BiRRTOpt: A COMBINED SOFTWARE FRAMEWORK FOR MOTION PLANNING APPLIED ON ATLAS ROBOT

by

Lening Li

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

May 2016

APPROVED:

Professor Michael Gennert, Major Thesis Advisor

Professor Dmitry Berenson, Reader

Abstract

The rise of robots is becoming unstoppable judging by how much effort and money has been invested in this Robotics field so far just these years. Carl Frey and Michael Osbourne in Oxford University released a paper in 2013 and claimed that around 47 percent of current jobs would be automated in the next two decades[9]. But planning robot motion still remains a major problem in Robotics regardless of countless approaches proposed in multiple aspects trying to solve it. TrajOpt(Trajectory Optimizer)[23] is a state-of-art optimization-based software framework for planning robot motions. TrajOpt generates trajectory through constrained sequential convex optimization given several initial guesses, meaning TrajOpt would focus on finding the local minimum guided by an initial guess. However, depending on the complex environment and robot mechanical structure, it sometimes would suffer from being stuck in the local minimum which is not a feasible trajectory. However, BiRRT(Bidirectional Rapidly exploring random tree)[16] is probabilistic complete. BiRRT is a sampling-based method. It has been widely used due to its property, probabilistic completeness. But without using any smoothing techniques, the trajectory generated by BiRRT mostly is inexecutable on the real robot. The objective of proposing this work is to use the sample-based method to enable the TrajOpt become probabilistic complete, which guarantees that considering the solution being present the planner has the capability of acquiring the optimized trajectory. I also intend to experimentally evaluate the performance of this improved method in the simulation called Gazebo[7] and on the real Atlas robot[5] over a wide range of environmental settings.

Acknowledgements

Firstly I would like to express my sincere gratitude to my advisor Professor Michael Gennert for his patient guidance and constructive ideas from the beginning of my experience at WPI(Worcester Polytechnic Institute). Without his help, I could never have finished this thesis work and learned so much during this process. Discussing with him about my confusion which I met during my research and implementation was always a treasurable and extraordinary experience for me. I consider it was an honor to work with him in DRC(Darpa Robotics Challenge) Laboratory and WPI Humanoid Robotics Laboratory.

Secondly besides my advisor, my thanks also goes to Professor Dmitry Berenson who has taught a fabulous motion planning class and provided valuable advices and insightful comments on my work.

Thirdly I am grateful for my teammates in the DRC lab who provided me with an incredible opportunity to join the team, which started my robotics journey. Being friends with them was the luckiest thing that happened to me during my life here. Their encouragement motivated me to carry on my study and research. And they were always helping clarify my thoughts when I got trapped throughout the project.

Furthermore, I especially thank Xianchao Long for his great help and knowledge on TrajOpt. Thank him for sharing some ides with me and his patience during my learning phase.

In the end, I would like to thanks my parents, Guang Li and Minli Lu. I could never had the chance to start my study and research journey at WPI without their support and encouragement that was required for make this tough decision.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

According to a report[19] published in 2014 by JM Ortman, right now the United States is experiencing a dramatic growth in its older population. Additionally, in this report it says that in 2050 there will be around 83.7 million individuals aged 65 and over throughout the whole country. At that time the older population would occupy the most percentage of the age structure. Supporting them would be a very heavy burden on the youths and middle ages' shoulders. In this reason, people are urgently looking for some intelligent agents to replace labor forces which have the capability of taking care of the elders especially in most developed countries.

Among those intelligent agents the robot is the most representative one out of those agents. Even though, just recently the AlphaGo defeated a legendary GO champion Lee, there is still a very long journey for human beings to be totally replaced by the robot and making robots highly intelligent. Presently an extremely active field in Robotics is motion planning. The motion planning problem known as navigation problem whose main purpose is to navigate or control one or multiple robots to move in an environment while satisfying some movement constraints including avoiding collisions, joint limits and full body balance for humanoid robot.

All videos corresponding to the results described in this work are available in my YouTube channel[18].

1.1 Problem Overview

If the DOF(degrees of freedom) are less than or equal to 6, as long as we know the trajectory composed by a series of points, an inverse kinematics approach could be easily implemented to calculate the joint configurations. However, if the DOF are larger than 6, motion planning would become necessary. There would not be an unique solution for a reachable pose of the end effector, which indicates that there are some redundancies among all joints. That is the main reason why we would need motion planning.

During the DRC, WPI-CMU team used TrajOpt as their major tool to manipulate the Atlas robot. Allowing individuals to set constrains for the robot and planing a collision-free path in an uncertain environment are huge advantages compared to other planning methods. However, there are still some shortcomings of TrajOpt existing. The main problem is that this optimization method highly depends on given sampled trajectories. That indicates that if the given trajectories are not feasible there is a high possibility that the TrajOpt is not able to find the solution, meaning TrajOpt gets stuck in a infeasible local minimum, even though a very obvious solution exists as shown in Figure 1.1. In such an environment, the robot could lift the left arm over the bar, which would prevent the left arm from hitting the bar, and then move the left arm forward.

BiRRT as an extension of RRT(Rapidly-exploring Random Tree) has been widely used in motion planning. Past applications has revealed that this sampling-based method has comparatively better performance in searching the high dimensional

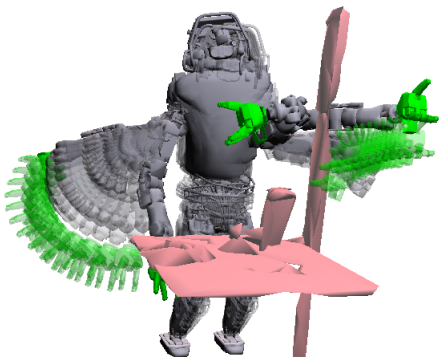

Figure 1.1: TrajOpt Gets Stucked in Infeasible Local minimum

space as shown in Figure 1.1. The search tree can efficiently explore the configuration space to find the solution. The major reason why sampling-based methods are very popular is the property of probabilistic completeness, meaning given unlimited time it would definitely find the solution as long as solutions exist. However, because BiRRT generates trajectories without considering stability and balance constraints applying this approach on bipedal robots becomes very difficult. That is, it may return a trajectory as shown in Figure 1.1 that satisfies the robot's kinematic constrains, but which, if executed, would result in the robot toppling over.

In order to fix those problems motioned above, the main goal of this thesis work is to combine the TrajOpt with BiRRT to generate the trajectory and perform some experiments on the real Atlas robot as shown in Figure 1.1. Additionally, I will also experimentally evaluate and compare with only using TrajOpt and only using BiRRT in several complicated environments.

1.2 Literature Review

It has been a long time that people have concentrated their attention on motion planning problems. Multiple aspects of algorithms to solve it has been proposed so

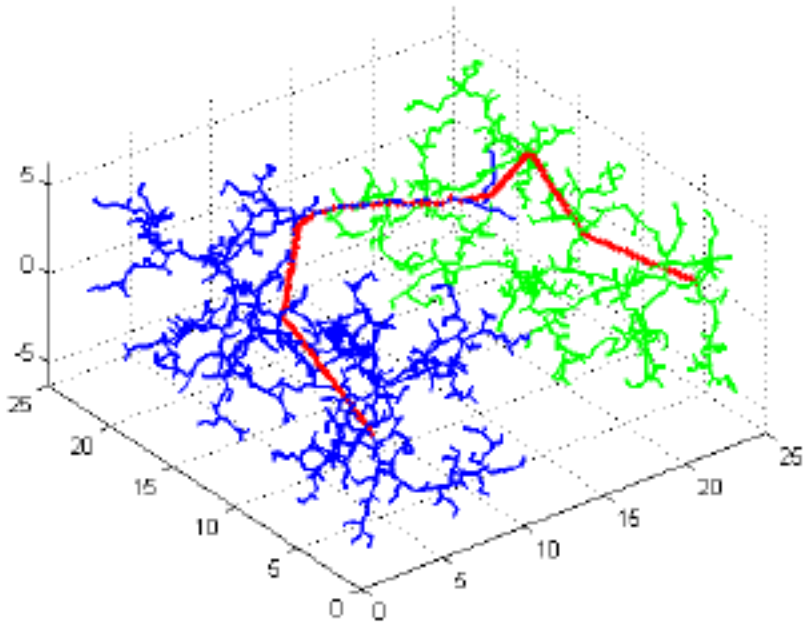

Figure 1.2: BiRRT Searching in Configuration [26]

far, including several notable search algorithms, Sampling-Based approaches, and Optimization-Based approaches.

1.2.1 Notable Search Algorithms

In 1959, there was a breakthrough in the search field. Dijkstra proposed a method called Dijkstra's algorithm[4] to find the shortest path between the start point and other points in the discrete space. Only after 9 years, in 1968 Perter Hart Nils Nilsson and Bertram Raphael presented another algorithm called A* search algorithm[11] which is an extension of Dijkstra's algorithm. In this algorithm instead of only storing the cost from the start to the current node, A* uses heuristics to guide its search, which achieves better performance. Additionally, the original D*[24] was introduced by Anthony Stentz in 1994. Actually D* is improved based on A*,

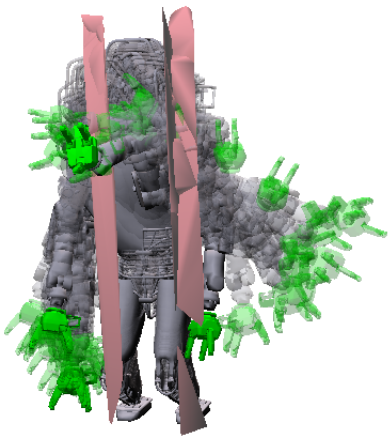

Figure 1.3: Trajectory Generated by BiRRT on Atals Robot

all the behaviours of both algorithms are the same except that the costs would dynamically change as the D* algorithm runs. Even now the original D* and its variants[25][15] have a very wide application in Robotics field for searching. All these search algorithms have comparatively good performances in the lower dimensional space which has no more than 3 dimensions. However, with the increasing the dimensions the performances of those methods would decrease dramatically. For planning in the configuration space which normally is a high-dimensional space, those classic search approaches is not adoptive.

1.2.2 Sampling-Based Approaches

Considering efficiently searching an accessing path in the configuration space, Steven LaValle in 1998 introduced a novel method called RRT[17] which would build a search tree to explore the high dimensional space. Just 2 years later a bidirectional version of RRT called BiRRT[16] was developed as described by Kuffner. This method would have comparatively good performance by exploring the space

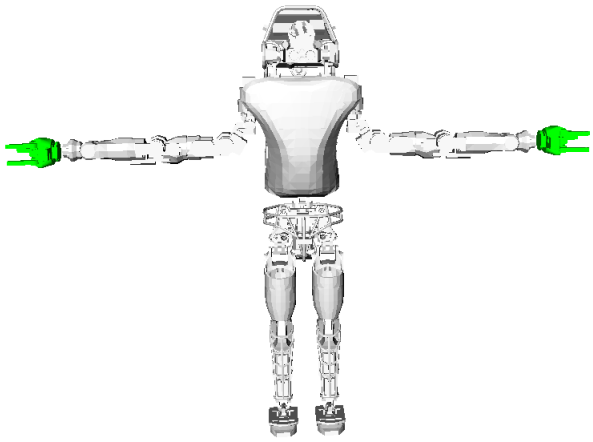

Figure 1.4: The Atals full Body Model

from the start and goal bidirectionally. Another new extension of RRT called pRRT(particle Rapidly-exploring Random Tree) was presented by Melchior in 2007. This work incorporates the uncertainty in its domain similar to how a particle filter works. In stead of extending along a straight line if we randomly increase the search tree, this extending step would be regarded as a stochastic procedure. Besides, combining stochastic optimization with sampling-based method, Jaillet introduced T-RRT(Transition-based Rapidly-exploring Random Tree)[12]. A high-quality trajectory will be generated use this approach comparing to other RRT invariants. Moreover, in 2009 Dmitry Berenson came up with a state-of-art approach called CBiRRT(Constrained Bi-directional Rapidly-Exploring Random Tree)[1]. This algorithm could handle multiple constraints while bidirectionally exploring the configuration space. Furthermore, in 2010 Karaman presented a distinguished approach called RRT*[14] which will generate optimal trajectory by using incremental sampling-based algorithm. Although those methods have a very nice property which probabilistic completeness, those approaches hardly can calculate the solutions while satisfy additional constraints including full body balance which

is essential for humanoid robot.

