02_Joint_Forces

April 17, 2025

1 Tutorial 02: Joint Forces

1.1 Tutorial Description

This tutorial covers creating the backend of a project using condynsate that applies torques to specific joints of a .URDF object. In this tutorial, we will cover: 1. Applying torques to continuous joints of a .URDF object. 2. Measuring the position and velocity of joints of a .URDF 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__` a pendulum is loaded using the same technique as tutorial 00. In `run`, we cover how to read the state of the joint of the pendulum and apply torques to the joint based on its state.

[1]:
```
class Project():
    def __init__(self):
        # Create a instance of the simulator
        self.s = con_sim(animation = False)

        # 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)

        '''
        ######################################################################
        After loading our pendulum, we want to set its joint so that it
        is at a slight angle. This is done using the
        condynsate.simulator.set_joint_position function. It has 5
```

```
        arguments:
        urdf_obj : URDF_Obj
            A URDF_Obj that contains that joint whose position is being
            set.
        joint_name : string
            The name of the joint whose position is set. The joint name
            is
            specified in the .urdf file.
        position : float, optional
            The position in rad to be applied to the joint.
            The default is 0..
        physics : bool, optional
            A boolean flag that indicates whether physics based
            poisiton controller will be used to change joint position
            or whether the joint position will be reset immediately
            to the target position with zero end velocity. The
            default is False.
        initial_cond : bool, optional
            A boolean flag that indicates whether the position set
            is an initial condition of the system. If it is an
            initial condition, when the simulation is reset using
            backspace, the joint position will be set again.

        Note that because this is an intial condition, we set the
        initial_cond flag to true.
        ######################################################################
        '''
        # Set the initial angle of the pendulum joint
        self.s.set_joint_position(urdf_obj = self.pendulum,
                                  joint_name = 'chassis_to_arm',
                                  position = 0.175,
                                  initial_cond = True,
                                  physics = False)


    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 joint torques
        forces to the pendulum 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, and velocity)
            of the pendulum joint. The name of the pendulum joint set in
            the .URDF file is "chassis_to_arm". For a given URDF object
            with joints that has already been loaded into the physics
            environment, we can measure this by using the
            condynsate.simulator.get_joint_state function. There are two
            arguments to this function:
                1) urdf_obj : URDF_Obj
                   A URDF_Obj whose joint state is being measured.
                2) joint_name : string
                   The name of the joint whose state is measured. The
                   joint name is specified in the .urdf file.

            state, which is returned by
            condynsate.simulator.get_joint_state has the following form:
            state : dictionary with the following keys
                'position' : float
                    The position value of this joint.
                'velocity' : float
                    The velocity value of this joint.
                'reaction force' : list shape(3,)
                    These are the joint reaction forces. Only read if
                    a torque sensor is enabled for this joint.
                'reaction torque' : list shape(3,)
                    These are the joint reaction torques. Only read if
                    a torque sensor is enabled for this joint.
                'applied torque' : float
                    This is the motor torque applied during the last
                    stepSimulation. Note that this only applies in
                    VELOCITY_CONTROL and POSITION_CONTROL. If you use
                    TORQUE_CONTROL then the applied joint motor torque is
                    exactly what you provide, so there is no need to
                    report it separately.
            ################################################################
            '''
            # Get the pendulum joint ("chassis_to_arm") state
            state = self.s.get_joint_state(urdf_obj = self.pendulum,
                                           joint_name = 'chassis_to_arm')


            '''
```

```
'''
######################################################################
Suppose we wanted to apply a returning torque whenever the
pendulum angle is above some threshold. To do this, we would
need to first measure the angle of the pendulum. We can do
this by extracting the position (angle for a continuous
joint) from its state.
######################################################################
'''
# Extract angle of the pendulum joint
angle = 180. * state['position'] / 3.141592654


'''
######################################################################
Now we write an if statement that applies a returning torque
to the joint if its angle is greater than 5.0 degrees. To
apply a torque about a joint, we use the
condynsate.simulator.set_joint_torque function. This function
has six arguments:
    urdf_obj : URDF_Obj
        A URDF_Obj that contains that joint whose torque is
        being set.
    joint_name : string
        The name of the joint whose torque is set. The joint
        name is specified in the .urdf file
    torque : float, optional
        The torque in NM to be applied to the joint. The
        default is 0..
    show_arrow : bool, optional
        A boolean flag that indicates whether an arrow will
        be rendered on the link 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 to offset the drawn arrow along the
        joint axis. 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.
######################################################################
'''
# If positive angle, apply negative torque
if angle > 5.:
```

```
                self.s.set_joint_torque(urdf_obj = self.pendulum,
                                        joint_name = 'chassis_to_arm',
                                        torque = -10,
                                        show_arrow = True,
                                        arrow_scale = 0.05,
                                        arrow_offset = 0.05)

            # If negative angle, apply positive torque
            elif angle < -5.:
                self.s.set_joint_torque(urdf_obj = self.pendulum,
                                        joint_name = 'chassis_to_arm',
                                        torque = 10,
                                        show_arrow = True,
                                        arrow_scale = 0.05,
                                        arrow_offset = 0.05)

            # If low angle magnitude, apply no torque
            else:
                self.s.set_joint_torque(urdf_obj = self.pendulum,
                                        joint_name = 'chassis_to_arm',
                                        torque = 0.0,
                                        show_arrow = True)

            '''
            ###################################################################
            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() to implement a PD controller.