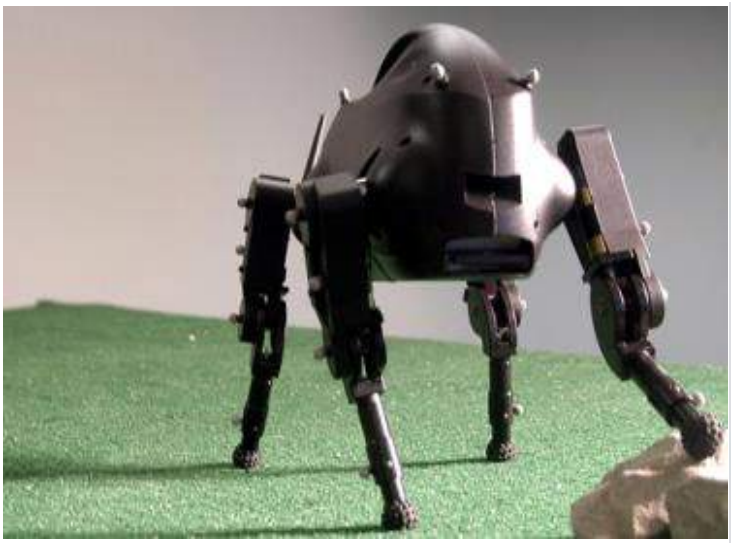
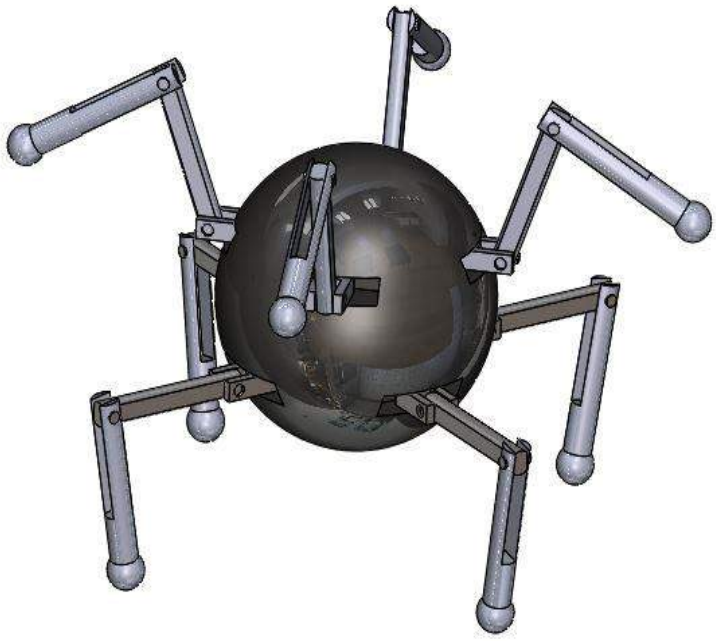
General Controller for Multi-Limb Robots for Pushing Task

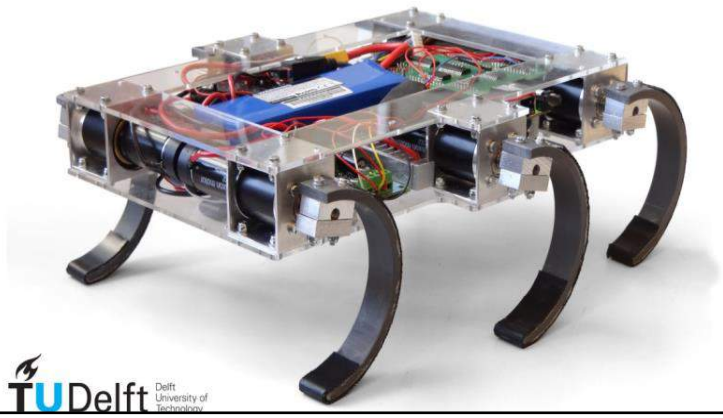
By Adapting Posture (Contacts and CoG)

Rajesh Subburaman

Master of Science Thesis







TU Delft Delft University of Technology

Delft Center for Systems and Control

General Controller for Multi-Limb Robots for Pushing Task

By Adapting Posture (Contacts and CoG)

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Rajesh Subburaman

September 27, 2015

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Multi-limb robots have the potential to be dexterous in manipulation and be able to locomote in challenging terrain. In the future they should match or supersede the movement capabilities of humans and animals, thus becoming useful to support humans in their daily activities or perform those activities individually in environments which are dangerous to humans. Humans and animals carry out such tasks by means of skills such as pushing, pulling, twisting and grasping, and for the multi-limb robots to perform such tasks, they also need to possess such skills. This thesis addresses the pushing skill by developing a general controller for multi-limb robots. The controller tries to execute the pushing task in a way humans would do. This controller is novel because it considers the possibility of using surrounding objects as potential support for high force generation. The resulting optimization-based motion is surprisingly similar to how humans push large objects.

The controller involves three major modules. The first module computes a **Posture Tree**, which takes geometric information about the environment and the robot, and generates a set of possible postures. These include the selected contacts in the robot and their respective location on the object. The second module is the **Posture Optimization**, which uses a cost function based on stability, friction at contact, joint and torque range to optimize each posture. The final module consists of an **Operational Space Controller (OSC)**, which selects the least costing posture and executes it by dividing the complete posture into several tasks. Each task is operated within the null space of its higher level task. The force is generated by modifying the end targets of the end-effectors (pushing contacts) and the robot resists the destabilizing moment resulting from the reaction force, by moving the center of gravity in the direction opposite to an approximation of the ZMP movement.

Initial verification involving first two modules was carried out with a humanoid robot of 29 Degree of Freedom (DOF) and its results are included in this report. For final verification, a more generic multi-limb robot of 8 end-effectors (Octobot 32 DOF) and a typical living environment were conceptualized, developed and evaluated. For validation of the complete controller a number of simulations were carried out in V-Rep. Simulation results depict the controller's ability to identify the suitable posture for any multi-limb robot, given some pushing task in any given environment. The results collectively demonstrate the controller's ability to handle heavy object manipulation, autonomous contact planning, adapting posture

according to friction at the contacts and adapting posture according to the pushing force requirement. Unlike the present controllers, the complete control module developed here is more generic both in terms of the system (robot) and the environment in which the system operates.

Table of Contents

List of Figures

List of Tables

Acknowledgements

During the entire course of this challenging graduation project, I had the opportunity to learn and apply new things, and enjoyed doing them so. I am grateful to a number of people from various fields, who helped me both professionally and personally during this project, without them it wouldn't have been possible for me to complete the project successfully. First and foremost, I would like to thank Dr. Gabriel Lopes for infusing this research topic in my mind and for constantly pushing me towards building a more generic concept. Without his constant pushing, the complete concept developed in this project wouldn't be so general as it is now. I would also like to thank him for his wise guidance throughout this project. The experience which I gained during my internship helped me a lot and enabled me to work without much assistance. For providing such a wonderful internship opportunity, I would like to thank Dr. Martijn Wisse, who was also my internship supervisor. I would also like to thank him for taking time to review my work and also for giving me valuable suggestions.

I would also like to thank Martin Felis, for taking time from his busy schedule to reply to my queries related to the Rigid Body Dynamics Library (RBDL) developed by him. I had to undergo a number of issues with V-Rep simulator during simulation and I couldn't have handled those issues without V-Rep admins. So, I would like to thank the v-rep technical administrators for replying swiftly and helping me as much as they could.

Above all, this master's degree would have remained as a dream for me, if my parents and uncle had not supported me. I don't have words to express my gratitude to them, for motivating me and supporting my financial needs all these years irrespective of their situation. Finally, I would like to thank my friends for being with me and motivating me all these years.

Delft, University of Technology — Rajesh Subburaman
September 27, 2015

Chapter 1

Introduction

Ever since the advent of multi-limb robots such as Fujitsu's HOAP, Waseda's WABOT, Honda's ASIMO, Boston's MiniDog, NASA's Athlete, etc., there has been a vast amount of research done to improve its stability and locomotion capabilities such as walking, running etc [14, 15, 16, 17, 18]. Concepts such as zero moment point (ZMP)[19], foot rotation indicator (FRI)[20] were introduced to quantify stability and these were further extended to design stable walking patterns by means of a Walking pattern generator. The vast amount of research done over many decades, has endowed robots with locomotion capabilities that allows them to traverse large distances autonomously. Yet, they are still limited to environments filled with less uncertainty. Through powerful path planning algorithms such as Rapidly exploring Random Trees (RRT) [21, 22], Probabilistic Road Map (PRM)[23], etc., the robot's ability to handle uncertainties and navigate through obstacles have been extended, but still limited. Though there are still researches being continued to extend these capabilities to highly unstructured environments, lately there has been a shift in the focus of the development of multi-limb robots. The focus is being laid upon the development of controllers to make them perform certain human tasks either individually or support them in doing so [24, 25, 26, 27].

The main purpose of designing multi-limb robots is to operate them in human environments alongside humans or to perform tasks in dangerous environments. With multi-limb robots emulating humans/animals in all aspects right from its structure to Degree of Freedom (DOF), they are expected to be capable of doing such tasks effectively. It is also relatively easy for highly redundant multi-limb robots such as humanoids to negotiate well with irregular grounds by making discrete contacts with it, which further enhances its suitability to perform complex tasks done by human. Above discussed similarities also makes them ideal to handle various equipment in human environments. All of the above challenges highly motivate researchers to develop controllers that enable multi-limb robots to perform all these tasks. Since by nature, humans perform any task in an energy efficient manner, researchers are developing controllers to drive multi-limb robots to imitate human in order to realize those tasks. This has been attempted by using whole body motion coordination, just as human do.

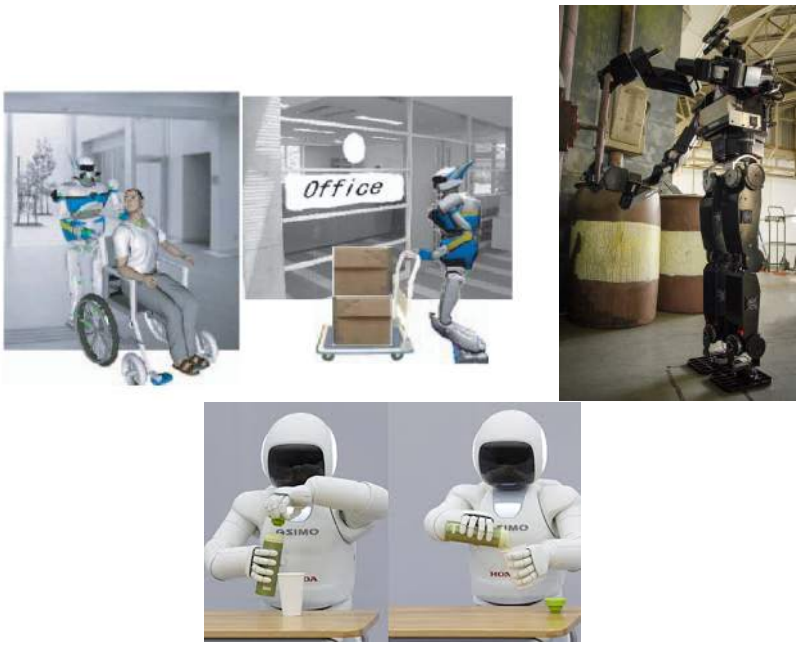


Figure 1-1: Biped robots performing various tasks (Pushing, turning and grasping).

1-1 Why Pushing Skill?

For multi-limb robots to successfully carry out any human task, such as opening doors, lifting a can, holding a bottle, drilling a hole, cutting wood, etc., some basic skills are required apart from maintaining stability and navigating safely. All these tasks require particular coordination of the body and it also involves continuous interaction with the environment, which subjects the robot to external forces. Some of the basic skill sets which a robot should possess for carrying out these tasks are pushing, pulling, grasping, turning/twisting, etc. Each of these skill sets in turn necessitates the robots to adapt their postures and contacts in a careful way, so that the tasks are performed in a stable manner.

Many applications in our day to day life require exerting a certain amount of force in certain directions in order to achieve the end objective. Pushing skill, not only caters to the need of manipulating some object in the horizontal direction, but it is also useful in handling other tasks like closing doors, pushing trolleys, carriage, assembly operations in industries, etc. Since a great deal of reaction force acts on the robot during the manipulation of the objects, it has to adapt its body optimally in order to counter those reaction forces and at the same time get the job done successfully. Even though the grasping skill is also required for many tasks, there has been a significant amount of research done already and further there are hardly any disturbing forces acting on the robot while executing such tasks. Pushing skill is also very important for robots carrying out exploration and rescue operations, where the environment is highly unstructured necessitating the skill of pushing to manipulate the obstacles in order to proceed ahead.

1-2 Previous Work & Motivation

Since the focus on whole body manipulation of humanoids just started in the late 20th century, there had been few controllers developed in the 21st century addressing the pushing task. The first controller developed [1] was based on ZMP, followed by [2] where it was proposed the force controlled manipulation of objects along with robot's gait. In [4], a controller was devised to modify the COM position by making use of the pushing force for manipulation. Other researchers [3] used the inertia of dynamic walking to push an object, while others [5] conceptualized separate controllers for pushing and locomotion. Several controllers based on optimization [6, 7, 8] were also proposed to take into account many constraints while manipulating the object, but their success has been limited. In order to reduce the complexity involved with pushing tasks, the uncertainties related to objects were neglected and much simpler controller was proposed as in [9, 10]. Recently, a slightly advanced controller[11] which can push, pull and orient objects in a particular direction was also proposed.

The above controllers, addresses mainly the aftermath of pushing as most of them were developed from walking based controller. Some controllers also discuss about the physical interference negotiation, minimizing torque, joint limit compatibility and partly discuss about friction at contacts. Almost all the controllers assume a static posture and predefined contacts irrespective of the object's mass and friction coefficient, which limits the magnitude of pushing force, robot's stability, general applicability to other applications and increases the energy consumption. Humans adapt their postures according to the force requirement for a particular task. These postures are found to be energy efficient as well. The main reason for the discrepancy observed in the present controllers is its inability to adapt the robot's posture according to the requirement of pushing force, environment, etc. Absence of whole body motion coordination in multi-limb robots, is also one of the main reasons for their below par performance while executing pushing task. As such, a part of this project focuses on developing a general control law to adapt the posture (by adapting CoG and contacts) of robots, as humans do in order to exert desired forces, and thereby address the above shortcomings which has not received sufficient attention.

1-3 Contribution of the Thesis

The following are the works contributed by me through this thesis in developing the general controller for multi-limb robots for pushing task.

- Identifying all possible postures, using the environment, robot and task details. This includes selecting suitable contacts in the robot, feasible pushing locations on the object to be pushed, choosing a favorable supporting object and corresponding locations on it.
- Devising cost functions to evaluate and optimize the posture in such a way that it mimics the way humans adapt according to the force requirement and environmental conditions.
- Designing controller to realize the optimum posture with the given robot from its initial state and execute the pushing task in a stable manner.

- Verification and Validation of the controller with a multi-limb robot and environment by carrying out simulations in V-REP.
- Creating a local repository [1] of the Rigid Body Dynamics Library (RBDL) (built by Martin Felis [2]) in Ubuntu 14.04, for kinematic and dynamic calculations related to the robot system model.
- ***Robot_Model:*** A C++ module is built for extracting specific details of the robot from urdf file such as, joint limits, predefined contact details, kinematic chain connecting the contacts to the main body, joint discretization and end-effector position computation (Monte-Carlo method),CoG Jacobian and Jacobian derivatives.
- ***Posture Tree Module:*** A C++ module for generating the posture tree, given a multi-limb robot model, environment details, object details, contact details and joint details as input. A data structure is created in the module for handling data related to the contacts, environment objects and postures. Some of the notable functions developed in this module for generating postures are selections of pushing, supporting & active planes, geometric projection & filtering techniques (plane-point distance, point projection on a plane, etc.,) for selecting contact points, object filtering, contact points xyz limit extraction on each plane, support polygon construction, stability checking, iterative inverse kinematic computation of joints given a target position, contact pruning and pushing & supporting posture generator.
- ***Posture Optimization Module:*** This module is also built in C++ and it optimizes the postures obtained in the previous module by means of cost functions, given an approximate pushing force as input. Similar to the above module, this module also contains data structure for handling details related to contacts and optimum postures. This module contains some of the functions developed in the previous module. Apart from those functions, there are specific functions built in this module for optimization purpose such as initializing the optimizer, extracting contact positions to optimize, approximate pushing force computation, contact force & control torque computation for a given posture, cost criteria (joint, torque, friction & stability) computation, cost function evaluation and storing optimum position details in the optimum posture structure.
- Integration of RBDL and Optimization library (dlib C++) with the above modules for generating postures and optimizing them.
- ***OSC Module:*** The Operational Space Control module is developed as a plugin to be used in the V-Rep simulator and it is written in C++. The control module is heavily based on the principle of Louis Sentis [28]. The module takes optimum posture details and it's corresponding optimum values as input, computes the OSC torque and sends it to the robot for execution. The prime functions developed in this module are, posture data, contacts & its optimum location extraction, segregation of the posture into separate OSC tasks, computation of inertia, coriolis & centrifugal forces and gravity forces in the operational space, computation of reference acceleration, computation of each task level forces within the null space of the higher level tasks, computation of

[1] Local repository is nothing but installing the complete library in static form. This lets the user to access/-modify the functions of the library for his/her own use.

[2] http://rbdl.bitbucket.org/index.html

OSC torque from the task level forces, optimum posture reach, exerting pushing force and ZMP compensation movement.

- OSC control principle & simulator's torque control validation with a simple double pendulum model.
- Integration of RBDL and OSC plugin with the simulator for carrying out dynamic simulations.
- 3D model development of multi-limb robot (OctoBot) and the environment (a typical living environment) considered for final evaluation of the complete controller is done using SolidWorks. The urdf file for the multi-limb robot is generated using the SolidWorks-URDF plugin and it is verified using rviz(ROS visualization tool).
- Generation of dynamic models of the multi-limb robot (OctoBot) and the environment using simple shapes in V-Rep in order to have a stable simulation and initial validation of the robot model with gravity compensation.
- Generation of Matlab GUI and .m files for visualizing and analyzing the results obtained with the posture tree generation, posture optimization and OSC modules by means of various plots.

1-4 Structure of the Report

The thesis report is structured/organized as follows:

Chapter 2 discusses briefly the general pushing dynamics, dynamics involved in human pushing, influencing factors, strategies adopted by humans, general controller requirements for pushing task and present controller limitations.

Chapter 3 introduces the general pushing concept developed through this thesis by briefly talking about its conceptualization, overview and short description of major modules involved.

Chapter 4 focuses on the Posture Tree Generation module. It contains the methodology, various inputs it requires, the process flow of the module and finally preliminary validation & results are discussed.

Chapter 5 talks about the Posture Optimization module. It involves parameter selection for optimization, cost functions, optimization method used and the module's validation & results.

Chapter 6 introduces different control methods and reasons for choosing OSC, control architecture and its validation with a simple model.

Chapter 7 presents the robot and the environment considered for evaluation, model preparation for the simulator, final simulation and the results obtained through it are discussed.

Chapter 8 concludes the report by summarizing the thesis work and stating future recommendations.

Chapter 2

Pushing Analysis and General Controller Requirements

"Pushing is an act of exerting force of a certain magnitude over an object, applied towards the direction in which we intend to drive or propel it."

2-1 General Pushing and Affecting Factors

Pushing force in general is greatly influenced by the friction force and also the inertial resistance of the object. Pushing is always resisted by the friction force (F) acting in the direction opposite to that of the force applied, and the magnitude of resistance depends on the coefficient of friction (μ) between the object and the floor. So in order to move the object, the pushing force should at least equal the friction force (F). The force required to push the equipment is always greater in the beginning (just before the object starts moving), which is due to the static friction. This is generally called as the initial or starting force. Fortuitously, this initial force prevails only for a short duration as it drops gradually to the sustained force level once certain acceleration is achieved. The force requirement is typically lower during the object's constant velocity motion. A significant increase in force during the motion can only occur, if there is a change in the acceleration direction (i.e. turning) or frequent deceleration and acceleration while precisely positioning the object.

Figure 2-1 shows the dynamics involved in pushing an object.

Affecting Factors

The following factors affect or influence the act of pushing in general.

- Dynamic or inertial forces: The initial push force is always higher as it includes in part the inertial forces, i.e. the force required to overcome the object's inertia. This force

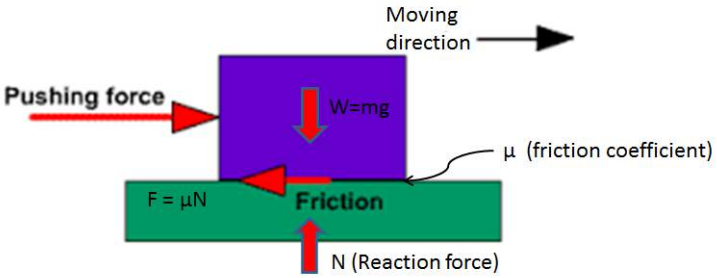


Figure 2-1: Dynamics involved in pushing

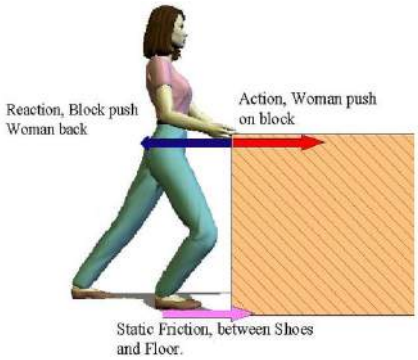


Figure 2-2: Forces acting on a human while pushing an object [12].

is directly proportional to the desired acceleration with which the object is required to move.

- Friction forces: Relative motion between two objects which are in contact will always result in friction, resisting the movement. The coefficient of friction defines the magnitude of this resistance, which in turn depends on the type of surface in contact. Friction can be categorized as either static (starting) or dynamic (rolling). The static forces are usually higher than the dynamic. Hence the initial pushing force to propel the object is always greater than the force required to sustain motion, as the former involves acceleration and also overcoming static friction forces.

2-2 Human Pushing Dynamics and Influencing Factors

Pushing is always preferred for humans for heavy object manipulation due to various physiological and efficiency reasons [13]. Physiologically, pulling while facing in the direction of travel puts the person in a mechanically awkward position and pulling from behind increases the risk of accidents. From a human perspective, pushing an object involves much more dynamics, apart from the general dynamics discussed in the previous section. This is shown in figure 2-2.
Humans need to have adequate strength to apply the minimum required initial or starting (as discussed earlier) force to push the object. According to Newton's third law, an equal

amount of reaction force acts on the human body during pushing, which tends to destabilize the posture. In order to apply the required force effectively, sufficient traction should also be adopted at the footholds in order to avoid slipping. The following figure shows the dynamic forces acting on a human during manipulation task which involves pushing.

2-2-1 Factors influencing human pushing

Human pushing skill is influenced by a number of factors, other than the general factors which affect pushing in general (discussed in earlier sections). Some of the major factors are discussed below.

- ***Handhold location:*** Handholds are very important, as they infer the person regarding where and how to exert force on the object. In the absence of such handholds, humans generally search for convenient or mechanically advantageous handholds for applying forces.
- ***Handhold height:*** This part plays a major role, as this defines partly what posture the person will assume, which in turn defines the magnitude and direction of effective pushing forces applied on the object.
- ***Body Posture:*** Human musculoskeletal system is essentially a series of mechanical levers. The length of lever arms is defined by the position of the joint and muscles. Some postures are mechanically advantageous than other postures as it may involve the use of muscle groups which are bigger and hence can generate more force when they are aligned optimally to exert force.
- ***Foot Positioning:*** Along with the handhold height, the pushing posture is defined by the location of the feet as well. Keeping the feet together reduces the magnitude of the pushing force and also the stability of the posture, while keeping the feet apart increases both.
- ***Friction forces/traction at the feet:*** The magnitude of pushing force is greatly limited by the friction at the feet. As stated earlier, the amount of force applied to an object reacts equally at the foot in the opposite direction. Increasing the pushing force beyond the friction maintainable/available at the feet, will result in slippage. However, pushing an object with some supporting object will completely neutralize this slipping effect, due to the reaction force generated at the supporting contact.
- ***Angle of Push force application:*** The ideal direction of applying force to move an object is horizontal. But in an actual situation, there won't be sufficient handholds at the desired height or there might be a necessity to increase the traction at the feet, so the pushing force has to be applied at an angle from the horizontal in order to prevent slipping. In such scenarios optimal angle should be selected to balance both.

There are also other factors such as length of travel involved during pushing, repetition rate for the task and the duration of the task. These factors reduce the generation of force, increases the metabolic demand, thereby reducing the muscle recovery period and increasing fatigue respectively.

Figure 2-3: Factors influencing a human while pushing an object [13].

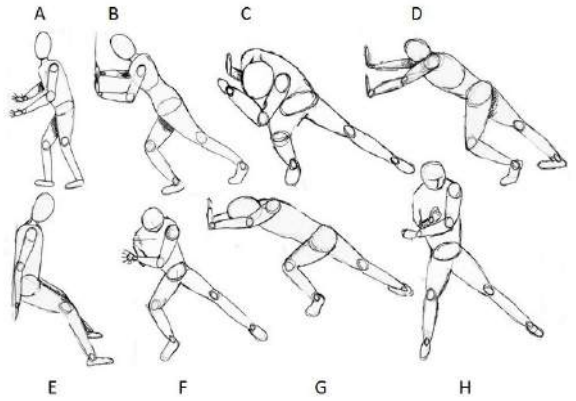


Figure 2-4: Different pushing methods adopted by human according to the requirement [29].

2-3 Different types of Postures

Humans adopt different strategies depending upon the requirement of pushing force, provided the influencing factors (discussed in the previous section) are supportive to do so. Figure 2-4 depicts the pictorial representations of various postures adopted by human for generating pushing force. Some of the common strategies adopted by human during various situations are discussed briefly below.

- ***Simple pushing postures:*** For objects which require very less force, humans tend to manipulate it by simple posture which involves keeping the feet together and generating force using hands.
- ***Pushing force and stability increasing postures:*** These postures are adopted when a considerable amount of force is required. In such cases, the same amount of reaction force act on the human which can destabilize him. To resist this, they tend to adopt the feet apart strategy in order to increase their support polygon. Doing so will keep the ZMP [1] point within the support polygon even though the reaction force tends to

[1]Since the human is pushing some object, there will be high reaction force (equal to pushing force) acting

Figure 2-5: Posture adopted to compromise both pushing force and friction at the feet(Left).Extreme posture adopted by human for pushing a car(Right).

move it. Increasing the distance between the feet proportionately increases the support polygon (stability) and hence the capability to apply more force as well. Postures A and B depicts those postures in figure 2-4.

- ***Stronger muscle usage postures:*** These postures are similar to the previous one, but here body posture factor plays a great role. The notion behind this is to adopt a configuration to make use of the stronger muscles in torso and legs, so that high force can be generated. Posture C, E and F shown in figure 2-4 fall under this category.
- ***Self weight utilizing postures:*** Other than generating forces through muscles, self weight of the body is also used partly for generating the required force by leaning onto the object. This reduces the amount of energy consumption to a great deal as part of the force is generated without activating excessive muscle fibers. In order to realize this posture, the object should be stable enough to handle the weight of the human when he leans on it. Posture D and F shown in figure 2-4 belong to this category.
- ***Friction/Traction postures:*** The ideal direction of generating pushing force is horizontal, but in situations which demand high pushing forces than the available friction force at the feet-floor contact, humans tend to adopt postures which balances these two. Handholds are placed at an angle from the ideal horizontal direction in order to balance both the force requirements. Angle of push plays a major role here. Figure 2-5 (left) shows this posture.
- ***Extreme Pushing postures:*** Humans adopt these postures when very high pushing forces are required. In these postures humans almost align their entire body in the horizontal direction. Such extreme postures are only possible if sufficient friction force is available at their feet. Figure 2-5 (right) shows this posture.

2-4 General Controller Requirements

With the general aspects of pushing skill, influencing factors and specific strategies adopted by the human to execute pushing skill efficiently been discussed briefly in the earlier sections,

at the contact which will tend to move the ZMP. So the ZMP computed will no longer be equal to CoG even if the acceleration is ignored (since the CoG acceleration observed during pushing motion is less) [30].

now the basic requirements for a controller to support any multi-limb robot to do the same i.e., to imitate humans have been listed below.

1. ***Minimal Object Knowledge:*** Should be able to determine the required force for a given object with minimal input about the objects (dimensions, type of contact) and the floor.
2. ***Autonomous contact planning:*** Ability to determine the feasible handholds on the object to manipulate it.
3. ***Different pushing strategies:*** Capability to assess the situation and adopt various strategies according to the requirements.
4. ***Varying Support Polygon:*** Ability to vary the support polygon by planning the foothold and handhold position and executing it stably.
5. ***Self Weight Utilization:*** Making use of its self weight to generate the required pushing force.
6. ***Torque Minimizing Postures:*** Adopting postures which balance between the energy consumption and the required pushing force.
7. ***Joint Limit Compatibility:*** Postures should be compatible with the joint and torque limits of the robot.
8. ***Traction check:*** Measures to check slipping at the footholds and handholds and adopt postures accordingly.
9. ***Physical interference negotiation:*** Capability to deal with physical interference which disrupts or blocks the smooth pushing motion of the object.
10. ***Aftermath of Pushing:*** Should possess suitable mechanism to handle the after pushing effects i.e once the object starts moving.

2-5 Present Controller Limitations

Present controllers for manipulating the object by means of pushing skill can be segregated into seven major types, based on the technique they are built upon. Considering each controller's merits & demerits and their ability to support the general requirements framed in the earlier section, their capabilities and limitations are shown in figure 2-6.

2-6 Requirements Addressed

The controller developed in this is focused more on the following requirements, which were either not addressed at all or not addressed satisfactorily by the present controllers.

- Minimal Object Knowledge

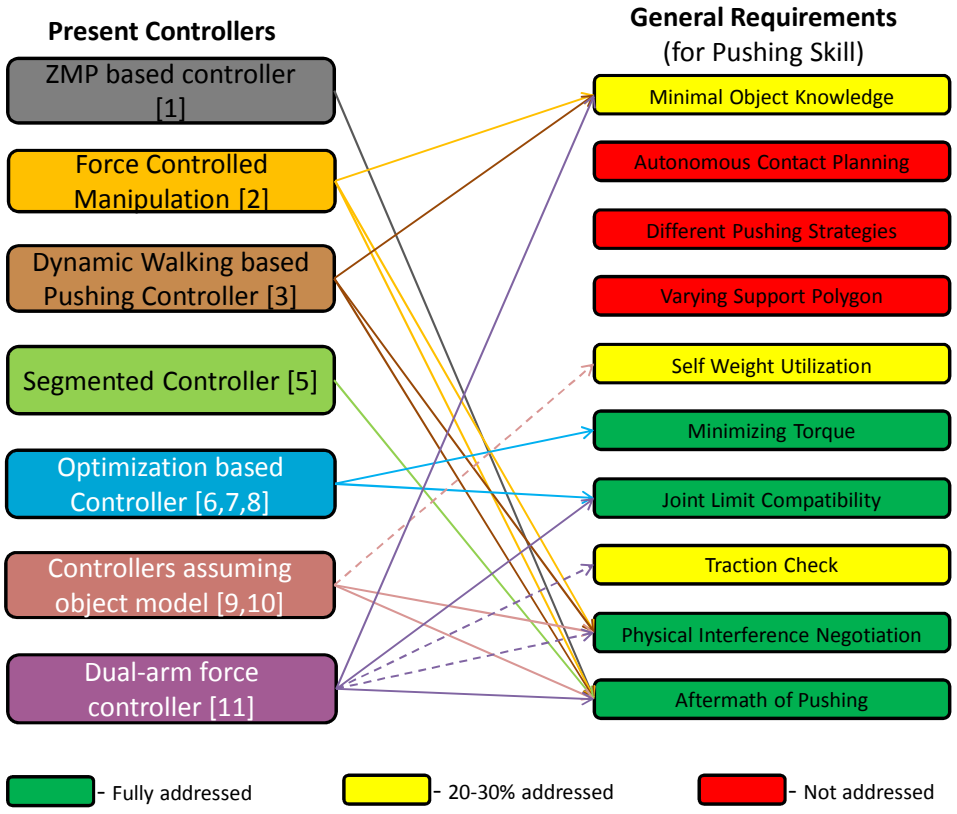


Figure 2-6: Present Controllers and the general requirements (pushing skill) which they support.

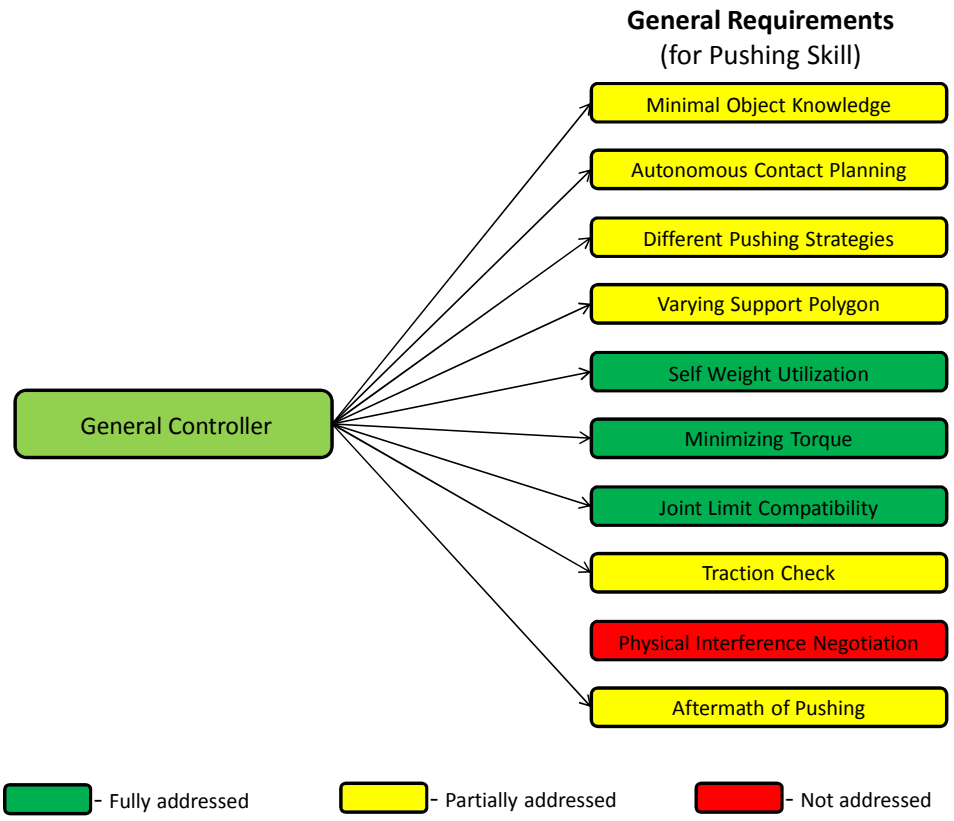


Figure 2-7: General requirements for pushing skill addressed by the controller developed in this thesis.

- Autonomous Contact Planning
- Different Pushing Strategies
- Varying Support Polygon
- Self Weight Utilization
- Traction Check

The overall requirements addressed by the controller developed in this thesis is shown in figure 2-7

Chapter 3

General Pushing Concept

The general pushing concept developed for multi-limb robots in this thesis is inspired by how a human would react to a pushing task which involves an object of unknown mass in some given environment. Humans tend to roughly estimate the force required to push an object by extracting (vision) and processing the object's details (material, type of floor contact, dimension, etc.) and also by using their past experiences of handling similar objects. The above specified attribute (approximating pushing force) of humans is not taken into account here. So the environment considered here includes an unknown object (light brown), a platform (dark brown) and some fixed object.

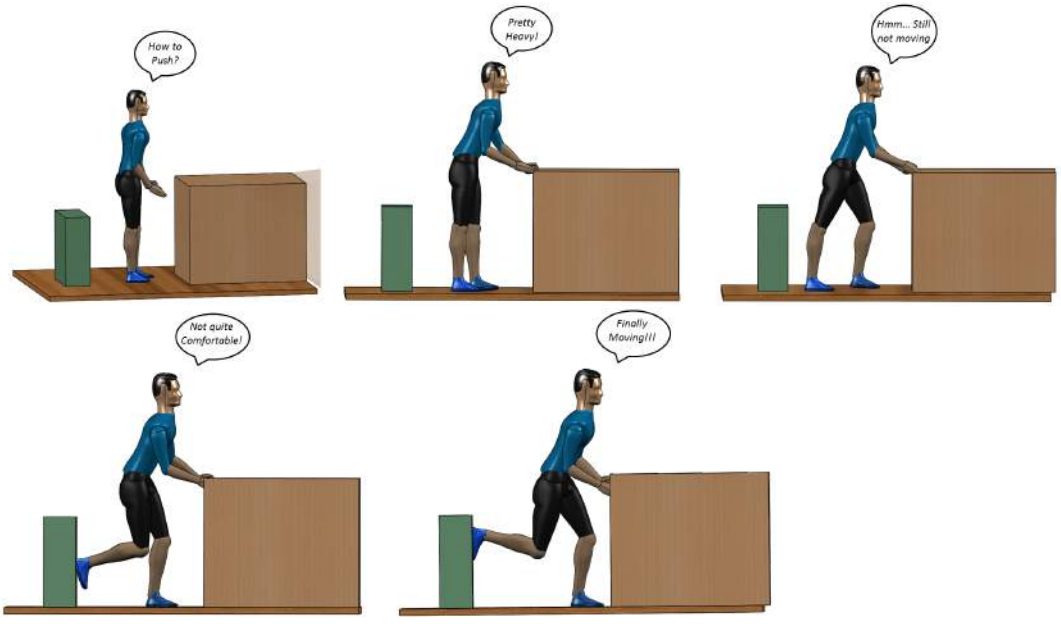


Figure 3-1: Different phases involved in a human pushing an unknown object (a specific case).

Assuming the human is able to visualize the environment, the various phases involved when

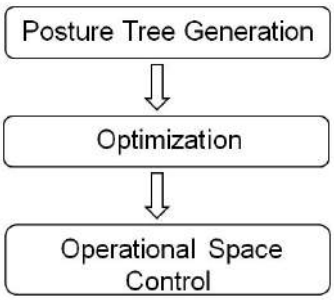


Figure 3-2: Overview of General Controller Concept for multi-limb robots.

he tries to push the object is shown in figure 3-1. The phases are numbered starting from top left.

Phase 1 and 2: The human evaluates the surrounding environment, identifies the object to be pushed, checks if the object is reachable with the available contacts and finally verifies if it requires less or more force.

Phase 3: If the object is found to be heavy in phase 2, then the human tries to exert more force by altering his posture.

Phase 4: Even after exerting more force if the object fails to move, then human goes into phase 4, wherein he looks for suitable objects in the environment which can be used as support for exerting very high force. Upon identifying such object, the available contacts reachability to that object is checked.

Phase 5: Finally, if the object is reachable, then the posture is adjusted in such a way that the human places himself in a comfortable position to exert more force in a stable manner.

In both phase 3 and phase 4, humans adjust their postures for comfortableness and it generally refers to a posture which yields more stability, consumes less energy, less slippery at contacts and feasible to do. In phase 3, apart from the posture shown in figure 3-1, humans adopt different kinds of postures as shown in figure 2-4.
In the latter part of phase 4 and phase 5, when humans start exerting pushing force on the object, there will be an equal amount of reaction force acting at the pushing contacts as shown in figure 2-2. Assuming there is enough friction at the foot contacts, this reaction force tend to destabilize the posture and humans resist this destabilizing moment by moving their body (CoG) forward.

