

# Cooperative Load Sharing by Nao Humanoid Robots

D S V Santosh Kumar

A Thesis Submitted to  
Indian Institute of Technology Hyderabad  
In Partial Fulfillment of the Requirements for  
The Degree of Master of Technology



Department of Mechanical and Aerospace Engineering

July 2015

## Declaration

I declare that this written submission represents my ideas in my own words, and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.

D.S.V. Santosh Kumar

(Signature)

D.S.V. Santosh Kumar

(D S V Santosh Kumar)

ME13M1007.

(Roll No)

## Approval Sheet

This thesis entitled Cooperative Load Sharing by Nao Humanoid Robots by D S V Santosh Kumar is approved for the degree of Master of Technology from IIT Hyderabad.



Dr. S Sireesh

Department of Civil Engineering

External Examiner



Dr. Viswanath Chinthapenta

Department of Mechanical and Aerospace Engineering

Internal Examiner



Dr. R Prasanth Kumar

Department of Mechanical and Aerospace Engineering

Adviser



Dr. C P Vyasrayani

Department of Mechanical and Aerospace Engineering

Chairman

## **Acknowledgements**

I wish to express my sincere gratitude to my advisor Dr. R. Prasanth Kumar for his encouragement, inspiration and patience during this work. I also wish to thank the Mr. Janardhan Vistapalli, Mr. Ratikanth, Mr. Surya, Mr. Vikram, Mr. Nikhil, Mr. Raj Kiran who helped me through out the project to make it more effective and complete it. And I wish to thank Mr. Suman Dhara and Mr. Raj Kiran who constantly motivated me through out the work.



## **Abstract**

Load Sharing is one of the major tasks in Cooperative Transportation by robots. When the members of the team are biped robots the Cooperative Load Sharing becomes even more difficult to handle as when compared to the other type of mobile robots due to increased Degrees Of Freedom and low stability margin. To share the load first the robots should be sure of the total load, number of robots and the percentage of the load they have to share. After knowing the load they have to detect the amount of load being put on them, share the information with other robots and cooperatively adjust themselves so that the total load is shared equally. To learn the force interactions the robots requires special force sensors but these are costly and are not present in many robotic platforms. In the current project, an Algorithm is developed which uses the Torque Resistance by motors in the shoulder joints of the robot to learn the force interactions. For the experiment purpose two Nao Humanoid robots are being used in which the force sensors are not present to sense the force exerted on them while cooperatively lifting an object and sharing the load. At first one robot is checked for the limits of the amount of load it can detect and handle without any physical damage. After knowing the load for single robot, the algorithm is checked on one robot. After making the necessary corrections, the algorithm is modified to work for both the robots.

# Contents

Declaration . . . . .	ii
Approval Sheet . . . . .	iii
Acknowledgements . . . . .	iv
Abstract . . . . .	v
<b>Nomenclature</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 General . . . . .	1
1.1.1 Understanding the environment . . . . .	2
1.1.2 Starting the sequence . . . . .	2
1.1.3 Sharing the Information . . . . .	2
1.1.4 Taking decisions . . . . .	2
1.1.5 Cooperating and completing the task . . . . .	3
1.2 Research Motivation . . . . .	3
1.3 Research Methodology . . . . .	3
<b>2 Literature Review</b>	<b>5</b>
2.1 General . . . . .	5
2.2 Humans and the robots . . . . .	5
2.3 Robots and Robots . . . . .	6
2.4 Nao Humanoid Robot . . . . .	9
<b>3 NAO Humanoid robot</b>	<b>11</b>
3.1 General . . . . .	11
3.2 Software In and Out of Nao . . . . .	12
3.3 OpenNAO . . . . .	12
3.3.1 User Accounts . . . . .	13
3.4 NAOqi . . . . .	13
3.5 NAOqi Framework . . . . .	14

3.5.1	The NAOqi process . . . . .	14
3.5.2	Modules . . . . .	16
3.6	Tools for Programmers . . . . .	17
3.7	SDK(Software Development Kit) . . . . .	18
3.8	Present Work . . . . .	19
<b>4</b>	<b>Programming and Experimental Work</b>	<b>20</b>
4.1	General . . . . .	20
4.2	Robot Sensing Load . . . . .	21
4.3	Determining Load Limits . . . . .	22
4.3.1	Finding Sensitivity Values . . . . .	23
4.3.2	Finding limiting load Values . . . . .	23
4.4	Object Lifting by Single Robot . . . . .	24
4.5	Object Lifting by Two Robots . . . . .	33
<b>5</b>	<b>Results and Discussions</b>	<b>50</b>
5.1	Results . . . . .	50
5.1.1	Single Robot Lifting the Object . . . . .	50
5.1.2	Two Robots Lifting the Object . . . . .	53
5.2	Conclusions . . . . .	55
<b>6</b>	<b>Further Work</b>	<b>56</b>
	<b>References</b>	<b>57</b>

# Chapter 1

## Introduction

### 1.1 General

Cooperative task is the nothing but **“Performing a Task Together by a group of members cooperating with each other”**. The group can be a combination of humans and robots or robots alone. Cooperative task between humans is obvious thing but, between a team of robots is one of the major emerging topics. It increases the scope of the usage of robots which can lead to the complete automation of the work. Not only in improving the level of automation, but also reduces the time for completion and also the flexibility of the place to accommodate more work capacity or different types of works.

The cooperative tasks are in general carried in two ways

- One of the group of the robot acts as the master robot and all the other robots acts as It's slaves. The main program that contains the code which controls the flow of the operations is programmed to this master robot and the remaining programs that contains the codes to perform certain tasks will be programmed to all the other robots including the master robot if it is also required to perform some tasks. The master robot controls the other robots as task scheduling is done by it.
- There will be a Central Server System to which the main program that contains the code which controls the flow of the operations is programmed and a group of robots to which and the remaining programs that contains the codes to perform certain tasks will be programmed. The Central Server System robot controls the other robots as task scheduling is done by it.

In general any cooperative task consists of 5 steps they are

1. understanding the environment.
2. Starting the sequence of actions.
3. Sharing the Information.
4. Taking the decisions and accordingly controlling the flow of actions of the members of the group.
5. Cooperating each other in performing and completion of the task.

#### **1.1.1 Understanding the environment**

If the robot involves motion, first robot needs to understand the environment around it. Where is it in the place? Where it has to move? What is the path? Are there any obstacles in the path? which object it has to detect? where are the walls? .. etc.

#### **1.1.2 Starting the sequence**

The robot must have programmed with a sequence of operations that it has to perform after identifying the environment. It has to start performing the sequence of tasks it is ordered to perform. The first task to start with may be to reach to a particular destination and start performing the remaining sequence of tasks there or first perform a task in the current place then move to other place.

#### **1.1.3 Sharing the Information**

The robots has to share information of their progress frequently. This is done through the communication protocol. This is one of the basic part in cooperative task. Until they communicate with each other cooperative task is impossible. For example if a group of robots are exploring an area, one robot should not go to a place which is already explored by one of the other robots. To avoid such situations, if one robot explores one area that should be updated to all the other robots so that other robots carry on with exploring the other places leaving the already explored places.

#### **1.1.4 Taking decisions**

In dynamic operations based on the situations the robots has to choose the sequence of operations they have to select to complete the task early and efficiently. The decision making is generally associated with the Master robot or the Central Server System that controls the group of robots.

### 1.1.5 Cooperating and completing the task

The robots have to repeat the above tasks, sharing the information so that Master robot or the Central Server System can decide how they share their part of work and what part of work is allocated to the group of robots to complete the task.

## 1.2 Research Motivation

In recent years, the Advancement and the scope of application of robotics is increasing quite fast that they are even made possible to use for domestic purpose and to perform a large number of operations without human involvement. The most striking and successful development in this context is the Humanoid robots like Honda Asimo, Nao, HRP etc.

Although the humanoid robots can be made usable for the domestic purpose, they are mainly used for the entertainment. Even though they can be used for helping humans, it requires cooperation from humans, that if the robot is needed to carry an object the object has to be handed over to the robot and take from it by humans but the robot can lift the heavier loads by itself balancing them and carry them to the required destination.

## 1.3 Research Methodology

The scope of this work is make two robots and dead load while cooperatively share the load and balancing it. Here the robot has no capability to understand the environment, it can be done by integrating the robot with other equipment over the network. So the focus has been kept on the communication and cooperation between the robots while carrying out the task of lifting the object sharing the load. This is carried out in 4 stages.

The **First** stage is to find out the way to make the robot detect the load because to lift the load or share the load the robot first needs to detect the load but Nao robots don't have the force sensors to detect the load or learn the force interaction

The **Second** stage to find out the limits of the amount of load the robot can handle because to detect low loads the sensitivity factor has to be too low then the robot may detect the load even with small perturbation without the actual load. Similarly to detect high load high sensitivity factor is required and the robot motors may get

damaged before reaching that high sensitivity factor

The **Third** stage is to develop a program to make a single robot lift load while sharing the load to it's both hands, balancing the weight. In this stage the load is used limits found out in the second stage. The program is also accordingly to be modified to make the robot carry out the task smoothly.

The **Fourth** stage is to develop a program to make two robots lift the load co-operatively by sharing the information of the part of the load they are handling and according move their hands to share the total load and lift it. And this program is also to be modified to make the robots carry out the task smoothly.

## Chapter 2

# Literature Review

### 2.1 General

A literature review was undertaken to accumulate information regarding the recent advancements on the various cooperative concepts needed to direct this study. The literature survey is done on the cooperative tasks that are carried out between the human and robots and between the robots. This literature survey is divided into 3 parts one is between the humans and the robots. The other is between the robots and last one is involving the Nao humanoid robot.

### 2.2 Humans and the robots

There is lot of research is carried out on the cooperative tasks that involves the humans and robots as partners.

One of such research work one is “*Cooperative Tasks between Humans and Robots in Industrial Environments*” [1]. In this work authors have performed in recent years in order to develop a human-robot interaction system which guarantees human safety by precisely tracking the complete body of the human and by activating safety strategies when the distance between them is too small. To do this they have used a motion capture system which is based on the inertial motion system to precisely track the human operators. They have proposed a fusion algorithm to learn information from the motion sensors and accordingly manipulate the links of the robotic manipulator arm to ensure the safety of the human operators around it without effecting it’s operations.



*“Load Sharing in Human-Robot Cooperative Manipulation”* [2]. In this research work the authors focused on the manipulation of bulky objects in narrow environments using the cooperation between human and a robot and Actuation redundancies arising in joint manipulation while load sharing among the interacting partners. They developed the effort sharing policies which are systematically derived from the geometric and dynamic task properties. Three policies are intuitively identified, resulting in unilateral and balanced effort distributions. These policies are evaluated within a novel hierarchical motion generation and control framework.

*“Cooperative Human Robot Interaction System: An Architecture for Learning and Executing Actions and Shared Plans”* [3]. In this research, the authors addressed the challenge of an important aspect of the robot behavior is the ability to acquire new knowledge of the cooperative tasks by observing and interacting with humans. They demonstrated a system which is capable to learn to recognize objects and to recognize actions as sequences of perceptual primitives, and to transfer this learning, and recognition, between different robotic platforms. The System is provided the ability to link actions into shared plans, that form the basis of human–robot cooperation, applying principles from human cognitive development to the domain of robot cognitive systems.

*“Physical Interaction Learning: Behavior Adaptation in Cooperative Human-Robot Tasks Involving Physical Contact”* [4]. In this research the authors addressed the problem of robots capabilities to adapt their behaviour in order to facilitate the interaction with a human partner. This is achieved using machine learning techniques. The learning is achieved for tightly coupled, physical touch interactions between the learning agent and the human partner. They presented a human in-the-loop learning scenarios and proposed a computationally cheap learning algorithm to achieve the goal.

## 2.3 Robots and Robots

There are two ways in which the robots can interact while performing the cooperative tasks. Either there will be a master robot or the central server system to control the group of robots. All task scheduling, communications or the mathematical and logical computations are carried out in this master robot or the central server system. there is also lots of research carried out in this sector they are presented below.

*“Multi-Robot Learning in a Cooperative observation task”* [5]. In this research the authors worked for the development of mechanisms that enable robot teams to autonomously generate cooperative behaviors. First they presented the Cooperative Multi-robot Observation of Multiple Moving Targets (CMOMMT) application as a rich domain for studying the issues of multi-robot learning of new behaviors. They have hand generated an algorithm for the purpose of CMOMMT. Finally they develop techniques for multi-robot learning and adaptation that will generalize to cooperative robot applications in many domains, thus facilitating the practical use of multi-robot teams in a wide variety of real-world applications.

*“Cooperative Object Transportation by Multiple Humanoid Robots”* [6]. In this research aims to create a framework of transporting an object by multiple humanoid robots. In this work, a leader-follower type cooperation method is used. In the leader-follower method, the leader robot is directly operated by a human operator, and the follower robot is programmed to follow the leader robot based on the force sensor information measured on its hands. This force is due to relative displacement of the robots and could induce unstable condition. They solved the problem falling down of the robots due to force in the worst case by proposing a system of easing the interactive force while moving in this paper. They generated a pattern for the follower robot by using the preview control of ZMP theory.

