TAMPERE UNIVERSITY OF TECHNOLOGY

Arash Rezaei

Force Control of a robot manipulator using a real-time PC-based Controller

Master of Science Thesis

Examiners: Professor Kalevi Huhtala and Dr. Reza Ghabcheloo
Examiners and topic approved by The Faculty Council of the Faculty of Engineering Sciences
14th August 2013

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Program in Machine Automation
Arash Rezaei: Force Control of a robot manipulator using a real-time PC-based Controller
Master of Science Thesis, 46 pages
2014
Major: Mechatronics
Examiner: Dr. Reza Ghabcheloo, Professor Kalevi Huhtala
Keywords: Realtime Systems, Robot force control, control in robotics, Real time operation systems, realtime linux, Interface programming of mechatronics devices, Force torque sensor, Force control strategy, Fast prototyping methods, Code generation methods

This master thesis is based on an autonomous wheeled mobile manipulator project, directed in Tampere university of technology (TUT). In the beginning, we carried out a comprehensive study on the linux-based real-time operating systems, meanwhile existing limitation in the real-time operating system and real-time consideration in the programming of the control application has been surveyed in more details. Then, we studied force control strategies and control theories of the manipulators under contact forces. After that, manipulator dynamic equations and internal motion control loops were simulated using Simulink environment of Matlab. In this step, three scheme of controller with different area application were designed and was simulated on the manipulator Simulink model. The implementation part is based on the rapid prototyping method using Matlab/Simulink code generation products, firstly, the interfacing software between the manipulator joint actuators and the main controller software which is run in the RTLinux Xenomai was developed. In the next step, a similar interfacing application was developed for acquiring force data from the force torque sensor. In the next step, force torque sensor was calibrated using a calibration method according to gravity effects, then calibrated data was provided as measuring(sensory) input of the force controller software. In the final step, force control loop was closed around the motion controller loop and stiffness control scheme was achieved.

PREFACE

This dissertation has been carried out at the department of Inteligent Hydraulic and Automation(IHA), Tampere University of Technology, Finland. I would like to specially thank Dr. Reza Ghabcheloo, my supervisor, for his help and guidance. Also, I would like to thank Professor Kalevi Huhtala for his supports. None of the text of the dissertation is taken directly from previously published or collaborative articles.The implementation methodology in chapter 4 is proposed and developed by Mr. Reza Oftadeh, researcher at IHA and the inverse kinematic of the Schunk manipulator was derived by Mr. Mohamad Mohammadi Aref, researcher at IHA.

I would love to thank specially to my best friend Mr. Reza Oftadeh and Mr. Mohamad Mohamadi Aref for their supports. My thanks also go to my family for their supports and encouragements through love and affection.

Finally, I would like to dedicate this thesis to my lovely fiancee Narges.

Arash Rezaei

2014 Tampere, Finland

CONTENTS

LIST OF FIGURES

NOMENCLATURE

CAN Control area network

DMA Direct memory access

DOF Degree of freedom

FTM Force Torque Sensor

$GPOS$ general-purpose operating system

IFC Indirect Force Control

ISR Interrupt service routine

J Jacobean of end effector

J_o Jacobean to angular velocities

J_p Jacobean to Linear velocities

P_E Desired position of the end-effector

q_i Joint value

R_E Desired orientation of the end-effector

RT real-time

$RTOS$ Real-Time operating System

TCP Tool Center Point

FTM Force-torque sensor module

1. INTRODUCTION

Wheeled mobile robots nowadays attract attention of robotic research groups all over the world. Many new applications of this category of the autonomous robots have emerged in last decades. As a result, extensive studies of mobile robots were directed in research centers and technical universities. This thesis is directed in according with design and fabrication of a wheeled mobile robot project called *iMoro* directed in TUT. This mobile platform has 8 degree of freedom and also is equipped with a light weight arm with six degree of freedom. Moreover, different type of sensors like encoder and laser scanner, IMU and camera are installed on the robot to increase the autonomy capability of the robot. The robot is also equipped with an embedded pc with a Linux-based realtime operating system to provide a platform to implement control algorithms. All the sensors and the actuators of the robot are connected to this embedded pc with various interface such as *control area network* (CAN) bus and IEEE 1394 bus.

This thesis is divided into chapters. In the first chapter, we will describe the realtime operating system and then the realtime concept of a control system is studied in details. In addition to the concept of realtime systems, we briefly examined the realtime operating system that has been installed on the embedded pc. Then, the software architecture of the control system and interfaces applications are described.

In the second chapter, interaction between the robot manipulator and environment is studied briefly. The interaction of the end effector and the environment is the main subject of this study. The exerted forces due to contacts are measured by means of a force-torque sensor module and acquired data are sent to the control unit through a CAN bus and then an appropriate action is done according to the measured values. In this part, force control strategies and variety of control methods are covered concisely.

In the third chapter, we will cover illustrate the implementation methodology and after that details of implementations of control algorithm will be described. Aftermath, we will describe the implemented software architecture and then, the force torque sensor module calibration process and underneath theory will be explained in details.

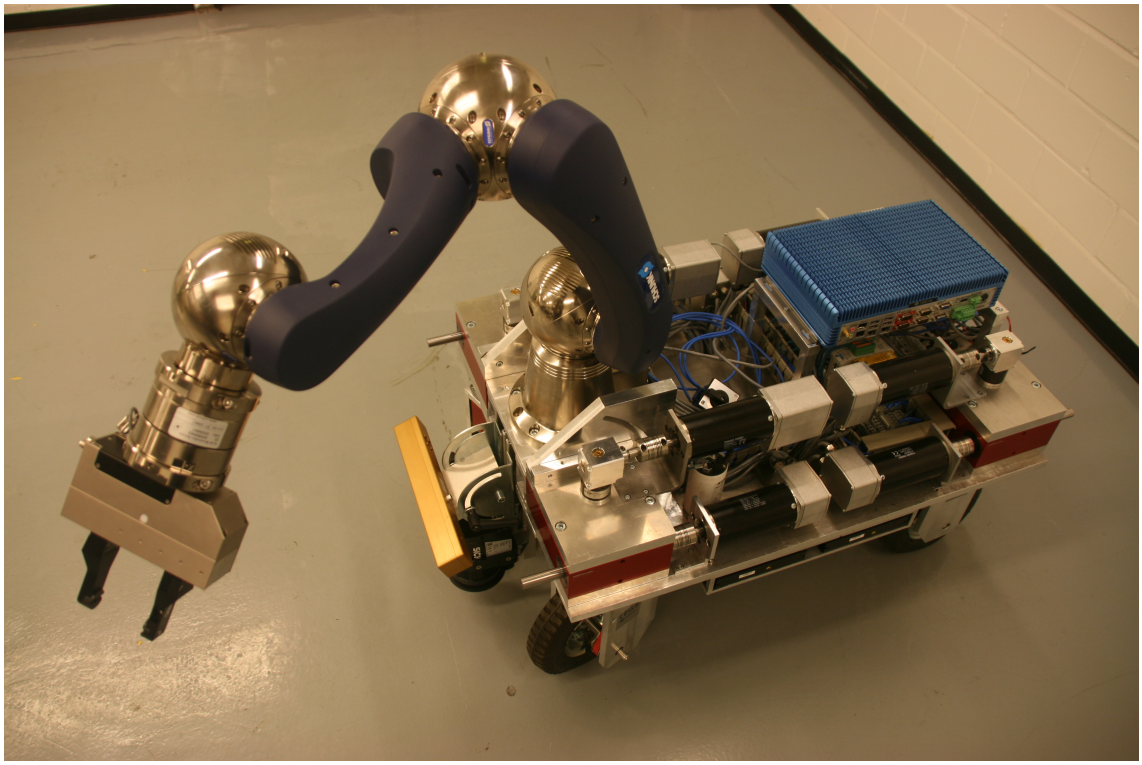

Figure 1.1: iMoro

1.1 System architecture

This thesis is based on *iMoro* project which is a four wheel steerable mobile robot with a 6-DOF robot manipulator installed on it [1; 2]. A force torque sensor module and a gripper are also installed on the end-effector of the manipulator. Mobile platform is consist of a rectangular rigid chassis with 4 legs on each corner. Each leg has two servo motors which control motion of the wheel independently. The first motor is for driving and produce movements in back and forth direction. In other words, it rotates the wheel around its rotation axes. The second motor is devoted to steering and it rotates the whole leg along its vertical axes. In fact, it changes the plane which wheel is rotating around. The manoeuvrability and motion control of the mobile platform are presented in [3; 4]. Motor actuators are Maxon servo EC Motors which are controlled individually by control drives called EPOS2. These drives provide both position and velocity control and their internal gains can be tuned accordingly. Each motor has an encoder on its own which feeds the internal control loop of the servo motor. In addition, there are two encoders installed on

the end of the wheel's shaft to provide extra measurements for wheels position and velocity. System architecture of the whole robot is depicted in figure (1.2).

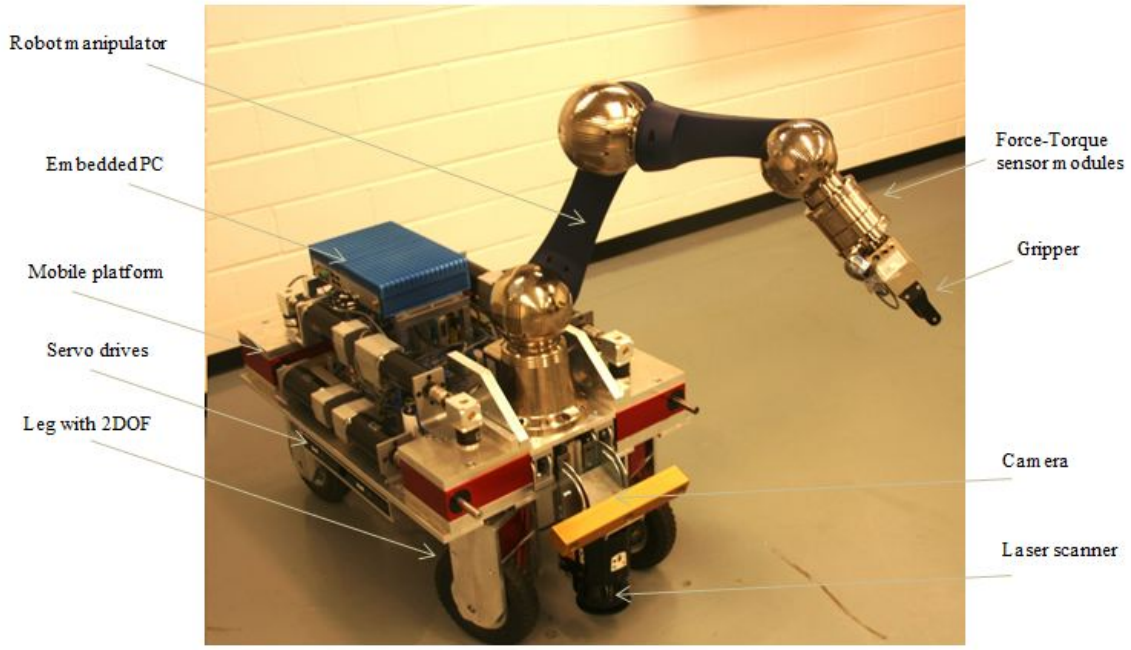

Figure 1.2: System architecture

As it is shown in Figure (1.2) a SCHUNK arm with 6 degree of freedom has been installed on top of the platform. This arm has been equipped with a gripper and a force torque sensor module as two separate modules. These auxiliaries modules are installed on the end effector of the manipulator to extend the application area of the robot. All drives of the mobile robot chassis and additionally all manipulator's drives and modules are connected to an embedded pc through the CAN bus networks spread all over the robot. This embedded pc plays the role of a control unit of our mechatronics system. All the measured data is passed to the embedded pc and the corresponding commands are sent through CAN networks to all the actuators and drives. In the following sections, we will give more explanation over embedded pc and data bus networks.

