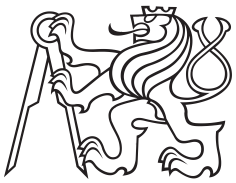
Master's Thesis



Czech Technical University in Prague

F1

Faculty of Civil Engineering
Department of Geomatics

Combinatorial computation of coordinates in the GNU Gama project

Bc. Petra Millarová

Supervisor: prof. Ing. Aleš Čepek, CSc.
Field of study: Geodesy and Cartography
May 2019

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta stavební
Thákurova 7, 166 29 Praha 6

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: Millarová Jméno: Petra Osobní číslo: 439263

Zadávající katedra: Katedra geomatiky

Studijní program: Geodézie a kartografie

Studijní obor: Geomatika

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce: Kombinatorický výpočet souřadnic v projektu GNU Gama

Název diplomové práce anglicky: Combinatorial computation of coordinates in the GNU Gama project

Pokyny pro vypracování:

Projekt GNU Gama [1] je věnován vyrovnání lokálních geodetických/zeměměřických sítí. Pokud nejsou známy souřadnice vyrovnávaných bodů, program přibližné souřadnice vypočte podle algoritmů publikovaných v [4] a implementovaných v [5]. Pořadí v jakém jsou přibližné souřadnice nemůže uživatel ovlivnit a přibližné souřadnice nejsou před vyrovnáním kontrolovány (předpokládá se, že měření neobsahují hrubé chyby). Tento postup může selhat v případě vyrovnání dlouhých navazujících polygonů, kde se běžné měřické chyby mohou kumulovat a znehodnotit odhady přibližných souřadnic.

Navrhněte a implementujte řešení, které bude kontrolovat vypočtené souřadnice přibližných bodů (například vícenásobným určením, pokud je to možné) a které bude explicitně vyhledávat a samostatně řešit polygonové pořady (určené oboustranně, jednostranně a vložené polygonové pořady).

Seznam doporučené literatury:

[1] Dokumentace projektu GNU Gama https://www.gnu.org/software/gama/
[2] B. Stroustrup, The C++ Programming language, 4th edition
[3] C++ reference, https://en.cppreference.com
[4] F. Charamza et al., Geolib / PC, VÚGTK 1989
[5] J. Veselý, Výpočet přibližných souřadnic bodů v C++, dipl.práce, 1999

Jméno vedoucího diplomové práce: prof. Ing. Aleš Čepek, CSc.

Datum zadání diplomové práce: 20. 2. 2019 Termín odevzdání diplomové práce: 20. 5. 2019

Údaj uveďte v souladu s datem v časovém plánu příslušného ak. roku

Podpis vedoucího práce Podpis vedoucího katedry

III. PŘEVZETÍ ZADÁNÍ

Beru na vědomí, že jsem povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je nutné uvést v diplomové práci a při citování postupovat v souladu s metodickou příručkou ČVUT „Jak psát vysokoškolské závěrečné práce" a metodickým pokynem ČVUT „O dodržování etických principů při přípravě vysokoškolských závěrečných prací".

21.2. 2019

Datum převzetí zadání Podpis studentky

Acknowledgements

First and foremost I would like to thank my supervisor for keeping me on the right track, providing great advice and encouraging me throughout the whole process of writing this thesis. I would also like to thank my rubber ducks and my friends, who have proved to be just as useful as the ducks and have always listened to me and provided the much needed silence and support when I was trying to debug my code.

Declaration

I hereby declare that I have written this thesis on Combinatorial computation of coordinates in the project GNU Gama by myself and only using the literature provided in the list of references at the end of this work.

Prague, May 15, 2019

Prohlašuji, že jsem svou magisterskou práci Kombinatorický výpočet souřadnic v projektu GNU Gama vypracovala samostatně a pouze za použití literatury uvedené na konci textu.

V Praze, 15. května 2019

Abstract

This work deals with the calculation of approximate coordinates in gama-local in the GNU Gama project. It describes the current state, where the solution fails with computation of long traverses due to the cumulation of measurement errors. The second half serves as documentation to the newly created code. The result of this thesis is a new solution, which iterates through the data and searches for suitable observations for calculation of the polar method and detection and calculation of traverses. The computed points are only added only if they can be checked by multiple solutions.

Keywords: GNU Gama, approximate coordinates, traverse, polar method, C++

Supervisor: prof. Ing. Aleš Čepek, CSc.

Abstrakt

Tato práce se zabývá výpočtem přibližných souřadnic v programu gama-local v projektu GNU Gama. Popisuje současný stav, kdy řešení selhává při výpočtech dlouhých polygonů díky hromadění měřických chyb. Druhá polovina práce slouží zároveň jako dokumentace k nově vzniklému kódu. Výstupem práce je pak nové řešení, které iterativně prochází data a vyhledává vhodné observace pro výpočet polární metody a sestavení a výpočet polygonů. Takto spočtené body přidává pouze pokud mohou být zkontrolovány vícenásobným určením.

Klíčová slova: GNU Gama, přibližné souřadnice, polygon, polární metoda, C++

Překlad názvu: Kombinatorický výpočet souřadnic v projektu GNU Gama

Contents

Figures

Tables

Chapter 1

Introduction

GNU Gama is a project aimed at adjustment of geodetic networks, especially those that were measured using traditional surveying methods. These are used today mainly in specialized measurements (e.g. mining or deformation surveys), where the use of Global Positioning Systems (GPS) is not possible [1]. The program calculates unknown coordinates using algorithms originally published in [6], that were first implemented by J.Veselý in [11]. This computation process is fully automated and the user only needs to provide measured data. Due to the nature of these algorithms, however, the program sometimes has difficulties with long traverses, where measurement errors can quickly add up. The existing class also expects data to not contain any gross errors. These are the two main reasons why this thesis aims to design and implement a set of classes which compute at least some of the approximate coordinates differently. This thesis also briefly introduces GNU Gama itself and the data structure necessary to understand the newly implemented classes, as well as the current solution.

GNU Gama and the presented new code are part of the GNU project, that means that it is free software. Free here does not necessarily mean free of cost (although GNU Gama is free in this sense as well), but rather free as in that the user has certain freedoms that they can enjoy. These four essential freedoms are the four building blocks of free software. First is the freedom to use the program for whatever reason the user wishes. Second is the freedom to study and edit the source code. Third and fourth are freedoms to redistribute copies of the program and to redistribute copies of the program with any changes you made to it. These last two points enable users to work together to modify the program, as was done for the purpose of this thesis. Free software puts the users' freedom and rights above everyone else's. That is why users should be able to modify and run software however best it fits their purpose. Proprietary software usually only has a handful of developers who can modify the software and the user is not even allowed to see the source code, that means that the software is not free because the developers have more power than the user [4].

Free software may evoke another term – open source, however, they are not the same. Although all free software is open source, not all open source programs can be called free software. Some software even presents itself as free,

but devices prevent you from running its modified executables, such as some Android products. Open source concentrates more on the practical benefits, such as better quality of software, while free software is more concerned with the ethics and users' freedom [5].

Chapter 2

GNU Gama

The manual for GNU Gama [2] states: *"Gama was originally inspired by Fortran system Geodet/PC (1990) designed by Frantisek Charamza. The GNU Gama project started at the department of mapping and cartography, faculty of Civil Engineering, Czech Technical University in Prague (CTU) about 1998 and its name is an acronym for geodesy and mapping. It was presented to a wider public for the first time at FIG Working Week 2000 in Prague and then at FIG Workshop and Seminar at HUT Helsinki in 2001."*

GNU Gama consists of several programs, one of which is the console application *gama-local*, which this thesis contributes to. The application performs adjustment of networks of observations in local Cartesian coordinate systems. Global coordinate systems are only partially supported in the *gama-g3* program and there is also a GUI available called *gama-q2*.

2.1 Licensing

The involved project and the newly presented code are both subjects to a copyleft license. Where copyright grants the author certain rights, copyleft also states that these rights must also be granted to others who use and/or modify the code [8], therefore any works derived from this software must use the same license as the original [10].

GNU licensing was written by Richard Stallman for the GNU project, which is a collaborative software project that originally started with the intention to create a free operating system under this license. Today the project is maintained and sponsored by the Free Software Foundation. GNU is a recursive abbreviation that stands for GNU's not Unix, which highlights the fact that GNU is a Unix-like system that does not include any Unix code and is licensed as free software [13].

2.1.1 GNU General Public License

GNU Gama is released under the GNU General Public License. GNU general public license (GNU GPL) is a free-software license, this means that any software using this license is freely available and can be modified, extended,

redistributed and used. GNU GPL is one of the original and most widely used of all the so–called copyleft licenses [12].

A person is not required to accept this license if they only intend to use a copy of a GNU GPL licensed program, however, changing the source code or distributing the work means you accept the license.

2.2 Format of input data

The console application gama-local uses a structured input of measured data and information about the geodetic network in the form of an xml file or the same file with the gkf suffix. The tag structure can be observed in Table 2.1, a whole file is available as Appendix B. Currently there can only be one `network` tag, but the option of adding more is planned in the future releases of the program [2]. This tag can also be used to specify axes orientation and right-handed or left-handed angles. The `description` tag is optional and is used for an annotation and/or comments about the data. The `parameters` tag is also optional and can be used to set additional attributes. These attributes can either be technical, such as encoding, language or output angle units or they can help refine the calculation. These include changing the numerical algorithm used, setting a different confidence interval for statistical tests, choosing which ellipsoid to use or adjusting the *a priori* square root of reference variance value. The `points-observation` tag is obligatory and contains information about points and observations and optionally can also set the implicit standard deviation values for different types of observations.

Within the `points-observations` tag you can specify which points are control points and which coordinates you want adjusted. You do so by using

```
<gama-local>
<network>
   <description> ... </description>
   <parameters ... />
   <points-observations>
                <point id = 'A' x='5000' y='1000' z='0' fix='xyz' />
                <point id='A' x='0'  y='0' fix='xy' />
                <point id='B' x='400'  y='0' fix='xy' />
                <point id='C' adj='xy' />

                <obs from='A'>
                        <angle bs='C' fs='B' val='100' />
                        <distance  to ='C' val='300' />
                </obs>

                <obs from='B'>
                        <angle bs='A' fs='C' val='40.9666' />
                        <distance  to ='C' val='500' />
                </obs>
   </points-observations>
</network>
</gama-local>
```

Table 2.1: Structure of data [2]

the `point` tag, in which you have to specify the ID if the point (which can be a number as well as a string of letters or a combination of both). Other attributes are optional and depend on what you want the program to do. You can specify known coordinates and and then which coordinates you do not want to change in the adjustment (using `fix`) and which coordinates you want the program to adjust (using `adj`).

Another tag which is used within the `points-observations` tag is `obs`, which is a pair tag used to group together a set of observations, usually those that are observed from one survey station. In this tag you can specify the type of measurement and the value, along with additional details. For a more elaborate description of the different observation types you can enter, please see the manual [2]. Additionally, you can also supply the variance-covariance matrix within the `obs` tag by using `cov-mat`.

The angular unit used in `gama-local` is implicitly gons, expressed using decimal numbers. If one wishes to use degrees instead, it is possible to enter the number in the DD-MM-SS.SS format, i.e. degrees, minutes and seconds each separated by a dash.

2.3 Format of output data

GNU Gama implicitly writes calculation results to standard output in xml format. This can be changed by setting the appropriate option while running the program. The xml, html and text options all provide the same following information, which can be divided into several categories.

The general parameters of the adjustment – these contain a summary of the computation, including the number of adjusted, constrained and fixed coordinates. The text and html formats also include the approximate coordinates summary which is done before the adjustment.

The coordinates – fixed coordinates provided by the user, followed by adjusted coordinates and the mean errors and parameters of the error ellipses.

Observations' summary – contains the numbers of each type of observation if present. For more information about observations see Chapter 3.1.1.

Project equations and standard deviations – the number of equations and degrees of freedom and information about the a priori and a posteriori standard deviations and which one was used in the computations. Also listed is the probability and confidence interval.

Point information – fixed and adjusted coordinates, in case of text and html output with their standard deviation confidence interval.

Adjusted observations – this includes the observations' original values, residuals and analysis.

The xml and text output formats include information about the version of the program and additionally the xml format also adds the information about the configuration of axes and angle orientation and the covariance matrix. It also lists the original indexes of points from the adjustment. The text format includes a summary of the approximate coordinates, which the other outputs lack.

The user can also choose for gama-local to produce an `*.m` file which consists of the following matrices that can be opened directly by Octave or Matlab.

These matrices are:

- `Points` - contains IDs of computed points
- `Indexes` - indexes of adjusted covariances
- `XYZ` - coordinates of the computed points
- `Cov` - the covariance matrix of adjusted coordinates
- `FixedPoints` - IDs of fixed points along with a matrix with their coordinates

Another output format is `*.svg`, which produces an image of the network configuration. Example outputs for input shown in Tab. 2.1 can be found in Appendix C.

2.4 Builds

As it was mentioned before, GNU Gama is part of the GNU Project. Following the GNU standards, it uses GNU build system to configure and build the program. The GNU Gama package has a `configure.ac` script, that adjusts the `Makefile` for the user's operating system and they can then build the project. The tools used to do this are collectively known as Autotools. These tools also enable conditional parameters, which allow GNU Gama to have different builds, depending on libraries that are available (SQLite3 database support, Valgrind memory tests etc). Along with not being able to run on Windows without the use of extra software, another disadvantage of Autotools is their complicated syntax. However, once it works it is a highly robust and reliable build suite.

The aforementioned `configure.ac` script contains the basic definition of build parameters. The whole file can be observed in Appendix D.1. An inexperienced reader might passively understand what each parameter means, but various definitions are far from being intuitive. After running the `configure` command (Fig. 2.1), a series of `Makefiles` is generated and the user can run the `make` command. After doing this, the new build is ready. It can be verified by running tests with the `make check` command, for more details on this see Chapter 5.1.

2.4.1 Alternative builds

Autotools are only available on Windows if the user has `Cygwin` or `MinGW` installed. This is why GNU Gama offers other alternative development platforms. A `gama-local.pro` project file for Qt Creator is available in the distribution for building gama-local. Another supported build system is `CMake`,

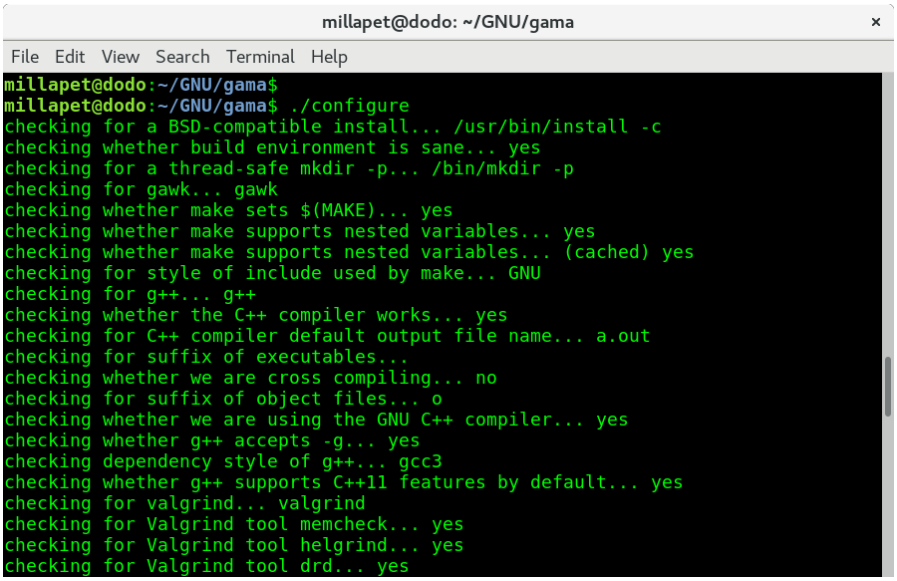


Figure 2.1: Configuring GNU Gama using Autotools

whose projects are supported by not only Qt Creator, but also Microsoft Visual Studio (see Fig. 2.2), among others. Thus GNU Gama is available on every platform that can run `CMake`.

It is good practice to create a separate *build* directory for the generated build files. That way it's easier to clean the project – by simply deleting the folder.

The easiest way to build a solution for Microsoft Visual Studio is using the Command Prompt for VS 2017. The only thing the user has to do is to create a build directory and then execute the `cmake` command (Fig. 2.3). After that, the user can either open the solution in MS Visual Studio IDE or stay in the command prompt and write `cmake -build . -config Release`, which will build a release version of the program (Fig. 2.4).

`CMake` is also available on Unix-style systems, where it produces `Makefiles` as output when the command `cmake` is used (see Fig. 2.5). These can then again be built with the `make` command (Fig. 2.6).

`CMake` projects are made up of a series of `CMakeLists.txt` files, where the build parameters are described. If we compare this file (see Appendix D.2) to `configure.ac`, which was mentioned in the previous chapter, we can clearly see that `CMake` project definitions are much more straightforward and simple. At the moment, not all functionalities of GNU Gama are implemented in the `CMake` projects, but unit tests created for verifying `Acord2` are available and are described in detail in Chapter 5.1.