3-1 Overview of Concept

The complete pushing concept for multi-limb robot is conceptualized from the above discussed human pushing process. The overall structure of the concept looks as shown in figure 3-2.

The complete concept involves three major modules.

- **Posture Tree Generation:** It takes geometric information about the environment, kinematic and dynamic details about the robot, object details and robot's predefined

contact details as input. With all these details, it selects the pushing and supporting planes on respective objects, where the robot can make contact. Using these selected planes and the available contacts, the possible postures for the given robot under the given environment for some assigned pushing task is generated. The complete set of postures form a posture tree, which will be used by the successive modules for further processing. So in principle, this module represents phase 1 and phase 2 of human pushing process, except for the fact that it doesn't try to identify how heavy the object is.

- **Posture Optimization:** The set of postures obtained in the previous module is taken as input, and each posture is optimized by varying each contact's location and the robot's COG. The evaluation of postures is done by means of cost functions based on stability, contact friction, joint torque and joint limits. Thus, each posture is assigned with certain cost, which represents the robot's comfortableness in that particular posture. This module tries to replicate what a human does in phase 3 or phase 5 to adjust its posture for comfortableness.

- **Operational Space Control:** The best posture, i.e. the least costing posture is selected from the optimized ones and it is executed by means of OSC. A control architecture is devised, which divides the selected posture into several sub-tasks for realizing the posture successfully. The module also tries to replicate the human way of resisting the destabilizing moment (due to reaction force) during pushing, by moving (with less acceleration) the CoG according to the reaction force experienced at the pushing contacts of the robot. This is explained in detail in chapter 6.

3-2 Model Classification and Terminology

The classification and terminologies considered for both the robot and the environment models are discussed below. This will be used for all the three modules described in the earlier section.

3-2-1 Robot Model

The robot model which will be considered for preliminary evaluation of the modules is shown in figure 3-3. It is a humanoid with 29 DOF and four contacts in total.

Classification of contacts:

The predefined contacts of the robot are classified into two types,

- **Active Contacts:** These are the contacts which will be in touch with the floor by default (i.e. initial configuration) and they define the stability of the robot at all configurations. For the model shown in figure 3-3, both the foot sole contacts (A_{c1} & A_{c2}) fall under this category.

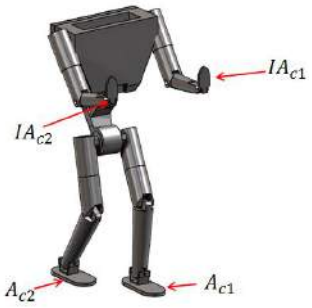


Figure 3-3: Robot model for preliminary evaluation of modules

- **InActive Contacts:** These contacts are initially free in the initial configuration and hence can be used readily for making either pushing contact or supporting contact with the corresponding objects. Both the hands (IA_{c1} & IA_{c2}) of the robot shown in figure 3-3 belong to this category.

3-2-2 Kinematic and Dynamic Calculations:

For kinematic and dynamic computations of the robot model, Rigid Body Dynamics Library (RBDL) developed by Martin Felis [1] is used in this thesis. The complete library built by Martin felis is in C++, and his codes are heavily based upon the pseudo code of the book "Rigid Body Dynamics Algorithms" by Roy Featherstone [31, 32].
The Dynamic algorithms are developed and expressed in 6D vectors (i.e. Spatial Vectors). These vectors contain both the linear and angular components of physical quantities like velocity, acceleration and force. Compared to the dynamics algorithms expressed in 3D vectors, spatial vectors greatly reduces the algebraic computations involved, makes it relatively easier to describe the dynamics and also simplifies the implementation process in a computer.
The complete algorithms are model based routines, which requires the system model to be given as input in specific format. The RBDL used here accepts the system model in two formats, Lua scripts and URDF files. The latter format is used here for doing the computations using RBDL.
For the humanoid shown in figure 3-3, the urdf file of Coman robot (built by IIT) is used. This file was generated by Enrico Mingo and Alessio, PhD students at Italian Institute of Technology (IIT), Italy.

3-2-3 Environment Model

The simple environment shown in figure 3-4, is considered for preliminary evaluation of the modules along with the robot model shown above.

Classification of Objects

The objects in the environment are classified as follows,

[1] http://rbdl.bitbucket.org/index.html

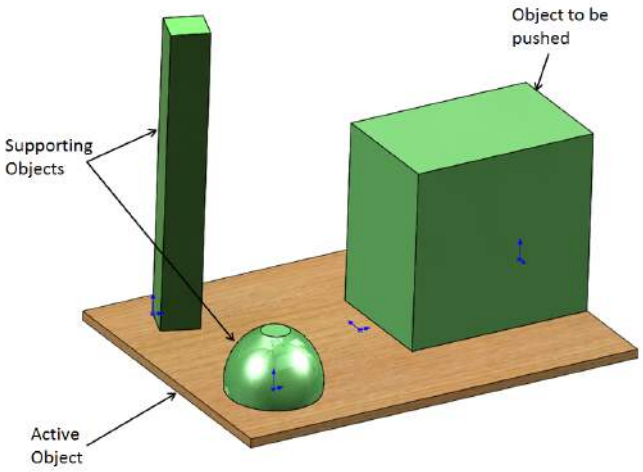


Figure 3-4: Environment model for preliminary evaluation of modules

- **Pushing Object:** This refers to the object which has to be pushed in the environment. Only one object which has to be manipulated by the robot can be given as pushing object.
- **Active Object:** The object(s) which remains in contact with the active contacts of the robot, in order to maintain its stability are termed as active object.
- **Supporting Objects:** All the other objects present in the environment will be considered as supporting objects, which will be considered by the robot for making support contact while pushing.

Chapter 4

Posture Tree Generation

4-1 Methodology

The Posture Tree Generation module takes five input files, which contains information about the environment and the robot. A data structure is created after extracting the details from the files, which is then processed to deliver a set of the possible postures for the given robot. The process details will be explained in the next section. Figure 4-1 shows the various files which the module takes in as input.

Input Files

- **Contact List:** It's a text file which contains details about the predefined contacts in the robot body. This file specifies which contacts are active and inactive in the initial configuration.
- **Environment Details:** This file contains information about the environment, where the robot will be operating. It is again a text file, which specifies the object to be pushed, objects which can be used as support and the active object, which the robot is already in contact with.

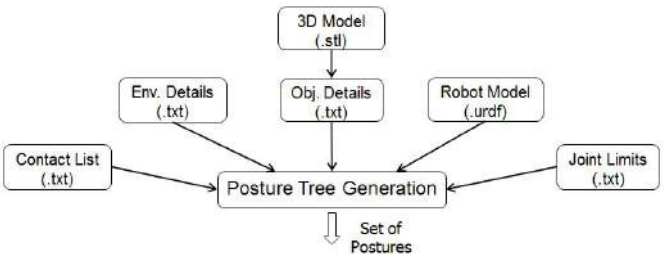


Figure 4-1: Posture Tree Generation module with input files

- **Object Details:** This file is extracted from a 3d model stored in .stl format. The STL format describes the outer surface of any 3d object by means of triangular meshes. Using an STL reader, each object specific details such as vertex, faces and norms are extracted in text files. These details give information about each object's location and its orientation in the given environment.
- **Robot Model:** This file is in .urdf format, which stands for Universal Robot Description Format (URDF). This file contains kinematic details about the robot such as type of joint, its location & rotation axis and its joint limit. It also contains dynamic details of each link such as its mass, inertia and CoG location.
- **Joint Limits:** This is a text file which contains details about the max and min limit of each joint present in the robot.

4-2 Process Flow

The complete process flow in generating the postures can be divided into three sections.

4-2-1 Contact Points Structure

The data are extracted from the input files and a data structure is created for the contacts and environment object, which stores the corresponding details. The possible positions each contact of the robot can reach is computed by discretizing the joints connecting the contact with the main body and using forward kinematics. The computed positions are stored in the contact structure. With the location and orientation details of the object and the robot, pushing and supporting plane are selected using some geometric techniques (explained later in this section). The contact points (computed earlier) closer to the selected planes are filtered, and those that are beyond the planes are projected (using point-plane distance & point projection on a plane) back into each plane and stored separately. These are the only possible positions where the robot's end-effectors can make contact with the object.

Kinematic chain & **Joint Discretization:**

The kinematic chain of any contact involves a series of joints which connects any given contact frame to the robot's local base frame. This extraction of kinematic chain is done using the model tree. The process of extracting kinematic chain and discretizing the joints is explained below with the robot model considered for preliminary evaluation (i.e. Humanoid).

Figure 4-3, shows the active and inactive contact frames along with the kinematic chains for RSoftHand and R_sole.
Kinematic chain of RSoftHand (inactive contact) - J1,J2,J3,J4,J5,J6,J7,J8,J9 & J10.
Kinematic chain of R_sole (active contact) - J11,J12,J13,J14,J15 & J16.
Similarly, the kinematic chains for other contacts can also be extracted. The selected joints which form a kinematic chain for a particular contact are discretized within their respective range, and using forward kinematics the contact's possible positions are computed. Since for the inactive contacts (LSoftHand & RSoftHand), first 3 joints are associated with changing their orientation, only joints J4-J10 are considered for discretization. Similarly, for the active

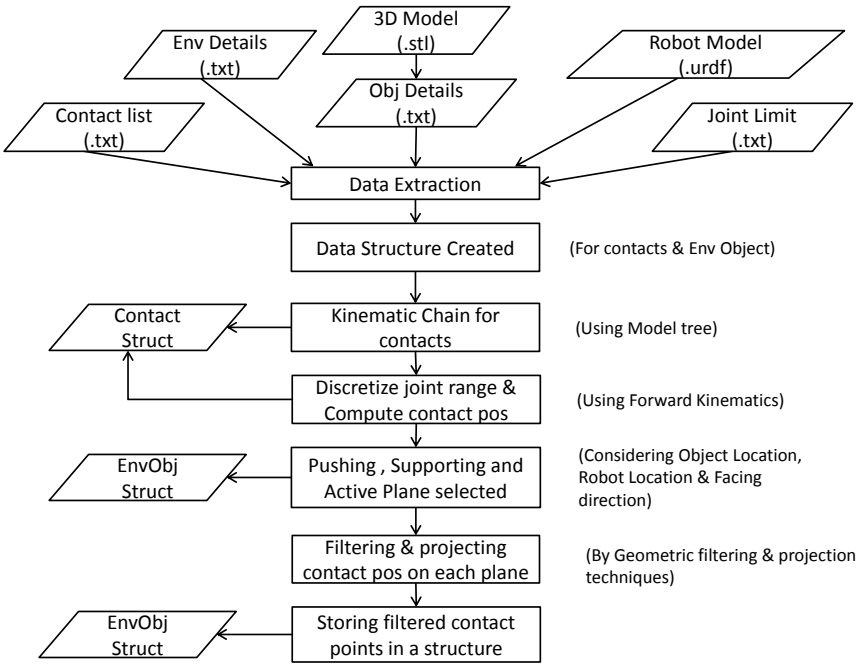


Figure 4-2: Flow chart for Contact Points data structure

contacts (L_sole & R_sole) only joints J11-J14 are considered for discretization, for the same reason. The total possible positions for R_sole and RSoftHand are computed as shown below.

For active contact - R_sole:
Joints considered - J11,J12,J13 & J14.
No. of Joints, N_J - 4.
Discretization size, d_s - 8.
No. of possible contact positions - $d_s^{N_J} = 8^4 = 4096$.

For active contact - RSoftHand:
Joints considered - J4,J5,J6,J7,J8,J9 & J10.
No. of Joints, N_J - 7.
Discretization size, d_s - 4.
Number of possible contact positions - $d_s^{N_J} = 4^7 = 16384$.

Plane Selection:

The general concept of selecting planes is briefly explained by means of a specific case as shown in figure 4-4. A norm vector, R_{N1} is constructed for the robot in the facing direction and the angle between this vector and the face norm vectors of the pushing object ($P_{N1}, P_{N2} \ldots$) are computed using 4-1. The plane angle (P_{ang}) which falls within 135° - 180° are selected, which forms the pushing plane. In this way, the pushing planes are selected by the module and the maximum allowable orientation of the pushing object is 45°.

$$P_{ang} = atan2(norm(R_{N1} \times P_{N1}), (R_{N1}.P_{N1})) \tag{4-1}$$

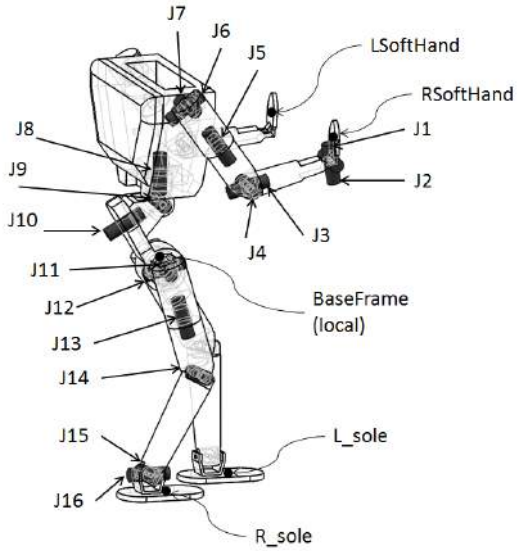


Figure 4-3: Humanoid robot with active (L_sole & R_sole), inactive (LSoftHand & RSoftHand) contacts and its local base frame (BaseFrame).J1-J10 is the kinematic chain of RSoftHand and J11-J16 is the kinematic chain of R_sole.

Similarly, the supporting planes are also selected, but since the supporting objects can be anywhere behind the robot and it can be in any orientation, it is necessary to filter out the objects which can effectively support the robot in the pushing direction. This is done by dividing the region behind the robot into five, with each region spanning 15°. Each region is set with an S_{ang} limit and any object which falls within that region, and if it's orientation is within the stipulated limit, only then the object will be considered for support. S_{ang} is calculated the same way as P_{ang} was calculated earlier, but the angle is computed between the selected pushing plane and all the face norms of the supporting objects. Since the supporting objects can be on either side of the robot (behind), orientation signs are checked while selecting the planes. The signs can be computed as $sgn = sign(P_{N2}.(P_{N1} \times S_{N1}))$ (assuming P_{N1} is the selected pushing plane). This way the pushing and supporting planes are extracted from the given data. A detailed explanation of selecting the planes is demonstrated with an example in appendix D.

Validation & Results

For validation, the environment and robot shown in figures 3-3 and 3-4 are used. The results so obtained are shown below in figure 4-5.

4-2-2 Pushing Posture Generation

With Contacts, EnvObj structure and the robot model, the type of contacts (Active or InActive or both) which should be considered for making pushing contact is selected by means of certain heuristics. One of the heuristic is based on the height ratio between the robot's base and the object's maximum height. The selected contacts further undergoes distance pruning,

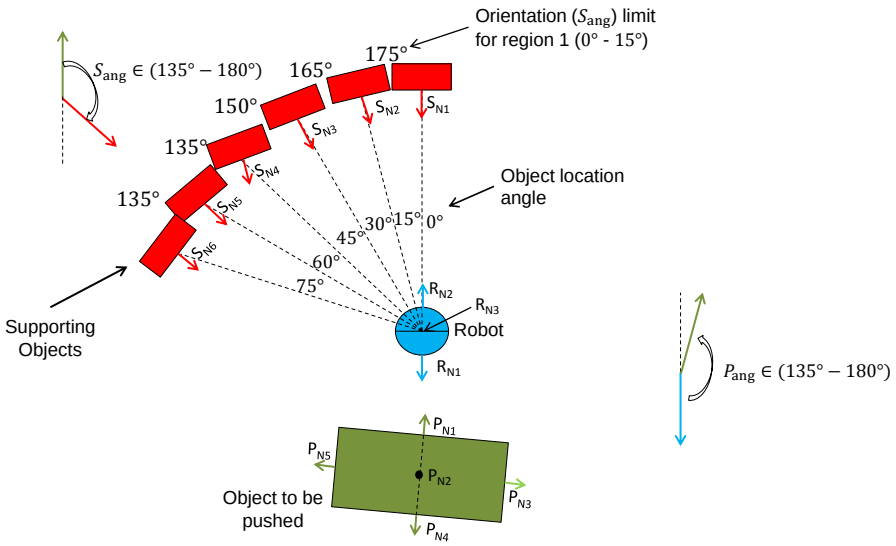


Figure 4-4: Pushing and Supporting plane selection in a specific case which involves a pushing object (green) and 6 supporting objects placed behind the robot. Robot norm vectors ($R_{N1}\ldots$), pushing object norm vectors ($P_{N1}\ldots$) and supporting object norm vectors ($S_{N1}\ldots$).

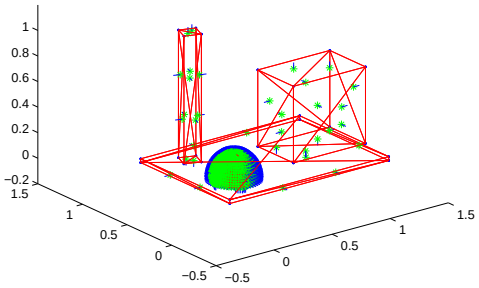
which prunes out the contacts farther to the pushing plane. From the filtered contacts, each contact is selected and moved to the pushing plane by inverse kinematics and the resulting posture is subjected to the selection criteria. Here the posture's end target reachability, stability and joint limit are checked. Upon satisfying the criteria successfully, the posture is added to the Posture Set structure and the contacts are updated. This is repeated for all the filtered contacts. Figure 4-6 shows the process flow for generating pushing posture.

Posture Selection Criteria

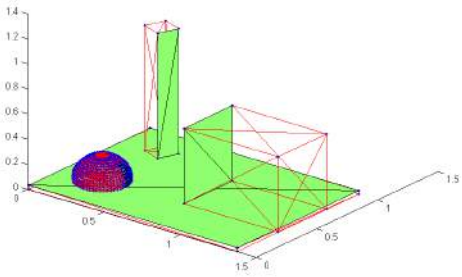
The following criteria are used for preliminary evaluation of the posture before it is selected.

- Distance Pruning:
  With the initial configuration of the robot, the distance (perpendicular) between the selected plane and the available contacts are computed. From the computed distance of each contact, the one with the minimum distance is selected and also the contacts which are closer to the minimum one ($\approx$ 50 mm) are selected.

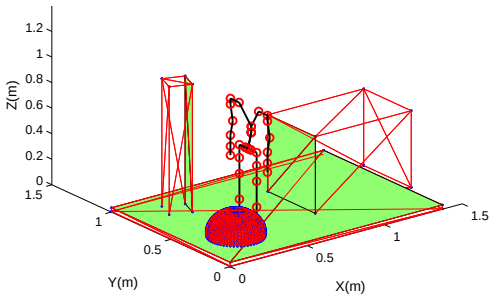
- Heuristics:
  Some basic heuristics are devised to further prune the above selected contacts, in order to achieve postures which are both reasonable and feasible. The following are the heuristics used while selecting pushing postures. Figure 4-7, shows the dimensions of the object to be pushed, its base, robot's base frame location and its height.



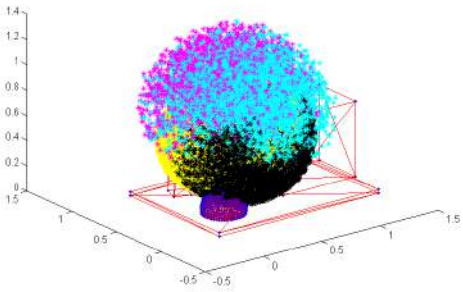
(a) Environment Constructed from .stl file. Green points & blue lines represent the centroids and normals of each triangle.



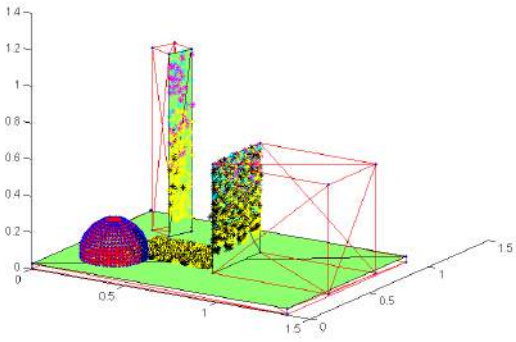
(b) Selected pushing, supporting and active planes (green shaded) in the model.



(c) Robot positioned in the environment with the selected pushing and supporting planes.



(d) Each contact's possible positions. Magenta & Blue - Left and Right hand,Yellow & Black - Left and Right leg



(e) Filtered contact points on the selected planes (after projection).

Figure 4-5: Plane selection and contact points filtering process

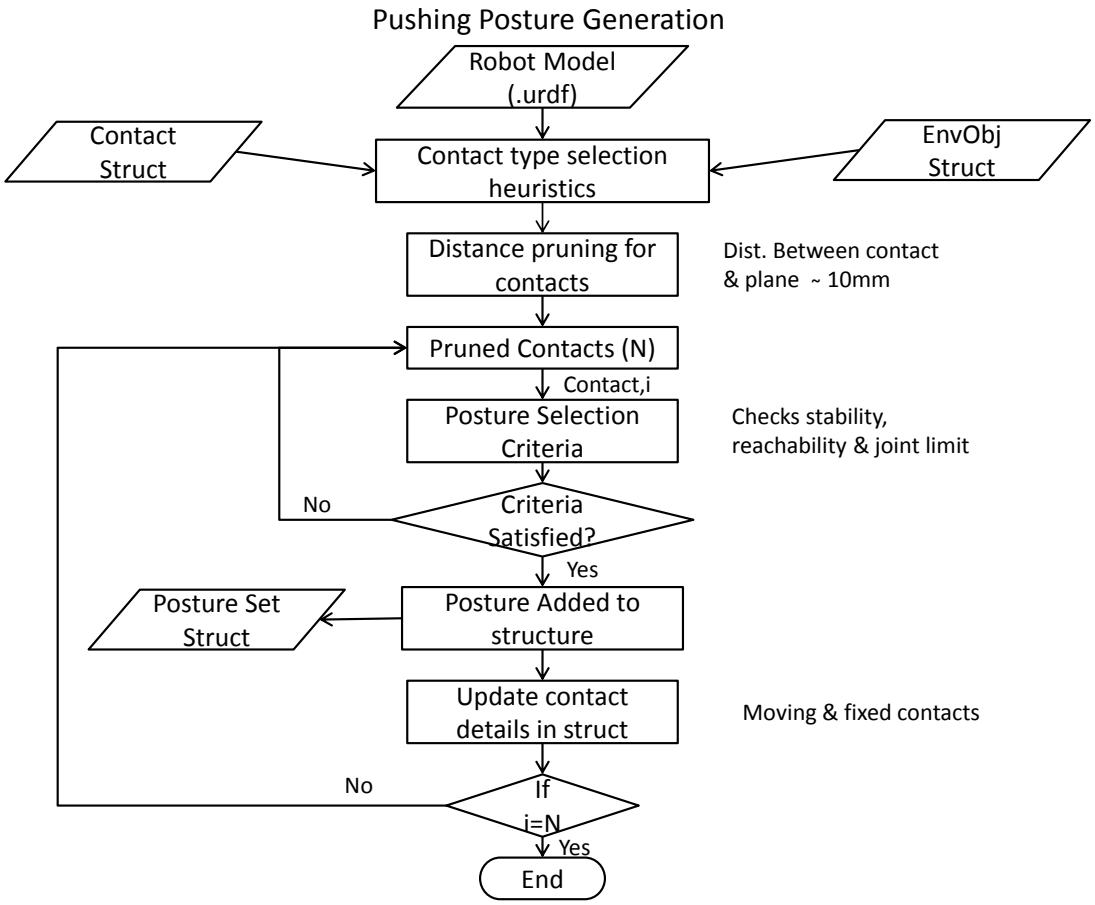


Figure 4-6: Pushing posture generation process flow

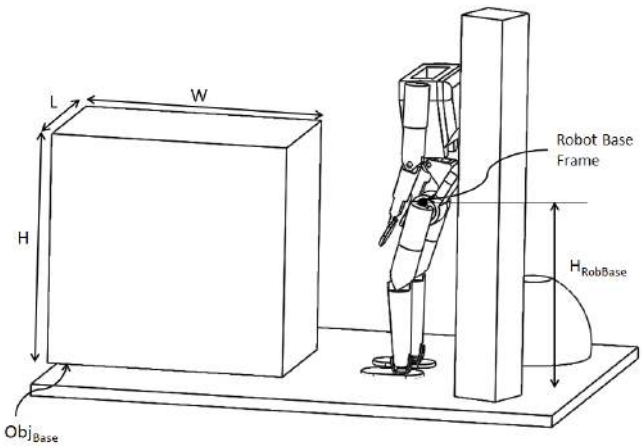


Figure 4-7: Object dimensions (L x W x H), its base (Obj_{Base}) and Robot's base frame height ($H_{RobBase}$) are shown.

- The selected contacts should have at least 10 possible positions on the selected pushing plane.
- The ratio between the robot's base frame height ($H_{RobBase}$) to the maximum pushing height ($H_{PushHeight}$) denoted by H_{ratio} is calculated. $H_{PushHeight}$ [1] is determined by comparing the object's width (W) and its height(H). If (W $\geq$ H), then $H_{PushHeight}$ = H, else $H_{PushHeight}$ = W. The type of contacts to be considered for making pushing contact are decided based on H_{ratio} and the object's base location (Obj_{Base})(This heuristic is used for handling object of any size placed at any height).
  If ($H_{ratio} >= 2$) & ($Obj_{Base} \leq (H_{RobBase}/2)$) - Only active contacts will be considered.
  If ($H_{ratio} < 0.75$) & ($Obj_{Base} \leq (H_{RobBase}/2)$) - Both active and inactive contacts will be considered.
  If (($H_{ratio} >= 0.75$) & ($H_{ratio} < 2$)) || ($Obj_{Base} > (H_{RobBase}/2)$) - Only inactive contacts will be considered.
- At least one active contact should remain in contact with the active object, in order to ensure stability of the robot.

- Joint Limit:
  The values of each joint are restricted to remain within their respective joint limit. A boundary of 5% is used on both the max and min limit of each joint (i.e $0.95(\theta_{min}) < \theta < 0.95(\theta_{max})$), in order to avoid postures with extreme joint values.

- Reachability:
  After moving the selected contacts to their respective target location, the norm of the difference between each contact's current position and its target position is computed. The postures are selected only if the norm is ≤ 0.025.

- Stability criteria:
  The stability of the robot is evaluated by computing the ZMP [2](static), and checking if it lies within the support polygon. If it falls outside, then the robot is moved (i.e CoG) to a stable position and its stability is re-checked. Since each posture involves a different set of contacts, both the ZMP and the support polygon are computed every time. The static force acting on both pushing and supporting contacts, when they are placed on the respective objects are also considered for computing ZMP. Figure 4-8 shows a humanoid making both pushing and supporting contact for manipulating an object, and ZMP is computed as shown below in equations 4-2 & 4-3.

$$x_{zmp} = \sum_{i=1}^{Nc} \frac{-Mg(x_G) - (z_i - z_{zmp})f_{xi} + (x_i)f_{zi}}{-Mg + f_{zi}} \tag{4-2}$$

[1] $H_{PushHeight}$ is chosen in such a way that the robot's maximum pushing contact height is not greater than the object's width. This prevents the toppling of the object while pushing.

[2] As long as the robot doesn't lean on to the object it is making contact with, the contact forces will be very small and in such cases ZMP = CoG. In cases where moving some contact (for ex. active contacts) puts a significant amount of weight onto the object (supporting/pushing), the contact forces will no longer be negligible and in such cases ZMP will differ from CoG.

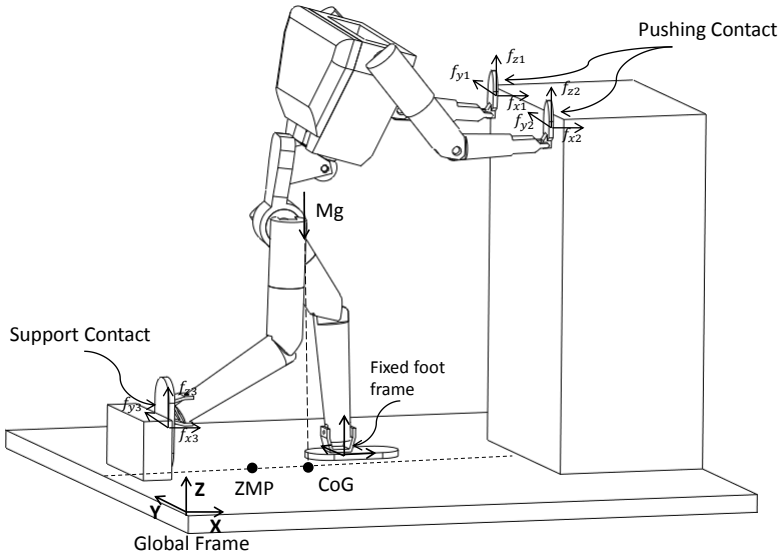


Figure 4-8: ZMP calculation for a humanoid making pushing and supporting contact.

$$y_{zmp} = \sum_{i=1}^{Nc} \frac{Mg(y_G) + (z_i - z_{zmp})f_{yi} - (y_i)f_{zi}}{Mg - f_{zi}} \tag{4-3}$$

where, $x_{zmp}, y_{zmp}, z_{zmp}$ - xyz position of ZMP.
x_G, y_G - xyz position of CoG.
M - total mass of the robot.
g - acceleration due to gravity.
N_c - No. of pushing and supporting contacts.
f_{xi}, f_{yi}, f_{zi} - i^{th} Contact forces in x, y & z direction.
x_i, y_i, z_i - xyz position of i^{th} contact.

4-2-3 Supporting Posture Generation

After generating pushing posture, each posture is selected and a corresponding supporting posture is generated with the remaining contacts (i.e. contacts apart from the one used for pushing) and the support planes. The process adopted for pruning contacts, evaluating posture and adding to the posture set is similar to pushing posture generation. The process flow for generating supporting posture is shown in figure 4-9.

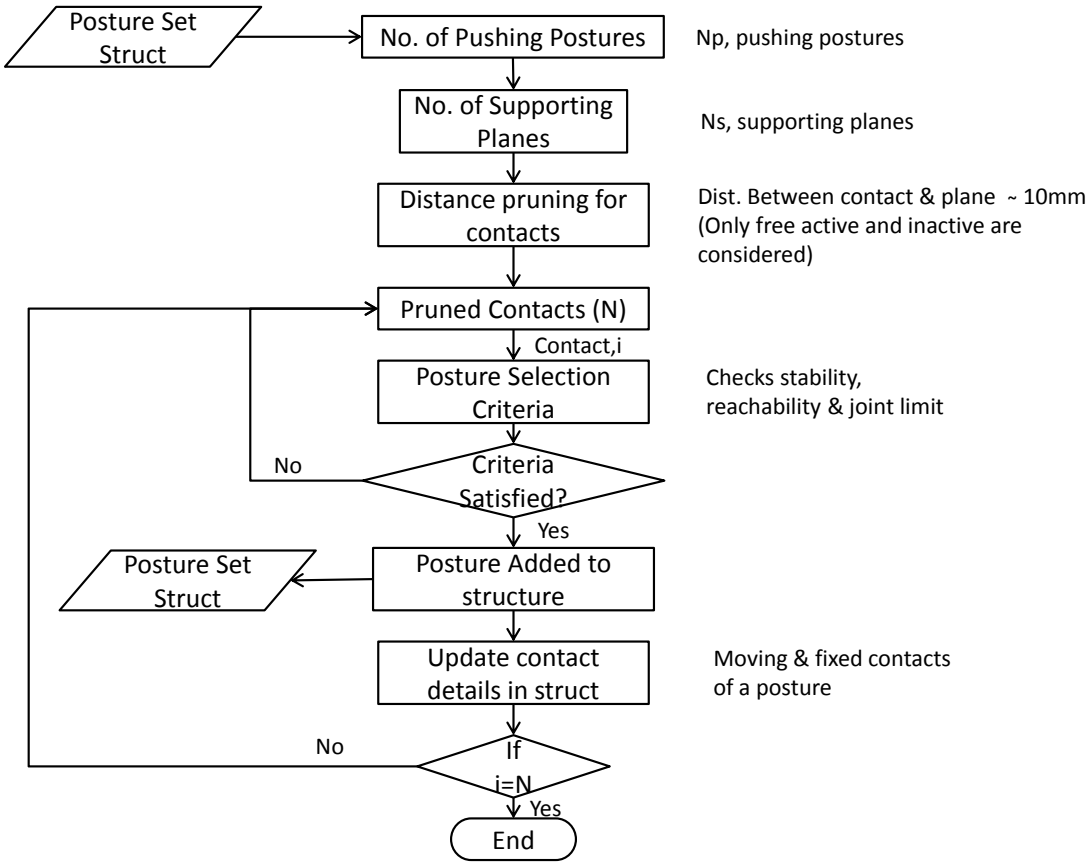


Figure 4-9: Supporting posture generation process flow

Total number of postures:
No. of possible pushing postures, $N_{PPos} = \sum_{i=1}^{N_P} \sum_{j=1}^{N_{PC}} PPos_{ij}$
No. of possible supporting postures, $N_{SPos} = \sum_{i=1}^{N_{PPos}} \sum_{j=1}^{N_{SObj}} \sum_{k=1}^{N_{SP}} \sum_{l=1}^{N_{SC}} SPos_{ijkl}$
Total Postures = $N_{PPos} + N_{SPos}$
where,
PPos - Pushing Posture.
SPos - Supporting Posture.
N_P - No. of pushing planes.
N_{PC} - No. of pushing contacts (after pruning).
N_{SObj} - No. of supporting objects.
N_{SP} - No. of supporting planes.
N_{SC} - No. of supporting contacts (after pruning).

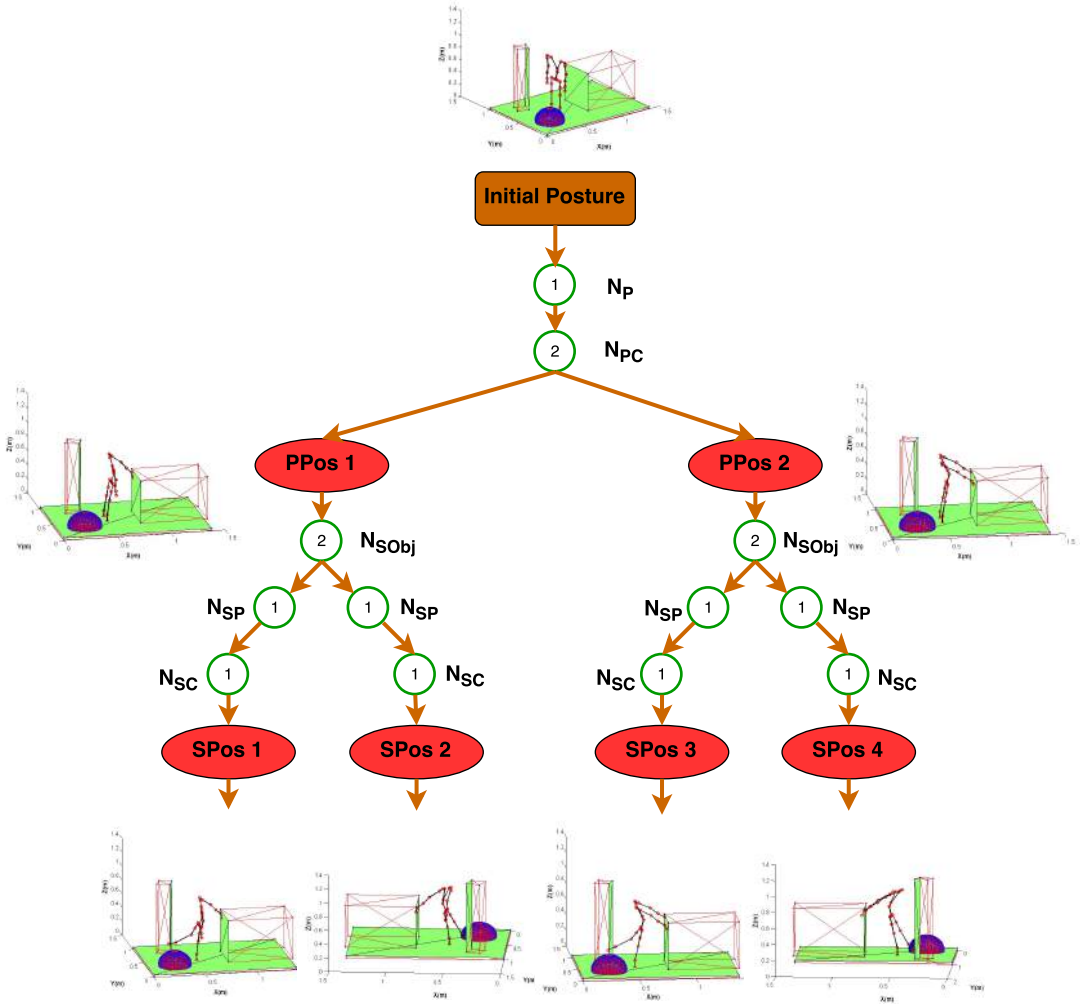
4-3 Validation & Results

The complete module is evaluated with the robot and environment shown in earlier chapter. The following is the result obtained.

Robot Details:
Robot type - Humanoid.
Degree of freedom - 29 + 6 (floating base).
No. of Active contacts - 2.
No. of InActive contacts - 2.

Plane & Postures Generated:
No. of pushing plane - 1.
No. of supporting planes - 2.
No. of pushing postures - 2.
No. of supporting postures - 4.

The generated posture tree is shown in figure 4-10.



- **N_P** - No. of Pushing Planes
- **N_{PC}** - No. of Pushing Contacts (after pruning)
- **N_{SObj}** - No. of Supporting Objects
- **N_{SP}** - No. of Supporting Planes
- **N_{SC}** - No. of Supporting Contacts (after pruning)
- **PPos** - Pushing Posture
- **SPos** - Supporting Posture

Figure 4-10: Posture tree generated for the given robot and environment.

Chapter 5

Posture Optimization

The postures obtained in the previous module will be optimized here with cost functions, to find the optimum one for the pushing task. For finding the optimum posture, the following things should be selected or defined wisely.

- Selection of suitable parameters for optimization
- Optimization Method
- Cost Functions

Considering all the available parameters for optimization will increase the time consumed by the optimizer enormously, while taking less parameters will result in a very bad posture. So, the parameters should be selected based on the impact it has on the task. Similarly, optimization method and cost functions play important roles in finding practically feasible postures within a reasonable amount of time. These three major elements are discussed in the following sections and finally, preliminary evaluation of the module will be carried and its results will be discussed.

5-1 Selection of Parameters

The robot's CoG and the contact's location are the two major elements considered here for optimization. Optimizing these two constitute the optimization of entire posture of the robot.

Center of Gravity (CoG)

The robot's CoG is optimized only in the x and z directions. Optimizing in the x direction, lets the robot to move forward and backward and thereby, it explores the possibility of applying its self weight for generating the pushing force. Similarly, optimizing in the z direction, gives the robot an option to place its body mass in line with its end-effector contacts, which can result in applying the pushing force efficiently on the object to be pushed.