Embedded pc and CAN Network

The embedded pc, that has been used in this system is a Fan-less Tank-720-Q67 which is equipped with an IntelÂ® Core i7 processor. Furthermore, an Intel VGA Card and an On-board 2GB DDR3 memory and also one DDR3 SO-DIMM slot have been installed on the main board of the embedded PC. In addition, 8-channel audio/video capture with couple of USB 2.0 and 3.0 ports beside one SATA 6Gb/s

port , and 2 PCAN-miniPCI Dual Channel Card have been included into the embedded pc. Moreover, two dual-band Wi-Fi for high speed wireless transmission plus a redundant dual DC power input (9V 36V) is set on the PC. As software point of view, a special operating system consist of a realtime framework cooperating with a linux kernel is installed on th PC. The realtime development framework is called Xenomai and it provides hard realtime support and processing time requirements to user-space applications run on this machine. For further information, we encourage you to read section (2.1) which is devoted to the details of the realtime system and the deterministic solutions in computer systems. The embedded PC Tank-720-Q67 is shown in figure (1.3).

Figure 1.3: Tank-720-Q67

As we mentioned earlier, CAN buses has been used to connect most of mechatronics devices of the system to the embedded pc as the main controller. All the actuator and sensor units such as motor drives of the mobile platform, manipulator joint derives as well as force torque sensor module and gripper boards have been connected to the embedded PC through CAN buses spread all over the system. CAN bus is a message-based protocol which had been designed at Robert Bosch GmbH for automotive and truck applications in 1983. CAN is a multi Node serial bus standard for electronics control unit connections, each node sends and receives messages containing specific information about sensory data and state mode of the control unit. Each sent message in the BUS have an identifier and bunch of bits devoted Data. Each Nodes in the CAN network must consist of a *Host processor, CAN controller* and a *transceiver* to establish a CAN communication. Host processor is basically the logical control unit that decides what the taken messages mean and what messages should be sent. CAN controller receive and send to the bus serially and the received messages are immediately fetched by host processor. Finally, transceiver transforms signal level from bust to the CAN controller and vice versa [5]. Data link layer and physical layer used in the CAN protocol are based on the ISO 11898-1 standard. Other protocol layers are arbitrary and can be based on the system developers choice. *Application layer, object layer, transfer layer, physical layer* are the main layers of CAN protocol which each of them has specific duties.

For instance, message filtering and status handling are done by the object layer of the CAN protocol. The physical layer and electrical details such like voltage and current levels have been specified in ISO 11898-2:2003. *CANopen* is a communication protocol which implements the network layer of this protocol. To have a better understanding of CAN protocol, readers are encouraged to read [6].

Generally, force control applications demand a heavy traffic of CAN packets in the CAN bus. Each individual Bus has a master host node which is basically the micro controller's transceiver of the CAN cards in the embedded pc. Additionally, there are couple of device nodes in each individual bus and as a matter of fact, the number of needed packets is a criteria to aggregate bunch of devices in a single bus. Furthermore, developer should consider the fact that the number of nodes are also dependent to the length of the wirings. In our case, three separate CAN bus has been deployed in the system. Each bus is devoted to a number of devices (nodes) in the system. Due to limitation for ratio of the number of nodes to the packet traffic, a baud rate of 1 Mb/s has been set, which was actually the highest possible option.

The first bus is devoted to the four legs of the mobile platform. Since each legs have two distinct servo motor drive, so there is eight nodes in the first CAN bus. The second bus is devoted to the six joint drives of the Schunk robot manipulator and the third bus is devoted to the force torque sensor module and the gripper. The arrangement of the robot CAN buses is depicted in figure (1.4).

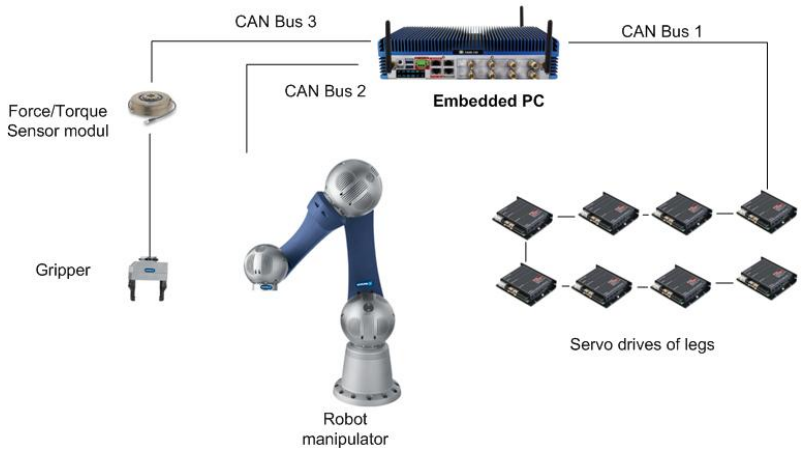

Figure 1.4: CAN Bus arrangement

Robot Manipulator

The installed manipulator on the iMoro is called Powerball Light Weight Arm LWA 4.6 and is a product of the SCHUNK GmBH. The payload is up to 6 kg and it needs a 24 V DC power supply. All the integrated control and drive electronics

are placed inside three ball shaped joint called *Powerball ERB* and each powerball provides two perpendicular axes of movement. A powerball diagram and its included elements are sketched in the figure (1.5).

Figure 1.5: powerball diagram

This actuators are connected to each other through internal CAN buses of the robot manipulator. Moreover, all the cabling for power supply and CAN bus are Internal and are done using especial design of SCHUNK GmBH company. This light manipulator with only 12 Kg weight is designed for mobile application as well as handling operations.

2. REAL-TIME SYSTEM

preview

In this chapter, firstly we will give a brief description over the embedded system and introduction to the realtime constraints which all have been faced during Implementation. Then, realtime operating system and their application in mechatronics system will be illustrated. Furthermore, Linux and its capability to be a realtime operating system will be surveyed. At the end, a widely used realtime linux called Xenomai and its architecture will be described in details.

2.1 Embedded systems

Nowadays, computer systems are found in every aspects of human life. In many cases, they are not even noticed by operator that a computing unit has been included into the device. The term *embedded system* is utilized for those system which are a combination of computer hardware and software as well as a number of mechanical and electrical elements incorporated into the system. Embedded systems are designed to carry out specific jobs. On the other hand, general-purpose computers have the potential to serve in different applications. General-purpose computers are used extensively in our daily lives. By the advent of world's first microprocessor namely 4004 chip which has been introduced by Intel in 1971, an evolution in computation devices had happened. This chip has been designed in a way that it could read and execute a series of instructions stored in the external memory chip. In fact, it was this instruction which was giving the device specific features. This product of Intel and its successors gain lots of attraction from designers. Especially, because of their applications in the aircraft and automotive control systems, developers brought embedded PCs in every subsystems of their projects [7]. Nowadays, there exist plenty of the small embedded systems, they contain a small size application or at most a moderate size one. In this dissertation , we are going to cover bigger embedded devices which typically use RTOS in their structure and have many applications running simultaneously.

2.2 Introduction to realtime systems

One subclass of embedded systems are *realtime systems* and in computer science disciplinary, it is referred to a system that has time constraints while data processing. In fact, they perform the calculations in a timely manner. These calculations should be done in a specific point in the future. We call these points *dead lines*. If the embedded PC misses any of these deadlines, it is a failure. Only Imagine, such a missed deadlines happens in the auto pilot control system of a landing aircraft with hundreds of passengers on board. This wrong answer could cause a catastrophic incident and puts the passengers lives in a hazardous situation. Likewise, in the robot surgery applications, any missed time-constraints may cause a fatal accident for the patience whom under the robotic surgery is [7]. In other word, in the realtime systems, whatever the computations and their results are, they must be done not only in proper form but also on a specific point in the time. Few beginners in programming think that the realtime concept means fast performance and it demands a super fast hardware for calculations. However, a true realtime application can be performed by a slow and very cheap hardware, which has been designed based on the realtime considerations. For converting a normal embedded PCs to a realtime one, we need an operating system capable of performing controls over processing and time management. In fact, realtiming guarantees the exact execution time of the critical operations, but not the speed of the operations. In practice, it is possible that the a realtime operating system scarifies the calculation results in favor of time constraints to reach a deadline [8].

2.3 Realtime operating systems

In general, an operation system provides principal services to the applications running on a computational platform. For instance, memory and process management as well as file systems are parts of the services which any operation system usually provides for the user applications. In addition to what just mentioned, facilitating programs execution service such as process creation and process scheduling is another crucial service, that the operation systems generally should provide. The way how a program requests a service from kernel is called *system calls*, they facilitate interfacing between processes and the operating system.

A computational platform alone, is not a practical controller in an embedded system. In fact, these are peripheral device of the PCs that enable PCs to receive data from the environment (sensor) and send them commands for performing an action

(actuator). Sensors or actuators of the robot in an embedded system are considered the peripheral devices or *I/O Devices*. Various vendors and companies all over the world produce these interfacing devices. These companies apply standard communication protocols in their products to facilitate a standard communication between devices with different vendors. Standard data buses connect these peripheral devices to the board cards included into the embedded PCs. These electronics board cards receive the analogue streams of data in low level of physical layers and transform them in the understandable digital streams to the higher levels of communication.

In other words, electronics cards supply 0 and 1 data for CPU and internal data bus of the PCs, in a real world which every phenomenon is inherently analogue. In figure (2.1), you can see how board cards are connected to the outside world.

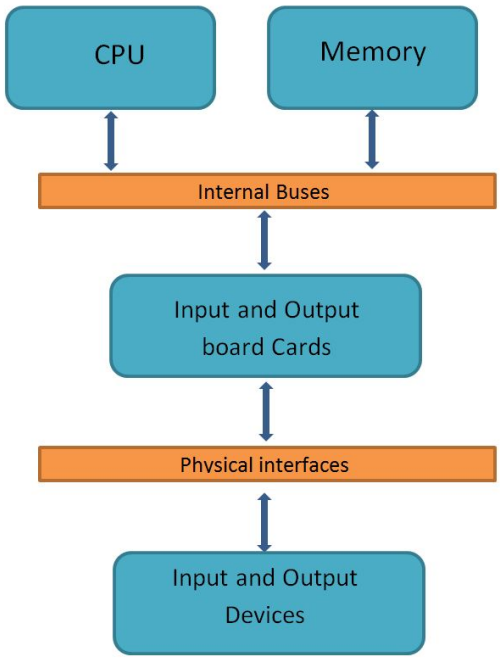

Figure 2.1: I/O commutation

Additionally, hardware related services are part of the operating systems duties based on programs demands. Example of such a services are write access to the printer and display screen or reading from the interfacing devices like keyboard or mouse and even communication bus board cards. These sort of service are provided by the system's kernel which is consist of hundreds of control algorithms. For instance, when an application wants to print some log information for the operator on the screen, it requests a *system call* from the operating system's kernel. Then, it is the duty of the system's kernel to communicate with the hardware and printing the related information on the screen. In the case of network applications like CAN communications, operation system commands CAN card to send the CAN packets

in the CAN Bus.

In the software point of view, these are *device drivers* that enables operating system to communicate with device cards. They make a hardware to react to a defined internal programming interface commands. Generally, device drivers are black boxes that hides the device internal operations to the users. In fact, they map some standardized system calls to the device specific operations. Software engineers who work in hardware manufacturer companies usually develops these device drivers because they have better understanding of their hardware design [9] than system developers. Moreover, vendors usually provide These device drivers for well known and popular operation systems not for uncommon or scares operating systems which may be employed by specific designers. In the case of using an uncommon OS, developers are obliged to compose the device drivers themselves. This would impose an extra burden of overwhelming low-level programmings to the developers.

