Interactive 3D Simulation of Flat Systems: The SpiderCrane as a Case Study

Davide Buccieri, José Sánchez, Sebastián Dormido, Philippe Mullhaupt, Dominique Bonvin

***Abstract*— In order to display the main characteristics of a well-known flat system, an interactive 3D simulation of SpiderCrane has been developed using *Ejs* (Easy Java Simulations) and Matlab/Simulink. The application allows users to set up different trajectories and introduce disturbances in a very visual and attractive way.**

I. Introduction

The control and stability problem of overhead cranes has recently received great attention [1]-[3]. It is worth mentioning that this problem is quite different from other nonlinear mechanical systems such as robot manipulators. For example, length variations during crane operation can produce a negative damping effect in the dynamics. Moreover, the number of degrees of freedom in cranes is larger than the number of actuators contrary to robots. Hence, the problem of designing automatic controllers for cranes described by highly nonlinear models is quite relevant and merits a certain degree of attention.

SpiderCrane is a new crane design. It was imagined in order to reduce, in a significant way, the time involved in carrying loads. The problem of classical cranes is the large inertia of the boom, which limits the crane dynamics. Hence, to improve the work rate, it is necessary to minimize the inertia of the system. In order to solve this problem, SpiderCrane is devoid of heavy mobile components.

The main contribution of this paper is to develop a complete virtual laboratory for the SpiderCrane system. A virtual laboratory is a distributed environment of software and estimation tools, intended to perform the interactive simulation of a mathematical model. Interactivity provides a flexible and user-friendly method to define the experiments performed on the model. During a simulation run, the user can modify the value of the model inputs and parameters and instantly view their influence on the overall dynamics. This strategy allows the user to design and conduct the experiments easily and, as a consequence, remain an active actor throughout the learning process.

The control choices made for the virtual laboratory controller have two parts. The first part is a feedforward controller that computes the open-loop input to the system so that it follows a trajectory specified by the user in the absence of perturbation and model mismatch. The trajectory is given in the virtual laboratory environment by choosing among a class of available trajectories (e.g. circular, polynomial, Lissajous curves and so forth). The second part of the control structure is a simple PD controller that takes care of eventual discrepancies both in the model and in the form of external disturbances. These disturbances are specified in the virtual laboratory environment in real-time by simple mouse clicks.

As mentioned above, the system is highly nonlinear. Hence, it is of no surprise that the most difficult part in the controller design lies in the computation of the open-loop input. To this effect, the full spectrum of nonlinear couplings are taken into account by exploiting the flatness property, which stipulates a one-to-one correspondence between the system variables (states and inputs) and the trajectory of a particular choice of system variables called the flat outputs. The importance of the dynamic possibilities offered by the use of this property is not always obvious by algebraic manipulations alone. Hence, the virtual laboratory is of great help in underlining the importance and simplicity of the results through interactively playing with different parameters of the system and the feedforward, and feedback controllers.

The paper is organized as follows. In Section II, the structure of SpiderCrane is presented together with some of its properties such as flatness, which is illustrated graphically. The foundation of Easy Java Simulations (*Ejs*), a software package that supports the creation of interactive dynamic simulation, is introduced in Section III. Section IV describes how *Ejs* can be employed in a very easy way in order to provide existing Simulink models with the level of interactivity required in a virtual control laboratory. The design and implementation of a SpiderCrane virtual laboratory is discussed in Section V, showing the suitability of *Ejs* in the development of complex 3D interactive interfaces. Finally, Section VI provides concluding remarks.

Manuscript received March 1, 2005.

D. Buccieri is with the Laboratoire d'Automatique, École Polytechnique Fédérale de Lausanne, Switzerland (e-mail: davide.buccieri@epfl.ch).

J. Sánchez is with the Department of Computer Science and Automatic Control, Universidad Nacional de Educación a Distancia, Madrid, Spain (e-mail: jsanchez@ dia.uned.es).