1.2.3 Optimization-Based Approaches

In another aspect, instead of searching in the configuration space, there are several optimization-based approaches proposed. In 2009, Ratliff introduced a novel method called CHOMP(Covariant Hamiltonian Optimization for Motion Planning)[21]. He used covariant gradient techniques to optimize some given initial guesses. In his work, it used pre-computed signed distance to check collisions. Also a new method called STMOP(Stochastic Trajectory Optimization for Motion Planning)[13]which claims that due to the stochastic nature of the algorithm the planner would not get stuck in the local minimum proposed by M. Kalakrishnan. Besides, ITMOP(Incremental trajectory optimization for real-time replanning in dynamic environments)[20] introduced by Chonhyon Park also use a stochastic trajectory optimization framework. However, it would avoid the collision of the changing environment and satisfy dynamic constraints. Furthermore, just 3 years ago John Schulman presents a sequential convex optimization approach while incorporating collision avoidance to optimize sampled initial guesses. But the performances of all those methods highly depends on the given initial guesses. Even though in some cases, those frameworks behave intelligently, in some complicated environments those cannot find the very obvious and existing solutions.

1.3 Structure of Thesis

This thesis is organized as follows:

Chapter 2 would introduce the theory behind this framework. This chapter would mainly focus on the reason why the TrajOpt would be stuck in the infeasible

local minimum and how we could use the BiRRT to prevent the TrajOpt from being trapped.

Chapter 3 would describe how I implement this new software framework, specifically for Atlas robot in details, including how we set costs and plan motions to maintain the full-body balance while satisfying constraints.

We have conducted several experiments over a wide range of environment settings. The details of each experiment's setting up would be revealed in Chapter 4. And in order to show the better performance, we will explain reasons why we set up those environments.

After acquiring data collected from the experiments described in Chapter 4, we evaluate and compare different performances among several methods in Chapter 5. But we mainly concentrate on the performance of the new software framework.

In Chapter 6 we come to a conclusion and summarize some advantages and disadvantages of this framework.

Chapter 2

Theory

This new software framework consists of two major parts as shown in the Figure 2. BiRRT in this framework serves as an initial guess generator which outputs a very jerky but feasible trajectory. And in the next phase the TrajOpt would optimize it into a smoothed and optimized trajectory. The pseudo code of this software framework is shown Figure 2.2.

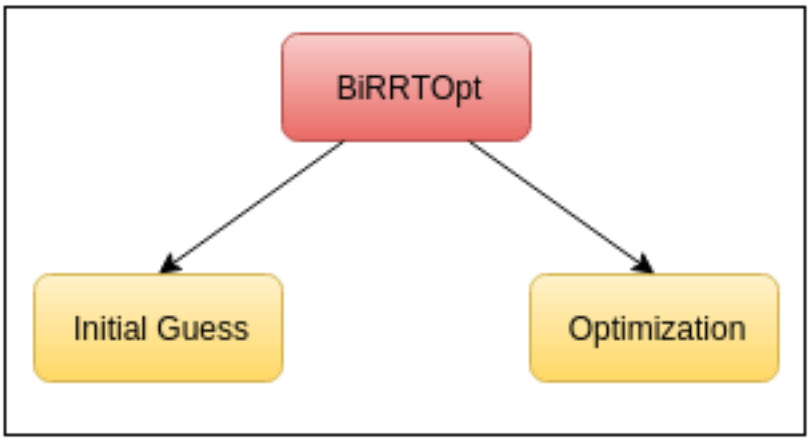

Figure 2.1: Framework of BiRRT

```
SOFTWARE FRAMEWORK (p_goal, manip)
(1) while TURE do
(2)    PREPARE_ENVIRONMENT()
(2)    q_goal ← INVERSE_KINEMATICS(p_goal)
(3)    q_init ← GET_CURRENT_STATE()
(4)    configs_arm ← BIRRT_PLANNER (q_init, q_goal)
(5)    configs_fullbody ← FULL_BODY_TRAJ (configs_arm, manip)
(6)    traj_fullbody ← OPTIMIZE_TRAJECTORY(configs_fullbody)
(7)    if accept = TRUE then
(8)       SEND_TO_CONTROLLER(traj_fullbody)
(9)    else
(10)      Continue
```

Figure 2.2: Pseudocode for This Software Framework

2.1 BiRRT: Initial Guess Generator

Admittedly, there are several sampling-based search algorithms widely applied in planning robot motion. The biggest advantage of a sampling-based search algorithm is its intrinsic property, probabilistic completeness. Considering that the TrajOpt would be trapped in the local minimum which still does not satisfy all constraints, meaning TrajOpt is not probabilistic complete, BiRRT is suitable to generate a feasible initial guess to promote TrajOpt to become probabilistic complete. The reasons why we adopt BiRRT instead of using other variants of RRT would be described in the following.

RRT is rapidly exploring toward one direction. But the BiRRT is bidirectional from the start point to the goal point, which means it is more efficient in exploring the configuration space. Even though the bias of picking up the goal point in RRT can be set very high, which sometimes would accelerate the speed of RRT's planning procedure in some simple environments, in some complex setting up the performance still cannot be compared to the BiRRT and the high bias would greatly help to get itself trapped. So in order to improve the performance of generating the initial

guess, we choose the BiRRT over the RRT. In spite of RRT* generating a very nice and optimized trajectory, there is a consideration that we would use TajOpt to optimize it in the next phase, no extra computational-power should be wasted on optimizing it in each iteration like RRT* does. CBiRRT gives individuals capabilities of setting constraints while planning. It is very useful when there is some certain rules needed to be satisfied in the environment. However, TrajOpt also provides this functionality while optimizing. And the time cost of CBiRRT cold not be less than BiRRT's, because every time a random point in the configuration space needs to be project on the constraints manifolds, which requires extra time.

BiRRT builds two search trees to search the solution in the configuration space. The pseudo code is shown in the Figure 2.3. We are able to see in the pseudo code a K which is the time of iterations would be set. However, if we consider K is infinite, meaning the algorithm would explore the whole space. This algorithm would definitely return a solution as long as the solution exits. During the exploring, a random point would be sampled in the configuration space, which prevents this method from getting trapped. Randomness is the most favorable major property of those sampling-based methods. In the original RRT algorithm, there is only one step called `EXTEND` to build the tree. However, in BiRRT we try to search from start and goal point. So not only `EXTEND` but also `CONNECT` is necessary for constructing an access. Every time we finish increasing the size of one tree by one, we swap those two trees and repeat the above operations. That is how this method search bidirectionally, which would dramatically improve the performance. `EXTEND` operation means we need to repeatedly sample new points along the line form the q_{near} to the q_{new} in the configuration space and add them into the exploring tree. The resolution of extending steps is very essential. Because if the resolution of joints in this procedure are too aggressive, the relative `step_size`s in the task space could

be huge, even though it could be small due to the different weights of joints along the arm. If the changing of the Euclidean distance, of the end effector, is larger than minimum radius of objects in the environment, which we cannot treat the trajectory continuous but discrete, it could result in being trapped again. The way in practice we avoid this situation to happen would be described in the next chapter in details. So overall, we decide to use BiRRT to generate our initial guess.

```
EXTEND (τ, q)
(1)   q_near ← NEAREST_NEIGHBOR (q, τ)
(2)   if NEW_CONFIG (q, q_near, q_new) then
(3)      τ .add_vertex (q_new)
(4)      τ. add_edge (q_near, q_new)
(5)      if q_new = q then
(6)         return REACHED
(7)      else
(8)         return ADVANCED
(9)   return TRAPPED
```

```
CONNECT (τ, q)
(1)   repeat
(2)      S ← EXTEND (τ, q)
(3)   until not (S = Advanced)
(4)   return S
```

```
BIRRT_PLANNER (q_init, q_goal)
(1)   τ_a .init (q_init) τ_b .init (q_goal)
(2)   for k = 1 to K do
(3)      q_rand ← RANDOM_CONFIG()
(4)      if not (EXTEND (τ_a, q_rand) = Trapped) then
(5)         if CONNECT(τ_b, q_new) = Reached then
(6)            return PATH(τ_a, τ_b)
(7)      SWAP(τ_a, τ_b)
(8)   return FAILURE
```

Figure 2.3: Pseudocode for BiRRT [16]

2.2 TrajOpt: Trajectory Optimizer

In the beginning, we need to make an assumption that TrajOpt applied in this work only considers kinematics problems, which does not involve any dynamic constraints. But in the wider aspect, this optimization-based method can be easily extended to find collision-free solutions while satisfying dynamics constraints. And in this phase, the TrajOpt would take the result of the initial guess generator as the guidance to find the local minimum.

2.2.1 Sequential Convex Optimization

TrajOpt is presented to show how to formulate the objective in robot motion and perform the numerical optimization. Normally robot motion planning problems can only be formulated as the following equations,

$$minimize\ f(x) \tag{2.1}$$

$$subject\ to \tag{2.2}$$

$$g_i(x) \leq 0, i = 1, 2, ..., n_{ineq} \tag{2.3}$$

$$h_i(x) = 0, i = 1, 2, ..., n_{eq} \tag{2.4}$$

where:

$f, g_i, h_i scalar functions.$

which are non-convex optimization problem. But currently there is no specific algorithm existing to solve the non-convex optimization problem. So TrajOpt tries to transform this non-convex problem into the convex problem and use existing

solver called Gurobi[10] to solve it. But this transform cannot be done in one iteration. In this reason, it repeatedly construct a convex sub-problem and ensure every time it is getting closer to the original problem. By doing so, TrajOpt would eventually find the local minimum. But there are two important ingredients of TrajOpt described as the following: (1) In each iteration, the progress cannot be made too aggressively, meaning if the step size is too big sometimes the approximation between the sub-problem and original problem cannot stay valid. (2) A method to transform constraints into cost, but in the end try to decrease the cost to zero, which meas all constrains are satisfied.

The pseudo code of TrajOpt would be shown in Figure 2.4. In the `PenaltyIteration` loop, each iteration of TrajOpt tries to increase penalty coefficient μ, if all constraints are satisfied, the solutions would be figured out. Otherwise, this progress would keep going on until the time of iterations reach the maximum times. The loop inside the `PenalytyIteration` loop is the `ConvexifyIteration` loop. In this loop we repeatedly formulate a convex sub-problem which is an improved approximation to the original problem and try to optimize it. The next loop(TrustRegionIteration) is to decide whether to increase the trust region or shrink the region depend on whether if it really makes some progress or not. Ideally, each time it would make some improvement on solving our non-convex problem.