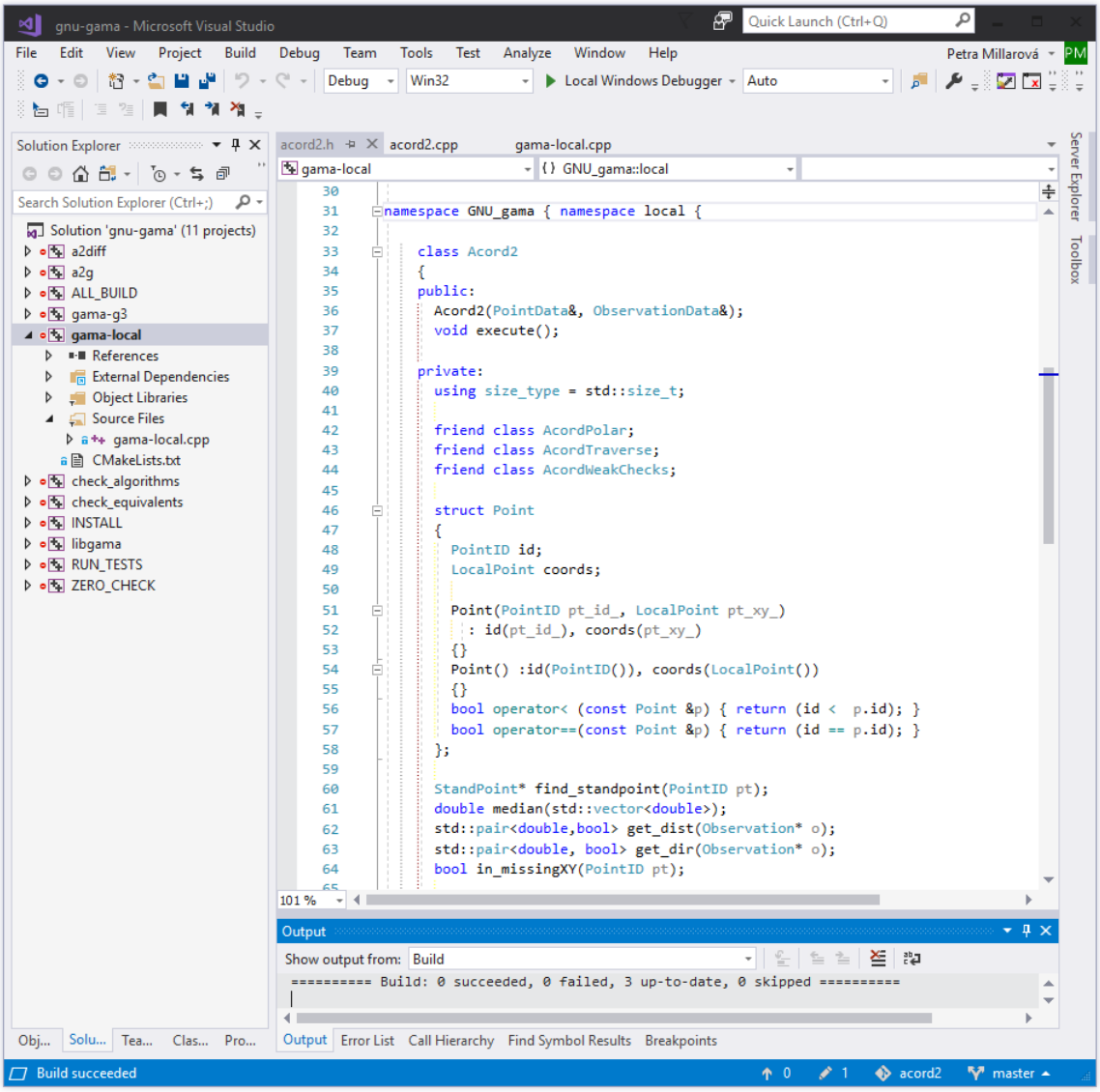


Figure 2.2: GNU Gama in Microsoft Visual Studio 2017

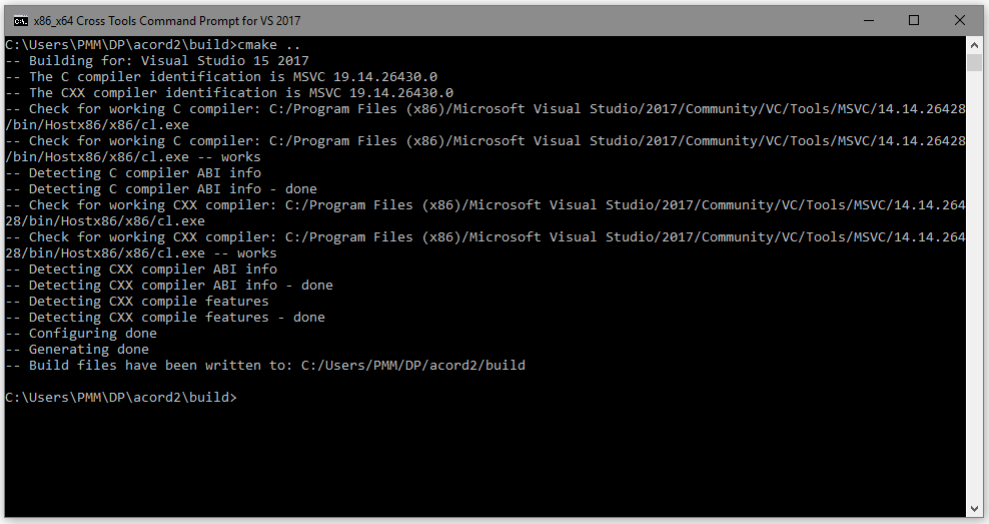


Figure 2.3: Preparation for building GNU Gama on Windows using `CMake`

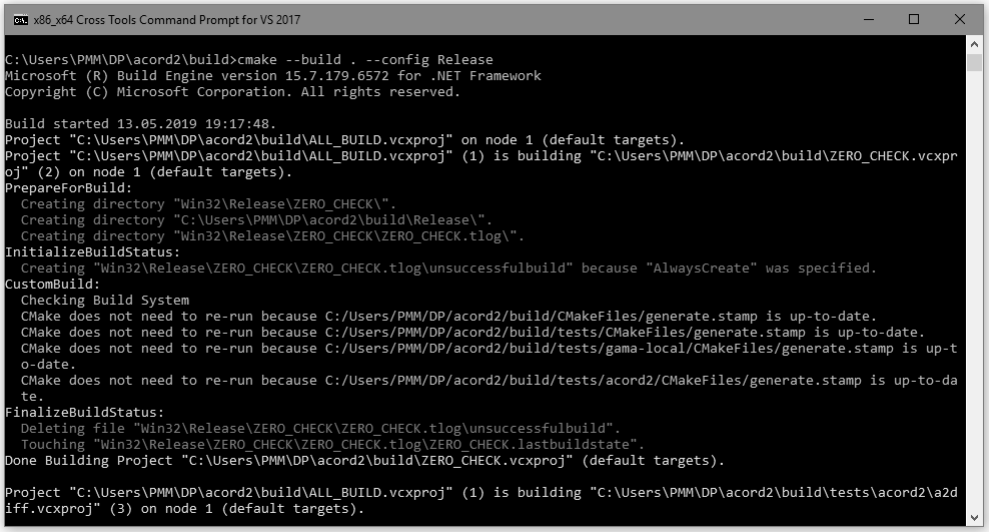


Figure 2.4: Building GNU Gama on Windows using `CMake`

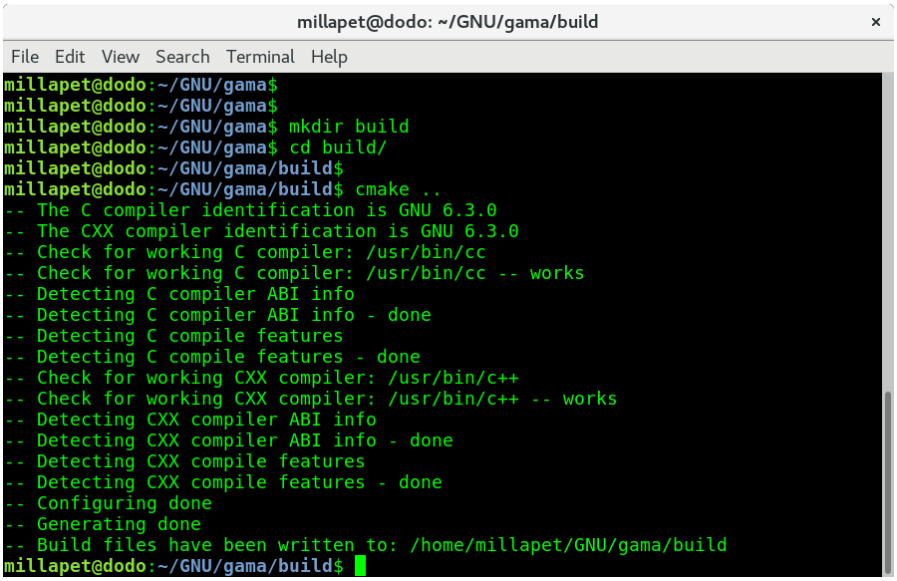


Figure 2.5: Generating Makefiles using `CMake`

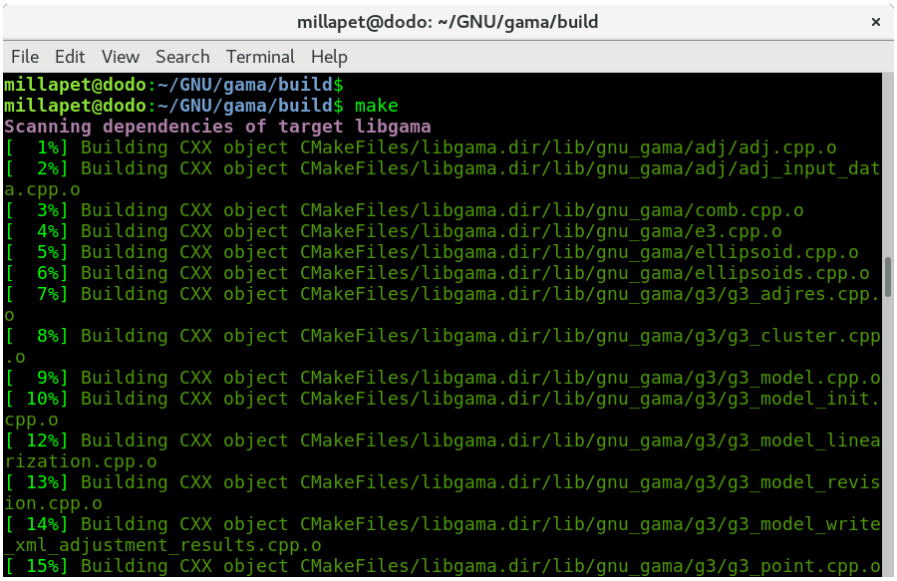


Figure 2.6: Building GNU Gama using Makefiles generated with `CMake`

Chapter 3

Problem analysis

Before gama-local starts the adjustment computations, it calculates approximate coordinates with intersection and resection methods. Unfortunately, this sometimes causes large errors while computing traverse points. This thesis proposes addition of other methods of calculation to prevent such errors from happening.

This chapter further elaborates on the structure of data in GNU Gama (3.1) and the existing class `Acord` (3.2).

3.1 Structure of data in GNU Gama

Gama includes several observation data structures, which are used to store measured data and allow adjustment of any combination of possibly correlated observations [1].

Both *gama-local* and *gama-g3* use template classes `ObservationData` and `Cluster` as their base classes. An `ObservationData` object stores all `Clusters`. These are objects with a common variance-covariance matrix and a list of pointers to specific `Observation` objects (see Chapter 3.1.1). Cluster types can be `Coordinates` (stores observed coordinates), `HeightDifferences` (data from leveling), `Vectors` (coordinate differences) or `StandPoint`, which holds observations (measurements) obtained with classic land surveying from one survey station and stores them in an `ObservationList`.

Clusters can also hold orientation shifts and thus allow them to be separated from the measurements and one survey station can have more orientations.

3.1.1 The Observation class and its derived classes

Each `Observation` is initialized with a value of type `double`, that holds the measured value, and two `PointID`s, which represent IDs of the beginning and end point of the observation. These IDs can be associated with their coordinates through the `PointData` class [11], which holds information about the local coordinate system and angular observations (left-handed angles vs right-handed angles) as well as the `PointID` and coordinates in a `LocalPoint` for every point with fixed or computed coordinates.

The `Observation` class has several derived classes, which each allow storage of a particular type of measurement.

These classes are:

- `Distance`
- `Direction`
- `Angle`
- `H_Diff` - height differences
- `Xdiff`, `Ydiff`, `Zdiff` - observed coordinate differences (vectors)
- `X`, `Y`, `Z` - observed coordinates
- `S_Distance` - slope distance
- `Z_Angle` - zenith angle
- `Azimuth`

3.2 The Acord class

This class was written by Ing. Jiří Veselý as part of his Master's thesis [11]. The `Acord` class takes a `PointData` and an `ObservationData` object and, after the `execute` member function is called, computes approximate coordinates from the information provided. It uses intersection and resection methods, namely intersection by bearings, intersection by distances and intersection by coordinated line and circle.

This class works fine with most input data, but a problem arises when one tries to compute long traverses, because the errors from each point propagate to the next. Another problem is that input data is expected to be free of any gross errors, which is not always the case. Both these problems are solved by introducing a class that offers a robust solution for most typical two dimensional inputs. It finds as many solutions as possible for each point and then compares them and stores their median. This is then followed by the current class before the network is adjusted. A closer look at how the algorithm works is provided in Chapter 4.2.

Chapter 4

Development

The first version of GNU Gama was introduced in the year 2000 and although it is written in compliance with the GNU Coding Standards [3], the source files are not completely uniform in style. Therefore the author decided to also use some recommendations from the Google Styleguide [7].

The requirements for the presented solution were as follows:

Functional requirements:

- additional methods of computation of approximate coordinates
- a method of checking coordinates before adding them to the previously computed and/or given coordinates

Non-functional requirements:

- implementation in C++11 (as it is one of the major milestones in C++ development)
- integration with existing code in `gama-local`

4.1 General principles

This chapter explains the basic calculations that were implemented as part of this thesis. The images and variables were all taken from [9].

4.1.1 Polar method

Polar method is one of the basic methods to determine the coordinates of one point. For this computation, one survey station with known coordinates (say P) is needed, along with a bearing (σ_{PK}) and distance (s_{PK}) measured to the point with unknown coordinates K. The coordinate differences are then computed from a triangle with simple sine and cosine functions:

$$\Delta x_{PK} = s_{PK} \cdot \cos \sigma_{PK} \tag{4.1}$$

$$\Delta y_{PK} = s_{PK} \cdot \sin \sigma_{PK} \tag{4.2}$$

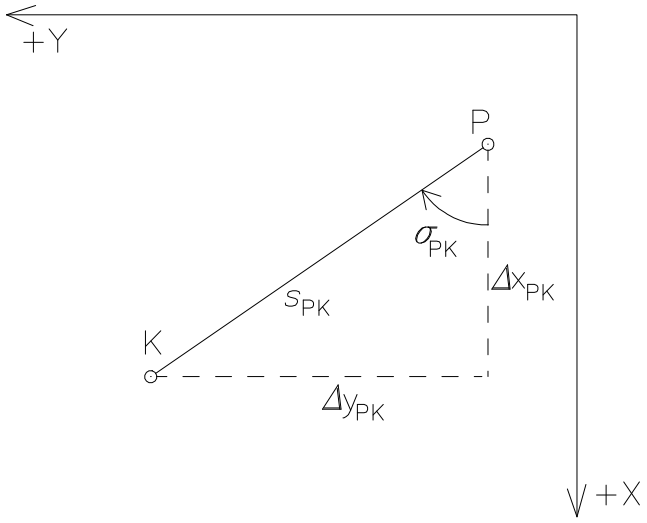


Figure 4.1: Polar method (Image source: [9])

This can be then added to the known coordinates to obtain coordinates of the unknown point.

$$X_K = X_P + \Delta x_{PK} \tag{4.3}$$

$$Y_K = Y_P + \Delta y_{PK} \tag{4.4}$$

4.1.2 Traverses

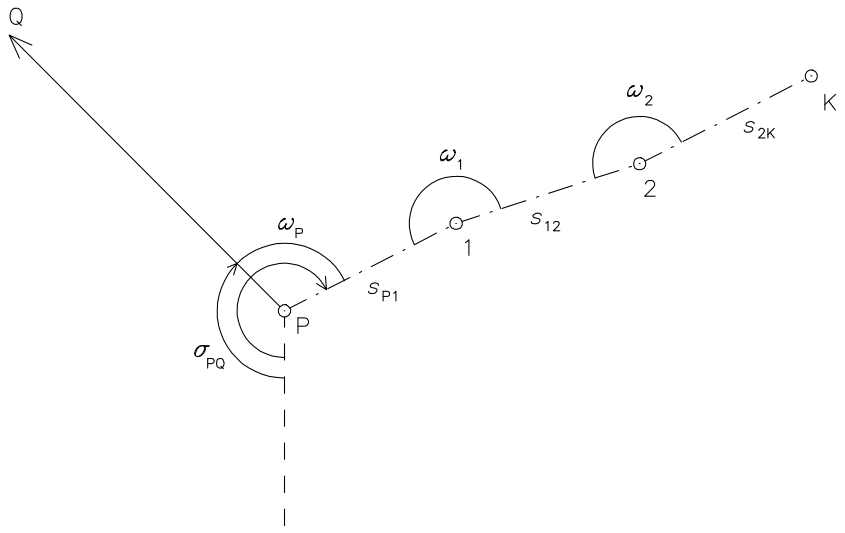


Figure 4.2: Traverse calculation (Image source: [9])

Traverse computation is an efficient method of computing more points at once. In this thesis the traverses were computed as open traverses with a known point and an orientation direction at the beginning, as can be seen in Fig. 4.2. If an orientation point was not available, traverses were computed as closed, with known points at the beginning and the end of the traverse and the first bearing was set to zero.

This ensures that in each case we have at least two known points to perform a similarity transformation afterwards. If the closing point of the traverse also has known coordinates then it's also used in the transformation. The reason for using similarity transformation after computing traverses is the minimization of errors and also to avoid having to calculate angular misclosures and adjusting the coordinate differences in the event of a closed traverse. By transforming the computed points, they are "fitted" to the coordinate system of the known points.

