03_The_Animator

April 17, 2025

1 Tutorial 03: The Animator

1.1 Tutorial Description

This tutorial covers use the animator interface to create a real time plot of data during a simulation. We will demonstrate this by creating a phase-space plot of a forced pendulum.

1.2 Imports

To begin, we import the same modules for the same reasons as tutorial 00.

[ ]:
```
from condynsate.simulator import Simulator as con_sim
from condynsate import __assets__ as assets
```

1.3 Building the Project Class

We now create a `Project` class with `__init__` and `run` functions. In `__init__` a pendulum is loaded using the same technique as tutorial 02. Additionally, the animator is set up the plot the phase diagram of the pendulum while the simulation is running. In `run`, we cover how send state data to the animator.

[ ]:
```
class Project():
    def __init__(self):
        '''
        ######################################################################
        This time we want to use the animator, so we set animation to
        True.
        ######################################################################
        '''
        # Create a instance of the simulator
        self.s = con_sim(animation = True)

        # Load the pendulum in the orientation we want
        self.pendulum = self.s.load_urdf(urdf_path = assets['pendulum'],
                                         position = [0., 0., 0.05],
                                         yaw = 1.571,
                                         wxyz_quaternion = [1., 0., 0., 0],
                                         fixed = True,
                                         update_vis = True)
```

```
# Set the initial angle of the pendulum joint
self.s.set_joint_position(urdf_obj = self.pendulum,
                          joint_name = 'chassis_to_arm',
                          position = 0.698,
                          initial_cond = True,
                          physics = False)

'''
#####################################################################
Once our URDF is loaded and initial conditions are set, we move
on to creating the animator window.
condynsate.simulator.add_subplot is how we tell the animator that
we want to add a subplot to our animation GUI. We may call this
function as many times as we like and each time a new subplot
will be added to the animation GUI. The arguments to this
function are as follows:
    n_artists : int, optional
        The number of artists that draw on the subplot
        The default is 1.
    subplot_type: either 'line' or 'bar', optional
        The type of plot. May either be 'line' or 'bar'. The
        default is 'line'.
    title : string, optional
        The title of the plot. Will be written above the
        plot when rendered. The default is None.
    x_label : string, optional
        The label to apply to the x axis. Will be written
        under the subplot when rendered. The default is None.
    y_label : string, optional
        The label to apply to the y axis. Will be written to
        the left of the subplot when rendered. The default is
        None.
    x_lim : [float, float], optional
        The limits to apply to the x axis of the subplot. A
        value of None will apply automatically updating
        limits to the
        corresponding bound of the axis. For example
        [None, 10.] will fix the upper bound to exactly 10,
        but the lower bound will freely change to show all
        data. The default is [None, None].
    y_lim : [float, float], optional
        The limits to apply to the y axis of the subplot.
        A value of None will apply automatically updating
        limits to the corresponding bound of the axis. For
        example [None, 10.] will fix the upper bound to
        exactly 10, but the lower bound will freely change
```

```
            to show all data.The default is [None, None].
        h_zero_line : boolean, optional
            A boolean flag that indicates whether a horizontal line
            will be drawn on the y=0 line. The default is false
        v_zero_line : boolean, optional
            A boolean flag that indicates whether a vertical line
            will be drawn on the x=0 line. The default is false
        colors : list of matplotlib color string, optional
            A list of the color each artist draws in. Must have
            length n_artists. If n_artists = 1, has the form
            ['COLOR']. When None, all artists will default to drawing
            in black. The default is None.
        labels : list of strings, optional
            A list of the label applied to each artist. For line
            charts, the labels are shown in a legend in the top right
            of the plot. For bar charts, the labels are shown on the
            y axis next to their corresponging bars. Must have length
            n_artists. If n_artists = 1, has the form ['LABEL'].
            When None, no labels will be made for any aritsts. The
            default is None.
        line_widths : list of floats, optional
            The line weigth each artist uses. For line plots, this is
            the width of the plotted line, for bar charts, this is
            the width of the border around each bar. Must be length
            n_artists. If n_artists = 1, has the form [LINE_WIDTH].
            When set to None, defaults to 1.0 for all lines. The
            default is None.
        line_styles : list of matplotlib line style string, optional
            The line style each artist uses. For line plots, this is
            the style of the plotted line, for bar charts, this
            argument is not used and therefore ignored. Must be
            length n_artists. If n_artists = 1, has the form
            ['LINE_STYLE']. When set to None, defaults to 'solid'
            for all lines. The default is None.
        tail : int, optional
            Specifies how many data points are used to draw a line.
