LEGO: A Modular Approach to Accelerator Alignment Data Analysis*

Catherine Le Cocq
Stanford Linear Accelerator Center, Stanford University,
Stanford, CA, USA

1. INTRODUCTION

The underlying unity of the numerous surveying computational methods is hidden by many practical differences in data acquisition. Traditional programming languages have added to the confusion by requiring programmers to describe the numeric data in very concrete and low-level structures (mostly arrays). In fact the algorithms behind all coordinate determination from surveying observations come down to basic methods of linear algebra. Lego uses the paradigm of object oriented programming (OOP) to more closely model the fundamental mathematical structures of all geodetic methods. Once the methods are in OOP form, the commonality across them becomes more obvious and a general architecture for a wide range of geodetic treatments becomes possible. This paper describes the fundamental concepts of this architecture and its advantages in terms of clarity (maintainability, testability and multi-author), portability and extensibility (observation types, resolution techniques and storage methods).

The very first version of Lego was built in 1994 as a set of C routines to be used for the adjustment of theodolite data and tracker data. The routines were organized into six modules. Each module answered a specific task. The tasks had been identified as followed: general implementation, input, generic surveying formulas, statistical functions, matrix manipulation and specific resolution technique. This organization was the reason for the name Lego, but more seriously the purpose of this separation was to make Lego easily adaptable to any environment and easily expandable to new resolution techniques. At a second look, it was also a cry for being converted into a more modern language. Because C++ is primarily a superset of C, most C++ compilers have no problems compiling regular C code and may also handle a mixture of C and C++. This made the transformation of Lego very fast and painless. Up to now Lego is still using C functions for file access and dynamic memory allocation but is organized into classes allowing stronger data typing and, most of all, data hiding. It also benefits from some of the more advanced concepts of object programming such as encapsulation and virtual functions. Unlike with the C version where different executables coexisted there is only one C++ Lego.

* Work supported by Department of Energy under Contract DE-AC03-76SF00515.

2. ARCHITECTURE

2.1 General Program Flow

In order for Lego to have a pure mathematical model it is important to shield the data structures used in computation from the details of the data acquisition and result reporting. In the course of the execution of the program, the information flows through 3 stages of data objects.

The first stage has 3 main objects representing the stations, the targets and the measures. It also has auxiliary objects needed in specific applications such as gravity related observations. Those objects are close to the description familiar to the user. They use the angle conventions, orientation and unit, reported by the instrument. They are an appropriate basis for a graphical user interface or a data base scheme. They encompass all the data known to the program and constitute the source for any number of successive computations.

All the objects of the second phase are built for efficient data crunching. They exist only during the active phase of computation. They are instantiated as needed and inaccessible to any user action. They follow an internally coherent system of units and conventions. Instead of representing the objects the user is familiar with, they represent the objects that the mathematical treatment is expecting. For instance, both stations and targets are grouped in points. Other objects arise from an expert analysis of the input data, like the lump transformation object which holds the deformation model of a solid object.

The last stage translates the output of the computation into the terms, units and conventions, familiar to the users. It holds permanently the results of an invocation of Lego. As such it can be used, for instance, to visually compare the results of successive invocations of Lego. As the program has been used at one site, the output is mostly tailored to produce the reports that those users are accustomed to. A wider audience will probably require some additional refinements.

Throughout the 3 stages, error reporting and user prompting are handled through an abstraction of the user interaction. This allows the Lego architecture to be independent of systems and user interfaces.

2.2 Class Organization

The main classes in Lego are called Feedback, Project, Lego, Approx and Method. The first 2 are designed to answer the simple task of data access and communication; they have to be customized to the user needs. The last ones encapsulate all the geodetic, mathematical and statistical analysis. They work together and have access to some of each other’s data members. To the user, they form a computational black box. To the designer of a new resolution technique or the implementer of a new type of observation, they are a toolbox of appropriately sized generic methods chunks. The grouping of the last classes makes Lego a 3-part package generally handled as 3 DLLs: Feedback, Project and Lego itself

The class Feedback gives the way to communicate with Lego. It has 2 public functions called simply feedback() and decision(). The first one allows you to output a character string in the environment of your choice. In the simple case of a console application it is equivalent to printf(). The second one allows you to input the result of a choice you have been asked to make.

The class Project holds the implementation of the first and last stages of objects. It can be customized to adapt to a different set of user conventions. Its goal is primarily to be used for interactive user editing and selecting. Up to now, the integrity of the data is not checked at that

stage. This may change in the future, putting more requirements on the design of new customizations of Project.