The principle of calculating traverse points is simple – we apply the polar method gradually to each newly computed point. Firstly, the angles between each two successive traverse lines are needed, as are the measured lengths of these traverse lines. With this information we can calculate the bearing to the first traverse line like so:

$$\sigma_{P1} = \sigma_{PQ} + \omega_P \tag{4.5}$$

And subsequently the rest of the bearings in the traverse:

$$\sigma_{i,i+1} = \sigma_{i-1,i} + \omega_i - 2R \tag{4.6}$$

Where i represents the i^{th} point of the traverse and $2R$ represents a straight angle. Now the coordinate differences can be computed:

$$\Delta x_{i,i+1} = d_{i,i+1} \cdot \cos(\sigma_{i,i+1}) \tag{4.7}$$

$$\Delta y_{i,i+1} = d_{i,i+1} \cdot \sin(\sigma_{i,i+1}) \tag{4.8}$$

And finally the coordinates of each of the points:

$$X_i = X_P + \sum_{i=P-1}^{K} \Delta x_{i-1,i} \tag{4.9}$$

$$Y_i = Y_P + \sum_{i=P-1}^{K} \Delta y_{i-1,i} \tag{4.10}$$

4.2 Implementation

The proposed solution works on the principle of storing calculated points in a variable from `Acord2` and only adding a point to the main PointData object if there are two or more computations of that point or the point's coordinates can be otherwise checked. This is the main difference from the the existing class `Acord` made by Ing. Jiří Veselý (see Chapter 3.2).

Acord2 is called from gama-local right before the Acord class and attempts to compute as many points as it can. The goal is for Acord2 to calculate as many points as possible, not all of them. This class is not meant to replace the current one, but merely to extend the existing calculation of approximate points as a whole. This is further supported by the fact that Acord2 only computes plane coordinates without height differences.

All the new classes were implemented in lib\gnu_gama\local\acord path of GNU Gama, the names of files correspond to all-lower-case names of the classes. All of the new code can be viewed in Appendix E.

4.2.1 Acord2

The solution consists of three classes, AcordPolar, AcordTraverse and AcordWeakChecks. They are all friend classes of Acord2, which allows them to access private variables of the main class. Acord2 controls the initialization of the three other classes and and stores information about the calculations (such as the number of new points, which points are still uncomputed etc.) and variables as well as functions that are used in more than one of the classes.

An additional class AcordStatistics was implemented for debugging purposes and output statistics. Class Acord2 takes objects of types PointData and ObservationData as references in the constructor (for more information about these types see Chapter 3.1.1). All the other new classes take an instance of Acord2 . This ensures all classes operate with the same data.

The following list presents some notable variables and functions from Acord2 that are used by the other classes.

Variables:

- MAX_NORM – variable that signifies the maximum accepted normal deviation of two values
- missingXY_ – set of all PointID objects with unknown coordinates
- obs_from_ – enables searching for observations from a point with a specific PointID
- obs_to_ – enables searching for observations that point to a point with a specific PointID
- same_points_ – stores computed coordinates along with their PointID, these have not been added to PointData yet
- SPClusters_ – all StandPoint objects from ObservationData
- traverses – stores all computed traverses that have not yet been added to PointData

Functions:

- get_dir – checks if supplied Observation is a Direction and returns a double and boolean pair with the value

- `get_dist` – checks if supplied `Observation` is a `Distance` and returns a double and boolean pair with the value
- `get_medians` – finds all points from `same_points_` that have the same `PointID` and checks if the maximal differences of computed coordinates are within `MAX_NORM` and if that is the case, stores the medians of coordinates in `PointData`
- `median` – returns a median value of supplied doubles
- `transform_traverse` – performs a similarity transformation upon a given traverse and stores the adjusted points into it

The flowchart in Fig. 4.3 shows how the individual classes are called from within `Acord2`.

4.2.2 AcordPolar

This class finds respective observations and computes approximate coordinates using the polar method, as described in Chapter 4.1.1. The class's `execute` function goes through all objects in `SPClusters_` and calls the `points_from_SPCluster` function on each `StandPoint`.

This function then takes the `StandPoint`, makes a copy of the list of local observations, sorts them by which point they lead to and processes all the observations that lead to a point with unknown coordinates. Sorting allows for effective processing, all observations for one given point are found and the function tries to apply the polar method if there is enough sufficient information.

Observations of the `Distance` type are stored in a vector and then only their median is used. Different `Direction` and `Angle` observations are processed separately and produce multiple results that are all stored in `same_points_` and are produced using the same median distance. If a stub point is detected then it is added straight to the main PointData object, since there is no other way to compute this point.

When the processing of `SPClusters_` ends and if newly added points are detected, the function calls `Acord2`'s function `get_medians`, for details of which see Chapter 4.2.1. This whole process repeats until either all clusters have been used, new points have not been added or there is nothing else to compute (`missingXY_` is empty). The whole process is summed up in Fig. 4.4.

4.2.3 AcordTraverse

The `AcordTraverse` class first finds all candidate points for the traverses: points that are only connected with two other points. Then the `execute` function goes over these points and finds the ones that make up a traverse. It then finds the end points of the traverse and determines the type of traverse depending on whether both end points are known (`closed_traverse`), none (`open_traverse`) or only one (`closed_start_traverse`).

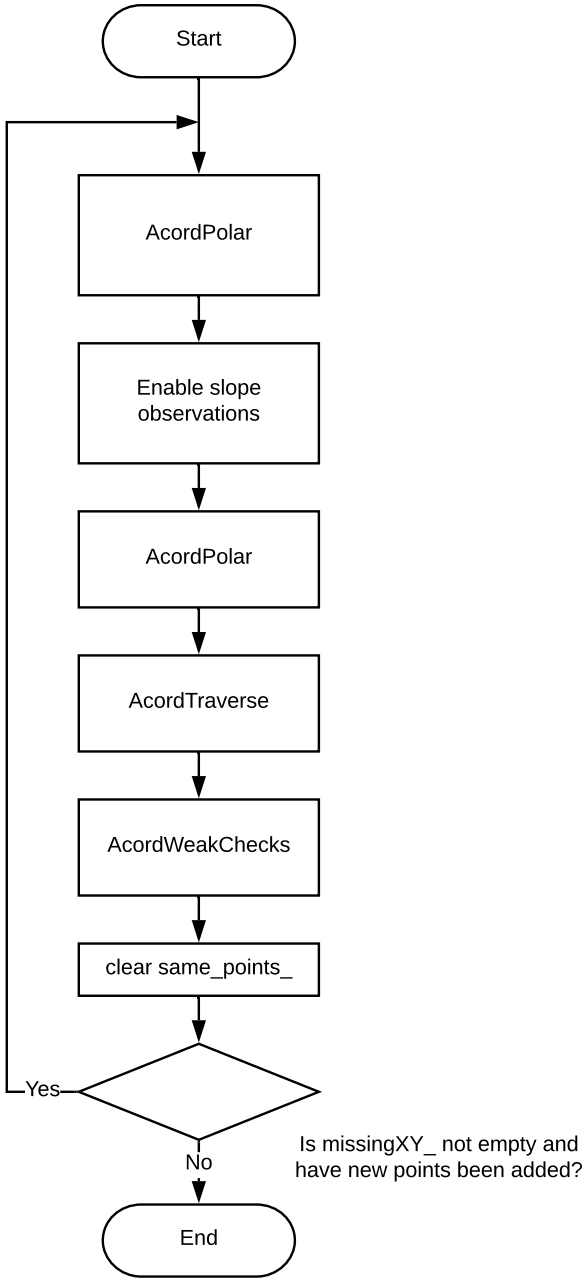


Figure 4.3: Flowchart for `Acord2.execute()`

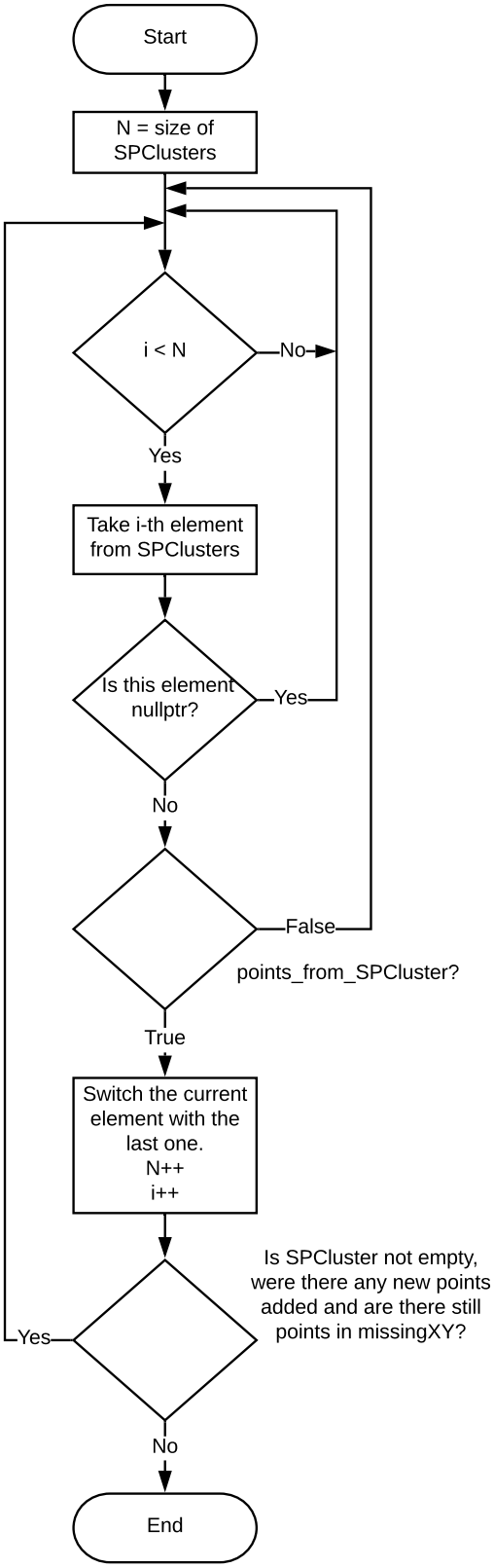


Figure 4.4: Flowchart for the `AcordPolar.execute()` function

The `calculate_traverse` function is then called, first finding appropriate distances and angles and then calculating the coordinates of all points. These coordinates are then transformed to fit within the existing points, thus eliminating the need of manually adjusting the traverse. All computed traverses are stored in a `traverses` object from class `Acord2` .

After processing all the candidate points we add closed traverses to the `PointData` object and search the rest for any missing points that appear in more than one traverse. If such points are found they are added to `same_points_` and the traverse is transformed again using the new coordinates and all new points are added to the `PointData` object. The flowchart in Fig. 4.5 summarizes the way `AcordTraverse` works.

4.2.4 AcordWeakChecks

The final friend class of `Acord2` searches end points of the traverses that have not yet been added to the `PointData` object and finds observations leading to endpoints with known coordinates. If these observations correspond to the values computed from coordinates, the point is considered to be computed correctly and is added to `PointData` and the rest of the traverse is transformed and added as well.

The remaining `same_points_` are treated in a similar fashion. First observations that lead to these points are found and if the values computed from coordinates match, the point is added.

4.2.5 AcordStatistics

This class is used for logging information about the state of computation of approximate coordinates. An instance of this class is declared before the first usage of `Acord2` and `Acord`. The `execute` function can then be run whenever there is need for a summary.

`AcordStatistics` provides the number of points with known coordinates, computed coordinates and the total number of points, along with the total number of observations and a boolean value of whether there are any missing coordinates or not. This information can then be passed to the existing template function `Approximate_coordinates`, which prints this information into the output file. In Tab. 4.1 we can see that each column concerns different combinations of coordinates; either just position (`xy`), height (`z`) or all three (`xyz`).

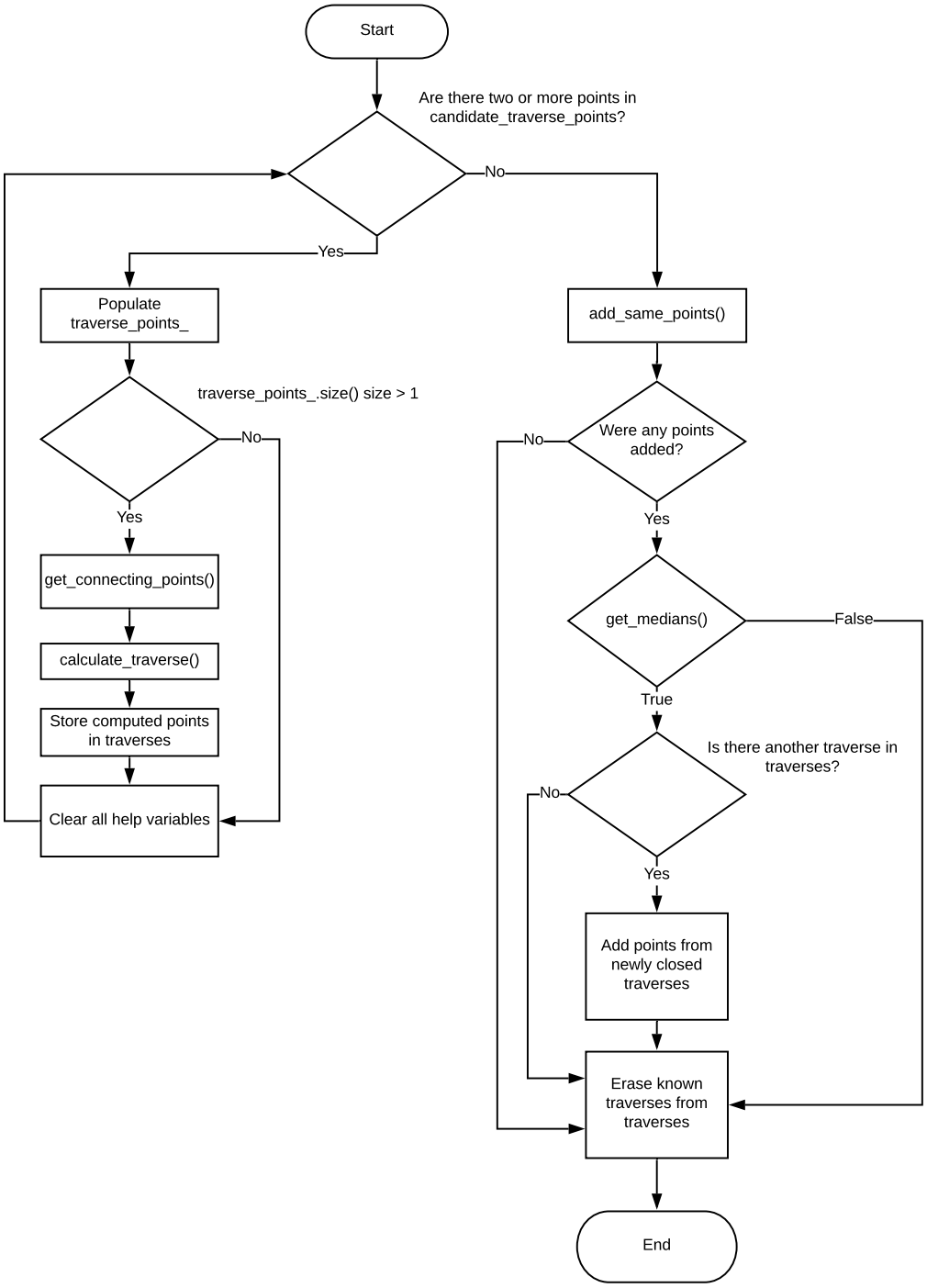


Figure 4.5: Flowchart for the `AcordTraverse.execute()` function

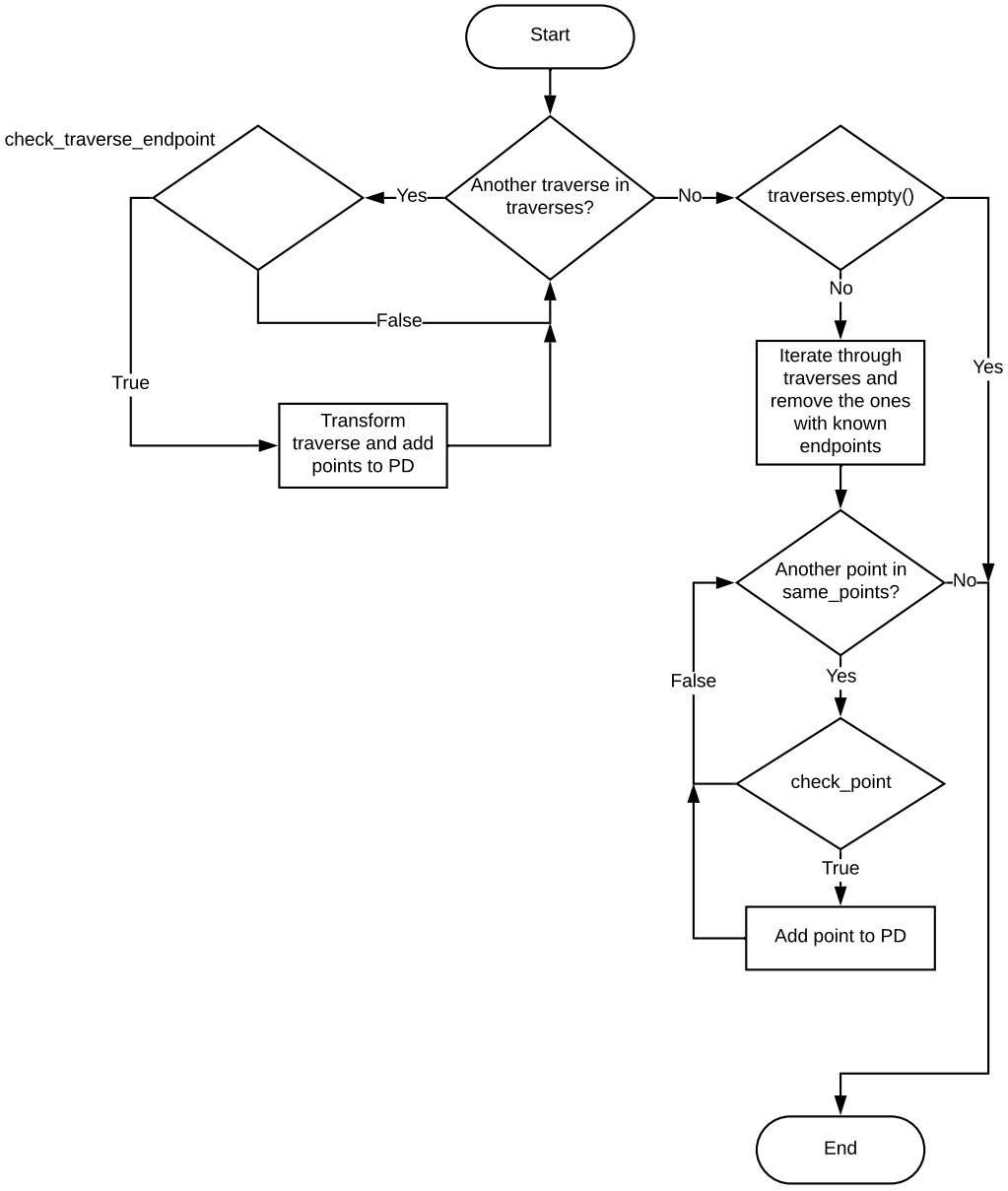


Figure 4.6: Flowchart for the `AcordWeakChecks.execute()` function

```
Approximate coordinates
***********************

coordinates           xyz         xy          z

given       :           0          3          0
computed    :           0         10          0
-----------------------------------------------
total       :           0         13          0

observations:          45
```