2.3.1 Linux operating system

To avoid these kind of problems, developers prefer to employ well known and supported operating system. Linux is a free and open source Unix-like operating system with the capability of conversion to the realtime operating system. At first, it has been designed for the Intel x86-based personal computers. Linux developer aimed to make an OS which could show acceptable performance under constant load conditions and had the ability to handle plenty of users in the same time. Linux as an open source operating system has lots of attributes along with its vast number of users and developers. These reasons would tempt designers to employ this practical OS in their designs. Due to these essential features, it is widely utilized in many robotic applications. Nowadays, even micro-controller versions of linux are used on many devices run by microprocessor. Moreover, annually hundreds of stable distributions of linux are released in the OS market. Each distribution is based on *linux kernels* which are released annually by linux community. These distributions are customized according to specific usability and special users. Due to its efficient performance, it is a popular operating system in community of meachatronics designers and system developers. Furthermore, linux community provides new device drivers form vendor companies as well as application software almost immediately. This features enable system developers to employ new sensors and devices with various interface in their design without the need of spending lots of time and effort on the implementation details. Normal linux kernel architecture is depicted in the figure (2.2).

As previously mentioned, a mechatronics system must react to the sensory inputs in a timely-deterministic manner to avoid catastrophic accidents. Let's check that if a standard linux distribution can achieve this exigent goal of operating system in

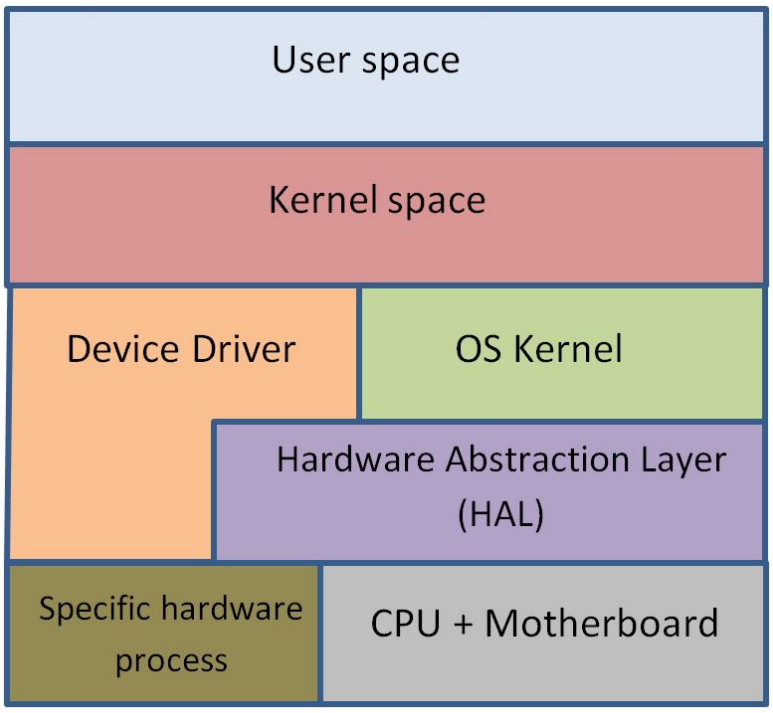

Figure 2.2: Linux kernel architecture

the mechatronics schemes. To this end, number of elementary concepts are pointed in the following paragraphs.

In standard linux, the processor which is executing a normal task, are warned from high-priority conditions by *interrupts*. In fact, interrupts inform the processor that it is required to cut off current code execution and advert to another event which needs attention. Equivalently, each interrupt has its own interrupt handler which is generally stored in the kernel core code of the operating system. If an external device emits such a signal, it is called hardware interrupt and if an executing program emits it, it called software interrupts.

To measure how much a realtime system is accurate, operating system developers introduce a concept that is called *latency*. It is the delay between occurrence of a triggering event and the time that the corresponding program begins to handle it. For example, *interrupt latency* is the delay that may occur between an interrupt and its correspond interrupt handler execution. There are many operations such as bus connection, cache misses, direct memory access (DMA) which can cause this delay to occur. To have a better understating over latency, imagine that after an interrupt handler awake a task to be run, it may take time for the kernel to complete its ongoing duties and concede the CPU to the task, this delay is called *scheduling latency* [8].

In figure (2.3), two kind of latency is demonstrated. The first one is the time between the moment which interrupt is sent and the moment which interrupt service routine (ISR) is awakened and the other one is to the moment in which the task in the charge of these events is practically resumed back.

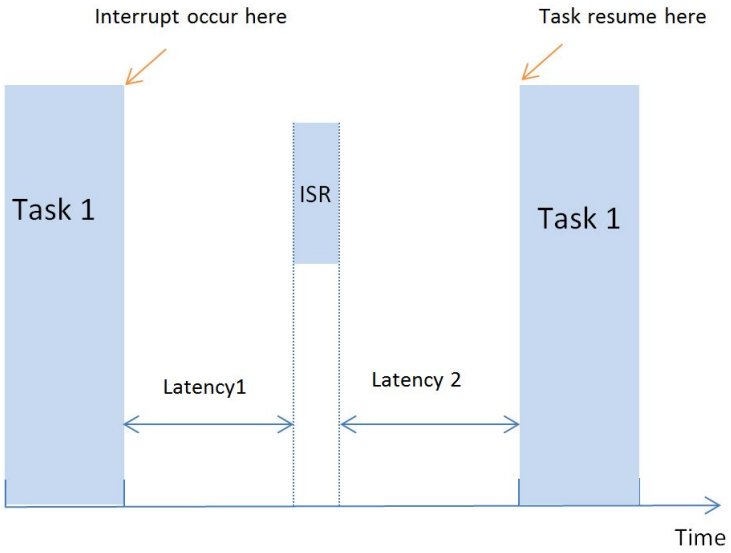

Figure 2.3: Latency

As it had been mentioned before, realtime applications respond frequently to the device interrupts to handle data acquisition process. To have a better understanding of this interaction, consider the definition of *Worst-case interrupt latency*. It is the longest time that it is needed for an OS kernel to run an interrupt handler software corresponds to its pending interrupt. Before designing any realtime software, it is significant to know that how high and accurate is this latency value for the target platform that the realtime application is going to run in [8].

We should also consider, while the standard subsystem of linux like virtual memory management, virtual file-system layer and the networking stack are performing their tasks, linux kernel masks the external interrupts for an unlimited amount of time. In fact, non-realtime linux do it to prevent running concurrent tasks on the very same processor from attaining non-shareable resource. Therefore, it is possible to say standard kernel prefer to keep a better throughput over responsiveness. Throughput-oriented design is best suitable case for the general-purpose operating system (GPOS). However, as we know a realtime system must be predictable and as it just mentioned, the potential interrupt latency in the standard linux kernel is not acceptable for realtime application [8]. As it is widely known, linux is a GPOS which has been initially designed to share resource fairly among the users.

Furthermore, today's realtime applications are constantly in interaction with non-realtime software and OS services. Furthermore, any mechatronics designs demand couple of conventional OS services such as networking capabilities as well as richness in implemented hardware driver which are inherently features of a GPOS. Traditional RTOSes don't encompass these features and designers must think of some ad hoc solution for them. Based on what has been mentioned above, system designer tried to add realtime capabilities to a viable GPOS like linux. In the following section, the concept of the realtime linux (RT linux) and possible architectures are surveyed.

2.3.2 Realtime linux

Actually, linux alone is not a realtime system, albeit it is possible to install a realtime micro kernel beneath linux kernel and run the realtime applications on it. In this case the realtime kernel has a higher priority than the normal linux kernel. In RT linux, realtime programs are made by creating realtime threads which are also called *realtime tasks*. Moreover, non-realtime programs are put in normal linux threads. Realtime micro kernel puts realtime tasks in the execution line and when there is no realtime task on the line of execution, normal linux programs are allowed to run. Actually, normal linux kernel is just an idle task for the realtime scheduler and only when the realtime kernel scheduler concedes the CPU to linux, it can execute its own scheduler and manage its own normal processes. Whenever a realtime task is ready to run, normal linux process is preempted by the realtime scheduler and the realtime task is executed immediately.

Mechanisms of interrupts in realtime linux

Basically, in multitasking computing, interrupts flow is a common technique for rescheduling higher priority tasks. They can be considered as the computer heartbeats. Schedulers as process management unit of the kernel and is driven by the timer interrupts from an internal clock in certain periods. A well-granular interrupt timer makes the system more accurate and provides more rescheduling point to handle the realtime tasks.
There could be some case where a normal linux thread is on execution line and on the same time a new interrupt which is corresponding to a waking realtime task arrives. At first, an interrupt handler set by the realtime kernel handle the interrupts and it awakes the realtime task corresponds to interrupts. Then, the realtime

scheduler which is also called by interrupt handler code, perceive that a realtime task is ready to run and then, it sets the linux kernel to sleep mode and set the realtime task in execution line.
Furthermore, there are some weak point of interrupt follow concept. Consider that the realtime kernel preempt a higher priority task after receiving an interrupt from a lower priority realtime task. After kernel resume this low-priority task execution and concede the CPU to that, then it realizes that the lower priority task is on the execution. However, it can not immediately stop this task and switch to high priority one. Hence, High priority task has to wait until kernel reach to the next rescheduling point. Mentioned case is a common problem which is called *priority inversion* and is one of the most complicated problem in realtime programming subject. Sufficient rescheduling points make a realtime operating system victorious to insure a limited scheduling latency [8].

Providing many rescheduling points for a realtime kernel enables it to switch as soon as possible between tasks and activates high-priority task that needs attention. Needed time for this switch is called the *preemption granularity* and *worst-case dispatch latency* is the longest time that it takes for a kernel to accomplish a pending rescheduling. In past, rescheduling could be done only when kernel duties have been finished and it was out of kernel context. In other words, while kernel was performing activities in order to accomplish a demands from an application or a system call, it wouldn't stop anything to resume a high-priority task to execution. *Robert Love* who is an open source software developer initiate a new approach in linux kernel which enables it to be preempted while performing its internal duties, this approach is called *preemptible kernel support*. It has been included into linux 2.6 as a standard feature [8].

Considering what has been mentioned above about possible mechanism of a RT linux, numerous versions of RT linux are suggested during last decade. Three most important, common and free RT linux-based OS are as follow:

1. Realtime Application Interface (RTAI).
2 .RTLinux.
3. Xenomai (It is basically a new version of RTAI linux)

In the following section, Xenomai operating system is surveyed with more details and some information over Xenomai skins, scheduling mechanisms and realtime object are given.

2.4 Xenomai Project

The Xenomai project has begun in August 2001 to facilitates porting of the realtime industrial program to a free linux-based operating system while keeping realtime constrains. Gradually, it adds supports of different computer architecture in embedded platforms such as PowerPC32, PowerPC64, Blackfin, ARM, x86, x86_64 and ia64. As it has been mentioned in the previous sections, this OS is based on a small co-kernel patched into a standard linux kernel and runs beneath it on the same computer hardware.

Scheduler, realtime objects, services are the components that has been built into the Xenomai's kernel on the top of a nucleus. Nucleus provides a set of traditional RTOS services for realtime objects which has been implemented into the Xenomai. These objects are basically kernel constructs such as *tasks, semaphores and message queues* which assist developers to create realtime applications [10]. On the top of the Xenomai nucleus, number of generic building blocks called *skins* are provided. These blocks provide traditional APIs of the RTOSes and they include the specific interfaces to the realtime objects which has been created by the realtime framework.

The following skins on the generic core of Xenomai are included :

- POSIX
- pSOS+
- VxWorks
- VRTX
- Native
- uITRON
- RTAI

Now, this question may arise that why these skins are introduced in the Xenomai's structure?