S. Dormido is with the Department of Computer Science and Automatic Control, Universidad Nacional de Educación a Distancia, Madrid, Spain (e-mail: sdormido@ dia.uned.es).

Ph. Mullhaupt is with the Laboratoire d'Automatique, École Polytechnique Fédérale de Lausanne, Switzerland (e-mail: philipe.muellhaupt@epfl.ch).

D. Bonvin is with the Laboratoire d'Automatique, École Polytechnique Fédérale de Lausanne, Switzerland (e-mail: dominique.bonvin@epfl.ch).

II. THE SPIDERCRANE DESIGN AND ITS PROPERTIES

SpiderCrane is made of three fixed pylons and a fixed gibbet. A pulley is mounted at the top of each pylon, allowing the sliding of a cable. These three cables are attached to a ring and, by varying their length, the ring can be moved in the surrounding space. The end of the gibbet is above the plane formed by the three pulleys and at the centre of the triangle formed by the pylons. At the end of the gibbet, another pulley is mounted, allowing the passage of the main cable. This cable goes through the centre of the ring and is attached to the load. The position of the load in space is done by adjusting both the position of the ring and the length of the main cable. All the cables are controlled by means of motors equipped with encoders, making it possible to measure the length as well as the speed of the cables.

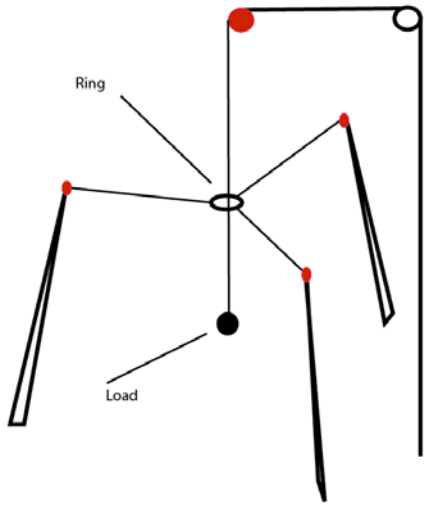


Fig.1 SpiderCrane.

The system has four inputs, namely three voltages applied to the motors for pulling the cables attached to the ring and the voltage applied to the motor for winching the main cable.

The dynamic equations of SpiderCrane are obtained using a Lagrange formalism with constraints [4]. The derivation of the dynamical model is given in [5].

SpiderCrane is a member of the class of cranes defined in [6]. According to this general formulation for 3D cranes, SpiderCrane has the following properties.

- The dimension of the working space is $p = 3$.
- There is no rigid articulated actuated system.
- The number of motors is $s+1 = 4$.
- The main pulley (the ring) moves in a manifolds of dimension $n=3$.

Naturally, SpiderCrane has the same property as its class; in particular, it is a flat system [6]. The flat outputs are given by the load positions (x,y,z) and the height of the ring. Notice that, for some mechanical systems, it is not always the case that the outputs to be steered are among the flat outputs (for instance, in the vertical take-off and landing aircraft (VTOL), the flat output is not at the position of the pilot, but lies below him [7]). Here, however, it is the case, which greatly simplifies motion planning for crane displacements (for example in obstacle avoidance problems).

This flatness property indicates a correspondence between the flat outputs and their derivatives on the one hand and the state of the system and the inputs on the other. Consequently, if the flat output describes a specific trajectory, the states and inputs will automatically follow corresponding trajectories. This is extremely useful for designing a feedforward controller. First, a trajectory with sufficient differentiability with respect to time is specified, which corresponds to the task at hand (lifting the load, changing the load from the current position to another pre-specified position, performing fancy trajectories, i.e. circular, Lissajous or polynomial, avoiding obstacles, etc.). Then, by the mere definition of flatness, there corresponds an input expression of only the flat outputs and their time derivatives up to a fixed determined order. Hence, a fixed set of closed-form formula exists (one for each input) that express the inputs necessary to follow exactly the trajectories specified, without integrating the system's differential equations, and independent of the type of trajectories chosen as long as they are known to have a sufficiently high (but fixed) level of differentiability.