*“Evolving Cooperative Control on Sparsely Distributed Tasks for UAV Teams Without Global Communication”* [7]. In this research the authors investigated a controller design using multi-objective genetic programming for a multi-robot system to solve a highly constrained problem, where multiple unmanned aerial vehicles (UAVs) must monitor targets spread sparsely throughout a large area. In research the authors tackled different problems like UAVs having a small communication range, limited and noisy sensor information, UAVs taking an indefinite amount of time monitoring a target, and they evolved controllers that continue to perform well even as the number of UAVs and targets changes and also evolved task selection controller that dynamically chooses a target for the UAV based on sensor information and communication. They also developed controllers using several communication schemes comparing on problem scenarios of varying size, and suggested that their approach can evolve effective controllers if communication is limited to the nearest other UAV.

*“Cooperative Sweeping by Multiple Mobile Robots”* [8]. In this research the authors proposed an off-line planning algorithm for cooperative sweeping task of multiple

mobile robots . In this research Sweeping described as a motion that a robot covers a 2-dimensional area by its effector. Sweeping of a whole work area is fundamental and essential task of mobile robots in this research. For efficient cooperation, they set an appropriate burden onto each robot to handle the problem of interference of robots and overlaps of their effectors making the efficiency low. The cost of sweeping depends on both sweeping ability of a robot and a shape of a work area to be swept. Evaluation and distribution of the cost are most important issues to deal with in this research. Here they proposed an algorithm in which the cost is evaluated by means of length on which robot should move. They introduce both edges of the configuration space and Voronoi diagram so as to compute paths in the whole area. They generated the a tour for traversing all the paths by applying the algorithm of the Chinese Postman Problem., they assigned appropriate paths of the tour to each robot according to the cost evaluation.

*“Cooperative Transport by Multiple Mobile Robots in Unknown Static Environments Associated With Real-Time Task Assignment”* [9]. In this work the authors dealt with a task assignment architecture for cooperative transport by multiple mobile robots in an unknown static environment. They developed and architecture that satisfies three features: deal with a variety of tasks in time and space, deal with a large number of tasks compared with the number of robots, and decide behavior in real time. The authors proposed the following approach: consider the unit of task (task instance) as the job that should be done in a short time by one robot. Based on environmental information, task instances are dynamically generated using task templates. The priority of task instances is evaluated dynamically based on the number of robots and the configuration in the workspace. In addition, they avoided generating too many task instances by suppressing object motion. The main part of the architecture consists of two real-time planners: a priority-based task-assignment planner solved by using a linear programming method, and motion planners based on short-time estimation.

*“Cooperative Self-Organization in a Heterogeneous Swarm Robotic System”* [10]. in this research the authors focused on how a swarm robotic system consisting of two different types of robots can solve a foraging task. They considered two types of robotic systems for their research. The first type of robots are small wheeled robots, called foot-bots, and the second type are flying robots that can attach to the ceiling, called eye-bots. While the foot-bots perform the actual foraging, that is they move back and forth between a source and a target location, the eye-bots are deployed in

stationary positions against the ceiling, with the goal of guiding the foot-bots. The key component of their approach is a process of mutual adaptation, in which foot bots execute instructions given by eye-bots, and eye-bots observe the behavior of foot-bots to adapt the instructions they give. Through a simulation study, they showed that this process allows the system to find a path for aging in a cluttered environment. They are also able to converge onto the shorter of two paths, and spread over different paths in case of congestion.

*“Object Transportation by Two Humanoid Robots using Cooperative Learning”* [11]. In this research they proposed an approach to the behavior acquisition required for humanoid robots to learn a cooperative transportation task. They tackled the problem of mutual position shifts due to the body swinging of robots that occur in the case of object transportation with two humanoid robots by correcting the position in a real-time manner. They put efforts to develop the position shift correction system. they propose to solve the problem by learning required behaviors with two learning algorithms. they Successfully demonstrated cooperation of two HOAP-1 humanoid robots in the transportation task obtained by Classifier System and Q-learning

## 2.4 Nao Humanoid Robot

*“Using Human Motion Estimation for Human-Robot Cooperative Manipulation”* [12]. In this research the authors developed an algorithm for Nao Humanoid robot to cooperatively assist a human to manipulate the objects by taking the human movements as inputs and follow them or lead them. To take the inputs from the users they use the visual capabilities of the robot that observe a series of led lights attached to the object. They proposes a framework to endow robots with a human capability of developing a mutual understanding while performing the collaborative task. In this research Behavior of the robot is controlled by two types of controllers such as reactive and proactive controllers. The reactive controller causes the robot to behave as a follower and the proactive controller causes it to behave as the leader. The proactive controller suggests proactive actions based on human motion prediction. The framework relies on a novel technique to compute a measure of confidence for the prediction. This confidence measure determines the leader/follower role of the robot. Hence, the robot can switch roles during the task autonomously and dynamically.

*“Inferring Guidance Information in Cooperative Human-Robot Tasks”* [13]. In this research the authors focused on human guiding the robot by exerting forces,

either through direct physical interaction or indirectly via a jointly manipulated object. They tried to compensate the physical forces that perturb the robot's behavior execution. Typically, this problem is tackled by means of special purpose force sensors which are, however, not available on Nao robotic platform. In contrast, they proposed a machine learning approach based on sensor data, such as accelerometer and pressure sensor information. In the training phase, a statistical model of behavior execution is learned that combines Gaussian Process Regression with a novel periodic kernel. During behavior execution, they continuously compared the predictions from the statistical model with stability parameters derived from current sensor readings. Differences between predicted and measured values exceeding the variance of the statistical model are interpreted as guidance information and used to adapt the robot's behavior.

## Chapter 3

# NAO Humanoid robot

### 3.1 General

The present research is done entirely on the Nao Humanoid robot [14]. The specifications and architecture of the Nao robot has a strong affect on the implementation of the proposed algorithm. The capabilities and limits of the robot played a crucial role in development of the programs out of the algorithm. So this chapter dedicated to the explanation of Nao humanoid robot.

- **Nao** (pronounced *now*) is an autonomous, programmable humanoid robot developed by Aldebaran Robotics, a French Robotics company
- Nao operates on OS called **OpenNAO** developed in Linux.
- OpenNAO runs a number of software out of them one called **NAOqi** is the most important one which consists of different all the predefined Modules and Methods which can only be used in programming to control Nao's hardware.
- The Programs for Nao can be developed in C++, Python, .NET, JAVA, Matlab, Urbi. The Programs thus developed must be associated with **SDK**(Software Development Kit) software that comes with the robot which does the cross compilation.
- Along with SDK other software also comes Nao which are **Choreographer**, which is another tool to do visual program development for Nao. **Monitor**, to visualize the sensor values live from the Nao while in operation. **Webots**, where the developed programs can be tested on a simulated robot in simulated environment and **Nao Flasher** which can be use to update the Nao.

## 3.2 Software In and Out of Nao

Nao comes with two types of softwares.

- **Embedded software**, which is running on the motherboard located in the head of the robot, allowing autonomous behaviors.
- **Desktop software**, which will run on your computer, allowing creation of new behaviors, programs and the remote control of the robot.

Open Nao and NAOqi are embedded softwares where SDK, Choregraphe, Monitor, Nao Flasher and Webots are the Desktop Softwares. Naoqi can also be installed on one's computer so that one can test the programs developed in programming languages in association with SDK on a simulated robot before running them on the actual robot.

## 3.3 OpenNAO

OpenNAO is a the operating system of the Robot installed in it's motherboard. It is a GNU/Linux distribution based on Gentoo. It's an embedded GNU/Linux distribution specifically developed to fit the Nao robot needs. OpenNAO provides numbers of programs and libraries, among these, all the required one by NAOqi, which is the main software on the robot that provides access to the hardware of the robot.

OpenNAO OS provides and runs numerous programs on the robot, which are mentioned in Table 3.1

Program	Description
Connman	the network manager
NAOqi	the software that allows the robot to move, and so on.
htop	monitor process activity
iftop	monitor network activity
ldd	list library dependencies
gdb-remote	start a remote gdb server

Table 3.1: Softwares in OpenNAO

All this softwares run inside OpenNAO for the internal purpose. Among all these NAOqi is the most important one to the programmers, which can be used outside the robot.

OpenNAO also provides Text editors. the available text editors in OpenNAO are *nano* *gemacs* (alias as *e* and *emacs*) and *vim* (alias as *vi*)

### 3.3.1 User Accounts

Unlike any operating systems OpenNAO also provides user accounts to log on into it. The main user is *nao*, and like any GNU/Linux system, there is the super-user *root*. By default, passwords are user names. So, changing user to *root* using the *su* command will request the password *root*. Logging in as *root* over ssh is disabled. However the *su* command remains available.

## 3.4 NAOqi

NAOqi is the name of the main software that runs on the robot and controls it.

- It runs on the robot, under OpenNAO distribution.
- It can also run on one's computer in order to test the developed code on a simulated robot.

NAOqi is automatically launched while switching on the robot as the it consists modules and methods that control the robot, if the NAOqi doesn't start at starting of the robot, the robot can't be operated also the robot has check for some parameters like temperature of the joints, battery level etc which is not possible without NAOqi.

The script `"/etc/init.d/naoqi"` manages NAOqi's start at starting of the robot

Starting of the NAOqi means module has been successfully cross-compiled and copied on the robot that is the modules that are already present on the robot will load into NAOqi which are actually stored in the memory of the robot and maintained by OpenNAO

If any module is there which is not loaded by default in NAOqi, it has to be made to load at the beginning of NAOqi manually. To do so, the *autoload.ini* file must be modified which consists of the details of the libraries, modules and methods to be loaded into the NAOqi at startup.

If a custom module that is developed by a programmer requires other libraries, first it should be made sure that they are properly cross-compiled and copied to the robot. Then, to allow naoqi to find these libraries, the library path on the robot should be changed to the path of these current libraries.



## 3.5 NAOqi Framework

The NAOqi Framework is the programming framework used to program NAO.

It answers to common robotics needs including: parallelism, resources, synchronization, events.

The framework developed to have the following specialties as

- cross-platform, which means that it is possible to develop with it on Windows, Linux or Mac.
- cross-language, with an identical API for both C++ and Python. That is the methods advertised by it can be directly called in any programming language without much change in the syntax of the method.

### 3.5.1 The NAOqi process

The NAOqi executable which runs on the robot is a broker. When it starts, it loads a preferences file called *autoload.ini* that defines which libraries it should load. Each library contains one or more modules that use the broker to advertise their methods.

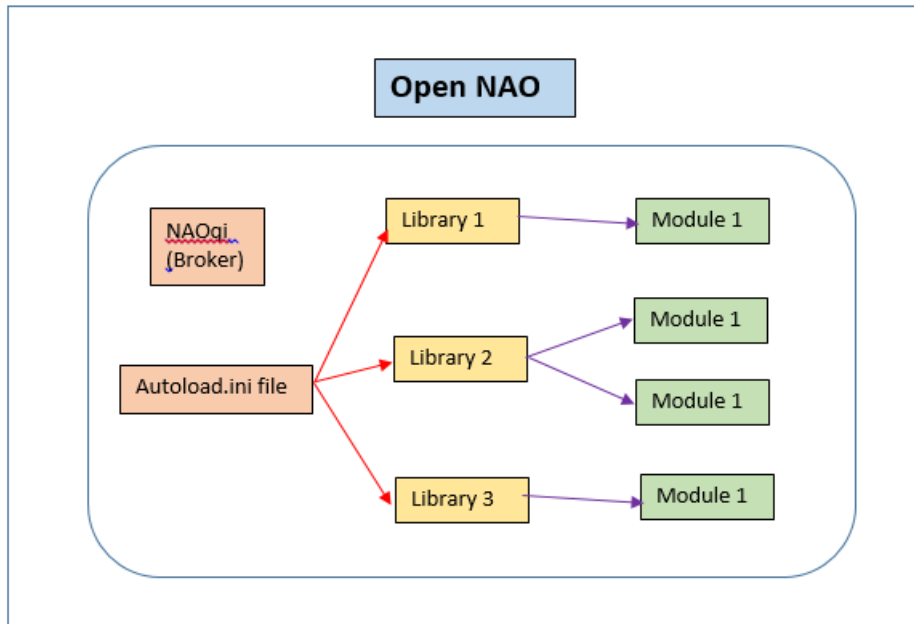


Figure 3.1: NAOqi Structure inside OpenNAO

### 3.5.1.1 Broker

A broker is an object that provides two main roles:

- It provides directory services: Allowing you to find modules and methods.
- It provides network access: Allowing the methods of attached modules to be called from outside the process.

The broker provides lookup services so that any module in the tree or across the network can find any method that has been advertised.

