the choice
            FillCoordGrid();
        }
        // handle the calculation event
        private void btnCalc_Click(object sender, EventArgs e)
        {
            // Make the calculations
            CalculateCoordinates();
        }
        // calculate the objectcoordiantes
        private void CalculateCoordinates()
        {
            // generate the coorect wherestatment
            string whereStatement = GenerateWhereStatment();
            SqlCeConnection conn = new SqlCeConnection();

            int objId = 0, preObjId = 0;
            int posCounter = 0, objCounter = 0, errorCounter = 0;
```

```
            double posX = 0, posY = 0, posZ = 0, dist = 0,
instHeight = 0, instConst = 0;
            double signHeight = 0, preSignHeight = 0, signConst =
0;
            double dist1 = 0, dist2 = 0, dist3 = 0, objPrec = -1;
            double distPrec = 0, precSum = 0, precCounter = 0;
            string comment = "", posName = "", objName = "";
            bool skippPos = false, skippObj = false;

            // declare some points
            point p1 = new point();
            point p2 = new point();
            point p3 = new point();
            point pObj = new point();
            point pPrec = new point();

            try
            {
                conn = da.GetDbConnection();
                SqlCeCommand command = conn.CreateCommand();

                // select the objects whos positions is to be
calculated
                command.CommandText = "SELECT O.objId, O.objName,
P.posId, P.posName" +
                                      "     , P.xCoord AS
posX, P.yCoord AS posY, P.zCoord AS posZ" +
                                      "     ,
D.measuredDist" +
                                      "     , P.instHeight"
+
                                      "     , P.instConst"
+
                                    "     , O.signHeight" +
                                      "     , O.signConst"
+
                                      "     , P.useForPrec"
+
                                    " FROM distances D" +
                                    "     JOIN knownPositions P
ON P.posId = D.posid" +
                                    "     JOIN objects O ON
O.objId = D.objId" +
                                    " WHERE P.projId = " +
projId.ToString() +
                                    " AND P.posName <> 'F'" +
                                    //" AND P.posName <> 'A'" +
                                    whereStatement +
                                    " ORDER BY O.objId,
P.useForPrec, D.measuredDist";
                                    //" ORDER BY O.objId,
P.posId, P.useForPrec DESC";

                SqlCeDataReader dr = command.ExecuteReader();

                // loop the objects
                while(dr.Read())
                {
                    // check the data
                    try
                    {
```

```
                        objId = Convert.ToInt32(dr["objId"]);
                        objName = Convert.ToString(dr["objName"]);
                        posName = Convert.ToString(dr["posName"]);
                        posX = Convert.ToDouble(dr["posX"]);
                        posY = Convert.ToDouble(dr["posY"]);
                        posZ = Convert.ToDouble(dr["posZ"]);
                        dist =
Convert.ToDouble(dr["measuredDist"]);
                        instHeight =
Convert.ToDouble(dr["instHeight"]);
                        instConst =
Convert.ToDouble(dr["instConst"]);
                        signHeight =
Convert.ToDouble(dr["signHeight"]);
                        signConst =
Convert.ToDouble(dr["signConst"]);
                        skippPos = false;
                    }
                    catch
                    {
                        comment = comment + "Data Error: " +
posName + ", ";
                        skippPos = true;
                    }

                    // check if its a new object or just another
position
                    if (objId != preObjId && preObjId != 0)
                    {
                        // if less then 3 pos or skipp obj: just
write comment
                        if (posCounter <= 2 || skippObj == true)
                        {
                            UpdateObjectComment(preObjId,
comment);
                        }
                        else
                        {
                            objPrec = precSum / precCounter;
                            // Update object.
                            UpdateObject(preObjId, pObj,
preSignHeight, objPrec, comment);
                            objCounter++;
                        }

                        // reset variables for new object
                        posCounter = 1;
                        objPrec = -1;
                        comment = "";
                        skippObj = false;
                        precSum = 0;
                        precCounter = 0;
                    }
                    else
                    {
                        posCounter++;
                    }

                    if (skippObj) {} // do nothing more with this
object, wait for update
                    if (skippPos) {posCounter--;}
```

```
                    else
                    {
                        switch (posCounter)
                        {
                            case 1: // first position
                                p1 = new point(posX, posY, posZ +
instHeight);
                                // add instrument constant
(adapter) and signal constant
                                // (target radius) to measured
distance
                                dist1 = dist + instConst +
signConst;
                                comment = comment + posName;
                                break;
                            case 2: // second position
                                p2 = new point(posX, posY, posZ +
instHeight);
                                // add instrument constant
(adapter) and signal constant
                                // (target radius) to measured
distance
                                dist2 = dist + instConst +
signConst;
                                comment = comment + posName;

