01_External_Forces

April 17, 2025

1 Tutorial 01: External Forces

1.1 Tutorial Description

This tutorial covers creating the backend of a project using condynsate in which external forces are applied to the center of mass of an unjointed .URDF object. We will cover: 1. Appling forces and torques to the center of mass of an object. 2. Measuring the position, orientation, velocity, and rotational velocity of the center of mass of that object.

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__` solid ground and a sphere are loaded using the same technique as tutorial 00. In `run`, we cover how to read the position and orientation of the sphere and apply external forces based on the sphere's state.

[ ]:
```
class Project():
    def __init__(self):
        # Create a instance of the simulator
        self.s = con_sim(animation = False)

        # Load the ground
        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)

        # Load the sphere so that it's resting on the ground
        self.sphere = self.s.load_urdf(urdf_path = assets['sphere'],
                                       position=[0., 0., 0.5],
                                       wxyz_quaternion=[1., 0., 0., 0],
                                       fixed = False,
```

```
                                        update_vis = True)


    def run(self, max_time=None):
        '''
        ########################################################################
        This run function does all the same basic functions as in
        tutorial 00 but with the added functionality of applying external
        forces to the sphere during the simulation loop
        ########################################################################
        '''
        # 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):
            '''
            ####################################################################
            First we want to measure the state (position, orientation,
            velocity, and angular vel) about the center of mass of the
            sphere. For a given URDF object that has already been loaded
            into the physics environment, we can measure this by using
            the condynsate.simulator.get_base_state function. There are
            two arguments to this function:
                1) urdf_obj: the unique URDF object ID that is returned
                   when condynsate.simulator.load_urdf is called
                2) body_coords: A boolean flag that indicates whether the
                   velocity and angular velocity is given in world coords
                   (False) or body coords (True). World coords are
                   defined around the axes defined in the URDF file.
            In this case, we want the world coords, so we set body_coords
            to False

            state, which is returned by
            condynsate.simulator.get_base_state has the following form:
            state : a dictionary with the following keys:
                'position' : array-like, shape (3,)
                    The (x,y,z) world coordinates of the base of the
                    urdf.
                'roll' : float
                    The Euler angle roll of the base of the urdf
                    that define the body's orientation in the world.
                    Rotation of the body about the world's x-axis.
                'pitch' : float
```

```
            The Euler angle pitch of the base of the urdf
            that define the body's orientation in the world.
            Rotation of the body about the world's y-axis.
        'yaw' : float
            The Euler angle yaw of the base of the urdf
            that define the body's orientation in the world.
            Rotation of the body about the world's z-axis.
        'R of world in body' : array-like, shape(3,3):
            The rotation matrix that takes vectors in world
            coordinates to body coordinates. For example,
            let V_inB be a 3vector written in body coordinates.
            Let V_inW be a 3vector written in world coordinates.
            Then: V_inB = R_ofWorld_inBody @ V_inW
        'velocity' : array-like, shape (3,)
            The linear velocity of the base of the urdf in
            either world coords or body coords. Ordered as
            either (vx_inW, vy_inW, vz_inW) or
            (vx_inB, vy_inB, vz_inB).
        'angular velocity' : array-like, shape (3,)
            The angular velocity of the base of the urdf in
            either world coords or body coords. Ordered as either
            (wx_inW, wy_inW, wz_inW), or
            (wx_inB, wy_inB, wz_inB). When written in world
            coordinates, exactly equal to the roll rate, the
            pitch rate, and the yaw rate.
    ##################################################################
    '''
    # Get the base state of the sphere
    state = self.s.get_base_state(urdf_obj = self.sphere,
                                  body_coords = False)

    '''
    ##################################################################
    Suppose we wanted to apply an upward force to the center of
    mass of the sphere if it is less than 1.0 meters above the
    ground. To do this, we would need to first measure its height
    off of the ground. We can do this by extracting the height
    from its state.
    ##################################################################
    '''
    # Extract the height of the center of mass of the sphere
    position = state['position']