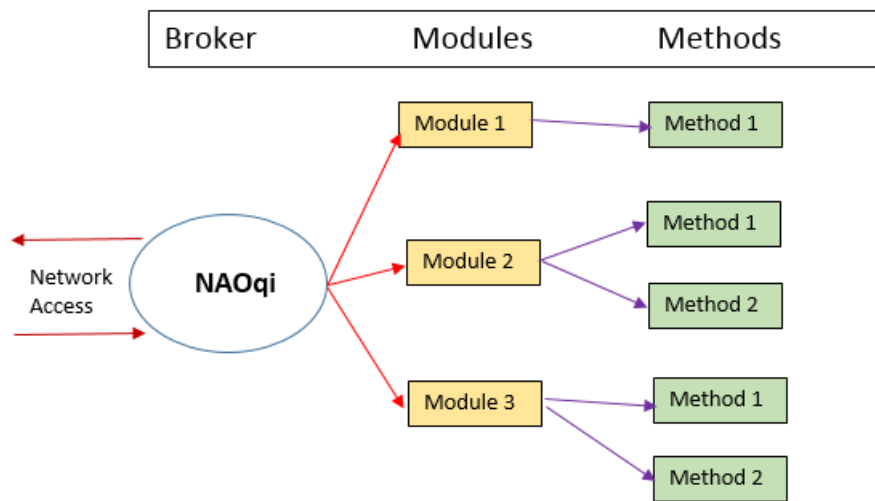


Figure 3.2: Modules and Methods inside a Broker

### 3.5.1.2 Proxy

A proxy is an "Object" that will behave as the module it represents.

If in a program there is a need to use the motion of arms of the robot. Then a Proxy has to be created to a module that contains the method to control the arms of the robot. By creating a proxy to a module, any methods inside that module can be called into the program. For example, if you create a proxy to the Motion module, you will get an object containing all the Motion methods.

To create a proxy to a module, (and thus calling the methods of a module) there are two ways:

- Simply use the name of the module. In this case, the program that is using the module and the module to which the program calling the methods must be in the same broker. Then that is called a *Local call*.
- If the module and the program that is calling the methods from the module are not in the same broker then methods of such module can be called by using the name of the module, and the IP and port of a broker which contains the module. This type calling is called *Remote call*.

### 3.5.2 Modules

Typically each Module is a "Class" within a library. When the library is loaded from the *autoload.ini* (which manages the starting of OpenNAO). It will automatically instantiate the module class. When a module class is instantiated the methods in it are also instantiated.

The methods are not simply implemented inside the module they are added to the module through a function called "bind". Then only the module can advertises the names and signatures of the method to the broker so that they become available to others. The new modules can also be developed implanting a set of it's own methods or a mixture of it's own methods which calls the methods that are already defined in the modules present in the broker of the robot or completely using only the methods of modules of the robot.

A module can be either remote or local. The module can be called as remote or local relative to other modules of a robot it is using.

- A *Remote Module* is module which is compiled as an executable file on a computer or outside the robot in the broker of another robot , and can be run outside the robot. Remote modules are easier to use and can be debugged easily from the outside, but are less efficient in terms of speed and memory usage since they are compiled outside to use the methods of the modules inside a broker they have to communicate to the module through the broker with a proxy to the module.
- A *Local Module* is a module which is present in the same broker as the module which methods it is using.it is compiled as a library, and can only be used on the robot as both the modules are present in the same broker and one broker is dedicated to one broker. However, they are more efficient than a remote module. Each module contains various methods. Among them, some methods are bound, which means they can be called from outside the module, for example inside

another module, from an executable etc. The way to call these bound functions does not vary if the module is remote or local: the module automatically adapts.

#### **3.5.2.1 Local modules**

Local modules are two (or more) modules launched in the same process. They speak to each other using only ONE broker.

Since local modules are in the same process, they can share variables and call each other's methods without serialization nor networking. This allow the fastest possible communication between them.

If there is a need to do some close loop (enslavement for example), only the local modules must be used.

#### **3.5.2.2 Remote modules**

Remote modules are modules which communicate using the network. A remote module needs a broker to speak to other modules. The broker is responsible for all the networking part. It must be known that remote modules work using SOAP over the network. One cannot do fast access using remote module (direct memory access for example).

#### **3.5.2.3 Connection between remote modules**

Remote modules can speak with other modules by connecting their brokers to other module's brokers using a proxy.

- A connection Broker to Broker opens a mutual communication. Modules from both brokers can talk to each others.
- A Proxy to Broker connection opens a single way of communication. The proxy can access to all modules registered to the broker BUT the modules registered to the broker cannot access to the module that owns the proxy.

## **3.6 Tools for Programmers**

There are two different tools available for the programs to program Nao they are

- SDKs (C++, Python, Mat Lab, JAVA. .NET, Urbi)
- Choregraphe software.

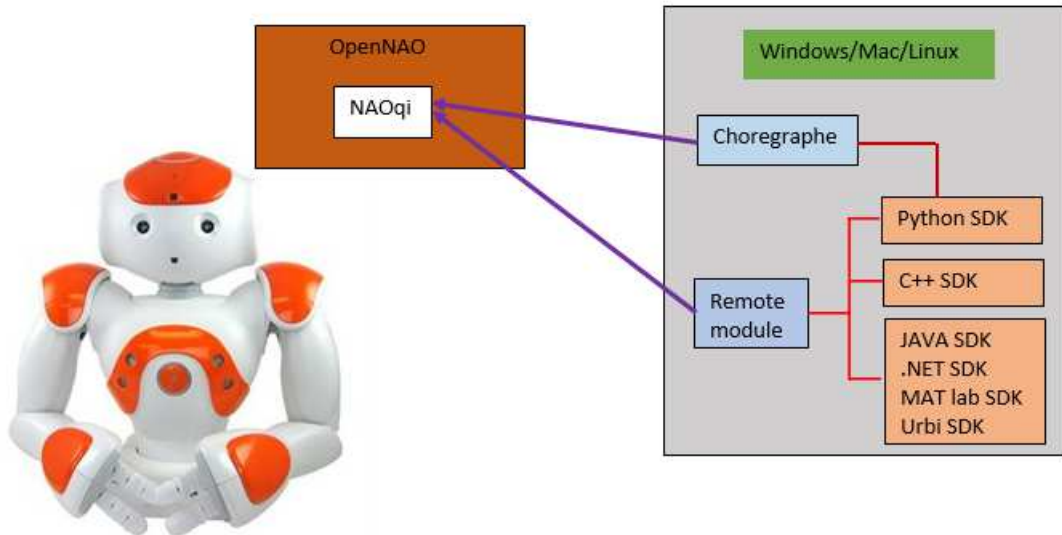


Figure 3.3: Programming NAO

Depending on the chosen language, using SDK one can:

- create code to enrich Choregraphe box library (Python),
- create a code remotely controlling the robot (all SDKs),
- create new NAOqi modules and upload them on the robot (C++, Python).

### 3.7 SDK(Software Development Kit)

SDK (Software Development Kit) comes with Nao it is a desktop software that one needs to install on their system. It consists of all the libraries and the modules that are in a broker so that there will be no problem in linking the object files while compiling the programs. There are different types of SDKs depending on Programming Language one choose to develop with. The main duty of a SDK is to cross compile the program that is written in a different Language from that of NAOqi and in different OS that is different from OpenNAO. It make the program developed remotely in different OS and language compatible to communicate with the Broker.

To create new NAOqi modules, the programs must be developed in C++ or Python Modules since all the Behaviors in a broker in the robot are are developed in Python and it's services in C++.

The sample program developed with C++ SDK looks like

### Sample Program:

```
#include <alproxies/altexttospeechproxy.h>
int main()
{
    AL::ALTextToSpeechProxy tts("<IP of your robot>", 9559);
    tts.say("Hello world from c plus plus");
    return 0;
}
```

In the above example *alproxies* is a library, *ALTextToSpeech* is a audio module, *tts* is the proxy (object) to that module and *say("string variable")* is the method of the module *ALTextToSpeech*

## 3.8 Present Work

In the present work, the programs are developed, calling the methods from different modules from brokers of the two robots in a sequence as per the proposed algorithm and are compiled as Remote modules in the computer in Linux and are executed as Remote Executables on the Robot from the computer over the WI-Fi network. These modules are developed using the C++ SDK.

## Chapter 4

# Programming and Experimental Work

### 4.1 General

As described in the introduction out of the Five steps the first step is neglected as the robot is not moving from one place to other. so the present cooperative task is starts from the second step.

The main objective of this works is to make two Na Humanoid robots lift an object while sharing load and balance the object. To achieve the target an algorithm is developed.

But to share the load the robot has to be sensitive to loads or the external forces so that it can detect the load and after detecting the load or force the robot can share the information of the load it detected by communicating trough the network.

To detect the load or the external forces, the robots usually consists of the force sensors. but the Nao humanoid robots lacks the force sensors. This makes the first challenge is to make the robot detect the load.

After making the robot to detect the load there is a need to know the limits of the load that the robot can handle.as if the load of the object is too heavy for the robot the motors in the joints of the robot get damaged or if the load of the object is too low the robot may not detect the load at all

After finding out load limits and the way to make robot detect the load. an algorithm is developed for single robot and after making the necessary corrections the algorithm is accordingly modified and tested on two robots.

## 4.2 Robot Sensing Load

As the robot lacks the force sensors to detect the load, the only way to make the robot sensitive to loads is using the motors in joints of the robot.

Although the entire hardware of the robot is controlled by NAOqi frame work it can't directly control the motors of the robot as it is high level language and motors work at the machine level language which is binary.

Hence NAOqi uses another sub software under it called as DCM(Device Communication Language).This DCM software is installed on the chipboards that are located at each and every motor in the robot. while commands to control the motors of the robot to perform certain tasks like lifting an hand, walking etc comes from NAOqi from the motherboard located in the head of the robot, these commands are not directly sent to the motors. these commands are received by the DCM in the electronic boards and converts them to machine level language to control the amount of the current to the motor through the electronic board from the battery.

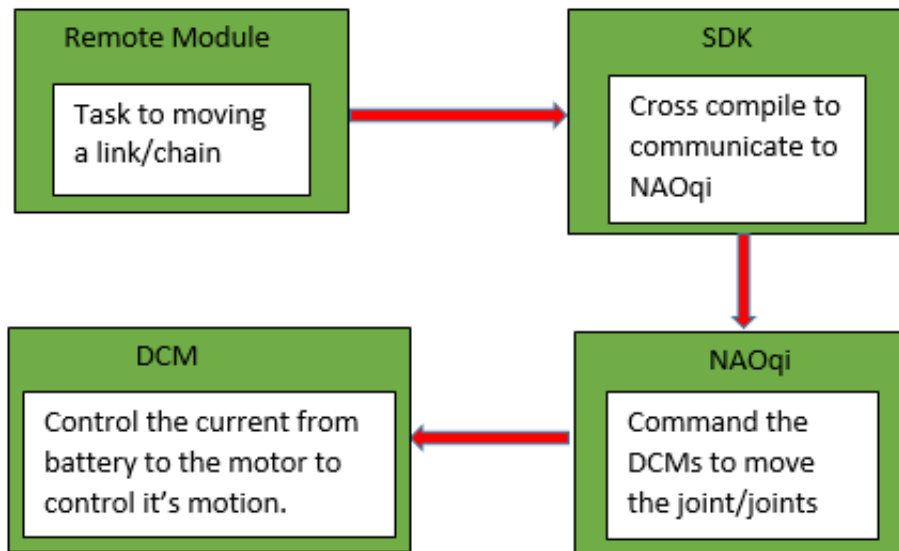


Figure 4.1: flow of commands



Hence the current to the motor has to be controlled to make the robot sensitive to the loads. to do so there is a feature in NAOqi called **smart stiffness** which controls to the current to the motors by reducing the current while they are in idle condition and increasing the current to the motors while there is command from NAOqi to DCM to move the joints.so that battery power can be optimized. Not only increasing and decreasing current based on the work and idle conditions it also increases the current to the joint to resist the movement of the joint when there are commands from NAOqi to DCM to move the joints to prevent the unexpected movements of the links of the robots. It is mainly developed to prevent robot from falling.

so the due smart stiffness feature the the motors resist the movement due to external loads when not ordered to move by increasing their Torque resistance. The torque values of all the joints of the robots are updated for every 10 milliseconds by every DCM in the memory of the robot.

Hence the difference in the torque values in the memory of the robot due to change in the torque resistance of the motors while an external load is applied can be used as the sensitivity values for the load detection by the robot. the next stage is to determine change in the difference values to find out the limits of the load that can be detected and handled by the robot.

### 4.3 Determining Load Limits

The torque values that are being written in the memory of the robot by the DCM are generally relative value starting from 0 to 1. Where 0 is "*No torque*" condition and 1 is "*Maximum Torque*" condition. these values are termed as hardness values. so the difference values are also relative and are decimal values.

Only the hardness differences of the shoulder joints are taken into account as including the other joint hardness difference values are giving a complicate and inaccurate results.

To determine the load limits, first the limits of the sensitivity values has to be determined. subsequently the limits of the load can be found out by checking where the load crosses the sensitivity value.