The class Lego holds the main data repository during computation. When a Lego object is created, the data are converted from Project members, a user view, to Lego members, a mathematical view. Consistency and completeness are checked at that point. Minor problems that can be unambiguously isolated are automatically fixed with a warning (presence of vertical angles in a 2D adjustment for instance). Secondary objects are created as needed, for example the lump transformation object if any lump is recognized as being active. Then, the parameters to be solved later are identified; all of them must have approximate values. The quality of these approximate values is very critical to ensure a proper convergence for any resolution process. If needed or upon request, Lego instanciates a new class, called Approx, which computes all the necessary approximate parameters in the case of networks observed with theodolites and/or trackers as long as they are semi-leveled. At the end of this data preparation, the Lego object is in a state that is an appropriate starting point for any of several algorithms. Using the object oriented paradigm, intrinsically different methods present the same public interfaces as explained in the next subsection. One of those methods is instanciated and called. At the end of the execution of the method, the destructor of the Lego object copies the relevant information into the result member of the Project object, and releases the memory used.

2.3 General Method Interface and Inheritance

Up to the point where a particular method is called, there is not one single line of code that assumes anything about the kind of algorithm that will be used for the adjustment. A common interface that fits any geodetic method will be presented now as well as the implementation of different specific ones. This will constitute a demonstration of the thesis that any geodetic method can be described as an object exporting the interface proposed. This particular set of primitives has been developed through successive implementations of adjustment programs at SLAC.

The embodiment of this concept in OOP is called a virtual function. This means that the base class, Method, provides the implementation of all the functions and data members common to many algorithms. It also provides the interface description of those parts of a geodetic method that change depending on the algorithm chosen. Each specific method provides a particular implementation that respects the base class interface. It can also add new functions and data members which are not part of the base class Method because they are specific to that particular resolution. The mechanism by which a class adheres to all the interfaces of a given class is called, in OOP terms, inheritance.

Lego maintains 3 algorithms for the reasons explained in subsection 3.2. The first one is a L1 norm minimization. The last two are variations for L2 norm minimization. The classes Simplex (for the L1 norm), Compact and Jordan (for the L2 norm) explicitly provide the implementation of each function that was declared virtual in Method. The virtual functions may be split into 2 categories: the utility group and the mathematics group. The utility functions perform the following accessory tasks: identification output, debug output and memory freeing. The other virtual functions come from the understanding that any geodetic adjustment may be represented mathematically by a system of linear equations: $\mathbf{M} \cdot \boldsymbol{\delta} = \boldsymbol{\upsilon}$ and performed by a 3-steps process. This formal linear system is derived from the overdetermined system of linearized observation equations: $\mathbf{v} = \mathbf{A} \cdot \boldsymbol{\delta} - \mathbf{l}$ after addition of a condition on the residual vector $\mathbf{v}$ and treatment of a

potential rank deficiency in the coefficient matrix **A**. The first of the 3 steps takes care of all the space allocation necessary for the final linear system and the extra entities required by the chosen algorithm; it is performed by only one virtual function call. The second step builds and solves the final linear system. It is based on the sequential approach of observations processing. It is performed by 3 virtual functions (named simply after their actions: initialize, build and solve) and an always expanding battery of partial derivatives computation functions to deal with individual types of observations. This second step may be inside an iterative loop. The last step is not always present, it may be called assessment of the results and has a particular meaning in the L2 resolution framework. It requires one call to a virtual function to invert the normal matrix of the system, the sole provider of information for statistical analysis either on the residuals or on the parameters.

3. PRACTICAL VALIDATIONS - FUTURE

3.1 Actual Implementations

For testing and development purposes Lego is operated in the simplest environment possible. It is launched as a console application getting its data by reading files. To avoid the multiplicity of format for input files, Lego uses the same input file as the one designed for its predecessor, the SLAC bundle program 3DCD. Because the scope of the other program is not as broad, all the specific entries of Lego not present in the old input file system are obtained in a complementary file with a system of key-words. The validation of the results is made by an analysis of the output file; there is no explicit use of the third stage of data structure. The output file of Lego may be better viewed as a log file reporting all the steps in the analysis. It has right now 3 levels of output: the common one, a more detailed one for classical debugging purposes and a complete one with the output of the normal matrix (not to be used without any good reason!). One more time to avoid the multiplicity of presentations, the output file follows the same format as 3CDC.

The second implementation of Lego is within a Windows program allowing on-line modification of entries with the use of a tabbed dialog box. The main part of the data acquisition is still made with the input file mentioned above; no other file is required as the user may review, complete, change and select most of the entries. The output is also presented with a tabbed dialog box in an organization symmetrical to the input one. The version developed with a Borland C++ compiler is tested and available, but not currently maintained and therefore lacks some of the new capabilities present in Lego.

