A Robot Simulation of the Tower of Hanoi Puzzle using OpenRAVE with Sensor Feedback

By

Tuo Shi

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

August, 2013

Nashville, Tennessee

Approved by:

Professor Kazuhiko Kawamura

Professor Richard Alan Peters II

ACKNOWLEDGMENTS

First of all, I would like to express the deepest thanks to Dr. Kawamura and Dr. Peters for the guidance and support they rendered both in my way to seek the knowledge and in research of this thesis.

In addition, I would like to thank Huan Tan, Erdem Erdemir and Jing Fan and all my friends in Vanderbilt University for the great help in my academic research and my life in Nashville.

I also would like to thank my girlfriend Yun for the encouragement she has given and the company going through the difficult times.

My special thanks to my parents for their endless love and patience. I warmly appreciate the full support and understanding to me of my family.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLE

LIST OF ABBREVIATIONS

OpenRAVE	Open Robotics Automation Virtual Environment
CRL	Cognitive Robotics Laboratory
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
API	Application Programming Interface
DOF	Degrees Of Freedom
ODE	Open Dynamic Engine
SOFA	Simulation Open Framework Architecture
COTS	Commercial-Off-The-Shelf
FPS	Frames per Second
RRT	Rapidly-exploring Random Trees

Chapter I

INTRODUCTION

In recent years, more and more efforts have been put on creating robots that operate in domestic environments. Such environments are seldom well structured, which makes the operation of a robot a real challenge. In the future, robots will need to accomplish more complex tasks in such an environment. A robotic simulation environment for testing, developing and deploying motion planning is extremely necessary for both research and real-world robotics applications. In this thesis, a newly released robotic real-time simulations software library called Open Robotics Automation Virtual Environment (OpenRAVE) will be used to simulate a robot playing the Tower of Hanoi using sensor feedback. The background of the Tower of Hanoi will be briefly discussed in this chapter. In Chapter II, the method, algorithm, the principle of sensors, and codes used in OpenRAVE of this thesis will be introduced in detail. In Chapter III, the manipulation planning algorithms of OpenRAVE will be briefly introduced. In Chapter IV, we will discuss the robot and the environment in terms of robot and environmental parameters. In Chapter V, the results of simulations will be shown and evaluated. The discussion and future works will be elaborated in Chapter VI.

I.1.Problem Description

The purpose of this thesis is to use the Tower of Hanoi puzzle as a test example to demonstrate and evaluate the performance of a robotic simulator with sensor feedback. We choose a commonly used simulation software library named the Open Robotics

Automation Virtual Environment which was founded by Rosen Diankov at the Quality of Life Technology Center in the Carnegie Mellon University Robotics Institute [1] as the robotic simulator. A 6-DOF robotic arm with a gripper and an indoor environment will be created in OpenRAVE. Real-time simulator then plays the puzzle with the force of gravity and friction. The robot will make decisions to orderly pick up and drop the disks onto small pegs to finish the puzzle.

I.2.A Tower of Hanoi Simulation with Sensor Feedback

The famous Tower of Hanoi puzzle, invented by the French mathematician Édouard Lucas in 1883 [2], consists of three pegs called A, B, and C, and a set of n, pierced disks of different diameters that can be stacked on the pegs. Usually, the disks are numbered from 1 to n in order of increasing diameter. The tower is formed by stacking the disks onto peg A in decreasing order from bottom to top. The task is to transport the tower to peg C by moving the disks one at a time from peg to peg, adhering to the following rules [3]:

Rule 1. Move only one disk at a time.

Rule 2. A larger disk may not be placed on top of a smaller disk.

Rule 3. All disks, except the one being moved, must be on a peg.

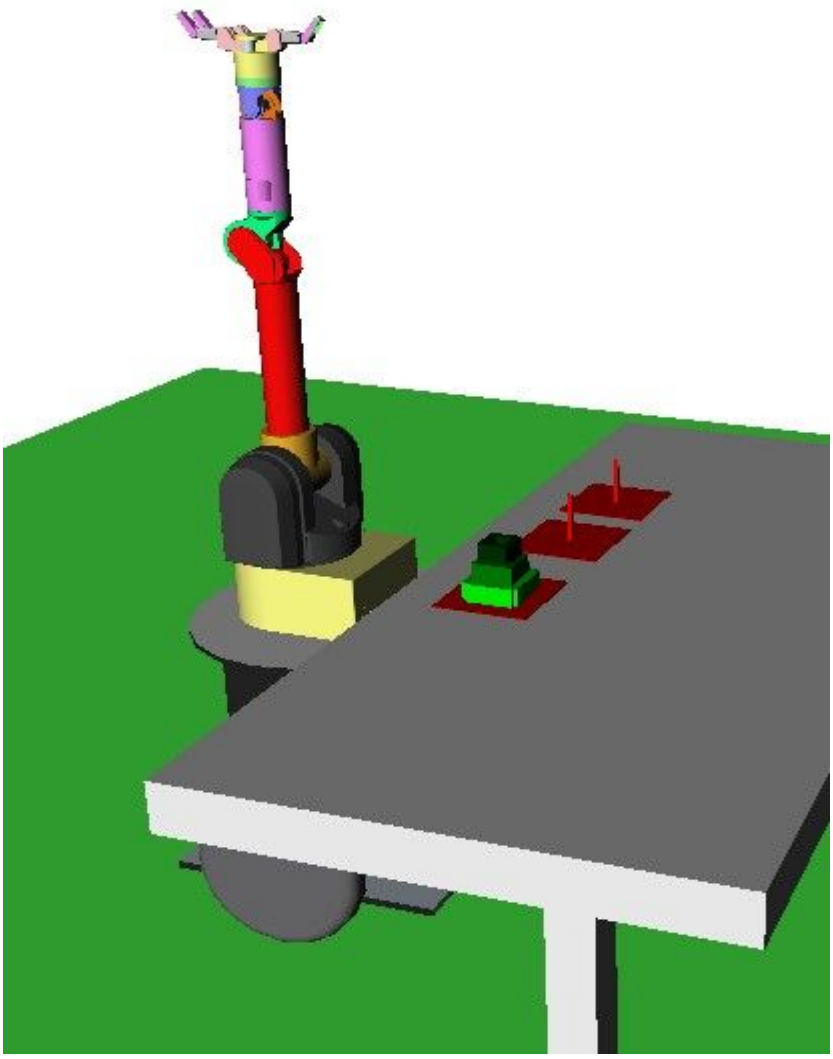


Figure 1: Tower of Hanoi puzzle built in a simulation environment

As shown in Figure 1, a Tower of Hanoi puzzle with three different sizes disks which is built in OpenRAVE will be simulated in this thesis. The three fingers of robotic arms are equipped with sensors that will detect whether there is a contact or not between each finger and the object. Once three fingers have contacted the disk, the fingers will stop closing and grab the disk firmly. The trajectory planner built in OpenRAVE calculates the motions of robotic arm without any collision. Then the robotic arm will take the disk to the desired location and waiting for a new commend

after all the fingers have been released.

The Tower of Hanoi puzzle can be played with any number of disks. The minimum number of moves that can solve the puzzle requires $2^n - 1$ steps, where n is the number of disks [4]. Several major types of solutions including iterative solution [5], recursive solution [6], binary solution [7] and gray code solution have been posted previously. Based on binary solution and gray code solution, a new method which plays the Tower of Hanoi in many different ways will be used in this thesis. The simulator will demonstrate all of these procedures and further tests the suitability and reliability itself.

Chapter II

OVERVIEW OF SIMULATION

In this chapter, the Tower of Hanoi methodology, approach, and algorithm used in the simulation will be introduced in detail.

II.1.Tower of Hanoi

In Chapter I, the Tower of Hanoi puzzle was introduced. In order to extend the diversity of playing the Tower of Hanoi puzzle and further test the simulator, different initial positions were tested in this thesis. Consider the following Tower of Hanoi puzzle with random disks positions, for example, shown in Figure 2.

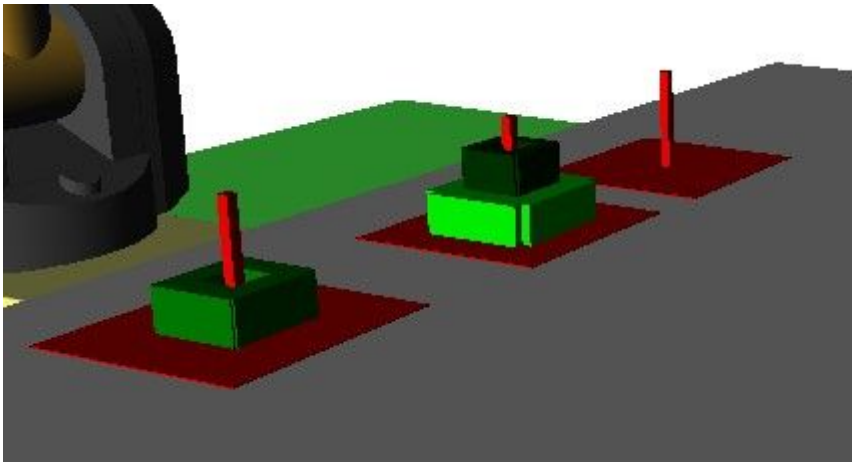


Figure 2: A Tower of Hanoi puzzle with random disk positions

The goal is still to form a complete tower onto the right peg. In this case, the solution would be different compared with the traditional one.

When human plays this puzzle, the most brute-force strategy is to try all possible moves. By searching all the possible situations of disks under the rules of puzzle, the task can eventually be accomplished. Robot can do the same despite the searching complexity. Intelligent robot, however, should use adaptability and flexibility in

decision-making [7]. The Cognitive Robotics Laboratory (CRL) in Vanderbilt University has contributed a great deal on designing decision-making software for intelligent robots, involving learning affordance [8], imitation learning [9] and so on. In this thesis, the robot in a simulation environment will mimic this procedure as a strategy to solve the Tower of Hanoi puzzle.

II.1.1.Searching Algorithm for Tower of Hanoi

The game can be represented by an undirected graph, the nodes representing the positions of disks and the edges representing the moves. A three disks game will be discussed, and the graph is as following:

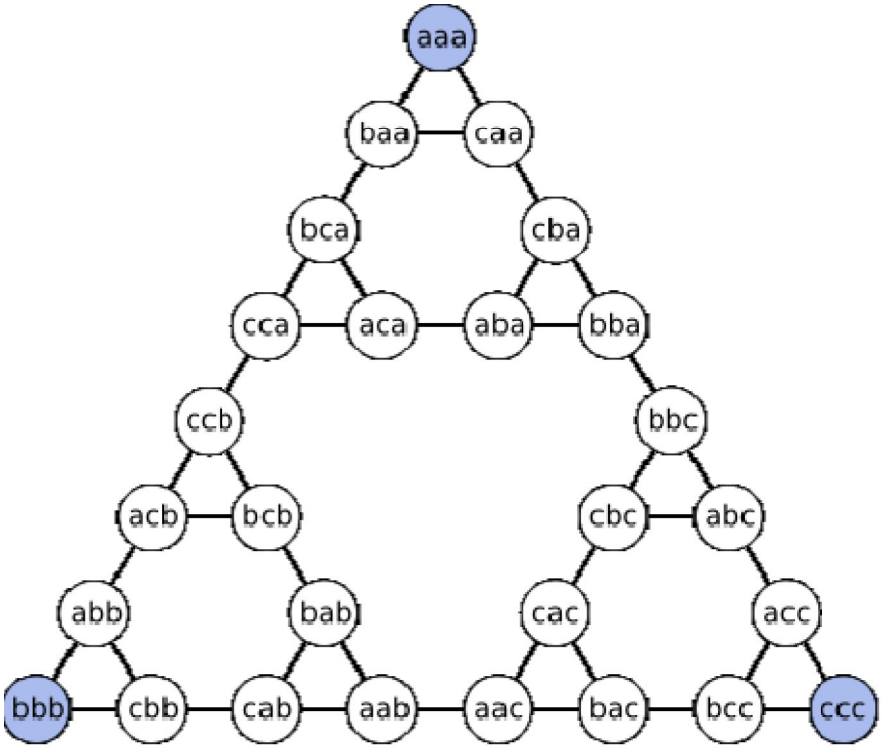


Figure 3: 27 possible locations of the puzzle

In Figure 3, pegs are called as *a*, *b* and *c*, which peg *a* is the starting peg and *c* is the destination. Each node represents the disk positions from left to right in order of increasing size. The purpose of searching algorithm is to reconstruct this graphical representation under the puzzle rules given an arbitrary node until the target node is

found (*ccc* for the regular Tower of Hanoi puzzle).

Based on three rules of the puzzle mentioned in previous chapter, three criteria can be developed, described as following:

Criterion 1 Only move the top disk on the peg.

Criterion 2 Larger disk cannot put on the top of a smaller one.

Criterion 3 Disk must be moved from one peg to another.

Figure 4 shows how three criteria have applied on the searching algorithm and built the graphical representation of Tower of Hanoi puzzle

```
StateTree[0].append(331)    # start node
level=0
unit=0
Search=1
SearchTarget=333    # target node

while(Search and level<20) :
    while ((Search and unit<len(StateTree[level])) and level<20) :
        TempNewState=[]
        TempNewStateRevised=[]
        NewState=0
        Dzeropos=int(StateTree[level][unit])%10
        Donepos=int((StateTree[level][unit]/10))%10
        Dtwopos=int(StateTree[level][unit]/100)
        Prev=float(unit)/10
        prev=float ("%.1f" % Prev)
        NewState=(Dtwopos+1)*100+Donepos*10+Dzeropos+prev
        TempNewState.append(NewState)
        NewState=(Dtwopos-1)*100+Donepos*10+Dzeropos+prev
        TempNewState.append(NewState)
        if Donepos+1!= Dtwopos and Donepos+1!= Dtwopos+3 and Donepos!=Dtwopos:
            NewState=Dtwopos*100+(Donepos+1)*10+Dzeropos+prev
            TempNewState.append(NewState)
        if Donepos-1!= Dtwopos and Donepos-1!= Dtwopos-3 and Donepos!=Dtwopos:
            NewState=Dtwopos*100+(Donepos-1)*10+Dzeropos+prev
            TempNewState.append(NewState)
```

```
            if Dzeropos+1!= Dtwopos and Dzeropos+1!= Dtwopos+3 and Dzeropos+1!= Donepos and Dzeropos+1!= Donepos+3 and
Donepos!=Dzeropos and Dzeropos!=Dtwopos:
                NewState=Dtwopos*100+Donepos*10+Dzeropos+1+prev
                TempNewState.append(NewState)
            if Dzeropos-1!= Dtwopos and Dzeropos-1!= Dtwopos-3 and Dzeropos-1!= Donepos and Dzeropos-1!= Donepos-3 and
Donepos!=Dzeropos and Dzeropos!=Dtwopos:
                NewState=Dtwopos*100+Donepos*10+Dzeropos-1+prev
                TempNewState.append(NewState)
            for NewState in TempNewState:
                if NewState/100 > 3:
                    NewState=NewState-300
                if (NewState/10)%10 > 3:
                    NewState=NewState-30
                if NewState%10 > 3:
                    NewState=NewState-3
                if NewState/100 < 1:
                    NewState=NewState+300
                if (NewState/10)%10 < 1:
                    NewState=NewState+30
                if NewState%10 < 1:
                    NewState=NewState+3
                TempNewStateRevised.append(NewState)
            Nextlevel=level+1
            for NewStateRevised in TempNewStateRevised:
                i=0
                j=0
                k=0
                for i in range(0,Nextlevel):
                    for j in range(0,len(StateTree[i])):
                        if   int (NewStateRevised) ==   int(StateTree[i][j]) :
                            k=k+1
                if k==0:
                    StateTree[Nextlevel].append(NewStateRevised)
                if   int(NewStateRevised) == SearchTarget :
                    Search=0
            unit=unit+1
        level=level+1
```

```
    unit=0


for x in range(0,level+1):
    print '\n'
      print 'Level',x,
      for y in range(0,len(StateTree[x])):
          z = StateTree[x][y]
          print "%.1f " % z,


for x in range(level,-1,-1):
      for y in range(0,len(StateTree[x])):
          if int(StateTree[x][y])==SearchTarget:
              Z=int(round(10*(StateTree[x][y]%1)))
              StateRev.append(int(StateTree[x][y]))
          if x!=level:
              if y==Z:
                  Z=int(round(10*(StateTree[x][y]%1)))
                  StateRev.append(int(StateTree[x][y]))
                  break


for x in range(level,-1,-1):
      State.append(StateRev[x])


print '\n\nState sequence',State
```

Figure 4: Searching algorithm code

Binary and Gray Code solutions have provided us an alternative way of solving the puzzle. Searching algorithm is based on a numeral system which is convenient an efficient for computer programming.

According to three criteria, from the given node as root, the searching algorithm is trying to look for every possible movement of next level. Each disk will be check in order. All the possible movements will be recorded into the list as well as their parent node. For example, the node *aaa* in Figure 3 represents as 111 in digital, since the rightmost digit represents the largest disk and the number indicates the location of

disk. (1 for start peg, 2 for middle peg, 3 for destination peg). There is one digit for each disk. The child nodes of 111 are 211.0 and 311.0. The 0 after decimal point means the order number starting with 0 from left to right of their parent node in previous level which helps to build the graphical representation constructionally and intuitionally. In each level, the algorithm will check every node and search for next possible movement until the target node is found. Then, we trace the target node all the way back to the root node; hence the solution of the puzzle is generated as state sequence shown in following figure:

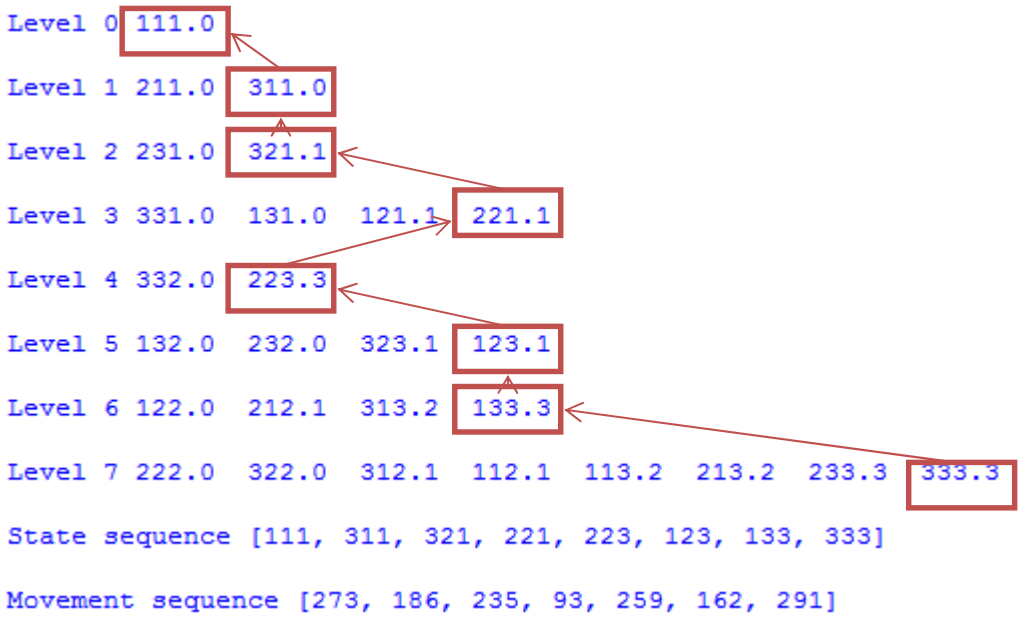


Figure 5: The graphical presentation generated by searching algorithm

In this case, the searching algorithm successfully constructs the graphical representation of Tower of Hanoi puzzle in Figure 3 and finds the solution correctly. Additionally, arbitrary disks starting positions such as position mentioned in Figure 2 can be also calculated accurately. Consequently, the experiments of robotic simulations can be conducted in various ways and the simulator will be tested more thoroughly and comprehensively

II.1.2.Movement Translation

We discussed the state sequence obtained by the searching algorithm. However, the robot still cannot play the puzzle since the disk location of picking up or dropping off is remaining unknown. A movement translation needs to be performed based on state sequence.

For a Tower of Hanoi puzzle with three disks, there are nine possible locations of disks shown in Figure 6.

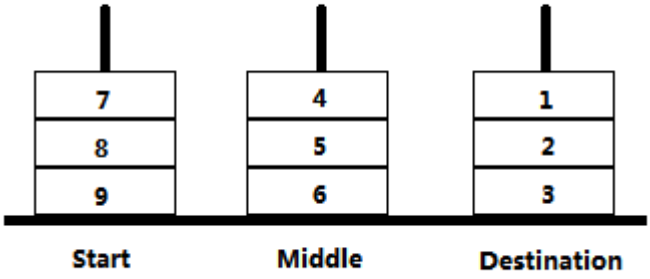


Figure 6: The possible disk locations of puzzle

By subtracting the nearby two elements in state sequence, the disk that is going to be moved and the numbers of disk on each peg can be calculated; hence the disk movement sequence is also determined. We represent these movements as a three digit number and generate a new movement sequence as shown in Figure 5. For example, the first element 273 means move the disk #2 (disk numbers increasing in order of decreasing size, disk #2 is the smallest disk) from location 7 to location 3 as shown in Figure 6. The leftmost digit represents the disk number that will be moved, the location number of this disk is at the middle and the destination location number is at the rightmost digit.

```
for x in range(0,len(State)-1):
    ANum=0
    BNum=0
    CNum=0
    Dzeropos=State[x]%10
    Donepos=(State[x]/10)%10
    Dtwopos=State[x]/100
    if Dzeropos==1:
        ANum=ANum+1;
    elif Dzeropos==2:
        BNum=BNum+1;
    else:
        CNum=CNum+1;
    if Donepos==1:
        ANum=ANum+1;
    elif Donepos==2:
        BNum=BNum+1;
    else:
        CNum=CNum+1;
    if Dtwopos==1:
        ANum=ANum+1;
    elif Dtwopos==2:
        BNum=BNum+1;
    else:
        CNum=CNum+1;
    Target=State[x+1]-State[x]
    if abs(Target)==100 or abs(Target)==200:
        disc=2
        src=Dtwopos-1
        dst=src+Target/100
        if dst<0:
            dst=dst+3
     elif abs(Target)==10 or abs(Target)==20:
        disc=1
        src=Donepos-1
        dst=src+Target/10
        if dst<0:
            dst=dst+3
```

```
        elif abs(Target)==1 or abs(Target)==2:
            disc=0
            src=Dzeropos-1
            dst=src+Target
            if dst<0:
                dst=dst+3
        if disc == 1:
            D = 100
        elif disc == 2:
            D = 200
        else :
            D = 0
        if src == 0:
            F = 100-ANum*10
        elif src == 1:
            F = 70-BNum*10
        else :
            F = 40-CNum*10
        if dst == 0:
            T = 9-ANum
        elif dst == 1:
            T = 6-BNum
        else :
            T = 3-CNum
        seq.append(D+F+T)
    print '\nMovement sequence',seq,'\n'
```

Figure 7: Movement translation code

Figure 7 shows the algorithm to translate the disks movement from state sequence and generating a new sequence named movement sequence. By obtaining the correct movement sequence, robot is able to finish the Tower of Hanoi puzzle successfully in the simulation environment.

II.2.Approach for a Robotic Simulator

Since a robust robotic simulator with sensor feedback is needed to be used and tested, we chose OpenRAVE as our simulation environment due to its strong functionality, visualized user interface and user friendliness.

II.2.1.OpenRAVE

OpenRAVE was founded by Rosen Diankov at the Quality of Life Technology Center in the Carnegie Mellon University Robotics Institute. The project was started in 2006 and started as a complete rewrite of RAVE to support plugins [1][11]. The main focus of OpenRAVE is to increase the reliability of motion planning systems to make integration easy. With the various functions and innovated algorithm built in, such as grasping, solving 6D transformations, motion planning and so on forth, OpenRAVE provides powerful features and extreme friendly user interface and is soon becoming popular as a robotics software environment. Moreover, it offers a tool called IKFast which is a Robot Kinematics Compiler. Unlike most inverse kinematics solvers, IKFast can analytically solve the kinematics equations of any complex kinematics chain with an extremely stable solution that can run as fast as 5 microseconds on recent processors [12]. The OpenRAVE grants a great simulator to let the user concentrate on the development of planning and scripting aspects of a problem without having to explicitly manage the details of robot kinematics, dynamics, collision detection, world updates and robot control.

II.2.2.Robot Manipulation with Sensor Feedback

With the movement sequence derived by the algorithm discussed previously, the robot is able to orderly pick up and drop off objects according to 3D coordinates with the help of collision-free motion planner, inverse kinematics transformation and

grasping functions in OpenRAVE. By reading the list of movement sequence one after another, the robot will execute the commands in order and display the message of current status. The simulator with sensor feedback simulates all the procedures in real-time. When the puzzle is accomplished, the robot will be reset and move back to the start position. The code for robot manipulation is shown in Figure 8.

```
seq_length = len(seq)
i=0


while (i<seq_length) :
    PosFrom = (seq[i]/10)%10
    PosTo = seq[i]%10
    DeskNum = seq[i]/100
    print "Step",i+1,": Disk",DeskNum,"     ",
    if PosFrom<=3:
        print 'C',
    elif PosFrom<=6:
        print 'B',
    else :
        print 'A',
    print "--->",
    if PosTo<=3 :
        print   'C',
    elif PosTo<=6 :
        print   'B',
    else :
        print   'A',
    if PosFrom == 1 :
        TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.148],[0,0,0,1]])
    elif PosFrom == 2 :
        TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.107],[0,0,0,1]])
    elif PosFrom == 3 :
        TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.066],[0,0,0,1]])
    elif PosFrom == 4 :
        TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.148],[0,0,0,1]])
    elif PosFrom == 5 :
        TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.107],[0,0,0,1]])
```

```
elif PosFrom == 6 :
    TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.066],[0,0,0,1]])
elif PosFrom == 7 :
    TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.148],[0,0,0,1]])
elif PosFrom == 8 :
    TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.107],[0,0,0,1]])
elif PosFrom == 9 :
    TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.066],[0,0,0,1]])
#TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.145],[0,0,0,1]])
resFrom = manipprob.MoveToHandPosition(matrices=[TgoalFrom],seedik=10)
robot.WaitForController(0)
with env:
    sol = manip.FindIKSolution(TgoalFrom, IkFilterOptions.CheckEnvCollisions)
    #with robot:
        #robot.SetDOFValues(sol,manip.GetArmIndices())
#manipprob.MoveManipulator(sol)
#robot.WaitForController(0)
env.SetPhysicsEngine(None)
taskprob.CloseFingers()
robot.WaitForController(0)
with env:
    if DeskNum == 0 :
        robot.Grab(env.GetKinBody('disk0'))
    elif DeskNum == 1 :
        robot.Grab(env.GetKinBody('disk1'))
    elif DeskNum == 2 :
        robot.Grab(env.GetKinBody('disk2'))
err=0.01
if PosTo == 1 :
    TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.148+err],[0,0,0,1]])
elif PosTo == 2 :
    TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.107+err],[0,0,0,1]])
elif PosTo == 3 :
    TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.066+err],[0,0,0,1]])
elif PosTo == 4 :
    TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.148+err],[0,0,0,1]])
elif PosTo == 5 :
    TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.107+err],[0,0,0,1]])
```

```
        elif PosTo == 6 :
            TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.066+err],[0,0,0,1]])
        elif PosTo == 7 :
            TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.148+err],[0,0,0,1]])
        elif PosTo == 8 :
            TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.107+err],[0,0,0,1]])
        elif PosTo == 9 :
            TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.066+err],[0,0,0,1]])
        env.SetPhysicsEngine(physics)
        '"with env:
            sol = manip.FindIKSolution(TgoalTo, IkFilterOptions.CheckEnvCollisions)
            with robot:
                time.sleep(3)"'
        #robot.SetDOFValues(sol,manip.GetArmIndices())
        #manipprob.MoveManipulator(sol)
        #robot.WaitForController(0)
        resTo = manipprob.MoveToHandPosition(matrices=[TgoalTo],seedik=10)
        robot.WaitForController(0)
        env.SetPhysicsEngine(None)
        if DeskNum == 0 :
            taskprob.ReleaseFingers(target=env.GetKinBody('disk0'))
        elif DeskNum == 1 :
            taskprob.ReleaseFingers(target=env.GetKinBody('disk1'))
        elif DeskNum == 2 :
            taskprob.ReleaseFingers(target=env.GetKinBody('disk2'))
        robot.WaitForController(0)
        env.SetPhysicsEngine(physics)
        print '    done!'
        i=i+1

    print '\nGame over !!!',
    manipprob.MoveManipulator(numpy.zeros(len(manip.GetArmIndices())),jitter=0.04)
    robot.WaitForController(0)
```

Figure 8: Robot manipulation code

As shown in Figure 9, the whole processes have been represented as a flowchart.