Flatness is essentially a differential algebraic property [8] with a clear geometric meaning at least in the differential geometric sense [9]. However, flatness is not always easy to perceive and understand from a mechanical and physical point of view. The purpose of the forthcoming discussion is to illustrate with simple sketches how the algebra and geometry translate into tractable quantities such as positions, velocities, acceleration and forces of the constitutive elements of SpiderCrane. For a complete analytical treatment, the reader is invited to consult [5].

As mentioned previously, a choice of flat outputs for SpiderCrane is the load position (x,y,z) and the height of the ring position. Therefore, when trajectories for the flat outputs are chosen, their values at time t specify the position of the load in an unambiguous way. However, the position of the ring is still partly undefined, and only its height is given by one of the flat outputs (Figure 2.i).

In order to obtain the other coordinates of the ring, it is necessary to consider the second derivatives of the load position, i.e. its acceleration. Knowing the gravity direction and intensity, it is then possible to compute the resulting force acting on the load (Figure 2.ii). This force lies necessarily along the direction of the cable and thus, after intersection with the plane corresponding to the constant height of the ring, the ring position can be deduced (Figure 2.iii).

Pursuing this unraveling mechanism one step further by differentiating the ring position again twice (whence increasing the order of differentiability of the flat outputs), the ring acceleration is obtained. Together with its position and the direction and intensity of the gravity, it is then possible to deduce the forces to be applied to the cables. From this, the inputs of the system, which are forces, can be

obtained (Figure 2.iiii). The correspondence is then complete and the feedforward inputs to be applied can be obtained.

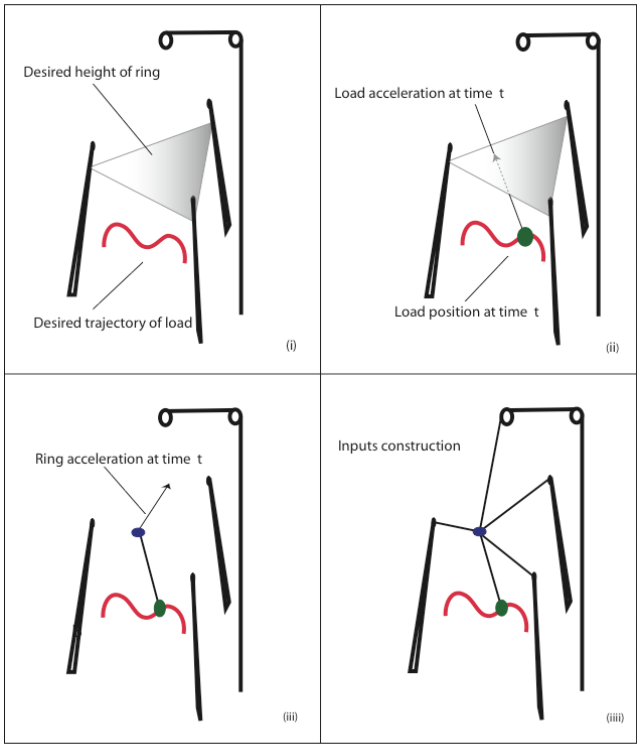


Fig. 2. Sketches illustrating the feedforward inputs computation based on flatness.

III. EASY JAVA SIMULATIONS FUNDAMENTALS

Easy Java Simulations (*Ejs*) is a freeware, open source, Java-based tool intended to create interactive dynamic simulations [10]. *Ejs* was originally designed to be used by students for interactive learning, under the supervision of educators with a low programming level. As a consequence, simplicity was a requirement. *Ejs* guides the user in the process of creating interactive simulations. This process includes the definition of the *model* and the *view*. *Ejs* implements its own procedure to define the model: a simple structure that the user needs to complete in order to specify the model variables and equations. In addition, *Ejs* version 3.4 supports the option of describing and simulating the model using Matlab/Simulink: (1) Matlab code and calls to any Matlab function (either built-in or defined in an M-file) can be used at any point in the *Ejs* model; and (2) the *Ejs* model can be partially or completely developed using Simulink block diagrams.