Generally speaking, while importing realtime software application codes to a RT linux, the main columns of these software applications do not experience substantial changes. Specifically, realtime APIs of RTOS which are considers these main columns of a realtime application software. In fact, a realtime API does mean certain specific semantics and represents particular behaviors in an application. Furthermore, action line of these applications are often depend on the structure of these OS API. In other words, whole operation of an application could be manipulated if a set of similar APIs from another OS is used instead of original ones. For instance, two realtime APIs from two different OS may be similar superficially. Although,

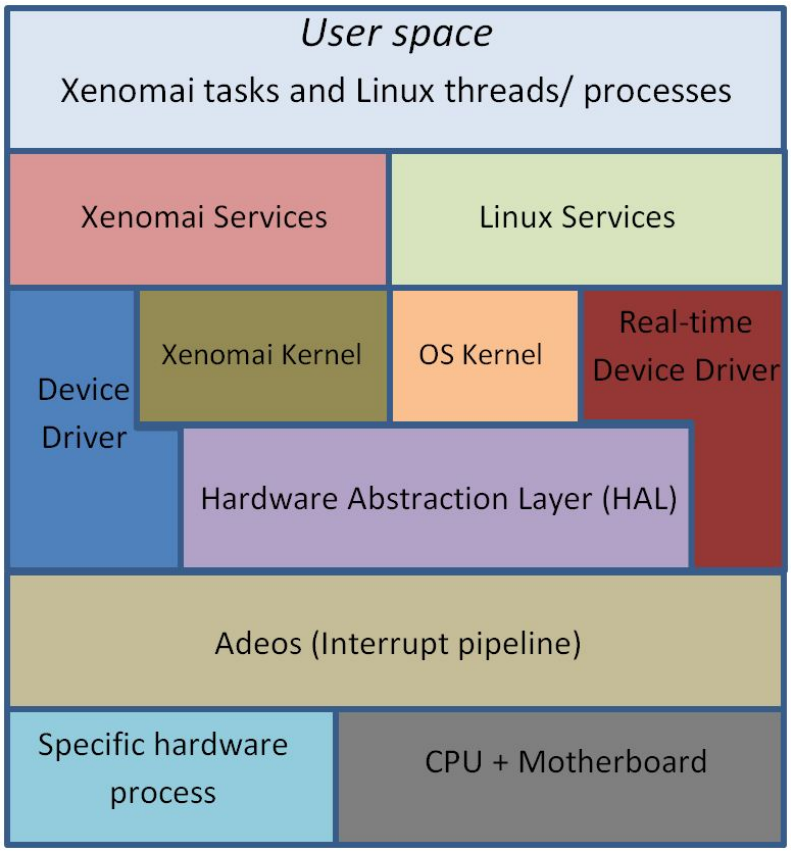

Figure 2.4: Xenomai Architecture

they still have some subtle differences which could change the way applications actually behave if the original system calls have been simply mapped to their closest equivalents in the target environment [8]. In figure 2.4 Xenomai Architecture is depicted.

3. FORCE CONTROL OF MANIPULATOR

preview

At the beginning of this chapter, we present robot manipulator applications in the modern industry and then the control models and dynamic of manipulator is covered. Then, various force control strategies are described briefly and at the end, the compliance control model is described in more details.

3.1 Industrial robot manipulator

In last decades, robot manipulators are one of the crucial devices which have been used in the automation lines of a modern factories. Fast production rate and low maintenance expenses caused vast utilization of these robots in the industry. Due to the advances in technology and evolution of the sensors and advent of the modern sensor (e.g. force torque sensor) as well as emerging of more powerful computation units, the implementation of more advanced operations like contact tasks are feasible now. It is worth mentioning, that the contact tasks are those in which the robot manipulator either grasps or pushes a workpiece in the work cell or perform an operation in which an specific amount of force or torque are needed. For instance, grinding, assembling and deburring are the tasks where the mechanical interaction between robot manipulator and environment is an important issue. Furthermore, modern robots must have the ability to work more closely with the human and even in the case of medical robotic application, they sometimes are employed to interact with the human soft body tissues.

Control of the robot manipulators demands a good understanding of their mathematical models. In the following sections, motion control models as well as dynamical models of the manipulator is studied to clarify the underlying concept of the force control strategies.

3.2 Motion Control Models

As we know, the robot manipulator is composed of many segments, such as links,

joints, actuators, sensors, data buses, power supply, actuator drives and the main controller which could be an embedded PC equipped with a RTOS. The mechanical structure of a manipulator comprises a sequence of links which are connected through joints. According to the desired planed task, a tool or sort of end-effector is attached on the tip of the manipulator. Joints are independently driven by means of motor actuators to provide maneuverability and flexibility for the manipulator to perform complex tasks in a spatial space. Joint values are measured by the encoders to enable the implementation of a servo control system in each individual joint. There is a kinematic relationship between the position and orientation of the robot Manipulator's end-effector and the joint values.

These equations are usually dependent on the relative geometrical situation of the joints in the space and also the length of the connecting links. Whenever the joint values are known, the end-effector position and orientation can be calculated by substituting the joint values into the *forward kinematic* equation.

$$P_e = P_e(q) \tag{3.1}$$

And for the orientation we have:

$$R_e = R_e(q) \tag{3.2}$$

where q is a (6×1) joint value vector and P_e is the desired position which in fact, is a (3×1) vector. Additionally, R_e is a (3×3) rotation matrix describing the origin and the end-effector desired orientation frame.

On the other hand, whenever an end-effector position or orientation are desired, the *inverse kinematic* equation can be solved to obtain the joint values equivalent to the desired position. In the latter case, there could be multiple solution which one of them needs to be selected by the robot controller based on the initial configuration of the manipulator. Generally speaking, keeping tool center point of a robot on a desired path is called *motion control*. In the case of no contact force, the customary way of solving such a problem is to find the desired joint values by solving inverse kinematic equation according to the given end-effector position and orientation in the space. After that, desired joint values are sent as the input to the designed controller of the joint actuators. This strategy is called *kinematic control* [11].

To obtain the governing equations, we need to define the following conventions. Let's assume that $\dot{q}$ denotes the time derivative of the joint values, $\dot{P}_e$ denotes the (3×1) linear velocities of the end effector and ω_e denotes (3×1) rotational velocity vector of the end effector. So, we can show the *end effector velocity* of the robot with v_e and define it in the following equation.

$$v_e = \begin{bmatrix} \dot{P}_e \\ \omega_e \end{bmatrix} \tag{3.3}$$

Joint values derivatives could be separated from the left side of the above equation and therefore, it can then rewritten as below:

$$v_e = J(q)\dot{q} \tag{3.4}$$

Where J is the $(6 \times n)$ *Jacobian* of the end effector which is dependent to joint values and the length of the manipulator's links and type of the connecting joints. This can also be written as below.

$$J = \begin{bmatrix} J_p \\ J_o \end{bmatrix} \tag{3.5}$$

By this, we can separately study the contribution of the joint space velocities to the linear and angular velocities of the end effector in the Cartesian space. Where J_p is the Jacobean effecting on the linear velocities and J_o is the Jacobean effecting on the angular velocities.

Singularity

By obtaining the Jacobean matrix, we have linear transformation which relates the joint velocities to the cartesian velocities. Now, let's consider the inverse relationships of the equation (3.4) which leads to the following equation.

$$\dot{q} = J^{-1}(q)v_e \tag{3.6}$$

As you can see above, in equation (3.6), the inverse of the Jacobian is utilized to calculate th necessary joint velocities to accomplish the desired cartesian velocities. In such a situation, a question could be posed that if the Jacobian matrix is invertible for all value of q? on the next step, we calculate the q vectors on which Jacobian matrix is not invertible. Such a q is called singularities point of the robot which are inherently dependent to the robot architecture [12]. There are two different type of singularities, the first type is called **workspace-boundary singularities** which happens at the end of the workspace limits of the robots when the arm is completely stretched out and the other one is called **workspace-interior singularities** which usually happen when two or more joint axes are lining up [12].

3.3 Dynamics of the manipulator

So far, we have focused only on the kinematic considerations of the manipulators. In this stage, we also consider the external forces applied to the manipulator as well as the torques applied by the actuators, in motion equations . There are two type of problem in the study of manipulators dynamics, the first one which is called *Inverse Dynamics* and is when a trajectory of points and velocities are given and joint torques are required. The other type of problem is called *Forward Dynamics* and is the case when the joint torques are given and the resulting end effector motion variables are desired[12]. It is worth mentioning that, the solution to the first problem possess a crucial importance in control point of view.

There are two approach in deriving the dynamical equations of a manipulator, the first one is the *Newton-Euler* approach which is based on two basic dynamic laws namely newton's second law and Euler formula. It is based on the analysis of forces and moments of the links. The other approach is called *Lagrangian dynamic formulation*. This approach is base on a energy point of view. The final results of the both approach are similar, however as it is widely known the Lagrangian dynamic formulation is briefer and faster[12]. In the following paragraphs the Lagrangian approach is illustrated in more details.
As it has been mentioned before, Lagrange equation is derived from the potential and kinetic energy calculations of the mechanical systems. It is shown in the following equation.

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = \xi_i \qquad i = 1, ..., n \tag{3.7}$$

where,

$$\mathcal{L} = \mathcal{T} - \mathcal{U} \tag{3.8}$$

ξ_i is a general force at the joints of the manipulator, type of this general force depends on the utilized joint type in the manipulator architecture. This general force is considered force in prismatic joints and torque in revolute joints respectively. Moreover, $\mathcal{L}$, the Lagrangian variable is the difference between kinetic energy and potential energy of the dynamic system that we consider.

Potential and kinetic energy of the system can be calculated based on the energy equations of the bodies. By substituting them into the equation (3.8) , the Lagrangian expression can be obtained. Then, by substitution of the Lagrangian expression into the equation (3.7) the dynamics equation of the manipulator is obtained. The general form of this equation is usually as follow.

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \xi \tag{3.9}$$

Where, B(q) is the $(n \times n)$ inertia matrix of the manipulator, ξ is a $(n \times 1)$ vector of the torques exerted on the joints, g(q) is the torques exerted on the manipulator caused by gravitational forces of the manipulator links and $C(q,\dot{q})\dot{q}$ is a $(n \times 1)$ vector of the torques caused by Coriolis and centrifugal forces.

Now, let's consider the elements of the general force ξ_i in the equation 3.7 in more details. ξ_i can be decomposed to the following elements.

$$\xi = \tau_i - \tau_{fi} - \tau_{ei} \tag{3.10}$$

Where, τ_i is the torque exerted on the joint i by the driving mechanism which it can be the combination of a servo motor with a gear or a timing belt or even in some specific cases only a motor and a coupling. τ_{fi} is the torque caused by the frictional forces in the joint i of the manipulator and the $\tau_{ei}$is torque on the joint i which is the effect of the external contact forces or the moments exerted on the end effector from the environment.

While modeling τ_f we only consider viscous friction for simplicity. By defining F as a diagonal matrix of the friction coefficients we can write as bellow.

$$\tau_f = F\dot{q} \tag{3.11}$$

Now, by applying the virtual work principal and based on the application of the Jacobean matrix, we can write the τ_e as follow.

$$\tau_e = J^T(q)h \tag{3.12}$$

Where $J^T(q)$ is the Transpose of Jacobean matrix and h is the external force and torque matrix which is defined in the following way:

$$h = \begin{bmatrix} f \\ \mu \end{bmatrix} \tag{3.13}$$

Where, f is a (3×1) force vector exerted on the end effector of the manipulator and μ is a (3×1) moment vector exerted on the end effector.

By substituting (3.12), (3.11) in the (3.10) and using the result to substitute in the dynamical model equation which has been developed earlier in (3.9), we will have a more general form of the manipulator dynamical model as follow:

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + F\dot{q} + g(q) = \tau - J^T(q)h \tag{3.14}$$

This equation is a basis for the further study of force control theories which are going to be described in the following sections.