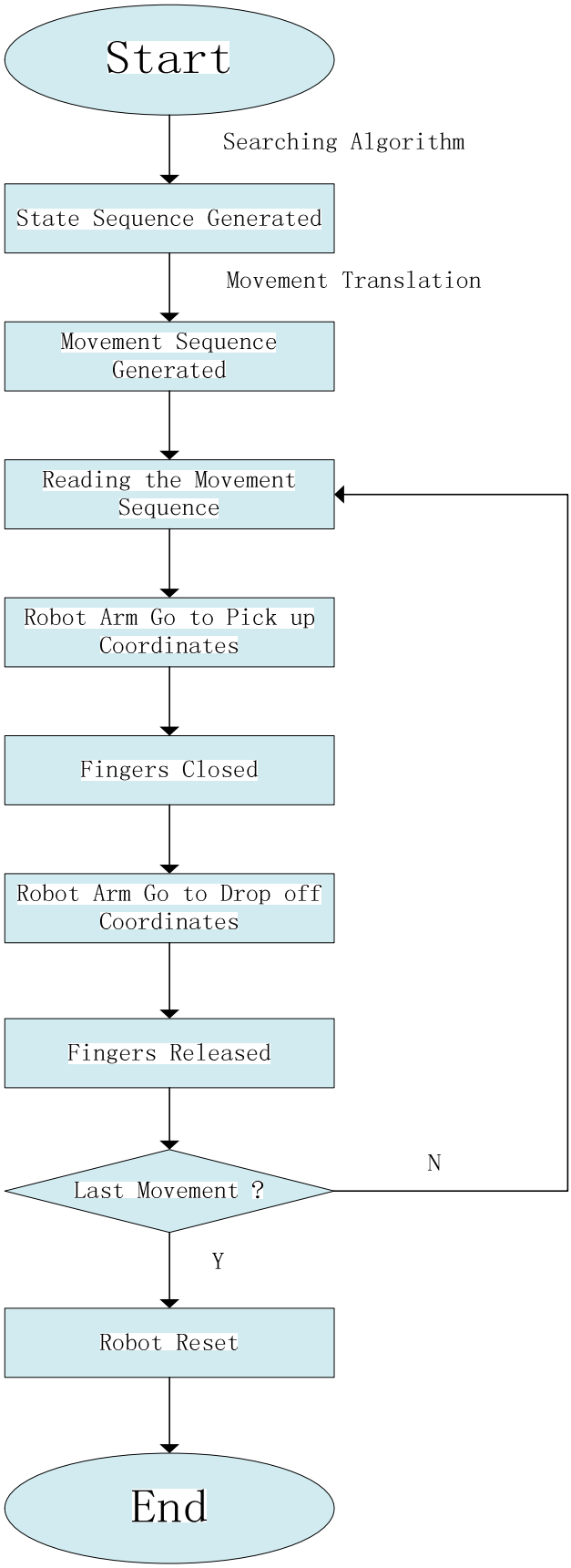


Figure 9: Flowchart of the Tower of Hanoi puzzle simulation

As the fingers of robotic arms are closing to grab the objects, the sensors will detect whether or not there is a contact between fingers and target object. Once all fingers have contacted on the object in the meanwhile both the object and the fingers are stabilized, the controller will execute next command to pick up the object toward its destination. When the robot drops off the object, it will take the object slightly higher than the target coordinates in case a collision which results in an error on the simulator. The object will fall due to the gravity.

During the grasppping, when all three fingers have contacted the object as shown in Figure 10, the object can be grasppped sucessfully.

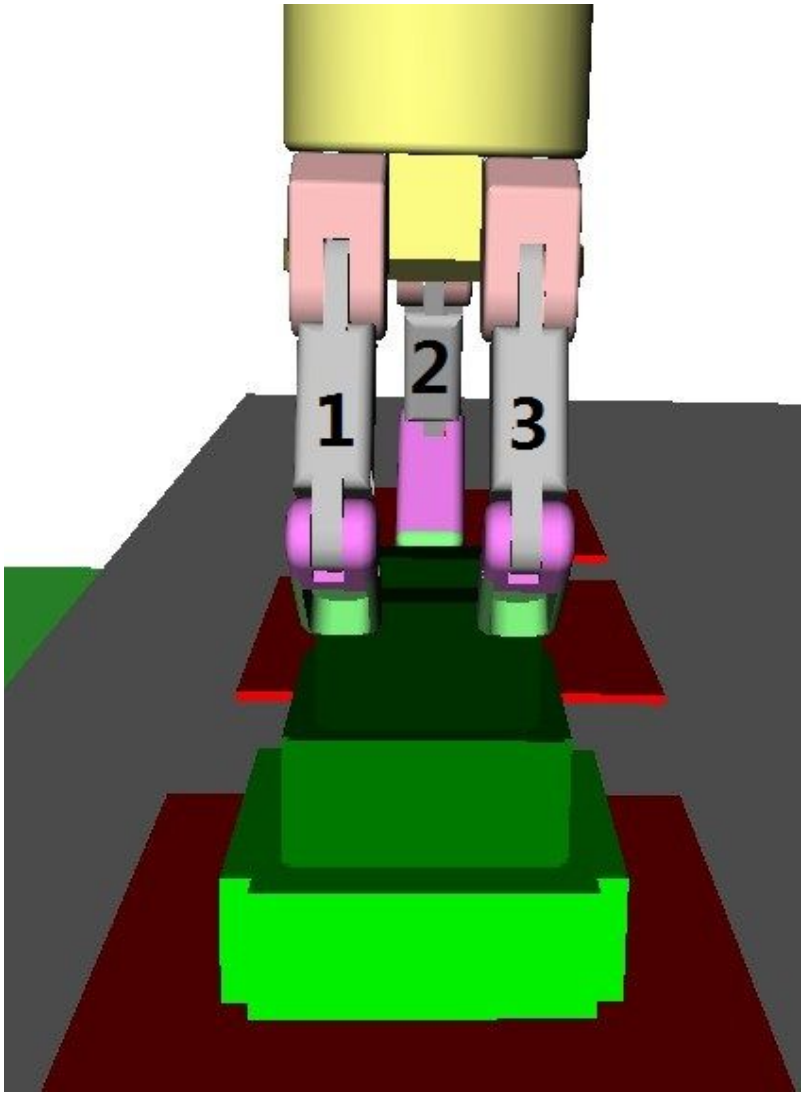


Figure 10: 3 fingers contacted the object

When only two fingers has contacted the object due to either the inaccuracy of 3D coordinates, as shown in Figure 11, or other errors, the robot can still pick up the object. In this case, finger #3 is missing the contact with the object. However, using the finger #1 and #2 robot can perform the grasping task successfully.

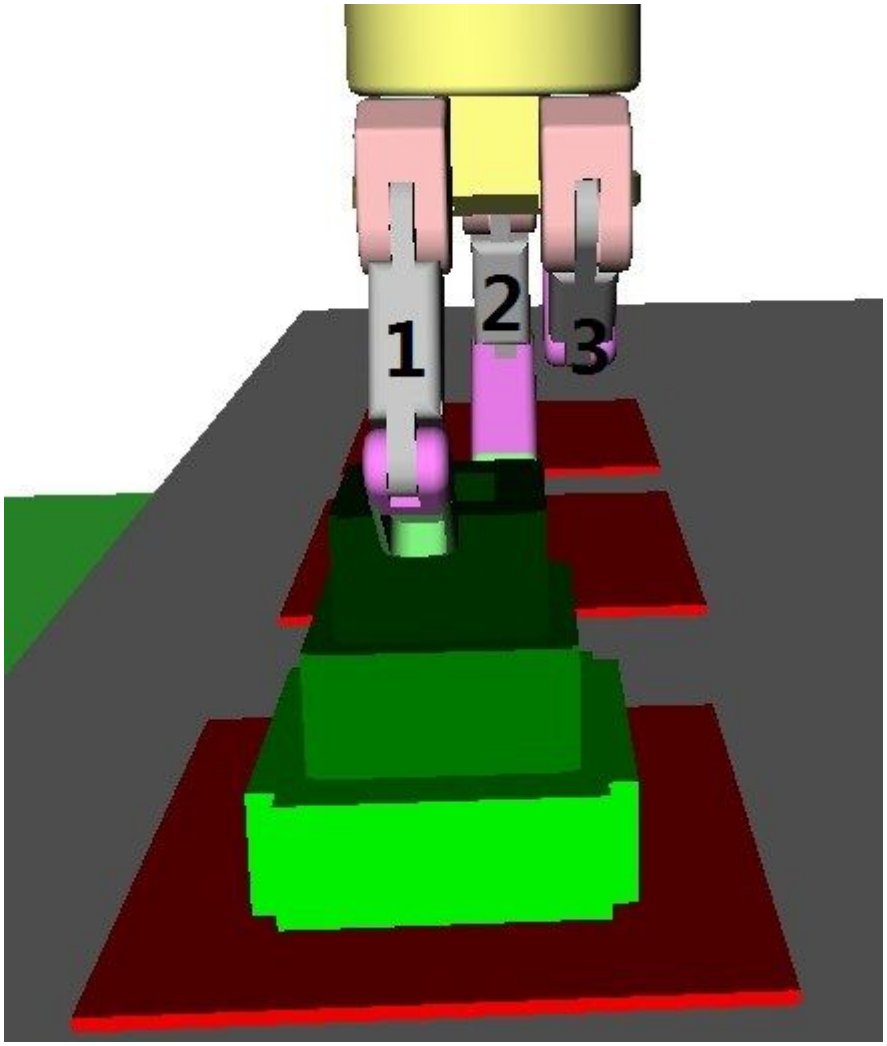


Figure 11: 2 fingers contacted the object

In the worst case, only one finger touches the object as shown in Figure 12. The object cannot be picked up by merely finger #1. After performing the finger closed command, the simulator will return an error and the simulation stops.

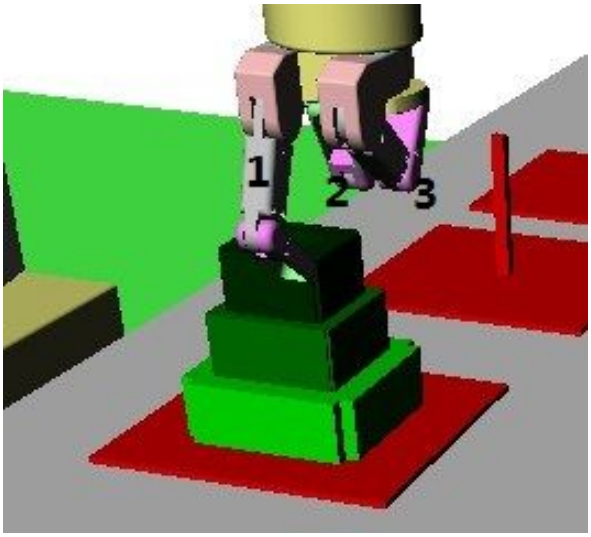


Figure 12: 1 finger contacted the object

II.2.3.Software Platform

In this thesis, the OpenRAVE version we used is 0.8.2 which was released on October 18, 2012. The robots database provides plenty of robot modules in this version with a high success testing rate. OpenRAVE can be installed on either Windows or Unix-based system. The computer for the test is running Windows 7 Ultimate 32bit. With Graphical User Interface (GUI) in OpenRAVE, as shown in Figure 13, a 3D environment for robotic simulation can be built and demonstrate the all the procedures intuitively.

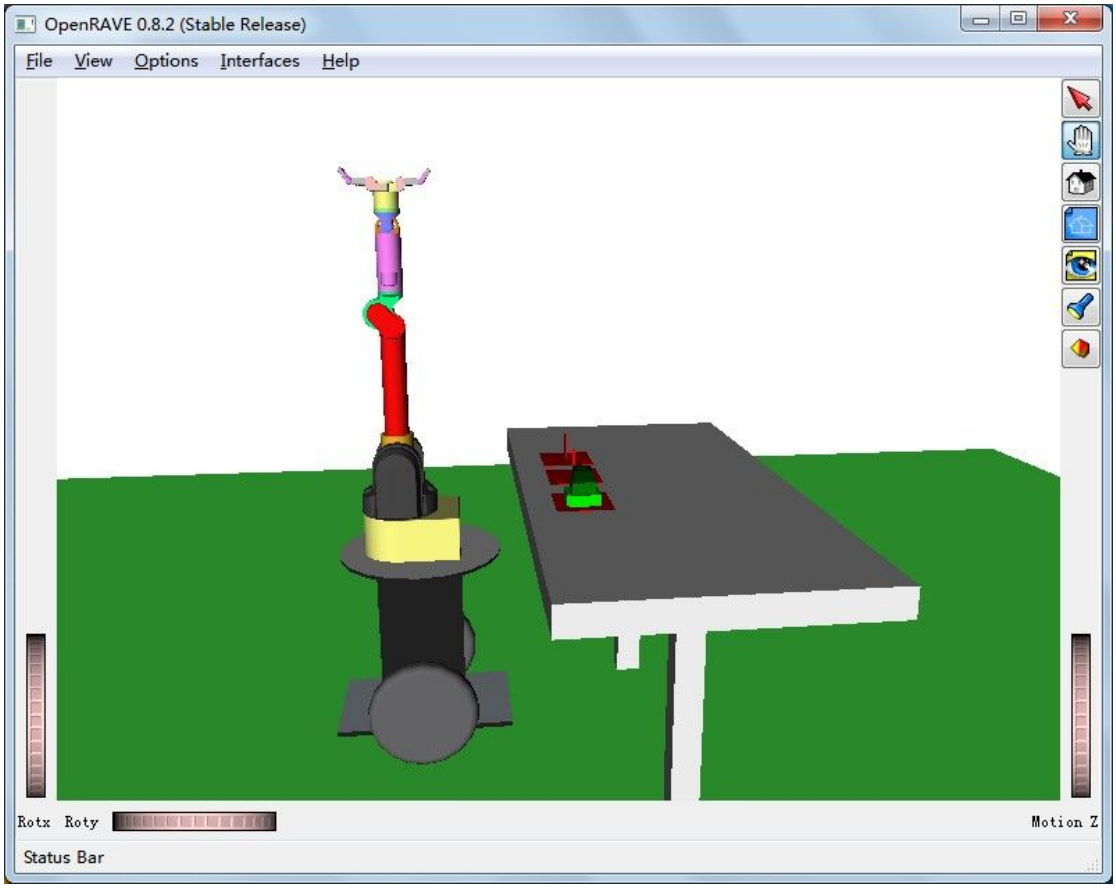


Figure 13: OpenRAVE GUI

All the objects in simulation environment, including robot, segway, disks and pegs, are written in Hyper Text Markup Language (HTML). The view point can be set and rotated freely along x, y and z axis with zoom in and out function. All the robot parameters and environment parameters such as the force of gravity, friction coefficient, density and etc are also programmed in HTML.

OpenRAVE also provides powerful Application Programming Interface (API) for many software platforms, including Python, Octave and Matlab. Currently, Python API offers more comprehensive functions than any other APIs; hence it will be selected for fulfill the purpose of this thesis. The GUI of Python is shown in Figure 14.

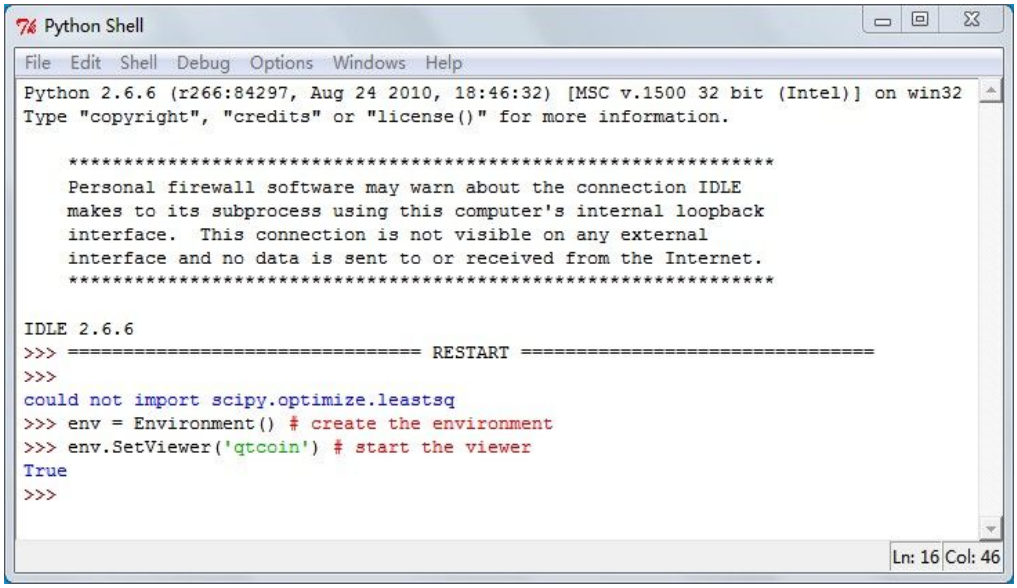


Figure 14: Python GUI

Python is a general-purpose, high-level programming language allowing programmers to express concepts in fewer lines of code than in other languages such as C [13] [14]. Python 2.6.6 has been installed in the testing computer. All the algorithms mentioned in this chapter and robot manipulation codes are programmed in Python.

Chapter III

Manipulation Planning of OpenRAVE

The manipulation planning algorithms of OpenRAVE are responsible for finding a path without collision in the robot configuration space between an initial configuration to a goal-satisfying configuration while maintaining constraints on the search space. The planner has divided into two components: a generator for feasible, goal-satisfying robot configurations, and a search algorithm that explores the feasible space and composes the robot path.

III.1.Robot Configurations Generator

When the simulator is planning for grasping an object, all grasping operations are pre-calculated and validated using the GRASPVALIDATOR function [1]. Its algorithm is shown on Figure 15.

Algorithm : GRASPVALIDATOR($T^{world}_{gripper}$,q,v^{world},δ)

```
1  SETCONFIGURATION(gripper,q)
   /* Check collision with gripper along negative direction.   */
2  GripperTransforms ← ∅
3  for d ∈ [δ, δ + ε] do
4      T ← [ I_3x3   -dv^world ] T^world_gripper
           [   0         1    ]
5      SETTRANSFORM(gripper,T)
6      if CHECKCOLLISION(gripper) then
7          return ∅
8      ADD(GripperTransforms,T)
9  end
10 SETTRANSFORM(gripper, T^world_gripper)
11 q_new ← GRASPSTRATEGY()
12 SETCONFIGURATION(gripper,q_new)
13 if ALLCOLLISIONS(gripper) ⊆ {target} then
14     return GripperTransforms
15 return ∅
```

Figure 15: The GRASPVALIDATOR algorithm

Before a grasp can be determined valid in the current environment, the gripper needs to be collision-free from other obstacles and have a room to maneuver along its direction of approach. In Figure 16, some examples are shown to be invalid in the environment. Green block is the target object.

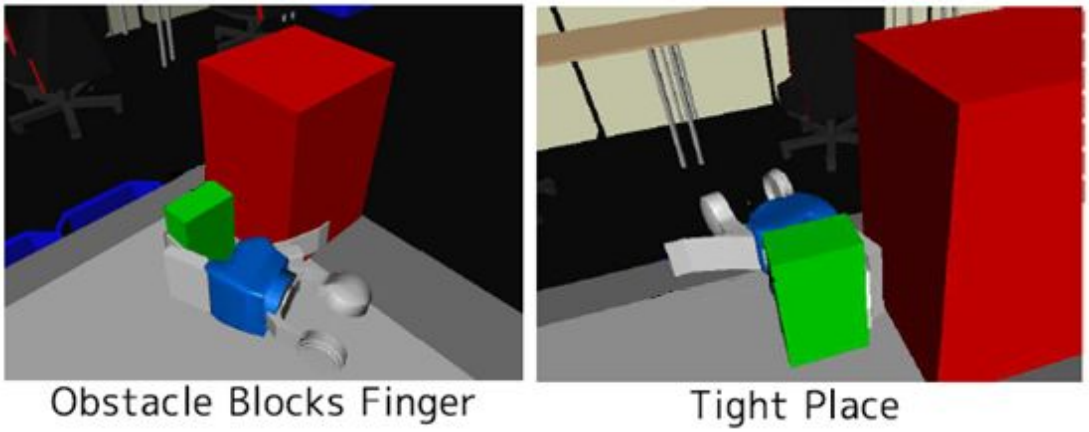


Figure 16: Example of invalid grasp in the environment

First, the gripper is set to its pre-shape and checked for the environment collisions. Then it is set to the final gripper final configuration to make sure only the gripper's contacts are from the target object. The GOALSAMPLER_GRASPS_ARM function [1] which is shown in Figure 17 validates the robot arm with the current environment.

Algorithm : GOALSAMPLER_GRASPS_ARM()

1 SETCONFIGURATION(gripper,q)
2 **for** (**target**, $T^{target}_{gripper}, q_{gripper}, v^{target}, \delta) \in \mathcal{G}_{stable}$ **do**
3 GripperTransforms $\leftarrow$ GRASPVALIDATOR($T^{world}_{target} T^{target}_{gripper}, q_{gripper}, T^{world}_{target} v^{target}, \delta$)
/* Validate all gripper transforms. */
4 **if** $\forall_{T \in GripperTransforms}$, $\exists\, q_{arm}$ **s.t.** $q_{arm} \in IK(T) \bigwedge (q, q_{gripper}) \in \mathcal{C}_{\text{free}}$ **then**
5 $T \leftarrow$ LASTELEMENT(GripperTransforms)
6 **for** $q_{arm} \in IK(T)$ **do**
7 **if** $(q_{arm}, q_{gripper}) \in \mathcal{C}_{\text{free}}$ **then**
8 **return** $(q_{arm}, q_{gripper})$
9 **end**
10 **end**

Figure 17: The GOALSAMPLER_GRASPS_ARM algorithm

All collisions are checked by the ALLCOLLISIONS function. Then the set of

collision-free transforms returned by GRASPVALIDATOR are checked for inverse kinematics solutions. If there is a grasp transformation that does not have a solution, the robot will not be able to complete the task. Otherwise, the transformation is generated as a goal robot configuration to feed the search algorithms.

III.2.Path search algorithms

The search algorithms used by OpenRAVE are Rapidly-exploring Random Trees (RRT) [17] [18] [19] and A* [20] [21] [22]. RRT have been shown to explore the feasible solution space quickly, so they are ideal for high dimensional search spaces like manipulator arms since the unique advantage of RRT is that they can be directly applied to nonholonomic and kinodynamic planning. This advantage stems from the fact that RRT do not require any connections to be made between pairs of configurations (or states), while probabilistic roadmaps typically require tens of thousands of connections [31]. A* algorithms give more power to heuristics to guide the path search. It uses a best-first search and finds a least-cost path from a given initial node to one goal node (out of one or more possible goals). As A* traverses the graph, it follows a path of the lowest expected total cost or distance, keeping a sorted priority queue of alternate path segments along the way [32].

The RRT algorithm is designed for efficiently searching non-convex, high-dimensional search spaces. The tree is constructed in such a way that any sample in the space is added by connecting it to the closest sample already in the tree [15]. Based on the RRT-Connect algorithm [16], some modifications have been made in OpenRAVE and named as RRTCONNECT* [1] shown in Figure 18.

Algorithm : RRTCONNECT*(q_{init})

```
/* ρ ∈ [0,1] - uniform random variable                                */
1  INIT(T_a, q_init); INIT(T_g, SAMPLE(C_goal)); T_b ← T_g
2  for iteration = 1 to N do
3      if ρ ≤ GOALSAMPLEPROBABILITY() then
4          ADDROOT(T_g,SAMPLE(C_goal))
5      q_new ← EXTEND(T_a,SAMPLE(C_free))
6      if q_new ≠ ∅ then
7          if CONNECT(T_b,q_new) then
8              return PATH(T_a,T_b)
9      SWAP(T_a,T_b)
10 end
11 return ∅
```

Figure 18: RRTCONNECT* algorithm

The RRTCONNECT* algorithm maintains a tree of configurations, with edges representing the ability for the robot to move from one configuration to the other. In every step the RRT randomly grows trees simultaneously from both the initial and goal configurations by first sampling a random configuration of 6 DOF in this case then extend both trees to it in order to find the path more efficiently and rapidly. The sampling is completed by the function SAMPLEWITHCONSTRAINTS [1] shown in Figure 19 and extending the tree by EXTENDWITHCONSTRAINTS [1] shown in Figure 20.

Algorithm : SAMPLEWITHCONSTRAINTS($\mathcal{C}$)

```
1 while true do
2     q ← PROJECTCONSTRAINTS(f_C,SAMPLE(C))
3     if q ∈ C then
4         return q
5 end
```

Figure 19: The SAMPLEWITHCONSTRAINTS algorithm

Algorithm : EXTENDWITHCONSTRAINTS($\mathcal{T}$,q)

```
/* Uses definitions from [Kuffner and LaValle (2000)]        */
q_near ← NEARESTNEIGHBOR(q,T)
q_new ← PROJECTCONSTRAINTS(f_C,NEWCONFIG(q,q_near))
if q_new ∈ C_free then
    ADDVERTEX(T,q_new)
    ADDEDGE(T,q_near,q_new)
    return q_new
return ∅
```

Figure 20: The EXTENDWITHCONSTRAINTS algorithm

The motion planner checks if two trees can be connected and a path is computed after every extension. Each robot configuration added to the tree is check by the ALLCOLISION function to guarantee a collision-free path. A* algorithm which are used for problems whose state space is much smaller helps plotting an efficiently traversable path between nodes. The path search algorithms take an initial robot configuration and attempts to find a path to the goal configuration while maintaining constraints. By iterations, the algorithm is trying to find a solution path and return it. Once the path has been found, the manipulation planning is complete.

It is very difficult to measure the time-complexities of manipulation planner because of the high dependency on the environment and sampling distribution. Additionally, the RRT families of planners are known to have long tails for planning time distributions, making it difficult to prove theoretical time bounds for them [1]. Therefore, we will test the response of OpenRAVE manipulation planner in Chapter V and the results are also going to be discussed.

Chapter IV

SIMULATION ENVIRONMENT

In this chapter, we will discuss the setup of simulation environment including a robot and other objects in OpenRAVE. Also, the parameters of physical engine will be introduced.

IV.1.Robotic Simulation

All models in OpenRAVE are stored in XML files [23]. In general, a model can be a definition of an object, a robot or a set of objects and robots. In the XML-format used by OpenRAVE, the definition of an object must be the type of *Kinbody.* A *Kinbody* can consist of several rigid bodies connected by joints. A definition of a robot is of the type *Robot*, although the robot will still consist of at least one *Kinbody* definition [24].

OpenRAVE provides a robot database with ample robot models for simulation. Among them, the 6 Degrees Of Freedom (DOF) robot arm is one of the commonly used robots for object grasping. DOF is a term describing the number of parameters that define a robot's state [25]. In the Cognitive Robotics Laboratory at Vanderbilt University, there is a 6-DOF robot arm *MOTOMAN HP3JC* as shown in Figure 21, which can perform real-world tests in the future. In this thesis, we will use the 6-DOF robot named *barrettsegway.robot.xml* provided by the OpenRAVE Robots Database, as shown in Figure 22.

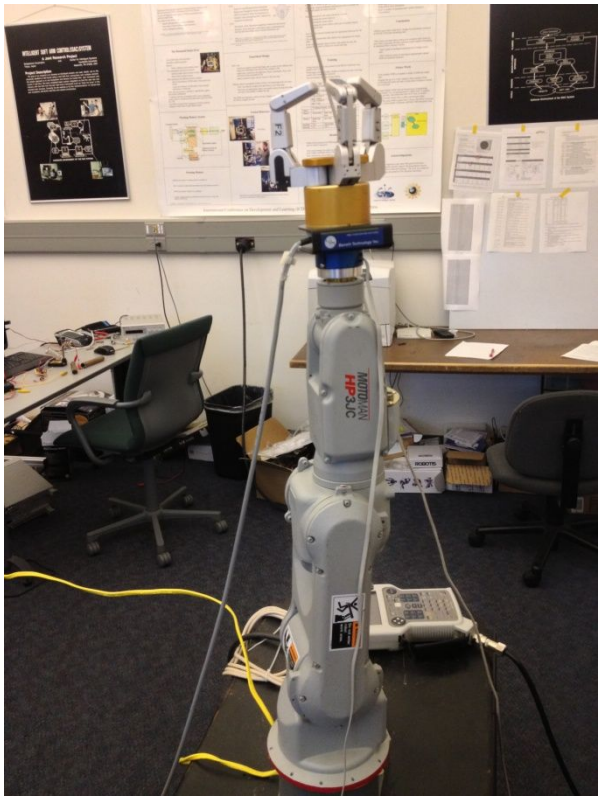


Figure 21: *MOTOMAN HP3JC* in the Cognitive Robotic Laboratory

Figure 22: The 6-DOF robot arm model for simulation test

The model shown in figure 22 is consists of a robot arm and the Segway mobile base. Since the robot will be stationary in the simulation, the segway is locked during the simulation test. The end-effector of this robot is a three fingers system. During the grasping, three fingers will be closed simultaneously.