### 4.3.1 Finding Sensitivity Values

To find the sensitivity value limits an algorithm is developed that first reads the stiffness values from the memory of the robot for once after lifting it's hands to particular height and then runs a continuous loop where it again continuously reads the stiffness values from the memory of the robot and continuously calculates the difference between the values that are previously read and just read in the loop for every particular period of time of 2 seconds and prints the difference for every loop. while algorithm is running, an external force is applied manually on the hands of the robot gradually and sensitivity values are recorded from continuous loop where there is a significant change in the values at less force and where the robot becomes unstable at high forces and are taken as the limiting sensitivity values for the robot.

#### Algorithm:

```
lift both the hands to specific height
get the hardness values
    while ( continuous loop)
    {
        again get the values and calculate
        the difference values and print them
    }
```

### 4.3.2 Finding limiting load Values

To find the limiting load values the same algorithm used for finding sensitivity values is used. but now, the sensitivity values found out in the previous case are used as limits conditions in the continuous loop to find the load values

An experimental setup is made where the loads are continuously added onto hands of the robot and the algorithm is slightly modified to make the robot use it's audio module to inform when one of the limiting load values is reached.

The algorithm is modified as

#### Algorithm:

```
lift both the hands to specific height
```

```

get the hardness values
while ( continuous loop)
{
    again get the values and calculate
    the difference
    if ( required difference detected is in limit for both hands)
        { stop the loop }
    otherwise { inform the state and continue the while loop }
}

```

The results obtained are

Number of hands	Sensitivity value	Load value
Single hand	0.005	250 grams
	0.01	400 grams
Both hands	0.005	450 grams
	0.01	600 grams

Table 4.1: Load and Sensitivity Values

## 4.4 Object Lifting by Single Robot

As only the shoulder pitch torque values are used and only one robot is lifting the object, there will be two hardness difference values to be taken care of.

Now the robot has to lift the object by itself. so there should be continuous movement of the hands until both hands of the robot detect the load of the object.so for the continuous movement again a continuous loop is used. the robot has to lift the hands by small amount check the difference and if the load of the object is detected the loop has to stop and if load is not detected the loop has to continue

There is need to check to difference of the hardness values after every small movement of the hands. Here the difference can be calculated in two ways that is between the vales after every small movement and

- The values just before the small movement or
- the vales that are read before starting of the continuous loop

The sensitivity values calculated using the values in the second case (which is done till now)are found to be very high and inaccurate.

The reason for this inaccuracy of high values is that, till now there is no movement in the arms of the robot and the smart stiffness concept worked correctly but now there is small movements continuously in the arms of the robots. As a result of the continuous change in position of the arms the difference values also changes continuously and limiting values are reached early, just after two to three changes in the position of the arms without the robot actually touching the object.

Thus the first case is used to calculate the difference of the values. These differences also seemed high but not as high as the values obtained during the second case. consequently the limiting values has to be changed for same loads to make the robot detect the load after actually lifting it.

Hence the sensitivity values and load values used are:

Number of hands	Sensitivity value	Load value
Single hand	0.08	250 grams
	0.1	400 grams
Both hands	0.08	450 grams
	0.1	600 grams

Table 4.2: Load And sensitivity values for one robot

As while trying to lift the object there is no perfect chance that both hands will be detected or not detected simultaneously. Therefore there will be a chance for more than two possibilities to arise while trying to lift the object. In that case, as there are two parameters(two sensitivity values for two hands) are there, there will be " ${}^2C_r$ " possibilities to occur where  $r$  is the number hands detected at a time. Thus there can be

$${}^2C_0 + {}^2C_1 + {}^2C_2 = 1 + 2 + 1 = 4$$

possible cases to arise. Out of the 4 cases the last case is desired one. Therefore to take care of all the other 3 cases an algorithm with two nested while loops inside the continuous loop is developed which also checks for the last case in every loop, every time. When the last case arises the program exits all the loops irrespective of the loop it currently executing and starts gripping and lifting the object, which is the obvious immediate action to be performed by the robot.

In the algorithm, the conditions for loops includes both the limits of the sensitivity values which constrains the amount of load that can be handled by one hand so the final case will not arise until both the hands share the load.

The Algorithm first lifts both the hands to a specific height after that it starts the continuous loop in which it continuously lifts both hands simultaneously by small amounts until one of the hand is detected with in the limits. if both the hands are detected the program exits the continuous loop. if only one hand is detected it will move the other hand until it is detected and checks for the first hand every time but it is not bothered until the second is detected. if the second hand is detected and first is disturbed then it will move first hand again until it is again detected. if second is disturbed again it comes out of the " first hand move loop" and starts to move the second hand while checking first hand but not bothering it until the second hand is detected. if at any point of time both hands gets disturbed the program jumps to the continuous loop where it simultaneously moves both hands. **As of the three cases every two cases are tackled inside the remaining one case the loop will not stop until the fourth case is detected**

The method used to move an arm looks like

```
algInterpolation("ArmName", "PositionValue", "time")
```

The Algorithm looks like

#### Algorithm:

```
lift both the hands to specific height
while( continuous loop)
{ get the hardness values
  lift both hands simultaneously by small amount
  again get the values and calculate the difference
  if ( required differencedetected)
    { stop the loop }
  else if ( required difference detected
            for any one hand)
    { while ( 1st hand is detected and the other does not )
      { lift the 2nd hand by small amount
        while ( 2nd hand is detected and other does not )
        { lift the 1st hand by small amount
        }
        while ( both are not detected )
        { repeat continuous while
        }
      }
    }
  }
```

```

    }
    repeat the 2nd while for the other " 2-C_1 " case
}
}

```

The Actual Program should be as follows

### Program :

```

#include <iostream>
#include <fstream>
#include <alerror/alerror.h>
#include <alproxies/alrobotpostureproxy.h>
#include <alproxies/almotionproxy.h>
#include <alproxies/almemoryproxy.h>
#include <alproxies/altexttospeechproxy.h>
#include <qj/os.hpp>

int main()
{
    try
    {
        AL::ALMotionProxy motion("192.168.8.149",9559);
        AL::ALTextToSpeechProxy tts("192.168.8.149", 9559);
        AL::ALRobotPostureProxy robotPosture("192.168.8.149",9559);
        AL::ALMemoryProxy memoryProxy("192.168.8.149", 9559);

        robotPosture.goToPosture("Crouch",0.5f);

        qj::os::msleep(700);

        const std::string
        lsph("Device/SubDeviceList/LShoulderPitch/Hardness/Actuator/Value"),
        lsrh("Device/SubDeviceList/LShoulderRoll/Hardness/Actuator/Value"),
        rsph("Device/SubDeviceList/RShoulderPitch/Hardness/Actuator/Value"),
        rsrh("Device/SubDeviceList/RShoulderRoll/Hardness/Actuator/Value");

        std::string nam = "LArm", nam2= "RArm";
        float stifnes = 1.0f;          float time = 1.0f;
        motion.stiffnessInterpolation(nam, stifnes, time);
        motion.post.stiffnessInterpolation(nam2, stifnes, time);

        AL::ALValue names = "LElbowYaw";
        AL::ALValue names2 = "RElbowYaw";
        AL::ALValue angellists; AL::ALValue angellists2;
        AL::ALValue angels = 0.5f;
        angellists = -1.57f; angellists2 = 1.57f;
    }
}

```

```

AL::ALValue timelists;AL::ALValue timelists2;
timelists = 2.0f; timelists2 = 2.0f;
bool isAbsolute = true;
    motion.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion.post.angleInterpolation(names2,angellists2,timelists2,isAbsolute);

names = "LWristYaw"; names2 = "RWristYaw";
angellists.clear(); angellists2.clear();
angellists = -1.57f; angellists2 = 1.57f;
timelists.clear(); timelists2.clear();
timelists = 2.0f; timelists2 = 2.0f;
isAbsolute = true;
    motion.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion.post.angleInterpolation(names2,angellists2,timelists2,isAbsolute);

names = "LElbowRoll"; names2 = "RElbowRoll";
angellists.clear(); angellists2.clear();
angellists = -2.0f ; angellists2 = 2.0f;
timelists.clear(); timelists2.clear();
timelists = 4.0f; timelists2 = 4.0f;
isAbsolute = true;
    motion.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion.post.angleInterpolation(names2,angellists2,timelists2,isAbsolute);

const std::string handname="LHand";
const std::string handname2="RHand";
    motion.post.openHand(handname);
    motion.post.openHand(handname2);

names = "LShoulderPitch";
names2 = "RShoulderPitch";
angellists.clear(); angellists = 1.25f; timelists.clear();
float a = angellists, b = a; timelists = 3.0f;
isAbsolute = true;
    motion.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion.post.angleInterpolation(names2,angellists,timelists,isAbsolute);

qi::os::sleep(1.0f);

AL::ALValue lp,rp, lp2,rp2, lp3,rp3;

lp = memoryProxy.getData(lsph); rp = memoryProxy.getData(rsph);

float lpv = lp, rpv = rp, lp2v,rp2v, lp3v,rp3v, lpd, rpd , k=-0.035f;

std::cout << "\n" << std::endl;
std::cout << "Left Shoulder Pitch Hardness : "<< lpv << std::endl;

```

```

        std::cout << "Right Shoulder Pitch Hardness : "<< rp3 << "\n"<<std::endl;

qi::os::sleep(2.0f);

while(1) // continuous loop begining
{
repeat :

        lp3 = memoryProxy.getData(lsph);          rp3 = memoryProxy.getData(rsph);
        lp3v = lp3;                                rp3v = rp3;

        a = a-0.05f;      b = b-0.05f;

        names = "LShoulderPitch";
        names2 = "RShoulderPitch";
        angellists.clear();      angels.clear();      angels = b;
        angellists = a;          timelists.clear();    timelists= 1.0f;
        isAbsolute = true;
        motion.post.angleInterpolation(names,angellists,timelists,isAbsolute);
        motion.angleInterpolation(names2,angels,timelists,isAbsolute);

        qi::os::msleep(500);

        lp1 = memoryProxy.getData(lsph);          rp1 = memoryProxy.getData(rsph);
        lp1v = lp1;                                rp1v = rp1;

        qi::os::sleep(1.0f);

        lp2=memoryProxy.getData(lsph); rp2=memoryProxy.getData(rsph);
        lp2v=lp2;      rp2v=rp2;

        lpd = lp3v - lp2v;      rpd = rp3v - rp2v;

        std::cout << "Left Shoulder Pitch Hardness differance (w1) : "<< lpd << std::endl;
        std::cout << "Right Shoulder Pitch Hardness differance (w1) : "<< rpd << "\n" << std::endl;

        if ( lpd <= k || rpd <= k )
        {
                if (lpd <= k && rpd >= k) tts.say("only left detected");
                if (rpd <= k && lpd >= k) tts.say("only Right detected");
                if (lpd <= k && rpd <= k) tts.say("both hands detected");

                while (rpd <= k && lpd >= k) // only right detected
                {
                        a = a-0.05f;
                        lp3 = memoryProxy.getData(lsph);    rp3 = memoryProxy.getData(rsph);

```



```

lp3v = lp3;                                rp3v = rp3;

motion.angleInterpolation("LShoulderPitch",a,1.0f,true);

qi::os::sleep(1.0f);

lp2 = memoryProxy.getData(lsph);  rp2 = memoryProxy.getData(rsph);
lp2v = lp2;                        rp2v = rp2;

lpd = lp3v - lp2v; rpd = rp3v - rp2v;

while (rpd >= k && lpd <= k) // right disturbed - left ok
{
    b = b - 0.05f;
    lp3 = memoryProxy.getData(lsph);    rp3 = memoryProxy.getData(rsph);
    lp3v = lp3;                        rp3v = rp3;

    motion.angleInterpolation("RShoulderPitch",b,1.0f,true);

    qi::os::sleep(1.0f);

    lp2 = memoryProxy.getData(lsph);    rp2 = memoryProxy.getData(rsph);
    lp2v = lp2;                        rp2v = rp2;

    lpd = lp3v - lp2v;  rpd = rp3v - rp2v;
}

if (rpd >= k && lpd >= k) // both disturbed
{
    goto repeat;
}
} // first while in if ending here

while (lpd <= k && rpd >= k) // only left detected
{
    b = b-0.05f;
    lp3 = memoryProxy.getData(lsph);    rp3 = memoryProxy.getData(rsph);
    lp3v = lp3;                        rp3v = rp3;

    motion.angleInterpolation("RShoulderPitch",b,1.0f,true);

    qi::os::sleep(1.0f);

    lp2 = memoryProxy.getData(lsph);    rp2 = memoryProxy.getData(rsph);
    lp2v = lp2;                        rp2v = rp2;

    lpd = lp3v - lp2v;    rpd = rp3v - rp2v;
}

