# 21

# Off-line Programming Industrial Robots Based in the Information Extracted From Neutral Files Generated by the Commercial CAD Tools

Vitor Bottazzi & Jaime Fonseca
*Department of Industrial Electronics - University of Minho*
*Portugal*

## 1. Introduction

In order for a robotic manipulator to perform useful work, it must be programmed to accomplish the desired task or motion cycle. Nowadays industrial robots generally require a tremendous amount of programming to make them useful. Their controllers are very sophisticated, the commercial robot programming environments are typically closed systems and the programming languages varies from manufacturer to manufacturer. Despite the great evolution of the industrial robots controllers, in the majority of the industrial applications, the robot programming is made, using one of the following ways:

- Manual on-line programming;
- Off-line programming;

Manual on-line programming refers to physically teaching a robot the required trajectory, through interaction with teach pendant or other similar device (Lee & ElMaraghy, 1990). This programming kind presents the following disadvantages: very slow, it needs that the robot is available, difficulty in the handling of equipments, need some practice in the language used by the robot, and technical knowledge to understand the operation of the equipment. These disadvantages are very expensive in the industry because the productive process needs to stop for a long time.

One simple approach to solve some disadvantages described above is the Off-line programming environments. These environments are based in graphical simulation platforms, in which the programming and execution process are shown using models of the real objects. Consequently, the robot programmer has to learn only the simulation language and not any of the robot programming languages. Other benefits in off-line programming environments include libraries of pre-defined high-level commands for certain types of applications, such as painting or welding, and the possibility to assess the kinematics feasibility of a move, thus enabling the user to plan collision-free paths. The simulation may also be used to determine the cycle time for a sequence of movements. These environments usually provide a set of primitives commonly used by various robots, and produce a sequence of robot manipulator language primitives such as ”move” or ”open gripper” that are then downloaded in the respective robot controllers.

However, the off-line programming tools based in graphically 3D representation presents several problems in many industry applications, particularly, when the robot task or the robot trajectory needs frequent changes, for example: in welding applications where the configuration of the pieces to weld change frequently (the size, the shape, etc.); the robot painting and gluing applications can have similar problems.
Nowadays, the CAD tools are often used in the industry to develop and to document the products and its manufacture. There are a lot of commercial CAD tools, like, AutoCAD, SolidWorks, Ideas and Cimatron, having each tool its own file format. However, it is possible to export the information of these pieces, in a neutral file format, namely: STL, IGES, STEP and SET formats.
This work presents one solution for programming different robots based in the relevant information extracted from neutral files. The solution implemented was tested in the industrial robots Mitsubishi (Mitsubishi Move Master Industrial Robot) and ABB (model IRB 140 with IRC5 controller). This chapter is organized as follows: section 2 presents an overview about the format of neutral files (STL, IGS, STEP and SET); in the section 3, the algorithms for extraction of the relevant information from the neutral files are described; in the section 4, the developed tool for code generation for different industrial robots is presented; section 5 and 6 present the results and conclusions; section 7 presents future work.

## 2. Neutral file formats

Computer Aided Design (CAD) technology for engineering, and manufacturing is now playing an increasingly important role in production industry. The importance of this technology to increase productivity in engineering design has been widely recognised. These technologies make it possible to shorten the time and lower the cost of development. Additionally, the reliability and the quality of the product can be improved. CAD systems have therefore been used in various fields of industry including automobile and aircraft manufacture, architecture and shipbuilding, and there are currently many commercial systems available. SolidWorks, Catia, Inventor and Cimatron are examples of available systems using CAD technology.
With the existence of a great diversity of CAD tools emerge the demand to import/export files between different CAD software. The emergence of neutral format files and neutral format file interfaces in order to exchange product data between CAD systems solve this problem. The most widely accepted formats have been the Initial Graphics Exchange Standard (IGES), the *Standard d'Echange et de Transfert* (SET), the STandard for the Exchange of Product model data (STEP) and the Standard Transform Language (STL).

### 2.1 Initial Graphics Exchange Standard (IGES)

IGES (Smith et al., 1988) was the first specification for CAD data exchange published in 1980 as a NBS (National Bureau of Standards) report (IGES 1, 1980) in USA.
The version IGES 5.2, provide the following capabilities:

- Geometry : 2D/3D wireframes, curves and surfaces; CSG (Constructive Solid Geometry) and B-Rep (boundary Representation) are supported;
- Presentation : Drafting entities for technical drawings;
- Application dependent elements : Piping and electronic schematics, AEC elements;
- Finite Element Modelling : Elements for FEM (Finite Element Method) systems

An IGES file consists of six sections: Flag, Start, Global, Directory Entry, Parameter Data, and Terminate. Each entity instance consists of a directory entry and parameter data entry. The directory entry provides an index and includes attributes to describe the data. The parameter data defines the specific entity. All the parameter data are defined by fixed length records according to the corresponding entity. Each entity instance has bi-directional pointers between the directory entry and the parameter data section (Vuoskoski, 1996).