Table 5-1: Parameters and its Optimizing direction

S.No	Parameters	Direction Optimized
1	CoG	x and z
2	Pushing contacts	z
3	Supporting contacts	z
4	Active contacts	x

Contact Location:

Pushing & **Supporting Contacts:** The possible directions in which the inactive contacts can be optimized are y and z. Since, keeping contacts closer to the body is more favored than that of keeping it away, and by default the closest (with respect to the contact) y position is always chosen, only the z direction is selected for optimization. Contacts closer to the body increases the pushing force magnitude and also lets the robot apply the force effectively onto the object, while contacts far away from the body reduces the magnitude of force in the pushing direction and also increases the chances of contact slipping.
The z position of the contact has an equal impact on the pushing force generation, maintaining traction at the feet, object's stability and robot's comfortability. The optimum location is identified by varying it and evaluating all the aforementioned elements by means of cost functions.

Active Contacts: For active contacts, x & y are the only possible directions in which the contacts can be optimized. Similar to the inactive contacts, moving the contacts away from the body i.e. y direction is not as advantageous as keeping it closer to the body. Hence, only in x direction the contacts will be optimized.
By moving the contacts x position in + and - direction, the support polygon of the robot is increased which in turn increases the stability of the robot. A robot which is more stable can exert high pushing force on the object in a stable manner. The selected parameters and its corresponding optimizing directions are tabulated in table 5-1. Figure 5-1 shows a supporting posture and the directions in which the selected parameters are optimized.

5-2 Force generation and measurement

The postures as such are stable and in order to evaluate its feasibility for some pushing task, the amount of force which it needs to generate during that task needs to be known. So, an approximate pushing force is given as input to this optimization module for evaluation. During pushing task equal amount of reaction force will act on the robot in the direction opposite to the one, the robot intends to push the object. Hence, the approximate force will be applied as an external force (F_{ext}) acting in the direction opposite to that of pushing, on the robot. If there are more than one pushing contact in a posture, then F_{ext} is divided equally among the contacts.
The external control torque required to resist the external force or generate the approximate pushing force is given by

$$\tau_{ext} = J^T_{eff,i}(F_{ext}/n) \tag{5-1}$$

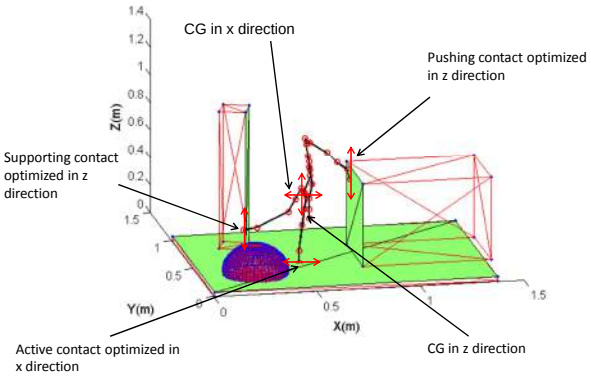


Figure 5-1: Posture with selected parameters and optimized directions.

where,
τ_{ext} - resisting control torque.
$J_{eff,i}$ - jacobian of i^{th} end-effector.
n - no. of pushing contacts.

Contact force measurement

It is necessary to measure the constraint forces at various contacts of the robot due to the external force which acts on it during the pushing task. The contact forces are vital in the evaluation of stability and friction. The general equation of motion for any floating base system can be written as shown below.

$$M(q)\ddot{q} + h(q,\dot{q}) = S^T\tau + J_c^T(q)\lambda \tag{5-2}$$

where,
M(q) $\in \mathbb{R}^{(n+6)\times(n+6)}$: Floating base inertia matrix.
$h(q,\dot{q}) \in \mathbb{R}^{(n+6)}$: floating base centripetal, coriolis and gravity forces.
S = $[I_{n\times n}\ 0_{n\times 6}]$: actuated joint selection matrix.
$\tau \in \mathbb{R}^n$: vector of actuated joint torques.
$J_c \in \mathbb{R}^{k\times(n+6)}$: jacobian of k linearly independent contact constraints.
$\lambda \in \mathbb{R}^k$: vector of k linearly independent constraint forces.

From equation 5-2, it can be seen that both the control torques, τ and the constraint forces, λ are interdependent and one cannot be measured without knowing the other. This problem was solved by Mistry [33], by representing the equation of motion in reduced dimensional space (n-k) through orthogonal decomposition. The complete equation 5-2, can be split into constrained and unconstrained part and can be written as follows.

$$S_cQ^T(M\ddot{q} + h) = S_cQ^TS^T\tau + R\lambda \tag{5-3}$$

$$S_u Q^T (M\ddot{q} + h) = S_u Q^T S^T \tau \tag{5-4}$$

Q,R is computed from the QR decomposition of the constraint jacobian matrix J_c.

where,
$S_c = \begin{bmatrix} I_{k\times k} & 0_{k\times(n+6-k)} \end{bmatrix}$: selects the constrained part of the equation.
$S_u = \begin{bmatrix} 0_{(n+6-k)\times k} & I_{(n+6-k)\times(n+6-k)} \end{bmatrix}$: selects the unconstrained part of the equation.
The control torque, τ can then be calculated from equation 5-4 as follows

$$\tau = (S_u Q^T S^T)^+ S_u Q^T (M\ddot{q} + h) \tag{5-5}$$

Substituting control torque, τ from equation 5-5 and external torque, τ_{ext} from equation 5-1, the contact forces can be calculated as follows.

$$\lambda = R^{-1} S_c Q^T (M\ddot{q} + h - \tau_{ext} - S^T \tau) \tag{5-6}$$

Since the contact forces are computed by maintaining the posture in static condition, $\ddot{q}$ and $\dot{q}$ are set to 0 in equations 5-5 and 5-6.

5-3 Cost Functions

To evaluate the feasibility and comfortability [1] of the posture for performing any particular pushing task, a set of criteria are devised as measurement. For each criteria, depending upon the robot's state, a cost is assigned by means of cost functions. The total cost computed gives a measure of how good the posture is for carrying out the given task. The cost functions are derived from the ones proposed by Yokokohji [34] for static evaluation of postures. The criteria considered here for the evaluation of posture are given below.

5-3-1 Evaluation Criteria

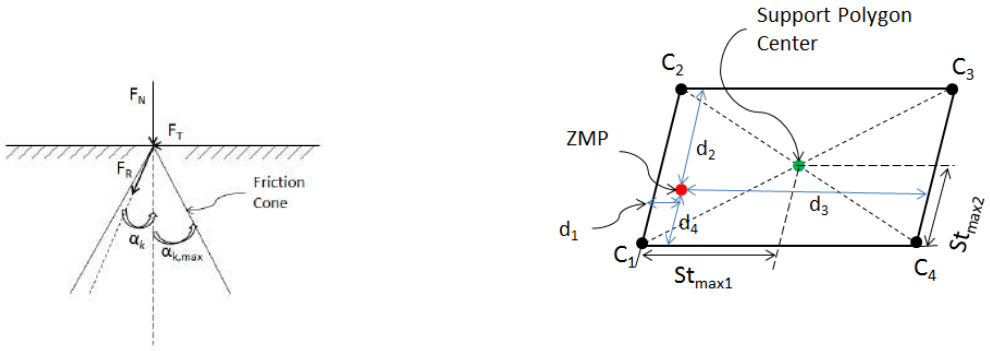
- ***Joint torque limit:***
  The cost function to limit the joint torque of the robot is given below. This function is used to minimize torques.

  $$C_t = \sum_{j=1}^{n} f(|\frac{\tau_j}{\tau_{j,max}}|) \tag{5-7}$$

  where n is the total number of joints of the robot, τ_j and $\tau_{j,max}$ denotes the current joint torque of j-th joint and its corresponding maximum torque. When $\tau_j \geq \tau_{j,max}$, the cost incurred due to the joint torque increases.

- ***Friction limit:***
  To prevent slipping at all contacts of the robot (pushing, supporting and active), the resultant force vector should stay within the friction cone. It is preferable to have some

[1] comfortableness refers to a posture which yields more stability, consumes less torque, less slippery at contacts and stays within the joint limit.



(a) Friction cone with normal force (F_N), tangential force (F_T) and resultant force (F_R).

(b) Support Polygon constructed with four contacts (C_1,C_2,C_3 and C_4) and ZMP with its distance from the boundary.

Figure 5-2: 5-2a shows the friction cone model used for friction cost calculation and 5-2b shows a typical support polygon for stability cost calculation.

margin from the friction cone limit so the robot can be robust against disturbances. The following function is used to do so.

$$C_f = \sum_{k=1}^{m} f(\frac{tan\,|\alpha_k|}{tan\,\alpha_{k,max}}) \tag{5-8}$$

where $\alpha_k = tan^{-1}\mu$ denotes the angle between the resultant force at k-th contact and its corresponding normal vector, $\alpha_{ik,max}$ represents the friction cone angle as shown in figure 5-2a and μ is the coefficient of friction. The cost function assumes a high value as the force vector goes closer to the edge of the friction cone, denoting that robot has high chances of slipping. The friction cone is shown in figure 5-2a. For supporting and pushing contact $F_N = F_x$ and $F_T = \sqrt{F_z^2 + F_y^2}$, while for active contacts $F_N = F_z$ and $F_T = \sqrt{F_x^2 + F_y^2}$. Using F_N and F_T, coefficient of friction can be calculated as, $\mu = \frac{F_T}{F_N}$.

- ***Joint motion range:***
  Every joint has its own, upper and lower limit and the configuration of the robot should fall within that in order to realize the posture successfully. It is preferable to have a certain margin from the upper/lower bounds of the motion range. The following function is used to evaluate the margin.

$$C_j = \sum_{i=1}^{n} f(x) \tag{5-9}$$

where x is determined by

$$x = \begin{cases} \frac{\theta_i - (\theta_{i,max} + 4.75\theta_{i,min})/5}{(\theta_{i,max} - \theta_{i,min})/10} & if \quad -1 \leq \bar{\theta}_i \leq -0.95 \\ 0 & if \quad -0.95 < \bar{\theta}_i < 0.95 \\ \frac{\theta_i - (4.75\theta_{i,max} + \theta_{i,min})/5}{(\theta_{i,max} - \theta_{i,min})/10} & if \quad 0.95 < \bar{\theta}_i \leq 1 \end{cases}$$

where θ_i denotes the angle of the i-th joint, $\bar{\theta_i} = \frac{\theta_i - (\theta_{i,max}+\theta_{i,min})/2}{(\theta_{i,max}-\theta_{i,min})/2}$ denotes normalized joint angle ($-1 \leq \bar{\theta_i} \leq 1$), and $\theta_{i,max}$ and $\theta_{i,min}$ are its upper and lower limits respectively. The cost C_r gets a larger value as the joint angle gets closer to the upper or lower limit.

- ***Stability margin:***
  Zero moment point (ZMP) is used to evaluate and define the stability margin of the robot. ZMP of the robot is computed as shown in equations (4-2 & 4-3) and its corresponding cost is calculated in the following way.

$$C_s = f(|\frac{St_{inst}}{St_{max}}|) \tag{5-10}$$

  where St_{inst} denotes the robot's instantaneous stability around some reference point and St_{max} represents the maximum stability possible. The stability is quantified by computing the ZMP and then measuring how close the ZMP is to the support polygon boundary. As the ZMP gets closer to the support polygon boundary, the stability gets lower and higher cost is incurred.
  A typical support polygon with 4 contacts (C_1,C_2,C_3 and C_4) is shown in figure 5-2b. As shown in the figure, the current ZMP is computed and its distance (d_1,d_2,d_3 and d_4) from each line forming the support polygon boundary are calculated. The least distance gives the boundary to which the ZMP is closer. From the figure, distance d_1 is minimum and St_{inst} can be calculated as $St_{inst} = St_{max1} - d_1$. Substituting St_{inst} and St_{max1} in equation 5-9, the stability cost can be computed.

Total cost function:
The total cost for any given posture is computed by summing up the cost incurred for each criteria shown in equations 5-7,5-8,5-9 and 5-10.

$$C = C_t + C_f + C_j + C_s \tag{5-11}$$

In the above equation 5-11, separate weighting terms (w_t,w_f,w_j and w_s) can be used to give more preference to certain criteria.

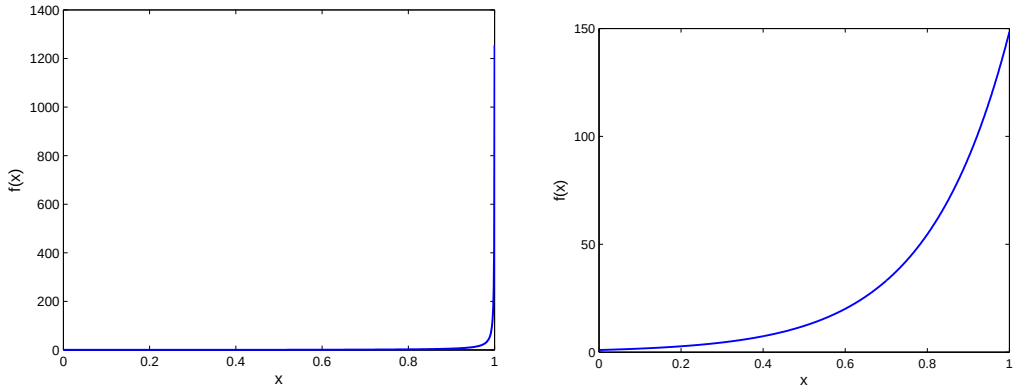
5-3-2 Cost Evaluation

The cost for the above criteria are calculated using the following two types of cost functions ($f(x)$).
Cosine based function:

The following formula is used for computing the cost.

$$f(x) = \frac{1 - cos\frac{\pi}{2}x}{cos\frac{\pi}{2}x} \tag{5-12}$$

With the above function $f(x)$, the cost incurred remains relatively low as long as $x < 1$ and goes to infinity when $x \approx 1$ (increases rapidly closer to 1). This function $f(x)$ makes C_i to get



(a) Cost profile obtained with cosine based function used for C_j only.

(b) Cost profile obtained with exponential based function used for C_t, C_f and C_s.

Figure 5-3: Cost profile for different cost functions.

a large value if the instantaneous/current values are close to their max values. This function is used for evaluating the cost for C_j.

Exponential based function:

The cost is computed as follows.

$$f(x) = e^{yx} \tag{5-13}$$

where, y is any integer to alter the exponential rate.
In the above function, the cost incurred increases exponentially, unlike the previous one where high cost is incurred only when it is closer to the boundary. As x becomes greater than 0, the cost starts increasing exponentially. This function is used for evaluating the cost of C_t, C_f and C_s. For all these criteria, the ideal value is to maintain a low torque/high friction/high stability and it is generally a specific point, unlike the case of joint motion, where it is a range. So, the moment robot starts moving away from the ideal value, cost should be incurred gradually depending upon how far the current value is from the ideal.

The cost profile for both the functions is shown in figure 5-3. Left one shows the cost profile for cosine based function and right one is for exponential based function.

5-4 Optimization Method

The parameters selected as explained in section 5-1 are optimized using some suitable optimization techniques. There are a number of off-the-shelf optimization techniques available, but the suitable method is chosen based on the following requirements.

- Nonlinear optimization - The method should be capable of handling nonlinear systems, since most of the systems considered here are highly nonlinear.

- Derivative free - Should be able to minimize the functions without requiring any derivatives of it.
- Large number of variables - Should be capable of optimizing the function even in the presence of more variables (at least 20).
- Bounded constraints - Constraint handling should be supported, which will greatly limit the search space and as a result, feasible values can be obtained and also the no. of function evaluations can be reduced.
- Fast convergence - Faster convergence rate is required for the robot to perform swiftly and efficiently in real scenarios.

Selected Method

Based on the above requirements, BOBYQA method was selected for optimizing the posture with cost functions.

BOBYQA algorithm:
BOBYQA stands for bounded optimization by quadratic approximation. It is a derivative free, iteration based method used for finding the minimum of a function $F(x)$, $x \in R^n$ subject to bounds $a < x < b$. The algorithm applies trust region method for finding the optimum value of the variables by employing a quadratic approximation Q to function $F(x)$. This quadratic approximation is done by means of some interpolation points, m defined by the user. Generally there is some freedom in the interpolation conditions, which is taken up by minimizing the forbenius norm of the change to the second derivative of the approximation model Q, hence no derivative of $F(x)$ is explicitly required. The values of the variables are constrained by the upper (a) and lower (b) bounds.

5-5 Validation & Results

The complete module is evaluated with the robot and environment shown in Chapter 3 (Figure 3-3 and 3-4). The pushing force considered for evaluation is 117N, calculated using the object dimensions and assuming its density (80 kg/m^3). The results so obtained are plotted in this section.

Cost Convergence

The cost convergence of both the pushing postures and supporting postures obtained over several iterations using BOBYQA method is plotted below. In order to ensure that the cost doesn't get converged at a local minimum point, the optimization is run consecutively for three times, with each successive run starting with the optimal values found in the previous run. In figure 5-4, it can be seen that for all postures, it initially starts with a high cost and as the iteration proceeds it gradually gets converged to a minimum cost and remains stable. Instances of high cost in the middle are due to the successive optimization runs as said earlier. Slight improvements in the cost are observed for Supporting Postures 1 and 3, due to the successive optimization runs.

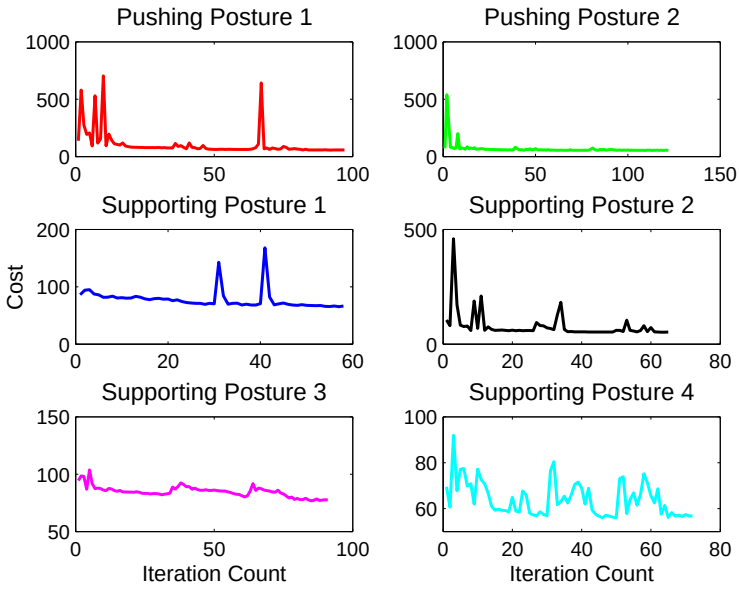


Figure 5-4: Cost convergence of both Pushing & Supporting postures for a pushing force of 117 N.

Minimum Cost

The various criteria cost as discussed in previous sections, are plotted for each posture at its optimum position (i.e minimum cost) determined through optimization. The various criteria cost plot obtained is shown in figure 5-5. The postures can be arranged in the following order according to the total cost incurred as follows,
PushPos2 < SupPos2 < PushPos1 < SupPos4 < SupPos1 < SupPos3.

- Torque costs are relatively higher for supporting postures 1,3 and 4 when compared to the other postures, as the former involves making supporting contacts with the objects which is slightly farther from the robot and the contact has to be made at a higher location to avoid contact slipping.
- Stability cost is higher for supporting postures 1 and 3, as the left foot contact is restricted to remain on the supporting plane, which restricts the robot from increasing its support polygon. However, the stability is kept within check due to the reaction force at the support contact, which counteracts the destabilizing moment.
- Friction cost is slightly lower for supporting postures 1, 2 and 4, but is higher for supporting posture 3. Since the pushing force to be exerted by the robot is less, there is not enough normal force at the support contact to lower the friction cone angle (less slipping).

Overall, PushPos2 seems to be the best posture that the robot can adapt for exerting a force of 117 N.

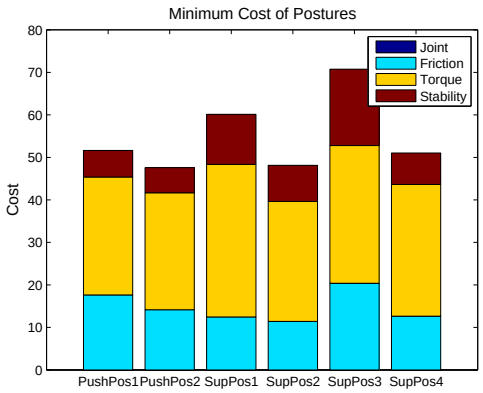


Figure 5-5: Various criteria cost for each posture to exert pushing force of 117 N.

Optimum Postures

Figures 5-6 and 5-7 show the initial and its corresponding optimum posture determined through optimization. The following observations are made from the posture plots.

- Significant difference is observed between the initial posture and the optimum posture obtained through optimization for all the 6 cases.
- In both the optimum postures obtained for pushing, the feet are kept apart to increase the support polygon, thereby the robot gets into a stable position for exerting the pushing force of 117 N.
- The pushing contacts are always maintained close to the CoG of the body, which results in exerting the pushing force effectively onto the object. The contacts are neither too low nor too high with respect to the torso of the body.
- The CoG of the body is moved forward appropriately, such that enough friction force is maintained at the feet and at the pushing contacts. Moving it, too forward will reduce the normal force at the feet and increase the tangential force at pushing contacts, both will result in high friction cost and eventually will lead to contact slipping.

Optimization with Increased Pushing Force

To verify how the optimization module behaves if the pushing force is increased, the force to be exerted by the robot is increased to 234 N, and the optimization is repeated for all the postures as done before. Figure 5-8 shows the various criteria cost for all 6 postures computed at their optimum configurations, and figure 5-9 shows the optimum postures obtained through optimization.

Changes in the Cost:
The posture can be ordered according to their respective overall cost as follows.
SupPos2 < SupPos1 < SupPos3 < SupPos4 < PushPos2 < PushPos1.
For generating high pushing forces, adapting simple pushing postures involve high stability

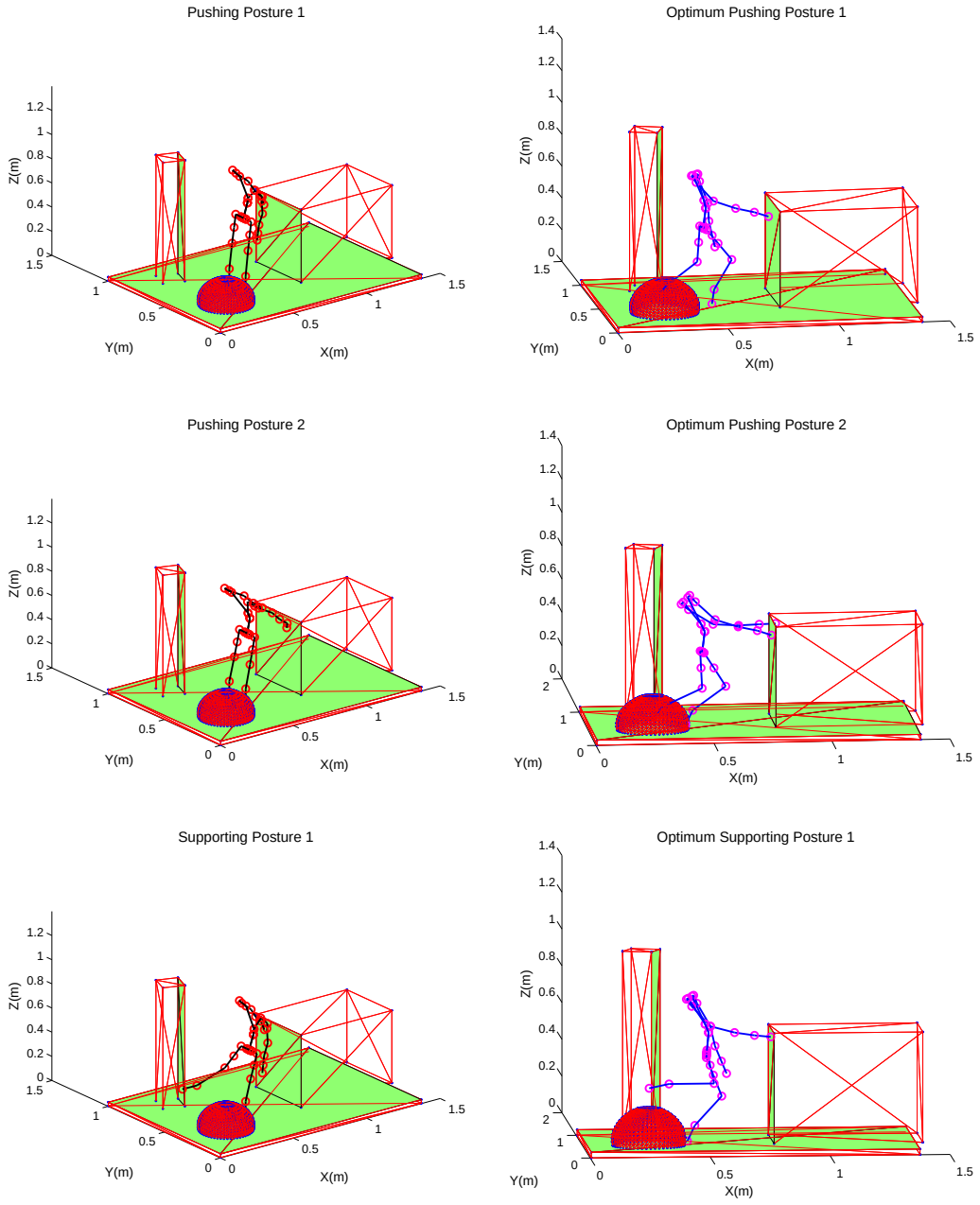


Figure 5-6: Initial and Optimum Configurations for Pushing Postures 1 & 2 and Supporting posture 1 for a humanoid to exert a force of 117 N.

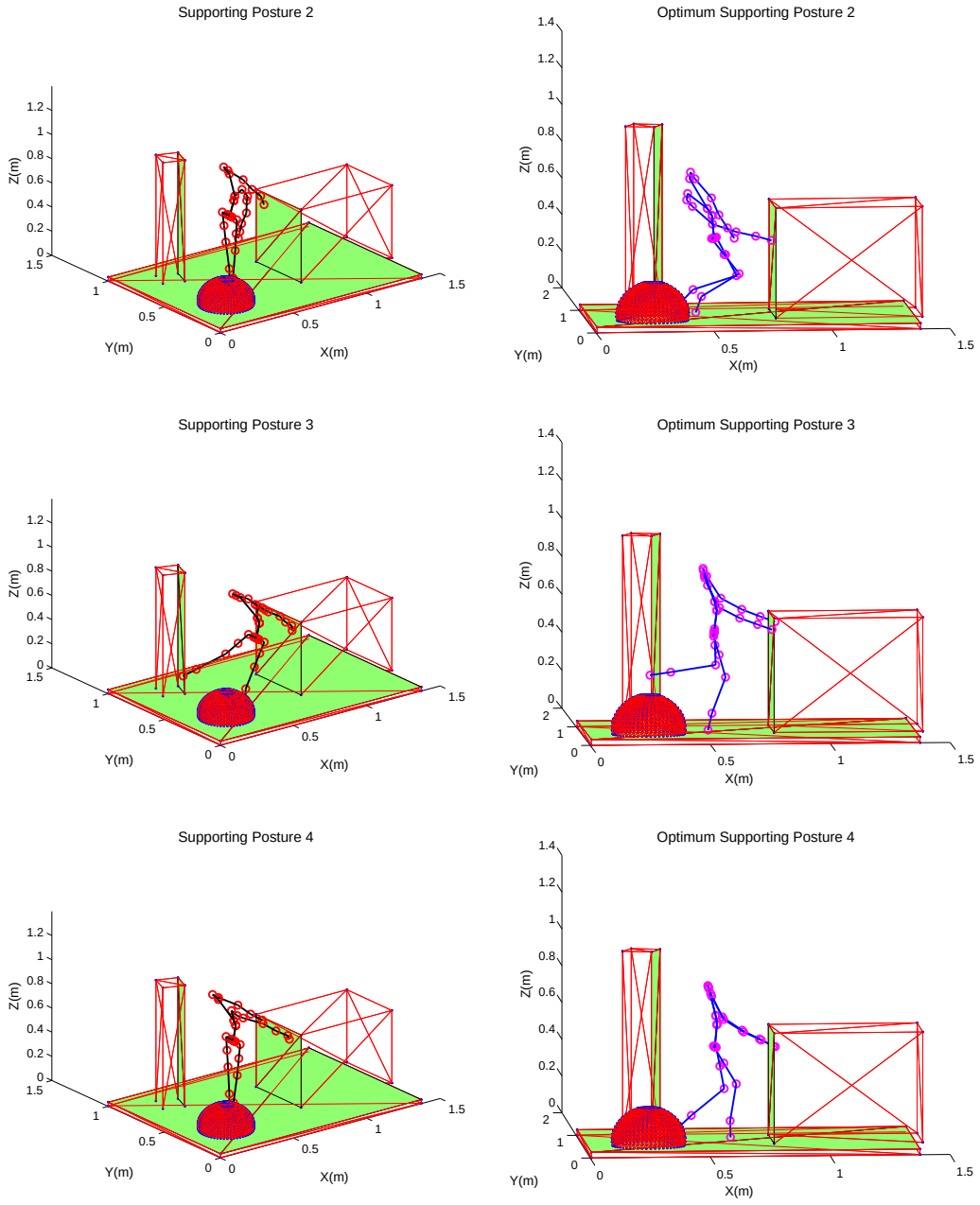


Figure 5-7: Initial and Optimum Configurations for Supporting Postures 2,3 & 4 for a humanoid to exert a force of 117 N.

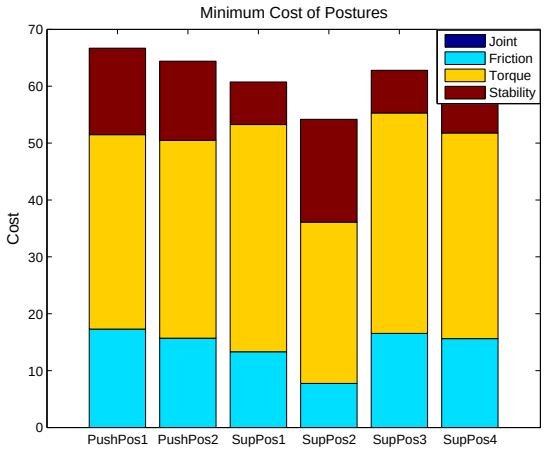


Figure 5-8: Various criteria cost for each posture to exert pushing force of 234 N.

and friction cost when compared to the supporting postures which involve relatively low friction and stability cost (except for SupPos2), but high torque cost. The best posture for exerting a force of 234 N can be either SupPos1 or SupPos3 as these postures are more stable.

Changes in the Posture:

- In both the pushing postures, support polygon is increased further by moving the feet apart. The destabilizing moment is countered by moving the CoG forward, but at the same time to maintain adequate friction at both the feet, the front foot is also moved forward. Pushing contacts are lowered to reduce the magnitude of destabilizing moment and proportionately CoG is also lowered to exert the pushing force, effectively.
- Similar to the pushing postures, in supporting postures 1 and 3 also lowering of pushing contacts and CoG is observed for the same reason as stated above.
- In supporting postures 2 and 3, the robot tries to increase the support polygon by moving the fixed leg backward (-x direction) and also tries to counteract the destabilizing moment by moving the CoG forward.

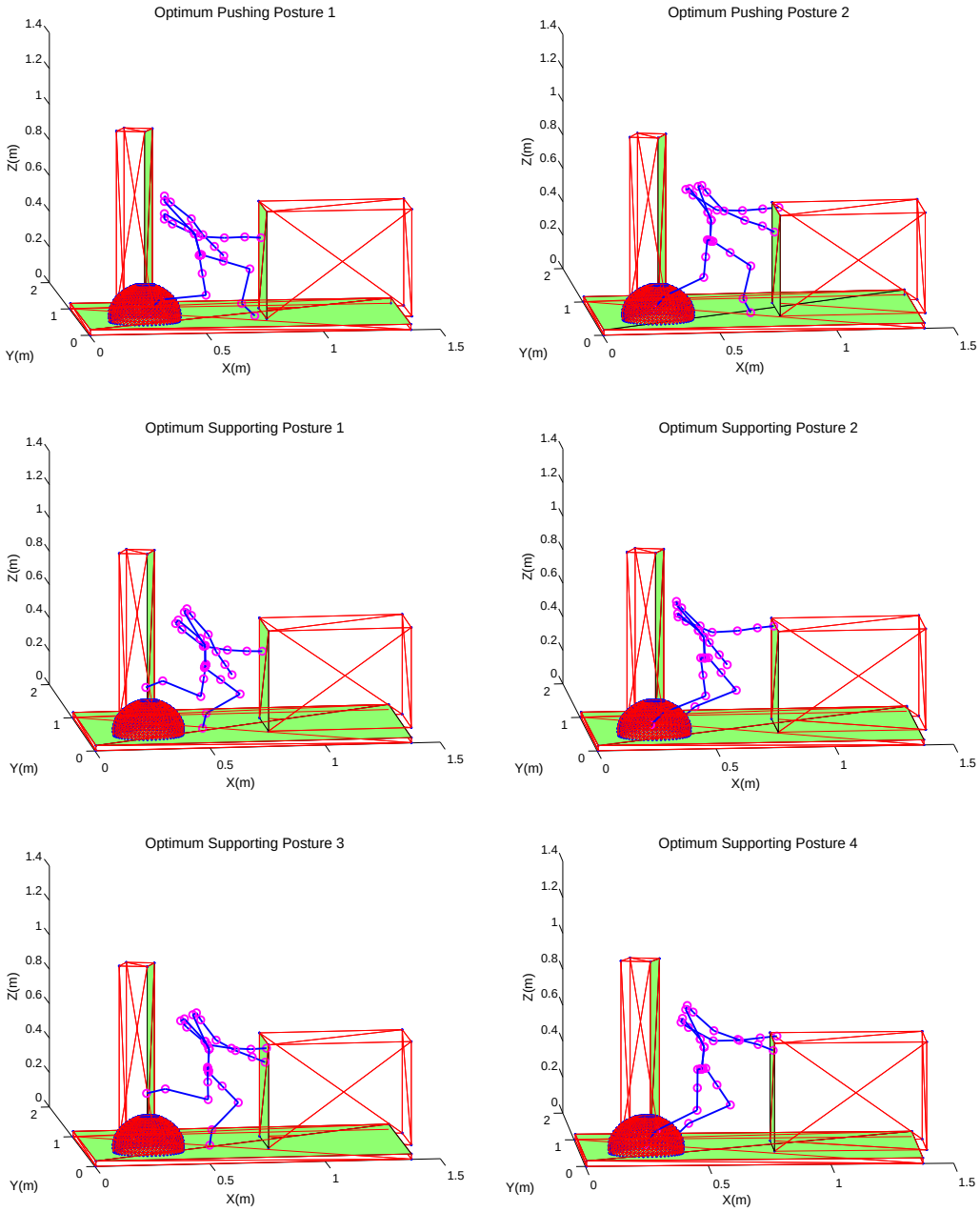


Figure 5-9: Optimum Configuration for all 6 postures, for a humanoid to exert a force of 234 N.

Chapter 6

Multitasking Control Framework

6-1 Different Control Frameworks

Multi-Limb robots are highly redundant in terms of their DOF, which proportionately increases their expected capabilities. Most of the tasks carried out by humans involve many sub-tasks or co-existing tasks which conflict with each other at times. For example, grasping a bottle is a major task and it includes several sub-tasks such as maintaining stability, keeping the feet in contact, etc. So humans tend to negotiate between these tasks, i.e. they try to do each task without violating other tasks or at least by compromising the less important tasks to some extent. So multi-limb robots which are anticipated to realize such tasks, should have some control framework to let them adapt human policy, as stated above.
Several control frameworks have been developed in reference to this, since the late 20th century. Oussama Khatib and Sentis [26, 35, 36] proposed an operational space control framework based on task prioritization. Mistry and Righetti [37, 38], proposed similar framework in joint space by making use of the null spaces of cartesian tasks. For making the priority levels compliant Federico Moro [39], came up with a notion of attractors for stability, joint limits, task, etc. and weighing it according to its importance. While researchers such as Inoue [40], devised general evaluation functions, assessing manipulability, stability and posture feasibility with respect to any given task and providing means to manipulate them so tasks becomes feasible. Some optimization based frameworks have also been developed such as Todorov's Contact Invariant Optimization [41], in which a multi-limb robot plans its own actions just with high level goals as inputs.
For the pushing task of multi-limb robots, which in itself has several sub-tasks, the OSC framework is selected. Compared to other frameworks, with OSC it is not only possible to operate low-level tasks within the null space of high-level tasks, but also it decouples the dynamics associated with each task. In this way, each task can be operated without any interference from other tasks. Due to this task dynamics decoupling OSC framework is preferred over other frameworks for the pushing task considered here.

6-2 OSC Framework

A general control framework proposed by Oussama Khatib [26] for achieving any task by means of certain desired whole body motion coordination of a robot is briefly explained. Based on the hypothesis that humans perform multi-tasks in a hierarchical manner, an operational space control (OSC) framework is proposed which prioritizes the tasks. Using OSC formulation the task level dynamics is obtained, which is then used to control the posture objectives in a dynamically consistent way. This formulation basically decouples the primary and secondary tasks by defining the secondary task objectives within the primary task space. In this way there is no interaction between the tasks and also the dynamics are compensated in their respective spaces.

Task and Posture decomposition: The task dynamic behavior is obtained by projecting the robot dynamics into the space associated with the task.

$$\bar{J}_t^T(A\ddot{q} + b + g = \Gamma) \Rightarrow \Lambda_t(x_t)\ddot{x}_t + \mu_t(x_t, \dot{x}_t) + p_t(x_t) = F_t \tag{6-1}$$

where $\Lambda_t(x_t)$ is the $m \times m$ kinetic energy matrix associated with the task, $\mu_t(x_t, \dot{x}_t), p_t(x_t)$ and F_t are respectively the centrifugal and Coriolis force vector, gravity force vector and generalized force vector acting in operational space.