The third implementation is for a specific use of Lego: error propagation for planning surveying networks. In this case the data acquisition is made through the graphical environment. There is a direct access between the data structures of SIMS (the graphic engine) and the ones of Lego. To benefit from the latest options of Lego the graphic data structures have been extended. The user may now enter individual standard deviation and epoch stamp on each observation. The station and point graphic structures have been augmented with 2 new pieces of information: a lump index and a connection status. Therefore the simulation may take into account the deformation of a part identified with the lump index within the time frame indicated by the epoch stamps on the observations. By a simple click the user may toggle the computation with or without deformation which may be very useful for planning a solid survey. The connected option is particularly useful in the case of multi-level networks.

3.2 Time and Space Requirements

A common question asked when rating an adjustment package is how many points it can handle. Lego's answer is how many can fit into your machine. Because all the data are dynamically allocated there is no a-priori limit on number of stations, points or observations. As in all adjustment packages, the main request for memory is at the resolution level and thus depends on the option chosen. Lego gives the user the choice between 2 types of resolution: L1 norm or L2 norm. In both cases the idea is to solve the system of linearized weighted observation equations with a condition on the residuals.

In the L1 norm resolution the condition is the minimization of the weighted sum of the absolute value of the residuals (i.e. the L1 norm). It is performed with a Simplex algorithm. The Simplex method has been developed within the frame of a linear programming problem using positive unknowns; it is usually presented in a tableau format. In the case of a network of n unknown parameters and m observations, the tableau involved has m lines and 2(n+m) columns. The columns represent the actual unknowns in the Simplex resolution and may be classified into 4 subsets: the basic variables, their negative counterparts, the non-basic variables and their negative counterparts. Lego keeps in memory only the part of the tableau related to the non-basic variables which is the equivalent of a matrix of m rows and n columns. This may be very demanding in the case of a large dense network. But because the use of the L1 norm is primarily recommended for blunder detection and not for coordinate determination, splitting the network in smaller areas and testing each one individually is often a simple solution. As part of experiments on robustness in blunder detection within less redundant networks, a method of balanced adjustment has been implemented. Its principle is to equalize the redundancy numbers of all observations. It is obtained in an iterative process requiring the building and inversion of the normal matrix. At that point, no special effort on storage savings and computation time for that specific task has been made leading to stable but slow results.

In the L2 norm resolution the condition is the minimization of the weighted sum of the square of the residuals (i.e. the L2 norm). It leads to the resolution of a normal system which may be summarized by a symmetric positive definite matrix whose rank is the number of parameters in the problem. The geodetic literature is full of methods for storage and resolution of such systems. Lego currently supports 2 of them, both based on Gaussian elimination. Other L2 methods drawn from the pool of both direct and iterative methods have been investigated but none has been proven to add any significant advantage over the 2 selected.

The first method uses the Gauss-Jordan algorithm with partial pivoting. This method is extremely stable and generates the inverse of the original matrix. Both the original matrix and its inverse are stored in an array of size n^2. Because of this high memory consumption and its relative slowness, it is mainly used as a check on the other method. The method recommended takes advantage of the inner structure of the normal matrix and is referred to as the compact method. In most large networks, because not every point is observed by every station, the normal matrix is relatively sparse. The distribution of the zeros depends on the order of the parameters. Lego chooses to consider first the coordinate unknowns then the nuisance parameters (rotations, offsets, lumps transformation parameters, etc) and finally the datum unknowns in the case of a free net adjustment. This arrangement leads to a concentration of the nonzero elements along the diagonal and in the last rows of the normal matrix. The compact method stores the normal matrix in an array filled row by row, keeping only the elements from the first nonzero element of the row to the diagonal. It has the double advantage of saving space and speeding the process of row

elimination. Used in the resolution stage (when only the solution of the linear system is required), this envelope technique has been proven to be very efficient. Used in the inversion stage, it may be too slow for very large networks. The remedy used at SLAC is to solve the normal system for the entire network and to break it into sub areas for a fully detailed analysis, eventually recombining the whole using the connected network approach.

Table1: Total run time for real-world networks done at SLAC.

200 MHz Pentium – 64 MB RAM		Network A	Network B
Size	Number of observations	293	3064
	Number of unknowns	128	1527
L1 method	Simplex algorithm	11s	2h 31min 3s
	Simplex algorithm with observations balancing	49s	N/A
L2 method	Number of iterations	5	7
	Compact algorithm	<1s	1min 59s
	Compact algorithm with inversion	1s	8min 6s
	Gauss-Jordan algorithm	4s	3h 9min 45s

Table 2: Timing for one iteration using the compact algorithm.