### 2.2 Standard d'Echange et de Transfert (SET)

SET is a French standard for the exchange and archiving of CAE data and is supported by several CAD systems. It was developed as a neutral file format for exchanging data between different CAD systems at Aerospatiale in 1983. The aim was to develop a more reliable alternative to IGES. It supports wireframe, surface and solid models, including CSG and B-Rep. Entities for drafting and connectivity applications, as well as scientific data and FEM (Finite Element Method) modelling are also included. It was considered to be important to have an unambiguously defined format that is compact in size, and is flexible enough to handle future demands from the CAD/CAM industry (Vuoskoski, 1996).
The structure of SET is based on a three-level hierarchy of data assemblies, data blocks, and data sub-blocks. Information that is common to several blocks or assemblies is stored in a so-called dictionary.

### 2.3 Standard for the Exchange of Product model data (STEP)

STEP (Owen, 1994) is a new International Standard (ISO 10303) for representing and exchanging product model information. It includes an object-flavoured data specification language, EXPRESS (Schenck& Wilson, 1994) to describe the representation of the data. STEP defines also implementation methods, for instance, a physical transfer file, and offers different resources, e.g. geometric and topological representation.
The objective of STEP is to offer system-independent mechanism to describe the product information in computer aided systems throughout its lifetime. It separates the representation of product information from the implementation methods. Implementation methods are used for data exchange.
The representation offers a definition of product information to many applications. STEP provides also a basis for archiving product information and a methodology for the conformance testing of implementations.
EXPRESS is a formal data specification language used to specify the representation of product information. The use of a formal data specification language facilitates development of implementation, and also enables consistency of representation. STEP specifies the implementation methods used for data exchange that support the representation of product information.
STEP does not only define the geometric shape of a product: it also includes topology, features, tolerance specifications, material properties, etc. necessary to completely define a product for the purposes of design, analysis, manufacture, test, inspection and product support. The use of STEP is still very modest but it is growing all the time. The majority of CAD system vendors have implemented or are implementing STEP pre- and post-processors for their CAD systems. STEP is

an evolving standard which will cover the whole product life cycle in terms of data sharing, storage and exchange. It is the most important and largest effort ever established in engineering domain and will replace current CAD exchange standards.

### 2.4 Standard Transform Language (STL)

STL is a file format native to the stereolithography CAD software created by 3D Systems of Valencia, CA, USA and is perhaps the main standard for rapid prototyping systems. STL files may be ASCII or binary data, although binary is far more common due to the resulting size of the CAD data when saved to the ASCII format.

**a) ASCII format**

The first line is a description line that must start with the word "solid" in lower case; it then normally contains the file name, author, date etc. The last line should be the keyword "endsolid". The lines between the above contain descriptions of 3 vertex facets including their normals, the ordering of the vertices should comply with the right hand rule.

The syntax for an ASCII STL file is as follows:

```
solid [name_of_object]
 facet normal x y z
    outer loop
      vertex x y z
      vertex x y z
      vertex x y z
    endloop
  endfacet
  facet normal x y z
    outer loop
      vertex x y z
      vertex x y z
      vertex x y z
    endloop
  endfacet
...
endsolid name_of_object
```

Normal vector components and vertex coordinate data are written in scientific notation (+-d.ddddddE+-ee). Often the normal's need not be provided and they will be generated by the parsing software/system. The main restriction placed upon the facets in STL files is that all adjacent facets must share two common vertices (figure 1).

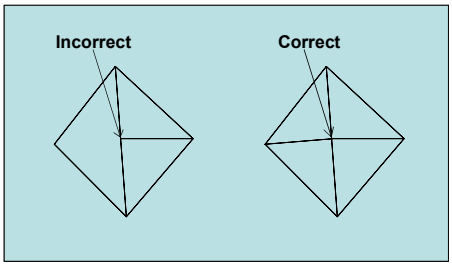


Fig. 1. STL representation restriction.

