Code in PDF

Unit Tests

```
import unittest
import creature
import pybullet as p

class TestCreature(unittest.TestCase):
    def testCreatExists(self):
        self.assertIsNotNone(creature.Creature)

    def testCreatureGetFlatLinks(self):
        c = creature.Creature(gene_count=4)
        links = c.get_flat_links()
        self.assertEqual(len(links), 4)

    def testExpLinks(self):
        c = creature.Creature(gene_count=25)
        links = c.get_flat_links()
        exp_links = c.get_expanded_links()
        self.assertGreaterEqual(len(exp_links), len(links))

    def testToXMLNotNone(self):
        c = creature.Creature(gene_count=2)
        xml_str = c.to_xml()
        self.assertIsNotNone(xml_str)

    def testLoadXML(self):
        c = creature.Creature(gene_count=20)
        xml_str = c.to_xml()
        with open('test.urdf', 'w') as f:
            f.write(xml_str)
        p.connect(p.DIRECT)
        cid = p.loadURDF('test.urdf')
        self.assertIsNotNone(cid)


    def testMotor(self):
        m = creature.Motor(0.1, 0.5, 0.5)
        self.assertIsNotNone(m)

    def testMotorVal(self):
        m = creature.Motor(0.1, 0.5, 0.5)
```

```
        self.assertEqual(m.get_output(), 1)

    def testMotorVal2(self):
        m = creature.Motor(0.6, 0.5, 0.5)
        m.get_output()
        m.get_output()
        self.assertGreater(m.get_output(), 0)

    def testDist(self):
        c = creature.Creature(3)
        c.update_position((0, 0, 0))
        d1 = c.get_distance_travelled()
        c.update_position((1, 1, 1))
        d2 = c.get_distance_travelled()
        self.assertGreater(d2, d1)


unittest.main()
```

```
# If you on a Windows machine with any Python version
# or an M1 mac with any Python version
# or an Intel Mac with Python > 3.7
# the multi-threaded version does not work
# so instead, you can use this version.

import unittest
import population
import simulation
import genome
import creature
import numpy as np

class TestGA(unittest.TestCase):
    def testBasicGA(self):
        pop = population.Population(pop_size=10,
                                    gene_count=3)
        #sim = simulation.ThreadedSim(pool_size=1)
        sim = simulation.Simulation()

        for iteration in range(1000):
            # this is a non-threaded version
            # where we just call run_creature instead
            # of eval_population
            for cr in pop.creatures:
                sim.run_creature(cr, 2400)
```

```
            #sim.eval_population(pop, 2400)
            fits = [cr.get_distance_travelled()
                    for cr in pop.creatures]
            links = [len(cr.get_expanded_links())
                    for cr in pop.creatures]
            print(iteration, "fittest:", np.round(np.max(fits), 3),
                  "mean:", np.round(np.mean(fits), 3), "mean links",
np.round(np.mean(links)), "max links", np.round(np.max(links)))
            fit_map = population.Population.get_fitness_map(fits)
            new_creatures = []
            for i in range(len(pop.creatures)):
                p1_ind = population.Population.select_parent(fit_map)
                p2_ind = population.Population.select_parent(fit_map)
                p1 = pop.creatures[p1_ind]
                p2 = pop.creatures[p2_ind]