Now we have introduced how the TrajOpt optimize and find the local minimum. The following part would explain the reason why if the TrajOpt has been given a feasible initial guess, it has the capability to generate a high-quality trajectory while satisfying all constraints. Initial guess phase would output an collision-free path as long as the solutions exist. And in the TrajOpt, the most significant constraints are self-collision and environment collisions, which exactly have been solved by BiRRT. That jerky trajectory should looks like a point on the curve where all points along

TrajOpt penalty method for sequential convex optimization

(1) **for** $PenaltyIteration = 1, 2, ...$ **do**
(2) **for** $ConvexifyIteration = 1, 2, ...$ **do**
(3) $\tilde{f}, \tilde{g}, \tilde{h} = ConvexifyProblem(f, g, h)$
(4) **for** $TrustRegionIteration = 1, 2, ...$ **do**
(5) $x \leftarrow arg_x min \tilde{f}(x) + \mu \sum_{i=1}^{n_i nq} |\tilde{g}_i(x)|^+ + \mu \sum_{i=1}^{n_{eq}} \left|\tilde{h}_i(x)\right|$
subject to trust region and linear constraints
(6) **if** $TrueImprove/ModelImprove > c$ **then**
(7) $s \leftarrow \tau^+ * s$ ▷ Expand trust region
(8) **break**
(9) **else**
(10) $s \leftarrow \tau^- * s$ ▷ Shrink trust region
(11) **if** $s < xtol$ **then**
(12) **goto** 15
(13) **if** converged according to tolerances xtol or ftol **then**
(14) **break**
(15) **if** converged satisfied to tolerance ctol **then**
(16) **break**
(17) **break**
(18) $\mu \leftarrow k * \mu$

Figure 2.4: Pseudocode for TrajOpt [23]

this part of curve should meet constraints as shown in Figure 2.2.1. Because the TrajOpt approximates the original problem to a convex problem, so along the curve, we are able to find the local minimum which is a smoothed trajectory optimized based on length of the path.

2.2.2 Discrete-time Collisions Checking

TrajOpt provides a very efficient way to check collisions in discrete-time. The collision penalty defined which needs to be satisfied in the TrajOpt is based on the translation Euclidean distance. TrajOpt uses a set of points to represent one object. So we could define the distance between two sets A, B which do not share the

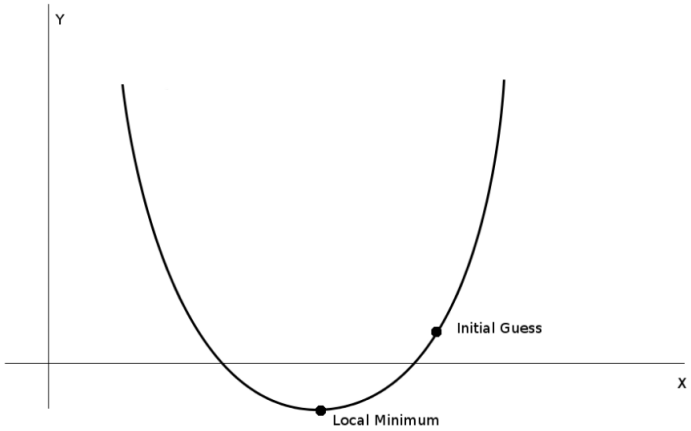

Figure 2.5: Initial Guess Guides Optimization

collision as:

$$dist(A, B) = inf\|T\| \,|(T + A) \cap B \neq \phi \tag{2.5}$$

And the distance of those two objects which are intersecting with each other is defined as:

$$penetration(A, B) = inf\|T\| \,|(T + A) \cap B = \phi \tag{2.6}$$

So The signed distance is represented as follows:

$$sd(A, B) = dist(A, B) - penetration(A, B) \tag{2.7}$$

where:

$A \subset \mathbb{R}^3$

$B \subset \mathbb{R}^3$

T = the length of the smallest translation that force two object in contact

As long as the signed distance is positive those two objects are non-colliding.

So if we go through all links of the robot and all obstacles in the environment, the collision constraints can be represented as that all signed distances should be larger than zero. But TrajOpt does some tricks on relaxing and representing self-collision and environment collisions. But those tricks would not be revealed here. Since the main idea of representing collision constraints has been introduced, if you are very interested in those tricks, you can look into [23].

2.2.3 Continuous-times Collision Free

In the last section, we have introduced the approach to formulation collision constraints, which makes sure that the the discrete movement generated for robot is collision-free. But trajectory is a serious of continuous movements, meaning it is not enough that in some certain states along the trajectory the robot is collision free. So another collision checking approach specifically for continuous-trajectory has been described here.

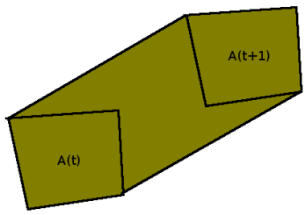

Figure 2.6: Swept-out Volume

Now we only consider two objects including a moving(only translation, not rotation) A and a stationary object B. And we define the volumes of A and B occupying the space are $A(t)$ and $B(t)$ at time t. By the definition, we can describe the volume occupied by object A is $A(t+1)$ at time $t+1$. Then the swept-out volume of moving object A can be treated as a bigger convex hull as shown in Figure 2.2.3. By doing that, we successfully transform the continuous movements into discrete states again.

The same method mentioned in the last section can be applied again here. We use the following equation to define the swept-out volume:

$$sd(convhull(A(t), A(t+1)), B) > d_{safe} + d_{arc} \tag{2.8}$$

where:

$convhull(A(t), A(t+1))$ = swept_out volume occupied
by A at continuous time from t to t+1

d_{arc} = correction(in practice we normally ignore that)

d_{safe} = the safe distance

So far we only have considered translation. However, the robot's links also undergo rotation, which result in expending the swept-out volume. Even though it is hard to estimate the exact number of the volume, we can easily calculate an upper-bound to that swept-out volume. By doing so, the method we described above is still applicable in orientation.

Chapter 3

Implementation

This new software framework mainly consists of two phases: Initial Guess and Optimization. Because the initial guess of our work is guaranteed to be feasible, so unlike the normal TrajOpt way we did in the DRC final, using multiple threads to optimize different initial guesses, we only use TrajOpt to optimize our initial guess in the Optimization phase.

3.1 Initial Guess Phase

In order to prevent the robot from toppling over by executing the trajectory generated by BiRRT, some assumptions need to be made. We assume that as long as the upper torso of the robot only moves up and down along the Z axis, meaning the center of mass usually would remain over the polygon of support, the robot is able to maintain the balance without losing too many DOF. In this case, we replace two legs of the robot with a rigid body, which forbids two legs moves along the X and Y axes. And We define the new joint called `base_pelvis` as a prismatic joint. This abstraction version of Atlas as shown in Figure 3.1 is load by OpenRAVE[2] to compute the initial guess.

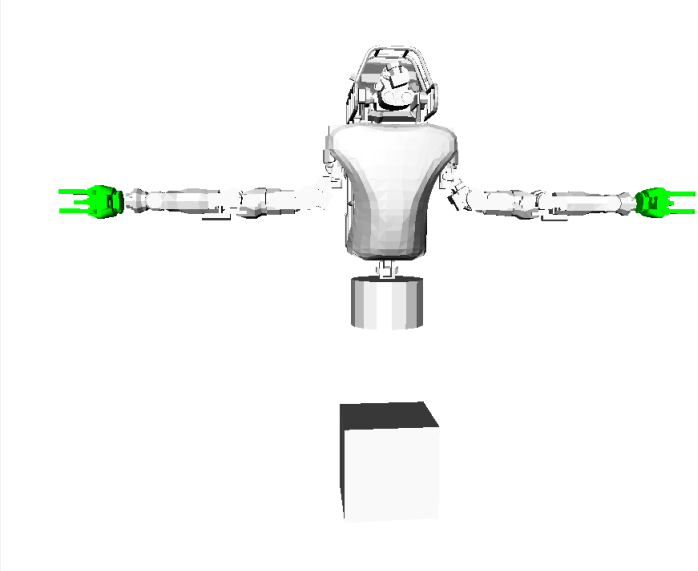

Figure 3.1: Atlas Abstraction Model

3.1.1 IK(Inverse Kinematics) computed by TrajOpt

Normally the input of this phase would be a pose including position and orientation instead of configurations at the start state. However, the BiRRT would only search the solutions in configuration space. In this reason, we map the pose into the configuration space by applying IK solver to compute all joints' values when the robot moves the end-effector to that pose. We considered to use IKFast to solve the IK solutions. However, it hardly can compute the solution when the DOF is larger than 8. In this case, there are 11 joints involved. On the other side,‘ TrajOpt can be treated as a very powerful tool to compute the IK solutions. In this procedure, we would compose a request for this problem. We set the number of step as 2, only including start state and goal state. And some constraints of `pelvis` and `utorso` and costs are added into the request. Also a series of random configurations are set for the joints as start state. We only use a naive initial guess which is a straight line to help find the IK solutions. More details of the composition of the request would

be introduced in optimization phase.

3.1.2 BiRRT Planning

After acquiring the start configuration from TrajOpt, we are able to use BiRRT to search the initial guess in the configuration space. But so far the environment has not loaded anything expect the abstraction robot, implying there is no collision at all but the robot self-collision. Thanks to the previous perception work of the WPI-CMU team, they have achieved transforming point clouds collected by MultiSense SL[22] produced by Carnegie Robotics into some polygon meshes and loading them into the environment. By doing so, we now have the capability of check collisions, which meaning the path found in the high dimensional space would be collision-free. Then we call our BiRRT planner to calculate the path for the robot.

3.1.3 Reconstruction of Whole Body's Configuration

If the solution exists, the BiRRT will definitely return a feasible solution. However, this solution is partial, which only contains the upper body's configurations. But whole body's configurations are necessary for TrajOpt's optimization and smoothing of the robot full body movement. So the next step is to reconstruct the full body's configuration. The strategy adopted here is to compute two leg's configurations while knowing the pelvis position. In spite of that we wanted to used the same procedure to compute the IK solutions, we also want to accelerate the speed of computing the IK solution. In this reason, a module called IKFast[3] is adopted here. Even though this method has some open issues including handling big decimal numbers and a limitation of DOF, it saves time on computing IK solutions. Since it create the database file and store it in the computer. Every time we do not need to spend our computational power on solving IK solutions, but just look into the

created table given end effector's pose(we define the pelvis is the end effect and the left leg is a manipulator) and find solutions. Thanks to the symmetry of two legs, we are not required to do the same to the other leg. In the end, only changing symbols of some special joints will return us the two legs' configurations. And now we have the full body's configurations by combining upper body configurations and lower body configurations together.

But note that TrajOpt has the difficulty in optimizing a trajectory which contains so many way-points. With the increasing of the way-points' scale, the time of optimization would grow exponentially. The size of way-points on the trajectory returned by BiRRT is normally over 100. But this far exceeds the ability of TrajOpt. So we have to eliminate some points. In the beginning, we just go through the trajectory and take one point every five points. However, soon we realize that by cause of different weights of each joint along the manipulator the distance in Cartesian space between every two points fluctuates a lot. In this case, sometime the TrajOpt would still get stuck in the local minimum as shown in Figure 3.1.3 even though the BiRRT return a feasible solution. We are able to see that the trajectory optimized by TrajOpt goes across the bar. For the sake of avoiding the TrajOpt from being trapped again, the step-size not in configuration space but in Cartesian space between two way-points should be small enough, which would lead the whole framework to become probabilistic complete. But here we need to apply the FK(Forward Kinematics) solver again to gain the each pose along the trajectory. Then we accumulate the distance from the start pose to the goal pose like the following equation:

$$Dist_{sum} = \sum_{Pose_{start}}^{Pose_{goal}} EuclideanDist(Pose_{new}, Pose_{old}) \tag{3.1}$$

And we assume this accumulated distance asymptotically approximates the arc of the trajectory. Since the default `num_step` has been set when the TrajOpt initialized, the `step_size` in Cartesian space can be computed by the next equation:

$$step\_size_{average} = \frac{Dist_{sum}}{num\_step_{default}} \tag{3.2}$$

As long as we can guarantee that the `step_size` is always smaller than the smallest radius among objects, the new software framework is definite probabilistic complete. By now, we have finished our initial guess phase.