```

```

while (lpd >= k && rpd <= k) // left disturbed - right ok
{
    a = a-0.05f;
    lp3 = memoryProxy.getData(lsph);    rp3 = memoryProxy.getData(rsph);
    lp3v = lp3;                        rp3v = rp3;

    motion.angleInterpolation("LShoulderPitch",a,1.0f,true);

    qi::os::sleep(1.0f);

    lp2 = memoryProxy.getData(lsph);    rp2 = memoryProxy.getData(rsph);
    lp2v = lp2;                        rp2v = rp2;

    lpd = lp3v - lp2v;                rpd = rp3v - rp2v;

}
if (rpd >= k && lpd >= k) // both disturbed
{
    goto repeat;
}
} // 2nd while in if is ending here

if (lpd <= k && rpd >= k) tts.say("only left detected");
if (rpd <= k && lpd >= k) tts.say("only Right detected");
if (lpd <= k && rpd <= k) tts.say("both hands detected");

break;
} // if loop ending here
} // main while loop ending here

motion.post.closeHand(handname);
motion.closeHand(handname2);

const std::string phraseToSay = "Got the object";
tts.say(phraseToSay);

qi::os::msleep(800);

motion.post.openHand(handname);
motion.openHand(handname2);

tts.say(" putting down the object");

qi::os::msleep(1000);

names = "LElbowRoll";    names2 = "RElbowRoll";

```

```

    angellists.clear();          angellists2.clear();
    angellists = 0.0f ;          angellists2 = 0.0f;
    timelists.clear();           timelists2.clear();
    timelists = 4.0f;            timelists2 = 4.0f;
    isAbsolute = true;
        motion.post.angleInterpolation(names,angellists,timelists,isAbsolute);
        motion.angleInterpolation(names2,angellists2,timelists2,isAbsolute);

    names = "LShoulderPitch";    names2 = "RShoulderPitch";
    angellists.clear();          angellists = 1.5f;
    timelists.clear();           timelists= 5.0f;
    isAbsolute = true;
        motion.post.angleInterpolation(names,angellists,timelists,isAbsolute);
        motion.angleInterpolation(names2,angellists,timelists,isAbsolute);

    robotPosture.goToPosture("Stand",0.5f);

    stifnes =0.0f;
        motion.stiffnessInterpolation(nam, stifnes, time);
        motion.post.stiffnessInterpolation(nam2, stifnes, time);
}
catch (const AL::ALError& e)
{
    std::cerr << "Caught exception: " << e.what() << std::endl;
    exit(1);
}
exit(0);
}

```

Here there is a possibility for the algorithm to become an **Infinite Loop**. Because of the " position value " parameter passed to the *angleInterpolation* method. It an absolute float value, not an incremental value. This problem is handled using other float variables and incrementing them during the loop and passing them as absolute value and if the same variable is used for both hands, again there can be a probability for infinite loop problem because the variable values of the continuous loop will be different from those of the case loop's variables and if at any point the program jumps from inside of one of the case loops due to " Both disturbed " condition to the continuous loop, it has to start from the same values at which it left the continuous loop for one of the case loops due to " Any one hand detected " condition. so the programs comes back to a position which it has passed once. Thus there may be a chance again for it to go through the same point where it has jumped back due to " Both disturbed " condition to one of the conditions( in main loop) it has passed and the cycle continues making the nested loop algorithm an infinite loop.

This problem is tackled by defining one separate variable for every hand before the continuous loop. and incrementing the corresponding variable for the corresponding hand in any loop. consequently if position of any hand is changed it's value is stored in a variable which is like global one as it is defined before the continuous loop. Therefore it does not matter which hand moves in which loop and wherever the program jumps as it does not effect the corresponding position values for the hands preventing the problem of passing through a position for a hand it has already passed. As a result the hands moves continuously in a perfect manner making the program to reach the final desired case.

## 4.5 Object Lifting by Two Robots

There are two robots now, which makes four shoulder pitch torque to be considered and which in turn makes four hardness difference values to be taken care of.

The sensitivity values and load values used are provided in the Table 4.3

Number of hands	Sensitivity value	Load value
Single robot	0.08	450 grams
	0.1	600 grams
Two robots	0.08	750 grams
	0.1	900 grams

Table 4.3: Load And sensitivity values for one robot

As there are four parameters now, there will be " ${}^4C_r$ " possibilities to occur where  $r$  is the number of hands detected. Thus there are

$${}^4C_0 + {}^4C_1 + {}^4C_2 + {}^4C_3 + {}^4C_4 = 1 + 4 + 6 + 4 + 1 = 16$$

possibilities to occur. Out of the 16 cases the last case of all hands detected is the desired one. The other 15 cases are:

${}^4C_1$	${}^4C_2$	${}^4C_3$
R1 L	R1 L , R1 R	R1 L , R1 R, R2 L
R1 R	R1 L , R2 L	R1 L , R1 R, R2 R
R2 L	R1 L , R2 R	R1 L , R2 L, R2 R
R2 R	R1 R , R1 L	R1 R , R2 L, R2 R
	R1 R , R2 R	
	R2 L , R2 R	

Figure 4.2: The 15 Cases

Therefore to take care of all the other 15 cases an algorithm having 3 nested loops inside a continuous loop is to be developed which also checks for the last case in every loop, every time. When that case occurs the program exits all the loops irrespective of the the loop it currently executing and start gripping and lifting the the object.

In the algorithm the conditions for loops includes both the limits of the sensitivity values which limits the amount of load that can be handled by any one hand so the final case will not arise until all the hands share the load.

In the previous case of only one robot lifting the object only three sets of cases arouse like "*no hand detected*", "*only one hand detected*" and "*all hands detected*". the first case is taken care of in the continuous loop and last case is the desired one and is checked through out the algorithm. And two other cases along with the first case, total 3 are handled by specifying the other two cases in remaining one case. But here there are 4 sets of cases. Even the last case is desired and checked everywhere in the program and first case is taken care of in the continuous loop. the other remaining loops including the first case, a total of 15 cases has to be taken care. In the previous case of single robot as the 3 has cases only two sets are there and one of the two sets contains only one case, the algorithm is formed with ease by fallowing the logic of specifying all the other in one case, but here such logic can not be fallowed as there are 3 sets of cases and every set consists more than one case.

If it is to specify all the other cases in one case, a total of 14 cases to be specified in one case and again inside of each of this 14 cases the remaining 13 cases has to be specified. which makes the algorithm very big and complex and can also make the loops an infinite one which will require very long time to execute.

This problem of redundancy is handled by keeping the cases in ascending order of the number of hands detected and checking for conditions of other cases in every set and making the program to jump to a particular set if it is detected inside any other set and fallowing two rules :

- The program should not go back in ascending order of sets arrangement.
- The program should not migrate from one loop to another loop in the same set.

Thus the algorithm should be like

while (continuous loop)

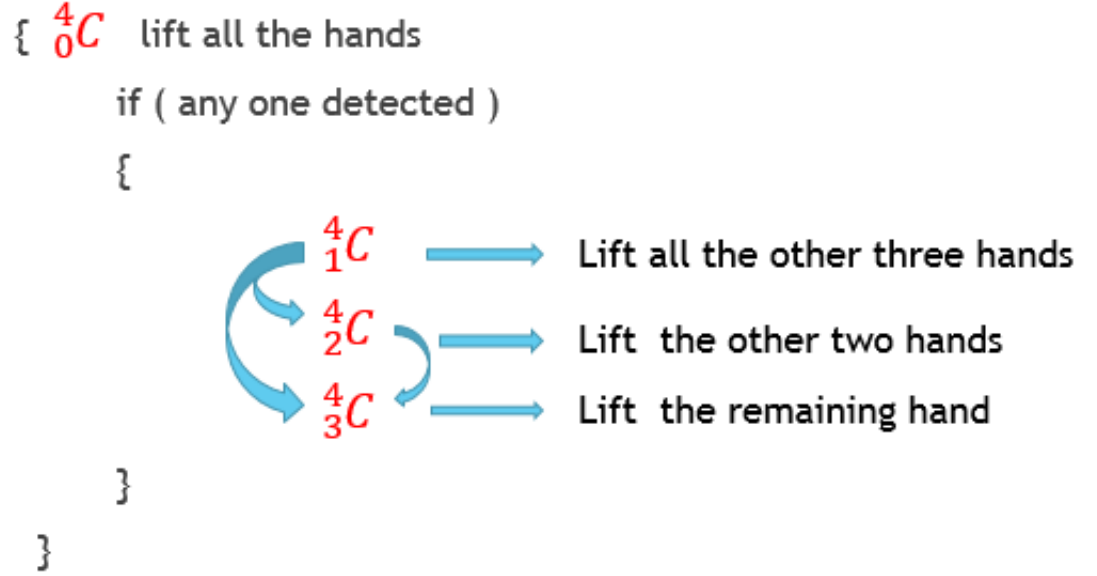


Figure 4.3: Two Robots program 1

In the case of  ${}^4C_1$  the algorithm not only move the other hands but also check for the possible cases in other sets and should prevent from migrating to other  ${}^4C_1$  case as per rule 1 of the formed rules. For example if the  ${}^4C_1$  case of  $R1 L$  is detected first it can jump to  $R1 L$  and  $R1 R$ ,  $R1 L$  and  $R2 L$ ,  $R1 L$  and  $R2 R$  cases of  ${}^4C_2$  or the first 3 cases of  ${}^4C_3$ , but not to other cases shown in the Fig 4.2. For the program to follow the proposed algorithm the  ${}^4C_1$  case of  $R1 L$  should be like

#### Algorithm:

```

While ( only one detected)
{
  lift all the other hands
  check the hardness difference values
  if (other case arises [detection of 1st one continued] )
  { jump to that case}
  if ( the detected one is disturbed )
  { lift the one }
}

```

Similarly the set 2 program should be like

### Algorithm:

```

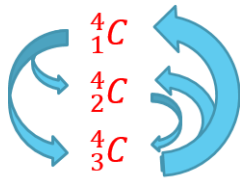
while ( any two detected )
{
    lift all the other hands
    check the hardness difference values
        if (other case arises [detection of the two continued] )
            { jump to that case}
        if ( any one of the detected two are disturbed)
            { *Here the 2-C_1 cses arises*
*1st 2-C_1 while <-- while ( 1st hand is detected and other does not )
            { lift the 2nd hand by small amount
    *subcase* <-- while ( 2nd hand is detected and other does not )
                { lift the 1st hand by small amount}
                while ( both are not detected)
                { both hands by small amounts }
                if ( all are disturbed )
                { repeat the main while loop }
                if (subcase is arises) { go to subcase line}
            }
            repeat the 2nd while for the other " 2-C_1 " case for
            subcase condition
        }
    }
}

```

up to the second set programs can be written following the formed rules but in the third set only one hand is not detected while moving that hand any other hand/hands may get disturbed so again all the 15 cases are possible here.so according to the formed rules, going back is not possible and dealing all the cases in one case is also impossible. so if the the rules are compromised and going back is allowed the program should look like. Fig 4.4

In this type of algorithm there can be chance for infinite loop again due to the "PositionValues" in method syntax for moving an arm but that problems already have solved by dedicating a separate variable for each arm. But still this type of program becomes the infinite loops due very frequent access to memory locations

```

while
{  $\begin{smallmatrix} 4 \\ 0 \end{smallmatrix}C$  lift all the hands
  if ( any one detected )
  {
    
  }
}

```

Figure 4.4: Two robots program 2

of the robots and very complex integration of the loops, the complexity and cost of computation has increased tremendously that the hardness values returned by robot are very close to **"ZERO"** making the program an infinite loop.

This problem is handled by modifying the two robot program as

```

while
{  $\begin{smallmatrix} 4 \\ 0 \end{smallmatrix}C$  lift all the hands
  if ( any one detected )
  {
     $\begin{smallmatrix} 4 \\ 1 \end{smallmatrix}C$ 
     $\begin{smallmatrix} 4 \\ 2 \end{smallmatrix}C$ 
     $\begin{smallmatrix} 4 \\ 3 \end{smallmatrix}C$ 

    if ( all are detected ) exit the main while loop
    if ( no one detected ) repeat the main while loop
    if ( any one detected ) repeat the main if loop
  }
}

```

Figure 4.5: Two robots final program

In the above algorithm the main changes are there is jumping of the program from one loop to other which drastically decreased the computational efficiency. and the



last line of the program **if (any one detected) repeat the main if loop** is used instead of the complex integration of the loops and it also prevents the program from exiting the continuous loop before all the hands are not detected and another if condition of all are not detected also prevents the program from exiting before all the hands are detected.