3.3.1 Static model-based Compensation

Consider that our task is to move the robot manipulator to a desired position through controlling torques exerted by the motors at the joints. Note that, for the simplicity we only consider position, linear velocities and forces on the end effector. Orientation and rotational velocities and moment on the end effector is an option for further developments. To this end, let's denote the desired position of the end effector by P_d and the actual current position of the end effector by P_e. In addition, the *position error* is defined as below:

$$\Delta P_{de} = P_d - P_e \tag{3.15}$$

Now, using mechanical intuition, suppose that for moving the robot end effector toward P_d, we need to exerted on the end effector, an equivalent controlling force γ_p. This equivalent force can be chosen to a proper action proportional to the position error.

$$\gamma = K_p \Delta P_{de} \tag{3.16}$$

Now by considering the equivalent controlling force, the joint torques could be estimated as follow:

$$\tau = J^T \gamma_p - K_D \dot{q} + g(q) \tag{3.17}$$

Where J^T is Jacobean Transpose matrix, g(q) is the compensation of the static gravity torque and the term $-K_D\dot{q}$ is designed to improve the transient response of the system and provide damping torques which can be tuned by changing the gain values in K_D that is a $(n \times n)$ matrix. The control scheme based on the (3.17) is called *proportional derivative control with gravity compensation.*

Now, let's find the dynamical behavior of the system after closing the control loop using this scheme. So, by substituting (3.17) into (3.14) we will have the closed-loop dynamical response as follow:

$$B(q)\ddot{q} + C(q,\dot{q})\dot{q} + (F + K_D)\dot{q} = J^T(q)\gamma_p \tag{3.18}$$

As you may know, Stability is the most important issue when a controller is suggested. To check the stability of this closed-loop controlled dynamical system the following Lyapunov function is suggested:

$$\nu = \frac{1}{2}\dot{q}^T B(q)\dot{q} + \mathcal{U}_p \tag{3.19}$$

The first term in the equation(3.19) is the kinetic energy of the manipulator and the second term is considered as the potential energy of the manipulator which is related to the position error of the end effector and it can be shown as follow:

$$\mathcal{U}_p = \frac{1}{2}\Delta P_{de}^T K_p \Delta P_{de} \tag{3.20}$$

Then the whole system is stable with the proof given in [11].

3.4 Force control strategies

Controller of the normal industrial robot manipulators are usually programmed using a motion control scheme, this enables them to accomplish non-contact tasks like welding or spraying. For this kind of controller, one can define only desired trajectory of tool center point which has been pre-programmed before run-time.

However, In some applications, it is more important to precisely control the contact forces applied by the end-effector on the environment rather than controlling the robots positioning. In this type of applications, a force control scheme must be integrated to the motion control scheme. Experiences have shown when the contact forces exist in a robotic operation, using merely motion control scheme is not feasible. For instance, in the case of hard contact surfaces, a small position error can lead to a big growth in the contact force and consequently causing a deviation of the TCP from the desired position. On the other hand, the control system reacts to compensate this error, and ultimately this interaction can cause unstable oscillations in the system and finally leads to huge contact forces until the saturation of the joint actuators or even breakage of the parts in the worse possible cases[11]. In this situation, even apparently easy contact tasks like wiping a surface could be inherently complicated. [13].

This issue can be solved if a compliant behavior while interaction with the environment is employed. This could be accomplish in two different approach, passive compliant method and active compliant method. The passive one can be obtained by installing a soft and flexible mechanical device (e.g. made of some spongy material) on the end-effector in contact to the environment, and the active might be obtained by employing a way to measure the interacting force along with a proper control strategy [11]. Although, in some specific situation, if the interaction task have been accurately planned, it could be performed using only the motion control schemes. To this end, a precise model of both the robot manipulator and the interacting environment is required. It is notable that, modeling a robot manipulator is much easier job than modeling the interacting environment [11].

As it has been mentioned above, a proper force control strategy along with a way of measuring the external force are the keys to achieve a force control schemes. Basically in these schemes, the output force or torque which are applied by the end-effector on an external object, are controlled to the desired force value. In some other cases, whole the robot manipulator shows a desired behavior to applied force to its end-effector.

To measure the interacting force, a force torque sensor module (FTM) can be mounted on the robot manipulator, typically between the wrist and the end-effector. The exerted forces and torques on the end effector are changed from the contact surfaces to the digital signals by means of strain gages installed on the sensor internal structure. After that, they are usually sent to a data bus to be used to close the force control loop in a digital PC-base controller[11].

Furthermore, force control schemes are similar to the position control schemes, but the complication in force control comes from the fact that the robot needs to follow a specific trajectory in the space in the specific directions and meanwhile a force control strategy must be performed in other directions. For instance, when a robot is polishing a surface, a specific amount of force is needed in the direction perpendicular to the surface and at the same time in tangent direction of the surface a position control strategy should be hold. Therefore, in the reality, the force control is a hybrid force/position control.

3.5 Direct and Indirect Force Control

Force control strategies can be categorized into two different group. The first group is *indirect force control* and the other one is *direct force control*. In the first one, force control is achieved via an inner loop motion control, but without explicitly using closure of force feedback loop and in the second one contact force is controlled directly to achieve desired force vector, by use of a force feedback loop.

In fact, if a proper model of the interacting environment has been formerly obtained, control methods of the second category is a feasible and viable choices. To clarify that , *hybrid position/force control* method which lies in the second category can be depicted. This method is designed in a way that the position is controlled in the unconstrained directions and force is controlled along the constrained directions. In the planar surfaces, a switching matrix is employed to select the proper control approach(position or force control) according to the model of the surface, while in a general curved surfaces task, a definite constraint equations must be utilized to achieve a proper control action[14].

However, As it has been mentioned earlier, obtaining a precise and detailed model of the interacting environment is not a straightforward task. Albeit, there is still an effective scheme in the first category which is called *inner/outer motion/force control*

and can be used in such a situation. In this strategy, a force control loop is closed around a position control loop and in the case of no contact force (unconstrained task), the desired input motion of the TCP is the input signal of the inner motion control loop. Via this control structure, the force and motion control are active in a parallel way ,whiles the motion control section of the scheme is dominated by the force control section to ensure force control in the case of contact force [15], [16].

3.5.1 Indirect Force Control

Another type of force control strategy is achievable through motion control schemes and these strategies are basically grouped in the indirect force control category. For instance, *compliance control* and *impedance control* are achievable by using only motion control with some small modification in the structure. In this type of strategy the position error is directly associated to the contact force and torques through a group of tunable variable which define the behavior of the manipulator. In another words, these variable determine how a manipulator reacts to an external excitation. The general form of these behavior is identical to a dynamical system, composed of spring, damper and mass. These approaches will be described in more details in the following sections.

Compliance control

Active compliance control is categorized under the indirect force control schemes and it is also called *stiffness control*. As it has been mentioned earlier in the previous section, indirect force control schemes are based on the idea of controlling the force or moment through changing the position or orientation of the end effector. Moreover, it has been mentioned that we can design the controller in a way that the whole dynamical systems shows an specific behavior to the force excitations.

To illustrate this issue, it is worth reanalyzing the static model-based compensation control approach which has been covered earlier in the section (3.3.1). Consider the position error definition in the (3.15) as well as choosing γ in the (3.16) to substitute in the equation (3.17) which was the control action estimation of torques. The proper torque estimation will be as follow:

$$\tau = J^T K_p \Delta P_{de} - K_D \dot{q} + g(q) \tag{3.21}$$

Now by assuming steady state condition ($\dot{q} = 0, \ddot{q} = 0$) and substituting the control torque estimation into the system dynamical equation (3.14), the response of the system to this specific control action will be as follow:

$$J^T K_p \Delta P_{de} = J^T f \tag{3.22}$$

Now, let's assume that the Jacobean is a full-rank matrix, therefore we can rewrite the equation (3.22) as below:

$$\Delta P_{de} = K_p^{-1} f \tag{3.23}$$

The equation(3.23) indicates that the manipulator in the steady state condition, responds to the contact forces on the end effector similar to a spring and make a position error correspond to the exerted force. K_p^{-1} is called *Active compliance* and one can tune its element to change the stiffness of the manipulator springy behavior and ensure the compliant response of the controlled system. Using the equation (3.15), we can rewrite the equation (3.23) by the force as follow :

$$X_e = K_p^{-1} f + X_d \tag{3.24}$$

Control system architecture of the compliance control is depicted in the figure(3.1).

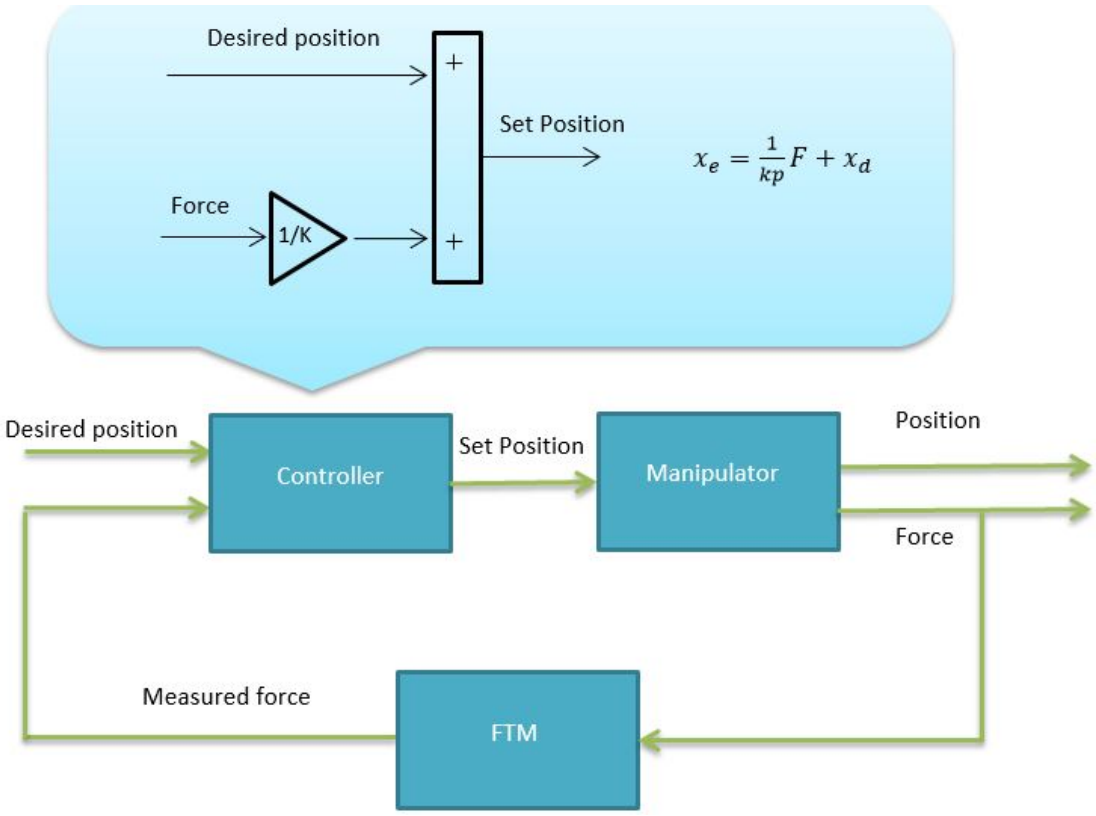

Figure 3.1: Compliance(Stiffness) control architecture

The properties of the environment surface can influence the selection of the element of K_p^{-1} . In this project stiffness control scheme has been implemented on a Schunck robot manipulator. In the following chapter, we will describe how this control method has been implemented in the robot manipulator.