As an example consider the following except from a STL file

```
solid
 facet normal  0.000000e+000 9.971213e-001 -7.582376e-002
  outer loop
        vertex 7.293332e+002 2.200000e+002 1.183396e+003
        vertex 7.295713e+002 2.200000e+002 1.183396e+003
        vertex 7.295713e+002 2.190000e+002 1.170246e+003
       endloop
 endfacet
 facet normal  5.202612e-003 9.971148e-001 -7.572907e-002
       outer loop
        vertex 7.295713e+002 2.190000e+002 1.170246e+003
        vertex 7.293332e+002 2.190000e+002 1.170229e+003
        vertex 7.293332e+002 2.200000e+002 1.183396e+003
       endloop
 endfacet
 facet normal  0.000000e+000 9.971076e-001 -7.600333e-002
  outer loop
   vertex 7.295713e+002 2.200000e+002 1.183396e+003
   vertex 7.298094e+002 2.200000e+002 1.183396e+003
   vertex 7.298094e+002 2.190000e+002 1.170277e+003
  endloop
 endfacet
endsolid
```

**b) BINARY format**

Binary STL files consist of an 80 byte header line that can be interpreted as a comment string. The following 4 bytes interpreted as a long integer give the total number of facets. What follows is a normal and 3 vertices for each facet, each coordinate represented as a 4 byte floating point number (12 bytes in all). There is a 2 byte spacer between each facet. The result is that each facet is represented by 50 bytes, 12 for the normal, 36 for the 3 vertices, and 2 for the spacer.

<STL file> := <name> <facet number> <facet 1> <facet 2> ... <facet n>
<name> := 80 bytes file name, filled with blank
<facet number> := 4 bytes long int integer
<facet> := <normal> <vertex 1> <vertex 2> <vertex 3> <fill-bytes>
<normal> := Nx, Ny, Nz
<vertex> := X Y Z
<fill-bytes> := 2 fill bytes

## 3. Extracting relevant information from STL file

After compare the different types of neutral file formats in this work, the STL format was chosen because it presents the following advantages:

- The information about the 3D coordinates of the points that compose the object it is easily extracted;

- The dimensions of the object for using in the Off-line programming it is easily imported from the STL files;
- The loss of information about the layers, colours, and other attributes in the export process for the STL format don't bring significant losses for this specific application;
- The main commercially CAD tools export the information in STL format.

### 3.1 Implemented Algorithms for STL file reading

Extract 3D coordinates data from ASCII STL files is very easy like was described at the section 2.4 in this chapter. The easy handling syntax used to access its files stimulates it choice. Read binary file information is also usual, attempting to the correct size of byte arrays used in blocks data reading (section 2.4 – b) ).
Two algorithms will be shown exhibiting a suggestion to reading data implementation using pseudo language.

**Reading Binary STL file:**

```
Open File
To reserve 80 bytes array to header reading
To reserve 4 bytes array to facets number reading
To read header
To read facets number
While not EOF
    Read normal vector float xyz coordinates
    While not end of Triangle
        Read triangle vertexes float xyz coordinates
    EndW
EndW
Close File
```

**Reading ASCII STL file:**

```
Open File
To read line
While read line is not "endsolid"
    If read line is equal "vertex"
        While not End of Triangle
            Read triangle vertexes
        EndW
    EndIf
To read line
EndW
Close File
```

## 4. Used Project Techniques

Abstraction is a very important feature when thinking about industrial machine programming. Code abstraction makes possible generalize implementation reaching high code re-use and less memory allocation. This abstract programming way makes feasible and easy, merge code from isolated software packages to new and specialized applications fields. With an abstract modelling of basic mechanisms structures like: joints, axes, programs and points for example. It can facilitate the routines implementation and strings manipulation interaction. To compose motion

commands through many languages that will run over different robot controllers. At this section will be explained how programming techniques helps to extrapolate these structures and organize sequentially machine control commands.

### 4.1 Design Patterns

Every object oriented effective architecture is full of patterns. Because to use design patterns during the development will provide a smaller, simplest and maintainable architecture, when compared with other paradigms (Gamma et al., 1998).
Experience is an evaluated characteristic in every kind of business. Also in oriented object programming (OOP) is useful adopt renamed programmers successful experiments and relates, called Design Patterns to solve common implementation problems.
Code reusability is one of the benefits brought by OOP. But, projecting reusable object oriented software is a hard work. For that reason specific development problems are targeted by object oriented design patterns bringing flexibility, elegance and code recycle. Resuming, design patterns (DPs) is a high level mechanism that relates the best programmers' practices (successful experiences) to solve software projects common problems. The DPs used at present work was: Factory, Singleton, Facade, Model/View/Controller, Memento, Command and Template Method (Gamma et al., 1998). Next we will discuss this OOP techniques applied in the robot code generator development, his labels, it micro-architecture, common problems solved by this practices, and implantation consequences.

