Planar 2-links robot simulator: report

Luca Tarasi

2019/09/18

Introduction

0.1 Purpose

This package has the purpose of simulating a planar robot with two links. The links are ideal (uni-dimensional) and the actuator is modelled as a material point. Degree of freedom of the simulation are the lengths of the links, the maximum linear speed of the robot, the simulation step, the way the state of the simulation is to be plotted (or not) during the simulation itself, and, very importantly, the way the signal is to be provided to the simulator. There are three ways to do this:

1. Mode 1: the signal is already stored in some variables, in a row array format.
2. Mode 2: an arbitrary number of continuous signals can be plotted by the user by cursor; they will be stored and then simulated later.
3. Mode 3: an arbitrary number of continuous signals can be plotted by the user by cursor, so that the simulated robot will follow them in real time.

The package offers two user interface functions, `prepare_simulation` and `start_simulation`. The former takes the parameter listed above (and, in case, other user inputs) as inputs, and returns a custom structure, which contains the parameters needed for the real simulation and is then to be given as input to `start_simulation`, which will effectively run the simulation. This call returns the significant signals of the simulation in a column vector format. Details on these functions are to be found in Part I. Other functions, used internally by these two, are specified in Part II. Finally, the Simulink model and the MATLAB interpreted functions used in it are described in Part III.

0.2 Control method

As with any manipulator, we have $\dot{\mathbf{x}} = \mathbf{J}\dot{\mathbf{q}}$, where $\mathbf{x}$ is the position of the actuator, $\mathbf{q} = (q1, q2)$ is the vector of generalized coordinates and $\mathbf{J}$ is the Jacobian of the system. Define $\mathbf{x}^*(t)$ as the reference signal (the trajectory we want our robot to follow), which we require to be differentiable (and, therefore, continuous); observe that, since we work on a discrete (computer) system, "differentiable" will mean that the difference quotient, computed with a step h_{step}, exists and is less, in absolute value, than an arbitrary maximum value v_{max} (both h_{step} and v_{max} will be passed to `prepare_sim` by the user as input parameters).

Define $\boldsymbol{\delta}\mathbf{x}(t) \coloneqq \mathbf{x}(t) - \mathbf{x}^*(t)$. Now, if J is invertible and well conditioned, i.e. we are sufficiently far from the singularity points, we can impose the following input signal,

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}(\mathbf{x}^*(t) - \mathbf{K}\boldsymbol{\delta}\mathbf{x}(t))$$

where $\mathbf{K}$ is a matrix with positive eigenvalues. We obtain

$$\boldsymbol{\delta}\dot{\mathbf{x}}(t) = \dot{\mathbf{x}}(t) - \dot{\mathbf{x}}^*(t) = \mathbf{J}\dot{\mathbf{q}} - \dot{\mathbf{x}}^*(t) = -\mathbf{K}\boldsymbol{\delta}\mathbf{x}(t)$$

which yields $\boldsymbol{\delta}\mathbf{x} = e^{-\mathbf{K}t}\boldsymbol{\delta}\mathbf{x}(t_0)$, where t_0 is the starting time of simulation. So, in general we'll have

$$\lim_{t\to\infty} \boldsymbol{\delta}\mathbf{x}(t) = \mathbf{0}$$

while, if $\boldsymbol{\delta}\mathbf{x}(t_0) = 0$, $\boldsymbol{\delta}\mathbf{x}(t) = \mathbf{0}$ for all $t \geq t_0$. In the program, we will always aim to have null initial error, in order to have null error in the whole simulation (except for numerical errors); for this reason, we'll have to know the initial value of the signal and position the robot in the corresponding configuration before the start of the simulation.

0.3 Decisions

During the development of the package, a few decisions had to be taken.

1. Under what abnormal circumstances does the simulation stop with error? At first, it was decided for this to happen when the reference point is unreachable and when the reference signal's speed exceeds the maximum one of the robot. In the end, this is what was done for the `OFFLINE` modes. In the `ONLINE` mode, it is hard for the user to properly control the speed of the cursor as the plot is drawn in real-time; so, if the maximum speed is exceeded, the robot moves at its maximum speed possible, i.e. it tries to keep up with the user. Hopefully, the user will slow down to adapt to the robot's maximum pace. Of course, even in `ONLINE` mode, an unreachable point causes the simulation to stop with error.

2. In the `ONLINE` mode, the reference signal is not known before the beginning of the simulation; however, in order to have null error for each time instant (at least theoretically), the robot is supposed to start at the initial conditions that correspond to the signal. To see how this can be handled, suppose that the robot is initially in an arbitrary configuration. When the user starts drawing the signal, in general a jump will happen (from the arbitrary initial point to the first drawn point); this is not realistic, as it would imply a very high (theoritically infinite) speed of the links. However, suppose there was a mechanism that, when the user starts drawing, computes a valid trajectory for the robot to reach the new point. In this simulator, it was decided to suppose that such a mechanism exists, and to consider as valid only the results of the simulations after the initial conditions have been reached. For details on how this is implemented, consult the Simulink model documentation.

3. The simulator can be further developed, for example by creating a complete user interface, detailing the actuator's state (by attaching a frame of reference to it), associating a SimScape model to the Simulink scheme, considering situations with noise, and many other things. Because of time constraints, it was decided to omit these features for now.

0.4 Conventions used in this report

- In the description of the exceptions, the term `caller` stands for the name of the function that originally throwed the exception.

Part I
User interface functions documentation

1 prepare_simulation

1.1 Definition

```
function sim_struct = prepare_simulation (mode, plot_mode, vmax, l1, l2, h_step, tmin, tmax, xs, ys)
```

1.2 Brief description

Returns a data structure containing data that will be used in the simulation (see 1.4), lets the user draw an arbitrary number of reference curves in mode 2, and opens a figure with a drawing of the reference signal in red in mode 1.

1.3 Input parameters

1.3.1 mode

- Type: string (valid values: `OFFLINE_GIVEN`, `OFFLINE_MANUAL`, `ONLINE`).
- Description: `OFFLINE_GIVEN` for mode 1, `OFFLINE_MANUAL` for mode 2, `ONLINE` for mode 3.

1.3.2 plot_mode

- Type: string (valid values: `NO_PLOT`, `PLOT_NO_LINKS`, `PLOT_WITH_LINKS`).
- Description: `NO_PLOT` if no plot is desired during the simulation, `PLOT_NO_LINKS` if only the plot of the robot position is wanted, `PLOT_WITH_LINKS` if at each step the robot's links are to be plotted. First choice is not valid if `mode = ONLINE`.

1.3.3 vmax

- Type: double (positive).
- Description: maximum absolute value of any component of the actuator's velocity.

1.3.4 l1

- Type: double (positive).
- Description: length of link 1 of the robot.

1.3.5 l2

- Type: double (positive).
- Description: length of link 2 of the robot.

1.3.6 h_step

- Type: double (positive).
- Description: sampling step wanted for the simulation (in mode 1, it must be the sampling time of `xs` and `ys`).

1.3.7 tmin (only for mode 1)

- Type: double.
- Description: starting time of `xs` and `ys`.

1.3.8 tmax (only for mode 1)

- Type: double (greather than `tmin`).
- Description: final time of `xs` and `ys`.

1.3.9 xs (only for mode 1)

- Type: row vector of doubles of size equal to that of `ys`.
- Description: x component of the reference signal.

1.3.10 ys (only for mode 1)

- Type: row vector of doubles of size equal to that of `xs`.
- Description: y component of the reference signal.

1.4 Output parameters

1.4.1 sim_struct

A custom data structure with the following fields: `h_step`, `l1`, `l2`, `vmax`, `sw`, `plot_mode_number`, `n`, `tmin`, `tmax`, `q1_initial`, `q2_initial`, `xstar`, `xstardot` (the last 7 ones only in `OFFLINE` modes).

- The first 4 ones are the input parameters of `prepare_simulation` with the same name.
- `sw` is the switching parameter, and has value -1 for mode 3, 1 for mode 1, 2 for mode 2.
- `plot_mode_num` = `plot_mode_str2int(plot_mode)`.
- `n` = number of simulations requested by the user.
- The successive 4 fields are `n`-sized row vectors, where the `i`-th element contains, respectively, the initial time, final time, initial first angle and initial second angle of the `i`-th simulation (`i` = 1,..,`n`).
- `xstar` and `xstardot` are `n`-sized cell array, where the `i`-th element contains, respectively, the signal data (`xstar`) and derivative data (`xstardot`) of the `i`-th reference signal (`i` = 1,..,`n`). Each cell contains three column arrays (first = time instants, second = x component, third = y component), and each column has size $\lfloor \frac{t_{max}-t_{min}}{h_{step}} \rfloor + 1$.

1.5 Other outputs

1.5.1 Figure

In `OFFLINE` modes, a figure is opened, through call to `open_figure` (see II.7). In mode 1, it simply displays the reference signal in red. In mode 2, for an arbitrary number of times, the user is able to draw an arbitrary continuous curve in the figure; at the end of each drawing, a message box is shown, asking the user if he/she wants to draw another signal.

1.5.2 Message box

As described in 1.5.1.

1.6 Throwable custom exceptions

1.6.1 caller:ARG_ERROR

- Cause: a numeric parameter was not in the specified range.
- Error string: depends on the parameter.