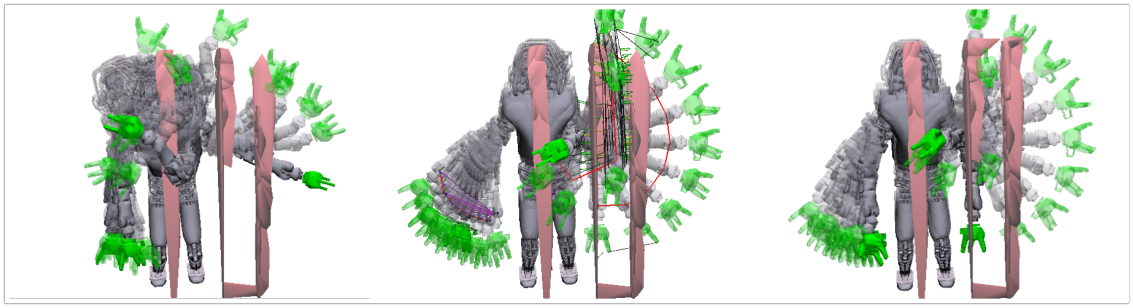

Figure 3.2: Big Changes in Pose Results in Being Trapped

3.2 Optimize Phase

In order to maintain the consistency of the environment, we choose the same environment which is adopted in the last phase. And this environment would be saved for future work. The reason will be disclosed in the experiments chapter.

First we need to construct a request. One request along with the environment would construct normally a non-convex problem which could be solved by TrajOpt algorithm. There are several aspects of information included in a request, including `basic_info`, `costs`, `constraints` and `init_info`. The `basic_info` describes some default settings including how many times we run the optimizations. In this work,

we set the iteration times as 30. The performance or the time cost of the TrajOpt would highly depend on the iteration times, which goes exponentially as the time of iterations increases. Usually this number is under 100. And the `manip` also should be defined in `basic_info`. Since we currently are trying to optimize the whole body trajectory, then we just define the whole robot as one manipulation.

The values in `costs` are set based on personal experience. We reveal our parameters in Equations (3.3) (3.4) (3.5) (3.6). And those vectors should be added into the request for helping construct the non-convex problem.

$$\begin{aligned} vel\_cost = \{&10, 10, 10, 10, \\ &4, 4, 1, 1, 1, 4, \\ &4, 4, 1, 1, 1, 4, \\ &1, 1, 1, 1, 1, 1, 0.1, \\ &1, 1, 1, 1, 1, 1, 0.1\} \end{aligned} \tag{3.3}$$

$$\begin{aligned} tau_cost = \{&0, 0, 0, 0, \\ &0, 0, 0, 0, 0, 0, \\ &0, 0, 0, 0, 0, 0, \\ &0, 0.5, 0, 0, 0, 0, \\ &0, 0.5, 0, 0, 0, 0\} \end{aligned} \tag{3.4}$$

$$
\begin{aligned}
pos_cost = \{&100.0, 1000.0, 10.0, 0.0, \\
&0.0, 0.0, 0.0, 1.5, 0.0, 0.0, \\
&0.0, 0.0, 0.0, 1.5, 0.0, 0.0, \\
&0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, \\
&0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0\}
\end{aligned} \tag{3.5}
$$

$$
\begin{aligned}
pos_vals = \{&100.0, 1000.0, 10.0, 0.0, \\
&0.0, 0.0, 0.0, 1.5, 0.0, 0.0, \\
&0.0, 0.0, 0.0, 1.5, 0.0, 0.0, \\
&0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, \\
&0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0\}
\end{aligned} \tag{3.6}
$$

There are so many different constraints can be added in the TrajOpt, including the target configuration or the target pose of the `end_effector`. Note that in this work, we only consider the translation but not orientation by `LoadGains`$(1, 1, 1, 0, 0, 0, 0, 0, 0)$. The first vector indicates the position where robot need to reach. And the second vector is for rotation, all 0's shows we do not constrain the robot to reach that orientation. The third vector is the offset to the end-effector, which now is zero. But this work can be straightforwardly extended to be applicable in constraining rotations. The example of setting the constraints is attached in the appendixA.1.

The last step is to add the initial guess into the request. In this procedure, instead of trying multiple initial guesses, we only add the initial guess which generated by BiRRT. Now we eventually combine our sampling-based method with optimization-

based method, and let the former method to guide the optimization of the latter one. Then we have composed a very complete request for this robot motion problem.

We are required to transform this request into json-formatted string. As long as the problem has been constructed, we call a function named `OptimizeProblem` which would calculate our results. In practice, we need to have an exam mechanism to check if quality of the trajectory output by this phase is good or not, which prevents the robot from behaving weirdly. Adding the human being into the control loop would dramatically decrease the percentage of failing. But usually if we let the initial guess generated by BiRRT to guild the optimization, the trajectory always looks good.

Chapter 4

Experiments

4.1 Setting Up

- Software:

  In order to run the proposed software framework, ROS(Robot Operation System)[6], OpenRAVE 0.9, TrajOpt, Python 2.7.3 [8] and Gurobi 6.0.* are required to be installed on the OCU(Operator Control Unit). There are also several necessary dependencies. We have a script to set up the environment of the computer, but due to the whole project is not public. You will need to ask for the permission to view the code.

- Operation System:

  OCU is running the Ubuntu 12.04. There are three on-board computers on the robot running 14.04. The OCU would run one ROS master, and all three on-board computers would run another ROS master and sharing it. The different operating systems running separately on the OCU and on-board computers would effect nothing in my experiments.

- Code:

  And also the code used by WPI-CMU team in the DRC final is recommended. But in the interface code between the robot and field computer, we've helped to integrate several features specially for this software framework into it.

- Robot:

  All the experiments are conducted in the Gazebo simulation and on the real second generation Atlas Robot which produced by Boston Dynamics. This robot is hydraulic with 32 DOF. In each experiments, the robot would not make steps at all. And it will just be placed in front of the experimental environment.

4.2 One Table and One Bar

4.2.1 Environment Description

In this section, one table with several items on it and one bar are placed in this environment as shown in Figure 4.2.1. Two small experiments are conducted here. In the first experiment, we set the start position P_{start} as $[0.6, -0.3, 0.5]$ and the goal position P_{goal} as $[0.4, -0.55, -0.2]$ for the right arm. Basically, the robot is going to move the right arm form a point which in under the table to a point which is over the table without colliding obstacles. After reaching the goal, the robot would repeat the above movement inversely, meaning the arm would be moved under the table again. And in the other experiment, the robot moves its left arm. The initial position P_{start} is $[0.6, 0.6, 0.6]$ which locates on the left side with the respect of the robot. However, the goal position P_{goal} as $[0.6, -0.2, 0.6]$ which we set is on the other side of the bar. This time the robot would have to avoid the bar and

the table. In those experiments, the BiRRT uses the following configuration resolution: $[0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001]$. And the length of the Atlas robot arm is around 1 meter, more specifically 94 cm. So even though the robot moves that first joint along the arm form the shoulder to the hand, the maximum changing distance of the end effector would not be more than 1cm. And in the reconstruction of the whole body configurations part, the `unit_step` in the euclidean space is as 5cm. So we would get our initial guess one point every 5cm when we go through the trajectory generated by BiRRT. Since the width of the bar in the simulation is 5cm, the initial guess we acquire would not be interrupted by that bar. We experimentally compare the performance of my new software framework with applying TrajOpt alone and BiRRT separately. By the way, the initial guess we adopt here is a straight line form the start position to the goal position. Because the start line would be the most naive and reasonable initial guess in every environment, but trying to use multiple initial guesses is difficult, which relates to environments very closely. The results acquired from those experiments would be analyzed in details in the next chapter.

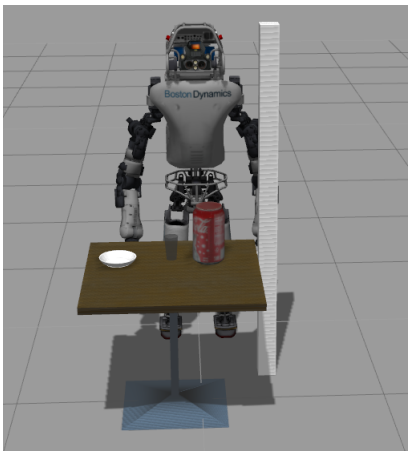

Figure 4.1: The Setting Up of One Table and One Bar Experiment

4.2.2 Reasons

The reasons why we present this experiment are described as the following. Human beings meet a table in normal life basically everyday. Individuals eat their breakfasts, lunches and dinners on tables. And they always love to play dishes on the surface, which involve a lot of arm movements. So in order to make the humanoid robot become more useful and applicable in the environment specially designed for human beings, we need to deal with such an environment in our experiment. The reason why we add a bar here is to increase the difficulty for arm manipulation. In some cases in normal our lives, the surroundings are very crowded, but our framework would give the capability to the robot of handling complex surroundings. And we also want to prove that in such environment TrajOpt is not suitable. TrajOpt might have the ability to deal with this environment except the bar. But the intrinsic property of the TrajOpt decides it is going to fail given a straight line from the start to the goal point. During the arm being moved from one side of the bar to another side of the bar, the penalty of obstacles would increase at the same time the cost to reach the goal is decreasing. By this reason, the TrajOpt would get sucked in the local minimum. The trajectory generated by TrajOpt alone would be interrupted by the bar along the movement.

4.3 Two-bars Experiment

4.3.1 Environment Description

In this experiment, we put two standing bars in front of the robot. Those two bars are separately placed in $[0.65, 0.37, 0.50]$ and $[0.67, 0.00, 0.50]$ in the world frame of the Gazebo as shown in Figure 4.3.1. And the goal position is set to $[0.70, 0.00, 0.50]$,

which means the robot is going to insert the left arm between those two bars. Once the robot have inserted its left arm between two bars, the next step is to take it out and lift the left arm up. We also use $[0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.001, 0.0$ as our configuration resolution to ensure that the the trajectory generated by BiRRt can be treated as a continuous path, which guarantee its feasibility. In this experiment, the same `unit_step` is adapted here.

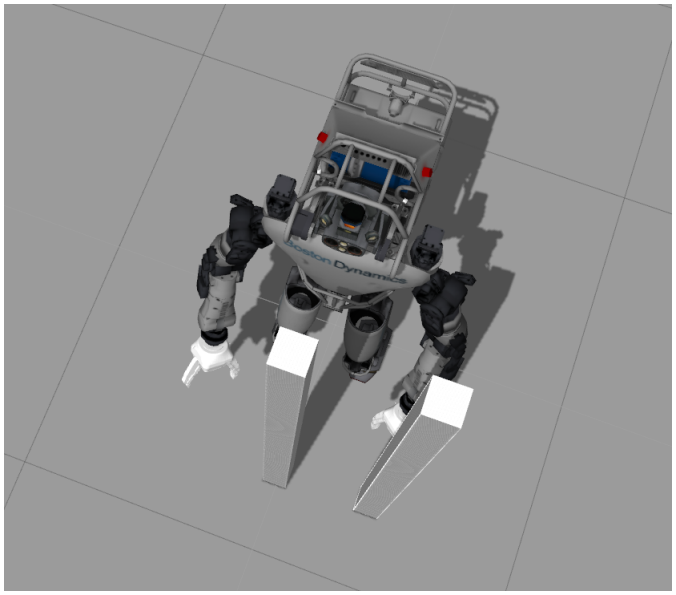

Figure 4.2: The Setting Up of Two-bars Experiment

4.3.2 Reasons

The birth of the Atlas is in the wake of the Fukushima nuclear disaster. In some unsuitable environments the robot would not face spacious surroundings to stretch arms at will but a very crowded space. In those potential rescue actions, the robot need to insert its arm into a gap and pass some food to people whose life is in danger. So we set up a gap for robot simulation.

