Advanced Engineering Platform for Industrial Development

M. A. González-Palacios

Universidad de Guanajuato, Campus Irapuato-Salamanca, División Ingenierías
Carretera Salamanca-Valle de Santiago km 3.5+1.8, Salamanca, Gto., México
maxg@ugto.mx

ABSTRACT
This paper introduces a full description of a software development platform involving libraries that allow the creation of software packages focused not only on industrial applications, but also on applications where design, modeling and/or on-line simulation are required. The flexibility of the main classes simplifies the generation of modules that constitute an application developed with this platform. Furthermore, any custom application starting from scratch contains by default a set of functions that facilitates the developer firstly to build the graphical environment with capabilities to interact with the pointing device, and secondly, to accomplish machinery control tasks while communicating with input/output components; such is the case of digital-analog cards or modules connected remotely. Besides, any fully developed application can be considered as a platform to generate another with a higher level of specialization. Several applications built with this platform are reported here as case studies.

Keywords: Software development, mechanical-system simulation, modeling, object-oriented programming, machinery process control.

RESUMEN
Este artículo introduce una descripción completa de una plataforma de desarrollo que involucra bibliotecas que permiten la creación de paquetes de software dedicados no solo a aplicaciones industriales, sino también a aquellas aplicaciones donde el diseño, modelado y/o simulación en línea son requeridas. La flexibilidad de las clases principales simplifica la generación de módulos que constituyen una aplicación desarrollada con esta plataforma. Más aún, cualquier aplicación que empieza a desarrollarse desde las bases de esta plataforma de desarrollo contiene de forma predeterminada un conjunto de funciones que facilita al desarrollador, en primer lugar, construir un ambiente gráfico con capacidad de interactuar con el ratón, y en segundo, realizar tareas de control de maquinaria al comunicarse con dispositivos de entrara/salida; tal es el caso de tarjetas digital/analógicas o módulos conectados de forma remota. Además, cualquier aplicación completamente desarrollada, puede ser considerada como una plataforma para generar otra aplicación con mayor nivel de especialización. Varias aplicaciones construidas con esta plataforma son reportadas aquí como casos de estudio.

1. Introduction

The constant need to implement automation algorithms to control mechanical systems while executing industrial R&D projects, and the continuous needs to implement applications that help understanding the mechanical devices in engineering education, motivated the creation of an interactive platform which combines these control algorithms with the tools needed to build 3D models [1-10] and applications using MFC (Microsoft Foundation Classes) [11-12]. Furthermore, the 3D modeling tools were implemented gradually while several software packages for design and simulation of mechanical systems were introduced in the past [13-16].

Written in Visual Studio® C++ [17-20], and structured with the object-oriented programming concept [21-23], these applications together with other related with industrial control and automation are continuously updated and integrated into the library set, that led to the platform introduced here, which is called *ADvanced Engineering platForm for Industrial Development,* and for short will be referred as ADEFID. Although ADEFID was originally created to develop industrial applications, the integration of the OpenGL libraries extended the applications to research and training. Furthermore, this paper represents the unification

and extension of several years of work and a summary of a modern textbook on mechanical design in which the author is working on.

There are a number of professional powerful software packages with wide capabilities to perform the applications related to those that have been implemented with the ADEFID platform, such as Simulink [24], an environment for multidomain simulation and model-based design for dynamic and embedded systems; Matlab [25] and Maple [26], which provide a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numerical computation; Pro/Engineer [27], SolidWorks [28] and Autodesk Inventor [29], integrated 3D CAD/CAM/CAE systems that provide solutions for mechanical design, product simulation, tooling creation, etc. Nevertheless, since ADEFID applications are either specific or specialized, the investment on a dedicated software package might not be justifiable or the solution might not fully satisfy the user's needs. Therefore, these arguments open the door to ADEFID since its main objective is to provide a flexible platform from which the developer is able to build those types of applications. Furthermore, any ADEFID application can be considered as a platform to generate another application with a higher level of specialization.

In order to understand the structure of an application derived from an ADEFID project, the relationship with MFC and ADEFID libraries is introduced in Fig. 1. The developer is able to create custom applications starting from scratch (Application A), or from an already developed ADEFID application or applications (Application B, …, Application n), with the aim to produce, as mentioned above, a higher level of specialization.

The organization of this paper is as follows: Section 2 is dedicated to present information on the most relevant libraries and the description of their related classes. Then, the main features of an ADEFID project to generate an application are introduced in Section 3, while Section 4 is reserved to present sample code regarding the CAD implemented in ADEFID. Section 5 includes a set of case-studies of applications already developed under this platform.

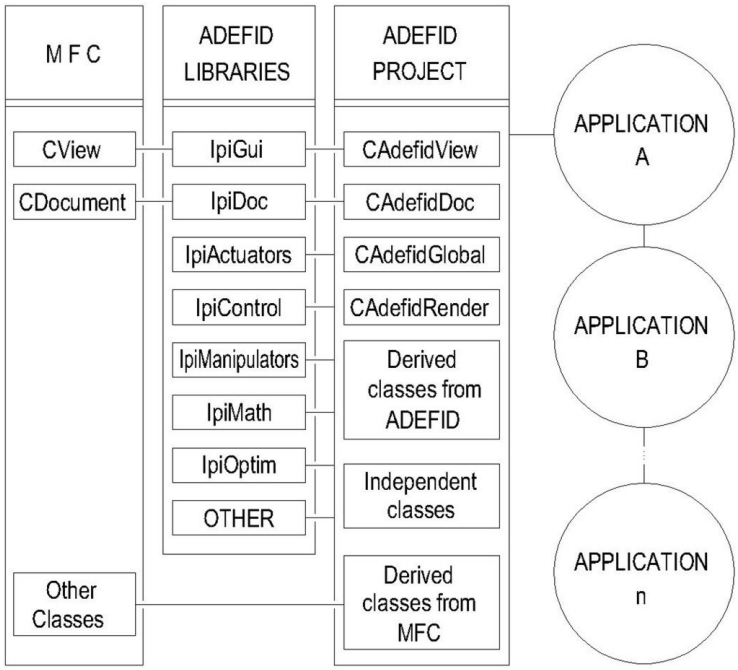

Figure 1. Structure of an ADEFID application.