The torque corresponding to the task behavior and the torque that only affects posture behavior in the null space can be obtained as follows

$$\Gamma = \Gamma_{task} + \Gamma_{posture} \tag{6-2}$$

$$\begin{aligned} \Gamma_{task} &= J_t^T F_t \\ \text{where, } F_t &= \hat{\Lambda}_t F_t^* + \hat{\mu}_t + \hat{p}_t \end{aligned} \tag{6-3}$$

F_t^* is the force level input and $\hat{\cdot}$ refers to the estimates of the various components of the dynamic model.

$$\begin{aligned} \Gamma_{posture} &= N_t^T \Gamma_p \\ \text{where, } N_t^T(q) &= [I - J_t^T \bar{J}_t^T] \end{aligned} \tag{6-4}$$

$$\Gamma = \Gamma_{task} + \Gamma_{posture} = J_t^T F_t + N_t^T \Gamma_p \tag{6-5}$$

Posture dynamic behavior and control: The task dependent torque decomposition from equation 6-5 can be rewritten as

$$\Gamma = J_t^T F_t + N_t^T (J_p^T F_p) \tag{6-6}$$

The posture torque can be rewritten as

$$\Gamma_{posture} = (J_p N_t)^T F_p \tag{6-7}$$

where, $J_{p|t} = J_p N_t$ is the instantaneous space of posture motion that is consistent with the task. $J_{p|t}$ is called the task-consistent posture Jacobian. The full description of the sub-task dynamic behavior from the projection can be written as

$$\bar{J}_{p|t}^T[A\ddot{q} + b + g = \Gamma_{task} + \Gamma_{posture}] \Rightarrow \Lambda_{p|t}\ddot{x}_{p|t} + \mu_{p|t} + p_{p|t} = F_{p|t} \tag{6-8}$$

6-2-1 Support Consistent OSC

In the previous section, OSC was explained and how torque associated with different tasks are calculated and finally added together to form a single vector of control torques which will be applied for task realization. In this section, OSC for floating base systems which maintain active support contacts while carrying out task will be explained and also various dynamic terms involved in computing the control torques. Since the multi-limb robot which will be simulated later for the evaluation of the General Pushing controller is a floating base system, the concept briefed here will be applied directly.
Supporting contact at the robot's feet affect the passive DOF of a floating base system, which in turn will affect the kinematics, dynamics and control torques. These contacts also define the stability of the robot. So, in order to perform any task stably, the task dynamics should be consistent with the support contacts, i.e. velocity (v_s) and accelerations ($\dot{v}_s$) at the contact points should be equal to 0. The equation of motion of a floating base system with support contacts and reaction forces (F_r) at those contacts due to gravity can be written as follows.

$$A \begin{bmatrix} \dot{v}_b \\ \ddot{q} \end{bmatrix} + b + g + J_s^T F_r = \begin{bmatrix} 0 \\ \Gamma \end{bmatrix} \tag{6-9}$$

where, $A \in \mathbb{R}^{(6+n)\times(6+n)}$ is inertia matrix, b and $g \in \mathbb{R}^{6+n}$ are coriolis and gravity forces, $J_s \in \mathbb{R}^{k\times(6+n)}$ is the jacobian for k support contacts and $\Gamma \in \mathbb{R}^n$ is the actuated control torque vector.
Equation of motion at the supporting contacts can be obtained by multiplying the above term 6-9 by $J_s A^{-1}$ and applying the following equality.

$$\dot{v}_s = J_s \begin{bmatrix} \dot{v}_b \\ \ddot{q} \end{bmatrix} + \dot{J}_s \begin{bmatrix} v_b \\ \dot{q} \end{bmatrix} \tag{6-10}$$

$$\dot{v}_s - \dot{J}_s \begin{bmatrix} v_b \\ \dot{q} \end{bmatrix} + J_s A^{-1}(b+g) + J_s A^{-1} J_s^T F_r = J_s A^{-1} S^T \Gamma \tag{6-11}$$

where, $S = \begin{bmatrix} 0_{n\times 6} & I_{n\times n} \end{bmatrix}$ is the selection matrix for selecting the actuated control torques. Applying the constraint $\dot{v}_s = 0$ to equation 6-11 and solving for F_r, the following estimate can be obtained.

$$F_r = \bar{J}_s^T S^T \Gamma - \bar{J}_s^T (b+g) + \Lambda_s \dot{J}_s \begin{bmatrix} v_b \\ \dot{q} \end{bmatrix} \tag{6-12}$$

where,

$$\Lambda_s = (J_s A^{-1} J_s^T)^{-1} \tag{6-13}$$

is the inertia at the supporting contacts.

$$\bar{J}_s = A^{-1} J_s^T \Lambda_s \tag{6-14}$$

is the dynamically consistent generalized inverse of J_s.
With the above equation, the equation of motion given in 6-9 can be written in the following form which is consistent with the support contacts.

$$A \begin{bmatrix} \dot{v}_b \\ \ddot{q} \end{bmatrix} + N_s^T (b+g) + J_s^T \Lambda_s \dot{J}_s \begin{bmatrix} v_b \\ \dot{q} \end{bmatrix} = N_s^T S^T \Gamma \tag{6-15}$$

where,

$$N_s = I - \bar{J}_s J_s \tag{6-16}$$

is the dynamically consistent null space of supporting contacts.

Task Dynamics and Control

Now considering that J is the jacobian of the task that has to be executed, with the support constraints satisfied, the generalized inverse of the task consistent with the support constraints ($J_{t|s}$) can be obtained as follows

$$\bar{J}_{t|s} = A^{-1} J_{t|s}^T \Lambda_{t|s} \tag{6-17}$$

where,

$$\Lambda_{t|s} = (J_{t|s} A^{-1} J_{t|s}^T)^{-1} \tag{6-18}$$

is the task specific inertia consistent with the support constraints.
The task space equation of motion can be obtained by multiplying the equation 6-15 by $\bar{J}_{t|s}^T$

$$\Lambda_{t|s}\ddot{x} + \mu_{t|s} + p_{t|s} = \bar{J}_{t|s}^T (S N_s)^T \Gamma \tag{6-19}$$

where,

$$\mu_{t|s} = \bar{J}_{t|s}^T b - (\Lambda_{t|s}\dot{J}_{t|s} + \bar{J}_{t|s}^T J_s^T \Lambda_s \dot{J}_s) \begin{bmatrix} v_b \\ \dot{q} \end{bmatrix} \tag{6-20}$$

is the Coriolis & Centrifugal forces associated with the task, consistent with the support constraints.

$$p_{t|s} = \bar{J}_{t|s}^T g \tag{6-21}$$

is the gravitational force associated with the task, consistent with the support constraints.
The vector of control forces F, driving the task in task space can be written as

$$F = \Lambda_{t|s}\ddot{x} + \mu_{t|s} + p_{t|s} \tag{6-22}$$

By setting $\ddot{x} = \ddot{x}_{ref}$, the desired task can be achieved and $\ddot{x}_{ref}$ is computed by adopting the ***velocity saturation principle***. This prevents erratic increase in task velocity and also ensures goal directed movement. This is implemented as follows.

$$\ddot{x}_{ref} = -k_v(\dot{x}_{inst} - v_v v_{des}) \tag{6-23}$$

where,

$$v_{des} = \frac{k_p}{k_v}(x_{inst} - x_{goal}), \quad v_v = min(1, \frac{v_{max}}{||v_{des}||}) \tag{6-24}$$

In equations 6-23 and 6-24, k_v and k_p are the proportional and derivative gains, x_{inst} and $\dot{x}_{inst}$ are instantaneous task position and velocity, v_{max} is the maximum task velocity allowed, v_{des} is the desired task velocity, v_v is the velocity saturation factor and x_{goal} is the target position.
If $v_{des} < v_{max}$, then equation 6-23 becomes,

$$\ddot{x}_{ref} = -k_v(\dot{x}_{inst}) - k_p(x_{inst} - x_{goal}) \tag{6-25}$$

If $v_{des} > v_{max}$, then

$$\ddot{x}_{ref} = -k_v(\dot{x}_{inst} - v_{max}\frac{v_{des}}{||v_{des}||}) \tag{6-26}$$

Using equations 6-23 and 6-22, the torque vector Γ yielding linear control of task force and acceleration can be written as follows.

$$\Gamma = (S\bar{N}_s)^T \bar{J}_{t|s}^T F \tag{6-27}$$

where, $S\bar{N}_s = A^{-1}(SN_s)^T\ (SN_sA^{-1}(SN_s)^T)^+$ is the support consistent generalized inverse of SN_s.
For multiple tasks the control torque can be computed using the following equation.

$$\Gamma = J^{*T}F + N^{*T}\Gamma_0 \tag{6-28}$$

where, Γ_0 is the redundant torque which can be used for additional tasks, J^{*T} is nothing but $(S\bar{N}_s)^T\bar{J}_{t|s}^T$ and $N^* = I - \bar{J}^*J^*$ is the null space of the previous task.
The second term in equation 6-24 projects the control force associated with the second task into the null space of task forces and accelerations of the first task.
The above explained method is recursively applied for additional tasks and the total torque Γ is used for controlling the robot. Please refer [28] for additional information and detailed explanation. The control framework developed for the pushing task of multi-limb robots is based on the above principle.

6-3 Posture Data Preparation

To build a general control architecture for the pushing task of multi-limb robots, the selected posture details are given in a standard format in the form of a ".txt" file. This file contains details such as Posture ID, number of moving active and inactive contacts and their respective IDs and number of fixed contacts and their IDs. The file also contains the IDs of moving objects, supporting objects and fixed active object. A sample text file with all the specified details is shown below.

```
Selected Posture for execution:
05
No. of moving Active contacts:
1
No. of moving InActive contacts:
2
%% Contact IDs
Moving Active Contacts:
0
Fixed Active Contacts:
1
Moving InActive Contacts:
0
1
```

```
No. of Active Pushing contacts:
0
No. of InActive Pushing contacts:
2
%% Object IDs
Moving Active Object:
2
Fixed Active Object:
1
Moving InActive Object:
0
0
```

A text file as specified above is forwarded to the OSC control architecture module to build a sequence of tasks.

6-4 Control Architecture

The control architecture designed for the pushing task of multi-limb robots is shown in figure 6-1. It requires two input ".txt" files, Optimal Posture Details and Optimal Posture Values, the former is shown in the earlier section and the latter contains the contact and the CoG optimal positions. With the input details, separate tasks are created for each moving contact and a single task for the CoG and fixed active contacts. Both CoG and the fixed active contact movement will be executed simultaneously by means of a single task, with the former being primary and the latter being secondary. For instance, if there are 2 moving active, 1 moving inactive and 2 fixed active contacts, then the no. of tasks will be 2+1+1 = 4 tasks. A simple proportional control on the joint is used to maintain the remaining free inactive contacts at a predefined configuration (for avoiding collision with surrounding objects). A task data structure is created and each task's details such as which contact is moved, target position, max velocity, corresponding jacobian, etc. are stored in it. The tasks are prioritized and each task is added iteratively upon the completion of previous task. The tasks are prioritized in the following order (higher - lower level).

- CoG + Fixed Active Contact Movement
- Pushing Contact Movement
- Supporting Contact Movement

Now with all the tasks having been executed i.e all the contacts and the CoG at their optimal positions, the pushing force is exerted onto the object which has to be moved. This is done by shifting the pushing contact's target position by dx amount in x direction i.e ($x_{goal} = x_{goal} + dx$). Upon pushing the object by dx amount, each task's target position are set back to the initial position, so that the robot gets back to its normal pose.

ZMP compensation:
During the pushing motion, the resisting force equal to the amount exerted on the object

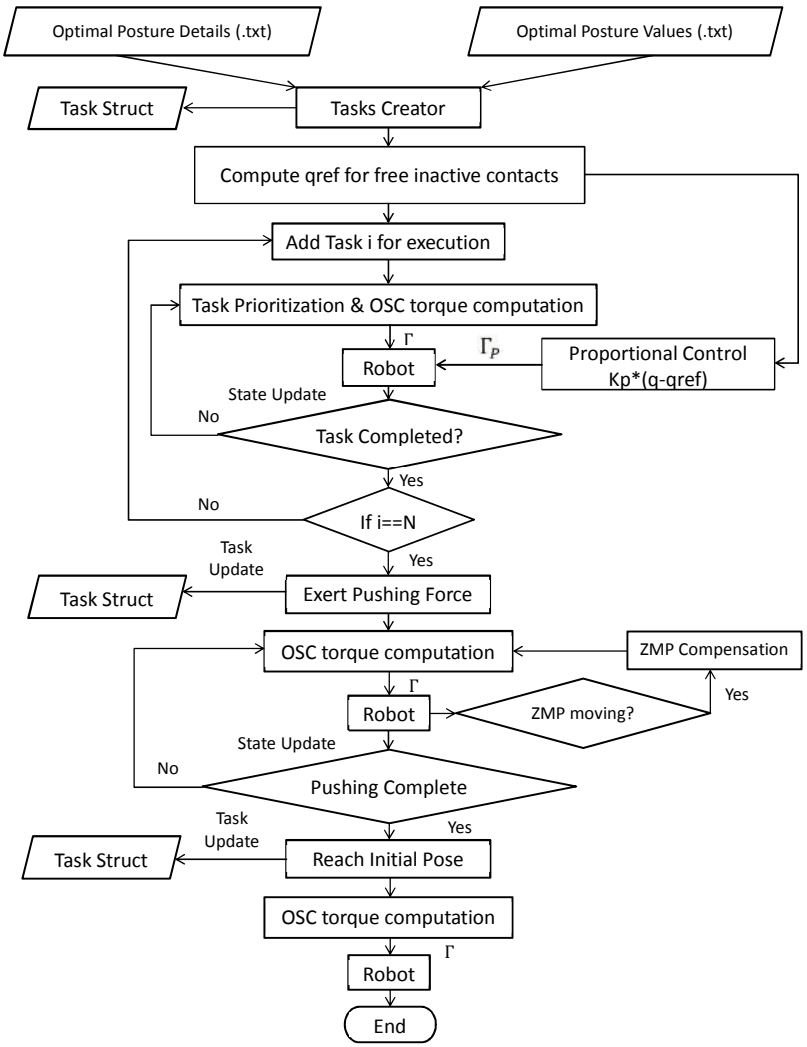


Figure 6-1: OSC Architecture.

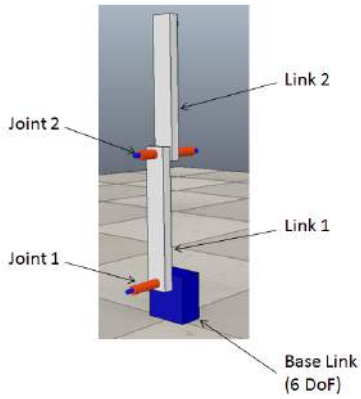


Figure 6-2: Double Pendulum model with 2 DOF for OSC validation

will be acting on the robot in the opposite direction, which will tend to destabilize the robot. Humans resist this force by propelling their CoG forward. Similarly, during the pushing motion, the ZMP of the robot is calculated and once it starts moving, the difference between the initial ZMP and the instantaneous ZMP is calculated and it is set as the target for CoG moving task. The ZMP computed here is quasi-static, i.e., at each time step the contact forces are measured and it is substituted in the equations 4-2 and 4-3 to determine the instantaneous ZMP values. The main reasons for adopting the quasi-static approximation of ZMP is, considering dynamics would necessitate the time-consuming computation of linear and angular momentum of the links which could affect (lower) the control update rate and also the accelerations of the links are not high enough (during pushing) to be considered here. This method of compensating the ZMP movement makes the robot to behave the same way as humans, i.e., counteract the resisting force and effectively apply the pushing force on the object.

6-5 OSC validation

The Operational Space Controller (OSC) is validated with a simple double pendulum model 8 DOF (2+6). The model has 2 actuated DOF and remaining 6 refers to the floating base DOF. Figure 6-2 shows the model with its links and joints.

OSC Tasks:
The OSC is validated by creating two tasks and executing the second task within the null space of the first one. The tasks assigned are given below.

1. Maintaining base contact - Primary task
2. Moving CoG in x direction ($X_{goal} = 0.015$m) - Secondary Task

The results obtained by simulating the model in V-Rep is shown in figure 6-3. In figure 6-4, it can be seen that the model's x value of CoG (X_{inst}) starts initially from 0 and it gradually moves towards the set target ($X_{goal} = 0.015$m). Since moving CoG is secondary and maintaining contact at the base link is primary, X_{inst} settles down close to 0.01m. The result obtained successfully validates the OSC framework implementation.

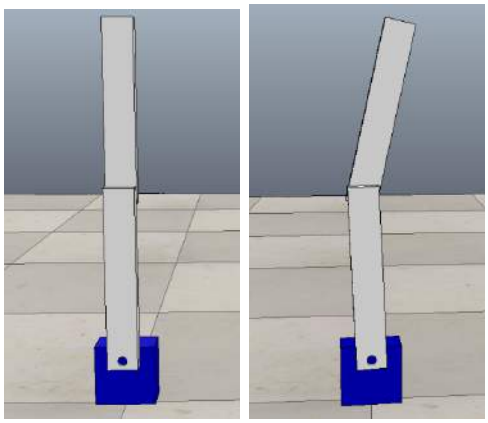


Figure 6-3: Left side figure shows the model at the beginning of simulation and right side figure shows the stable position of the model obtained with OSC

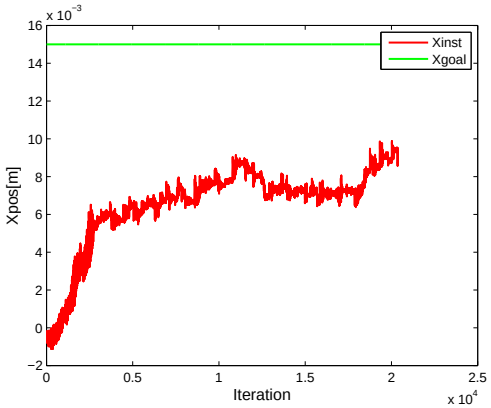


Figure 6-4: Instantaneous x value of CoG position, X_{inst} (red) of double pendulum and its corresponding goal position,X_{goal} (green)

Chapter 7

Final Evaluation and Simulation

7-1 3D Robot model and Environment

Robot Model

For the final evaluation of all the modules, a multi-limb robot which has enough active and inactive contacts to check extensively the capability of posture tree generation and posture optimization modules, and simple enough to make the OSC module slightly easier (in terms of control) was conceptualized. The multi-limb robot so designed for final evaluation is called 'OctoBot' shown below in figure 7-1. The robot model was designed and built using SolidWorks (3d modeling software).

Robot Specifications

Name - OctoBot
Mass - 35 kg

Figure 7-1: Multi-limb robot ('OctoBot') for final evaluation

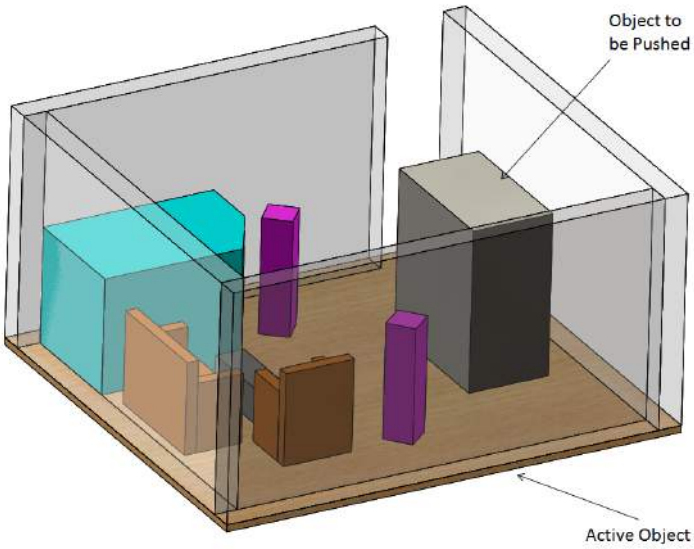


Figure 7-2: A typical living environment considered for final evaluation of the modules. Black rectangular box represents the object to be pushed, brown platform is the active object and all the other objects will be considered as potential supporting objects (except transparent ones, which are walls).

Degree of Freedom (DOF) - 32 (actuated) + 6(floating base)
No.of End-effectors - 8
No. of Active contacts - 4
No. of InActive Contacts - 4

Environment

The environment model considered here for final evaluation is more complex than the one considered for preliminary evaluation. The model is shown in figure 7-2, it's a typical living environment which has been considered here. Similar to the preliminary environment (figure 3-4), the environment considered here also has one pushing and one active object, but the latter has more supporting objects than the former one, making it more complex. So, with the environment shown in figure 7-2, the robot has more than 2 objects around it, to be considered for making support contact. Also in the present environment, no spherical, circular or curved objects are included, since it was understood during the preliminary evaluation of the modules that contacts with such objects need to be treated differently from other regular contacts and the methods to do that are not developed yet. One more reason for avoiding irregular shaped objects is, the creation of dynamic models (react to force) to represent the environment in the simulator becomes easier with regular shapes.

7-2 URDF generation

The 3d robot model shown in figure 7-1, has to be represented in a specific format in order to carry out dynamic and kinematic computations with the model for all three modules. The

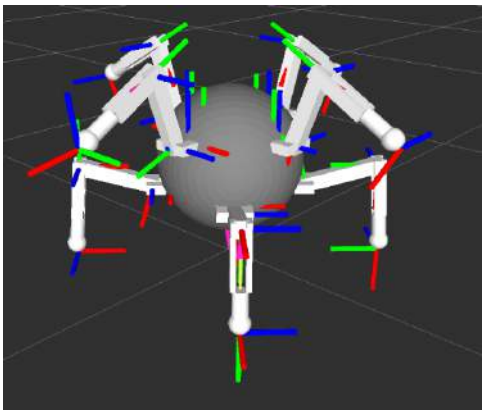


Figure 7-3: OctoBot.URDF file visualized in rviz (3d visualization tool for ROS).

format which is used here is URDF, which was already explained in Chapter 4. URDF can be generated manually for models with less DOF, since the model considered here has more DOF, it is generated automatically with minimal manual actions by using Solidworks-URDF plugin [1]. The generated URDF file is visualized and verified with rviz [2], a $3d$ visualization tool of ROS. The model visualized in rviz is shown in figure 7-3.

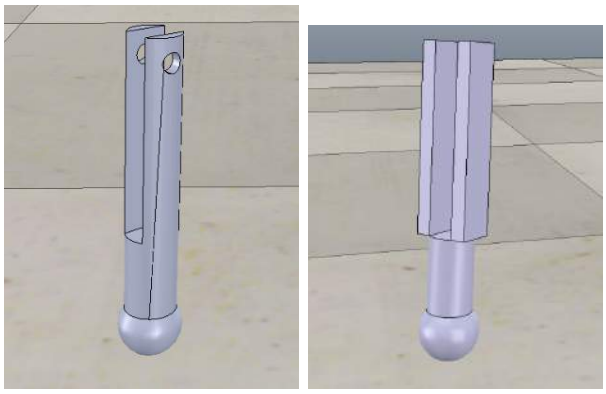
7-3 Simulator

The final evaluation of the General Pushing Controller developed in this thesis for multi-limb robots is done with the V-Rep simulator. V-Rep is a robot simulator, with an integrated development environment. Its control architecture is more distributed, i.e. each object/model can be controlled separately by means of embedded scripts, plugin, ROS node, remote API client or some custom solution, which makes it more versatile. The control scripts can be written in any languages such as C/C++, Python, Java, Lua, Matlab, Octave or Urbi.
The control mode selected for manipulating the multi-limb robot is torque mode and for it to perform well, the update rate has to be fast, without any communication lag. Since with ROS node and remote API client, there is high communication lag and with embedded scripts and Add-on, the execution speed is less, the best method is creating a controller plugin. Controller plugins are faster in executing the code and there is no communication lag, as it can operate synchronously. The controller plugin used here for controlling the multi-limb robot is written in C++ language.

7-4 Model preparation for simulator

For the simulator to perform well, the model should be made of pure shapes. Any $3d$ model developed with any modeling package usually contains a number of non-pure shapes and using it directly in the simulator for simulation will deteriorate its performance. Problems

[1] http://wiki.ros.org/sw_urdf_exporter
[2] http://wiki.ros.org/rviz



(a) Lower leg - visual model **(b)** Lower leg - dynamic model

Figure 7-4: Figure 7-4a represents the visual model and Figure 7-4b represents the dynamic model. The dynamic model is made of pure shapes (rectangles, cylinder and sphere).

such as, more time consumption for dynamic calculations, unstable simulations, jittery contacts, etc. are highly possible to occur. But making models with pure shapes wards off the aforementioned problems at the cost of its aesthetic appealing. So in order to balance both, two models are generally used in V-Rep. Model with non-pure shapes, i.e. model imported directly from a $3d$ modeling package are used as visual model and model with pure shapes are used for dynamic calculations. The latter model can be generated from the former one by shape editing or can be built separately. Figure 7-4a shows the visual and dynamic model of the lower leg part of OctoBot. In figure 7-4, it can be seen that the dynamic model doesn't have the curved edges and joint holes as observed in the visual model. For simulation purpose, the dynamic models are superimposed with the visual models and the former is hidden. All dynamic computations are performed with dynamic models, but its effect/movements are observed through the visual model.

Robot Model

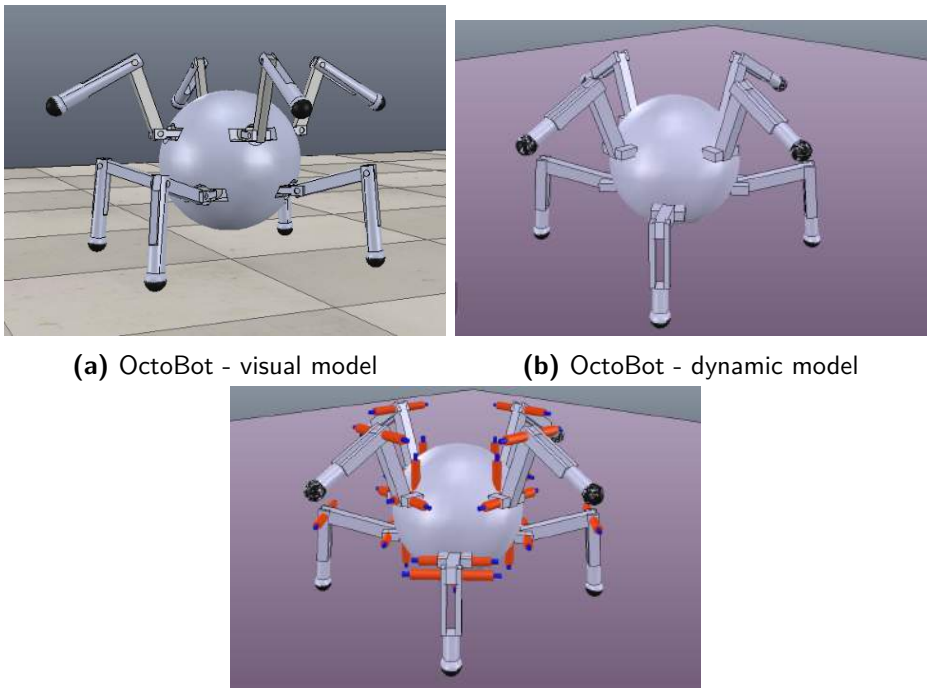
As explained above, similar operation was carried out for the entire robot model and the model so obtained is shown in figure 7-5.

Environment

The visual and dynamic model for the environment is also constructed the same as done for the robot and the resulting models are shown in figure 7-6. All objects are made static, except the one which has to be pushed.

7-5 Controller Modification

The control architecture shown in figure 6-1 is simplified in order to reduce the complexity of first task (i.e CoG and fixed active contact movement) and also to make it compatible



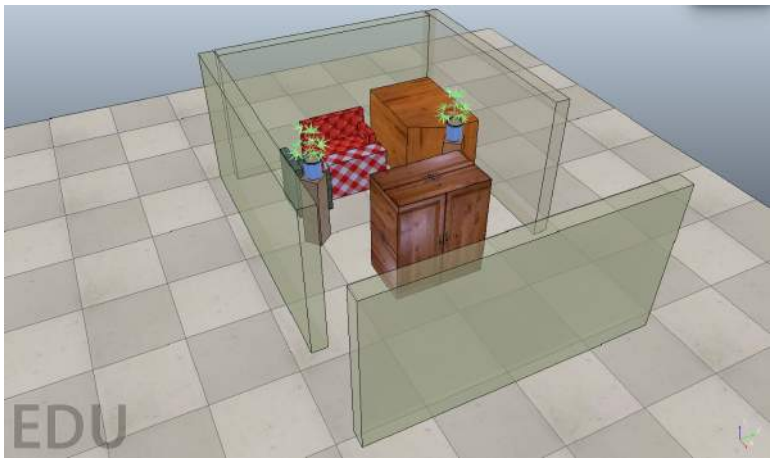
(a) OctoBot - visual model **(b)** OctoBot - dynamic model

(c) OctoBot - dynamic model with joints

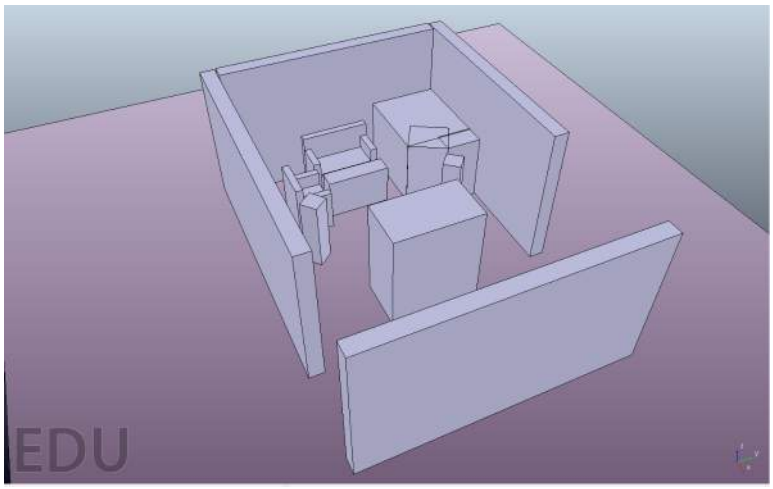
Figure 7-5: Figure 7-5a represents the visual model, Figure 7-5b represents the dynamic model (made of pure shapes) and Figure 7-5c shows the dynamic model along with the robot's joints.

with the simulator. The fixed active contacts (i.e. legs) are executed in parallel with the CoG moving task by means of a simple proportional control, instead of doing it by means of OSC. Since each task is operated within the null space of the previous tasks, making each fixed active contact movement as separate task results in frequent make and break contact scenario, which destabilizes the system. For the OctoBot robot considered here for simulation, the maximum possible number of tasks associated with moving fixed active contacts would be 4 and this is reduced to one, by operating it along with the CoG moving task, which is the primary task. By doing so, the active contacts (i.e. legs) are dragged to the desired position instead of making and breaking contact with the ground. This is done by pre-computing the joint values (q_{ref}) for the corresponding joints of each leg and with proportional gain k_p, the control torque is calculated as $\Gamma_p = k_p(q - q_{ref})$. This torque is added along with the OSC torque (Γ) and applied to the robot. Since the OSC gains are relatively higher than k_p, the fixed active contact tasks act as secondary to the primary CoG moving task. Hence, the chances of realizing the exact target positions for the fixed active contacts are minimal when compared to the CoG task. This method of controlling the fixed active contacts increases the stiffness of the legs and results in a more stable posture. The modified control architecture is shown in figure 7-7 and the OSC task order would be as shown below.

- CoG Movement
- Pushing Contact Movement
- Supporting Contact Movement



(a) Environment - visual model



(b) Environment - dynamic model

Figure 7-6: Figure 7-6a represents the visual model and Figure 7-6b represents the dynamic model(rectangles, cylinder and sphere).

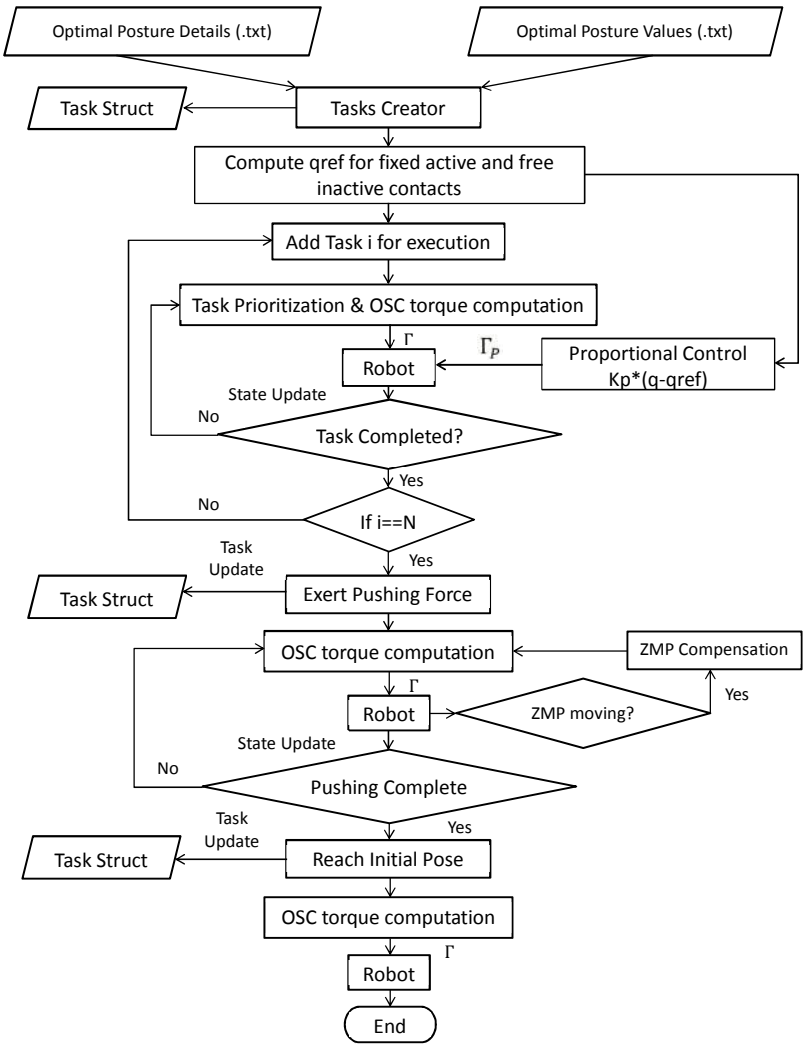


Figure 7-7: Simulation compatible OSC control Architecture.

Table 7-1: Posture Tree Generation Module Results

S.No	Item Name	No.
1	No. of Pushing Plane	1
2	No. of Supporting Objects	5
3	No. of Supporting Planes selected	7
4	No. of Active Plane	1
5	No. of Pushing Postures	3
6	No. of Supporting Postures	21

7-6 Posture Tree Generation Results

With the multi-limb robot (OctoBot) shown in figure 7-1 and the environment shown in figure 7-2, the Posture Tree Generation module is run and the results so obtained are shown in the following figure.
The numerical results are tabulated in table 7-1.
Figure 7-10 shows the generated posture tree for for the given multi-limb robot (OctoBot) and the environment. The total number of possible postures is 24, and they are shown in figures 7-11 and 7-12. The postures shown in the figure are in initial configuration, which will be optimized later.

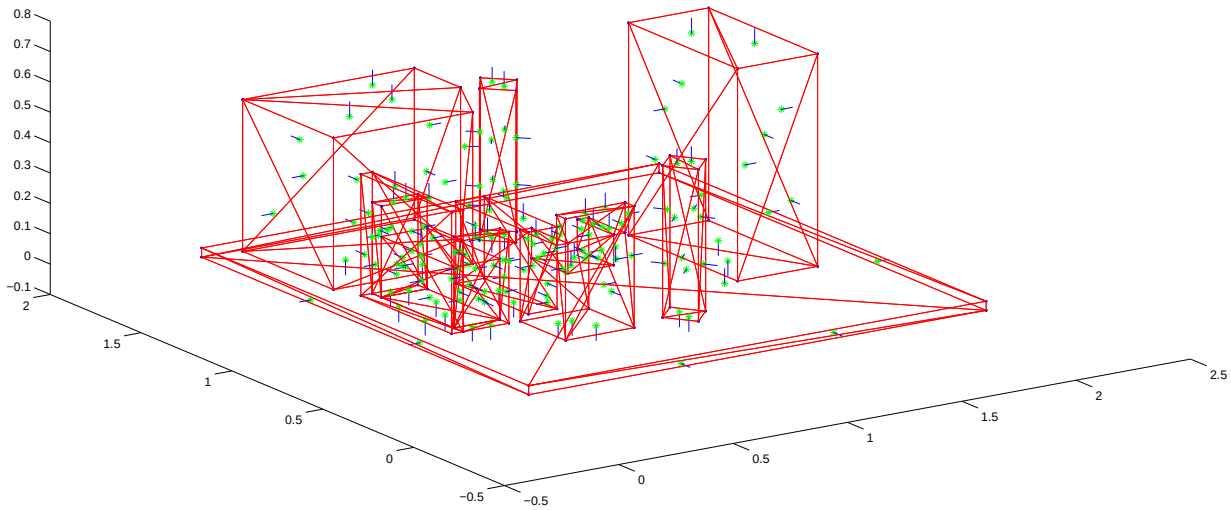
7-7 Posture Optimization Results

For running optimization, an approximate pushing force which the robot needs to exert on the object has to be given. The approximate pushing force is calculated as follows.

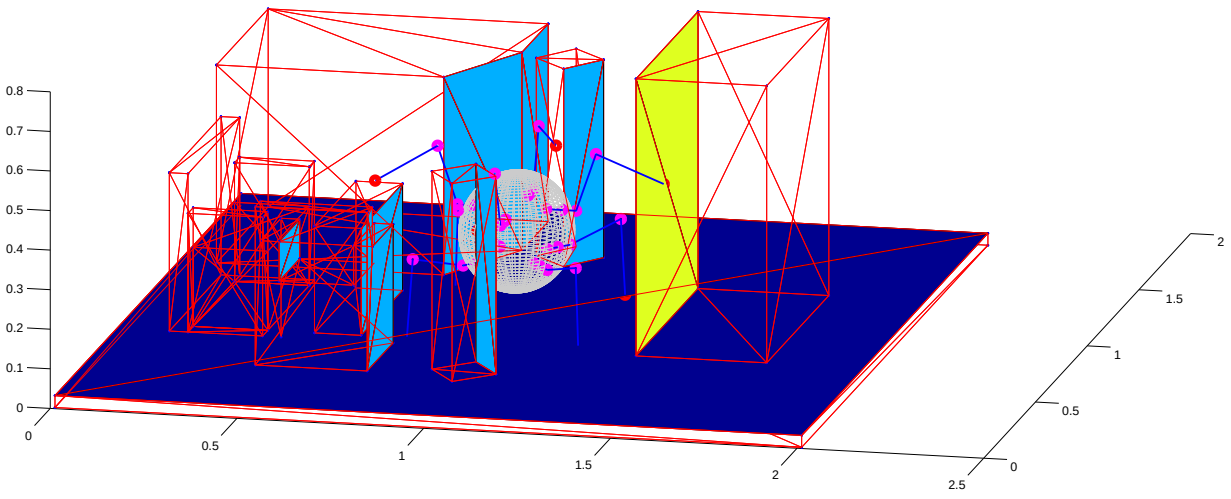
Pushing force calculation:
Density of the object, $\rho = 300\ kg/m^3$ (assumed)
Coefficient of friction, $\mu = 0.5$
Volume of the object, $V = L \times B \times H = 0.7 \times 0.6 \times 0.35 = 0.1470\ m^3$
Mass of the object, $m = \rho \times V = 300 \times 0.18 = 44.1\ kg$
Approximate pushing force, $F = \mu \times m.g = 0.5 \times (44.1 \times 9.8) = 220\ N$.

Criteria Cost for Postures:
All 24 postures are optimized for exerting pushing force, $F = 220\ N$. The optimization is carried out as done earlier for the Humanoid robot. Since there are too many postures to be optimized and multiple optimization runs didn't improve the cost significantly during preliminary evaluations, each posture is subject to only one optimization run. The criteria cost at optimum locations obtained for all postures are shown in figure 7-13. Based on the overall cost, less costing postures are selected. The selected postures along with their cost are tabulated in table 7-2. The optimal configuration determined through optimization for the least costing postures are shown in figures 7-14.