The view is the user-to-model interface of *Ejs* interactive simulations. It is intended to: (1) provide a visual representation of the relevant properties and dynamic behavior of the model; and (2) facilitate the user's interactive actions on the model. *Ejs* includes a set of ready-to-use visual elements that the modeler can use to compose a sophisticated view in a simple, drag-and-drop way. The properties of the view elements can be linked to the model variables, producing a bi-directional flow of information between the view and the model. Any change of a model variable value is automatically displayed by the view. Reciprocally, any user interaction with the view automatically modifies the value of the corresponding model variable.

Once the modeler has defined the model and the view of the interactive simulation, *Ejs* generates the Java source code of the simulation program, compiles the program, packs the resulting object files into a compressed file, and generates HTML pages containing the narrative and the simulation as an applet and an application.

IV. COMBINED USED OF *EJS* AND MATLAB/SIMULINK

Ejs 3.4 supports the option of describing and simulating the model (or just some parts) using Matlab/Simulink. In order to simulate the *Ejs* part of the model, *Ejs* implements a set of built-in ODE solvers, and it allows the modeler to program and use his own numerical algorithms. The Simulink part of the model is simulated by Matlab/Simulink, using Simulink numerical algorithms.

Let's focus our attention in the combined use of *Ejs* to create the interactive user interface and Simulink and develop the complete model. The procedure consists of three main steps: (1) adapt the Simulink model in order to be ruled by the *Ejs* interface, (2) develop the *Ejs* interface taking into account the inherent interactivity of the model, and (3) connect both parts by establishing the link between the *Ejs* variables and the Matlab/Simulink variables.

The first step, i.e. the adaptation of a Simulink model to be controlled by an *Ejs* view, consists essentially in the connection in the original Simulink diagram of specific blocks to produce the exchange of information with the Matlab workspace. At every integration step, the Simulink model must read the input variables and parameters from the Matlab workspace, simulate the system using this information, and write the resulting state model to the workspace (Figure 3). This cyclical operation is due to the fact that Matlab workspace is really the buffer that *Ejs* and Simulink use to exchange data. Every change in the *Ejs* views is sent to the Matlab workspace and read by Simulink blocks. At the same time, the outputs of Simulink are written in the workspace, read by *Ejs* and used to refresh the different simulation views.

Furthermore, *Ejs* operates in a similar cyclical way as Simulink: at every cycle, it reads the Simulink outputs from the Matlab workspace and refresh the Ejs view. At the same time, any change in an *Ejs* variable linked to a Simulink variable is sent to the Matlab workspace. Synchronization between the two worlds is guaranteed: a particular block must be included in the Simulink diagram, and the final *Ejs* application contains the necessary built-in Java methods.

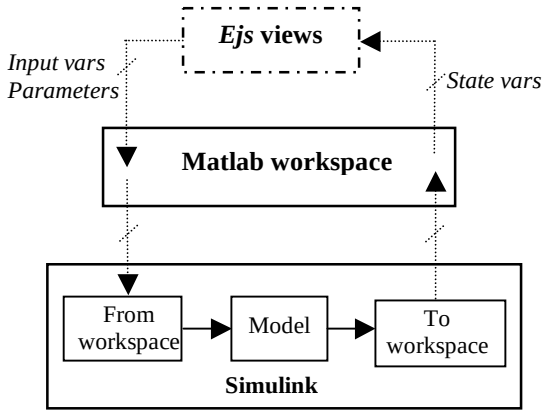


Fig. 3. Exchange of data between the *Ejs* views, the Matlab workspace, and the Simulink model. The "From workspace" and "To workspace" boxes represent the new blocks included in the Simulink block diagram to read/write the variables from/to the workspace.