                # now we have the parents!
                dna = genome.Genome.crossover(p1.dna, p2.dna)
                dna = genome.Genome.point_mutate(dna, rate=0.1, amount=0.25)
                dna = genome.Genome.shrink_mutate(dna, rate=0.25)
                dna = genome.Genome.grow_mutate(dna, rate=0.1)
                cr = creature.Creature(1)
                cr.update_dna(dna)
                new_creatures.append(cr)
            # elitism
            max_fit = np.max(fits)
            for cr in pop.creatures:
                if cr.get_distance_travelled() == max_fit:
                    new_cr = creature.Creature(1)
                    new_cr.update_dna(cr.dna)
                    new_creatures[0] = new_cr
                    filename = "elite_"+str(iteration)+".csv"
                    genome.Genome.to_csv(cr.dna, filename)
                    break

            pop.creatures = new_creatures

        self.assertNotEqual(fits[0], 0)

unittest.main()
```

```
# If you on a Windows machine with any Python version
# or an M1 mac with any Python version
# or an Intel Mac with Python > 3.7
# this multi-threaded version does not work
```

```
# please use test_ga_single_thread.py on those setups

import unittest
import population
import simulation
import genome
import creature
import numpy as np

class TestGA(unittest.TestCase):
   def testBasicGA(self):
       pop = population.Population(pop_size=10,
                                   gene_count=3)
       sim = simulation.ThreadedSim(pool_size=1)
       #sim = simulation.Simulation()

       for iteration in range(1000):
           sim.eval_population(pop, 2400)
           fits = [cr.get_distance_travelled()
                   for cr in pop.creatures]
           links = [len(cr.get_expanded_links())
                   for cr in pop.creatures]
           print(iteration, "fittest:", np.round(np.max(fits), 3),
                 "mean:", np.round(np.mean(fits), 3), "mean links",
np.round(np.mean(links)), "max links", np.round(np.max(links)))
           fit_map = population.Population.get_fitness_map(fits)
           new_creatures = []
           for i in range(len(pop.creatures)):
               p1_ind = population.Population.select_parent(fit_map)
               p2_ind = population.Population.select_parent(fit_map)
               p1 = pop.creatures[p1_ind]
               p2 = pop.creatures[p2_ind]
               # now we have the parents!
               dna = genome.Genome.crossover(p1.dna, p2.dna)
               dna = genome.Genome.point_mutate(dna, rate=0.1, amount=0.25)
               dna = genome.Genome.shrink_mutate(dna, rate=0.25)
               dna = genome.Genome.grow_mutate(dna, rate=0.1)
               cr = creature.Creature(1)
               cr.update_dna(dna)
               new_creatures.append(cr)
           # elitism
           max_fit = np.max(fits)
           for cr in pop.creatures:
               if cr.get_distance_travelled() == max_fit:
                   new_cr = creature.Creature(1)
```

```
                    new_cr.update_dna(cr.dna)
                    new_creatures[0] = new_cr
                    filename = "elite_"+str(iteration)+".csv"
                    genome.Genome.to_csv(cr.dna, filename)
                    break

            pop.creatures = new_creatures

        self.assertNotEqual(fits[0], 0)

unittest.main()
```

```
import unittest
import genome
import numpy as np
import os

class GenomeTest (unittest.TestCase):
    def testClassExists(self):
        self.assertIsNotNone(genome.Genome)


    def testClassExists(self):
        self.assertIsNotNone(genome.Genome)

    def testRandomGene(self):
        self.assertIsNotNone(genome.Genome.get_random_gene)

    def testRandomGeneNotNone(self):
        self.assertIsNotNone(genome.Genome.get_random_gene(5))

    def testRandomGeneHasValues(self):
        gene = genome.Genome.get_random_gene(5)
        self.assertIsNotNone(gene[0])

    def testRandomGeneLength(self):
        gene = genome.Genome.get_random_gene(20)
        self.assertEqual(len(gene), 20)

    def testRandGeneIsNumpyArrays(self):
        gene = genome.Genome.get_random_gene(20)
        self.assertEqual(type(gene), np.ndarray)
```

```
def testRandomGenomeExists(self):
    data = genome.Genome.get_random_genome(20, 5)
    self.assertIsNotNone(data)

def testGeneSpecExist(self):
    spec = genome.Genome.get_gene_spec()