From the least costing postures, simple postures i.e. postures with at least three legs in contact with the active object (floor) are selected for simulation. Pushing posture 1, Supporting postures 6,8,12 and 15 are selected for simulation. These simple postures are more stable during the transition from the robot's initial configuration to each posture's optimal

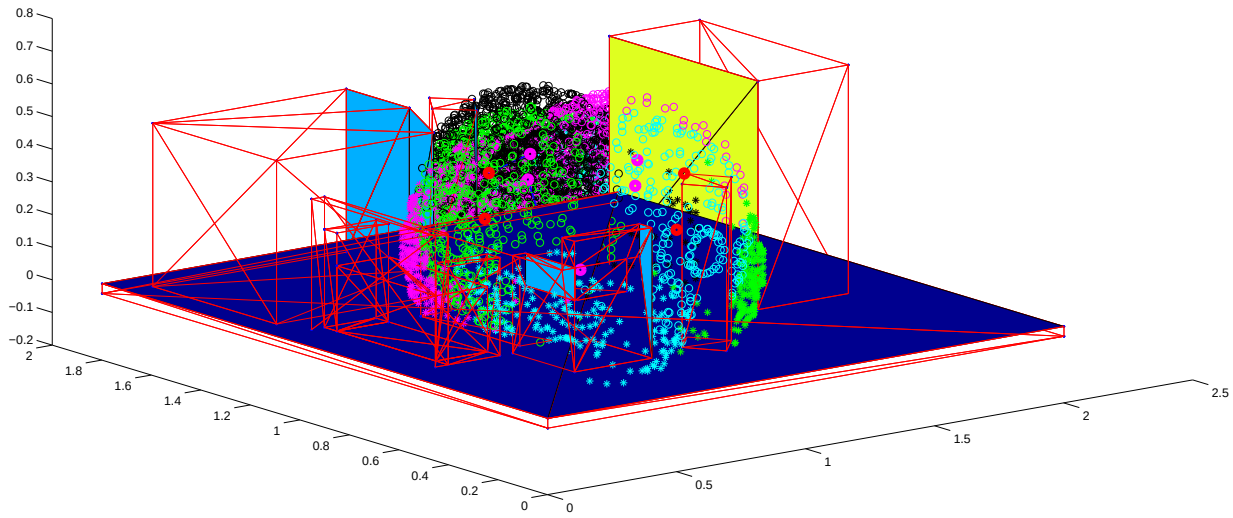


(a) Extracted environment from .stl files with stlreader.

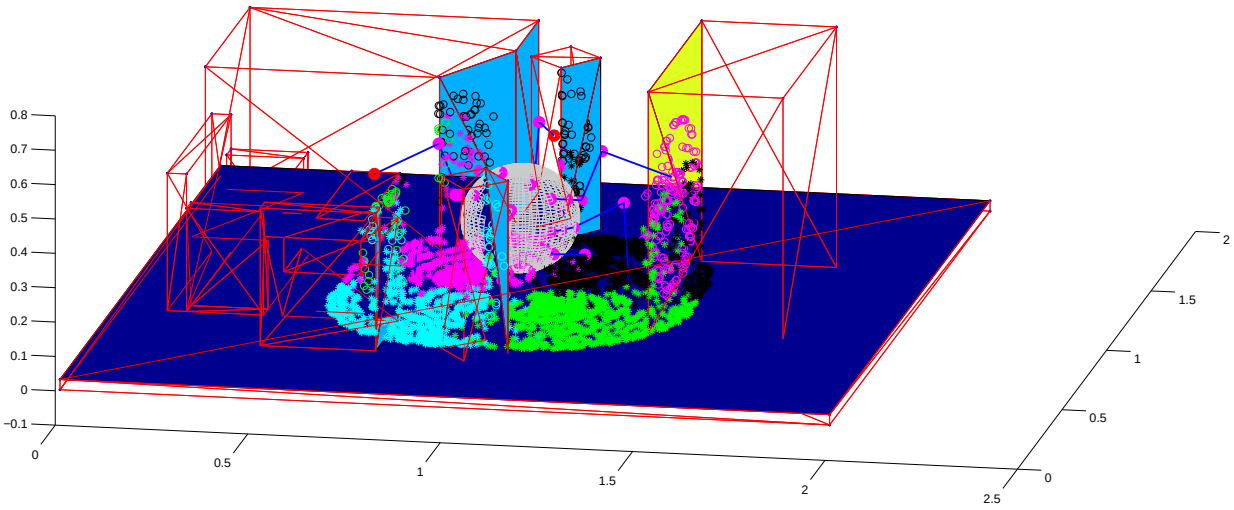


(b) Pushing (Shaded yellow), Supporting (light blue) and Active Planes (dark blue) selected from the environment.

Figure 7-8: Posture Tree Generation Module results for OctoBot robot with the living environment.Extracted environment and selected planes are shown here.



(a) All possible positions of each end-effector of OctoBot. Colored circles (magenta, cyan, green and black) represent inactive and colored stars represent active contacts.



(b) Filtered end-effectors points projected on to pushing, supporting and active planes.

Figure 7-9: Posture Tree Generation Module results for OctoBot robot with the living environment. 7-9a shows the possible positions of each end-effector of the robot and 7-9b shows the points filtered with respect to each plane.

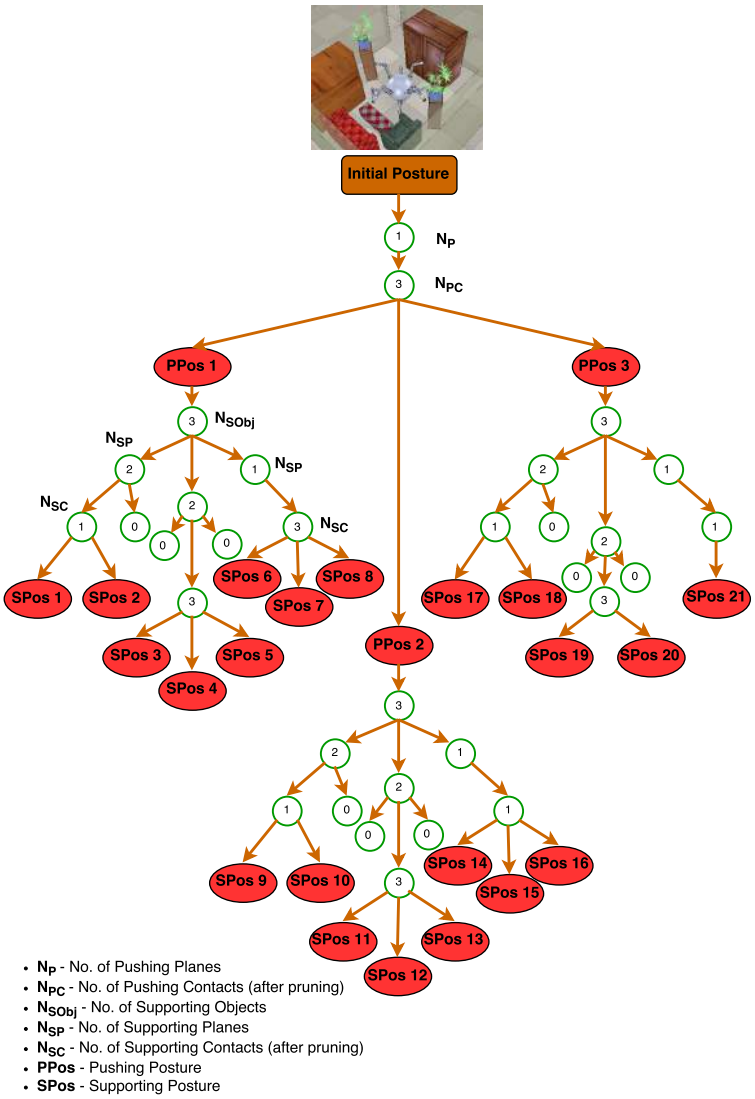
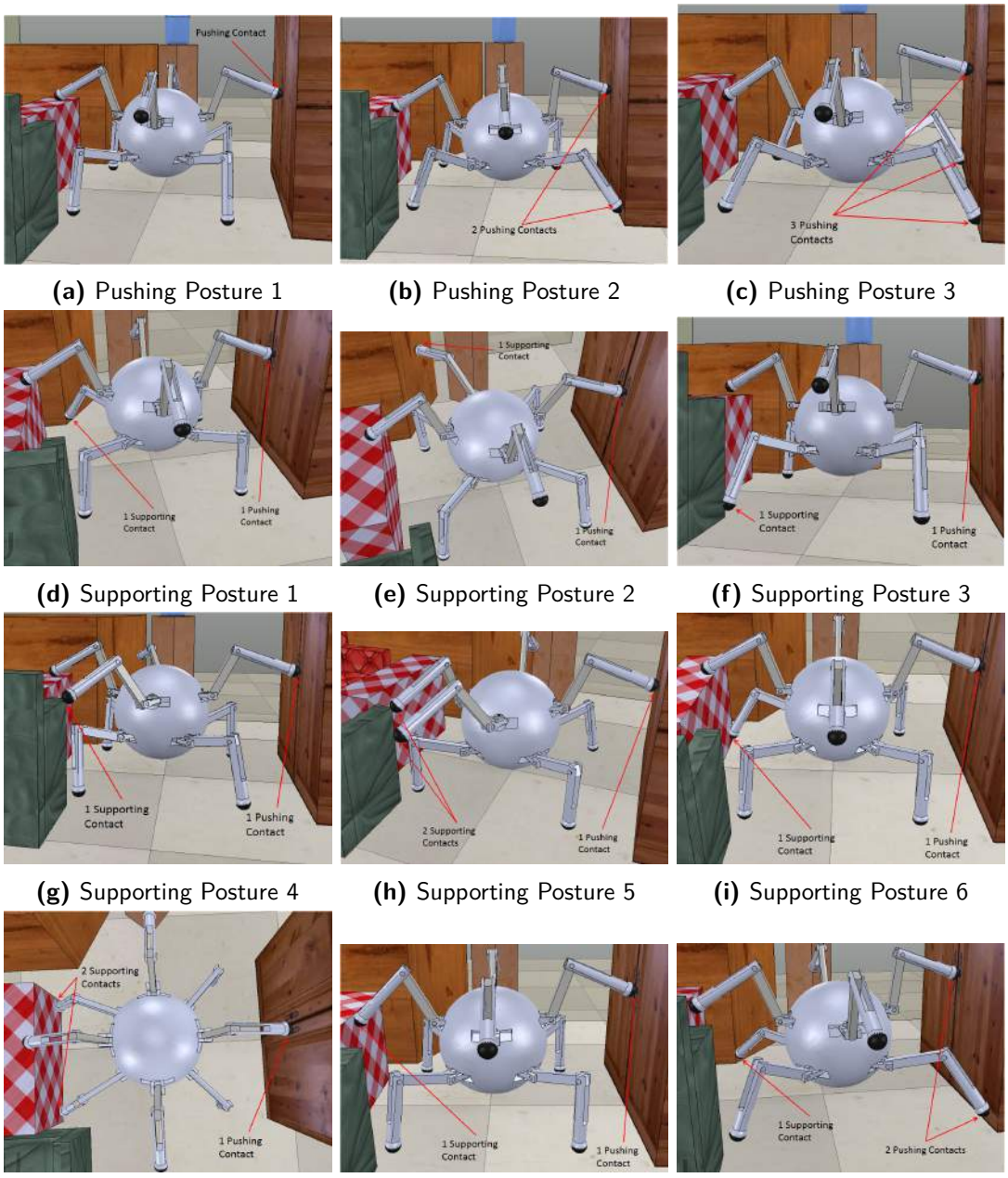


Figure 7-10: Schematic view of Posture Tree generated with the multi-limb robot (OctoBot) and the given environment.

Table 7-2: Least Costing Postures

S.No	Postures	Cost
1	PushPos1	76.9219
2	SupPos6	51.1061
3	SupPos7	56.9407
4	SupPos8	49.3240
5	SupPos12	76.8888
6	SupPos14	65.6599
7	SupPos15	58.1608
8	SupPos16	52.722



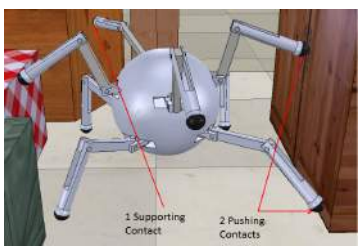
(a) Pushing Posture 1 **(b)** Pushing Posture 2 **(c)** Pushing Posture 3

(d) Supporting Posture 1 **(e)** Supporting Posture 2 **(f)** Supporting Posture 3

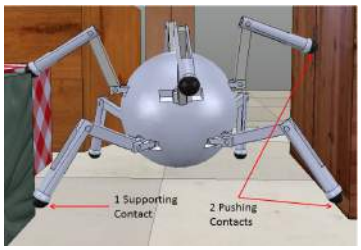
(g) Supporting Posture 4 **(h)** Supporting Posture 5 **(i)** Supporting Posture 6

(j) Supporting Posture 7 **(k)** Supporting Posture 8 **(l)** Supporting Posture 9

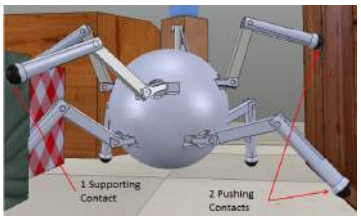
Figure 7-11: Posture generated for OctoBot with Posture Tree Generation module. Pushing Postures (1-3) and Supporting Postures (1-9) are shown.



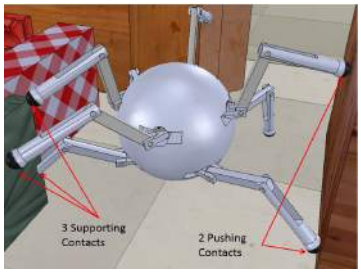
(a) Supporting Posture 10



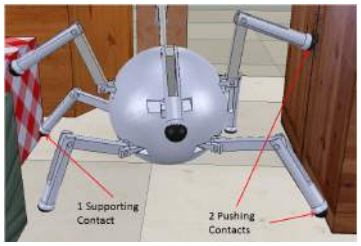
(b) Supporting Posture 11



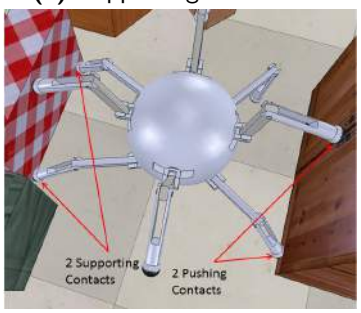
(c) Supporting Posture 12



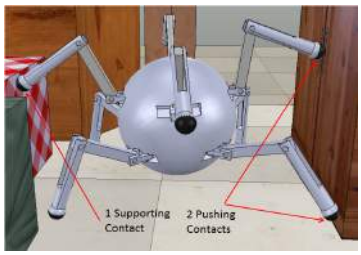
(d) Supporting Posture 13



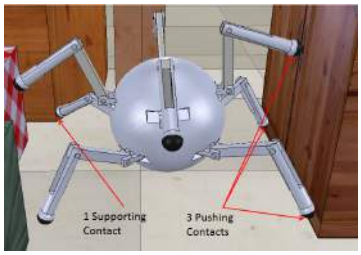
(e) Supporting Posture 14



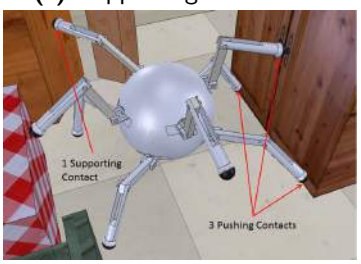
(f) Supporting Posture 15



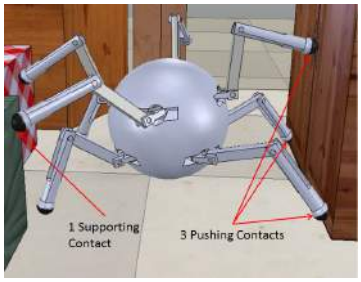
(g) Supporting Posture 16



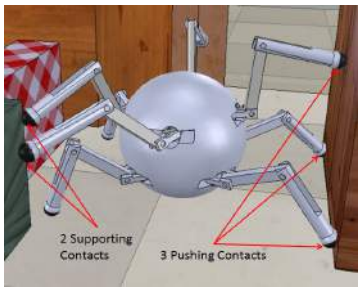
(h) Supporting Posture 17



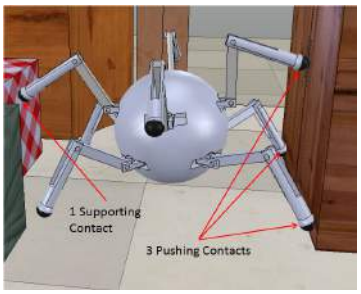
(i) Supporting Posture 18



(j) Supporting Posture 19



(k) Supporting Posture 20



(l) Supporting Posture 21

Figure 7-12: Posture generated for OctoBot with Posture Tree Generation module. Supporting Postures (10-21) are shown.

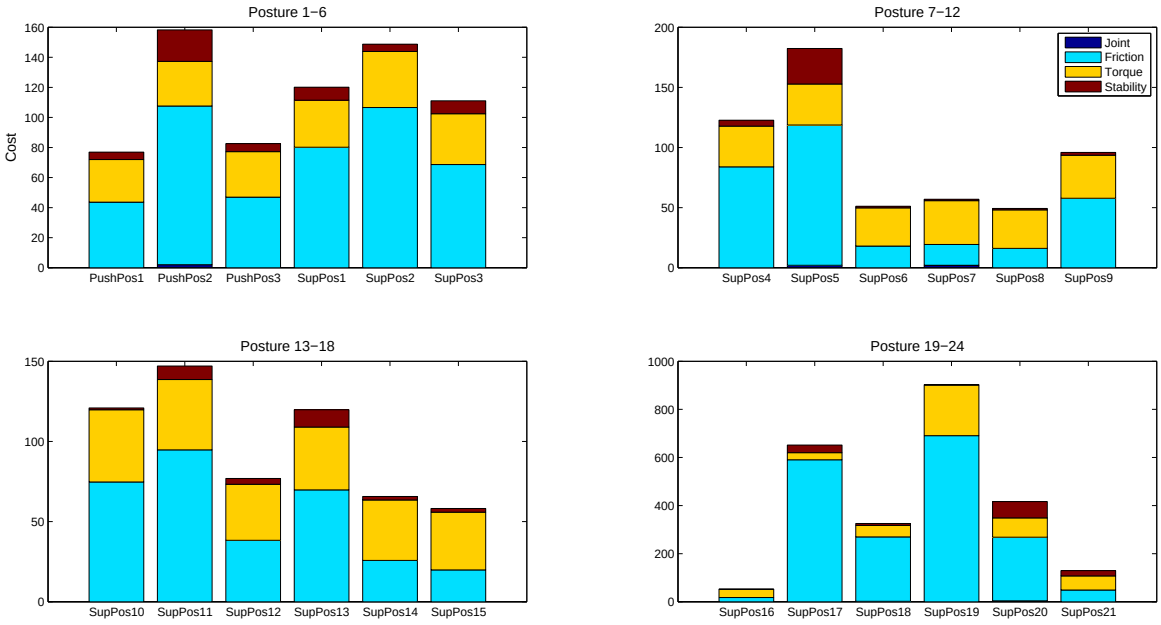


Figure 7-13: Criteria cost for all 24 postures computed at optimum locations after optimization.

configuration and hence easier to control. The remaining postures will not be simulated as they require more robust control for balancing dynamically and since these postures use supporting/pushing contacts for balancing, the controller should be capable of handling contact slipping, making the control more complicated.

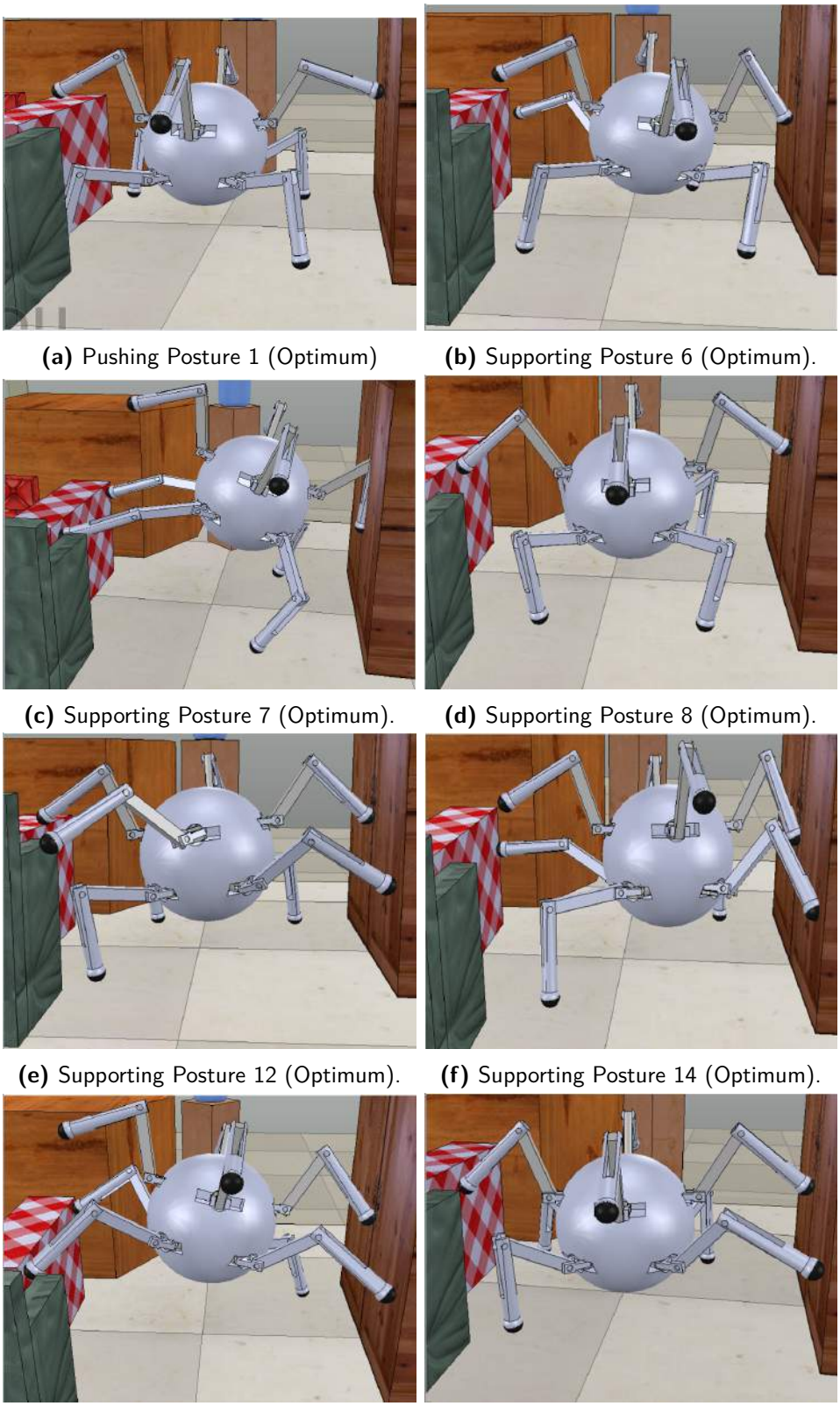
7-8 Simulation with OSC

To simulate the postures selected in the previous section, specific details about each posture has to be given as input to the controller in certain format. This was already discussed in chapter 6 under Posture Data Preparation section. So, two .txt files 'Optimum_Posture_PXX.txt' and 'Task_Targets_PXX.txt' are created. The former one includes details about the posture & the environment and the latter contains the optimal locations of each contact of the robot & its CoG.

The posture to be simulated is segregated into the following 3 types of tasks.

- CoG movement.
- Pushing contacts movement.
- Supporting contacts movement.

The simulation of postures can be divided into 3 stages and in each stage the data stored in the ***Task Struct*** are modified accordingly, as shown in the control architecture 7-7. The modifications done in each stage are explained below.



(a) Pushing Posture 1 (Optimum)
(b) Supporting Posture 6 (Optimum).
(c) Supporting Posture 7 (Optimum).
(d) Supporting Posture 8 (Optimum).
(e) Supporting Posture 12 (Optimum).
(f) Supporting Posture 14 (Optimum).
(g) Supporting Posture 15 (Optimum).
(h) Supporting Posture 16 (Optimum).

Figure 7-14: Optimal configuration of Pushing Postures 1 and Supporting Postures 6,7,8,12,14,15,& 16.

Stage1: Reach Pos

- CoG and fixed active contacts are moved simultaneously to their respective target positions (specified in 'Task_Targets_PXX.txt'). As specified earlier fixed active contacts will be moved with a proportional control.
- Pushing contact's target position is reduced by 0.02 m in the x direction and executed so that the contacts are just near the pushing plane.
- Supporting contact's target position is increased by 0.01 m in the x direction, and executed until the end-effector touches the support plane and generates some force. The task is stopped after making the supporting contact.

Stage2: Exert Force

- Depending upon the movement of ZMP in x direction, x value of CoG task is updated i.e, $\text{CoG}_x = ZMP_{ref} - ZMP_{comp}$. This is only done, if computed ZMP (ZMP_{comp}) is less than reference ZMP (ZMP_{ref}).
- The x position of pushing contacts is increased by 0.1 m and the task is executed to exert force on the object to be pushed.
- The x position of supporting contacts is increased by 0.05 m, so that the supporting contact remains in contact with the support plane, as the robot tries to push the object.

Stage3: Initial Pos

- If the object has been pushed to the predefined limit, the CoG task target position is set back to the initial position and executed.
- Both the pushing and supporting contacts are also set back to their respective initial position.

7-9 Simulation Results

The parameters shown in table 7-3 are set to certain fixed values inside the simulator and the simulation is carried out for all postures. The pushing movement of the object has been set to 0.05 m for all the simulations. The OSC parameters used for all the simulation is shown in table 7-4.

7-9-1 High Friction

The simulations carried out with the selected postures are done with high coefficient of friction ($\mu \approx 0.9$) at the contacts.

Table 7-3: Simulator Parameters and Corresponding Values

S.No	Parameters	Value
1	Dynamic Engine	ODE
2	Sampling time	5 ms
3	Contact Material	High friction (≈ 0.9)
4	Mass of the object	37.5 kg
5	Object to floor friction	Low friction (≈ 0.5)
6	Control Mode	Torque mode

Table 7-4: OSC Parameters and Corresponding Values

S.No	Parameters	Value
1	CoG task gains	k_p = -50000, k_v = -0.05k_p
2	Pushing contact task gains	k_p = -65000, k_v = -0.05k_p
3	Supporting contact task gains	k_p = -65000, k_v = -0.05k_p
4	Fixed active contact task gains	P = -5000
5	v_{max}	1.0 m/s

Pushing Posture 1

The tasks associated with Pushing Posture 1 are given below and the optimal positions for each task are tabulated in table 7-5.
Task 1: CoG.
Task 2: Pushing contact movement (End Effector 1).

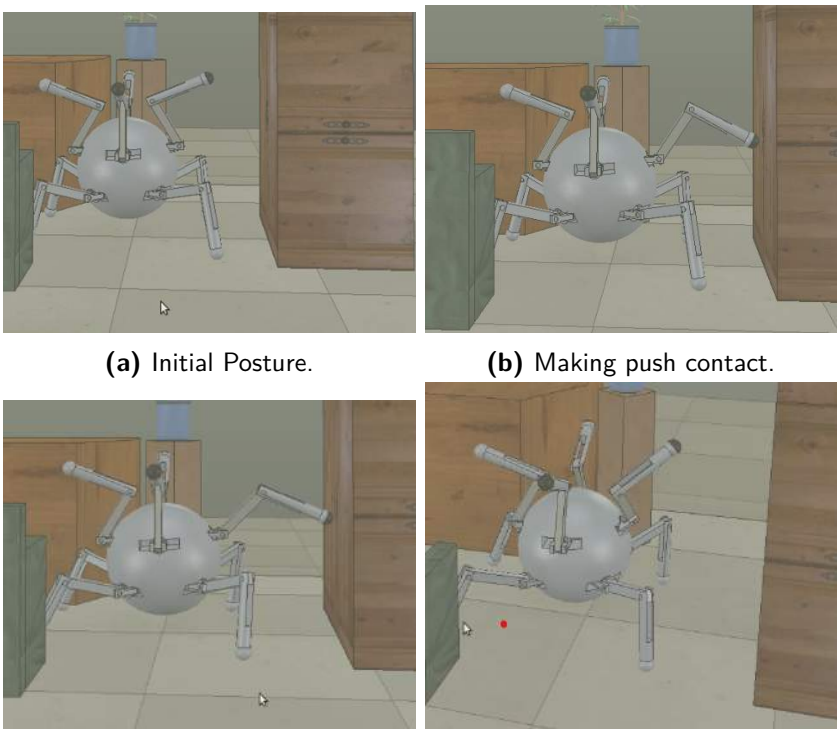
Results:

Simulation Pictures:

Figure 7-15 shows the snapshots taken while executing the pushing posture in the simulator. 7-15a is the initial posture assumed by the robot before making any contact, 7-15b shows the robot making pushing contact with end-effector 1, 7-15c shows the robot pushing the object and 7-15d shows the robot getting back to its initial position after successfully pushing the object.

Table 7-5: Optimal Position for Pushing Posture 1

S.No	Task	xyz values in m
1	Leg1 (Active)	[−0.31, 0.22, 0.00]
2	Leg2 (Active)	[−0.31, −0.22, 0.001]
3	Leg3 (Active)	[0.13, −0.22, −0.002]
4	Leg4 (Active)	[0.13, 0.22, −0.002]
5	End-effector 1 (Pushing)	[0.395, −0.007, 0.335]
6	CoG	[−0.09, −0.006, 0.23]



(a) Initial Posture. **(b)** Making push contact.

(c) Pushing the object. **(d)** Posture after pushing the object.

Figure 7-15: Snapshots showing the execution of Pushing Posture 1 in the simulator.

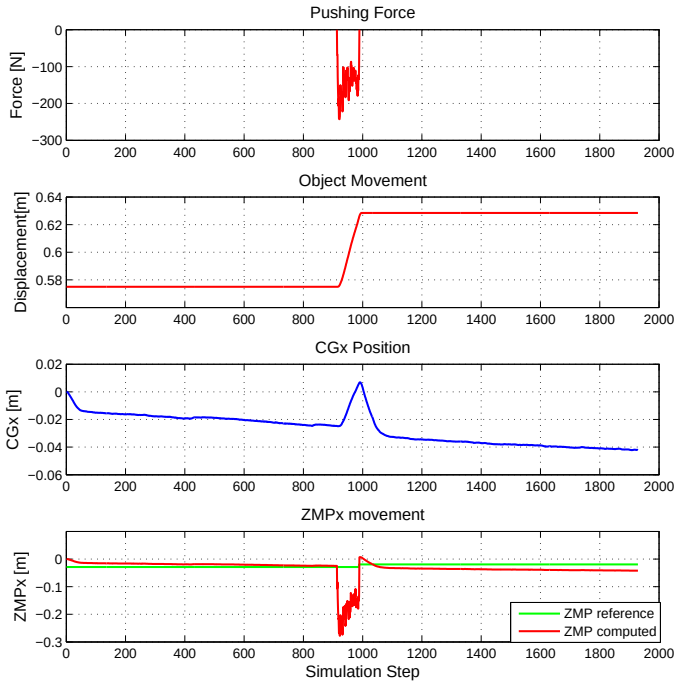
Pushing Data Plot:

Figure 7-16 plots the data obtained while simulating the Pushing Posture 1 in the simulator. 7-16a plots the generated pushing force, displacement of the object, CoG movement in x direction and computed ZMP movement in x direction along with its reference.

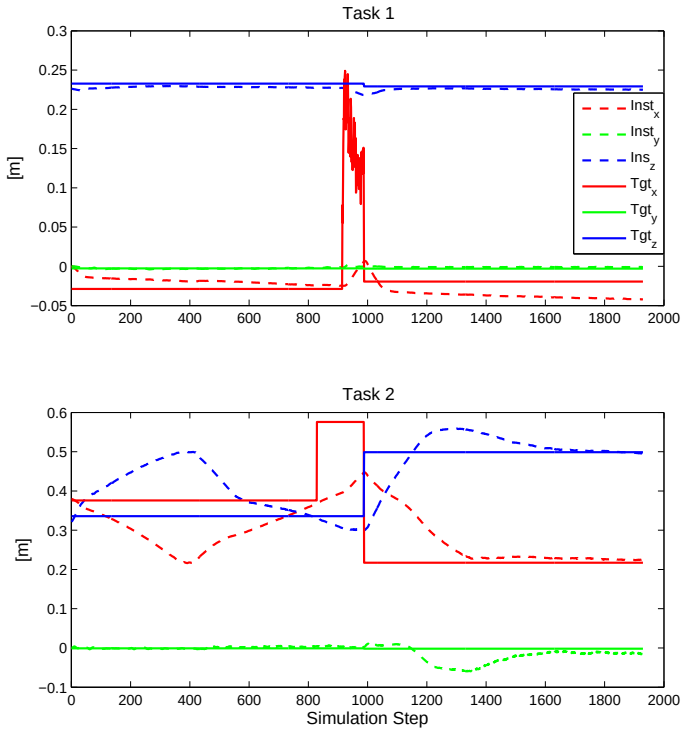
Observations:

- The pushing force plot in figure 7-16a shows the maximum pushing force generated at the pushing contact to be $\approx$ 240 N and it starts decreasing gradually once the object starts moving and finally reduces to 0 when the object has been pushed to 0.05 m.
- Object movement plot in figure 7-16a shows the smooth movement of the object from 0.575 m to 0.625 m.
- In the ZMP_x movement plot it can be seen that the computed ZMP, ZMP_{comp} (red line) gets negative value and it falls below the reference ZMP, ZMP_{ref} (green line). This is due to the reaction force acting on the end-effector when it is trying to push the object, and it is trying to destabilize the robot. The controller as described earlier, reacts by computing the ZMP difference ($ZMP_{ref} - ZMP_{comp}$) and setting this difference as target to the robot's CoG. In the CGx [3] position plot, it can be seen that around the simulation step 900, the CoG x position increases and as it increases the ZMP becomes

[3] CoG movement in x direction



(a) Pushing Posture 1 data plot



(b) Instantaneous and targets positions of the tasks.

Figure 7-16: Simulation data plot for Pushing Posture 1.

Table 7-6: Optimal Position for Supporting Posture 6

S.No	Task	xyz values in m
1	Leg1 (Supporting)	[−0.34, 0.15, 0.26]
2	Leg2 (Active)	[−0.13, −0.23, −0.007]
3	Leg3 (Active)	[0.27, −0.22, 0.001]
4	Leg4 (Active)	[0.13, 0.22, 0.001]
5	End-effector 1 (Pushing)	[0.399, −0.008, 0.27]
6	CoG	[0.046, −0.035, 0.263]

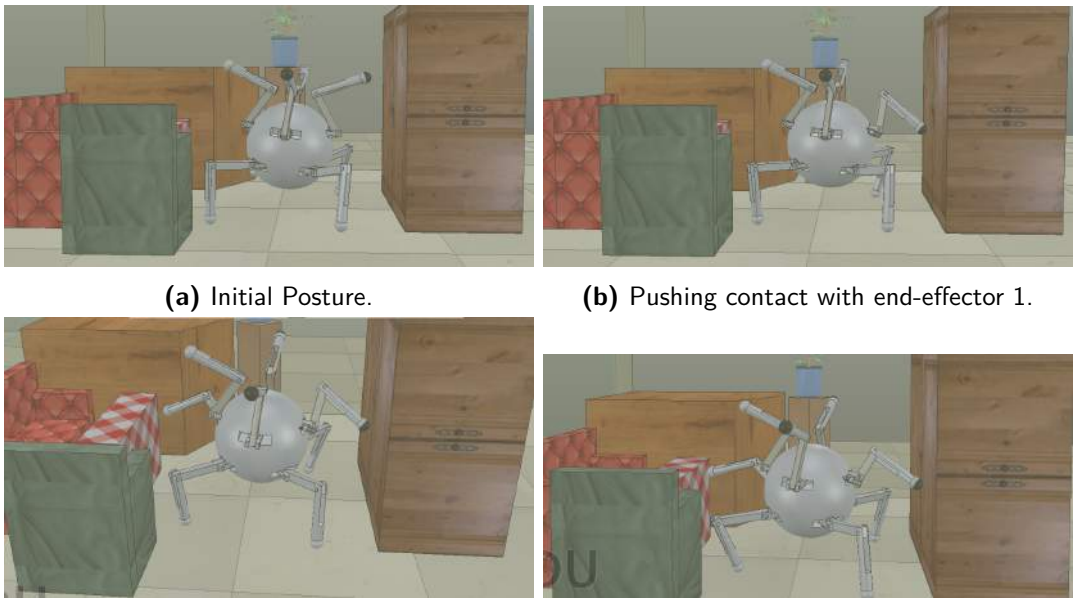
less negative (i.e. starts reducing). In this way, the CoG movement compensates the ZMP movement in -x direction and thus stabilizes the robot to apply the desired pushing force.

- In figure 7-16b, Task 1 plot refers to the CoG position of the robot. It can be visualized that y and z positions of the robot closely follow the target positions. The peaks observed in the target's x position between the simulation steps 900-1000, represents the ZMP compensation movement as explained earlier.

- Task 2 in figure 7-16b represents the pushing contact movement (end-effector 1). xyz positions of the end-effectors, starts following their respective target positions around the simulation step 400 and continues to do so until the end. The step observed in the end-effector's x target position just over 800 simulation steps, is for the end-effector to exert force on the object. There is a huge difference between the instantaneous position of the end-effector and its respective target position until step 400. This is because, each new task is added only if the previous task has reached its target position or it had run for at least 1000 steps (whichever is earlier). In this case, Task 1 (CoG movement) reaches its goal around 400 steps, as a result Task 2 gets added after 400 steps. It can be seen that x and z positions of Task 2 start following their respective target positions after 400 simulation steps. The downward step observed in z target position and upward step observed in x target position immediately after the pushing motion (after 1000 simulation steps) is for the end-effector to reach its initial position.

Supporting Posture 6

The tasks associated with Supporting Posture 6 are given below. Here the active contacts are Leg2, Leg3 and Leg4, pushing is done with end-effector 1 and leg1 is used for making support contact. The optimal positions for the tasks are tabulated in table 7-6.
Task 1: CoG movement.
Task 2: Pushing contact movement (End-Effector 1).
Task 3: Supporting contact movement (Leg 1).



(a) Initial Posture. **(b)** Pushing contact with end-effector 1.

(c) Supporting contact with leg 1. **(d)** OctoBot pushing the object.

Figure 7-17: Snapshots showing the execution of Supporting Posture 1 in the simulator.

Results:

Simulation Pictures:

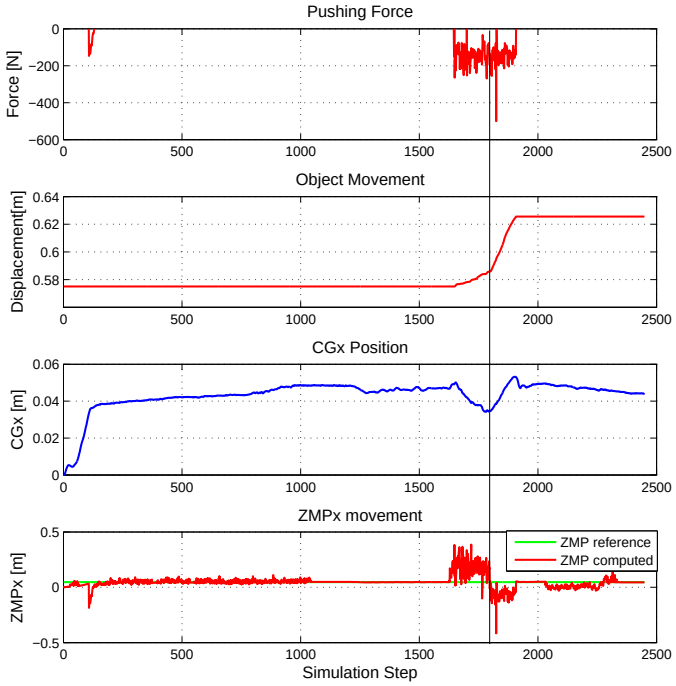
Figure 7-17 shows the execution of supporting posture 6 in the simulator. 7-17a is the initial posture of the robot before making any contact, 7-17b shows the pushing contact made with end-effector 1, 7-17c shows the supporting contact made with leg 1 and 7-17d displays how the object is being pushed by the robot.