And in this experiment, moving the arm into a position which locates between

two bars is probably very difficult for TrajOpt, because the goal is between two bars, but TrajOpt would treat those two bars as collisions which would push the arm away, which probably would dramatically decrease the success rate. But in the other side, an feasible initial guess would guild the optimization and find the local minimum in a feasible range. This experiment would show that how well our software framework can behave.

4.3.3 Perception Issue

Due to some constraints in our Atlas perception code, sometimes there are some issues during the experiments. For example, in this two-bars experiment or three -bars experiment after the robot moving the left arm between those two bars, the robot cannot see the part of left bar even though it is not moved and exists as shown in Figure 4.3.3. So in this case, when the BiRRT try to plan the trajectory, it sometimes would escape from the miss part. And since TrajOpt also is not able to see that, this initial guess would be considered as a feasible one and try to optimize it, which would result in the robot being toppled over. In order to take care of that, we choose to manually draw two bars which locates in the same positions in the OpenRAVE, instead of loading the point cloud into the environment. By doing that, the BiRRT is not able to escape through the missing part again. You might notice that in the description part the goal position is almost the same as the position where one bar locates. The reason is that there is always some shifting in loading transformed point clouds. Where the two pieces of meshed transformed by the point could is not the original coordinates anymore.

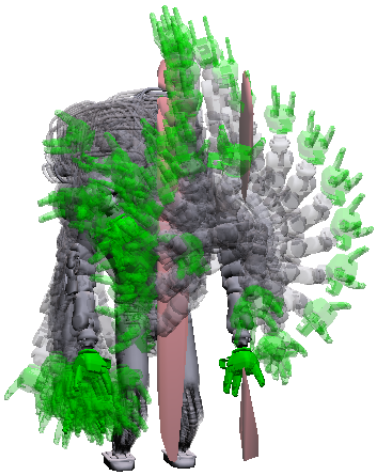

Figure 4.3: The Part of Left Bar is Missing

4.4 Three-Bars Experiment

4.4.1 Environment Description

Compared to the last experiment, the only difference between those two experiments is that another standing bar is placed next to the left bar with the respect to the robot. The third bar's position is $[0.65, 0.7, 0.79]$ in the world frame. The robot would act the same two movements. In the beginning, the robot moves the left arm from the initial position to $[0.7, 0, 0.6]$ whose projection on Y axis locates between two bars. Then the robot lift the arm to the same position. But in the experiment, we change the `step_size` to 15cm, which is around 3 times to the original one. Because the bigger `step_size` is, the less waypoints along the trajectory are taken for initial, which would greatly accelerate the speed of optimization. But in the environment, the minimum width of the object is the bar, which is 5cm.

4.4.2 Reasons

We have the following considerations to put another bar and pay attention to add another bar here. Add one more bar into the environment would increase the difficulty of finding the solution, basically your explorable configuration space is shrunk by more constraints. But our work can handle this situation very well. We notice that the `step_size` is increased, and we know that the speed of optimization highly depends on the size of the initial guess, we want to compare the performance between last experiment and this experiment. We want to show that by adjust some parameters of algorithm, our work behave really well while dealing with this environment.

Chapter 5

Analysis

We repeat all movements in each environment by using our BiRRTOpt, BiRRT and TrajOpt 10 times. Due the the page limitations, all details of results would be attached in Appendix.

5.1 One Table and One Bar

In this experiment, we run two small experiments including left movements and right movements. All the performances of BiRRTOpt, TrajOpt and BiRRT would be separately evaluated in both two small experiments,

5.1.1 Right Hand

In this right-hand experiment, we collect our data from running 10 times of BiRRTOpt for each movement, and compute the average time of each step. How much percentage of each sub-procedure occupying the whole time are shown in Figure5.1.1.

Comparing average computational time of solving IK with BiRRT, this Figure 5.1.1 indicates that it takes longer time to compute the IK solution, which is

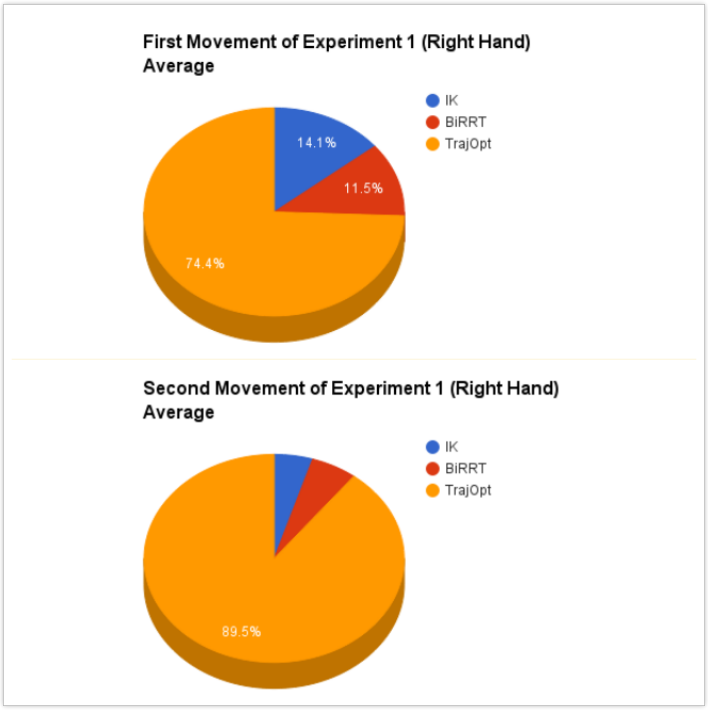

Figure 5.1: Percentage of Steps Occupying in Experiment One(Right Hand)

understandable. Because we use TrajOpt in our BiRRTOpt to compute the IK solution. Even though initially the number of steps has been set to 2, the straight line between default start point and gaol point would go across some obstacles, which means another random point needs to be sampled to construct the straight line for computing the IK solution until that straight line does not go across some obstacles. And also even in the simplest environment, the time of optimization normally would be a little bit longer than BiRRT's. So in this setting, it is ordinary that computing IK solution would take more time. According to the Figure 5.1.1we are able to see that BiRRT only occupies a small portion of the whole time to calculate the solution. Since when we are trying to compute the initial guess by sampling a collision-free random point in the configuration space, due to the simplicity of the environment, there is a very high possibility that by applying sampled random configurations on

the robot, the right arm's projection on Z axis with respect of the world frame is higher than table's height, which leads to that most of the time there is only two iterations needed, which are sampling a random point and connecting with the goal point. However, on the other side, the Figure 5.1.1 reveals that TrajOpt spends most of time on optimizing the initial guess on this problem. The time cost of the optimization procedure highly depends on the size of the problem, meaning also the size of the initial guess. And since we set our `step_size` as 5cm, which results in that the size of the initial guess is around 30. So that is the major reason why this procedure takes so long time.

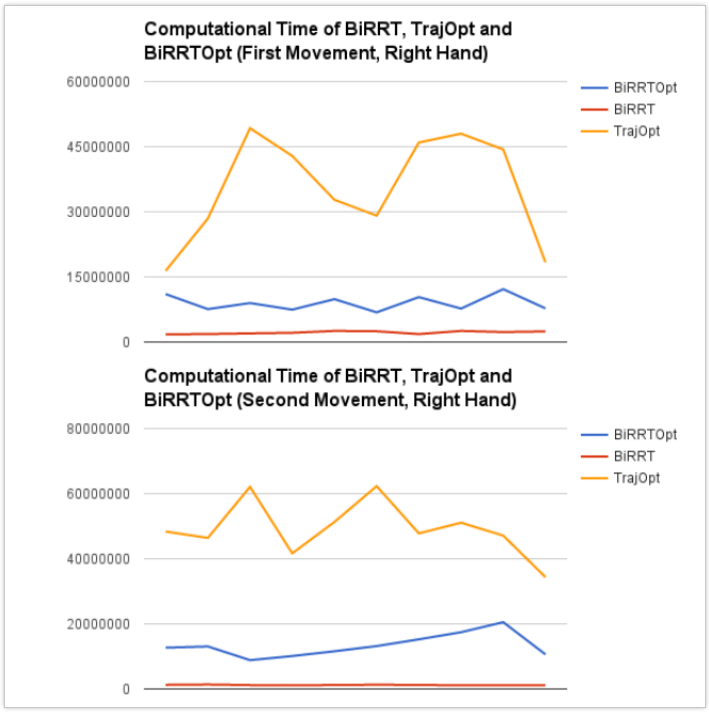

Figure 5.2: Computational Time of BiRRT, TrajOpt and BiRRT in Experiment One(Right Hand)

We compare the performance of BiRRt, TrajOpt and BiRRTOpt separately and show the results in Figure 5.1.1. When we only use BiRRT to compute the trajectory for both movements, thanks to the intrinsic probabilistic completeness, from

the resultsA.2A.2 this method is always able to find the solution as long as the solution exists in the configuration space. Comparing to the line of the TrajOpt, the line of BiRRT looks very flat. And also we are able to see that the average time cost of BiRRT is much less then TrajOpt. Those two mean that applying BiRRT alone can efficiently find the solutions in such naive environment with a very stable performance. Thanks to the TrajOpt viewer, it gives us the visualization of how the trajectory looks like if it is going to be executed on the robot since we draw our transparent trajectory in the viewer. We define the `unit_step` very small in this case, so those discrete movements seems in the viewer would not result in robot toppled over. However, when we are using TrajOpt alone to complete those two actions, the success rates change. In the first experiment in which the robot moves its right arm to a point which is over the table, the 70% success rate is still acceptable. TrajOpt has the capability of twisting the initial guess which is given as a straight line while avoiding the table and satisfying other constraints. So in the former experiment, the TrajOpt most of time is able to find the solution. But in the latter movement, the success rate drops dramatically. It only succeed once out 10 times, which barely can claim that it can move the right arm to the gal point. Through visualization of the optimization procedure, we find the reason why it cannot handle this situation. This target point seems very close to the corner of the table. So in this case, while TrajOpt making effort to reach the goal, the table seems would try to push the arm toward the inverse direction. Even though TrajOpt has the ability of twisting initial guess, which is the reason why it only succeeds once, but the initial guess is not twisted enough to be feasible, which promotes that the rest of experiments fail. The Figure 5.1.1 reveals that by using the default size of TrajOpt, which is 30, in each movement would take comparatively longer time to calculate the result, even though the time fluctuates a lot. The reason why in some experiments

it would take longer time than the rest is that in those experiments the TrajOpt gets trapped and iterates the optimization procedure until it reaches the maximum iteration times. But the on the other side, if it find the local minimum it would exit the loop. That's why the Figure 5.1.1 looks like this. Our work BiRRTOpt's line is between those two lines, but the cost time is further less than TrajOpt's and comparatively close to BiRRT's. Also the line of BiRRTOpt seems very flat, which means the performance is stable. And similar to the BiRRT, the success rate is always 100% in each movement. Because the BiRRT has already provided the optimizer with an infeasible initial guess which would help the optimization easily find an accepted local minimum. By doing that, it would helpfully guarantee the success rate and increase the performance.

