DYNAMIC GRASP ANALYSIS AND PROFILING OF GAZEBO

By

Jielei Wang

A Thesis Submitted to the Graduate Faculty of Rensselaer Polytechnic Institute in Partial Fulfillment of the Requirements for the Degree of MASTER OF SCIENCE

Major Subject: COMPUTER SCIENCE

Exchanging Committee:

Jeffrey Trinkle, Thesis Adviser

Christopher Carothers, Member

Randolph Franklin, Member

Rensselaer Polytechnic Institute
Troy, New York

November 2013
(For Graduation December 2013)
CONTENTS

LIST OF TABLES .............................................................. iv
LIST OF FIGURES ............................................................ v
ACKNOWLEDGMENTS ........................................................... v
ABSTRACT ........................................................................ vi
1. BACKGROUND ................................................................. 1
   1.1 Related Work of Grasp Analysis ................................. 3
      1.1.1 Static Grasp Analysis .................................... 3
      1.1.2 Dynamic Grasp Analysis ................................. 4
   1.2 Related Work of Profiling .......................................... 5
   1.3 Outline ................................................................. 5
2. INTRODUCTION .............................................................. 6
   2.1 Basic Concepts ....................................................... 6
      2.1.1 Grasping ....................................................... 6
      2.1.2 Static Grasp Analysis .................................... 6
      2.1.3 Dynamic Grasp Analysis ................................. 6
      2.1.4 Wrench and Wrench Space ................................ 6
      2.1.5 Force Closure ................................................. 7
      2.1.6 Joint Parameters ........................................... 8
      2.1.7 Inverse Kinematics ....................................... 8
      2.1.8 Gazebo and ODE ........................................... 9
      2.1.9 ROS .......................................................... 9
      2.1.10 OProfile ................................................... 9
      2.1.11 DRC(DARPA Robotics Challenge) ..................... 10
      2.1.12 Rigid Multibody Dynamics Simulation .............. 10
      2.1.13 LCP ......................................................... 11
   2.2 Motivation ............................................................. 12
   2.3 Summary of Current Work and Results ...................... 13
      2.3.1 Dynamic Grasp Analysis ................................ 13
      2.3.2 Profiling of Gazebo and ODE ......................... 14
## LIST OF TABLES

| 3.1 | Hardware conditions of computer for profiling of Gazebo | 21 |
| 3.2 | OProfile report for picking up hose end simulation | 22 |
| 3.3 | Timer report for boxes simulation | 25 |
| 3.4 | Timer report for Atlas simulation | 26 |
| 3.5 | Timer report for Atlas grasping standpipe simulation | 26 |
| 3.6 | Timer report for Atlas picking up hose simulation | 27 |
| 3.7 | Percentages of time consumed by LCP procedure | 27 |
LIST OF FIGURES

2.1 Sandia hand grasping a drill and its corresponding convex hull computed by GraspIt! .......................................................... 7
2.2 Workspace of Atlas left arm ......................................................... 8
2.3 Grasp hose task in DRC ............................................................ 10
2.4 Pose error caused by dynamics....................................................... 12
3.1 Grasp drill task ........................................................................ 15
3.2 Grasps generated by GraspIt! ........................................................ 16
3.3 Dynamic grasp evaluation process ............................................... 17
3.4 Pre-grasp and grasp ................................................................. 18
3.5 Sandia hand with tactile sensors ............................................... 19
3.6 Dynamic grasp analysis result ...................................................... 19
3.7 Simulation scenes ................................................................. 20
3.8 Comparison of simulation 1 and simulation 2 of profiling results .......... 23
3.9 Comparison of simulation 2 and simulation 3 of profiling results .......... 24
3.10 Comparison of simulation 3 and simulation 4 of profiling results .......... 24
3.11 Profiling of ODE result .............................................................. 28
ACKNOWLEDGMENTS

First of all, I would like to express the deepest gratitude to my advisor and chair of committee members Jeff Trinkle for taking me in computer science department in 2012 when I had never heard of grasp planning or LCP. Without his patience, the work presented in this thesis cannot be achieved.

Secondly, I would like to thank my committee members Professor Randolph Franklin and Professor Christopher Carothers for participating in my defense and giving me precious advise on my thesis.

Thirdly, I would like to thank Claude Lacoursire, Shuai Li, Jun Dong, Ying Lv, Emma Zhang, Danial Flickinger and Jed Williams. They have been helpful whenever I have come up with problems. They are also nice friends too in the spare time.

Last but not least, I would like to thank my husband Kun Huang. He is always supportive either at home or at school and patient for my stubborness and all the troubles I have made.
ABSTRACT

In this work, we consider dynamics and uncertainty of the pose of the object during grasp analysis of Atlas robot simulated in Gazebo. Considering dynamics is crucial during grasping process in dynamic environment as well as pose error of the object. The dynamic grasp analysis start with taking grasps from grasp database generated by GraspIt! which is based on static object assumption. We then evaluate the grasps using Monte Carlo simulation. The goal is to identify grasps selected by GraspIt! that are robust to object position uncertainty and accidental bump. We have shown from the result that the success rates of the grasps obtained from GraspIt! differ a lot when in dynamic environment.

Sometimes the simulations are very slow and sometimes are unstable. Thus we find it necessary to determine the bottleneck of the simulator which is another work presented in this thesis. Using a profiling tool, we are able to analyze the program and also the critical section of the code can be identified[1]. We use OProfile as the profiling tool for Gazebo. We have also added our own timing library using the standard counter: rdtsc to ODE and counted the time that each function took in miliseconds within one time step when profiling ODE.
1. BACKGROUND

Analyzing the quality of grasp of robot hand has been an important and difficult problem in robotics since its inception. Normally, grasp quality analysis is based on the ability of the contact points between the hand and the object to resist external forces applied to the object, e.g., gravity and contact forces arising during assembly. This is a static analysis based on geometric and kinematic models of the hand and object and the equations of equilibrium[2].

When real robotic systems perform tasks, the robot needs to grasp objects and manipulate them. To do this robustly, the robot should use the most secure grasps that it can achieve. Such grasps are termed “force closure” grasps. When a grasp has force closure, then the grasp can be maintained despite arbitrary (but bounded) disturbing forces and torques applied to the object. Because finding the best force closure grasps is very time consuming, potential grasps are analyzed off-line and stored in a database so that the robot can select a good grasp in real-time.

The most widely-used grasp analysis and database tools are GraspIt![3] and the Columbia Grasp Database[4]. GraspIt! seeks for grasps that are robust considering the geometric and kinematic models of the object and hand. The search proceeds at random in the very high-dimensional space that represents the configuration of the object relative to the palm and the configurations of the fingers. The dimension of this space is six plus the number of independently controlled finger joints. On average, GraspIt! requires about 15 minutes on a modern desktop computer to find about 10 force closure grasps of an object.