1.6.2 caller:MODE_ERROR

- Cause: the `mode` parameter holds an invalid value, or `plot_mode` and `mode` are not compatible.
- Error string: `Given mode is not valid` or `Mode and plot mode are not compatible`.

1.6.3 caller:FIRST_POINT_NOT_REACHABLE_ERROR

- Cause: first point of at least one to-be-prepared simulation is not reachable by the robot.
- Error string: `['First point of reference signal ',num2str(ii),' is not reachable.']`, where `ii` is the index of the indicted simulation.

1.6.4 caller:INFINITE_SIGNAL_ERROR

- Cause: at least one point of at least one to-be-prepared simulation is not finite.
- Error string: `['Reference signal ',num2str(ii),' is not finite.']`, where `ii` is the index of the indicted simulation.

1.6.5 Others

All the exceptions throwable by `prepare_offline_given`, `prepare_offline_manual`, `plot_mode_str2int` (see Part II).

2 start_simulation

2.1 Definition

```
function [sim_time, q1out, q2out, xout, yout, deltax, deltay] = start_simulation (sim_struct)
```

2.2 Brief description

Runs the simulation(s) as indicated by the input structure. In `OFFLINE` modes, there is no user interaction. In the `ONLINE` mode, the user must draw (by cursor) an arbitrary number of signals to be followed in real time.

2.3 Input parameters

2.3.1 sim_struct

A structure as specified by the output of `prepare_simulation` (see 1.4.1). Note that, as pointed out in 1.4.1, the last 7 fields (in the order seen in 1.4.1) are not necessary in `ONLINE` mode (if they are provided, they will simply be ignored). If the output of a call to `prepare_simulation` is given as input to `start_simulation` (as it should be), there is no need to worry about this detail, as everything is handled automatically.

2.4 Output parameters

2.4.1 sim_time

- Column vector of doubles.
- Description: time associated with the simulation, which is the original signal time in mode 1 and the real simulation time in modes 2 and 3.

2.4.2 q1out

- Column vector of doubles.
- Desctiption: array of values of the first generalized coordinate during the simulation.

2.4.3 q2out

- Column vector of doubles.
- Desctiption: array of values of the second generalized coordinate during the simulation.

2.4.4 xout

- Column vector of doubles.
- Desctiption: array of values of the first component of the actuator's position during the simulation.

2.4.5 yout

- Column vector of doubles.
- Desctiption: array of values of the second component of the actuator's position during the simulation.

2.4.6 deltax

- Column vector of doubles.
- Desctiption: array of values of the first component of $\boldsymbol{\delta}\mathbf{x}$ during the simulation.

2.4.7 deltay

- Column vector of doubles.
- Desctiption: array of values of the second component of $\boldsymbol{\delta}\mathbf{x}$ during the simulation.

2.5 Other outputs

2.5.1 Figure

In `ONLINE` mode, a figure is opened, through call to `open_figure` (see II.7). For an arbitrary number of times, the user is able to draw an arbitrary continuous curve in the figure, and the robot (=Simulink model) follows it in real time; at the end of each drawing, a message box is shown, asking the user if he/she start a new simulation.

2.5.2 Message boxes

1. The first kind is as described in 2.5.1.
2. After all simulations end, a message box containing the message `The simulation has ended successfully.` is opened.

2.6 Throwable custom exceptions

2.6.1 caller:INVALID_STRUCT_ERROR

- Cause: an exception of type `MATLAB:nonExistentField` or `MATLAB:structRefFromNonStruct` was thrown, i.e. the input `sim_struct` lacks some field or it was not a structure data type.
- Error string: `The argument sim_struct lacks at least one requested field.`

2.6.2 caller:ARG_ERROR

- Cause: a numeric parameter was not in the specified range.
- Error string: depends on the parameter.

2.6.3 Others

All exceptions throwable by `compute_screen_parameters` (see II.3.6).

2.7 Notes

- A `FOR` loop is used to loop through the simulations. In the `OFFLINE` modes, the number of simulations is provided as the `n` field of `sim_struct`; in the `ONLINE` mode, such number is not known, so a `WHILE` loop, with a `BREAK` condition on local variable `no_more_simulations_needed` becoming 1 (see next note), would be theoretically the most "clean" solution. However, to make the code more compact, it was decided to use a `FOR` loop with an "infinite" number of loops (and the `BREAK` condition stated above). However, such a loop raises the warning `Warning: Too many FOR loop iterations. Stopping after 9223372036854775806 iterations.`; since infinite loops are undoable anyway, it was then opted to use, as number of loops, the number indicated in the warning (9223372036854775806). However, the warning was still issued. The number used in the end was 92233720368547757 2 (which should be enough for all cases).

- A local variable, `no_more_simulations_needed`, is initialized to 0; the loop on the simulations has a `BREAK` condition on `no_more_simulations_needed` becoming 1. This change of value happens when the user refuses to run a new simulation, choosing "No" in the message box displayed by the MATLAB interpreted function `read_cursor_input`. When that happens, the instruction `assignin('caller', 'no_more_simulations_needed', 1);` (executed in `read_cursor_input`) injects the value 1 into `no_more_simulations_needed`. Note that the use of function `assignin` is discouraged as it is slow and hides data dependencies; however, as a special case it was choosen to use it for his compactness and the fact that it is executed at most once in the run of `starting_simulation`.

Part II
Internal functions documentation

3 compute_screen_parameters

3.1 Definition

```
function [screen_width, screen_height, dist_left, dist_lower, dist_right, dist_upper]
   = compute_screen_parameters(f)
```

3.2 Brief description

Computes some parameters that relate the input figure and the machine's screen.

3.3 Input parameters

3.3.1 f

- Type: figure handle.
- Description: an arbitrary figure handle.

3.4 Output parameters

3.4.1 screen_width

- Type: double.
- Description: width of the machine's screen.

3.4.2 screen_height

- Type: double.
- Desctiption: height of the machine's screen.

3.4.3 dist_left

- Type: double.
- Description: distance between the left border of the drawing area and the left border of the screen (as in Figure 1).

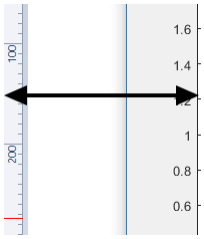


Figure 1: Value of dist_left.

3.4.4 dist_lower

- Type: double.
- Description: distance between the lower border of the drawing area and the lower border of the screen.

3.4.5 dist_right

- Type: double.
- Description: distance between the right border of the drawing area and the right border of the screen.

3.4.6 dist_upper

- Type: double.
- Description: distance between the upper border of the drawing area and the upper border of the screen.

3.5 Other outputs

3.6 Throwable custom exceptions

3.6.1 caller:DELETED_FIGURE_ERROR

- Cause: an `InvalidHandle` exception has been thrown.
- Error string: `The figure (drawing area) must not be closed manually before the end of the running process.`

3.7 Where it is used

1. In online mode, at the first loop it is used by `start_simulation` before the simulation begins (the output values are used in the Simulink model).

4 exit_with_error

4.1 Definition

```
function exit_with_error(id, errstr)
```

4.2 Brief description

Closes all open resources, clears persistent variables in other internal functions, opens a message box pointing the error out and throws an exception in function of the parameters.

4.3 Input parameters

4.3.1 id

- Type: string.
- Description: considering `caller:str2` as the exception-to-be-thrown's identifier, this is `str2`.

4.3.2 errstr

- Type: string.
- Description: the exception-to-be-thrown's error string.

4.4 Output parameters

4.5 Other outputs

4.5.1 Message box

An error message box is opened in the user's screen. The title has the form `caller:id`; the message box text is `errstr`.

4.6 Throwable custom exception

4.6.1 caller:id

- Cause: call to `exit_with_error`.
- Error string: `errstr.`

4.7 Where it is used

Every procedure that needs to exit after an error uses this function.

5 get_signals

5.1 Definition

```
function [xstar, xstardot, q1_initial, q2_initial] = get_signals(h_step, tmin, tmax, xs, ys, l1,
l2, sw)
```

5.2 Brief description

Converts `xs` and `ys`, signals in a row vector format, into a `[time; xs'; ys']` format, where `time` is the time column vector. Also returns the derivatives of `xs` and `ys` in the same format and the initial angles needed for the signal.

5.3 Inputs description

5.3.1 h_step

- Type: double (positive).
- Description: sampling step of `xs` and `ys`.

5.3.2 tmin

- Type: double.
- Description: starting time of `xs` and `ys`.

5.3.3 tmax

- Type: double (greather than `tmin`).
- Description: final time of `xs` and `ys`.

5.3.4 xs

- Type: row vector of doubles of size equal to that of `ys`.
- Description: x component of the reference signal.

5.3.5 ys

- Type: row vector of doubles of size equal to that of `xs`.
- Description: y component of the reference signal.

5.3.6 l1

- Type: double (positive).
- Description: length of link 1 of the robot.

5.3.7 l2

- Type: double (positive).
- Description: length of link 2 of the robot.

5.3.8 sw

- Type: integer (can be -1, 1 or 2).
- Description: simulation switching parameter.

5.4 Output parameters

5.4.1 xstar

- Type: matrix of doubles.
- Description: `[t xs' ys']` where `t=(tmin:h:tmax)'`.