    self.assertIsNotNone(spec)

def testGeneSpecHasLinkLength(self):
    spec = genome.Genome.get_gene_spec()
    self.assertIsNotNone(spec['link-length'])

def testGeneSpecHasLinkLength(self):
    spec = genome.Genome.get_gene_spec()
    self.assertIsNotNone(spec['link-length']["ind"])

def testGeneSpecScale(self):
    spec = genome.Genome.get_gene_spec()
    gene = genome.Genome.get_random_gene(20)
    self.assertGreater(gene[spec["link-length"]["ind"]], 0)


def testGeneToGeneDict(self):
    spec = genome.Genome.get_gene_spec()
    gene = genome.Genome.get_random_gene(len(spec))
    gene_dict = genome.Genome.get_gene_dict(gene, spec)
    self.assertIn("link-recurrence", gene_dict)

def testGenomeToDict(self):
    spec = genome.Genome.get_gene_spec()
    dna = genome.Genome.get_random_genome(len(spec), 3)
    genome_dicts = genome.Genome.get_genome_dicts(dna, spec)
    self.assertEqual(len(genome_dicts), 3)

def testFlatLinks(self):
    links = [
      genome.URDFLink(name="A", parent_name=None, recur=1),
      genome.URDFLink(name="B", parent_name="A", recur=2),
      genome.URDFLink(name="C", parent_name="B", recur=2)
    ]
    self.assertIsNotNone(links)

def testExpandLinks(self):
    links = [
```

```
            genome.URDFLink(name="A", parent_name="None", recur=1),
            genome.URDFLink(name="B", parent_name="A", recur=1),
            genome.URDFLink(name="C", parent_name="B", recur=2),
            genome.URDFLink(name="D", parent_name="C", recur=1),
        ]
        exp_links = [links[0]]
        genome.Genome.expandLinks(links[0], links[0].name, links, exp_links)
        self.assertEqual(len(exp_links), 6)

    def testCrossover(self):
        g1 = [[1], [2], [3]]
        g2 = [[4], [5], [6]]
        for i in range(10):
            g3 = genome.Genome.crossover(g1, g2)
            self.assertGreater(len(g3), 0)

    def test_point(self):
        g1 = np.array([[1.0], [2.0], [3.0]])
        g2 = genome.Genome.point_mutate(g1, rate=1, amount=0.25)
        self.assertFalse(np.array_equal(g1, g2))

    def test_point_range(self):
        g1 = np.array([[1.0], [0.0], [1.0], [0.0]])
        for i in range(100):
            g2 = genome.Genome.point_mutate(g1, rate=1, amount=0.25)
            self.assertLessEqual(np.max(g2), 1.0)
            self.assertGreaterEqual(np.min(g2), 0.0)

    def test_shrink(self):
        g1 = np.array([[1.0], [2.0]])
        g2 = genome.Genome.shrink_mutate(g1, rate=1.0)
        # should def. shrink as rate = 1
        self.assertEqual(len(g2), 1)

    def test_shrink2(self):
        g1 = np.array([[1.0], [2.0]])
        g2 = genome.Genome.shrink_mutate(g1, rate=0.0)
        # should not shrink as rate = 0
        self.assertEqual(len(g2), 2)

    def test_shrink3(self):
        g1 = np.array([[1.0]])
        g2 = genome.Genome.shrink_mutate(g1, rate=1.0)
        # should not shrink if already len 1
        self.assertEqual(len(g2), 1)
```

```
def test_grow1(self):
    g1 = np.array([[1.0], [2.0]])
    g2 = genome.Genome.grow_mutate(g1, rate=1)
    self.assertGreater(len(g2), len(g1))

def test_grow2(self):
    g1 = np.array([[1.0], [2.0]])
    g2 = genome.Genome.grow_mutate(g1, rate=0)
    self.assertEqual(len(g2), len(g1))

def test_tocsv(self):
    g1 = [[1,2,3]]
    genome.Genome.to_csv(g1, 'test.csv')
    self.assertTrue(os.path.exists('test.csv'))

def test_tocsv_content(self):
    g1 = [[1,2,3]]
    genome.Genome.to_csv(g1, 'test.csv')
    expect = "1,2,3,\n"
    with open('test.csv') as f:
        csv_str = f.read()
    self.assertEqual(csv_str, expect)

def test_tocsv_content2(self):
    g1 = [[1,2,3], [4,5,6]]
    genome.Genome.to_csv(g1, 'test.csv')
    expect = "1,2,3,\n4,5,6,\n"
    with open('test.csv') as f:
        csv_str = f.read()
    self.assertEqual(csv_str, expect)

def test_from_csv(self):
    g1 = [[1,2,3]]
    genome.Genome.to_csv(g1, 'test.csv')
    g2 = genome.Genome.from_csv('test.csv')
    print(g1, g2)
    self.assertTrue(np.array_equal(g1, g2))

def test_from_csv2(self):
    g1 = [[1,2,3], [4,5,6]]
    genome.Genome.to_csv(g1, 'test.csv')
    g2 = genome.Genome.from_csv('test.csv')
    print(g1, g2)
    self.assertTrue(np.array_equal(g1, g2))
```

```
unittest.main()
```

```
import unittest
import population
import numpy as np

class TestPop(unittest.TestCase):
   ## check for a parent id in the range 0-2
   def testSelPar(self):
       fits = [2.5, 1.2, 3.4]