IV.2.Test Environment Setup

In order to perform the simulation test of Tower of Hanoi puzzle, all other objects need to be created in OpenRAVE including disks, pegs and a table. Moreover, parameters of simulation environment are also need to be set. Three simulation environments with different initial disk positions have been created for the test.

IV.2.1.Pegs Setup

There are three identical pegs in the simulation environment as shown in Figure 23.

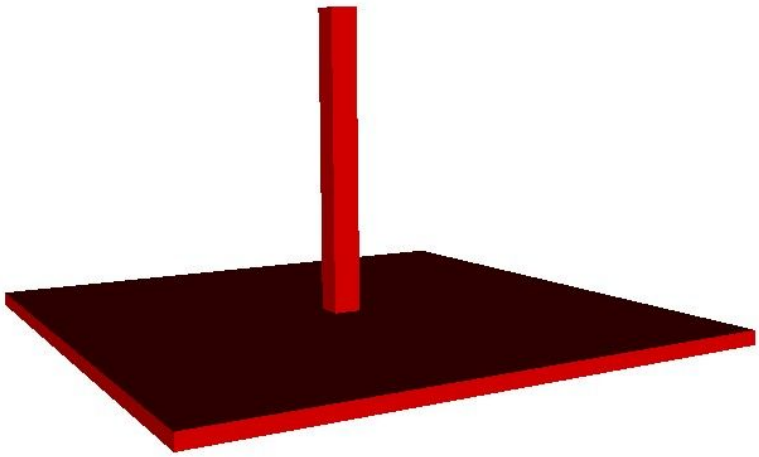


Figure 23: The Peg model in OpenRAVE

They placed on the table in parallel. Each peg is consists of two cuboids, one as the base and another as the column. The peg is set to be dynamic so it might be

moved on the table. The density is $1000\ \mathrm{kg/m^3}$.

IV.2.2.Disks Setup

Three disks in different sizes have been created in environment as shown in Figure 24.

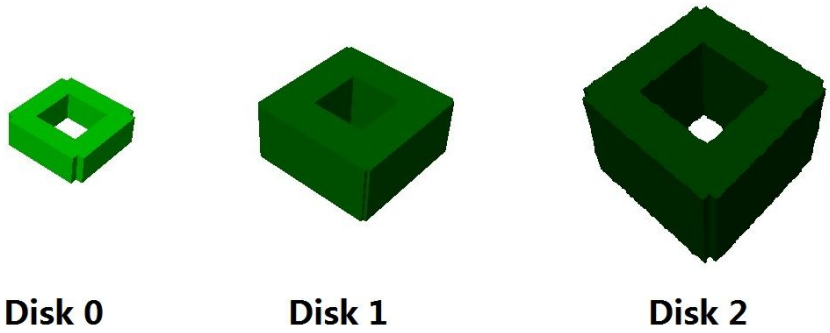


Figure 24: Disks model in OpenRAVE

Four cuboids as each side form the disk. The size of disks increases with the deeper color. Inside the disks, there are holes that should be in correct sizes, since the disk on the top can be dropped into the hole if it is too large or the column of peg cannot be go through if it is too small. The heights of disks are identical. The disks are set to dynamic with the density of $1000\ \mathrm{kg/m^3}$.

IV.2.3.Table and Floor Setup

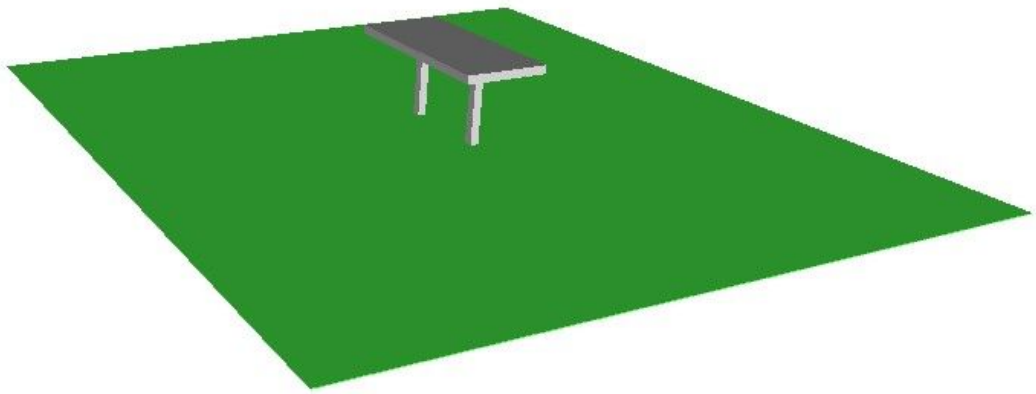


Figure 25: Floor and table in OpenRAVE

As shown in Figure 25, floor and table are two surfaces parallel to the *xy* plate. Both of them are set to be static so they cannot be fall along the *z*-axis due to the gravity. The height of table above the floor is set to be within the working space of robot.

IV.3.Physics Engine Setup

The physics engine is computer software that provides an approximate simulation of certain physical systems, such as rigid body dynamics (including collision detection), soft body dynamics, and fluid dynamics, of use in the domains of computer graphics and so on. There are generally two classes of physics engines: real-time and high-precision. High-precision physics engines require more processing power to calculate very precise physics and are usually used by scientists and computer animated movies. Real-time physics engines—as used in simulation and other forms of interactive computing—use simplified calculations and decreased accuracy to compute in time to respond at an appropriate rate for real-time simulation [26]. Many real-time physics engines are open source including Open Dynamic

Engine (ODE) [27], Simulation Open Framework Architecture (SOFA) [28], Open Physics Initiative [29], Tokamak physics engine [30] and etc.

OpenRAVE uses Open Dynamics Engine as its physical engine. It is a fast, flexible and robust industrial quality library with built-in collision detection for simulating articulated rigid body dynamics. ODE is designed to be used in real-time simulation [27].

In OpenRAVE, we turn on the physics engine and apply the gravity along z axis with the gravity constant $G \quad 9.8\ \mathrm{m/s}$. The density of both pegs and disks are set to be $1000\ \mathrm{kg/m^3}$ as mentioned before. The friction coefficient is set to be 0.3. In order to perform the real-time simulation fast and smoothly, we set the time step to be 0.01 second.

Chapter V

SIMULATION EXPERIMENTS

This chapter covers the procedures and results of the simulation experiments. Three different Tower of Hanoi scenes with different initial disks positions have been built for the tests. The results will be discussed later.

V.1.Goal and Procedures

The goal of this thesis is to use the Tower of Hanoi puzzle as a test example to develop a robotic real-time simulation with sensor feedback and evaluate the performance of the simulator. The robot should pick up and drop off the disks in correct order without any grasping, collision or other problems. Obstacles will be added to further test the ability of simulator trajectory planning. Also, the simulator should be suitable and reliable enough to complete the whole tests. Ultimately, the robot in the simulation environment should successfully accomplish the Tower of Hanoi puzzle in all test scenes. OpenRAVE is the simulator we choose in this thesis. We will measure and demonstrate the time in real-time simulation environment and actual running time of simulation. The computer which is running OpenRAVE is COTS (commercial-off-the-shelf) and readily available in the market. The specification of computer is listed in the Figure 26:

Processor	Intel Core i5 M450 @ 2.40GHz
Chipset	Intel QM57
System Memory	4 GB DDR3 1067MHz
Hard Drive	500 GB 7200rpm
Display Adapters	NVIDIA NVS 3100M 512MB
Operating System	Windows 7 Home 32bit SP1

Figure 26: The specification of computer for simulation tests

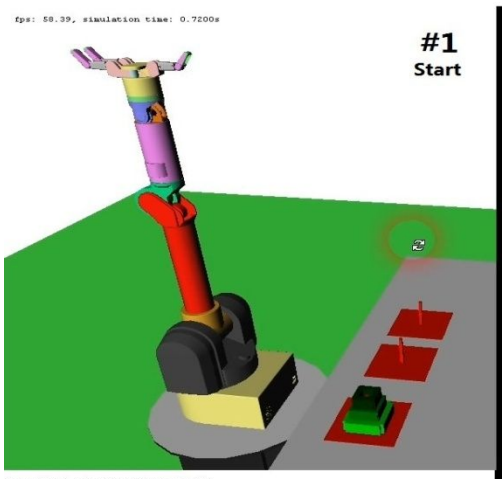
Four scenes files which are *hanoi111.xml, hanoi212.xml, hanoi331.xml* and *hanoi331obs.xml* with different initial disks positions have been created in OpenRAVE for the test. The robot will try to accomplish these three puzzles in the following simulation experiments.

V.1.1.Simulation Experiment I

hanoi111.xml which is the traditional Tower of Hanoi puzzle will be loaded in this experiment. As shown in Figure 27, the searching algorithm successfully finds the target (*333*) and the movement sequence has generated correctly by the movement translation. Then, the robot begins to execute seven movement steps orderly as shown in Figure 28. In Figure 28, each picture has been marked a distinctive increasing number as time elapses. Sixteen pictures represent the whole procedure about how the robot accomplishes the traditional Tower of Hanoi puzzle by grasping and dropping the disks by order.

```
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 2.6.6
>>> ================================ RESTART ================================
===
>>>
could not import scipy.optimize.leastsq
It's time to play !!


Level 0 331.0

Level 1 131.0  231.0  332.0

Level 2 121.0  211.1  132.2  232.2

Level 3 221.0  321.0  311.1  111.1  122.2  212.3

Level 4 223.0  222.4  322.4  312.5  112.5

Level 5 323.0  123.0  113.4

Level 6 313.0  133.1  213.2  313.2

Level 7 233.1  333.1

State sequence [331, 131, 121, 221, 223, 123, 133, 333]

Movement sequence [228, 136, 285, 93, 259, 162, 291]

Step 1 : Disk 2     C ---> A   done!
Step 2 : Disk 1     C ---> B   done!
Step 3 : Disk 2     A ---> B   done!
Step 4 : Disk 0     A ---> C   done!
Step 5 : Disk 2     B ---> A   done!
Step 6 : Disk 1     B ---> C   done!
Step 7 : Disk 2     A ---> C   done!

Game over !!!
```

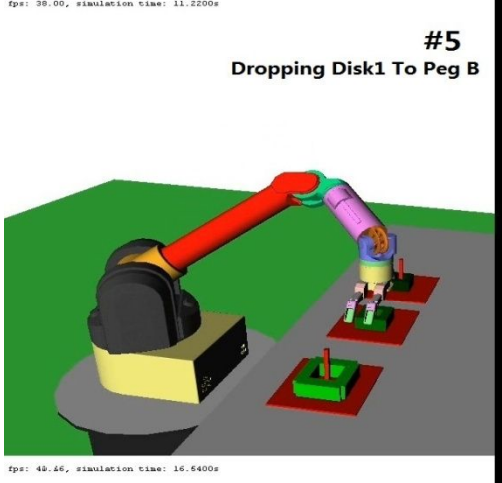
Figure 27: The simulation result of *Hanoi111.xml*

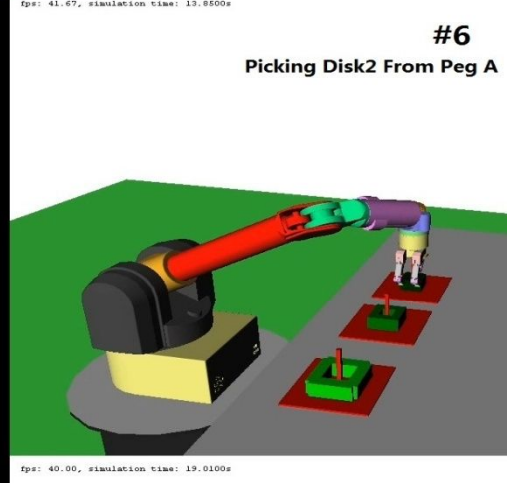

fps: 58.39, simulation time: 0.7200s
#1
Start

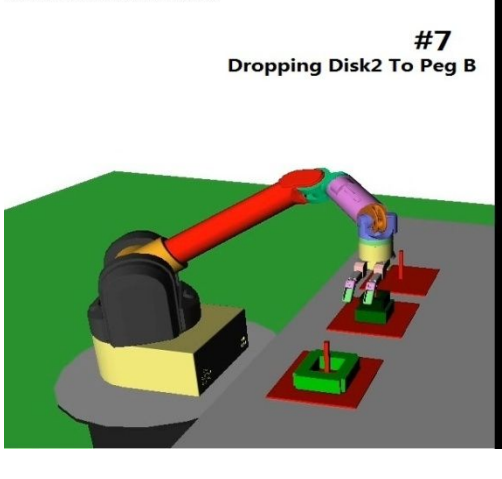

fps: 40.10, simulation time: 3.2100s
#2
Picking Disk2 From Peg C

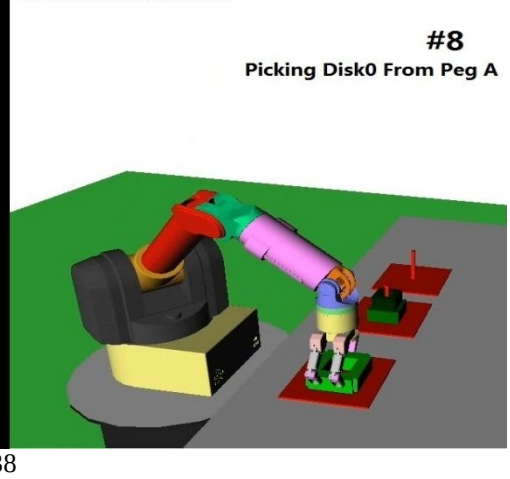

fps: 39.29, simulation time: 6.2000s
#3
Dropping Disk2 To Peg A


fps: 40.80, simulation time: 8.6800s
#4
Picking Disk1 From Peg C


fps: 38.00, simulation time: 11.2200s
#5
Dropping Disk1 To Peg B

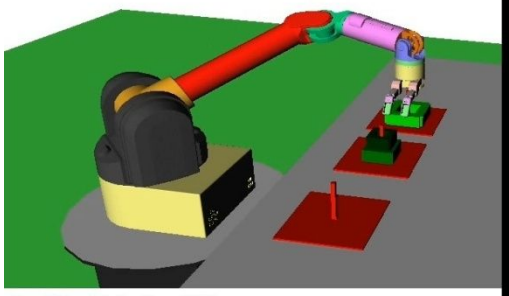

fps: 41.67, simulation time: 13.8500s
#6
Picking Disk2 From Peg A


fps: 40.46, simulation time: 16.6400s
#7
Dropping Disk2 To Peg B


fps: 40.00, simulation time: 19.0100s
#8
Picking Disk0 From Peg A

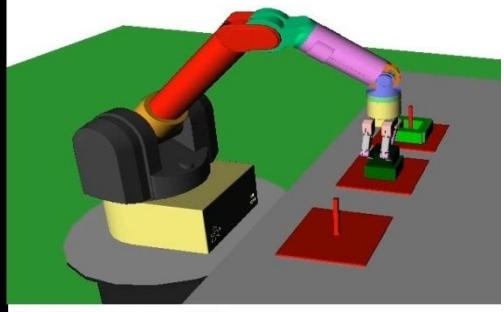
fps: 44.32, simulation time: 21.6500s

#9
Dropping Disk0 To Peg C



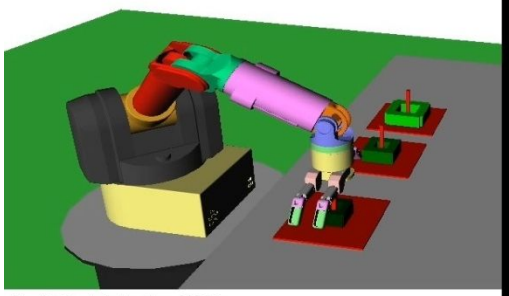
fps: 43.72, simulation time: 24.1400s

#10
Picking Disk2 From Peg B



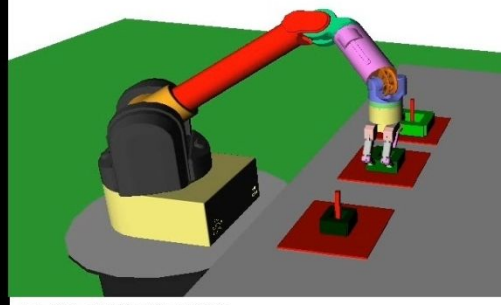
fps: 50.31, simulation time: 27.2300s

#11
Dropping Disk2 To Peg A



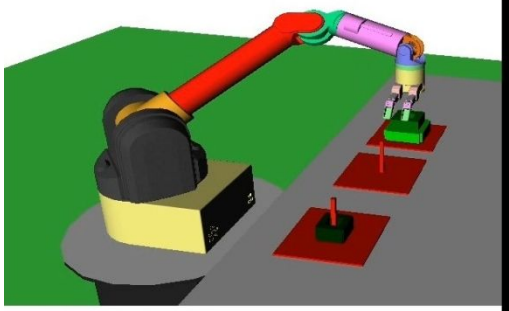
fps: 43.87, simulation time: 29.7600s

#12
Picking Disk1 From Peg B



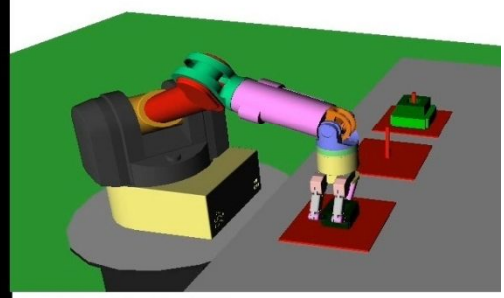
fps: 50.00, simulation time: 31.0000s

#13
Dropping Disk1 To Peg C



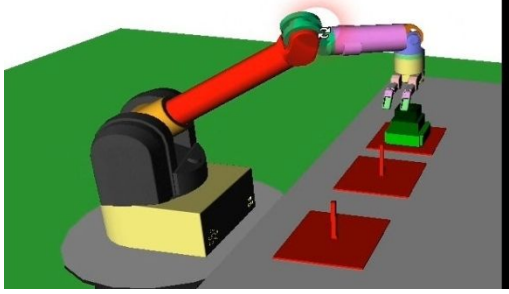
fps: 43.60, simulation time: 32.6500s

#14
Picking Disk2 From Peg A



fps: 40.10, simulation time: 34.5400s

#15
Dropping Disk2 To Peg C



fps: 41.67, simulation time: 37.0100s

#16
Done

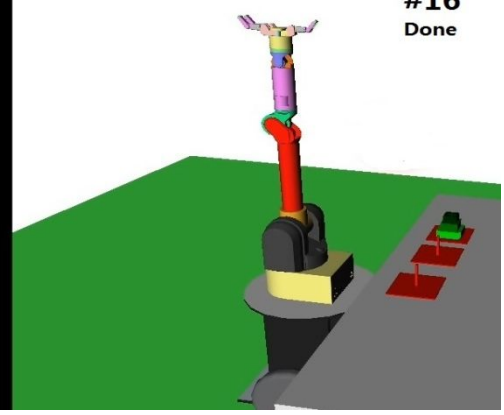


Figure 28: The OpenRAVE simulation of experiment I

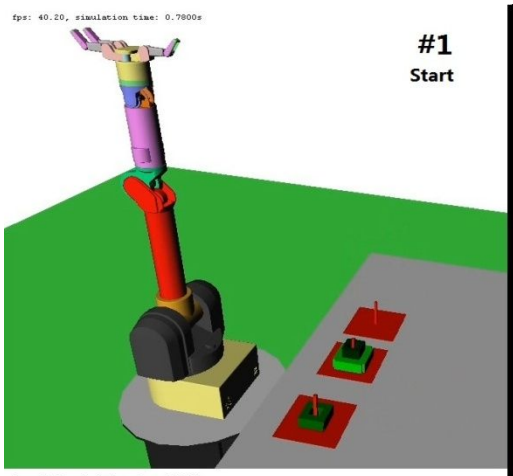
As shown in picture #1 of Figure 28, the initial disks positions are traditional Tower of Hanoi puzzle. In picture #2, robot picks the Disk2 up and drops it off in picture #3. This movement step has been displayed on the GUI of Python, as shown in Figure 27. All of the rest movements are shown in each picture. In picture #16, after taking seven movement steps (fourteen pickups and drop-offs), the robot successfully completes the traditional Tower of Hanoi puzzle without any problems and reset to the initial position.

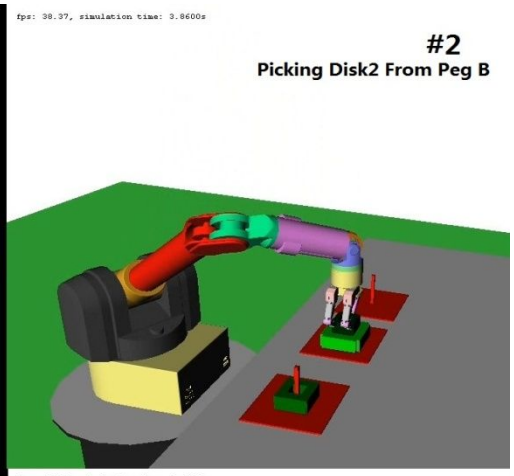
V.1.2.Simulation Experiment II

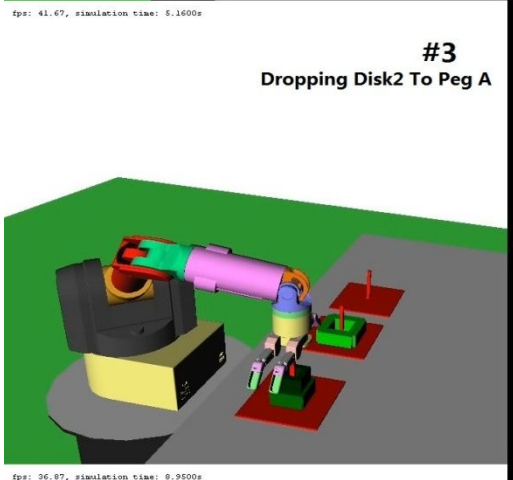
In this simulation experiment, we will load the scene *hanoi212.xml* with a random initial disks position as we discussed at the beginning of chapter II. After running the program, the GUI of Python displays as following in Figure 29:

```
IDLE 2.6.6
>>> ================================ RESTART ================================
>>>
could not import scipy.optimize.leastsq
It's time to play !!


Level 0 212.0

Level 1 312.0  112.0  232.0

Level 2 322.0  113.1  332.2  132.2

Level 3 122.0  222.0  213.1  313.1  331.2  122.3

Level 4 233.2  323.3  131.4  231.4

Level 5 333.0  133.0

State sequence [212, 112, 113, 213, 233, 333]

Movement sequence [258, 63, 286, 192, 261]

Step 1 : Disk 2     B ---> A   done!
Step 2 : Disk 0     B ---> C   done!
Step 3 : Disk 2     A ---> B   done!
Step 4 : Disk 1     A ---> C   done!
Step 5 : Disk 2     B ---> C   done!
```

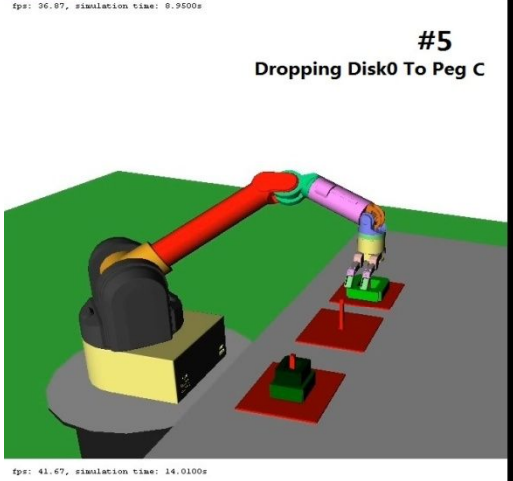
Figure 29: The simulation result of *Hanoi212.xml*


fps: 40.20, simulation time: 0.7800s
#1
Start

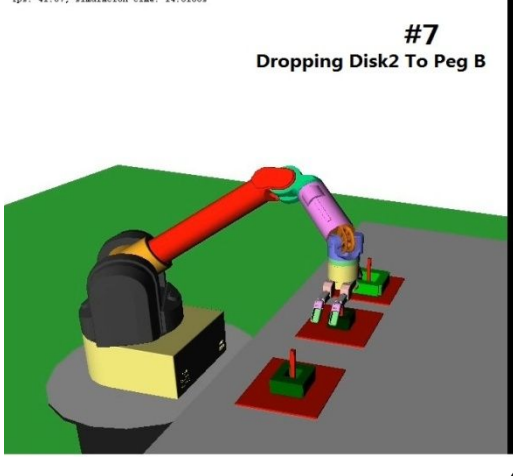

fps: 38.37, simulation time: 3.8600s
#2
Picking Disk2 From Peg B

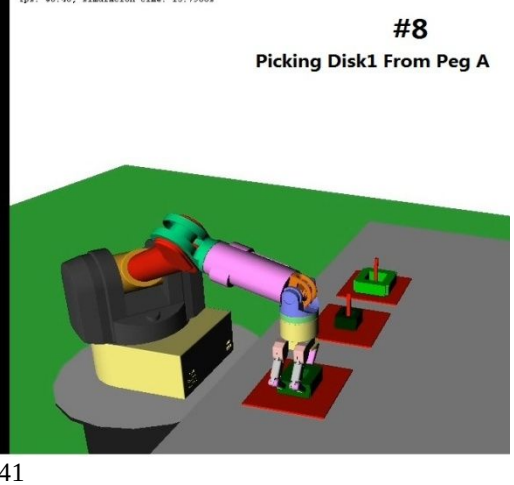

fps: 41.67, simulation time: 5.1600s
#3
Dropping Disk2 To Peg A


fps: 40.00, simulation time: 7.6700s
#4
Picking Disk0 From Peg B


fps: 36.87, simulation time: 8.9500s
#5
Dropping Disk0 To Peg C


fps: 40.00, simulation time: 11.4400s
#6
Picking Disk2 From Peg A


fps: 41.67, simulation time: 14.0100s
#7
Dropping Disk2 To Peg B


fps: 38.40, simulation time: 15.7900s
#8
Picking Disk1 From Peg A

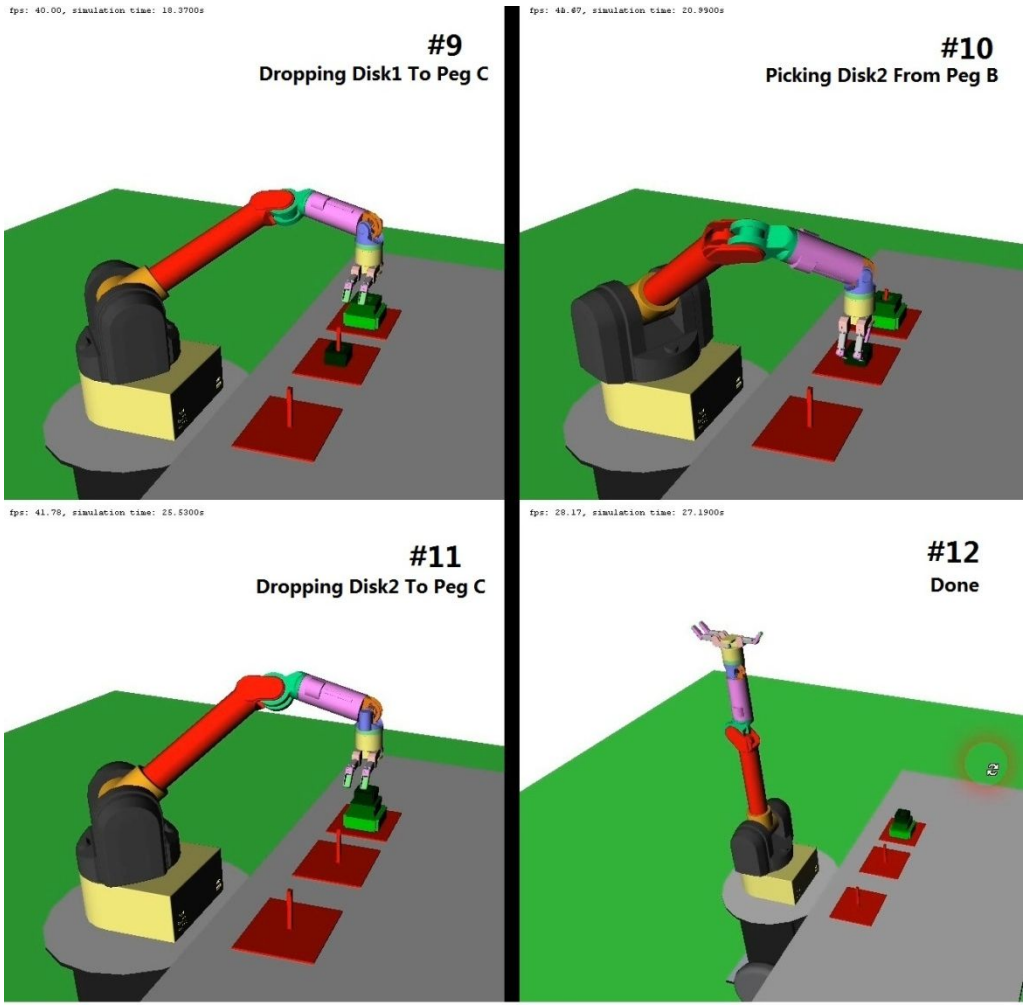


Figure 30: The OpenRAVE simulation of experiment II

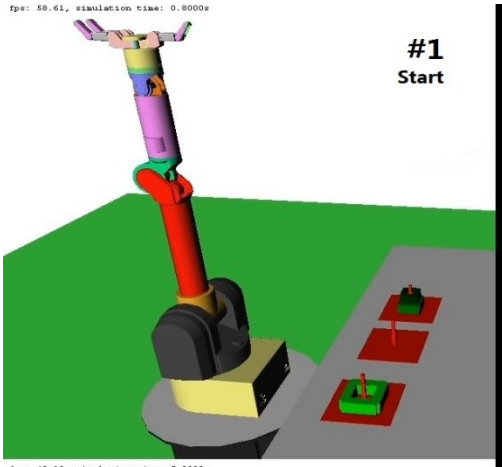
The searching algorithm has successfully discovered the connections between each note in each level and solved the puzzle with a random initial disks position this time. Unlike the traditional puzzle, only five movement steps have been derived to complete the puzzle in this situation. As shown in picture #1 of Figure 30, the initial disks position is exactly the one given in chapter II. The robot executes the grasping and dropping disks commands as the movement sequences indicated. Still, when the robot performs each movement, the information has been displayed on the GUI of Python. The simulator has accomplished this scene of puzzle without any grasping, collision or other errors as it shows in picture #12.