#### 4.1.1 Factory

It can be understood like an objects factory. This pattern centralizes the objects creation with high changing probability in the project. The demand of a class that centralize the objects creation was the motivation to idealize it. Prohibiting object directly instance inside of business classes. The source problem is: if a signature method changes in a specific class, it will be necessary change all of direct instances to it class, becoming hard and complex work, considering that this object is being instanced by many classes. The proposed solution was developing a centralized solution class, responsible to create and return instance references from all called objects. If some object charger method changes, in the factory class, it will be centralized in the getInstanceX() method, where "X" is the name of a class that is moulding the object identity.
The strategy was analyze the list of classes that has an big probability to change, and implements its getInstance() methods. All GetInstance() methods implements the objects creation inside of a centralized factory class. Figure 2 shows the class Program requesting through Factory a reference to new Point object instance.

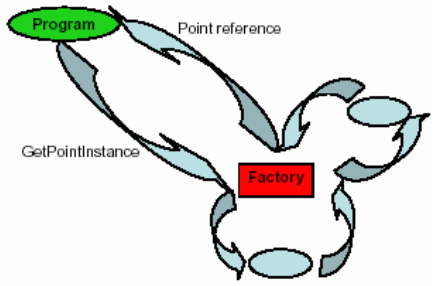


Fig. 2. Point instance request.

### 4.1.2 Singleton

It is a pattern that prohibits the duplication of objects in RAM memory. Your meaning is "unique instance". It was idealized because during the program execution many objects are recreated without real requirement, reflecting in memory and processing overhead. The main problem attacked by it DP is; the memory wastefulness caused by the sub-utilization of created objects in memory. It will decrease the overhead caused by java virtual machine garbage collector service, that monitories and cleans the unreferenced objects in memory.

The proposed solution was, to verify during object creation if it still exists in memory. The direct consequences to it practice: decreases process overhead caused by garbage collector and increase the objects life cycle, reflecting in a better memory resources allocation. The follow strategy was represented in figure 3.

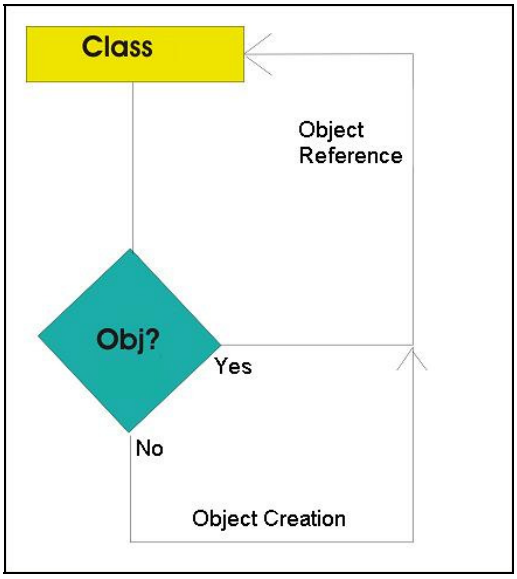


Fig. 3. If the object still exists in memory is returned a reference to him, else the object is created and also is returned a pointer to it new object.

### 4.1.3 Facade

The facade pattern decouple user interface and business classes, through an acknowledged default data input point. The meaning of facade can be understood like "consensus". It was implemented to make possible connect different interfaces with the same business classes, through a centralized data input, decoupling interface layer from business layer. That problem is: if the interface changes, also will be necessary changes the businesses classes to compatible it. This practicum suggests the creation of a class that hide interface layer system complexity and allows interaction with any kind of input, like command prompts, database queries, applets or encoded data input applications. The direct consequences will be decrease interface interaction complexity and decouple interface from business layer.

Thus all dependencies between involved entities in the use cases will be transparent to the user. Another visible gain is, the software maintenance becomes easier. Reflect in fast interface modifying and new interface input creation.
The strategy was implements a charger class, that knows how to talk with the business layer and known by interfaces that want to use the service. This way will make possible change the presentation layer without big software set adjustments. Figure 4 shows how charger class, talks with business layer, and it data inputs centralized model.

