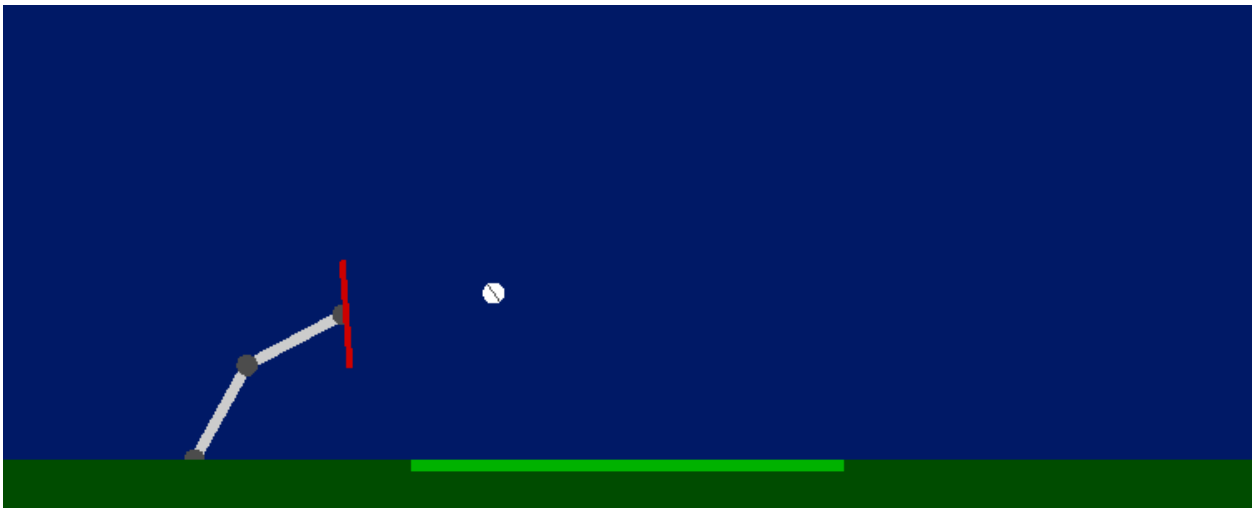
Roboball2d Documentation

Release 0.1.07

Nicolas Guetler and Vincent Berenz

Oct 11, 2022

Contents



CHAPTER 1

Overview

Roboball2d is a lightweight python API to simulated 3 dofs torque controlled robot(s) and ball(s), along with a renderer. It is based on Box2d and pyglet. It can be used for anything you want, but was created with reinforcement learning in mind.

Roboball2d has tested on ubuntu 18.04 and python 3.5.2.

1.1 Installation

```
# pip install roboball2d
```

1.2 Getting Started

- *Roboball2d Example* - A commented example of usage

After installation, you may also run the demos:

```
# roboball2d_demo
# roboball2d_balls_demo
# roboball2d_rendering_demo
# roboball2d_mirroring_demo
```

1.3 Source Code

- https://github.com/intelligent-soft-robots/roboball2d

1.4 License Information

Roboball2d is licensed under the BSD-3-Clause. See the LICENSE file for more information.

- https://github.com/intelligent-soft-robots/roboball2d/blob/master/LICENSE

1.5 Authors and Maintainers

- Nico Gürtler
- Vincent Berenz

Intelligent Soft Robots Laboratory, Max Planck Institute for Intelligent Systems

CHAPTER 2

Roboball2d Example

Here the source code of roboball2d_demo. You may find the source of other demos:

- https://github.com/intelligent-soft-robots/roboball2d/tree/master/roboball2d/demos

```
import math,random,time

from roboball2d.physics import B2World
from roboball2d.robot import DefaultRobotConfig
from roboball2d.robot import DefaultRobotState
from roboball2d.robot import PDController
from roboball2d.ball import BallConfig
from roboball2d.ball_gun import DefaultBallGun


def run(rendering=True):

    """
    Runs the balls demo, in which the robot moves using a PD controller
    as a ball bounces around.
    You may run the executable roboball2d_demo after install to
    see it in action.

    Parameters
    ----------

    rendering :
        renders the environment if True

    """


    if rendering:
        from roboball2d.rendering import PygletRenderer
        from roboball2d.rendering import RenderingConfig
```

(continues on next page)

(continued from previous page)

```
    # configurations, using default
    robot_config = DefaultRobotConfig()
    ball_config = BallConfig()
    visible_area_width = 6.0
    visual_height = 0.05

    # physics engine
    world = B2World(robot_config,
                    ball_config,
                    visible_area_width)

    # graphics renderer
    if rendering:
        renderer_config = RenderingConfig(visible_area_width,
                                          visual_height)
        renderer = PygletRenderer(renderer_config,
                                  robot_config,
                                  ball_config)


    # ball gun : specifies the reset of
    # the ball (by shooting a new one)
    # Here using default ball gun
    ball_gun = DefaultBallGun(ball_config)

    # basic PD controller used to compute torque
    controller = PDController()
    references = [-math.pi/8.0,-math.pi/8.0,-math.pi/8.0]

    # init robot state : specifies the reinit of the robot
    # (e.g. angles of the rods and rackets, etc)
    init_robot_state = DefaultRobotState(robot_config)

    # tracking the number of times the ball bounced
    n_bounced = 0

    # we add a fixed goal
    # starting at x=3 and finishing at x=6
    goal = (2,4)
    goal_color = (0,0.7,0)
    goal_activated_color = (0,1,0)

    n_episodes = 0

    # running 5 episodes
    for episode in range(5):

        episode_end = False

        # resetting the robot and shooting
        # the ball gun
        world_state = world.reset(init_robot_state,
                                  ball_gun)

        # keeping track of the number of times the
        # ball bounced
        n_bounced = 0
```

(continues on next page)

(continued from previous page)

```
    while not episode_end:


        # running controller
        angles = [joint.angle for joint
                  in world_state.robot.joints]
        angular_velocities = [joint.angular_velocity for joint
                              in world_state.robot.joints]
        torques = controller.get(references,
                                 angles,
                                 angular_velocities)

        #
        # One simulation step
        #

        # returns a snapshot of all the data computed
        # and updated by the physics engine at this
        # iteration (see below for all information managed)
        # relative=True : torques are not given in absolute value,
        # but as values in [-1,1] that will be mapped to
        # [-max_torque,+max_torque]
        world_state = world.step(torques,relative_torques=True)


        # keeping track number of times the ball bounced
        if world_state.ball_hits_floor :
            n_bounced += 1
        if n_bounced >= 2 :
            # if bounced more than 2 : end of episode
            episode_end = True


        #
        # Rendering
        #

        if rendering:

            # was the goal hit ?

            color = goal_color
            if world_state.ball_hits_floor :
                p = world_state.ball_hits_floor
                if p>goal[0] and p<goal[1]:
                    # yes, using activated color
                    color = goal_activated_color

            # the renderer can take in an array of goals
            # to display

            goals = [(goal[0],goal[1],color)]

            # render based on the information provided by
            # the physics engined
            renderer.render(world_state,goals,time_step=1.0/60.0)
```

(continues on next page)

(continued from previous page)

CHAPTER 3

Indices and tables

- genindex
- modindex
- search