5.4.2 xstardot

- Type: matrix of doubles.
- Description: `[t xsd' ysd']` where `t=(tmin:h:tmax)'`, and `[xsd ysd]` is the numerical derivative of `[xs ys]`, computed with Euler's method.

5.4.3 q1_initial

- Type: double
- Desctiption: initial value of q1 for the reference signal.

5.4.4 q2_initial

- Type: double
- Desctiption: initial value of q2 for the reference signal.

5.5 Other outputs

5.6 Throwable custom exceptions

5.6.1 caller:INVALID_INPUT_SIZE_ERROR

- Cause: parameter sizes are not compatible.
- Error string: `At least one component of the given reference signal has invalid dimension. Each component must be a row vector with floor((tmax-tmin)/h)+1 columns.`

5.7 Where it is used

In `prepare_offline_given` (to convert the user input into the correct format) and in `prepare_offline_manual` (to convert the `drawfreehand` output into the correct format).

5.8 Notes

- When executing the instruction `b=diff(a)`, where `a` is an array of size `n`, `b` turns out to have `n - 1` elements. This is because the `i`-th element of `b` is computed as `b(i) = a(i+1)-a(i)`, and there is no `n + 1`-th element to use. Because of this, the last element of `xsd` and `ysd` is computed separately: if the reference signal has more than one sample, we use a backwards formula, `b(n) = a(n)-a(n-1)`; otherwise, the signal was a constant anyway, so we set the last (and only) element to 0.

6 get_simulation_results

6.1 Definition

`function [sim_time, q1out, q2out, xout, yout, deltax, deltay] = get_simulation_results(sw, q1q2, outs)`

6.2 Brief description

Takes the simulation results in the original Simulink format and converts them into a column vector format.

6.3 Input parameters

6.3.1 sw

- Type: integer (can be -1, 1 or 2).
- Description: simulation switching parameter.

6.3.2 q1q2

- Type: Simulink signal structure with two signals.
- Description: robot generalized coordinates during the simulation.

6.3.3 outs

- Type: Simulink signal structure with six signals.
- Description: components of actuator position, components of $\boldsymbol{\delta}\mathbf{x}$, simulation variable `starting_time`, and real time of the simulation.

6.4 Output parameters

The output are those of `start_simulation` (see I.2.4).

6.5 Other outputs

6.6 Throwable exceptions

6.7 Where it is used

In `start_simulation`, after the end of each simulation.

6.8 Notes

- Call `sim_time_arr` the array of simulation times. In `OFFLINE` modes, the values of the signals during the whole simulation run have to be returned. On the other hand, in `ONLINE` mode, the results that are to be considered are those after the initial conditions have been reached; in this case, therefore, the only values that will be returned are those with index $\geq$ `first_time_index`, where `first_time_index` is such that `sim_time_arr(first_time_index)` is equal to the instant where the initial conditions are reached.

7 open_figure

7.1 Definition

```
function f = open_figure(sw, l1, l2)
```

7.2 Brief description

Closes all open figures, opens and returns a new figure after applying some customization for the simulation (see below).

7.3 Input parameters

7.3.1 sw

- Type: integer (can be -1, 1 or 2).
- Description: simulation switching parameter.

7.3.2 l1

- Type: double (positive).
- Description: length of link 1 of the robot.

7.3.3 l2

- Type: double (positive).
- Description: length of link 2 of the robot.

7.4 Output parameters

7.4.1 f

- Type: figure handle.
- Description: handles a figure with axis limits `[-(l1+l2), l1+l2]` in both coordinates, maximized dimension, square shape if not in online mode, `hold on` active, grid, pixel as axes units. Furthermore, two (one if `l1 = l2`) circles are drawn in black, respectively with radius `l1 + l2` and `abs(l1 - l2)`.

7.5 Other outputs

7.5.1 f

`f` is opened as well as returned.

7.6 Throwable exceptions

7.7 Where it is used

1. In `start_simulation`, `prepare_offline_given`, `prepare_offline_manual`, it is called in order to open the simulation figure for the first time.
2. In `plot_robot_position` and `start_simulation`, it is called if it turns out the figure has been closed during simulation, and must be re-opened.

8 plot_mode_str2int

8.1 Definition

```
function plot_mode = plot_mode_str2int(plot_mode)
```

8.2 Brief description

Converts a valid plot mode string into an integer value.

8.3 Input parameters

8.3.1 plot_mode

- Type: string (valid values: `NO_PLOT`, `PLOT_NO_LINKS`, `PLOT_WITH_LINKS`).
- Description: `NO_PLOT` if no plot is desired during the simulation, `PLOT_NO_LINKS` if only the plot of the robot position is wanted, `PLOT_WITH_LINKS` if at each step the drawing of the robot links are wanted.

8.4 Output parameters

8.4.1 plot_mode

- Integer.
- Description: 0 if input = `NO_PLOT`, 1 if input = `PLOT_NO_LINKS`, 2 if input = `PLOT_WITH_LINKS`.

8.5 Other outputs

8.6 Throwable custom exceptions

8.6.1 caller:PLOT_MODE_ERROR

- Cause: input has an invalid value.
- Error string: `Given plot mode is not valid`.

8.7 Where it is used

At the beginning of `prepare_simulation`.

9 pos2angle

9.1 Definition

```
function [q1, q2] = pos2angle (x, y, l1, l2)
```

9.2 Brief description

Given the actuator position and the length of the links, the corresponding robot angles are returned.

9.3 Input parameters

9.3.1 x

- Type: double.
- Desctiption: first component of actuator's position.

9.3.2 y

- Type: double.
- Desctiption: second component of actuator's position.

9.3.3 l1

- Type: double (positive).
- Desctiption: length of link 1 of the robot.

9.3.4 l2

- Type: double (positive).
- Desctiption: length of link 2 of the robot.

9.4 Output parameters

9.4.1 q1

- Type: double.
- Description: first generalized coordinate associated with `(x, y)`.

9.4.2 q2

- Type: double.
- Description: second generalized coordinate associated with `(x, y)`.

9.5 Other outputs

9.6 Throwable custom exceptions

9.7 Where it is used

In `get_signals` and `read_cursor_input`, in order to compute the initial conditions for the Simulink model (if `x`, `y` are reachable).

10 prepare_offline_given

10.1 Definition

```
function [n, q1_initial, q2_initial, xstar, xstardot, sw]
   = prepare_offline_given(l1, l2, h_step, tmin, tmax, xs, ys, plot_mode)
```

10.2 Brief description

In `OFFLINE_GIVEN` mode, assigns 1 to n and `sw` and computes the other output parameters through a call to `get_signals`. Afterwards, it opens a figure and plots the reference signal in red.

10.3 Input, output parameters

Input and output parameters are those of `prepare_sim`, from where this function is called in case the `OFFLINE_GIVEN` mode is requested.

10.4 Other outputs

10.4.1 Figure

A figure is opened through a call to `open_figure`. The reference signal is plotted on it in red, as an xy curve.

10.5 Throwable custom exceptions

10.5.1 caller:ARG_ERROR

- Cause: `tmin > tmax`.
- Error string: `tmin cannot be greater than tmax.`

10.5.2 Other

The exceptions throwable in `get_signals` can be thrown.

11 prepare_offline_manual

11.1 Definition

```
function [n, q1_initial, q2_initial, tmin, tmax, xstar, xstardot, sw]
   = prepare_offline_manual(l1, l2)
```

11.2 Brief description

In `OFFLINE_MANUAL` mode, opens a figure and lets the user draw an arbitrary number of continuous curves by using the `drawfreehand` function. At the end of each drawing, a message asks the user if he/she wants to draw another curve. The number of simulations and 1, respectively, are assigned to `n` and `sw`.

11.3 Input, output parameters

Input and output parameters are those of `prepare_sim`, from where this function is called in case the `OFFLINE_MANUAL` mode is requested.

11.4 Other outputs

11.4.1 Figure

A figure is opened through call to `open_figure` if `plot_mode` is positive.

11.4.2 Message box

At the end of each drawing, a message box asks the user if he/she wants to draw another curve.

11.5 Throwable custom exceptions

11.5.1 caller:DELETED_FIGURE_ERROR

- Cause: a `MATLAB:class:InvalidHandle` exception is thrown by the `drawfreehand` function as a result of closing the figure during the drawing process.
- Error string: `The figure (drawing area) must not be closed manually before the end of the running process.`

11.5.2 Others

The exceptions throwable in `get_signals` can be thrown.

11.6 Notes

- To obtain the curve drawn by the user, a call to `drawfreehand`, a MATLAB system function, is performed. Since this function returns all the plotted points after the call has terminated, i.e. after the plot is completed, it could not be used in the `ONLINE` mode, where the drawn points are needed as soon as possible by the simulator. However, it was decided to use it here, and not to use the custom routine implemented for the `ONLINE` mode, as `drawfreehand` is more precise and more performing.
- If the figure is closed during the simulation, `drawfreehand` returns a null object; therefore the process has to be stopped with error (see 11.5.1).

Part III
Simulink model documentation

After an introduction, each of the following sections is associated to an area of the Simulink model.