Pushing Data Plot:

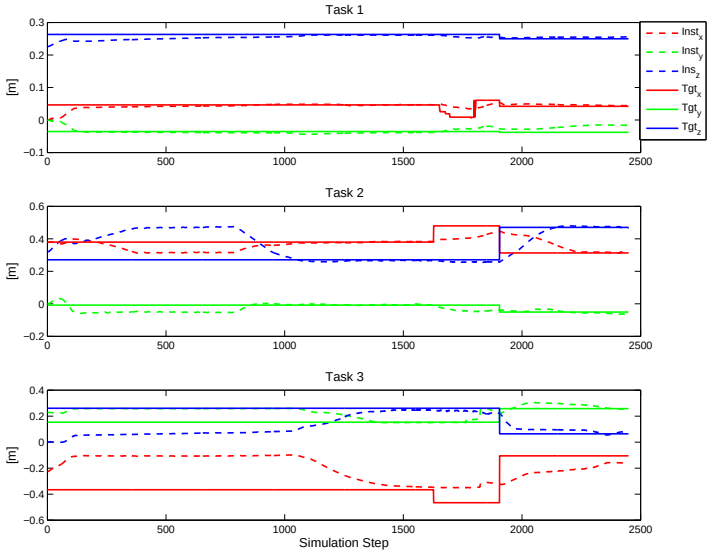
Figure 7-18 plots the data obtained while simulating the Pushing Posture 1 in the simulator. 7-18a plots the generated pushing force, displacement of the object, CoG movement in x direction and computed ZMP movement in x direction along with its reference.
Figure 7-18b plots the instantaneous and target positions of each task. $Inst_x$, $Inst_y$ and $Inst_z$ represents the instantaneous xyz positions of each task and Tgt_x, Tgt_y and Tgt_z represents their corresponding xyz target positions.

Observations:

- The pushing force plot in figure 7-18a is similar to the one observed for Pushing posture 1. There is a momentary peak force ($>$ 400 N) around 1750th simulation step which could be due to the sudden increase in CoG acceleration. The maximum force is $\approx$ 240 N as observed earlier.
- The object movement plot shows that it is not very smooth as observed in the case of Pushing posture 1 and it also takes more time to push the object from 0.575 m to 0.625 m.



(a) Supporting Posture 6 data plot



(b) Instantaneous and targets positions of the tasks for Supporting Posture 6.

Figure 7-18: Simulation data plot for Supporting Posture 6.

Table 7-7: Optimal Position for Supporting Posture 8

S.No	Task	xyz values in m
1	Leg1 (Active)	[−0.12, 0.22, −0.002]
2	Leg2 (Active)	[−0.12, −0.23, −0.001]
3	Leg3 (Active)	[0.26, −0.22, 0.001]
4	Leg4 (Active)	[0.25, 0.22, 0.001]
5	End-effector 1 (Pushing)	[0.399, −0.007, 0.267]
6	End-effector 3 (Supporting)	[−0.35, 0.04, 0.269]
7	CoG	[0.042, 0.009, 0.214]

- The black vertical line represents the iteration at which ZMPx[4] movement takes negative value and in the CoG plot it can be seen that it is counteracted by accelerating the CoG in the $+x$ direction, i.e ZMP compensation movement. The CoG movement is relatively lesser when compared to the Pushing Posture 1, this is due to the supporting contact which counteracts the destabilizing moment and hence reduces the ZMP difference.

- After pushing the object (around 1900th simulation step), the end-effector stops exerting force and the ZMP gets back to its initial reference position and follows it closely until the end.

- In figure 7-18b, the convergence of instantaneous position of each task to its corresponding target positions is shown. It can be seen that first Task 1 starts converging to its target position, followed by Task 2 at 750th simulation step and finally Task 3 starts converging around 1200th simulation step.

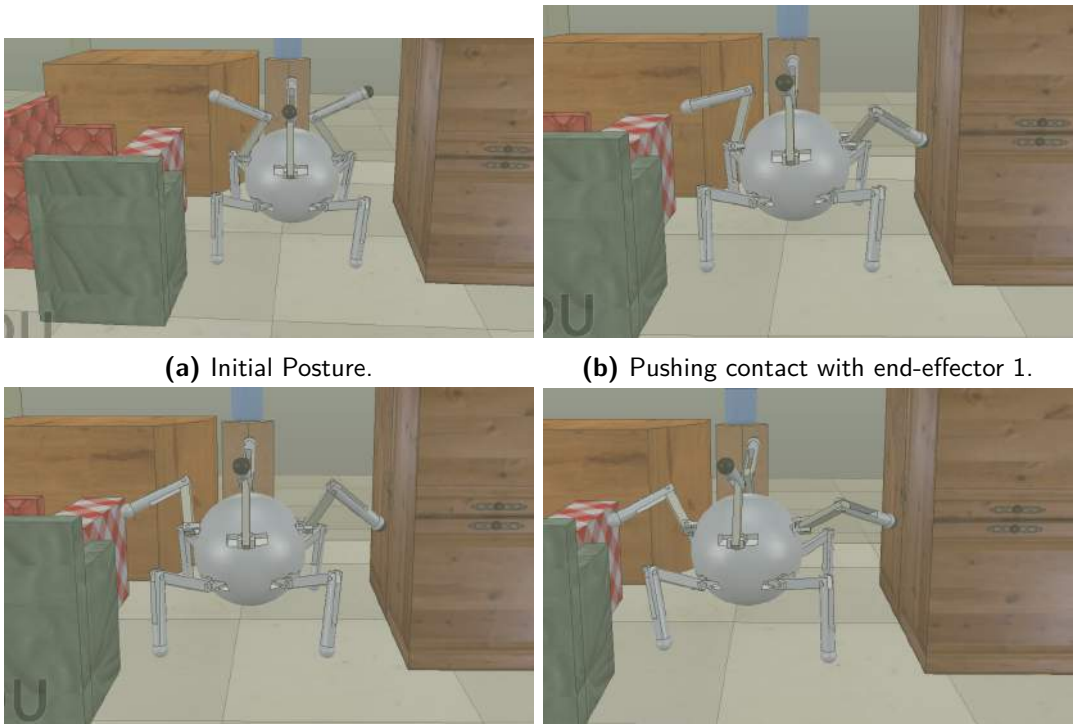
- The step observed in CoG refers to the ZMP compensation movement as explained earlier. The step in Task 2 around 1600th simulation step is for the end-effector to generate pushing force and downward step observed in the Task 3 plot around the same iteration is for leg 1 to remain in contact with the supporting object, so that pushing can be supported effectively.

- After pushing, it can be seen that the target positions are set back to their respective initial values and each task closely follows it and converges as well.

Supporting Posture 8

The tasks associated with Supporting Posture 8 are given below. Here the active contacts are Leg1, Leg2, Leg3 and Leg4, pushing is done with end-effector 1 and end-effector 3 is used for making support contact. The optimal positions for the tasks are tabulated in table 7-7.
Task 1: CoG.
Task 2: Pushing contact movement (End-effector 1).
Task 3: Supporting contact movement (End-effector 3).

[4]ZMP movement in x direction.



(a) Initial Posture. **(b)** Pushing contact with end-effector 1.

(c) Supporting contact with end-effector 3. **(d)** OctoBot pushing the object.

Figure 7-19: Snapshots showing the execution of Supporting Posture 8 in the simulator.

Results:

Simulation Pictures:

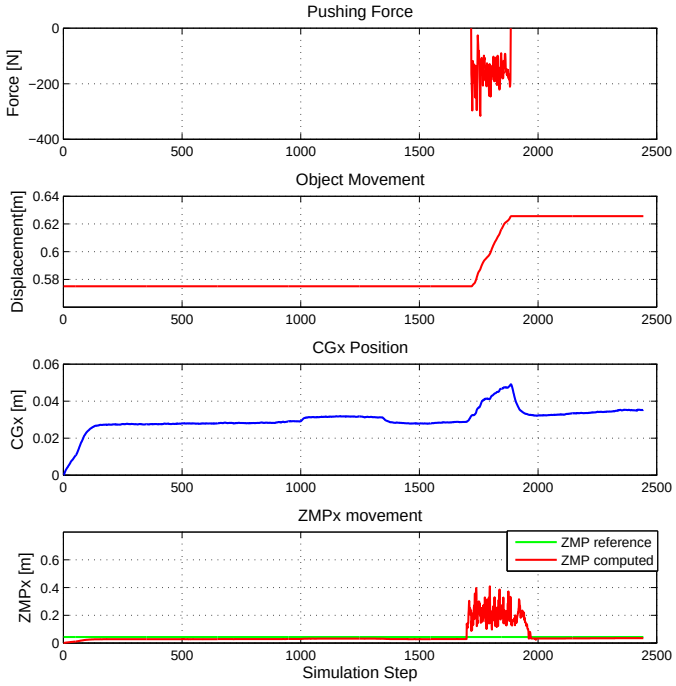
Figure 7-19 shows the execution of supporting posture 8 in the simulator. 7-19a is the initial posture of the robot before making any contact, 7-19b shows the pushing contact made with end-effector 1, 7-19c shows the supporting contact made with end-effector 3 and 7-19d displays how the object is being pushed by the robot.

Pushing Data Plot:

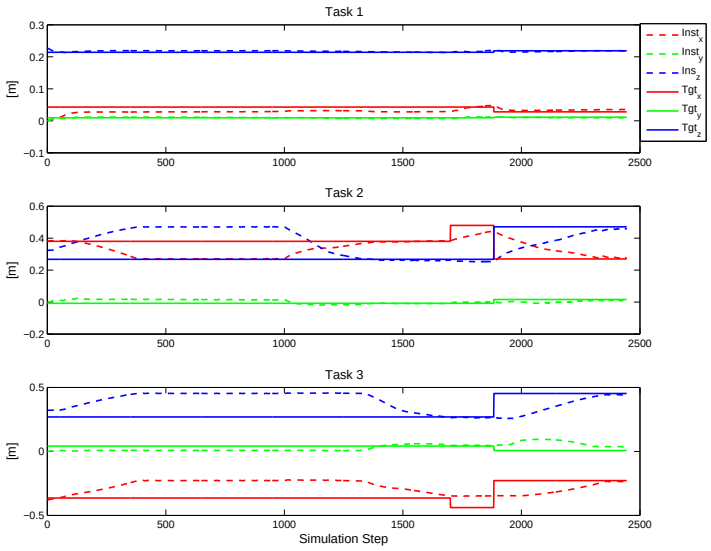
Figure 7-20 shows the data obtained while simulating the Supporting Posture 8 in the simulator. 7-20a plots the generated pushing force, displacement of the object, CoG movement in x direction and computed ZMP movement in x direction along with its reference.
Figure 7-20b plots the instantaneous and target positions of each task. $Inst_x$, $Inst_y$ and $Inst_z$ represents the instantaneous xyz positions of each task and Tgt_x, Tgt_y and Tgt_z represents their corresponding xyz target positions.

Observations:

- Pushing force and object movement plots in figure 7-20a, is similar to the other postures.
- Compared to the previous postures, a major difference is observed in the ZMPx movement plot for this posture. The ZMP hardly moves in -x direction, as a result very less



(a) Supporting Posture 8 data plot



(b) Instantaneous and targets positions of the tasks for Supporting Posture 6.

Figure 7-20: Simulation data plot for Supporting Posture 8.

CoG movement in $+x$ direction is observed while pushing the object. This suggests that the supporting contact (end-effector 3) fully neutralizes the destabilizing moment generated due to the reaction force at the end-effector 1. This makes the robot more stable. ZMP moves in the $+x$ direction considerably because the reaction force generated at the supporting contact is more than that generated at the pushing contact. However, this doesn't destabilize the robot, as it can be seen that after pushing the computed ZMP follows the reference ZMP perfectly.

- Figure 7-20b behaves in the same way as observed for the previous postures.

Supporting Posture 11 and 15

Similar to the other postures, Supporting posture 11 and 15 are also simulated and figures 7-21 and 7-22 shows the snapshots taken during the simulation, depicting the various phases the robot undergoes in reaching the target posture and pushing the object.

7-9-2 Reduced Friction

Simulation results discussed in the previous section were carried out with high friction contact material (≈ 0.9). The friction coefficient of the contacts were reduced to ≈ 0.7 and the simulation was repeated for pushing posture 1 and supporting posture 7. This is done to compare how the postures performance differ from the previous case of high friction and also to compare the relative difference in performance between the pushing and supporting posture as μ is reduced.

Results: Pushing Posture 1

Simulation Pictures:

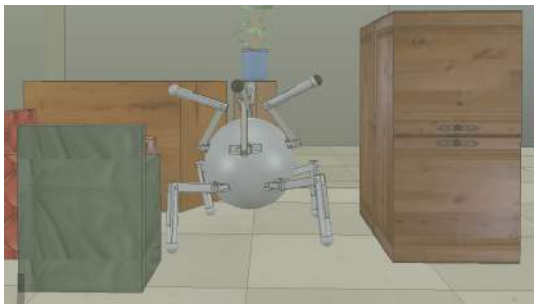
The tasks and its corresponding targets are maintained the same except for the friction at the active contacts. Figure 7-23 shows the snapshots taken during the simulation.

Pushing Data Plot:

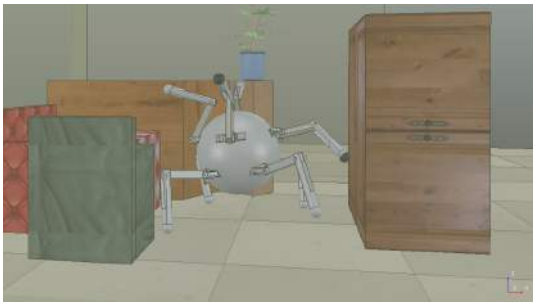
As done in the previous section, the pushing data obtained during the simulation are plotted in figure 7-24.

Observations:

- The force plot in figure 7-24 shows the magnitude plummeting to 0 several times and rising immediately during the pushing motion, suggesting that it is relatively steady than the high friction case. However, the overall force profile looks similar to the high friction case.
- The object movement is relatively less smoother and it takes slightly more time in the beginning to get the object to move.



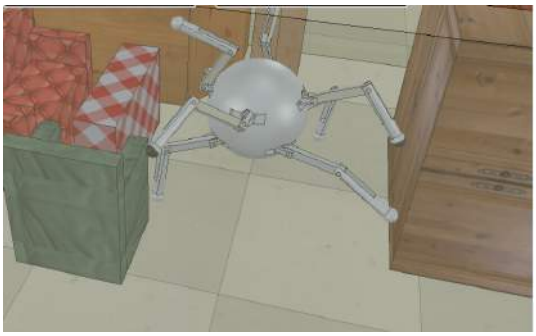
(a) Initial Posture.



(b) Pushing contact with end-effector 1.

(c) Pushing contact with Leg 3.

(d) Supporting contact with end-effector 2.

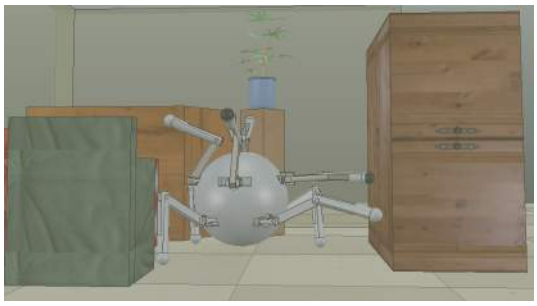


(e) OctoBot pushing the object.

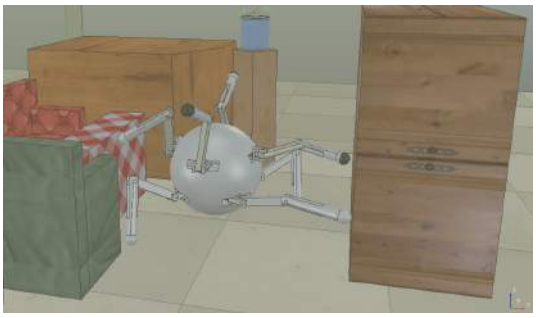
Figure 7-21: Snapshots showing the execution of Supporting Posture 11 in the simulator.

(a) Initial Posture.

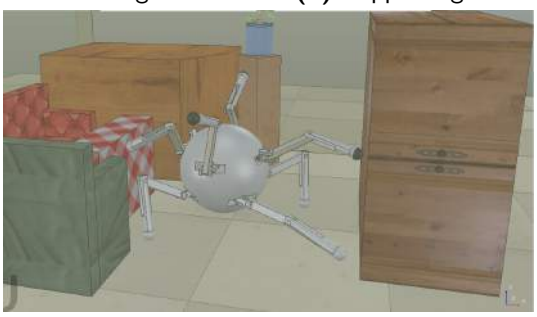
(b) Pushing contact with end-effector 1.



(c) Pushing contact with Leg 3.

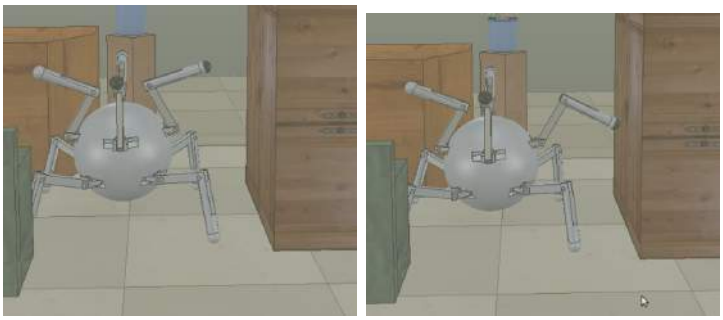


(d) Supporting contact with end-effector 3.



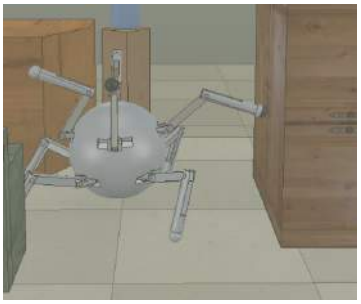
(e) OctoBot pushing the object.

Figure 7-22: Snapshots showing the execution of Supporting Posture 15 in the simulator.



(a) Initial Posture.

(b) Making push contact.



(c) Pushing the object.

(d) Posture after pushing the object.

Figure 7-23: Snapshots showing the execution of Pushing Posture 1 in the simulator with μ $\approx$0.7.

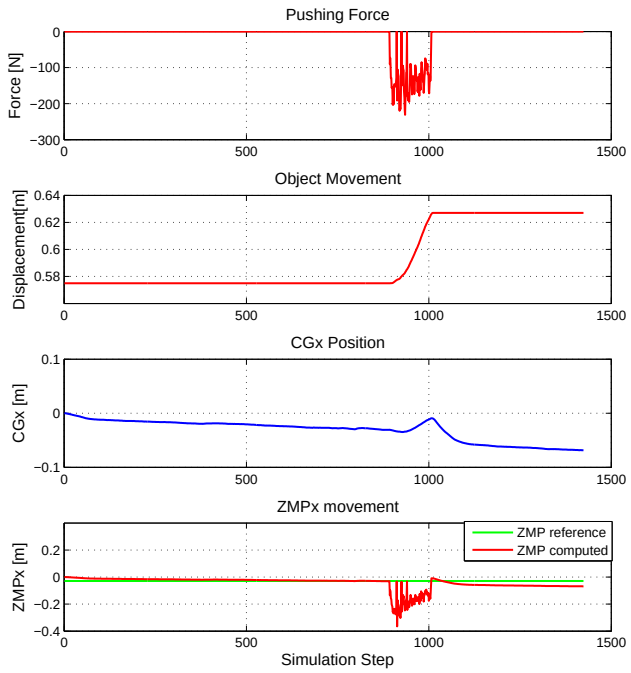
- The ZMP movement plot shows relatively high value ($\approx$ -0.4 m) in the -x direction when compared to the high friction case ($\approx$ -0.275 m). The reason is evident in the CoG movement plot. The controller restricts/resists the movement of ZMP in -x direction by moving the CoG appropriately. Since the friction at the active contacts is reduced, the controller is not able to move the CoG as much it intends to. The CoG movement (i.e. ZMP compensation) is considerably less when compared to the high friction case.

- Task 1 plot refers to the CoG movement of the robot and both Tgt_x and $Inst_x$ reflects the same scenario discussed above.

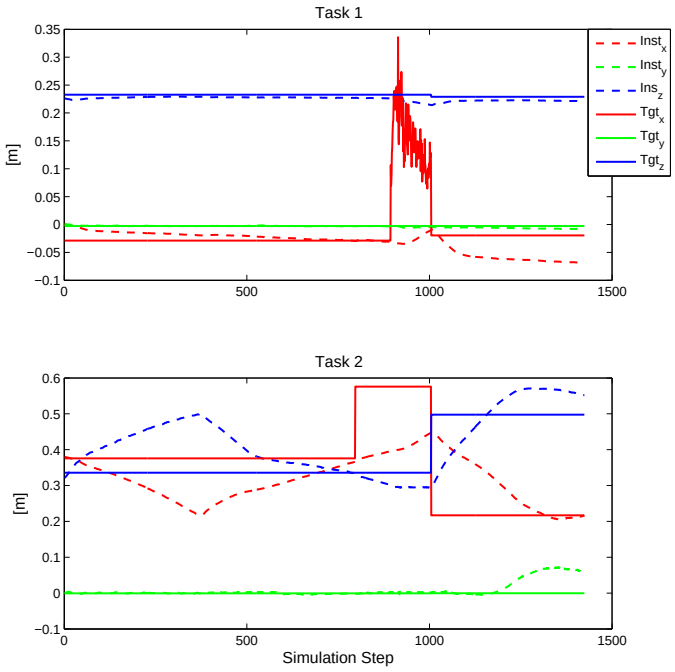
- There is not much change observed in the Task 2 plot which refers to the pushing contact movement.

Results: Supporting Posture 8

Figure 7-25 shows the pushing data plot obtained by simulating the supporting posture 8, with active contact friction (μ) maintained at $\approx$ 0.7. The figure shows only the plots of end-effector force, object movement, CoG movement in x direction and ZMP movement in x direction. Since there weren't any significant differences observed in the task plot, it is not shown below.



(a) Pushing Posture 1 data plot



(b) Instantaneous and targets positions of the tasks for Pushing Posture 1.

Figure 7-24: Simulation data plot for Pushing Posture 1 pushing an object of mass 37.5 kg with contact friction $\mu \approx 0.7$.

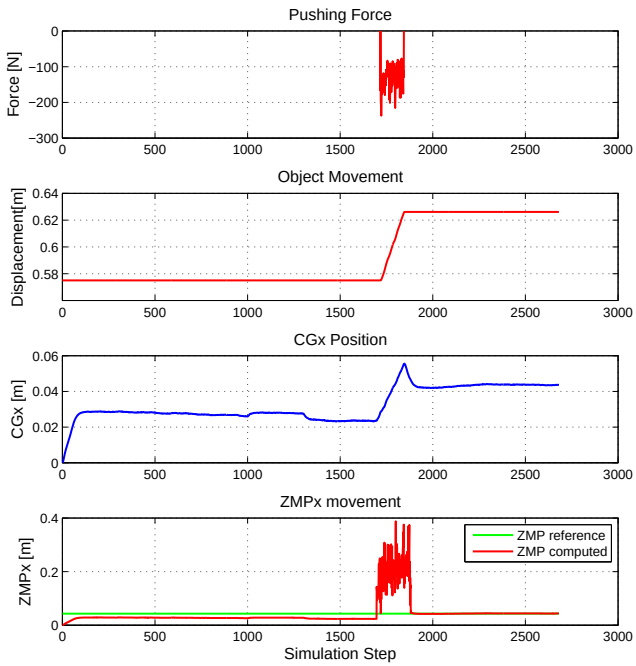


Figure 7-25: Simulation data plot for Supporting Posture 8 pushing an object of mass 37.5 kg with contact friction $\mu \approx 0.7$.

Pushing Data Plot:

Observations:

- Not much difference is observed in any of the plots, except for the CoG movement (x direction) plot. Due to less friction at the contacts, the robot is not able to get back to its initial CoG state after the pushing motion. The difference in the CoG values between the simulation step 1500 and 2000 suggest this.

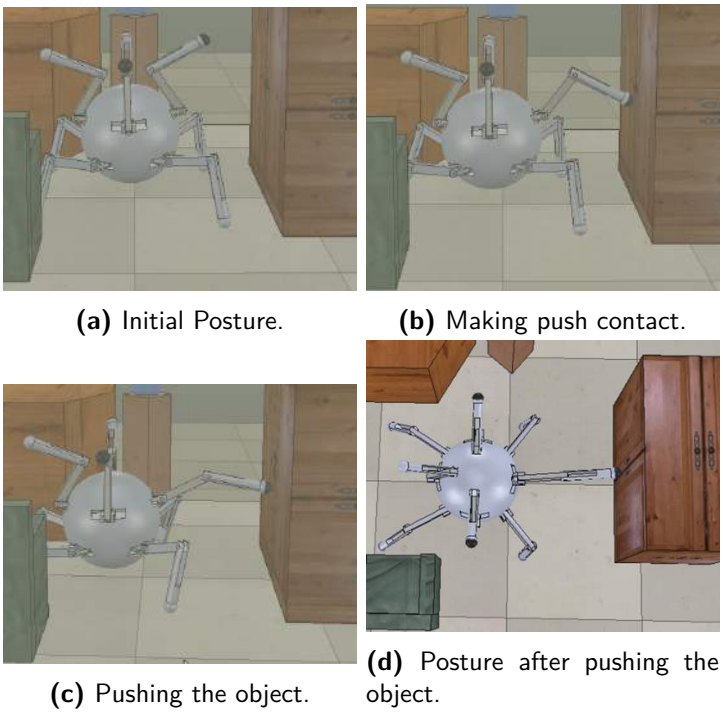
7-9-3 Low Friction

The simulations are repeated for pushing posture 1 and supporting posture 8 with a contact friction $\mu \approx 0.5$.The results so obtained are discussed below in the following sections.

Results: Pushing Posture 1

Simulation Pictures:

Figure 7-26 shows the simulation snapshots. The pictures clearly shows the slipping of the robot as it tries to push the object.



(a) Initial Posture.

(b) Making push contact.

(c) Pushing the object.

(d) Posture after pushing the object.

Figure 7-26: Snapshots showing the execution of Pushing Posture 1 in the simulator with μ $\approx$0.5.

Pushing Data Plot:

As done in the previous section, the pushing data obtained during the simulation are plotted in figure 7-27.

Observations:

- The force plot and the object movement plot shows the inability of the robot to generate high force (>200 N) to initiate the object motion and maintain substantial amount of force to continue pushing the object.
- The negative slope observed in the x movement of CoG between the simulation steps 1100 and 1300, represents the slipping motion of the robot due to the reaction force acting on it as a result of pushing.
- The ZMP plot shows that it attains high negative value once the pushing starts and continues to fluctuate around that value throughout the pushing motion, due to the inability of the controller to compensate its movement, because of low friction. The ZMP value reduces after the pushing motion, but it continues stay well below the reference.

Results: Supporting Posture 8

With the same simulator settings, supporting posture 8 is simulated and the results obtained are shown below in figure 7-28.

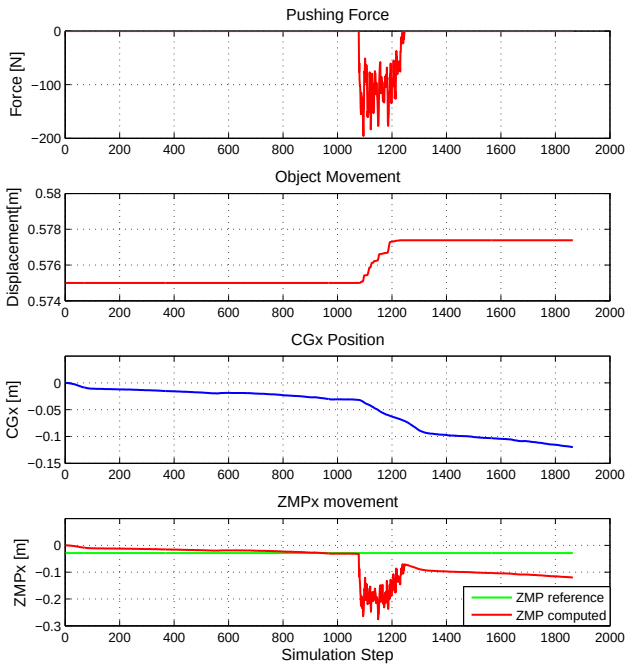
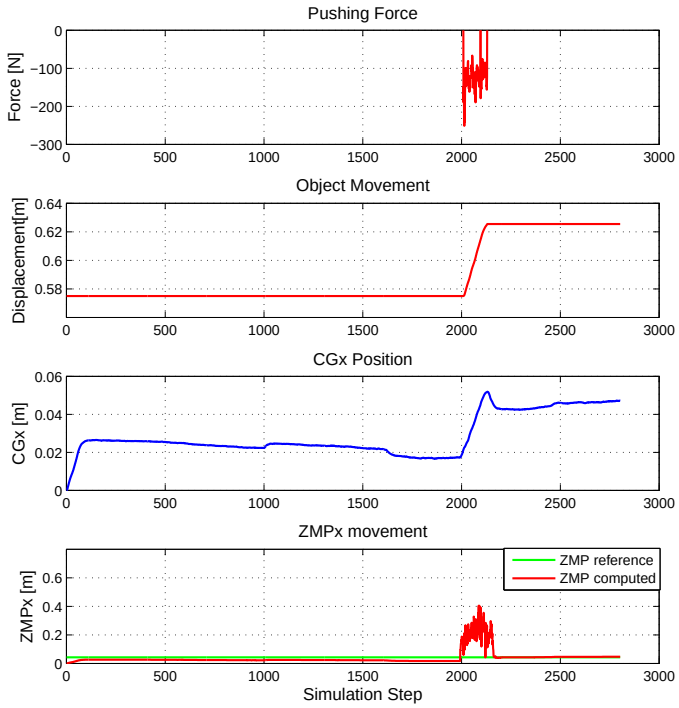


Figure 7-27: Simulation data plot for Pushing Posture 1 pushing an object of mass 37.5 kg with contact friction $\mu \approx 0.5$.

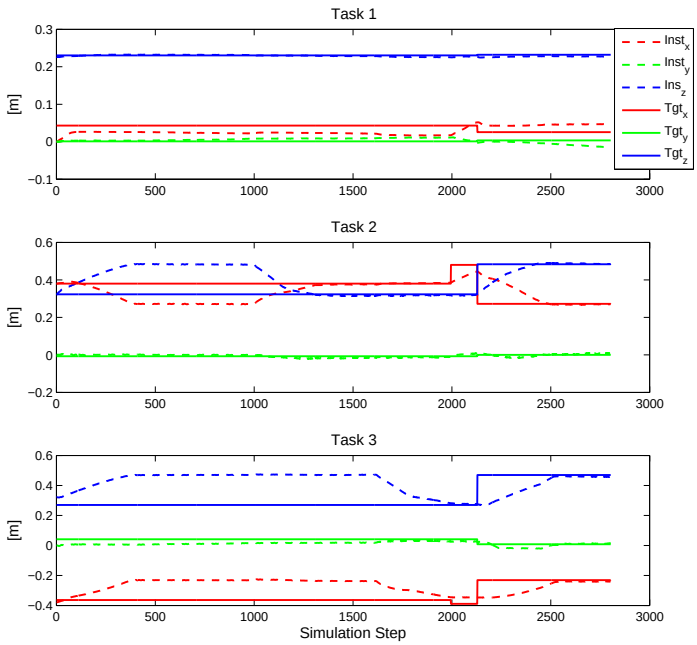
Pushing Data Plot:

Observations:

- Due to the supporting contact, the robot manages to successfully push the object even in the less friction case.
- There is not much difference observed except for the fact that the pushing motion is delayed by 500 steps. In the case of high friction and reduced friction case, the pushing motion starts after 1750 simulation steps, while in this case it starts after 2000 simulation steps. This is due to slipping which makes the robot difficult to achieve its Task 1 (i.e. CoG movement.
- The above discussed phenomena is clearly visible in the task plots shown in figure 7-28b. Task 1 (CoG movement) never reaches its set target in the x direction as a result it runs for 1000 simulation steps and then adds Task 2. In the task 2 (pushing contact) plot it can be seen that the task's xyz values starts following its respective target positions after 1000th step and it runs until 1750 steps, which is then followed by Task 3 (supporting contact).
- As observed in the reduced friction case, the controller tries to get back the robot's CoG to its initial state, but it is not able to do due, to very less friction. The difference between Tgt_x and $Inst_x$ of Task 1 at simulation step 2500, clearly suggest this.



(a) Supporting Posture 8 data plot



(b) Instantaneous and targets positions of the tasks for Supporting Posture 8.

Figure 7-28: Simulation data plot for Supporting Posture 8 pushing an object of mass 37.5 kg with contact friction $\mu \approx 0.5$.

Overall, the performance of the pushing posture gradually deteriorated as the contact friction was reduced and in the case of supporting posture 8, there was very less deterioration observed. To conclude, the simulation results closely corroborates the high cost evaluated for pushing posture 1 and the low cost evaluated for supporting posture 8 in the posture optimization module.

Chapter 8

Conclusion and Recommendations

8-1 Conclusion

In this thesis, a general controller applicable for any multi-limb robot, for pushing task under any given environment has been developed. Unlike the present controllers as discussed in chapter 1, the controller developed here finds the optimal (could be a local optimal) configuration for the pushing task by means of finding the optimal location of each contact and its CoG. The controller also exhibits ZMP compensation movement, which is inspired from the human way of pushing. Apart from optimizing the posture, for heavy object manipulation, the robot tries to use surrounding objects as support for exerting high pushing force. The complete concept involves three major modules Posture Tree Generation, Posture Optimization and Operational Space Controller (OSC) module. First two modules are subjected to preliminary evaluations with a Humanoid robot and a simple environment, and the complete concept is validated with a multi-limb robot 'OctoBot' and a typical living environment by means of simulation in V-Rep. The major findings/contributions of the work are summarized below.

Posture Tree Generation

- The environment in which the robot will be operating is constructed using a $3d$ software package (Solidworks used here) and saved in'.stl' format. Using an stl reader the environment is reconstructed, and with predefined robot contacts, the possible positions which the robot can access are determined using forward kinematics. For practical application, the environment can be reconstructed using $3d$ vision and details such as norms of the planes, centroids, vertices, etc., can be extracted as done in the stl reader. As such, the developed concept can still be applied directly in real time applications for generating postures.
- For preliminary evaluation of the modules, simple objects (flat shaped) were considered for pushing, and slightly complex objects (curved) were considered for supporting.

Including more complex shapes (irregular shaped objects, objects with holes, etc.) for both pushing and supporting is possible, but this will proportionately increase the number of planes which will in turn increase the consumption of time. However, this can be controlled with suitable heuristics. The typical living environment considered for final evaluation involves only simple objects in order to keep it simple and also to ease the process of creating dynamic environment in the simulator.

- The concept adopted here for generating postures, works well in finding the planes on objects suitable for making contact, prunes the available contact based on its reachability and also performs well in filtering out the objects not suitable for making support contact. All these were evident from the results obtained during preliminary and final evaluations.

Posture Optimization

- Four cost criteria joint, torque, friction and stability are adopted for optimizing the posture, with cosine based cost profile for joint cost and exponential based cost profile for the remaining criteria. The rate of costing (exponential rate y in e^{yx}) is kept same for all criteria which are based on exponential. As observed in the preliminary and final evaluation results of the optimization module, with the current cost rate setting, the robot can adapt its posture to any given environment for the given task. However, to give more preference to a particular criteria, either the rate of costing (y) can be increased accordingly, or a weighting term (w) can be multiplied with each cost term before summing it up to find the total cost.
- An approximate pushing force which the robot needs to generate is given as input for the optimizer to generate optimum postures. This is done to simplify the posture optimization process. In practical scenarios this can be replaced with a separate controller, which determines the approximate pushing force by means of $3d$ vision and active tactile sensing (by extracting object details such as its dimension, approximate mass, type of object-floor contact, etc).
- The posture/configuration of the robot as a whole involves a number of variables and it is optimized indirectly by means of optimizing the contacts and the CoG locations. Since optimizing too many variables will exponentially increase the time consumed, the variables are selected based on its relative impact it has on the pushing task. As a result, for pushing and supporting contacts z location is optimized, for active contacts x location is optimized and the robot's CoG is optimized in x and z directions. The results were convincing enough, looking at the simulations carried out with OctoBot for various postures. However, for Supporting Posture 11, the supporting contact (end-effector 2) broke instantaneously, once the object started moving, which destabilized the posture. This was due to the lower reachability of the end-effector, because of the extreme y location of the contact and it was corrected manually by lowering the value of y position. A similar situation was experienced with Supporting Posture 15, where one of the pushing contact (leg 3) broke immediately after the object movement. This was also due to the reachability limit of the contact. The posture was, however successful as it had another pushing contact. All these issues are kept in check by limiting the movement of CoG in x direction and also by limiting the pushing movement.

- To verify, if the optimizer explores the posture according to the amount of pushing force the robot needs to generate, the desired pushing force was doubled and the optimization was repeated during the preliminary evaluation of the Optimization module. The resulting postures were found to be different, with increased support polygons, lower CoG and pushing contact location to minimize the destabilizing moment and the final cost of the postures also reflected significant difference. During this process, it was observed that for some postures in figure 5-9, the orientation of the main body (i.e. torso) seemed to erratic/uncontrolled in nature. It is understood that for robots such as humanoids which has certain DOFs for its main body, it would be better to include its orientation as well in the variables to be optimized, in order to realize more controlled postures.
- In order to ward-off the local minimum convergence while optimizing the posture, the optimization method was run 3 times, with each consecutive run starting from the optimal values of the previous run. This was experimented during the preliminary evaluation of the module, but since the improvements in the cost were not significant, single optimization run has been adopted for the final evaluations. The optimal postures obtained as a result were successfully validated by means of simulations and the postures were found to be doing well during the pushing task. However, it is not certain that the postures obtained during preliminary and final evaluation through optimization are globally optimum, it is most likely to be locally optimum ones.
- The time consumed for optimizing the variables increases significantly with the number of variables being optimized and it also depends on the limit range of each variable. The maximum time consumed during the preliminary evaluation was observed to be ≈ 20 mins/posture and in the final evaluation it took approximately 35 mins/posture. The time reported here are for one complete optimization run. However, this time can be significantly reduced with multi-core machines, which gives the possibility of optimizing several postures simultaneously.