200 MHz Pentium Pro – 256 MB RAM	Network A	Network B	Network C
Number of stations	5	65	399
Number of targets	29	377	2734
Number of observations	293	3064	16454
Number of unknowns	128	1527	11001
1 iteration of Compact solution	<<1s	15s	50 min

3.3 A-Posteriori Implementation of New Observation Types

Lego has evolved over time following the demand of its users, demonstrating its modularity and the versatility of its data structures. Using the “bundle approach”, Lego had originally no direct way to take into account observations referenced to gravity in its 3D model. Leveled theodolites were treated as standard instrument carrying their 3 rotations. Precise leveling data could be incorporated into a 3D adjustment but only after being corrected. Answering the demand for less pre-treatment of the observations, Lego incorporated the concept of an earth model into its mathematical model. A new data structure was created to regroup all the information necessary to identify a model of earth and its relationship to the reference system used in the adjustment. Because the scope of applications for Lego is microgeodetic works, primarily accelerator survey, a simple spherical model is implemented. Now the user can enter leveled theodolite observations simply by indicating that the first 2 rotations of the station are zero and that a model of earth will be used to internally correct both horizontal and vertical angles generated at this station. To be even closer to field reality, corrections for station and target offsets are performed automatically using a similar algorithm.

An even better example of the expandability of Lego is the addition of photogrammetric observations. As far as the Lego class was concerned, only one new object had to be created in order to store and manage all the possible camera parameters involved in the adjustment process.

No special change had to be done to any existing objects. Because the treatment of image coordinates by Lego is still in the development phase, no specific addition to the output object has been made yet. At the input stage, 2 new objects have been introduced. The first one is used to enter the a priori knowledge of all the cameras present in the project. It tells also how the calibration parameters are going to be treated in the bundle. This is the source of information for the Lego object. The second one is temporary and based on the actual (and transitory) method to get the image coordinates through files. A more coherent approach would have been to mimic the method applied to the other types of observation. The general method interface described in subsection 2.3 works perfectly. The class Method grows with the addition of the specific routines to deal with image coordinates namely the computation of the partial derivatives and the calculated values associated with the 2 image-coordinates observation equation.

3.4 Discussion and Future

The way the photogrammetry has been integrated into Lego as far as creation of objects and relationship between objects could be applied to distance measurements. The common models for a distance observation are based on the introduction of 2 possible parameters: an additional offset and a multiplicative factor. Right now, these 2 parameters representative of the distance model are integrated inside the station object. As any parameters, they have a-priori values and codes to tell how they will be treated later in the bundle (unknown, fixed, weighted). If a new object characterizing the distance model were created at the input stage in a similar way as the camera object, it would be easier to handle the distance models. The actual way could be simply reproduced by creating as many new objects as stations and having a one to one correspondence; but it would be easy to make a group of stations sharing the same distance model. In fact, this dissociation between station and instrument is closer to the field and creating objects representative of reality is the end goal in object oriented programming.

The treatment of photogrammetric data has just being touched. Lots of testing work has to be undergone before the bug-free seal could be awarded! An interesting subject for studies will be the analysis of lumps deformations based on image coordinates observations. Of course, another direction is the production of an integrated package from the images acquisition and treatment to the bundle solution.

Another direction for expansion is the addition of new types of observations. Several preliminary studies have been conducted around the idea of offset measurements. They will be finalized as soon as some instrumentation will be commonly operational. The next addition will be the transfer of points along the vertical. The need for direct observations between multi-level networks has arisen recently and Lego will answer the user demand.

4. CONCLUSION

After an investment that is estimated to be under 2 person-years, Lego offers a geodetic bundle package which has proven to be both reliable and accurate on the one hand, flexible and extensible on the other hand. Lego can calculate networks that are limited in size only by the computer memory (and user patience) available. While the traditional presentation of position measurement sciences split geodesy and photogrammetry into completely different domains, Lego can accept data from cameras, theodolites, trackers and levels simultaneously.

These results are a validation of the claims of productivity and simplicity from the object oriented programming advocates. Furthermore, the object-oriented framework brings to light the common foundation of many geodetic methods, allowing incremental additions and non-traditional mixes. This modular approach to alignment surveying should become even more fruitful as many more methods and instruments are added to Lego while keeping the program clear and manageable.

5. ACKNOWLEGMENTS

I would like to thank C. Salsberg for helpful programming discussions and for providing the various graphical user interfaces driving Lego. Many thanks to M. Gaydosh for his patience as a test user and for requesting interesting new features. Last but not least, I appreciate the support and trust of the whole alignment team and especially of its manager R. Ruland.

6. REFERENCES

[1] C. Le Cocq, *LEGO: The Mathematical Model*, Internal Paper 1995.

[2] R. Ruland and C. Le Cocq, *Relocation: A Concept for Dealing with Measurement Object Expansion or Contraction*, SLAC-Pub 7414 presented at the Boeing Tracker Workshop, Renton, WA, Jan 14,15, 1997.