The Actual program should be as follows

### Program :

```
#include <iostream>
#include <fstream>
#include <alerror/alerror.h>
#include <alproxies/alrobotpostureproxy.h>
#include <alproxies/almotionproxy.h>
#include <alproxies/almemoryproxy.h>
#include <alproxies/altexttospeechproxy.h>
#include <qi/os.hpp>

int main()
{
    try
    {
        AL::ALMotionProxy motion("192.168.8.149",9559);
        AL::ALTextToSpeechProxy tts("192.168.8.149", 9559);
        AL::ALRobotPostureProxy robotPosture("192.168.8.149",9559);
        AL::ALMemoryProxy memoryProxy("192.168.8.149", 9559);

        AL::ALMemoryProxy memoryProxy2("192.168.8.101", 9559);
        AL::ALMotionProxy motion2("192.168.8.101",9559);
        AL::ALTextToSpeechProxy tts2("192.168.8.101", 9559);
        AL::ALRobotPostureProxy robotPosture2("192.168.8.101",9559);

        robotPosture.goToPosture("Crouch",0.5f);
        robotPosture2.goToPosture("Crouch",0.5f);

        qi::os::msleep(500);

        const std::string
        lsph("Device/SubDeviceList/LShoulderPitch/Hardness/Actuator/Value"),
        lsrh("Device/SubDeviceList/LShoulderRoll/Hardness/Actuator/Value"),
        rsph("Device/SubDeviceList/RShoulderPitch/Hardness/Actuator/Value"),
        rsrh("Device/SubDeviceList/RShoulderRoll/Hardness/Actuator/Value");

        std::string nam = "LArm",      nam2= "RArm";
        float stifnes = 1.0f,          time = 1.0f;
```

```

        motion.post.stiffnessInterpolation(nam, stifnes, time);
        motion.post.stiffnessInterpolation(nam2, stifnes, time);
        motion2.post.stiffnessInterpolation(nam, stifnes, time);
        motion2.stiffnessInterpolation(nam2, stifnes, time);

qi::os::msleep(200);

AL::ALValue names = "LElbowYaw";
AL::ALValue names2 = "RElbowYaw";
AL::ALValue angellists;    AL::ALValue angellists2;
angellists = -1.57f;        angellists2 = 1.57f;
AL::ALValue timelists;    AL::ALValue timelists2;
timelists= 2.0f;            timelists2= 2.0f;
bool isAbsolute = true;
    motion.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion2.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion.post.angleInterpolation(names2,angellists2,timelists2,isAbsolute);
    motion2.post.angleInterpolation(names2,angellists2,timelists2,isAbsolute);

names = "LWristYaw";        names2 = "RWristYaw";
angellists.clear();          angellists2.clear();
angellists = -1.57f;          angellists2 = 1.57f;
timelists.clear();            timelists2.clear();
timelists= 2.0f;              timelists2= 2.0f;
isAbsolute = true;
    motion.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion2.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion.post.angleInterpolation(names2,angellists2,timelists2,isAbsolute);
    motion2.post.angleInterpolation(names2,angellists2,timelists2,isAbsolute);

names = "LElbowRoll";        names2 = "RElbowRoll";
angellists.clear();          angellists2.clear();
angellists = -2.0f;          angellists2 = 2.0f;
timelists.clear();            timelists2.clear();
timelists = 3.0f;            timelists2= 3.0f;
isAbsolute = true;
    motion.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion2.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion.post.angleInterpolation(names2,angellists2,timelists2,isAbsolute);
    motion2.post.angleInterpolation(names2,angellists2,timelists2,isAbsolute);

const std::string handname="LHand";
const std::string handname2="RHand";
    motion.post.openHand(handname);
    motion2.post.openHand(handname);
    motion.post.openHand(handname2);
    motion2.openHand(handname2);

```

```

names = "LShoulderPitch";
names2 = "RShoulderPitch";
angellists.clear();          angellists = 1.1f;
timelists.clear();          timelists= 3.0f;
isAbsolute = true;
    motion.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion2.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion.post.angleInterpolation(names2,angellists,timelists,isAbsolute);
    motion2.angleInterpolation(names2,angellists,timelists,isAbsolute);

AL::ALValue lp11, rp11, lp21, rp21, lp12, rp12, lp22, rp22, lp1, rp1, lp2, rp2;
float lp11v, rp11v, lp21v, rp21v, lp1v, rp1v, lp2v, rp2v;
float lp1d, rp1d, lp2d, rp2d, lp12v, rp12v, lp22v, rp22v,
a = angellists, b, c, d, k, l, m, n;
b = c = d = a;
float k = -0.009f; l = -0.009f; m = -0.009f; n = -0.009f;

lp1 = memoryProxy.getData(lsph);          rp1 = memoryProxy.getData(rsph);
lp1v = lp1;                               rp1v = rp1;

lp2 = memoryProxy2.getData(lsph);          rp2 = memoryProxy2.getData(rsph);
lp2v = lp2;                               rp2v = rp2;

Ostd::cout << "\n" << std::endl;
std::cout << "Left Shoulder Hardness Robo 1 : "<< lp1v << std::endl;
std::cout << "Right Shoulder Hardness Robo 1 : "<< rp1v << std::endl;
std::cout << "Left Shoulder Hardness Robo 2 : "<< lp2v<< std::endl;
std::cout << "Right Shoulder Hardness Robo 2 : "<< rp2v << std::endl;
std::cout << "\n" << std::endl;

AL::ALValue angel1 = 0.0f, angel2 =0.0f, angel3 = 0.0f, angel4 = 0.0f;

while(1)
{
repeat:

    lp11 = memoryProxy.getData(lsph);          rp11 = memoryProxy.getData(rsph);
    lp11v = lp11;                               rp11v = rp11;
    lp21 = memoryProxy2.getData(lsph);          rp21 = memoryProxy2.getData(rsph);
    lp21v = lp21;                               rp21v = rp21;

    names = "LShoulderPitch";          names2 = "RShoulderPitch";
    a = a - 0.05f;    b = b - 0.05f;    c = c - 0.05f;    d = d - 0.05f;
    angel1.clear();          angel2.clear();
    angel3.clear();          angel4.clear();
    angel1 = a;  angel2 = b;  angel3 = c;  angel4 = d;

```

```

timelists.clear();          timelists = 1.0f;
isAbsolute = true;
    motion.post.angleInterpolation(names,angel1,timelists,isAbsolute);
    motion2.post.angleInterpolation(names,angel3,timelists,isAbsolute);
    motion.post.angleInterpolation(names2,angel2,timelists,isAbsolute);
    motion2.angleInterpolation(names2,angel4,timelists,isAbsolute);

qi::os::sleep(1.0f);

lp12 = memoryProxy.getData(lsph);      rp12 = memoryProxy.getData(rsph);
lp12v = lp12;                          rp12v = rp12;
lp22 = memoryProxy2.getData(lsph);      rp22 = memoryProxy2.getData(rsph);
lp22v = lp22;                          rp22v = rp22;

lp1d = lp11v - lp12v;                  rp1d = rp11v - rp12v;
lp2d = lp21v - lp22v;                  rp2d = rp21v - rp22v;

std::cout << "Left Shoulder Hardness differance Robo 1 : "<< lp1d << std::endl;
std::cout << "Right Shoulder Hardness differance Robo 1 : "<< rp1d << std::endl;
std::cout << "Left Shoulder Hardness differance Robo 2 : "<< lp2d << std::endl;
std::cout << "Right Shoulder Hardness differance Robo 2 : "<< rp2d << std::endl;
std::cout << "\n" << std::endl;

if ( lp1d <= k || rp1d <= l || lp2d <= m || rp2d <= n ) // any 1 detected
{
    repeatif :

        if (lp1d <= k && rp1d <= l && lp2d <= m && rp2d <= n)
        { tts.say("My both hands detected"); tts2.say("mine too"); goto finalbreak;
        }

    while ( lp1d >= k && rp1d <= l && lp2d <= m && rp2d <= n ) // lp1d not detected
    {
        a = a-0.05f;

        lp11 = memoryProxy.getData(lsph);      rp11 = memoryProxy.getData(rsph);
        lp11v = lp11;                          rp11v = rp11;
        lp21 = memoryProxy2.getData(lsph);      rp21 = memoryProxy2.getData(rsph);
        lp21v = lp21;                          rp21v = rp21;

        motion.angleInterpolation("LShoulderPitch",a,1.0f,true); qi::os::sleep(1.0f);

        lp12 = memoryProxy.getData(lsph);      rp12 = memoryProxy.getData(rsph);
        lp12v = lp12;                          rp12v = rp12;
        lp22 = memoryProxy2.getData(lsph);      rp22 = memoryProxy2.getData(rsph);
        lp22v = lp22;                          rp22v = rp22;

```

```

        lp1d = lp11v - lp12v;          rp1d = rp11v - rp12v;
        lp2d = lp21v - lp22v;          rp2d = rp21v - rp22v;
    }

while ( rp1d >= k && lp1d <= 1 && lp2d <= m && rp2d <= n) // rp1d not detected
{
    b = b-0.05f;

    lp11 = memoryProxy.getData(lsph);  rp11 = memoryProxy.getData(rsph);
    lp11v = lp11;                      rp11v = rp11;
    lp21 = memoryProxy2.getData(lsph); rp21 = memoryProxy2.getData(rsph);
    lp21v = lp21;                      rp21v = rp21;

    motion.angleInterpolation("RShoulderPitch",b,1.0f,true); qi::os::sleep(1.0f);

    lp12 = memoryProxy.getData(lsph); rp12 = memoryProxy.getData(rsph);
    lp12v = lp12;                      rp12v = rp12;
    lp22 = memoryProxy2.getData(lsph); rp22 = memoryProxy2.getData(rsph);
    lp22v = lp22;                      rp22v = rp22;

    lp1d = lp11v - lp12v;          rp1d = rp11v - rp12v;
    lp2d = lp21v - lp22v;          rp2d = rp21v - rp22v;
}

while ( lp2d >= k && lp1d <= 1 && rp1d <= m && rp2d <= n) // lp2d not detected
{
    c = c-0.05f;

    lp11 = memoryProxy.getData(lsph);  rp11 = memoryProxy.getData(rsph);
    lp11v = lp11;                      rp11v = rp11;
    lp21 = memoryProxy2.getData(lsph); rp21 = memoryProxy2.getData(rsph);
    lp21v = lp21;                      rp21v = rp21;

    motion2.angleInterpolation("LShoulderPitch",c,1.0f,true); qi::os::sleep(1.0f);

    lp12 = memoryProxy.getData(lsph); rp12 = memoryProxy.getData(rsph);
    lp12v = lp12;                      rp12v = rp12;
    lp22 = memoryProxy2.getData(lsph); rp22 = memoryProxy2.getData(rsph);
    lp22v = lp22;                      rp22v = rp22;

    lp1d = lp11v - lp12v;          rp1d = rp11v - rp12v;
    lp2d = lp21v - lp22v;          rp2d = rp21v - rp22v;
}

while ( rp2d >= k && lp1d <= 1 && rp1d <= m && lp2d <= n) // rp2d not detected
{
    d = d-0.05f;

```

```

lp11 = memoryProxy.getData(lsph);    rp11 = memoryProxy.getData(rsph);
lp11v = lp11;                        rp11v = rp11;
lp21 = memoryProxy2.getData(lsph);    rp21 = memoryProxy2.getData(rsph);
lp21v = lp21;                        rp21v = rp21;

motion2.angleInterpolation("RShoulderPitch",d,1.0f,true); qi::os::sleep(1.0f);

lp12 = memoryProxy.getData(lsph);    rp12 = memoryProxy.getData(rsph);
lp12v = lp12;                        rp12v = rp12;
lp22 = memoryProxy2.getData(lsph);    rp22 = memoryProxy2.getData(rsph);
lp22v = lp22;                        rp22v = rp22;

lp1d = lp11v - lp12v;                rp1d = rp11v - rp12v;
lp2d = lp21v - lp22v;                rp2d = rp21v - rp22v;
}

while ( lp1d >= k && rp1d >= l && lp2d <= m && rp2d <= n) // lp1d and rp1d not
    detected
{
    a = a-0.05f;                b = b-0.05f;

    lp11 = memoryProxy.getData(lsph);    rp11 = memoryProxy.getData(rsph);
    lp11v = lp11;                        rp11v = rp11;
    lp21 = memoryProxy2.getData(lsph);    rp21 = memoryProxy2.getData(rsph);
    lp21v = lp21;                        rp21v = rp21;

    motion.post.angleInterpolation("LShoulderPitch",a,1.0f,true);
    motion.angleInterpolation("RShoulderPitch",b,1.0f,true); qi::os::sleep(1.0f);

    lp12 = memoryProxy.getData(lsph);    rp12 = memoryProxy.getData(rsph);
    lp12v = lp12;                        rp12v = rp12;
    lp22 = memoryProxy2.getData(lsph);    rp22 = memoryProxy2.getData(rsph);
    lp22v = lp22;                        rp22v = rp22;

    lp1d = lp11v - lp12v;                rp1d = rp11v - rp12v;
    lp2d = lp21v - lp22v;                rp2d = rp21v - rp22v;
}

while ( lp1d >= k && lp2d >= l && rp1d <= m && rp2d <= n) // lp1d and lp2d not
    detected
{
    a = a-0.05f;                c = c-0.05f;

    lp11 = memoryProxy.getData(lsph);    rp11 = memoryProxy.getData(rsph);
    lp11v = lp11;                        rp11v = rp11;
    lp21 = memoryProxy2.getData(lsph);    rp21 = memoryProxy2.getData(rsph);