                                break;
                            case 3: // third position, time for
some calculations
                                p3 = new point(posX, posY, posZ +
instHeight);
                                // add instrument constant
(adapter) and signal constant
                                // (target radius) to measured
distance
                                dist3 = dist + instConst +
signConst;

                                comment = comment + posName;

                                //make the calculation
                                bool calcOk =
pObj.CalculatePointOfInterection(p1, dist1, p2, dist2, p3, dist3);

                                if (calcOk)
                                {
                                    // TO BE IMPLEMENTED
                                    //point pDir = new point();
                                    // check if there is a
direction target for this sensor
                                    //if (pDir.GetDirPos(objId))
                                    //{
                                    //    double dir, elev;
                                    //    dir = CalcPosDir(p3,
pDir);
                                    //    elev = CalcPosElev(p3,
pDir);
                                    //    // move p3 position in
the oposit direction of pDir and the length of signHeight
                                    //    p3.ModPosByPosDir(dir,
elev, signHeight);
```

```
                                    //}
                                }
                                else
                                {
                                    //comment = "Calculation
error";
                                    comment = "E:" + comment;
                                    errorCounter++;
                                    skippObj = true;
                                }
                                break;
                            default: // calculate the precision
with the rest of the positions
                                pPrec = new point(posX, posY, posZ
+ instHeight);
                                // add instrument constant
(adapter) and signal constant
                                // (target radius) to measured
distance
                                distPrec = dist + instConst +
signConst;
                                precSum = precSum +
pObj.CalculatePrecision(pPrec, distPrec);

                                if (precCounter == 0)
                                    comment = comment + "+" +
posName;
                                else
                                    comment = comment + posName;

                                precCounter++;
                                break;
                        }
                    }
                    // remeber the last objId
                    preObjId = objId;
                    preSignHeight = signHeight;
                }

                // Calculate the last object too
                if (posCounter <= 2 || skippObj == true)
                {
                    UpdateObjectComment(preObjId, comment);
                }
                else
                {
                    objPrec = precSum / precCounter;
                    // Update object.
                    UpdateObject(preObjId, pObj, preSignHeight,
objPrec, comment);
                    objCounter++;
                }
                // update statusbar
                statusBar.Text = objCounter.ToString() + "
position(s) calculated. " + errorCounter + " error(s) occured.";
                // update grid
                FillCoordGrid();
            }
            catch (Exception ex)
            {
                Console.Write(ex.ToString());
```

```
            }
            finally
            {
                conn.Close();
            }
        }
        // updates one object with coordinates
        private void UpdateObject(int objId, point objPos, double
objSignHeight, double objPrec, string comment)
        {
            string objPrecString;

            // round of the precision
            if (objPrec == -1)
                objPrecString = "NULL";
            else
                objPrecString = Math.Round(objPrec, 3).ToString();

            // add the signalheight
            objPos.z = objPos.z - objSignHeight;

            SqlCeConnection conn = new SqlCeConnection();
            SqlCeCommand command = new SqlCeCommand();

            try
            {
                conn = da.GetDbConnection();
                command = conn.CreateCommand();

                // make the update
                command.CommandText = "UPDATE objects SET xCoord =
" + FixDbString(Math.Round(objPos.x, 3).ToString()) +
                                                ", yCoord = " +
FixDbString(Math.Round(objPos.y, 3).ToString()) +
                                                ", zCoord = " +
FixDbString(Math.Round(objPos.z, 3).ToString()) +
                                                ", prec = " +
FixDbString(objPrecString) +
                                                ", comment = '" +
comment +
                                                "' WHERE objId = " +
objId.ToString();
                command.ExecuteScalar();
            }
            catch (Exception ex)
            {
                Console.Write(ex.ToString());
            }
            finally
            {
                conn.Close();
            }
        }
        // update an objects comment
        private void UpdateObjectComment(int objId, string
comment)
        {
            SqlCeConnection conn = new SqlCeConnection();
            SqlCeCommand command = new SqlCeCommand();

            try
```

```
            {
                conn = da.GetDbConnection();
                command = conn.CreateCommand();
                // make the update
                command.CommandText = "UPDATE objects SET xCoord =
NULL, yCoord = NULL, zCoord = NULL, prec = NULL" +
                                                ", comment = '" +
comment +
                                                "' WHERE objId = " +
objId.ToString();

                command.ExecuteScalar();
            }
            catch (Exception ex)
            {
                Console.Write(ex.ToString());
            }
            finally
            {
                conn.Close();
            }
        }
        // handle the export-button click
        private void btnExport_Click(object sender, EventArgs e)
        {
            ExportToFile();
        }
        // generates a filename to the file to be exported
        private string GenerateFileName()
        {
            DateTime dateTime = new DateTime();
            dateTime = DateTime.Now;