The second step, i.e. the construction of the *Ejs* view, is performed to take advantage of the inherent interactivity of the simulation system. The idea is that the designer creates the view to illustrate the relationships among the state variables and the parameters of the model [11]. During the interactive simulation run, the user can change the values of the system inputs and parameters and instantly see how these changes affect system behavior. Interacting with an instructional simulation can help learners gain a better understanding of a real process or a phenomenon through exploring, testing hypotheses, and discovering explanations for the mechanisms and relationships. This interactivity may provide opportunities for students to modify their mental models, by comparing the outputs of the simulated system with their expectations, and to explore and associate actions with effects, which will lead to better understanding.

Hence, the creation of an interactive *Ejs* view consists of establishing the connections needed for:

a) The correct visualization of the state of the phenomenon being simulated, and

b) the appropriate interaction of the user with the view (either to modify this state or to perform the actions defined by the model).

For this, *Ejs* provides a broad range of elements that let designers build sophisticated interactive interfaces without deep knowledge of programming: 2D and 3D objects, buttons, sliders, scopes, labels, etc. If we want this interaction to have certain relevance on the program, the gestures on the interface need to trigger actions that affect the model variables. By doing some gestures (such as clicking or dragging the mouse, hitting the keyboard, or moving a joystick) with the computer peripherals on the program interface (or view), the view itself can be used to control the simulation. In order to do that, every element has a panel of properties to allow users to interact with them by writing Java code or using built-in *Ejs* methods.

The third and last step to complete the interactive simulation is to set up the bi-directional link between the *Ejs* view and the Simulink model. In short, this step consists of writing a plain file with the list of Simulink variables to be linked and, finally, importing this file in *Ejs* (Figure 4). The linking process is done by explicitly defining pairs of *Ejs* and Simulink variables. When the final application is compiled, the Java code contents the built-in methods to do the link among the pairs of variables in a way transparent to the user.

```
model='Simulgrue.mdl';          %EJS Model
t;                              %EJS Variable
ChoixTrajectoire;               %EJS Variable
g;                              %Ejs Variable
m;                              %Ejs Variable
……
……
KP;                             %Ejs Variable
KD;                             %Ejs Variable
```

Fig. 4. Excerpt of the text file showing some of the Simulink variables used in the model of the crane. The first line lets *Ejs* application know the name of the Simulink block diagram corresponding to the spider crane model. The text file must content all the Simulink variables to be linked with *Ejs* variables.

V. Case Study: Interactive Simulation of SpiderCrane

To demonstrate the suitability of Ejs in the development of 3D interactive views of simulated systems, SpiderCrane has been selected as the case study. The reasons for such a selection are:

a) The mathematical model has been developed in Matlab/Simulink. So, the *Ejs* is just used to develop the interactive view, and

b) the system is visually very interesting since it is composed of elements with mobile components. It allows one to use a 3D representation of the mechanical system in the view using some of the *Ejs* graphical objects and to draw the trajectories of the load and the effect of the disturbances.

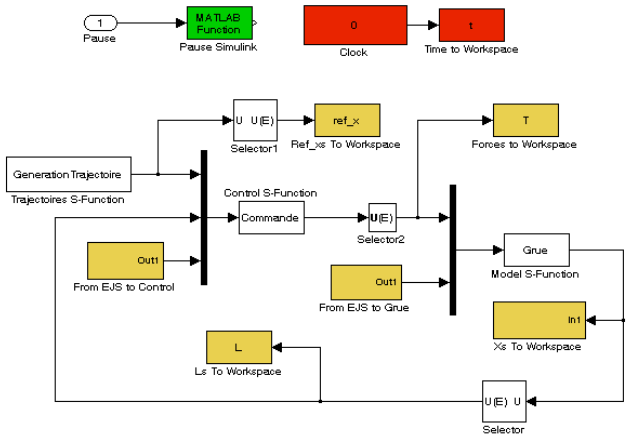


Fig. 5. Simulink diagram of SpiderCrane with the blocks to exchange data with the *Ejs* views.

