00_Introduction

April 17, 2025

1 Tutorial 00: Introduction

1.1 Tutorial Description

This tutorial covers creating the backend of a new project using condynsate. In this tutorial, we will cover how to: 1. Import physics objects in the condynsate physics environment. 2. Test that physics objects behave as expected.

We will accomplish these goals by importing a cube 1 meter above a solid ground plane, starting the physics engine, then observing the dynamics of the cube. Note that this project will not coverhow to create your own .URDF files. Instead, we recommend reviewing https://wiki.ros.org/urdf.

1.2 Imports

To begin, we import the required dependencies. In general, for projects that will simulate the physics of a set of urdf objects, the only module needed from the condynsate project is `condynsate.simulator`. Note that we also import `condynsate.__assets__` so that we can load and use a condynsate default cube .URDF file.

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

To see what default .URDF files are available to us, we can list the keys of `assets`:

[ ]:
```
assets.keys()
```

1.3 Building the Project Class

The `Project` class is that object that front end users interact with. In its simplest form, the `Project` class contains two class functions: `__init__` and `run`. In `__init__` the physics engine is initialized, .URDF files are imported into the engine, and initial conditions are set. In `run` initial conditions are reset, and the physics engine is turned on.

[ ]:
```
class Project():
    def __init__(self):
        '''
        ##################################################################
        STEP 1: Initialize an instance of the condynsate.simulator class.
        Note that for this project, we do not want to use animation.
        This means that we do not want to plot things in real time.
```

```python
    To avoid condynsate from assuming that we want to do this and
    running some things in the background that take compute power,
    we simply set the animation flag in the initialization function
    to be false.
    ###################################################################
    '''
    self.s = con_sim(animation = False)

    '''
    ###################################################################
    STEP 2: Add solid ground to the physics environment. Condynsate
    already includes several default .URDF files that are listed in
    the  condynsate.__assets__ variable. To load any one of these,
    we simply  select the path to the asset we want and call the
    load_urdf function.

    load_urdf has 5 arguments. urdf_path tells the function where the
    .URDF file is that we want to load. For the ground we will use
    'plane_big'.

    position and wxyz_quaternion define the position and orientation
    in which the URDF object will be placed. These positions and
    orientations are defined around the parent axes of the URDF
    object. By setting position = [0, 0, 0] and
    wxyz_quaternion = [1, 0, 0, 0], we place the plane at (0, 0, 0)
    aligned to the XY plane.

    By setting fixed = True, we are telling the simulator NOT to
    update the physics of the base of the object. It will still have
    collision, but no other forces, including gravity, will affect it.

    By setting update_vis = False, we are telling the simulator NOT
    to send updates to the visualizer for this object. This flag is
    usually set to False for unjointed, fixed URDFs to reduce the
    amount of compute power required by the visualization
    ###################################################################
    '''
    self.ground = self.s.load_urdf(urdf_path = assets['plane_big'],
                                   position = [0., 0., 0.],
                                   wxyz_quaternion = [1., 0., 0., 0],
                                   fixed = True,
                                   update_vis = False)

    '''
    ###################################################################
    STEP 3: Load a cube into the simulator as a physics object. For
    this we will use the 'cube' asset.
```

```python
        By setting position = [0, 0, 1.5] and
        wxyz_quaternion = [1, 0, 0, 0], we place the cube at (0, 0, 1.5)
        in the orientation defined by the URDF parent axes. As it turns
        out, the cube is 1x1x1, so that a center position of [0, 0, 1.5]
        places the bottom of the cube exactly 1 meter above the ground.

        By setting fixed = False, we are telling the simulator apply
        physics to this .URDF when the engine is running.

        By setting update_vis = True, we are telling the simulator to
        send updates to the visualizer for this object. This is usually
        set to  true when an object is not fixed and will change its
        position and/or orientation.
        ###################################################################
        '''
        self.cube = self.s.load_urdf(urdf_path = assets['cube'],
                                     position = [0., 0., 1.5],
                                     wxyz_quaternion = [1., 0., 0., 0],
                                     fixed = False,
                                     update_vis = True)

    def run(self, max_time=None):
        '''
        ###################################################################
        Now that we have created an initialization function, we move on
        to the run function. This function will run a physics simulation
        using condynsate. Essentially, we will do three things:
            1) Reset the simulation environment to the initial
               conditions. Whereas this will not do anything for this
               particular example, it is best practice to always call
               condynsate.simulator.reset before running a simulation
               loop.
            2) Wait for a user to press enter. This is acheived with the
               condynsate.simulator.await_keypress(key='enter') function
               call.
            3) Run a simulation to completion. In this case, completion
               is defined as the simulator reaching max_time or the user
               pressing the 'esc' key. We will describe how to do this
               using a while loop and condynsate.simulator.step below.
        ###################################################################
        '''

        '''
        ###################################################################
        STEP 1: Reset the simulator. It is best practice to do this at
        the beginning of every run function.
```

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

        '''
        ################################################################
        STEP 2: Tell the run function to wait for the user to press enter
        before continuing to the simulation loop. We do this using
        await_keypress. This function call makes it so that the simulator
        will fully suspend until the user presses the enter key. Note
        that this is not a necessary function to call. If you want the
        simulation to run as soon as Project.run is called, exclude this
        function call.
        ################################################################
        '''
        # Await run command.
        self.s.await_keypress(key = 'enter')

        '''
        ################################################################
        STEP 3: Run the simulation loop. condynsate.simulator cannot run
        an entire simulation by itself. Instead, it can take one time
        step of 0.01 seconds. Therefore, the way we run an entire
        simulation is to place the condynsate.simulator.step function
        inside a while loop that executes until the boolean flag
        condynsate.simulator.is_done is True. Note that this flag will
        automatically be set to true when max_time is reached or the user
        presses the esc key. If the max_time argument to  is None,
        is_done will notn be set to true until the esc key is pressed.
        Further, if max_time is None AND keyboard interactivity is
        disabled, the is_done boolean flag will be set to true at 10.0
        seconds.
        ################################################################
        '''
        # Run the simulation loop until done
        while(not self.s.is_done):
            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 added challenge, think of how you would modify `__init__` so that two cubes, one above the other, are loaded but only the top one has physics applied to it.