    height = position[2]

    '''
    ##################################################################
    Now we write an if statement that applies a force to the
```

```
        center of mass if the height is less than 1.0 and applies 0
        force if the height is greater than 1.0. To apply a force, we
        use the condynsate.simulator.apply_force_to_com function.
        This function has six arguments:
            urdf_obj : URDF_Obj
                A URDF_Obj to which the force is applied.
            force : array-like, shape (3,)
                The force vector in either world or body coordinates
                to apply to the body.
            body_coords : bool, optional
                A boolean flag that indicates whether force is
                given in body coords (True) or world coords (False).
                The default is False.
            show_arrow : bool, optional
                A boolean flag that indicates whether an arrow
                will be rendered on the CoM to visualize the applied
                force. The default is False.
            arrow_scale : float, optional
                The scaling factor that determines the size of the
                arrow. The default is 0.4.
            arrow_offset : float, optional
                The amount by which the drawn force arrow will be
                offset from the center of mass along the direction
                of the applied force. The default is 0.0.

        In this case, we want to draw the force arrow so we set
        show_arrow to True and adjust arrow_scale and arrow_offset
        until the size and position of the arrow look correct,
        respectively.
        ################################################################
        '''
        # Apply an upward force if low
        if height <= 1.0:
            self.s.apply_force_to_com(urdf_obj = self.sphere,
                                      force = [0.,0.,20.],
                                      body_coords = False,
                                      show_arrow = True,
                                      arrow_scale = 0.05,
                                      arrow_offset = 0.5)

        # Apply no forces if high
        else:
            self.s.apply_force_to_com(urdf_obj = self.sphere,
                                      force = [0.,0.,0.])

        '''
        ################################################################
```

```
Suppose we also wanted to apply a torque about the center of
mass of the sphere near the top of its trajectory. To do this
we would need to measure not only its height, but also its
upward speed. We can do this by extracting the upward speed
from its state.
######################################################################
'''
# Extract the upward speed of the center of mass of the sphere
velocity = state['velocity']
upward_speed = abs(velocity[2])

'''
######################################################################
Now we write an if statement that applies a torque to the
center of mass if the height is more than 1.0 and its upward
speed is less than 2.0 in magnitude. Otherwise it applies 0
torque. To apply a torque about the center of mass of a URDF
object, we use the
condynsate.simulator.apply_external_torque function. This has
five arguments:
    urdf_obj : URDF_Obj
        A URDF_Obj to which the torque is applied.
    torque : array-like, shape(3,)
        The torque vector in world coordinates to apply to
        the body.
    body_coords : bool, optional
        A boolean flag that indicates whether torque is
        given in body coords (True) or world coords (False).
        The default is False.
    show_arrow : bool, optional
        A boolean flag that indicates whether an arrow will
        be rendered on the com to visualize the applied
        torque. The default is False.
    arrow_scale : float, optional
        The scaling factor that determines the size of the
        arrow. The default is 0.1.
    arrow_offset : float, optional
        The amount by which the drawn torque arrow will be
        offsetnfrom the center of mass along the direction
        of the applied torque. The default is 0.0.

In this case, we want to draw the torque arrow so we set
show_arrow to True and adjust arrow_scale and arrow_offset
until the size and position of the arrow look correct,
respectively.
######################################################################
'''
```

```
            # Apply torque at top of trajectory
            if height > 1.0 and upward_speed < 2.0:
                self.s.apply_external_torque(urdf_obj = self.sphere,
                                             torque = [0.,0.,0.1],
                                             body_coords = False,
                                             show_arrow = True,
                                             arrow_scale = 25.0,
                                             arrow_offset = 0.0)

            # Apply no torque at bottom of trajectory
            else:
                self.s.apply_external_torque(urdf_obj = self.sphere,
                                             torque = [0.,0.,0.])

            '''
            ################################################################
            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 added challenge, think of how you would modify the Project.run() function so that the force applied is porportional to the magnitude of the velocity of the sphere.