2. Adefid libraries

As mentioned in the previous section, besides the support provided by the MFC and OpenGL libraries, an ADEFID project is supported by a number of ADEFID libraries, which have been created based on the needs that have been arisen during the development of previous software packages, namely, USyCaMs, SixPaq, SVIS, etc., and those integrated libraries have been successfully implemented in the software packages that are discussed as case-studies in this paper. In the subsections below, a description of the representative libraries is included. Needless to say, every library is continuously updated and new classes are being implemented as the needs arise during the implementation of new applications.

2.1 IpiActuators

This library comprises classes of objects intended to drive mechanical components either with translational or rotational motions. In addition, they have member functions allowing the control of the motion program of devices such as motors, or member functions to retrieve pulses from sensors, such as encoders, see Table 1 for the description of the most relevant classes.

2.2 IpiControl

This library is devoted to develop those classes focused on applications where the communication with control devices is required through embedded devices or through Ethernet-connected systems, the only constrain being that they should have compatible drivers to both the operating system and the programming language. The most representative classes are listed in Table 2.

2.3 IpiGUI

Those classes whose main objective is to support the graphical interface, as well as to define objects being frequently used to create mechanical components are concentrated in this library. The most relevant are listed in Table 3.

Class Name	Description
CIncEncoder	It is possible to reset, read the pulses and interpret the direction of rotation.
CLinearMotor	The motor starts and stops according to a given period and a given motion program (smooth start and smooth stop).
CRotaryActuator	
CStepMotor	

Table 1. IpiActuator Library Classes.

Class Name	Description
CConveyorO22	This is a sample class derived from CMachine that includes the algorithm to control a single digital input as well as a single digital output.
CIOBrain CIOCard	A set of functions and/or variables from which it is possible to communicate with embedded I/O cards and/or Opto22 brain components.
CMachine CMachineO22	Fundamental functions, such as reading a sensor signal or writing a digital or analog output. It also keeps track of the custom defined timers.

Table 2. IpiControl Library Classes.

Class Name	Description
C3DPlot	Given a function *f*(*x*,*y*), it is possible to plot it in a user-defined range.
CHelix	Contains functions to create a helicoidal segment by means of member variables. The helix can be a line or a polygonal shape extruded.
CIpiGLView	This class involves all window messages required to interact with the mouse as well as all the setup and initialization for OpenGL.
CIpiSModel	Contains a set of predefined models most commonly used in material-handling processes.
CPlanarPlot	Similar to C3DPlot, the difference being the function, which in this case is a single-variable function *f*(*x*).

Table 3. IpiGUI Library Classes.

2.4 IpiMath

This library was mainly created to facilitate the symbolic definition of mathematical functions of one or two independent variables to develop several applications. In the case studies, the packages named OptimPlot2d and OptimPlot3D use this library to interpret a function given as a string of characters to be analyzed and plotted.

2.5 IpiOptim

This library is intended to assist those applications requiring functions whose objective is to evaluate critical points while analyzing a mathematical function. Those points can be roots, minima or maxima, as listed in Table 4. Examples of software packages that apply this library are OptimPlot2D and OptimPlot3D.

Class Name	Description
CBisection	Classes applied to obtain roots of a single-variable function *f(x)*.
CFalsePosition	
CFixedPoint	
CNewtonRaphson	
CSecant	
CGoldenSection	Class applied to find critical points of a single-variable function.
CDavFlePow	Methods applied to get minimum values of a *n*-variable function.
CDirCon	
CModMar	
CSteepestDescent	
COptimFunctions	A base class with virtual functions to define those required to analyze critical points

Table 4. IpiOptim Library Classes.

2.6 IpiManipulators

This library concentrates those classes related with the study of robots, the main areas being the direct and inverse kinematics of serial manipulators. The most relevant classes are described in Table 5. An application that makes an intensive use of these classes whose name is ADRS is discussed as a case study.

3. An adefid application

A discussion is included in this section on how an ADEFID project is organized and also a discussion on the main default features a developer can obtain by means of this platform.

An ADEFID project is derived from a MFC application template and has a Document/View architecture support. As shown in Figure 1, CAdefidView is derived from CIpiGLView (component of IpiGUI library), while CAdefidDoc, from CIpiGLDoc (component of IpiDoc library).

Besides CAdefidView and CAdefidDoc, there are other two classes that are standard components of an ADEFID project as well, namely, CAdefidGlobal and CAdefidRender. The former is derived from CMachine; it was created to support control processes, when global system components are required such as safety inputs like emergency stop buttons. Whereas CAdefidRender was created to keep the rendering properties of an ADEFID project; it contains two virtual functions, namely, SetupScene() and RenderUScene(), which override those originally defined in CIpiGLView.

SetupScene() was created to define the components of the scene; therefore, the function is called when the application begins, while RenderUScene() was created to update the scene if any transformation (rotation, translation, etc.) takes place.

3.1 The Main Process

The main process of an ADEFID project resides in CAdefidDoc, and the process is synthesized as indicated in the flowchart in Figure 2. When the application is initialized, a message box appears indicating that the system is connecting to embedded devices (I/O cards) and/or to remote I/O systems (Ethernet connection). Once the connection is established, the next step is the initialization of the objects derived from the CMachine Class. The message box is a component of the CMainInitDlg class where, through its member function Start(), the developer is able to customize both the connection to the I/O devices and the machines initialization. A machine can be a physical device, as those defined in SVIS or MetalCoating applications, or a virtual device, as those defined in ADRS or Mechanism-O applications. These applications are described in Subsection 3.2.

Class Name	Description
CLinearPath	Given the end poses, it defines the poses of a manipulator end-effector along a straight line.
Clink	Contains all properties to provide the graphical representation of a link.
CRobot	Base class from which CRoboKin and CRoboDyn are derived.
CRoboKin	All properties related to robot kinematics such as direct and inverse kinematics.
CRoboDyn	This class is intended to keep all robot properties regarding its dynamics.