Multiple grasps are generated due to that in some configurations, some grasps cannot be achieved because the object to be grasped is too close to some other object that the robot should not touch. The goal of grasp planning is to store a set of grasps to a grasp database such at at least one can be executed for all possible situations in which the robot will be required to grasp that object. We need to select a large set of grasps because the quality of the grasp set being used in the planning directly affects the manipulation task[5]. Hence it could be hundreds of grasps that
could require a CPU day to generate.

This work is done in Gazebo which is a dynamic simulator. It means the robot has to accomplish grasp tasks in dynamic environment. Since in tool like GraspIt!, it assumes the objects are all static when generating the “good” grasps. This work explores obtaining “good” grasps when dynamics is considered.

Our grasping experiments are all running in simulation, hence we care about the performance of the simulator because it can largely affect the quality of grasp. That’s why we profile the Gazebo simulator and look at its bottleneck of performance. It is another part of this thesis work.

During the simulation in Gazebo, both real time and simulation time are recorded. Real time is computational time which is synchronized with the clock in the computer. Simulation time refers to the time in simulation.

For example, in the simulation world, when an item is dropped at a height \( h \) and initially it is still, it will take \( T = \sqrt{\frac{2h}{g}} \) seconds according to Newton’s law to fall to the ground in which \( g \) is the gravitational constant. \( T \) is simulation time. However probably it will take the computer \( 2T \) to compute the free body falling process. Or if the computation process is fast, it may take the computer only \( 0.5T \) to finish the whole process.

Real time factor is used to measure the difference of the real time and simulation time. Real time factor calculated in Gazebo is the ratio of simulation time and real time and it is a common metric to measure how fast the simulation runs. When it is smaller than one, it means the real computation time the simulation takes is longer than the time in simulation. If it is one or larger than one, it means the process is done in real time. Sometimes the real time factor is much smaller than one which indicates a poor performance because computation takes an excessive amount of time.

During our grasp experiments using Gazebo, the real time factor is usually around 0.4 and that is pretty low. So it means the computation time is very long and far from what we expected. That’s why it is interesting for us what code section takes significant amount of processor time.

During the grasp experiments, Gazebo sometimes terminates by itself too. It
is very much likely that in Gazebo, memory leak exists. We can also use the profiling
tool to detect what code causes memory leak. And from the results, we are able to
know how to optimize the software.

However, the computer system has become so complex which makes it difficult
to determine what code consumes the processor time more than it is expected. So
the tools we use to do profiling are very essential[6]. They should be able to evaluate
how well programs performs on new architecture, to find out how well the instruction
scheduling algorithm performs and they should also analyze programs and identify
critical sections of code[1].

1.1 Related Work of Grasp Analysis

There is a large amount of work done in the field of grasp analysis. We only
include the most relevant work here.

1.1.1 Static Grasp Analysis

In the work [7], Li and Sastry discussed optimal grasping of an object by
multifingered robot hand. They have proposed three ways of quality measurements
for evaluation of a grasp. They proposed a grasp evaluation method by obtaining
smallest singular value of grasp matrix G and the smallest volume in wrench space,
which shares similar idea with GraspIt!. They have also introduced a task-oriented
quality measurement as the evaluation of grasp. Ferrari and Canny proposed in
their work evaluating grasp using the radius of largest ball inside the convex hull of
the wrench space[8]. They described two methods of constructing a GWS. In one
method, the convex hull covers the origin of the wrench space and the largest ball
radius is the criterion of the quality of grasp. Another method of defining grasp
quality is the ratio of the magnitude of the task wrenches to the magnitude of the
contact forces.

In the first step of the dynamic grasp analysis of our work, we take grasps
generated by GraspIt! It uses force closure as the evaluation of grasp which uses the
magnitude of the ball radius of the convex hull and it is similar to the work stated
above.
1.1.2 Dynamic Grasp Analysis

Kim and Iwamoto proposed a grasp quality measures considering both dynamics and pose uncertainty\cite{5}. They have shown in their work by a real robot system that it is very effective to predict the actual grasp success rate. They simulated the robot hand grasping an object using their own physics simulation tool. And they have claimed that the trend of success rate of real robot is similar with their simulation results however the results obtained by static object model doesn’t have similar trend with their results. Hence, they concluded that both object dynamics and pose uncertainty were important. In the grasp analysis part of this thesis work, we have also considered the dynamics and pose uncertainty. But the motion of the object is not calculated in this work and our work is done in simulation only. We can consider adding that in our future work because it can predict the position of the object more precisely. We can also consider doing the simulation in real robotic system. Hsiao and Kaelbling presented in their work a decision-theoretic approach to problems that require accurate placement of a robot relative to an object of known shape\cite{9}. They claimed in their paper that “The decision process is applied to a robot hand with tactile sensors, to localize the object on a table and ultimately achieve a target placement by selecting among a parameterized set of grasping and information-gathering trajectories”. They have considered the uncertainty of the pose error of the object. Dogar and Siddhartha have presented in their work a planner for the pickup, push, slide, and sweep with robot hand and arm\cite{10}. Their algorithm considered about uncertainty of object poses from the robot perception system. In the paper \cite{11}, Kim, Gribovskaya and Billard proposed an approach to time the motion accurately while catching a moving object. In their work, they claimed they are able to control the timing of motions of the robot if it is able to synchronize its movement with a moving object. They have also considered dynamics and calculated accurately the motion of the end-effectors and the object’s position.
1.2 Related Work of Profiling

William E. Cohen presents an introduction to OProfile in his paper[6]. He first gives an overall introduction to the architecture of processor. He then introduces the architecture of OProfile and then gave several examples of how to OProfile. One example given is image processing applications that convert large images from one format to another being profiled by OProfile. In the example, performances of profiling results before optimization and after optimization are compared. He claims in the work the run time has decreased from 19.48 seconds to 16.37 seconds. Cashing is also introduced in this paper. In the work [12], profiling of ROS system is presented. Two techniques were used to do the profiling work. One is CPU usage sampling and that is similar with what we use to profile Gazebo. The other is compute kernel timing by adding timing code around the core compute kernel and that is similar with what we did to profile ODE. They have also optimized the system and did comparison of the results before and after optimization. That is what we consider to accomplish in our future work.

1.3 Outline

This thesis includes four chapters. The first chapter gives a brief background of the area of this work and introduces the related work. The further outline of this thesis is as follows:

In chapter two, an introduction of some basic concepts is included such as multibody dynamics, inverse kinematics, grasp analysis etc. And then it looks at the motivation of the thesis.

Chapter three covers the experiment details of dynamic grasp analysis work and also profiling of Gazebo and ODE. Later the results are discussed.

Finally in chapter four, future work is explored and a conclusion of grasp analysis and profiling work are made.
2. INTRODUCTION

2.1 Basic Concepts

2.1.1 Grasping

When a robot is interacting with objects in the environment, grasping is one of the primary ways in the interaction. Picking up and placing objects, opening a door are tasks that are highly relevant to grasping.