As was described in the previous section, the first step is to modify the original Simulink block diagram in order to exchange information with the *Ejs* views. Figure 5 shows the diagram with the inclusion of these new blocks. It is noticed that there are two groups of blocks: source blocks to read Matlab variables from the workspace, and sink blocks to

write the model state to the workspace. In this case study, some of the variables to be read are general parameters (gravity, mass), control parameters (KD, KD), trajectories configurations (type, destination point, time, velocity), and load disturbances. The values to be written at every integration step are position and velocity of the load and ring, cable lengths, and motor forces.

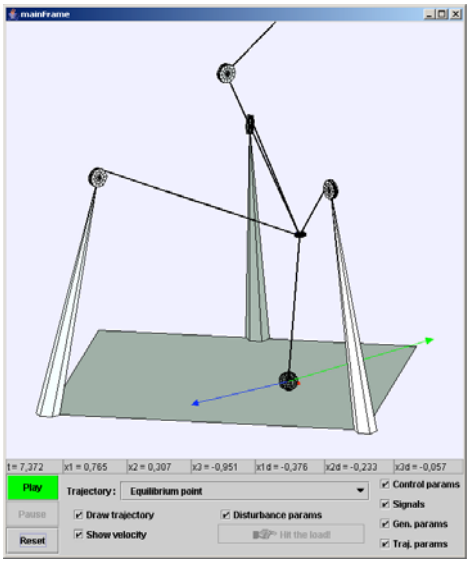


Fig. 6. Main window of the SpiderCrane application. The frame shows the load being moved to reach the equilibrium point. The two vectors represent the composition of the velocity and disturbance in every simulation step.

Figure 6 presents the main window of the 3D graphical user interface developed by *Ejs* using the previous Simulink diagram as the model. This window is composed of two elements to get a complete control of the flat system: the view panel and the control panel. The view panel presents a 3D representation of SpiderCrane in agreement with the existing physical set-up (positions of the pylons, pulleys, gibbet, ring, and load). This view lets us visualize in a very realistic fashion the movement of the cables, the ring and the load in accordance with the trajectory and the control parameters specified by the user; also, the view shows two vectors corresponding to the instant velocity of the load, and the magnitude of the applied disturbance. At the bottom of the view panel, there are seven text fields showing the time, and the load position and velocity.

The control panel is located at the lower part of the main window. It gathers all the elements necessary for control of the flat system. The left part of the panel has three buttons to run the crane: once the reference trajectory is specified (type of displacement, initial and final positions, duration, etc.), the user has to press the *Play* button to start the simulation; if the *Pause* is pressed, the simulation is stopped and a new trajectory can be specified; the *Reset* button reinitializes both *Ejs* interface and Matlab/Simulink environment.

The four check buttons on the right-hand side of the control panel are to open/close new windows. These windows let change various parameters (Figure 7) and visualize scopes with the states of the crane (Figure 8).

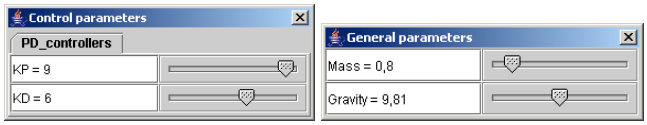


Fig. 7. The parameters of the PD controllers of the pulleys can be modified with the window on the left. Also, the mass of the load and the gravitational acceleration can be changed to simulate different situations.