Table 5. IpiManipulators Library Classes.

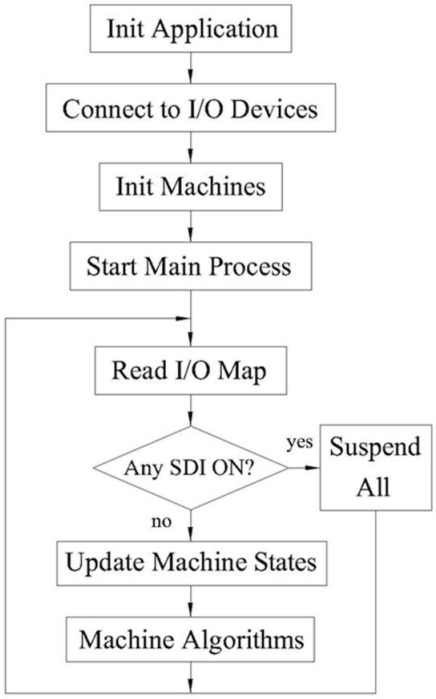

Figure 2. Main ADEFID's process flowchart.

The *Start Main Process* is indeed an infinite loop which ends only if the user closes the application, this way allowing an application to run as stand-alone, mostly when it is an industrial application. At every cycle, the I/O map is read and, if any of the *safety device inputs* (SDI) is activated, the system is set to a suspend state and no further action is taken. On the contrary, if all the SDIs remain deactivated, the state of every machine is evaluated and updated according to their corresponding I/Os state. Then, every machine is accessed through its own Algorithm() member function.

3.2 The Graphical Interface

In most cases, there are two levels of interaction with the rendering area. One level is where the user can rotate, move, translate, zoom-in or zoom-out the scene. In this case these features are standard operations that can be handled with the mouse; the code of these window messages is written in CIpiGLView class, and the user can access the setting parameters through the dialog named Render Dialog. The other level is considered when the user wishes to modify the topology of a model, for which purpose a custom dialog should be created. Nevertheless, in case the scenery´s parameters are required to change, they can also be modified interactively with the mouse according to the instructions provided in Table 6.

4. On Computer Aided Design

The objects rendered in ADEFID are built with the application of OpenGL functions. Most of them are generated from basic primitives such as cylinders, boxes, faces, etc., but there are other objects requiring the use of more complex primitives. Nevertheless, all geometric primitives are described in terms of their vertices [1]. Thus, in this section is presented a discussion on how these vertices are handled to generate those complex primitives. Consequently, in the IpiGUI library a class called CPrimitives is dedicated to write this type of primitives.

Mouse button	Action
Left	Rotation
Right	Translation
Left and Right	Zoom in and zoom out

Table 6. Interacting with the Render Window.

To illustrate the logical structure of an ADEFID primitive, two functions are discussed, namely, DrawLineSegment() and DrawSurfSegment(). The former, is applied to generate any three-dimensional curve, whereas the latter, to generate a three-dimensional surface. These two functions are written in CPrimitives. In this class, three more virtual functions are defined: GetCurvePoint(), GetSurfPoint() and GetNormal(). Since these are public virtual functions, the user can redefine them in a derived class to establish the mathematical behaviour of either the curve or the surface to be generated.

The DrawLineSegment() pseudo-code is shown in Table 7, where *u* varies from 0 to 1 and m represents the number of points along the curve segment required. Likewise, Table 8 shows the DrawSurfSegment() pseudo-code, where both *u* and *v* range from 0 to 1 and the product *mn* defines the number of points of the surface.

```
beginfun CPrimitives::DrawLineSegment( u, m )
    glBegin(GL_LINE_STRIP)
        For i = 0 till m do
            p ← GetCurvePoint(iu / m)
            glVertex3dv( p )
            i ← i + 1
        enddo
    glEnd()
endfun
```

Table 7. Pseudo-code to generate a curvilinear segment.

```
beginfun CPrimitives::DrawSurfSegment( u, m, v, n )
    For j = 0 till j = n − 1 do
        glBegin(GL_QUAD_STRIP)
            For i = 0 till i = m do
                j ← j + 1
                For k = 1 till k = 2 do
                    n ← GetNormal(iu / m, jv / n)
                    p ← GetSurfPoint(iu / m, jv / n)
                    glNormal3fv( n )
                    glVertex3dv( p )
                    k ← k + 1
                    j ← j − 1
                enddo
                i ← i + 1
                j ← j + 1
            enddo
        glEnd()
        j ← j + 1
    enddo
endfun
```

Table 8. Pseudo-code to generate a surface.

One can verify in the pseudo-code in Table 8 that the inner most loop is applied to generate two points (**p**) with their corresponding normal unit vectors (**n**). Thus, with the variation of *i*, strips with *m* faces are built with the aid of $2(m+1)$ generated points, and finally, with the variation of *j*, *n* strips are constructed to complete the surface. It is important to note that the main structure of both DrawLineSegment() and DrawSurfSegment() are visualization techniques included directly in the graphics library, i.e., OpenGL´s evaluator facility [1,2]. Nevertheless, what makes the difference here is the call to GetCurvePoint() on DrawLineSegment() and the calls to GetNormal() and GetSurfPoint() in GetSurfSegment().

Furthermore, a representative example to create objects like springs or screws is presented below. For this purpose, CHelix is derived from CPrimitives, having the pitch (*h*), the radius of the base cylinder (*R*), the radius of the extruded polygon (*r*), and the position vector of the reference point (**c**), as global variables. These variables are illustrated in Figures 3 and 4.

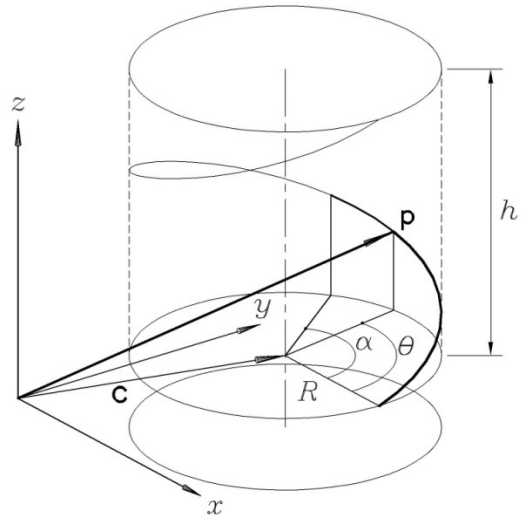

Figure 3. Parameters to define a segment of a helix.