Operational Space Controller (OSC)

- OSC developed for executing postures is based on the principle proposed by Luis Sentis [28] for whole body motion. The selected posture is segregated into separate tasks, and each lower level task is executed within the null space of the higher level tasks. In order to simplify the controller and also to make it compatible with the simulator, both CoG and fixed active contacts moving tasks are executed simultaneously making it a single task. While the movement of CoG is controlled by high OSC gains, the fixed active contacts are controlled by low proportional gain, making the former primary and the latter secondary. Hence the CoG always reaches its target position, but the fixed active contacts reach their target positions only when it is not in conflict with the primary task (i.e CoG). This works well with OctoBot (4 legged robot), but with 2 legged robots such as Humanoid, both CoG and fixed active contacts should be operated as separate tasks. This is necessary because, humanoids being inherently unstable, the active contacts have to be moved to their respective target location in order to increase its stability.
- The pushing force on the object is exerted by increasing the pushing contact's target location in the x direction by 0.1 m. This makes OSC to generate a linear force at

the pushing contact in the x direction. Upon pushing the object to a predefined limit, the pushing is stopped by setting the pushing contact's target location to initial value. This method of generating force was successfully verified with all the selected postures through simulation.

- The reaction force acting at the pushing contact and the destabilizing moment acting on the robot because of it are resisted by ZMP compensation movement of CoG. This principle exactly mimics the human way of pushing an object. For the computation of ZMP, the linear and angular momentum of the robot is assumed to be zero, since the acceleration of CoG is less while pushing the object from a static position. So in principle the ZMP computed here is quasi-static. This works well for all the postures which have been evaluated in the simulator and has continuously yielded desired results. However, in the case of moving the object (i.e. after pushing) it would be more appropriate to compute the ZMP considering the momentum as well, i.e. including the dynamics.

- The controller developed here is applicable only for simple postures, i.e postures which can remain statically stable without any object support. Most of the postures which have at least 3 fixed active contacts falls under this category. For executing complex postures ($\leq$ 2 fixed active contacts), a more robust controller is required for tackling the slipping issues at the supporting contacts (while making some other movement), and also the controller needs to handle the stability of the robot once the object starts moving.

- The postures were simulated with three different values of contact friction (μ = 1,0.7 and 0.5). For high friction case, pushing posture 1 and supporting postures 6,8,11 and 15 were simulated, while for the other two cases only pushing posture 1 and supporting posture 8 were simulated. As the friction was reduced, the performance of pushing posture was found to be decreasing and for the low friction case, pushing posture 1 was hardly able to move the object. While for the supporting posture, there were only minor issues such as delay in executing the pushing motion and difficulty in getting back to the initial state (after pushing) was observed. The results closely reflects the posture costs evaluated in the posture optimization module, where μ was taken to be 0.5. The cost obtained for pushing posture 1 was relatively higher than that of supporting posture 8, with friction criteria playing a significant part in the former case.

Overall, the controller developed in this thesis meets the general requirements of pushing skill defined in chapter 2, as shown in figure 8-1.

8-2 Future Recommendations

The general controller developed for multi-limb robots for pushing task can be used for both light and heavy weight object manipulation in any given environment, under certain predefined conditions. Compromising the aforementioned conditions, will extend the controller's application further in real time scenarios. The following works are recommended, in order to do so.

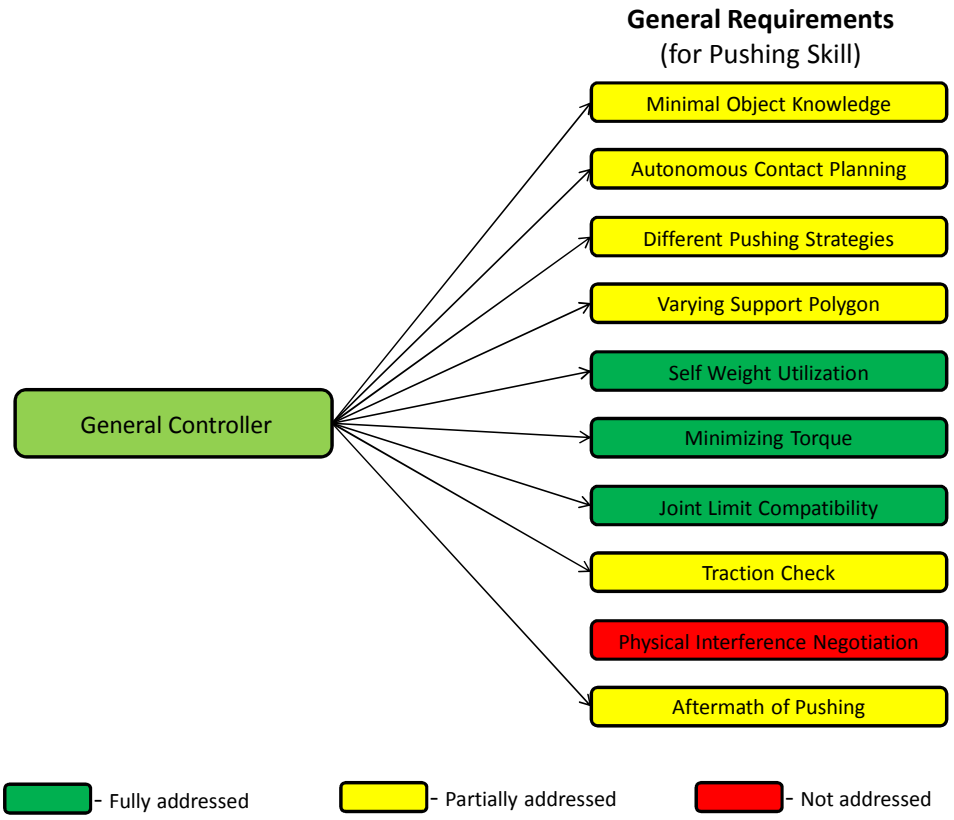


Figure 8-1: General requirements of pushing skill addressed by the General Controller developed in this thesis.

- ***Object handling:*** Currently, the object considered for evaluation are of simple shapes, but in practical case the objects contain much more complex features such as curves, holes, dents, irregular profiles etc. Such complex objects will increase the number of potential pushing and supporting planes. The algorithm developed in this thesis should be suitably modified to handle such objects.

- ***Inter-Object Support handling:*** Under posture tree generation, the algorithm evaluates each supporting object separately and determines the potential contact which forms a supporting posture. The algorithm doesn't evaluate the possibility of combining different supporting postures, doing so will yield supporting postures which use more than one object at the same time for support. These postures can be more effective in supporting the pushing task, involving high force.

- ***End-effector Orientation:*** The complete concept developed here assumes that the end-effectors of the robot to be spherical and capable of making only single point contact with the objects in the environment. This assumption doesn't call for the control of the end-effector's orientation, at any stage of the pushing task. This works well with most of the multi-limb robot such as 4-legged, 6-legged, 8-legged, etc. But with 2-legged robots such as Humanoids, the end-effectors are not spherical anymore and they generally have a series of contact points defined, effectively forming a plane. So, for handling such robots specific end-effector orientation has to be maintained, in order to execute pushing task in a stable manner.

- ***Optimal Posture:*** The time consumed by the optimizer is one of the major concern as far as the optimization is concerned. Though this can be reduced by using multi-core machines with more processing power, it would still be in minutes. For real time applications, it would be much better if the robot can decide the optimal posture within a minute instead of taking 20-25 mins. This can be addressed by using Multi Layer Perceptron (MLP) of neural networks. This basically requires some series of data sets as input and it generates a linear function. Using this linear function, given any pushing force as input it will automatically output the optimal posture.

- ***Complex Posture Handling:*** As stated in the conclusion part, the present controller is capable of executing only simple postures. For handling complex postures, a more robust control is required. One option could be to use both PD control and OSC. For making contact, OSC can be used and then control for the chain of links connected to that contact can be stiffened by switching to PD control and finally, after reaching the desired posture, the force can be exerted as usual by means of OSC as described earlier.

- ***Aftermath of Pushing:*** Presently, the controller tries to stabilize the robot after pushing the object. This can be extended to moving the robot along with the object, which will result in actual manipulation of the object. This can be done by connecting the present controller with the walking pattern generator of the multi-limb robot. Since the walking pattern generator works by ZMP trajectory planning, the trajectory can be modified according to the motion of the object and the pushing force experienced at the end-effectors.

Appendix A

BOBYQA

Bounded Optimization by Quadratic Approximation (BOBYQA), developed by M.J.D Powell [42] is a method for optimizing a function in the absence of derivative information. This method seeks the least value of a function of many variables, by applying a trust region method that forms quadratic models by interpolation. There is usually some freedom in the interpolation conditions, which is taken up by minimizing the Frobenius norm of the change to the second derivative of the model, beginning with the zero matrix. The values of the variables are constrained by upper and lower bounds.

A-1 Function Usage

```
double find_min_bobyqa (
        const funct& f,
        T& x,
        long npt,
        const U& x_lower,
        const U& x_upper,
        const double rho_begin,
        const double rho_end,
        const long max_f_evals
    );
```

A-1-1 Arguments

f	A function that returns the value of the objective at the specified set of parameters x.
x	A numeric vector of starting estimates of the parameters of the objective function.
npt	The number of points used to approximate the objective function via a quadratic approximation.
x_lower	A numeric vector of lower bounds on the parameters.
x_upper	A numeric vector of upper bounds on the parameters.
rho_begin	Initial value of a trust region radius.
rho_end	Final value of a trust region radius. It indicates the convergence accuracy of the final values of the variables.
max_f_evals	The maximum allowed number of function evaluations. If this is exceeded, the method will terminate.

A-1-2 Details

f	The function f must return a scalar numeric value in double.
x	The value of x should be within the lower and upper bounds, i.e it should satisfy the following constraints min(x-x_lower)$\geq$ 0 and min(x_upper-x)$\geq$ 0.
npt	The value of npt must be in the interval $[n+2, (n+1)(n+2)/2]$ where n is the number of parameters in par. Choices that exceed $2n+1$ are not recommended. If not defined, it will be set to $min(2n; n+2)$.
rho_begin	It must be positive, with 0 < rho_end < rho_begin. Typically rhobeg should be about one tenth of the greatest expected change to a variable. If the user does not provide a value, this will be set to $min(0.95, 0.2 * max(abs(par)))$. Note also that smallest difference $abs(upper - lower)$ should be greater than or equal to $rhobeg * 2$. If this is not the case then rho_begin will be adjusted.
rho_end	If not defined, then 1e-6 times the value set for rhobeg will be used.
max_f_evals	It should be greater than 1.

Appendix B

URDF Generation

The Double Pendulum model used for OSC validation is considered here for explaining how to generate URDF. The following figure shows the various links of the model and their corresponding local co-ordinates.

B-1 URDF file Description

The urdf file given below describes the kinematic & dynamic details of Base link and its joint details with its parent link.

```
1. <robot name="Double_Pendulum">
2.   <link name="model_base"/>
3.   <link name="base_link">
4.     <visual>
```

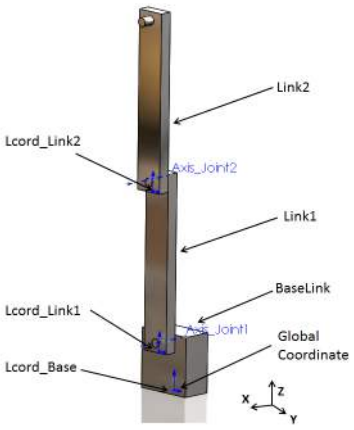


Figure B-1: Links and local co-ordinates of the Double Pendulum model.

```
5.       <geometry>
6.         <mesh filename="file://home/home/rajesh/robmod/src/double_pend/meshes
            /sld_fixed.stl"/>
7.       </geometry>
8.       <origin xyz="0.0 0.0 0.0" rpy="0 0 0"/>
9.       <material name="blue">
10.         <color rgba="0 0 .8 1"/>
11.       </material>
12.     </visual>
13.     <collision>
14.       <geometry>
15.         <mesh filename="file://home/home/rajesh/robmod/src/double_pend/meshes
            /sld_fixed.stl"/>
16.       </geometry>
17.     </collision>
18.     <inertial>
19.       <origin rpy="0.0 0.0 0.0" xyz="0.0 0.0 0.05"/>
20.       <mass value="3.928"/>
21.       <inertia ixx="0.006" ixy="0.0" ixz="0.0" iyy="0.004" iyz="0.0" izz="0.004"/>
22.     </inertial>
23.   </link>

24.   <joint name="base_joint" type="floating">
25.     <parent link="model_base"/>
26.     <child link="base_link"/>
27.     <origin rpy="0 0 0" xyz="0 0 0.0"/>
28.   </joint>
29. </robot>
```

- Line 1, specifies the name of the robot (user defined) and this is followed by the link details.
- Line 2, defines a dummy frame (model_base) representing the global frame of the robot. Since it is a virtual link, no kinematic & dynamic details are specified.
- Line 3-23, defines the Base_link of the robot, by specifying all its kinematics and dynamics details of the link.
- In lines 4-12, the visual properties of the link are defined. It starts with the geometry model which should be used for visualization (line 5-7), its location & orientation wrt global coordinates (line 8) and finally its color code (line 9-11).
- In lines 13-17, the model which should be used for collision is specified. This collision model will be used in the simulator for dynamic computations. Generally a simpler model is preferred in order to save computation time and for stable simulations.
- The inertial properties of the link are specified in lines 18-22. It includes the CoG of the link with respect to the global frame, mass and moment of inertia at its CoG.

- The joint connecting the Base_link to the dummy frame model_base is described in the lines 24-28. First, a name is specified for the joint (user defined), followed by the type of joint. The joint type can be either prismatic or revolute or fixed or floating. Here, floating joint type is used for connecting the two links. These are followed by the name of the parent & child link and the joint origin. For first joint, its origin is specified with respect to the global frame and for other successive joints, their origins are specified with respect to the origin of the previous joints.

The urdf file for other links are also written in the same way as explained above and the complete urdf file for the model is shown below.

```
<?xml version="1.0"?>
<robot name="Double_Pendulum">
  <link name="model_base"/>
  <link name="base_link">
    <visual>
      <geometry>
        <mesh filename="file://home/home/rajesh/robmod/src/double_pend/meshes
        /sld_fixed.stl"/>
      </geometry>
      <origin xyz="0.0 0.0 0.0" rpy="0 0 0"/>
      <material name="blue">
        <color rgba="0 0 .8 1"/>
      </material>
    </visual>
    <collision>
      <geometry>
        <mesh filename="file://home/home/rajesh/robmod/src/double_pend/meshes
        /sld_fixed.stl"/>
      </geometry>
    </collision>
    <inertial>
      <origin rpy="0.0 0.0 0.0" xyz="0.0 0.0 0.05"/>
      <mass value="3.928"/>
      <inertia ixx="0.006" ixy="0.0" ixz="0.0" iyy="0.004" iyz="0.0" izz="0.004"/>
    </inertial>
  </link>

  <joint name="base_joint" type="floating">
    <parent link="model_base"/>
    <child link="base_link"/>
    <origin rpy="0 0 0" xyz="0 0 0.0"/>
  </joint>

  <link name="link1">
    <visual>
      <geometry>
```

```
      <mesh filename="file://home/home/rajesh/robmod/src/double_pend/meshes
      /sld_link1.stl"/>
    </geometry>
    <origin xyz="0.0 0.015 -0.0" rpy="1.570 0 0"/>
    <material name="white">
      <color rgba="1 1 1 1"/>
    </material>
  </visual>
  <collision>
    <geometry>
      <mesh filename="file://home/home/rajesh/robmod/src/double_pend/meshes
      /sld_link1.stl"/>
    </geometry>
  </collision>
  <inertial>
    <origin rpy="0.0 0.0 0.0" xyz="0.0 -0.153 0.0"/>
    <mass value="2.340"/>
    <inertia ixx="0.0006" ixy="0.0" ixz="0.0" iyy="0.017" iyz="0.0" izz="0.018"/>
  </inertial>
</link>

<joint name="base_link1" type="continuous">
  <parent link="base_link"/>
  <child link="link1"/>
  <origin rpy="1.570 3.140 0" xyz="0.0 -0.040 0.085"/>
  <axis xyz="0 0 1"/>
  <limit effort="20" velocity="4.0"/>
  <joint_properties damping="0.0" friction="0.0"/>
</joint>


<link name="link2">
  <visual>
    <geometry>
      <mesh filename="file://home/home/rajesh/robmod/src/double_pend/meshes
      /sld_link1.stl"/>
    </geometry>
    <origin xyz="0.0 0.015 0.0" rpy="1.570 0 0"/>
    <material name="white">
      <color rgba="1 1 1 1"/>
    </material>
  </visual>
  <collision>
    <geometry>
      <mesh filename="file://home/home/rajesh/robmod/src/double_pend/meshes
      /sld_link1.stl"/>
    </geometry>
```

```
    </collision>
    <inertial>
      <origin rpy="0.0 0.0 0.0" xyz="0.0 -0.153 0.0"/>
      <mass value="2.340"/>
      <inertia ixx="0.0006" ixy="0.0" ixz="0.0" iyy="0.017" iyz="0.0" izz="0.018"/>
    </inertial>
  </link>

  <joint name="link1_link2" type="continuous">
    <parent link="link1"/>
    <child link="link2"/>
    <origin rpy="0 0 0" xyz="0.0 -0.270 -0.025"/>
    <axis xyz="0 0 1"/>
    <limit effort="20" velocity="4.0"/>
    <joint_properties damping="0.0" friction="0.0"/>
  </joint>

<link name="end_eff"/>
<joint name="eff_joint" type="fixed">
<parent link="link2"/>
<child link="end_eff"/>
<origin rpy="0 0 0" xyz="0.0 -0.27 0.0"/>
</joint>

</robot>
```

Appendix C

Posture Tree Generation - Input Files

The input files shown below are the ones for the environment and the multi-limb robot ('OctoBot') considered for final evaluation of the complete controller. The following are the input files required for posture tree generation module.

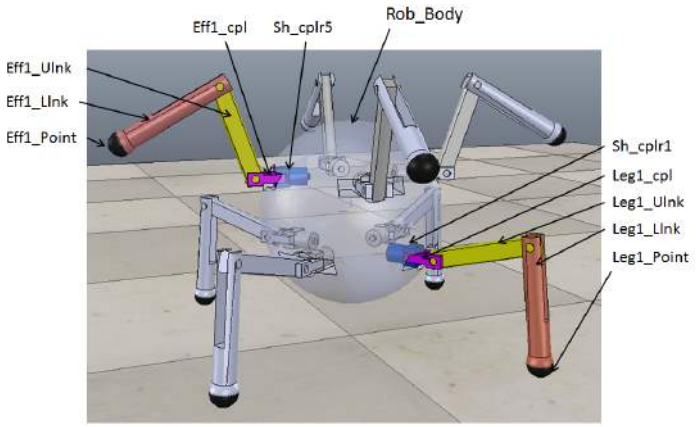
- Contact List
- Robot Model
- Joint Limits
- Env. Details
- Obj. Details

Multi-Limb Robot

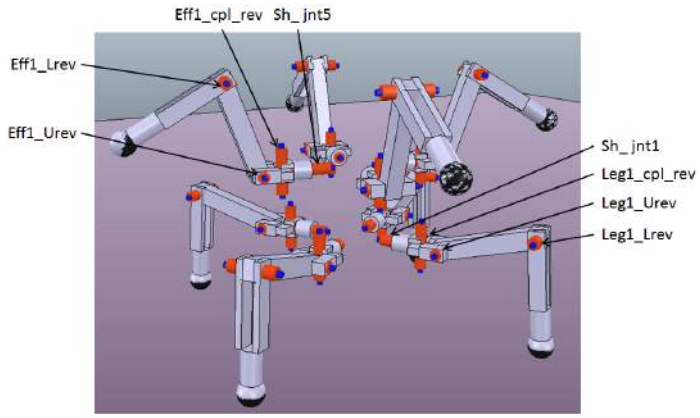
Figure C-1 shows the naming convention adopted for the links and joints of the multi-limb robot, OctoBot. In the figure, the link and joint names are shown only for Leg1 and End-Effector1. Similar convention is followed for other legs and end-effectors. This naming convention is used throughout the input files, which will be shown in the following sections.

C-1 Contact List

```
RobotBase
Rob_Body
ActiveContacts
Leg1_Point
```



(a) Link and contact points naming convention of OctoBot.



(b) Naming convention adopted for OctoBot joints.

Figure C-1: Naming convention of OctoBot (Leg1 & End-Effector1).

```
Leg2_Point
Leg3_Point
Leg4_Point
InactiveContacts
Eff1_Point
Eff2_Point
Eff3_Point
Eff4_Point
```

C-2 Robot Model

```
<?xml version="1.0"?>
<robot name="OctoBot_Glob_Orient">
 <link name="model_base"/>
  <link name="Rob_Body">
    <inertial>
      <origin
        xyz="-2.4355014081623E-18 2.43550140816235E-18 -1.49131450191244E-34"
        rpy="0 0 0" />
      <mass value="14.1371669411541" />
      <inertia
        ixx="0.127234502470387"
        ixy="-1.68052222445238E-48"
        ixz="2.88380513280089E-34"
        iyy="0.127234502470387"
        iyz="7.70371977754894E-34"
        izz="0.127234502470387" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Rob_Body.STL" />
      </geometry>
      <material name="">
        <color rgba="0.384313725490196 0.384313725490196 0.384313725490196 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Rob_Body.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="base_joint" type="floating">
    <parent link="model_base"/>
```

```
    <child link="Rob_Body"/>
    <origin rpy="0 0 0" xyz="0 0 0.21238"/>
  </joint>
  <link name="Sh_cplr1">
    <inertial>
      <origin xyz="-3.16192461262062E-18 0 0.00723224561176804"
        rpy="0 0 0" />
      <mass value="0.238764004385893" />
      <inertia
        ixx="5.64246E-04"
        ixy="-5.1848511750621E-22"
        ixz="1.8412173417568E-21"
        iyy="5.30469E-04"
        iyz="4.54179004542963E-21"
        izz="2.72889E-04" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr1.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr1.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Sh_jnt1" type="revolute">
    <origin xyz="-0.06364 0.06364 -0.06"
      rpy="1.5708 -4.3298E-17 -2.3562" />
    <parent link="Rob_Body" />
    <child link="Sh_cplr1" />
    <axis xyz="0 0 1" />
    <limit lower="-1.57" upper="1.57" effort="5" velocity="1" />
  </joint>
  <link name="Leg1_cpl">
    <inertial>
      <origin xyz="-2.33916908241978E-17 -0.0199261992619927 -1.38777878078145E-17"
        rpy="0 0 0" />
      <mass value="0.42276" />
      <inertia
        ixx="0.0011078"
```

```
        ixy="-4.02027537470174E-20"
        ixz="-5.99160882215575E-21"
        iyy="0.001917"
        iyz="8.18468148262408E-20"
        izz="0.0028438" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg1_cpl.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg1_cpl.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Leg1_cpl_rev" type="revolute">
    <origin xyz="0 0 0.0315"
      rpy="-1.5708 5.0532E-16 -3.7492E-33" />
    <parent link="Sh_cplr1" />
    <child link="Leg1_cpl" />
    <axis xyz="0 0 1" />
    <limit lower="-0.92" upper="0.92" effort="5" velocity="1" />
  </joint>
  <link name="Leg1_Ulnk">
    <inertial>
      <origin xyz="-0.0774999999999994 -9.43689570931383E-16 -4.83508993838015E-17"
        rpy="0 0 0" />
      <mass value="1.02375" />
      <inertia
        ixx="0.0001301015625"
        ixy="2.79859685772821E-18"
        ixz="6.68313867232251E-19"
        iyy="0.002666015625"
        iyz="2.09781145090998E-20"
        izz="0.0026894765625" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg1_Ulnk.STL" />
```

```
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg1_Ulnk.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Leg1_Urev" type="revolute">
    <origin xyz="0 -0.0375 0" rpy="1.5708 0.2618 1.5708" />
    <parent link="Leg1_cpl" />
    <child link="Leg1_Ulnk" />
    <axis xyz="0 0 1" />
    <limit lower="-1.2" upper="1.35" effort="5" velocity="1" />
  </joint>
  <link name="Leg1_Llnk">
    <inertial>
      <origin xyz="0.102726769503843 -0.0089874277722275 -6.66389039863417E-17"
        rpy="0 0 0" />
      <mass value="1.02455526413118" />
      <inertia
        ixx="0.000222746338725306"
        ixy="-0.000323367079269386"
        ixz="6.31268140547286E-19"
        iyy="0.00389055801417182"
        iyz="-1.54168092421445E-19"
        izz="0.00384800912401538" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg1_Llnk.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg1_Llnk.STL" />
      </geometry>
    </collision>
```

```
</link>
<joint name="Leg1_Lrev" type="revolute">
  <origin xyz="-0.155 0 0"
          rpy="1.5848E-17 5.9144E-17 -1.309" />
  <parent link="Leg1_Ulnk" />
  <child link="Leg1_Llnk" />
  <axis xyz="0 0 1" />
  <limit lower="-1.75" upper="1.13" effort="5" velocity="1" />
</joint>
 <link name="Leg1_Point"/>
<joint name="Leg1_Con" type="fixed">
<parent link="Leg1_Llnk"/>
  <child link="Leg1_Point"/>
  <origin rpy="0 0 0" xyz="0.19323 -0.0125 0"/>
</joint>
<link name="Sh_cplr2">
  <inertial>
    <origin xyz="4.04797106920574E-17 0 0.00723224561176805"
      rpy="0 0 0" />
    <mass value="0.238764004385893" />
    <inertia
      ixx="5.64246E-04"
      ixy="-5.1848511750621E-22"
      ixz="1.8412173417568E-21"
      iyy="5.30469E-04"
      iyz="4.54179004542963E-21"
      izz="2.72889E-04" />
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr2.STL" />
    </geometry>
    <material name="">
      <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr2.STL" />
    </geometry>
  </collision>
</link>
<joint name="Sh_jnt2" type="revolute">
  <origin xyz="-0.06364 -0.06364 -0.06"
    rpy="1.5708 -4.3298E-17 -0.7854" />
```

```
    <parent link="Rob_Body" />
    <child link="Sh_cplr2" />
    <axis xyz="0 0 1" />
    <limit lower="-1.57" upper="1.57" effort="5" velocity="1" />
  </joint>
  <link name="Leg2_cpl">
    <inertial>
      <origin xyz="1.06434770357789E-17 -0.0199261992619927 -6.93889390390723E-18"
        rpy="0 0 0" />
      <mass value="0.42276" />
      <inertia
        ixx="0.0011078"
        ixy="-4.02027537470174E-20"
        ixz="-5.99160882215575E-21"
        iyy="0.001917"
        iyz="8.18468148262408E-20"
        izz="0.0028438" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg2_cpl.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg2_cpl.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Leg2_cpl_rev" type="revolute">
    <origin xyz="0 0 0.0315" rpy="-1.5708 0 0" />
    <parent link="Sh_cplr2" />
    <child link="Leg2_cpl" />
    <axis xyz="0 0 1" />
    <limit lower="-0.92" upper="0.92" effort="5" velocity="1" />
  </joint>
  <link name="Leg2_Ulnk">
    <inertial>
      <origin
        xyz="-0.0775 -2.77555756156289E-17 -7.02788191778439E-18"
        rpy="0 0 0" />
      <mass value="1.02375" />
```

```
      <inertia
        ixx="0.0001301015625"
        ixy="1.54159996400283E-18"
        ixz="-1.16630931592887E-20"
        iyy="0.002666015625"
        iyz="-2.10902956568898E-20"
        izz="0.0026894765625" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg2_Ulnk.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg2_Ulnk.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Leg2_Urev" type="revolute">
    <origin xyz="0 -0.0375 0" rpy="1.5708 0.2618 1.5708" />
    <parent link="Leg2_cpl" />
    <child link="Leg2_Ulnk" />
    <axis xyz="0 0 1" />
    <limit lower="-1.2" upper="1.35" effort="5" velocity="1" />
  </joint>
  <link name="Leg2_Llnk">
    <inertial>
      <origin xyz="0.103119169074701 -1.11022302462516E-16 -2.48480781519943E-16"
        rpy="0 0 0" />
      <mass value="1.02455526413118" />
      <inertia
        ixx="0.000194455385131744"
        ixy="1.10961316090313E-18"
        ixz="6.85118389467933E-19"
        iyy="0.00391884896776538"
        iyz="-2.13920594394574E-19"
        izz="0.00384800912401538" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
```

```
      <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg2_Llnk.STL" />
    </geometry>
    <material name="">
      <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg2_Llnk.STL" />
    </geometry>
  </collision>
</link>
<joint name="Leg2_Lrev" type="revolute">
  <origin xyz="-0.155 0 0"
   rpy="2.3183E-16 7.9653E-17 -1.3963" />
  <parent link="Leg2_Ulnk" />
  <child link="Leg2_Llnk" />
  <axis xyz="0 0 1" />
  <limit lower="-1.75" upper="1.13" effort="5" velocity="1" />
</joint>
<link name="Leg2_Point"/>
<joint name="Leg2_Con" type="fixed">
<parent link="Leg2_Llnk"/>
  <child link="Leg2_Point"/>
  <origin rpy="0 0 0" xyz="0.19323 0 0"/>
</joint>
<link name="Sh_cplr3">
  <inertial>
    <origin xyz="-2.34707904719658E-18 0 0.00723224561176809"
      rpy="0 0 0" />
    <mass value="0.238764004385893" />
    <inertia
      ixx="5.64246E-04"
      ixy="-5.1848511750621E-22"
      ixz="1.8412173417568E-21"
      iyy="5.30469E-04"
      iyz="4.54179004542963E-21"
      izz="2.72889E-04" />
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr3.STL" />
    </geometry>
    <material name="">
      <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
```

```
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr3.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Sh_jnt3" type="revolute">
    <origin xyz="0.06364 -0.06364 -0.06"
      rpy="1.5708 4.3298E-17 0.7854" />
    <parent link="Rob_Body" />
    <child link="Sh_cplr3" />
    <axis xyz="0 0 1" />
    <limit lower="-1.57" upper="1.57" effort="5" velocity="1" />
  </joint>
  <link name="Leg3_cpl">
    <inertial>
      <origin xyz="-1.0719981327889E-17 -0.0199261992619926 -6.93889390390723E-18"
        rpy="0 0 0" />
      <mass value="0.42276" />
      <inertia
        ixx="0.0011078"
        ixy="-4.02027537470174E-20"
        ixz="-5.99160882215575E-21"
        iyy="0.001917"
        iyz="8.18468148262408E-20"
        izz="0.0028438" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg3_cpl.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg3_cpl.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Leg3_cpl_rev" type="revolute">
```

```
    <origin xyz="0 0 0.0315" rpy="-1.5708 1.6081E-16 0" />
    <parent link="Sh_cplr3" />
    <child link="Leg3_cpl" />
    <axis xyz="0 0 1" />
    <limit lower="-0.92" upper="0.92" effort="5" velocity="1" />
  </joint>
  <link name="Leg3_Ulnk">
    <inertial>
      <origin xyz="-3.33066907387547E-16 -0.0775 2.38388902011233E-17"
        rpy="0 0 0" />
      <mass value="1.02375" />
      <inertia
        ixx="0.002666015625"
        ixy="9.46982835030308E-18"
        ixz="-6.48600318130043E-20"
        iyy="0.0001301015625"
        iyz="-4.40735398246417E-19"
        izz="0.0026894765625" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg3_Ulnk.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg3_Ulnk.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Leg3_Urev" type="revolute">
    <origin xyz="0 -0.0375 0" rpy="-1.5708 1.309 -1.5708" />
    <parent link="Leg3_cpl" />
    <child link="Leg3_Ulnk" />
    <axis xyz="0 0 1" />
    <limit lower="-1.2" upper="1.35" effort="5" velocity="1" />
  </joint>
  <link name="Leg3_Llnk">
    <inertial>
      <origin xyz="-0.0089874277722275 -0.102726769503843 -3.30657197023219E-17"
        rpy="0 0 0" />
      <mass value="1.02455526413118" />
```

```
      <inertia
        ixx="0.00389055801417182"
        ixy="0.000323367079269385"
        ixz="-5.95155684607128E-20"
        iyy="0.000222746338725305"
        iyz="1.09747995874938E-19"
        izz="0.00384800912401538" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg3_Llnk.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg3_Llnk.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Leg3_Lrev" type="revolute">
    <origin xyz="0 -0.155 0"
      rpy="1.0377E-15 2.0452E-16 1.8326" />
    <parent link="Leg3_Ulnk" />
    <child link="Leg3_Llnk" />
    <axis xyz="0 0 1" />
    <limit lower="-1.75" upper="1.13" effort="5" velocity="1" />
  </joint>
  <link name="Leg3_Point"/>
  <joint name="Leg3_Con" type="fixed">
  <parent link="Leg3_Llnk"/>
    <child link="Leg3_Point"/>
    <origin rpy="0 0 0" xyz="-0.0125 -0.19323 0"/>
  </joint>
  <link name="Sh_cplr4">
    <inertial>
      <origin xyz="6.78435459185761E-17 -6.93889390390723E-18 0.00723224561176809"
        rpy="0 0 0" />
      <mass value="0.238764004385893" />
      <inertia
        ixx="5.64246E-04"
        ixy="-5.1848511750621E-22"
        ixz="1.8412173417568E-21"
```

```
        iyy="5.30469E-04"
        iyz="4.54179004542963E-21"
        izz="2.72889E-04" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr4.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr4.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Sh_jnt4" type="revolute">
    <origin xyz="0.06364 0.06364 -0.06"
      rpy="1.5708 4.3298E-17 2.3562" />
    <parent link="Rob_Body" />
    <child link="Sh_cplr4" />
    <axis xyz="0 0 1" />
    <limit lower="-1.57" upper="1.57" effort="5" velocity="1" />
  </joint>
  <link name="Leg4_cpl">
    <inertial>
      <origin xyz="4.36858531168106E-16 -0.0199261992619928 -1.38777878078145E-17"
        rpy="0 0 0" />
      <mass value="0.42276" />
      <inertia
        ixx="0.0011078"
        ixy="-4.02027537470174E-20"
        ixz="-5.99160882215575E-21"
        iyy="0.001917"
        iyz="8.18468148262408E-20"
        izz="0.0028438" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg4_cpl.STL" />
      </geometry>
      <material name="">
```

```
      <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg4_cpl.STL" />
    </geometry>
  </collision>
</link>
<joint name="Leg4_cpl_rev" type="revolute">
  <origin xyz="0 0 0.0315" rpy="-1.5708 0 0" />
  <parent link="Sh_cplr4" />
  <child link="Leg4_cpl" />
  <axis xyz="0 0 1" />
  <limit lower="-0.92" upper="0.92" effort="5" velocity="1" />
</joint>
<link name="Leg4_Ulnk">
  <inertial>
    <origin xyz="-3.19189119579733E-16 -0.0774999999999999 -5.3022794168959E-16"
      rpy="0 0 0" />
    <mass value="1.02375" />
    <inertia
      ixx="0.002666015625"
      ixy="1.03304138247134E-17"
      ixz="9.96865318295566E-21"
      iyy="0.0001301015625"
      iyz="1.72147472888006E-17"
      izz="0.0026894765625" />
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg4_Ulnk.STL" />
    </geometry>
    <material name="">
      <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg4_Ulnk.STL" />
    </geometry>
  </collision>
</link>
<joint name="Leg4_Urev" type="revolute">
```

```
    <origin xyz="0 -0.0375 0" rpy="-1.5708 1.309 -1.5708" />
    <parent link="Leg4_cpl" />
    <child link="Leg4_Ulnk" />
    <axis xyz="0 0 1" />
    <limit lower="-1.2" upper="1.35" effort="5" velocity="1" />
  </joint>
  <link name="Leg4_Llnk">
    <inertial>
      <origin xyz="0.00898742777222589 0.102726769503843 -5.69290400573669E-16"
        rpy="0 0 0" />
      <mass value="1.02455526413118" />
      <inertia
        ixx="0.00389055801417183"
        ixy="0.000323367079269333"
        ixz="3.31431132106402E-19"
        iyy="0.000222746338725296"
        iyz="-5.41283427566867E-19"
        izz="0.00384800912401538" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg4_Llnk.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Leg4_Llnk.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Leg4_Lrev" type="revolute">
    <origin xyz="0 -0.155 0"
      rpy="1.7046E-15 3.4284E-15 -1.309" />
    <parent link="Leg4_Ulnk" />
    <child link="Leg4_Llnk" />
    <axis xyz="0 0 1" />
    <limit lower="-1.75" upper="1.13" effort="5" velocity="1" />
  </joint>
  <link name="Leg4_Point"/>
  <joint name="Leg4_Con" type="fixed">
  <parent link="Leg4_Llnk"/>
    <child link="Leg4_Point"/>
```

```
    <origin rpy="0 0 0" xyz="0.0125 0.19323 0"/>
  </joint>
  <link name="Sh_cplr5">
    <inertial>
      <origin xyz="9.22872889219661E-16 0 0.0184822456117681"
        rpy="0 0 0" />
      <mass value="0.238764004385893" />
      <inertia
        ixx="5.64246E-04"
        ixy="-5.1848511750621E-22"
        ixz="1.8412173417568E-21"
        iyy="5.30469E-04"
        iyz="4.54179004542963E-21"
        izz="2.72889E-04" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr5.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr5.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Sh_jnt5" type="revolute">
    <origin xyz="0.07875 0 0.06"
      rpy="1.5708 6.1232E-17 1.5708" />
    <parent link="Rob_Body" />
    <child link="Sh_cplr5" />
    <axis xyz="0 0 1" />
    <limit lower="-1.57" upper="1.57" effort="5" velocity="1" />
  </joint>
  <link name="Eff1_cpl">
    <inertial>
      <origin xyz="7.7715611723761E-16 -0.0199261992619926 -2.08166817117217E-17"
        rpy="0 0 0" />
      <mass value="0.42276" />
      <inertia
        ixx="0.0011078"
        ixy="-4.02027537470174E-20"
```

```
        ixz="-5.99160882215575E-21"
        iyy="0.001917"
        iyz="8.18468148262408E-20"
        izz="0.0028438" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff1_cpl.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff1_cpl.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Eff1_cpl_rev" type="revolute">
    <origin xyz="0 0 0.04275"
      rpy="-1.5708 -7.2164E-16 5.3248E-30" />
    <parent link="Sh_cplr5" />
    <child link="Eff1_cpl" />
    <axis xyz="0 0 1" />
    <limit lower="-0.92" upper="0.92" effort="5" velocity="1" />
  </joint>
  <link name="Eff1_Ulnk">
    <inertial>
      <origin xyz="0.0775000000000001 -5.55111512312578E-17 1.68342942118537E-17"
        rpy="0 0 0" />
      <mass value="1.02375" />
      <inertia
        ixx="0.0001301015625"
        ixy="6.67461962436389E-19"
        ixz="7.55426414561436E-19"
        iyy="0.002666015625"
        iyz="-1.20512021073499E-19"
        izz="0.0026894765625" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff1_Ulnk.STL" />
      </geometry>
```

```
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff1_Ulnk.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Eff1_Urev" type="revolute">
    <origin xyz="0 -0.0375 0" rpy="-1.5708 -1.2217 -1.5708" />
    <parent link="Eff1_cpl" />
    <child link="Eff1_Ulnk" />
    <axis xyz="0 0 1" />
    <limit lower="-0.95" upper="2.62" effort="5" velocity="1" />
  </joint>
  <link name="Eff1_Llnk">
    <inertial>
      <origin xyz="1.11022302462516E-16 -0.103119169074701 4.68871513908462E-15"
        rpy="0 0 0" />
      <mass value="1.02455526413118" />
      <inertia
        ixx="0.00391884896776538"
        ixy="-3.69983991360678E-18"
        ixz="-2.256527547601E-19"
        iyy="0.000194455385131744"
        iyz="-1.66477148208726E-16"
        izz="0.00384800912401539" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff1_Llnk.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff1_Llnk.STL" />
      </geometry>
    </collision>
  </link>
```

```
<joint name="Eff1_Lrev" type="revolute">
  <origin xyz="0.155 0 0"
    rpy="4.4777E-14 6.0936E-16 -2.9671" />
  <parent link="Eff1_Ulnk" />
  <child link="Eff1_Llnk" />
  <axis xyz="0 0 1" />
  <limit lower="-1.75" upper="1.13" effort="5" velocity="1" />
</joint>
<link name="Eff1_Point"/>
<joint name="Eff1_Con" type="fixed">
<parent link="Eff1_Llnk"/>
  <child link="Eff1_Point"/>
  <origin rpy="-1.57 3.14 1.22" xyz="0 -0.19323 0"/>
</joint>
<link name="Sh_cplr6">
  <inertial>
    <origin xyz="-9.92261828258734E-16 -6.93889390390723E-18 -0.011017754388242"
      rpy="0 0 0" />
    <mass value="0.238764004385893" />
    <inertia
      ixx="5.64246E-04"
      ixy="-5.1848511750621E-22"
      ixz="1.8412173417568E-21"
      iyy="5.30469E-04"
      iyz="4.54179004542963E-21"
      izz="2.72889E-04" />
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr6.STL" />
    </geometry>
    <material name="">
      <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr6.STL" />
    </geometry>
  </collision>
</link>
<joint name="Sh_jnt6" type="revolute">
  <origin xyz="0 -0.10825 0.06"
    rpy="1.5708 -8.9979E-31 -1.8097E-14" />
  <parent link="Rob_Body" />
```

```
    <child link="Sh_cplr6" />
    <axis xyz="0 0 1" />
    <limit lower="-1.57" upper="1.57" effort="5" velocity="1" />
  </joint>
  <link name="Eff2_cpl">
    <inertial>
      <origin xyz="1.5404344466674E-15 -0.0199261992619777 1.55084278752327E-14"
        rpy="0 0 0" />
      <mass value="0.42276" />
      <inertia
        ixx="0.0011078"
        ixy="-4.02027537470174E-20"
        ixz="-5.99160882215575E-21"
        iyy="0.001917"
        iyz="8.18468148262408E-20"
        izz="0.0028438" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff2_cpl.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff2_cpl.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Eff2_cpl_rev" type="revolute">
    <origin xyz="0 0 0.01325" rpy="-1.5708 -1.1507E-13 0" />
    <parent link="Sh_cplr6" />
    <child link="Eff2_cpl" />
    <axis xyz="0 0 1" />
    <limit lower="-0.92" upper="0.92" effort="5" velocity="1" />
  </joint>
  <link name="Eff2_Ulnk">
    <inertial>
      <origin xyz="-0.077500000000008 1.65145674912992E-14 -5.29467394465628E-17"
        rpy="0 0 0" />
      <mass value="1.02375" />
      <inertia
        ixx="0.0001301015625"
```

```
        ixy="-2.03287907341032E-19"
        ixz="-2.15501867080603E-19"
        iyy="0.002666015625"
        iyz="3.00538471363261E-19"
        izz="0.0026894765625" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff2_Ulnk.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff2_Ulnk.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Eff2_Urev" type="revolute">
    <origin xyz="0 -0.0375 0" rpy="1.5708 1.2217 1.5708" />
    <parent link="Eff2_cpl" />
    <child link="Eff2_Ulnk" />
    <axis xyz="0 0 1" />
    <limit lower="-0.95" upper="2.62" effort="5" velocity="1" />
  </joint>
  <link name="Eff2_Llnk">
    <inertial>
      <origin xyz="-6.27276008913213E-15 -0.103119169074678 -4.79584686425135E-17"
        rpy="0 0 0" />
      <mass value="1.02455526413118" />
      <inertia
        ixx="0.00391884896776538"
        ixy="-9.48676900924816E-20"
        ixz="-2.38929191124371E-20"
        iyy="0.000194455385131744"
        iyz="3.63747477629144E-18"
        izz="0.00384800912401538" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff2_Llnk.STL" />
      </geometry>
```

```
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff2_Llnk.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Eff2_Lrev" type="revolute">
    <origin xyz="-0.155 0 0"
       rpy="-2.8371E-16 -1.902E-15 0.17453" />
    <parent link="Eff2_Ulnk" />
    <child link="Eff2_Llnk" />
    <axis xyz="0 0 1" />
    <limit lower="-1.75" upper="1.13" effort="5" velocity="1" />
  </joint>
  <link name="Eff2_Point"/>
  <joint name="Eff2_Con" type="fixed">
  <parent link="Eff2_Llnk"/>
    <child link="Eff2_Point"/>
    <origin rpy="0 0 0" xyz="0 -0.19323 0"/>
  </joint>
  <link name="Sh_cplr7">
    <inertial>
      <origin xyz="-6.93889390390723E-18 0 0.0184822456117681"
        rpy="0 0 0" />
      <mass value="0.238764004385893" />
      <inertia
        ixx="5.64246E-04"
        ixy="-5.1848511750621E-22"
        ixz="1.8412173417568E-21"
        iyy="5.30469E-04"
        iyz="4.54179004542963E-21"
        izz="2.72889E-04" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr7.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
```

```
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr7.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Sh_jnt7" type="revolute">
    <origin xyz="-0.07875 0 0.06"
      rpy="1.5708 -6.1232E-17 -1.5708" />
    <parent link="Rob_Body" />
    <child link="Sh_cplr7" />
    <axis xyz="0 0 1" />
    <limit lower="-1.57" upper="1.57" effort="5" velocity="1" />
  </joint>
  <link name="Eff3_cpl">
    <inertial>
      <origin xyz="-2.64371857738865E-15 -0.0199261992619926 0"
        rpy="0 0 0" />
      <mass value="0.42276" />
      <inertia
       ixx="0.0011078"
       ixy="-4.02027537470174E-20"
       ixz="-5.99160882215575E-21"
       iyy="0.001917"
       iyz="8.18468148262408E-20"
       izz="0.0028438" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff3_cpl.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff3_cpl.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Eff3_cpl_rev" type="revolute">
    <origin xyz="0 0 0.04275" rpy="-1.5708 1.6753E-13 0" />
    <parent link="Sh_cplr7" />
```

```
    <child link="Eff3_cpl" />
    <axis xyz="0 0 1" />
    <limit lower="-0.92" upper="0.92" effort="5" velocity="1" />
  </joint>
  <link name="Eff3_Ulnk">
    <inertial>
      <origin xyz="-0.0775 0 3.9883570575756E-17" rpy="0 0 0" />
      <mass value="1.02375" />
      <inertia
        ixx="0.0001301015625"
        ixy="-2.3039296165317E-19"
        ixz="2.5598267864923E-19"
        iyy="0.002666015625"
        iyz="1.35206956266724E-19"
        izz="0.0026894765625" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff3_Ulnk.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff3_Ulnk.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Eff3_Urev" type="revolute">
    <origin xyz="0 -0.0375 0" rpy="1.5708 1.2217 1.5708" />
    <parent link="Eff3_cpl" />
    <child link="Eff3_Ulnk" />
    <axis xyz="0 0 1" />
    <limit lower="-0.95" upper="2.62" effort="5" velocity="1" />
  </joint>
  <link name="Eff3_Llnk">
    <inertial>
      <origin xyz="0 -0.103119169074701 2.39217396293074E-16"
        rpy="0 0 0" />
      <mass value="1.02455526413117" />
      <inertia
        ixx="0.00391884896776538"
        ixy="-5.42101086242752E-19"
```

```
        ixz="-5.76598101194397E-20"
        iyy="0.000194455385131744"
        iyz="-8.63226220374726E-18"
        izz="0.00384800912401538" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff3_Llnk.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff3_Llnk.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Eff3_Lrev" type="revolute">
    <origin xyz="-0.155 0 0"
       rpy="1.93E-15 -4.4066E-16 0.17453" />
    <parent link="Eff3_Ulnk" />
    <child link="Eff3_Llnk" />
    <axis xyz="0 0 1" />
    <limit lower="-1.75" upper="1.13" effort="5" velocity="1" />
  </joint>
   <link name="Eff3_Point"/>
  <joint name="Eff3_Con" type="fixed">
  <parent link="Eff3_Llnk"/>
    <child link="Eff3_Point"/>
    <origin rpy="0 0 0" xyz="0 -0.19323 0"/>
  </joint>
  <link name="Sh_cplr8">
    <inertial>
      <origin xyz="-9.78384040450919E-16 6.93889390390723E-18 -0.0110177543882316"
        rpy="0 0 0" />
      <mass value="0.238764004385893" />
      <inertia
        ixx="5.64246E-04"
        ixy="-5.1848511750621E-22"
        ixz="1.8412173417568E-21"
        iyy="5.30469E-04"
        iyz="4.54179004542963E-21"
        izz="2.72889E-04" />
```

```
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr8.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh         filename="package://OctoBot_Glob_Orient/meshes/Sh_cplr8.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Sh_jnt8" type="revolute">
    <origin xyz="0 0.10825 0.06"
      rpy="1.5708 1.0107E-30 3.1416" />
    <parent link="Rob_Body" />
    <child link="Sh_cplr8" />
    <axis xyz="0 0 1" />
    <limit lower="-1.57" upper="1.57" effort="5" velocity="1" />
  </joint>
  <link name="Eff4_cpl">
    <inertial>
      <origin xyz="7.43849426498855E-15 -0.0199261992619926 0"
        rpy="0 0 0" />
      <mass value="0.42276" />
      <inertia
         ixx="0.0011078"
        ixy="-4.02027537470174E-20"
        ixz="-5.99160882215575E-21"
        iyy="0.001917"
        iyz="8.18468148262408E-20"
        izz="0.0028438" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff4_cpl.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
```

```
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff4_cpl.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Eff4_cpl_rev" type="revolute">
    <origin xyz="0 0 0.01325" rpy="-1.5708 -4.044E-13 0" />
    <parent link="Sh_cplr8" />
    <child link="Eff4_cpl" />
    <axis xyz="0 0 1" />
    <limit lower="-0.92" upper="0.92" effort="5" velocity="1" />
  </joint>
  <link name="Eff4_Ulnk">
    <inertial>
      <origin xyz="-0.0775 0 -2.51911638309394E-17"
        rpy="0 0 0" />
      <mass value="1.02375" />
      <inertia
        ixx="0.0001301015625"
        ixy="-4.16740210049116E-19"
        ixz="-1.84352862786801E-18"
        iyy="0.002666015625"
        iyz="-2.97979504365207E-19"
        izz="0.0026894765625" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff4_Ulnk.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff4_Ulnk.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Eff4_Urev" type="revolute">
    <origin xyz="0 -0.0375 0" rpy="1.5708 1.2217 1.5708" />
    <parent link="Eff4_cpl" />
    <child link="Eff4_Ulnk" />
```

```
    <axis xyz="0 0 1" />
    <limit lower="-0.95" upper="2.62" effort="5" velocity="1" />
  </joint>
  <link name="Eff4_Llnk">
    <inertial>
      <origin xyz="-5.55111512312578E-17 -0.103119169074701 -6.68272278592785E-17"
        rpy="0 0 0" />
      <mass value="1.02455526413118" />
      <inertia
        ixx="0.00391884896776538"
        ixy="3.46944695195361E-18"
        ixz="3.17701401910254E-19"
        iyy="0.000194455385131744"
        iyz="4.39395007935499E-18"
        izz="0.00384800912401539" />
    </inertial>
    <visual>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff4_Llnk.STL" />
      </geometry>
      <material name="">
        <color rgba="0.752941176470588 0.752941176470588 0.752941176470588 1" />
      </material>
    </visual>
    <collision>
      <origin xyz="0 0 0" rpy="0 0 0" />
      <geometry>
        <mesh filename="package://OctoBot_Glob_Orient/meshes/Eff4_Llnk.STL" />
      </geometry>
    </collision>
  </link>
  <joint name="Eff4_Lrev" type="revolute">
    <origin xyz="-0.155 0 0"
      rpy="-7.6705E-16 -3.0468E-16 0.17453" />
    <parent link="Eff4_Ulnk" />
    <child link="Eff4_Llnk" />
    <axis xyz="0 0 1" />
    <limit lower="-1.75" upper="1.13" effort="5" velocity="1" />
  </joint>
   <link name="Eff4_Point"/>
  <joint name="Eff4_Con" type="fixed">
  <parent link="Eff4_Llnk"/>
    <child link="Eff4_Point"/>
    <origin rpy="0 0 0" xyz="0 -0.19323 0"/>
  </joint>
</robot>
```