```

```

lp21v = lp21;                                rp21v = rp21;

motion.post.angleInterpolation("LShoulderPitch",a,1.0f,true);
motion2.angleInterpolation("LShoulderPitch",c,1.0f,true); qi::os::sleep(1.0f);

lp12 = memoryProxy.getData(lsph); rp12 = memoryProxy.getData(rsph);
lp12v = lp12;                                rp12v = rp12;
lp22 = memoryProxy2.getData(lsph); rp22 = memoryProxy2.getData(rsph);
lp22v = lp22;                                rp22v = rp22;

lp1d = lp11v - lp12v;                        rp1d = rp11v - rp12v;
lp2d = lp21v - lp22v;                        rp2d = rp21v - rp22v;
}

while ( lp1d >= k && rp2d >= l && rp1d <= m && lp2d <= n) // lp1d and rp2d not
    detected
{
    a = a-0.05f;            d = d-0.05f;

    lp11 = memoryProxy.getData(lsph);    rp11 = memoryProxy.getData(rsph);
    lp11v = lp11;                        rp11v = rp11;
    lp21 = memoryProxy2.getData(lsph); rp21 = memoryProxy2.getData(rsph);
    lp21v = lp21;                        rp21v = rp21;

    motion.post.angleInterpolation("LShoulderPitch",a,1.0f,true);
    motion2.angleInterpolation("RShoulderPitch",d,1.0f,true); qi::os::sleep(1.0f);

    lp12 = memoryProxy.getData(lsph); rp12 = memoryProxy.getData(rsph);
    lp12v = lp12;                        rp12v = rp12;
    lp22 = memoryProxy2.getData(lsph); rp22 = memoryProxy2.getData(rsph);
    lp22v = lp22;                        rp22v = rp22;

    lp1d = lp11v - lp12v;                rp1d = rp11v - rp12v;
    lp2d = lp21v - lp22v;                rp2d = rp21v - rp22v;
}

while ( rp1d >= k && lp2d >= l && lp1d <= m && rp2d <= n) // rp1d and lp2d not
    detected
{
    b = b-0.05f;            c = c-0.05f;

    lp11 = memoryProxy.getData(lsph);    rp11 = memoryProxy.getData(rsph);
    lp11v = lp11;                        rp11v = rp11;
    lp21 = memoryProxy2.getData(lsph); rp21 = memoryProxy2.getData(rsph);
    lp21v = lp21;                        rp21v = rp21;

    motion.post.angleInterpolation("RShoulderPitch",b,1.0f,true);

```

```

motion2.angleInterpolation("LShoulderPitch",c,1.0f,true); qi::os::sleep(1.0f);

lp12 = memoryProxy.getData(lsph); rp12 = memoryProxy.getData(rsph);
lp12v = lp12;                      rp12v = rp12;
lp22 = memoryProxy2.getData(lsph); rp22 = memoryProxy2.getData(rsph);
lp22v = lp22;                      rp22v = rp22;

lp1d = lp11v - lp12v;              rp1d = rp11v - rp12v;
lp2d = lp21v - lp22v;              rp2d = rp21v - rp22v;
}

while ( rp1d >= k && rp2d >= 1 && lp1d <= m && lp2d <= n) // rp1d and rp2d not
    detected
{
    b = b-0.05f;          d = d-0.05f;

    lp11 = memoryProxy.getData(lsph); rp11 = memoryProxy.getData(rsph);
    lp11v = lp11;          rp11v = rp11;
    lp21 = memoryProxy2.getData(lsph); rp21 = memoryProxy2.getData(rsph);
    lp21v = lp21;          rp21v = rp21;

    motion.post.angleInterpolation("RShoulderPitch",b,1.0f,true);
    motion2.angleInterpolation("RShoulderPitch",d,1.0f,true); qi::os::sleep(1.0f);

    lp12 = memoryProxy.getData(lsph); rp12 = memoryProxy.getData(rsph);
    lp12v = lp12;                      rp12v = rp12;
    lp22 = memoryProxy2.getData(lsph); rp22 = memoryProxy2.getData(rsph);
    lp22v = lp22;                      rp22v = rp22;

    lp1d = lp11v - lp12v;              rp1d = rp11v - rp12v;
    lp2d = lp21v - lp22v;              rp2d = rp21v - rp22v;
}

while ( lp2d >= k && rp2d >= 1 && lp1d <= m && rp1d <= n) // lp2d and rp2d not
    detected
{
    c = c-0.05f;          d = d-0.05f;

    lp11 = memoryProxy.getData(lsph); rp11 = memoryProxy.getData(rsph);
    lp11v = lp11;          rp11v = rp11;
    lp21 = memoryProxy2.getData(lsph); rp21 = memoryProxy2.getData(rsph);
    lp21v = lp21;          rp21v = rp21;

    motion2.post.angleInterpolation("LShoulderPitch",c,1.0f,true);
    motion2.angleInterpolation("RShoulderPitch",d,1.0f,true); qi::os::sleep(1.0f);

    lp12 = memoryProxy.getData(lsph); rp12 = memoryProxy.getData(rsph);

```



```

lp12v = lp12;                rp12v = rp12;
lp22 = memoryProxy2.getData(lsph); rp22 = memoryProxy2.getData(rsph);
lp22v = lp22;                rp22v = rp22;

lp1d = lp11v - lp12v;        rp1d = rp11v - rp12v;
lp2d = lp21v - lp22v;        rp2d = rp21v - rp22v;
}

while ( lp1d >= k && rp1d >= l && lp2d >= m && rp2d <= n) // rp2d detected
{
    a = a-0.05f;            b = b-0.05f;            c = c-0.05f;

    lp11 = memoryProxy.getData(lsph);  rp11 = memoryProxy.getData(rsph);
    lp11v = lp11;                rp11v = rp11;
    lp21 = memoryProxy2.getData(lsph); rp21 = memoryProxy2.getData(rsph);
    lp21v = lp21;                rp21v = rp21;

    motion.post.angleInterpolation("LShoulderPitch",a,1.0f,true);
    motion.post.angleInterpolation("RShoulderPitch",b,1.0f,true);
    motion2.angleInterpolation("LShoulderPitch",c,1.0f,true); qi::os::sleep(1.0f);

    lp12 = memoryProxy.getData(lsph); rp12 = memoryProxy.getData(rsph);
    lp12v = lp12;                rp12v = rp12;
    lp22 = memoryProxy2.getData(lsph); rp22 = memoryProxy2.getData(rsph);
    lp22v = lp22;                rp22v = rp22;

    lp1d = lp11v - lp12v;        rp1d = rp11v - rp12v;
    lp2d = lp21v - lp22v;        rp2d = rp21v - rp22v;
}

while ( lp1d >= k && rp1d >= l && rp2d >= m && lp2d <= n) // lp2d detected
{
    a = a-0.05f;            b = b-0.05f;            d = d-0.05f;

    lp11 = memoryProxy.getData(lsph);  rp11 = memoryProxy.getData(rsph);
    lp11v = lp11;                rp11v = rp11;
    lp21 = memoryProxy2.getData(lsph); rp21 = memoryProxy2.getData(rsph);
    lp21v = lp21;                rp21v = rp21;

    motion.post.angleInterpolation("LShoulderPitch",a,1.0f,true);
    motion.post.angleInterpolation("RShoulderPitch",b,1.0f,true);
    motion2.angleInterpolation("RShoulderPitch",d,1.0f,true); qi::os::sleep(1.0f);

    lp12 = memoryProxy.getData(lsph); rp12 = memoryProxy.getData(rsph);
    lp12v = lp12;                rp12v = rp12;
    lp22 = memoryProxy2.getData(lsph); rp22 = memoryProxy2.getData(rsph);
    lp22v = lp22;                rp22v = rp22;
}

```

```

        lp1d = lp11v - lp12v;          rp1d = rp11v - rp12v;
        lp2d = lp21v - lp22v;          rp2d = rp21v - rp22v;
    }

while ( lp1d >= k && lp2d >= l && rp2d >= m && rp1d <= n) // rp1d detected
{
    a = a-0.05f;          c = c-0.05f;          d = d-0.05f;

    lp11 = memoryProxy.getData(lsph);  rp11 = memoryProxy.getData(rsph);
    lp11v = lp11;                    rp11v = rp11;
    lp21 = memoryProxy2.getData(lsph); rp21 = memoryProxy2.getData(rsph);
    lp21v = lp21;                    rp21v = rp21;

    motion.post.angleInterpolation("LShoulderPitch",a,1.0f,true);
    motion2.post.angleInterpolation("LShoulderPitch",c,1.0f,true);
    motion2.angleInterpolation("RShoulderPitch",d,1.0f,true); qi::os::sleep(1.0f);

    lp12 = memoryProxy.getData(lsph); rp12 = memoryProxy.getData(rsph);
    lp12v = lp12;                    rp12v = rp12;
    lp22 = memoryProxy2.getData(lsph); rp22 = memoryProxy2.getData(rsph);
    lp22v = lp22;                    rp22v = rp22;

    lp1d = lp11v - lp12v;          rp1d = rp11v - rp12v;
    lp2d = lp21v - lp22v;          rp2d = rp21v - rp22v;
}

while ( rp1d >= k && lp2d >= l && rp2d >= m && lp1d <= n) // lp1d detected
{
    b = b-0.05f;          c = c-0.05f;          d = d-0.05f;

    lp11 = memoryProxy.getData(lsph);  rp11 = memoryProxy.getData(rsph);
    lp11v = lp11;                    rp11v = rp11;
    lp21 = memoryProxy2.getData(lsph); rp21 = memoryProxy2.getData(rsph);
    lp21v = lp21;                    rp21v = rp21;

    motion.post.angleInterpolation("RShoulderPitch",b,1.0f,true);
    motion2.post.angleInterpolation("LShoulderPitch",c,1.0f,true);
    motion2.angleInterpolation("RShoulderPitch",d,1.0f,true); qi::os::sleep
        (1.0f);

    lp12 = memoryProxy.getData(lsph); rp12 = memoryProxy.getData(rsph);
    lp12v = lp12;                    rp12v = rp12;
    lp22 = memoryProxy2.getData(lsph); rp22 = memoryProxy2.getData(rsph);
    lp22v = lp22;                    rp22v = rp22;

    lp1d = lp11v - lp12v;          rp1d = rp11v - rp12v;

```

```

        lp2d = lp21v - lp22v;          rp2d = rp21v - rp22v;
    }

    if (lp1d <= k && rp1d <= l && lp2d <= m && rp2d <= n)
    { tts.say("My both hands detected"); tts2.say("mine too"); goto finalbreak; }

    if (lp1d >= k && rp1d >= l && lp2d >= m && rp2d >= n)
    { tts.say("My both hands are not detected");
      tts2.say("mine too are not detected"); goto repeat;
    }

    if (lp1d <= k || rp1d <= l || lp2d <= m || rp2d <= n) goto repeatif;

finalbreak: break;

    }// end of main if loop
} // end of main while loop

names = "LShoulderPitch";
names2 = "RShoulderPitch";
a = a - 0.15f;      b = b - 0.15f;      c = c - 0.15f;      d = d - 0.15f;
angel1.clear();     angel2.clear();     angel3.clear();     angel4.clear();
angel1 = a; angel2 = b; angel3 = c; angel4 = d;
timelists= 4.0f;     isAbsolute = true;
    motion.post.angleInterpolation(names,angel1,timelists,isAbsolute);
    motion2.post.angleInterpolation(names,angel3,timelists,isAbsolute);
    motion.post.angleInterpolation(names2,angel2,timelists,isAbsolute);
    motion2.angleInterpolation(names2,angel4,timelists,isAbsolute);

    motion.post.closeHand(handname);
    motion2.post.closeHand(handname);
    motion.post.closeHand(handname2);
    motion2.closeHand(handname2);

const std::string phraseToSay = "Got the object";
tts.say(phraseToSay);

qi::os::msleep(500);

motion.post.openHand(handname);
motion2.post.openHand(handname);
motion.post.openHand(handname2);
motion2.openHand(handname2);

const std::string phraseToSay2 = "putting down the object";
tts.say(phraseToSay2);

```

```

qi::os::sleep(1.0f);

names = "LElbowRoll";
names2 = "RElbowRoll";
angellists.clear();          angellists2.clear();
angellists = 0.0f ;          angellists2 = 0.0f;
timelists.clear();          timelists2.clear();
timelists = 3.0f;          timelists2 = 3.0f;
isAbsolute = true;
    motion.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion2.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion.post.angleInterpolation(names2,angellists2,timelists2,isAbsolute);
    motion2.post.angleInterpolation(names2,angellists2,timelists2,isAbsolute);

names= "LShoulderPitch";
names2= "RShoulderPitch";
angellists.clear();          angellists = 1.5f;
timelists.clear();          timelists= 3.0f;
isAbsolute = true;
    motion.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion2.post.angleInterpolation(names,angellists,timelists,isAbsolute);
    motion.post.angleInterpolation(names2,angellists,timelists,isAbsolute);
    motion2.angleInterpolation(names2,angellists,timelists,isAbsolute);

qi::os::msleep(300);

robotPosture.goToPosture("Crouch",0.5f);
robotPosture2.goToPosture("Crouch",0.5f);

qi::os::msleep(300);

stifnes =0.0f;
    motion.post.stiffnessInterpolation(nam, stifnes, time);
    motion.post.stiffnessInterpolation(nam2, stifnes, time);
    motion2.post.stiffnessInterpolation(nam, stifnes, time);
    motion2.stiffnessInterpolation(nam2, stifnes, time);
}
catch (const AL::ALError& e)
{
    std::cerr << "Caught exception: " << e.what() << std::endl;
    exit(1);
}
exit(0);
}