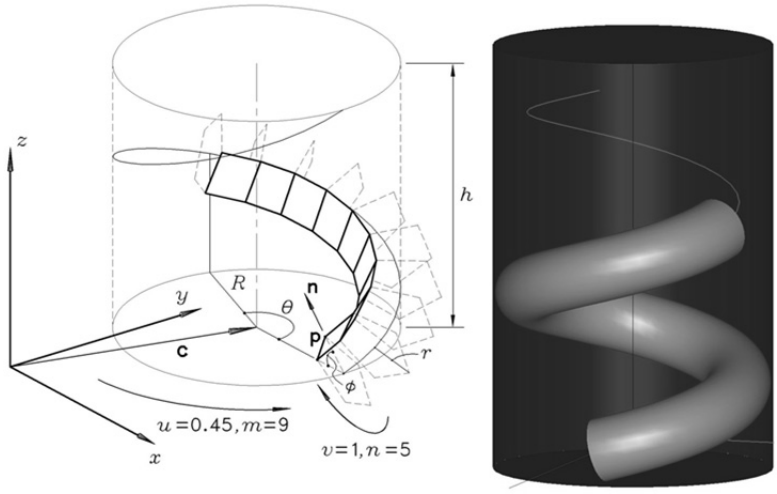

Figure 4. Left: Parameters to define a segment of an extruded polygonal surface in a helix. Right: A helix and an extruded surface rendered.

As mentioned previously, in order to define the properties of helical curves and extruded helical surfaces, GetCurvePoint(), GetSurfPoint() and GetNormal() are redefined in CHelix. The pseudo-code of these functions is shown in Table 9.

beginfun CHelix::GetCurvePoint(u)

$$\theta \leftarrow 2u\pi$$

$$\mathbf{p} \leftarrow \begin{bmatrix} R\cos\theta \\ R\sin\theta \\ h\theta/2\pi \end{bmatrix} + \mathbf{c}$$

$$\texttt{GetPoint} \leftarrow \mathbf{p}$$

endfun

beginfun CHelix::GetSurfPoint(u, v)

$$\mathbf{p} \leftarrow \texttt{GetCurvePoint}(u) + r\texttt{GetNormal}(u,v)$$

$$\texttt{GetPoint} \leftarrow \mathbf{p}$$

endfun

beginfun CHelix::GetNormal(u, v)

$$b \leftarrow h/2\pi$$

$$f \leftarrow \sqrt{b^2 + R^2}$$

$$\theta \leftarrow 2u\pi$$

$$\phi \leftarrow 2v\pi$$

$$\mathbf{n} \leftarrow \begin{bmatrix} \cos\theta\cos\phi + (b/f)\sin\theta\sin\phi \\ \sin\theta\cos\phi - (b/f)\cos\theta\sin\phi \\ (R/f)\sin\phi \end{bmatrix}$$

$$\texttt{GetNormal} \leftarrow \mathbf{n}$$

endfun

Table 9. Auxiliary pseudo-code to generate helical curves and extruded surfaces.

For the case presented in the left side of Figure 4, the loop with $i = 0$ has finished and one of the five strips, with 9 faces each, is displayed. Moreover, the right side of the same figure shows a rendered image of a helix and an extruded surface. Both objects have the following variable values: $h = 1.7$, $R = 1.0$ and $r = 0.3$. Additionally, they are called, respectively, with the following arguments:

DrawLineSegment(2,60) and
DrawSurfSegment(1.2, 60, 1.0, 30).

Note that for this case, due to the nature of this type of objects, *u* can be greater than 1.

With variations of the DrawSurfSegment() function, objects like springs can be generateded, as the examples illustrated in Fig. 5 where three nested springs with different parameters and shapes are rendered. With an appropiate pitch, it is also possible to design screws, as shown in Figure 6, where the helical shape is built on top of a cylindrical surface.

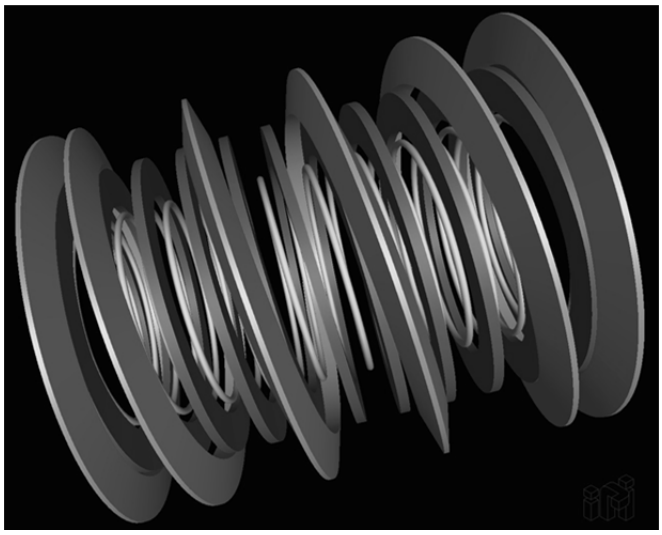

Figure 5. Three nested springs generated with variations in the application of the DrawSurfSegment() function.