12 Introduction

The model uses a set of variables; each one is associated to a GOTO block. In particular, `l1`, `l2`, `sw`, `vmax`, `tmin`, `plot_mode` are used in all modes with obvious role, as well as the screen parameters. Other variables with obvious meaning include `xy`, `xstar`, `xstardot`, `q1`, `q2`, q, etc.

The variable `ended` is computed at every step in the Reference signal acquisition area, and is valued 0 before the simulation has ended, 1 afterwards. Since the non-abnormal end of the simulation is issued as an input of 1 to the `STOP` block, this parameter can be used by the other areas in the remaining execution of the last step of simulation.

The variables `q1iws`, `q2iws` are the initial angles computed in the workspace (ws = workspace), and are used in `OFFLINE` modes only.

The variable `starting_time` is computed at every step in the Achievement of initial position area, and holds value `tmin` in `OFFLINE` modes; in `ONLINE` mode, it has value -1 before the initial conditions are reached, and their time of achievement afterwards.

The variable `reset` is computed at every step in the Reference signal acquisition area, and is always valued 0 in `OFFLINE` modes. In `ONLINE` mode, it is valued 0 before the user has started drawing, and 1 afterwards.

IMPORTANT: the Simulink model is run from the `start_simulation` function (through a call to the `sim` function), and the simulation parameters, including the simulation step, are given as parameters there. HOWEVER, don't put different settings in the Simulink configuration parameters settings. For example, calling `sim` with a fixed step parameter AND setting the solver to `auto` in the solver configuration parameters settings will cause Simulink to set as actual simulation step the GCD of the `sim` parameter and a Simulink-generated step. This WILL cause confusion, so be careful.

13 Initial assignments area

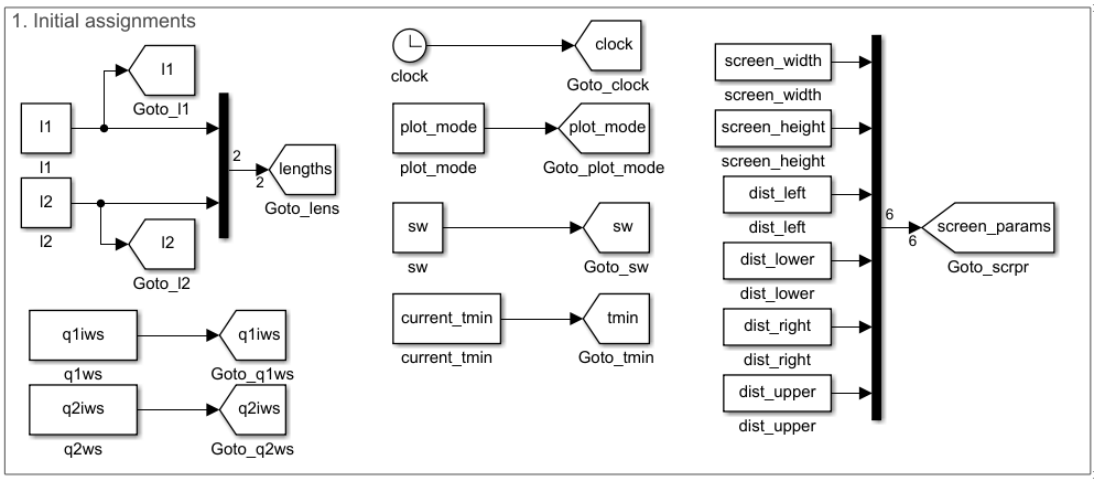


Figure 2: Initial assignments8 area.

In this area several parameters (`l1`, `l2`, `q1ws`, `q2ws`, `clock`, `plot_mode`, `sw`, `current_tmin` and all the screen parameters) from the workspace are assigned to a `GOTO` block. This way, if one or moreof those parameters get their name changed in a future development of the software, the model has to be changed in one place only.

14 Reference signal acquisition area

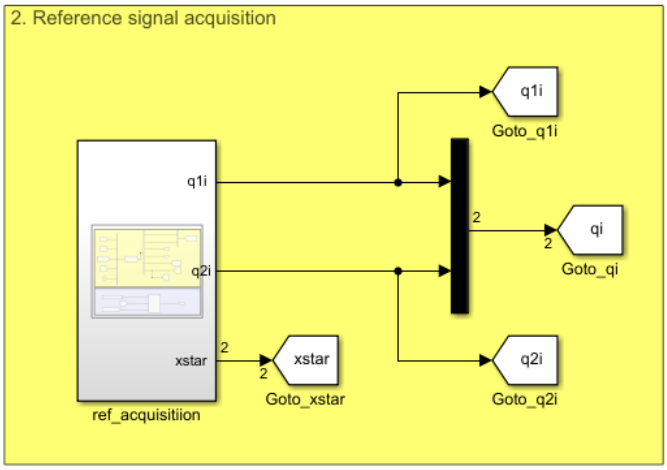


Figure 3: Reference signal acquisition.

The `ref_acquisition` subsystem is composed of three areas: the Online reference processing area, the Choice of Reference area and the vmax computation area.

14.1 Online reference processing area

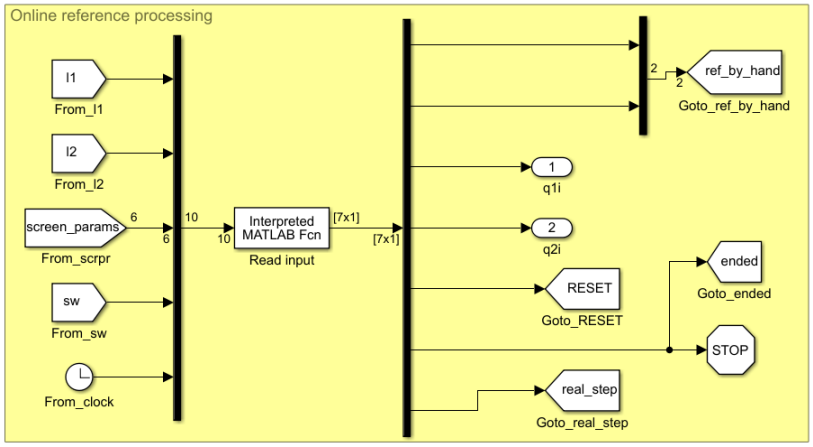


Figure 4: `ref_acquisition` subsystem.

In `ONLINE` mode, this area reads and stores the cursor input in the `GOTO` block `ref_by_hand`, as well as assigning a few other variables (see 1.4.1.1).

14.1.1 MATLAB interpreted function read_cursor_input

Definition `function out = read_cursor_input (in)`.

Output parameters The output signal `out` is a column array of size 6. In `OFFLINE` modes, this function is not used, therefore the outputs always have a value of 0. In `ONLINE` mode, its elements are the following (listed from first to last):

- `xstar`, `ystar` = zero vector before the user starts drawing; the two components of the reference signal, provided by the user through the use of the cursor, afterwards.

- `q1i`, `q2i` = respectively 0 and 4 (arbitrary values) before the user starts drawing; the initial angles associated with the reference signal drawn by the user after the user has started drawing.
- `RESET` = 0 before the user starts drawing, 1 afterwards. The change of value of this variable happens the first time that the mouse results clicked and the clicked point is inside the drawing area.
- `ended` = 0 before the simulation has terminated, 1 afterwards. The change of value of this variable happens if the mouse is not clicked and `RESET` is equal to 1, i.e. the mouse has been released by the user.
- `real_step` = 0 in `OFFLINE` modes; in `ONLINE` mode, it is equal to the difference of the time at which the latest point was drawn and the time at which the previous point was drawn.

Other outputs After the user has started drawing, i.e. when `RESET > 0`, the points provided by the user through the cursor are plotted in red on the opened simulation figure.

Custom exceptions An exception of id `caller:FIRST_POINT_NOT_REACHABLE_ERROR` and error string `First point of the reference signal is not reachable.` can be thrown if the first point plotted by the user is outside of the reachability space.

Note At implementation time, a convenient way to capture the position of the cursor at each instant was seeked. Consider the command `get(gca, 'CurrentPoint');`; it returns the cursor's coordinates with respect to the current axes, but only returns the last clicked or unclicked point, ignoring the points over which the cursor hovered. Oddly enough, the command `get(0, 'PointerLocation');` returns the points with respect to the screen coordinates, but at all times, not only when the user clicks or unclicks. The second instruction is therefore used; however, the obtained point must be converted to the figure/axes coordinates, and this is the reason for which the values computed by `compute_screen_parameters` are provided as input to `read_cursor_input`. The formulas to convert the points were obtained through a simple transformation of coordinates.

For instance, consider the x coordinate. Define $L := l_1+l_2$, $d_L :=$ `dist_left`, $d_R :=$ `dist_right`, $w :=$`screen_width`. To transform the value x_s (relative to screen coordinates) to x_a (relative to axes coordinates), we need the straight line going through the points ($x_s = d_L$, $x_a = -2L$) and ($x_s = w - d_R$, $x_a = 2L$), that is, the straight line satisfying the equations

$$-2L = m_x d_L + q_x$$

$$2L = m_x(w - d_R) + q_x$$

Solving for m_x and q_x easily gives $m_x = \frac{4L}{w-d_R-d_L}, q_x = -2L - \frac{4L}{w-d_R-d_L}d_L$, which are the relationships used in the program (rearranged in order to avoid multiple computations of $\frac{4L}{w-d_R-d_L}$). For the y coordinate, an analogous procedure (or noticing the simmetry of the problem) yields $m_y = \frac{4L}{h-d_D-d_U}, q_y = -2L - \frac{4L}{h-d_D-d_U}d_D$, where $h =$ `screen_height`, $d_D =$ `dist_lower`, $d_U =$ `dist_upper`.