Table 4.1: Example output of the `Approximate_coordinates` function

Chapter 5

Usage of Classes

Class `Acord2` is called in the file `gama-local.cpp` before the existing `Acord` class, as shown in Tab. 5.1 on line 4, where an instance of the class is created and on line 5, where the `execute` function is called. `AcordStatistics` is created before both of the classes for approximate coordinate calculation and executed right before it is used in `ApproximateCoordinates` on lines 23 and 24.

```
1  ...
2  AcordStatistics stats(IS->PD, IS->OD);
3
4  Acord2 acord2(IS->PD, IS->OD);
5  acord2.execute();
6
7  Acord acord(IS->PD, IS->OD);
8  acord.execute();
9
10 ReducedObservationsText(IS,&(acord.RO), cout);
11
12 if (IS->correction_to_ellipsoid())
13   {
14     using namespace GNU_gama;
15     gama_ellipsoid elnum = ellipsoid(IS->ellipsoid().c_str());
16     Ellipsoid el;
17     GNU_gama::set(&el, elnum);
18     ReduceToEllipsoid reduce_to_el(IS->PD, IS->OD, el, IS->latitude());
19     reduce_to_el.execute();
20     ReducedObservationsToEllipsoidText(IS, reduce_to_el.getMap(), cout);
21   }
22
23 stats.execute();
24 ApproximateCoordinates(&stats, cout);
25 ...
```

Table 5.1: Excerpt from `gama local.cpp`

An additional preprocessor directive `DEBUG_ACORD2` was added and enables debug output which shows how many points were computed in which iteration and by which class. The console output of this directive is shown in Fig. 5.1.

5.1 Tests

As `Acord2` is part of the gama-local application, it must pass all the tests of the gama-local test suite. Additional tests `a2g` and `a2diff` were written

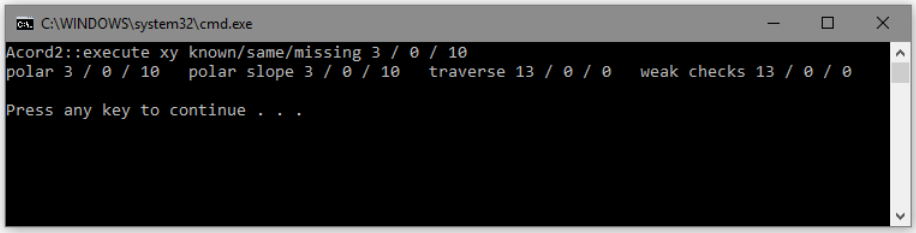


Figure 5.1: Output with the `DEBUG_ACORD2` preprocessor directive

to test the new class specifically. All these tests were written by A. Čepek. Like with all GNU projects, the tests can be run by performing *make check* on the top-level GNU Gama directory [2] or by running *ctest* if using `CMake` (see Fig. 5.2). The following output shows the results of a successful test run in Microsoft Visual Studio 2017:

```
1>------ Build started: Project: RUN_TESTS, Configuration: Release Win32 ------
1>Test project C:/Users/PMM/DP/acord2/build
1>      Start  1: check_algorithms_gama-local
1> 1/29 Test  #1: check_algorithms_gama-local .....................   Passed    0.05 sec
1>      Start  2: check_algorithms_zoltan-test_2d_dms
1> 2/29 Test  #2: check_algorithms_zoltan-test_2d_dms .............   Passed    0.13 sec
1>      Start  3: check_algorithms_zoltan-test_2d_gon
1> 3/29 Test  #3: check_algorithms_zoltan-test_2d_gon .............   Passed    0.13 sec
1>      Start  4: check_algorithms_zoltan-test_3d_dms
1> 4/29 Test  #4: check_algorithms_zoltan-test_3d_dms .............   Passed    0.37 sec
1>      Start  5: check_algorithms_zoltan-test_3d_gon
1> 5/29 Test  #5: check_algorithms_zoltan-test_3d_gon .............   Passed    0.37 sec
1>      Start  6: check_algorithms_tst-tetrahedron-1
1> 6/29 Test  #6: check_algorithms_tst-tetrahedron-1 ..............   Passed    0.02 sec
1>      Start  7: check_algorithms_tst-tetrahedron-2
1> 7/29 Test  #7: check_algorithms_tst-tetrahedron-2 ..............   Passed    0.02 sec
1>      Start  8: check_algorithms_tst-tetrahedron-3-deg
1> 8/29 Test  #8: check_algorithms_tst-tetrahedron-3-deg ..........   Passed    0.02 sec
1>      Start  9: check_algorithms_tst-tetrahedron-3-gon
1> 9/29 Test  #9: check_algorithms_tst-tetrahedron-3-gon ..........   Passed    0.02 sec
1>      Start 10: check_algorithms_fixed-azimuth
1>10/29 Test #10: check_algorithms_fixed-azimuth ...................   Passed    0.03 sec
1>      Start 11: check_algorithms_azimuth-angle
1>11/29 Test #11: check_algorithms_azimuth-angle ...................   Passed    0.03 sec
1>      Start 12: check_algorithms_azimuth-azimuth
1>12/29 Test #12: check_algorithms_azimuth-azimuth .................   Passed    0.03 sec
1>      Start 13: check_algorithms_azimuth-distance
1>13/29 Test #13: check_algorithms_azimuth-distance ................   Passed    0.03 sec
1>      Start 14: check_algorithms_triangle-1
1>14/29 Test #14: check_algorithms_triangle-1 ......................   Passed    0.02 sec
1>      Start 15: check_algorithms_triangle-2
1>15/29 Test #15: check_algorithms_triangle-2 ......................   Passed    0.02 sec
1>      Start 16: check_algorithms_stroner-levelling-a
1>16/29 Test #16: check_algorithms_stroner-levelling-a .............   Passed    0.02 sec
1>      Start 17: check_algorithms_stroner-levelling-b
1>17/29 Test #17: check_algorithms_stroner-levelling-b .............   Passed    0.02 sec
1>      Start 18: check_algorithms_extern-azimuth-distance
1>18/29 Test #18: check_algorithms_extern-azimuth-distance .........   Passed    0.03 sec
1>      Start 19: check_algorithms_extern-tst-tetrahedron-3-gon
1>19/29 Test #19: check_algorithms_extern-tst-tetrahedron-3-gon ...   Passed    0.02 sec
1>      Start 20: check_equivalents_zoltan_2d
1>20/29 Test #20: check_equivalents_zoltan_2d ......................   Passed    0.09 sec
1>      Start 21: check_equivalents_zoltan_3d
1>21/29 Test #21: check_equivalents_zoltan_3d ......................   Passed    0.18 sec
```

```
1>      Start 22: check_equivalents_scale
1>22/29 Test #22: check_equivalents_scale ........................   Passed    0.01 sec
1>      Start 23: check_equivalents_fixed
1>23/29 Test #23: check_equivalents_fixed ........................   Passed    0.02 sec
1>      Start 24: check_equivalents_gama-local
1>24/29 Test #24: check_equivalents_gama-local ...................   Passed    0.02 sec
1>      Start 25: check_equivalents_stroner
1>25/29 Test #25: check_equivalents_stroner ......................   Passed    0.02 sec
1>      Start 26: check_equivalents_tetrahedron
1>26/29 Test #26: check_equivalents_tetrahedron ..................   Passed    0.02 sec
1>      Start 27: check_equivalents_azimuth
1>27/29 Test #27: check_equivalents_azimuth ......................   Passed    0.02 sec
1>      Start 28: acord2-a2g
1>28/29 Test #28: acord2-a2g .....................................   Passed    0.11 sec
1>      Start 29: acord2-a2diff
1>29/29 Test #29: acord2-a2diff ..................................   Passed    0.05 sec
1>
1>100% tests passed, 0 tests failed out of 29
1>
1>Total Test time (real) =   2.03 sec
========== Build: 1 succeeded, 0 failed, 1 up-to-date, 0 skipped ==========
```

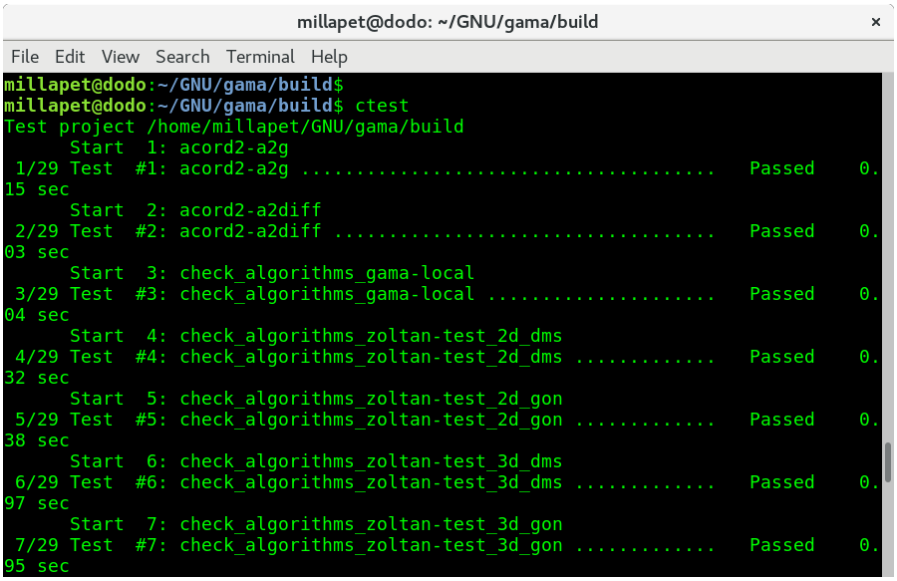


Figure 5.2: Running tests using `CMake` on a Unix system

a2diff

This test loads a network from a gama configuration xml file, computes approximate coordinates with both `Acord` and `Acord2` separately and then compares the final coordinates.

a2g

This test loads data from a file in a2g format, which is a set of coordinates and types of measurements in the counter-clockwise east-north coordinate system (see Tab. 5.2). The program then generates input files in gkf format for

gama-local in all possible combinations of coordinate axes for both clockwise and counter-clockwise angles and runs these in gama-local and then compares the results to the original coordinates.

```
! A 216.123 170.456   # fixed point A
! B 800.230 301.564   # fixed point B
? C 311.809 521.981   # adjusted point C with unknown xy
? D 622.672 608.490   # adjusted point D with unknown xy

( A     # standpoint A
= C     # distance + direction to C
> B     # direction to C
)       # end of~the~cluster

( B     # standpoint B
> C     # direction to C
> D     # direction to D
& A C   # angle A-B-C
& C A   # angle C-B-A
)

( C     # standpoint C
> A     # direction to A
> B     # direction to B
> D     # direction to D
- D     # distance  to D
& D A   # angle D-C-A
)
```

Table 5.2: Example input file for a2g file format

Chapter 6

Conclusion

The author has successfully implemented a working set of classes that compute approximate coordinates by detecting polar observations and traverses and use multiple computations to check the results before storing them and using them further. The solution successfully passed all test from the gama-local test suite and the classes were released in the official GNU Gama 2.05.

The undeniable advantage of `Acord2` is its versatility. More classes to compute coordinates from a range of different observations can now be easily added, as is planned for future releases and mentioned in Chapter 6.1.

6.1 Further improvements

The author of this thesis remains a contributor to the GNU Gama project, as there is of course room for improvement on the classes presented with this thesis.

The main aim for future releases of GNU Gama is making `Acord2` a virtual base class, with the other classes being derived from this class. The derived classes that could be added in the future include algorithms for computing height differences or processing azimuths. `Acord2` would then analyse the input data and call only the algorithms that would be relevant (e.g. no point in calling an algorithm for calculating height differences if none were provided in the `ObservationList`).

Another improvement could be the ability of `AcordTraverse` to detect and compute loop traverses. At the moment the algorithm only considers adding points that are not already in the traverse.

Bibliography

[1] Aleš Čepek. *GNU Gama - Adjustment of geodetic networks*, 2.00 edition, March 2018.

[2] Aleš Čepek. GNU Gama 2.02. https://www.gnu.org/software/gama/manual/gama.html, December 2018.

[3] The Free Software Foundation. GNU coding standards. https://www.gnu.org/prep/standards/standards.html, 2018.

[4] The Free Software Foundation. What is free software? https://www.gnu.org/philosophy/free-sw.html, 2019.

[5] The Free Software Foundation. Why open source misses the point of free software. https://www.gnu.org/philosophy/open-source-misses-the-point.html, 2019.

[6] František Charamza, Aleš Čepek, and Pavel Dubišar. *Knihovna základních procedur Geolib/PC*. VÚGTK Zdiby, 1989. Programové vybavení osobních počítačů, č. dokumentace 1/1989.

[7] Google. Google C++ style guide. https://google.github.io/styleguide/cppguide.html, 2018.

[8] František Kučera. Co je to copyleft. https://svobodnysoftware.frantovo.cz/drupal/node/7, 2010.

[9] Ing. Jana Mansfeldová. Geodetické výpočty, 1.část. http://spszem.cz/storage/files/56/Geodetick-vpoty-1-25-6-13.pdf, 2008.

[10] The Linux Information Project. Copyleft definition. http://www.linfo.org/copyleft.html, 2016.

[11] Jiří Veselý. Výpočet přibližných souřadnic bodů v C++. Master's thesis, Faculty of Civil Engineering, CTU in Prague, 1999.

[12] Wikipedia. GNU General Public License. https://en.wikipedia.org/wiki/GNU_General_Public_License, 2019.

[13] Wikipedia. GNU Project. https://en.wikipedia.org/wiki/GNU_Project, 2019.

Appendix A

Where to access GNU Gama

GNU Gama can be downloaded through a mirror from `http://ftpmirror.gnu.org/gama`, with version 2.05 being the first one with the changes presented in this thesis.

