Interface: The Furuta pendulum

INTRODUCTION

This document describes the environment developed with Ejss for the Furuta pendulum. The simulation interface is shown in figure 1. On the other hand, the Remote laboratory interface is similar to the previous one, except its controls, which are applied to the inverted pendulum as shown in figure 2.

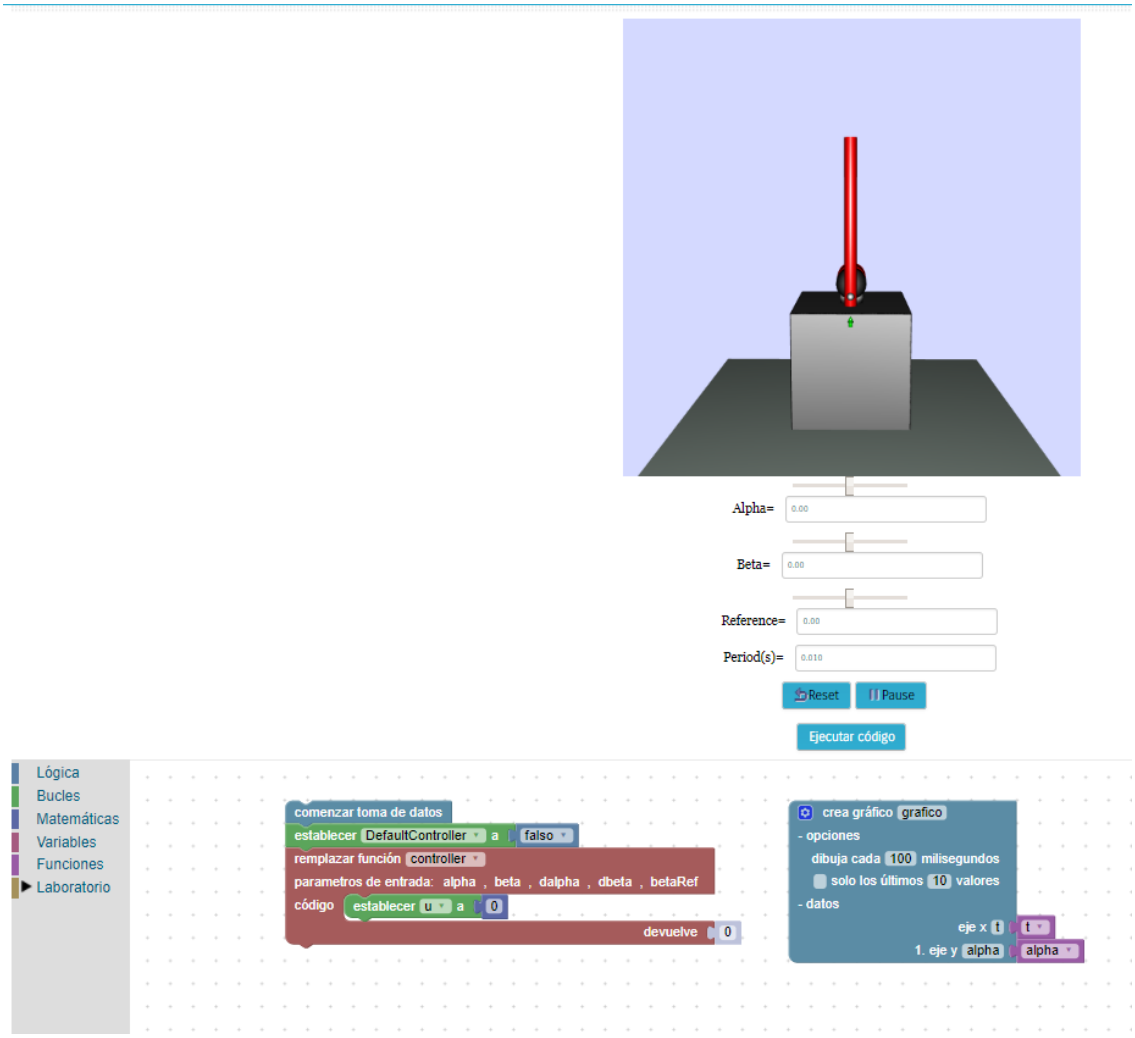


Fig. 1. Simulation interface.

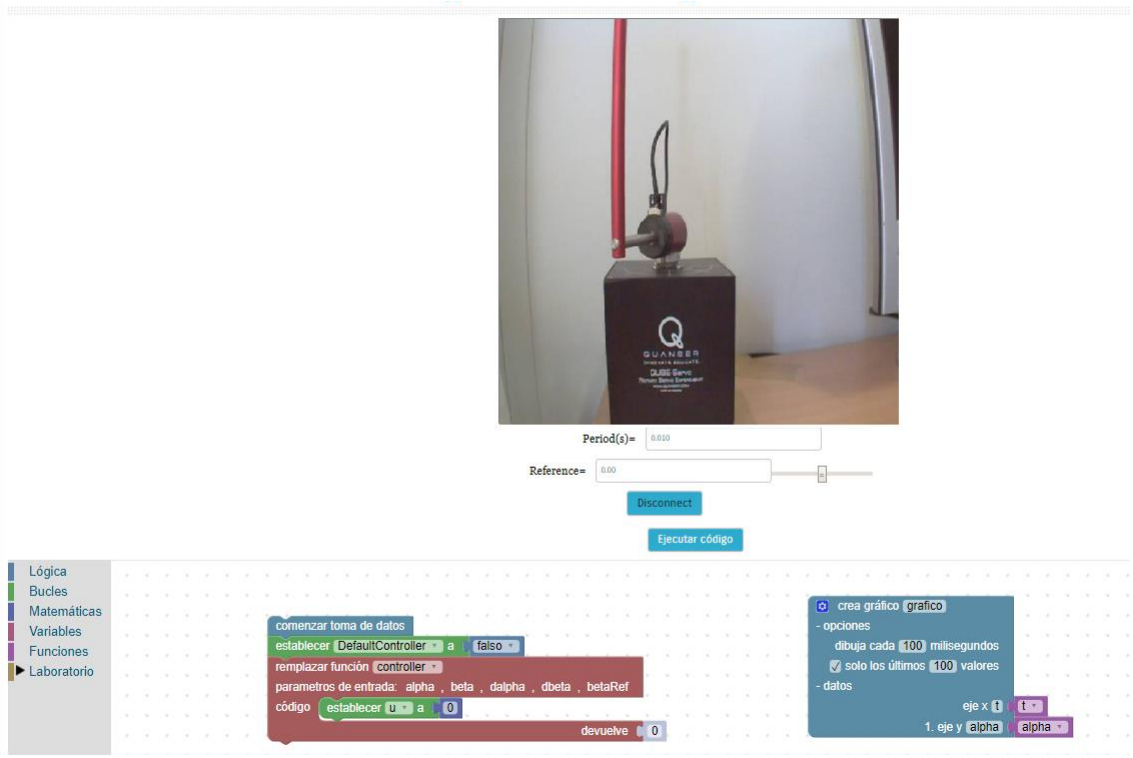


Fig. 2. Connection interface of the real plant.

1. USE OF SIMULATION

In this section the operation of the virtual laboratory (simulation) is explained.

1.1. Main view

The interface consists of several parts:

First of all, there is a view that allows to interact with the pendulum that it is represented in figure 3.

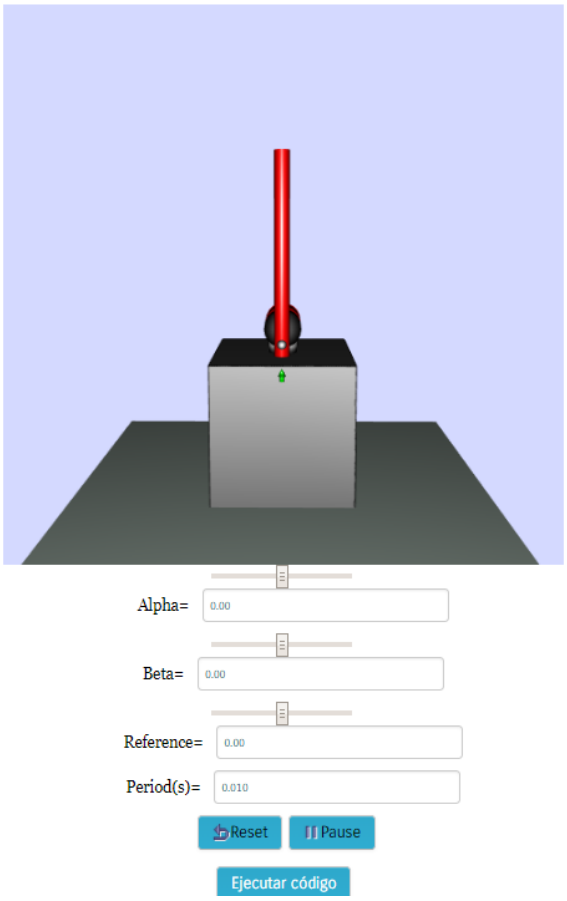


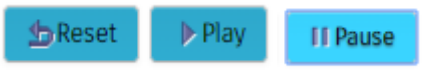
Fig. 3. Main view of the pendulum and controls.

The view contains controls that allow you to modify the Alpha and Beta angles, as well as the reference and the sampling period. If, for example, the value of Alpha is modified you can observe a surprising behavior... The pendulum does not fall!

It is not an error in the simulation, what happens is that the pendulum includes a default controller that keeps it in its upper position. However, if the Alpha angle is modified energetically it is possible to see how the pendulum falls and then gets up again.

It is also interesting to modify the Beta angle reference to check how the default control works, since it is an example of the control that should be implemented in practice.

At any time, the simulation can be paused, resumed or restarted with the following buttons:



Note that the play / pause buttons are displayed alternately depending on whether the simulation is paussed or running.

1.2. Code editing panel

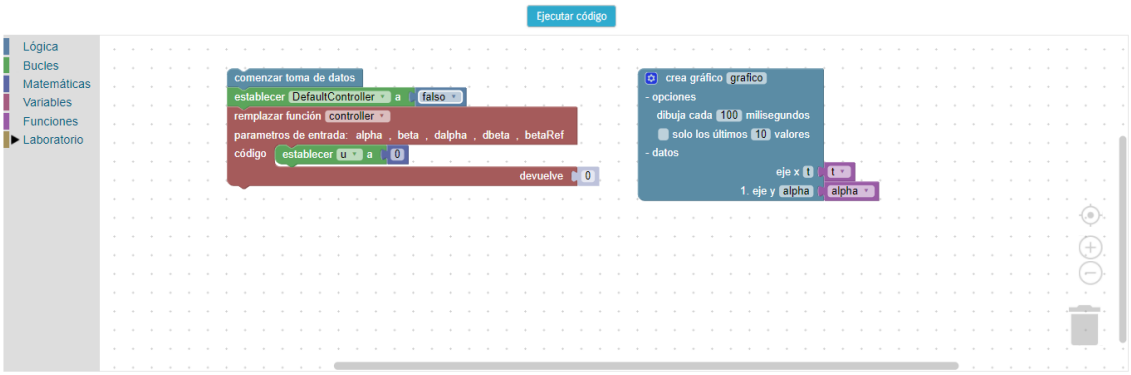


Fig. 3. Code editor

To change the behavior of the simulation, there is (just below it) a code editor that is shown in figure 3. The code is created using a graphic editor called Blockly. Upon entering the laboratory, a code appears by default. For that code to take effect you have to click on the "Ejecutar codigo" (Run code) button.

In doing so, two changes will be appreciated. First, a graph to the right of the situation appears, secondly the Furuta pendulum falls off. Both things can be seen in figure 4.

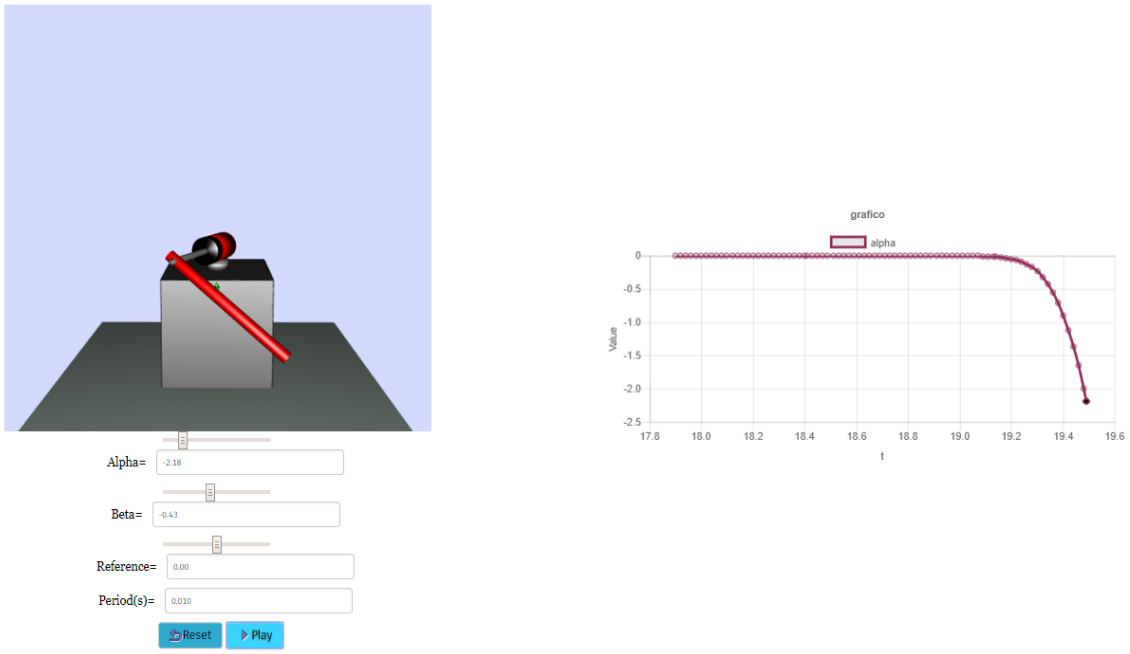


Fig. 4. Execution of the default code

To recover the normal behavior and for the pendulum to rise again, it is enough to modify the value of the "DefaultController" variable and set it to "verdadero" (true). For this, simply modify the code as shown in figure 5. Next, press the button "Ejecutar código" (Run code) and check that the pendulum rises again.

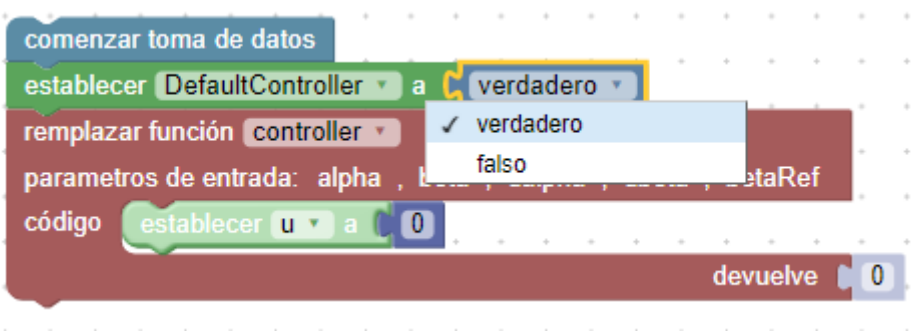


Fig. 5. First edition of the code

1.3. File panel

A fundamental part of the environment is the file panel that is in the right side of the page and shown in figure 6. In this panel, you can save the code files that are being developed and return to previous versions.

Fig. 6. File panel.

As an example, to save the modified code we will press the button "Mostrar" (Show) of the section "Blocky". When doing so, the following additional options appear: "Grabar código" (Save code) and "Cargar código" (Load code).