14.2 Choice of reference area

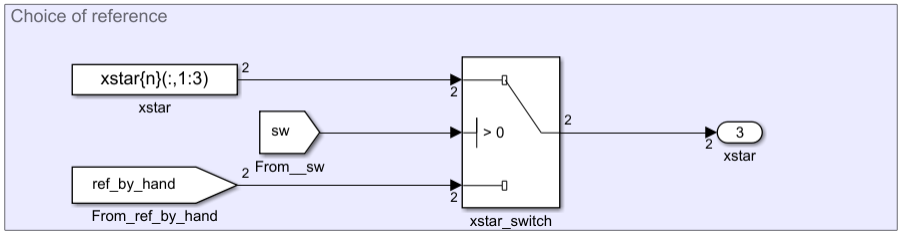


Figure 5: Choice of reference area.

In `ONLINE` mode, this area assigns the `ref_by_hand` signal (the cursor input) to `xstar`; in `OFFLINE` modes, the `n`-th cell of `xstar` (workspace variable) is assigned instead.

14.3 vmax computation area

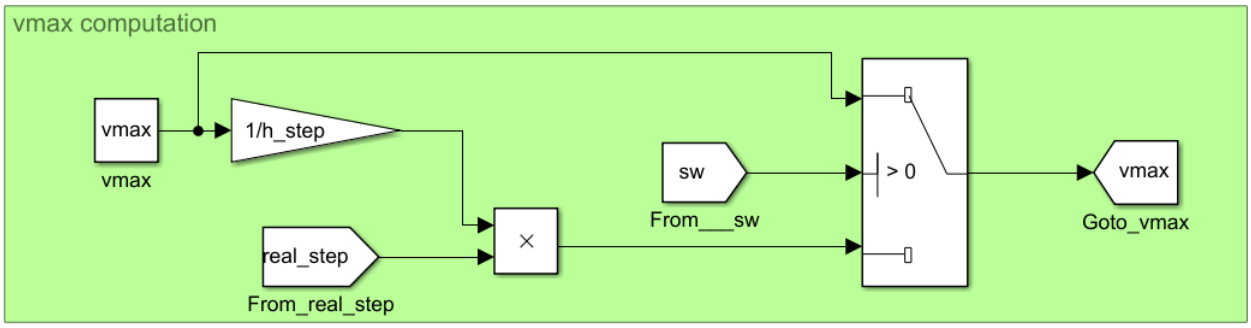


Figure 6: vmax computation area.

In `OFFLINE` modes, vmax is simply imported from the workspace.

In `ONLINE` mode, it is multiplied by the positive quantity `real_step/h_step`. This is because the internal clock of the simulation is `h_step`, but the actual step time at each simulation pass `real_step`. (usually `h_step < real_step`). The maximum speed in m/s is `vmax`; if one simulation step corresponded with one real second, we'd simply have to divide by `h_step`; however, one simulation step corresponds to one real step, i.e. it contains `real_step` seconds.

15 xstardot computation area

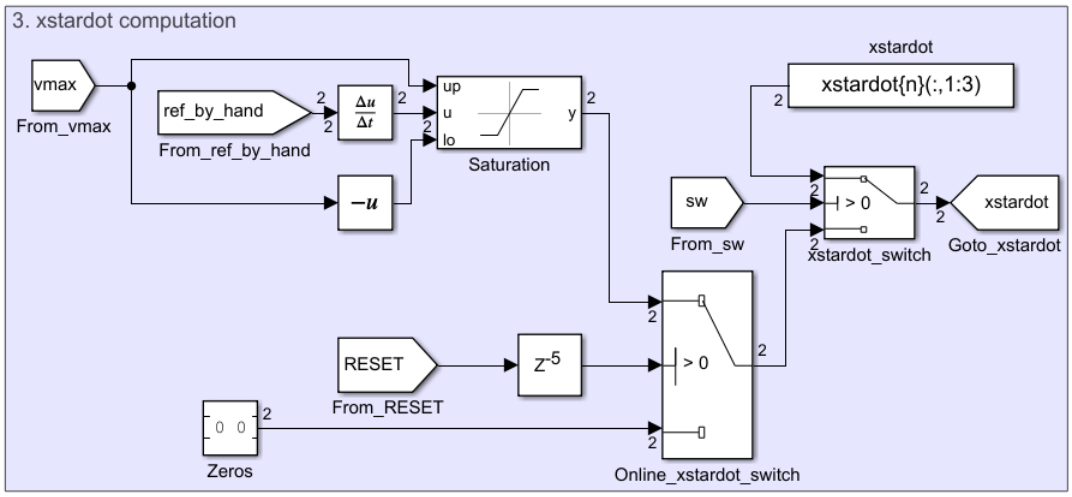


Figure 7: xstardot computation area.

This area selects the derivative of the reference. If `sw` > 0, the n-th element of the workspace cell array `xstardot` is trivially chosen. In the other case, it is more complicated.

For now, ignore the delay block. Before `RESET` is set to a positive value, i.e. before the user has started drawing, a null derivative is provided (any finite signal would do, as this part is going to be ignored). Roughly after `RESET` changes sign, a saturated version of the derivative of the hand(cursor)-drawn reference signal is provided instead.

The saturation allows the simulation not to crash on the user when the latter exceeds the maximum speed of the robot (however, the robot will proceed at its maximum velocity, forcing the user to slow down). Finally, the delay block is necessary because, when the user starts drawing, a jump will happen, i.e. the derivative of the cursor-drawn signal will be very high. We want the robot to start as still, so we provide null derivative for a few more time steps.

16 qdot computation area

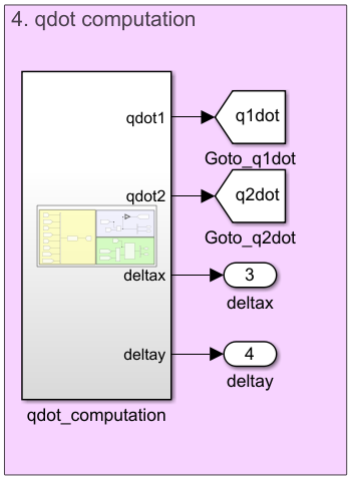


Figure 8: qdot computation area.

The `qdot_computation` subsystem faithfully implements the robot control logic seen in the introduction, with `K` = 10×`I`. Internally, it has three areas: the Inversion of J area, the K×Deltax computation area and the Final computation area.

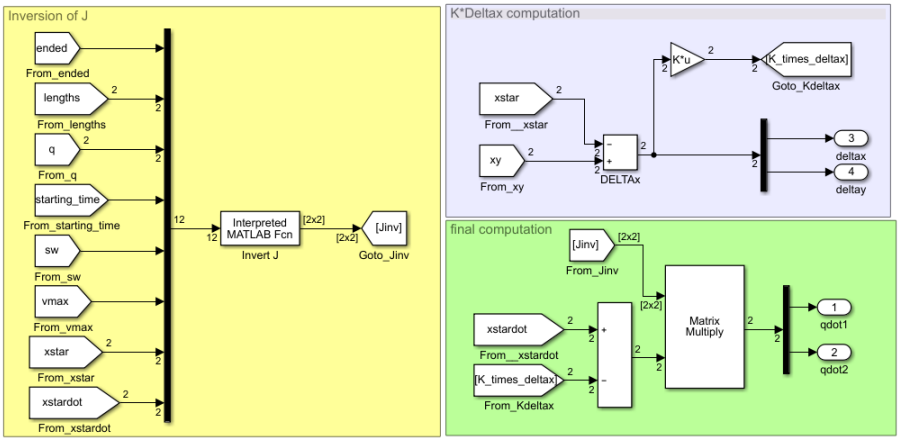


Figure 9: qdot_computation subsystem

As said, the calculations are those explained in the introduction.

16.0.1 Interpreted MATLAB function invert_J

If inputs `xstar`, `ystar`, `xstardot` and `ystardot` specify a valid signal (that is, not exceeding maximum speed `vmax` and being reachable), the output is the inverse of the Jacobian of the system. Otherwise, an exception is thrown; it can be of two types:

1. id = `caller:MAX_SPEED_EXCEEDED_ERROR`, error text = `Maximum speed exceeded.`
2. id = `caller:POINT_NOT_REACHABLE_ERROR`, error text = `Point is not reachable.`

17 Arm area

The Arm subsystem uses `qdot`, computed in the qdot computation area, to calculate the actuator's position (multiplexed into `xy`) and the associated angles (multiplexed into `q`).

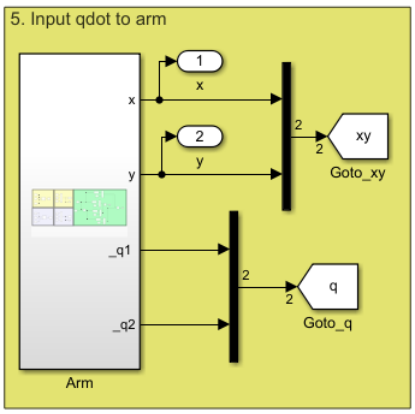


Figure 10: Arm area.