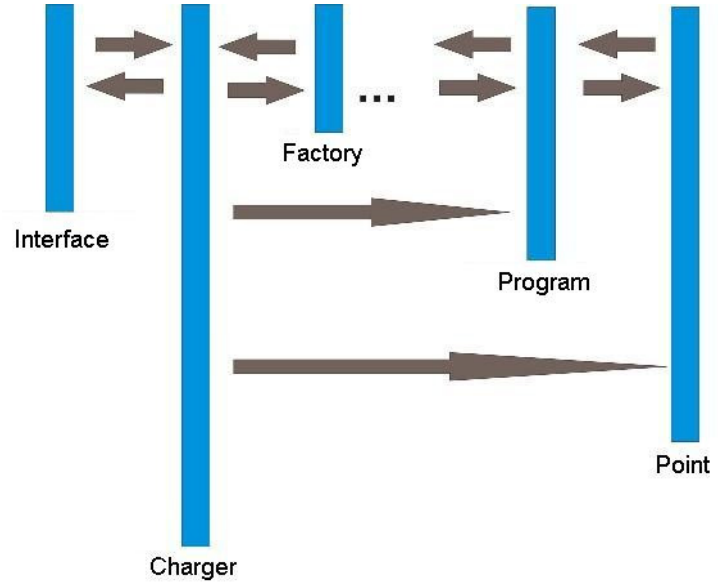


Fig. 4. The charger intermediation.

#### 4.1.4 MVC

The triple MVC is an acronym to Model/View/Controller. It is used to develop object oriented interface keeping consistence between model and views through the controller.
This pattern is composed by 3 object types basically:

- The model that is the application object.
- The View that is the graphic representation.
- The Controller, who defines the interface behaviour reacting a data input.

Before MVC pattern creation this functionalities were grouped in a singular object. This pattern separates views from model establishing a subscription/notification protocol between them. The view has to reflect the model state, and if the model changes else all dependent views have to be notified. The main target of this pattern is link an unrestricted number of views to a model, bringing capability of many different data representations. It able inserts new views, without huge code modifications.
The aim of this pattern is separate data (Model) from user interface (View) and application flow (Controller). The gain of this practice is share the same business logic to be accessed through different interfaces, like is pointed at figure 5.
In MVC architecture, the Model does not know how much Views are showing its state. This pattern possibly append easily new human interface technologies to a specific solution, like

holograms for example, only attaching the needed hardware and adding the minimum code required to access the model behaviour.
Interactive applications require dynamic interfaces. Therefore reducing the coupling between interface and business classes will reflect in less development effort and easiest interface maintenance to new versions with new functionalities.

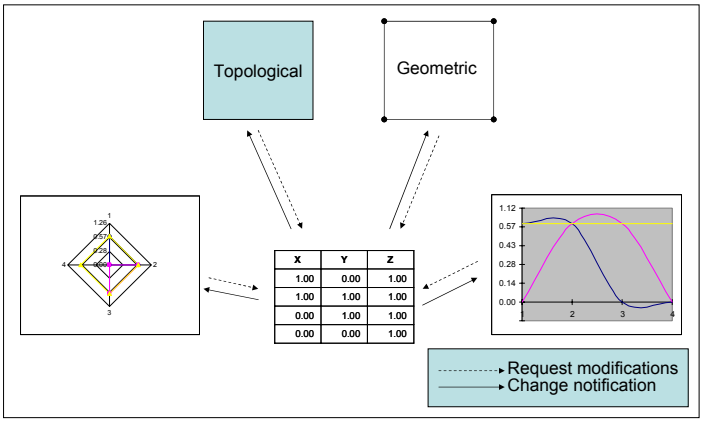


Fig. 5. Model View Controler Iteraction.

### 4.1.5 Memento

It is a pattern that stores the last recent memory interactions with the software, making possible recoup effected operations. Can be understood like "Undo". This demand was perceived when the user, seeking a program creation though line needs to undo some interactions in the code. The problem detected is: how to store the working structures? The proposed solution was store the most recently objects states in a limited size list, to restore it if necessary. Thus the user can undo a fix number of executed steps. The main objective of this pattern is creating a user-friendly interface.

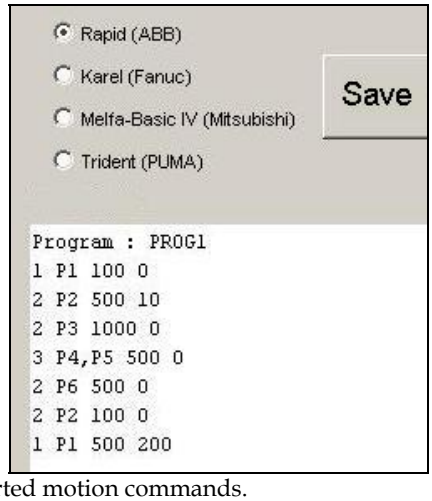


Fig. 6. Generic list of inserted motion commands.