       fitmap = population.Population.get_fitness_map(fits)
       pid = population.Population.select_parent(fitmap)
       self.assertLess(pid, 3)

   ## parent id should be 1 as the first fitness is zero
   ## second is 1000 and third is 0.1 , so second should
   ## almost always be selected
   def testSelPar2(self):
       fits = [0, 1000, 0.1]
       fitmap = population.Population.get_fitness_map(fits)
       pid = population.Population.select_parent(fitmap)
       self.assertEqual(pid, 1)

unittest.main()
```

```
import unittest
import population
import numpy as np

class TestPop(unittest.TestCase):
   ## check for a parent id in the range 0-2
   def testSelPar(self):
       fits = [2.5, 1.2, 3.4]
       fitmap = population.Population.get_fitness_map(fits)
       pid = population.Population.select_parent(fitmap)
       self.assertLess(pid, 3)

   ## parent id should be 1 as the first fitness is zero
   ## second is 1000 and third is 0.1 , so second should
   ## almost always be selected
   def testSelPar2(self):
       fits = [0, 1000, 0.1]
```

```
        fitmap = population.Population.get_fitness_map(fits)
        pid = population.Population.select_parent(fitmap)
        self.assertEqual(pid, 1)

unittest.main()
```

Simulations

```
import pybullet as p
from multiprocessing import Pool

class Simulation:
    def __init__(self, sim_id=0):
        self.physicsClientId = p.connect(p.DIRECT)
        self.sim_id = sim_id

    def run_creature(self, cr, iterations=2400):
        pid = self.physicsClientId
        p.resetSimulation(physicsClientId=pid)
        p.setPhysicsEngineParameter(enableFileCaching=0, physicsClientId=pid)

        p.setGravity(0, 0, -10, physicsClientId=pid)
        plane_shape = p.createCollisionShape(p.GEOM_PLANE, physicsClientId=pid)
        floor = p.createMultiBody(plane_shape, plane_shape, physicsClientId=pid)

        xml_file = 'temp' + str(self.sim_id) + '.urdf'
        xml_str = cr.to_xml()
        with open(xml_file, 'w') as f:
            f.write(xml_str)

        cid = p.loadURDF(xml_file, physicsClientId=pid)

        p.resetBasePositionAndOrientation(cid, [0, 0, 2.5], [0, 0, 0, 1],
physicsClientId=pid)


        for step in range(iterations):
            p.stepSimulation(physicsClientId=pid)
            if step % 24 == 0:
                self.update_motors(cid=cid, cr=cr)

            pos, orn = p.getBasePositionAndOrientation(cid, physicsClientId=pid)
            cr.update_position(pos)
            #print(pos[2])
```

```
            #print(cr.get_distance_travelled())


    def update_motors(self, cid, cr):
        """
        cid is the id in the physics engine
        cr is a creature object
        """
        for jid in range(p.getNumJoints(cid,
                                        physicsClientId=self.physicsClientId)):
            m = cr.get_motors()[jid]

            p.setJointMotorControl2(cid, jid,
                    controlMode=p.VELOCITY_CONTROL,
                    targetVelocity=m.get_output(),
                    force = 5,
                    physicsClientId=self.physicsClientId)


    # You can add this to the Simulation class:
    def eval_population(self, pop, iterations):
        for cr in pop.creatures:
            self.run_creature(cr, 2400)


class ThreadedSim():
    def __init__(self, pool_size):
        self.sims = [Simulation(i) for i in range(pool_size)]

    @staticmethod
    def static_run_creature(sim, cr, iterations):
        sim.run_creature(cr, iterations)
        return cr

    def eval_population(self, pop, iterations):
        """
        pop is a Population object
        iterations is frames in pybullet to run for at 240fps
        """
        pool_args = []
        start_ind = 0
        pool_size = len(self.sims)
        while start_ind < len(pop.creatures):
            this_pool_args = []
            for i in range(start_ind, start_ind + pool_size):
```

```
                if i == len(pop.creatures):# the end
                    break
                # work out the sim ind
                sim_ind = i % len(self.sims)
                this_pool_args.append([
                            self.sims[sim_ind],
                            pop.creatures[i],
                            iterations]
                )
            pool_args.append(this_pool_args)
            start_ind = start_ind + pool_size

        new_creatures = []
        for pool_argset in pool_args:
            with Pool(pool_size) as p:
                # it works on a copy of the creatures, so receive them
                creatures = p.starmap(ThreadedSim.static_run_creature, pool_argset)
                # and now put those creatures back into the main
                # self.creatures array
                new_creatures.extend(creatures)
        pop.creatures = new_creatures
```

Genomes

```
import numpy as np
import copy
import random

class Genome():
    @staticmethod
    def get_random_gene(length):
        gene = np.array([np.random.random() for i in range(length)])
        return gene

    @staticmethod
    def get_random_genome(gene_length, gene_count):
        genome = [Genome.get_random_gene(gene_length) for i in range(gene_count)]
        return genome

    @staticmethod
    def get_gene_spec():
        gene_spec =  {"link-shape":{"scale":1},
            "link-length": {"scale":2},
            "link-radius": {"scale":1},
```

```
            "link-recurrence": {"scale":3},
            "link-mass": {"scale":1},
            "joint-type": {"scale":1},
            "joint-parent":{"scale":1},
            "joint-axis-xyz": {"scale":1},
            "joint-origin-rpy-1":{"scale":np.pi * 2},
            "joint-origin-rpy-2":{"scale":np.pi * 2},
            "joint-origin-rpy-3":{"scale":np.pi * 2},
            "joint-origin-xyz-1":{"scale":1},
            "joint-origin-xyz-2":{"scale":1},
            "joint-origin-xyz-3":{"scale":1},
            "control-waveform":{"scale":1},
            "control-amp":{"scale":0.25},
            "control-freq":{"scale":1}
            }
        ind = 0
        for key in gene_spec.keys():
            gene_spec[key]["ind"] = ind
            ind = ind + 1
        return gene_spec

    @staticmethod
    def get_gene_dict(gene, spec):
        gdict = {}
        for key in spec:
            ind = spec[key]["ind"]
            scale = spec[key]["scale"]
            gdict[key] = gene[ind] * scale
        return gdict

    @staticmethod
    def get_genome_dicts(genome, spec):
        gdicts = []
        for gene in genome:
            gdicts.append(Genome.get_gene_dict(gene, spec))
        return gdicts

    @staticmethod
    def expandLinks(parent_link, uniq_parent_name, flat_links, exp_links):
        children = [l for l in flat_links if l.parent_name == parent_link.name]
        sibling_ind = 1
        for c in children:
            for r in range(int(c.recur)):
                sibling_ind  = sibling_ind +1
                c_copy = copy.copy(c)
```

```
                c_copy.parent_name = uniq_parent_name
                uniq_name = c_copy.name + str(len(exp_links))
                #print("exp: ", c.name, " -> ", uniq_name)
                c_copy.name = uniq_name
                c_copy.sibling_ind = sibling_ind
                exp_links.append(c_copy)
                assert c.parent_name != c.name, "Genome::expandLinks: link joined to
itself: " + c.name + " joins " + c.parent_name
                Genome.expandLinks(c, uniq_name, flat_links, exp_links)

    @staticmethod
    def genome_to_links(gdicts):
        links = []
        link_ind = 0
        parent_names = [str(link_ind)]
        for gdict in gdicts:
            link_name = str(link_ind)
            parent_ind = gdict["joint-parent"] * len(parent_names)
            assert parent_ind < len(parent_names), "genome.py: parent ind too high:
" + str(parent_ind) + "got: " + str(parent_names)
            parent_name = parent_names[int(parent_ind)]
            #print("available parents: ", parent_names, "chose", parent_name)
            recur = gdict["link-recurrence"]
            link = URDFLink(name=link_name,
                            parent_name=parent_name,
                            recur=recur+1,
                            link_length=gdict["link-length"],
                            link_radius=gdict["link-radius"],
                            link_mass=gdict["link-mass"],
                            joint_type=gdict["joint-type"],
                            joint_parent=gdict["joint-parent"],
                            joint_axis_xyz=gdict["joint-axis-xyz"],
                            joint_origin_rpy_1=gdict["joint-origin-rpy-1"],
                            joint_origin_rpy_2=gdict["joint-origin-rpy-2"],
                            joint_origin_rpy_3=gdict["joint-origin-rpy-3"],