```

## Chapter 5

# Results and Discussions

### 5.1 Results

#### 5.1.1 Single Robot Lifting the Object

The sensitivity and the load values used for one robot are provided in Table 4.2

1. The robot just lifted it's hands to a specific height



Figure 5.1: single robot just reached specific height

2. The robot just touched it's hands to the object.

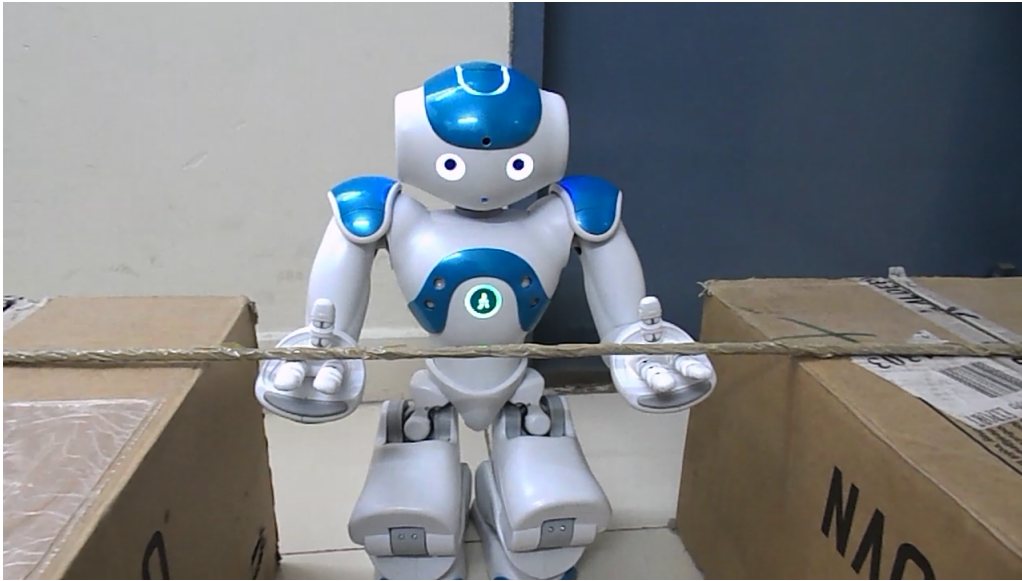


Figure 5.2: single robot just touching the object

3. The robot Left detected hand detected first(one of the  ${}^2C_1$  case raised). It can be seen that Center of Mass of the object towards left so left is First detected.



Figure 5.3: single robot Left hand detected

4. The robot right hand raised until it is detected in order to share the load and also balance it.



Figure 5.4: single robot Right hand raised to share Load

5. Again the robot Left hand is raised (sub case of  ${}^2C_1$  case) in order to share the load and also balance it. the load is finally shared and both hands are detected.



Figure 5.5: single robot left hand raised to share Load



### 5.1.2 Two Robots Lifting the Object

the Sensitivity and load values used are provided in Table 4.3

1. The Two robots just lifted their hands to a specific height

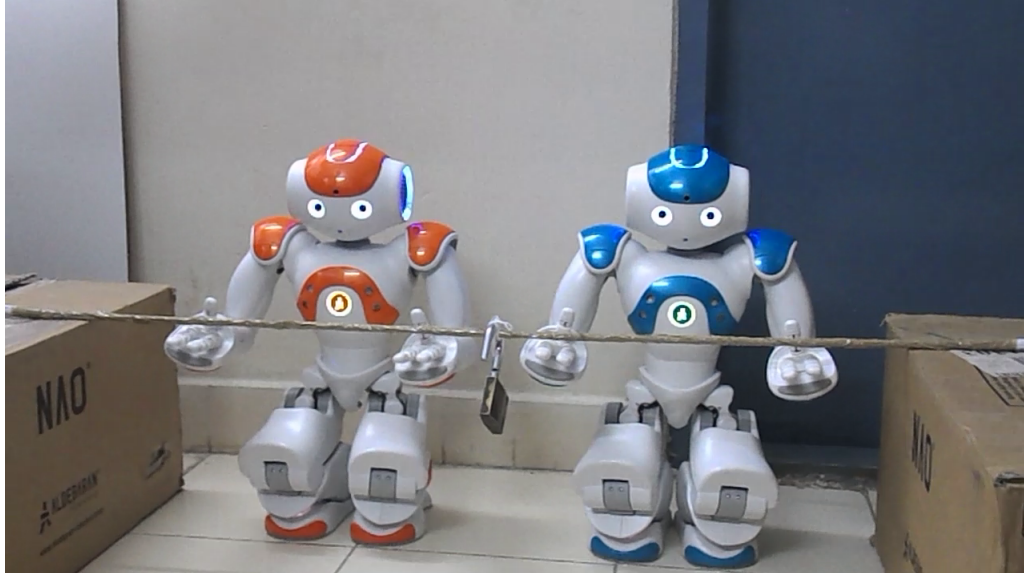


Figure 5.6: Two robots just reached specific height

2. The Two robots just touched their hands to object

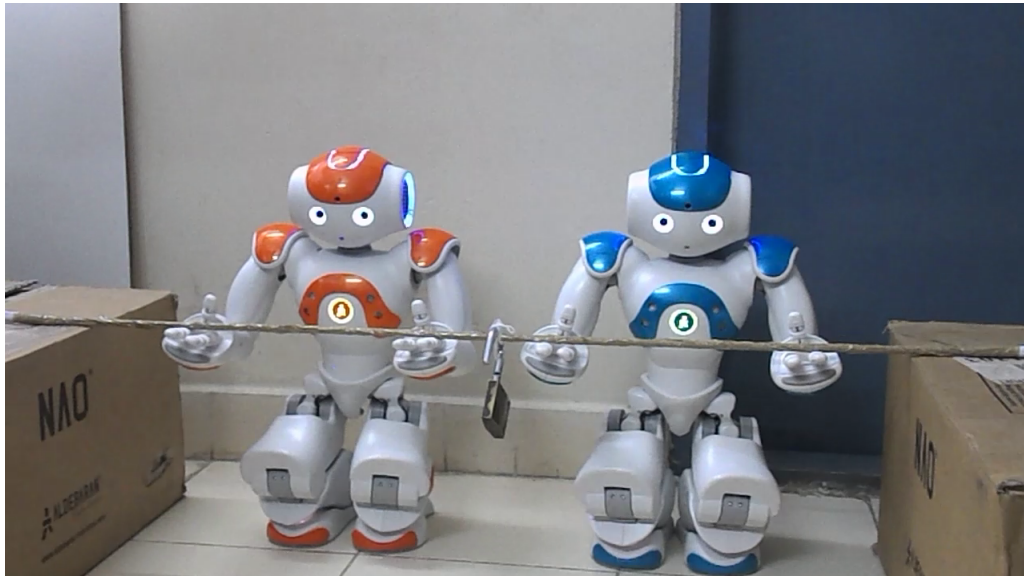


Figure 5.7: Two robots just touching the object



3. The Two robots cooperatively lifting the object.

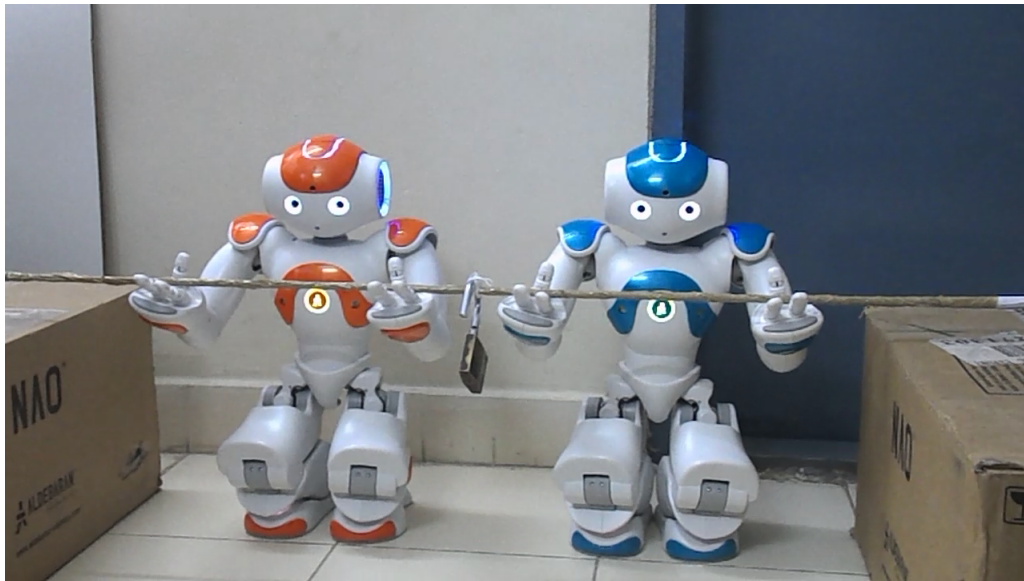


Figure 5.8: Two robots cooperatively lifting the object

4. All the hands are detected. The Two robots sharing the load.



Figure 5.9: Two robots Sharing the load

## 5.2 Conclusions

The Algorithms developed for Cooperative Load Sharing for single robot and two robots are successfully tested, optimized for computational cost effectiveness and verified by experimenting on the Nao Humanoid Robots. The Algorithm developed for two Robots can be easily extended to any number of robots, It requires just adding on the cases to the program.

## Chapter 6

### Further Work

1. To modify the Algorithm for the robots to lift non-stiff objects like clothes, wires etc.
2. To extend the program and add to the Robot Library
3. To extend the algorithm including the other motions like walking
4. TO modify the algorithm to include obstacle detection.

# Bibliography

- [1] J. A. Corrales Ramón, G. J. García Gómez, F. Torres Medina, and V. Perdereau. Cooperative tasks between humans and robots in industrial environments .
- [2] M. Lawitzky, A. Mörtl, and S. Hirche. Load sharing in human-robot cooperative manipulation. In RO-MAN, 2010 IEEE. IEEE, 2010 185–191.
- [3] S. Lallée, U. Pattacini, S. Lemaignan, A. Lenz, C. Melhuish, L. Natale, S. Skachek, K. Hamann, J. Steinwender, E. A. Sisbot et al. Towards a platform-independent cooperative human robot interaction system: Iii an architecture for learning and executing actions and shared plans. *Autonomous Mental Development, IEEE Transactions on* 4, (2012) 239–253.
- [4] S. Ikemoto, H. Ben Amor, T. Minato, H. Ishiguro, and B. Jung. Physical interaction learning: Behavior adaptation in cooperative human-robot tasks involving physical contact. In Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on. IEEE, 2009 504–509.
- [5] L. E. Parker and C. Touzet. Multi-robot learning in a cooperative observation task. In Distributed autonomous robotic systems 4, 391–401. Springer, 2000.
- [6] M.-H. Wu, A. Konno, and M. Uchiyama. Cooperative object transportation by multiple humanoid robots. In System Integration (SII), 2011 IEEE/SICE International Symposium on. IEEE, 2011 779–784.
- [7] G. J. Barlow, C. K. Oh, and S. F. Smith. Evolving cooperative control on sparsely distributed tasks for UAV teams without global communication. In Proceedings of the 10th annual conference on Genetic and evolutionary computation. ACM, 2008 177–184.
- [8] D. Kurabayashi, J. Ota, T. Arai, and E. Yoshida. Cooperative sweeping by multiple mobile robots. In Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on, volume 2. IEEE, 1996 1744–1749.

- [9] N. Miyata, J. Ota, T. Arai, and H. Asama. Cooperative transport by multiple mobile robots in unknown static environments associated with real-time task assignment. *Robotics and Automation, IEEE Transactions on* 18, (2002) 769–780.
- [10] F. Ducatelle, G. A. Di Caro, and L. M. Gambardella. Cooperative self-organization in a heterogeneous swarm robotic system. In Proceedings of the 12th annual conference on Genetic and evolutionary computation. ACM, 2010 87–94.
- [11] Y. Inoue, T. Tohge, and H. Iba. Object transportation by two humanoid robots using cooperative learning. In Evolutionary Computation, 2004. CEC2004. Congress on, volume 1. IEEE, 2004 1201–1208.
- [12] A. Thobbi, Y. Gu, and W. Sheng. Using human motion estimation for human-robot cooperative manipulation. In Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on. IEEE, 2011 2873–2878.
- [13] E. Berger, D. Vogt, N. Haji-Ghassemi, B. Jung, and H. Ben Amor. Inferring guidance information in cooperative human-robot tasks. In Humanoid Robots (Humanoids), 2013 13th IEEE-RAS International Conference on. IEEE, 2013 124–129.
- [14] A. Robotics. This is NAO humanoid robo Documentation, <http://doc.aldebaran.com/1-14/> 2015.