            // remove illegal characters
            string tmpProjName = projName;
            tmpProjName = tmpProjName.Replace("?", "");
            tmpProjName = tmpProjName.Replace(":", "");
            tmpProjName = tmpProjName.Replace("\\", "");
            tmpProjName = tmpProjName.Replace("/", "");
            tmpProjName = tmpProjName.Replace("*", "");
            tmpProjName = tmpProjName.Replace("\"", "");
            tmpProjName = tmpProjName.Replace("<", "");
            tmpProjName = tmpProjName.Replace(">", "");
            tmpProjName = tmpProjName.Replace("|", "");
            tmpProjName = tmpProjName.Replace(" ", "");

            // add date and time
            return tmpProjName + dateTime.ToString("_yyMMdd-
HHmmss");
        }
        // export to textfile
        private void ExportToFile()
        {
            int objCounter = 0;

            // get the application path
            string appName =
Assembly.GetExecutingAssembly().GetName().CodeBase;
            string appPath = Path.GetDirectoryName(appName);
            string path = appPath + "\\exports\\" +
GenerateFileName() + ".txt";
```

```
            DateTime dateTime = new DateTime();
            dateTime = DateTime.Now;

            // generate a wherestatment
            string whereStatement = GenerateWhereStatment();

            // Create directory if it doesn't exists.
            if (Directory.Exists(path) == false)
                Directory.CreateDirectory(appPath + "\\exports");

            // Delete the file if it exists.
            if (File.Exists(path))
                File.Delete(path);

            //Create the file.
            using (FileStream fs = File.Create(path))
            {
                // add a fileheader
                AddText(fs, "Export from DistoPos\r\n");
                AddText(fs, "Project: " + projName + "\r\n");
                AddText(fs, "Date: " + dateTime.ToString("yyyy-MM-
dd HH:mm") + "\r\n\r\n");
                // add column headers
                AddText(fs, "Nr;X;Y;Z;Prec;Type;Comment\r\n");

                try
                {
                    SqlCeConnection conn = new SqlCeConnection();
                    conn = da.GetDbConnection();

                    SqlCeCommand command = conn.CreateCommand();
                    // select all objects
                    string sSQL = "SELECT O.objName, O.xCoord,
O.yCoord, O.zCoord, " +
                                                "O.prec,
O.comment, OT.objTypeName " +
                                                 "FROM objects O
" +
                                                 "   LEFT OUTER
JOIN objTypes OT ON O.objTypeId = OT.objTypeId " +
                                                 "WHERE O.projId
= " + projId.ToString() +
                                                 whereStatement +
                                                 " ORDER BY
O.objName";

                    command.CommandText = sSQL;

                    SqlCeDataReader dr = command.ExecuteReader();

                    // loop the objects
                    while (dr.Read())
                    {
                        string objectToString = dr["objName"] +
";" + dr["xCoord"]
                                                + ";" +
dr["yCoord"] + ";" + dr["zCoord"]
                                                + ";" + dr["prec"]
+ ";" + dr["objTypeName"]
```

```
                                                    + ";" +
dr["comment"] + "\r\n";
                        // write the object to file
                        AddText(fs, objectToString);
                        objCounter++;
                    }
                    dr.Close();

                    // tell the user that the export is done
                    statusBar.Text = objCounter.ToString() + "
objects exported to file";
                    MessageBox.Show("Coordinates exported to file:
'" + path + "'");
                }
                catch { }
                finally { }
            }
        }
        // add text to a filestream
        private static void AddText(FileStream fs, string value)
        {
            byte[] info = new UTF8Encoding(true).GetBytes(value);
            fs.Write(info, 0, info.Length);
        }
        #endregion