Overall, our work successfully deals with this situation under an acceptable time, and since add BiRRT into generating the initial guess phase only take a little, it is reasonable to add it to make the whole work become probabilistic complete.

5.1.2 Left Hand

In the left-hand experiment, the robot would be required to move the left arm from one side of the bar to another side, we fist analyse the data collected while using the BiRRTOpt and compare it with other two methods. The average cost time of finding the solution through our BiRRTOpt is shown in Figure 5.1.2.

The Figure 5.1.2 indicates that there is a obvious changing of percentage between two movements while adopting BiRRTOpt. In the first movement, the percentage of the average time spent on calculating the IK solution is higher than that of BiRRT. However, in the second movement the situation is opposite. BiRRTOpt spends more time on planning the initial guess than calculating the IK solution. In both movements the IK solution is computed by TrajOpt. However, in the first

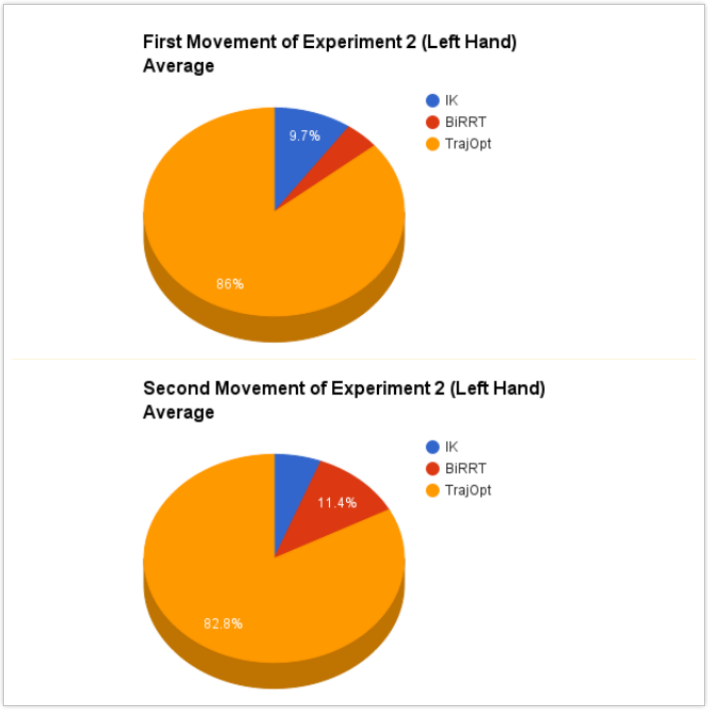

Figure 5.3: Percentage of Steps Occupying in Experiment One(Left Hand)

movement, BiRRT planner does not need to plan in such a naive environment, since from the start point the planner is able to see the goal point, which means it can directly connect the goal by extending small steps along the line between start and goal point. Compared to the fist experiment, the straight line from start point to the goal point in the second movement would traverse across the bar, then the planner would have to try exploring the configuration space and building search trees to find a access path, which takes time. Also the Figure 5.1.2 shows that the TrajOpt occupies the most time of this computation while optimization the initial guess, which is a very similar phenomenon happening in the last small experiment. Moreover, the success rate of BiRRTOpt applied in this experiment is 100% as shown in Table A.2A.2.

Now let's change our focus from the percentage of our BiRRTOpt to time cost

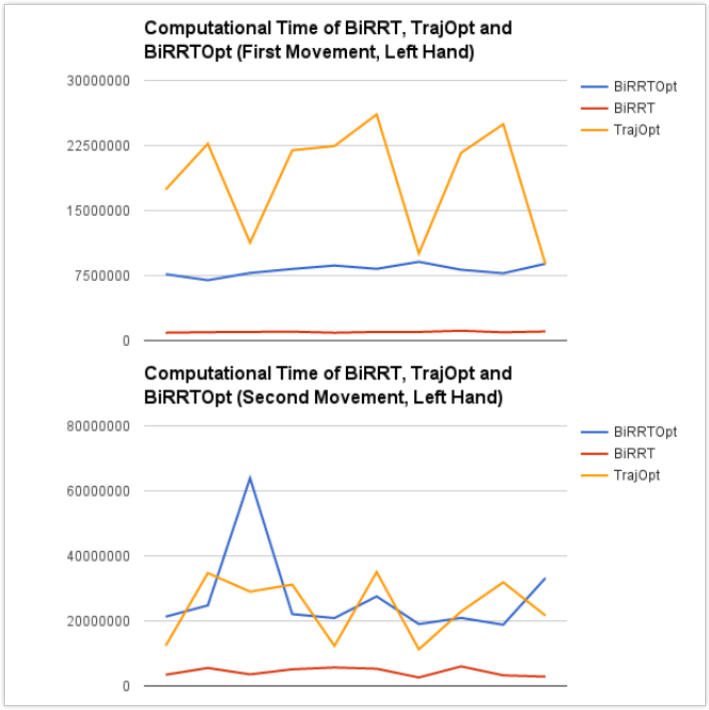

Figure 5.4: Computational Time of BiRRT, TrajOpt and BiRRT in Experiment One(Left Hand)

of those three approach, including BiRRT, TrajOpt and BiRRTOpt. In the Figure 5.1.2, we are able to see that in both charts the lowest line is always BiRRT, which means it takes the least time to compute the trajectory. On the ohter side, the TrajOpt would require much more time to compute the trajectory compared to other two approaches. Moreover, a very interesting phenomenon happens during this this experiment, which cannot be shown through those both chats. The robot cannot move the left arm from one side of the standing bar to another side by only using TrajOpt given a straight line initial guess. In such environment, the TrajOpt fails 10 times out 10 running shown as Table A.2. The success rate of that movement is 0%. Also form the Figure 5.1.2, the time spent on the TrajOpt normally would be several times to the time of the BiRRT, and also that fluctuate a lot. In this case, the TrajOpt not only requires much longer time but also fails every time

calculate the solution. We can claim that TrajOpt cannot handle it at all. On the other hand, in same environment according to the Figure 5.1.2 the performance of the BiRRTOpt in the first movement is faster than TrajOpt, and in the second experiment it looks similar to the TrajOpt. Here the performance only are considered as time cost. However, the reason why in this environment the BiRRTOpt overrides the TrajOpt is that compared to the TrajOpt the BiRRTOpt has much higher success rate see Table A.2 A.2 A.2 A.2. The success rate of BiRRTOpt is 100%. Additionally, while we compare BiRRT and BiRRTOpt, we separately draw both trajectories in the viewer. Since the BiRRT needs to plan in the second movement, the trajectory would become crazy. However, we would apply this trajectory on a humanoid robot, so that jerky trajectory is not suitable.

In general, in this experiment BiRRTOpt needs equivalent time to compute the trajectory compared to TrajOpt. But the it can deal with much complicated environment. On the other side, even though BiRRT also has the ability of handling hard problem, the trajectory of BiRRT is not as high-quality as that of BiRRTOpt. In both computations, BiRRTOpt has a better performance than others in this setting.

5.2 Two Bars

In this two-bar experiment, the left arm of the robot needs to be inserted into a position which locates between two standing bars. The data as shown in Table A.2 A.2 A.2 A.2 A.2 A.2is collected from 10 times running of BiRRT, TrajOpt and BiRRTOpt. We will analyze the processed data and compare those three methods synthetically.

The Figure 5.2 shows that the each step in BiRRTOpt occupies the whole time.

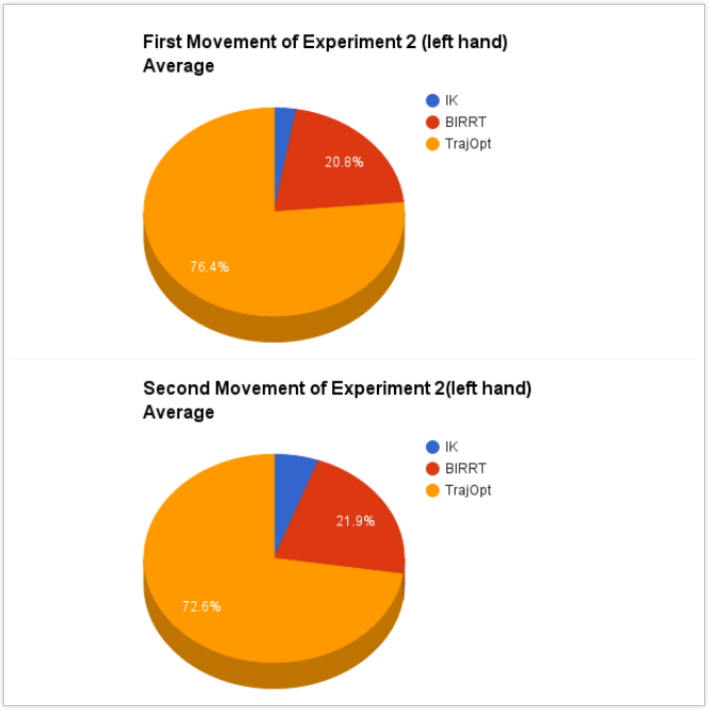

Figure 5.5: Percentage of Steps Occupying in Experiment Two

We are able to see that the percentage of BiRRT increase a lot. The major reason of that increasing of is that the environment used in this experiment compared to the last experiment is more complicated. Not matter in the first movement or the second the movement, we can image that the part of the trajectory which are still between two standing bars are very deterministic. Because the arm needs to be moves comparatively parallel to the bars, so robot either move from downside to that position through minimum distance or draw a huge circle moving from upside while avoiding the standing bars and insert the left into bars. That means leads to that the BiRRT probably needs to plan through a narrow passage to find the solution. And by sampling to go through a narrow passage in the configuration space is very hard, which requires more iteration times to find that intial guess. However, even though the time of planning the initial guess increases a lot, the time

cost compared to TrajOpt is still minor occupation. But according to the results of Table A.2 A.2our BiRRTOpt can easily deal with this environment. It does not fail at all in both 10 times running. Note that in the Figure 5.2, the 10th experiment of the first movement takes much longer time than the time which it normally takes. But this only happens once out of 10 times. In the analysis, we treat this time an unusual case.