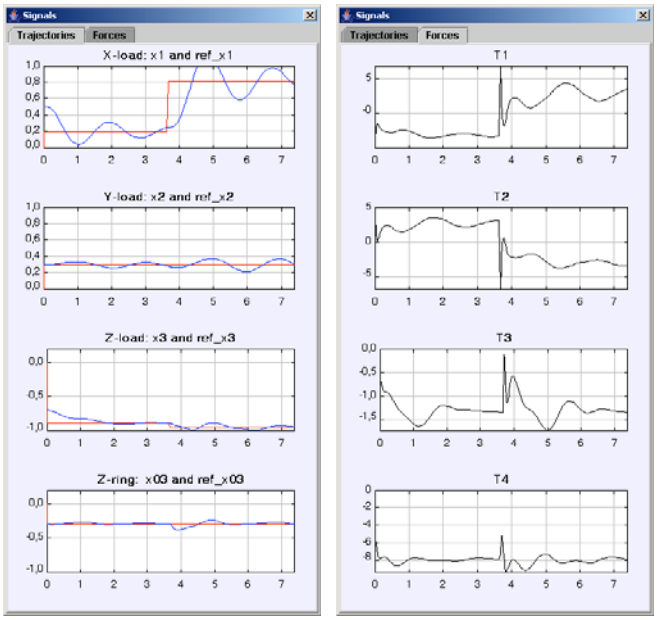
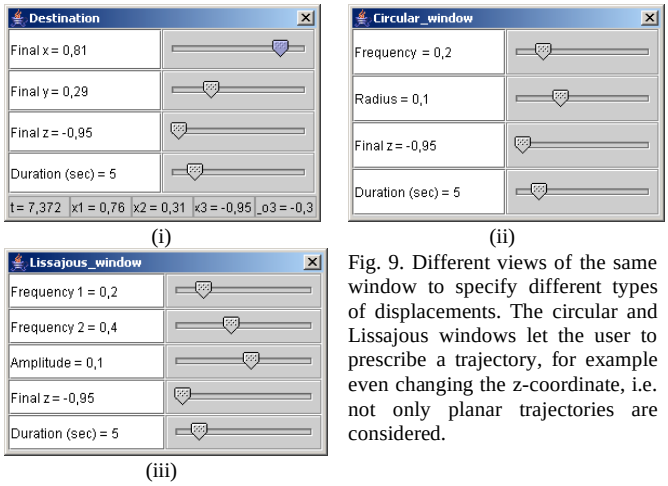


Fig. 8. The same window is used to present two groups of scopes with the states of the crane. The scopes on the left give the reference (red) and the real (blue) trajectory. The scopes on the right present the motor forces.

Figures 9.i-iii present different views of the window to set up the reference trajectory of the load. The set of parameters changes according to the type of displacement chosen. The application has four types of trajectories: equilibrium point, linear, circular and Lissajous.



(i) (ii) (iii)

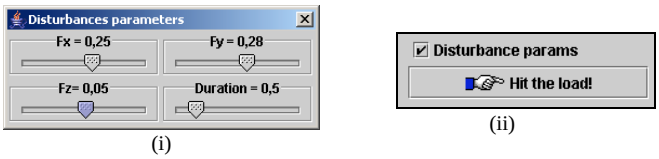
Fig. 9. Different views of the same window to specify different types of displacements. The circular and Lissajous windows let the user to prescribe a trajectory, for example even changing the z-coordinate, i.e. not only planar trajectories are considered.

The first two are used for transporting the load from an initial to a destination point but with different conditions. When linear trajectory is chosen, the displacement of the load is done in limited time, taking into account the fact that

the derivatives up to the 4th order of the initial and destination points of the trajectory are set to zero. If equilibrium point is chosen, a destination point must be selected and the crane moves the load regardless of any derivatives. This type of displacement induces big oscillations in the load. In both cases, it is possible to drag the destination point directly on the view panel using the mouse.

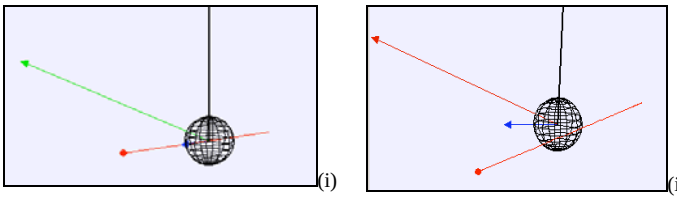
The other two types of trajectories are specified in Figures 9.ii-iii. They are very similar as sinusoidal expressions are used to calculate the *x-y* coordinates. In both cases, the derivatives up to the 4th order of the *x-y* sinusoidal expressions are needed. Also, to calculate the *z* displacement, a linear trajectory is used so that the derivatives up to the 4th order in the initial and final *z* have to be set to zero.