The strategy is to create a collection that memorize the user steps, defining a structure that stores and restore the objects attributes, making it persistent in memory. This collection clones the list showed in figure 6, but keeping persistent the inserted/deleted motion commands to undo procedure.

### 4.1.6 Command

This pattern generates interface command decoupling actions of it causing events. The name of this pattern can give an idea of his functionality. Command pattern is necessary because the user is forced to execute system functionalities through data input interface objects(Menu, Button, CheckBox,...). The user interface generation tool has menus and buttons objects responsible for "commands" entered by the user. Action is responsibility of business classes. The user interface objects should not implement the explicit action, because only the business layer should know how to do it.
The solution adopted is showed in figure 7. All current interface instanced objects know only a reference to the method responsible to execute the user desired action. It decouples interface and business layer functions. The chosen strategy is; the interface objects only will know the system default input point (see Facade pattern, section 4.1.3). Thus interface objects invoked by the user, have to reference the method located in the business layer to execute actions, using the common input point, the charger class.

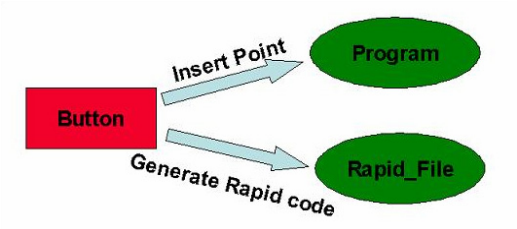


Fig. 7. Button commanding objects action.

### 4.1.7 Template Method

This pattern helps to define a skeleton algorithm. This skeleton will be used specializing subclasses that inherit the object abstract common model.

It allows a father class refinement from child classes realizing it reality showed in figure 8.

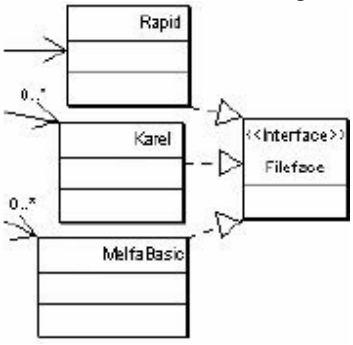


Fig. 8. Template method pattern used to persistence layer.

This technique consists in create a template to be specialized by child classes. It classes will materialize its behaviour overcharging father class methods through polymorphism. The TM direct consequence is increase code reuse. Although reflects in couple, caused by the implemented inheritance between the abstract class and child classes. This practice allied to dynamic binding is the bases to a framework building. The foundation of frameworks will be introduced in the next section.

### 4.2 Frameworks

It is a work layer where all knowledge about the predefined activity is encapsulated. Some characteristics of OO like dynamic binding, polymorphism, and inheritance make easy structures modelling it. These techniques were widely used to reusable, abstract, and specialized object creation. Make objects interact and communicate dynamically is the most important thing in the framework building. The framework concept was crucial to develop the robot code generator presented in this research, because it has to generate many different specific robot languages. Sometimes to implant the framework model is necessary redesign all the application to support its powerful interaction. Wherefore, the OODP and framework documentation is so important, prevents completely remodel the projected system if appear some located change demand, like insert a new brand robot language formulation.

### 4.3 RoBott

The RoBott Trajectory Generator (figure 9) was projected using OO concepts allied with UML specification to project analysis phase. Beyond it, all showed patterns concepts were considered in the software development phase, and applied using iterative incremental development process (Jacobson et al., 1998). The language used to develop this Off-line programming tool is Java Enterprise Edition (JEE) version 1.4.2_03, and the development environment is Oracle JDeveloper version 9.0.5.1(Build 1605).