Some robots like PR2 use parallel grippers to accomplish pick and place tasks. Some robots like Atlas robot accomplish those tasks using multifinger hands, such as Sandia hand, iRobot hand, etc. The tool that is attached to the end of the robot arm is called end effector. It is a kind of gripper which consists of two, three, four or five fingers[13].

2.1.2 Static Grasp Analysis

Static grasp analysis only consider grasping objects in static environment. That is during the grasping process, the object stays at the given position and orientation and can never be knocked down or change position even when applied a large force and torque.

2.1.3 Dynamic Grasp Analysis

Dynamic grasp analysis consider the dynamics of the object while executing grasp. For example, if an object is placed on a table without sticking to the table, when the end-effector is grasping the object, it will cause change of the position or orientation or both of the object before the grasp task has been accomplished. So we have to consider the movement of the object while we do the dynamic grasp analysis.

2.1.4 Wrench and Wrench Space

For 2 dimensional object, the force applied to it is 2 dimentional and the torque caused by the force is perpendicular to the force. So together they form a 3
dimensional space. For 3 dimensional object, we know the force and torque applied to it form a 6 dimensional space. We name the force and torque together wrench and the space formed by the wrench wrench space.

2.1.5 Force Closure

A grasp is determined to have force closure if it is able to resist any external forces and torques without loosing contact with the object[8]. There is also a simple geometrical interpretation of force closure. If the wrenches exerted by the fingers on the contact points of the object is able to form a convex hull in the wrench space and it contains the origin of the wrench space, then force closure of the grasp exists. The quality of the grasp is represented as the radius of the largest ball contained in the convex hull.

![Figure 2.1: We generated these two figures by GraspIt!. The one on the left shows Sandia hand grasping a drill. And the green polygon in the picture on the right is the corresponding convex hull according to the contact information.](image)

In figure 2.1, the picture on the right shows the wrench space and the convex hull of Sandia hand grasping a drill like it is shown in the picture on the left. They are generated by GraspIt!. Since we cannot show 6 dimesional in a picture, so we can only show the wrench space in 3 dimensions with either forces or torques. The picture on the right displays the convex hull constructed using forces in 3 dimensions.
2.1.6 Joint Parameters

Joint parameters refer to the displacements of the joints between links of a robot. In case of rotary or revolute joint, joint parameters are called joint angles and for prismatic joints, joint parameters are called joint offset.

2.1.7 Inverse Kinematics

When a robot is trying to grasp an object with the position known, joint angles of each joint need to be obtained in order to grasp the object. Joint angles of joints in the robot are determined when the kinematic equations and joint limits are satisfied and the end effector has reached the goal position[14].

Since the robot we are working with is a humanoid robot. Like human hands is not able to reach every point near the body like for example somewhere near the back of the body due to our joint limits. Robot arms also have joint limits. Hence to construct a workspace which is constructed of all the reachable positions by the end effectors of the robot is important in grasping problem. Figure 2.2 shows the workspace of Atlas robot.

Figure 2.2: We generated this figure using OpenRAVE. It shows the workspace of the left arm of Atlas robot with eight degrees of freedom. Reachability of each point within the ball centered at the shoulder joint is calculated. The red region can be reached by the left hand with most configurations (different joint parameters). And orange, yellow, green and blue have less configurations.
2.1.8 Gazebo and ODE

Dynamic simulators are simulators that simulates the time varying behavior of a system using a computer program.

Gazebo is a 3D dynamic simulator. It is able to reproduce any dynamic environments a robot may encounter in the real world[15]. ODE (Open Dynamics Engine) was created by Russel Smith, which is a widely used physics engine. This engine is able to simulate unilateral and bilateral constraints, perform collision detection, calculate mass and rotational functions. Gazebo is importing these features and creates both normal and abstract objects, for example, laser rays and boxes while the ODE is still functioning properly. Bullet physics engine is another physics engine Gazebo will be using.

Gazebo is now a widely used simulator. DRC simulator was used during DRC(DARPA Robotics Challenge) held by DARPA during the software part in which we participated and it was built on Gazebo software package.

2.1.9 ROS

ROS is a robotic software platform that provides operating system-like functionality across multiple computers[16]. On the ros wiki, it says: “It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.” Gazebo is utilizing some of ROS features.

2.1.10 OProfile

OProfile is a sample-based profiler for Linux. It measures the frequency and duration of function calls and its samples are taken periodically to indicate the detailed information of the program executed in the operating system[6]. OProfile is very useful for identifying processor performance bottlenecks. In this work, we choose OProfile as the tool to profile Gazebo.
2.1.11 DRC (DARPA Robotics Challenge)

DRC is a robotics competition held by DARPA[17]. The goal of this competition is to develop robots that are able to accomplish complicated tasks like walking and grasping objects in many kinds of difficult environments. DRC consists of two primary events, the software challenge and the hardware challenge. The software part is simulation only and the robot has to complete several tasks within a certain amount of time like picking up drill and hose, attaching the hose to standpipe and align them, turning the valve and driving, etc. Figure 2.3 shows the Atlas robot is trying to pick up the hose end and attach to the standpipe on the wall.

Figure 2.3: We simulated the scene in Gazebo. In the picture, the Atlas robot is trying to pick up the fire hose on the table with both hands and attach it to the standpipe.

2.1.12 Rigid Multibody Dynamics Simulation

Rigid multibody dynamics problem is usually solved by constructing discrete time step equations. The equations can be converted to LCP problem. There are many solvers to solve LCP problem.

For each time step, a discretized equation and a complementarity condition are used to solve for force information for the current time step when solving LCP problem. The acceleration of the rigid body is then known from the force information. With the previous configuration, such as position and velocity, we can compute the new velocity and position of the rigid bodies in the simulation. With
this information, it is ready to move on to the next time step and obtain the new force information.

Newton-Euler equation and equation for the unilateral and bilateral constraints together form the discrete time step equation.

Unilateral constraints refer to the constraints that prevent penetration of two rigid bodies. The gap between two bodies is often larger than zero. For example, when a ball is dropped to the ground, the ground is a unilateral constraint for the ball. Bilateral constraints refer to the joints of two bodies such as revolute joint, spherical joint, etc. The gap between two bodies is often equal to zero.

All the physics engines including ODE and bullet mentioned above provide approximate solutions to the discretized equation while satisfying the complementarity condition.

2.1.13 LCP

Linear complementarity problem (LCP) solves for problems satisfying the following conditions in which w, z and q are vectors with dimension n, M is a n by n matrix.

\[ w = Mz + q \] (2.1)

\[ 0 \leq w \perp z \geq 0 \] (2.2)

Equation 2.2 is the complementarity condition. If there are n elements in both w and z, the condition tells us for any i (i is any number between 0 and n), w_i multiply by z_i is zero and none of them are negative, which means for each pair of w_i and z_i, at least one of them is zero and the other is zero or positive number[18].