                            joint_origin_xyz_1=gdict["joint-origin-xyz-1"],
                            joint_origin_xyz_2=gdict["joint-origin-xyz-2"],
                            joint_origin_xyz_3=gdict["joint-origin-xyz-3"],
                            control_waveform=gdict["control-waveform"],
                            control_amp=gdict["control-amp"],
                            control_freq=gdict["control-freq"])
            links.append(link)
            if link_ind != 0:# don't re-add the first link
                parent_names.append(link_name)
            link_ind = link_ind + 1
```

```
        # now just fix the first link so it links to nothing
        links[0].parent_name = "None"
        return links

    @staticmethod
    def crossover(g1, g2):
        x1 = random.randint(0, len(g1)-1)
        x2 = random.randint(0, len(g2)-1)
        g3 = np.concatenate((g1[x1:], g2[x2:]))
        if len(g3) > len(g1):
            g3 = g3[0:len(g1)]
        return g3

    @staticmethod
    def point_mutate(genome, rate, amount):
        new_genome = copy.copy(genome)
        for gene in new_genome:
            for i in range(len(gene)):
                if random.random() < rate:
                    gene[i] += 0.1
                if gene[i] >= 1.0:
                    gene[i] = 0.9999
                if gene[i] < 0.0:
                    gene[i] = 0.0
        return new_genome

    @staticmethod
    def shrink_mutate(genome, rate):
        if len(genome) == 1:
            return copy.copy(genome)
        if random.random() < rate:
            ind = random.randint(0, len(genome)-1)
            new_genome = np.delete(genome, ind, 0)
            return new_genome
        else:
            return copy.copy(genome)

    @staticmethod
    def grow_mutate(genome, rate):
        if random.random() < rate:
            gene = Genome.get_random_gene(len(genome[0]))
            new_genome = copy.copy(genome)
            new_genome = np.append(new_genome, [gene], axis=0)
            return new_genome
```

```
        else:
            return copy.copy(genome)


    @staticmethod
    def to_csv(dna, csv_file):
        csv_str = ""
        for gene in dna:
            for val in gene:
                csv_str = csv_str + str(val) + ","
            csv_str = csv_str + '\n'

        with open(csv_file, 'w') as f:
            f.write(csv_str)

    @staticmethod
    def from_csv(filename):
        csv_str = ''
        with open(filename) as f:
            csv_str = f.read()
        dna = []
        lines = csv_str.split('\n')
        for line in lines:
            vals = line.split(',')
            gene = [float(v) for v in vals if v != '']
            if len(gene) > 0:
                dna.append(gene)
        return dna

class URDFLink:
    def __init__(self, name, parent_name, recur,
                 link_length=0.1,
                 link_radius=0.1,
                 link_mass=0.1,
                 joint_type=0.1,
                 joint_parent=0.1,
                 joint_axis_xyz=0.1,
                 joint_origin_rpy_1=0.1,
                 joint_origin_rpy_2=0.1,
                 joint_origin_rpy_3=0.1,
                 joint_origin_xyz_1=0.1,
                 joint_origin_xyz_2=0.1,
                 joint_origin_xyz_3=0.1,
                 control_waveform=0.1,
                 control_amp=0.1,
```

```
                control_freq=0.1):
        self.name = name
        self.parent_name = parent_name
        self.recur = recur
        self.link_length=link_length
        self.link_radius=link_radius
        self.link_mass=link_mass
        self.joint_type=joint_type
        self.joint_parent=joint_parent
        self.joint_axis_xyz=joint_axis_xyz
        self.joint_origin_rpy_1=joint_origin_rpy_1
        self.joint_origin_rpy_2=joint_origin_rpy_2
        self.joint_origin_rpy_3=joint_origin_rpy_3
        self.joint_origin_xyz_1=joint_origin_xyz_1
        self.joint_origin_xyz_2=joint_origin_xyz_2
        self.joint_origin_xyz_3=joint_origin_xyz_3
        self.control_waveform=control_waveform
        self.control_amp=control_amp
        self.control_freq=control_freq
        self.sibling_ind = 1

    def to_link_element(self, adom):
        #         <link name="base_link">
        #     <visual>
        #       <geometry>
        #         <cylinder length="0.6" radius="0.25"/>
        #       </geometry>
        #     </visual>
        #     <collision>
        #       <geometry>