C-3 Joint Limits

First & second column refers to the lower and upper limit of the joints respectively. First 6 rows represent the 6 floating DOF (not actuated), which are set to 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
-1.57 1.57
-0.92 0.92
-1.20 1.35
-1.75 1.13
-1.57 1.57
-0.92 0.92
-1.20 1.35
-1.75 1.13
-1.57 1.57
-0.92 0.92
-1.20 1.35
-1.75 1.13
-1.57 1.57
-0.92 0.92
-1.20 1.35
-1.75 1.13
-1.57 1.57
-0.92 0.92
-0.95 2.62
-1.75 1.13
-1.57 1.57
-0.92 0.92
-0.95 2.62
-1.75 1.13
-1.57 1.57
-0.92 0.92
-0.95 2.62
-1.75 1.13
-1.57 1.57
-0.92 0.92
-0.95 2.62
-1.75 1.13
```

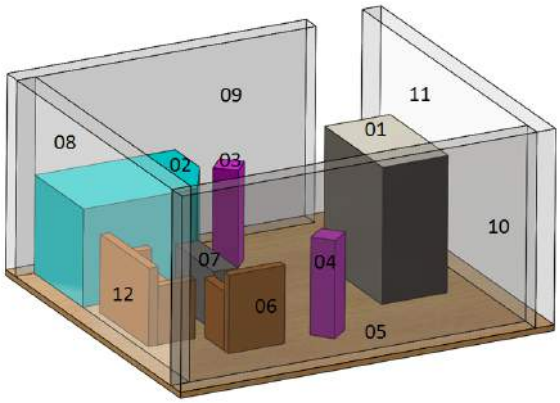


Figure C-2: Object IDs considered for referring each object in the environment, inside the input files (IDs are placed on the object they refer to).

Environment Model

Figure C-2, shows the IDs used for referring each object present in the environment. These IDs are used to name the object detail files extracted from '.stl' files using the stlreader. The vertex, face and norm files of a particular object are named as vXX, fXX and nXX respectively, where XX refers to the object. For example, if the object ID is 11, then it's vertex, face and norm detail files are named as v11, f11 & n11 respectively.

C-4 Environment Details (Env.Details)

The pushing, supporting and active object file names, which contain details such as vertex, face and norm are specified separately for the Posture tree generation module to access respective object files and store the data for further processing.

```
PushingObj:Vert
v01
PushingObj:Face
f01
PushingObj:Norm
n01
PushingEnd
SupportingObj:Vert
v02
v03
v04
v06
v07
SupportingObj:Face
f02
f03
```

```
f04
f06
f07
SupportingObj:Norm
n02
n03
n04
n06
n07
SupportingEnd
ActiveObj:Vert
v05
ActiveObj:Face
f05
ActiveObj:Norm
n05
ActiveEnd
```

C-5 Object details (Obj. Details)

The vertex, face and norm details of pushing object (object ID - 01) and one supporting object (object ID: 07) are shown below. Similarly, for other objects its respective details are extracted from stl file using the stlreader.

Pushing Object

v01.txt - xyz values of the vertices

```
1.400000 1.209000 0.030300
1.400000 1.209000 0.730300
1.750000 1.209000 0.030300
1.750000 1.209000 0.030300
1.400000 1.209000 0.730300
1.750000 1.209000 0.730300
1.400000 0.609000 0.030300
1.400000 1.209000 0.030300
1.750000 0.609000 0.030300
1.750000 0.609000 0.030300
1.400000 1.209000 0.030300
1.750000 1.209000 0.030300
1.400000 0.609000 0.730300
1.400000 0.609000 0.030300
1.750000 0.609000 0.730300
1.750000 0.609000 0.730300
1.400000 0.609000 0.030300
```

```
1.750000 0.609000 0.030300
1.400000 1.209000 0.730300
1.400000 0.609000 0.730300
1.750000 1.209000 0.730300
1.750000 1.209000 0.730300
1.400000 0.609000 0.730300
1.750000 0.609000 0.730300
1.400000 0.609000 0.030300
1.400000 0.609000 0.730300
1.400000 1.209000 0.030300
1.400000 1.209000 0.030300
1.400000 0.609000 0.730300
1.400000 1.209000 0.730300
1.750000 0.609000 0.730300
1.750000 0.609000 0.030300
1.750000 1.209000 0.730300
1.750000 1.209000 0.730300
1.750000 0.609000 0.030300
1.750000 1.209000 0.030300
```

f01.txt - each row contains the index of the vertices, which forms a triangular mesh.

```
1.000000 2.000000 3.000000
4.000000 5.000000 6.000000
7.000000 8.000000 9.000000
10.000000 11.000000 12.000000
13.000000 14.000000 15.000000
16.000000 17.000000 18.000000
19.000000 20.000000 21.000000
22.000000 23.000000 24.000000
25.000000 26.000000 27.000000
28.000000 29.000000 30.000000
31.000000 32.000000 33.000000
34.000000 35.000000 36.000000
```

n01.txt - each row represents the norm vector of the triangles.

```
0.000000 1.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 -1.000000
0.000000 0.000000 -1.000000
0.000000 -1.000000 0.000000
0.000000 -1.000000 0.000000
0.000000 0.000000 1.000000
0.000000 0.000000 1.000000
-1.000000 0.000000 0.000000
-1.000000 0.000000 0.000000
```

```
1.000000 0.000000 0.000000
1.000000 0.000000 0.000000
```

Supporting Object

v07.txt

```
0.525000 1.059000 0.030300
0.525000 1.059000 0.300300
0.650000 1.059000 0.030300
0.650000 1.059000 0.030300
0.525000 1.059000 0.300300
0.650000 1.059000 0.300300
0.525000 0.659000 0.030300
0.525000 1.059000 0.030300
0.650000 0.659000 0.030300
0.650000 0.659000 0.030300
0.525000 1.059000 0.030300
0.650000 1.059000 0.030300
0.525000 0.659000 0.300300
0.525000 0.659000 0.030300
0.650000 0.659000 0.300300
0.650000 0.659000 0.300300
0.525000 0.659000 0.030300
0.650000 0.659000 0.030300
0.525000 1.059000 0.300300
0.525000 0.659000 0.300300
0.650000 1.059000 0.300300
0.650000 1.059000 0.300300
0.525000 0.659000 0.300300
0.650000 0.659000 0.300300
0.525000 0.659000 0.030300
0.525000 0.659000 0.300300
0.525000 1.059000 0.030300
0.525000 1.059000 0.030300
0.525000 0.659000 0.300300
0.525000 1.059000 0.300300
0.650000 0.659000 0.300300
0.650000 0.659000 0.030300
0.650000 1.059000 0.300300
0.650000 1.059000 0.300300
0.650000 0.659000 0.030300
0.650000 1.059000 0.030300
```

f07.txt

```
1.000000 2.000000 3.000000
```

```
4.000000 5.000000 6.000000
7.000000 8.000000 9.000000
10.000000 11.000000 12.000000
13.000000 14.000000 15.000000
16.000000 17.000000 18.000000
19.000000 20.000000 21.000000
22.000000 23.000000 24.000000
25.000000 26.000000 27.000000
28.000000 29.000000 30.000000
31.000000 32.000000 33.000000
34.000000 35.000000 36.000000
```

n07.txt

```
0.000000 1.000000 0.000000
0.000000 1.000000 0.000000
-0.000000 0.000000 -1.000000
-0.000000 0.000000 -1.000000
-0.000000 -1.000000 0.000000
-0.000000 -1.000000 0.000000
0.000000 0.000000 1.000000
0.000000 0.000000 1.000000
-1.000000 0.000000 0.000000
-1.000000 0.000000 0.000000
1.000000 -0.000000 0.000000
1.000000 -0.000000 0.000000
```

Appendix D

Preliminary Environment - Plane Selection

The selection of pushing and supporting planes for the environment considered during preliminary evaluation is explained in detail in the following sections. Figure D-1 shows the environment constructed from .stl file. The figure shows how each object's outer surface are represented by triangular meshes. The figure also includes the IDs assigned to each object present in the environment.

D-1 Pushing plane selection

Figure D-2, shows the position of the robot along with its norm R_{N1}. The figure also shows the norms of pushing object ($P_{N1}\ldots$) and the norms of supporting object ($S_{N1}\ldots$)). Since each face of the object is represented by two triangular meshes as shown in figure D-2, the objects have two norms on each face, one for each triangle. The supporting object 01 is represented by a number of triangular meshes (1348 meshes) and in order to keep the figure simple and clear the norms for those triangles are not shown here.

As explained in chapter 4 (section:Contact Points Structure), the angle between the robot norm (R_{N1}) and each norm of the pushing object is calculated using the following formula.

$$P_{ang} = atan2(norm(R_{N1} \times P_{N1}), (R_{N1}.P_{N1})) \tag{D-1}$$

The angle so obtained are tabulated below in table D-1. These angles represent the orientation of each face of the object with respect to the robot. From the computed angles, the norms whose angles which fall within the range of 135° - 180° are filtered. Absolute angles are considered here during the filtering process. For the case considered here, norms P_{N9} and P_{N10} will get selected. From the figure D-2, we can see that these norms belong to the face which is facing the robot. This way the pushing plane gets selected.

Selected triangular meshes forming pushing plane (Object: 04) - 9 and 10.

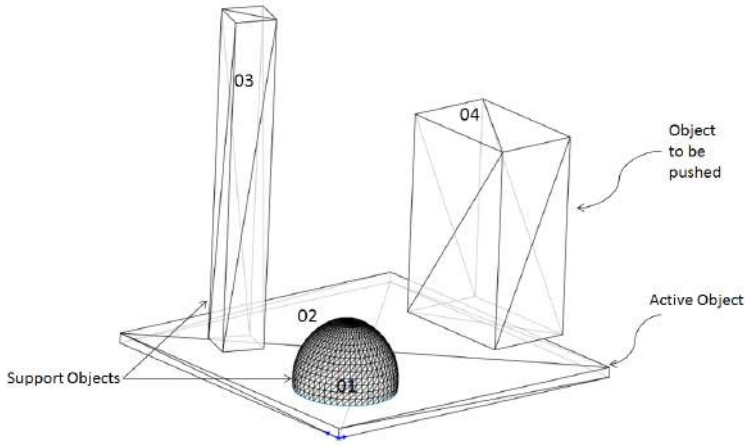


Figure D-1: Environment considered for preliminary evaluation constructed from stl file. Objects along with their respective IDs are shown.

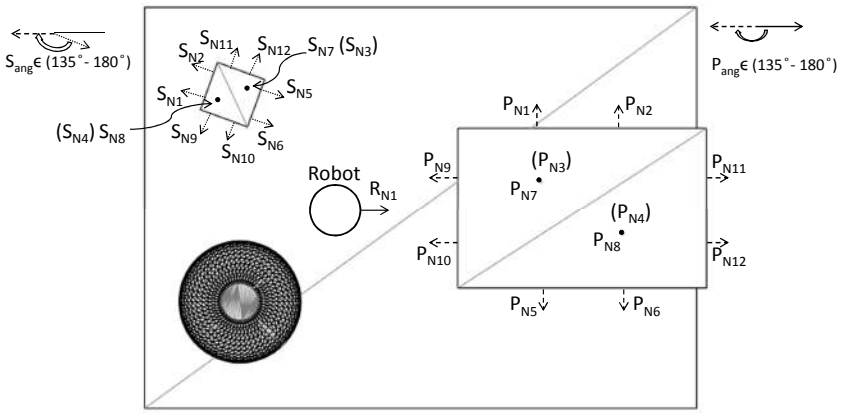


Figure D-2: Top view of the environment with norms for the robot (R_{N1}), pushing object ($P_{N1}\ldots$) and supporting object ($S_{N1}\ldots$)).

Table D-1: Pushing Plane angles

S.No	P_{ang}	Angle (in deg)
1	P_{N1}, P_{N2}	90, 90
2	P_{N3}, P_{N4}	90, 90
3	P_{N5}, P_{N6}	-90, -90
4	P_{N7}, P_{N8}	90, 90
5	P_{N9}, P_{N10}	-180, -180
6	P_{N11}, P_{N12}	0, 0

Table D-2: Supporting Plane angles

S.No	S_{ang}	Angle (in deg)
1	S_{N1}, S_{N2}	-20, -20
2	S_{N3}, S_{N4}	90, 90
3	S_{N5}, S_{N6}	160, 160
4	S_{N7}, S_{N8}	90, 90
5	S_{N9}, S_{N10}	70, 70
6	S_{N11}, S_{N12}	-110, -110

D-2 Supporting plane selection

For the selection of supporting planes, the angle between the norm of the selected pushing plane and the norms of the supporting object are computed. Since for the selected pushing plane both the norms are same, any one norm can be used for computing the angle (here P_{N9} is used). The supporting object 03 is taken for selecting the supporting planes. The angles are computed using D-1 as done for the P_{ang} case, shown in previous section. The computed angles are denoted by S_{ang} and they are tabulated in table. Again filtering the norms whose angles fall within the range (135° - 180°), the norms S_{N5} and S_{N6} will get selected. Since the selected norms of the triangular meshes are same, these meshes together form the supporting plane for that object.

Filtering based on plane orientation

The supporting object in general can be present on either side (wrt norms P_{N9} and P_{N10}) of the pushing object (04). If the supporting object 03 is present near the supporting object 01, then also the same plane will get selected as potential support plane for the object 03. But in such a case, the orientation of the supporting plane (P_{N9} and P_{N10}) will not be towards the pushing plane, as a result if the robot makes contact with that plane, the resulting reaction force will not aid the pushing motion of the robot. Such scenarios are handled by filtering the supporting plane based on its location. This is done as shown in figure D-3. Since the selected supporting plane is formed by 2 triangular meshes, two centroid location vectors CL_1 and CL_2 are constructed from S_{C5} and S_{C6} to any one of the centroids (since both the norms are same) of the selected pushing plane. The centroid considered here for the pushing plane is P_{C9}. With the vectors P_{N9} and CL_1, the cross product is computed, which is nothing but a vector perpendicular to both the vectors considered for cross product. This resulting

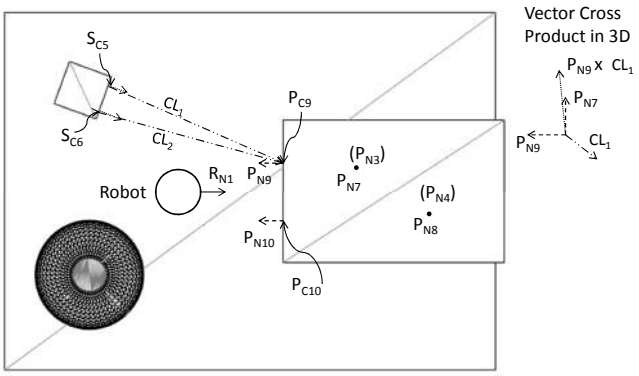


Figure D-3: Supporting plane filtering based on object location wrt pushing plane.(P_{C9} and P_{C10} - Pushing plane centroids, S_{C5} and S_{C6} - Supporting plane centroids and CL_1 and CL_2 are vectors connecting centroid between pushing and supporting plane.

vector (i.e $P_{N9} \times CL_1$) is subjected to dot product with a norm vector in the +ve z direction (here P_{N7} is taken). The sign of the resulting dot product is extracted and it is compared with the sign of the angles S_{ang} (angles corresponding to the selected norms) computed in the earlier section. If the signs are same then the planes are selected, if not they are filtered out. Same sign represents the fact that the orientation of the selected supporting planes are directed towards the pushing plane and hence will support the robot effectively during the pushing motion. The formula used for extracting the sign using the vectors P_{N9}, CL_1 and P_{N7} is given below. This is repeated for the other centroid location vector (i.e CL_2) as well. For the case considered here, the signs are same and the planes are selected.

$$sgn = sign(P_{N7}.(P_{N9} \times CL_1) \tag{D-2}$$

Object location based filtering

The supporting objects can be anywhere behind the robot and it can be in any orientation, it is necessary to filter out the objects which can effectively support the robot in the pushing direction. This is done by dividing the region behind the robot into five, with each region spanning 15°. Each region is set with some S_{ang} limit and any object which falls within that region and if it's orientation is within the stipulated limit, only then that object will be considered for support. The orientation limit (S_{ang}) for each region is tabulated below.

For the supporting object (03) considered here, the plane location wrt the robot can be determined as follows using the vectors R_{N1}, R_{N2}, RC_1 and RC_2 as follows.

$$R_{ang} = atan2(norm(R_{N2} \times RC_1), (R_{N2}.RC_1)); \tag{D-3}$$

Table D-3: Support plane Orientation limit

S.No	Region	Orientation limit
1	Region 1(0° - 15°)	175°
2	Region 2(15° - 30°)	165°
3	Region 3(30° - 45°)	150°
4	Region 4(45° - 60°)	135°
5	Region 5(60° - 75°)	135°

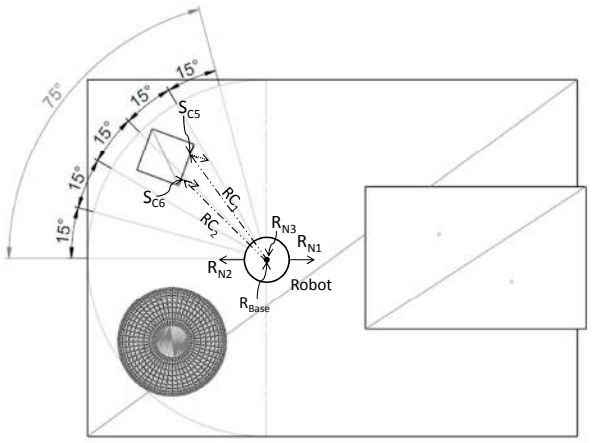


Figure D-4: Filtering supporting object based on its location wrt robot. The region behind the robot is split into 5, each spanning $15°$ (S_{C5} and S_{C6} - Supporting plane centroids,R_{N1}, R_{N2} and R_{N3} - robot norm vectors, R_{Base} - robot base position and RC_1 and RC_2 - robot base to supporting plane centroid vectors.

The location angle so obtained are $-46.43°$ and $-52.48°$ respectively, which puts the selected plane in region 3 and 4. The S_{ang} for the selected meshes were computed to be $160°$ and this is well within the orientation limit set for the respective regions. Similarly the supporting planes can be selected for supporting object 01 also.

Selected triangular meshes forming supporting plane (Object: 03) - 5 and 6.

Bibliography

[1] K. Harada, S. Kajita, K. Kaneko, and H. Hirukawa, "Pushing manipulation by humanoid considering two-kinds of zmps," in *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, vol. 2, pp. 1627–1632, IEEE, 2003.

[2] K. Harada, S. Kajita, F. Kanehiro, K. Fujiwara, K. Kaneko, K. Yokoi, and H. Hirukawa, "Real-time planning of humanoid robot's gait for force-controlled manipulation," *Mechatronics, IEEE/ASME Transactions on*, vol. 12, no. 1, pp. 53–62, 2007.

[3] T. Takubo, K. Inoue, and T. Arai, "Pushing an object considering the hand reflect forces by humanoid robot in dynamic walking," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 1706–1711, IEEE, 2005.

[4] T. Takubo, K. Inoue, K. Sakata, Y. Mae, and T. Arai, "Mobile manipulation of humanoid robots-control method for com position with external force," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 2, pp. 1180–1185, IEEE, 2004.

[5] M. Ikebe, T. Tsuji, Y. Sato, K. Tsuji, and K. Ohnishi, "Pushing motion of humanoid robot on dynamic locomotion," in *Industrial Technology, 2004. IEEE ICIT'04. 2004 IEEE International Conference on*, vol. 1, pp. 90–95, IEEE, 2004.

[6] K.-J. Choi, Y.-K. Hwang, D. S. Hong, W. J. Chung, and I.-H. Park, "Optimization of cooperative motion for humanoid robots using a genetic algorithm," in *ICMIT 2005: Control Systems and Robotics*, pp. 60423Q–60423Q, International Society for Optics and Photonics, 2005.

[7] I.-H. Park, K.-J. Choi, and D. S. Hong, "Optimization and generalization of whole-body cooperative motion of a humanoid robot using a genetic algorithm and neural networks," in *Control, Automation and Systems, 2007. ICCAS'07. International Conference on*, pp. 2699–2704, IEEE, 2007.

[8] Y. K. Hwang, K. J. Choi, and D. S. Hong, "Self-learning control of cooperative motion for a humanoid robot," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 475–480, IEEE, 2006.

[9] T. Takubo, K. Inoue, and T. Arai, "Pushing operation for humanoid robot using multipoint contact states," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pp. 1887–1892, IEEE, 2005.

[10] M. Stilman, K. Nishiwaki, and S. Kagami, "Learning object models for whole body manipulation," in *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pp. 174–179, IEEE, 2007.

[11] S. Nozawa, Y. Kakiuchi, K. Okada, and M. Inaba, "Controlling the planar motion of a heavy object by pushing with a humanoid robot using dual-arm force control," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1428–1435, IEEE, 2012.

[12] "Actionandreaction,http://dallaswinwin.com/newtonslaws/newton.html,"

[13] "The ergonomics of manual material handling - pushing and pulling tasks,"

[14] Q. Huang, K. Kaneko, K. Yokoi, S. Kajita, T. Kotoku, N. Koyachi, H. Arai, N. Imamura, K. Komoriya, and K. Tanie, "Balance control of a piped robot combining off-line pattern with real-time modification," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 4, pp. 3346–3352, IEEE, 2000.

[15] M. Gienger, K. Loffler, and F. Pfeiffer, "A biped robot that jogs," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 4, pp. 3334–3339, IEEE, 2000.

[16] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Yokoi, and H. Hirukawa, "A realtime pattern generator for biped walking," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 1, pp. 31–37, IEEE, 2002.

[17] F. Kanehiro, M. Inaba, and H. Inoue, "Development of a two-armed bipedal robot that can walk and carry objects," in *Intelligent Robots and Systems' 96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, vol. 1, pp. 23–28, IEEE, 1996.

[18] M. Inaba, T. Igarashi, S. Kagami, and H. Inoue, "A 35 dof humanoid that can coordinate arms and legs in standing up, reaching and grasping an object," in *Intelligent Robots and Systems' 96, IROS 96, Proceedings of the 1996 IEEE/RSJ International Conference on*, vol. 1, pp. 29–36, IEEE, 1996.

[19] M. Vukobratović and B. Borovac, "Zero-moment pointâĂŤthirty five years of its life," *International Journal of Humanoid Robotics*, vol. 1, no. 01, pp. 157–173, 2004.

[20] A. Goswami, "Postural stability of biped robots and the foot-rotation indicator (fri) point," *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 523–533, 1999.

[21] D. Hsu, J.-C. Latcombe, and S. Sorkin, "Placing a robot manipulator amid obstacles for optimized execution," in *Assembly and Task Planning, 1999.(ISATP'99) Proceedings of the 1999 IEEE International Symposium on*, pp. 280–285, IEEE, 1999.

[22] S. M. LaValle and J. J. Kuffner Jr, "Rapidly-exploring random trees: Progress and prospects," 2000.

[23] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, pp. 566–580, 1996.

[24] E. Yoshida, C. Esteves, O. Kanoun, M. Poirier, A. Mallet, J.-P. Laumond, and K. Yokoi, "Planning whole-body humanoid locomotion, reaching, and manipulation," in *Motion Planning for Humanoid Robots*, pp. 99–128, Springer, 2010.

[25] A. Dietrich, C. Ott, and A. Albu-Schaffer, "Multi-objective compliance control of redundant manipulators: Hierarchy, control, and stability," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 3043–3050, IEEE, 2013.

[26] O. Khatib, L. Sentis, J. Park, and J. Warren, "Whole-body dynamic behavior and control of human-like robots," *International Journal of Humanoid Robotics*, vol. 1, no. 01, pp. 29–43, 2004.

[27] Y. Nishihama, K. Inoue, T. Arai, and Y. Mae, "Mobile manipulation of humanoid robots-control method for accurate manipulation," in *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 2, pp. 1914–1919, IEEE, 2003.

[28] L. Sentis, *Synthesis and control of whole-body behaviors in humanoid systems.* PhD thesis, Citeseer, 2007.

[29] "Pushing action analysis,http://amberroseactionanalysis.weebly.com/push.html,"

[30] Y. Jun, A. Alspach, and P. Oh, "Controlling and maximizing humanoid robot pushing force through posture," in *Ubiquitous Robots and Ambient Intelligence (URAI), 2012 9th International Conference on*, pp. 158–162, IEEE, 2012.

[31] R. Featherstone, *Rigid body dynamics algorithms.* Springer, 2014.

[32] R. Featherstone, "A beginner's guide to 6-d vectors (part 1)," *Robotics & Automation Magazine, IEEE*, vol. 17, no. 3, pp. 83–94, 2010.

[33] M. Mistry, J. Buchli, and S. Schaal, "Inverse dynamics control of floating base systems using orthogonal decomposition," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 3406–3412, IEEE, 2010.

[34] Y. Yokokohji, S. Nomoto, and T. Yoshikawa, "Static evaluation of humanoid robot postures constrained to the surrounding environment through their limbs," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 2, pp. 1856–1863, IEEE, 2002.

[35] L. Sentis and O. Khatib, "Control of free-floating humanoid robots through task prioritization," in *IEEE International Conference on Robotics and Automation*, vol. 2, p. 1718, IEEE; 1999, 2005.

[36] L. Sentis and O. Khatib, "A whole-body control framework for humanoids operating in human environments," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 2641–2648, IEEE, 2006.

[37] M. Mistry, J. Nakanishi, G. Cheng, and S. Schaal, "Inverse kinematics with floating base and constraints for full body humanoid robot control," in *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, pp. 22–27, IEEE, 2008.

[38] M. Mistry, J. Nakanishi, and S. Schaal, "Task space control with prioritization for balance and locomotion," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pp. 331–338, IEEE, 2007.

[39] F. L. Moro, M. Gienger, A. Goswami, N. G. Tsagarakis, and D. G. Caldwell, "An attractor-based whole-body motion control (wbmc) system for humanoid robots,"

[40] K. Inoue, H. Yoshida, T. Arai, and Y. Mae, "Mobile manipulation of humanoids-real-time control based on manipulability and stability," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 3, pp. 2217–2222, IEEE, 2000.

[41] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics (TOG)*, vol. 31, no. 4, p. 43, 2012.

[42] M. J. Powell, "The bobyqa algorithm for bound constrained optimization without derivatives," 2009.

/

Glossary

List of Acronyms

OSC	Operational Space Controller
DOF	Degree of Freedom
URDF	Universal Robot Description Format
CoG	Center of Gravity
RBDL	Rigid Body Dynamics Library