In multibody dynamics problem, equation 2.1 forms the discrete time step equation. The complementarity condition comes from the force law of unilateral contact which addresses the relation between contact force and the gap between two surfaces of two rigid bodies. When contact force is larger than zero, the gap is zero and when the contact force is zero, the gap is larger than or equals to zero.
2.2 Motivation

The main problem motivating this thesis is that the search for grasps is done under the assumption that the hand can achieve a desired grasp, but this is not true. Grasping an object is a dynamic process in which the object can be jostled significantly during the grasping attempt by the robot. As a result, when a grasp is achieved, it is not the intended grasp, so the analysis is not valid like it is displayed in figure 2.4.

Figure 2.4: We simulated the grasps in Gazebo. The picture on the left shows Sandia hand is going to grasp the drill on the table. The picture on the right shows Sandia hand grasping the drill. We can clearly see that the position and orientation of the drill have both changed after the grasp. Since the hand won’t adjust grasp due to the move of the object and the desired grasp is computed in the frame of torso. So once the object’s position or orientation changes, the relative position of the hand and object doesn’t match the computed grasp from GraspIt!. Thus the grasp cannot be the desired grasp because desired grasp is obtained with the assumption that the object is static.

The main goal of this thesis is to use dynamic simulation of robot grasping to approximate the effects of accidental bumping on the final grasp. This will be done by injecting errors in the position of the object, without “telling” the robot. When the robot reaches toward the object to grasp it, the object will be in a slightly different position than assumed when designing the reaching trajectory. Therefore, the hand will disturb the object a bit. Usually the robot will achieve a secure grasp, but it will be different than the planned grasp. In the worst case the grasp will fail.

The secondary goal of this thesis is to use Monte Carlo simulation to identify
grasps selected by GraspIt! that are robust to object position uncertainty and accidental bump. For each grasp in the database, grasp acquisition will be attempted many times, each time, with the object in an uncertain position selected at random from an uncertainty “ball” in the object’s configuration space. The object with the highest grasp success rate over thorough sampling of the uncertainty ball is the most robust grasp.

The robustness of a grasp is also highly related to the performance of the dynamic simulator.

Since we want to simulate grasp just like in real world, so firstly, the simulator has to be accurate.

Secondly, we execute grasps many times, sometimes, thousands of times depending on how many samples we are choosing for the uncertainty of the position of the object. Hence we want the simulator run as fast as possible.

Last but not least, the simulator has to be constructed properly so that it doesn’t have memory leak. Memory leak can cause the simulation to be stuck at some point.

In this work, we want to time-based analysis of the performance of Gazebo. If the time is abnormally longer than the code section should take, then it needs some kind of optimization. Thus another goal we want to achieve in this work is to find out which part of Gazebo should be optimized to improve the performance of the simulation.

2.3 Summary of Current Work and Results

This work includes two parts: dynamic grasp analysis and profiling of Gazebo and ODE.

2.3.1 Dynamic Grasp Analysis

In this work, we use simulation to test large amount of different grasps and select some grasps that have highest success rate to accomplish the grasp task.

Uncertainty of the object pose is considered in this work. We start with taking a grasp generated by GraspIt! then inject pose error to the position of the
drill which is being grasped and execute that grasp multiple times. Finally success rate is calculated for ten grasps.

The results show the success rates obtained by dynamic grasp analysis differ quite a lot with what was calculated by GraspIt!. Which means grasp analysis based on static object assumption is not completely reliable in dynamic environment.

2.3.2 Profiling of Gazebo and ODE

Four benchmark simulations are used in Gazebo environment when doing profiling. We profile Gazebo using OProfile as the tool to analyze the time cost by each part of the program. It turned out Gazebo is not so efficient and some redundant part can be removed or optimized.

When profiling ODE, a timing library using a standard counter: rdtsc(Real Time-Stamp Counter) is added to ODE quickstep code from which CPU cycles each function in the solver costs are calculated. We then convert the CPU cycles to the time the functions consume. Same four benchmark simulation are used for analyzing timing of ODE solver as we do for profiling Gazebo. Same four benchmark simulations are used as profiling of Gazebo. Based on the results of the experiments, ODE is quite efficient right now.
3. EXPERIMENT DETAILS AND RESULTS

3.1 Dynamic Grasp Analysis

Figure 3.1: The scene was simulated in Gazebo. The picture shows the Atlas robot has grasped and picked up the drill on table in Gazebo.

In the work of dynamic grasp analysis, pick up drill task is used like it’s shown in figure 3.1. For this task, 656 grasps are generated by GraspIt! and we put them in grasp database. To test these grasps one by one will take a large amount of time, and we know although the grasps are generated from GraspIt! and they all have high success rate according to the force closure test, however, we have only imported the handle of the drill to GraspIt! so some grasps will definitely fail. For example, if the hand reaches the handle with three fingers coming from the top and the thumb coming from the bottom, that type of grasps will fail because it didn’t consider the top and bottom part of the drill when it was generated. Hence we filtered the 656 grasps before we do dynamic grasp analysis.
Figure 3.2: The picture shows two chosen grasps and it was simulated using Panda3D. In the picture, you can see some penetrations because collision is not considered in the visualization of the grasp. Without considering dynamics, desired grasps can be seen more clearly generated by GraspIt!®. There are eight more grasps which are not shown.

Ten out of the 656 grasps were chosen from grasp database according to the angle between hand frame and the object. The grasp is chosen if the resulting hand frame is within 0.3 rad with the frame of the object. Two of the chosen grasps are displayed in figure 3.2.

After we obtained the 10 chosen grasps, for each grasp, we start with taking the grasp from grasp database like it is shown in the flow chart in figure 3.3. What we get from the grasp database is the relative hand position to the object. Both pre-grasp and grasp positions are included as shown in figure 3.4. Since we use the torso frame as the reference frame when computing joint parameters, transformations of the hand positions are made from the object frame to torso frame.

The torso is moving all the time during the whole simulation, joint angles calculated using IK with respect to the initial torso frame were calculated in advance for pre-grasp and grasp movements.

For each simulation we measure the quality of the simulated grasp and gather the quality scores of all simulations we ran for a particular grasp as raw data for evaluation of the quality of the grasp under uncertainty.

The uncertainty model is parametrized by $[x, y, \theta]$. The configurations of the drill were changed with deviations of 1cm in $x$, 1cm in $y$, and 1.1 degrees in $\theta$ of the object. For each grasp, 64 simulations are sampled to perform grasp evaluation with different object configuration.