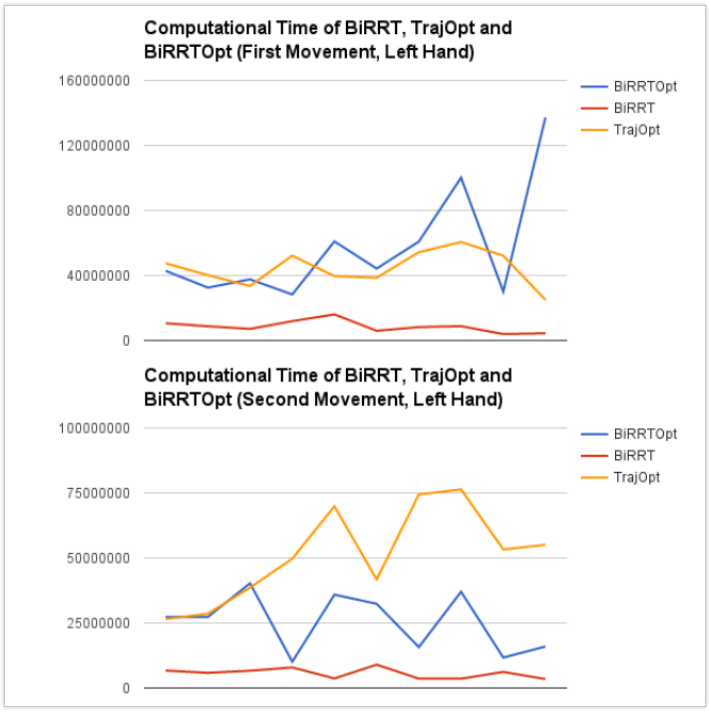

Figure 5.6: Computational Time of BiRRT, TrajOpt and BiRRT in Experiment Two

For this environment, we focus on the first movement which compared to the second one is more challenging. As shown in Figure 5.2 We are able see that in first chats even though the lines of TrajOpt and BiRRTOpt fluctuate, however, generally those two lines matches. This matching also means that TrajOpt and BiRRTOpt would require equivalent time to compute the trajectory in such a complicated environment. Also form that we know BiRRT servers as an initial guess generator

only take a little time to plan. But while TrjaOpt trying to optimize the initial guess, with or without a nice initial guess in this environment it still require a lot of time for optimization. Because, while optimizing the part trajectory which locates between the bars, in spite of the new efficient way to do the collision checking, it is still computational costly. However, in the second chart the time of BiRRTOpt and TrajOpt differ. Even though in the beginning, the line of TrajOpt seems matching the BiRRTOpt, which is because of it's soon failing, in the latter runs the time of TrajOpt becomes several times to BiRRTOpt. By the way, in the aspect of the success rate, the 100% success rate of BiRRTOpt overwhelm the 60% success rate of TrajOpt'. In this comparison, we claim BiRRTOpt is more competitive than TrajOpt. The line of BiRRT also seems the lowest in both charts in this experiment. However, in this experiment the trajectory becomes more crazy, which the trajectory generated by BiRRT would definitely let the robot toppled over. And for planning the robot motion, the balance needs to be place in the first priority, so BiRROpt in this aspect is better than BiRRT.

So in this two-bar environment, the BiRRTOpt takes comparatively less time but it guarantee the success rate for planning the robot motion.

5.3 Three Bars

Compared to the last two-bar experiment in which two bars are stranding in front of the ground, in this three-bars experiment there are three bars. The robot needs to insert its left arm into a gap which is between the rightmost bar and the middle bar. We run BiRRT, TrajOpt and BiRRTOpt separately and collect all the data which is processed and attached in Table A.2 A.2 A.2 A.2 A.2 A.2. Besides the difference we have motioned above, there is another difference between experiment

and two-bar experiment. In this experiment, we set the `step_size` as 15 cm which is larger than the minimum width of bars there. But increase the step size would result in decrease of the size of the intial guess, which would accelerate the speed of the optimizaition.

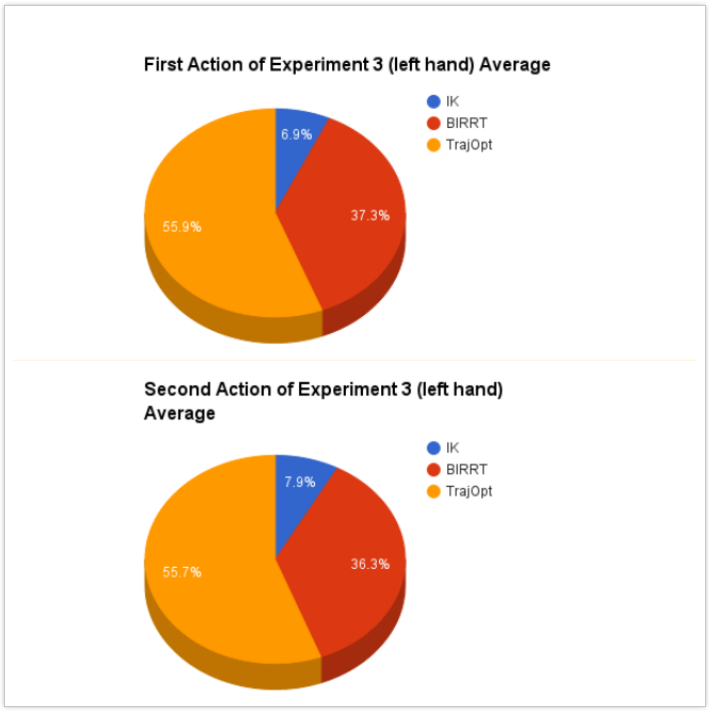

Figure 5.7: Percentage of Steps Occupying in Experiment Three

From the Figure 5.3, we are able to see that the percentage of BiRRT occupying the whole time increases again in both charts compared to the two-bar experiment. In the two-bar experiment, the percentage is around 20%. However, in this experiment this percentage increase to 30%. The reason is that a another standing has has been placed next to the currently middle bar. By doing that, the collision-free spaced has been further compressed. So in this case, the BiRRT would require more time to plan an access and collision-free path in the configuration space. One thing we need to notice is that in the first movement according to the result shown in

TableA.2 it fails twice out of 10 times. Theoretically, initial guess is generated by BiRRT, which would guarantee the probabilistic complete. There are two major reasons causing this phenomenon happening. In our 6th experiment of the first movement, the initial guess looks like a huge circle. However, due to the too aggressive step size we set, the initial guess cannot be treated as a continuous trajectory, which would make the TrajOpt suffer from getting stuck between two left bars again. Another reason causes the failure of 4th experiment. As we mentioned before in the last chapter, we noticed that there is some issue with the Atlas perception, we places three bars in the OpenRAVE instead of loading the point clouds. However, there is some mismatching existing, which makes the 4th experiment fail. When the initial guess generated by BiRRT, that is not strictly collision free if we optimize it loading the point clouds. When the TrajOpt optimize it, it tries to twist the trajectory. There is a possibility that the one point on the initial guess is pushed into the wrong direction, which results in that point getting stuck. But overall the 80% success rate of BiRRTOpt is much better than 0% of TrajOpt.

The Figure 5.3 shows that the time of 10 runs of each movement. In both charts TrajOpt normally use the most time, and BiRRT used the least time, and line of the BiRRTOpt is in the middle, even though there is one time BiRRTOpt spend more time than TrajOpt. This information tells us that the by adding the BiRRT which servers as initial guess generator would help the performance of the TrajOpt, since TrajOpt has no chance to succeed in such complicated environment. That feasible initial guess guides optimization and decrease the time to an acceptable one.

In this experiment, we are able to see that there is always a balance between success rate and performance. If you increase the step size, the success rate would decrease.

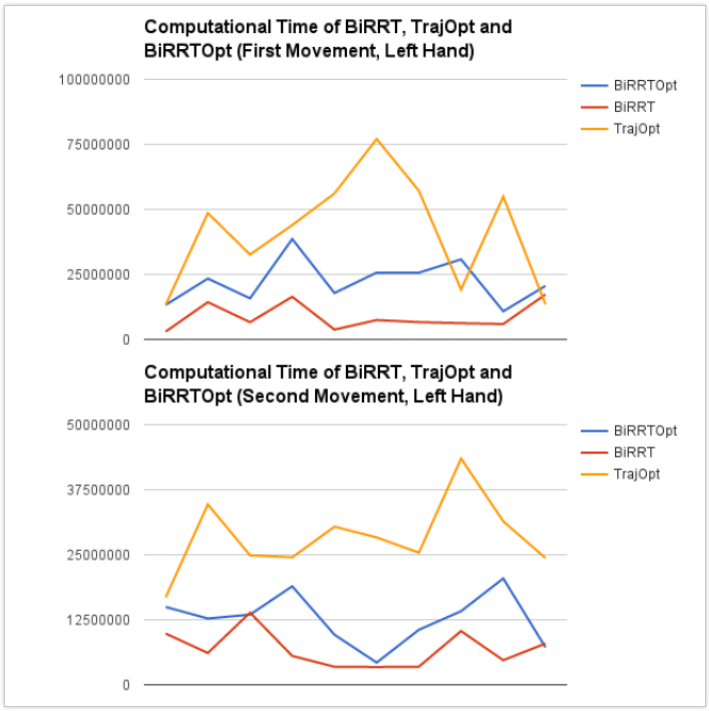

Figure 5.8: Computational Time of BiRRT, TrajOpt and BiRRT in Experiment Three

Chapter 6

Conclusion and Future Work

6.0.1 Conclusion

While WPI-CMU team's DRC code has been fully developed and TrajOpt has been applied on the real Atlas robot, there is still a lot of effort to be made. We presented a new software framework combing sampling-based method with optimization-based method for planning the robot motion. The sampling-based methods are concisely introduced and the optimization-based methods are covered with details. Then the theory behind this software framework is thoroughly explained. In that section, we described how the BiRRT explores and builds the tress in the configuration space. And the reason why it would not get trapped is explained in details. Then we woud show how the TrajOpt optimizes the trajectory given some environment information. We revealed that introducing the BiRRT into the TrajOpt's initial guess phase, would result in this improved framework becoming probabilistic complete.

Several assumptions has been made in this thesis. But those assumptions would not effect the performance of this framework, which just handles with some issues we met during the implementation on the real robot. This framework was applied not also in the Gazebo[7] simulation but also on the actual Atlas robot. According to

the data collected in the experiments, this method has the capability of generating a smoothed, optimized, and executable trajectory compared with the original BiRRT method. Also in the aspect of prevent from being stuck in the infeasible local minimum, the improved framework overwhelms the original TrajOpt.

6.0.2 Future Work

The ultimate goal of this thesis work is to plan the robot motion in the cluttered environments while satisfying some constraints, including self-constraints and other environment constraints. However, the perception subscribed by the robot sometimes does not match the actual environment because of low-quality filtering. Better perception would definitely help the performance of this framework.

Also currently this method cannot plan the motion in real time because it takes time to figure out the solution. In the future, we hope to increase the efficiency of the algorithm, which probably could lead it to plan real-time motion. Also in order to accelerate the speed, we can try to run it on the cloud where we assume there would be unlimited computational power.

This work lays some foundation to develop fully autonomous manipulation on the real Atlas robot considering the constraint information.

6.1 Other Thoughts

In this section, we would introduce other thoughts while we are doing our research and implementing our method.

There is a thought which has not been studied deeply and implemented in practice due to limited time. We are thinking about how instead of searching the solution in the configuration space, we can search not a path but a feasible range in the Carte-

sian space. In this case, because the searching happens in the lower dimensional space, we are able to use some notable search algorithms such as A* or D* to find that range. Then we can use that range to guide the optimization of the TrajOpt.