4. IMPLEMENTATION

Firstly in this chapter, we are going to describe the simulation scheme of the control strategy which has been illustrated earlier and then, the methodology of implementation is depicted briefly. Finally, the implementation process of the control theory which has been mentioned in the previous chapters will be described in more details.

4.1 Simulation of controller performance

After a precise study on the theories of the manipulator force control and describing the conventional strategy of solving the force control problems in the robotic world, some of these control schemes have been simulated using the Simulink environment of the MATLAB. This will help us to anticipate the behavior of the force controlled manipulator under the different excitation contact forces. To this end, firstly we need to model the dynamic behavior of the servo controlled joints of the manipulator mathematically. Namely , the response of the joint control systems to a change in their desired position has been modeled as a conventional second-order dynamic system with a mass, spring and a damper. By tunning these variables, the desired behavior of the manipulator such as response time, rise time and bandwidth are achievable. In the next section compliance controller is simulated.

4.1.1 Simulation of compliance controller

After obtaining a rough mathematical model of the joint controlled manipulator, other elements of a force control system are modeled. To this end, excitation force and motion control elements are also modeled mathematically in the simulation software. Now, by placing previously designed compliance force controller from equation (3.24) into the modeled system, we can simulate the behavior of a compliance controlled manipulator under different forms of excitation force. As it has been illustrated earlier, the output vector of a compliance controller is in the cartesian space and the input to a robot manipulator motion controller is vector of desired joint values. Moreover, we consider that we need to send joint value commands to the servo controllers in an specific manner, to this end, a *motion planner* block is usually added to the controller of industrial robots. In our simulation, it was not necessary to design a substantial and practical motion planner sub-system model. However, in the implementation phase on the SCHUNK Arm designing such a block

is a mandatory step. The elements of such a block will be described later. The simulation architecture and blocked are depicted below in the figure (4.1)

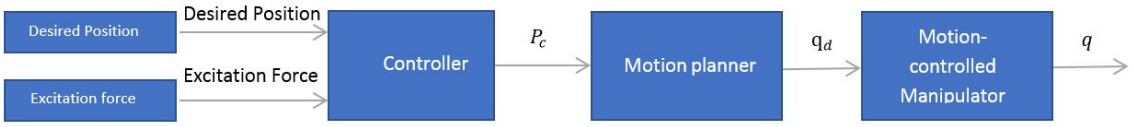

Figure 4.1: simulation architecture

Now, let's describe the procedure of the simulation, at first an initial position and orientation is assumed for the manipulator and based on them, joint values of the manipulator are calculated using inverse kinematic. As we know, if a springy behavior is desired, an equilibrium point must be defined for the manipulator oscillations. In this case, we assumed that TCP is located at [x=50, y=30, z=100] with respect to the robot base frame. Additionally, we assumed that a vector of excitation force is exerted on the TCP. The following figure illustrates how the excitation force was selected.

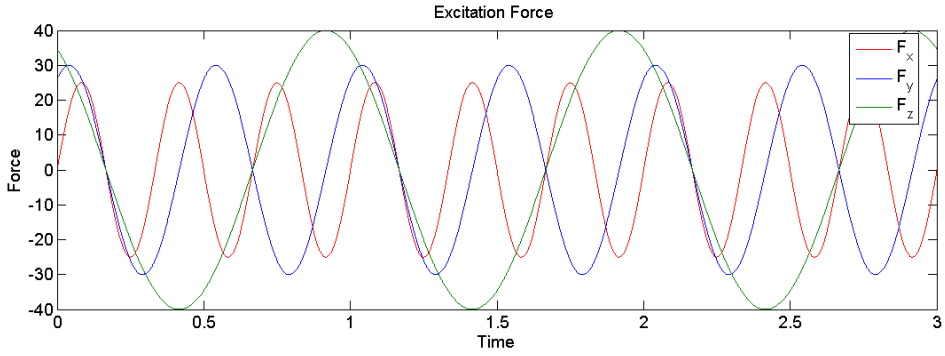

Figure 4.2: Excitation Forces

As you can see in the figure (4.2), the excitation force are exerted in three direction in the cartesian space with respect to the axes of the base frame. Force magnitude is [25 30 40] and the excitation frequency is [3 2 1].

For the first experiment, we set the the stiffness factor of the compliance controller to the 10 and run the simulation.

As you can see in the figure (4.3) the compliance position of the end effector is oscillating corresponds to oscillations in the excitation forces. Note that, the bigger force magnitudes would cause the the bigger oscillation amplitude of the end effector position.

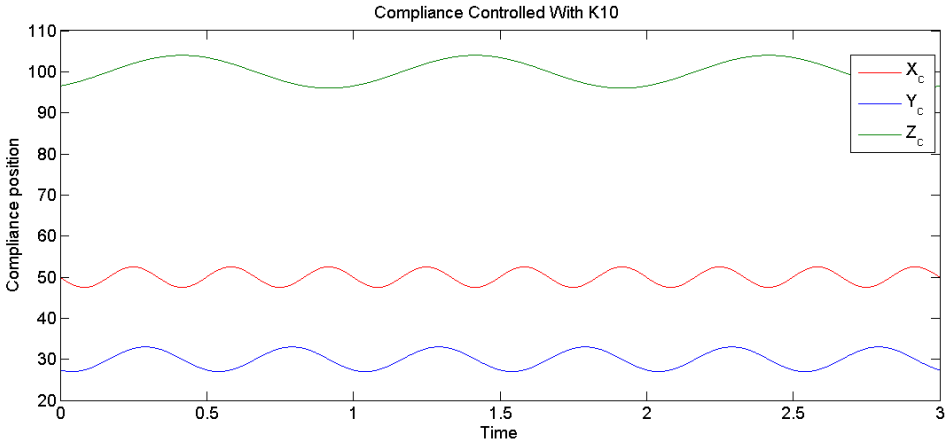

Figure 4.3: Compliance Controlled With K=10

Now, For the second experiment we set the the stiffness factor of the compliance controller to the 1 and run the simulation. here is the result :

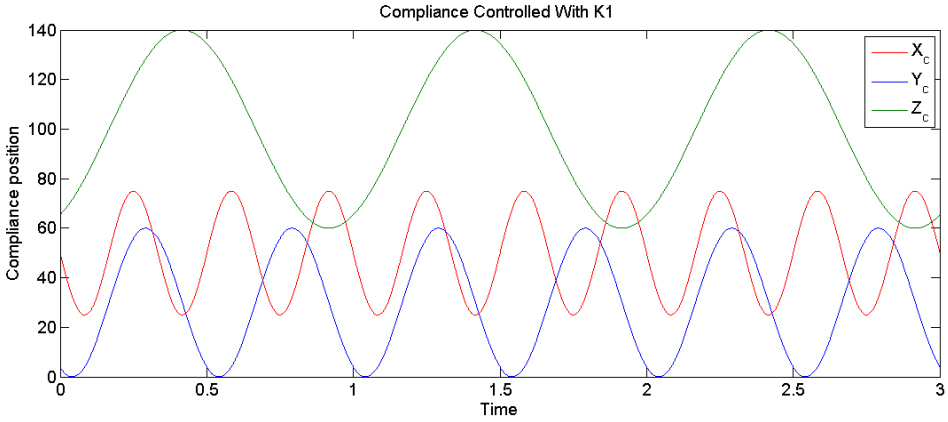

Figure 4.4: Compliance Controlled With K=1

As you can see in the figure (4.4) by decreasing the stiffness factor of the controller, the robot show a softer response to the external forces and it is possible to change the position of the end effector with a smaller force. In other words,the pliability of the springy behavior has been increased.

Now, let's see what if the stiffness factor is set to 100 in the figure , the resultant has been shown in figure (4.5), as we can see a stiffer response of the manipulator to the external forces.

4.2 Implementation methodology

Developing mechatronics systems means intergeneration of the various areas of science in a single system. A system developer needs to firstly study and comprehend

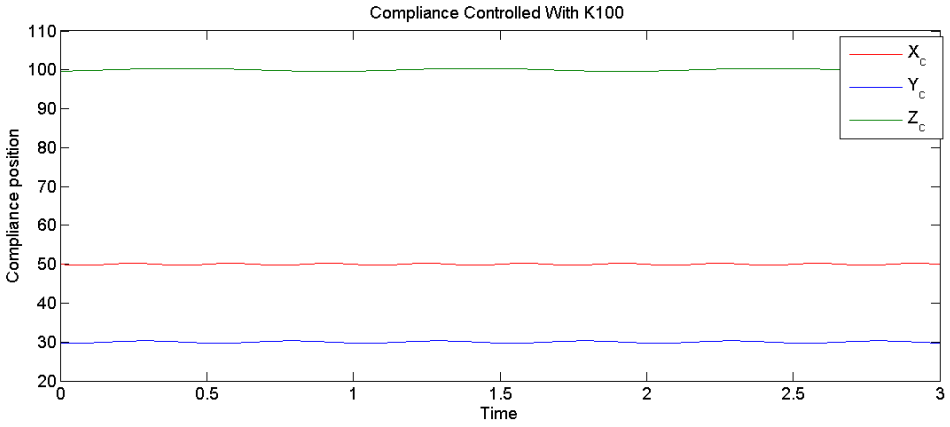

Figure 4.5: Compliance Controlled With K=100

the theory which leads to a viable and feasible solution of the problem. Then, she or he implements the results of his efforts on the intelligent system whose intelligence come from a means of embedded controller. In general, system developers must have a good commands of the computer science, specifically in the computer programming languages. Furthermore, there are mathematical tools like MATLAB and Simulink which provide ready to use functions and algorithm to facilitate developing control theories. Although, these high level programming tools aren't suitable when a realtime tasks in realtime operating system is developed[17]. In this regard, several research has been done to provide a framework to facilitate code generation of the realtime system in MATLAB.

In fact, the most complex task while developing such realtime systems is interfacing the computational controllers developed in a mathematic tool like MATLAB with the data buses of sensors and actuators. To this end, data bus card driver like CAN or serial port which are provided usually by manufacturer must have the ability to work in realtime operating system under realtime constraints. Developing control layers and strategies merely in traditional C, C++ environment demands a lot of time and efforts. Although, in this situation, it would be easier for system developers to integrate device drivers to the control application.

In this section, a software development environment will be described which will enable us to develop realtime control application of a mechatronics system on the Matlab and Simulink Softwares. In this method, there are two PC involved, the one which is called developing PC and the other one which called target PC and play the role of embedded PC to host the execution of the control application. The developing PC hosts a MATLAB software whose all the code generation tool boxes installed with. Generally speaking, a realtime specialized for realtime linux (in our case Xenomai) is generated from a Simulink project. The generated code is conveyed to the Target PC and is compiled there versus realtime libraries. Finally,

a standalone control application is created from the generated source codes [17].

As it has been mentioned earlier, the most complex job is the integration of the device drivers and the control applications. To this end ,an interface applications is incorporated into the code-generation Sumulink project. As you may know, these interface applications are usually developed using C, C++ and they are related to their API library which the manufacturer provides with device driver for the developers. To integrate interface applications into Simulink model, They are needed to turn into Simulink block libraries. This can be done using S-Function programming which is a attribute of the MATLAB software [17]. In fact ,S-functions can represent as a Simulink block. In the case where, the interface applications are developed earlier, S-functions are generated using Matlab Legacy Code Tool and TLC files. In code generation process, the interface code which is extracted for the S-function is connected to the main body of the generated code [17].

4.3 Detail of implementation

The methodology which has been used to implement the force control theory has been described in the previous section. Based on the system architecture which was described in the section (1.1), detail of implementation will be illustrated. The universal control application of this manipulator is run on an embedded PC which we call it Black box pc in our convention. As it was mentioned earlier in (1.1), a realtime operating system kernel called Xenomai is installed on the top of a linux kernel in the black box embedded pc.