Two ways can be used to evaluate grasps in this work. One way is to evaluate
grasp without tactile sensor. We judge a grasp is successful if the height of the object is larger than 0.5 meters while the height of the table is 1.0 meter. A listener program was called by dynamic simulation program to obtain the height of the object. We set the height lower than the height of table because after deleting the table the hand and the object will lower a little due to gravity. We give score 0 if the height of the drill is smaller than 0.5 meters (which means the object falls to the ground) and we score it 1 if the height of the drill is larger than 0.5 meters (which
Figure 3.4: The two grasps were simulated in Gazebo. The left picture shows the pre-grasp position of the grasp drill task. The picture on the right shows the grasp position.

means the object remains in the hand)

The number of successful grasps and the total number of grasps are displayed after each run. They can also be written to the old database file. The corresponding success rate is the ratio of the number of successful grasps and the total number of grasps.

Another way is grasp evaluation using tactile sensor. Figure 3.5 shows the sandia hand with tactile sensor on the fingers. A simple version of low pass filter was added to the tactile sensor. We first listen to a rostopic to get the raw data from tactile sensor. Then we use a low pass filter to get rid of the noise and publish another rostopic to publish the score of grasp. We give score 0 if there are no contact at all on the hand. We give score 1 if there are only contacts on the upper part of the fingers (finger tip). We give score 2 if the contacts are on the lower part of the fingers and we give score 3 which is the highest score if there are contacts on the palm.

In this work, the first way is used as the criteria of grasp evaluation.

Figure 3.6 displays the dynamic grasp analysis results. The height of the boxes represent the success rate in percentage of the 10 chosen grasps. We know those 10 grasps are obtained from GraspIt!. They should all have high success rate according
Figure 3.5: The above figure shows the sandia hand and the sensors location on the fingers and palm[19].

Figure 3.6: The above figure is generated according to the success rate results from dynamic grasp analysis of the chosen 10 grasps based on the first grasp evaluation method.
to force-closure test. However the success rates differ quite a lot in dynamic analysis. The lowest one only has success rate of 18.5%. It means in dynamic environment, contact information has changed a lot so the result differs quite a lot with force-closure test result. So we cannot rely only on the grasps obtained by GraspIt!. If we need grasps with success rate higher than 70%, from the dynamic grasp analysis results, grasp 3, 6, 7, 8 can be chosen as good grasps. If we need success rate to be higher than 90%, only grasp 3 can be chosen. Thus we can see we definitely need to use dynamic grasp analysis to filter the grasps.

3.2 Profiling of Gazebo and ODE

Figure 3.7: The pictures display the four simulation scenes chosen for profiling of Gazebo and ODE. We simulated the four scenes using Gazebo.
Table 3.1: Hardware conditions of computer for profiling of Gazebo

<table>
<thead>
<tr>
<th>Processor</th>
<th>RAM</th>
<th>Graphics Card</th>
<th>OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel Core i7 3770 @ 3.40 GHz</td>
<td>15.6 GB</td>
<td>NVIDIA GTX 580</td>
<td>Ubuntu 12.04</td>
</tr>
</tbody>
</table>

3.2.1 Profiling of Gazebo

Four benchmark simulations are used for the profiling of Gazebo (as in the figure 3.7). Hardware conditions of the computer used for simulation are displayed in table 3.1. The main purpose in choosing these four simulations is to compare the results from them and determine how much time the physics engine in Gazebo takes under different circumstances. For example, we want to see how much time does the physics engine takes when there is a robot and compare it to the simulation without the robot.

In the first simulation, there are only some boxes. Obviously there are no joints in the scene. In the second simulation, Atlas robot is added. Since there are multiple bodies and joints, in theory, the physics engine should spend more time than the previous one. In the third simulation, Atlas is grasping a pipe. In the fourth one, Atlas is picking up a hose end. It is expected that the duration of the last simulation should be greater than the other three because it contains the largest amount of movement. We would like to analyze how much time it takes for the physics engine when there are no joints in simulation, when there are multiple joints, when the robot is standing still and when the robot is making some movements. If there are not many differences, we can ask the robot to complete much more difficult tasks without worrying if the simulator can handle it.

Table 3.2 is a fraction of the profiling result of Atlas picking up hose end simulation from OProfile. The first column is the total number of calls of each function during the simulation. The second column shows the percentage of number of calls among the total function calls for each function. The program name refers to the name of the code that contains the function. Image name is the dynamic libraries the code belongs to.

Looking at the table, it is obvious that quickstep.cpp was called most during the simulation in which LCP was solved. And it belongs to the physics engine.
### Table 3.2: OProfile report for picking up hose end simulation

<table>
<thead>
<tr>
<th>samples</th>
<th>%</th>
<th>program name</th>
<th>image name</th>
</tr>
</thead>
<tbody>
<tr>
<td>249362</td>
<td>14.9158</td>
<td>quickstep.cpp:519</td>
<td>libgazebo_ode.so.1.4.0</td>
</tr>
<tr>
<td>77001</td>
<td>4.6059</td>
<td>(no location information)</td>
<td>libstdc++.so.6.0.16</td>
</tr>
<tr>
<td>74469</td>
<td>4.4544</td>
<td>(no location information)</td>
<td>libnvidia-glcore.so.295.4</td>
</tr>
<tr>
<td>42989</td>
<td>2.5714</td>
<td>malloc.c:3415</td>
<td>libc-2.15.so</td>
</tr>
<tr>
<td>42464</td>
<td>2.5400</td>
<td>malloc.c:2910</td>
<td>libc-2.15.so</td>
</tr>
<tr>
<td>37194</td>
<td>2.2248</td>
<td>malloc.c:3906</td>
<td>libc-2.15.so</td>
</tr>
<tr>
<td>24997</td>
<td>1.4952</td>
<td>OPC_OBBCollider.cpp:530</td>
<td>libgazebo_opcode.so.1.4.0</td>
</tr>
<tr>
<td>20642</td>
<td>1.2347</td>
<td>quickstep.cpp:1401</td>
<td>libgazebo_ode.so.1.4.0</td>
</tr>
<tr>
<td>18941</td>
<td>1.1330</td>
<td>(no location information)</td>
<td>libros.cpp.so</td>
</tr>
<tr>
<td>18533</td>
<td>1.1086</td>
<td>State.cc:39</td>
<td>libgazebo_physics.so.1.4.0</td>
</tr>
<tr>
<td>16434</td>
<td>0.9830</td>
<td>(no location information)</td>
<td>libstdc++.so.6.0.16</td>
</tr>
<tr>
<td>14626</td>
<td>0.8749</td>
<td>pthread_mutex_lock.c:47</td>
<td>libpthread-2.15.so</td>
</tr>
<tr>
<td>13652</td>
<td>0.8166</td>
<td>OgreOctreeNode.cpp:177</td>
<td>Plugin_OctreeSceneManager.so</td>
</tr>
<tr>
<td>13332</td>
<td>0.7975</td>
<td>memcpysse3-back.S:60</td>
<td>libc-2.15.so</td>
</tr>
<tr>
<td>13308</td>
<td>0.7960</td>
<td>(no location information)</td>
<td>glC4mhpZ (deleted)</td>
</tr>
<tr>
<td>13264</td>
<td>0.7934</td>
<td>pthread_mutex_unlock.c:289</td>
<td>libpthread-2.15.so</td>
</tr>
<tr>
<td>13102</td>
<td>0.7837</td>
<td>malloc.c:4192</td>
<td>libc-2.15.so</td>
</tr>
<tr>
<td>13029</td>
<td>0.7793</td>
<td>(no location information)</td>
<td>python2.7</td>
</tr>
<tr>
<td>11836</td>
<td>0.7080</td>
<td>SDF.cc:551</td>
<td>libgazebo_sdf_interface.so.1.4.0</td>
</tr>
<tr>
<td>11210</td>
<td>0.6705</td>
<td>(no location information)</td>
<td>libpthread-2.15.so</td>
</tr>
<tr>
<td>10384</td>
<td>0.6211</td>
<td>pthread_mutex_lock.c:47</td>
<td>libpthread-2.15.so</td>
</tr>
<tr>
<td>10003</td>
<td>0.5983</td>
<td>memcmp-sse4.S:52</td>
<td>libc-2.15.so</td>
</tr>
<tr>
<td>8707</td>
<td>0.5208</td>
<td>(no location information)</td>
<td>libtinyxml.so.2.6.2</td>
</tr>
<tr>
<td>8268</td>
<td>0.4946</td>
<td>ModelState.cc:151</td>
<td>libgazebo_physics.so.1.4.0</td>
</tr>
</tbody>
</table>

That’s why we are interested in the profiling of the physics engine.

Besides physics engine, the performance of the rendering system is also essential because the physics and rendering parts should consume most of the time in theory. In Gazebo, OGRE is used as the rendering tool. From the profiling result, the rendering system did play an important role in all four simulations.

From the profiling result, we have noticed that for every simulation, a code called malloc.c takes significant amount of time. So in the results, the percentages of time taken by malloc.c are also displayed.

The percentages of time for physics engine, rendering system and malloc.c for all four simulations were computed according to the OProfile results. Figure 3.8 - 3.10 display the comparison results for the four simulations. In the graph,
ODE(25.4%) means during this simulation, ODE takes 25.4% of the total time. The others are similar.

From figure 3.8, we can notice there is an increase of the rendering part when the Atlas robot is added to the simulation instead of the boxes. However there is a big decrease of the ODE part. It is not very normal because when an Atlas robot with about 100 links and joints is added to the simulation, physics should take more time.

In figure 3.9, it can be seen that the percentage of time of ODE for simulation 3 is a little bit higher than simulation 2. This is due to that in simulation 3, Atlas robot is grasping the pipe and makes a movement, while in simulation 2 Atlas is standing still. This is why physics engine takes more time to compute.

In figure 3.10, except for ODE, percentages of time for other parts of simulation 4 are higher than simulation 3. However in simulation 4, the robot picks up the hose from table and in simulation 3, the robot just closes the fingers of right hand. Obviously the movement of the robot is larger than in simulation 3. It is expected that it will take longer for physics engine to calculate in simulation 4.

Figure 3.8: The pie graphs display comparison of the percentages of time ODE, rendering (OGRE) and malloc.c consumed during the first and second simulations.

From the profiling Gazebo results, we can see there are several problems with Gazebo.

Firstly, the physics engine and rendering system together take no more than 40% percent of the total computation time which is not very ideal. For dynamic simulation, the physics engine and rendering should take at least 50% of the computation time, however right now they are only a small fraction.
Secondly, the malloc.c program takes about 9% on average in all four simulations. It is only used to allocate and free memory. However it takes more computation time than it is expected. We do think it is worthwhile to make optimization to this part.

Lastly, the percentages taken by the physics engine is not what we expected. Like in figure 3.10, the physics engine should take more computation time in simulation 4 than in simulation 3, but it does not. However in figure 3.9, it behaves as we expected. This means the simulator lacks stability sometimes.

3.2.2 Profiling of ODE

The same four simulations are used to analyze the timing of the Gauss-seidel method. It is the solver ODE is using currently. Our own timing library is added using a standard counter: rdtsc(Real Time-Stamp Counter) to ODE quickstep code
to analyze the timing information.

\[
\text{CPU cycles} = \text{end instruction} - \text{start instruction} \quad (3.1)
\]

\[
\text{time cost} = \frac{\text{CPU cycles}}{\text{CPU frequency}} \quad (3.2)
\]

By calling rdtsc at the start of the function, we are able to get the start instructions of that function. And then rdtsc is called at the end of the function to get the end instructions of that function. By subtracting the end instruction and start instructions, we can obtain the CPU cycles that this function costs. Finally, we divide the CPU cycles by the CPU frequency to obtain the time cost of the function.

<table>
<thead>
<tr>
<th>Events</th>
<th>time/ms</th>
<th>max/ms</th>
<th>min/ms</th>
<th>stddev/ms</th>
<th>cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>preprocessing</td>
<td>0.0015</td>
<td>0.0495</td>
<td>0.0001</td>
<td>0.0005</td>
<td>4964</td>
</tr>
<tr>
<td>create J</td>
<td>0.0012</td>
<td>0.0832</td>
<td>0.0001</td>
<td>0.0006</td>
<td>4080</td>
</tr>
<tr>
<td>compute rhs</td>
<td>0.0005</td>
<td>0.0198</td>
<td>0.0000</td>
<td>0.0002</td>
<td>1656</td>
</tr>
<tr>
<td>compute rhs_precon</td>
<td>0.0004</td>
<td>0.0359</td>
<td>0.0000</td>
<td>0.0002</td>
<td>1200</td>
</tr>
<tr>
<td>solving LCP problem</td>
<td>0.0131</td>
<td>0.6133</td>
<td>0.0033</td>
<td>0.0147</td>
<td>44676</td>
</tr>
<tr>
<td>start pgs rows</td>
<td>0.0114</td>
<td>0.5715</td>
<td>0.0031</td>
<td>0.0139</td>
<td>38736</td>
</tr>
<tr>
<td>wait for threads</td>
<td>0.0000</td>
<td>0.0095</td>
<td>0.0000</td>
<td>0.0000</td>
<td>102</td>
</tr>
<tr>
<td>threads done</td>
<td>0.0010</td>
<td>0.0328</td>
<td>0.0001</td>
<td>0.0003</td>
<td>3374</td>
</tr>
<tr>
<td>velocity update due to cf</td>
<td>0.0002</td>
<td>0.0094</td>
<td>0.0000</td>
<td>0.0001</td>
<td>586</td>
</tr>
<tr>
<td>compute velocity update</td>
<td>0.0001</td>
<td>0.0301</td>
<td>0.0000</td>
<td>0.0001</td>
<td>400</td>
</tr>
<tr>
<td>update position</td>
<td>0.0030</td>
<td>0.0348</td>
<td>0.0002</td>
<td>0.0008</td>
<td>10294</td>
</tr>
<tr>
<td>velocity update due to cf</td>
<td>0.0001</td>
<td>0.0172</td>
<td>0.0000</td>
<td>0.0001</td>
<td>330</td>
</tr>
<tr>
<td>tidy up</td>
<td>0.0002</td>
<td>0.0169</td>
<td>0.0000</td>
<td>0.0001</td>
<td>834</td>
</tr>
</tbody>
</table>

Table 3.3-3.6 are generated according to the timer results of the four simulations. The left column displays the main functions of the Gauss-Seidel method. “preprocessing” is used to get the masses, compute inertia tensor, add gravity forces, get joint information, etc for all bodies. “create J” creates jacobian matrix from the constraints information. “Compute rhs” means compute right hand side. “compute rhs precon” is compute preconditioned right hand side. In “solving LCP problem”, LCP is solved and also the forces are obtained. Also, the time taken by the solver PGS is also calculated in “start pgs rows”. “wait for threads” and “threads done” are for multi threads processing. After LCP is solved, all the forces or impulses are
Table 3.4: Timer report for Atlas simulation

<table>
<thead>
<tr>
<th>Events</th>
<th>time/ms</th>
<th>max/ms</th>
<th>min/ms</th>
<th>stddev/ms</th>
<th>cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>preprocessing</td>
<td>0.0185</td>
<td>0.0802</td>
<td>0.0131</td>
<td>0.0039</td>
<td>63011</td>
</tr>
<tr>
<td>create J</td>
<td>0.0060</td>
<td>0.0541</td>
<td>0.0047</td>
<td>0.0021</td>
<td>20300</td>
</tr>
<tr>
<td>compute rhs</td>
<td>0.0029</td>
<td>0.0385</td>
<td>0.0027</td>
<td>0.0013</td>
<td>9819</td>
</tr>
<tr>
<td>compute rhs_precon</td>
<td>0.0023</td>
<td>0.0709</td>
<td>0.0022</td>
<td>0.0012</td>
<td>7761</td>
</tr>
<tr>
<td>solving LCP problem</td>
<td>0.4068</td>
<td>3.4665</td>
<td>0.3399</td>
<td>0.0621</td>
<td>1383661</td>
</tr>
<tr>
<td>start pgs rows</td>
<td>0.3917</td>
<td>0.9169</td>
<td>0.3269</td>
<td>0.0411</td>
<td>1332256</td>
</tr>
<tr>
<td>wait for threads</td>
<td>0.0000</td>
<td>0.0084</td>
<td>0.0000</td>
<td>0.0001</td>
<td>73</td>
</tr>
<tr>
<td>threads done</td>
<td>0.0006</td>
<td>3.5415</td>
<td>0.0002</td>
<td>0.0289</td>
<td>1925</td>
</tr>
<tr>
<td>velocity update due to cf</td>
<td>0.0019</td>
<td>0.1872</td>
<td>0.0010</td>
<td>0.0019</td>
<td>6143</td>
</tr>
<tr>
<td>compute velocity update</td>
<td>0.0006</td>
<td>0.0175</td>
<td>0.0005</td>
<td>0.0003</td>
<td>1880</td>
</tr>
<tr>
<td>update position</td>
<td>0.0137</td>
<td>0.3386</td>
<td>0.0107</td>
<td>0.0042</td>
<td>46633</td>
</tr>
<tr>
<td>velocity update due to cf</td>
<td>0.0003</td>
<td>0.0113</td>
<td>0.0003</td>
<td>0.0003</td>
<td>1100</td>
</tr>
<tr>
<td>tidy up</td>
<td>0.0006</td>
<td>0.0259</td>
<td>0.0004</td>
<td>0.0004</td>
<td>1886</td>
</tr>
</tbody>
</table>

Table 3.5: Timer report for Atlas grasping standpipe simulation

<table>
<thead>
<tr>
<th>Events</th>
<th>time/ms</th>
<th>max/ms</th>
<th>min/ms</th>
<th>stddev/ms</th>
<th>cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>preprocessing</td>
<td>0.0167</td>
<td>0.1960</td>
<td>0.0008</td>
<td>0.0067</td>
<td>56752</td>
</tr>
<tr>
<td>create J</td>
<td>0.0072</td>
<td>0.0629</td>
<td>0.0004</td>
<td>0.0029</td>
<td>24549</td>
</tr>
<tr>
<td>compute rhs</td>
<td>0.0033</td>
<td>0.0218</td>
<td>0.0001</td>
<td>0.0015</td>
<td>11355</td>
</tr>
<tr>
<td>compute rhs_precon</td>
<td>0.0027</td>
<td>0.0187</td>
<td>0.0001</td>
<td>0.0012</td>
<td>9103</td>
</tr>
<tr>
<td>solving LCP problem</td>
<td>0.4060</td>
<td>2.7401</td>
<td>0.0055</td>
<td>0.1597</td>
<td>1380850</td>
</tr>
<tr>
<td>start pgs rows</td>
<td>0.3893</td>
<td>2.6152</td>
<td>0.0049</td>
<td>0.1531</td>
<td>1324147</td>
</tr>
<tr>
<td>wait for threads</td>
<td>0.0000</td>
<td>0.0049</td>
<td>0.0000</td>
<td>0.0000</td>
<td>36</td>
</tr>
<tr>
<td>threads done</td>
<td>0.0004</td>
<td>0.1000</td>
<td>0.0002</td>
<td>0.0009</td>
<td>1212</td>
</tr>
<tr>
<td>velocity update due to cf</td>
<td>0.0022</td>
<td>0.1978</td>
<td>0.0000</td>
<td>0.0019</td>
<td>7456</td>
</tr>
<tr>
<td>compute velocity update</td>
<td>0.0006</td>
<td>0.0107</td>
<td>0.0000</td>
<td>0.0003</td>
<td>2173</td>
</tr>
<tr>
<td>update position</td>
<td>0.0146</td>
<td>0.2821</td>
<td>0.0010</td>
<td>0.0059</td>
<td>49640</td>
</tr>
<tr>
<td>velocity update due to cf</td>
<td>0.0004</td>
<td>0.0090</td>
<td>0.0000</td>
<td>0.0002</td>
<td>1194</td>
</tr>
<tr>
<td>tidy up</td>
<td>0.0005</td>
<td>0.0173</td>
<td>0.0000</td>
<td>0.0003</td>
<td>1644</td>
</tr>
</tbody>
</table>

computed, so velocity and position are able to update according to the forces. They are updated in “velocity update due to cf” (in which cf means constraint force), “compute velocity update”, and “update position”.