            Only the most recently added data points are kept. Any
            data points added more than tail data points ago are
            discarded and not plotted. Only valid for line plots, and
            applied to all artists in the plot. For bar plots, this
            argument is ignored and not used. A value of None means
            that no data is ever discarded and all data points added
            to the animator will be drawn. The default is None.

    The function returns:
        subplot_index : int
            A integer identifier that is unique to the subplot
```

```
            created. This allows future interaction with this subplot
            (adding data points, etc.).
        artist_inds : tuple of ints
            A tuple of integer identifiers that are unique to the
            artist created. This allows future interaction with these
            artists (adding data points, etc.).

    subplot_index can be considered a pointer to a specific
    subplot. If you want to make edits to a subplot, you must first
    identify which subplot you are modifying. This is done with
    subplot_index. Each subplot can have multiple artists. You can
    think of an artist as a pen. If you want to draw two lines on a
    subplot at the same time, you will need two pens, i.e. two
    artists. Each time you add a subplot, you will receive a list
    of artist_inds. You can again think of this as a list of
    pointers to each artists in that subplot.
    ########################################################################
    '''
    # Make plot for phase space
    self.p, self.a = self.s.add_subplot(n_artists = 1,
                                        subplot_type = 'line',
                                        title = "Phase Space",
                                        x_label = "Angle [Deg]",
                                        y_label = "Angle Rate [Deg / Sec]",
                                        colors = ["k"],
                                        line_widths = [2.5],
                                        line_styles = ["-"],
                                        x_lim = [-40.,40],
                                        y_lim = [-275.,275],
                                        h_zero_line = True,
                                        v_zero_line = True)

    '''
    ########################################################################
    Once we are done adding subplots to the animator, we open the
    animator GUI.
    ########################################################################
    '''
    # Open the animator GUI
    self.s.open_animator_gui()

def run(self, max_time=None):
    '''
    ########################################################################
    This run function does all the same basic functions as in
    tutorial 02 but with the added functionality of real time
    animation of the phase of the pendulum.
```

```
        ######################################################################
        '''
        # Reset the simulator.
        self.s.reset()

        # Await run command.
        self.s.await_keypress(key = 'enter')

        # Run the simulation loop until done
        while(not self.s.is_done):
            # Get the pendulum's joint state
            state = self.s.get_joint_state(urdf_obj = self.pendulum,
                                           joint_name = 'chassis_to_arm')

            # Get the angle and angular velocity of the pendulum
            angle = 180. * state['position'] / 3.142
            angle_vel = 180. * state['velocity'] / 3.142

            # Apply a proportional torque
            torque = -angle - 0.01*angle_vel
            self.s.set_joint_torque(urdf_obj = self.pendulum,
                                    joint_name = 'chassis_to_arm',
                                    torque = torque,
                                    show_arrow = True,
                                    arrow_scale = 0.02,
                                    arrow_offset = 0.05)

            '''
            ######################################################################
            This is how we modify a subplot in real time. Essentially, we
            identify which artist of which subplot we would like to draw
            a data point, and then we specify that data point. Remember
            artist_inds, which is returned by
            condynsate.simulator.add_subplot, is ALWAYS A TUPLE.
            Therefore, you will need to reference which artist you would
            like to use, even if there is only one artist.
            ######################################################################
            '''
            # Add (angle, angle_vel) point to subplot self.p
            self.s.add_subplot_point(subplot_index = self.p,
                                     artist_index = self.a[0],
                                     x = angle,
                                     y = angle_vel)

            '''
            ######################################################################
            As usual, at the bottom of the run function we step the
```

```
            simulation.
            ##################################################################
            '''
            self.s.step(max_time=max_time)
```

1.4 Running the Project Class

Now that we have made the `Project` class, we can test it by initializing it and then calling the `run` function. Remember to press the enter key to start the simulation and the esc key to end the simulation.

[ ]:
```
# Create an instance of the Project class.
proj = Project()

# Run the simulation.
proj.run(max_time = None)
```

1.5 Challenge

This tutorial is now complete. For an additional challenge, think of how you might create another subplot that plots both the angle and angular velocity as a function of time.