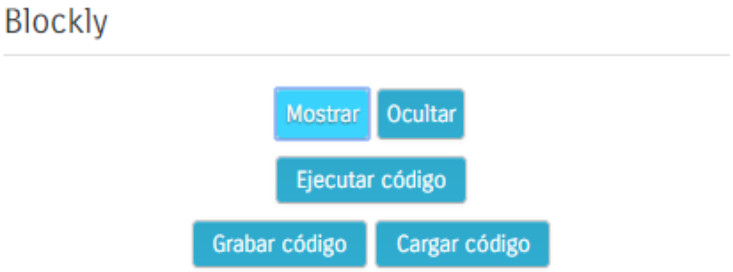
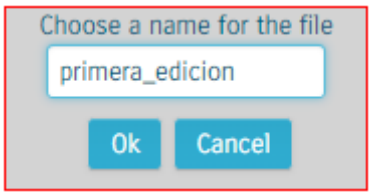


Fig. 6. Blocky controls

Press the first of them to make appear a dialogue asking for the name:

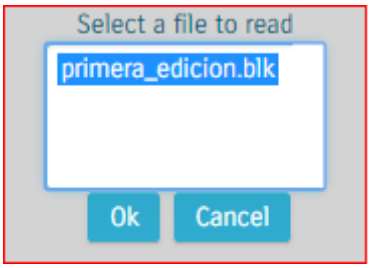


Enter the name "primera_edicion" and press OK. When doing so, a new file "primera_edicion.blk" appears in the file panel:

Fig. 7. Created files

You will be able to reload the code at any time by clicking on the file in the panel of figure 7.

Another alternative option is to click on the "Cargar código" button in the Blokly section (figure 6). In this case we can select the code to be read and load.



Although it may seem redundant, both options have their specific use. If pressed on a file in figure 7, the simulation is completely reinitiated with the code loaded in said file. If, on the other hand, you click on the "Cargar código" button of Figure 6, only the code is modified leaving the simulation unaltered.

If you want to go back to the default code, simply exit and re-enter the laboratory.

Finally, if you click on the "Gestionar archivos" (Manage Files) button of figure 7 you can access the saved files (see figure 8) as well as download or upload new files. This is very useful to send the code to the teacher. Saying panel also allows you to delete unnecessary files or download all files like a zip.

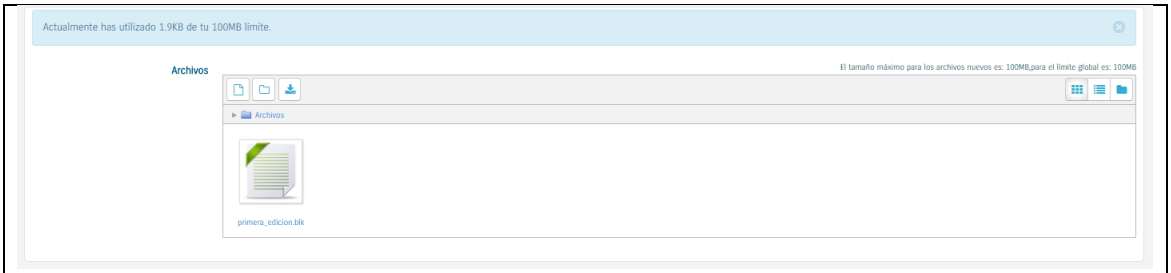


Fig. 8. File management

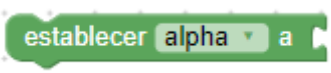
1.4. Code creation

Once accustomed to the use of the tool, it is time to create a code to control the simulation. To do this we will first delete all the blocks that appear by default. For this, just select them with the mouse and press the "delete" key or drag the blocks to the trash:

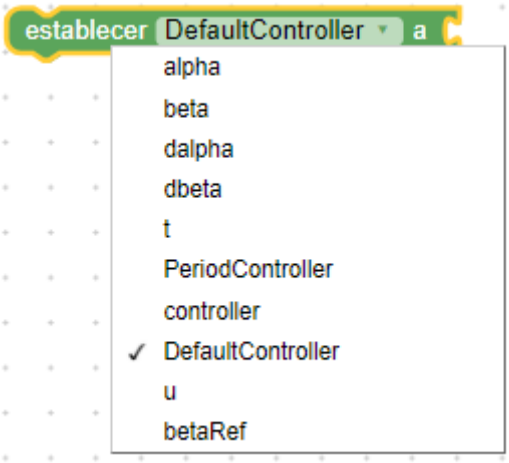
Then, as a preparatory exercise, we will reconstruct the code by default to explain its operation.

First, we must disable the default driver, that is achieved clicking on the left side panel on the tab "Laboratorio" (Laboratory) and then on "Variables":

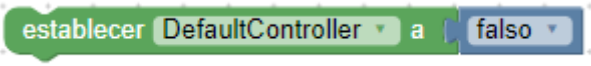
Click then on the "establecer" (establish) block and grad it to the code zone.



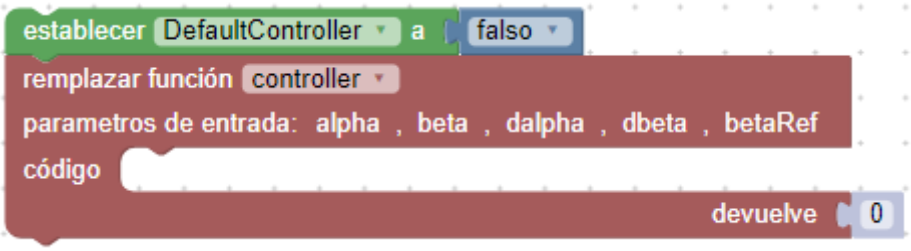
Then, click on the drop-down menu in which alpha is selected and select "defaultController" instead:



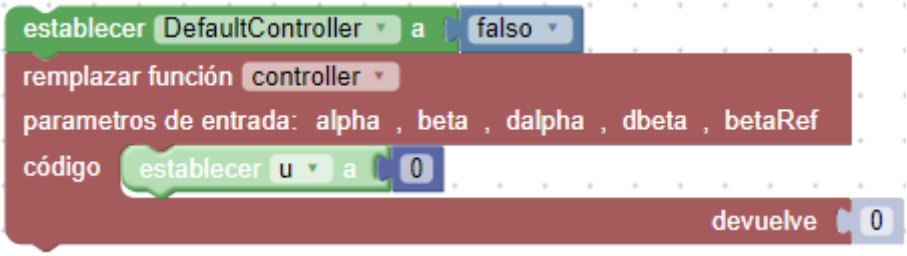
Once this is done, select the "verdadero" (true) constant from the "Lógica" (Logic) panel and drag it to the code zone for connecting it to the previous block as if it they were puzzle pieces. Finally change "verdadero" (true) to "falso" (false):



Done this, in the functions panel take the "reemplazar función" ("replace function") block and connect it to the rest of the blocks as follows:



In doing so, a space remains to include the code that will replace the controller. In this space put a "establecer" (set) block with the variable "u" to the value zero (which is in the "Matemáticas" (Mathematics) panel). The steps are similar to the case of the DefaultController variable. The final result should be:



There are several shortcuts to put the block "establecer" (establish), on the one hand we can select block and use the copy and paste functions (ctrl-c and ctrl-v). Another way is to select the blocks that are already in the view and right-click on

them to then select the option "duplicate". These options are more useful if you want to copy a set of large code blocks. Another useful option of the menu that appears right-clicking the elements, is to disable the code (similar to adding a comment in a text code).

It would be very long to describe all the options blockly has, but this section has shown that creating code is very intuitive and we encourage the students to check how the different options work for themselves.

Since we have worked hard, we will save the code with the name "Test 1".

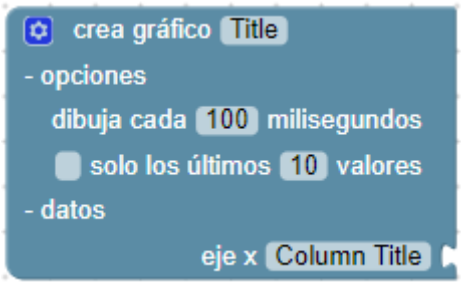
If the code is executed we will see that we have managed to disable the controller by default and replaced the code of the controller with the code "u = 0". Done this, the pendulum is out of control and falls freely.

Obviously, using the blocks that can be found in the section "Mathematics", it is possible to build a much more sophisticated and useful controller. This is the purpose of practice.