The second column of the report displays the time cost by each function calculated with equation 3.1 and 3.2 during the current time step. The results also show the maximum, minimum and standard deviation of the time each function cost
Table 3.6: Timer report for Atlas picking up hose simulation

<table>
<thead>
<tr>
<th>Events</th>
<th>time/ms</th>
<th>max/ms</th>
<th>min/ms</th>
<th>stddev/ms</th>
<th>cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>preprocessing</td>
<td>0.0199</td>
<td>0.5565</td>
<td>0.0004</td>
<td>0.0104</td>
<td>67529</td>
</tr>
<tr>
<td>create J</td>
<td>0.0121</td>
<td>0.0535</td>
<td>0.0005</td>
<td>0.0055</td>
<td>41262</td>
</tr>
<tr>
<td>compute rhs</td>
<td>0.0046</td>
<td>0.0238</td>
<td>0.0001</td>
<td>0.0022</td>
<td>15603</td>
</tr>
<tr>
<td>compute rhs_precon</td>
<td>0.0037</td>
<td>0.0233</td>
<td>0.0001</td>
<td>0.0017</td>
<td>12579</td>
</tr>
<tr>
<td>solving LCP problem</td>
<td>0.6883</td>
<td>6.1464</td>
<td>0.0108</td>
<td>0.3011</td>
<td>2341028</td>
</tr>
<tr>
<td>start pgs rows</td>
<td>0.6621</td>
<td>2.4873</td>
<td>0.0101</td>
<td>0.2721</td>
<td>2251791</td>
</tr>
<tr>
<td>wait for threads</td>
<td>0.0000</td>
<td>0.0084</td>
<td>0.0000</td>
<td>0.0001</td>
<td>47</td>
</tr>
<tr>
<td>threads done</td>
<td>0.0005</td>
<td>0.0249</td>
<td>0.0001</td>
<td>0.0004</td>
<td>1597</td>
</tr>
<tr>
<td>velocity update due to cf</td>
<td>0.0033</td>
<td>0.2099</td>
<td>0.0001</td>
<td>0.0028</td>
<td>11338</td>
</tr>
<tr>
<td>compute velocity update</td>
<td>0.0007</td>
<td>0.0254</td>
<td>0.0000</td>
<td>0.0005</td>
<td>2324</td>
</tr>
<tr>
<td>update position</td>
<td>0.0167</td>
<td>0.3123</td>
<td>0.0004</td>
<td>0.0094</td>
<td>56635</td>
</tr>
<tr>
<td>velocity update due to cf</td>
<td>0.0004</td>
<td>0.0213</td>
<td>0.0000</td>
<td>0.0003</td>
<td>1215</td>
</tr>
<tr>
<td>tidy up</td>
<td>0.0005</td>
<td>0.0146</td>
<td>0.0000</td>
<td>0.0003</td>
<td>1780</td>
</tr>
</tbody>
</table>

during the whole simulation. “CPU cycles” is the CPU cycles cost by this function during the simulation.

Table 3.7: Percentages of time consumed by LCP procedure

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>percentage of total time taken by LCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>four simulations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>61.5%</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>89.6%</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>89.3%</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>91.7%</td>
</tr>
</tbody>
</table>

The “solving LCP problem” just as how it is named, is the function in which LCP is solved. It is the dominant part according to the timer report. The percentages of the time of solving LCP function are calculated for all four simulations. Table 3.7 presents the percentages of time consumed by LCP procedure. Figure 3.11 was generated according to table 3.7. It can be seen that in the first simulation, LCP procedure only takes 61.5% of the total time. It takes the physics engine less time to compute compared with the other three simulations because there are only unilateral constraints and no bilateral constraints in the simulation. The other three simulations have higher percentages of time taken by solving LCP problem and the last simulation has the highest percentage. This is because in the last simulation,
Figure 3.11: This figure shows the percentage of time consumed by solving LCP problem in quickstep.

Atlas robot made the largest movements among the four simulations. It is obvious from the results that solving LCP problem took most of the time among the function calls. This makes sense because the solver should take most of the computation time. And we have also noticed that even the maximum time of solving LCP problem within one time step is about 2.5 milliseconds which is quite a promising result.
4. FUTURE WORK AND CONCLUSION

4.1 Future Work

4.1.1 Dynamic Grasp Analysis

In this work, we planned to test all the grasps obtained from grasp database, however, it turned out that Gazebo is not able to run for more than 300 simulations continuously at one time. So we ended up with filtering the grasps first by looking at the angles of the hand frame and object frame and eventually only tested 10 grasps. In future work, we will look for a way to run Gazebo continuously and test all the grasps so we can make better use of grasp database.

Since we know static object assumption can lead to incorrect contact information in dynamic environment but force-closure is a standard way to evaluate grasp quality. Thus another thing we would like to accomplish is to add force-closure as another grasp evaluation during the dynamic grasp analysis. Hence the score given to each grasp will be the ball radius of the largest ball inside the convex hull in wrench space.

It would also be worthwhile to do a more precise prediction of the motion of the object and calculate the motion of the end effectors of the robot accordingly to achieve a good grasp.

To apply the dynamic grasp analysis to a real robot hand is what we are most interested. Robust grasp is able to obtained by calculating the success rate for real robot hand.

4.1.2 Profiling of Gazebo and ODE

Since the profiling results of the last three simulations don’t differ quite a lot, it means the movements of atlas not large enough. So firstly, designing some simulations in which atlas is making larger movements like falling to massive amount of balls, hit atlas with large force, etc would be important. In these simulations, the time cost by the solver is expected to increase a lot.

Secondly, during the dynamic grasp simulation, Gazebo usually stops at some
point. That is the simulation is not running sometimes and the simulation time just “freezes”. Obviously something is wrong with the simulator during the simulation. We believe memory leak exists. That’s why we would also like to use the dynamic grasp analysis simulation to profile Gazebo again to see which section of the code in Gazebo causes the problem.

Another work we would like to achieve in future is to optimize Gazebo for example the malloc.c code can be modified so it costs less time. Then we are able to compare the profiling results before and after the optimization and see if the optimization yields better performances.

4.2 Conclusion

4.2.1 Dynamics Grasp Analysis

The dynamic grasp analysis result shows the grasps generated from grasp database based on force closure test are not very reliable. Some of the grasp’s success rate is only 18.5%. That’s far from what is expected. The analysis result indicate to us that to filter the grasps generated by GraspIt! using dynamic grasp analysis is an important step in achieving robust grasp in dynamic environment.

4.2.2 Profiling of Gazebo and ODE

From the profiling results, we can see the physics engine takes most of the computation time among all parts which makes sense because that is the core of the simulator. But it is not very stable sometimes. The results show when there is a bigger movement of the robot, physics engine sometimes take less time than when the movement is smaller, however it is expected to behave in the opposite way. And also the allocating memory code malloc.c takes significant amount of time. This component represents a more compute time than it should take.

From the results of the profiling of ODE it can be seen even the maximum time of the function is only 2.5 miliseconds for one time step. So it means ODE is in a good shape. There is not much that needs to be improved based on the results from this work.
REFERENCES