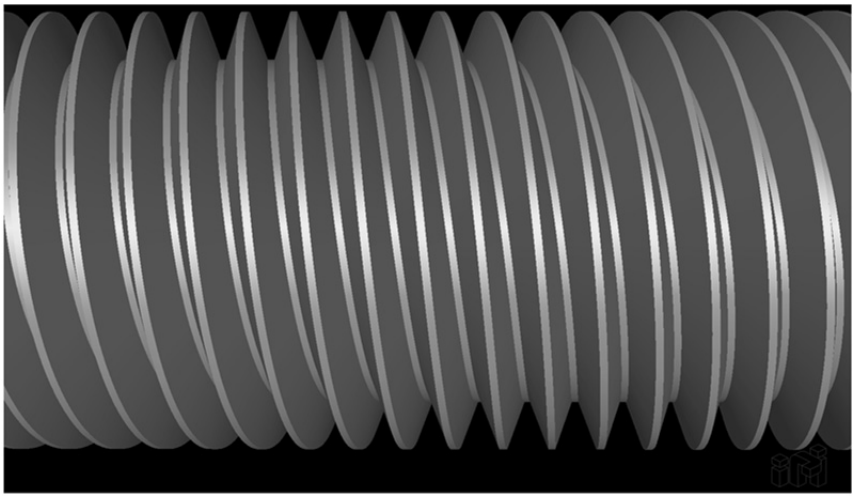

Figure 6. A screw generated with variations in the application of the DrawSurfSegment() function.

5. Case Studies

In this section, representative applications developed with ADEFID are described, some of which were developed for training and research purposes, the case of *Mechanism-O*, *Vibrato*, *OptimPlot2D*, *OptimPlot3D*, and *ADRS*; others are oriented to industrial applications, like *MetalCoating* and *SVIS*. There are video files presenting a simulation for each case study. To have access to these files, the reader may contact the author for further information.

5.1 Mechanism-O

The experience in the classroom while teaching courses of mechanisms for engineering students as well as the success in generating innovative mechanical devices using software packages already developed, e.g., USyCaMs, from which Speed-o-Cam was created [14, 32-35], motivated the development of this package. Figure 7 shows a still image of a simulation of the well-known Whitworth mechanism, with the corresponding dialog box where the user is able to change interactively the parameters indicated, and to visualize an instantaneous updating in the graphics, including the plots for the position, velocity and acceleration of the slider while the mechanism is in motion. This way, the user is able to visualize in real time how the kinematic behavior is affected by a modification of any of the link lengths, thereby giving an immediate picture of the mechanism behavior during the design process. Likewise, Fig. 8 shows a snapshot of a four-bar linkage while plotting the trajectory of a coupler point. In this case, if the user changes any of the parameters interactively, the coupler curve is updated simultaneously.

5.2 Vibrato

Vibrato is dedicated to the study of vibrations of mechanical systems. It was created as an auxiliary tool for the theory of vibration courses. Although this package is still under development, some systems can be simulated such as lumped-parameter systems with one or two degree of freedom [36], or continuous systems like strings [37]. Figure 9 illustrates a one-degree-of-freedom system.

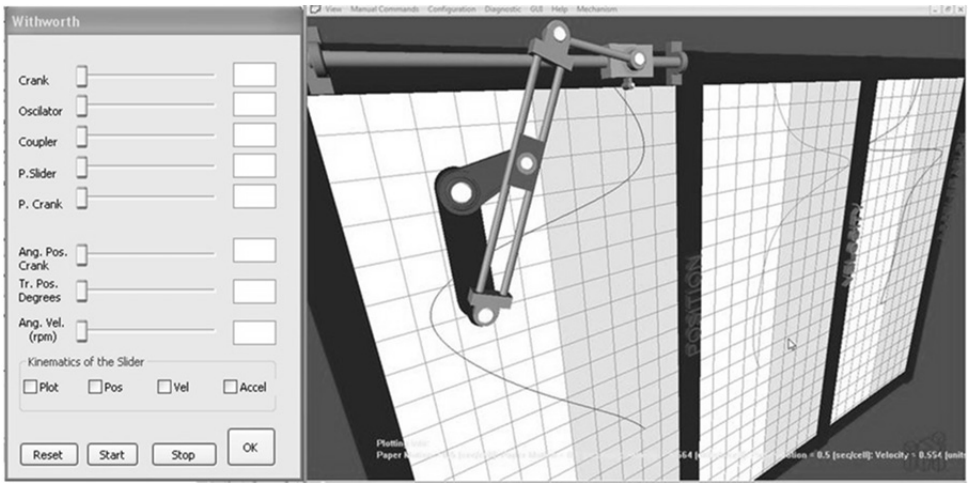

Figure 7. Simulation of a Whitworth mechanism.

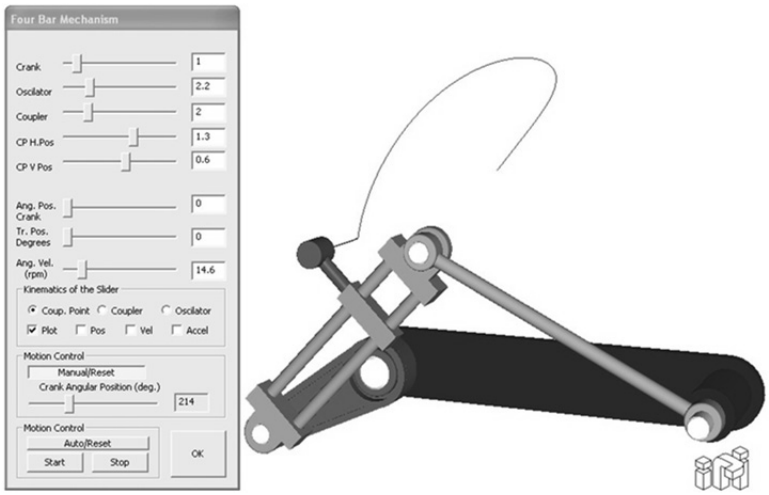

Figure 8. A four-bar mechanism while plotting a coupler curve.