17.1 Arm subsystem

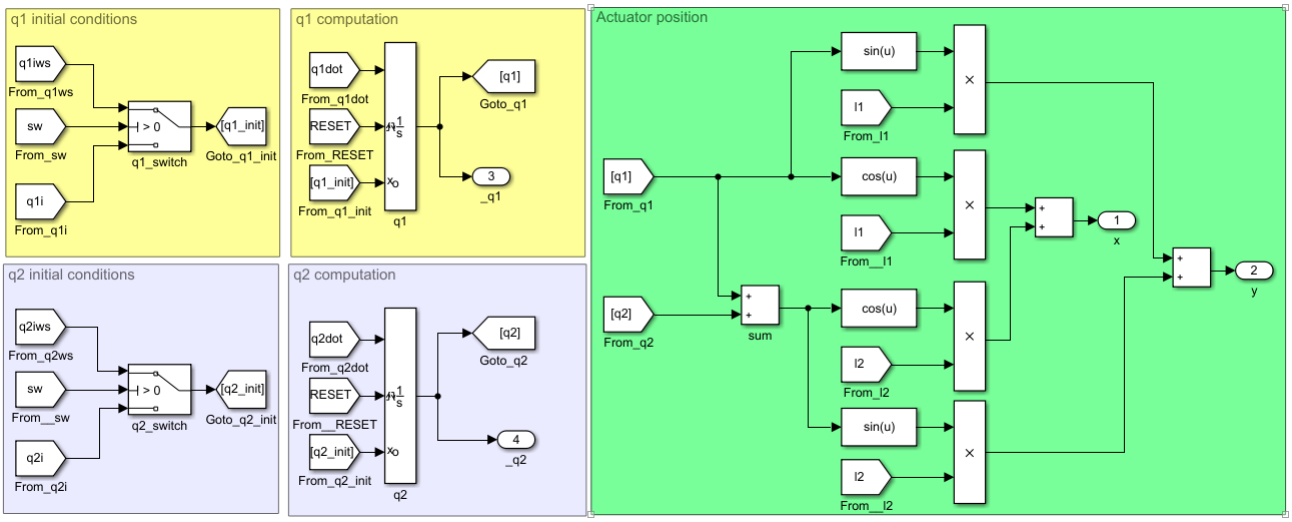


Figure 11: Arm subsystem.

17.1.1 Initial conditions areas

The two leftmost areas select the correct initial angles. If `sw` > 0, i.e. the simulation is in an `OFFLINE` mode, the signals `q1i` and `q2i`, computed during the simulation in the Reference signal acquisition area, are chosen. Otherwise, the signals `q1iws` and `q2iws`, computed before the simulation in the workspace, are chosen.

17.1.2 Angles computation

The remaining areas perform the integration of the derivatives of `q1` and `q2`, in order to obtain `q1` and `q2`, which are then returned as third and fourth outputs.

The integrators reset to the initial condition signals `q1_init` and `q2_init` when the signal `RESET` changes sign; in the `OFFLINE` modes this never happens, while in the `ONLINE` mode this happens after the user has started drawing, and `q1_init`, `q2_init` have been assigned to the needed initial angles by the `read_user_input` function (see 17.1).

17.1.3 Actuator position area

This area outputs the components of the actuator's position, `x` and `y`, after being computed from the angles `q1` and `q2` according to the relationships

$$x = l_1 \cos q_1 + l_2 \cos(q_1 + q_2)$$

$$y = l_1 \sin q_1 + l_2 \sin(q_1 + q_2)$$

18 Time handling area

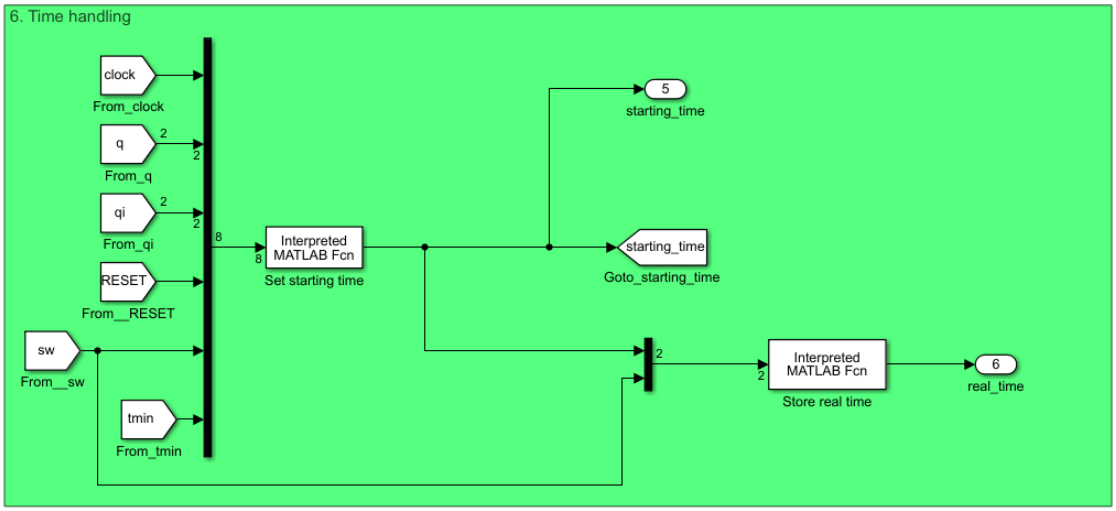


Figure 12: Achievement of initial position area.

18.1 MATLAB interpreted function set_starting_time

18.1.1 Definition

```
function starting_time = set_starting_time(in)
```

18.1.2 Output parameters

1. `starting_time` = `tmin` in `OFFLINE` modes. In `ONLINE` mode, `starting_time` = -1 if the initial condition have yet to be reached; otherwise, `starting_time` = the instant in which they have been reached. In `ONLINE` mode, the value of `starting_time` is used in the other parts of the model to determine if the initial condition have been reached.

18.1.3 Other outputs

18.1.4 Throwable custom exceptions

18.2 Interpreted MATLAB function store_real_time

18.2.1 Definition

```
function t = store_real_time (in)
```

18.2.2 Output parameters

1. `t` = 0 in mode 1 (this output is not significant). In mode 2, `t = toc`. In mode 3, `t = toc` if `tic` has been called (i.e. `starting_time` $\geq$ 0, see 14.1), otherwise `t` = 0.

18.2.3 Other outputs

18.2.4 Throwable custom exceptions

18.2.5 Notes

- In the `OFFLINE` modes, `set_starting_times` runs `tic` at the first simulation step, so `store_toc` (which executes after `set_starting_times` as it needs its output as input) can execute the `toc` command always. In `ONLINE` mode, `tic` is only run by `set_starting_times` at the step where initial conditions have been reached, so `toc` can't be executed until then, and 0 will be returned instead.

19 Plot area

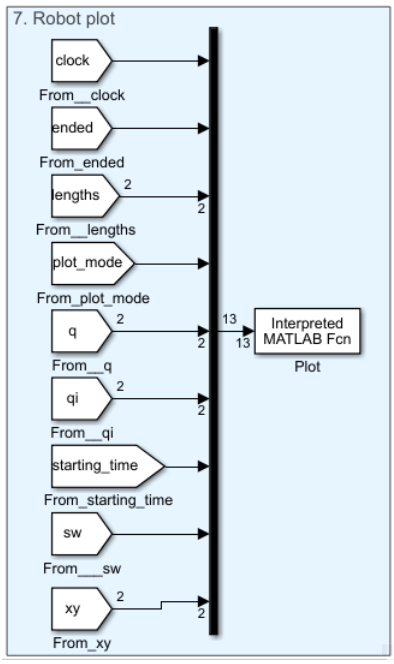


Figure 13: Plot area.

19.1 MATLAB interpreted function plot_robot_position

19.1.1 Definition

```
function plot_robot_position(in)
```

19.1.2 Output parameters

19.1.3 Other outputs

- If `plot_mode = 1` (and, in `ONLINE` mode, if `starting_time` ≥ 0), the function plots, in blue, a straight line that goes from the previous position of the actuator to the current one, if the two positions are different; it draws a point if they are equal, and at the first step (when a "previous position" doesn't exist).
- If `plot_mode = 2` (and, in `ONLINE` mode, if `starting_time` ≥ 0), the function plots, at every step, two lines in the position of the robot's links, in blue. It also deletes the links drawn at the previous step, if any.

- Otherwise, it does nothing.
- If the figure is closed by the user while the simulation is running and `plot_mode` is not 0, the function re-opens the figure and issues a warning, with message `It is recommended not to close the figure during the simulation.`

19.1.4 Throwable custom exceptions

19.1.5 Notes

Note 1 This function, when plotting is enabled, is the bottleneck of the whole simulator, and dominates the execution time. For example, with $t_{min} = 0$, $t_{max} = 6$ s, $h_{step} = 0.001$, $\mathbf{x}^*(t) = (0.7\cos(10t), 0.7\sin(10t) + 0.01)$, $v_{max} = 100$, $l_1 = 1$, $l_2 = 0.8$, the simulation takes:

- 10.931683 seconds with `plot_mode = PLOT_WITH_LINKS`.
- 10.247887 seconds with `plot_mode = PLOT_NO_LINKS`.
- 1.670445 seconds with `plot_mode = NO_PLOT`.