        #region Disto communication
        // enum to handle connection status
        private enum connectionStatus { Connecting = 1, Connected,
Disconneting, Disconnected }
        // communication variables
        connectionStatus connStatus =
connectionStatus.Disconnected;
        private double receivedDist, lastUpdatedDist = 0;
        private bool updateDist = false;
        // connects the PDA to the disto
        private void startDistoConn()
        {
            connStatus = connectionStatus.Connecting;
            receivedDist = 0;
            lastUpdatedDist = 0;

            try
            {
                // check if the port is allready open
                if (serialPort.IsOpen)
                    serialPort.Dispose();

                serialPort.Open();
                statusBar.Text = "Connection open";
                connStatus = connectionStatus.Connected;
            }
            catch (Exception e)
            {
                // update status
                Console.Write(e.ToString());
                statusBar.Text = "Failed to open connection";
                connStatus = connectionStatus.Disconnected;
            }
            finally
            {
```

```
                //serialPort.Close();
            }
        }
        // disconnects the PDA to the disto
        private void endDistoConn()
        {
            connStatus = connectionStatus.Disconneting;

            try
            {
                // check if the port is open
                if (serialPort.IsOpen)
                {
                    try
                    {
                        // clear bufferts
                        serialPort.DiscardInBuffer();
                        serialPort.DiscardOutBuffer();
                    }
                    catch { }
                    try
                    {
                        serialPort.Dispose();
                        serialPort.Close();
                    }
                    catch { }

                    // update status
                    statusBar.Text = "Connection closed";
                    connStatus = connectionStatus.Disconnected;
                }
            }
            catch (Exception e)
            {
                Console.Write(e.ToString());

                if (serialPort.IsOpen)
                {
                    connStatus = connectionStatus.Connected;
                    statusBar.Text = "Failed to close connection.
Open.";
                }
                else
                {
                    connStatus = connectionStatus.Disconnected;
                    statusBar.Text = "Failed to close connection.
Closed.";
                }
            }
            finally
            {
            }
        }
        // handle Received dataevent
        private void serialPort_DataReceived(object sender,

System.IO.Ports.SerialDataReceivedEventArgs e)

        {
            string sInData, sDist, sMoreData;
            double dDist;
```

```
            try
            {
                // read data from the COM-port
                sInData = serialPort.ReadLine();
                while (serialPort.BytesToRead > 0)
                    sMoreData = serialPort.ReadLine();

                // Parse the string
                // Dataformat: "31..00+00015917 \r" (= 15.917m)
                char cZero = '0';
                sDist = sInData.Substring(8, 8).TrimStart(cZero);
                dDist = Convert.ToDouble(sDist) / 1000;

                receivedDist = dDist;
                // tell the timer to update the distance textbox
                updateDist = true;
            }
            catch
            {
            }
        }
        // handle open/close click
        private void btnOpenCloseConn_Click(object sender,
EventArgs e)
        {
            if (connStatus == connectionStatus.Connected)
            {
                // end connection
                endDistoConn();
            }
            else if (connStatus == connectionStatus.Disconnected)
            {
                // start connection
                startDistoConn();
            }
            // updat status
            updateConnStatusLbl();
        }
        // update connection status label
        private void updateConnStatusLbl()
        {
            // connected
            if (connStatus == connectionStatus.Connected)
            {
                btnOpenCloseConn.Text = "Close";
                lblConnStatus.Text = "Open";
                lblConnStatus.ForeColor = Color.Green;
            }
            // disconnected
            else if (connStatus == connectionStatus.Disconnected)
            {

                btnOpenCloseConn.Text = "Open";
                lblConnStatus.Text = "Closed";
                lblConnStatus.ForeColor = Color.Red;
            }
            else
            {
                btnOpenCloseConn.Text = "Open";
                lblConnStatus.Text = "Unknown";
```

```
                    lblConnStatus.ForeColor = Color.Blue;
                }
            }
            // timer-event to update distandetextbox on distance tab
            private void timerNewDist_Tick(object sender, EventArgs e)
            {
                // if new data is received
                if (updateDist)
                {
                    txtObjDist.Text = receivedDist.ToString();
                    lastUpdatedDist = receivedDist;
                    statusBar.Text = "New data received";
                    updateDist = false;
                }
            }
            #endregion
        }
    }
```

Geometry.cs