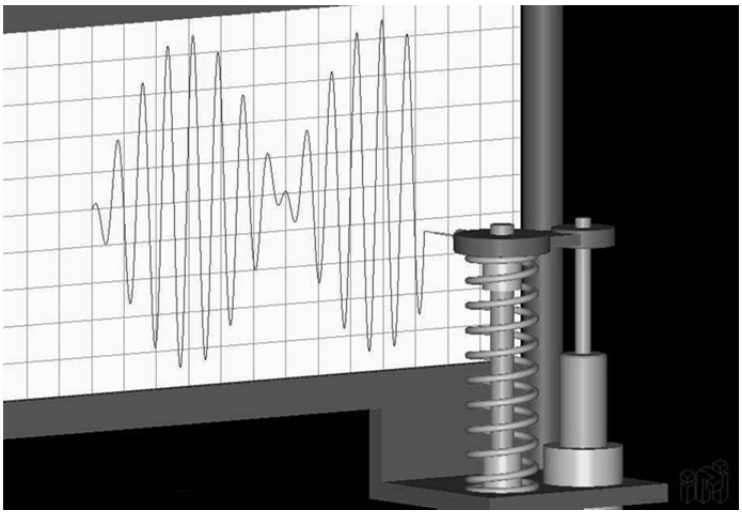

Figure 9. Vibrato: One-degree-of-freedom system.

5.3 OptimPlot2D

With this software, the user inputs any function of the type $y = f(x)$, by typing it as a string of characters in a command line. Then, the curve is plotted and the user is able to navigate on it by moving the cursor horizontally; then, a small sphere moves along the curve pointing the $f(x)$ value of the corresponding x position of the cursor. It is possible to define the range of the plotting area, so that the user can zoom in the area of interest, see Figure 10 where a dialog box shows the plotting settings. Besides, by using numerical methods, it is possible to obtain critical points interactively such as roots, maxima and minima. Details on this application are given in [38].

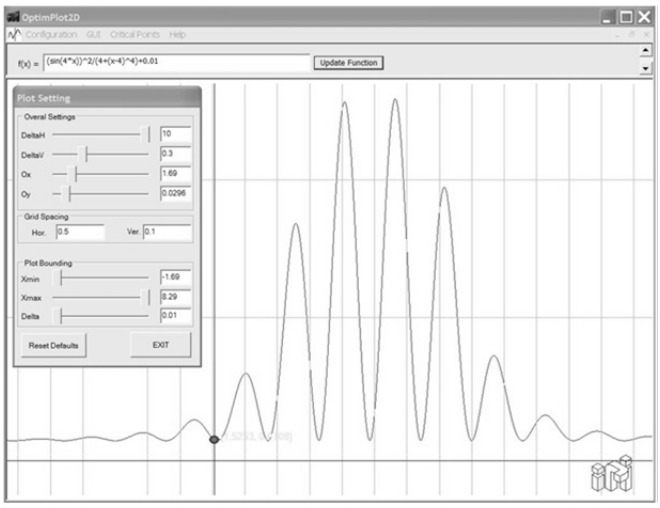

Figure 10. Still Frame from OptimPlot2D.

5.4 OptimPlot3D

This package is similar to OptimPlot2D but in this case the function is of the type $z = f(x, y)$. The surface is plotted within the range defined by the user. By moving the pointing device on the screen, x and y values are defined and the $f(x, y)$ value is calculated, and a small sphere is placed on the surface to indicate the z value. Figure 11 shows a still image during a navigation process on the surface. Furthermore, by using iterative techniques, it is possible to obtain, interactively, points of interest for optimization purposes. A full description of this application was introduced in [39]. Most of the OptimPlot3D features can be achieved by packages like Matlab [25] or Maple [26], such as zoom-in and -out and rotate, or even change rendering properties [30, 31], however, and to the best of knowledge, properties like the interactive navigation on the surface to evaluate in line the z-value of the function as the user moves the x-y point with the mouse or the interactive evaluation of a critical point (minimum or maximum) as the user approaches it to provide with the auxiliary mouse button the initial point to trigger any of the numerical methods listed in Table 4, are OptimPlot3D original features.

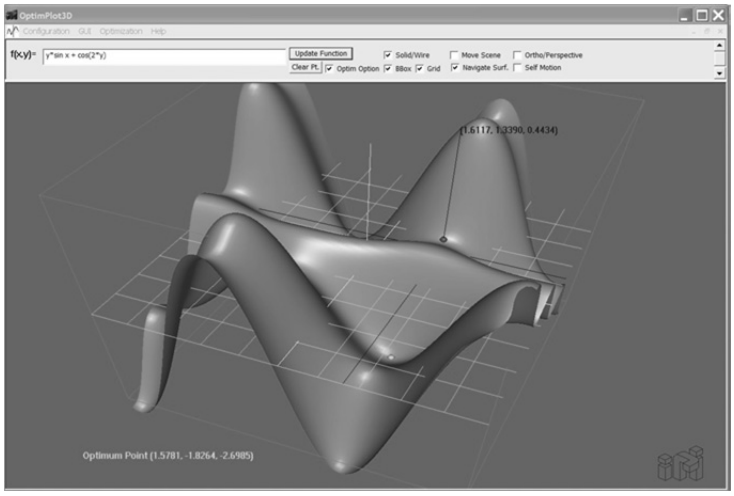

Figure 11. Still Frame from OptimPlot2D.

5.5 ADRS

The name of this package stands for *Architecture Design and Robot Simulation* (ADRS). This package was created as an auxiliary tool to help the designer of serial manipulators. Although the package is still under development, its current stage provides the user with the tools to design interactively a desired robotic architecture; in other words, design parameters like length, offset and twist angle of any link can be modified on line while the model is continuously updated. Thanks to the GUI, when it is in the *Selection* option, the user is able to select a link with the pointing device, while it is highlighted, to obtain the corresponding dialog box and interact with its design parameters at the same time the modifications are updated in the model.

ADRS was developed with simulation features like path tracking, path generation, palletizing, direct and inverse kinematics. Figure 12 illustrates the case of path tracking.

According to Fig. 1, this is an example of an *Application B*, since it has been developed and improved with innovative features compared with its base Application *A* [40]. For this matter, with few changes in the Visual Studio's project definition properties of Application B, the user departs from this platform to create a new application, for instance, a pick & place operation, which is now considered an *Application C*, as shown in Figure 13, where the user does not need to rewrite the code for the manipulator, being this an object of the CManipulator class, which is in turn derived from the classes of the IpiManipulators Library.