        #         <cylinder length="0.6" radius="0.25"/>
        #       </geometry>
        #     </collision>
        #     <inertial>
        #       <mass value="0.25"/>
        #       <inertia ixx="0.0003" iyy="0.0003" izz="0.0003" ixy="0" ixz="0"
iyz="0"/>
        #     </inertial>
        #   </link>
         link_tag = adom.createElement("link")
        link_tag.setAttribute("name", self.name)
        vis_tag = adom.createElement("visual")
        geom_tag = adom.createElement("geometry")
        cyl_tag = adom.createElement("cylinder")
        cyl_tag.setAttribute("length", str(self.link_length))
```

```
        cyl_tag.setAttribute("radius", str(self.link_radius))

        geom_tag.appendChild(cyl_tag)
        vis_tag.appendChild(geom_tag)



        coll_tag = adom.createElement("collision")
        c_geom_tag = adom.createElement("geometry")
        c_cyl_tag = adom.createElement("cylinder")
        c_cyl_tag.setAttribute("length", str(self.link_length))
        c_cyl_tag.setAttribute("radius", str(self.link_radius))

        c_geom_tag.appendChild(c_cyl_tag)
        coll_tag.appendChild(c_geom_tag)

        #     <inertial>
        #       <mass value="0.25"/>
        #       <inertia ixx="0.0003" iyy="0.0003" izz="0.0003" ixy="0" ixz="0"
iyz="0"/>
        #     </inertial>
        inertial_tag = adom.createElement("inertial")
        mass_tag = adom.createElement("mass")
        # pi r^2 * height
        mass = np.pi * (self.link_radius * self.link_radius) * self.link_length
        mass_tag.setAttribute("value", str(mass))
        inertia_tag = adom.createElement("inertia")
        # <inertia ixx="0.0003" iyy="0.0003" izz="0.0003" ixy="0" ixz="0" iyz="0"/>
        inertia_tag.setAttribute("ixx", "0.03")
        inertia_tag.setAttribute("iyy", "0.03")
        inertia_tag.setAttribute("izz", "0.03")
        inertia_tag.setAttribute("ixy", "0")
        inertia_tag.setAttribute("ixz", "0")
        inertia_tag.setAttribute("iyx", "0")
        inertial_tag.appendChild(mass_tag)
        inertial_tag.appendChild(inertia_tag)



        link_tag.appendChild(vis_tag)
        link_tag.appendChild(coll_tag)
        link_tag.appendChild(inertial_tag)

        return link_tag

    def to_joint_element(self, adom):
        #           <joint name="base_to_sub2" type="revolute">
```

```
        #     <parent link="base_link"/>
        #     <child link="sub_link2"/>
        #     <axis xyz="1 0 0"/>
        #     <limit effort="10" upper="0" lower="10" velocity="1"/>
        #     <origin rpy="0 0 0" xyz="0 0.5 0"/>
        #   </joint>
        joint_tag = adom.createElement("joint")
        joint_tag.setAttribute("name", self.name + "_to_" + self.parent_name)
        if self.joint_type >= 0.5:
            joint_tag.setAttribute("type", "revolute")
        else:
            joint_tag.setAttribute("type", "revolute")
        parent_tag = adom.createElement("parent")
        parent_tag.setAttribute("link", self.parent_name)
        child_tag = adom.createElement("child")
        child_tag.setAttribute("link", self.name)
        axis_tag = adom.createElement("axis")
        if self.joint_axis_xyz <= 0.33:
            axis_tag.setAttribute("xyz", "1 0 0")
        if self.joint_axis_xyz > 0.33 and self.joint_axis_xyz <= 0.66:
            axis_tag.setAttribute("xyz", "0 1 0")
        if self.joint_axis_xyz > 0.66:
            axis_tag.setAttribute("xyz", "0 0 1")

        limit_tag = adom.createElement("limit")
        # effort upper lower velocity
        limit_tag.setAttribute("effort", "1")
        limit_tag.setAttribute("upper", "-3.1415")
        limit_tag.setAttribute("lower", "3.1415")
        limit_tag.setAttribute("velocity", "1")
        # <origin rpy="0 0 0" xyz="0 0.5 0"/>
        orig_tag = adom.createElement("origin")

        rpy1 = self.joint_origin_rpy_1 * self.sibling_ind
        rpy = str(rpy1) + " " + str(self.joint_origin_rpy_2) + " " +
str(self.joint_origin_rpy_3)

        orig_tag.setAttribute("rpy", rpy)
        xyz = str(self.joint_origin_xyz_1) + " " + str(self.joint_origin_xyz_2) + "
" + str(self.joint_origin_xyz_3)
        orig_tag.setAttribute("xyz", xyz)

        joint_tag.appendChild(parent_tag)
        joint_tag.appendChild(child_tag)
        joint_tag.appendChild(axis_tag)
```

```
        joint_tag.appendChild(limit_tag)
        joint_tag.appendChild(orig_tag)
        return joint_tag
```