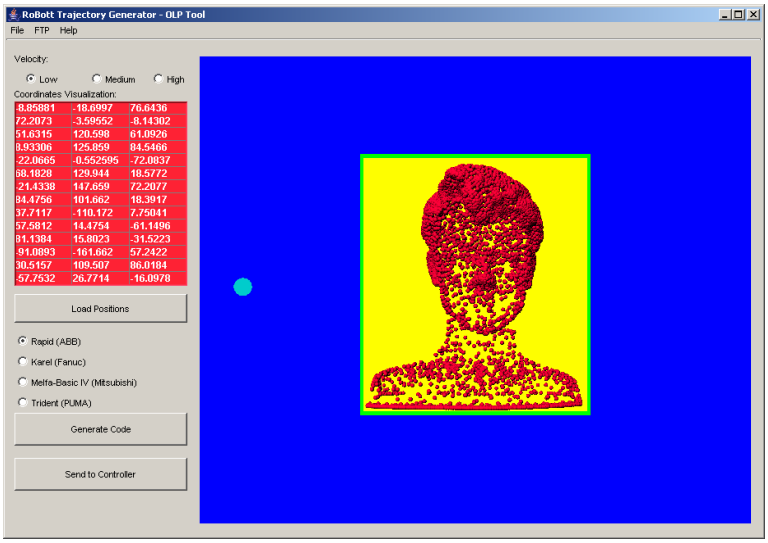


Fig. 9. RoBott OLP Tool.

## 5. Results

The tests were implemented using two different robot constructors, ABB and Mitsubishi industrial robots. The complete Off-line programming test was done using the ABB robot. First, it was necessary to setup references of ABB IRB140 working area (Murray, 1994), represented in figure 10 as a green rectangle. To get the robot real working area references is required to position the piece that will be worked. The blue point represents the robot base were is connected to the axis 1. After that, the neutral file has to be read to extract the outline of the piece, and it can be positioned as well using rotation and translation matrixes, respecting the working area bounds. After piece placement, the select layer (yellow rectangle) can be moved to touch the interesting points selecting the respective coordinates that will be used to generate the specific robot program. The file transfer to the ABB Robot Controller (IRC5) was done using FTP RoBott capability.

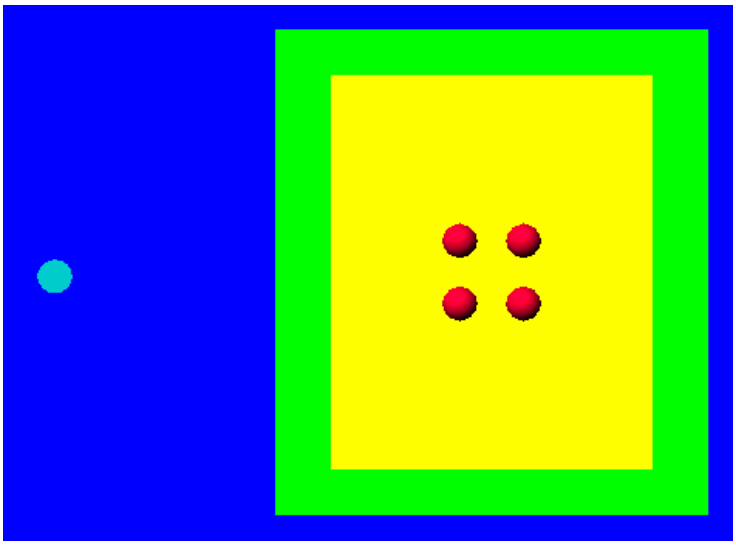

Fig. 10. Selected points from a top of a Cube placed in the ABB IRB140 work area.

After working area reference setup, cube boundary extraction from a STL file, and point cloud selection, it is possible generate a robot program (ABB, 2000) like the following example:

```
%%%
    VERSION:1
    LANGUAGE:ENGLISH
%%%
MODULE cubo
PERS robtarget Point0:=[[505.5,200.0,450.5], [0.043,0.384,0.921,0.021],…];
PERS robtarget Point1:=[[505.5,210.0,450.5], [0.043,0.384,0.921,0.021],…];
PERS robtarget Point2:=[[515.5,210.0,450.5], [0.043,0.384,0.921,0.021],…];
PERS robtarget Point3:=[[515.5,200.0,450.5],[0.043,0.384,0.921,0.021],…];
    PROC main()
     ConfL \Off;
```

```
        MoveL Point0,v10,fine,Tool0;
        MoveL Point1,v10,fine,Tool0;
        MoveL Point2,v10,fine,Tool0;
        MoveL Point3,v10,fine,Tool0;
        MoveL Point0,v10,fine,Tool0;
        ConfL \On;
    ENDPROC
ENDMODULE
```

Some tests were done also with Mitsubishi Move Master Industrial Robot, but restricted only to Melfa Basic IV(Mitsubishi, 2000) motion commands generation. The 3D coordinates input to Mitsubishi platform was done by teaching.