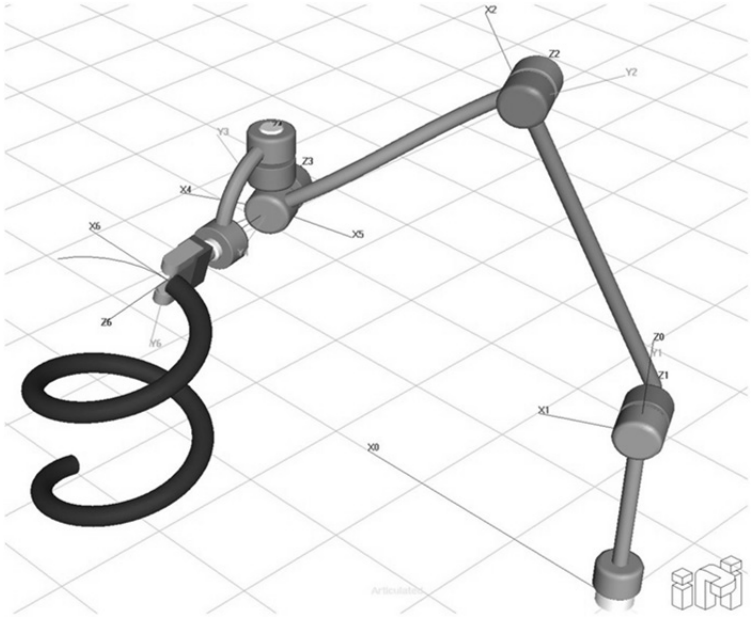

Figure 12. A path tracking application (*Application B*, see Fig.1).

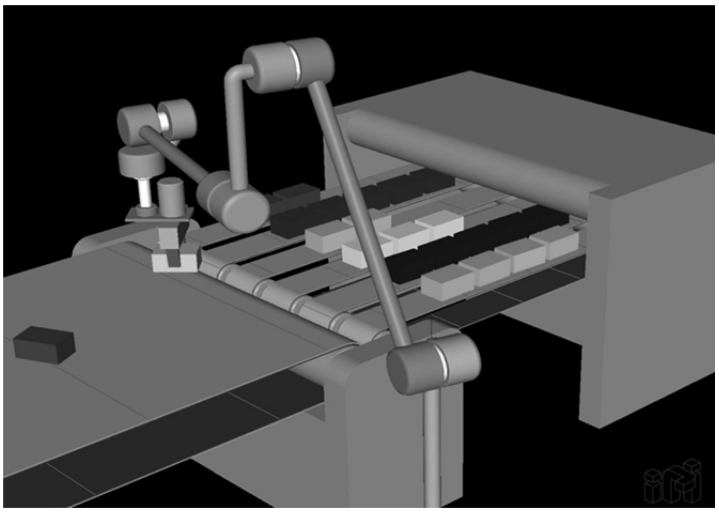

Figure 13. ADRS: A pick & place operation (*Application C*, see Fig.1).

5.6 SVIS

As mentioned before, this control program was created for an industrial application which has been successfully implemented at Placage Unique, Inc. (Canada). The main objective of this application is to keep in operation a system dedicated to the production of veneer. SVIS stands for Spliced Veneer Integrated System and it is an example in which all devices shown in Fig. 14 are defined each as independent machines. Then, the code to synchronize their operation with the overall process is written in the block labelled Update Machine State, as indicated in Fig. 2. The actual layout of Placage Unique, Inc., the owner of this application, is shown in Figure 15.

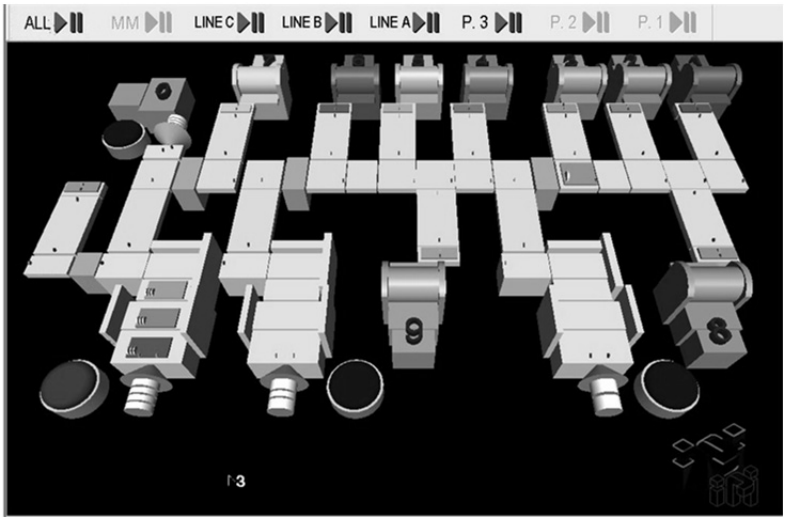

Figure 14. SVIS: An industrial application (Courtesy of Placage Unique, Inc.).

Figure 15. Layout of Placage Unique, Inc. (Canada).

6. Conclusions

In this paper, a set of continuously growing libraries devoted to the development of software applications, not only for industrial, but also for training and research purposes, have been introduced as a software development platform named ADEFID (*Advanced Engineering platForm for Industrial Development*). The flexibility of their classes allows the generation of new modules as well as custom-tailored applications to satisfy industrial, research or training needs. These features have been pointed out within ADEFID applications introduced here as case studies.

Acknowledgements

The author acknowledges the support from SNI, (Sistema Nacional de Investigadores), Mexico.

References

[1] Shreiner D., The Khronos OpenGL ARB Working Group, OpenGL Programming Guide, 7 ed. Adison Wesley, 2009.

[2] Shreiner D., OpenGL Reference Manual, 4 ed. Adison Wesley. 2007.

[3] Fosner R., OpenGL Programming for Windows 95 and Windows NT. Adison Wesley, 1999.

[4] Angel E., Shreiner D., Teaching a shader-based introduction to computer graphics, IEEE Computer Graphics and Applications, Vol. 31, No. 2, 2011, pp. 9-13.

[5] Angel E., Interactive Computer Graphics: A Top-down Approach using OpenGL, 3 ed. Adison Wesley, 2003.