The PGP signature is:

```
-----BEGIN PGP SIGNATURE-----

iF0EABECAB0WIQQpzR0yCsBdv48umCuvLWpBLF9uuAUCXNhgbwAKCRCvLWpBLF9u
uPsHAJwOvrIg76V4IYMa9IFHOO9NVL+HAgCfZyorQHHLrizx5Yod15X/8v1JsIM=
=O19L
-----END PGP SIGNATURE-----
```

Appendix B

Example input file for gama-local

```
<?xml version='1.0' ?>

<gama-local xmlns='http://www.gnu.org/software/gama/gama-local'>
<network axes-xy = 'en' angles='left-handed'>

<description>
trivial-01
</description>

<parameters  sigma-apr = '10'
             conf-pr = '0.95'
             tol-abs = '1000'
             sigma-act = 'aposteriori'
             update-constrained-coordinates = 'yes'/>

<points-observations distance-stdev='5'
                     direction-stdev='10'
                     angle-stdev='10'
                     azimuth-stdev='10'>

<point id='A' x='0'  y='0' fix='xy' />
<point id='B' x='400'  y='0' fix='xy' />
<point id='C' adj='xy' />

<obs from='A'>
   <angle bs='C' fs='B' val='100' />
   <distance  to ='C' val='300' />
</obs>

<obs from='B'>
   <angle bs='A' fs='C' val='40.9666' />
   <distance  to ='C' val='500' />
</obs>

</points-observations>
</network>
</gama-local>
```

Appendix C

Example outputs of gama-local

C.1 Example xml output

```
<?xml version="1.0"?>
<gama-local-adjustment xmlns="http://www.gnu.org/software/gama/gama-local-adjustment">

<description>
trivial-01
</description>

<network-general-parameters
   gama-local-version="2.02"
   gama-local-algorithm="envelope"
   gama-local-compiler="msvc++ 19.14"
   axes-xy="en"
   angles="left-handed"
/>

<network-processing-summary>

<coordinates-summary>
   <coordinates-summary-adjusted>    <count-xyz>0</count-xyz> <count-xy>1</count-xy> <count-z>0</count-z>
</coordinates-summary-adjusted>
   <coordinates-summary-constrained> <count-xyz>0</count-xyz> <count-xy>0</count-xy> <count-z>0</count-z>
</coordinates-summary-constrained>
   <coordinates-summary-fixed>       <count-xyz>0</count-xyz> <count-xy>2</count-xy> <count-z>0</count-z>
</coordinates-summary-fixed>
</coordinates-summary>

<observations-summary>
   <distances>2</distances>
   <directions>0</directions>
   <angles>2</angles>
   <xyz-coords>0</xyz-coords>
   <h-diffs>0</h-diffs>
   <z-angles>0</z-angles>
   <s-dists>0</s-dists>
   <vectors>0</vectors>
   <azimuths>0</azimuths>
</observations-summary>

<project-equations>
   <equations>4</equations>
   <unknowns>2</unknowns>
   <degrees-of-freedom>2</degrees-of-freedom>
   <defect>0</defect>
   <sum-of-squares>1.5951422e-01</sum-of-squares>
   <connected-network/>
```

```
</project-equations>

<standard-deviation>
   <apriori>1.0000000e+01</apriori>
   <aposteriori>2.8241301e-01</aposteriori>
   <used>aposteriori</used>

   <probability>0.950</probability>
   <ratio>0.028</ratio>
   <lower>0.159</lower>
   <upper>1.921</upper>
   <failed/>

   <confidence-scale>4.3026527e+00</confidence-scale>
</standard-deviation>

</network-processing-summary>

<coordinates>

<fixed>
   <point> <id>A</id> <x>0.000000</x> <y>0.000000</y> </point>
   <point> <id>B</id> <x>400.000000</x> <y>0.000000</y> </point>
</fixed>

<approximate>
   <point> <id>C</id> <x>0.000111</x> <y>300.000148</y> </point>
</approximate>

<!-- capital X,Y,Z denote constrained coordinates -->
<adjusted>
   <point> <id>C</id> <x>0.0000596351711768</x> <y>300.0000845145732455</y> </point>
</adjusted>

<orientation-shifts>
</orientation-shifts>

<!-- upper part of symmetric matrix band by rows -->
<cov-mat>
<dim>2</dim> <band>1</band>
<flt>1.0712147e-02</flt> <flt>-1.8883286e-03</flt> <flt>1.2645763e-02</flt>
</cov-mat>

<!-- original indexes from the adjustment -->
<original-index>
<ind>1</ind>
<ind>2</ind>
</original-index>

</coordinates>

<observations>

<angle> <from>A</from> <left>C</left> <right>B</right>
   <obs>100.0000000000000000</obs><adj>99.9999873450249197</adj> <stdev>0.2196326403108037</stdev>
   <qrr>0.395</qrr> <f>22.230</f> <std-residual>0.713</std-residual>
   <err-obs>-0.320</err-obs> <err-adj>-0.194</err-adj>
   </angle>
<distance> <from>A</from> <to>C</to>
   <obs>300.0000000000000000</obs> <adj>300.0000845145747803</adj> <stdev>0.1124533873148578</stdev>
   <qrr>0.091</qrr> <f>20.362</f> <std-residual>0.990</std-residual>
   <err-obs>0.231</err-obs> <err-adj>0.147</err-adj>
   </distance>
<angle> <from>B</from> <left>A</left> <right>C</right>
   <obs>40.9665999999999926</obs><adj>40.9665661042017675</adj> <stdev>0.1493681818632762</stdev>
   <qrr>0.720</qrr> <f>47.110</f> <std-residual>1.414</std-residual>
```

```
   <err-obs>-0.471</err-obs> <err-adj>-0.132</err-adj>
   </angle>
<distance> <from>B</from> <to>C</to>
   <obs>500.0000000000000000</obs> <adj>500.0000030006110592</adj> <stdev>0.0979563851152626</stdev>
   <qrr>0.130</qrr> <f>30.629</f> <std-residual>0.030</std-residual>
   <err-obs>0.006</err-obs> <err-adj>0.003</err-adj>
   </distance>

</observations>

</gama-local-adjustment>
```

C.2 Example Octave output

```
% gama-local : simplified adjustement results for GNU Octave
% version 1.00
%

%  Addjusted points are stored in four matrix objects
%
%  * Points   points ids
%  * Indexes  indexes of adjusted covatiances
%  * XZY      ajusted coordinates (zero if not available)
%  * Cov      covariance matrix of adjusted coordinates
%

Points = [
   'C'
];

Indexes = [
      1    2    0
];

XYZ = [
            0.000060         300.000085             0
];

Cov = [
  1.0712147e-02 -1.8883286e-03;
 -1.8883286e-03  1.2645763e-02;
];

% Fixed points are store in a cell array of the size n x 2
%
% The first column contains points ids, the second row vectors of
% coordinates [x y z], [x y] or [z]
%
% Example: [id xyz] = FixedPoints{1,:}
%

FixedPoints = {
   'A'  [         0.000000  -0.000000 ]
   'B'  [       400.000000  -0.000000 ]
};
```

C.3 Example svg output

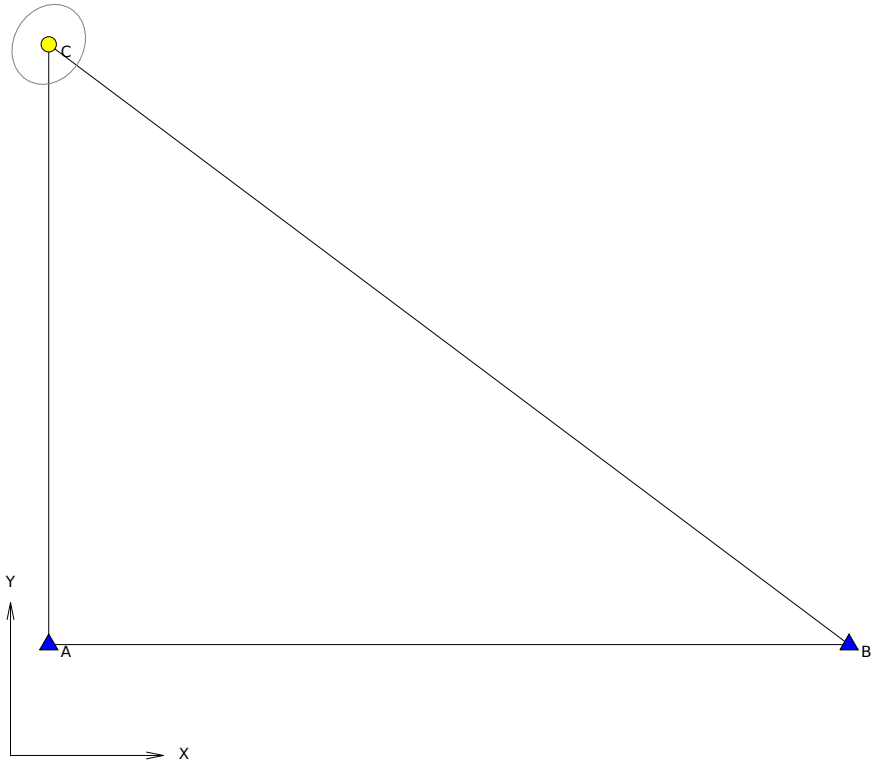


Figure C.1: Example of svg output from gama-local

C.4 Example html output

GNU Gama

Network description

trivial-01

General parameters of the adjustment

Coordinates	xyz	xy	z
Adjusted	0	1	0
Constrained *	0	0	0
Fixed	0	2	0
Total	0	3	0

Number of angles 2
Number of distances 2
Total of observations 4

Number of project equations 4 Number of unknowns 2
Degrees of freedom 2 Network defect 0

m0 apriori 10.00
m0" aposteriori 0.28 [pvv] 1.59514e-01

During statistical analysis we work
- with aposteriori standard deviation 0.28
- with confidence level 95 %

Ratio m0' aposteriori / m0 apriori: 0.028
95 % interval does not contain value m0'/m0 (0.159 , 1.921)
m0''/m0 (distances): 0.018
m0'/m0 (angles): 0.034
Confidence coeficient: 4.30265

Maximal decrease of m0''/m0 on elimination of one observation: 0.000

Maximal studentized residual 1.41 exceeds critical value 1.41
on significance level 5 % for observation #3
<angle from="B" bs="A" fs="C" val="40.9666" stdev="10.0" />

Points

Fixed points

point	x	y
A	0.000	0.000
B	400.000	0.000

Adjusted coordinates

i	point	approximate value	correction [m]	adjusted value	std.dev [mm]	conf.i. [mm]
	C					
1	x	0.00011	-0.00005	0.00006	0.1	0.4
2	y	300.00015	-0.00006	300.00008	0.1	0.5

Mean errors and parameters of error ellipses

point	mp [mm]	mxy [mm]	mean error ellipse a [mm]	b	alpha [g]	confidence a' [mm]	b'	g
C	0.2	0.1	0.1	0.1	134.9	0.7	0.6	0.1

Observations

Adjusted observations

i	standpoint	target		observed value	adjusted [m|g]	std.dev [mm|cc]	conf.i. [mm|cc]
1	A	C					
		B	angle	100.000000	99.999987	0.2	0.9
2		C	dist.	300.00000	300.00008	0.1	0.5
3	B	A					
		C	angle	40.966600	40.966566	0.1	0.6
4		C	dist.	500.00000	500.00000	0.1	0.4

Residuals and analysis of observations

i	standpoint	target		f[%]	r [mm|cc]	|r'|	e–obs [mm|cc]	e–adj
1	A	C						
		B	angle	22.2	-0.127	0.7	-0.3	-0.2
2		C	dist.	20.4	0.085	1.0	0.2	0.1
3	B	A						
		C	angle	47.1	-0.339	1.4 m	-0.5	-0.1
4		C	dist.	30.6	0.003	0.0	0.0	0.0

Figure C.2: Example of html output from gama-local

C.5 Example text output

```
Adjustment of local geodetic network        version: 2.02-envelope / msvc++ 19.14
***********************************
http://www.gnu.org/software/gama/


Approximate coordinates
***********************

coordinates          xyz        xy         z

given       :          0         2         0
computed    :          0         1         0
---------------------------------------------
total       :          0         3         0

observations:          4



Network description
*******************

trivial-01


General parameters of the adjustment
************************************

Coordinates           xyz     xy       z

Adjusted       :       0       1       0
Constrained  * :       0       0       0
Fixed          :       0       2       0
-----------------------------------------
Total          :       0       3       0

Number of angles           :     2
Number of distances        :     2
Total of observations      :     4

Number of project equations:     4            Number of unknowns:     2
Degrees of freedom         :     2            Network defect     :     0

m0  apriori    :    10.00
m0' aposteriori:     0.28         [pvv] : 1.59514e-01

During statistical analysis we work

- with aposteriori standard deviation 0.28
- with confidence level 95 %

Ratio m0' aposteriori / m0 apriori: 0.028
95 % interval (0.159, 1.921) does not contain value m0'/m0
m0'/m0 (distances): 0.018   m0'/m0 (angles): 0.034

Maximal decrease of m0''/m0 on elimination of one observation: 0.000

Maximal studentized residual 1.41 exceeds critical value 1.41
on significance level 5 % for observation #3
<angle from="B" bs="A" fs="C" val="40.9666" stdev="10.0" />


Fixed points
```

```
************

       point         x              y
=======================================

           A        0.000          0.000
           B      400.000          0.000


Adjusted coordinates
********************

  i        point    approximate  correction  adjusted    std.dev conf.i.
====================== value ====== [m] ====== value ========== [mm] ===

              C
  1           x         0.00011  -0.00005       0.00006     0.1     0.4
  2           y       300.00015  -0.00006     300.00008     0.1     0.5


Mean errors and parameters of error ellipses
********************************************

       point     mp      mxy      mean error ellipse  conf.err. ellipse   g
=============== [mm] == [mm] ==== a [mm] b  alpha[g] ==== a' [mm] b' ========

           C     0.2     0.1     0.1     0.1   134.9     0.7     0.6     0.1


Adjusted observations
*********************

   i   standpoint       target           observed     adjusted std.dev conf.i.
============================================== value ==== [m|g] ====== [mm|cc] ==

   1            A            C
                             B angle   100.000000    99.999987     0.2     0.9
   2                         C dist.    300.00000    300.00008     0.1     0.5
   3            B            A
                             C angle    40.966600    40.966566     0.1     0.6
   4                         C dist.    500.00000    500.00000     0.1     0.4


Residuals and analysis of observations
**************************************

   i   standpoint       target        f[%]        v    |v'|     e-obs.  e-adj.
============================================== [mm|cc] =========== [mm|cc] ===

   1            A            C
                             B angle  22.2      -0.127  0.7      -0.3   -0.2
   2                         C dist.  20.4       0.085  1.0       0.2    0.1
   3            B            A
                             C angle  47.1      -0.339  1.4 m    -0.5   -0.1
   4                         C dist.  30.6       0.003  0.0       0.0    0.0
```

Appendix D

Build files

D.1 The configure.ac script

```
#                                               -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.
#
AC_PREREQ(2.69)
AC_INIT([gama], [2.05], [bug-gama@gnu.org])
AC_CONFIG_SRCDIR([lib/gnu_gama/version.cpp])
AC_CONFIG_HEADER([config.h])
AC_CONFIG_AUX_DIR([scripts/config.aux])

AM_INIT_AUTOMAKE
AM_SILENT_RULES([yes])

# You can comment out the following macro if you have the package
# autoconf-archive installed
#
AC_CONFIG_MACRO_DIR([m4])
#
# Install package autoconf-archive or download from
# git://git.sv.gnu.org/autoconf-archive.git
#
# https://www.gnu.org/software/autoconf-archive/ax_cxx_compile_stdcxx_11.html
# https://www.gnu.org/software/autoconf-archive/ax_valgrind_check.html
#
AX_CXX_COMPILE_STDCXX([11], [noext], [mandatory])
AX_VALGRIND_DFLT([sgcheck], [off])
AX_VALGRIND_CHECK

# Checks for programs.
AC_PROG_CXX
AC_PROG_CC
AC_PROG_CPP
AC_PROG_LN_S
AC_PROG_MAKE_SET
AC_PROG_RANLIB

AM_CONDITIONAL([GNU_GAMA_LOCAL_TEST_SQLITE_READER], [false])
AM_CONDITIONAL([GNU_GAMA_LOCAL_TEST_EXPAT_1_1], [false])

AC_ARG_ENABLE([gama-g3],
   [AS_HELP_STRING([--enable-gama-g3[[=yes|no]]],
                   [Turn on/off build with gama-g3] (default is yes))],
   [AM_CONDITIONAL([GNU_GAMA_G3_ENABLED],
                   [AS_IF([test "x$enable_gama_g3" != "xyes"],
                          enable_gama_g3=no)]
                   [test "x$enable_gama_g3" = "xyes"])],
```

```
   [AM_CONDITIONAL([GNU_GAMA_G3_ENABLED], [true])
    enable_gama_g3=yes]
)
AC_MSG_NOTICE([checking whether gama-g3 build is enabled... $enable_gama_g3])

# Checks for libraries libexpat1-dev, libsqlite3-dev

AC_CHECK_LIB([expat], [XML_ParserCreate])
if test "x$ac_cv_lib_expat_XML_ParserCreate" != "xyes"; then
       AC_MSG_WARN([Build with local copy of expat 1.1 XML parser])
       AM_CONDITIONAL([GNU_GAMA_LOCAL_TEST_EXPAT_1_1], [true])
       AC_DEFINE([GNU_gama_expat_1_1],1,
                 [Optional support for expat 1.1 from local sources])
       CPPFLAGS="${CPPFLAGS} -DGNU_gama_expat_1_1"
else
       AC_CHECK_HEADER(expat.h, , [AC_MSG_ERROR(
                [Cannot find expat.h ... please install libexpat1-dev] )])
fi

AC_CHECK_LIB([sqlite3], [sqlite3_open])
if test "x$ac_cv_lib_sqlite3_sqlite3_open" = "xyes"; then
       AM_CONDITIONAL([GNU_GAMA_LOCAL_TEST_SQLITE_READER], [true])
       AC_DEFINE([GNU_GAMA_LOCAL_SQLITE_READER],1,
                 [Conditional support for sqlite3 databases])
       CPPFLAGS="${CPPFLAGS} -DGNU_GAMA_LOCAL_SQLITE_READER"
fi

# Check for xmllint

AC_CHECK_PROG(GNU_GAMA_LOCAL_TEST_XMLLINT, xmllint, yes, no)
AM_CONDITIONAL([GNU_GAMA_LOCAL_TEST_XMLLINT],
               [test "x$GNU_GAMA_LOCAL_TEST_XMLLINT" = "xyes"])

if test "x$GNU_GAMA_LOCAL_TEST_XMLLINT" != "xyes"; then
   AC_MSG_WARN([*** xmllint not found, XML files will not be validated])
fi

# Check for octave

AC_CHECK_PROG(GNU_GAMA_LOCAL_TEST_OCTAVE, octave, yes, no)
AM_CONDITIONAL([GNU_GAMA_LOCAL_TEST_OCTAVE],
               [test "x$GNU_GAMA_LOCAL_TEST_OCTAVE" = "xyes"])

if test "x$GNU_GAMA_LOCAL_TEST_OCTAVE" != "xyes"; then
   AC_MSG_WARN([*** octave not found, .m files will not be validated])
fi

# Checks for header files.
AC_HEADER_STDC

AC_OUTPUT([
   Makefile
   lib/Makefile
   bin/Makefile
   scripts/Makefile
   doc/Makefile
   xml/Makefile
   tests/Makefile
   tests/matvec/Makefile
   tests/statan/Makefile
   tests/acord2/Makefile
   tests/acord2/input/a2g/Makefile
   tests/acord2/input/a2diff/Makefile
   tests/gama-local/Makefile
   tests/gama-local/input/Makefile
   tests/gama-local/scripts/Makefile
```

```
    tests/gama-g3/Makefile
    tests/gama-g3/input/Makefile
    tests/gama-g3/scripts/Makefile
])
```