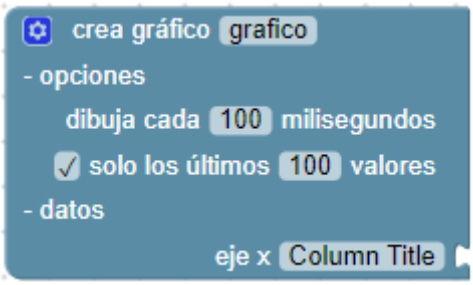
1.5. Graph definition

The program described so far has a problem, it is not easy to visualize the result of the execution of the code since graphics are not shown with the variables.

To create the graphs, select the block "crea gráfico" (create graphic) that is in the "Laboratorio" (Laboratory) category, subsection "Gráficas" (Graphs). This code runs in parallel with the previous one, so you do not have to connect it with anything else.

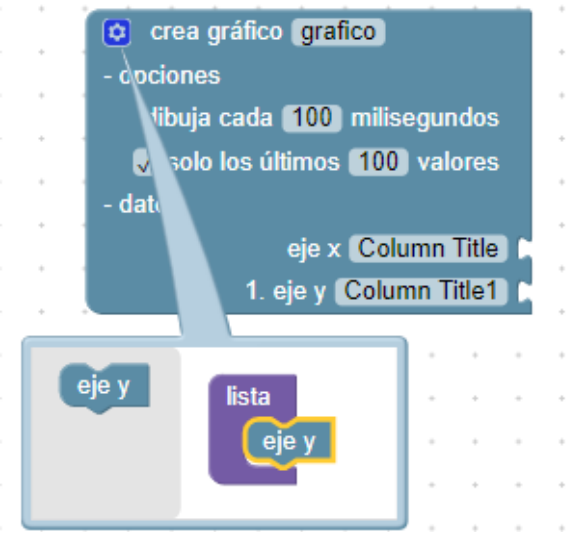


It is necessary, however, to configure the data you want to paint. For doing so, click on the title and write "grafico". Indicate that the graph should only paint 100 values every 100 ms (which results in a total of 10 seconds of data).

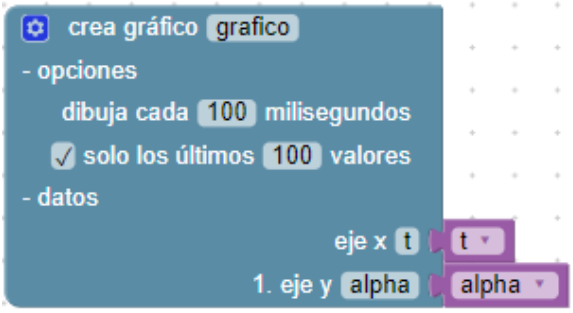


Then, it's time to add the data axes. This is done by clicking on the gear-shaped button located in the upper left part of the block and dragging as many "eje y" (y axis) pieces as we

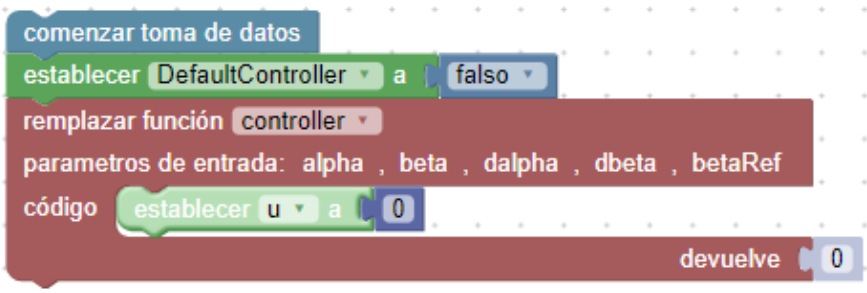
want to represent in the "lista" (list) block. This makes spaces appear to connect the different variables of the drawing:



Press again on the gear to close the editing block. Finally, edit the titles and connect the variables to be painted. For this, go to the section "Laboratorio" (Laboratory), subsection "variables", and take the variables that you want to connect. Configure the block as follows:



If we execute the code as it is, we will see that a graphic appears, but it nothing is painted on it. To solve it, indicate the simulation to start the data collection. For this, select the block "comenzar toma de datos" (start registering data), located in the"Gráficas" (Graphics) subsection, and place it at the beginning (above) of the block that changed the controller:

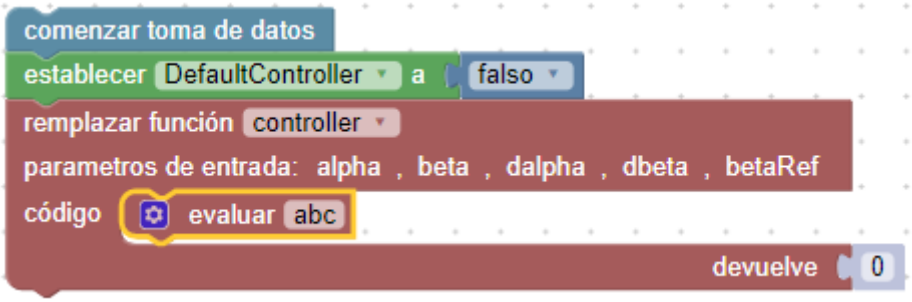


After all this work, we finally have a control code by default with some data plotted and we can save it for later use with name "default".

1.6. Advanced code editing

All the code can be created using the Mathematics blocks and the variables of the laboratory. However, to create a complex control code, it may be useful to edit code in textual form. For this, it is possible to use the code block "evaluar" (evaluate) that is located in the section "Laboratorio" (laboratory), subsection "funciones".

To illustrate its general operation, we will erase the code "establecer u a 0" (establish u to 0, see last figure in previous page) and we will replace it with the "evaluar" (evaluate) block:



By clicking on the gear we can edit the code more easily:

Replace the "abc" code with the following:

```
set('u',0);
```

Which obviously has the same effect as the previous block code (establishes the control signal u to a value of zero). Close the edit box by clicking again on the wheel and test the operation of the block.

An example of a more general code is the following:

```
/*obtain the variables*/

var alpha=get('alpha');
var beta=get('beta');
var betaRef=get('betaRef');
var dbeta=get('dbeta');
var dalpha=get('dalpha');
```

```
/*calculate the control action */
var u= complex function in Javascript; …
/*apply the control action */
set('u',u);
```

Example 1. Generic controller code template

This code can be used as a generic template to build complicated controllers using JavaScript code. Note that the general procedure is the following: first, obtain the variables of the simulation with the "get" methods, then calculate the control action and finally apply it with "set".

2. USE OF THE REMOTE LABORATORY

The operation of the remote laboratory is very similar to the simulation, so this section focuses on the main differences between both.

2.1. Interaction with the laboratory

The first difference between the simulation and the real laboratory is that since you work with the physical laboratory, the actions that can be applied are more limited. To establish the connection, just press the button "Connect".

Una vez conectado el botón dejará de mostrarse y aparecerá un botón "Disconect". También podemos comprobar que la conexión está iniciada cuando aparece la imagen del péndulo real (ver figura 10).

Fig. 9. Control of the real laboratory.

As can be seen in Figure 9, the controls in the real system are more limited than in the simulation. In particular, we can only change the reference and the control sampling period.

The rest of the variables can not be manipulated remotely in the real system. The interaction with the laboratory can neither be "paused" or "resumed".

The action that can imitate the restart of the laboratory consists of pressing the button "Disconnect" and then the "Connect" button.

2.2. Advanced code editing

The editing and execution of the code using the blockly elements is totally analogous to the virtual laboratory. The only important difference is that it is not possible to access all the variables that we had in simulation.

In particular, the "evaluar" (evaluate) function that allows writing Javascript code can not "set" nor "get" simulation variables.

The remote laboratory automatically measures the real alpha, beta, betaRef, dalpha and dbeta variables before executing the code, executes the code and then applies the control signal u to the pendulum automatically.

In this way, to test the code in the real laboratory, it is enough to comment the sections where the "get" and the "set" were used in the template of Example 1:

```
/*the remote lab automatically obtains the variables.
Therefore, this part of the code is commented
var alpha=get('alpha');
var beta=get('beta');
var betaRef=get('betaRef');
var dbeta=get('dbeta');
var dalpha=get('dalpha');*/

/*calculate the control action */

var u= función muy complicada en javascrip;
…

/*The application of the control signal is also
automatic, so this part of the code is also commented
set('u',u); */
```

Example 2. Modification in the code to execute it with the remote lab.