```
using System;
using System.Collections.Generic;
using System.Text;


namespace DistoPos
{
    // class to handle one position
    class point
    {
        public double x, y, z;
        // constructor in inparameters
        public point(double x, double y, double z)
        {
            this.x = x;
            this.y = y;
            this.z = z;
        }
        // constructor without inparameters
        public point()
        {
            this.x = 0;
            this.y = 0;
            this.z = 0;
        }
        // calculates the distance between two points by
phytagoras
        private double CalculateDistance(point p1, point p2)
        {
            return Math.Sqrt(Math.Pow(p1.x - p2.x, 2) +
Math.Pow(p1.y - p2.y, 2) + Math.Pow(p1.z - p2.z, 2));
        }
        // calculates the precision
        public double CalculatePrecision(point p, double
prec_dist)
        {
            return Math.Abs(prec_dist - CalculateDistance(this,
p));
        }
        // calculates the point of intercestion for the points p1,
p2 and p3
        public bool CalculatePointOfInterection(point p1, double
r1, point p2, double r2, point p3, double r3)
        {
            // declare vaiables
            double x1, y1, z1, x2, y2, z2, x3, y3, z3;
            double x1_2, y1_2, z1_2, r1_2, x2_2, y2_2, z2_2, r2_2,
x3_2, y3_2, z3_2, r3_2;
            double x, y, z;
            double kx, ky, b_x, b_y;
            double kx_2, ky_2, b_x_2, b_y_2;

            // put the IN-parameters in different variables
            x1 = p1.x;
            y1 = p1.y;
            z1 = p1.z;
            x2 = p2.x;
```

```
            y2 = p2.y;
            z2 = p2.z;
            x3 = p3.x;
            y3 = p3.y;
            z3 = p3.z;

            x1_2 = Math.Pow(p1.x, 2);
            y1_2 = Math.Pow(p1.y, 2);
            z1_2 = Math.Pow(p1.z, 2);
            r1_2 = Math.Pow(r1, 2);
            x2_2 = Math.Pow(p2.x, 2);
            y2_2 = Math.Pow(p2.y, 2);
            z2_2 = Math.Pow(p2.z, 2);
            r2_2 = Math.Pow(r2, 2);
            x3_2 = Math.Pow(p3.x, 2);
            y3_2 = Math.Pow(p3.y, 2);
            z3_2 = Math.Pow(p3.z, 2);
            r3_2 = Math.Pow(r3, 2);

            // collect cofficients for y
            ky = (-0.5) * (-2 * x3 * z2 + 2 * x3 * z1 - 2 * x2 *
z1 - 2 * x1 * z3 + 2 * x2 * z3 + 2 * x1 * z2) / (x1 * y2 - x2 * y1
+ x3 * y1 - x3 * y2 + y3 * x2 - x1 * y3);
            b_y = (-0.5) * (x2 * x1_2 - x3 * r2_2 - x3_2 * x2 -
y3_2 * x2 - z3_2 * x2 - x3 * y1_2 + x2 * y1_2 + x2 * z1_2 - x2 *
r1_2 - x3 * z1_2 + x3 * r1_2 + x3 * x2_2 + x3 * y2_2 + x3 * z2_2 -
x1 * r3_2 + x1 * r2_2 + x1 * x3_2 - x1 * z2_2 + r3_2 * x2 - x1 *
x2_2 - x1 * y2_2 + x1 * y3_2 + x1 * z3_2 - x1_2 * x3) / (x1 * y2 -
x2 * y1 + x3 * y1 - x3 * y2 + y3 * x2 - x1 * y3);
            // collect cofficients for x
            kx = (0.5) * (2 * z3 * y2 + 2 * y1 * z2 - 2 * z1 * y2
+ 2 * z1 * y3 - 2 * y3 * z2 - 2 * y1 * z3) / (x1 * y2 - x2 * y1 +
x3 * y1 - x3 * y2 + y3 * x2 - x1 * y3);
            b_x = (0.5) * (x1_2 * y2 - x1_2 * y3 - y1 * r3_2 + y1
* r2_2 + y1 * x3_2 + y1_2 * y2 - y1_2 * y3 + z1_2 * y2 + y2 * r3_2
- y2 * x3_2 + y2_2 * y3 + z2_2 * y3 - r2_2 * y3 + y1 * y3_2 + y1 *
z3_2 - y1 * y2_2 - y1 * z2_2 - z1_2 * y3 - r1_2 * y2 + r1_2 * y3 +
x2_2 * y3 - y2 * y3_2 - y2 * z3_2 - y1 * x2_2) / (x1 * y2 - x2 *
y1 + x3 * y1 - x3 * y2 + y3 * x2 - x1 * y3);

            kx_2 = Math.Pow(kx, 2);
            ky_2 = Math.Pow(ky, 2);
            b_x_2 = Math.Pow(b_x, 2);
            b_y_2 = Math.Pow(b_y, 2);