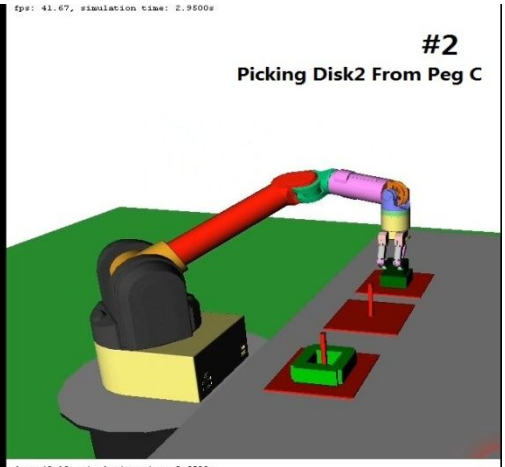
V.1.3.Simulation Experiment III

We will test another case of Tower of Hanoi puzzle with random initial disks position. In this simulation, we try to finish the puzzle without obstacles first. Then, in comparison, three obstacles have been put on the table to test whether or not the robot is able to complete the puzzle without hitting the obstacles. Three blue obstacles are set to be dynamic; which means they can fall onto the table or slide if the robot arm hits them. We demonstrate the result of the simulation as Figure 31, and the graphical OpenRAVE simulation is shown as Figure 32 and 33.

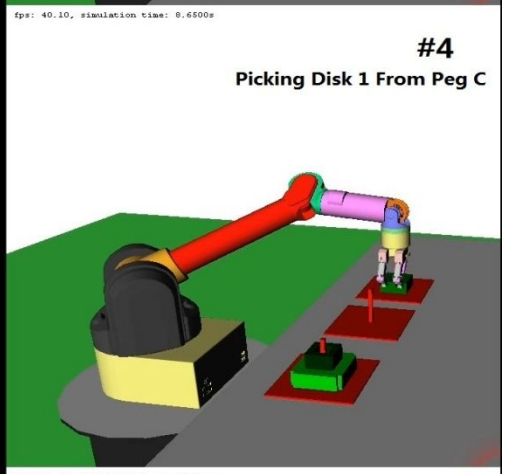
```
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 2.6.6
>>> ================================ RESTART ================================
===
>>>
could not import scipy.optimize.leastsq
It's time to play !!


Level 0 331.0

Level 1 131.0  231.0  332.0

Level 2 121.0  211.1  132.2  232.2

Level 3 221.0  321.0  311.1  111.1  122.2  212.3

Level 4 223.0  222.4  322.4  312.5  112.5

Level 5 323.0  123.0  113.4

Level 6 313.0  133.1  213.2  313.2

Level 7 233.1  333.1

State sequence [331, 131, 121, 221, 223, 123, 133, 333]

Movement sequence [228, 136, 285, 93, 259, 162, 291]

Step 1 : Disk 2     C ---> A   done!
Step 2 : Disk 1     C ---> B   done!
Step 3 : Disk 2     A ---> B   done!
Step 4 : Disk 0     A ---> C   done!
Step 5 : Disk 2     B ---> A   done!
Step 6 : Disk 1     B ---> C   done!
Step 7 : Disk 2     A ---> C   done!
```

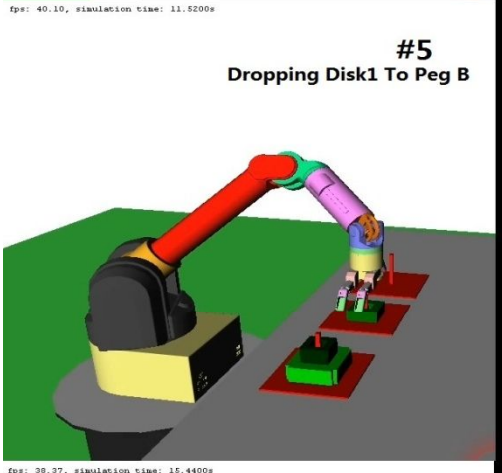
Figure 31: The simulation result of experiment III


fps: 58.61, simulation time: 0.8000s
#1
Start


fps: 41.67, simulation time: 2.9500s
#2
Picking Disk2 From Peg C


fps: 40.10, simulation time: 5.9000s
#3
Dropping Disk2 To Peg A


fps: 40.10, simulation time: 8.6500s
#4
Picking Disk 1 From Peg C


fps: 40.10, simulation time: 11.5200s
#5
Dropping Disk1 To Peg B


fps: 41.67, simulation time: 13.3200s
#6
Picking Disk2 From Peg A


fps: 38.37, simulation time: 15.4400s
#7
Dropping Disk2 To Peg B


fps: 43.60, simulation time: 17.1200s
#8
Picking Disk0 From Peg A

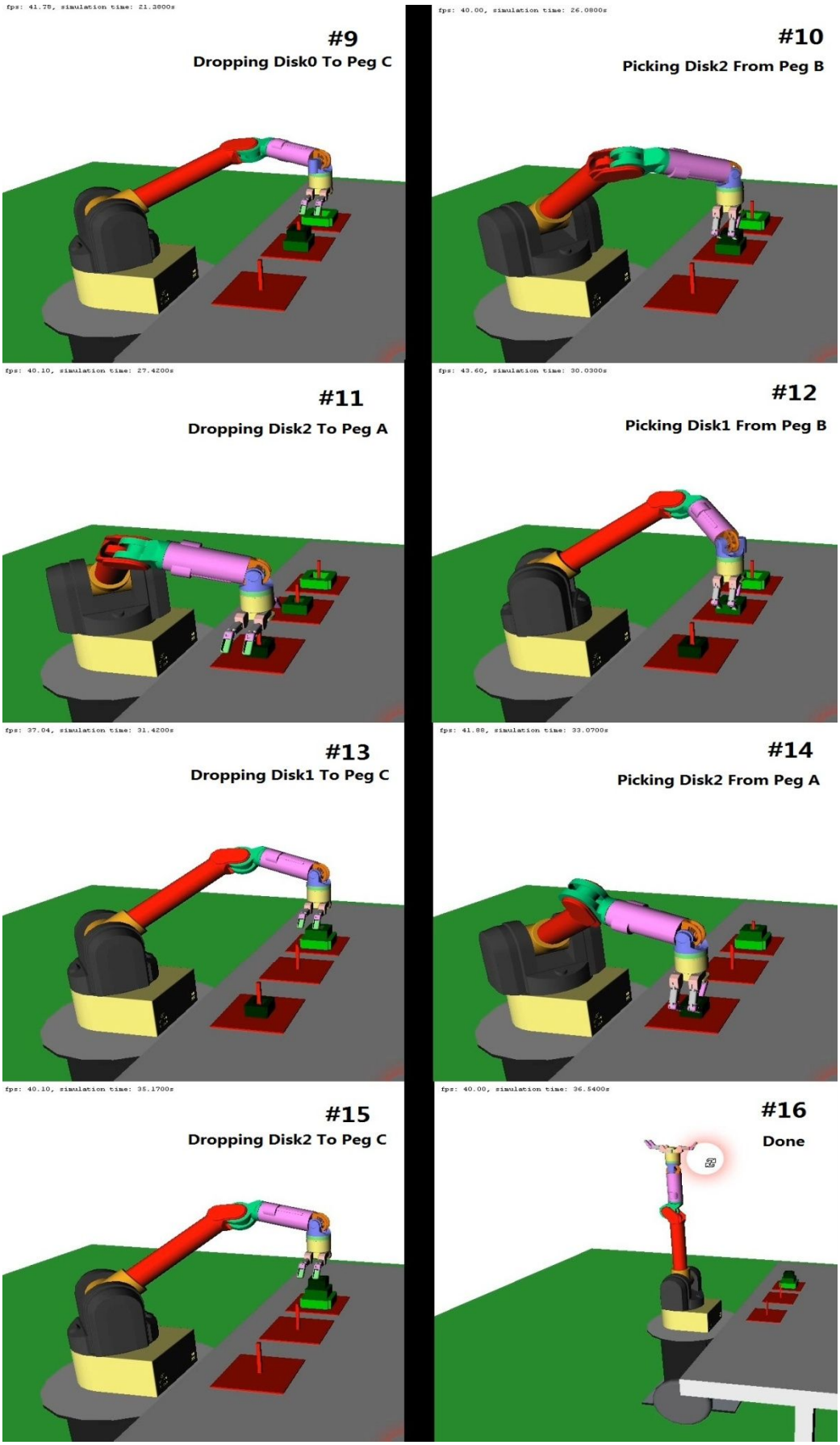
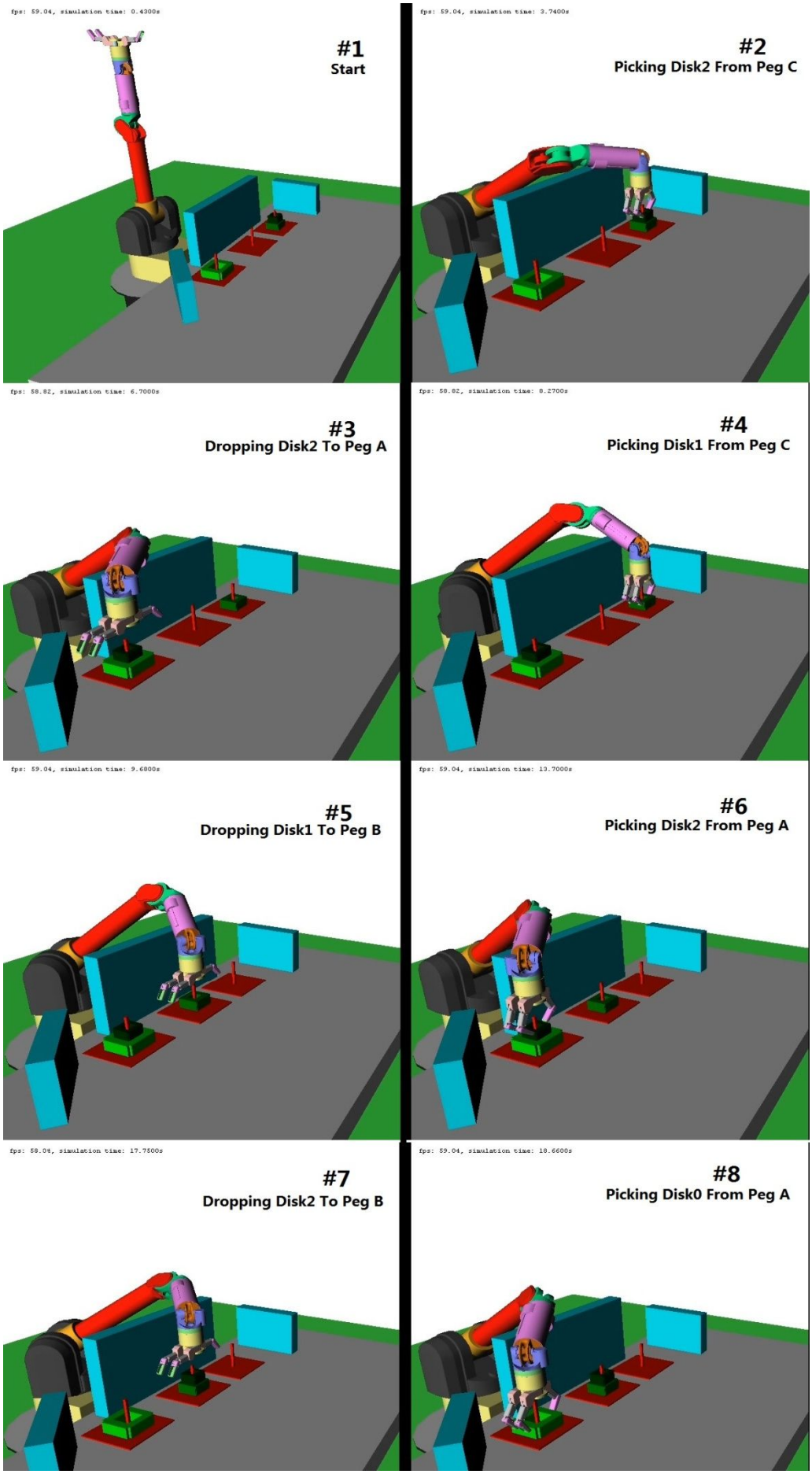


Figure 32: The OpenRAVE simulation of *Hanoi331.xml*


fps: 59.04, simulation time: 0.4300s
#1
Start
fps: 59.04, simulation time: 3.7400s
#2
Picking Disk2 From Peg C
fps: 58.82, simulation time: 6.7000s
#3
Dropping Disk2 To Peg A
fps: 58.82, simulation time: 8.2700s
#4
Picking Disk1 From Peg C
fps: 59.04, simulation time: 9.6800s
#5
Dropping Disk1 To Peg B
fps: 59.04, simulation time: 13.7000s
#6
Picking Disk2 From Peg A
fps: 58.04, simulation time: 17.7500s
#7
Dropping Disk2 To Peg B
fps: 59.04, simulation time: 18.6600s
#8
Picking Disk0 From Peg A

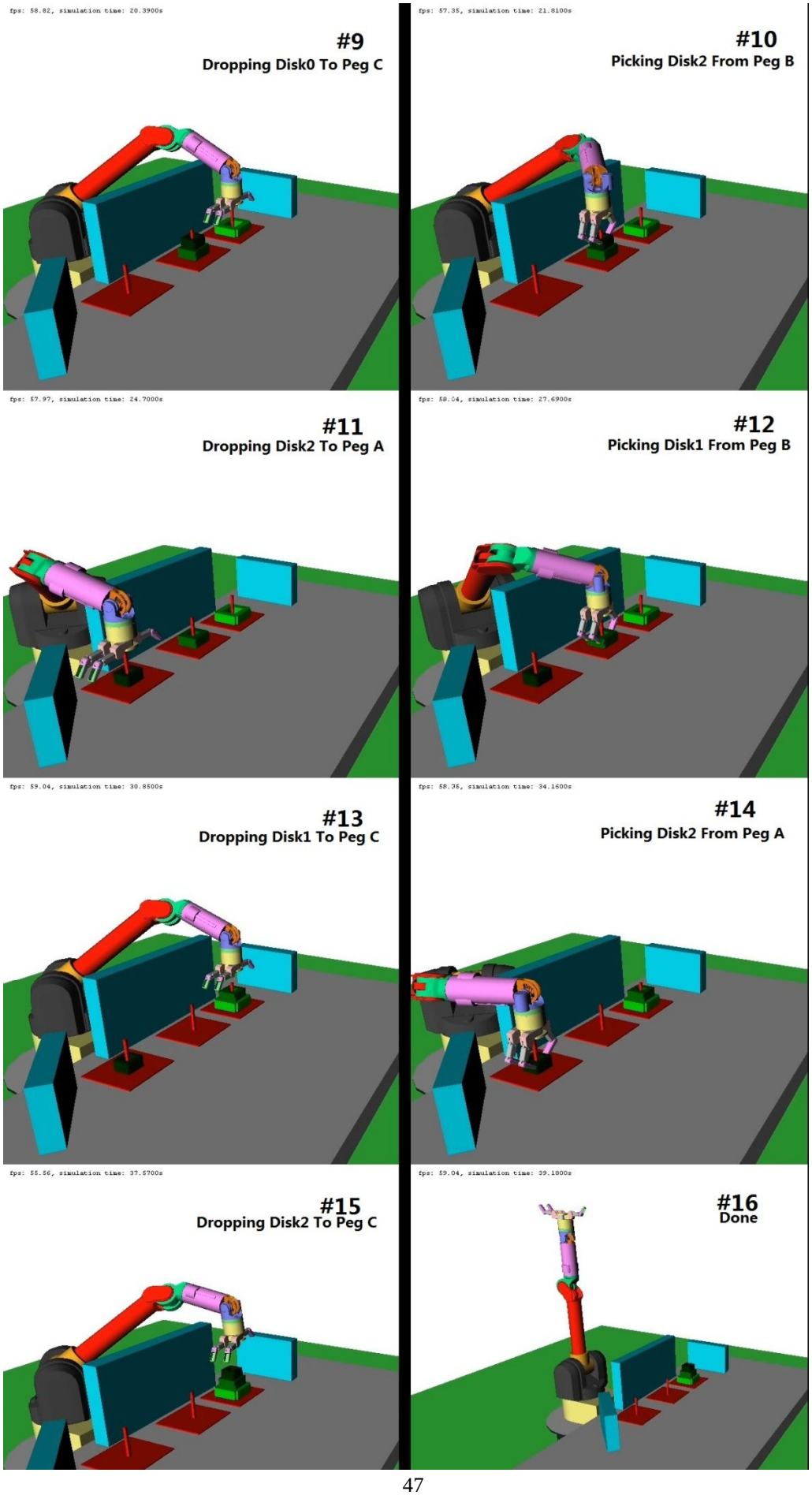

fps: 58.82, simulation time: 20.3900s
#9
Dropping Disk0 To Peg C
fps: 57.35, simulation time: 21.6100s
#10
Picking Disk2 From Peg B
fps: 57.97, simulation time: 24.7000s
#11
Dropping Disk2 To Peg A
fps: 58.04, simulation time: 27.6900s
#12
Picking Disk1 From Peg B
fps: 59.04, simulation time: 30.8500s
fps: 58.36, simulation time: 34.1600s
#13
Dropping Disk1 To Peg C
#14
Picking Disk2 From Peg A
fps: 55.56, simulation time: 37.5700s
fps: 59.04, simulation time: 39.1800s
#15
Dropping Disk2 To Peg C
#16
Done

Figure 33: The OpenRAVE simulation of *Hanoi331obs.xml*

The initial disks positions are shown in picture #1 in Figure 32 and 33. Unlike the previous simulation experiment, the robot needs seven movement steps which is the maximum steps a three disks Tower of Hanoi puzzle can be achieved to complete the puzzle. The algorithm once again reaches the correct order and the disks are precisely grasped and dropped. In the scene without obstacles, the robot in the simulation environment has successfully accomplished the Tower of Hanoi puzzle. When the obstacles have been added, the manipulation planner amazingly let the robot avoid all the obstacles in all movement without hitting them, although the task of grasping and dropping objects is extremely challenging since three objects blocks three directions of the targets. The robot configurations of each movement with or without obstacles can be different due to the generation of distinct final goal robot configuration. We will discuss the manipulation planning algorithms of OpenRAVE later. The simulator once again meets the challenge and fulfills the task in this simulation experiment. Ultimately, the robot in the simulation environment has successfully accomplished the Tower of Hanoi puzzle in all test scenes.

V.2.Results and Discussion

We have accomplished the tests of Tower of Hanoi puzzle in four different scenes. The performance of OpenRAVE which simulates a 6-DOF robot arm using sensor feedback turns to be competent and reliable enough for handling the robot simulations including object grasping, solving inverse kinematics, motion planning and so on forth.

V.2.1.Real-time and Response of OpenRAVE

Evaluating the OpenRAVE is based on a somewhat subjective impression; therefore, we will try to quantify some of the standards such as time spent during the simulation.

Action of Picture Number	Real-time Simulation Timeline (Seconds)	Actual Timeline (Seconds)	Waiting Time (Seconds)
1	0.72	1.0	---
2	3.21	4.0	0.3
3	6.20	35.0	24.8
4	7.69	39.0	1.7
5	11.22	48.0	5.2
6	13.85	57.0	2.8
7	16.54	97.0	35.5
8	19.01	102.0	1.9
9	21.65	107.0	2.2
10	24.14	112.0	1.8
11	27.23	158.0	40.7
12	29.70	165.0	2.9
13	31.00	175.0	7.8
14	32.65	180.0	1.1
15	34.54	193.0	10.9
16	37.01	196.0	0.6

Table 1 Simulation time of experiment I

Action of Picture Number	Real-time Simulation Timeline (Seconds)	Actual Timeline (Seconds)	Waiting Time (Seconds)
1	0.78	1.0	---
2	3.86	4.0	0.2
3	5.16	12.0	6.7
4	7.67	18.0	1.9
5	8.95	29.0	8.8
6	11.44	34.0	1.9
7	14.01	66.0	29.2
8	15.79	71.0	1.1
9	18.37	90.0	14.9
10	20.99	96.0	1.4
11	25.53	131.0	25.8
12	27.19	137.0	2.1

Table 2 Simulation time of experiment II

Action of Picture Number	Real-time Simulation Timeline (Seconds)	Actual Timeline (Seconds)	Waiting Time (Seconds)
1	0.80	1.0	---
2	2.95	3.0	0.2

3	5.90	9.0	3.4
4	8.65	21.0	2.1
5	11.52	67.0	35.3
6	13.32	73.0	1.3
7	15.44	94.0	17.7
8	17.12	99.0	1.1
9	21.38	144.0	32.4
10	26.08	164.0	3.2
11	27.42	180.0	13.6
12	30.03	188.0	2.2
13	31.42	201.0	11.2
14	33.07	206.0	1.2
15	35.17	238.0	27.5
16	36.54	243.0	1.0

Table 3 Simulation time of test scene *Hanoi331.xml*

Action of Picture Number	Real-time Simulation Timeline (Seconds)	Actual Timeline (Seconds)	Waiting Time (Seconds)
1	0.43	1.0	---
2	3.74	8.0	0.2
3	6.70	38.0	22.1
4	8.27	42.0	1.4
5	9.68	49.0	4.2
6	13.70	68.0	7.4
7	17.75	96.0	20.1
8	18.60	105.0	5.9
9	20.39	122.0	13.7
10	21.81	129.0	5.3
11	24.70	165.0	30.4
12	27.69	185.0	9.1
13	30.85	206.0	13.2
14	34.16	235.0	18.6
15	37.57	265.0	21.8
16	39.18	273.0	1.0

Table 4 Simulation time of test scene *Hanoi331obs.xml*

The real-time simulation is performed under physics engine ODE. In Table 1, 2 3 and 4, several terms need to be explained. Action of picture number is a corresponding picture number in each figure of OpenRAVE simulation experiment. It represents a moment of the robot state which is shown in each individual picture. When the robot reaches a moment as shown in previous figures, a time is recorded into the table. Real-time simulation timeline is a time estimated by the simulator what

the robot should behave in the real world. The time step is set to be 0.01 sec, so the simulator will update the robot state one hundred times in a second of simulation environment. As the simulator solving the inverse kinematics and planning the trajectory, this time clock will stop. When the motion planning is finished and the robot is moving again, the clock will start. The waiting time is the time spent when the simulator is calculating the corresponding actions. The actual timeline records the time when the simulator is running on the computer during the test.

Shorten the time step will significantly impact the time performance of simulator, but obtaining a more precise and accurate robot trajectory. The gap between real-time simulation timeline and actual timeline is mostly because of the waiting time shown in the table. According to the three tables, it can be seen that the response of the pick-up action is much faster than the drop-off. When the robot drops off the Disk2 which has the smallest hole inside, the robot response is becoming extremely slow. We assume that the simulator is busy at searching the path mostly in the waiting time. In simulation III, due to the existence of obstacles in the test scene, the response of both pick-up and drop-off is becoming much slower compared with the scene without obstacles. Pick-up action can be much easier to planning since nothing is attached on the end-effector of robot. When dropping off the disks such as Disk2, the hole inside the disk needs to pass through the column of peg without any collision, which makes the trajectory planning much more complicated.

Besides the action of dropping of the Disk2, the rest actions respond accurately and rapidly. The average waiting time in experiment I, II and III are 9.35s, 8.55s, 10.23s and 11.62s respectively. As we can see, the OpenRAVE grants a fast response robot simulator. The waiting time can be further shortened by improving the performance of computer.

V.2.2.Simulator with sensor feedback

We have demonstrated three different Tower of Hanoi tests in the robot simulation environment. In order to simulate the robot playing the Tower of Hanoi puzzle, there are some built-in functions provided in the GUI of python that need to be mentioned.

- *ikmodel = openravepy.databases.inversekinematics.InverseKinematicsModel(robot,iktype=IkParameterizationType.Transform6D)*

  This command manages compiled inverse kinematics files for robots using ikfast. The database generator uses IKFast: The Robot Kinematics Compiler to generate optimized and stable analytic inverse kinematics solvers for any robot manipulator. The manipulator's arm joints are used for obtaining the joints to solve for.

- *FindIKSolution((Manipulator)arg1, (object)param, (int)filteroptions[, (bool)ikreturn[, (bool)releasegil]])*

  This command finds a close solution to current robot's joint values.

- *Environment.SetPhysicsEngine((Environment)arg1, (PhysicsEngine)physics)*

  This command sets the physics engine of simulation environment.

- *MoveManipulator(goal=None, maxiter=None, execute=None, outputtraj=None, maxtries=None, goals=None, steplength=None, outputtrajobj=None, jitter=None, releasegil=False)*

  This command will move the arm joints of active manipulator to a given set of joint values without collision by motion planner built in OpenRAVE. The trajectory of robot is derived from manipulation planning algorithms of OpenRAVE

- *MoveToHandPosition(matrices=None, affinedofs=None, maxiter=None, maxtries=None, translation=None, rotation=None, seedik=None, constraintfreedoms=None, constraintmatrix=None, constrainterrorthresh=None, execute=None, outputtraj=None, steplength=None, goalsamples=None, ikparam=None, ikparams=None, jitter=None, minimumgoalpaths=None, outputtrajobj=None, postprocessing=None, jittergoal=None,*

constrainttaskmatrix=None, constrainttaskpose=None, goalsampleprob=None, goalmaxsamples=None, goalmaxtries=None)

This command moves the manipulator's end effector to reach a set of 6D poses without collision similar to *MoveManipulator()*.

- *taskprob.CloseFingers()*

This command closes the active manipulator fingers using the built in grasp planner. The fingers will be closing until the sensor detects a contact collision.

- *taskprob.ReleaseFingers()*

This command releases the active manipulator fingers using the built in grasp planner. It also releases the given object.

- *Robot.WaitForController(*(Robot)arg1,(float)timeout*)*

This command waits until the robot controller is done.

When the robot arm has manipulated the gripper to the target coordinates for grasping, the fingers will be closed and grab the disks. The sensors will detect whether or not there is a contact between fingers and the target disk. Once the finger has detected a contact, it will stop closing instead of keep on closing the finger. We discussed three cases in chapter II. While the fingers and the disk are stabilized, the robot will calculate a trajectory without collision for its arm and gripper along with the attached disk; then grab the disk to its destination location.

In simulation experiment I, the robot has successfully accomplished the traditional Tower of Hanoi puzzle; which proves the simulator with sensor feedback can deal with the basic tasks such as grasping and placing the object, motion planning without collision, inverse kinematics equation solving and some robot manipulations.

In simulation experiment II, in order to further test the reliability and robustness of the simulator, we choose a case of random initial position puzzle as the test scene in OpenRAVE. Although the movement order and the 3D coordinates of pick-up and

drop-off are much different from pervious simulation, the simulator still completes the puzzle without any problem. It also demonstrates the algorithms that have been developed in this thesis generate the correct action orders and fulfill the task of testing simulator.

In the last simulation, another case of random initial disks puzzle has been tested. We tested the scene without obstacles first, then increasing the complexity of accomplishing the puzzle by adding three obstacles in three directions that the robot arm could come from. The obstacles can be easily hit or fall onto the table if the simulator motion planner is not able to calculate the trajectory precisely and accurately. Besides the existence of obstacles and different initial disks position, the robot in the simulator once again finishes the puzzle without any collisions. We mentioned that the robot final configuration of each movement could be different depending on the obstacles. Since the planning algorithms pre-calculate its manipulation and validate the final goal robot configurations of both arm and gripper to be collision free, it is not possible that the final robot configurations of the same movement are identical when the obstacles block the arms in the simulation. When obstacles exist, the GRASPVALIDATOR function is able to validate the grapper in the environment since the obstacles do not block it as the fingers are releasing or closing. However, the GOALSAMPLER_GRASPS_ARM function cannot validate the robot arm if the robot configuration is identical to the one when obstacles do not exist since the arm will collide with obstacles in the scene. In this case, the robot configuration generator will generate a new configuration until a robot configuration without collisions has been found. When both robot gripper and arm have been validated, the final robot configuration is created. In fact, during the simulation, the collisions cannot be happened, since if the collision-free robot configurations or the

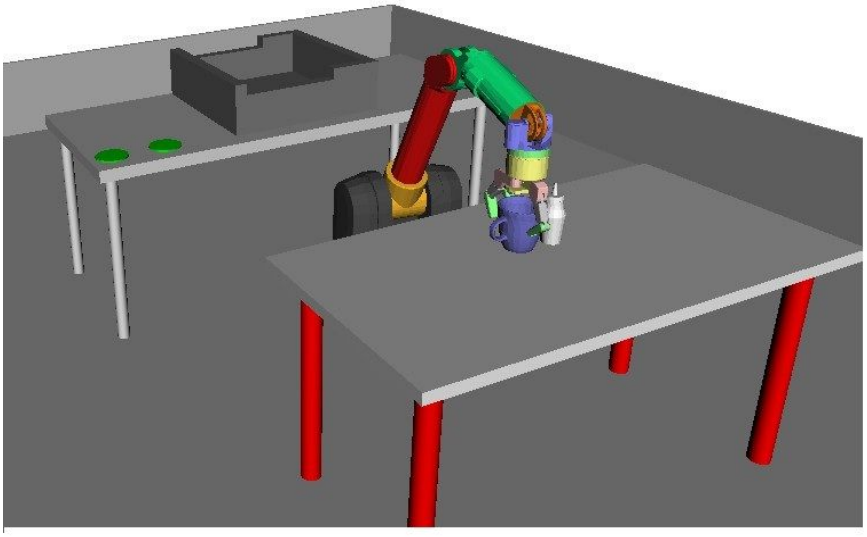
paths with no collision in pre-calculation do not exist, an error message will be returned on the python GUI before arm manipulation. The manipulation planner of OpenRAVE operates automatically when the functions involving robot manipulations have been called in python. Despite the existence of obstacles in different scenes, the functions are using exactly the same without changing the code.

The physics engine provided by OpenRAVE offers an intuitional visual view of the robot actual behavior. During three tests, the FPS (Frames per Second) of graphical simulation interface maintains around 40 to 60; which makes the whole simulation smooth and easy to watch. With all features mentioned above, the robot simulators such as OpenRAVE will be using in more boarder areas of robotics applications in a low-cost, efficient and express way.

Thanks to abundant functions provided by OpenRAVE, the coding work could be significantly reduced which leaves the program concise and easy to read. The Python which can call most functions of OpenRAVE is easy to learn. Similarly, building a 3D simulation environment in OpenRAVE is still not complicated. Normally, the code written in HTML language can complete a scene within one hundred lines. The robot database provides plentiful models for simulation. Also, the robot can be created by the user with the parameters of torque range, mass, dimensions and etc.

With successful completion of all three simulations along with the time spent and experience gained by this thesis, we can say that OpenRAVE is an easy-to-use and powerful robot simulator without dealing with the problems such as collision detection, robot kinematics, robot control and etc. The feedback sensors provide a steady and reliable performance of grasping objects in simulation environment. Also, the manipulation planner works excellent in collision free robot manipulation. During four the Tower of Hanoi puzzle tests, the feedback sensors successfully help the robot

arm grasping the disks without errors and all puzzles are accomplished at last. With a real-time simulator with sensor feedback, the robot simulation could be more close to actual robot manipulation in reality. More complex tasks are able to accomplish in a simulation environment such as OpenRAVE. For example, as shown in Figure 34, with the help of feedback sensors, the robot could easily grasping much more complicated objects such as ketchup bottle or mug in this case and put them into a washing basket.



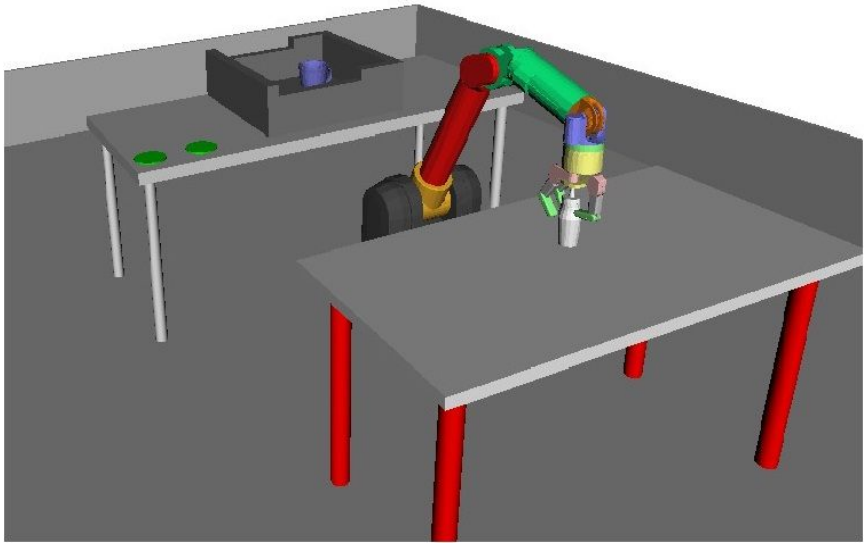


Figure 34: Robot grasping object of complicated shape in OpenRAVE

Such an example is more related to the real-world robot applications. With the help of sensor feedback, we believe the robot simulator for testing, developing and deploying motion planning such as OpenRAVE will be applied more thoroughly and comprehensively in both research and industry areas.

Chapter VI

CONCLUSION

This thesis developed and evaluated the robotic simulation of the Tower of Hanoi puzzle with sensor feedback using the robotic simulation environment called OpenRAVE. We will discuss the findings of this thesis and the related future work in this chapter.

VI.1.Discussion

We have built the robotic simulation environment in OpenRAVE and successfully accomplished the simulation of the Tower of Hanoi puzzle in different circumstances in this research. The Tower of Hanoi puzzle which is briefly introduced in Chapter I is an ideal case for testing the robot simulator with sensor feedback since grasping and dropping as well as trajectory planning works are needed to complete the puzzle. In order to let the robot play the puzzle, we developed a searching algorithm and a movement translation; therefore the robot is able to realize the correct order to complete the puzzle. The searching algorithm can solve the puzzle with different initial disk positions so that multiple simulation test scenes could be created to thoroughly and comprehensively measure the performance of the robot simulator. The code and the mechanism of robot manipulation with sensor feedback along with the software platform are also elaborated. We created four Tower of Hanoi puzzle test scenes with different initial disk positions and obstacles such as walls in one of them. The physics engine of the simulator ODE provided a real-time robot simulation with gravity and fiction force applied.

Four Tower of Hanoi puzzle tests have been successfully accomplished despite the different initial disk positions and existence of the obstacles. The response of the simulator is relatively fast. With the help of sensor feedback, the robot arm completed the actions of pick-up and drop-off without any problem. Also, the trajectory planner of simulator avoided all the obstacles, which is a challenging task since the obstacles surround three directions of the target disks. The simulation experiments and the experience gained in the research proved that OpenRAVE is a powerful and easy-to-use robot simulator with strong reliability and suitability for robotic research.

The robot simulator with sensor feedback makes it possible to test grasping objects of complicated shapes such as a ketchup bottle or a mug and improves the success rate of grasping in the simulation environment. OpenRAVE allows to create real-world robot applications and simulate it in the virtual environment with gravity, forces in real-time. I believe that such robot simulator with sensor feedback will further contribute and be broadly applied to both academic and industry applications.

VI.2.Future Work

A robot arm has been created and played the Tower of Hanoi puzzle in the simulation environment in this thesis. As mentioned in Chapter IV, *MOTOMAN HP3JC,* a 6-DOF robot arm similar to the robot we used in the simulation, should be able to perform the test in the real-world. The grasping, trajectory planning and other works we have done in the simulator can be applied to a real robot. With the help of the OpenRAVE using its physics engine, more tasks can be simulated in a virtual environment which is highly similar to the real world using its physics engine. With a sensor feedback, the simulator can execute complex actions such as grasping the irregular or complex objects with a higher successful rate. The research related to the

object grasping could be further studied using the OpenRAVE. Moreover, it would be great if the calculation result of trajectory planner in OpenRAVE can output its commands of each joint to the variety of robots and let the simulator guide it to conduct tasks in the real world. If so, we are able to conduct many robotic research works extremely efficient and simple with the assistance of OpenRAVE.

It is possible that errors exist in the real object grasping when a robot arm is approaching the target. If the error is large enough, the end-effector of the robot arm will be too far from its target coordinates and grasping tasks may fail due to the slide or target miss. The simulator with sensor feedback may simulate the situation of target miss if the actual coordinates are far from desired while an error will be returned, but the solution has not been studied in this thesis. A mechanism that can avoid this error and improve the successful rate of grasping is necessary to be established. The corresponding solutions can be tested in the simulator which is timesaving and economical then applied to the actual robot. Currently, OpenRAVE doesn’t have the ability to calculate the contact forces between the objects and robot hands. We believe that judges whether or not the contact force is large enough to grasp the object, the simulator will be extremely suitable for the robot simulation research of object grasping.

APPENDIX A: The Code for Building the 6-DOF Robot Arm

barrettsegway.robot.xml

```
<Robot>
  <KinBody>
    <!-- add a segway model to the base link -->
    <Body name="segway">
      <translation>0 0 0.742</translation>
      <Geom type="trimesh">
        <Data>models/segwayrmp/segwayrmp.iv 1.0</Data>
        <Render>models/segwayrmp/segwayrmp.iv   1.0</Render>
      </Geom>
      <mass type="custom">
        <total>40</total>
      </mass>
    </Body>
  </KinBody>
  <Robot file="barrettwam.robot.xml"></Robot>
  <KinBody>
    <body name="wam0">
      <!-- shift wam0 to align correctly with segway base -->
      <translation>0.22 0.14 0.346</translation>
      <translation>-0.15 -0.14 0.742</translation>
    </body>
    <joint name="dummy0" type="hinge" enable="false">
      <body>segway</body>
      <body>wam0</body>
      <limits>0 0</limits>
    </joint>
  </KinBody>
</Robot>
```

barrettwam.robot.xml

```
<Robot name="BarrettWAM">
```

```
  <KinBody file="wam7.kinbody.xml"/>
  <KinBody file="barretthand.kinbody.xml"/>

 <kinbody>
   <body name="handbase">
     <offsetfrom>wam7</offsetfrom>
   </body>
   <joint name="dummyhand" type="hinge" enable="false">
     <body>wam7</body>
     <body>handbase</body>
     <limits>0 0</limits>
   </joint>
 </kinbody>

  <Manipulator name="arm">
    <base>wam0</base>
    <effector>wam7</effector>
    <iksolver>WAM7ikfast 0.05</iksolver>
    <Translation>0 0 0.16</Translation>
    <joints>JF1 JF2 JF3 JF4</joints>
    <closingdirection>1 1 1 0</closingdirection>
    <direction>0 0 1</direction>
  </Manipulator>
</Robot>
```

wam7.kinbody.xml

```
<KinBody name="WAM7">
  <Body name="wam0" type="dynamic">
    <Translation>-0.22 -0.14 -0.346</Translation>
    <Geom type="trimesh">
      <Data>models/WAM/wam0.iv 1.0</Data>
      <Render>models/WAM/wam0.iv   1.0</Render>
    </Geom>
    <mass type="custom">
      <total>9.97059584</total>
      <com>0.19982328999999999 0.14 0.079952939999999972</com>
      <inertia>0.10916849   0.00640270     0.02557874   0.00640270   0.18294303   0.00161433   0.02557874   0.00161433
```

```
0.11760385</inertia>
        </mass>
      </Body>

      <Body name="wam1" type="dynamic">
        <offsetfrom>wam0</offsetfrom>
        <Translation>0.22 0.14 0.346</Translation>
        <rotationaxis>1 0 0 -90</rotationaxis>
        <Geom type="trimesh">
          <rotationaxis>1 0 0 90</rotationaxis>
          <Data>models/WAM/wam1.iv 1.0</Data>
          <Render>models/WAM/wam1.iv   1.0</Render>
        </Geom>
        <mass type="custom">
          <total>8.3936</total>
          <com> -0.00443422 0.12189039 -0.00066489</com>
          <inertia>0.13488033   -0.00213041   -0.00012485   -0.00213041   0.11328369   0.00068555   -0.00012485   0.00068555
0.09046330</inertia>
        </mass>
      </Body>

      <Joint name="Shoulder_Yaw" type="hinge">
        <Body>wam0</Body>
        <Body>wam1</Body>
        <offsetfrom>wam0</offsetfrom>
        <axis>0 0 1</axis>
        <anchor>0.22 0.14 0.346</anchor>
        <limitsdeg>-150 150</limitsdeg>
        <maxvel>1.5708</maxvel>
        <weight>1.92154</weight>
        <resolution>0.25</resolution>
        <rotorinertia>0.0 0.201 0.0</rotorinertia>
        <maxtorque>77.3</maxtorque>
      </Joint>
      <Body name="wam2" type="dynamic">
        <offsetfrom>wam1</offsetfrom>
        <rotationaxis>1 0 0 90</rotationaxis>
        <Translation>0   0   0</Translation>
```

```
      <Geom type="trimesh">
        <rotationaxis>1 0 0 -90</rotationaxis>
        <Data>models/WAM/wam2.iv   1.0</Data>
        <Render>models/WAM/wam2.iv   1.0</Render>
      </Geom>
      <mass type="custom">
        <total>3.87493756</total>
        <com> -0.00236983 0.03105614 0.01542114</com>
        <inertia>0.02140958   0.00027172   0.00002461   0.00027172   0.01377875    -0.00181920   0.00002461   -0.00181920
0.01558906</inertia>
      </mass>
    </Body>
    <Joint name="Shoulder_Pitch" type="hinge">
      <Body>wam1</Body>
      <Body>wam2</Body>
      <offsetfrom>wam1</offsetfrom>
      <axis>0 0 1</axis>
      <limitsdeg>-113 113</limitsdeg>
      <weight>0.91739941</weight>
      <maxvel>1.0472</maxvel>
      <resolution>0.5</resolution>
      <rotorinertia>0 0.182 0</rotorinertia>
      <maxtorque>160.6</maxtorque>
    </Joint>
    <Body name="wam3" type="dynamic">
      <offsetfrom>wam2</offsetfrom>
      <rotationaxis>1 0 0 -90</rotationaxis>
      <translation>0.045 0 0.55</translation>
      <Geom type="trimesh">
        <rotationaxis>1 0 0 90</rotationaxis>
        <translation>-0.045 0.55 0</translation>
        <Data>models/WAM/wam3.iv   1.0</Data>
        <Render>models/WAM/wam3.iv   1.0</Render>
      </Geom>
      <mass type="custom">
        <total>1.80228141</total>
        <com>-0.03825858 0.20750770 0.00003309</com>
        <inertia>0.05911077   -0.00249612   0.00000738   -0.00249612   0.00324550   -0.00001767   0.00000738   -0.00001767
```

```
0.05927043</inertia>
        </mass>
      </Body>
      <Joint name="Shoulder_Roll" type="hinge">
        <Body>wam2</Body>
        <Body>wam3</Body>
        <offsetfrom>wam2</offsetfrom>
        <axis>0 0 1</axis>
        <limitsdeg>-157 157</limitsdeg>
        <weight>0.882397</weight>
        <maxvel>2.0944</maxvel>
        <resolution>0.5</resolution>
        <rotorinertia>0 0.067 0</rotorinertia>
        <maxtorque>95.6</maxtorque>
      </Joint>
      <Body name="wam4" type="dynamic">
        <offsetfrom>wam3</offsetfrom>
        <rotationaxis>1 0 0 90</rotationaxis>
        <translation>-0.045 0 0</translation>
        <Geom type="trimesh">
          <rotationaxis>1 0 0 -90</rotationaxis>
          <translation>0.045 0 0</translation>
          <Data>models/WAM/wam4.iv   1.0</Data>
          <Render>models/WAM/wam4.iv   1.0</Render>
        </Geom>
        <mass type="custom">
          <total>2.40016804 </total>
          <com>0.00498512 -0.00022942 0.13271662</com>
          <inertia>0.01491672   0.00001741   -0.00150604   0.00001741   0.01482922   -0.00002109   -0.00150604   -0.00002109
0.00294463</inertia>
        </mass>
      </Body>
      <Joint name="Elbow" type="hinge">
        <Body>wam3</Body>
        <Body>wam4</Body>
        <offsetfrom>wam3</offsetfrom>
        <axis>0 0 1</axis>
        <limitsdeg>-50 180</limitsdeg>
```

```
      <weight>0.45504</weight>
      <maxvel>2.0944</maxvel>
      <resolution>1</resolution>
      <rotorinertia>0 0.034 0</rotorinertia>
      <maxtorque>29.4</maxtorque>
    </Joint>
    <Body name="wam5" type="dynamic">
      <offsetfrom>wam4</offsetfrom>
      <translation>0 0 0.3</translation>
      <rotationaxis>1 0 0 -90</rotationaxis>
      <Geom type="trimesh">
        <translation>0 0.3 0</translation>
        <rotationaxis>1 0 0 90</rotationaxis>
        <Data>models/WAM/wam5.iv  1.0</Data>
        <Render>models/WAM/wam5.iv  1.0</Render>
      </Geom>
      <mass type="custom">
        <total>0.12376019</total>
        <com>0.00008921 0.00511217 0.00435824</com>
        <inertia>0.00005029  0.00000020  -0.00000005  0.00000020  0.00007582  -0.00000359  -0.00000005  -0.00000359
0.00006270</inertia>
      </mass>
    </Body>
    <Joint name="Wrist_Yaw" type="hinge">
      <Body>wam4</Body>
      <Body>wam5</Body>
      <offsetfrom>wam4</offsetfrom>
      <axis>0 0 1</axis>
      <limitsdeg>-275 75</limitsdeg>
      <weight>0.40141</weight>
      <maxvel>4.1888</maxvel>
      <resolution>1.11</resolution>
      <rotorinertia>0 0.0033224 0</rotorinertia>
      <maxtorque>11.6</maxtorque>
    </Joint>

    <Body name="wam6" type="dynamic">
      <offsetfrom>wam5</offsetfrom>
```

<rotationaxis>1 0 0 90</rotationaxis>

<Geom type="trimesh">

<rotationaxis>1 0 0 -90</rotationaxis>

<Data>models/WAM/wam6.iv 1.0</Data>

<Render>models/WAM/wam6.iv 1.0</Render>

</Geom>

<mass type="custom">

<total>0.41797364</total>

<com>-0.00012262 -0.01703194 0.02468336</com>

<inertia>0.00055516 0.00000061 -0.00000074 0.00000061 0.00024367 -0.00004590 -0.00000074 -0.00004590 0.00045358</inertia>

</mass>

</Body>

<Joint name="Wrist_Pitch" type="hinge">

<Body>wam5</Body>

<Body>wam6</Body>

<offsetfrom>wam5</offsetfrom>

<axis>0 0 1</axis>

<limitsdeg>-90 90</limitsdeg>

<weight>0.22841245</weight>

<maxvel>4.1888</maxvel>

<resolution>1.62</resolution>

<rotorinertia>0 0.0033224 0</rotorinertia>

<maxtorque>11.6</maxtorque>

</Joint>

<Body name="wam7" type="dynamic">

<offsetfrom>wam6</offsetfrom>

<Translation>0.0 0.0 0.06</Translation>

<Geom type="trimesh">

<Translation>0.0 0.0 -0.06</Translation>

<data>models/WAM/wam7_nohand.iv 1.0</data>

<Render>models/WAM/wam7_nohand.iv 1.0</Render>

</Geom>

<mass type="custom">

<total>0.06864753</total>

<com>-0.00007974 0.00016313 -0.00323552</com>

<inertia>0.00003773 -0.00000019 0.00000000 -0.00000019 0.00003806 0.00000000 0.00000000 0.00000000

```
0.00007408</inertia>
        </mass>
      </Body>

      <Joint name="Wrist_Roll" type="hinge">
        <Body>wam6</Body>
        <Body>wam7</Body>
        <offsetfrom>wam6</offsetfrom>
        <axis>0 0 1</axis>
        <limitsdeg>-172 172</limitsdeg>
        <weight>0.20178</weight>
        <maxvel>1.0472</maxvel>
        <resolution>1.62</resolution>
        <rotorinertia>0 0 0.000466939</rotorinertia>
        <maxtorque>2.7</maxtorque>
      </Joint>

      <adjacent>wam1 wam3</adjacent>
      <adjacent>wam4 wam6</adjacent>
      <adjacent>wam4 wam7</adjacent>
    </KinBody>
```

barretthand.kinbody.xml

```
    <KinBody name="BarrettHand">
      <Body name="handbase" type="dynamic">
        <Geom type="trimesh" modifiable="false">
          <Translation>0.0   0.0   -0.06</Translation>
          <data>models/barrett/palm.iv 1</data>
          <Render>models/barrett/palm.iv 1</Render>
        </Geom>
        <Geom type="trimesh" modifiable="false">
          <Translation>0.0   0.0   0.0915</Translation>
          <RotationMat>-1 0 0 0 -1 0 0 0 1</RotationMat>
          <data>models/barrett/link1.iv 0.001</data>
          <Render>models/barrett/link1.iv 0.001</Render>
        </Geom>
        <mass type="custom">
```

```
      <!-- cylinder -->
      <total>1.1</total>
      <com>0 0 0.04796</com>
      <inertia>0.000556875 0 0 0 0.000556875 0 0 0 0.00111375</inertia>
    </mass>
  </Body>

  <!-- finger 0-->
  <Body name="Finger0-0" type="dynamic">
    <offsetfrom>handbase</offsetfrom>
    <Translation>0 -0.025 0.0915</Translation>
    <Geom type="trimesh" modifiable="false">
      <data>models/barrett/link1.iv 0.001</data>
      <Render>models/barrett/link1.iv 0.001</Render>
    </Geom>
    <mass type="custom">
      <!-- approximate as box -->
      <total>0.1</total>
      <com>0.026484 0.000000 -0.007214</com>
      <inertia>0.000025 0 0 0 0.000069 0 0 0 0.000459</inertia>
    </mass>
  </Body>
  <Body name="Finger0-1" type="dynamic" >
    <offsetfrom>Finger0-0</offsetfrom>
    <Translation>0.05 0 0</Translation>
    <RotationMat>1 0 0 0 0 -1 0 1 0</RotationMat>
    <Geom type="trimesh" modifiable="false">
      <Translation>0 0 0</Translation>
      <data>models/barrett/link2.iv 0.001</data>
      <Render>models/barrett/link2.iv 0.001</Render>
    </Geom>
    <mass type="custom">
      <!-- approximate as box -->
      <total>0.1</total>
      <com>0.033500 0.000830 0.000050</com>
      <inertia>0.000009 0 0 0 0.000074 0 0 0 0.000514</inertia>
    </mass>
  </Body>
```

```
<Joint type="hinge" name="JF1">
  <Body>Finger0-0</Body>
  <Body>Finger0-1</Body>
  <offsetfrom>Finger0-1</offsetfrom>
  <weight>0.03846</weight>
  <limitsdeg>0 140</limitsdeg>
  <axis>0 0 1</axis>
  <maxvel>2</maxvel>
  <resolution>4.2</resolution>
  <maxtorque>4.6875</maxtorque>
</Joint>
<Body name="Finger0-2" type="dynamic" >
  <offsetfrom>Finger0-1</offsetfrom>
  <Translation>0.07 0 0</Translation>
  <RotationMat>1 0 0 0 1 0 0 0 1</RotationMat>
  <Geom type="trimesh" modifiable="false">
    <Translation>0 0 0</Translation>
    <data>models/barrett/link3.iv 0.001</data>
    <Render>models/barrett/link3.iv 0.001</Render>
  </Geom>
  <mass type="custom">
    <!-- approximate as box -->
    <total>0.1</total>
    <com>0.023250 0.000000 0.000100</com>
    <inertia>0.000007 0 0 0 0.000040 0 0 0 0.000352</inertia>
  </mass>
</Body>
<Joint name="JF1mimic" type="hinge" enable="false" mimic_pos="JF1/3+0.8727" mimic_vel="|JF1
0.3333333333333333" mimic_accel="|JF1 0.3333333333333333">
  <Body>Finger0-1</Body>
  <Body>Finger0-2</Body>
  <offsetfrom>Finger0-2</offsetfrom>
  <weight>0.03846</weight>
  <limitsdeg>0 97</limitsdeg>
  <axis>0 0 1</axis>
  <maxvel>1</maxvel>
  <resolution>4.2</resolution>
</Joint>
```

```
<!-- finger 1-->
<Body name="Finger1-0" type="dynamic">
  <offsetfrom>handbase</offsetfrom>
  <Translation>0 0.025 0.0915</Translation>
  <Geom type="trimesh" modifiable="false">
    <data>models/barrett/link1.iv 0.001</data>
    <Render>models/barrett/link1.iv 0.001</Render>
  </Geom>
  <mass type="custom">
    <!-- approximate as box -->
    <total>0.1</total>
    <com>0.026484 0.000000 -0.007214</com>
    <inertia>0.000025 0 0 0 0.000069 0 0 0 0.000459</inertia>
  </mass>
</Body>
<Body name="Finger1-1" type="dynamic" >
  <offsetfrom>Finger1-0</offsetfrom>
  <Translation>0.05 0 0</Translation>
  <RotationMat>1 0 0 0 0 -1 0 1 0</RotationMat>
  <Geom type="trimesh" modifiable="false">
    <Translation>0 0 0</Translation>
    <data>models/barrett/link2.iv 0.001</data>
    <Render>models/barrett/link2.iv 0.001</Render>
  </Geom>
  <mass type="custom">
    <!-- approximate as box -->
    <total>0.1</total>
    <com>0.033500 0.000830 0.000050</com>
    <inertia>0.000009 0 0 0 0.000074 0 0 0 0.000514</inertia>
  </mass>
</Body>
<Joint type="hinge" name="JF2">
  <Body>Finger1-0</Body>
  <Body>Finger1-1</Body>
  <offsetfrom>Finger1-1</offsetfrom>
  <weight>0.03846</weight>
  <limitsdeg>0 140</limitsdeg>
```

```
    <axis>0 0 1</axis>
    <maxvel>2</maxvel>
    <resolution>4.2</resolution>
    <maxtorque>4.6875</maxtorque>
  </Joint>
  <Body name="Finger1-2" type="dynamic" >
    <offsetfrom>Finger1-1</offsetfrom>
    <Translation>0.07 0 0</Translation>
    <RotationMat>1 0 0 0 1 0 0 0 1</RotationMat>
    <Geom type="trimesh" modifiable="false">
      <Translation>0 0 0</Translation>
      <data>models/barrett/link3.iv 0.001</data>
      <Render>models/barrett/link3.iv 0.001</Render>
    </Geom>
    <mass type="custom">
      <!-- approximate as box -->
      <total>0.1</total>
      <com>0.023250 0.000000 0.000100</com>
      <inertia>0.000007 0 0 0 0.000040 0 0 0 0.000352</inertia>
    </mass>
  </Body>
  <Joint name="JF2mimic" type="hinge" enable="false" mimic_pos="JF2/3+0.8727" mimic_vel="|JF2
0.3333333333333333" mimic_accel="|JF2 0.3333333333333333">
    <Body>Finger1-1</Body>
    <Body>Finger1-2</Body>
    <offsetfrom>Finger1-2</offsetfrom>
    <weight>0.03846</weight>
    <limitsdeg>0 97</limitsdeg>
    <axis>0 0 1</axis>
    <maxvel>1</maxvel>
    <resolution>4.2</resolution>
  </Joint>


  <!-- finger 2-->
  <Body name="Finger2-1" type="dynamic" >
    <offsetfrom>handbase</offsetfrom>
    <Translation>-0.05 0 0.0915</Translation>
```

```
    <RotationMat>-1 0 0 0 0 1 0 1 0</RotationMat>
    <Geom type="trimesh" modifiable="false">
      <Translation>0 0 0</Translation>
      <data>models/barrett/link2.iv 0.001</data>
      <Render>models/barrett/link2.iv 0.001</Render>
    </Geom>
    <mass type="custom">
      <!-- approximate as box -->
      <total>0.1</total>
      <com>0.033500 0.000830 0.000050</com>
      <inertia>0.000009 0 0 0 0.000074 0 0 0 0.000514</inertia>
    </mass>
  </Body>
  <Joint type="hinge" name="JF3">
    <Body>handbase</Body>
    <Body>Finger2-1</Body>
    <offsetfrom>Finger2-1</offsetfrom>
    <weight>0.03846</weight>
    <limitsdeg>0 140</limitsdeg>
    <axis>0 0 1</axis>
    <maxvel>2</maxvel>
    <resolution>4.2</resolution>
    <maxtorque>4.6875</maxtorque>
  </Joint>
  <Body name="Finger2-2" type="dynamic" >
    <offsetfrom>Finger2-1</offsetfrom>
    <Translation>0.07 0 0</Translation>
    <RotationMat>1 0 0 0 1 0 0 0 1</RotationMat>
    <Geom type="trimesh" modifiable="false">
      <Translation>0 0 0</Translation>
      <data>models/barrett/link3.iv 0.001</data>
      <Render>models/barrett/link3.iv 0.001</Render>
    </Geom>
    <mass type="custom">
      <!-- approximate as box -->
      <total>0.1</total>
      <com>0.023250 0.000000 0.000100</com>
      <inertia>0.000007 0 0 0 0.000040 0 0 0 0.000352</inertia>
```

```
    </mass>
  </Body>
  <Joint name="JF3mimic" type="hinge" enable="false" mimic_pos="JF3/3+0.8727" mimic_vel="|JF3
0.3333333333333333" mimic_accel="|JF3 0.3333333333333333">
    <Body>Finger2-1</Body>
    <Body>Finger2-2</Body>
    <offsetfrom>Finger2-2</offsetfrom>
    <weight>0.03846</weight>
    <limitsdeg>0 97</limitsdeg>
    <axis>0 0 1</axis>
    <maxvel>1</maxvel>
    <resolution>4.2</resolution>
  </Joint>

  <!-- spread -->
  <Joint name = "JF4" type="hinge">
    <Body>handbase</Body>
    <Body>Finger0-0</Body>
    <offsetfrom>Finger0-0</offsetfrom>
    <weight>0.14894</weight>
    <limitsdeg>-1 181</limitsdeg>
    <axis>0 0 -1</axis>
    <!-- <anchor>0.025 0 0</anchor> -->
    <maxvel>2</maxvel>
    <resolution>1.7</resolution>
    <maxtorque>2</maxtorque>
  </Joint>
  <Joint name="JF4mimic" type="hinge" enable="false" mimic_pos="JF4" mimic_vel="|JF4 1" mimic_accel="|JF4 1">
    <Body>handbase</Body>
    <Body>Finger1-0</Body>
    <offsetfrom>Finger1-0</offsetfrom>
    <weight>0.14894</weight>
    <axis>0 0 1</axis>
    <!-- <anchor>-0.025 0 0</anchor> -->
    <maxvel>2</maxvel>
    <resolution>1.7</resolution>
    <limitsdeg>-1 181</limitsdeg>
  </Joint>
```

```
  <!-- add adjacent links, necessary in order to prevent bogus collisions -->
  <adjacent>Finger0-0 Finger1-0</adjacent>
  <adjacent>Finger0-1 Finger0-2</adjacent>
  <adjacent>Finger1-1 Finger1-2</adjacent>
  <adjacent>Finger2-1 Finger2-2</adjacent>
  <adjacent>handbase Finger0-1</adjacent>
  <adjacent>handbase Finger0-2</adjacent>
  <adjacent>handbase Finger1-1</adjacent>
  <adjacent>handbase Finger1-2</adjacent>
  <adjacent>handbase Finger2-1</adjacent>
  <adjacent>handbase Finger2-2</adjacent>
</KinBody>
```

APPENDIX B: The Code for Building OpenRAVE Test Scenes

Hanoi111.xml

```
<Environment>
  <camtrans>-0.717526 -3.060319 1.947468</camtrans>
  <camrotationaxis>-0.997344 0.058089 -0.043943 103.988655</camrotationaxis>
  <physicsengine type="ode">
    <odeproperties>
      <friction>0.9</friction>
      <selfcollision>0</selfcollision>
    </odeproperties>
  </physicsengine>
  <Robot file="robots/barrettsegway.robot.xml" name="BarrettWAM">
    <RotationAxis>0 0 1 180</RotationAxis>
    <translation>-0.8824     0.1310      0</translation>
  </Robot>
  <KinBody name="table">
    <RotationAxis>1 0 0 90</RotationAxis>
    <Body type="static">
      <Geom type="box">
        <extents>0.5 0.05 1</extents>
        <translation>0 0.95 0</translation>
      </Geom>
      <Geom type="box">
        <extents>0.05 0.45 0.05</extents>
        <translation>0 0.45 -0.5</translation>
      </Geom>
      <Geom type="box">
        <extents>0.05 0.45 0.05</extents>
        <translation>0 0.45 0.5</translation>
      </Geom>
    </Body>
  </KinBody>
  <KinBody name="floor">
    <RotationAxis>1 0 0 90</RotationAxis>
```

```
  <Body type="static">
    <Translation>0 -0.015 0</Translation>
    <Geom type="box">
      <extents>4 0.005 4</extents>
      <diffuseColor>.3 1 .3</diffuseColor>
     <ambientColor>0.3 1 0.3</ambientColor>
    </Geom>
  </Body>
</KinBody>

<KinBody name="srcpeg">
  <translation>-0.30162 -0.16691 1</translation>
  <Body type="dynamic">
     <Geom type="box">
       <Mass type="mimicgeom">
         <density>1000</density>
       </Mass>
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.1 0.0025 0.1</extents>
      <translation>0 0 -0.002</translation>
      <diffusecolor>1 0 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.005 0.055 0.005</extents>
      <translation>0 0 0.055</translation>
      <diffusecolor>1 0 0</diffusecolor>
    </Geom>
  </Body>
</KinBody>
<KinBody name="destpeg">
  <translation>-0.28877 0.40770 1</translation>
   <Body type="dynamic">
     <Mass type="mimicgeom">
       <density>1000</density>
     </Mass>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
```

```
        <extents>0.1 0.0025 0.1</extents>
        <translation>0 0 -0.002</translation>
        <diffusecolor>1 0 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.005 0.055 0.005</extents>
        <radius>0.01</radius>
        <height>0.2</height>
        <translation>0 0 0.055</translation>
        <diffusecolor>1 0 0</diffusecolor>
      </Geom>
    </Body>
  </KinBody>
  <KinBody name="peg">
    <translation>-0.29019 0.14937 1</translation>
     <Body type="dynamic">
       <Mass type="mimicgeom">
         <density>1000</density>
       </Mass>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.1 0.0025 0.1</extents>
        <translation>0 0 -0.002</translation>
        <diffusecolor>1 0 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.005 0.055 0.005</extents>
        <translation>0 0 0.055</translation>
        <diffusecolor>1 0 0</diffusecolor>
      </Geom>
    </Body>
  </KinBody>
  <KinBody name="disk0">
    <translation>-0.30162 -0.16691 1.021</translation>
     <Body type="dynamic">
       <Mass type="mimicgeom">
```

```
        <density>1000</density>
      </Mass>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.05 0.02 0.014</extents>
        <translation>0 0.045 0</translation>
        <diffusecolor>0 1 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.05 0.02 0.014</extents>
        <translation>0 -0.045 0</translation>
        <diffusecolor>0 1 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.014 0.02 0.05</extents>
        <translation>-0.045 0 0 </translation>
        <diffusecolor>0 1 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.014 0.02 0.05</extents>
        <translation>0.045 0 0</translation>
        <diffusecolor>0 1 0</diffusecolor>
      </Geom>
    </Body>
  </KinBody>
  <KinBody name="disk1">
    <translation>-0.30162 -0.16691 1.062</translation>
     <Body type="dynamic">
       <Mass type="mimicgeom">
         <density>1000</density>
       </Mass>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.04 0.02 0.012</extents>
        <translation>0 0.03 0</translation>
```

```
      <diffusecolor>0 0.5 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.04 0.02 0.012</extents>
      <translation>0 -0.03 0</translation>
      <diffusecolor>0 0.5 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.012 0.02 0.04</extents>
      <translation>-0.03 0 0 </translation>
      <diffusecolor>0 0.5 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.012 0.02 0.04</extents>
      <translation>0.03 0 0</translation>
      <diffusecolor>0 0.5 0</diffusecolor>
    </Geom>
  </Body>
</KinBody>
<KinBody name="disk2">
  <translation>-0.30162 -0.16691 1.103</translation>
   <Body type="dynamic">
     <Mass type="mimicgeom">
       <density>1000</density>
     </Mass>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.03 0.02 0.008</extents>
      <translation>0 0.025 0</translation>
      <diffusecolor>0 0.2 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.03 0.02 0.008</extents>
      <translation>0 -0.025 0</translation>
```

```
      <diffusecolor>0 0.2 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.008 0.02 0.03</extents>
      <translation>-0.025 0 0 </translation>
      <diffusecolor>0 0.2 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.008 0.02 0.03</extents>
      <translation>0.025 0 0</translation>
      <diffusecolor>0 0.2 0</diffusecolor>
    </Geom>
  </Body>
 </KinBody>
</Environment>
```

Hanoi212.xml

```
<Environment>
  <camtrans>-0.717526 -3.060319 1.947468</camtrans>
  <camrotationaxis>-0.997344 0.058089 -0.043943 103.988655</camrotationaxis>
  <physicsengine type="ode">
    <odeproperties>
      <friction>0.9</friction>
      <selfcollision>0</selfcollision>
    </odeproperties>
  </physicsengine>
  <Robot file="robots/barrettsegway.robot.xml" name="BarrettWAM">
    <RotationAxis>0 0 1 180</RotationAxis>
    <translation>-0.8824     0.1310      0</translation>
  </Robot>
  <KinBody name="table">
    <RotationAxis>1 0 0 90</RotationAxis>
    <Body type="static">
      <Geom type="box">
        <extents>0.5 0.05 1</extents>
```

```
      <translation>0 0.95 0</translation>
    </Geom>
    <Geom type="box">
      <extents>0.05 0.45 0.05</extents>
      <translation>0 0.45 -0.5</translation>
    </Geom>
    <Geom type="box">
      <extents>0.05 0.45 0.05</extents>
      <translation>0 0.45 0.5</translation>
    </Geom>
  </Body>
</KinBody>
<KinBody name="floor">
  <RotationAxis>1 0 0 90</RotationAxis>
   <Body type="static">
    <Translation>0 -0.015 0</Translation>
    <Geom type="box">
      <extents>4 0.005 4</extents>
      <diffuseColor>.3 1 .3</diffuseColor>
     <ambientColor>0.3 1 0.3</ambientColor>
    </Geom>
  </Body>
</KinBody>

<KinBody name="srcpeg">
  <translation>-0.30162 -0.16691 1</translation>
   <Body type="dynamic">
    <Mass type="mimicgeom">
      <density>1000</density>
    </Mass>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.1 0.0025 0.1</extents>
      <translation>0 0 -0.002</translation>
      <diffusecolor>1 0 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
```

```
        <extents>0.005 0.055 0.005</extents>
        <translation>0 0 0.055</translation>
        <diffusecolor>1 0 0</diffusecolor>
      </Geom>
    </Body>
  </KinBody>
  <KinBody name="destpeg">
    <translation>-0.28877 0.40770 1</translation>
     <Body type="dynamic">
       <Mass type="mimicgeom">
         <density>1000</density>
       </Mass>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.1 0.0025 0.1</extents>
        <translation>0 0 -0.002</translation>
        <diffusecolor>1 0 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.005 0.055 0.005</extents>
        <radius>0.01</radius>
        <height>0.2</height>
        <translation>0 0 0.055</translation>
        <diffusecolor>1 0 0</diffusecolor>
      </Geom>
    </Body>
  </KinBody>
  <KinBody name="peg">
    <translation>-0.29019 0.14937 1</translation>
     <Body type="dynamic">
       <Mass type="mimicgeom">
         <density>1000</density>
       </Mass>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.1 0.0025 0.1</extents>
        <translation>0 0 -0.002</translation>
```

```
      <diffusecolor>1 0 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.005 0.055 0.005</extents>
      <translation>0 0 0.055</translation>
      <diffusecolor>1 0 0</diffusecolor>
    </Geom>
  </Body>
</KinBody>
<KinBody name="disk0">
  <translation>-0.29019 0.14937 1.021</translation>
   <Body type="dynamic">
     <Mass type="mimicgeom">
       <density>1000</density>
     </Mass>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.05 0.02 0.014</extents>
      <translation>0 0.045 0</translation>
      <diffusecolor>0 1 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.05 0.02 0.014</extents>
      <translation>0 -0.045 0</translation>
      <diffusecolor>0 1 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.014 0.02 0.05</extents>
      <translation>-0.045 0 0 </translation>
      <diffusecolor>0 1 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.014 0.02 0.05</extents>
      <translation>0.045 0 0</translation>
```

```
        <diffusecolor>0 1 0</diffusecolor>
      </Geom>
    </Body>
</KinBody>
<KinBody name="disk1">
  <translation>-0.30162 -0.16691 1.021</translation>
    <Body type="dynamic">
      <Mass type="mimicgeom">
        <density>1000</density>
      </Mass>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.04 0.02 0.012</extents>
        <translation>0 0.03 0</translation>
        <diffusecolor>0 0.5 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.04 0.02 0.012</extents>
        <translation>0 -0.03 0</translation>
        <diffusecolor>0 0.5 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.012 0.02 0.04</extents>
        <translation>-0.03 0 0 </translation>
        <diffusecolor>0 0.5 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.012 0.02 0.04</extents>
        <translation>0.03 0 0</translation>
        <diffusecolor>0 0.5 0</diffusecolor>
      </Geom>
    </Body>
</KinBody>
<KinBody name="disk2">
  <translation>-0.29019 0.14937 1.062</translation>
```

```
    <Body type="dynamic">
      <Mass type="mimicgeom">
        <density>1000</density>
      </Mass>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.03 0.02 0.008</extents>
        <translation>0 0.025 0</translation>
        <diffusecolor>0 0.2 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.03 0.02 0.008</extents>
        <translation>0 -0.025 0</translation>
        <diffusecolor>0 0.2 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.008 0.02 0.03</extents>
        <translation>-0.025 0 0 </translation>
        <diffusecolor>0 0.2 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.008 0.02 0.03</extents>
        <translation>0.025 0 0</translation>
        <diffusecolor>0 0.2 0</diffusecolor>
      </Geom>
    </Body>
  </KinBody>
</Environment>
```

Hanoi331.xml

```
<Environment>
  <camtrans>-0.717526 -3.060319 1.947468</camtrans>
  <camrotationaxis>-0.997344 0.058089 -0.043943 103.988655</camrotationaxis>
```

```
<physicsengine type="ode">
  <odeproperties>
    <friction>0.9</friction>
    <selfcollision>0</selfcollision>
  </odeproperties>
</physicsengine>

<Robot file="robots/barrettsegway.robot.xml" name="BarrettWAM">
  <RotationAxis>0 0 1 180</RotationAxis>
  <translation>-0.8824     0.1310      0</translation>
</Robot>
<KinBody name="table">
  <RotationAxis>1 0 0 90</RotationAxis>
  <Body type="static">
    <Geom type="box">
      <extents>0.5 0.05 1</extents>
      <translation>0 0.95 0</translation>
    </Geom>
    <Geom type="box">
      <extents>0.05 0.45 0.05</extents>
      <translation>0 0.45 -0.5</translation>
    </Geom>
    <Geom type="box">
      <extents>0.05 0.45 0.05</extents>
      <translation>0 0.45 0.5</translation>
    </Geom>
  </Body>
</KinBody>
<KinBody name="floor">
  <RotationAxis>1 0 0 90</RotationAxis>
  <Body type="static">
    <Translation>0 -0.015 0</Translation>
    <Geom type="box">
      <extents>4 0.005 4</extents>
      <diffuseColor>.3 1 .3</diffuseColor>
     <ambientColor>0.3 1 0.3</ambientColor>
    </Geom>
  </Body>
```

```
</KinBody>


<KinBody name="srcpeg">
  <translation>-0.30162 -0.16691 1</translation>
  <Body type="dynamic">
    <Mass type="mimicgeom">
      <density>1000</density>
    </Mass>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.1 0.0025 0.1</extents>
      <translation>0 0 -0.002</translation>
      <diffusecolor>1 0 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.005 0.055 0.005</extents>
      <translation>0 0 0.055</translation>
      <diffusecolor>1 0 0</diffusecolor>
    </Geom>
  </Body>
</KinBody>
<KinBody name="destpeg">
  <translation>-0.28877 0.40770 1</translation>
  <Body type="dynamic">
    <Mass type="mimicgeom">
      <density>1000</density>
    </Mass>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.1 0.0025 0.1</extents>
      <translation>0 0 -0.002</translation>
      <diffusecolor>1 0 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.005 0.055 0.005</extents>
      <radius>0.01</radius>
```

```
        <height>0.2</height>
        <translation>0 0 0.055</translation>
        <diffusecolor>1 0 0</diffusecolor>
      </Geom>
    </Body>
</KinBody>
<KinBody name="peg">
    <translation>-0.29019 0.14937 1</translation>
     <Body type="dynamic">
       <Mass type="mimicgeom">
         <density>1000</density>
       </Mass>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.1 0.0025 0.1</extents>
        <translation>0 0 -0.002</translation>
        <diffusecolor>1 0 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.005 0.055 0.005</extents>
        <translation>0 0 0.055</translation>
        <diffusecolor>1 0 0</diffusecolor>
      </Geom>
    </Body>
</KinBody>
<KinBody name="disk0">
    <translation>-0.30162 -0.16691 1.021</translation>
     <Body type="dynamic">
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.05 0.02 0.014</extents>
        <translation>0 0.045 0</translation>
        <diffusecolor>0 1 0</diffusecolor>
      </Geom>
      <Mass type="mimicgeom">
        <density>1000</density>
      </Mass>
```

```
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.05 0.02 0.014</extents>
        <translation>0 -0.045 0</translation>
        <diffusecolor>0 1 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.014 0.02 0.05</extents>
        <translation>-0.045 0 0 </translation>
        <diffusecolor>0 1 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.014 0.02 0.05</extents>
        <translation>0.045 0 0</translation>
        <diffusecolor>0 1 0</diffusecolor>
      </Geom>
    </Body>
  </KinBody>
  <KinBody name="disk1">
    <translation>-0.28877 0.40770 1.021</translation>
     <Body type="dynamic">
      <Mass type="mimicgeom">
        <density>1000</density>
      </Mass>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.04 0.02 0.012</extents>
        <translation>0 0.03 0</translation>
        <diffusecolor>0 0.5 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.04 0.02 0.012</extents>
        <translation>0 -0.03 0</translation>
        <diffusecolor>0 0.5 0</diffusecolor>
      </Geom>
```

```
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.012 0.02 0.04</extents>
        <translation>-0.03 0 0 </translation>
        <diffusecolor>0 0.5 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.012 0.02 0.04</extents>
        <translation>0.03 0 0</translation>
        <diffusecolor>0 0.5 0</diffusecolor>
      </Geom>
    </Body>
  </KinBody>
  <KinBody name="disk2">
    <translation>-0.28877 0.40770 1.062</translation>
     <Body type="dynamic">
       <Mass type="mimicgeom">
         <density>1000</density>
       </Mass>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.03 0.02 0.008</extents>
        <translation>0 0.025 0</translation>
        <diffusecolor>0 0.2 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.03 0.02 0.008</extents>
        <translation>0 -0.025 0</translation>
        <diffusecolor>0 0.2 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.008 0.02 0.03</extents>
        <translation>-0.025 0 0 </translation>
        <diffusecolor>0 0.2 0</diffusecolor>
      </Geom>
```

```
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.008 0.02 0.03</extents>
        <translation>0.025 0 0</translation>
        <diffusecolor>0 0.2 0</diffusecolor>
      </Geom>
    </Body>
  </KinBody>
</Environment>
```

Hanoi331obs.xml

```
<Environment>
  <camtrans>-0.717526 -3.060319 1.947468</camtrans>
  <camrotationaxis>-0.997344 0.058089 -0.043943 103.988655</camrotationaxis>

  <physicsengine type="ode">
    <odeproperties>
      <friction>0.9</friction>
      <selfcollision>0</selfcollision>
    </odeproperties>
  </physicsengine>

  <Robot file="robots/barrettsegway.robot.xml" name="BarrettWAM">
    <RotationAxis>0 0 1 180</RotationAxis>
    <translation>-0.8824     0.1310      0</translation>
  </Robot>
  <KinBody name="table">
    <RotationAxis>1 0 0 90</RotationAxis>
    <Body type="static">
      <Geom type="box">
        <extents>0.5 0.05 1</extents>
        <translation>0 0.95 0</translation>
      </Geom>
      <Geom type="box">
        <extents>0.05 0.45 0.05</extents>
        <translation>0 0.45 -0.5</translation>
```

```
    </Geom>
    <Geom type="box">
      <extents>0.05 0.45 0.05</extents>
      <translation>0 0.45 0.5</translation>
    </Geom>
  </Body>
</KinBody>
<KinBody name="floor">
  <RotationAxis>1 0 0 90</RotationAxis>
  <Body type="static">
    <Translation>0 -0.015 0</Translation>
    <Geom type="box">
      <extents>4 0.005 4</extents>
      <diffuseColor>.3 1 .3</diffuseColor>
     <ambientColor>0.3 1 0.3</ambientColor>
    </Geom>
  </Body>
</KinBody>

<KinBody name="srcpeg">
  <translation>-0.30162 -0.16691 1</translation>
  <Body type="dynamic">
    <Mass type="mimicgeom">
      <density>1000</density>
    </Mass>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.1 0.0025 0.1</extents>
      <translation>0 0 -0.002</translation>
      <diffusecolor>1 0 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.005 0.055 0.005</extents>
      <translation>0 0 0.055</translation>
      <diffusecolor>1 0 0</diffusecolor>
    </Geom>
  </Body>
```

```
</KinBody>
<KinBody name="destpeg">
  <translation>-0.28877 0.40770 1</translation>
  <Body type="dynamic">
    <Mass type="mimicgeom">
      <density>1000</density>
    </Mass>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.1 0.0025 0.1</extents>
      <translation>0 0 -0.002</translation>
      <diffusecolor>1 0 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.005 0.055 0.005</extents>
      <radius>0.01</radius>
      <height>0.2</height>
      <translation>0 0 0.055</translation>
      <diffusecolor>1 0 0</diffusecolor>
    </Geom>
  </Body>
</KinBody>
<KinBody name="peg">
  <translation>-0.29019 0.14937 1</translation>
   <Body type="dynamic">
     <Mass type="mimicgeom">
       <density>1000</density>
     </Mass>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.1 0.0025 0.1</extents>
      <translation>0 0 -0.002</translation>
      <diffusecolor>1 0 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.005 0.055 0.005</extents>
```

```
        <translation>0 0 0.055</translation>
        <diffusecolor>1 0 0</diffusecolor>
      </Geom>
    </Body>
</KinBody>
<KinBody name="disk0">
   <translation>-0.30162 -0.16691 1.021</translation>
     <Body type="dynamic">
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.05 0.02 0.014</extents>
        <translation>0 0.045 0</translation>
        <diffusecolor>0 1 0</diffusecolor>
      </Geom>
      <Mass type="mimicgeom">
         <density>1000</density>
      </Mass>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.05 0.02 0.014</extents>
        <translation>0 -0.045 0</translation>
        <diffusecolor>0 1 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.014 0.02 0.05</extents>
        <translation>-0.045 0 0 </translation>
        <diffusecolor>0 1 0</diffusecolor>
      </Geom>
      <Geom type="box">
        <RotationAxis>1 0 0 90</RotationAxis>
        <extents>0.014 0.02 0.05</extents>
        <translation>0.045 0 0</translation>
        <diffusecolor>0 1 0</diffusecolor>
      </Geom>
    </Body>
</KinBody>
<KinBody name="disk1">
```

```
  <translation>-0.28877 0.40770 1.021</translation>
   <Body type="dynamic">
    <Mass type="mimicgeom">
      <density>1000</density>
    </Mass>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.04 0.02 0.012</extents>
      <translation>0 0.03 0</translation>
      <diffusecolor>0 0.5 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.04 0.02 0.012</extents>
      <translation>0 -0.03 0</translation>
      <diffusecolor>0 0.5 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.012 0.02 0.04</extents>
      <translation>-0.03 0 0 </translation>
      <diffusecolor>0 0.5 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.012 0.02 0.04</extents>
      <translation>0.03 0 0</translation>
      <diffusecolor>0 0.5 0</diffusecolor>
    </Geom>
  </Body>
</KinBody>
<KinBody name="disk2">
  <translation>-0.28877 0.40770 1.062</translation>
   <Body type="dynamic">
     <Mass type="mimicgeom">
       <density>1000</density>
     </Mass>
    <Geom type="box">
```

```
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.03 0.02 0.008</extents>
      <translation>0 0.025 0</translation>
      <diffusecolor>0 0.2 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.03 0.02 0.008</extents>
      <translation>0 -0.025 0</translation>
      <diffusecolor>0 0.2 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.008 0.02 0.03</extents>
      <translation>-0.025 0 0 </translation>
      <diffusecolor>0 0.2 0</diffusecolor>
    </Geom>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.008 0.02 0.03</extents>
      <translation>0.025 0 0</translation>
      <diffusecolor>0 0.2 0</diffusecolor>
    </Geom>
  </Body>
</KinBody>
<KinBody name="Object1">
  <translation>-0.44476 0.15526 1.15404</translation>
  <Body type="dynamic">
    <Mass type="mimicgeom">
      <density>1000</density>
    </Mass>
    <Geom type="box">
      <RotationAxis>1 0 0 90</RotationAxis>
      <extents>0.03 0.15 0.30</extents>
      <translation>0 0 -0.002</translation>
      <diffusecolor>0 1 1</diffusecolor>
    </Geom>
  </Body>
```

```
  </KinBody>
  <KinBody name="Object2">
    <translation>-0.2798 -0.44027 1.10260</translation>
     <Body type="dynamic">
      <Mass type="mimicgeom">
        <density>1000</density>
      </Mass>
      <Geom type="box">
        <RotationAxis>1 0 45 45</RotationAxis>
        <extents>0.03 0.15 0.10</extents>
        <translation>0 0 0</translation>
        <diffusecolor>0 1 1</diffusecolor>
      </Geom>
    </Body>
  </KinBody>
  <KinBody name="Object3">
    <translation>-0.2798 0.68027 1.10260</translation>
    <Body type="dynamic">
      <Mass type="mimicgeom">
        <density>1000</density>
      </Mass>
      <Geom type="box">
        <RotationAxis>1 0 90 90</RotationAxis>
        <extents>0.03 0.15 0.10</extents>
        <translation>0 0 0</translation>
        <diffusecolor>0 1 1</diffusecolor>
      </Geom>
    </Body>
  </KinBody>
</Environment>
```