Appendix A

More to say

A.1 Setting TrajOpt Constraints

```
AddRequestHead(request);
AddCostHead(request, vel_cost);
AddContinueCollisionCost(request,
                         collision_cost,
                         dist_pen,
                         0,
                         num_step-1);
AddDiscontinueCollisionCost(request,
                            collision_cost,
                            dist_pen,
                            0,
                            num_step-1);
AddPoseCostorConstraint(request,
                        "utorso",
                        {0,0,0},
                        l_foot_quat,
                        {0,0,0},
```

```
                        {500,500,0},
                        1,
                        num_step-1,
                        {0,0,0});
    AddPoseCostorConstraint(request,
                            "pelvis",{0,0,0.75},
                            l_foot_quat,
                            {0,0,10},
                            {10,10,10},
                            1,num_step-1,
                            {0,0,0});
    AddJointPositionCostorConstraint(request,
                                     pos_cost,
                                     pos_vals);
                                     AddCostEnd(request);
                                     AddConstraintHead(request);
    AddPoseCostorConstraint(request,
                            "l_foot",
                            l_foot_xyz,
                            l_foot_quat,
                            {10,10,10},
                            {10,10,10},
                            1,
                            num_step-1,
                            {0,0,0});
    AddPoseCostorConstraint(request,
                            "r_foot",
                            r_foot_xyz,
                            r_foot_quat,
                            {10,10,10},
                            {10,10,10},
```

```
                            1,
                            num_step-1,
                            {0,0,0});
    AddPoseCostorConstraint(request,
                            hand_str,
                            xyz_target,
                            quat_target,
                            pos_gains,
                            rot_gains,
                            num_step-1,
                            num_step-1,
                            hand_offset);
    AddPoseCostorConstraint(request,
                            other_hand_str,
                            other_hand_xyz_target,
                            other_hand_quat_target,
                            other_hand_pos_gains,
                            other_hand_rot_gains,
                            1,
                            num_step-1,
                            other_hand_offset);
    AddConstraintEnd(request,request_traj);
```

A.2 Experiments Results

Description	Total	Average
IK	12666227	1266622.7
BiRRT	10303629	1030362.9
TrajOpt	66784620	6678462
IK + BiRRT + TrajOpt	89754476	8975447.6
BiRRT inside(second)	7.475	0.7475
Initial Guess Size	245	24.5
Success	10	100%

Table A.1: One Table and One Bar(BiRRT+TrajOpt), First Movement with Right Hand

Description	Total	Average
IK	6387688	638768.8
BiRRT	7603492	760349.2
TrajOpt	119268032	11926803.2
IK + BiRRT+TrajOpt	133259212	13325921.2
BiRRT inside(second)	4.388	0.4388
Initial Guess Size	330	33
Success	10	100%

Table A.2: One Table and One Bar(BiRRT+TrajOpt), Second Movement with Right Hand

Description	Total	Average
IK	7922193	792219.3
BiRRT	3493162	349316.2
TrajOpt	70026172	7002617.2
IK + BiRRT + TrajOpt	81441527	8144152.7
BiRRT inside(second)	2.196	0.2196
Initial Guess Size	180	18
Success	10	100%

Table A.3: One Table and One Bar(BiRRT+TrajOpt), First Movement with Left Hand

Description	Total	Average
IK	15944429	1594442.9
BiRRT	31039694	3103969.4
TrajOpt	225503947	22550394.7
IK + BiRRT + TrajOpt	272488070	272488070
BiRRT inside(second)	24.841001	2.4841001
Initial Guess Size	481	48.1
Success	10	100%

Table A.4: One Table and One Bar(BiRRT+TrajOpt), Second Movement with Left Hand

Description	Total	Average
IK	12046212	1204621.2
BiRRT	9815625	981562.5
IK + BiRRT	21861837	2186183.7
BiRRT inside(second)	6.539	0.6539
Path Size	1556	155.6
Success	10	100%

Table A.5: One Table and One Bar(BiRRT), First Movement with Right Hand

BiRRT	Totoal	Average
IK	5662747	566274.7
BiRRT	6399158	639915.8
IK + BiRRT	12061905	1206190.5
BiRRT inside(second)	3.811	0.3811
Path Size	1258	125.8
Success	10	100%

Table A.6: One Table and One Bar(BiRRT), Second Movement with Right Hand

Description	Totoal	Average
TrajOpt time	355548849	35554884.9
Size	300	30
Success	7	70%

Table A.7: One Table and One Bar(TrajOpt), First Movement with Right Hand

Description	Totoal	Average
TrajOpt time	492454645	49245464.5
Size	300	30
Success	1	10%

Table A.8: One Table and One Bar(TrajOpt), Second Movement with Right Hand

BiRRT	**Totoal**	**Average**
IK	6594427	659442.7
BiRRT	3303986	330398.6
IK + BiRRT	9898413	989841.3
BiRRT inside(second)	1.958	0.1958
Path Size	640	64
Success	10	100%

Table A.9: One Table and One Bar(BiRRT), First Movement with Left Hand

Description	**Totoal**	**Average**
IK	13089151	1308915.1
BiRRT	30222313	3022231.3
IK + BiRRT	43311464	4331146.4
BiRRT inside(second)	22.806	2.2806
Path Size	3274	327.4
Success	10	100%

Table A.10: One Table and One Bar(BiRRT), Second Movement with Left Hand

Description	**Totoal**	**Average**
TrajOpt time	187355853	18735585.3
Size	300	30
Success	10	100%

Table A.11: One Table and One Bar(TrajOpt), First Movement with Left Hand

Description	**Totoal**	**Average**
TrajOpt time	242264198	24226419.8
Size	300	30
Success	0	0%

Table A.12: One Table and One Bar(TrajOpt), Second Movement with Left Hand

Description	**Total**	**Average**
IK	15840645	1584064.5
BiRRT	119594324	11959432.4
TrajOpt	439228186	43922818.6
IK + BiRRT + TrajOpt	574663155	57466315.5
BiRRT inside(second)	111.903005	11.1903005
Initial Guess Size	494	49.4
Success	10	100.00%

Table A.13: Two Bars(BiRRT+TrajOpt), First Movement with Left Hand

Description	Totoal	Average
IK	14020009	1402000.9
BiRRT	55478479	5547847.9
TrajOpt	184055558	18405555.8
IK + BiRRT + TrajOpt	253554046	25355404.6
BiRRT inside(second)	47.706001	4.7706001
Initial Guess Size	414	41.4
Success	10	100%

Table A.14: Two Bars(BiRRT+TrajOpt), Second Movement with Left Hand

Description	Totoal	Average
IK	12871661	1287166.1
BiRRT	72873241	7287324.1
IK + BiRRT	85744902	8574490.2
BiRRT inside(second)	65.696003	6.5696003
Path Size	3153	315.3
Success	10	100%

Table A.15: Two Bars(BiRRT), First Movement with Left Hand

Description	Totoal	Average
IK	8106788	810678.8
BiRRT	48435952	4843595.2
IK+BIRRT	56542740	5654274
Path Size	3756	375.6
Success	10	100%
BiRRT inside(second)	39.431	3.9431

Table A.16: Two Bars(BiRRT), Second Movement with Left Hand

Description	Totoal	Average
TrajOpt time	443648889	44364888.9
Success	6	60%
Size	300	30

Table A.17: Two Bars(TrajOpt), First Movement with Left Hand

Description	Totoal	Average
TrajOpt time	514166005	51416600.5
Success	3	30%
Size	300	30

Table A.18: Two Bars(TrajOpt), Second Movement with Left Hand

First movement	Totoal	Average
IK	15251086	1525108.6
BiRRT	82882920	8288292
TrajOpt	124244704	12424470.4
IK + BiRRT + TrajOpt	222378710	22237871
BiRRT inside(second)	74.360001	7.4360001
Size	178	17.8
Success	8	80.00%

Table A.19: Three Bars(BiRRT+TrajOpt), First Movement with Left Hand

Description	Totoal	Average
IK	10047569	1004756.9
BiRRT	45923421	4592342.1
TrajOpt	70428418	7042841.8
IK + BiRRT + TrajOpt	126399408	12639940.8
BiRRT inside(second)	37.313	3.7313
Size	166	16.6
Success	10	100%

Table A.20: Three Bars(BiRRT+TrajOpt), Second Movement with Left Hand

Description	Totoal	Average
IK	12008106	1200810.6
BiRRT	75585301	7558530.1
IK + BiRRT	87593407	8759340.7
BiRRT inside(second)	67.668003	6.7668003
Path Size	3534	353.4
Success	10	100%

Table A.21: Three Bars(BiRRT), First Movement with Left Hand

Description	Totoal	Average
IK	9108350	910835
BiRRT	59621771	5962177.1
IK+BiRRT	68730121	6873012.1
BiRRT inside(second)	48.738	4.8738
Path Size	4599	459.9
Success	10	100%

Table A.22: Three Bars(BiRRT), First Movement with Left Hand

Description	**Totoal**	**Average**
TrajOpt time	416391792	41639179.2
Success	2	20%
Size	300	30

Table A.23: Three Bars(TrajOpt), First Movement with Left Hand

Description	**Totoal**	**Average**
TrajOpt time	284310210	28431021
Success	5	50%
Size	300	30

Table A.24: Three Bars(TrajOpt), Second Movement with Left Hand

Bibliography

[1] Dmitry Berenson, Siddhartha S Srinivasa, Dave Ferguson, and James J Kuffner. Manipulation planning on constraint manifolds. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 625–632. IEEE, 2009.

[2] Rosen Diankov. Openrave. `http://openrave.org`, 2016(Online; accessed 01-April-2016).

[3] Rosen Diankov. ikfast module. `http://openrave.org/docs/0.6.6/openravepy/ikfast/#ikfast-the-robot-kinematics-compiler`, 2016(Online; accessed 26-March-2016).

[4] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[5] Boston Dynamics. Atlas robot. `http://www.bostondynamics.com/robot_Atlas.html`, 2016(Online; accessed 27-March-2016).

[6] Open Source Robotics Foundation. Robot operation system. `http://www.ros.org/`, 2016(Online; accessed 01-April-2016).

[7] Open Source Robotics Foundation. Gazebo simulator. `http://gazebosim.org/`, 2016(Online; accessed 30-March-2016).

[8] Python Software Foundation. Python. `https://www.python.org/`, 2016(Online; accessed 01-April-2016).

[9] Carl Benedikt Frey and Michael A Osborne. The future of employment: how susceptible are jobs to computerisation. *Retrieved September*, 7:2013, 2013.

[10] Gurobi. Gurobi optimizer. `http://www.gurobi.com/`, 2016(Online; accessed 01-April-2016).

[11] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[12] Léonard Jaillet, Juan Cortés, and Thierry Siméon. Transition-based rrt for path planning in continuous cost spaces. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2145–2150. IEEE, 2008.

[13] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4569–4574. IEEE, 2011.

[14] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *arXiv preprint arXiv:1005.0416*, 2010.

[15] Sven Koenig and Maxim Likhachev. Fast replanning for navigation in unknown terrain. *Robotics, IEEE Transactions on*, 21(3):354–363, 2005.

[16] J Kuffner and S LaValle RRT-Connect. An efficient approach to single-query path planning ieee international conference on robotics and automation. *San Francisco*, pages 473–479, 2000.

[17] Steven M LaValle. Rapidly-exploring random trees a ew tool for path planning. 1998.

[18] Lening Li. Birrtopt. `https://www.youtube.com/channel/UCd7BFnE5SDCUN01Ue3yOYCA`, 2016(Online; accessed 30-March-2016).

[19] Jennifer M Ortman, Victoria A Velkoff, Howard Hogan, et al. An aging nation: the older population in the united states. *Washington, DC: US Census Bureau*, pages 25–1140, 2014.

[20] Chonhyon Park, Jia Pan, and Dinesh Manocha. Itomp: Incremental trajectory optimization for real-time replanning in dynamic environments. In *ICAPS*, 2012.

[21] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 489–494. IEEE, 2009.

[22] Carnegie Robotics. Multisense sl. `http://carnegierobotics.com/multisense-sl`, 2016(Online; accessed 26-March-2016).

[23] John Schulman, Jonathan Ho, Alex X Lee, Ibrahim Awwal, Henry Bradlow, and Pieter Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, pages 1–10. Citeseer, 2013.

[24] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317. IEEE, 1994.

[25] Anthony Stentz et al. The focussed d* algorithm for real-time replanning. In *IJCAI*, volume 95, pages 1652–1659, 1995.

[26] Yu Yan, Emilie Poirson, and Fouad Bennis. Integrating user to minimize assembly path planning time in plm. In *Product Lifecycle Management for Society*, pages 471–480. Springer, 2013.