            //calcualte z
            z = (z1 + y1 * ky - kx * b_x + x1 * kx - ky * b_y +
Math.Sqrt(-ky_2 * x1_2 + ky_2 * r1_2 + 2 * y1 * b_y - kx_2 * y1_2
- kx_2 * b_y_2 + kx_2 * r1_2 - ky_2 * z1_2 - ky_2 * b_x_2 + 2 * kx
* b_x * ky * b_y - b_y_2 + 2 * ky_2 * x1 * b_x + 2 * kx_2 * y1 *
b_y - 2 * z1 * ky * b_y + 2 * z1 * x1 * kx - 2 * z1 * kx * b_x + 2
* z1 * y1 * ky - 2 * y1 * ky * kx * b_x + 2 * y1 * ky * x1 * kx +
2 * x1 * b_x + r1_2 - x1_2 - y1_2 - b_x_2 - 2 * x1 * kx * ky * b_y
- kx_2 * z1_2)) / (1 + kx_2 + ky_2);
            //calcualte y
            y = (-0.5) * (x2 * x1_2 - 2 * x1 * z * z3 + 2 * x2 * z
* z3 + 2 * x1 * z * z2 + 2 * x3 * z * z1 - x3 * y1_2 + x2 * y1_2 +
x2 * z1_2 - x2 * r1_2 - x3 * z1_2 + x3 * r1_2 + x3 * x2_2 + x3 *
y2_2 + x3 * z2_2 - x3 * r2_2 - x3_2 * x2 - y3_2 * x2 - z3_2 * x2 +
r3_2 * x2 - x1 * x2_2 - x1 * y2_2 - x1 * z2_2 - 2 * x2 * z * z1 -
2 * x3 * z * z2 - x1_2 * x3 + x1 * y3_2 + x1 * z3_2 - x1 * r3_2 +
```

```
x1 * r2_2 + x1 * x3_2) / (x1 * y2 - x2 * y1 + x3 * y1 - x3 * y2 +
y3 * x2 - x1 * y3);
            //calcualte x
            x = (0.5) * (x1_2 * y2 - x1_2 * y3 - 2 * y3 * z * z2 -
2 * y1 * z * z3 + 2 * y1 * z * z2 - 2 * z * z1 * y2 + 2 * z * z1 *
y3 + 2 * z * z3 * y2 + y2 * r3_2 - y2 * x3_2 + y2_2 * y3 + z2_2 *
y3 - r2_2 * y3 + y1 * y3_2 + y1 * z3_2 - y1 * r3_2 + y1 * r2_2 +
y1 * x3_2 + y1_2 * y2 - y1_2 * y3 + z1_2 * y2 - z1_2 * y3 - r1_2 *
y2 + r1_2 * y3 + x2_2 * y3 - y2 * y3_2 - y2 * z3_2 - y1 * x2_2 -
y1 * y2_2 - y1 * z2_2) / (x1 * y2 - x2 * y1 + x3 * y1 - x3 * y2 +
y3 * x2 - x1 * y3);

// Mizrahi (2003)

            // check the values
            if (Double.IsNaN(x) || Double.IsNaN(y) ||
Double.IsNaN(z))
            {
                return false;
            }
            else
            {
                this.x = x;
                this.y = y;
                this.z = z;

                return true;
            }
        }
    }
}
```

DataAccess.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlServerCe;
using System.Reflection;
using System.IO;

namespace DistoPos
{
    // class to handle the database connection
    class DataAccess
    {
        private SqlCeEngine engine;
        private string connectionString;
        // get the connection
        public SqlCeConnection GetDbConnection()
        {
            try
            {
                SqlCeConnection connection = new
SqlCeConnection(engine.LocalConnectionString);
                connection.Open();
                return connection;
            }
            catch
            {
                return null;
            }
        }
        // initialize the database connection
        public bool InitDB(out string connString)
        {
            try
            {
                // get the application path
                string sAppName =
Assembly.GetExecutingAssembly().GetName().CodeBase;
                string sAppPath = Path.GetDirectoryName(sAppName);

                // make the connectionstring
                connectionString = "Data Source='" + sAppPath +
"\\DistoPosDB.sdf';";
                connString = sAppPath + "\\DistoPosDB.sdf";

                engine = new SqlCeEngine(connectionString);

                return true;
            }
            catch
            {
                connString = "";
                return false;
            }
        }
    }
}
```