APPENDIX C: The Code for Test the Puzzle

MyHanoi111.py

```
from openravepy import *
import numpy,time

env = Environment()
env.SetViewer('qtcoin')
env.Load('envtest/Hanoi111.xml') # Load the test scene

timestep=0.01 #Set real-time simulation time step

robot=env.GetRobots()[0]
manip = robot.GetActiveManipulator()
ikmodel = databases.inversekinematics.InverseKinematicsModel(robot,iktype=IkParameterization.Type.Transform6D)
if not ikmodel.load():
    ikmodel.autogenerate()
manipprob = interfaces.BaseManipulation(robot)
taskprob = interfaces.TaskManipulation(robot)


with env:
    raw_input("It's time to play !!")

with env:
    # set a physics engine
    physics = RaveCreatePhysicsEngine(env,'ode') #use ODE engine
    env.SetPhysicsEngine(physics)
    physics.SetGravity(numpy.array((0,0,-9.8))) #set gravity

    robot = env.GetRobots()[0]
    robot.GetLinks()[0].SetStatic(True) #Lock the wheel
    env.StopSimulation()
    env.StartSimulation(timestep) #Start real-time simulation
```

```
    manipprob.MoveManipulator(numpy.zeros(len(manip.GetArmIndices())),jitter=0.04) #Reset robot
    robot.WaitForController(0)

    State = []
    StateRev = []
    StateTree= [[] for i in range(20)]
    Movement = 0
    MovementTree = []
    seq=[]

    StateTree[0].append(111) # Initial state of puzzle
    level=0
    unit=0
    Search=1
    SearchTarget=333 # Searching target state of puzzle

    while(Search and level<20) : #Searching algorithm
        while ((Search and unit<len(StateTree[level])) and level<20) :
            TempNewState=[]
            TempNewStateRevised=[]
            NewState=0
            Dzeropos=int(StateTree[level][unit])%10
            Donepos=int((StateTree[level][unit]/10))%10
            Dtwopos=int(StateTree[level][unit]/100)
            Prev=float(unit)/10
            prev=float ("%.1f" % Prev)
            NewState=(Dtwopos+1)*100+Donepos*10+Dzeropos+prev
            TempNewState.append(NewState)
            NewState=(Dtwopos-1)*100+Donepos*10+Dzeropos+prev
            TempNewState.append(NewState)
            if Donepos+1!= Dtwopos and Donepos+1!= Dtwopos+3 and Donepos!=Dtwopos:
                NewState=Dtwopos*100+(Donepos+1)*10+Dzeropos+prev
                TempNewState.append(NewState)
            if Donepos-1!= Dtwopos and Donepos-1!= Dtwopos-3 and Donepos!=Dtwopos:
                NewState=Dtwopos*100+(Donepos-1)*10+Dzeropos+prev
                TempNewState.append(NewState)
            if Dzeropos+1!= Dtwopos and Dzeropos+1!= Dtwopos+3 and Dzeropos+1!= Donepos and Dzeropos+1!= Donepos+3
and Donepos!=Dzeropos and Dzeropos!=Dtwopos:
```

```
                    NewState=Dtwopos*100+Donepos*10+Dzeropos+1+prev
                    TempNewState.append(NewState)
                if Dzeropos-1!= Dtwopos and Dzeropos-1!= Dtwopos-3 and Dzeropos-1!= Donepos and Dzeropos-1!= Donepos-3 and
Donepos!=Dzeropos and Dzeropos!=Dtwopos:
                    NewState=Dtwopos*100+Donepos*10+Dzeropos-1+prev
                    TempNewState.append(NewState)
                for NewState in TempNewState:
                    if NewState/100 > 3:
                        NewState=NewState-300
                    if (NewState/10)%10 > 3:
                        NewState=NewState-30
                    if NewState%10 > 3:
                        NewState=NewState-3
                    if NewState/100 < 1:
                        NewState=NewState+300
                    if (NewState/10)%10 < 1:
                        NewState=NewState+30
                    if NewState%10 < 1:
                        NewState=NewState+3
                    TempNewStateRevised.append(NewState)
                Nextlevel=level+1
                for NewStateRevised in TempNewStateRevised:
                    i=0
                    j=0
                    k=0
                    for i in range(0,Nextlevel):
                        for j in range(0,len(StateTree[i])):
                            if   int (NewStateRevised) ==   int(StateTree[i][j]) :
                                k=k+1
                    if k==0:
                        StateTree[Nextlevel].append(NewStateRevised)
                    if   int(NewStateRevised) == SearchTarget :
                        Search=0
                unit=unit+1
            level=level+1
            unit=0


        for x in range(0,level+1): #Print State tree
```

```
        print '\n'
        print 'Level',x,
        for y in range(0,len(StateTree[x])):
            z = StateTree[x][y]
            print "%.1f " % z,



for x in range(level,-1,-1): #Generate State sequence
        for y in range(0,len(StateTree[x])):
            if int(StateTree[x][y])==SearchTarget:
                Z=int(round(10*(StateTree[x][y]%1)))
                StateRev.append(int(StateTree[x][y]))
            if x!=level:
                if y==Z:
                    Z=int(round(10*(StateTree[x][y]%1)))
                    StateRev.append(int(StateTree[x][y]))
                    break



for x in range(level,-1,-1):
        State.append(StateRev[x])


print '\n\nState sequence',State


for x in range(0,len(State)-1): #Movement Translation
        ANum=0
        BNum=0
        CNum=0
        Dzeropos=State[x]%10
        Donepos=(State[x]/10)%10
        Dtwopos=State[x]/100
        if Dzeropos==1:
            ANum=ANum+1;
        elif Dzeropos==2:
            BNum=BNum+1;
        else:
            CNum=CNum+1;
        if Donepos==1:
```

```
        ANum=ANum+1;
    elif Donepos==2:
        BNum=BNum+1;
    else:
        CNum=CNum+1;
    if Dtwopos==1:
        ANum=ANum+1;
    elif Dtwopos==2:
        BNum=BNum+1;
    else:
        CNum=CNum+1;
    Target=State[x+1]-State[x]
    if abs(Target)==100 or abs(Target)==200:
        disc=2
        src=Dtwopos-1
        dst=src+Target/100
        if dst<0:
            dst=dst+3
    elif abs(Target)==10 or abs(Target)==20:
        disc=1
        src=Donepos-1
        dst=src+Target/10
        if dst<0:
            dst=dst+3
    elif abs(Target)==1 or abs(Target)==2:
        disc=0
        src=Dzeropos-1
        dst=src+Target
        if dst<0:
            dst=dst+3
    if disc == 1:
        D = 100
    elif disc == 2:
        D = 200
    else :
        D = 0
    if src == 0:
        F = 100-ANum*10
```

```
    elif src == 1:
        F = 70-BNum*10
    else :
        F = 40-CNum*10
    if dst == 0:
        T = 9-ANum
    elif dst == 1:
        T = 6-BNum
    else :
        T = 3-CNum
    seq.append(D+F+T)

print '\nMovement sequence',seq,'\n'


seq_length = len(seq)
i=0

while (i<seq_length) :
    PosFrom = (seq[i]/10)%10
    PosTo = seq[i]%10
    DeskNum = seq[i]/100
    print "Step",i+1,": Disk",DeskNum,"     ",
    if PosFrom<=3:
        print 'C',
    elif PosFrom<=6:
        print 'B',
    else :
        print 'A',
    print "--->",
    if PosTo<=3 :
        print   'C',
    elif PosTo<=6 :
        print   'B',
    else :
        print   'A',
    if PosFrom == 1 : #Setting object loaction
        TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.148],[0,0,0,1]])
```

```
elif PosFrom == 2 :
    TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.107],[0,0,0,1]])
elif PosFrom == 3 :
    TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.066],[0,0,0,1]])
elif PosFrom == 4 :
    TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.148],[0,0,0,1]])
elif PosFrom == 5 :
    TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.107],[0,0,0,1]])
elif PosFrom == 6 :
    TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.066],[0,0,0,1]])
elif PosFrom == 7 :
    TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.148],[0,0,0,1]])
elif PosFrom == 8 :
    TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.107],[0,0,0,1]])
elif PosFrom == 9 :
    TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.066],[0,0,0,1]])
resFrom = manipprob.MoveToHandPosition(matrices=[TgoalFrom],seedik=10)
robot.WaitForController(0)
with env:
    sol = manip.FindIKSolution(TgoalFrom, IkFilterOptions.CheckEnvCollisions)
env.SetPhysicsEngine(None)
taskprob.CloseFingers()
robot.WaitForController(0)
with env:
    if DeskNum == 0 :
        robot.Grab(env.GetKinBody('disk0'))
    elif DeskNum == 1 :
        robot.Grab(env.GetKinBody('disk1'))
    elif DeskNum == 2 :
        robot.Grab(env.GetKinBody('disk2'))
err=0.01
if PosTo == 1 : #Setting destination
    TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.148+err],[0,0,0,1]])
elif PosTo == 2 :
    TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.107+err],[0,0,0,1]])
elif PosTo == 3 :
    TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.066+err],[0,0,0,1]])
elif PosTo == 4 :
```

```
        TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.148+err],[0,0,0,1]])
    elif PosTo == 5 :
        TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.107+err],[0,0,0,1]])
    elif PosTo == 6 :
        TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.066+err],[0,0,0,1]])
    elif PosTo == 7 :
        TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.148+err],[0,0,0,1]])
    elif PosTo == 8 :
        TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.107+err],[0,0,0,1]])
    elif PosTo == 9 :
        TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.066+err],[0,0,0,1]])
    env.SetPhysicsEngine(physics)
    resTo = manipprob.MoveToHandPosition(matrices=[TgoalTo],seedik=10)
    robot.WaitForController(0)
    env.SetPhysicsEngine(None)
    if DeskNum == 0 :
         taskprob.ReleaseFingers(target=env.GetKinBody('disk0'))
    elif DeskNum == 1 :
         taskprob.ReleaseFingers(target=env.GetKinBody('disk1'))
    elif DeskNum == 2 :
         taskprob.ReleaseFingers(target=env.GetKinBody('disk2'))
    robot.WaitForController(0)
    env.SetPhysicsEngine(physics)
    print '   done!'
    i=i+1

print '\nGame over !!!',
manipprob.MoveManipulator(numpy.zeros(len(manip.GetArmIndices())),jitter=0.04)
robot.WaitForController(0)
#env.SetPhysicsEngine(None)
```

MyHanoi212.py

```
from openravepy import *
import numpy,time

env = Environment()
env.SetViewer('qtcoin')
```

```
env.Load('envtest/Hanoi212.xml') # Load the test scene

timestep=0.01 #Set real-time simulation time step

robot=env.GetRobots()[0]
manip = robot.GetActiveManipulator()
ikmodel = databases.inversekinematics.InverseKinematicsModel(robot,iktype=IkParameterization.Type.Transform6D)
if not ikmodel.load():
    ikmodel.autogenerate()
manipprob = interfaces.BaseManipulation(robot)
taskprob = interfaces.TaskManipulation(robot)



with env:
    raw_input("It's time to play !!")

with env:
    # set a physics engine
    physics = RaveCreatePhysicsEngine(env,'ode') #use ODE engine
    env.SetPhysicsEngine(physics)
    physics.SetGravity(numpy.array((0,0,-9.8))) #set gravity

    robot = env.GetRobots()[0]
    robot.GetLinks()[0].SetStatic(True) #Lock the wheel
    env.StopSimulation()
    env.StartSimulation(timestep) #Start real-time simulation

manipprob.MoveManipulator(numpy.zeros(len(manip.GetArmIndices())),jitter=0.04) #Reset robot
robot.WaitForController(0)

State = []
StateRev = []
StateTree= [[] for i in range(20)]
Movement = 0
MovementTree = []
seq=[]

StateTree[0].append(212) # Initial state of puzzle
```

```
    level=0
    unit=0
    Search=1
    SearchTarget=333 # Searching target state of puzzle

    while(Search and level<20) : #Searching algorithm
        while ((Search and unit<len(StateTree[level])) and level<20) :
            TempNewState=[]
            TempNewStateRevised=[]
            NewState=0
            Dzeropos=int(StateTree[level][unit])%10
            Donepos=int((StateTree[level][unit]/10))%10
            Dtwopos=int(StateTree[level][unit]/100)
            Prev=float(unit)/10
            prev=float ("%.1f" % Prev)
            NewState=(Dtwopos+1)*100+Donepos*10+Dzeropos+prev
            TempNewState.append(NewState)
            NewState=(Dtwopos-1)*100+Donepos*10+Dzeropos+prev
            TempNewState.append(NewState)
            if Donepos+1!= Dtwopos and Donepos+1!= Dtwopos+3 and Donepos!=Dtwopos:
                NewState=Dtwopos*100+(Donepos+1)*10+Dzeropos+prev
                TempNewState.append(NewState)
            if Donepos-1!= Dtwopos and Donepos-1!= Dtwopos-3 and Donepos!=Dtwopos:
                NewState=Dtwopos*100+(Donepos-1)*10+Dzeropos+prev
                TempNewState.append(NewState)
            if Dzeropos+1!= Dtwopos and Dzeropos+1!= Dtwopos+3 and Dzeropos+1!= Donepos and Dzeropos+1!= Donepos+3
and Donepos!=Dzeropos and Dzeropos!=Dtwopos:
                NewState=Dtwopos*100+Donepos*10+Dzeropos+1+prev
                TempNewState.append(NewState)
            if Dzeropos-1!= Dtwopos and Dzeropos-1!= Dtwopos-3 and Dzeropos-1!= Donepos and Dzeropos-1!= Donepos-3 and
Donepos!=Dzeropos and Dzeropos!=Dtwopos:
                NewState=Dtwopos*100+Donepos*10+Dzeropos-1+prev
                TempNewState.append(NewState)
            for NewState in TempNewState:
                if NewState/100 > 3:
                    NewState=NewState-300
                if (NewState/10)%10 > 3:
                    NewState=NewState-30
```

```
            if NewState%10 > 3:
                NewState=NewState-3
            if NewState/100 < 1:
                NewState=NewState+300
            if (NewState/10)%10 < 1:
                NewState=NewState+30
            if NewState%10 < 1:
                NewState=NewState+3
            TempNewStateRevised.append(NewState)
        Nextlevel=level+1
        for NewStateRevised in TempNewStateRevised:
            i=0
            j=0
            k=0
            for i in range(0,Nextlevel):
                for j in range(0,len(StateTree[i])):
                    if   int (NewStateRevised) ==   int(StateTree[i][j]) :
                        k=k+1
            if k==0:
                StateTree[Nextlevel].append(NewStateRevised)
            if   int(NewStateRevised) == SearchTarget :
                Search=0
        unit=unit+1
    level=level+1
    unit=0

for x in range(0,level+1): #Print State tree
    print '\n'
    print 'Level',x,
    for y in range(0,len(StateTree[x])):
        z = StateTree[x][y]
        print "%.1f " % z,



for x in range(level,-1,-1): #Generate State sequence
    for y in range(0,len(StateTree[x])):
        if int(StateTree[x][y])==SearchTarget:
            Z=int(round(10*(StateTree[x][y]%1)))
```

```
                StateRev.append(int(StateTree[x][y]))
            if x!=level:
                if y==Z:
                    Z=int(round(10*(StateTree[x][y]%1)))
                    StateRev.append(int(StateTree[x][y]))
                    break


for x in range(level,-1,-1):
    State.append(StateRev[x])

print '\n\nState sequence',State

for x in range(0,len(State)-1): #Movement Translation
    ANum=0
    BNum=0
    CNum=0
    Dzeropos=State[x]%10
    Donepos=(State[x]/10)%10
    Dtwopos=State[x]/100
    if Dzeropos==1:
        ANum=ANum+1;
    elif Dzeropos==2:
        BNum=BNum+1;
    else:
        CNum=CNum+1;
    if Donepos==1:
        ANum=ANum+1;
    elif Donepos==2:
        BNum=BNum+1;
    else:
        CNum=CNum+1;
    if Dtwopos==1:
        ANum=ANum+1;
    elif Dtwopos==2:
        BNum=BNum+1;
    else:
        CNum=CNum+1;
```

```
Target=State[x+1]-State[x]
if abs(Target)==100 or abs(Target)==200:
    disc=2
    src=Dtwopos-1
    dst=src+Target/100
    if dst<0:
        dst=dst+3
elif abs(Target)==10 or abs(Target)==20:
    disc=1
    src=Donepos-1
    dst=src+Target/10
    if dst<0:
        dst=dst+3
elif abs(Target)==1 or abs(Target)==2:
    disc=0
    src=Dzeropos-1
    dst=src+Target
    if dst<0:
        dst=dst+3
if disc == 1:
    D = 100
elif disc == 2:
    D = 200
else :
    D = 0
if src == 0:
    F = 100-ANum*10
elif src == 1:
    F = 70-BNum*10
else :
    F = 40-CNum*10
if dst == 0:
    T = 9-ANum
elif dst == 1:
    T = 6-BNum
else :
    T = 3-CNum
seq.append(D+F+T)
```

```
print '\nMovement sequence',seq,'\n'



seq_length = len(seq)
i=0


while (i<seq_length) :
    PosFrom = (seq[i]/10)%10
    PosTo = seq[i]%10
    DeskNum = seq[i]/100
    print "Step",i+1,": Disk",DeskNum,"     ",
    if PosFrom<=3:
        print 'C',
    elif PosFrom<=6:
        print 'B',
    else :
        print 'A',
    print "--->",
    if PosTo<=3 :
        print   'C',
    elif PosTo<=6 :
        print   'B',
    else :
        print   'A',
    if PosFrom == 1 : #Setting object loaction
        TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.148],[0,0,0,1]])
    elif PosFrom == 2 :
        TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.107],[0,0,0,1]])
    elif PosFrom == 3 :
        TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.066],[0,0,0,1]])
    elif PosFrom == 4 :
        TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.148],[0,0,0,1]])
    elif PosFrom == 5 :
        TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.107],[0,0,0,1]])
    elif PosFrom == 6 :
        TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.066],[0,0,0,1]])
    elif PosFrom == 7 :
```

```
    TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.148],[0,0,0,1]])
elif PosFrom == 8 :
    TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.107],[0,0,0,1]])
elif PosFrom == 9 :
    TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.066],[0,0,0,1]])
resFrom = manipprob.MoveToHandPosition(matrices=[TgoalFrom],seedik=10)
robot.WaitForController(0)
with env:
    sol = manip.FindIKSolution(TgoalFrom, IkFilterOptions.CheckEnvCollisions)
env.SetPhysicsEngine(None)
taskprob.CloseFingers()
robot.WaitForController(0)
with env:
    if DeskNum == 0 :
        robot.Grab(env.GetKinBody('disk0'))
    elif DeskNum == 1 :
        robot.Grab(env.GetKinBody('disk1'))
    elif DeskNum == 2 :
        robot.Grab(env.GetKinBody('disk2'))
err=0.01
if PosTo == 1 : #Setting destination
    TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.148+err],[0,0,0,1]])
elif PosTo == 2 :
    TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.107+err],[0,0,0,1]])
elif PosTo == 3 :
    TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.066+err],[0,0,0,1]])
elif PosTo == 4 :
    TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.148+err],[0,0,0,1]])
elif PosTo == 5 :
    TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.107+err],[0,0,0,1]])
elif PosTo == 6 :
    TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.066+err],[0,0,0,1]])
elif PosTo == 7 :
    TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.148+err],[0,0,0,1]])
elif PosTo == 8 :
    TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.107+err],[0,0,0,1]])
elif PosTo == 9 :
    TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.066+err],[0,0,0,1]])
```

```
    env.SetPhysicsEngine(physics)
    resTo = manipprob.MoveToHandPosition(matrices=[TgoalTo],seedik=10)
    robot.WaitForController(0)
    env.SetPhysicsEngine(None)
    if DeskNum == 0 :
        taskprob.ReleaseFingers(target=env.GetKinBody('disk0'))
    elif DeskNum == 1 :
        taskprob.ReleaseFingers(target=env.GetKinBody('disk1'))
    elif DeskNum == 2 :
        taskprob.ReleaseFingers(target=env.GetKinBody('disk2'))
    robot.WaitForController(0)
    env.SetPhysicsEngine(physics)
    print '    done!'
    i=i+1

print '\nGame over !!!',
manipprob.MoveManipulator(numpy.zeros(len(manip.GetArmIndices())),jitter=0.04)
robot.WaitForController(0)
#env.SetPhysicsEngine(None)
```

MyHanoi331.py

```
from openravepy import *
import numpy,time

env = Environment()
env.SetViewer('qtcoin')
env.Load('envtest/Hanoi331.xml') # Load the test scene

timestep=0.01 #Set real-time simulation time step

robot=env.GetRobots()[0]
manip = robot.GetActiveManipulator()
ikmodel = databases.inversekinematics.InverseKinematicsModel(robot,iktype=IkParameterization.Type.Transform6D)
if not ikmodel.load():
    ikmodel.autogenerate()
manipprob = interfaces.BaseManipulation(robot)
taskprob = interfaces.TaskManipulation(robot)
```

```
with env:
    raw_input("It's time to play !!")

with env:
    # set a physics engine
    physics = RaveCreatePhysicsEngine(env,'ode') #use ODE engine
    env.SetPhysicsEngine(physics)
    physics.SetGravity(numpy.array((0,0,-9.8))) #set gravity

    robot = env.GetRobots()[0]
    robot.GetLinks()[0].SetStatic(True) #Lock the wheel
    env.StopSimulation()
    env.StartSimulation(timestep) #Start real-time simulation

manipprob.MoveManipulator(numpy.zeros(len(manip.GetArmIndices())),jitter=0.04) #Reset robot
robot.WaitForController(0)

State = []
StateRev = []
StateTree= [[] for i in range(20)]
Movement = 0
MovementTree = []
seq=[]

StateTree[0].append(331) # Initial state of puzzle
level=0
unit=0
Search=1
SearchTarget=333 # Searching target state of puzzle

while(Search and level<20) : #Searching algorithm
    while ((Search and unit<len(StateTree[level])) and level<20) :
        TempNewState=[]
        TempNewStateRevised=[]
        NewState=0
        Dzeropos=int(StateTree[level][unit])%10
```

```
            Donepos=int((StateTree[level][unit]/10))%10
            Dtwopos=int(StateTree[level][unit]/100)
            Prev=float(unit)/10
            prev=float ("%.1f" % Prev)
            NewState=(Dtwopos+1)*100+Donepos*10+Dzeropos+prev
            TempNewState.append(NewState)
            NewState=(Dtwopos-1)*100+Donepos*10+Dzeropos+prev
            TempNewState.append(NewState)
            if Donepos+1!= Dtwopos and Donepos+1!= Dtwopos+3 and Donepos!=Dtwopos:
                NewState=Dtwopos*100+(Donepos+1)*10+Dzeropos+prev
                TempNewState.append(NewState)
            if Donepos-1!= Dtwopos and Donepos-1!= Dtwopos-3 and Donepos!=Dtwopos:
                NewState=Dtwopos*100+(Donepos-1)*10+Dzeropos+prev
                TempNewState.append(NewState)
            if Dzeropos+1!= Dtwopos and Dzeropos+1!= Dtwopos+3 and Dzeropos+1!= Donepos and Dzeropos+1!= Donepos+3
and Donepos!=Dzeropos and Dzeropos!=Dtwopos:
                NewState=Dtwopos*100+Donepos*10+Dzeropos+1+prev
                TempNewState.append(NewState)
            if Dzeropos-1!= Dtwopos and Dzeropos-1!= Dtwopos-3 and Dzeropos-1!= Donepos and Dzeropos-1!= Donepos-3 and
Donepos!=Dzeropos and Dzeropos!=Dtwopos:
                NewState=Dtwopos*100+Donepos*10+Dzeropos-1+prev
                TempNewState.append(NewState)
            for NewState in TempNewState:
                if NewState/100 > 3:
                    NewState=NewState-300
                if (NewState/10)%10 > 3:
                    NewState=NewState-30
                if NewState%10 > 3:
                    NewState=NewState-3
                if NewState/100 < 1:
                    NewState=NewState+300
                if (NewState/10)%10 < 1:
                    NewState=NewState+30
                if NewState%10 < 1:
                    NewState=NewState+3
                TempNewStateRevised.append(NewState)
            Nextlevel=level+1
            for NewStateRevised in TempNewStateRevised:
```

```
                i=0
                j=0
                k=0
                for i in range(0,Nextlevel):
                    for j in range(0,len(StateTree[i])):
                        if   int (NewStateRevised) ==   int(StateTree[i][j]) :
                            k=k+1
                if k==0:
                    StateTree[Nextlevel].append(NewStateRevised)
                if   int(NewStateRevised) == SearchTarget :
                    Search=0
            unit=unit+1
        level=level+1
        unit=0

for x in range(0,level+1): #Print State tree
        print '\n'
        print 'Level',x,
        for y in range(0,len(StateTree[x])):
            z = StateTree[x][y]
            print "%.1f " % z,



for x in range(level,-1,-1): #Generate State sequence
        for y in range(0,len(StateTree[x])):
            if int(StateTree[x][y])==SearchTarget:
                Z=int(round(10*(StateTree[x][y]%1)))
                StateRev.append(int(StateTree[x][y]))
            if x!=level:
                if y==Z:
                    Z=int(round(10*(StateTree[x][y]%1)))
                    StateRev.append(int(StateTree[x][y]))
                    break



for x in range(level,-1,-1):
        State.append(StateRev[x])
```

```
print '\n\nState sequence',State


for x in range(0,len(State)-1): #Movement Translation
    ANum=0
    BNum=0
    CNum=0
    Dzeropos=State[x]%10
    Donepos=(State[x]/10)%10
    Dtwopos=State[x]/100
    if Dzeropos==1:
        ANum=ANum+1;
    elif Dzeropos==2:
        BNum=BNum+1;
    else:
        CNum=CNum+1;
    if Donepos==1:
        ANum=ANum+1;
    elif Donepos==2:
        BNum=BNum+1;
    else:
        CNum=CNum+1;
    if Dtwopos==1:
        ANum=ANum+1;
    elif Dtwopos==2:
        BNum=BNum+1;
    else:
        CNum=CNum+1;
    Target=State[x+1]-State[x]
    if abs(Target)==100 or abs(Target)==200:
        disc=2
        src=Dtwopos-1
        dst=src+Target/100
        if dst<0:
            dst=dst+3
    elif abs(Target)==10 or abs(Target)==20:
        disc=1
        src=Donepos-1
        dst=src+Target/10
```

```
            if dst<0:
                dst=dst+3
        elif abs(Target)==1 or abs(Target)==2:
            disc=0
            src=Dzeropos-1
            dst=src+Target
            if dst<0:
                dst=dst+3
        if disc == 1:
            D = 100
        elif disc == 2:
            D = 200
        else :
            D = 0
        if src == 0:
            F = 100-ANum*10
        elif src == 1:
            F = 70-BNum*10
        else :
            F = 40-CNum*10
        if dst == 0:
            T = 9-ANum
        elif dst == 1:
            T = 6-BNum
        else :
            T = 3-CNum
        seq.append(D+F+T)

print '\nMovement sequence',seq,'\n'


seq_length = len(seq)
i=0

while (i<seq_length) :
    PosFrom = (seq[i]/10)%10
    PosTo = seq[i]%10
    DeskNum = seq[i]/100
```

```
print "Step",i+1,": Disk",DeskNum,"     ",
if PosFrom<=3:
    print 'C',
elif PosFrom<=6:
    print 'B',
else :
    print 'A',
print "--->",
if PosTo<=3 :
    print   'C',
elif PosTo<=6 :
    print   'B',
else :
    print   'A',
if PosFrom == 1 : #Setting object loaction
     TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.148],[0,0,0,1]])
elif PosFrom == 2 :
     TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.107],[0,0,0,1]])
elif PosFrom == 3 :
     TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.066],[0,0,0,1]])
elif PosFrom == 4 :
     TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.148],[0,0,0,1]])
elif PosFrom == 5 :
     TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.107],[0,0,0,1]])
elif PosFrom == 6 :
     TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.066],[0,0,0,1]])
elif PosFrom == 7 :
     TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.148],[0,0,0,1]])
elif PosFrom == 8 :
     TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.107],[0,0,0,1]])
elif PosFrom == 9 :
     TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.066],[0,0,0,1]])
resFrom = manipprob.MoveToHandPosition(matrices=[TgoalFrom],seedik=10)
robot.WaitForController(0)
with env:
     sol = manip.FindIKSolution(TgoalFrom, IkFilterOptions.CheckEnvCollisions)
env.SetPhysicsEngine(None)
taskprob.CloseFingers()
```

```
robot.WaitForController(0)
with env:
    if DeskNum == 0 :
        robot.Grab(env.GetKinBody('disk0'))
    elif DeskNum == 1 :
        robot.Grab(env.GetKinBody('disk1'))
    elif DeskNum == 2 :
        robot.Grab(env.GetKinBody('disk2'))
err=0.01
if PosTo == 1 : #Setting destination
    TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.148+err],[0,0,0,1]])
elif PosTo == 2 :
    TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.107+err],[0,0,0,1]])
elif PosTo == 3 :
    TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.066+err],[0,0,0,1]])
elif PosTo == 4 :
    TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.148+err],[0,0,0,1]])
elif PosTo == 5 :
    TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.107+err],[0,0,0,1]])
elif PosTo == 6 :
    TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.066+err],[0,0,0,1]])
elif PosTo == 7 :
    TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.148+err],[0,0,0,1]])
elif PosTo == 8 :
    TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.107+err],[0,0,0,1]])
elif PosTo == 9 :
    TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.066+err],[0,0,0,1]])
env.SetPhysicsEngine(physics)
resTo = manipprob.MoveToHandPosition(matrices=[TgoalTo],seedik=10)
robot.WaitForController(0)
env.SetPhysicsEngine(None)
if DeskNum == 0 :
     taskprob.ReleaseFingers(target=env.GetKinBody('disk0'))
elif DeskNum == 1 :
     taskprob.ReleaseFingers(target=env.GetKinBody('disk1'))
elif DeskNum == 2 :
     taskprob.ReleaseFingers(target=env.GetKinBody('disk2'))
robot.WaitForController(0)
```

```
    env.SetPhysicsEngine(physics)
    print '    done!'
    i=i+1

print '\nGame over !!!',
manipprob.MoveManipulator(numpy.zeros(len(manip.GetArmIndices())),jitter=0.04)
robot.WaitForController(0)
#env.SetPhysicsEngine(None)
```