Although qualitatively, these results show the fact that the plotting heavily affects the running time. To avoid even more overhead, in the case of `plot_mode = PLOT_WITH_LINKS` the two links are plotted through a single instruction.

Note 2 This function uses a local persistent variable, `plotted`, to store the latest element plotted in the figure in `PLOT_WITH_LINKS` plot mode. At each call, if the variable is not empty, i.e. after the first links configuration is plotted, the command `delete(plotted)` is issued, in order to erase the previous links configuration plot.

In `start_simulation`, after each simulation, the functions that use persistent variables are cleared (i.e. their persistent variables are reset to empty values), except for this one. This is because, after the end of a simulation, if there is a next one plotted will contain the latest links configuration of the previous simulation, and will be automatically erased.

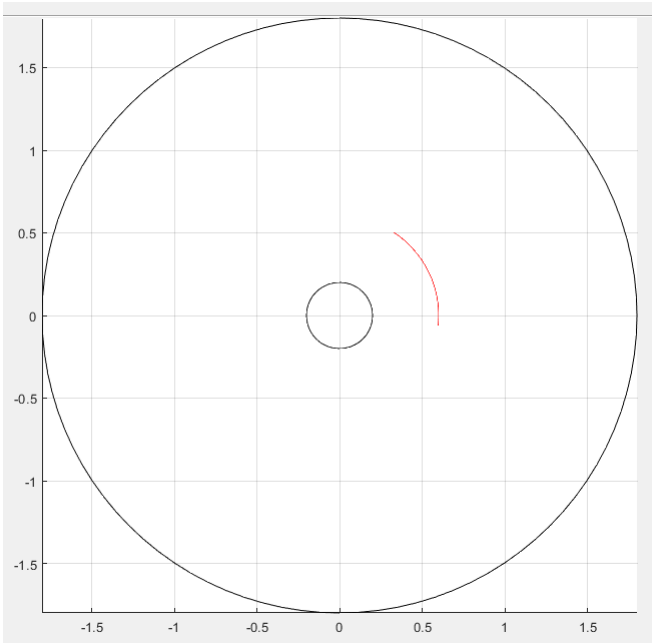
Note that the command `clear plot_robot_position` must be issued before `start_simulation` returns (and, to cover the case of error exit, in `exit_with_error`, too).

Part IV
Examples of use

20 Example 1

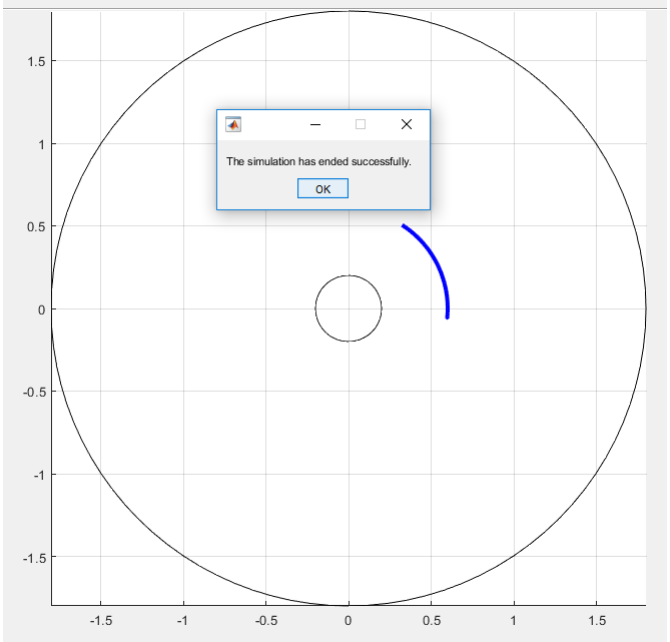
```
tmin=-0.1; tmax=1; h_step=0.001; t=tmin:h_step:tmax;
sim_struct = prepare_simulation('OFFLINE_GIVEN', 'PLOT_NO_LINKS', 1, 1, 0.8, h_step, tmin, tmax,
0.6*cos(t), 0.6*sin(t));
```

Output:



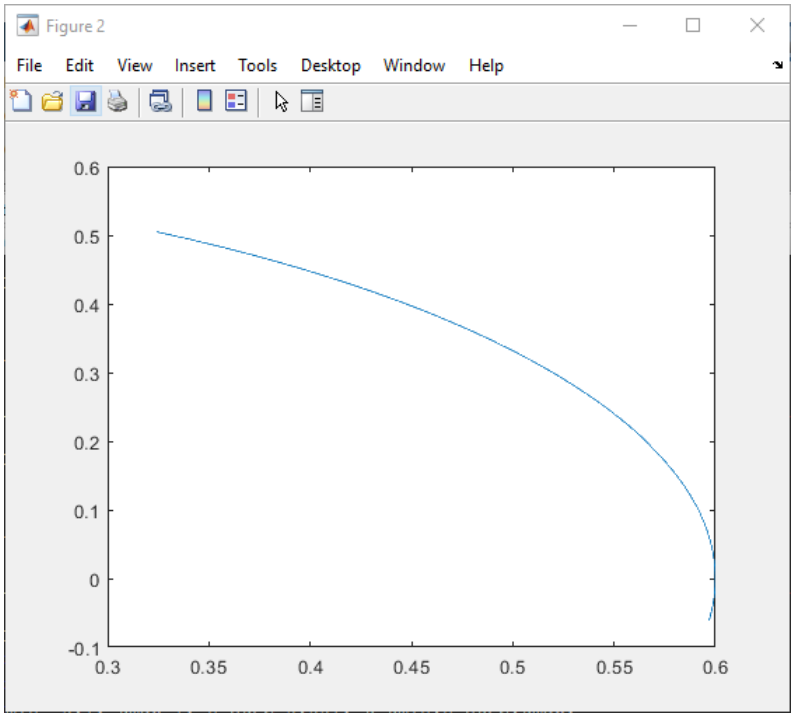

```
[t, q1, q2, x, y, deltax, deltay] = start_simulation(sim_struct);
```

Output:




```
figure(2); plot(x{1}, y{1});
```

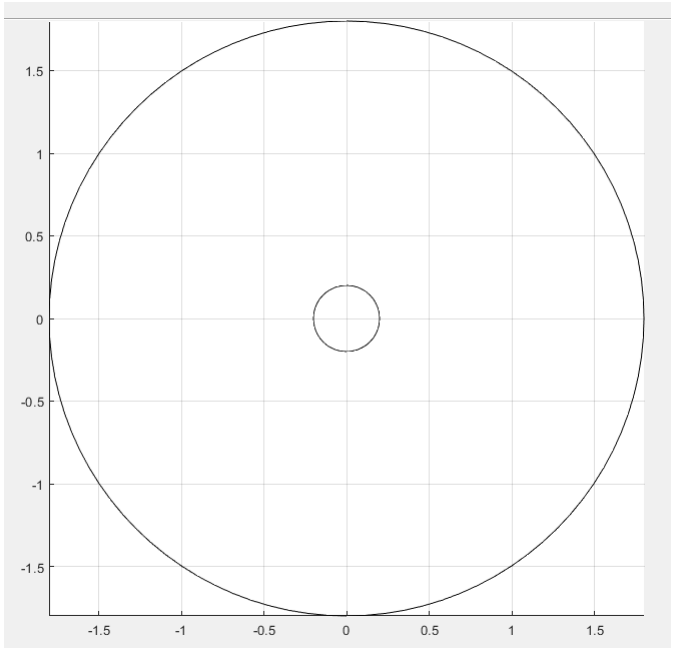
Output:



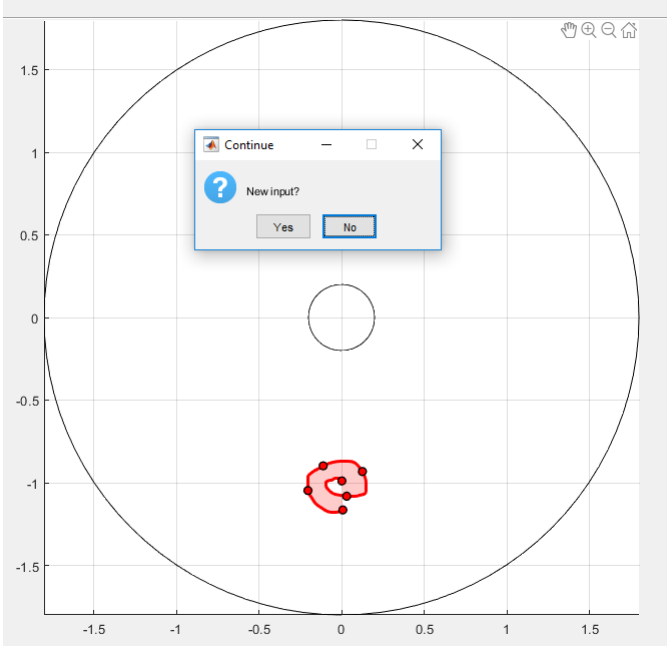
21 Example 2

```
sim_struct = prepare_simulation('OFFLINE_MANUAL', 'PLOT_WITH_LINKS', 1, 1, 0.8, 0.001);
```