D.2 The CMakeLists.txt file

```
cmake_minimum_required(VERSION 3.5)
project (gnu-gama VERSION 2.05)
set(CMAKE_CXX_STANDARD 11)

add_definitions(-DGNU_gama_expat_1_1)
#add_definitions(-DDEBUG_ACORD2)
#add_definitions(-DA2G_DEBUG)

# Library
file(GLOB_RECURSE SRC_GAMA lib/gnu_gama/*.cpp lib/gnu_gama/*.h)

set(SRC_EXPAT
   lib/expat/xmltok/xmltok.c
   lib/expat/xmltok/xmlrole.c
   lib/expat/xmlwf/codepage.c
   lib/expat/xmlparse/xmlparse.c
   lib/expat/xmlparse/hashtable.c
)

include_directories(lib lib/expat/xmlparse lib/expat/xmltok)

add_library(libgama OBJECT ${SRC_GAMA} ${SRC_EXPAT})

# Binaries
add_executable(gama-local bin/gama-local.cpp $<TARGET_OBJECTS:libgama>)
add_executable(gama-g3 bin/gama-g3.cpp $<TARGET_OBJECTS:libgama>)

# Installation
install(TARGETS gama-local gama-g3 DESTINATION bin)

# Tests
enable_testing()
add_subdirectory(tests)
```

Appendix E

Source files

E.1 acord2.h

```
/*
  GNU Gama -- Approximate coordinates version 2
  Copyright (C) 2018  Petra Millarova <petramillarova@gmail.com>

  This file is part of the GNU Gama C++ library.

  GNU Gama is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GNU Gama is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GNU Gama.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef GAMA_LOCAL_ACORD2_H_
#define GAMA_LOCAL_ACORD2_H_

#include <gnu_gama/local/gamadata.h>
#include <cstddef>
#include <vector>
#include <set>
#include <map>
#include <utility>

namespace GNU_gama { namespace local {

    class Acord2
    {
    public:
      Acord2(PointData&, ObservationData&);
      void execute();

    private:
      using size_type = std::size_t;

      friend class AcordPolar;
      friend class AcordTraverse;
      friend class AcordWeakChecks;
```

```
      struct Point
      {
        PointID id;
        LocalPoint coords;

        Point(PointID pt_id_, LocalPoint pt_xy_)
          : id(pt_id_), coords(pt_xy_)
        {}
        Point() :id(PointID()), coords(LocalPoint())
        {}
        bool operator< (const Point &p) { return (id <  p.id); }
        bool operator==(const Point &p) { return (id == p.id); }
      };

      StandPoint* find_standpoint(PointID pt);
      double median(std::vector<double>);
      std::pair<double,bool> get_dist(Observation* o);
      std::pair<double, bool> get_dir(Observation* o);
      bool in_missingXY(PointID pt);

      double MAX_NORM = 0.1;
      bool slope_observations_ = false;

      PointData&       PD_;
      ObservationData& OD_;
      std::vector<StandPoint*> SPClusters_;
      std::set<PointID> missingXY_;
      std::set<PointID> missingZ_;
      std::multimap<PointID,LocalPoint> same_points_;
      std::multimap<PointID,Observation*> obs_from_;
      std::multimap<PointID,Observation*> obs_to_;
      bool get_medians();
      size_type new_points_;

      using Traverse = std::vector<Point>;
      enum Traverse_type
        {
          open_traverse = 0,
          closed_start_traverse = 1,
          closed_traverse = 2
        };
      bool transform_traverse(Acord2::Traverse& traverse);

      std::vector<std::pair<Traverse, Traverse_type>> traverses;
    };

  }} //namespace GNU_gama::local
#endif
```

E.2 acord2.cpp

```
/*
  GNU Gama -- Approximate coordinates version 2
  Copyright (C) 2018  Petra Millarova <petramillarova@gmail.com>

  This file is part of the GNU Gama C++ library.

  GNU Gama is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GNU Gama is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GNU Gama.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <gnu_gama/local/acord/acord.h>
#include <gnu_gama/local/acord/acord2.h>
#include <gnu_gama/local/acord/acordpolar.h>
#include <gnu_gama/local/acord/acordtraverse.h>
#include <gnu_gama/local/acord/acordweakchecks.h>
#include <gnu_gama/local/acord/acordstatistics.h>
#include <gnu_gama/local/bearing.h>
#include <gnu_gama/local/orientation.h>
#include <gnu_gama/local/median/g2d_helper.h>
#include <gnu_gama/local/gamadata.h>
#include <cmath>

using namespace GNU_gama::local;

#ifndef DEBUG_ACORD2
#define DEBUG_ACORD2
#endif

#ifdef DEBUG_ACORD2

#include <iostream>
#include <algorithm>

using Pair = std::pair<PointID, LocalPoint>;
using std::count_if;
using std::cerr;
using std::endl;

#define DBG_algo(text)(cerr << "" << text << " "\
<< count_if(PD_.begin(),PD_.end(),[](Pair p){return p.second. test_xy();})\
<< " / " << same_points_.size() << " / "  \
<< count_if(PD_.begin(),PD_.end(),[](Pair p){return !p.second. test_xy();}))\
<< "   ";

#define DBG_prnt(text)(cerr << text);

#else
#define DBG_algo(text)
#define DBG_prnt(text)
#endif


Acord2::Acord2(PointData& pd, ObservationData& od)
```

```
  : PD_(pd), OD_(od)
{
  for (auto i : OD_.clusters)
    {
      StandPoint* sp = dynamic_cast<StandPoint*>(i);
      if (sp != nullptr)
        {
          SPClusters_.push_back(sp);
        }
    }

  for (PointData::const_iterator i = PD_.begin(); i != PD_.end(); ++i)
    {
      const PointID& c = (*i).first;
      const LocalPoint& p = (*i).second;
      if (p.active_xy() && !p.test_xy())
        { // this means the a network point xy were not set and are
          // part of the network
          // find and add all missing points into one set
          missingXY_.insert(c);
        }
      if (p.active_z() && !p.test_z())
        {
          missingZ_.insert(c);
        }
    }

  for (auto c : OD_.clusters)
    {
      StandPoint* sp = dynamic_cast<StandPoint*>(c);
      if (sp != nullptr)
        {
          for (auto obs : sp->observation_list)
            {
              PointID from = obs->from();
              obs_from_.insert({obs->from(), obs});
              if (Angle* angle = dynamic_cast<Angle*>(obs))
                {
                  obs_to_.insert({angle->bs(), obs});
                  obs_to_.insert({angle->fs(), obs});
                }
              else
                {
                  obs_to_.insert({obs->to(), obs});
                }
            }
        }
    }
}


StandPoint* Acord2::find_standpoint(PointID pt)
{
  for (auto sp : SPClusters_)
    {
      if (sp == nullptr) continue;
      if (sp->station == pt) return sp;
    }
  return nullptr;
}


void Acord2::execute()
{
  DBG_prnt("Acord2::execute xy known/same/missing")
  DBG_algo("")
```

```
  DBG_prnt("\n")
  size_type after {},  before = missingXY_.size();
  if (before == 0) return;

  do  // while some points need to be solved
    {
      before = missingXY_.size();

      AcordPolar polar(this);

      polar.execute();
      DBG_algo("polar")


      polar.enable_slope_observations();
      polar.execute();
      DBG_algo("polar slope")

      AcordTraverse traverse(this);
      traverse.execute();
      DBG_algo("traverse")

      AcordWeakChecks weakchecks(this);
      weakchecks.execute();
      DBG_algo("weak checks")

      after = missingXY_.size();
  get_medians();
      same_points_.clear();
  traverses.clear();
    }
  while (after != 0 && after < before);

  DBG_prnt("\n\n")
}


// median: calculates median for a vector of doubles

double Acord2::median(std::vector<double> v)
{
  std::sort(v.begin(), v.end());
  double res = 0;
  if (v.size() % 2 == 0) //size is even
    {
      size_type i = v.size() / 2;
      size_type j = i - 1;
      res = (v[i] + v[j]) / 2;
    }
  else //size is odd
    {
      res = v[v.size()/2];
    }
  return res;
}

// dist_or_dir: function takes an observation and nullptr distance and
// direction and decides if observation is/can be converted to
// Distance or Direction type and places it in the appropriate
// variable

std::pair<double, bool> Acord2::get_dir(Observation* o)
{
  if (Azimuth* a = dynamic_cast<Azimuth*>(o))
    {
      return {a->value() + PD_.xNorthAngle(), true};
```

```
    }
  /*if (Angle* a = dynamic_cast<Angle*>(o))
    {
    auto d = angle_to_dir(a);
    if (d.second)
    {
    return {d.first, true};
    }
    }*/
  if (Direction* d = dynamic_cast<Direction*>(o))
    {
      return {d->value(), true};
    }
  return {0, false};
}


std::pair<double,bool> Acord2::get_dist(Observation* o)
{
  if (Distance* d = dynamic_cast<Distance*>(o))
    {
      return {d->value(),true};
    }
  if (slope_observations_)
    {
      if (S_Distance* sd = dynamic_cast<S_Distance*>(o))
        {
          //search for zenith angle so we can calculate horizontal distance
          auto i = obs_from_.lower_bound(o->from());
          auto ei = obs_from_.upper_bound(o->from());
          while (i != ei)
            {
              if (Z_Angle* za = dynamic_cast<Z_Angle*>(i->second))
                {
                  if (o->to() == za->to())
                    {
                      double distval = sd->value()*std::fabs(sin(za->value()));
                      return {distval,true};
                    }
                }
              ++i;
            }
        }
    }
  return {0,false};
}

// in_missingXY: returns true if point is in the missingXY set

bool Acord2::in_missingXY(PointID pt)
{
  std::set<PointID>::iterator it_stpt = missingXY_.find(pt);
  if (it_stpt == missingXY_.end()) return false;
  else return true;
}

// get_medians: goes through same_points and if there are two or more
// entries for one PointID, decides if it can add the point to PD and
// calculates new coords

bool Acord2::get_medians()
{
  bool res = false;
  // get unique keys from same_points_
  std::set<PointID> keys;
  for (auto i : same_points_) keys.insert(i.first);
```

```
// for (std::vector<Acord2::Point>::iterator j, i =
// same_points_.begin(), e = same_points_.end(); i != e; )
for (auto pt : keys)
  {
    auto p = same_points_.equal_range(pt);
    if (p.first == p.second) continue;    // should never happen

    auto i = p.first;
    auto j = p.second;

    // i and j now mark the beginning and end of an interval with
    // the "same" points
    std::vector<double> all_x;
    std::vector<double> all_y;

    double min_x = i->second.x();
    double max_x = i->second.x();
    double min_y = i->second.y();
    double max_y = i->second.y();

    while (i != j)
      {
        double x = i->second.x();
        double y = i->second.y();
        all_x.push_back(x);
        all_y.push_back(y);
        if (min_x > x) min_x = x;
        if (max_x < x) max_x = x;
        if (min_y > y) min_y = y;
        if (max_y < y) max_y = y;
        ++i;
      }

    if ((all_x.size()>1) && in_missingXY(pt))
      {
        //norm check
        double max_dx = std::fabs(max_x - min_x);
        double max_dy = std::fabs(max_y - min_y);
        double max_diff = std::max(max_dy, max_dx);

        if (slope_observations_) MAX_NORM = 0.5;

        if (max_diff <= MAX_NORM)
          {
            double med_x = median(all_x);
            double med_y = median(all_y);
            // only setting the coords, leaving other info as is
            PD_[pt].set_xy(med_x, med_y);
            missingXY_.erase(pt);
            new_points_++;
res = true;
          }
      }
  }

// same_points cleanup - we can delete already computed points
for (auto p : keys)
  {
    if (!in_missingXY(p))
      {
        same_points_.erase(p);
      }
  }
return res;
```

```
};

// takes a traverse and transforms its coordinates

bool Acord2::transform_traverse(Traverse& traverse)
{
  //now we must somehow transform the coords from local
  PointData local_traverse;
  PointIDList unknown_traverse_pts;
  for (auto p : traverse)
    {
      local_traverse[p.id] = p.coords;
      if (in_missingXY(p.id)) unknown_traverse_pts.push_back(p.id);
    }
  if (unknown_traverse_pts.size() == 0) return true;
  SimilarityTr2D transformation(PD_, local_traverse, unknown_traverse_pts);
  transformation.calculation();
  if (transformation.state() < unique_solution)
    {
      // Transformation failed: No unique solution.
    }
  PointData res = transformation.transf_points();
  if (!res.empty())
    {
  for (auto &pt : traverse)
{
if (res[pt.id].test_xy ())
  {
pt.coords = res[pt.id];
  }

}
      return true;
    }
  else
    {
      return false;
    }
}
```

E.3 acordpolar.h

```
/*
  GNU Gama -- Approximate coordinates by Polar Method
  Copyright (C) 2018  Petra Millarova <petramillarova@gmail.com>

  This file is part of the GNU Gama C++ library.

  GNU Gama is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GNU Gama is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GNU Gama.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef GAMA_LOCAL_ACORDPOLAR_H
#define GAMA_LOCAL_ACORDPOLAR_H

#include <gnu_gama/local/gamadata.h>
#include <gnu_gama/local/acord/reduce_observations.h>
#include <gnu_gama/local/acord/reduce_to_ellipsoid.h>
#include <gnu_gama/local/acord/acord2.h>


namespace GNU_gama { namespace local
  {
    class AcordPolar
    {
    public:
      AcordPolar(Acord2* acord2);
      void execute();

      void enable_slope_observations()  { AC.slope_observations_ = true; }
      void enable_weak_checks()         { weak_checks_ = true; }
      void disable_slope_observations() { AC.slope_observations_ = false; }
      void disable_weak_checks()        { weak_checks_ = false; }

    private:
      Acord2& AC;
      PointData & PD;
      ObservationData& OD;

      bool weak_checks_ = false;

      struct Measurement
      {
        PointID standpoint;
        PointID ori;
        double dir;
        double dist;
        StandPoint* sp; //for orientation shift

        Measurement(PointID standpoint_val, PointID ori_val,
                    double dir_val, double dist_val, StandPoint* sp_val)
          : standpoint(standpoint_val), ori(ori_val),
            dir(dir_val), dist(dist_val), sp(sp_val)
        {
        }
```

```
    };

    bool points_from_SPCluster(StandPoint* sp);
    LocalPoint calculate_polar(Measurement m);
    bool stub(PointID p);
  };

}} //namespace GNU_gama::local
#endif
```

E.4 acordpolar.cpp

```
/*
  GNU Gama -- Approximate coordinates by Polar Method
  Copyright (C) 2018  Petra Millarova <petramillarova@gmail.com>

  This file is part of the GNU Gama C++ library.

  GNU Gama is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GNU Gama is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GNU Gama.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <numeric>
#include <algorithm>
#include <gnu_gama/local/acord/acordpolar.h>
#include <gnu_gama/local/orientation.h>
#include <gnu_gama/local/median/g2d_point.h>

using namespace GNU_gama::local;

AcordPolar::AcordPolar(Acord2* acord2)
  : AC(*acord2), PD(acord2->PD_), OD(acord2->OD_)
{
  try
    {
    }
  catch (...)
    {
      throw;
    }
}

void AcordPolar::execute()
{
  if (AC.missingXY_.size() == 0) return;

  Acord2::size_type before_cnt {}, after_cnt {};
  Acord2::size_type N = AC.SPClusters_.size();
  do
    {
      AC.new_points_ = 0;
      before_cnt = AC.same_points_.size();

      for (Acord2::size_type i = 0; i < N ; i++)
        {
          if (AC.SPClusters_[i] != nullptr)
            {
              if (!points_from_SPCluster(AC.SPClusters_[i]))
                {
                  // AC.SPClusters_[i] = nullptr;
                  --N;
                  std::swap(AC.SPClusters_[i], AC.SPClusters_[N]);
                  --i;
                }
            }
```

```
        }
      after_cnt = AC.same_points_.size();
      AC.get_medians();
    }
  while (N > 0 && (AC.new_points_ > 0) && AC.missingXY_.size() > 0);
}

// points_from_SPCluster: goes through a StandPoint cluster and if it
// finds enough information, calls the calculate_polar function

bool AcordPolar::points_from_SPCluster(StandPoint* sp)
{
  bool res = true;
  if (AC.in_missingXY(sp->station)) return res;

  // local observation list
  ObservationList lol = sp->observation_list;   // deep copy

  // orientation shift can be defined explicitly in the input XML
  // if this is the case it has the precedence over computed values
  if (!sp->test_orientation())
    {
      Orientation ori(PD, lol);
      double orientation_shift;
      int n;
      ori.orientation(sp, orientation_shift, n);
      if (n > 0) sp->set_orientation(orientation_shift);
    }

  std::map<PointID, double> sp_distances;   // <to where, median value>
  std::vector<Angle*> sp_angles;
  std::map<PointID, double> sp_directions;

  lol.sort([](Observation* a, Observation* b) { return a->to() < b->to(); });
  for (PointID pid : AC.missingXY_)
    {
      std::vector<double> tmp_dists;
      bool skip = true;
      for (Observation* obs : lol)
        {
          if (Angle* a = dynamic_cast<Angle*>(obs))
            {
              sp_angles.push_back(a);
              continue;
            }

          if (obs->to() != pid)
            {
              if (skip) continue;    // skip leading obs not targeting to pid
              else      break;       // ignore trailing obs ...
            }

          skip = false;

          auto dir = AC.get_dir(obs);
          if (dir.second)
            {
              sp_directions.insert({ pid, dir.first });
            }
          else
            {
              auto dist = AC.get_dist(obs);
              if (dist.second)
                {
                  tmp_dists.push_back(dist.first);
                }
```

```
          }
      }
    if (!tmp_dists.empty())
      {
        sp_distances.insert({ pid, AC.median(tmp_dists) });
      }

  }

// go through directions and compute
if (sp->test_orientation())
  {
    for (auto o : sp_directions)
      {
        auto dist_it = sp_distances.find(o.first);
        if (dist_it != sp_distances.end())
          {
            Measurement m(sp->station, o.first, o.second,
                          dist_it->second, sp);
            LocalPoint calc_pt = calculate_polar(m);
            if (stub(o.first))
              {
                 //only setting the coords, leaving other info as is
                PD[o.first].set_xy(calc_pt.x(), calc_pt.y());
                AC.missingXY_.erase(o.first);
                AC.new_points_++;
                res = false;
              }
            else
              {
                AC.same_points_.insert({ o.first, calc_pt });
                res = false;
              }
          }
      }
  }

// add coordinates computed from angle observation
if (sp_angles.size() == 0) return res;

std::map<PointID, LocalPoint> secondary_points;
Acord2::size_type N = sp_angles.size();

Acord2::size_type N_before {};
do
  {
    N_before = N;
    for (Acord2::size_type i=0; i<N; i++)
      {
        Angle* a = sp_angles[i];
        if (a == nullptr) continue;

        PointID bsid = a->bs();
        PointID fsid = a->fs();
        LocalPoint bs_local_point, fs_local_point;
        int known_xy = 0;   // bitmap 00, 10, 01, 11

        { // find xy coordinates for bs ray if exists
          auto bsiter = PD.find(bsid);
          if (bsiter != PD.end() && bsiter->second.test_xy())
            {
              known_xy += 1; // set 1st bit
              bs_local_point = bsiter->second;
            }
          else
            {
```

```
      auto bsiter = secondary_points.find(bsid);
      if (bsiter != secondary_points.end() &&
          bsiter->second.test_xy())
        {
          known_xy += 1; // set 1st bit
          bs_local_point = bsiter->second;
        }
    }
}

{ // find xy coordinates for fs ray if exists
  auto fsiter = PD.find(fsid);
  if (fsiter != PD.end() && fsiter->second.test_xy())
    {
      known_xy += 2; // set 2nd bit
      fs_local_point = fsiter->second;
    }
  else
    {
      auto fsiter = secondary_points.find(fsid);
      if (fsiter != secondary_points.end() &&
          fsiter->second.test_xy())
        {
          known_xy += 2; // set 2nd bit
          fs_local_point = fsiter->second;
        }
    }
}

if (known_xy != 1 && known_xy != 2) continue;

// now we know that for the given angle there is exactly one
// ray with known xy
LocalPoint target;
LocalPoint stand_point = PD[a->from()];
if (bs_local_point.test_xy())
  {
    double b = bearing(stand_point,  bs_local_point);
    b += a->value();
    //double d = sp_distances[fsid];
    auto dist_it = sp_distances.find(fsid);
    if (dist_it != sp_distances.end())
      {
        double d = dist_it->second;
        double x = stand_point.x() + d * std::cos(b);
        double y = stand_point.y() + d * std::sin(b);
        target.set_xy(x, y);
        AC.same_points_.insert({ fsid,target });
        secondary_points.insert({ fsid,target });
      }

  }
if (fs_local_point.test_xy())
  {
    double b = bearing(stand_point,  fs_local_point);
    b -= a->value();
    //double d = sp_distances[bsid];
    auto dist_it = sp_distances.find(bsid);
    if (dist_it != sp_distances.end())
      {
        double d = dist_it->second;
        double x = stand_point.x() + d * std::cos(b);
        double y = stand_point.y() + d * std::sin(b);
        target.set_xy(x, y);
        AC.same_points_.insert({ bsid,target });
        secondary_points.insert({ bsid,target });
```

```
                }

            }

           // remove processed angle from the list
           sp_angles[i] = nullptr;
         }
      }
    while(N > 0 && N_before > N);

  return res;
}

bool AcordPolar::stub(PointID p) //returns true if point is a stub
{
  auto it = AC.obs_to_.lower_bound(p);
  auto eit = AC.obs_to_.upper_bound(p);
  PointID comp_pt = it->second->from();
  while (it != eit)
    {
      if (comp_pt != it->second->from()) return false;
      ++it;
    }
  return true;
}

// calculate_polar: calculates coords from measurement

LocalPoint AcordPolar::calculate_polar(Measurement m)
{
  const LocalPoint& sp_xy = PD[m.standpoint];
  double bearing = m.sp->orientation() + (m.dir);
  double x = sp_xy.x() + (m.dist*std::cos(bearing));
  double y = sp_xy.y() + (m.dist*std::sin(bearing));
  return LocalPoint(x, y);
}
```

E.5 acordtraverse.h

```
/*
  GNU Gama -- Approximate coordinates by Polar Method
  Copyright (C) 2018  Petra Millarova <petramillarova@gmail.com>

  This file is part of the GNU Gama C++ library.

  GNU Gama is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GNU Gama is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GNU Gama.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef GAMA_LOCAL_ACORDTRAVERSE_H
#define GAMA_LOCAL_ACORDTRAVERSE_H

#include <set>

#include <gnu_gama/local/acord/acord2.h>


namespace GNU_gama { namespace local {

    class AcordTraverse
    {
    public:
      AcordTraverse(Acord2* acord2);
      void execute();

    private:
      friend class AcordWeakChecks;

      Acord2& AC;
      PointData & PD;
      ObservationData& OD;

      Acord2::Traverse_type tr_type = AC.open_traverse;
      Acord2::Traverse traverse;

      std::set<PointID>    candidate_traverse_points_;
      std::vector<PointID> traverse_points_;

      std::set<PointID> get_neighbours(PointID pt);
      bool  candidateTraversePoint(PointID);
      void get_traverse_pts(PointID pt);
      Acord2::Traverse_type get_connecting_points();

      std::set<PointID> add_same_points();

      bool calculate_traverse();
    };

  }} //namespace GNU_gama::local
#endif
```

E.6 acordtraverse.cpp

```
/*
  GNU Gama -- Approximate coordinates by Polar Method
  Copyright (C) 2018  Petra Millarova <petramillarova@gmail.com>

  This file is part of the GNU Gama C++ library.

  GNU Gama is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GNU Gama is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GNU Gama.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <map>
#include <gnu_gama/local/acord/acordtraverse.h>
#include <gnu_gama/local/orientation.h>


using namespace GNU_gama::local;

AcordTraverse::AcordTraverse(Acord2* acord2)
  : AC(*acord2), PD(acord2->PD_), OD(acord2->OD_)
{
  try
    {
      for (PointID p : AC.missingXY_)
        // if (!PD[p].test_xy()) ... AC.missingXY_ cannot have xy
        {
          std::set<PointID> t = get_neighbours(p);
          if (t.size() == 2) candidate_traverse_points_.insert(p);
        }
    }
  catch (...)
    {
      throw;
    }
}


void AcordTraverse::execute() //to keep in line with the acord class
{
  std::set<PointID>::iterator it = candidate_traverse_points_.begin();
  while (candidate_traverse_points_.size() > 1 && it != candidate_traverse_points_.end())
    {
      // If there is one then add pt to traverse_pts and remove from candidates,
      traverse_points_.push_back(*it);
      get_traverse_pts(*it);   // populating the traverse_points_

      if (traverse_points_.size() > 1)
        {
          tr_type = get_connecting_points();
          //if there are at least two points in traverse_pts then I can calculate the polygon and insert into traverses
          bool success = calculate_traverse();
          if (success)
            {
              AC.traverses.push_back({ traverse, tr_type });
```

```
          }
        else
          {
            // Computation failed
          }

        for (auto t : traverse_points_)
          {
            candidate_traverse_points_.erase(t);
          }
        it = candidate_traverse_points_.begin();
        traverse_points_.clear();
        traverse.clear();
      }
    else
      {
        for (auto t : traverse_points_)
          {
            candidate_traverse_points_.insert(t);
          }
        traverse_points_.clear();
        traverse.clear();
        ++it;
      }

  }

// now we have all traverses we can try to find same points
std::set<PointID> added_points = add_same_points();
if (!added_points.empty())
  {
    auto cnt_before = AC.missingXY_.size();
    AC.get_medians();
    auto cnt_after = AC.missingXY_.size();
    if (cnt_before != cnt_after)
      {
        /* check which traverses contain the points that are now known and edit
         * them and their traverse type
         */
        for (auto tr : AC.traverses)
          {
            // if it's a closed traverse we already know coords of both points and we can add them
            if (tr.second == AC.closed_traverse) continue;
            PointID tr_pointid = tr.first.back().id;

            // if the point is in missingXY than we know it was not added
            if (AC.in_missingXY(tr_pointid)) continue;
            auto found_it = std::find(added_points.begin(), added_points.end(), tr_pointid);
            if (found_it != added_points.end())
              {
                tr.first.back().coords = PD[tr_pointid]; //set the newly computed coords
                tr.second = AC.closed_traverse;

                // now we can add computed points to PD and remove them from missingXY
                if (AC.transform_traverse(tr.first))
                  {
                    for (auto pt : tr.first)
                      {
                        if (AC.in_missingXY(pt.id))
                          {
                            // only setting the coords, leaving other info as is
                            PD[pt.id].set_xy(pt.coords.x(), pt.coords.y());
                            AC.missingXY_.erase(pt.id);
                            AC.new_points_++;
                          }
                      }
```

```
                    }
                }
            }
        }
    }

  // now we just iterate through the traverses one more time and if the
  // last point is known, we can safely remove them from the list

  AC.traverses.erase(std::remove_if(AC.traverses.begin(), AC.traverses.end(), [this](std::pair<Acord2::Traverse, Acord2

}

// find same points in traverses and add them to AC.same_points,
// returns true if points are added

std::set<PointID> AcordTraverse::add_same_points()
{
  std::set<PointID> added_points;
  for (auto itr = AC.traverses.begin(); itr!= AC.traverses.end(); ++itr )
    {
//first check if any points have been computed before
bool found = false;
  for (auto pt : (*itr).first)
{
  auto fit = AC.same_points_.find (pt.id);
  auto ait = added_points.find (pt.id);

  if (fit != AC.same_points_.end() && ait == added_points.end())
{
  AC.same_points_.insert({pt.id, pt.coords});
  found = true;
}
}
  if (found && AC.get_medians())
{
  AC.transform_traverse ((*itr).first);
}
      //we know both endpoints are known - we can add the whole traverse
      //straight to PD
      if ((*itr).second == AC.closed_traverse)
        {
          for (auto pt : (*itr).first)
            {
              if (AC.in_missingXY(pt.id))
                {
                  PD[pt.id].set_xy(pt.coords.x(), pt.coords.y());
                  AC.missingXY_.erase(pt.id);
                  AC.new_points_++;
                }
            }
        }
      /* tr_type should now only be closed_start_traverse
       * this is the only unknown point that can occur in more traverses
       */
      Acord2::Point comp_point = (*itr).first.back();
      bool found_already = false;
      if (AC.traverses.size() < 2 || itr == AC.traverses.end() - 1) continue;
      for (auto initr = ++itr; initr != AC.traverses.end(); ++initr)
        {
          if ((*itr).second == AC.closed_traverse) continue;
          if (comp_point == (*initr).first.back())
            {
              if (!found_already)
                {
                  AC.same_points_.insert({ comp_point.id, comp_point.coords });
```

```
                found_already = true;
                added_points.insert(comp_point.id);
              }
            AC.same_points_.insert({ (*initr).first.back().id, (*initr).first.back().coords });
          }
      }

  }
 return added_points;
}

// finds all points that are connected and their coords are unknown
// and adds them to traverse_points in the right order

void AcordTraverse::get_traverse_pts(PointID pt)
{
  std::set<PointID> neighbours = get_neighbours(pt);
  for (auto c : neighbours)
    {
      auto found_traverse_point  = std::find(traverse_points_.begin(), traverse_points_.end(), c);
      auto found_candidate_point = std::find(candidate_traverse_points_.begin(), candidate_traverse_points_.end(), c);

      // if the point is not added in traverse points yet
      if (found_traverse_point == traverse_points_.end())
        {
          // if the point is a candidate traverse point then add it and delete from candidates
          if (found_candidate_point != candidate_traverse_points_.end())
            {
              traverse_points_.push_back(c);
              candidate_traverse_points_.erase(c);
              get_traverse_pts(c);
            }
        }
    }
}

///looks for known points at start or end of traverse, sets type of traverse and add the point to  the right place
Acord2::Traverse_type AcordTraverse::get_connecting_points()
{
  tr_type = AC.open_traverse;
  //find all neighbours of this point and check if they can be added
  std::set<PointID> last_neighbours = get_neighbours(traverse_points_.back());
  bool end_pt = false;
  for (auto pid : last_neighbours)
    {
      auto found_tr = std::find(traverse_points_.begin(), traverse_points_.end(), pid);
      if (found_tr == traverse_points_.end()) //point not yet in traverse
        {
          traverse_points_.push_back(pid);
          if (!AC.in_missingXY(pid)) end_pt = true;
          break;
        }
    }
  //find all neighbours of this point and check if they can be added
  std::set<PointID> first_neighbours = get_neighbours(traverse_points_.front());
  for (auto pid : first_neighbours)
    {
      auto found_tr = std::find(traverse_points_.begin(), traverse_points_.end(), pid);
      if (found_tr == traverse_points_.end()) //point not yet in traverse
        {
          traverse_points_.insert(traverse_points_.begin(), pid);
          if (!AC.in_missingXY(pid))
            {
              if (end_pt) tr_type = AC.closed_traverse;
              else tr_type = AC.closed_start_traverse;
            }
```

```
          else //this means the traverse only has a known point at the end
            {
              //would it be better to use reverse iterators later instead?
              std::reverse(traverse_points_.begin(), traverse_points_.end());
              tr_type = AC.closed_start_traverse;
            }
          break;
        }
    }
  return tr_type;
}

/// returns a set of neighbours
std::set<PointID> AcordTraverse::get_neighbours(PointID pt)
{
  std::set<PointID> s;
  auto it = AC.obs_from_.lower_bound(pt);
  auto eit = AC.obs_from_.upper_bound(pt);

  while (it != eit)
    {
      //if (PD[it->second->to()].test_xy()) break;
      s.insert(it->second->to());
      ++it;
    }

  it = AC.obs_to_.lower_bound(pt);
  eit = AC.obs_to_.upper_bound(pt);
  while (it != eit)
    {
      //if (PD[it->second->from()].test_xy()) break;
      s.insert(it->second->from());
      ++it;
    }

  return  s;
}

bool AcordTraverse::candidateTraversePoint(PointID pid)
{
  if (PD[pid].test_xy()) return false;

  return  get_neighbours(pid).size() == 2;
}

///calculates traverse coordinates if traverse is not open and if finds front orientation
bool AcordTraverse::calculate_traverse()
{
  if (tr_type == AC.open_traverse) return false;

  std::pair<double, bool> front_ori = {0, false };
  std::vector<double> angles;
  std::vector<double> distances;

  for (Acord2::size_type i = 0; i<traverse_points_.size(); i++ )
    {
      PointID pid = traverse_points_[i];
      std::vector<double> tmp_dists;
      std::vector<double> tmp_bearings;
      int known_dirs = 0; //11 - both, 10 - front, 01 - back, 00 - none
      double angle_from_dirs = 0;

      //check for distances, directons and angles from this point
      auto it = AC.obs_from_.lower_bound(pid);
      auto eit = AC.obs_from_.upper_bound(pid);
      while (it != eit && i != traverse_points_.size()-1)
```

```
  {
    Angle* a = dynamic_cast<Angle*>(it->second);
    if (i != 0 && a != nullptr)
      {
        if (a->bs() == traverse_points_[i - 1] && a->fs() == traverse_points_[i + 1]) tmp_bearings.push_back(a->v
        if (a->fs() == traverse_points_[i - 1] && a->bs() == traverse_points_[i + 1]) tmp_bearings.push_back(2 *
      }
    auto dir = AC.get_dir(it->second);
    if (dir.second)
      {
        if (it->second->to() == traverse_points_[i + 1] && (known_dirs == 00 || known_dirs == 01))
          {
            known_dirs += 10;
            angle_from_dirs += dir.first;
          }
        else if ((known_dirs == 00 || known_dirs == 10))
          {
            if (i == 0 && !AC.in_missingXY(it->second->to()) && !front_ori.second)
              {
                StandPoint* sp = AC.find_standpoint(it->second->from());
                if (sp != nullptr)
                  {
                    if (!sp->test_orientation())
                      {
                        Orientation ori(PD, sp->observation_list);
                        double orientation_shift;
                        int n;
                        ori.orientation(sp, orientation_shift, n);
                        if (n > 0) sp->set_orientation(orientation_shift);
                      }
                    if (sp->test_orientation())
                      {
                        traverse.push_back({ it->second->to(),PD[it->second->to()] });
                        known_dirs += 1;
                        angle_from_dirs -= (dir.first + sp->orientation());
                        front_ori = { dir.first + sp->orientation(), true };
                      }
                  }
              }
            else if (i!=0)
              {
                if (it->second->to() == traverse_points_[i - 1])
                  {
                    known_dirs += 1;
                    angle_from_dirs -= dir.first;
                  }
              }
          }
      }

    auto d = AC.get_dist(it->second);
    if (d.second && it->second->to() == traverse_points_[i + 1]) tmp_dists.push_back(d.first);

    if (known_dirs == 11)
      {
        while (angle_from_dirs > 2 * M_PI)
          angle_from_dirs -= 2 * M_PI;
        while (angle_from_dirs < 0)
          angle_from_dirs += 2 * M_PI;
        tmp_bearings.push_back(angle_from_dirs);
        angle_from_dirs = 0;
        known_dirs = 0;
      }
    ++it;
  }
//check for directions and distances TO this point
```

```
auto itt = AC.obs_to_.lower_bound(pid);
auto eitt = AC.obs_to_.upper_bound(pid);
while (itt != eitt && i != traverse_points_.size() - 1)
  {
    auto d = AC.get_dist(itt->second);
    if (d.second && itt->second->from() == traverse_points_[i + 1]) tmp_dists.push_back(d.first);

    auto dir = AC.get_dir(itt->second);
    if (dir.second)
      {
        if (itt->second->from() == traverse_points_[i + 1] && (known_dirs == 00 || known_dirs == 01))
          {
            known_dirs += 10;
            double dirval = M_PI - dir.first;
            angle_from_dirs -= dirval;
          }
        else if ((known_dirs == 00 || known_dirs == 10))
          {
            if (i == 0 && !AC.in_missingXY(itt->second->to()) && !front_ori.second)
              {
                StandPoint* sp = AC.find_standpoint(itt->second->from());
                if (sp != nullptr)
                  {
                    if (!sp->test_orientation())
                      {
                        Orientation ori(PD, sp->observation_list);
                        double orientation_shift;
                        int n;
                        ori.orientation(sp, orientation_shift, n);
                        if (n > 0) sp->set_orientation(orientation_shift);
                      }
                    if (sp->test_orientation())
                      {
                        traverse.push_back({ itt->second->to(),PD[itt->second->to()] });
                        known_dirs += 1;
                        double dirval = M_PI - dir.first;
                        angle_from_dirs += (dirval + sp->orientation());
                        front_ori = { dir.first + sp->orientation(), true };
                      }
                  }
              }
            else if (i != 0)
              {
                if (itt->second->from() == traverse_points_[i - 1])
                  {
                    known_dirs += 1;
                    double dirval = M_PI - dir.first;
                    angle_from_dirs += dirval;
                  }
              }
          }
      }

    if (known_dirs == 11)
      {
        while (angle_from_dirs > 2 * M_PI)
          angle_from_dirs -= 2 * M_PI;
        while (angle_from_dirs < 0)
          angle_from_dirs += 2 * M_PI;
        tmp_bearings.push_back(angle_from_dirs);
        angle_from_dirs = 0;
        known_dirs = 0;
      }
    ++itt;
  }
if (!tmp_dists.empty() &&!tmp_bearings.empty())
```

```
      {
        distances.push_back(AC.median(tmp_dists));
        angles.push_back( AC.median(tmp_bearings));
      }
    else if(i<traverse_points_.size()-1)
      {
        //if the point is in the first place and traverse
        //type is closed, we can try to switch the order and
        //start again
        if ((i == 0) && tr_type == AC.closed_traverse)
          {
            front_ori = { 0, false };
            tr_type = AC.closed_start_traverse;
            std::reverse(traverse_points_.begin(), traverse_points_.end());
            traverse_points_.pop_back();
            i = -1;
          }
        else
          {
            //if the point is in the middle we can try
            //to end the traverse here and compute the
            //stuff we have got
            tr_type = AC.closed_start_traverse;
            for (auto t = traverse_points_.size()-1; t<i-1; --t)
              {
                candidate_traverse_points_.insert(traverse_points_[t]);
                traverse_points_.pop_back();
              }
            if (traverse_points_.size() < 1) return false;


          }
      }
  }

//calculate bearings and coord differences
std::vector<double> bearings;
std::vector<double> dX;
std::vector<double> dY;
std::vector<LocalPoint> XY;
if (!front_ori.second && tr_type != AC.closed_traverse) return false;
if (front_ori.second)
  {
    double b = front_ori.first + angles.front();
    while (b > 2*M_PI)
      b -= 2 * M_PI;
    while (b < 0)
      b += 2 * M_PI;
    bearings.push_back(b);
    dX.push_back(distances.front() * std::cos(b));
    dY.push_back(distances.front() * std::sin(b));
  }
if (distances.size() != angles.size()) return false;
for (Acord2::size_type j = 1; j < angles.size();++j)
  {
    double b = angles[j] + bearings[j - 1] - M_PI;
    while (b > 2 * M_PI)
      b -= 2 * M_PI;
    while (b < 0)
      b += 2 * M_PI;
    bearings.push_back(b);
    dX.push_back(distances[j] * std::cos(b));
    dY.push_back(distances[j] * std::sin(b));
  }

//caculate point coordinates
```

```
  double X = PD[traverse_points_.front()].x();
  double Y = PD[traverse_points_.front()].y();
  traverse.push_back({ traverse_points_.front(), PD[traverse_points_.front()] });
  for (Acord2::size_type k = 0; k<dX.size();++k)
    {
      X += dX[k];
      Y += dY[k];
      XY.push_back(LocalPoint(X,Y));
      traverse.push_back({ traverse_points_[k + 1], LocalPoint(X,Y) });
    }

  //look for second orientation
  if (tr_type == AC.closed_traverse && !AC.in_missingXY(traverse_points_.back()))
    {
      std::set<PointID> neighbours = get_neighbours(traverse_points_.back());
      std::pair<double, bool> ori_back = { 0, false };
      for (auto n : neighbours)
        {
          if (!AC.in_missingXY(n))
            {
              traverse.push_back({n, PD[n]});
              ori_back.second = true;
            }
        }
    }
  return AC.transform_traverse(traverse);
}
```