MyHanoi331obs.py

```
from openravepy import *
import numpy,time

env = Environment()
env.SetViewer('qtcoin')
env.Load('envtest/Hanoi331obs.xml') # Load the test scene

timestep=0.01 #Set real-time simulation time step

robot=env.GetRobots()[0]
manip = robot.GetActiveManipulator()
ikmodel = databases.inversekinematics.InverseKinematicsModel(robot,iktype=IkParameterization.Type.Transform6D)
if not ikmodel.load():
    ikmodel.autogenerate()
manipprob = interfaces.BaseManipulation(robot)
taskprob = interfaces.TaskManipulation(robot)


with env:
    raw_input("It's time to play !!")

with env:
    # set a physics engine
    physics = RaveCreatePhysicsEngine(env,'ode') #use ODE engine
    env.SetPhysicsEngine(physics)
    physics.SetGravity(numpy.array((0,0,-9.8))) #set gravity
```

```
    robot = env.GetRobots()[0]
    robot.GetLinks()[0].SetStatic(True) #Lock the wheel
    env.StopSimulation()
    env.StartSimulation(timestep) #Start real-time simulation

manipprob.MoveManipulator(numpy.zeros(len(manip.GetArmIndices())),jitter=0.04) #Reset robot
robot.WaitForController(0)

State = []
StateRev = []
StateTree= [[] for i in range(20)]
Movement = 0
MovementTree = []
seq=[]

StateTree[0].append(331) # Initial state of puzzle
level=0
unit=0
Search=1
SearchTarget=333 # Searching target state of puzzle

while(Search and level<20) : #Searching algorithm
    while ((Search and unit<len(StateTree[level])) and level<20) :
        TempNewState=[]
        TempNewStateRevised=[]
        NewState=0
        Dzeropos=int(StateTree[level][unit])%10
        Donepos=int((StateTree[level][unit]/10))%10
        Dtwopos=int(StateTree[level][unit]/100)
        Prev=float(unit)/10
        prev=float ("%.1f" % Prev)
        NewState=(Dtwopos+1)*100+Donepos*10+Dzeropos+prev
        TempNewState.append(NewState)
        NewState=(Dtwopos-1)*100+Donepos*10+Dzeropos+prev
        TempNewState.append(NewState)
        if Donepos+1!= Dtwopos and Donepos+1!= Dtwopos+3 and Donepos!=Dtwopos:
            NewState=Dtwopos*100+(Donepos+1)*10+Dzeropos+prev
            TempNewState.append(NewState)
```

```
            if Donepos-1!= Dtwopos and Donepos-1!= Dtwopos-3 and Donepos!=Dtwopos:
                NewState=Dtwopos*100+(Donepos-1)*10+Dzeropos+prev
                TempNewState.append(NewState)
            if Dzeropos+1!= Dtwopos and Dzeropos+1!= Dtwopos+3 and Dzeropos+1!= Donepos and Dzeropos+1!= Donepos+3
and Donepos!=Dzeropos and Dzeropos!=Dtwopos:
                NewState=Dtwopos*100+Donepos*10+Dzeropos+1+prev
                TempNewState.append(NewState)
            if Dzeropos-1!= Dtwopos and Dzeropos-1!= Dtwopos-3 and Dzeropos-1!= Donepos and Dzeropos-1!= Donepos-3 and
Donepos!=Dzeropos and Dzeropos!=Dtwopos:
                NewState=Dtwopos*100+Donepos*10+Dzeropos-1+prev
                TempNewState.append(NewState)
            for NewState in TempNewState:
                if NewState/100 > 3:
                    NewState=NewState-300
                if (NewState/10)%10 > 3:
                    NewState=NewState-30
                if NewState%10 > 3:
                    NewState=NewState-3
                if NewState/100 < 1:
                    NewState=NewState+300
                if (NewState/10)%10 < 1:
                    NewState=NewState+30
                if NewState%10 < 1:
                    NewState=NewState+3
                TempNewStateRevised.append(NewState)
            Nextlevel=level+1
            for NewStateRevised in TempNewStateRevised:
                i=0
                j=0
                k=0
                for i in range(0,Nextlevel):
                    for j in range(0,len(StateTree[i])):
                        if   int (NewStateRevised) ==   int(StateTree[i][j]) :
                            k=k+1
                if k==0:
                    StateTree[Nextlevel].append(NewStateRevised)
                if   int(NewStateRevised) == SearchTarget :
                    Search=0
```

```
            unit=unit+1
        level=level+1
        unit=0

for x in range(0,level+1): #Print State tree
        print '\n'
        print 'Level',x,
        for y in range(0,len(StateTree[x])):
                z = StateTree[x][y]
                print "%.1f " % z,



for x in range(level,-1,-1): #Generate State sequence
        for y in range(0,len(StateTree[x])):
                if int(StateTree[x][y])==SearchTarget:
                        Z=int(round(10*(StateTree[x][y]%1)))
                        StateRev.append(int(StateTree[x][y]))
                if x!=level:
                        if y==Z:
                                Z=int(round(10*(StateTree[x][y]%1)))
                                StateRev.append(int(StateTree[x][y]))
                                break



for x in range(level,-1,-1):
        State.append(StateRev[x])


print '\n\nState sequence',State


for x in range(0,len(State)-1): #Movement Translation
        ANum=0
        BNum=0
        CNum=0
        Dzeropos=State[x]%10
        Donepos=(State[x]/10)%10
        Dtwopos=State[x]/100
        if Dzeropos==1:
                ANum=ANum+1;
```

```
elif Dzeropos==2:
    BNum=BNum+1;
else:
    CNum=CNum+1;
if Donepos==1:
    ANum=ANum+1;
elif Donepos==2:
    BNum=BNum+1;
else:
    CNum=CNum+1;
if Dtwopos==1:
    ANum=ANum+1;
elif Dtwopos==2:
    BNum=BNum+1;
else:
    CNum=CNum+1;
Target=State[x+1]-State[x]
if abs(Target)==100 or abs(Target)==200:
    disc=2
    src=Dtwopos-1
    dst=src+Target/100
    if dst<0:
        dst=dst+3
elif abs(Target)==10 or abs(Target)==20:
    disc=1
    src=Donepos-1
    dst=src+Target/10
    if dst<0:
        dst=dst+3
elif abs(Target)==1 or abs(Target)==2:
    disc=0
    src=Dzeropos-1
    dst=src+Target
    if dst<0:
        dst=dst+3
if disc == 1:
    D = 100
elif disc == 2:
```

```
            D = 200
        else :
            D = 0
        if src == 0:
            F = 100-ANum*10
        elif src == 1:
            F = 70-BNum*10
        else :
            F = 40-CNum*10
        if dst == 0:
            T = 9-ANum
        elif dst == 1:
            T = 6-BNum
        else :
            T = 3-CNum
        seq.append(D+F+T)

print '\nMovement sequence',seq,'\n'


seq_length = len(seq)
i=0

while (i<seq_length) :
    PosFrom = (seq[i]/10)%10
    PosTo = seq[i]%10
    DeskNum = seq[i]/100
    print "Step",i+1,": Disk",DeskNum,"     ",
    if PosFrom<=3:
        print 'C',
    elif PosFrom<=6:
        print 'B',
    else :
        print 'A',
    print "--->",
    if PosTo<=3 :
        print   'C',
    elif PosTo<=6 :
```

```
    print   'B',
else :
    print   'A',
if PosFrom == 1 : #Setting object loaction
    TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.148],[0,0,0,1]])
elif PosFrom == 2 :
    TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.107],[0,0,0,1]])
elif PosFrom == 3 :
    TgoalFrom = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.066],[0,0,0,1]])
elif PosFrom == 4 :
    TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.148],[0,0,0,1]])
elif PosFrom == 5 :
    TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.107],[0,0,0,1]])
elif PosFrom == 6 :
    TgoalFrom = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.066],[0,0,0,1]])
elif PosFrom == 7 :
    TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.148],[0,0,0,1]])
elif PosFrom == 8 :
    TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.107],[0,0,0,1]])
elif PosFrom == 9 :
    TgoalFrom = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.066],[0,0,0,1]])
resFrom = manipprob.MoveToHandPosition(matrices=[TgoalFrom],seedik=10)
robot.WaitForController(0)
with env:
    sol = manip.FindIKSolution(TgoalFrom, IkFilterOptions.CheckEnvCollisions)
env.SetPhysicsEngine(None)
taskprob.CloseFingers()
robot.WaitForController(0)
with env:
    if DeskNum == 0 :
        robot.Grab(env.GetKinBody('disk0'))
    elif DeskNum == 1 :
        robot.Grab(env.GetKinBody('disk1'))
    elif DeskNum == 2 :
        robot.Grab(env.GetKinBody('disk2'))
err=0.01
if PosTo == 1 : #Setting destination
    TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.148+err],[0,0,0,1]])
```

```
    elif PosTo == 2 :
        TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.107+err],[0,0,0,1]])
    elif PosTo == 3 :
        TgoalTo = numpy.array([[0,-1,0,-0.28877],[-1,0,0,0.40770],[0,0,-1,1.066+err],[0,0,0,1]])
    elif PosTo == 4 :
        TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.148+err],[0,0,0,1]])
    elif PosTo == 5 :
        TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.107+err],[0,0,0,1]])
    elif PosTo == 6 :
        TgoalTo = numpy.array([[0,-1,0,-0.29019],[-1,0,0,0.14937],[0,0,-1,1.066+err],[0,0,0,1]])
    elif PosTo == 7 :
        TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.148+err],[0,0,0,1]])
    elif PosTo == 8 :
        TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.107+err],[0,0,0,1]])
    elif PosTo == 9 :
        TgoalTo = numpy.array([[0,-1,0,-0.30162],[-1,0,0,-0.16691],[0,0,-1,1.066+err],[0,0,0,1]])
    env.SetPhysicsEngine(physics)
    resTo = manipprob.MoveToHandPosition(matrices=[TgoalTo],seedik=10)
    robot.WaitForController(0)
    env.SetPhysicsEngine(None)
    if DeskNum == 0 :
         taskprob.ReleaseFingers(target=env.GetKinBody('disk0'))
    elif DeskNum == 1 :
         taskprob.ReleaseFingers(target=env.GetKinBody('disk1'))
    elif DeskNum == 2 :
         taskprob.ReleaseFingers(target=env.GetKinBody('disk2'))
    robot.WaitForController(0)
    env.SetPhysicsEngine(physics)
    print '    done!'
    i=i+1

print '\nGame over !!!',
manipprob.MoveManipulator(numpy.zeros(len(manip.GetArmIndices())),jitter=0.04)
robot.WaitForController(0)
#env.SetPhysicsEngine(None)
```

APPENDIX D: An Example Code for Grasping Ketchup Bottle and Mug

Test.py

```
from openravepy import *
import numpy
env = Environment()
env.SetViewer('qtcoin')
env.Load('envtest/test2.dae')
raw_input("Press any key to Start")

recorder = RaveCreateModule(env,'viewerrecorder')
env.AddModule(recorder,'')
filename = 'baobao.mpg'
codec = 13 # mpeg4
recorder.SendCommand('Start 640 480 30 codec %d timing realtime filename %s\nviewer %s'%(codec,filename,env.GetViewer().GetName()))

robot=env.GetRobots()[0]
manip = robot.GetActiveManipulator()
manipprob = interfaces.BaseManipulation(robot)
manipprob.MoveManipulator(goal=[1.421,0.473,0,1.403,0,1.259,0])
robot.WaitForController(0)
taskprob = interfaces.TaskManipulation(robot)
taskprob.CloseFingers()
robot.WaitForController(0)
with env:
    robot.Grab(env.GetKinBody('mug1'))
    manipprob.MoveManipulator(goal=[-1.548,0.621,0,0.945,0,1.571,0])
robot.WaitForController(0)
taskprob.ReleaseFingers(target=env.GetKinBody('mug1'))
robot.WaitForController(0)
manipprob.MoveManipulator(numpy.zeros(len(manip.GetArmIndices())),jitter=0.04)
robot.WaitForController(0)
```

```
manipprob.MoveManipulator(goal=[1.6,0.473,0,1.403,0,1.259,0])
robot.WaitForController(0)
taskprob.CloseFingers()
robot.WaitForController(0)
with env:
    robot.Grab(env.GetKinBody('Target'))
    manipprob.MoveManipulator(goal=[-1.3,0.621,0,0.945,0,1.571,0])
robot.WaitForController(0)
taskprob.ReleaseFingers(target=env.GetKinBody('Target'))
robot.WaitForController(0)
manipprob.MoveManipulator(numpy.zeros(len(manip.GetArmIndices())),jitter=0.04)

recorder.SendCommand('Stop')
```

APPENDIX E. User Manual of OpenRAVE

OpenRAVE provides an environment for testing, developing, and deploying motion planning algorithms in real-world robotics applications. The main focus is on simulation and analysis of kinematic and geometric information related to motion planning. OpenRAVE's stand-alone nature allows is to be easily integrated into existing robotics systems. It provides many command line tools to work with robots and planners, and the run-time core is small enough to be used inside controllers and bigger frameworks. An important target application is industrial robotics automation.

Short examples of code will help you to understand the OpenRAVE functionally. Code can be directly executed inside the python interpreter.

Example 1: Simple Environment Loading

In this example, an environment will be loaded up. It also attaches the OpenRAVE viewer, loads a scene, and requests information about the robot.

```
   from openravepy import *
   env = Environment() # create openrave environment
   env.SetViewer('qtcoin') # attach viewer (optional)
   env.Load('data/lab1.env.xml') # load a simple scene
   robot = env.GetRobots()[0] # get the first robot

   with env: # lock the environment since robot will be used
       raveLogInfo("Robot "+robot.GetName()+" has
"+repr(robot.GetDOF())+" joints with
values:\n"+repr(robot.GetDOFValues())) # logging the robot
infomation
       robot.SetDOFValues([0.5],[0]) # set joint 0 to value 0.5
       T = robot.GetLinks()[1].GetTransform() # get the transform of
link 1
       raveLogInfo("The transformation of link 1 is:\n"+repr(T)) #
logging the robot information
```

The spaces before the code in each line should be exactly followed by the above example, since it represents the code architecture in Python (like the { and } in C language). Otherwise, the code may not work functionally.

Example 2: Rotating Body

This example rotates all bodies along world z-direction by 45 degrees.

```
from openravepy import *
import numpy
env = Environment() # create openrave environment
env.SetViewer('qtcoin') # attach viewer (optional)
env.Load('data/lab1.env.xml') # load a simple scene

Tz = matrixFromAxisAngle([0,0,numpy.pi/4]) # set the angle
   with env: # lock the environment
    for body in env.GetBodies():
       body.SetTransform(numpy.dot(Tz,body.GetTransform())) #
rotates 45 degrees
```

Example 3: Using a BiRRT Planner

This example uses a planner to get a collision free pate to configuration space goal.

```
from openravepy import *
env = Environment() # create the environment
env.SetViewer('qtcoin') # start the viewer
env.Load('data/lab1.env.xml') # load a scene
robot = env.GetRobots()[0] # get the first robot
RaveSetDebugLevel(DebugLevel.Debug) # set output level to debug

manipprob = interfaces.BaseManipulation(robot) # create the
interface for basic manipulation programs
manipprob.MoveManipulator(goal=[-0.75,1.24,-0.064,2.33,-1.16,-1.5
48,1.19]) # call motion planner with goal joint angles
```

The command

```
manipprob.MoveManipulator(goal=[-0.75,1.24,-0.064,2.33,-1.16,-1.548,1.19])
```

has 7 values in the goal matrix depending on which types of robot exists in the

scene.

Example 4: Inverse Kinematics: Transform6D

This example shows how to get all 6D IK solutions.

```
from openravepy import *
import numpy, time
env = Environment() # create the environment
env.SetViewer('qtcoin') # start the viewer
env.Load('data/pr2test1.env.xml') # load a scene
robot = env.GetRobots()[0] # get the first robot

manip = robot.SetActiveManipulator('leftarm_torso') # set the
manipulator to leftarm
ikmodel =
databases.inversekinematics.InverseKinematicsModel(robot,iktype=I
kParameterization.Type.Transform6D)
if not ikmodel.load():
    ikmodel.autogenerate()

with env: # lock environment
    Tgoal =
numpy.array([[0,-1,0,-0.21],[-1,0,0,0.04],[0,0,-1,0.92],[0,0,0,1]
])
    sol = manip.FindIKSolution(Tgoal,
IkFilterOptions.CheckEnvCollisions) # get collision-free solution
    with robot: # save robot state
        robot.SetDOFValues(sol,manip.GetArmIndices()) # set the
current solution
        Tee = manip.GetEndEffectorTransform()
        env.UpdatePublishedBodies() # allow viewer to update new robot
        time.sleep(10) # sleep 10 seconds

    raveLogInfo('Tee is: '+repr(Tee))
```

Tgoal is the target 3D coordination of robot's end-effector. In this case, the coordination is (-0.21, 0.04, 0.92). Tee is the inverse kinematics solution which has been logged in the last command.

Example 5 Plan to End Effector Position

This example uses a planner to get a collision free path to a workspace goal of the end-effector.

```
from openravepy import *
import numpy
env = Environment() # create the environment
env.SetViewer('qtcoin') # start the viewer
env.Load('data/pr2test1.env.xml') # load a scene
robot = env.GetRobots()[0] # get the first robot

robot.SetActiveManipulator('leftarm_torso') # set the manipulator to
leftarm + torso
ikmodel =
databases.inversekinematics.InverseKinematicsModel(robot,iktype=I
kParameterization.Type.Transform6D)
if not ikmodel.load():
    ikmodel.autogenerate()

manipprob = interfaces.BaseManipulation(robot) # create the
interface for basic manipulation programs
Tgoal =
numpy.array([[0,-1,0,-0.21],[-1,0,0,0.04],[0,0,-1,0.92],[0,0,0,1]
])
res = manipprob.MoveToHandPosition(matrices=[Tgoal],seedik=10) #
call motion planner with goal joint angles
robot.WaitForController(0) # wait
```

Still, the Tgoal is the target coordination of end-effector. Note, after each robot manipulation, `manipprob.MoveToHandPosition` in this case, `robot.WaitForController(0)` should be the next command waiting for the robot response.

Example 6: Grabbing Object with Planner

This example shows how to use a planner to close and open a gripper using planning.

```
from openravepy import *
import numpy
env = Environment() # create openrave environment
env.SetViewer('qtcoin') # attach viewer (optional)
env.Load('data/lab1.env.xml') # load a simple scene
```

```
robot=env.GetRobots()[0]
manip = robot.GetActiveManipulator()
ikmodel =
databases.inversekinematics.InverseKinematicsModel(robot,iktype=I
kParameterization.Type.Transform6D)
if not ikmodel.load():
    ikmodel.autogenerate()

manipprob = interfaces.BaseManipulation(robot) # create the
interface for basic manipulation programs
Tgoal =
numpy.array([[0,-1,0,-0.23],[-1,0,0,-0.1446],[0,0,-1,0.85],[0,0,0
,1]])
res = manipprob.MoveToHandPosition(matrices=[Tgoal],seedik=10) #
call motion planner with goal joint angles
robot.WaitForController(0) # wait

taskprob = interfaces.TaskManipulation(robot) # create the interface
for task manipulation programs
taskprob.CloseFingers() # close fingers until collision
robot.WaitForController(0) # wait
with env:
    robot.Grab(env.GetKinBody('mug4'))

manipprob.MoveManipulator(numpy.zeros(len(manip.GetArmIndices()))
,jitter=0.04) # move manipulator to all zeros, set jitter to 0.04 since
cup is initially colliding with table

robot.WaitForController(0) # wait
```

In this case, a mug has been grabbed and lifted. To open the finger, use command `taskprob.ReleaseFingers(target=env.GetKinBody('mug4'))`.

Example 7: Setting Physics Engine

In this example, we will show how to set up physics engine using example 3.

```
from openravepy import *
env = Environment() # create the environment
env.SetViewer('qtcoin') # start the viewer
env.Load('data/lab1.env.xml') # load a scene
robot = env.GetRobots()[0] # get the first robot
```

```
RaveSetDebugLevel(DebugLevel.Debug) # set output level to debug

with env:
    # set a physics engine
    physics = RaveCreatePhysicsEngine(env,'ode')
    env.SetPhysicsEngine(physics)
    physics.SetGravity(numpy.array((0,0,-9.8))) # set gravity and
direction

manipprob = interfaces.BaseManipulation(robot) # create the
interface for basic manipulation programs
manipprob.MoveManipulator(goal=[-0.75,1.24,-0.064,2.33,-1.16,-1.5
48,1.19]) # call motion planner with goal joint angles
env.SetPhysicsEngine(None) # stop physics engine
```

In this case, ODE has been chosen as the physics engine. The gravity set to be -9.8 along Z-axis.

Example 8: Setting Real-time Simulation

In this example, we will show how to set up real-time simulation still using example 3.

```
from openravepy import *
env = Environment() # create the environment
env.SetViewer('qtcoin') # start the viewer
env.Load('data/lab1.env.xml') # load a scene

robot = env.GetRobots()[0] # get the first robot
RaveSetDebugLevel(DebugLevel.Debug) # set output level to debug
timestep=0.001 # set simulation time step
with env:
    env.StartSimulation(timestep) # start real-time simulation
manipprob = interfaces.BaseManipulation(robot) # create the
interface for basic manipulation programs
manipprob.MoveManipulator(goal=[-0.75,1.24,-0.064,2.33,-1.16,-1.5
48,1.19]) # call motion planner with goal joint angles
env.StopSimulation() # stop simulation
```

In this case, we set time step of simulation to be 0.001s. Shorter time step will improve the response of simulator.

Example 9: Saving Viewer Image

We show how to save a 640*480 image here.

```
from openravepy import *
import scipy
import time
env = Environment() # create openrave environment
env.SetViewer('qtcoin')
env.Load('data/lab1.env.xml') # load a simple scene
time.sleep(1) # wait for viewer to initialize

env.GetViewer().SendCommand('SetFiguresInCamera 1') # also shows the
figures in the image
I = env.GetViewer().GetCameraImage(640,480,
env.GetViewer().GetCameraTransform(),[640,640,320,240])
scipy.misc.imsave('openrave.jpg',I)
```

Example 10: Recording Videos

We show how to start and stop recording videos using Python API in this example.

```
from openravepy import *
import time

env = Environment() # create openrave environment
env.SetViewer('qtcoin')
env.Load('data/lab1.env.xml') # load a simple scene

recorder = RaveCreateModule(env,'viewerrecorder')
env.AddModule(recorder,'')
codecs = recorder.SendCommand('GetCodecs') # linux only
filename = 'openrave.mpg'
codec = 13 # mpeg4
recorder.SendCommand('Start 640 480 30 codec %d timing realtime
filename %s\nviewer %s'%(codec,filename,env.GetViewer().GetName()
))
time.sleep(5)
recorder.SendCommand('Stop') # stop the video
env.Remove(recorder) # remove the recorder
```

Illustration of Testing Tower of Hanoi Puzzle Code

env.Load('envtest/hanol331.xml') command will load the test scene.

StateTree[0].append(331) command sets the initial state of puzzle.

SearchTarget=333 sets the target state to be achieved.

For example, the node *aaa* in Figure 3 represents as 111 in digital, since the rightmost digit represents the largest disk and the number indicates the location of disk. (1 for start peg, 2 for middle peg, 3 for destination peg). There is one digit for each disk. Each node represents the disk positions from left to right in order of increasing size.

After setting these three lines, the code can be executed. Then press any key to start the game.

When ‘Game over !!!’’ has been displayed on the GUI, the puzzle is finished.

REFERENCES

[1] Rosen Diankov, Automated Construction of Robotics Manipulation Programs, Robotics Institute, Carnegie Mellon University, August , 2010

[2] Lucas, ´Edouard, R´ecr´eations Math´ematiques, vol. III, Gauthier-Villars, Paris, 1893

[3] Simon Fraser University, http://www.cs.sfu.ca/~tamaras/recursion/Rules_Towers_Hanoi.html

[4] Petković, Miodrag, Famous Puzzles of Great Mathematicians, AMS Bookstore, pp. 197, 2009

[5] Troshkin, M. Doomsday Comes: A Nonrecursive Analysis of the Recursive Towers-of-Hanoi Problem, Focus, 95(2): 10-14

[6] Y.Alavi, D. R. Lick, and A. Schwenk The Average Distance between Nodes in the Cyclic Tower of Hanoi Digraph, in Combinatorics, Graph Theory, and Algorithms, Vol. II, ed., pp. 799--808. New Issues Press, Kalamazoo, 1999

[7] Wikipedia, http://en.wikipedia.org/wiki/Tower_of_hanoi

[8] Kawamura, K., Gordon, S.M., Ratanaswasd, P., Erdemir, E. and J.F. Hall,"Implementation of Cognitive Control for a Humanoid Robot." International Journal of Humanoid Robotics (IJHR), Vol 5 No 4, pp. 547-586, 2008

[9] Erdemir, E., Frankel, C. B., Kawamura, K., Gordon, S.M., Thorton, S., and B.Ulutas,"Towards a Cognitive Robot that Uses Internal Rehearsal to Learn Affordance Relations." International Conference on Intelligent RObots and Systems (IROS), Nice, France, September 22-26, 2008.

[10] Huan Tan, Erdem Erdemir, Kazuhiko Kawamura, and Qian Du, "A Potential

Field Method-based Extension of the Dynamic Movement Primitive Algorithm for Imitation Learning with Obstacle Avoidance", Proceedings of the 2011 IEEE International Conference on Mechatronics and Automation (ICMA 2011), Beijing, China, August 2011.

[11] Wikipedia, https://en.wikipedia.org/wiki/OpenRAVE

[12] OpenRAVE documentation, http://openrave.org/docs/latest_stable/openravepy/ikfast/#ikfast-the-robot-kinematics-compiler

[13] Mark Summerfield, *Rapid GUI Programming with Python and Qt*, Prentice Hall, 2008

[14] Von Steve McConnell, *Code Complete*, second edition, Microsoft Press, pp.100, 2004

[15] Wikipedia, http://en.wikipedia.org/wiki/Rapidly_exploring_random_tree

[16] Kuffner J, LaValle S, RRT-Connect: An Efficient Approach to Single-Query Path Planning, IEEE International Conference on Robotics and Automation (ICRA), 2000

[17] LaValle S, Kuffner J, Randomized kinodynamic planning. International Journal of Robotics Research, 20(5):378-400,2001

[18] Kuffner J, Nishiwaki K, Kagami S, Inaba M, Inoue H, Motion planning for humanoid robots, International Symposium on Robotics Research (ISRR), 2003

[19] Oriolo G, Vendittelli M, Freda L, Troso G, The SRT Method: Randomized strategies for exploration, IEEE International Conference on Robotics and Automation (ICRA), 2004

[20] Ferguson D, Stentz A, Delayed D*: The Proofs. Tech. Rep. CMU-RI-TR-04-51, Carnegie Mellon Robotics Institute, 2004

[21] Ferguson D, Stentz A, The Field D* Algorithm for Improved Path Planning and Replanning in Uniform and Non-uniform Cost Environments. Tech. Rep. CMU-RI-TR-05-19, Carnegie Mellon School of Computer Science, 2005

[22] Diankov R, Kuffner J, Randomized statistical path planning, IEEE International Conference on Intelligent Robots and Systems (IROS), 2007

[23] Rosen Diakov et.al. OpenRAVE custom xml format http://openrave.programmingvision.com/wiki/index.php/Format:XML,April, 2012

[24] Erik Fahlen and Josef Sunesson Robot Door Opening report http://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand12/Group1Marten/final/Erik.Fahlen.Josef.Sunesson.report.pdf

[25] J.J.Uicker, G.R.Pennock, and J.E.Shigley, *Theory of machines and mechanism,* Oxford University Press, New York, 2003

[26] Physical Engine http://en.wikipedia.org/wiki/Physics_engine

[27] Open Dynamics Engine http://ode-wiki.org/wiki/index.php?title=Manual:_Introduction

[28] Simulation Open Framework Architecture http://www.sofa-framework.org/

[29] AMD Announces Open Physics Initiative Designed to Bring New Levels of Realism to Gaming, Simulations, Popular Applications http://www.amd.com/us/press-releases/Pages/amd-announces-new-levels-of-realism-2009sept30.aspx

[30] Tokamak Open Source Physics Engine http://www.tokamakphysics.com/

[31] Lavalle, S.M, Rapidly-exploring random trees: A new tool for path planning, Computer Science Dept, Iowa State University, Tech Rep. TR: 98–11, 1998

[32] A* Search Algorithm, http://en.wikipedia.org/wiki/A*_search_algorithm