This operating system provides realtime services and realtime environment for control applications to run with hard realtime constraints. Additionally, this embedded pc has two built-in CAN card and two Ethernet card ports. As it is widely known, the Ethernet connection is necessary for SSH or SFTP communications. Developer can transfer their realtime control applications sources which have been developed previously in developing PC to the black box and then compile and run it in Xenomai environment.

The controller is connected to other parts of the system through 2 CAN buses. One of them is devoted to manipulator drives and the other one is reserve for the Force Torque Sensor Module and Gripper. Force Torque sensor Module (FTM) is installed on the robot arm between the 6th joint and the end effector which in our application is a gripper. In the following picture the CAN port plate of the Schunk manipulator is shown.

Note that, the CAN buses baud rates are set to 1 Mbit/s. The baud rate configuration is dependent on the vendor of the CAN card. It could be either configured in the *Service Application* which will be run beside the control application or in the realtime operating system RTCAN file.

Figure 4.6: CAN port plate of the Schunk manipulator

There exist two service applications run on the embedded pc beside the generated control application. They are developed in C programming language using realtime API which is provided by Xenomai. These applications are deigned to convey the data between the CAN buses and realtime control application. Realtime queues are implemented in the service application to provide IPC tools. They are binded by the previously implemented queues of the interfacing part of the control application.

As it has been mentioned earlier in methodology section, the C, C++ project of the controller application has been generated from a Simulink model using the Simulink coder toolbox of Matlab. This simulink model contains several S-functions which have been developed using Legacy Code Toolbox of Matlab. These S-functions are basically the implementation of the realtime queues which in fact is not possible to implement in Matlab environment. The legacy code tool creates S-functions from the formerly developed C or C ++ codes.

These codes are usually device drivers and basically they are dependent to the vendors or the operating system APIs. Created S-function can be incorporated to the Simulink model of the controller and then using Simulink coder an appropriate C code can be created to run on the embedded PC.

In our case, the generated code is transferred to the embedded PC, where using a generated make file which again has been created by Simulink coder can be compiled, linked and run respectively. Development scheme and system outline is depicted in the figure (4.7). As you can see in the figure, CAN buses are depicted with the blue line and the red line represents the Ethernet connection for SSH or SFTP communications.

The software architecture of the system is depicted in the figure (4.8). As you can see in this figure, the control application consists of two main part, the control

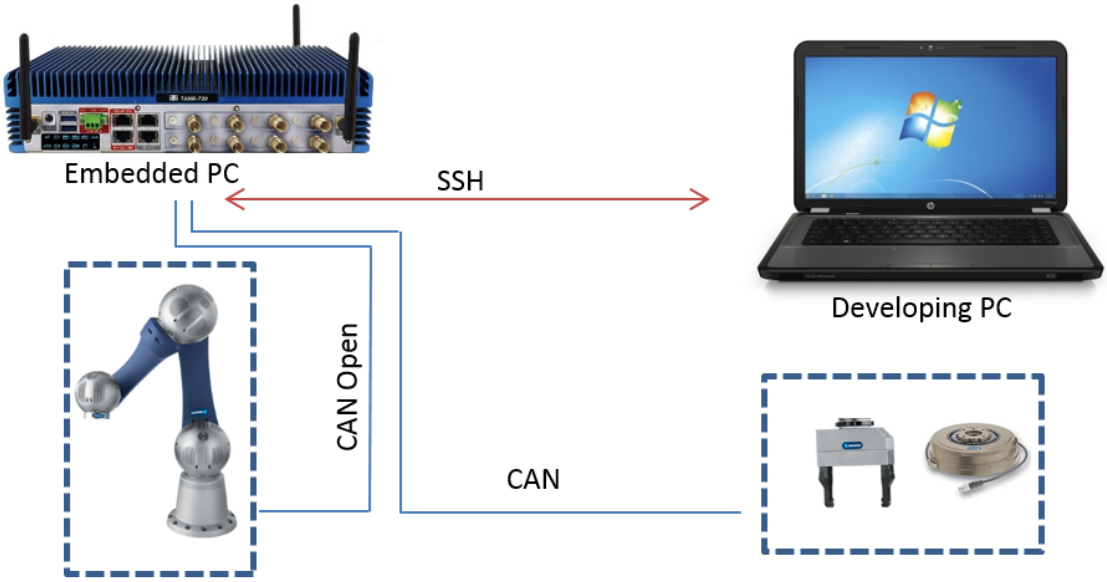

Figure 4.7: Development scheme

algorithms and the interfacing application. The procedure of obtaining the interfacing application and its performance has been described in the previous sections. It was also mentioned implicitly that, what are its control layers and how the control algorithm works.

4.3.1 Force Torque Sensor Module

Force/Torque sensors has extensive applications in the industrial robotic and research projects. They are employed in various applications such as product testing, robotic assembly, grinding and polishing in the industry. Robotic surgery, rehabilitation, haptic and neurology research are also related application of these sensors. Basically a force/torque sensor system consists of strain gauges and the gauge values are related to the amount of force and torque exerted to the sensor module. Strain gauges of the force-torque sensor are installed in a specific manner on a particular mechanical structure inside the sensor module. Their analogue signals are usually converted to the digital signal via the micro-controller built into the module.

Then,the raw signals are sent through the communication buses to an embedded pc or some sort of controller unit to serve as sensory data in a control level layer. These sensors mostly need reliable and high-speed data transmission protocols to communicate with the controller.
In the controller unit, these raw signals are captured in a realtime manner. Then, using an initial calibration matrix, gauges values are related to the amount of the force and torque exerted to the sensor module expressed in the sensor coordinates.

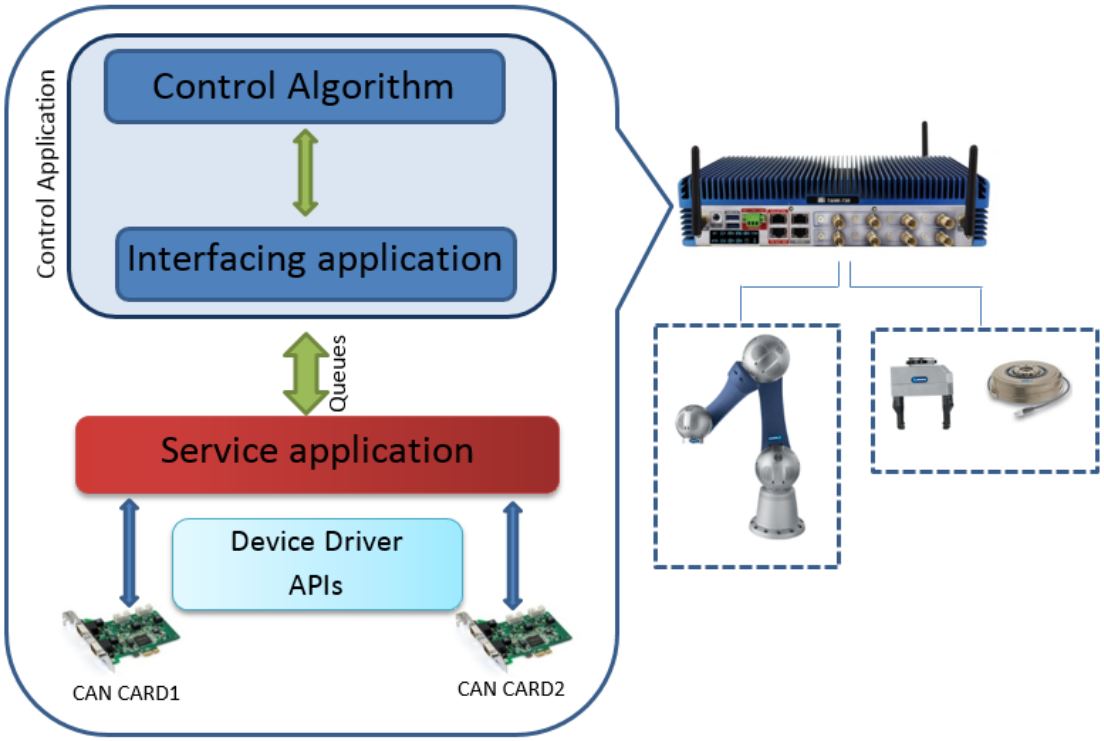

Figure 4.8: Software Architecture

Usually, measured force and torques values are extremely noisy and contain bias values and a variable gravitational force value. Before utilization of this sensory data in the control layer, they must be calibrated accurately. To calibrate sensor values, basically we need to reduce the bias values from the read values. Additionally, the gravitational compensation terms must be applied to the remaining values to neutralize FTM weight effects.
To illustrate that how noisy the acquired data was, figures (4.9) and (4.10) are provided.

In figure (4.9), we can see the force values of non-calibrated data sample captured from a FTM.

This data has been obtained when the joint values were $\begin{bmatrix} 30 & -30 & 0 & 0 & -40 & 0 \end{bmatrix}$. As you can see they are too noisy and inaccurate it is. Furthermore, the equivalent torque values captured from the FTM is shown in the below figure as well.

In the following section, underlying mathematical equations in the FTM calibration and relevant process are discussed in more details.

4.3.2 Force torque sensor calibration method

In our method, force torque module values are acquired in two different position and orientation of the robot. Then to reduce the noise of data, an average value

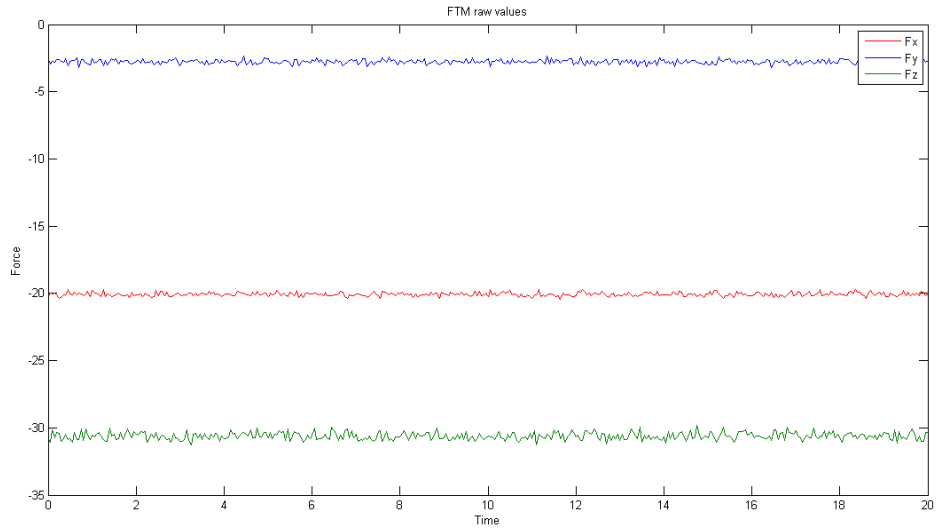

Figure 4.9: non-calibrated force data sample of the FTM

is taken over the acquired time period. Then, the derived equations are solved using captured data in two position by a non-linear numerical solver. Now, let's consider the governing equation in the FTM calibration. As we mentioned earlier, in each sampling time, values read from the strain gauges are multiplied to an initial calibration matrix. Resulted vector is a vector whose first three elements are force exerted on the sensor and its last three elements are exerted torques and all of them are expressed in the sensor frame. Read value F_{read} before calibration is assumed to include the following terms.

$$F_{read} = F_{ex} + F_{bias} + F_{grv} \tag{4.1}$$

F_{ex} is assumed to be the exerted force on the FTM, F_{bias} is assumed to be a constant term which is usually depend on the mechanical installation. The final term is F_{grv} which is the gravitational effect originating from the weight of the module top section and the gripper to the the strain gouges.