E.7 acordweakchecks.h

```
/*
  GNU Gama -- Approximate coordinates by Polar Method
  Copyright (C) 2019  Petra Millarova <petramillarova@gmail.com>

  This file is part of the GNU Gama C++ library.

  GNU Gama is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GNU Gama is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GNU Gama.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef GAMA_LOCAL_ACORDWEAKCHECKS_H
#define GAMA_LOCAL_ACORDWEAKCHECKS_H

#include <gnu_gama/local/gamadata.h>
#include <gnu_gama/local/acord/reduce_observations.h>
#include <gnu_gama/local/acord/reduce_to_ellipsoid.h>
#include <gnu_gama/local/acord/acord2.h>

namespace GNU_gama { namespace local {

    class AcordWeakChecks
    {
    public:
      AcordWeakChecks(Acord2* acord2);
      void execute();
      std::pair<bool, PointID> check_traverse_endpoint(Acord2::Point pt);
      bool check_point(Acord2::Point pt);

    private:
      Acord2 & AC;
      PointData& PD;
      ObservationData& OD;
    };

}} //namespace GNU_gama::local

#endif
```

E.8 acordweakchecks.cpp

```
/*
  GNU Gama -- Approximate coordinates by Polar Method
  Copyright (C) 2019  Petra Millarova <petramillarova@gmail.com>

  This file is part of the GNU Gama C++ library.

  GNU Gama is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GNU Gama is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GNU Gama.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <gnu_gama/local/acord/acordweakchecks.h>
#include <gnu_gama/local/bearing.h>
#include <gnu_gama/local/orientation.h>

using namespace GNU_gama::local;

AcordWeakChecks::AcordWeakChecks(Acord2* acord2)
  : AC(*acord2), PD(acord2->PD_), OD(acord2->OD_)
{
}

void AcordWeakChecks::execute()
{
  // iterate through AC.traverses which now contains only unchecked
  // traverses
  for (auto tr : AC.traverses)
    {
      if (tr.second == AC.closed_start_traverse)
        // they should all be of this type by now but just in case...
        {
          auto check_res = check_traverse_endpoint(tr.first.back());
          if (check_res.first)
            {
              tr.first.push_back(Acord2::Point(check_res.second,
                                               PD[check_res.second]));
              // transform points first
              if (AC.transform_traverse(tr.first))
                {
                  for (auto pt : tr.first)
                    {
                      if (AC.in_missingXY(pt.id))
                        {
                          // only setting the coords, leaving other info as is
                          PD[pt.id].set_xy(pt.coords.x(), pt.coords.y());
                          AC.missingXY_.erase(pt.id);
                          AC.new_points_++;
                            }
                      }
                  }
              tr.first.pop_back(); //popping back the point we added
                                   //earlier for simpler erasing
            }
        }
```

```
    }
  if (!AC.traverses.empty())
    {
      // now we just iterate through the traverses one more time and
      // if the last point is known, we can safely remove them from
      // the list
      AC.traverses.erase(std::remove_if(
          AC.traverses.begin(),
          AC.traverses.end(),
          [this](std::pair<Acord2::Traverse, Acord2::Traverse_type> tr) {
            return (!AC.in_missingXY(tr.first.back().id));
          }), AC.traverses.end());
    }

  // similar procedure for points in same_points
  for (auto pt : AC.same_points_)
    {
      if (check_point(Acord2::Point(pt.first, pt.second)) &&
          AC.in_missingXY(pt.first))
        {
          PD[pt.first].set_xy(pt.second.x(), pt.second.y());
          AC.missingXY_.erase(pt.first);
          AC.new_points_++;
        }
    }
}

// check traverse endpoint: find if there are any observations from
// known points to the free end point if there are, try to compute the
// value from your computed cocrdinates, if the difference is
// acceptable, add traverse to PD returns true if point passes the
// check, along with pointID of the point that connects to the
// traverse

std::pair<bool, PointID>
AcordWeakChecks::check_traverse_endpoint(Acord2::Point pt)
{
  auto it = AC.obs_to_.lower_bound(pt.id);
  auto eit = AC.obs_to_.upper_bound(pt.id);

  while (it != eit)
    {
      Observation* obs = (*it).second;
      if (!AC.in_missingXY(obs->from()))
        // this means there is an observation from a known point to our
        // endpoint
        {
          auto dist = AC.get_dist(obs);
          if (dist.second)
            {
              double calc_dist = distance(PD[obs->from()], pt.coords);
              // if the point is within tolerance then we can add it
              if (std::abs(dist.first - calc_dist) <= AC.MAX_NORM)
                return {true, obs->from()};
            }
          auto dir = AC.get_dir(obs);
          if (dir.second)
            {
              StandPoint* sp = AC.find_standpoint(obs->from());
              if (sp == nullptr)
                {
                  ++it;
                  continue;
                }
              if (!sp->test_orientation())
                {
```

```
                  Orientation ori(PD, sp->observation_list);
                  double orientation_shift;
                  int n;
                  ori.orientation(sp, orientation_shift, n);
                  if (n > 0) sp->set_orientation(orientation_shift);
                }
              if (sp->test_orientation())
                {
                  double calc_dir = bearing(PD[obs->from()], pt.coords);
                  double calc_dist = distance(PD[obs->from()], pt.coords);
                  while (calc_dir > 2 * M_PI)
                    calc_dir -= 2 * M_PI;
                  while (calc_dir < 0)
                    calc_dir += 2 * M_PI;
                  if (std::sin(std::abs(calc_dir - dir.first))*calc_dist <
                      AC.MAX_NORM)
                    {
                      return { true, obs->from() };
                    }
                }
            }
        }
      ++it;
    }
  return {false, PointID()};
}

// check point: find if there are any observations from/to known
// points to/from a point from same_points if there are, try to
// compute the value from your computed cocrdinates, compare the
// biggest diff with NORM_MAX returns true if point passes the check

bool AcordWeakChecks::check_point(Acord2::Point pt)
{
  double maxdiff = -1;
  auto it = AC.obs_to_.lower_bound(pt.id);
  auto eit = AC.obs_to_.upper_bound(pt.id);

  while (it != eit)
    {
      Observation* obs = (*it).second;
      // this means there is an observation from a known point to our
      // point
      if (!AC.in_missingXY(obs->from()))
        {
          auto dist = AC.get_dist(obs);
          if (dist.second)
            {
              double calc_dist = distance(PD[obs->from()], pt.coords);
              double diff = std::abs(dist.first - calc_dist);
              if (diff > maxdiff) maxdiff = diff;
            }
          auto dir = AC.get_dir(obs);
          if (dir.second)
            {
              StandPoint* sp = AC.find_standpoint(obs->from());
              if (sp == nullptr)
                {
                  ++it;
                  continue;
                }
              if (!sp->test_orientation())
                {
                  Orientation ori(PD, sp->observation_list);
                  double orientation_shift;
                  int n;
```

```
                ori.orientation(sp, orientation_shift, n);
                if (n > 0) sp->set_orientation(orientation_shift);
              }
            if (sp->test_orientation())
              {
                double calc_dir = bearing(PD[obs->from()], pt.coords);
                double calc_dist = distance(PD[obs->from()], pt.coords);
                while (calc_dir > 2 * M_PI)
                  calc_dir -= 2 * M_PI;
                while (calc_dir < 0)
                  calc_dir += 2 * M_PI;
                double diff = std::sin(std::abs(calc_dir - dir.first +
                                        sp->orientation()))*calc_dist;
                if (diff > maxdiff) maxdiff = diff;
              }
          }
      }
    ++it;
  }

it = AC.obs_from_.lower_bound(pt.id);
eit = AC.obs_from_.upper_bound(pt.id);

while (it != eit)
  {
    Observation* obs = (*it).second;
    // this means there is an observation from a known point to our
    // point
    if (!AC.in_missingXY(obs->to()))
      {
        auto dist = AC.get_dist(obs);
        if (dist.second)
          {
            double calc_dist = distance(PD[obs->to()], pt.coords);
            double diff = std::abs(dist.first - calc_dist);
            if (diff > maxdiff) maxdiff = diff;
          }
        auto dir = AC.get_dir(obs);
        if (dir.second)
          {
            StandPoint* sp = AC.find_standpoint(obs->from());
            if (sp == nullptr)
              {
                ++it;
                continue;
              }
            if (!sp->test_orientation())
              {
                Orientation ori(PD, sp->observation_list);
                double orientation_shift;
                int n;
                ori.orientation(sp, orientation_shift, n);
                if (n > 0) sp->set_orientation(orientation_shift);
              }
            if (sp->test_orientation())
              {
                double calc_dir = bearing(pt.coords, PD[obs->from()]);
                double calc_dist = distance(pt.coords, PD[obs->from()]);
                while (calc_dir > 2 * M_PI)
                  calc_dir -= 2 * M_PI;
                while (calc_dir < 0)
                  calc_dir += 2 * M_PI;
                double diff = std::sin(std::abs(calc_dir - dir.first +
                                        sp->orientation()))*calc_dist;
                if (diff > maxdiff) maxdiff = diff;
              }
```

```
            }
          }
        ++it;
      }

    if (maxdiff < 0) return false;

    return (maxdiff<AC.MAX_NORM);
  }
```

E.9 acordstatistics.h

```
/*
  GNU Gama -- Approximate coordinates by Polar Method
  Copyright (C) 2019  Petra Millarova <petramillarova@gmail.com>

  This file is part of the GNU Gama C++ library.

  GNU Gama is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GNU Gama is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GNU Gama.  If not, see <http://www.gnu.org/licenses/>.
*/

#ifndef GAMA_LOCAL_ACORDSTATISTICS_H
#define GAMA_LOCAL_ACORDSTATISTICS_H

#include <gnu_gama/local/gamadata.h>
#include <gnu_gama/local/acord/acord2.h>

namespace GNU_gama { namespace local
{
  class AcordStatistics
  {
  public:
    AcordStatistics(PointData&, ObservationData&);
    void execute();

    int observations;
    int  given_xy, given_z, given_xyz;
    int  computed_xy, computed_z, computed_xyz;
    int  total_xy, total_z, total_xyz;
    bool missing_coordinates;
    PointData & PD;
    ObservationData& OD;
  };

}} //namespace GNU_gama::local


#endif
```

E.10 acordstatistics.cpp

```
/*
  GNU Gama -- Approximate coordinates by Polar Method
  Copyright (C) 2019  Petra Millarova <petramillarova@gmail.com>

  This file is part of the GNU Gama C++ library.

  GNU Gama is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  GNU Gama is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with GNU Gama.  If not, see <http://www.gnu.org/licenses/>.
*/

#include <gnu_gama/local/acord/acordstatistics.h>

using namespace GNU_gama::local;

AcordStatistics::AcordStatistics(PointData& pd, ObservationData& od)
  : PD(pd), OD(od)
{
  missing_coordinates = false;
  computed_xy = computed_z = computed_xyz = 0;
  given_xy = given_xyz = given_z = 0;
  total_xy = total_xyz = total_z = 0;
  observations = 0;

  for (PointData::const_iterator i = PD.begin(); i != PD.end(); ++i)
    {
      const PointID& c = (*i).first;
      const LocalPoint& p = (*i).second;
      bool cp = p.test_xy();
      bool hp = p.test_z();

      if (cp && hp) given_xyz++;
      else if (cp) given_xy++;
      else if (hp) given_z++;

      if (p.active_xy() && !cp) missing_coordinates = true;
      if (p.active_z() && !hp) missing_coordinates = true;

      if (p.active_xy() && p.active_z()) total_xyz++;
      else if (p.active_xy())  total_xy++;
      else if (p.active_z())  total_z++;
    }

  for (ObservationData::const_iterator
         i = OD.begin(), e = OD.end(); i != e; ++i, ++observations);
}

void AcordStatistics::execute()
{
  int known_now_z, known_now_xy, known_now_xyz;
  known_now_z = known_now_xy = known_now_xyz = 0;
  for (PointData::const_iterator i = PD.begin(); i != PD.end(); ++i)
    {
      // const PointID& c = (*i).first;
```

```
      const LocalPoint& p = (*i).second;
      bool cp = p.test_xy();
      bool hp = p.test_z();

      if (cp && hp) known_now_xyz++;
      else if (cp) known_now_xy++;
      else if (hp) known_now_z++;
    }
  computed_xy = known_now_xy - given_xy;
  computed_xyz = known_now_xyz - given_xyz;
  computed_z = known_now_z - given_z;

  if ((total_xy + total_xyz + total_z) ==
      (known_now_xy + known_now_xyz + known_now_z))
    {
      missing_coordinates = false;
    }
}
```