When a trajectory has been specified, it is depicted in the view panel by a red path. During the displacement, disturbances in the acceleration of the load can be applied. For this, it is necessary to fix the magnitude, direction, and duration of the disturbance (Figure 10.i). The disturbance is represented in the view panel by a green arrow. Since this arrow is enabled, the user can drag it to change the magnitude and direction of the disturbance too.



(i) (ii)

Fig. 10. (i) Sliders to pre-set the components of a disturbance. (ii) When the simulation is running, the *Hit the load!* button is enabled in the view panel to introduce the disturbance previously specified in the disturbance parameters panel.



(i) (ii)

Fig. 11. (i) A linear displacement where the green vector represents the disturbance before its application. (ii) The disturbance is applied, resulting in the load moving away from the reference. The disturbance is now in red, and the blue vector represents the velocity at that instant.

Once the perturbation is pre-set and the simulation is running, the user can apply the disturbance by pressing the *Hit the load!* button (Figure 10.ii). From this moment on, the disturbance begins to be applied and lasts the time fixed in the disturbance panel. Figures 11.i-ii are two frames of a trajectory, before and after the disturbance.

VI. Conclusions

In order to illustrate the performance of SpiderCrane, an interactive 3D application has been developed using *Ejs* and Matlab/Simulink. This application lets the user program different types of load displacement, simulate disturbances, and change many parameters to test the performance of the modeled system.

The application can be improved to explore new paths for both research and teaching. From a pedagogical point of view, the application can be used as a training step before operation with a real crane. For this purpose, the control of the crane by a joystick with force feedback is presently being programmed in Ejs. This will allow the user to feel the movement of the load when it is transported between two positions and under different configurations.

References

[1] Y. Fang, W.E. Dixon, D.M. Dawson, and E. Zergeroglu, "Nonlinear Coupling Control Laws for an Underactuated Overhead Crane System", *IEEE/ASME Transactions on Mechatronics*, vol. 8, no. 3, pp. 418-423, 2003.

[2] H.H. Lee, "A new approach for the anti-swing control of overhead cranes with high-speed load hoisting", *Int. J. Control*, vol. 76, no. 15, pp .1493-1499.

[3] T. Gustafsson, "On the design and implementation of a rotary crane controller", *European J. Control*, vol. 2, no. 3, pp. 166-175, 1996.

[4] D.T. Greenwood, *Classical Dynamics*. Englewood Cliffs, NJ: Prentice-Hall, 1977.

[5] D. Buccieri, Ph. Mullhaupt, and D. Bonvin, "SpiderCrane: Model and properties of a fast weight-handling equipment", *IFAC World Congress*, Prague, 2005.

[6] B. Kiss, J Lévine, and Ph. Mullhaupt, "Modelling, flatness and simulation of a class of cranes", *Periodica Polytechnica, Ser. El. Eng.*, vol. 43, no. 3, pp. 215-225.

[7] P. Martin, S. Devasia, and B. Paden, "A different look at output tracking: Control of a VTOL aircraft.", *Automatica* 32, no.1, pp. 101-107, 1996.

[8] M. Fliess, J. Lévine, Ph Martin, and P. Rouchon "Flatness and defect of nonlinear systems: introductory theory and applications", *Int. J. Control*, vol. 61, no. 6, pp. 1237-1361, 1995

[9] M. Fliess, J. Lévine, Ph. Martin, and P. Rouchon "A Lie-Bäcklund approach to equivalence and flatness of nonlinear systems", *IEEE Transactions on Automatic Control*, vol 38, pp. 700-716, 1999.

[10] F. Esquembre, "Easy Java Simulations: A software tool to create scientific simulations in Java", *Comp. Phys. Comm.*, vol. 156, pp. 199-204, 2004.

[11] J. Sánchez, S. Dormido, F. Esquembre. "The learning of control concepts using interactive tools (Accepted for publication)". *Computer Applications in Engineering Education*, to be published.