## 6. Conclusion

Actually, the robot programming is still a hard work (Wrn, 1998). Some causes are: difficulty of available equipment reserve to improve it, complex handling, and technologic approach demanded to learn it. This research demonstrates that the manufacturing cell integration can be accelerated, the communication between different platforms of robots can be optimized and costs with specialized people can be reduced.
The off-line programming method was created to minimise the integration cell time. But the contemporary off-line programming has not brought significative gains to the manufacture cell integration, also to reduce the robot programmer working hours. The contemporary programming tools to manufacture cells were projected without the necessary abstraction, to generalize the robot programming problem. The available tools present in robot kits, can program and interact only with its platform, files and libraries.
The demand grows for a unified tool that interact between different manufacturers solutions, turning easy robot programming to the companies. The development focus creates portable software, capable to write robot programs to different manufacturer's languages. Hence actually, it research is able to extract reference coordinates from a neutral project CAD file, generate motion commands to different robot platforms, through different input interfaces, running over different operating systems.
Summarizing the process:

- The points cloud is extracted from a STL file
- The coordinates references are transformed by scaling, translations and rotations matrixes
- The interesting points are selected according the planned task
- The proprietary robot programs are generated
- The program is sent to the robot controller and runs.

So, the effective Off-line programming was tested successfully.

## 7. Future Research

The abstraction used through robot programming interface development allows it application to program many robot tasks. Practicing a right point selection in the

points cloud, it is possible program tasks to welding, polishing, painting, assembly, inspection, every else manufacture area or robotized services. It is also possible insert new robot platforms with his respective proprietary languages.
The off-line robot programming can be used without restrictions to reduce manufacture cell integration time. It is also useful in areas where the contingent of workers was reduced by repeatable and insalubrious features tasks.
To future works I suggest specialized algorithms implementation using:

- Graphs theory to points selection refinement, task and path planning to:
  - o Painting and Polishing using Smoothing paths
  - o Welding using Waving paths
  - o Or assembly and inspection using Template Matching
- Finite Elements theory (FEM) to calculate the force applied in the tool, working over a piece composed by some known material, like the sculpture process for example.
- Real time module receiving feedback from an extensometer network mounted between the tool and the robot flange, correcting in real time the tool attack angle and the tension applied to robot servo motors.

Acknowledgements. This research was carried out in the context of a MSc Degree in Industrial Electronic Engineering. Project Supported by the Programme ALBAN, the European Union Programme of High level Scholarships for Latin America, scholarship no. E04M033540BR. Thanks to CIMATEC - Technology and Manufactory Integrated Centre of SENAI-BA, Brazil, and Electronic Engineering Department (DEI), University of Minho, Portugal.

## 8. References

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Longman, ISBN 0-201-63361-2, 1st edition.

Jacobson, I., Booch, G. & Rumbaugh, J.(1998). *The Unified Software Development Process*, Addison Wesley Longman, 1998.

Lee, D. M. A. & ElMaraghy, W. H. (1990). ROBOSIM: a CAD-based off-line programming and analysis system for robotic manipulators. *Computer-Aided Engineering Journal,* Vol. 7, No. 5, (October 1990) page numbers (141-148), ISSN: 0263-9327

Murray, R. M. (1994). A Mathemetival Introduction to Robotic Manipulation. *CRC Press, Florida*, 1st edition.

Owen, J. (1994). STEP : An Introduction. *Information Geometers Ltd,* (April 1994).

Schenck, D. A. & Wilson, P. R. (1994). Information Modeling: The EXPRESS way. *Oxford University Press,* (1994)

Smith, B. M., Rinaudot, G. R., Reed, K. A. & Wright, T. (1988). Initial Graphics Exchange Specification (IGES). *Version 4.0. IGES/PDES Organization,* (June 1988), Gaithersburg, MD.

Vuoskoski, J. (1996). Exchange of Product Data between CAD systems and a Physics Simulation Program. *Tampere University of Technology, Pori Unit,* (April 1996) page numbers (19-23).

Wrn, H. (1998). Automatic off-line programming and motion planning for industrial robots. *In ISR98, 29th International Symposium on Robotics 1998*. ISR Press.

ABB (2000). ABB Robotics AB, *RAPID Reference Manual* – Características Gerais, Västeras.2000, 172p (3HAC 5780-1).

IGES 1 (1980), NBS, US. Department of Commerce, Washington DC 20234, 1980.

Mitsubishi (2000). Mitsubishi Electronics Corporation. *Mitsubishi Industrial Robot – Instruction Manual, Detailed explanations of functions and operations*. Nagoya, 2000. 152p (BFP-A5992-C).