[6] Kilian M., Mitra N., Pottmann H., Geometric Modeling in Shape Space, ACM Trans. Graphics, Vol. 26, No. 3, Article 64, 2007, 8 pages.

[7] Schreiner J., Asirvatham A., Praun E., Hoppe H., Inter-surface mapping, ACM Trans. Graphics, Vol. 23, No. 3, pp. 870-877.

[8] Randi Rost, OpenGL Shading Language, Second Edition, Addison-Wesley, 2006.

[9] Georgli J., Westermann R., A multigrid framework for real-time simulation of deformable bodies, Computers & Graphics, Vol. 30, No. 3, 2006, pp. 408-415.

[10] Diaz-Gutierrez P., Bhushan A., Gopi M., Pajarola R., Single-strips for fast interactive rendering, The Visual Computer, Vol. 22, No. 6, 2006, pp. 372-386.

[11] Blaszczak M., Professional MFC with Visual C++ 6. Wrox Press, 1999.

[12] Murray W.H., C.H. Pappas, MFC Programming in C++ with the Standard Template Libraries. Prentice Hall PTR, 2000.

[13] González-Palacios M.A., Angeles J., Cam Synthesis. Springer, 1993.

[14] González-Palacios M.A., Angeles J., USyCaMs: A Software Package for the Interactive Synthesis of Cam Mechanisms, 1st IDMME Conference, Nantes, France, Vol. 1, 1996, pp. 485-494

[15] González-Palacios M.A., An Algorithm for the Synthesis of Bevel Gears. Proc. 9th IFToMM World Congress on the Theory of Machines and Mechanisms, Milan, Italy, Vol. 1, 1995, pp. 570-574

[16] González-Palacios M.A., Angeles J., SIXPAQ: A Comprehensive Software Package for the Analysis and Synthesis of Six-Bar Dwell Linkages. ASME International Computers in Engineering Conference, Santa Clara, CA, Vol. 1, 1991, pp. 301-308

[17] Stroustrup B., The C++ Programming Language, 3 ed. Adison Wesley, 1997.

[18] Horton I., Beginning Visual C++ 2010. Wrox Press, 2010.

[19] Walnun K., C++ Master Reference. IDG Books Worldwide Inc., 1999.

[20] Leinecker R.C, T. Archer, Visual C++ 6 Bible. IDG Books Worldwide Inc., 1998.

[21] Lafore R., Object-Oriented Programming in C++, 3rd edn. Sams Publishing, 1999.

[22] Kecskeméthy A., Lange C., Grabner G., Object-Oriented Modeling of Multibody Dynamics Including Impacts, European Conference on Computational Mechanics, Cracow, Poland, 2001, pp. 1-28.

[23] Tändl M., Stark T., Erol N., Löer F., Kecskeméthy A., An object-oriented approach to simulating human gait motion based on motion tracking, International Journal of Applied Mathematics and Computer Science, Vol. 19, No. 3, 2009, pp. 469-483.

[24] Simulink (2009) http://www.mathworks.com/products/simulink/

[25] Mathlab 7.9 (2009) http://www.mathworks.com/products/matlab/

[26] Maple 13 (2009) http://www.maplesoft.com/

[27] Pro/Engineer (2009) http://www.ptc.com/products/proengineer/

[28] SolidWorks 2010 (2009) http://www.solidworkslaunch.com/

[29] Autodesk Inventor (2009) http://usa.autodesk.com

[30] Runiter Company, Graphing Calculator 3D, (2011) http://www.calculator.runiter.com/graphing-calculator.

[31] Brown A., Torrey Pines H.S., Three Dimensional Graphing, (2008), http://www.youtube.com/watch?v=JRBDpg6awWs.

[32] González-Palacios M.A., Angeles J., The Design of a Novel Mechanical Transmission for Speed Reduction, J. Mech. Des., Vol. 121, 1999, pp. 538-543.

[33] González-Palacios M.A., Angeles J., The Design of a Novel Pure-Rolling Transmission to Convert Rotational into Translational Motion, J. Mech. Des., Vol. 121, 2002, pp. 1-3.

[34] Bai S., Angeles J., The design of spherical multilobe-cam mechanisms. Proc. IMechE, Part C: J. Mechanical Engineering Science, Vol. 223, No. C2, 2009, pp. 473-482.

[35] Chen C., Zhang X., Angeles J., Kinematic and geometric analysis of a pure-rolling epicyclic train, J. Mech. Des., Vol. 129, No. 8, 2007, pp. 852-857.

[36] Moreno-Báez M.A., González-Palacios M.A., Colín-Venegas J., Aguilera-Cortés L.A., Implementación de un Modelo de Simulación para el Programa VIBRATO. I Reunión Nacional de Estudiantes de Posgrado, Cuernavaca, México, Vol. 1, 2008, pp. 1-7.

[37] Rocha-Aguilera G., González-Palacios M.A., Colín-Venegas J., Aguilera Cortés L.A., Simulación en ADEFID del Movimiento Vibratorio de una Cuerda. XIV Congreso Internacional Anual de la SOMIM, Puebla, México, Vol. 1, 2008, pp. 1438-1452.

[38] González-Palacios M.A., Peña-Gallo R., Aguilera-Cortés L.A., OptimPlot2D: A Novel and Interactive Software Package to Analyze Single Variable Functions. CERMA Workshop on Innovation of the 2009 Electronics, Robotics and automotive Mechanics Conference, Cuernavaca, México, 2009, pp. 1-7.

[39] González-Palacios M.A., Bernal-Martínez C.A., Aguilera-Cortés L.A., OptimPlot3D: A Novel and Interactive Software Package for Analysis of Three Dimensional Surfaces. CERMA Proceedings of the 2009 Electronics, Robotics and automotive Mechanics Conference, Cuernavaca, México, Vol. 1, 2009, pp. 137-142.

[40] González-Palacios M.A., González-Barbosa E.A, Aguilera Cortés L.A., SnAM: A Simulation Software on Serial Manipulators, Engineering with Computers, Springer, 2012, pp.1-8. DOI: 10.1007/s00366-011-0246-6.