As it has been mentioned in the sensor specification, the output signal of force torque sensor is expressed in the sensor frame coordinate. Although, the relative orientation of the sensor frame and the joint frames of the manipulator is unknown. hence , in addition to the force biases, the frame orientation should also be obtained to calibrate the FTM and accordingly getting more accurate sensor data.

Now, if no external force is exerted on the FTM which means $F_{ex} = 0$,we can rewrite the equation(4.1) in the sensor frame as follow:

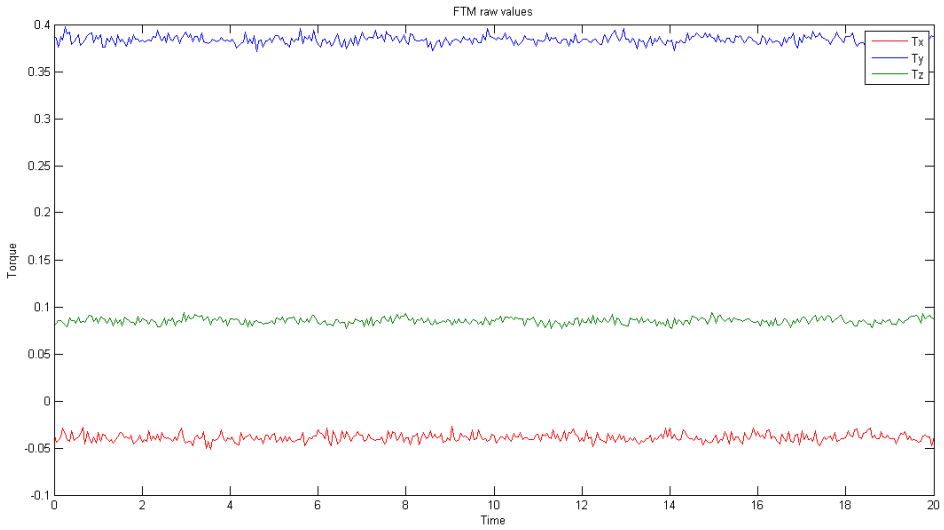

Figure 4.10: non-calibrated torque data sample of the FTM

$$ {}^{S}F_{read} = {}^{S}F_{bias} + {}^{S}_{U}R \; {}^{U} \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{4.2} $$

The ${}^{S}F_{bias}$ is a vector which is expressed in sensor frame and it can be shown as follow:

$$ {}^{S}F_{bias} = {}^{U} \begin{bmatrix} f_{bx} \\ f_{by} \\ f_{bz} \end{bmatrix} \tag{4.3} $$

Based on representation law of rotation matrix we can rewrite ${}^{S}_{U}R$ as follow:

$$ {}^{S}_{U}R = {}^{S}_{5}R \, {}^{5}_{b}R \, {}^{b}_{U}R \tag{4.4} $$

${}^{S}_{5}R$ in equation (4.4) is the rotation matrix between the sensor frame and the frame on the joint number five. ${}^{5}_{b}R$ in this equation is the rotation matrix of the base frame of the robot with respect to fifth joint frame which can be calculated easily by substituting joint angles in the forward kinematic equation of the manipulator.

The last term in above equation is ${}^{b}_{U}R$ which is the rotation matrix of the world frame with respect to the base frame of the robot. As we know the direction of the weight vector is along the Z axis of the world frame , as we know, the base frame of the manipulator and the world frame are not coinciding. In fact, the exact orientation of the base frame to the earth frame is vague. Hence the rotation matrix ${}^{b}_{U}R$ is considered as our unknown variables. Now, let's consider each element of the

equation (4.4) individually.

Firstly, in considering ${}^{S}_{5}R$ let's assume that the Z axis of the fifth joint frame and sensor frame are in the same directions. Therefore, we can say that only a single rotation angle above z axis is enough to obtain ${}^{S}_{5}R$, hence we can write as follow:

$$ {}^{S}_{5}R = \begin{bmatrix} cos(\theta_{5S}) & -sin(\theta_{5S}) & 0 \\ sin(\theta_{5S}) & cos(\theta_{5S}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.5} $$

As we know the second term in equation (4.4) is dependent to the manipulator joint value or in other words it depends on the position and orientation of the manipulator end effector. ${}^{b}_{U}R$ is the dependent strongly on the way that the manipulator has been installed on its platform and we consider the general form of the rotation matrix between the base frame and the world frame. In addition, by considering the equation (4.2) we can see this matrix is directly multiplied into the weight vector which is expressed in world frame ${}^{U}\begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}$. hence the resultant vector would be as follow:

$$ {}^{S}_{5}R = \begin{bmatrix} -sin(p)mg \\ cos(p)sin(r)mg \\ cos(p)cos(r)mg \end{bmatrix} \tag{4.6} $$

where p and r are the pitch and roll rotation angles of the manipulator base frame and the world frame.

Now, by substituting result of equation (4.4), (4.5), (4.6) we can rewrite ${}^{S}_{U}R$ as follow:

$$ {}^{S}_{U}R = \begin{bmatrix} cos(\theta_{5S}) & -sin(\theta_{5S}) & 0 \\ sin(\theta_{5S}) & cos(\theta_{5S}) & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{5}_{b}R \begin{bmatrix} -sin(p)mg \\ cos(p)sin(r)mg \\ cos(P)cos(r)mg \end{bmatrix} \tag{4.7} $$

Now, by substituting the results from equation (4.7) and (4.3), we can write the final calibration equation for the ${}^{S}F_{read}$ as follow :

$$ {}^{S}F_{read} = {}^{U}\begin{bmatrix} f_{bx} \\ f_{by} \\ f_{bz} \end{bmatrix} + \begin{bmatrix} cos(\theta_{5S}) & -sin(\theta_{5S}) & 0 \\ sin(\theta_{5S}) & cos(\theta_{5S}) & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{5}_{b}R \begin{bmatrix} -sin(p)mg \\ cos(p)sin(r)mg \\ cos(p)cos(r)mg \end{bmatrix} \tag{4.8} $$

In practice, the obtained value for the orientation angle of the base frame to the earth frame p and r are negligible. by considering p=0 and r=0 degree we will have:

$$ {}^{S}F_{read} = {}^{U}\begin{bmatrix} f_{bx} \\ f_{by} \\ f_{bz} \end{bmatrix} + \begin{bmatrix} cos(\theta_{5S}) & -sin(\theta_{5S}) & 0 \\ sin(\theta_{5S}) & cos(\theta_{5S}) & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^{5}_{b}R \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{4.9} $$

As you can see in the above equation, $\begin{bmatrix} f_{bx} & f_{by} & f_{bz} & mg & \theta_{5S} \end{bmatrix}$ are 5 unknown variables, so we need to solve 5 independent equations to obtain them. By studying equation (4.10) more deeply, we can see that for any ${}^{5}_{b}R$ there exists three equations which are dependent to the position and orientation of the manipulator end effector. In our case we moved the manipulator in two different position and by recording both force torque sensor values and joint values 6 non-linear equation are solved to obtain the 5 unknown.

These set of non-linear equations has been solved using *fmincon* function of the MATLAB software which is based on computational iteration using an initial guess of the unknown variables. By obtaining these variables from above equations we can use them to compensate the weight term in a realtime manner from F_{read} variable which is captured every 10 millisecond from the FTM.

Calibration operation has been done 6 time with a set of joint values and the resultant calibration data were as follow:

$$ \begin{bmatrix} -18.2398 & -18.9903 & -19.2907 & -18.8303 & -18.8042 & -18.2215 \\ -0.0728 & -0.4023 & 0.0270 & -0.4224 & -0.4377 & -0.0158 \\ -14.4664 & -8.6122 & -8.5442 & -9.2661 & -9.3822 & -13.298 \\ 17.0953 & 17.1397 & 17.1007 & 17.1635 & 17.1730 & 17.1988 \\ 0.5144 & 0.3956 & 0.3582 & 0.4045 & 0.4038 & 0.3985 \end{bmatrix} \tag{4.10} $$

The first three row in the above matrix are force bias calbrtaion data in X, Y and Z directions, the fourth row is devoted to the wheight of the FTM module and is in newton. The last row of the matrix represents the θ_{5S} and is in radians..

We should note that the bias value of the FTM are possible to be altered, when the CAN bus power is off. Therefore, whenever the robot manipulator power is cut off or the CAN buses are rebooted, the FTM should be recalibrate . In this case we just need to find $\begin{bmatrix} f_{bx} & f_{by} & f_{bz} \end{bmatrix}$, Since the weight of top section would never change and θ_{5S} could not alter unless some modification has been done on the FTM mounting flange or the installation angle. Moreover, the pitch and roll angel would not change unless the robot manipulator has been remounted or it has been transported to another place or somehow the base flange orientation is altered.

SOURCES

[1] "IHA Mobile Robot (iMoro)." `http://webhotel2.tut.fi/iha/puresafe/iMoro.shtml`, 2013.

[2] R. Oftadeh, M. M. Aref, R. Ghabcheloo, and J. Mattila, "System integration for real-time mobile manipulation," *INTERNATIONAL JOURNAL OF ADVANCED ROBOTIC SYSTEMS*, vol. 11, 2014.

[3] R. Oftadeh, R. Ghabcheloo, and J. Mattila, "A novel time optimal path following controller with bounded velocities for mobile robots with independently steerable wheels," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 4845–4851, IEEE, 2013.

[4] R. Oftadeh, R. Ghabcheloo, and J. Mattila, "Time optimal path following with bounded velocities and accelerations for mobile robots with independently steerable wheels," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 2925–2931, IEEE, 2014.

[5] R. Oftadeh, M. M. Aref, R. Ghabcheloo, and J. Mattila, "Mechatronic design of a four wheel steering mobile robot with fault-tolerant odometry feedback," *IFAC Mechatronic Systems*, vol. 1, no. 1, pp. 663–669, 2013.

[6] K. Etschberger, R. Hofmann, C. Schlegel, J. Stolberg, and S. Weiher, *Controller Area Network: Basics, Protocols, Chips and Applications*. IXXAT Automation GmbH, 2001.

[7] M. Barr and A. Massa, *Programming Embedded Systems: With C and GNU Development Tools*. O'Reilly Media, 2009.

[8] K. Yaghmour, J. Masters, G. Ben-Yossef, and P. Gerum, *Building Embedded Linux Systems*. O'Reilly Media, 2009.

[9] J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers*. O'Reilly Media, 2009.

[10] Q. Li and C. Yao, *Real-Time Concepts for Embedded Systems*. CMP books, CMP Books, 2003.

[11] B. Siciliano and L. Villani, *Robot Force Control*. Kluwer international series in engineering and computer science: Robotics: vision, manipulation and sensors, Springer US, 1999.

[12] J. Craig, *Introduction to Robotics: Mechanics and Control*. Addison-Wesley series in electrical and computer engineering: control engineering, Pearson Education, Incorporated, 2005.

[13] N. Hogan, "Stable execution of contact tasks using impedance control," in *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, vol. 4, pp. 1047–1054, 1987.

[14] T. Yoshikawa, "Dynamic hybrid position/force control of robot manipulators–description of hand constraints and calculation of joint driving force," *Robotics and Automation, IEEE Journal of*, vol. 3, no. 5, pp. 386–392, 1987.

[15] S. Chiaverini, B. Siciliano, and L. Villani, "Force/position regulation of compliant robot manipulators," *Automatic Control, IEEE Transactions on*, vol. 39, no. 3, pp. 647–652, 1994.

[16] J. De Schutter and H. Van Brussel, "Compliant robot motion ii. a control approach based on external control loops," *The International Journal of Robotics Research*, vol. 7, no. 4, pp. 18–33, 1988.

[17] R. Oftadeh, M. M. Aref, R. Ghabcheloo, and J. Mattila, "Unified framework for rapid prototyping of linux based real-time controllers with matlab and simulink," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics, AIM*, pp. 274–279, 2012.