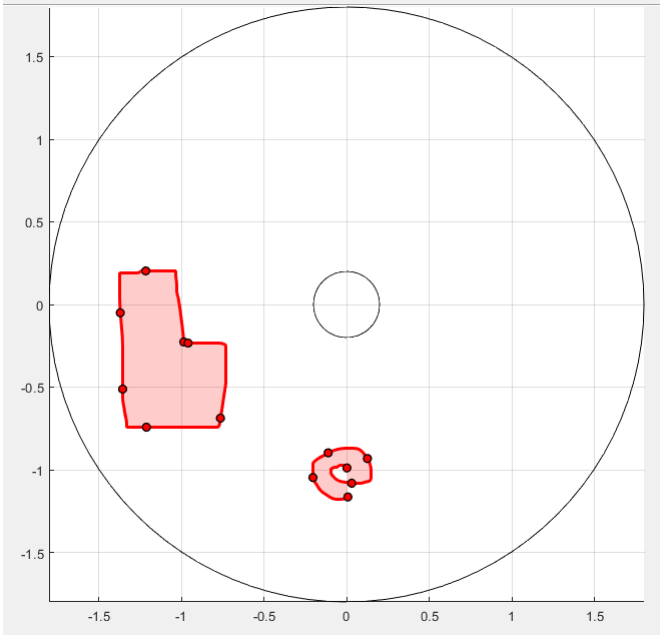
Output:



After drawing one signals:

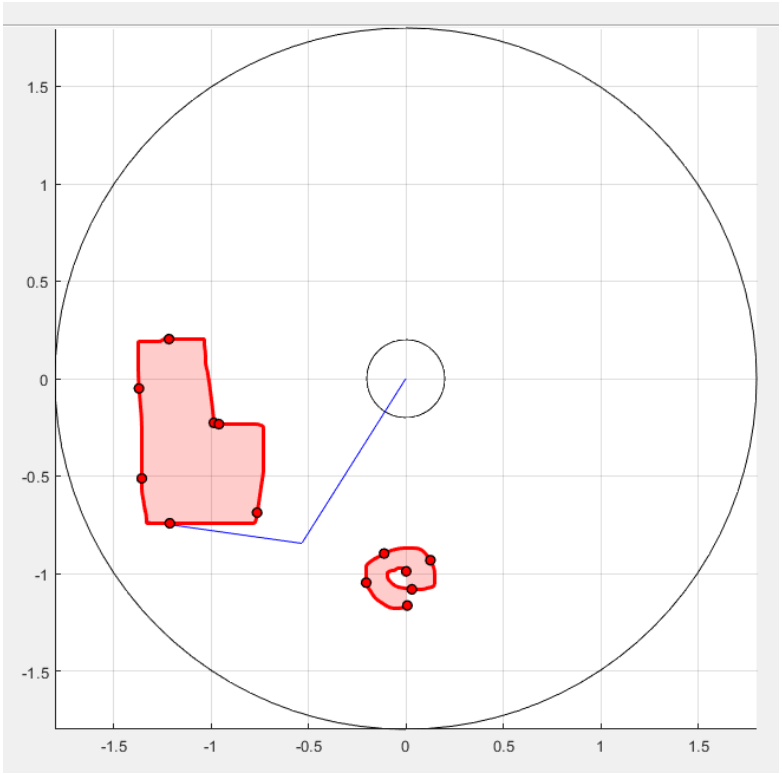


After choosing “Yes”, drawing another signal and choosing “No”:



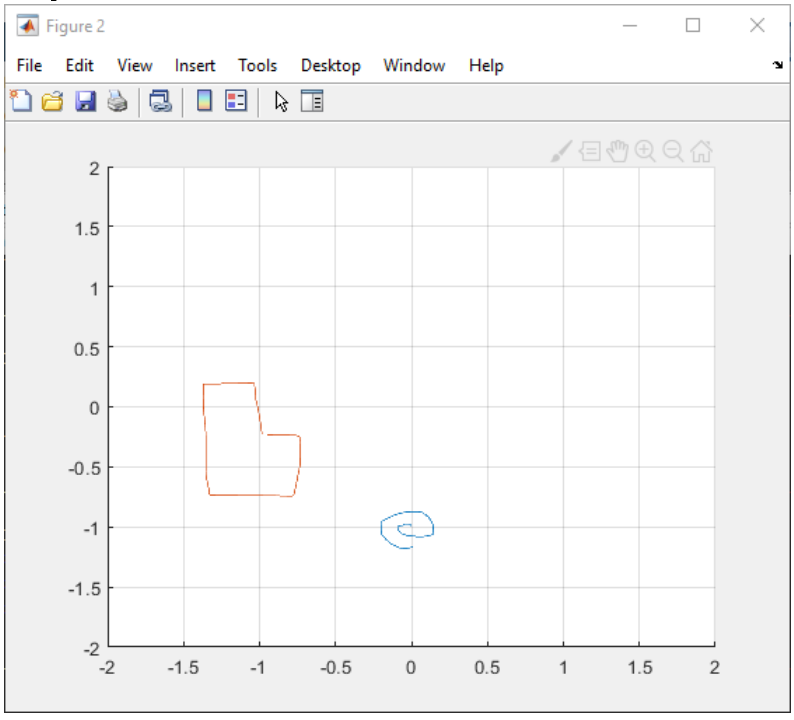
```
[t, q1, q2, x, y, deltax, deltay] = start_simulation(sim_struct);
```

While plotting:




```
figure(2); hold on; grid on; axis([-2 2 -2 2]); plot(x{1}, y{1}, x{2}, y{2});
```

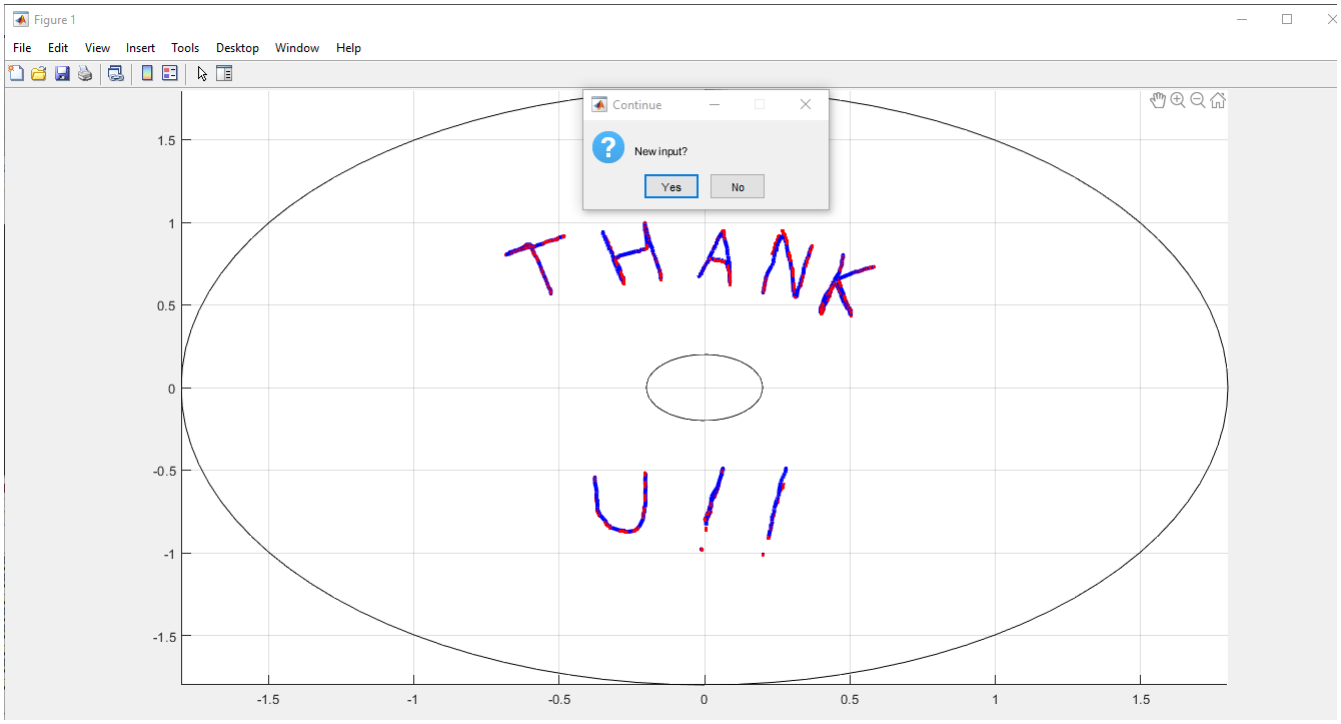
Output:



22 Example 3

```
sim_struct = prepare_simulation('ONLINE', 'PLOT_NO_LINKS', 1, 1, 0.8, 0.00001);
[t, q1, q2, x, y, deltax, deltay] = start_simulation(sim_struct);
```

Output after plotting some curves:




```
figure(2); hold on; for ii = 1 :  max(size(x)) plot(x{ii}, y{ii},'.-'); end
```

Output: