DEVELOPMENT OF IMPEDANCE SENSING TECHNOLOGY AND AN INTELLIGENT CONTROL SYSTEM FOR ROBOT-AUTOMATED PROCESSING OF FLEXIBLE AND NATURAL OBJECTS

By

Jianhua Scott Gu

B.A.Sc., Tsinghua, China; M.Sc., Chongqing, China; M.A.Sc., Brit. Col.

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

MECHANICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

July, 1999

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

MECHANICAL ENGINEERING
THE UNIVERSITY OF BRITISH COLUMBIA
2324 MAIN MALL
VANCOUVER, CANADA
V6T 1Z4

DATE:

October 12, 1999

Abstract

General tasks of robotic manipulation may be broadly categorized as gross/hard manipulation and flexible manipulation. In gross/hard manipulation, robotic tasks are defined by motion commands based on predefined and precise positions. In this case, the high-level task control typically adopts an open-loop approach where low-level servo controllers drive the robot with the objective of achieving a predefined joint trajectory that is consistent with the task trajectory as planned at an upper level, but the sensory information is not fed into the task level of the robot control system. In flexible manipulation, as defined in this thesis, robot motion cannot be pre-defined in terms of precise positions and motions, due to task uncertainties. Then, feedback of sensory information to upper levels of the control system, with appropriate preprocessing and abstraction, will facilitate high-level task control and improved process performance.

Control systems of the existing commercial robots by and large fall into the category of gross/hard manipulation, with associated shortcomings. Focus of the present research is on fine/flexible manipulation, which is important in robotic processing of flexible and inhomogeneous, natural materials, such as fish and meat, where, the detection of material properties and transition regions in the processed object is usually important for high-level, intelligent task control. Mechanical impedance at the process interface of a robotic task, is considered to provide the information that is needed in detecting the required process characteristics, and is investigated as a significant sensing approach in flexible manipulation. The research presented in this thesis concerns technology development for flexible manipulation of robotic processes.

The main goals of the present research are threefold; (1) investigate and develop an impedance-based task sensing method for flexible manipulation, (2) investigate, design, and develop a control architecture that has the capability of on-line task monitoring and interpretation,

high-level feedback of information, and knowledge-based decision making, which is suitable for executing flexible manipulation of robotic processes, and (3) implement and evaluate the developed sensing and control system technologies with regard to practical applications. To achieve these objectives, the following tasks are carried out:

Analytical framework for dynamics, sensing, and control. Models are formulated for robot dynamics and process impedance, with linearizing feedback and task separation for gross motions and fine manipulation. This framework is intended for facilitating on-line estimation of process impedance and implementation of knowledge-based task monitoring and control.

Impedance sensing technologies. Methods of estimating mechanical impedance at the process interface of a robotic processing task are developed, which use actuator effort and the joint motion signals as the input information to the estimator. Methods of signal conditioning and estimation are developed based on the Kalman filter approach. Further interpretation and utilization of impedance information at the task level is made through knowledge-based decision making. This is intended to accommodate task uncertainties, specifically, position variables and disturbances.

Control architecture. The use of mechanical impedance as intended in the present research requires feedback of sensory information to the task control level. To accommodate this and other requirements of flexible manipulation, a real-time open-architecture control system (ROACS) is designed and developed in this thesis. The control system has a hierarchical structure with the capability of multi-mode control. A systematic method is developed for intelligent task control, where tasks may be represented in a descriptive manner with unknown task variables which may be determined and assigned during operation.

Design of implementation model and system prototyping. A client-server type robot control system is designed and developed that satisfies the ROACS model. The system contains three servers: servo controller, motion planner, and data analyzer; and three client applications: intelligent task control, information updating module, and a graphics user interface which

includes a robot control shell and a real-time task language (RTTL) as developed in the present research. The entire system adopts a cooperative model between the real-time subsystem and the artificial-intelligence subsystem, which can efficiently carry out both low-level sensing and control tasks, and high-level process monitoring and knowledge-based decision making tasks in an integrated manner.

The developed robot controller is implemented for operation with a Puma 560 robot in the Industrial Automation Laboratory. A salmon-slicing task, which produces boneless salmon steaks, is performed by using the developed system, to demonstrate the performance of ROACS and the feasibility of employing the developed technology in an industrial application. The developed technology is also implemented in an industrial CNC router machine to accurately sense the material surface of the processed object for system referencing for subsequent processing.

Table of Contents

List of Figures

List of Tables

List of Symbols

$\mathbf{A}$	System matrix of a linear state space model.
B	Viscous friction coefficient of a rotational robot joint.
$\mathbf{B}$	Input matrix of a linear state space model.
$\mathbf{C}$	Output matrix of a linear state space model.
C_i	$\cos(\theta_i)$
C_{ij}	$\cos(\theta_i + \theta_j)$
$\mathbf{D}(\mathbf{q})$	Matrix of Coriolis terms in a model of robot dynamics.
$\mathbf{E}(\mathbf{q})$	Matrix of centrifugal terms in a model of robot dynamics.
F, $F(s)$	Force.
G	Gear ratio.
$\mathbf{G}(\mathbf{q})$	Gravity torques (vector).
J	Moment of inertia of a mechanical system.
$\mathbf{J}$	Jacobian matrix.
K	Kinetic energy of a mechanical system.
$\mathbf{K}_d$	The derivative gain matrix of a PD controller.
$\mathbf{K}_p$	The proportional gain matrix of a PD controller.
L	Lagrangian.
L_m	Motor inductance.
$\mathbf{M}(\mathbf{q})$	Inertia matrix of a robot manipulator.
P	Potential energy of a mechanical system.
P	Error covariance.
$P_j(t)$	Cubic spline function.

Q	Covariance matrix of process noise.
R	Covariance matrix of measurement noise.
R_m	Motor resistance.
R_s	Shunt resistance.
S_i	$\sin(\theta_i)$
S_{ij}	$\sin(\theta_i + \theta_j)$
$T_R(X, \alpha)$	Rotation about axis X for an angle α.
$T_R(Z, \theta)$	Rotation about axis Z for an angle θ.
$T_T(X, a)$	Translation along axis X for a distance a.
$T_T(Z, d)$	Translation along axis Z for a distance d.
${}^{i-1}T_i$	Homogeneous transformation from frame $i-1$ to frame i.
T_x	Period of path planning.
V, $V(s)$, v, $v(t)$	Generalized velocity in frequency/time domain.
Z, $Z(s)$, z, $z(t)$	Mechanical impedance in frequency/time domain.
Z_e, $Z_e(s)$, z_e, $z_e(t)$	Mechanical impedance at the process interface.
Z_m, $Z_m(s)$, z_m, $z_m(t)$	Mechanical impedance of the manipulator itself.
a	Offset between axes of robot arms.
$\mathbf{a}$	Joint acceleration (vector).
b, $\mathbf{b}$	Viscous damping coefficient/vector in the robot joint space.
d	Distance along links of a robot manipulator.
e_b	Back EMF.
f, $f(t)$	Force in time domain.
f_d	Force of disturbance.
f_e	External force.
f_g	Gravity force.
f_m	Motor magnetic force.

i_m	Motor current.
k	Stiffness.
k_E	Back EMF constant.
k_a	Amplifier gain.
k_i	Kinetic energy of the ith joint of a robot.
k_τ	Motor torque constant.
m	Mass.
p_i	Potential energy of the ith joint of a robot.
$\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$	Generalized displacement, velocity, and acceleration of robot joints in the joint coordinates.
s	Laplace transform variable.
t	Time.
v_d	Input control voltage of a joint drive-motor.
v_s	Shunt voltage.
w	Weight.
$\mathbf{w}$	Process noise (vector).
$\mathbf{y}_k$	System output.
$\mathbf{x}$, $\dot{\mathbf{x}}$, $\ddot{\mathbf{x}}$	Generalized displacement, velocity, and acceleration of a robot end-effector in the world coordinates.
Γ	Input matrix of a discrete time state space model.
Φ	System matrix of a discrete time state space model.
$\sum_i$	The ith coordinate frame.
α	Twist angle between axes of robot arms.
ϵ	Speed value that separates stable and unstable friction regions.
ν	Measurement noise.

ω, Ω	Angular speed.
τ	Force/torque, particularly, the torque used to overcome the manipulator dynamic force and gravity.
τ_c	Force/torque of Coulomb friction.
τ_d	Force/torque of disturbances.
τ_e	Force/torque reflected at the joints from the force/torque exerted on the robot end-effector.
τ_e	Time constant of an electric system.
τ_f	Force/torque of friction.
τ_m	Motor magnetic force/torque.
τ_m	Time constant of a mechanical system.
τ_s	Force/torque of stiction.
$\theta, \dot{\theta}, \ddot{\theta}$	The angle, velocity, and acceleration of a robot joint.
$\hat{\theta}$	Estimates of parameters.

Superscript

$^+$	Positive direction.
$^-$	Negative direction.

Subscript

$_d$	Disturbance.
$_e$	External.
$_i$	The ith joint.
$_l$	Link.
$_m$	Motor.

Technical Terms

CNC	Computer numeric control.
Photon	Photon windowing system based on QNX operating system.
QNX	QNX real-time operating system.
ROACS	A real-time open architecture control system.
RTTL	A real-time task language.

Acknowledgment

First, I wish to take this opportunity to express my deep gratitude to my supervisor Prof. Clarence de Silva for his encouragement, support, and delightful guidance throughout the entire course of my Ph.D. program. Also, I wish to express my appreciation to professors in the Department of Mechanical Engineering, UBC, and the professors who visited the Industrial Automation Laboratory (IAL) from various parts of the world. In particular, Prof. Elizabeth Croft gave me some good advice on my study and my life; Prof. Vinod Modi offered me a special course on dynamics to strengthen my background in this area; and Prof. Chul-Guo Kang from the Kon-Kuk University, Korea, helped me and worked closely with me on robot kinematics. I had very fruitful discussions with Dr. Reuven Granot, from Israel, and Prof. Hae-Jay Park from Korea. I would also like to thank all my colleagues in IAL for their help and discussion.

Part of the thesis research has been carried out as a project of the networks of centers of excellence within the Institute for Robotics and Intelligent Systems (IRIS) program. Grants to Prof. C. W. de Silva through this program, and also from the Natural Sciences and Engineering Research Council of Canada (NSERC), B.C. Packers, Ltd. and Garfield Weston Foundation, are gratefully acknowledged.

Last, but not the least, I wish to thank my family; my wife Weili, my children Eric and Raymond, my parents, and my parents in law, for their unreserved love, support and patience in the past five years, without which this thesis and the previous research for my M.A.Sc. degree would not have been possible.

Chapter 1

Introduction

The primary objective of the research presented in this thesis is to investigate, develop, and implement some important control technologies for flexible manipulation of robotic processes. The main motivations of the work arose from importance of processing flexible and inhomogeneous material objects in a range of industrial applications, and the serious limitations presented by the control systems of common industrial robots, in carrying out such processes. The rationale of the work is provided by the relative benefits of the expected technology over what is currently available, and the belief that these advantages could be achieved without a significant increase in system control complexity. This chapter will emphasize and expand on this goal, motivations, and rationale, and also will give the background of the research.

The use of flexible manipulation in automated production systems will be discussed, and task uncertainties and sensor-based intelligent systems will be considered in this context. Mechanical impedance at the process interface of a robotic processing task is treated here as an important type of information for the recognition of process properties and for system monitoring and task-level control. Specifically, online detection of process properties based on mechanical impedance information is expected to facilitate intelligent processing of a variety of tasks where robot manipulation cannot be accurately described in terms of motion specification alone, in advance. In this context, open-architecture, real-time, intelligent systems as developed in the present work are increasingly in need for implementing advanced sensor-based robot control technologies.

1.1 Motivation and Rationale for Impedance Sensing

Robot manipulators are particularly suitable in non-rigid production operations where small batches of a range of different products are manufactured. This is true because robots can be easily programmed for carrying out different tasks, without needing extensive retooling or major changes in hardware and instrumentation. A large majority of successful industrial applications of robots are limited to motion-control tasks, such as parts transfer, seam welding, and spray painting. Manipulators equipped with motion control technologies alone have drawbacks in processing tasks where, for example, small errors in motion could lead to large variation of force, which is the case of motion of a tool against a hard constraint. Also, such control systems cannot properly accommodate robotic processes with position uncertainties caused by such reasons as the flexibility and inadequate holding of the processed object, changing of process conditions during operation, and unknown disturbances. Furthermore, motion control alone cannot provide acceptable performance, in general, for robotic processing tasks that are governed by force-motion relations between the end-effector and the processed object, in conjunction with constraining environment. In such situations, detection and control of either the processing force or the process impedance would be valuable. Assembly tasks, food processing tasks, and material identification fall into robotic processes that can benefit from impedance information at the process interface. This aspect will be elaborated in the course of the thesis.

By definition, a standard robot manipulation involves controlled operation of a robot manipulator in order to move an object or to apply a force to a process environment. There are three basic types of manipulation that are defined in terms of task requirements: position specifications only; both position and force requirements, with motion and force specified in different sub-spaces; and general impedance requirements specified as force-motion relations. Corresponding to these general categories of task specifications, the robot manipulation can be

grouped according to the associated control requirements, as follows:

Position control only. Parts transfer, seam welding, and spray painting are typical tasks where only motion requirements are specified in terms of known positions and required time points or trajectories. Accordingly, position control alone is adequate in these manipulations. Most commercial industrial robots are designed for carrying out tasks, where accurate positioning would be important [83, 89], and should be accurately specified in the robotic task commands. Force and impedance considerations are of minor significance here and are neither specified in the task description nor provided by the robot control system.

Hybrid position and force control. Here, both motion and force are important for proper execution of the robotic task, but they are specified separately in independent directions. Accordingly, motion control and force control are developed as two separate subsystems, so that the forces are controlled in some constrained directions while positions are controlled in other unconstrained directions. Switching of control laws would be required for transition from free space to contact with an environment. Characteristics of the environment have to be known for this purpose. Relative weighting of the two control tasks should be considered when integrating the two subsystems, and a proper balance should be reached. Grinding is a typical task of this type, and hybrid position and force control [76] has been developed for this type of applications.

Impedance control. This third type of manipulation accommodates a general case of task requirements, where force and motion are dynamically coupled in the same degrees of freedom, and force-motion relations need to be specified for proper execution of the task. Impedance control has been suggested to meet the requirements of force-motion relations in manipulation [54, 55, 56]. Impedance control is inherently a model based method. Two basic approaches exist; namely, position-based impedance control and torque-based impedance control [68]. In the position based approach, positions are commanded while forces are measured. In the torque based approach, positions and forces are measured while torques are commanded. It is clear,

however, that the impedance itself should be measured (or estimated) online, and used in proper control strategy, in order to achieve proper impedance. The present research is built upon this fundamental understanding. Impedance control essentially allows a robot system to emulate another simpler system that satisfies task requirements, assuming the new behavior is within the capabilities of the robot system.

Clearly, robot manipulation depends directly on the task requirements. This aspect should be considered not only in low-level direct control, but also in task-level control. Traditionally, robot tasks are specified as motion requirements. However, there are many other process applications where tasks cannot be given in position specifications alone, or position requirements are less significant than force or impedance requirements. For example, in robotic tasks of contacting a rigid environment (e.g., a parts assembly process), a very small change in displacement against a hard surface would result in a large force. In such tasks, position control can be viewed as an ill-posed problem, where exact positioning is almost impossible. Force control would be relatively easy and more appropriate in these circumstances. In robotic food processing that involves cutting tasks, such as separation of meat from bone (see Figure 1.1), measurement of

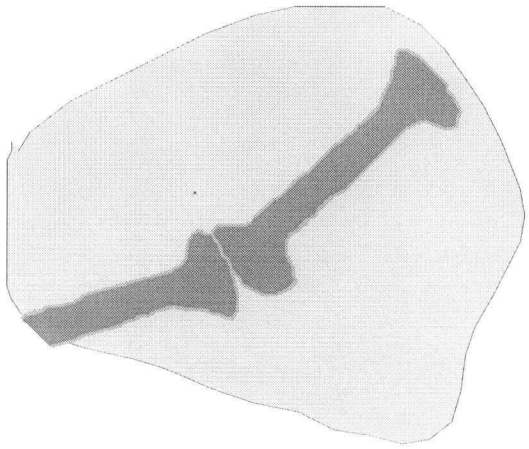


Figure 1.1: A section of meat with bone.

the exact bone position is very difficult and could be infeasible. Hence, online detection of bone

by sensing the mechanical impedance, with some *a priori* knowledge of impedance differences that are manifested in cutting meat and bone, would be much more efficient and effective. Execution of such types of tasks demand flexible manipulation that should handle uncertainties in task specifications through online feedback of sensory data and intelligent decision-making.

Flexible manipulation. Flexible manipulation is defined in the present research to mean the robot manipulation where interpretation and high-level feedback of online sensory data along with process knowledge and intelligent decision-making would be employed to resolve uncertainties and variations in a "flexible" task specification. The associated "unknowns" may be positions that are required for robot motion control, or process-dynamic characteristics. Flexible manipulation has two major features:

- Sensor-based manipulation. Without online sensory information, position unknowns may not be determined and corrected.
- Intelligent manipulation. Here, online detection of process characteristics would be based on process knowledge which also may incorporate high-level interpretation (abstraction) of low-level sensory data, and will require intelligent decision-making procedures.

Flexible manipulation of this type challenges the traditional robot sensing and control technologies. First, the entire robot control system, including task-level control, not just the low-level robot servo control, will require process information and online feedback. Reliable and cost-effective sensing technologies are essential in flexible manipulation. Secondly, a traditional position-based task control system will not be effective in a wide range of robotic processing tasks, as noted before, and cannot generally handle high-level information feedback and intelligent decision-making. As will be emphasized in the sequel of this thesis, impedance-based task control is particularly appropriate in this regard. An open-architecture control system is necessary for this purpose. The system should be both "real-time operational" for

online control and "intelligent" for high-level decision-making. The conflicts between real-time performance and intelligent decision-making should be resolved properly in such a control system. The research presented here is motivated by the fact that commercial robot controllers typically do not possess these capabilities for flexible manipulation. A primary goal of the research is to systematically study and develop a robot control system that possesses these capabilities.

The reminder of the chapter is organized into seven sections. Section 1.2 will provide an introductory background of flexible manipulation, especially in robotic food processing. Section 1.3 will present a literature review of robot sensing and manipulation. Section 1.4 will discuss the sensing technologies that will be developed in this research. Section 1.5 will address the concept and principles of development of a sensor-based real-time intelligent robot control system. Section 1.6 will explicitly summarize the objectives of the research presented in this thesis. Section 1.7 will discuss an example domain which will be implemented subsequently in the thesis as an application and demonstration of the developed technologies. Section 1.8 will outline the plan for the entire thesis.

1.2 Robotic Food Processing

When processing natural objects that consist of inhomogeneous and anisotropic biological material, which is a common situation in the food processing industry, it is desirable to detect the transition regions within the material for proper control of the process. This will be crucial particularly in improving the yield, the product quality, and the overall process efficiency. For example, consider the task of head removal in processing a fish [28]. If the useful meat is removed with the head which is discarded, the yield and the revenues will drop and a useful natural resource will be wasted. If parts of the head are left behind with the body, the product quality and aesthetic appear will suffer, and furthermore, an additional processing effort will

be needed to remove the unwanted parts [28, 36].

A conceptual robotic fish processing operation is shown in Figure 1.2. It is neither conve-

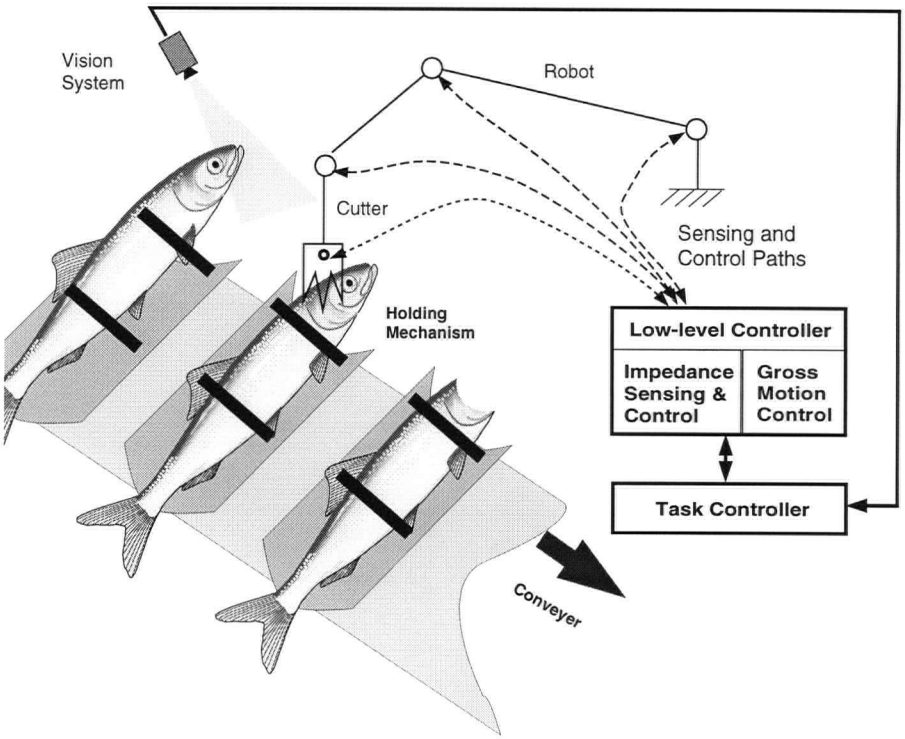


Figure 1.2: Conceptual representation of a robotic fish processing operation.

nient nor economical to instrument the fish to determine various regions such as skin, bone, and useful meat within its body. In fact, it could be totally infeasible to do so. A better approach would be to determine these regions online as the cutter interacts with the fish during processing. An operation of separating meat from bones for the production of boneless steaks will need similar sensing techniques to detect different regions within the body (See Figure 1.1). Online sensing is useful not only for ascertaining characteristics of the object that is being processed, but also in monitoring and controlling the process operation. Impedance information can be used for monitoring process loads, which is an example in computer numeric control (CNC)

machine operations.

Sensing of mechanical impedance, which is defined as:

$$\frac{\text{Generalized Force}}{\text{Generalized Velocity}},$$

will provide a means of online characterization of the object-cutter interface. This may serve as a gradient measure, for example, which could be employed to steer the cutter either towards or away from certain regions. Furthermore, as illustrated in [26], an impedance control policy may be developed whereby the process impedance may be adjusted so as to optimize an appropriate performance index. In this manner impedance control may be viewed as a case of optimal control as well.

The direct way to sense mechanical impedance of a robotic process would be through simultaneous measurement of force and velocity at the process interface. Since incorporation of an explicit force sensor will increase the mass and also the instrumentation cost of the end-effector and, furthermore, create difficulties in a food processing environment that is contaminated with debris and goods, in the present research the motor currents will be used as the sensory data for estimating the force parameters. The lack of delicate sensors at the mechanical-processing interface is a further advantage in such an approach as this will not hinder processing of biological matter in an industrial environment, and also will improve the operating flexibility and design robustness of the end-effector. Also, an explicit velocity sensor will not be used in the present work, and the built-in optical encoders of the robot will be employed for sensing speed. The sensory signals obtained in this manner may not represent the desired variables for impedance estimation, for reasons such as dynamic interference, nonlinearities, external disturbances, and noise. The measurements have to be conditioned and further processed for online estimation of mechanical impedance at the interface of the object and the end-effector. This procedure may be assisted by a dynamic model of the process interface. The same model may be useful in performance evaluation and model-based control

as well [34]. Furthermore, the low-level information may be preprocessed and used in a high-level, intelligent, supervisory control systems for the process, with assistance from other sensory means such as vision [35, 47], as illustrated in Figure 1.2.

1.3 Literature Review on Robot Sensing and Manipulation

Research on robot manipulation may be categorized into two classes: unconstrained manipulation and manipulation in a constrained environment. The former category has been subjected to extensive research, and many applications have been realized in the past two decades. The formulation and solution of kinematics, dynamics, and motion control problems have been well addressed [6, 42]. The latter class which considers robotic manipulation with environmental interactions has drawn much attention in recent years [60, 61, 11, 13, 24, 69, 91, 92]. By definition, robot manipulation is typically intended to handle objects and to perform movement and processing tasks. In open-loop manipulation (i.e., without task-level sensing and feedback control), it is difficult for a robot to know and control the performance errors and to adapt to unmodeled or random characteristics of the environment. For example, in robotic food processing, the object (e.g., fish or meat) and its environment (e.g., holding mechanism and conveyer) cannot be exactly modeled. Perception of the object and its environment, and proper control of the robotic task (e.g., cutting) on that basis, will be important here. Sensing of both force and motion will be useful in this context. Force sensing is particularly useful in fine manipulation, where application of a desired level of force would be important. Mechanical-impedance characteristics [54, 26] can provide valuable information of the processed object, the environment, and will be important in monitoring and controlling a robotic task.

Two primary techniques are widely used in sensing the dynamic environment of a manipulation task; namely, vision and contact sensing. Although hearing [77] and smell [14] have been employed as well in some special-purpose robotic applications, they may be neither feasible

nor economically viable in a large majority of industrial processes; vision [42] and contact sensing [9] are of primary significance in applications such as manufacturing, food processing, and spacecraft.

Visual observation is an important means of perception. Information such as shape, size (including volume), color, position, velocity, and even surface texture of an object can be extracted through image processing. Consequently, robot vision has been an active area of robotic research in recent years. Progress has been made in both theory and applications [2, 62] in this context. Major problems in vision as a sensing method arise from the inaccuracy of image processing, the need for processing large quantities of information with associated restrictions on speed, and the difficulty of three-dimensional (3D) interpretation from 2D images. Vision takes advantage of the measurement of relative location and orientation of an object, but precision and accurate detection of the object may be compromised. This may be remedied through the use of redundant sensors to assist the vision system. Specifically, sensor fusion techniques [2, 74, 82] are used to overcome the drawbacks of using vision as the only means of sensing the process information. Another problem in vision is that images lack a tactile "feel" of an object. Object properties such as flexibility, firmness, and internal texture are not directly contained in visual images, although some special-purpose images, such as depth images produced by ultrasonic scanners, can reflect such material properties of an object. Also, vision systems can be too expensive for some applications.

Contact sensing [57] can be important as it gives direct information about an object and its interface. Object properties can be "felt" by the interior (e.g., joint) sensors of the robot as well. In many applications, explicit tactile sensing of an object is not necessary, but force and speed information at the object/end-effector interface can be quite important. Robots and their end-effectors have to be properly controlled in order to handle complex and flexible objects during processing operations, to avoid damage to the object and the manipulator, and also to achieve the required processing quality and accuracy. A general sensing scheme for a robot

and its environment is illustrated in Figure 1.3. Here four stages are shown:

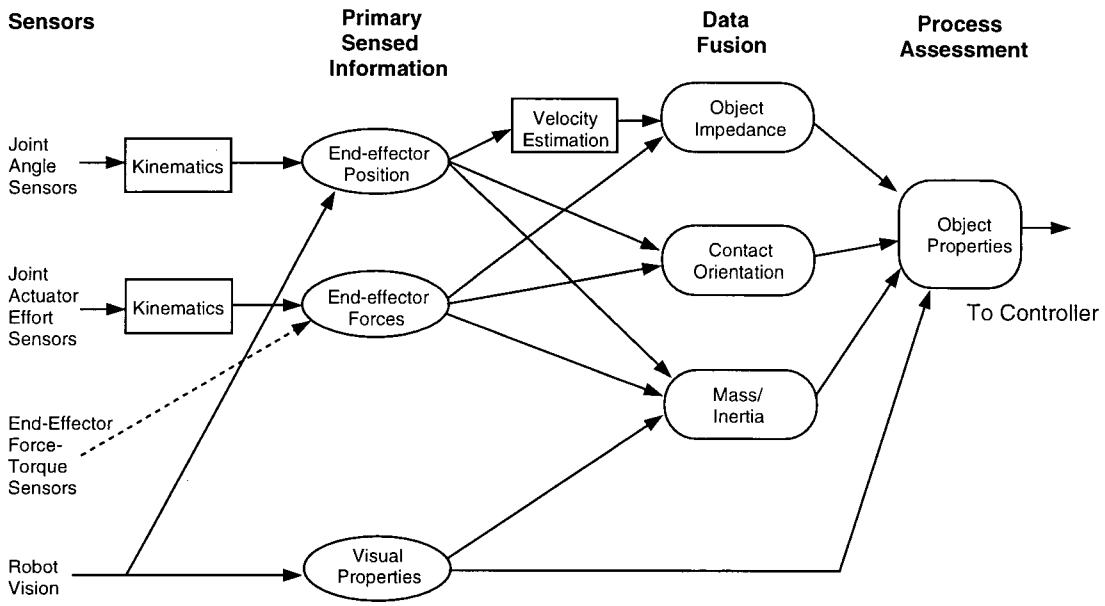


Figure 1.3: Diagram of robot sensing and data fusion for control.

- **Sensors.** This stage represents physical sensor units. The sensors include joint angle sensors, such as optical encoders, potentiometers, and resolvers; actuator effort sensors, such as motor current sensors; direct force/torque sensors of joints and end-effector; and robot vision.

- **Primary sensed information.** At this stage, signals are measured using appropriate hardware and software. Some preprocessing and coordinate transformation through robot kinematics would be needed. Typically, the obtained information includes positions and speeds of joints and end-effector, joint forces and end-effector forces, and visual properties of the processed object.

- **Data fusion.** Data fusion technologies are utilized to extract more complex and high-level information, such as process impedance from position signals and force signals; contact orientation from end-effector orientation and contact force direction; and

mass/inertia from visual properties together with the contact forces and motion. Some information redundancy would be present in data fusion, and may be utilized to improve the reliability and accuracy of the processed results.

- **Process assessment.** In this stage, the pre-processed sensory information is used to assess the robotic process (i.e., the end-effector and object interface). Typically, object properties may be obtained through data processing and interpretation. A dynamic model of the process interface and its environment may be necessary. Conventional estimation and identification techniques, and knowledge-based decision making may be required in this stage.

Mechanical impedance considerations are emphasized in this research, as they are important in manipulation and mechanical processing of natural objects with inhomogeneous and flexible mechanical characteristics; for example, as applied to robotic food processing. The next section will present some background information, and will lay the foundation for an analytical framework for impedance modelling and estimation.

1.4 Impedance Sensing and Estimation in Flexible Manipulation

Sensing and/or estimation of impedance would be primarily important in impedance control of a robot manipulator. Here the impedance of the robotic process and its environment is required for accurate control of the robot in executing a processing task while having dynamic interaction with the environment. Online identification will be required in general, since the impedance of the process interface and its environment is unknown or varying. Nagata, *et al.* proposed a method for impedance identification using fuzzy environment model [71]. In the current research, it is important that the impedance information is expressed in an explicit form for use in process interpretation and high-level intelligent task control. For this purpose, we now define the mechanical impedance.

Mechanical impedance is a measure of dynamic resistance to motion. It is particularly useful for identifying mechanical properties of inhomogeneous and flexible objects and associated environments during robotic manipulation. Specifically, mechanical impedance is defined as the ratio of the generalized force to the generalized velocity in a specified degree of freedom (coordinate direction of incremental motion),

$$z = \frac{f}{v}. \tag{1.1}$$

Conventionally, the frequency domain representation is employed, with the corresponding transfer function defined by,

$$Z(s) = \frac{F(s)}{V(s)}, \tag{1.2}$$

where, f (or F) represents the force exerted on the processed object by the robot end-effector, for example, cutter; v (or V) is the relative velocity of the end-effector at the process/object contact point in the specified direction of motion; z (or Z) represents the associated mechanical impedance at the interface of end-effector and the object; and s is the Laplace-transform variable. The lower case letters represent variables in the time domain, while the upper case letters represent the corresponding variables in the frequency domain. Generally, from the point of view of basic mechanical elements, impedance can be grouped into the following three basic types [26, 35, 47]:

1. Inertial impedance , $Z_e = ms$, which corresponds to the inertial resistance of a pure mass or moment of inertia;

2. Dissipative impedance , $Z_e = b$, which corresponds to mechanical resistance in energy-dissipation through a viscous-damping model;

3. Elastic impedance , $Z_e = \frac{k}{s}$, which corresponds to elastic resistance through recoverable deformation, as represented by a spring model.

The parameters of the type m, b, and k may be identified for an object/environment model (See Figure 1.4). It is clear, however, that due to process nonlinearities and limitations of

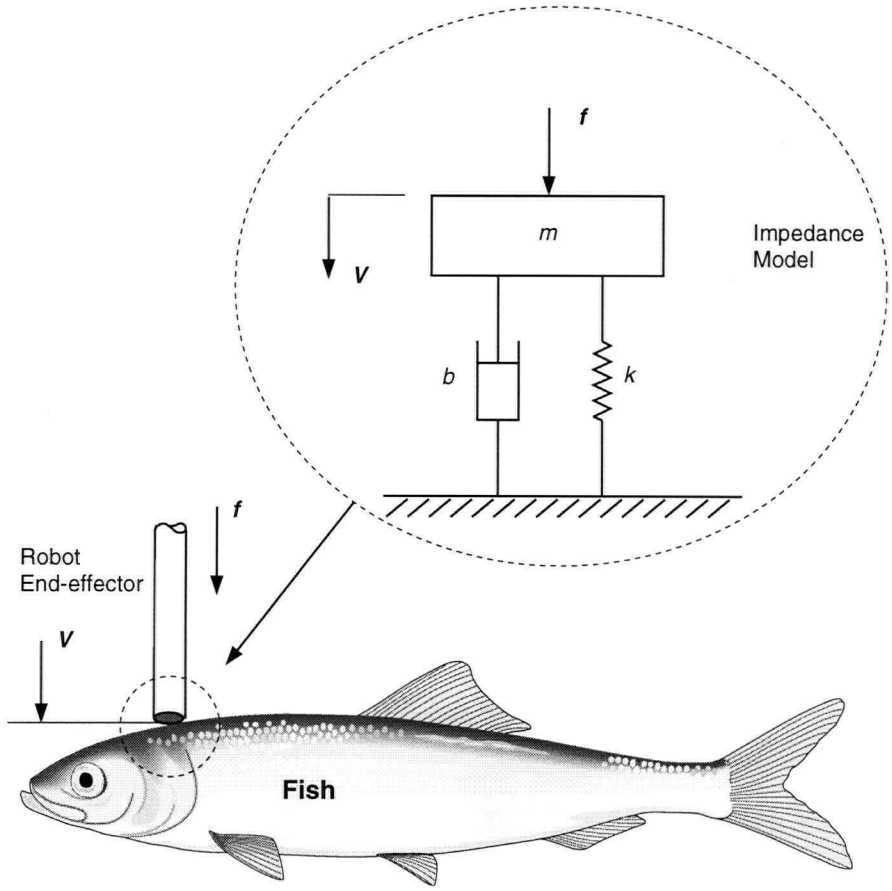


Figure 1.4: An impedance model for an end-effector and object/environment interface.

the impedance model that is used, the impedance parameters will vary with the point of operation (hence, with time). Direct measurement of mechanical impedance at the interface of a robotic task will require sensing of the force which the robot manipulator exerts on the object, and the relative velocity between the object and the end-effector at the contact point (interface) in a specified direction. It has been discussed [34] that, direct sensing of force and velocity may result in high cost, poor reliability, and a sluggish system. To overcome such drawbacks, an implicit sensing/estimation approach should be developed for force and velocity,

and consequently for impedance, using commonly available information of driving current (or, pressure in the case of a hydraulic manipulator) of the actuator and the associated position signal. In this context, the present research focuses on the idea of "contact" sensing with inherent determination of process characteristics, which emulates the information feedback in human touch.

The sensing of actuator loads and associated interpretation for determining task characteristics is intuitively appealing and natural, similar to how humans interpret process characteristics as they carry out mechanical tasks. This may present a basis for "intelligent" sensing and control in the present research. Also, since typically more than one actuator (and sensor) are involved, with associated sensor generalization, fusion, and interpretation, the approach is considered robust and reliable. Furthermore, no explicit force sensors are involved, particularly in the mechanical process environment; hence the approach will be cost effective as well.

Sensing and/or estimation of mechanical impedance can be accomplished in both time domain and frequency domains. In the time domain, first the external force and velocity should be measured/estimated, and conditioned, and then impedance could be obtained by using the definition given in equation (1.1). In the frequency domain, a second order linear model consisting of the three basic types of impedance can be "fitted" to the definition (equation (1.2)),

$$Z = \frac{F}{V} = ms + b + \frac{k}{s} \tag{1.3}$$

Impedance parameters may be identified, for example, using the least squares method. Details of the underlying approaches will be presented in Chapter 3. Note that the present discussion concerns mechanical impedance in a single direction of motion. But, in practice, a robot end-effector can have six degrees of freedom (three linear directions and three orientations). In this case, one needs to consider an impedance matrix that has the dimensions 6×6. If the degrees of freedom are chosen to be canonical ones, the impedance matrix becomes diagonal.

Furthermore, if orientations are neglected, only three impedance functions will be present. The extension of the impedance concepts to a multiple degree of freedom robot is addressed in Chapter 4.

1.5 Sensor-Based Real-Time Intelligent Robot Control

The specific objectives of the present research will lead to the development of a versatile control system for robotic processing. This system will take a hierarchical structure [31], as needed by the task requirements of flexible manipulation. Direct control of the robot actuators will be carried out at the lowest level, using low-level sensory information which may include mechanical impedance at the process interface. Servo control and other conventional direct control methods, and also direct impedance control techniques may be implemented in this layer, as appropriate. An intermediate layer may perform signal conditioning, information fusion, determination of task characteristics, and behavior planning. The top layer is considered the most "intelligent" module of the control system. It performs process monitoring and task control operations, at a relatively low bandwidth, using preprocessed sensory information from the lower layers.

Generally, the operation information; and particularly, the impedance information, should be fed back to all the levels in the control system. The task information at high level, motion path at intermediate level, and control signals at low level should be updated online using sensory information. Conventional robot controllers, particularly the old designs; for example, the Unimate Puma controller, are not suitable for implementing impedance based flexible manipulation. Usually commercial controllers have motion-based control techniques, and provide only a user interface at an upper level. Users cannot access low-level signals and modify the direct control algorithms in the controller, as needed in the present research. The host computers and the task programming languages that are provided in these commercial

control systems are usually too restrictive; they do not allow sensor-based control strategies to be implemented, and their computational power is very low. As a result, researchers who acquire commercial robots to investigate robot control techniques typically face the following scenario, due to lack of suitable robots or funds and time to develop new and appropriate robots [3]:

> A new robotics research lab purchases a commercial robot to undertake experimental control studies. The lab soon realizes that the programming language and the host computer are too limited, and undertakes to rip out the computer system and implement its own. The only catch is that a detailed specification of the servo system is needed, and if the lab is lucky the robot manufacturer will agree to provide this information.
>
> After 2 years the lab has finally managed to get the arm under its own computer control with a custom-made planning and control system. The lab then decides that it is important to include contact sensing, but the remaining servo system and bandwidth are not conducive for incorporating such sensors. ... ...

In order to perform flexible manipulation using a commercial robot, an open architecture controller is required, which will overcome the problems described in the above scenario, and mentioned previously in the chapter. The basic requirements of such a controller are discussed next. Its development and implementation will be presented in chapters 2, 7, and 8, where issues of hardware structure, implementation model and software architecture will be discussed. The main requirements of the controller can be summarized as follows:

- **Sensory feedback.** In most commercial designs of robot control systems, sensory data are fed back only to the position servo controllers. The high-level procedures of motion planning and task control adopt an open-loop architecture. As a result, tasks

have to be programmed in motion commands, with no flexibility of modification and error resolution at the task level. The controller that is developed in the present work should provide sensor feedback capabilities to both behavior (motion) planning and task control levels in addition to the servo level. This would provide the robot controller the capability to update online its operation and desired motion, based on sensory information, and thereby schedule new tasks accordingly, perhaps incorporating knowledge-based intelligent decision making.

- **Real-time intelligence.** In a robot control system, proper real-time performance of the low-level controller has to be guaranteed. At the same time, the control system has to perform some decision-making in order to ascertain the system performance and so as to plan the task properly. Again, knowledge-based intelligent decision making capability, in real time, will be important. The cooperation, synchronization and communication between the real-time and intelligent subsystems would be required here.

- **Open architecture.** Implementation of new functions when needed would be desirable of a flexible control system for robots. A client-server architecture can make a robot control system open to further development. Changes to robot functionality can be realized through modifying high-level modules, without altering the low-level, open-architecture control system.

1.6 Objectives of the Research

The general objective of the research presented in this dissertation is to investigate, develop, and implement some critical technologies of robotic flexible manipulation. Pertinent research issues include impedance sensing for online process monitoring and interpretation, and multi-level and multi-mode real-time intelligent control. Specific objectives will include developments and implementations in the following areas:

- Development of an analytic framework for impedance-based flexible robot manipulation;
- Development of impedance sensing and estimation technologies, assuming that the conventional motion sensor information is available, and the actuator effort may be sensed based on motor current;
- Processing and interpretation of impedance information to characterize the robotic process, and particularly, the processed material and its environment;
- Development and implementation of an open architecture real-time intelligent controller for the robotic system;
- Development of a high-level, intelligent, task control system;
- Application of the robotic system to food processing and other industrial situations.

As an important practical application, the research will focus on robotic fish processing [28]. Reliable, accurate, fast, and cost-effective techniques are needed for real-time sensing and control of such processes, in view of the present trend for automation of the associated industries. One other industrial application will be considered, however, in the field of industrial manufacturing, in order to demonstrate the diversity of the technology that is developed in the present research.

1.7 An Example Domain

Fish processing industries can considerably benefit from advanced robotic technology and soft automation [36]. Nevertheless, by and large, they employ outdated technology. In particular, many machines used in the fish processing industry are based on turn-of-the-century designs. These are known to be inefficient, wasteful, costly to operate and maintain, and not particularly user friendly. For example, the "Iron Butcher" machines which are commonly used in the head

cutting operation of fish, waste about 5% of the useful meat, each percentage point representing a loss of approximately $5 million a year for the province of British Columbia. Furthermore, the older technology of automation is based on "hard" or fixed mechanical configurations that lack flexibility. In "hard automation", product changeover may involve significant down times and retooling and fixturing costs. Instead, robot technology provides "flexible automation" where product changeover will involve fast and soft changes, based on software and programmable tools and fixtures. In view of varying consumer needs and demands, increased product costs, raw material limitations, and product quality concerns, the concept of small-batch production incorporating "product-on-demand" procedures would be important, which would be more amenable to flexible, robotic automation. Apart from increasing productivity and product quality, robotic technology will help release humans from intensive, monotonous and semi-skilled work in unpleasant and hazardous environments. Furthermore, due to the seasonal nature of some sectors of the food processing industry, it would be difficult to maintain a trained and high-paying workforce, which would introduce seasonal retraining costs, and will give rise to concerns of quality, efficiency, and reliability.

Fish are soft, deformable, inhomogeneous, and anisotropic natural objects. Instrumenting a fish for processing is not practical. Consider a salmon steak slicing task (see Figure 1.5) where a fish is sliced up to its backbone, and then the steaks are removed from the bone using a transverse cut. Steaks that are produced in this manner will be boneless. It is fairly difficult to measure the position of the backbone for each fish. It is also difficult to move the robotic cutter to the exact backbone position even if it is known, because the fish will be deformed during processing, and furthermore, some shifting of the fish will be possible due to soft handling. Position control only is not adequate to properly perform this task. Obviously, the mechanical impedance of cutting a bone would be much higher than that of cutting meat. Hence, online detection of the backbone using impedance information at the cutting interface would be very helpful for commanding the robot cutter in carrying out this task. In this manner,

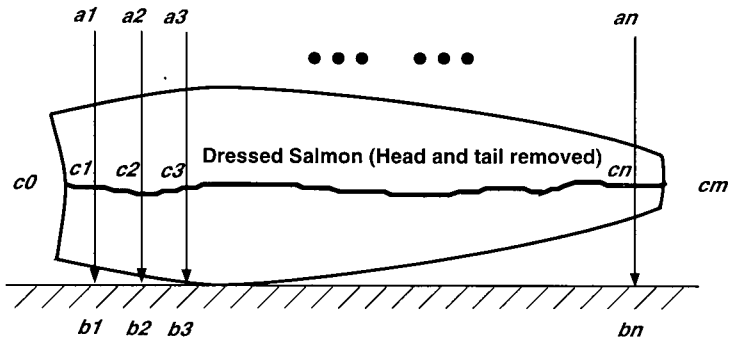


Figure 1.5: Illustration of a salmon slicing task.

the uncertainties of the cutting position and the process environment in task description could be remedied by using impedance specifications.

The removal operation of steaks from the backbone, in the present example, is somewhat more difficult. Again the position of the bone cannot be exactly specified. Cutting into bone would degrade the product and may need further processing, and may be interpreted as a failure of the process. Impedance information can overcome this problem, by detecting the backbone and changing the cutter orientation so as to slightly retract it in the transverse direction, thereby avoiding cutting into the bone. This operation provides an excellent test case for our research, with respect to the following considerations:

- Online estimation of mechanical impedance at the cutting interface is necessary in the detection of bones.
- Feedback of (pre-processed) sensory data to motion planning level and task control level of the system is required.
- Impedance interpretation and intelligent decision-making are essential in characterizing the fish and the processing interface, and in ascertaining the process performance.
- The performance of the real-time response of the cutter controller should be guaranteed for good cutting quality, while retaining the knowledge-based decision making capability.

- An available PUMA 560 robot has to be used, whose controller is not suitable for carrying out the particular task.

This example will be used as the test case throughout the thesis. In Chapter 8, a task program with position uncertainties and impedance requirements will be given for this task.

1.8 Thesis Outline

The thesis in its entirety consists of four main segments which will present: (1) introduction and relevant background; (2) development of an analytical framework for robot dynamics and process impedance, and implementation of impedance sensing technologies; (3) robot control for flexible manipulation using impedance information; and (4) development and implementation of an open architecture, real-time, intelligent control system for the present class of robotic applications.

The first segment of the thesis consists of chapters 1 and 2. Research objectives, motivation, rationale, and some background material have been given in the present chapter. Chapter 2 will discuss a research testbed consisting of a Puma 560 robot manipulator. Mechanical structure, kinematics, and the controller hardware will be addressed. The chapter will focus on the modeling of the system, formulation of the kinematics, and customizing the controller.

The second segment of the thesis will focus on modelling of process impedance in conjunction with robot dynamics, and development of related techniques of impedance sensing; which are covered in Chapter 3 and Chapter 4. Chapter 3 will discuss an appropriate impedance model, possible techniques of impedance estimation, and associated estimation of acceleration, velocity, and driving torque. To simplify the problem, the algorithms will be developed for a single-degree-of-freedom (DOF) system; for example, a single-link manipulator. In Chapter 4, these modelling concepts and the analytical basis will be extended to a multiple DOF robot system, using differential kinematics, and the corresponding algorithms of impedance

estimation will be given. Feedback linearization technology will be used to handle the highly nonlinear robot dynamics. An industrial application of online estimation and interpretation of process impedance, in a CNC router machine, will be presented to demonstrate the developed technology.

The third segment of the thesis will address control method in sensor-based flexible manipulation. A hierarchical system structure will be given in Chapter 5, which contains the following five layers: hardware layer, I/O layer, servo control layer, motion planning layer, and high-level task control layer. Information feedback will be designed for all levels of the system, in different formats. High-level intelligent task control is a new concept that is closely investigated in the present research, and the entire Chapter 6 will be devoted to this topic.

The fourth segment of the thesis will consist of Chapter 7 and Chapter 8. A software-oriented implementation model of real-time open-architecture robot control system (ROACS) will be developed and presented in Chapter 7. A client-server architecture will be adopted for the control system design, with emphases on the real-time performance of robot control servers and the artificial intelligence (AI) of client software. The prototyping of ROACS will be given in Chapter 8. The QNX real-time operating system has been selected as the implementation platform, in this context.

Chapter 9 will conclude the thesis. Major contributions and some limitations of the present research will be indicated. Directions of possible future work will be given as well, in the concluding chapter.

Chapter 2

Customization of the Control System of an Industrial Robot

Although the technologies developed in this research will be generally applicable to robot systems and machine tools, our testbed is based on a Puma 560 industrial manipulator. This robot has a number of limitations in implementing user-developed control schemes and, particularly, impedance sensing and control procedures as developed in the present work. Hence, considerable redesign and modification of the robot controller would be necessary. This chapter will first discuss relevant considerations of the manipulator; specifically, at the high level, the robot structure and kinematics; and at the low level, the characteristics of sensors and actuators, friction, and dynamics of the driving mechanism. Next, the characteristics and flexibility of robot control that is required in the present research, and associated complexity will be discussed. A suitable hardware system will be designed, with software that will enable the system to fully extend its flexibility. In many cases, robot control is taken to mean the control of a robotic workcell. This is true because, in order to perform a real task, a manipulator will have to cooperate with other machine tools and devices, and interact with the environment. Proper control and communication among workcell components, which are essential for this purpose, will be examined as well. In the following sections of this chapter, a Unimate robot controller will be analyzed, and then modified for use as a more flexible, open-system controller. The focus of the chapter is to design and develop a PC-based open architecture controller from both hardware and software points of view. In this sense, it will be a straightforward task to implement user-developed control schemes and algorithms. Implementation details of the controller will be given. Coordination of physical components as well as computational modules, will

be implemented and carried out by networking through the real-time operating system QNX, as important for robotic workcell control. The architecture, implementation model, and the realization of a sensor-based, real-time, open-architecture control system for robotic processes, based on the robot controller that is developed and presented in this chapter, will be discussed subsequently in the thesis.

The chapter is organized as follows: Section 2.1 will introduce the structure and driving mechanism of Puma 560 industrial robot. Section 2.2 will discuss the kinematics problems of the robot. Section 2.3 will give details on customized development of the commercial robot controller into an open architecture system. Section 2.4 will summarize the chapter.

2.1 Puma 560 Structure and Driving Mechanism

Puma 560 robot manipulator is quite popular in both laboratory research and industrial applications [21]. It is designed to adapt to a wide range of applications [83]. The robot arm consists of six revolute joints connected in series, each controlled by a permanent-magnet DC motor (see Figure 2.1). It possesses six degrees of freedom, and consequently, the end-effector can be moved to an arbitrary position with arbitrary orientation in a three-dimensional workspace. Therefore, it is sufficiently flexible for performing a variety of tasks.

2.1.1 Puma 560 structure

The components of the robot arm are the trunk, shoulder, upper arm, forearm, and the wrist, all consisting of revolute joints. The first joint is mounted within the trunk, and rotates the rest of the manipulator about a vertical axis. The second and third joints move the upper arm at the shoulder and the forearm at the elbow, respectively, about horizontal axes. Joint one is called the waist joint, and the joints 2 and 3 are called arm joints. With the waist and arm joints, the robot end-effector can be moved to an arbitrary position in a three-dimensional work space;

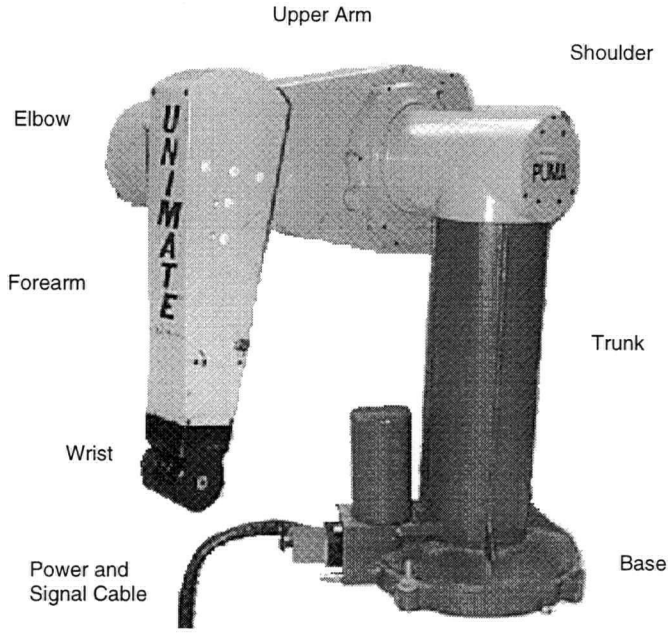


Figure 2.1: A view of the Puma 560 robot arm.

however, the orientation cannot be arbitrarily controlled, which will need further three degrees of freedom. The wrist has three joints which are used for end-effector orientation, as required. Joints 4, 5, and 6 rotate the wrist along the forearm, bend the end-effector about a horizontal axis at the wrist, and rotate the end-effector about the bent wrist axis, respectively. These three revolute wrist axes pass through a common point — the wrist center, forming a spherical wrist, and thereby somewhat simplifying the robot kinematics.

All six joints are equipped with gears. Although the gears amplify the motor torques, the effect of the joint friction will be amplified as well. Furthermore, a gear transmission makes the actuator less back-drivable. While this characteristic is an advantage in motion control, it somewhat hinders the implementation of implicit sensing technologies in our sensor-based control procedure. Although Puma 560 is not an ideal robot for use in industrial applications that incorporate implicit sensing technologies, it is a robust device which has shown to work in laboratory experiments, as will be illustrated in this thesis.

Each motor in the Puma robot arm contains an incremental encoder, and a potentiometer that is driven through a 116:1 gear reduction mechanism. The potentiometers are used to initialize the Puma; that is, to establish its absolute position. Due to wear and tear of the present robot, the potentiometers do not work properly. Hence the initialization is done by reading the initial positions from a data file. The incremental encoders are mounted on the shafts of the motors and provide the incremental rotation and velocity of these motors, for the joint controllers of the robot. Encoders are the only original sensors in the robot arm that we can use.

The servomotors for the major axes (joints 1, 2, 3) are equipped with electromagnetic brakes. These brakes are activated when power is off. For example, when power is removed from the motors, the brake power would be removed as well, thereby locking the robot arm in that position. This safety feature eliminates the risk of injury or damage that could result from the arm collapsing if power is accidentally removed. Some specifications that are useful in the design of the robot control system are listed in Table 2.1.

2.1.2 Joint friction

A typical friction versus velocity behavior of a robot joint is shown in Figure 2.2. Since the joints are not directly driven, but employ gear transmissions, friction is amplified by the gear reduction, which can cause significant problems in joint control.

From Figure 2.2, it is clear that the frictional dissipation may be primarily separated into three components: stiction, Coulomb friction, and viscous friction. Stiction occurs at standstill, and contributes an unstable (negative slop) friction effect at low relative speeds; Coulomb friction (force) is constant under given loading conditions, and viscous friction varies linearly with velocity. Specifically, stiction effects are dominant at very low speeds, while viscous friction is dominant at high speeds. Also, these three components do not have identical characteristics in opposite moving directions. In view of this directional property, we use τ_s^+ and τ_s^- for stiction,

Table 2.1: Specifications of Puma 560 robot.

Item	**Specifications**	**Comments**
Axes	6 revolute joints	Wrist joint axes intersect at a common point.
Position sensors	Incremental encoders and potentiometers	Potentiometers malfunction.
Force sensors	None	
Actuators	PM DC motors	
Static force at tool	58N (13.0lb) maximum	
Accuracy	±0.1mm	
Tool acceleration	1g maximum	
Joint motion limits	320, 250, 270, 300, 200, 532	in degrees.
Encoder index resolution	5.7557, 3.3392, 6.7098, 4.7358, 5.0062, 4.6918	degrees per motor revolution.
Max. joint speed	82.1, 53.5, 122.1, 227.8, 241.4, 227.8	degrees per second.
Max. linear velocity of end-effector	468	mm per second.
Max. joint acceleration	12.7, 8.3, 38.0, 70.9, 75.2, 70.9	degrees/s^2
Max. linear acceleration of end-effector	4.17	m/s^2

τ_c^+ and τ_c^- for Coulomb friction, and B^+ and B^- for viscous friction coefficient, in the positive and the negative directions of relative motion, as shown in Figure 2.2.

There is a critical point ϵ which is of importance. The region where moving speed is less than ϵ is termed the low speed regime. In this regime, stiction is dominant, and due to the negative damping curve, control is unstable. Also the friction in this regime is inconsistent and nonlinear. When speed is greater than ϵ, viscous friction becomes dominant, and hence the frictional behavior would be quite linear with speed and stable, in this high-speed regime. The friction in the high-speed regime is quite estimatable, whereas it is not so in the low speed regime. Even though a representative curve of friction for the entire regime is indicated in the

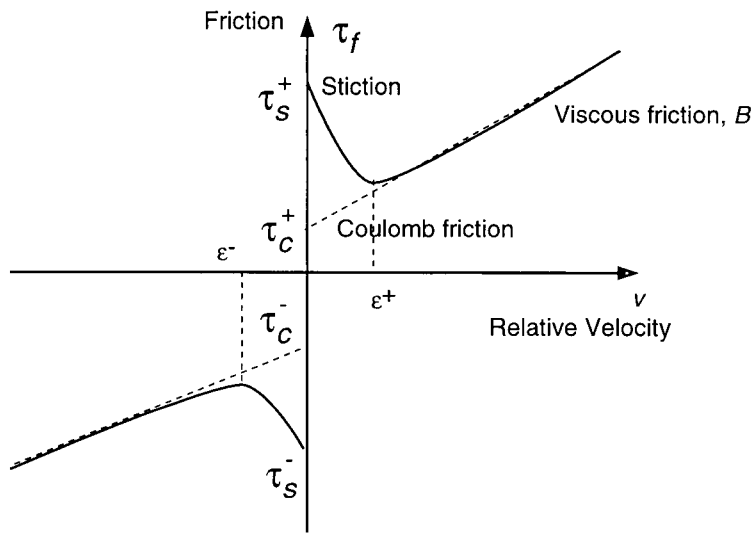


Figure 2.2: A typical friction versus velocity behavior of a robot joint.

figure, the friction-velocity behavior may not exactly follow this model. Moreover, the joint motion at low speeds is known to be jerky due to stiction and possesses a negative (unstable) friction curve. This makes estimation of friction and inertia forces extremely difficult at low speed.

In general, the dashed line in Figure 2.2 represents the simplified friction model,

$$\tau_f = B\dot{\theta} + \tau_c \tag{2.1}$$

where τ_f is the frictional torque, θ is the joint angle, slope B represents the viscous damping constant, and the offset τ_c represents the Coulomb frictional torque. The Coulomb frictional torque is modeled as:

$$\tau_c = \begin{cases} 0 & \text{if } \dot{\theta} = 0 \\ \tau_c^+ & \text{if } \dot{\theta} > 0 \\ \tau_c^- & \text{if } \dot{\theta} < 0 \end{cases} \tag{2.2}$$

The friction model can be modified by including the stiction, as

$$\tau_f = \begin{cases} B\dot{\theta} + \tau_c^+ & \text{if } \dot{\theta} > 0 \\ \tau_s^+(\text{max.}) & \text{if } \dot{\theta} = 0, \text{ and } \Delta\theta > 0 \\ \tau_s^-(\text{min.}) & \text{if } \dot{\theta} = 0, \text{ and } \Delta\theta < 0 \\ B\dot{\theta} + \tau_c^- & \text{if } \dot{\theta} < 0 \end{cases} \tag{2.3}$$

where $\Delta\theta$ denotes the error between the commanded position θ_d and the actual position θ.

This model is used for friction estimation in impedance sensing, and in compensation for control. The parameters, according to Armstrong and Corke [5, 22], are listed in Table 2.2.

Table 2.2: Friction parameters of Puma 560 — motor referenced (Nm and Nms/rad). Positive and negative joint velocity are indicated by the superscripts. The column B is the mean of B^+ and B^-.

Joint	τ_s^+	τ_c^+	B^+	τ_s^-	τ_c^-	B^-	B
1	0.569	0.435	$1.46e^{-3}$	-0.588	-0.395	$-1.49e^{-3}$	$1.48e^{-3}$
2	0.141	0.126	$0.928e^{-3}$	$-95.1e^{-3}$	$-70.9e^{-3}$	$-0.705e^{-3}$	$0.817e^{-3}$
3	0.164	0.105	$1.78e^{-3}$	-0.158	-0.132	$-0.972e^{-3}$	$1.38e^{-3}$
4	$14.7e^{-3}$	$11.2e^{-3}$	$64.4e^{-6}$	$-21.8e^{-3}$	$-16.9e^{-3}$	$-77.9e^{-6}$	$71.2e^{-6}$
5	$5.72e^{-3}$	$9.26e^{-3}$	$93.4e^{-6}$	$-13.1e^{-3}$	$-14.5e^{-3}$	$-71.8e^{-6}$	$82.6e^{-6}$
6	$5.44e^{-3}$	$3.96e^{-3}$	$40.3e^{-6}$	$-9.21e^{-3}$	$-10.5e^{-3}$	$-33.1e^{-6}$	$36.7e^{-6}$

In the experiments related to the present research, the point ϵ, which represents the lowest speed where model (2.1) is still valid, was found to be approximately 1°/s. This is usually outside the region of interest in continuous robot operation.

2.1.3 Electro-mechanical dynamics

Two different sizes of motors are used in the Puma 560 robot: a larger and more powerful unit for the base and arm axes (joints 1 through 3) and a smaller unit for the wrist axes (joints 4 through 6). Figure 2.3 shows a simplified joint model of the dynamics, including

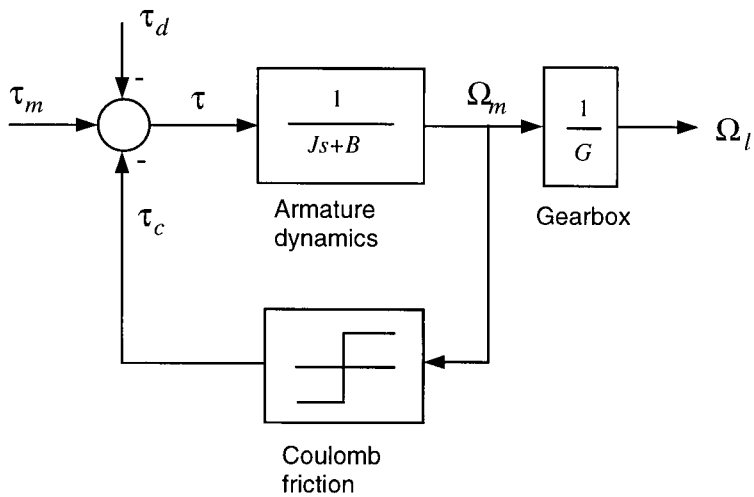


Figure 2.3: Block diagram of a model for motor mechanical dynamics.

the motor and the gearbox. In this figure, τ_m is the magnetic torque generated by the motor, part of which is applied to overcome the equivalent mechanical impedance of the motor-joint mechanism $(Js + B)$, resulting in the motor speed Ω_m. Note that J and B are the equivalent moment of inertia and the angular viscous damping constant, respectively, of the motor and joint combination. Joint speed Ω_l is simply the output speed of the joint transmission, which is the motor speed reduced by the gear ratio G. The remainder of τ_m is used to overcome the Coulomb friction τ_c and the external disturbances (load from the rest of the manipulator) τ_d. The pertinent equations are:

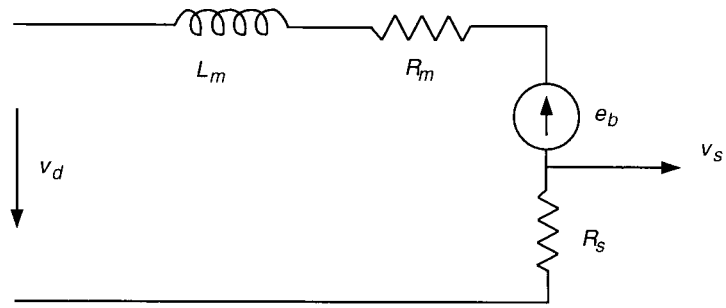
$$\tau_m = k_\tau i_m \tag{2.4}$$

$$\Omega_m = \frac{\tau}{Js + B} \tag{2.5}$$

$$\Omega_l = \frac{\Omega_m}{G} \tag{2.6}$$

$$\tau = \tau_m - \tau_c - \tau_d. \tag{2.7}$$

Here k_τ is the motor torque constant and i_m is the armature current, as shown in Figure 2.4, which illustrates the motor current loop. The parameters L_m and R_m are the leakage inductance



R_m= 2.1Ω for base motors, 6.7Ω for wrist motors;
R_s = 0.2Ω for base motors, 0.392Ω for wrist motors.

Figure 2.4: Electrical model of robot servo motors.

and the resistance, respectively, of the motor armature; e_b is the back EMF; and R_s is a shunt resistance which is very small and in series with the motor armature. The shunt resistance is of particular interest in the motor drive; specifically, the shunt voltage v_s is used as a feedback signal in the power amplifier to maintain the armature current proportional to the input control signal v_d. This signal is also used to measure the motor magnetic torque τ_m in the controller that is developed in the present research, as described in Section 2.3 of this chapter. Measured values of the motor inertia, torque constant parameters, and gear ratio are given in Table 2.3.

Table 2.3: Motor inertia and torque constant values.

Joint	1	2	3	4	5	6
J (kgm^2)	$200e^{-6}$	$200e^{-6}$	$200e^{-6}$	$33e^{-6}$	$33e^{-6}$	$33e^{-6}$
k_τ (Nm/A)	0.223	0.226	0.240	0.069	0.072	0.066
G	62.61	107.36	53.69	76.01	71.91	76.63

The motors are driven by linear current power-amplifiers. The output currents (motor armature currents) of the power amplifiers are determined by the feedback relation:

$$i_m = \frac{v_d}{CR_s} \tag{2.8}$$

Here, R_s is the shunt resistance and C is a feedback gain constant with $C = 6.06$ for Unimate

amplifiers. Note that equation (2.8) represents steady state conditions. The bandwidth of the amplifiers reaches 1000Hz which is much higher than the approximate value of 3Hz ($\sqrt{\frac{B}{J}}$) for the mechanical system. Hence equation (2.8) predominantly holds in the present work.

The maximum control input for Puma 560 is $v_d = 10$V. The shunt resistances for the base joints are 0.2Ω, and for the wrist joints they are 0.392Ω. The maximum current can be calculated as:

$$i_{m_{max}} = v_{d_{max}}/(6.06 \times R_s) = \begin{cases} 10/(6.06 \times 0.2) = 8.25\text{A} & \text{for base joints} \\ 10/(6.06 \times 0.392) = 4.21\text{A} & \text{for wrist joints} \end{cases} \tag{2.9}$$

Note that the power supply to the amplifiers provides a voltage of 40V and the armature resistance is about 2Ω for the base motors and 6Ω for the wrist motors. It follows that the maximum armature current is controlled by the amplifier rather than the armature resistance.

2.1.4 Calibration of Puma 560

Calibration of sensors and actuators will be described in this section, while the calibration of robot kinematics will be discussed in the next section. Puma 560 uses both optical encoders and potentiometers for joint position sensing. The potentiometers are rated from 0V to 5V. This analog voltage signal represents the absolute position of the joints, and is quite noisy. The optical incremental encoders generate digital pulses that can be counted to determine the relative joint positions, and are more reliable than the absolute positions, in terms of noise protection. The calibration constants of the encoders, specifically, count per motor-revolute ($m.rev$) and count per link-radius ($l.rad$), are listed in Table 2.4.

The control voltage, armature current, motor torque, and the joint torque of the actuators are shown in Table 2.5.

Table 2.4: Encoder resolution.

Joint	1	2	3	4	5	6
Count/m.rev	1000	800	1000	1000	1000	1000
Count/l.rad	9965	13727	8548	12102	11447	12204

Table 2.5: Joint torque parameters of Puma 560.

Joint	1	2	3	4	5	6
Control Voltage (V)	1	1	1	1	1	1
Output Current (A)	0.825	0.825	0.825	0.421	0.421	0.421
Motor Torque (Nm)	0.184	0.186	0.198	0.0290	0.0303	0.0278
Joint Torque (Nm)	11.52	20.02	10.63	2.21	2.18	2.13

2.2 Kinematics

2.2.1 Home position and DH parameters

The home position of the laboratory robot is set as shown in Figure 2.5. This configuration is convenient in fish cutting applications. In particular, the shoulder and elbow singularities (see Chapter 5) can be avoided by this means. The joint coordinate frames are set as illustrated in the right hand side of Figure 2.5. The kinematics of Puma 560 are dependent upon four non-zero length parameters and a number of axis twist angles which are taken as either exactly 0 or exactly 90 degrees. The lengths, which may be directly measured, are:

1. Distance between the shoulder and elbow axes along the upper arm link (a_2);

2. Distance from the elbow axis to the center of the spherical wrist along the forearm (d_4);

3. Offset between the axes of joint 4 and the elbow (a_3);

4. Offset between the waist axis and the joint 4 axis (d_3).

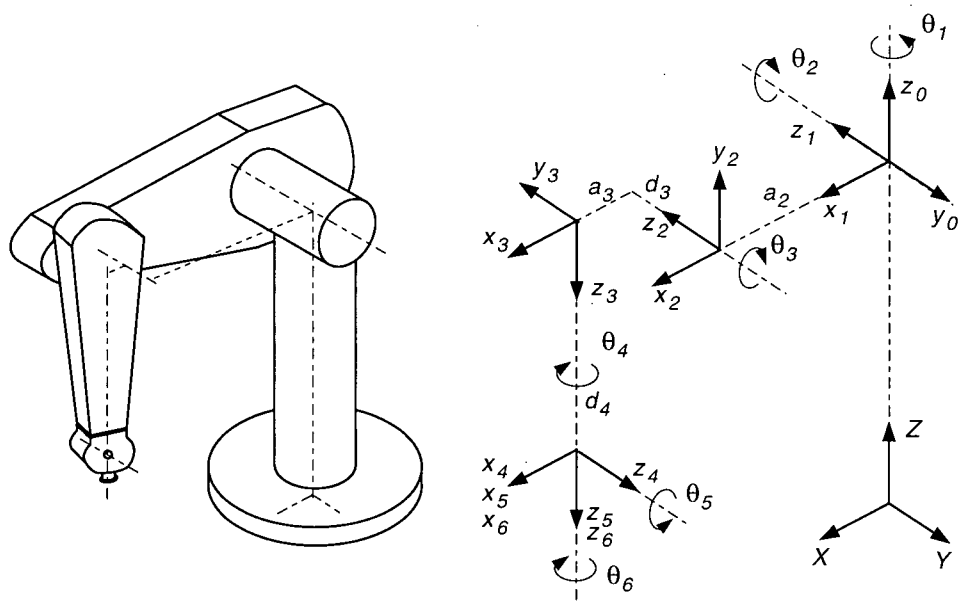


Figure 2.5: Home configuration and coordinate frames for the PUMA 560 manipulator.

The Denavit-Hartenberg (DH) parameters [10, 23, 42] based on the joint coordinate frames of Figure 2.5, are listed in Table 2.6.

Table 2.6: The DH parameters of Puma 560.

i	α_i (rad)	a_i (m)	d_i (m)	θ_i (rad)
1	$\frac{\pi}{2}$	0	0	-2.79< θ_1 < 2.79
2	0	0.4318	0	-1.5< θ_2 <2.86
3	$\frac{\pi}{2}$	0.0203	0.1254	-1.5 < θ_3 < 3.2
4	$\frac{\pi}{2}$	0	0.4318	-2.2 < θ_4 < 2.62
5	$-\frac{\pi}{2}$	0	0	-1.74 < θ_5 < 1.74
6	0	0	0	-4.64 < θ_6 < 4.64

2.2.2 Forward kinematics

The homogeneous transformation that describes the relation between the coordinates $\sum_i$ and $\sum_{i-1}$ is [42]

$$^{i-1}T_i = T_T(Z_i, d_i)T_R(Z_i, \theta_i)T_T(X_{i-1}, a_{i-1})T_R(X_{i-1}, \alpha_{i-1}) \tag{2.10}$$

where $T_T(X, a)$ denotes a translation along the X axis through a distance a, and $T_R(X, \alpha)$ denotes a rotation about the X axis by an angle α. From equation (2.10) we obtain,

$$\begin{aligned} ^{i-1}T_i &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &\times \begin{bmatrix} 1 & 0 & 0 & a_{i-1} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_{i-1} & -\sin\alpha_{i-1} & 0 \\ 0 & \sin\alpha_{i-1} & \cos\alpha_{i-1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \cos\alpha_i\sin\theta_i & \cos\alpha_i\cos\theta_i & -\sin\alpha_i\cos\theta_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{2.11}$$

Note that $^{i-1}T_i$ is a function of the joint angle variable θ_i, as the link parameters are constants given in Table 2.6. By repetitive application of this relation, the overall homogeneous transform relating $\sum_n$ to $\sum_0$ can be expressed as,

$$^0T_n = {}^0T_1\ {}^1T_2\ \cdots\ {}^{n-1}T_n. \tag{2.12}$$

It follows that ^0T_n is a function of θ_1, θ_2, ..., θ_n.

Consider the homogeneous transform from the fixed world coordinate frame $\sum_{XYZ}$ to $\sum_0$, denoted as ${}^RT_0$, and the transform from $\sum_n$ to the end-effector coordinate $\sum_E$, denoted as ${}^nT_E$. Then, the forward kinematics can be expressed as,

$$ {}^RT_E = {}^RT_0 \, {}^0T_n \, {}^nT_E. \tag{2.13} $$

2.2.3 Inverse kinematics

The inverse kinematics problem is to solve for the joint positions corresponding to a specified configuration (position and orientation) of the end-effector, in world coordinates. The solution of the inverse kinematics problem is not unique. For example, there are singular configurations in which the solution is indeterminate. Furthermore, for a given end-effector configuration, there can be two or more link configurations. For a specific task we should avoid the singularities, and also resolve multiple solutions by choosing the configuration for inverse kinematics that is closest to the immediately prior configuration. In particular, note that multiple solutions can be avoided by using the differential kinematics or the incremental coordinate (Jacobian) formulation. A detailed discussion on singularities will be given in Chapter 5.

Although analytic solutions do not always exist for inverse kinematics for an arbitrary robot configuration. Puma 560 adopts a special structure where inverse kinematics is solvable. The last three axes (wrist joints) intersect at one point, the wrist center. Therefore, the inverse kinematics problem is decoupled, and can be solved by using the wrist center as a common point for both the base arms and the wrist joints. In this manner, the inverse kinematics problem of Puma 560 can be solved analytically. The solution is summarized below [75]. Given,

$$ {}^0T_n = \begin{bmatrix} \mathbf{n} & \mathbf{o} & \mathbf{a} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{2.14} $$

with the column vectors of orientation and position **n o a p** known, the inverse kinematics solution is:

(1) Solution for θ_1

$$\theta_1 = \tan^{-1}\frac{p_x}{p_y} + \sin^{-1}\frac{d_3}{r} \tag{2.15}$$

or

$$\theta_1 = \tan^{-1}\frac{p_x}{p_y} + \pi - \sin^{-1}\frac{d_3}{r} \tag{2.16}$$

where

$$r = \sqrt{p_x^2 + p_y^2}$$

(2) Solution for θ_2

$$\theta_2 = \tan^{-1}\frac{p_z}{V_{114}} + \phi \tag{2.17}$$

or

$$\theta_2 = \tan^{-1}\frac{p_z}{V_{114}} - \phi \tag{2.18}$$

where

$$V_{114} = C_1 p_x + S_1 p_y$$

and

$$\phi = \cos^{-1}\frac{a_2^2 - d_4^2 - a_3^2 + V_{114}^2 + p_z^2}{2a_2 r}$$

The abbreviated notations used here are,

$$C_i = \cos\theta_i; \quad \text{and } S_i = \sin\theta_i,$$

$$C_{ij} = \cos(\theta_i + \theta_j); \quad \text{and } S_{ij} = \sin(\theta_i + \theta_j).$$

(3) Solution for θ_3

$$\theta_3 = \tan^{-1}\frac{a_3}{d_4} - \tan^{-1}\frac{C_2 V_{114} + S_2 p_z - a_2}{C_2 p_z - S_2 V_{114}} \tag{2.19}$$

(4) Solution for θ_4

$$\theta_4 = \begin{cases} \tan^{-1}\frac{-V_{323}}{-V_{313}} & \text{if } S_5 > 0; \\ \tan^{-1}\frac{V_{323}}{V_{313}} & \text{if } S_5 < 0; \\ \text{undefined} & \text{if } S_5 = 0. \end{cases} \tag{2.20}$$

where

$$\begin{aligned} V_{323} &= C_1 a_y - S_1 a_x \\ V_{113} &= C_1 a_x + S_1 a_y \\ V_{313} &= C_{23} V_{113} + S_{23} a_z \end{aligned}$$

(5) Solution for θ_5

$$\theta_5 = \tan^{-1}\frac{S_5}{C_5} \tag{2.21}$$

where

$$\begin{aligned} S_5 &= -C_4 V_{313} - S_4 V_{323} \\ C_5 &= -S_{23} V_{113} + C_{23} a_z \end{aligned}$$

(6) Solution for θ_6

$$\theta_6 = \tan^{-1}\frac{S_6}{C_6} \tag{2.22}$$

where

$$\begin{aligned} S_6 &= -C_5 V_{412} - S_5 V_{422} \\ C_6 &= -V432 \end{aligned}$$

and

$$\begin{aligned} V_{412} &= C_4 V_{312} - S_4 V_{132} \\ V_{422} &= V_{332} \end{aligned}$$

$$
\begin{aligned}
V_{432} &= S_4 V_{312} + C_4 V_{132} \\
V_{312} &= C_{23} V_{112} + C_{23} o_z \\
V_{332} &= -S_{23} V_{112} + C_{23} o_z \\
V_{132} &= S_1 o_x - C_1 o_y \\
V_{112} &= C_1 o_x + S_1 o_y
\end{aligned}
$$

The appropriate choice of the inverse kinematics, among multiple solutions, should be determined by the preceding configuration of the manipulator.

2.2.4 Differential kinematics

The Jacobian matrix $\mathbf{J}$ relates the joint coordinate rates $\dot{\theta}_1$ through $\dot{\theta}_6$ to the Catesian coordinate rates of the end of the manipulator $\dot{\mathbf{x}}$, according to

$$\dot{\mathbf{x}} = \mathbf{J}\dot{\theta} \tag{2.23}$$

where

$$
\begin{aligned}
\dot{\mathbf{x}} &= [v_x \; v_y \; v_z \; \omega_x \; \omega_y \; \omega_z]^T \\
\dot{\theta} &= [\dot{\theta}_1 \; \dot{\theta}_2 \; \dot{\theta}_3 \; \dot{\theta}_4 \; \dot{\theta}_5 \; \dot{\theta}_6]^T
\end{aligned}
$$

In other words, the incremental coordinates $\delta\theta$ are related to $\delta\mathbf{x}$ in this manner.

The Jacobian is determined by the DH parameters and the configuration of the manipulator. For Puma 560 robot, it can be solved as,

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_{11} & 0 \\ \mathbf{J}_{21} & \mathbf{J}_{22} \end{bmatrix} \tag{2.24}$$

where

$$
\mathbf{J}_{11} = \begin{bmatrix} U_{211} d_3 + U_{421} U_{214} & U_{321} a_2 & -U_{411} d_4 + U_{521} a_3 \\ U_{212} d_3 + U_{422} U_{214} & U_{322} a_2 & -U_{412} d_4 + U_{522} a_3 \\ U_{213} d_3 + U_{423} U_{214} & U_{323} a_2 & -U_{413} d_4 + C_5 a_3 \end{bmatrix}
$$

$$\mathbf{J}_{21} = \begin{bmatrix} U_{221} & 0 & -U_{421} \\ U_{222} & 0 & -U_{422} \\ U_{223} & 0 & -U_{423} \end{bmatrix}$$

$$\mathbf{J}_{22} = \begin{bmatrix} U_{521} & -S_6 & 0 \\ U_{522} & -C_6 & 0 \\ C_5 & 0 & 1 \end{bmatrix}$$

and the U elements are defined by,

$$\begin{aligned}
U_{512} &= -C_5 S_6, \\
U_{522} &= -S_5 S_6, \\
U_{412} &= C_4 U_{512} - S_4 C_6, \\
U_{413} &= -C_4 S_5, \\
U_{422} &= S_4 U_{512} + C_4 C_6, \\
U_{423} &= -S_4 S_5, \\
U_{222} &= S_{23} U_{412} + C_{23} U_{522}, \\
U_{223} &= S_{23} U_{413} + C_{23} C_5, \\
U_{224} &= C_{23} d_4 + S_{23} a_3 + S_2 a_2, \\
U_{212} &= C_{23} U_{412} - S_{23} U_{522}, \\
U_{213} &= C_{23} U_{413} - S_{23} C_5, \\
U_{214} &= -S_{23} d_4 + C_{23} a_3 + a_2 C_2.
\end{aligned}$$

For a six DOF manipulator, $\mathbf{J}$ is a 6×6 square matrix. Then $\mathbf{J}^{-1}$ exists if the manipulator is not in a singular configuration; i.e., if $\det \mathbf{J} \neq 0$, and we have the inverse Jacobian relation,

$$\delta\theta = \mathbf{J}^{-1} \delta\mathbf{x}. \tag{2.25}$$

or

$$\dot{\theta} = \mathbf{J}^{-1}\dot{\mathbf{x}}. \tag{2.26}$$

from which the joint velocities can be computed, for a given end-effector velocity.

Let $\mathbf{f}_e$ denote the forces and torques exerted on the end-effector of the manipulator in the Cartesian space, and τ_e denote the joint torques which are needed to overcome these environmental forces. Then by the principle of virtual work, we obtain,

$$-\mathbf{f}_e^T \delta\mathbf{x} + \tau_e^T \delta\theta = 0 \tag{2.27}$$

for a given equilibrium configuration of the robot, where $-\mathbf{f}_e$ is the reaction force on the end-effector from the environment (hence the $-$ sign).

By substituting equation (2.25) into equation (2.27), we get,

$$\mathbf{f}_e^T \delta\mathbf{x} = \tau_e^T \mathbf{J}^{-1} \delta\mathbf{x}, \tag{2.28}$$

for some arbitary motion increment from the equilibrium configuration. Hence,

$$\mathbf{f}_e^T = \tau_e^T \mathbf{J}^{-1} \tag{2.29}$$

Thus,

$$\mathbf{f}_e = (\mathbf{J}^{-1})^T \tau_e \tag{2.30}$$

or,

$$\tau_e = \mathbf{J}^T \mathbf{f}_e. \tag{2.31}$$

Note that equation (2.31) gives the manner in which an end-effector force $\mathbf{f}_e$ is reflected at the joints of the robot. This should not be confused with the dynamic torque at the robot joints. Clearly, there can be non-zero dynamic torques at the joints even when the end-effector is free ($\mathbf{f}_e = \mathbf{0}$).

In the present research, only joint variables are measured. The speed and the contact force of the end-effector should be computed from joint variables using Jacobian transformation, after accounting for robot dynamics.

2.3 Open Architecture Robot Control System

Although from the electro-mechanical viewpoint the Puma 560 robot arm is useful in the present research, the commercial Unimate controller is completely unacceptable. Figure 2.6 illustrates the structure of the Unimate Mark II controller, where the host computer running on LSI-11/02

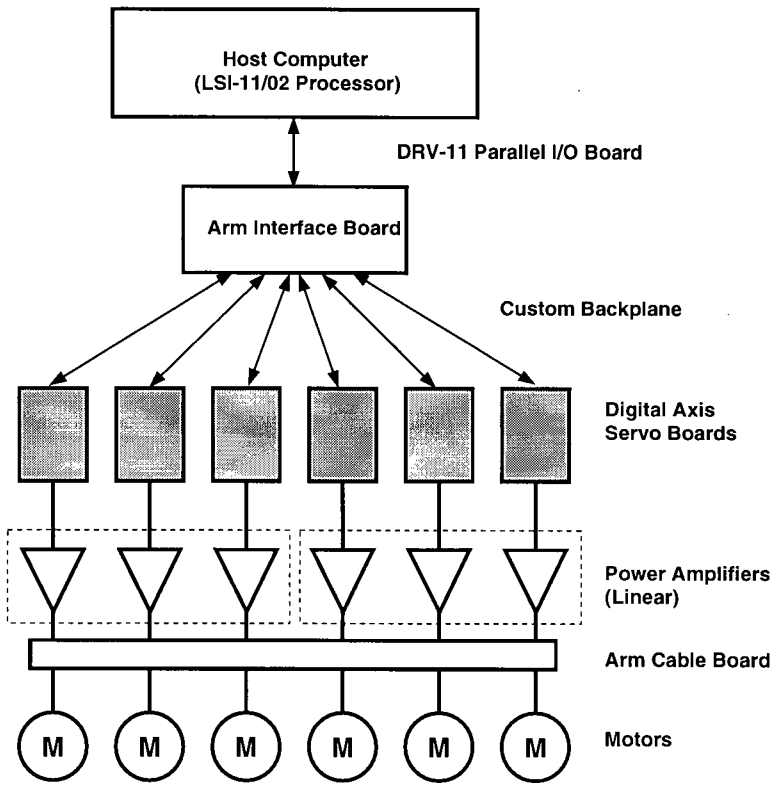


Figure 2.6: Structure of the Unimate Mark II controller.

CPU communicates with an arm interface board through a DRV-11 parallel input/output (I/O) board. The arm interface board, six joint servo boards, and a digital to analog (D/A) conversion board are connected through an internal bus. The analog outputs from the D/A board, which are the control signals, are wired to the linear power amplifiers that drive the joint motors. An arm cable I/O board provides the necessary communication capability to the robot, for sensing and actuation. The host computer also communicates with peripherals, such as, the teach pendant,

display device, and programming terminals, through DLV-11J serial ports. Task level motion commands are generated from these user-interface devices. The physical layout of the Mark II controller is shown in Figure 2.7.

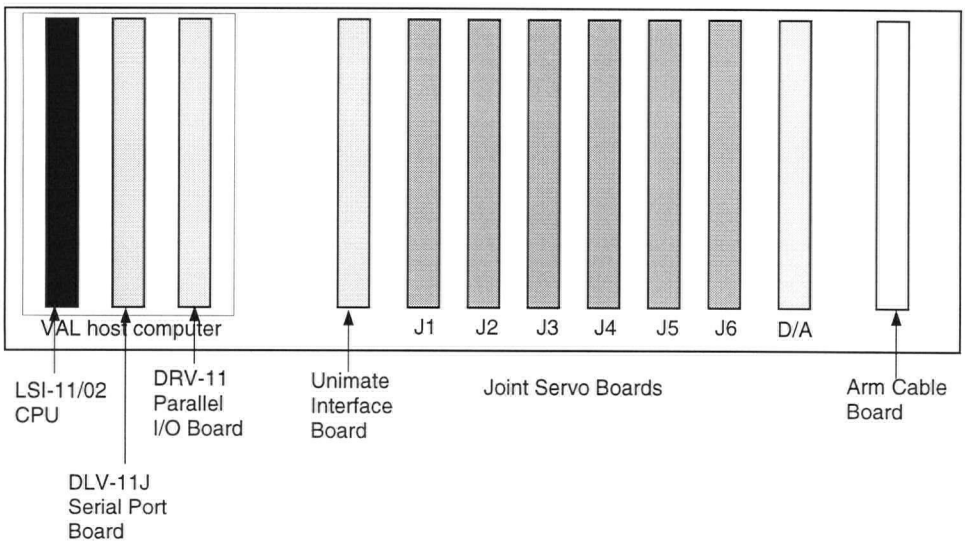


Figure 2.7: Physical layout of the commercial Mark II robot controller.

The host computer accepts commands from a pendant through serial ports and interprets them as motion specifications to the joints. These specifications, including target positions and velocities, are the commands to the joint servo modules. Joints are controlled using proportional plus derivative (PD) controllers separately by the joint servo boards. As a result, the commercial robot controller may be interpreted as a combination of separate servo systems. Only the kinematics computation of the robot is considered as an integrated activity in the host computer. All the programs in the host and the servo computers are embedded, and cannot be modified. It follows that the controller does not have flexibility or modifiability.

A new open-architecture controller is designed and implemented for the Puma 560 robot in this research, which replaces the Mark II controller. The new controller takes advantage of the existing Arm Cable Interface for sensor signals, and the power amplifiers for actuation. The entire computer system of the original controller is removed. A Pentium 90 MHz industrial PC

is used as the servo control system, and a Pentium 200 MHz machine is used as the upper level host. The system communication is based on a QNX [79] network, which will be described in Chapter 8. Additional torque sensing is also implemented in the new controller through measuring the shunt voltage. This feature is particularly essential in online impedance sensing and control. The evolution from a single customized controller into a more flexible and versatile network controller that integrates impedance sensing and control will be described in the sequel.

2.3.1 Open architecture controller

The low-level joint servo functions of the robot controller are performed by a Pentium 90 MHz industrial PC. In this servo system, the robot position signals and control efforts are measured, and made accessible to the user. Control algorithms are executed by a dedicated, high-priority process. This architecture avoids restrictions on signal processing and control algorithms due to hardware limitations, which is usually the case with DSP board-based systems. In this sense, the new controller is completely open to the user. This is a very useful feature for the controller of a research robot.

The system, as implemented in our laboratory, is illustrated in Figure 2.8. The servo controller is based on a PC ISA bus. It consists of a CPU board, two encoder cards which read the encoder signals of the six joints, one multi-functional analog to digital (A/D) conversion and digital output (DO) card which reads the potentiometer signals and the motor current signals and send digital signals to logic control devices such as the protection circuits and relay board, and one D/A card which sends analog signals to the power amplifiers. A network card provides the data communication capability to the controller.

A controller interface board rearranges the signals from the customized Mark II controller, and wires them to the connectors of the new PC based controller. Note that we have bypassed the computer systems of the original Mark II controller, and utilize the raw signals only.

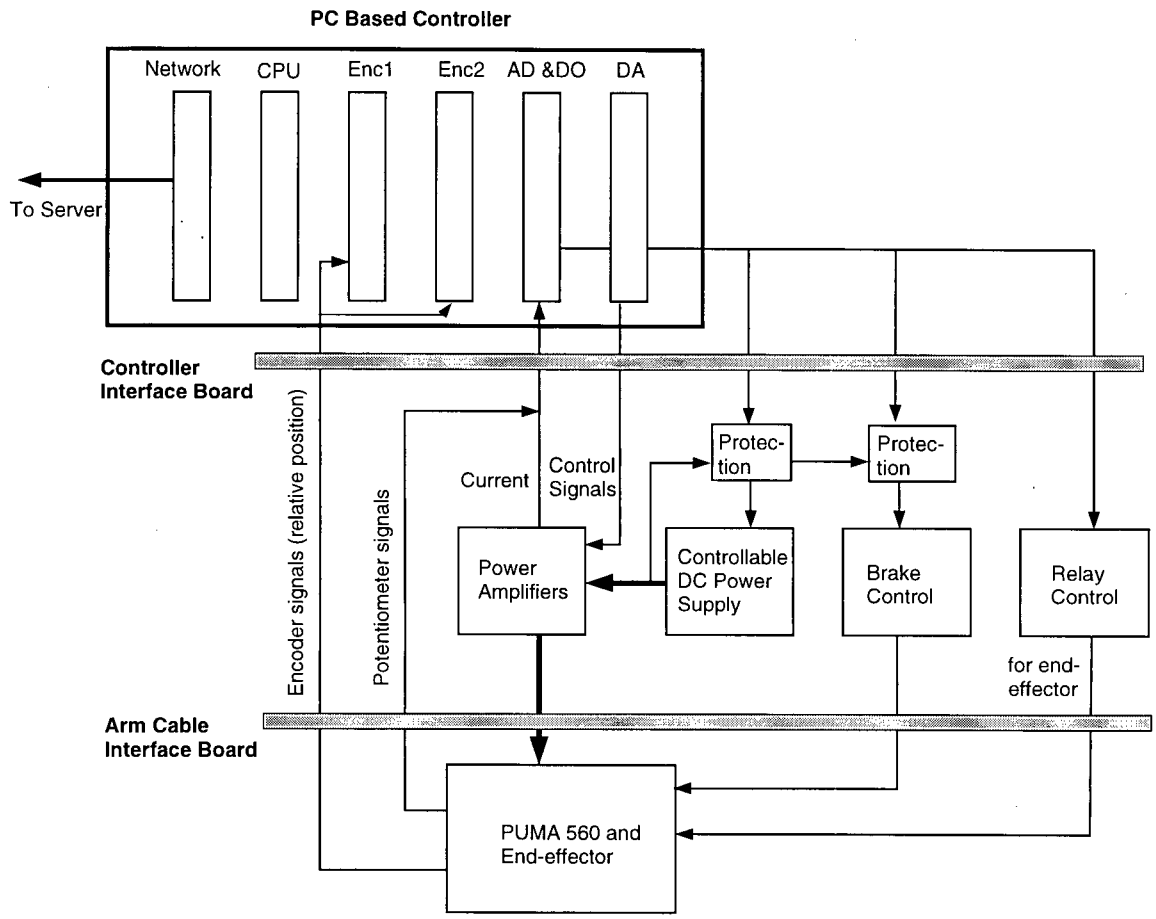


Figure 2.8: Configuration of the hardware system in the new controller.

2.3.2 Customization of the Unimate controller

The purpose of customizing the commercial Mark II controller is to acquire the sensor signals for use with the open controller and to have access to the actuators for implementing control commands. The original LSI-11/02 system and the joint servo systems have been removed. Only the arm cable interface board, the power amplifiers, and the power supplies are retained for use in the new system. The layout of the customized controller is indicated in Figure 2.9.

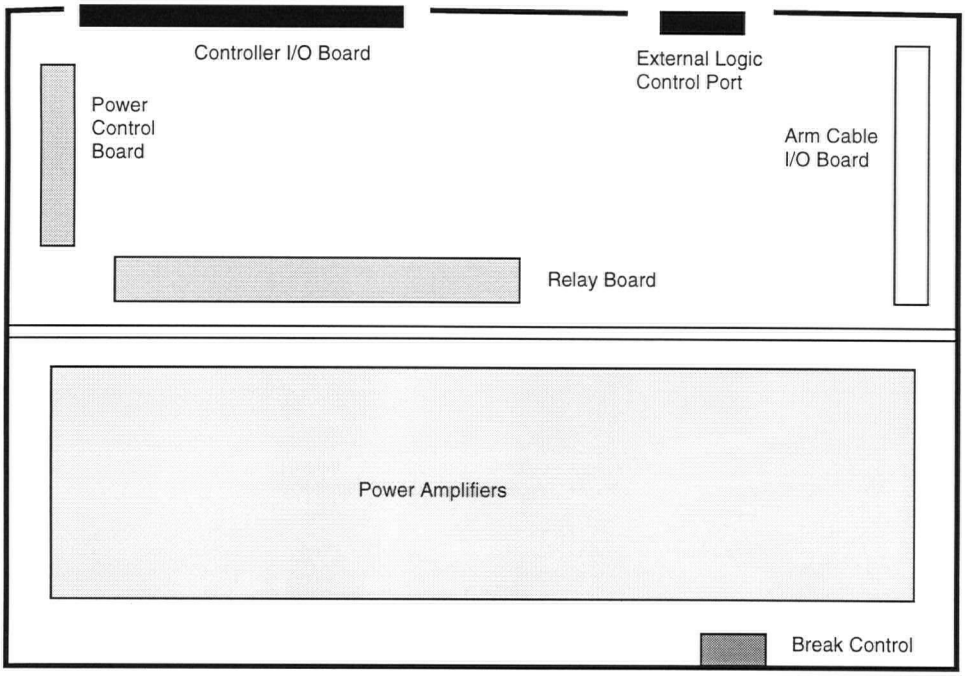
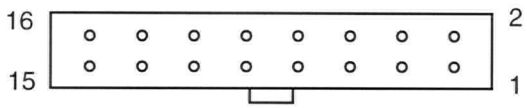


Figure 2.9: Layout of the new controller.

Power Amplifiers: The power amplifiers are connected to the D/A output. The corresponding pin assignment of the connector is shown in Figure 2.10.



2, 4, 8, 11, 14 --- Ground
1 --- Joint 2 (major assembly), joint 5 (minor assembly)
3 --- Joint 1 (major assembly), joint 4 (minor assembly)
5 --- Joint 3 (major assembly), joint 6 (minor assembly)

Figure 2.10: Connector pinout of the power amplifiers.

Arm I/O Board: The arm cable I/O board is retained unchanged. Some signals are directly connected to the controller I/O board. The signal layout is shown in Figure 2.11.

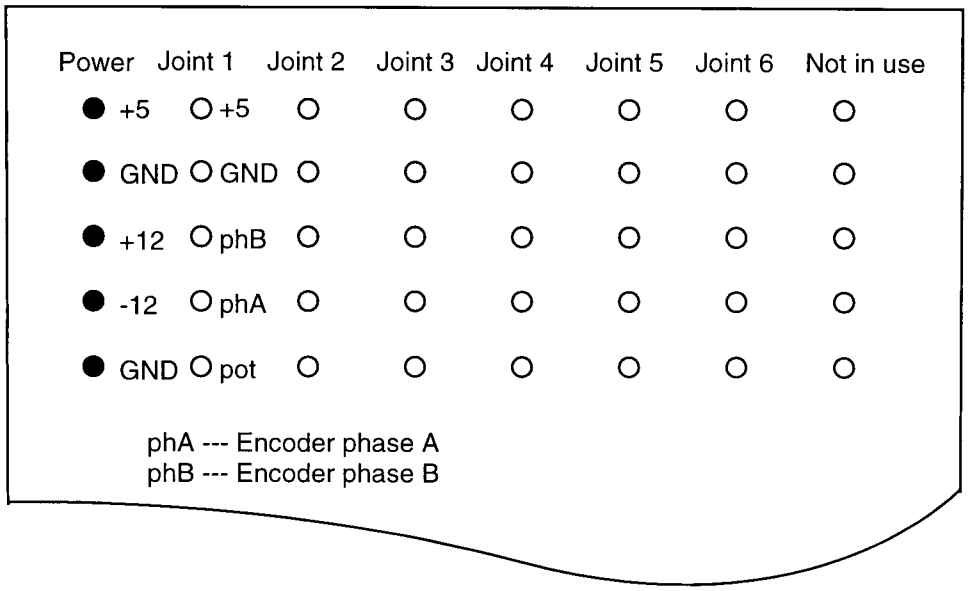


Figure 2.11: Signal configuation of the arm cable I/O board.

Relay Board: The relay board is directly connected to a DO port. Its 16 channels are fully controlled. The address of the DO port is 0x310 for the low byte and 0x311 for the high byte. The logic control circuit is shown in Figure 2.12, and described as follows:

- The channel Ch 15 is used for robot brake control. One terminal is connected to a +26V power source. Another terminal is connected to the robot brake through the connectors J99 and P99. If Ch 15 is switched on, the robot brake would be released.

- Ch 14 is used for DC power supply protection. One terminal is connected to the control windings of two power relays in series which are further connected to a 31V power supply. The other terminal is grounded. Hence, when Ch 14 is on, power relays will connect the power supplies to the current amplifiers. The relay circuit is shown in Figure 2.13. Control ports of the relays are 85 and 86.

- Ch 8 and Ch 9 are used for external AC control.

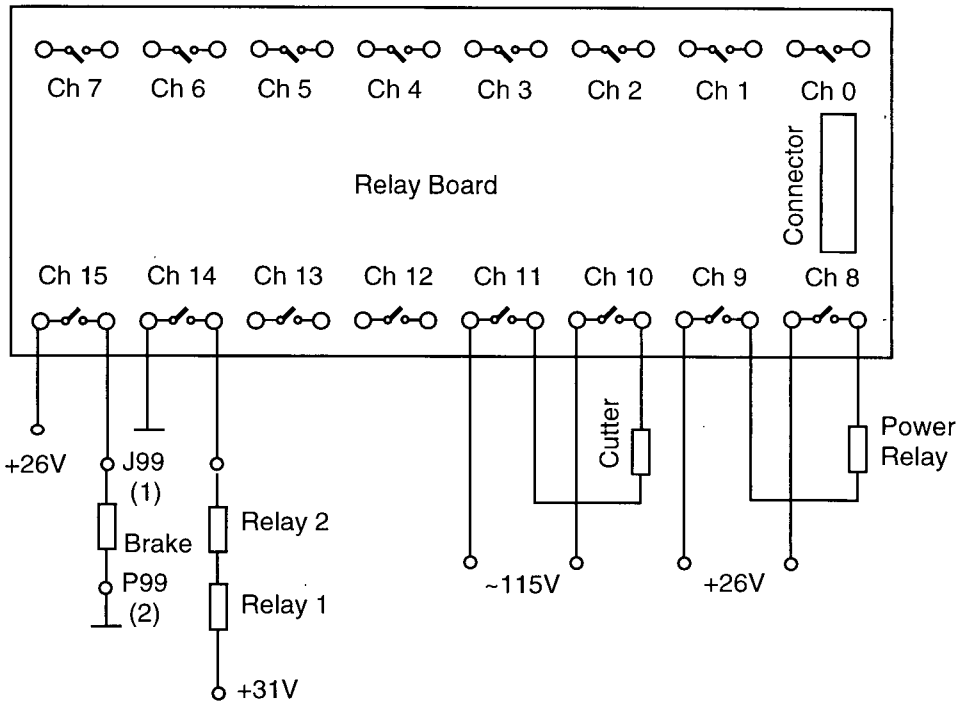


Figure 2.12: Control loops in the relay board.

- Ch 10 and Ch 11 are used for cutter (end-effector) control. Note that AC power and cutter power supplies are both controlled by two relay gates, to enhance the reliability.

Controller Interface Board: Figure 2.14 gives details of the interface board of the new controller. The pin assignment of connectors should coincide with that of the corresponding computer cards.

2.3.3 Implementation of signal input and output

Two, model M5312-4, quadrature encoder cards are used in the new control system. An M5312-4 card has four 24-bit multi-mode counters which can be cascaded together to form various 24-bit counter configurations such as two 48-bit counters. Each counter is capable of counting in numerous modes. Examples are:

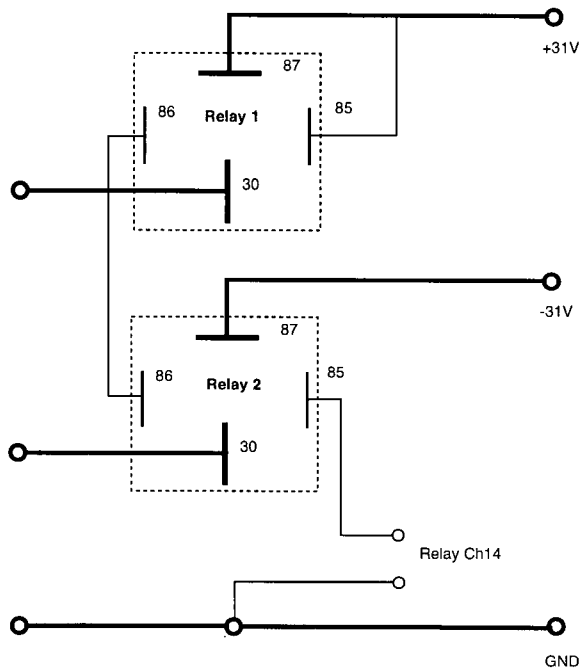
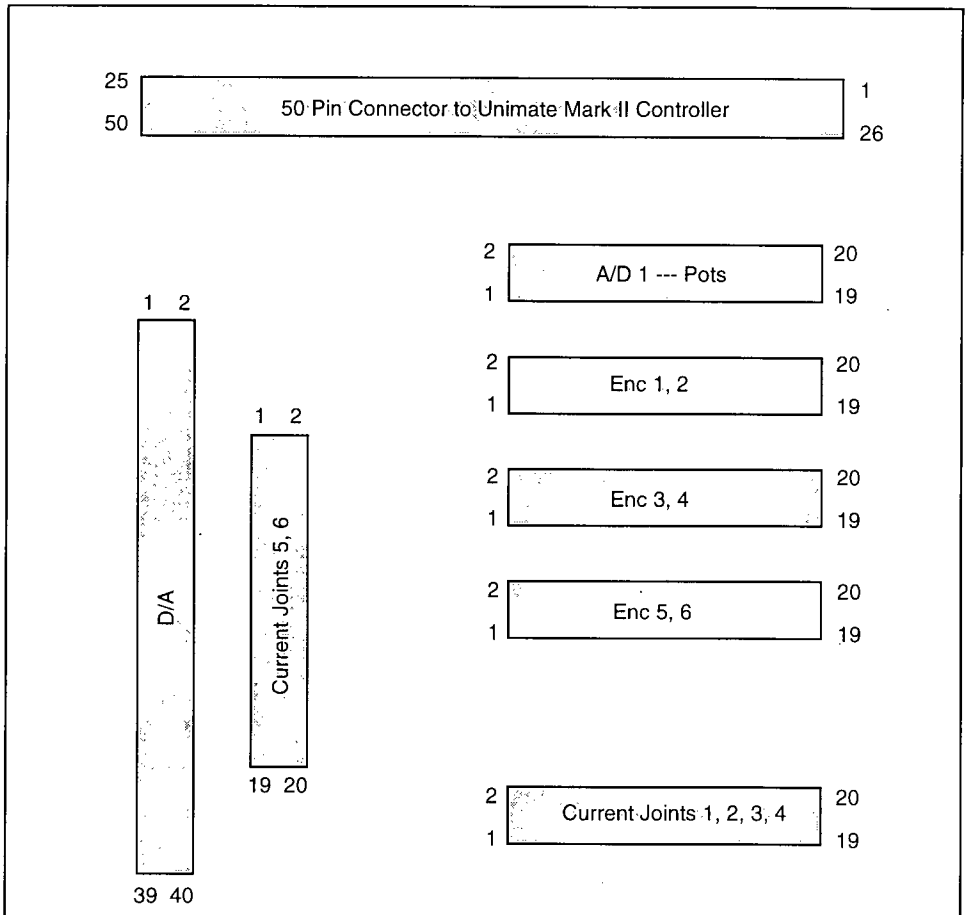


Figure 2.13: DC power protection circuit.

- A/B quadrature with a maximum input frequency of 0.333MHz.
- Up/down count with a maximum input frequency of 1.25 MHz.
- Count/direction with a maximum input frequency 1.25 MHz.

This speed is considerably greater than the typical operation speed of the robot. The maximum joint speed is about 4 radians/s, or 55k counts/s (see Table 2.4). All three inputs to the counter, phase A, phase B and index, can be single-ended TTL or, for greater noise immunity, differential type. Each input is digitally filtered using a sample clock rate. The connector (9 pin D-sub) pinout information for a channel is given in Table 2.7. Further details are found in the Model M5312 User Manual [85].

The base addresses of the two cards are set to 0x340 and 0x360. All four channels of the first card are used for joints 1 through 4. The channels 1 and 2 of the second card are used for



Pin assignment in 50-pin connector:

1, 2, 3: J1 pot, phA, phB;
26, 27,28: J2 pot, phA, phB;
4, 5, 6: J3 pot, phA, phB;
29, 30, 31: J4 pot, phA, phB;
7, 8, 9: J5 pot, phA, phB;
32, 33, 34: J6 pot, phA, phB;
11, 36: J1 current;
12, 37: J2 current;
13, 38: J3 current;
14, 39: J4 current;
15, 40: J5 current
16, 41: J6 current;
21: DA GND;
19, 18, 20, 44, 43, 45: DA to J1, J2, J3, J4, J5, J6;
47, 23; break;
50, 49, 25, 24: GND, +5V, -12V, +12V
48: +26 when mechanical switch on;
others: reserved.

Figure 2.14: I/O interface board of the new controller.

Table 2.7: Pin assignment of the M5312 connector.

1	GND output	6	Phase A input
2	+5V output	7	Phase B input
3	GND output	8	GND output
4	+5 output	9	Index input
5	GND output		

joints 5 and 6. No interrupt is enabled. Software polling is the basic method of communication, which allows the process to control the I/O procedure in a flexible manner.

The interface board uses a D/A card of Model DDA06, which has the base address 0x350. DDA06 is a 6 channel 12-bit digital-to-analog converter. Channels 1 through 6 are used for joints 1 through 6, respectively. The use of this card is quite straightforward. Specifically, the command:

$$\text{outp}(\textit{base\_address}+\textit{ch\_no},\ \textit{value})$$

will send *value* to channel *ch_no*. The output range is set to ±10V. The pinout of analog port is shown in Figure 2.15. Further information is found in the DDA06 User Manual [84].

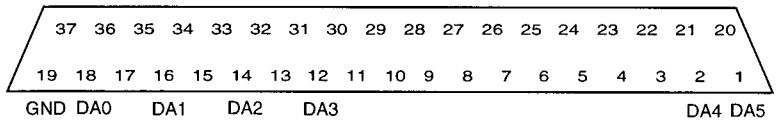


Figure 2.15: Pin assignment of the DDA06 connector.

The A/D card is a multiple purpose data acquisition system (DAS) card of model PCL814B [87]. It provides 16 differential 14-bit A/D channels. Maximum sampling rate is 100 kHz, which provides a bandwidth of upto 50kHz and is fast enough for most real-time robot control tasks. It also provides 16 channels of digital output. The output channels are used for relay control directly (address 0x310, 0x311).

The base address of this card is set to 0x310. All interrupts are disabled for the same reason as for the encoder cards. Software trigger is used in polling. A/D channels 1 through 6 are used for potentiometer signals, and the channels 9 through 14 for current sensing. Signal range varies from ± 1.25 to ± 5V depending on whether or not the physical signals are amplified. Pin assignment of the analog port is illustrated in Figure 2.16.

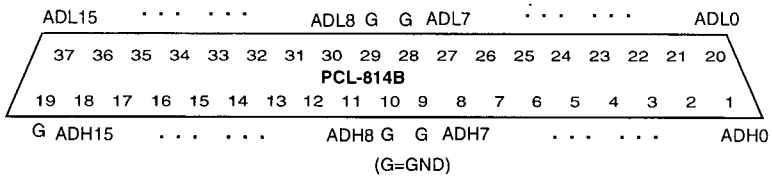


Figure 2.16: Pin assignment of the PCL814B analog signal connector.

The CPU card is a PCA-6157 Pentium 90MHz PCI/ISA card with 16M bytes of memory. The PCA-6157 User's Manual contains further information [86]. The network card is a QNX (see Chapter 8 for details on QNX) Arcnet card with boot ROM. Hence, the entire controller can be booted from a QNX network (Arcnet is chosen here for its real-time performance). In fact, the low-level controller is a diskless system. The whole system is mounted on an industrial ISA passive chassis, Model 9200.

2.3.4 Network controller

An important feature of the robot controller developed here is the use of QNX network. To quote the QNX OS manual of System Architecture [79]:

> In its simplest form, local area networking provides a mechanism for sharing files and peripheral devices among several interconnected computers. QNX goes far beyond this simple concept and integrate the entire network into a simple, homogeneous set of resources.

In this manner, QNX network makes the entire system completely transparent. Robot components can be controlled by different machines, but managed as if they are controlled by a single machine. No additional inter-machine communication is required in QNX applications.

Figure 2.17 shows an example of a QNX network controller. All the computers that are connected to the QNX network will function as a single computer, but with increased power for handling real-time events, since they are capable of parallel processing. No serial or parallel port communication is needed between a host and a low-level servo controller. This technique provides the much needed flexibility for a robot control system.

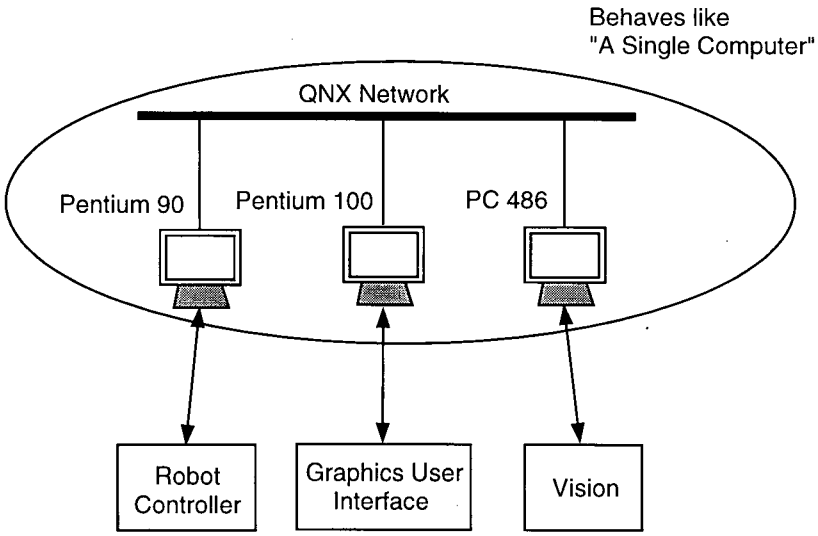


Figure 2.17: Schematic representation of a QNX network controller.

In the present research, the overall system is generated by combining the new low-level open architecture controller and the high-level QNX network, which is completely transparent, flexible, and expandable.

In order to make the overall control system including hardware, work in a desired manner, the control software has to be properly developed and incorporated as well. This aspect will be discussed in chapters 7 and 8. Impedance sensing and control which is an important feature of the robot control system, will be presented in the next chapter.

2.4 Summary

This chapter described the characteristics of the laboratory Puma 560 robot, with respect to joint friction, electro-mechanical dynamics of the axes and calibration constants. The forward and inverse kinematics solution of the robot were given. Robot control problems were investigated. To overcome the restrictions of the commercial controller of PUMA 560, particularly the non-accessibility of the low-level controller, and to handle complex and intelligent control procedures in a flexible manner, an open-architecture system was designed and developed. Both the low-level and the high-level systems were discussed.

Chapter 3

Impedance Estimation in a Single DOF System

Impedance estimation in a multiple degree-of-freedom (DOF) robot system can be quite complicated, as impedance functions in all DOF have to be determined in general. These impedance functions are coupled, and the associated estimation problem is further complicated by the fact that in the present research, the impedance for various directions of motion at the end-effector have to be determined using the driving-motor current and motion information of the robot joints. To simplify the problem, first a single-DOF system, such as a single joint along with an associated end-effector, is studied in this chapter. The methodology developed for single-DOF systems can be then extended to multiple DOF systems while exercising special care concerning dynamic coupling of joints, nonlinearities, and the degrees of freedom of the end-effector motion. This chapter will develop the necessary procedures for modelling and estimation of system states, external force, and impedance in a single degree-of-freedom system. Three methods of impedance estimation will be presented: (1) The impedance at the interface between end-effector and environment, is computed in the time domain as a ratio of the external force to the velocity at the interface. The force and the velocity are explicitly computed by using robot dynamics and position information. (2) The external force and velocity at the robot-environment interface are estimated by using a Kalman filter and based on a system model. (3) The parameters of object impedance are identified in the frequency domain using an environment model. All three methods will employ a robot model to obtain the dynamic force generated by the robot itself. In particular, the third method will use an environment model as well for parameter identification. The chapter will begin by presenting the modelling

considerations in Section 3.1. Then, the three impedance estimation procedures are presented in sections 3.2, 3.3, 3.4. A comparative study and a summary of the chapter will be given in Section 3.5 and Section 3.6, respectively.

3.1 System Modelling

3.1.1 Environment model

The objects that are mechanically processed in flexible manipulation may consist of inhomogeneous and flexible material, which is typically the case with natural objects (biological tissues) in food processing. Modelling of flexible and inhomogeneous objects of biological tissue is extremely difficult. As more research is done in the areas of biomechanics [41], better understanding of important aspects of the modelling problem is gained. The objects concerned are typically inconsistent and greatly varied. In essence, modelling of these objects in a reasonably simple form is virtually impossible. However, an approximate model may be adequate for mechanical analysis in this research, as the model parameters are assumed variable and are continuously estimated online through actual physical measurements.

An approximate linear model [41] for cutting is shown in Figure 3.1. The first part is a Kelvin-Voigt model [25] which simulates the material flexibility (visco-elastic) and associated dissipation during processing. The second part of the model simulates cutting friction. The mathematical model can be written as follows:

$$f = k'(x_1 - x_2) + b_1'(\dot{x}_1 - \dot{x}_2) + b_2'\dot{x}_2 + f_c \mathrm{sgn}(\dot{x}_2) \tag{3.1}$$

where f_c denotes Coulomb friction at the cutter; b_2' denotes the viscous damping coefficient of cutting; and k' and b_1' are the elastic stiffness and the damping coefficient, respectively, associated with material deformation prior to cutting. Also, x_i denote the displacement variables.

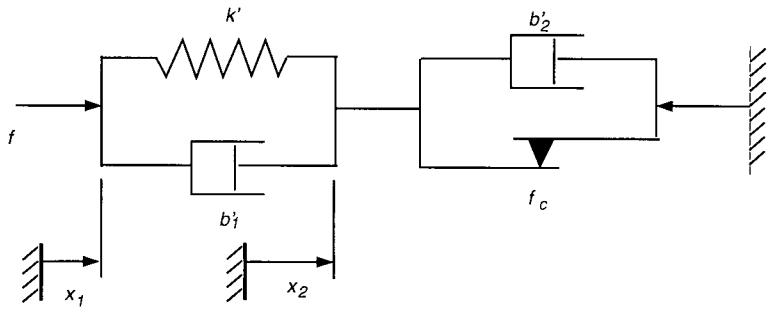


Figure 3.1: An approximate model for meat cutting.

This two degree-of-freedom model assumes lumped parameters corresponding to the coordinates x_1 and x_2. However this is not an accurate representation of a continuous object, which would be affected as a whole during cutting, and hence necessitating continuous spatial variables and partial differential equations. But, the present lumped-parameter model may be considered as an "equivalent" representation. The relationships between cutting force and cutting position, velocity, and acceleration are actually highly nonlinear as well, and may involve inertia forces. Therefore, a general model of the form,

$$f = f_i(\ddot{x}) + f_b(\dot{x}) + f_k(x) \tag{3.2}$$

would be more appropriate in the absence of continuous online estimation of model parameters. Here f, f_i, f_b, and f_k are the applied force, inertia force, damping dissipation force, and elastic force, respectively. As before, x denotes the displacement variable corresponding to the end-effector motion.

A single DOF cutting experiment was carried out on a salmon, in order to check the feasibility of determining mechanical impedance online. Only the third joint of the PUMA 560 was driven in the experiment. The cutter was mounted on the third robot link, and the cutting force was roughly measured by the actuator effort, and in turn by the motor current. Figure 3.2 shows the force/displacement relation as determined in the cutting trial.

Before the cutter hits the fish, only the joint friction is present. Upon touching the fish skin,

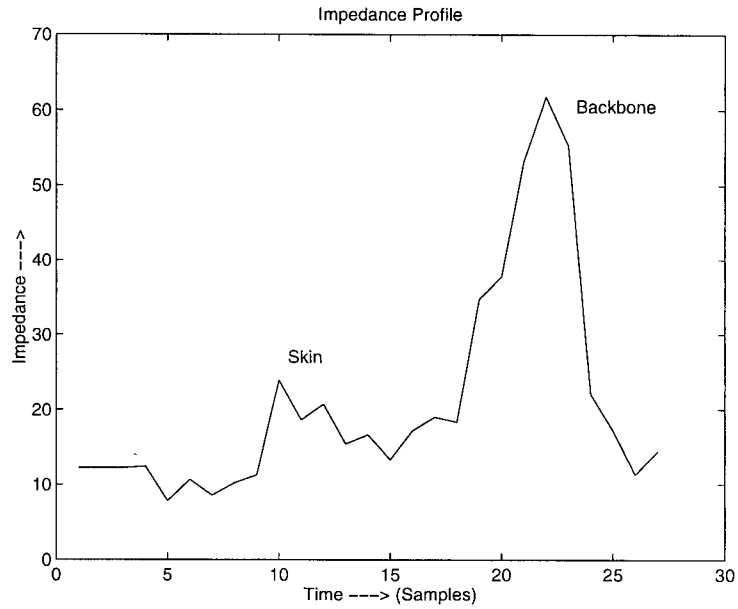


Figure 3.2: The force/displacement data from a single DOF cutting experiment of salmon (impedance units: N/cm/s, sampling rate: 1Hz).

the force increases as the cutter moves against the surface deformation of the fish. When the skin is cut through, the force drops a little. The subsequent force is needed to overcome the resistance in cutting the fish meat. The force jumps up rapidly when the cutter hits a bone (backbone of the salmon), and quickly drops after the bone is cut through. Note that, in estimating the cutting force at the object-cutter interface, we need to remove the components of inertial torque and the frictional torque at the robot joint. Figure 3.2 uses the total actuator torque, which includes the cutting force, the joint dynamic force, and the friction and disturbance forces. The dynamic force may be minimized by keeping the cutter speed somewhat constant. The particular result shown in Figure 3.2 is intended only to give a rough idea of the cutting process. A new accurate estimation of the cutting force will be discussed later in this chapter.

The cutting process itself can be summarized as follows:

- Skin: predominantly spring effect;

- Meat: soft, low stiffness, and predominantly dissipation;
- Bone: high dissipation and high stiffness.

As we can see, the force-motion relations vary considerably in a cutting process. A main objective of the present research is online estimation of process impedance, where it is important to measure or estimate the parameter variations that represent cutting properties of an object. By localizing (linearizing about an operating point) equation (3.2), we write,

$$\Delta f = \frac{\partial f_i}{\partial \ddot{x}}\Delta\ddot{x} + \frac{\partial f_b}{\partial \dot{x}}\Delta\dot{x} + \frac{\partial f_k}{\partial x}\Delta x \tag{3.3}$$

Define the parameters

$$m' = \frac{\partial f_i}{\partial \ddot{x}},\ b' = \frac{\partial f_b}{\partial \dot{x}},\ k' = \frac{\partial f_k}{\partial x}.$$

Then, we have

$$\Delta f = m'\Delta\ddot{x} + b'\Delta\dot{x} + k'\Delta x.$$

In the frequency domain, the corresponding relation is

$$\Delta F(s) = m's^2\Delta X(s) + b's\Delta X(s) + k'\Delta X(s)$$

which can be expressed in the transfer function form as

$$\frac{\Delta F(s)}{\Delta X(s)} = m's^2 + b's + k'$$

In this thesis, for notational convenience, the same symbol may be routinely used to denote both a time signal and its Laplace or Fourier transform.

Assuming a passive environment/object, mechanical impedance at the processing interface consists of three primary components: inertial impedance, dissipative impedance, and elastic impedance. Generally, then, mechanical impedance has the model (single DOF),

$$Z_e = \frac{\Delta F_e(s)}{\Delta V(s)} = \frac{\Delta F_e(s)}{\Delta sX(s)} = m's + b' + \frac{k'}{s}.$$

where F_e denotes a general force at the end-effector, and X denotes a general displacement along the degree-of-freedom (motion) of the end-effector in which direction F_e acts.

3.1.2 Process model

Again, we consider a single joint of the robot. As Puma 560 may be treated as a rigid body system with negligible flexibility of links and joints, a model for a single joint may be represented as,

$$f_m - f_e - f_d = m\ddot{x} + b\dot{x} \tag{3.4}$$

where f_m is the motor magnetic force, m and b are equivalent lumped inertia and viscous damping coefficient, respectively, of the joint. The variables f_e and f_d are the external force at the process interface and the joint disturbance (mainly Coulomb friction), respectively.

Generally, a process model consists of a combination of a robot arm model and an environment/object model, with a common variable, the velocity at the interface between the end-effector and the processed object. The single DOF model, as shown in Figure 3.3, may be expressed in the general form:

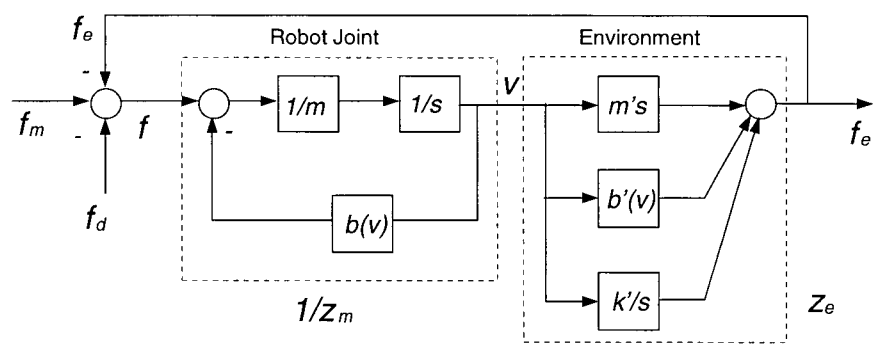


Figure 3.3: An impedance model for single DOF robot manipulation.

$$f_m - f_e - f_d = f_i(\dot{v}) + f_b(v) = z_m v \tag{3.5}$$

and

$$f_e = f_i'(\dot{v}) + f_b'(v) + f_k'(\int v) = z_e v \tag{3.6}$$

Here, z_m denotes the impedance of the robot joint including the motor rotor, transmission system and end-effector; z_e denotes the impedance exerted on the end-effector at the interface

between the end-effector and the processed object; and v denotes the velocity at the process interface. The main task is to estimate the impedance z_e at the process interface as given by

$$z_e = \frac{f_e}{v} \tag{3.7}$$

Here, f_e can be estimated from equation (3.5) where allowance should be made for joint dynamics, and v can be estimated from displacement of the joint. Note in particular that f_d which primarily includes Coulomb friction at the joint, should be taken into consideration in estimating f_e.

We have observed that, usually, an impedance model of the process interface of object/environment consists of a combination of the three basic components: inertial, dissipative and elastic impedances. One or two of these components may be neglected depending on the process conditions. For example, in fish cutting, where the object (fish) is usually fixed during the cutting process, and the mass of the moving elements of the object body is quite small, the inertial impedance may be neglected (however, m' is still included in the model, for generality), resulting in a model consisting primarily of dissipative and elastic impedance elements.

To represent impedance as a linear combination of inertial, dissipative and elastic elements, a linearized local model is derived as follows:

In the time domain,

$$f_e = f_{i'}(\dot{v}) + f_{b'}(v) + f_{k'}(\int v) \tag{3.8}$$

$$f = f_i(\dot{v}) + f_b(v) \tag{3.9}$$

$$f_m = h(f, f_e, f_d) \tag{3.10}$$

where

$$h(f, f_e, f_d) = f + f_e + f_d \tag{3.11}$$

or, in the Laplace domain,

$$h(F(s), F_e(s), F_d(s)) = F(s) + F_e(s) + F_d(s) \tag{3.12}$$

By localizing equations (3.9) through (3.10) and transforming into the s domain, we get

$$\Delta F_e(s) = m's\Delta V(s) + b'\Delta V(s) + \frac{k'}{s}\Delta V(s) \tag{3.13}$$

$$\Delta F(s) = ms\Delta V(s) + b\Delta V(s) \tag{3.14}$$

$$\Delta F_m(s) = \Delta h(F(s), F_e(s), F_d(s)) \tag{3.15}$$

The derivation of the first two equations has been discussed in the previous section. Equation (3.15) in general, can be further expressed as follows,

$$\Delta F_m(s) = \frac{\partial h}{\partial F(s)}\Delta F(s) + \frac{\partial h}{\partial F_e(s)}\Delta F_e(s) + \frac{\partial h}{\partial F_d(s)}\Delta F_d(s). \tag{3.16}$$

By substituting equation (3.12) into equation (3.16), we have

$$\Delta F_m(s) = \Delta h(F(s), F_e(s), F_d(s)) = \Delta F(s) + \Delta F_e(s) + \Delta F_d(s) \tag{3.17}$$

The impedance of the manipulator environment which is the interface between the end-effector and the processed object, can be written as,

$$Z_e(s) = \frac{\Delta F_e(s)}{\Delta V(s)} = m's + b' + \frac{k'}{s} \tag{3.18}$$

and the manipulator dynamics can be written as

$$Z_m(s) = \frac{\Delta F(s)}{\Delta V(s)} = ms + b \tag{3.19}$$

The model shown in Figure 3.3 may not be an exact representation of the process, and will contain many unknowns including joint friction, that are primarily represented by F_d. A digital filter of the Kalman type [65, 66] may be incorporated with this model, to obtain an online estimate of the mechanical impedance at the process interface. Specifically, the measured signals (motor current and joint displacement) will form the inputs to the filter/estimator and the mechanical impedance of the process interface, as modeled, will be its output (estimate).

3.1.3 Model-based impedance estimation

Three methods are considered here for impedance estimation. The first method adopts direct impedance computation. A general caution should be made regarding impedance computation in the time domain. Even though the ratio f_e/v of the time signals represents mechanical impedance as it varies with time, it is not the same as the mechanical impedance in the frequency domain, as given by the classical definition. As we will see later in the thesis, the impedance signals in both the time domain and frequency domain represent the process characteristics, hence are useful in flexible manipulation.

It is clear that in the time domain the impedance z_e can be computed from $\frac{f_e}{v}$. Since we do not measure the velocity directly, v is estimated from the position signal using a Kalman filter, and f_e is computed as,

$$f_e = f_m - f - f_d$$

where f_m is the motor torque/force, which can be measured from the current of the motor armature circuit; f is the dynamic force/torque component of the manipulator arm (a single link is considered here) primarily representing its inertia and damping effects, which can be computed from the robot dynamic equations; and f_d is the disturbance term which is dominated by Coulomb friction at the robot joint, and can be estimated using the friction model given in Chapter 2 for reasonably satisfactory results.

The error in friction estimation could be large. However, what is really significant is the change of the process impedance which represents the transition between different materials within a processed object. The effect of friction is assumed constant, and its variation is negligible in steady state processing. Hence, it will not significantly affect the elements of "change" in process impedance. The proposed impedance estimation is quite useful and accurate in cases where the detection of impedance changes is required. The direct computation method will be discussed in detail in Section 3.2.

The second method uses a Kalman filter to estimate system velocity, acceleration, and the contact force of the end-effector, and thereby to compute the process impedance using a single estimator. This algorithm will be discussed in Section 3.3. Note that, although both the methods use Kalman filters, the first method uses a Kalman filter more like a general filter to estimate velocity, whereas the second method uses a Kalman filter as an estimator based on the physical model of the system to identify all necessary process variables.

The third method utilizes a frequency domain localized model, and applies a parameter identification algorithm to estimate the impedance parameters. This method will be presented in Section 3.4.

Section 3.5 will compare the three methods, discuss their merits and drawbacks, illustrate situations where various impedance estimation methods may be applicable, and indicate the most appropriate method. Section 3.6 will summarize the chapter.

3.2 Direct Computation of Impedance

The first method of impedance computation, as described herein, is based on the definition of mechanical impedance as the ratio force/velocity in the time domain. However, the measured signals of motor armature current and angular position are usually too noisy for use in direct computation of impedance. A general purpose Kalman filter is constructed to filter out noise in position and current signals, and to obtain velocity and acceleration at the same time.

Let $x(t)$ be a signal corresponding to either position or current, within a small time interval Δt in the vicinity of time t_0. We write the Taylor series expansion,

$$x = x_0 + \dot{x}\Delta t + \frac{\ddot{x}}{2}\Delta t^2 + e(\Delta t) \tag{3.20}$$

where x is the signal at time $t = t_0 + \Delta t$, x_0 is the value of the signal at time t_0, and $e(\cdot)$ represents the $O(3)$ terms (residual) of the series.

Let

$$x_1 = x, \quad x_2 = \dot{x}, \quad x_3 = \ddot{x}. \tag{3.21}$$

Equation (3.20) can be written as,

$$x_1 = x_{1_0} + \Delta t x_{2_0} + \frac{\Delta t^2}{2} x_{3_0} + w_1. \tag{3.22}$$

For a general time point k, we have,

$$x_{1_k} = x_{1_{k-1}} + \Delta t x_{2_{k-1}} + \frac{\Delta t^2}{2} x_{3_{k-1}} + w_1. \tag{3.23}$$

Using the first order and the zero order expansions for x_2 and x_3, we have,

$$x_{2_k} = x_{2_{k-1}} + \Delta t x_{3_{k-1}} + w_2, \tag{3.24}$$

$$x_{3_k} = x_{3_{k-1}} + w_3. \tag{3.25}$$

where w_1, w_2, w_3 are the residuals of the Taylor series expansion. For a physical system, w_1, w_2, w_3 may be considered as the combination of expansion residuals and process noise.

The equations (3.23), (3.24), and (3.25) can be written in the general matrix form

$$\mathbf{x}_k = \Phi \mathbf{x}_{k-1} + \mathbf{w}, \tag{3.26}$$

together with output equation

$$\mathbf{y}_k = C\mathbf{x}_k + \nu, \tag{3.27}$$

where

$$\Phi = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix},$$

$$C = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix},$$

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}_k^T,$$

$$\mathbf{w} = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix}_k^T,$$

and y_k is the measured value, and ν is the measurement noise, which is assumed zero-mean, white Gaussian and independent of process noise $\mathbf{w}$. The covariance of ν is a scalar value given by,

$$R = E\{\nu\nu^T\} = r. \tag{3.28}$$

If we consider only the residual of the Taylor series expansion, the process noise, say w_1, can be represented by,

$$w_1 = \frac{\Delta t^3}{3!}\frac{d^3x}{dt^3} + \frac{\Delta t^4}{4!}\frac{d^4x}{dt^4} + \cdots, \tag{3.29}$$

Due to limited input energy, $\frac{d^n x}{dt^n}$ will be bounded, and has a result

$$\lim_{n\to\infty} \frac{\Delta t^n}{n!}\frac{d^n x}{dt^n} = 0.$$

Therefore, to simplify the problem, we approximate w_1 by the first term of the residual series. The resulting covariance matrix will provide some useful information on the noise properties, which may help us in selecting the parameters when constructing the Kalman filter. The approximate process noise components are

$$w_1 = \frac{\Delta t^3}{3!}\frac{d^3x}{dt^3}, \tag{3.30}$$

$$w_2 = \frac{\Delta t^2}{2!}\frac{d^3x}{dt^3}, \tag{3.31}$$

$$w_3 = \Delta t\frac{d^3x}{dt^3}. \tag{3.32}$$

The covariance matrix of $\mathbf{w}$ ($[w_1, w_2, w_3]^T$) is,

$$
\begin{aligned}
Q &= E\left\{\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}[w_1\ w_2\ w_3]\right\} \\
&= E\left\{\begin{bmatrix} w_1^2 & w_1w_2 & w_1w_3 \\ w_1w_2 & w_2^2 & w_2w_3 \\ w_1w_3 & w_2w_3 & w_3^2 \end{bmatrix}\right\} \\
&= (\frac{d^3x}{dt^3})^2 \begin{bmatrix} \frac{\Delta t^6}{3!3!} & \frac{\Delta t^5}{3!2!} & \frac{\Delta t^4}{3!} \\ \frac{\Delta t^5}{3!2!} & \frac{\Delta t^4}{2!2!} & \frac{\Delta t^3}{2!} \\ \frac{\Delta t^4}{3!} & \frac{\Delta t^3}{2!} & \Delta t^2 \end{bmatrix}
\end{aligned}
\tag{3.33}
$$

In practice, we find that the absolute values of the elements of Q do not significantly affect the filtering results, but the relative values are very important. This means $(\frac{d^3x}{dt^3})^2$ in Q can be ignored in practical implementations.

We have,

$$\mathbf{w} \sim N(0, Q),$$

and

$$\nu \sim N(0, R).$$

Since the measurement devices are independent of the process, we can assume that $\mathbf{w}$ and ν are independent. Suppose that the system initial condition is $E(\mathbf{x}_0) = \hat{\mathbf{x}}_0$ and the initial error covariance is P_0. A Kalman filter can be designed to estimate the state variables as follows [44]:

Time update:

$$\hat{\mathbf{x}}_{k+1}(-) = \Phi_k \hat{\mathbf{x}}_k(+), \tag{3.34}$$

Error covariance extrapolation:

$$P_{k+1}(-) = \Phi_k P_k \Phi_k^T + Q_k, \tag{3.35}$$

Kalman gain matrix:

$$K_{k+1} = P_{k+1}(-)C_{k+1}^T(C_{k+1}P_{k+1}(-)C_k^T + R_{k+1})^{-1}. \tag{3.36}$$

Measurement update:

$$\hat{\mathbf{x}}_{k+1}(+) = \hat{\mathbf{x}}_{k+1}(-) + K_{k+1}(\mathbf{y}_{k+1} - C_{k+1}\hat{\mathbf{x}}_{k+1}(-)) \tag{3.37}$$

Error covariance update:

$$P_{k+1}(+) = (I - K_{k+1}C_{k+1})P_{k+1}(-). \tag{3.38}$$

In the design and implementation of the Kalman filter, the covariance updates $P(-)$, and $P(+)$ should be checked for symmetry and positive definiteness. Failure to attain either condition may result in inaccurate estimates. To overcome this problem, another equivalent expression for $P_{k+1}(+)$, called the Joseph form, is employed as shown below

$$P_{k+1}(+) = (I - K_{k+1}C_{k+1})P_{k+1}(-)(I - K_{k+1}C_{k+1})^T + K_{k+1}R_{k+1}K_{k+1}^T \tag{3.39}$$

Note that the right-hand side of this equation is the summation of two symmetric matrices. The first part is positive definite and the second one is positive semidefinite, thereby making $P_{k+1}(+)$ a positive definite matrix.

Filtering of joint current. Denote the measurement of the motor current signal by y. Then the output x_1 of the Kalman filter as designed above, is a filtered signal of the motor current. As an example of the results from a real-time operation, the original signal and the filtered signal of motor current are shown in Figure 3.4. Clearly, the filtered signal is much smoother than the raw signal. To reduce the computation load, we may use a second order mathematical

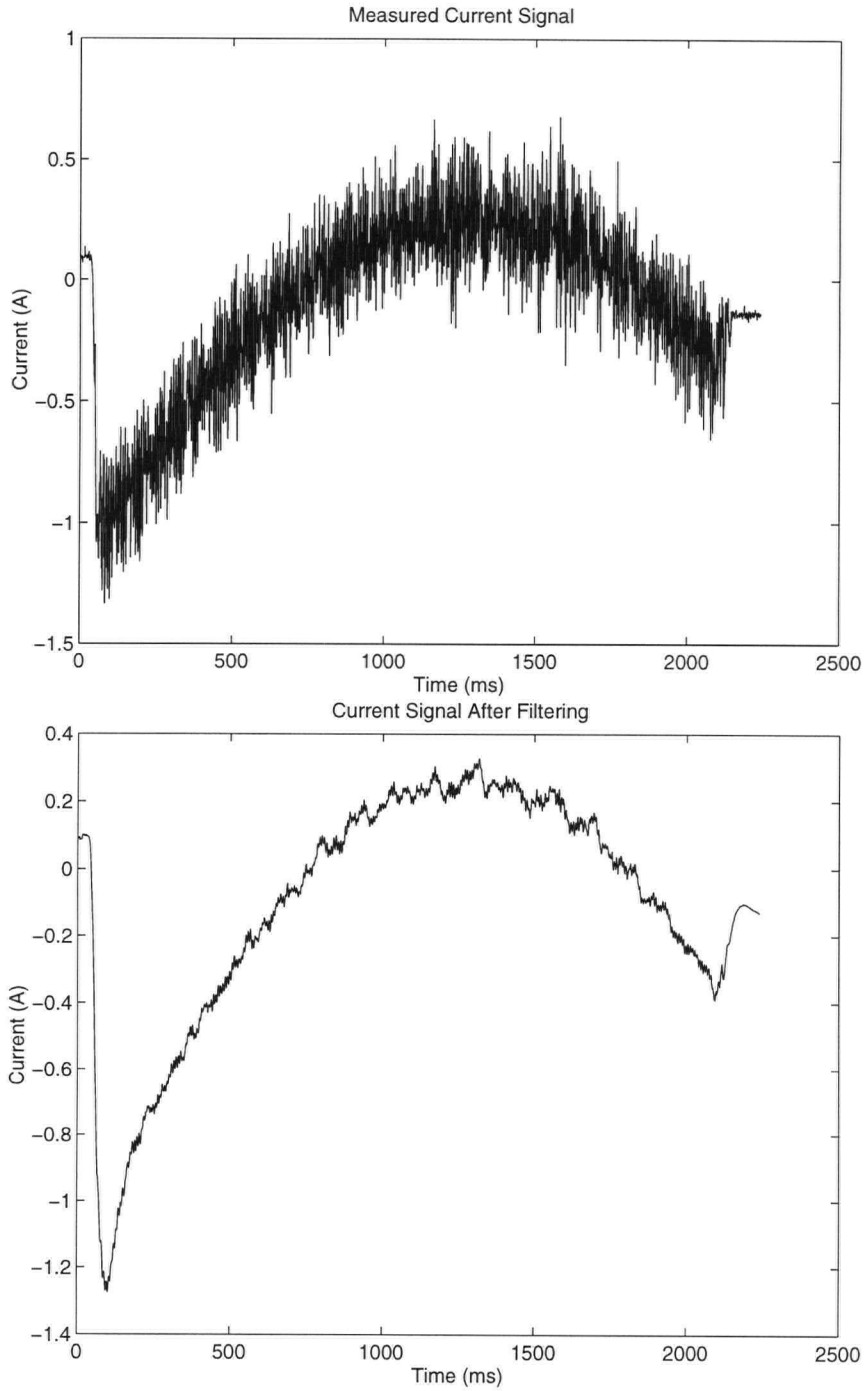


Figure 3.4: Measured and filtered current signals from the 3rd joint of Puma 560.

model instead of the third order model in equations (3.26) and (3.27) for the Kalman filter, as the derivatives of the current signal are not used.

The Q matrix in the Kalman filter follows equation (3.34), and is given by (operation data have been resampled at $\Delta t = 0.01$),

$$Q = \begin{bmatrix} 0.0001 & 0.02 \\ 0.02 & 4.0 \end{bmatrix}$$

and, in view of the high measurement noise, R is chosen as

$$R = 100.0$$

Note that the absolute values of Q and R are not important, but the relative values between Q and R (ratios of the elements) affect the performance of the filter.

Estimation of velocity and acceleration. The position signal from the encoder counter is quite accurate. Velocity and acceleration are not directly measured, but are necessary to compute the dynamic forces of the robot arm. As indicated in Chapter 1 of the thesis, direct measurement of velocity and acceleration would increase the instrumentation cost and decrease the system reliability. Differentiation of a position signal could render a reasonably good velocity signal. However, the acceleration signal obtained through double differentiation of position signal would be very noisy, and hence unacceptable.

The designed Kalman filter can be used to estimate the velocity and acceleration from an observed position signal. The variance of measurement noise is set to a low value here, since the measurement of position is found to be quite accurate and clean.

The noise covariance parameters used in the Kalman filter are,

$$Q = \begin{bmatrix} 0.1 & 0.0 & 0.0 \\ 0.0 & 100.0 & 0.0 \\ 0.0 & 0.0 & 10000.0 \end{bmatrix}$$

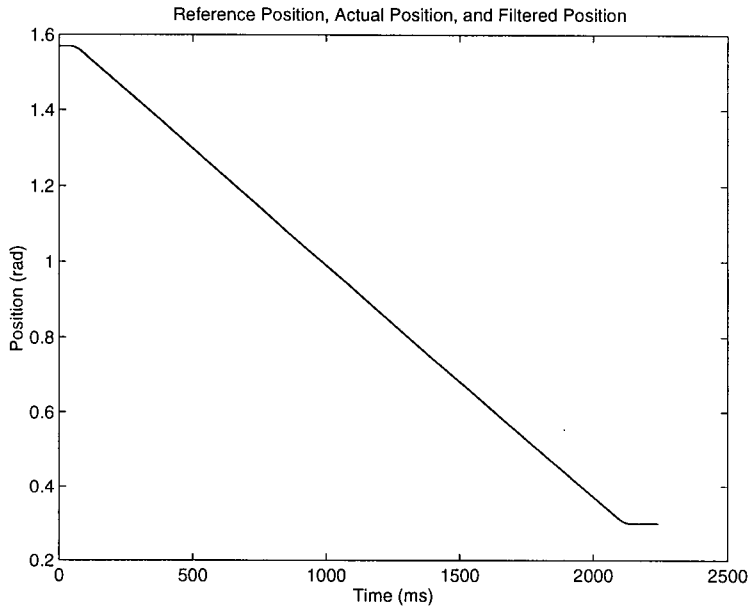


Figure 3.5: Reference, measured and filtered position signals from the 3rd joint of Puma 560 (All three signals virtually coincide).

and

$$R = [1.0].$$

The selection of the Q matrix is based on the fact that the process noise of position is very low, the noise of acceleration is very high, and the noise of velocity is somewhere in between. This can be seen from the results obtained by direct differentiation of a measured position signal (see figures 3.6 and 3.7 for velocity by direct differentiation, and acceleration by double differentiation). The filtered position signal is almost identical to the original measurement, and also to the specified reference signal, as shown in Figure 3.5. Velocities obtained through direct digital differentiation and Kalman filter estimation are shown in Figure 3.6. It is observed that the estimated signal is relatively noise free and would be much more useful in impedance computation. The more significant result, however, is the estimation of acceleration, which is shown in Figure 3.7. The acceleration computed by double differentiation is so noisy that it is

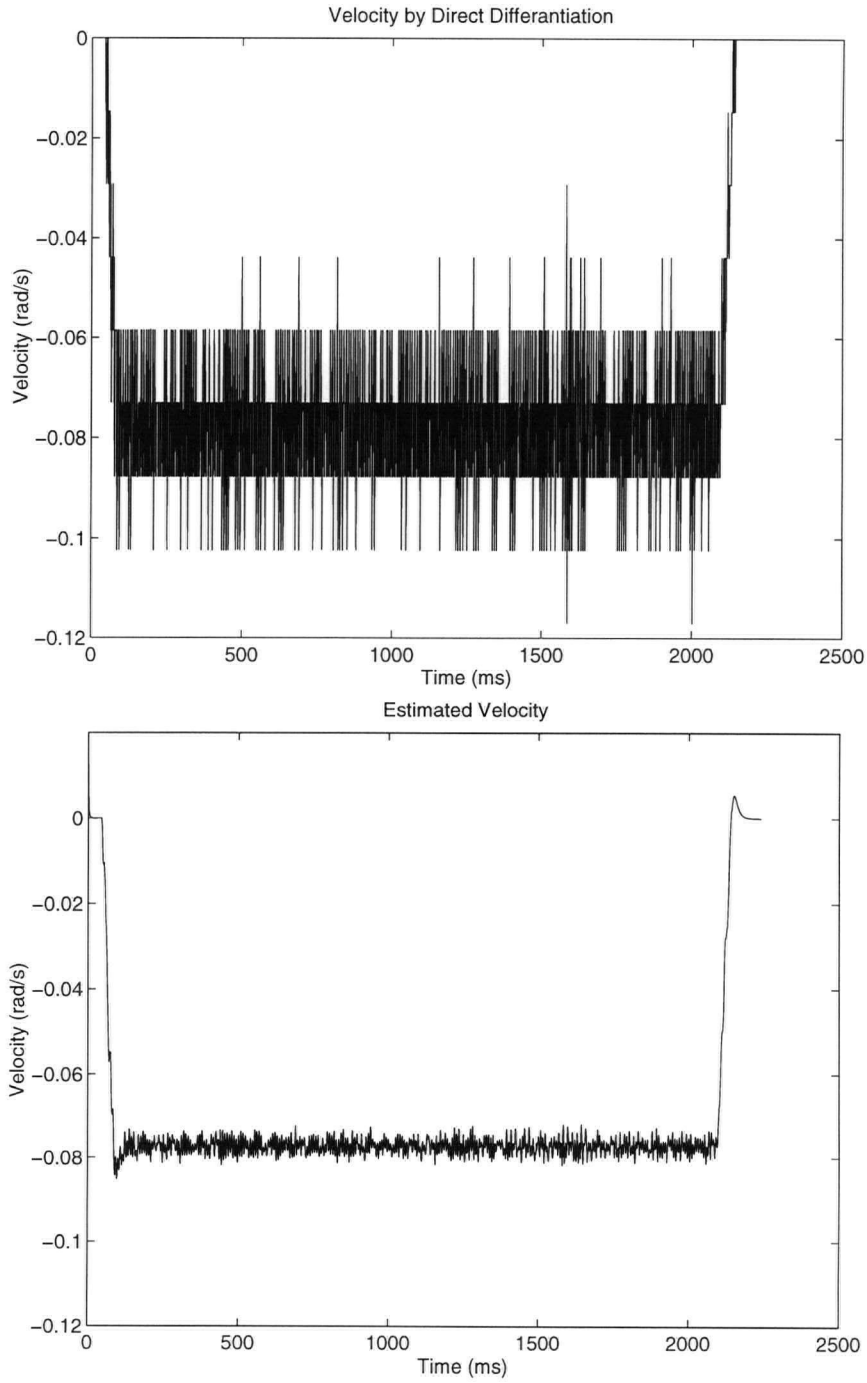


Figure 3.6: Velocity signals computed from measured position signal of the 3rd joint of Puma 560 by direct differentiation and Kalman filter estimation.

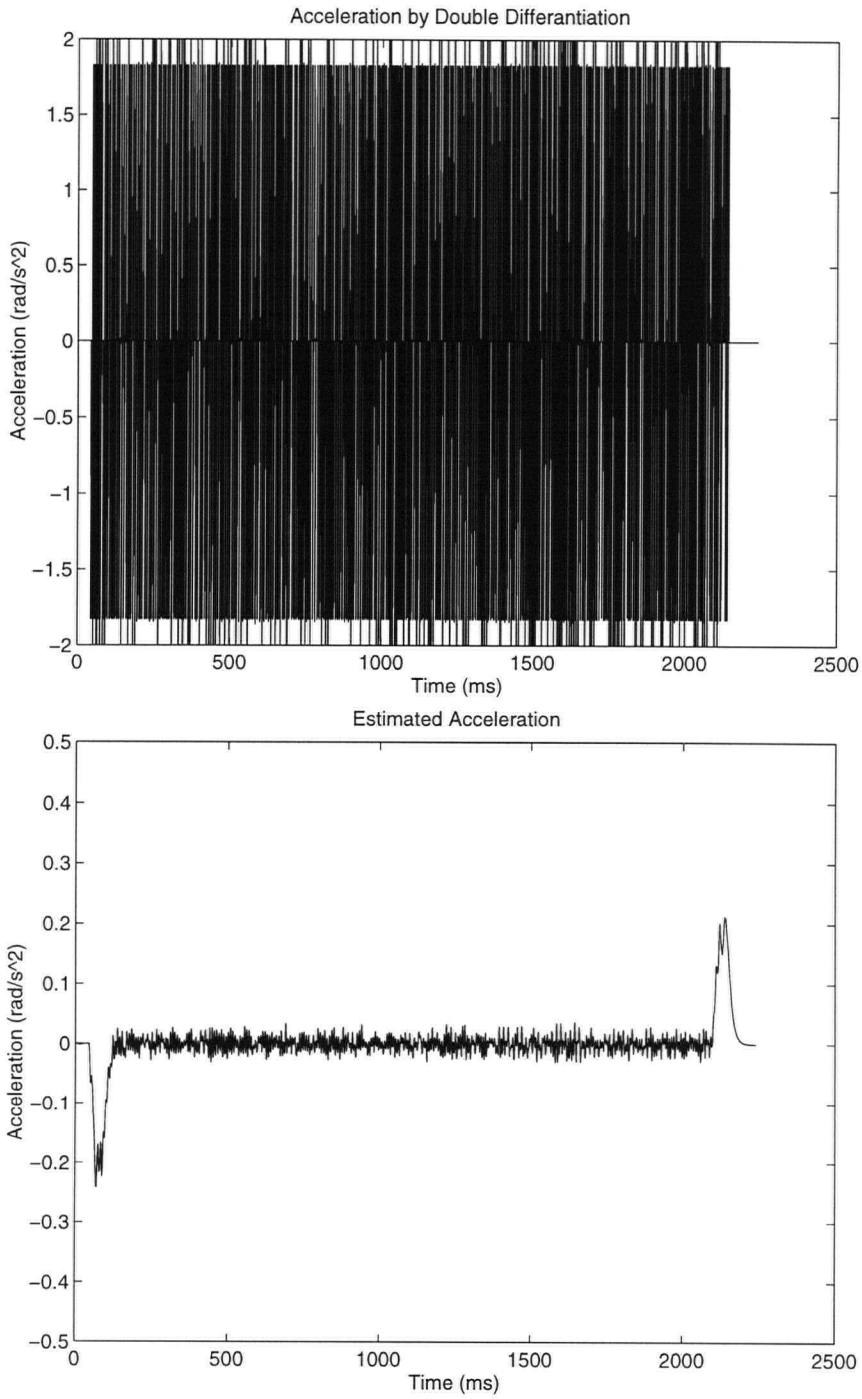


Figure 3.7: Acceleration signals computed from measured position signal of the 3rd joint of Puma 560 by double differatiation and Kalman filter estimation.

useless in our implementations. But, the acceleration result that is estimated through Kalman filter is found to be quite accurate.

The error covariance matrix P after 2000 points of estimation was still found to be positive definite and symmetric; thus,

$$P = 1.0e + 05 \begin{bmatrix} 0.0000 & 0.0001 & 0.0007 \\ 0.0001 & 0.0092 & 0.0659 \\ 0.0007 & 0.0659 & 1.8426 \end{bmatrix}.$$

Here, $P(1,1)$ is zero; the position signal is almost free of noise. However the acceleration signal is somewhat noisy as seen from the element $P(3,3)$ which is rather high.

A fourth order Kalman filter that applies to a fourth order model, instead of the third order model, in equations (3.26) and (3.27), has been constructed and tested. Similar results were obtained compared with those of the third order Kalman filter. Therefore, a third order Kalman filter is considered quite sufficient for the estimation of position, velocity, and acceleration, and is used henceforth, to save computational effort.

Computation of processing force. The external force/torque f_e acting on the end-effector, at the process interface, is given by,

$$f_e = f_m - f - f_d \tag{3.40}$$

where, f_m is the magnetic force/torque generated by the actuator current i_m; thus,

$$f_m = k_\tau i_m; \tag{3.41}$$

Here, k_τ is the motor torque constant (see Table 2.5). Also, f is the dynamic force/torque of the robotic joint:

$$f = m\ddot{q} + b\dot{q} + f_g(q); \tag{3.42}$$

where m denotes the joint mass/inertia, q denotes the generalized displacement of the joint, b denotes the viscous damping coefficient, and f_g denotes the gravity load. Note that only

the inertial force/torque, dissipation force/torque and gravitational force/torque are present in a single DOF system. Coriolis and centrifugal forces will present as well in a multiple DOF system, which will be discussed in the next chapter.

f_d is a disturbance term that includes nonlinear friction of the robot joint. There are several recent studies of measurement and estimation of friction [51, 80]. As the nature of friction can be quite complex and there can be inconsistencies in both manipulator joints and object processing interface, friction sensing/estimation remains an area of major difficulty in accurate sensing and control of a robot manipulator. In some processes; for example, meat cutting, friction can be approximately measured or estimated using an equivalent model, and the nature of friction does not significantly affect the "changes" of impedance at the process interface which one intends to detect. Hence the error in friction estimation by and large should not affect the detection of impedance changes in our application.

The disturbance (primarily Coulomb friction) of the robot joint is represented by,

$$f_d = \mathrm{sgn}(\dot{q}) f_c \tag{3.43}$$

By substituting f_m, f, and f_d into equation (3.40), we obtain the processing force

$$f_e = k_\tau i_m - m\ddot{q} - b\dot{q} - f_g(q) - \mathrm{sgn}(\dot{q}) f_c. \tag{3.44}$$

Here, k_τ, m, f_c, and b are taken to be constants. The system variables are i_m, q, $\dot{q}$, $\ddot{q}$, where i_m denotes the motor current, and q, $\dot{q}$, and $\ddot{q}$ denote direct measurements or estimates of joint displacement, velocity, and acceleration, respectively.

In the time domain, mechanical impedance is computed as

$$z_e(k) = \begin{cases} f_e(k)/\dot{q}(k), & \text{if } |\dot{q}(k)| \geq \delta; \\ z_e(k-1), & \text{if } |\dot{q}(k)| \leq \delta; \end{cases} \tag{3.45}$$

where z_e denotes the impedance at the process interface, which delineates the resistance exerted on the end-effector by the processed object; k denotes a discrete time point; and δ denotes a

small positive value for speed, below which the computation of impedance is known to be ill-posed. Also, when the velocity drops to zero, the impedance is not defined. For computational convenience, the previous value $z_e(k-1)$ is retained for the case of low values of speed. An impedance profile that is computed in this manner, where only joint 3 of the Puma 560 is driven and the end-effector is made to collide with two obstacles during motion, is shown in Figure 3.8. Note that the impacts are clearly detected by impedance estimation.

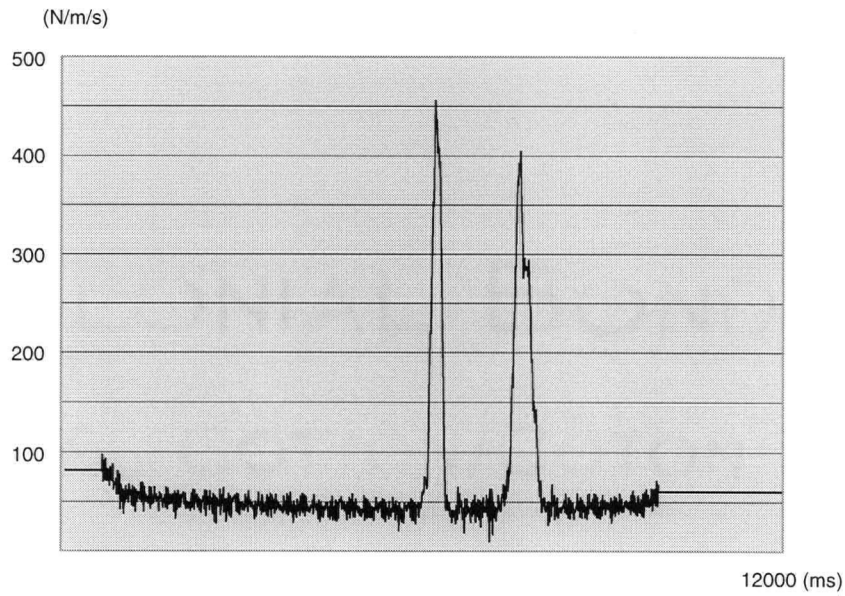


Figure 3.8: Computed impedance profile when joint 3 of the Puma 560 is driven and the end-effector collides with two obstacles during motion.

Clearly, the mechanical impedance for free motion should be zero. It has a small value here due to underestimation of friction. The slight change of impedance at the beginning and the end of the motion results from the computational error when the motion speed is very low.

3.3 Impedance Estimation Using Kalman Filter

The joint model of robot, which is shown in Figure 3.9, has two parts: the electrical subsystem and the mechanical subsystem. The two subsystems are connected through the motor torque constant k_τ and the back EMF constant k_E. The constant k_τ converts electric current into mechanical torque, and k_E provides velocity feedback into motor voltage. We consider the following two cases here:

- actuator current is known;
- actuator current is unknown or so noisy that filtering out of noise would be required.

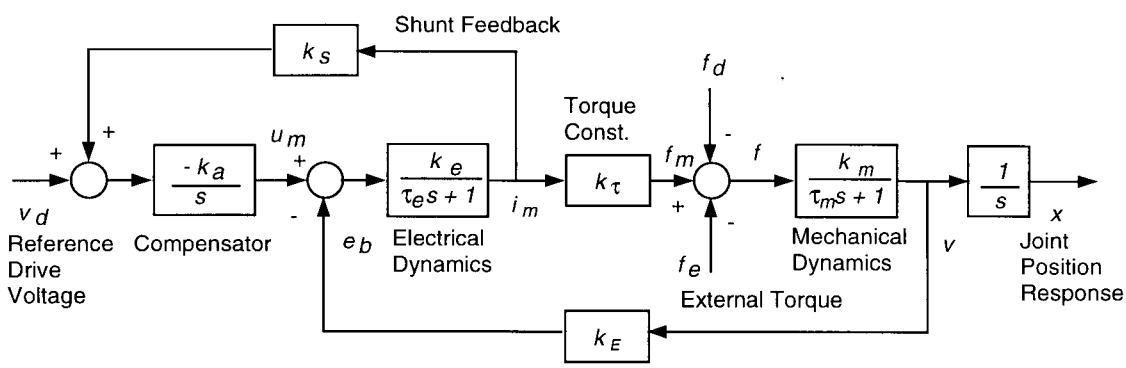


Figure 3.9: A model for a joint of Puma 560 for use in estimating equivalent external torque acting through end-effector.

For system estimation, only the model of the mechanical subsystem is needed in the first case. However, a complete system model should be considered in the second case.

Case I: i_m is known. The mechanical system can be written as,

$$\dot{q} = v \tag{3.46}$$

$$\dot{v} = -\frac{1}{\tau_m}v + \frac{k_m}{\tau_m}(k_\tau i_m - f_e - f_d) \tag{3.47}$$

where q and v are the displacement and the velocity of a joint, respectively, i_m is the motor current, f_e is the equivalent external force/torque acting at the joint as a result of the process

forces on the end-effector , and f_d denotes the disturbance force/torque that is mainly due to joint Coulomb friction. In the subsequent system models, f_d is ignored because it is assumed to be constant (Coulomb friction) and hence its influence on the change of f_e is negligible. The equation (3.47) can be written in the matrix state-space form,

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} \tag{3.48}$$

and

$$\mathbf{y} = C\mathbf{x} \tag{3.49}$$

where

$$\mathbf{x} = \begin{bmatrix} q & v \end{bmatrix}^T; \quad \mathbf{u} = \begin{bmatrix} i_m & f_e \end{bmatrix}^T; \quad \mathbf{y} = \begin{bmatrix} q \end{bmatrix},$$

and

$$A = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{1}{\tau_m} \end{bmatrix}; \; B = \begin{bmatrix} 0 & 0 \\ \frac{k_m k_\tau}{\tau_m} & -\frac{k_m}{\tau_m} \end{bmatrix}; \; C = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

The system is observable since $\begin{bmatrix} C^T & A^T C^T \end{bmatrix}^T$ has full rank as expected; thus,

$$\text{rank} \begin{bmatrix} C \\ CA \end{bmatrix} = \text{rank} \left\{ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right\} = 2$$

By converting this model into the discrete time form [7], we obtain,

$$\mathbf{x}_{k+1} = \Phi \mathbf{x}_k + \Gamma \mathbf{u}_k \tag{3.50}$$

where $\mathbf{u}_k = [i_{m_k}, \; f_{e_k}]^T$ and $\Gamma = [\Gamma_i, \; \Gamma_f]$. Here, q and v are state variables, and i_m and f_e are considered as inputs. In fact f_e is unknown. An input estimator can be constructed by adding f_e as a third state variable, and modifying the system model as follows:

$$\begin{bmatrix} \mathbf{x}_{k+1} \\ f_{e_{k+1}} \end{bmatrix} = \begin{bmatrix} \Phi & \Gamma_f \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ f_{e_k} \end{bmatrix} + \begin{bmatrix} \Gamma_i \\ 0 \end{bmatrix} i_{m_k} \tag{3.51}$$

$$\mathbf{y}_k = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ f_{e_k} \end{bmatrix} \tag{3.52}$$

The external force f_e, together with the other state variables (position and velocity) can be estimated using this model with input i_m and output $\mathbf{y}$ (position). As long as the processing force f_e and the processing velocity are obtained, the impedance can be computed using the formulas given in equation (3.45). Figure 3.10 shows a schematic diagram for the Kalman estimator that is based on this method. Note that the hat variables (^) denote the estimates, but for convenience, this notation is dropped in our formulation.

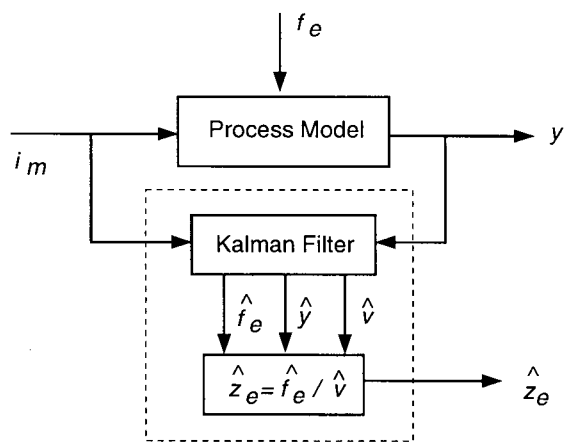


Figure 3.10: Schematic representation of an impedance estimator.

A simulation of this method is done by using the system model presented above. The external force f_e together with the joint speed can be estimated directly without computation using the force relation. The simulation is done for joint 3 of the PUMA 560 robot, with a simulated external force given by step changes shown in Figure 3.11.

The Kalman filter designed to estimate the external disturbance force takes the following parameters for noise characteristics:

$$Q = 1.0e^{-4} \begin{bmatrix} 0.001, & 0.0, & 0.0 \\ 0.0, & 0.001, & 0.0 \\ 0.0, & 0.0, & 4.0 \end{bmatrix}$$

and

$$R = 1.0e^{-4} \begin{bmatrix} 0.001 \end{bmatrix}$$

.

The simulation results are shown in Figure 3.11. It is observed that the estimated force tracks the actual force quite closely, but a delay of 0.5 seconds is present. Simulation results show that this method works quite well for estimation of the external force on the end-effector and, in general, for estimation of any form of system disturbance. However, in practical implementation, modelling error can affect the estimation accuracy quite significantly.

Case II: i_m is unknown or needs filtering. In this case v_d, the input control signal is accurate, and the electric model of the joint is known. Hence i_m can be estimated using a Kalman filter based on the model of the electrical subsystem. The electrical model of the joint is given by,

$$\dot{i}_m = -\frac{1}{\tau_e} i_m + \frac{k_e}{\tau_e}(u_m - e_b) \tag{3.53}$$

$$\dot{u}_m = -k_a(v_d + k_s i_m) \tag{3.54}$$

where i_m and u_m denote the state variables of motor current and voltage, respectively, and v_d denotes the input control signal. Also, e_b denotes the back EMF voltage

$$e_b = k_E v.$$

By combining the electrical subsystem and the mechanical subsystem, we obtain

$$\dot{x} = v \tag{3.55}$$

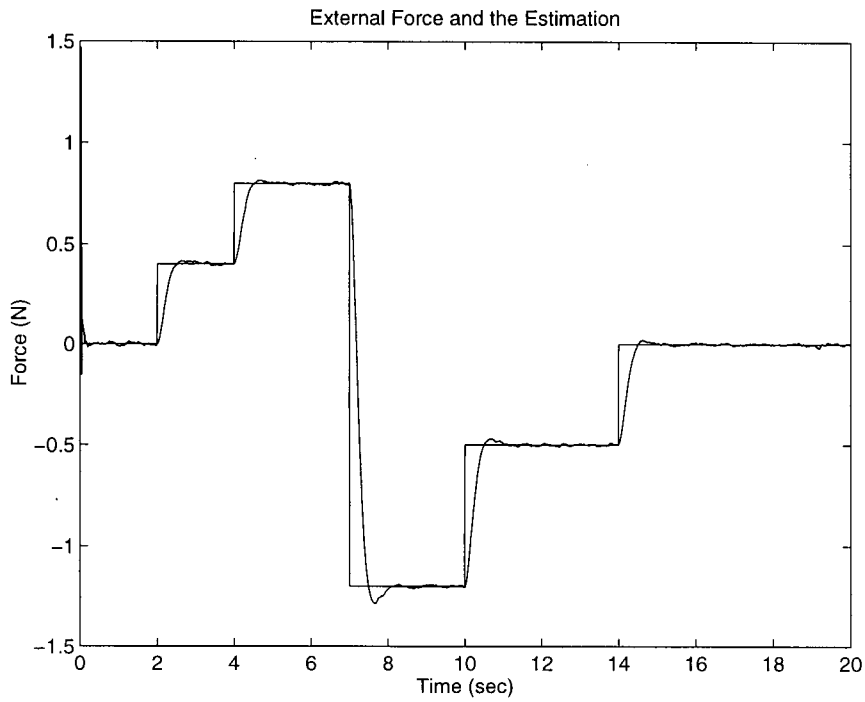


Figure 3.11: Estimation of external force using the mechanical model with position and motor current measurement only.

$$\dot{v} = -\frac{1}{\tau_m}v + \frac{k_m}{\tau_m}(k_\tau i_m - f_e) \tag{3.56}$$

$$\dot{i}_m = -\frac{1}{\tau_e}i_m + \frac{k_e}{\tau_e}(u_m - e_b) \tag{3.57}$$

$$\dot{u}_m = -k_a(v_d + k_s i_m) \tag{3.58}$$

The system has two inputs:

$$\mathbf{u} = [v_d,\ f_e]^T,$$

two outputs:

$$\mathbf{y} = [x,\ i_m]^T,$$

and four state variables,

$$\mathbf{x} = [x,\ v,\ i_m,\ u_m]^T.$$

The following two cases are considered:

1. i_m is measured. The entire system should be divided into two subsystems: the electrical subsystem which is represented by equations (3.57) and (3.58), and the mechanical subsystem which is represented by equations (3.55) and (3.56). The Kalman filters based on a reduced order system model is much more efficient than one based on the full order system model. Note that $\hat{i}_m$ is the output of the first system, which in turn is the input to the second system.

2. i_m is not measured. The entire system has the input vector $\mathbf{u} = [v_d,\ f_e]^T$, and the only output is the position $\mathbf{y} = [x]$.

Equations (3.55) through (3.58) can be rewritten in the matrix form as,

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} \tag{3.59}$$

$$\mathbf{y} = C\mathbf{x} \tag{3.60}$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{1}{\tau_m} & \frac{k_m k_\tau}{\tau_m} & 0 \\ 0 & -\frac{k_e k_E}{\tau_e} & -\frac{1}{\tau_e} & \frac{k_e}{\tau_e} \\ 0 & 0 & -k_a k_s & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & -\frac{k_m}{\tau_m} \\ 0 & 0 \\ -k_a & 0 \end{bmatrix}$$

and

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

As for the mechanical subsystem, we convert the overall system model into a discrete time model and add f_e as the fifth state variable. In this manner, we can estimate the external force

and the system velocity using a single Kalman filter. Due to the heavy computational load, however, this fifth order Kalman filter is not implemented online here. It may be implemented employing a more powerful computer system, or alternatively used off line for evaluation purposes.

3.4 Impedance Identification Using a Frequency Domain Model

The methods discussed in the previous sections are based on the manipulator models and primarily of time domain concepts. In these methods the environment is not explicitly modelled. The impedance of the environment is given by the ratio of its output (force) to its input (velocity). Extraction of material properties from such an impedance profile is difficult. Use of a linear combination of inertial, dissipative and elastic impedance components would provide useful information of the object characteristics, as discussed in Chapter 1. Since a global model would be highly nonlinear, a localized linear time-varying model (equations (3.18)) would be appropriate for representing the object model, where the time-variance of the parameters is used to represent the nonlinearity of the system.

We rewrite the localized linear model as,

$$\frac{\Delta F_e(s)}{\Delta V(s)} = m's + b' + \frac{k'}{s}. \tag{3.61}$$

In the frequency domain, as indicated before, the least squares estimation can be used to identify the parameters m', b', and k' of mechanical impedance at the process interface. The relationship between these physical system parameters and those estimated in a discrete-time model is given next.

The continuous time model may be rewritten as

$$\frac{\Delta X_e(s)}{\Delta F_e(s)} = \frac{1}{m's^2 + b's + k'} \tag{3.62}$$

It can be converted into the discrete-time Z domain as

$$\frac{\Delta X_e(z)}{\Delta F_e(z)} = \frac{b_1 z}{z^2 + a_1 z + a_2} \tag{3.63}$$

with

$$\begin{cases} a_1 &= -2e^{-\alpha T}\cos(\omega T) \\ a_2 &= e^{-2\alpha T} \\ b_1 &= \beta e^{-\alpha T}\sin(\omega T) \end{cases} \tag{3.64}$$

where, T is the sampling period, and

$$\begin{cases} \alpha &= \frac{b'}{2m'} \\ \omega &= \sqrt{\frac{k'}{m'} - \frac{b'^2}{4m'^2}} \\ \beta &= \frac{1}{m'\omega} \end{cases} \tag{3.65}$$

The inverse computation is carried out as

$$\begin{cases} m' &= \frac{1}{\beta\omega} \\ k' &= \frac{1}{\beta\omega}(\omega^2 + \alpha^2) \\ b' &= 2\frac{\alpha}{\beta\omega} \end{cases} \tag{3.66}$$

with

$$\begin{cases} \alpha &= -\frac{1}{2T}\ln a_2 \\ \omega &= \frac{1}{T}\cos^{-1}(\frac{-a_1}{2e^{-\alpha T}}) \\ \beta &= b_1/(\sqrt{a_2}\sqrt{1-(\frac{a_1}{2e^{-\alpha T}})^2}) \end{cases} \tag{3.67}$$

The recursive least squares estimation [7] may be utilized to identify the discrete time model of mechanical impedance of an object. Specifically, the Z transfer function of the impedance model is rewritten as

$$\begin{aligned} \frac{\Delta X_e(z)}{\Delta F_e(z)} &= \frac{b_1 z}{z^2 + a_1 z + a_2} \\ &= \frac{b_1 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}}, \end{aligned} \tag{3.68}$$

which can be expressed as a difference equation; thus,

$$\Delta x(k) = -a_1 \Delta x(k-1) - a_2 \Delta x(k-2) + b_1 \Delta f_e(k-1). \tag{3.69}$$

Now, let

$$\theta = \begin{bmatrix} a_1 & a_2 & b_1 \end{bmatrix}^T, \tag{3.70}$$

and

$$\phi(k) = \begin{bmatrix} -\Delta x(k-1), & -\Delta x(k-2), & \Delta f_e(k-1) \end{bmatrix}. \tag{3.71}$$

The recursive form of the least squares estimation can be written as follows:

$$K(k+1) = P(k)\phi^T(k+1)[\lambda + \phi(k+1)P(k)\phi^T(k+1)]^{-1} \tag{3.72}$$

$$\hat{\theta}(k+1) = \hat{\theta}(k) + K(k+1)[\Delta x(k+1) - \phi(k+1)\hat{\theta}(k)] \tag{3.73}$$

$$P(k+1) = [I - K(k)\phi(k+1)]P(k)/\lambda \tag{3.74}$$

where, $P(0)$ gives the initial condition of the error covariance matrix. The original parameters m', b' and k' can be evaluated once the discrete time model parameters b_1, a_1 and a_2 are identified in this manner.

Since a localized model is used in the identification process, $\phi(k)$ is referred to a local initial state of $x_n(0)$ and $f_n(0)$. Then we have

$$\Delta x_n(k) = x_n(k) - x_n(0) \tag{3.75}$$

$$\Delta f_{en}(k) = f_{en}(k) - f_{en}(0) \tag{3.76}$$

The subscript n denotes the nth segment of the model. The model is considered to be linear in each segment. The length of segment N depends on the bandwidth of the impedance which is a characteristic of the material being processed. At nth segment, the initial parameter is given by the last estimate in segment $n-1$,

$$\hat{\theta}_n(0) = \hat{\theta}_{n-1}(N-1) \tag{3.77}$$

By substituting equations (3.75), (3.76), and (3.77) into equations (3.71) through (3.74), we can estimate a new update of $\hat{\theta}$.

A non-recursive least squares procedure would also be suitable for the segmented parameter identification. For each segment we have,

$$\mathbf{y}_n = \Phi_n \theta_n \tag{3.78}$$

where,

$$\mathbf{y}_n = \begin{bmatrix} \Delta x_n(2) \\ \Delta x_n(3) \\ \cdots \\ \Delta x_n(N-1) \end{bmatrix}$$

$$\Phi_n = \begin{bmatrix} \phi_n(2) \\ \phi_n(3) \\ \cdots \\ \phi_n(N-1) \end{bmatrix}$$

in which,

$$\phi_n(k) = \begin{bmatrix} -\Delta x_n(k-1) & -\Delta x_n(k-2) & \Delta f_{en}(k-1) \end{bmatrix},$$

and

$$\theta_n = \begin{bmatrix} a_{n_1} \\ a_{n_2} \\ b_{n_1} \end{bmatrix}.$$

By the least squares method, we obtain,

$$\hat{\theta_n} = (\Phi_n^T \Phi_n)^{-1} \Phi_n^T \mathbf{y}_n, n = 1, 2, \cdots \tag{3.79}$$

Here the resolution of the estimation is one update per segment. However, this can be increased by over lapping the segments (see Figure 3.12)

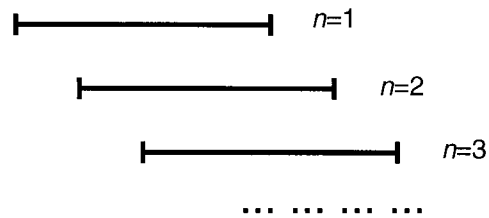


Figure 3.12: Overlapping of segments in estimation.

Figure 3.13 shows the estimated parameters of the discrete time impedance model of a cutting experiment (joint 5 of the PUMA 560 robot is employed here). The estimated parameter values are, approximately, $a_1 = -1.55$, $a_2 = 0.55$, and $b_1 = 0.2$. Then the inverse computation (equations (3.66) and (3.67)) is used to obtain the impedance parameters. The resulting values: $m' = 0.3764$, $b' = 2.25$, and $k' = 0.0$, are quite close to the true values. In fact, in the experiment, there was no spring-like connection; hence, $k' = 0.0$. Note that the dynamics of the manipulator joint is also included. As a result, the obtained parameters include the dynamic influence of the joint as well. As the parameters of the link, including the motor, are known, we can subtract them from the estimated parameters to obtain the process impedance.

3.5 Comparative Evaluation

The direct computation of mechanical impedance as the ratio of force/velocity is quite straightforward. But the resulting impedance estimates are found to be noisy. By using a general purpose Kalman filter, system velocity and acceleration are obtained at a reasonable accuracy for use in the computation of impedance. The estimated velocity and acceleration are much smoother than those obtained by direct digital differentiation and double differentiation of the position signal. A simplified physical system model with inertia and viscous damping (J, b) of the robot may be used in this method for computation of the processing force. Note that this model does need explicit information of the robot model. However, it is more robust than the second method of impedance estimation, with regard to robot modelling error.

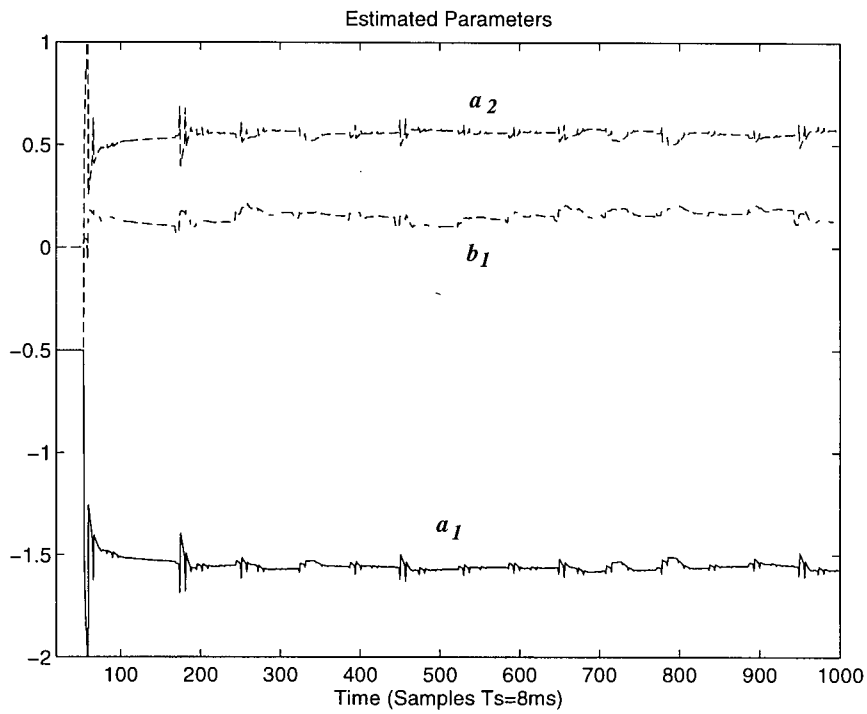


Figure 3.13: Estimated parameters of a frequency domain impedance model for joint 5 of the PUMA 560 manipulator.

The second method takes advantage of an accurately known physical model of the system (robot). It is actually not necessary to measure motor current in this method. However, this method relies on system model too much and the modelling error may affect the results significantly. Hence incorporation of the motor current sensing is desirable even in this approach.

The impedance transfer function could be extracted from the time series of input velocity and output force using fast Fourier transformation (FFT). Usually this conversion is time consuming, and not practical for online use. Therefore, if impedance parameters are required, the third method that is system identification should be applied to obtain the parameters. Since the model used in identification is a localized linear model, segment-based least squares identification is employed in the estimation of impedance parameters. This method provides more insight into the impedance information that reflects the material characteristics, particularly in terms of

inertial, dissipative, and elastic impedance characteristics.

3.6 Summary

Three basic methods were presented in this chapter for estimation of mechanical impedance. System modelling and estimation were the main focus of the chapter. In particular, for accurate estimation of process impedance, a satisfactory robot model should be incorporated.

Environmental modelling is rather difficult for robotic processing of objects that consist of inhomogeneous and flexible, natural material. In general, the model can be highly nonlinear and time-varying. However, a localized, time-varying linear model consisting of mass, spring, and damper components is derived for estimation of impedance parameters of the process interface. The time-variance of the model parameters is intended to account for the system nonlinearities.

Two methods were developed for impedance estimation primarily based on time domain models. The first method adopted a general mathematical model and constructs a Kalman filter based on this model. Both position and current of robot joint were measured and fed into this filter to yield a smooth output and estimated derivatives. Velocity and acceleration were obtained from these values. The second method employed a physical model of the system (robot) in estimation. The external interaction force was estimated instead of direct computation using noisy velocity and acceleration.

The frequency domain method could provide more details of the impedance information by identifying the basic impedance parameters pertaining to the inertial, flexible, and dissipative characteristics of the process interface. A localized model was used as before. Signals were localized in small segments in which a linear, model-based parameter identification was constructed. The different methods were further compared for identifying respective advantages and disadvantages using application examples and indicating situations where each approach would be appropriate.

Chapter 4

Extension and Application of Impedance Estimation

This chapter will extend the approaches for impedance estimation, which were developed in the previous chapter, to multiple degree-of-freedom (DOF) systems. A system model will be developed and discussed to extend impedance concepts in a single DOF system to a multiple DOF system, in the present context. Estimation of the external forces that are acting on the end-effector at the process interface are estimated using the Kalman filter method. Some special issues that arise in the impedance computation of multiple DOF system will be explored as well.

The impedance sensing technologies, in both single and multiple DOF systems, are applicable to commercial products in the computer numeric control (CNC) machining industry. Such applications will be discussed as well, in this chapter.

Section 4.1 will give an introduction to impedance sensing in multiple DOF systems. Section 4.2 will focus on robot dynamics and the feedback linearization approach. Section 4.3 will discuss the extension of the direct computation method of impedance to a multiple DOF system. Section 4.4 will present a Kalman filter-based method for impedance estimation in a multiple DOF system, using the feedback linearization approach. Section 4.5 will discuss applications of impedance sensing in industrial CNC machines which can be viewed as a 3 DOF robotic motion system with orthogonal (Cartesian) axes. Section 4.6 will summarize the chapter.

4.1 Extension of Impedance Sensing to Multiple DOF Systems

A robot typically consists of several joints with associated degrees of freedom. The end-effector is expected to utilize more than one DOF in carrying out a mechanical processing task. Under these conditions, the single DOF approaches developed in the previous chapter are not directly applicable. Manipulator configurations and coupled dynamics should be taken into consideration when extending to a multiple DOF robot system, the technologies developed for a single DOF system. The velocity and force at the end-effector are vectors which can have up to six components in the multiple DOF case. In particular, the differential equation that relates the joint displacements ($\mathbf{q}$) to the end-effector displacement vector ($\mathbf{x}$), is given by,

$$\delta\mathbf{x} = \mathbf{J}\delta\mathbf{q} \tag{4.1}$$

or

$$\dot{\mathbf{x}} = \mathbf{J}\dot{\mathbf{q}} \tag{4.2}$$

where $\mathbf{J}$ is the Jacobian matrix which can be computed using robot kinematic relations [42, 75].

Furthermore, if a force vector $\mathbf{f}_e$ is exerted on the end-effector (in the direction of $\delta\mathbf{x}$), the resulting torque vector τ_e that is needed at the robot joints in order to support this external force is obtained by applying the principle of virtual work; thus $-\mathbf{f}_e^T\delta\mathbf{x} + \tau_e^T\delta\mathbf{q} = 0$, (see equation (2.31)), which gives

$$\tau_e = \mathbf{J}^T\mathbf{f}_e \tag{4.3}$$

Then, assuming that the matrix $\mathbf{J}$ is square and nonsingular, we have

$$\mathbf{f}_e = (\mathbf{J}^T)^{-1}\tau_e \tag{4.4}$$

Note that τ_e here is not the drive torque vector of the robot joints, which has to overcome robot dynamics and other disturbances such as joint friction, but rather the torque component that corresponds to the forces on the end-effector due to mechanical processing of an object.

Of course, the system dynamics has a significant effect on the performance of a multiple DOF robot manipulator. A robot system cannot be treated as linear except for small, incremental motions and, furthermore, dynamic coupling between joints will be significant and should be considered. Therefore, the system dynamics should be included in the estimation models for robot impedance. In the present development, a linearized dynamic model of the robot with nonlinear feedback along with a Kalman filter will be employed in the estimation of the system state variables and mechanical impedance components at the process interface.

4.2 Robot Dynamics with a Linearizing Feedback Law

Lagrange's equation of motion for a robotic manipulator is given by,

$$\frac{d}{dt}(\frac{\partial L}{\partial \dot{\mathbf{q}}}) - \frac{\partial L}{\partial \mathbf{q}} = \tau \tag{4.5}$$

where vector $\mathbf{q}$ represents a set of generalized coordinates, which are typically the joint displacements, vector τ represents the generalized (non-conservative) forces, which are typically the drive torques at the joints and disturbance torques, and L is the Lagrangian which is defined as $L = K - P$, where, K and P are the kinetic and potential energies, respectively, of the manipulator. They may be computed by first determining the individual energies of the robot links and then forming the respective sums, as

$$K = \sum_{i=1}^{n} k_i \tag{4.6}$$

$$P = \sum_{i=1}^{n} p_i \tag{4.7}$$

Note that k_i and p_i are the kinematic and potential energies, respectively, of link i. In this manner, the dynamic model of a robot can be expressed in the form [5]:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{D}(\mathbf{q})[\dot{\mathbf{q}}\dot{\mathbf{q}}] + \mathbf{E}(\mathbf{q})[\dot{\mathbf{q}}^2] + \mathbf{G}(\mathbf{q}) = \tau \tag{4.8}$$

where

$$\mathbf{q} = [q_1, q_2, \cdots, q_n]^T$$

$$\dot{\mathbf{q}} = [\dot{q}_1, \dot{q}_2, \cdots, \dot{q}_n]^T$$

$$\ddot{\mathbf{q}} = [\ddot{q}_1, \ddot{q}_2, \cdots, \ddot{q}_n]^T$$

The symbols $[\dot{\mathbf{q}}\dot{\mathbf{q}}]$ and $[\dot{\mathbf{q}}^2]$ are used to denote the $n(n-1)/2$ vector of velocity products and the n vector of squared velocities, and are given by,

$$[\dot{\mathbf{q}}\dot{\mathbf{q}}] = [\dot{q}_1\dot{q}_2 \ \dot{q}_1\dot{q}_3 \ \cdots \ \dot{q}_1\dot{q}_n \ \dot{q}_2\dot{q}_3 \ \dot{q}_2\dot{q}_4 \ \cdots \ \dot{q}_{n-2}\dot{q}_n \ \dot{q}_{n-1}\dot{q}_n]^T,$$

$$[\dot{\mathbf{q}}^2] = [\dot{q}_1^2 \ \dot{q}_2^2 \ \cdots; \dot{q}_n^2]^T.$$

The terms in this dynamic equation may be identified as: inertial torques $\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}$ which are proportional to the joint accelerations $\ddot{\mathbf{q}}$; Coriolis torques $\mathbf{D}(\mathbf{q})[\dot{\mathbf{q}}\dot{\mathbf{q}}]$; centrifugal torques $\mathbf{E}(\mathbf{q})[\dot{\mathbf{q}}^2]$, and gravitational torques $\mathbf{G}(\mathbf{q})$. Also, the Coriolis terms and the centrifugal terms can be combined as $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}}$ in notation. Clearly, this model is quite nonlinear. Normally, it is not useful to linearize this model at a fixed operating point, as the robot operation covers its entire work space which can be quite extensive. Also even at a single operating point, the linearization can introduce substantial errors, for example, due to singularities. Instead, a feedback linearization approach is formulated below, which retains the full nonlinear model and linearizes its behavior through nonlinear feedback.

First, define a new input vector $\mathbf{a}$ as,

$$\mathbf{a} = \ddot{\mathbf{q}} \tag{4.9}$$

Also, define the state vector $\mathbf{x} = [\mathbf{q}^T, \dot{\mathbf{q}}^T]^T$. Then we have a linear state-space model given by,

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{a} \tag{4.10}$$

with

$$\mathbf{A} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 0 \\ I \end{bmatrix}$$

The relation between the original input τ and the transformed input $\mathbf{a}$ is obtained from equation (4.8) as,

$$\mathbf{M}(\mathbf{q})\mathbf{a} + \mathbf{D}(\mathbf{q})[\dot{\mathbf{q}}\dot{\mathbf{q}}] + \mathbf{E}(\mathbf{q})[\dot{\mathbf{q}}^2] + \mathbf{G}(\mathbf{q}) = \tau \tag{4.11}$$

which is the nonlinear feedback law. This relation can be expressed as,

$$\mathbf{a} = \mathbf{M}(\mathbf{q})^{-1}(\tau - \mathbf{D}(\mathbf{q})[\dot{\mathbf{q}}\dot{\mathbf{q}}] - \mathbf{E}(\mathbf{q})[\dot{\mathbf{q}}^2] - \mathbf{G}(\mathbf{q})) \tag{4.12}$$

In this manner, we have transformed the original nonlinear model into a linear model for the robot, as given by equations (4.10), using the nonlinear feedback law, as given by equation (4.12).

4.3 Impedance Computation at Process Interface

The method developed in Section 3.3 for computing mechanical impedance at the process interface can be extended to the present case of multiple DOF robot in a rather straightforward manner. By using the Kalman filter developed as equations (3.26), (3.27), and (3.34) – (3.38), the motor torques and velocities of all six joints can be estimated. The computation takes the same format as for the single DOF case, but the dynamic forces should be computed by using the multiple DOF system equations.

Figure 4.1 illustrates the structure and procedure of direct computation of mechanical impedance in the multiple DOF case. The position signals $\mathbf{q}$ and motor current signals $\mathbf{i_m}$ are measured directly from each joint as in the single DOF case. The Kalman filters that are

developed in Section 3.2 for smoothing the signals and estimating joint velocity and acceleration, are used for each joint separately, which yields the motor torques (converted into joint torques through gears), joint angles, velocities and accelerations.

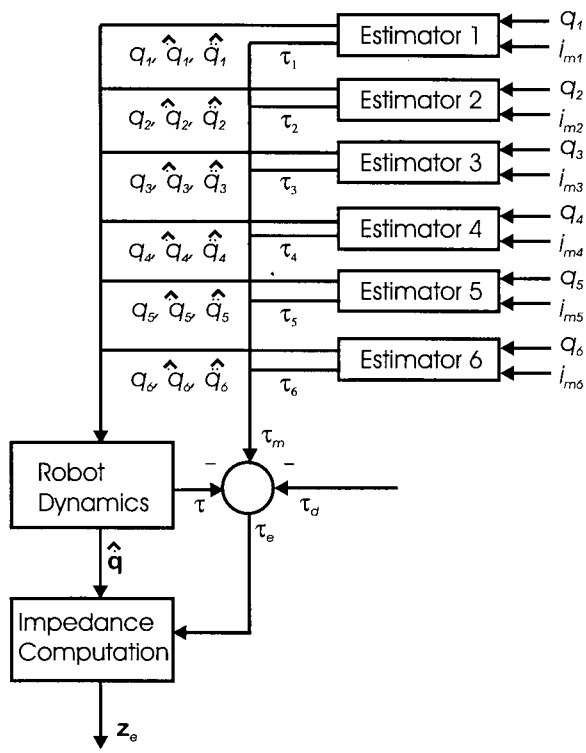


Figure 4.1: Illustration of impedance computation in a multiple DOF robot system.

The dynamic forces/torques of multiple DOF robot present a much more complicated problem than in a single DOF system. Specifically, we have

$$\tau = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{D}(\mathbf{q})[\dot{\mathbf{q}}\dot{\mathbf{q}}] + \mathbf{E}(\mathbf{q})[\dot{\mathbf{q}}^2] + \mathbf{G}(\mathbf{q}) \tag{4.13}$$

where

$$\tau = \tau_m - \tau_e - \tau_d - \mathrm{diag}(\mathbf{b})\dot{\mathbf{q}} \tag{4.14}$$

with τ_m denoting the motor drive torques, τ_e denoting torques reflected at the joints from the external (end-effector) forces, τ_d denoting disturbance torques, and $\mathrm{diag}(\mathbf{b})$ denoting a diagonal

matrix made from viscous damping coefficients at the joints ($\mathbf{b}$), i.e.,

$$[\mathrm{diag}(\mathbf{b})]_{ii} = b_i$$

Then, the torque reflected from the end-effector forces can be expressed as

$$\tau_e = \tau_m - \tau - \tau_d - \mathrm{diag}(\mathbf{b})\dot{\mathbf{q}} \tag{4.15}$$

The end-effector (external) forces in world coordinate can be computed by

$$\mathbf{f}_e = (\mathbf{J}^T)^{-1}\tau_e \tag{4.16}$$

where $\mathbf{J}$ is the Jacobian. The velocity vector at the end-effector is,

$$\mathbf{v} = \mathbf{J}\dot{\mathbf{q}} \tag{4.17}$$

By definition, the impedance in world coordinate $\mathbf{Z}_e$ is a matrix that satisfies the following relationship:

$$\mathbf{f}_e = \mathbf{Z}_e\mathbf{v} \tag{4.18}$$

Depending on the world coordinate frame that is chosen, $\mathbf{Z}_e$ may not be a diagonal matrix. However, the coordinate frame can be transformed (i.e., the directions of the world coordinates may be appropriately chosen) so that $\mathbf{Z}_e$ is a diagonal matrix. Then,

$$Z_{eii} = \frac{f_{ei}}{v_i} \tag{4.19}$$

4.4 Impedance Computation Based on Estimated External Force

Using a system model to estimate external force is not feasible due to the nonlinearity of the model, unless a local linear model is used, which is believed to be very limited in application due to large range of operation of robot arms. An alternative is to apply feedback linearization

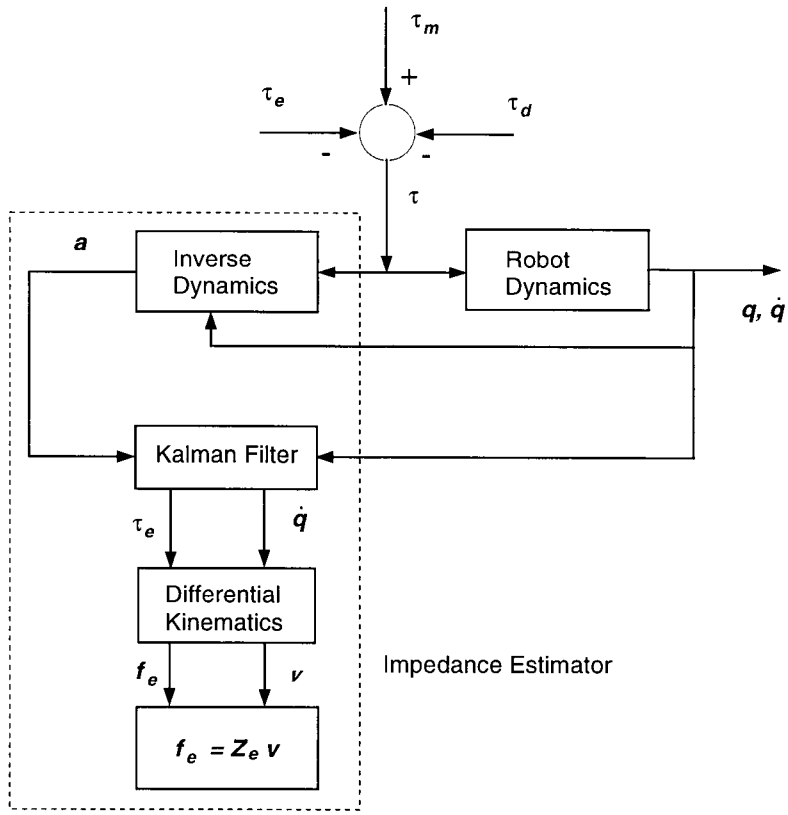


Figure 4.2: Estimation of impedance using feedback linearization technology.

technology first, and then use a Kalman filter. This approach is illustrated in Figure 4.2. As long as the joint force and the velocity are estimated, the rest of the computation of impedance is the same as in the direct computation.

Rewrite the linearized system equation (4.10) in the state-space form:

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{a} \tag{4.20}$$

together with the output equation,

$$\mathbf{y} = C\mathbf{x} \tag{4.21}$$

where

$$A = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ I \end{bmatrix}$$

$$C = \begin{bmatrix} I & 0 \end{bmatrix}$$

$\mathbf{a}$ is computed from the robot inverse dynamics; then,

$$\mathbf{a} = \mathbf{M}(\mathbf{q})^{-1}(\tau_m - \mathbf{D}(\mathbf{q})[\dot{\mathbf{q}}\dot{\mathbf{q}}] - \mathbf{E}(\mathbf{q})[\dot{\mathbf{q}}^2] - \mathbf{G}(q) - \tau_d - \tau_e - \mathrm{diag}(\mathbf{b})\dot{\mathbf{q}}) \tag{4.22}$$

This model can be converted into a discrete time difference model as,

$$\mathbf{x}_{k+1} = \Phi\mathbf{x}_k + \Gamma\mathbf{a}_k \tag{4.23}$$

$$\mathbf{y}_k = C\mathbf{x}_k \tag{4.24}$$

with nonlinear feedback,

$$\begin{aligned} \mathbf{a}_k &= \mathbf{M}_k^{-1}(\tau_{mk} - \mathbf{D}_k[\dot{\mathbf{q}}_k\dot{\mathbf{q}}_k] - \mathbf{E}_k[\dot{\mathbf{q}}_k^2] - \mathbf{G}_k - \tau_{dk} - \tau_{ek} - \mathrm{diag}(\mathbf{b})\dot{\mathbf{q}}_k) \\ &= \mathbf{M}_k^{-1}\tau_{mdk} - \mathbf{M}_k^{-1}\tau_{ek} \end{aligned}$$

where

$$\tau_{mdk} = \tau_{mk} - \mathbf{D}_k[\dot{\mathbf{q}}_k\dot{\mathbf{q}}_k] - \mathbf{E}_k[\dot{\mathbf{q}}_k^2] - \mathbf{G}_k - \tau_{dk} - \mathrm{diag}(\mathbf{b})\dot{\mathbf{q}}_k$$

Now consider τ_e as an input. Then we can expand the system as,

$$\mathbf{x}_{k+1} = \Phi\mathbf{x}_k + \Gamma\mathbf{M}_k^{-1}\tau_{mdk} - \Gamma\mathbf{M}_k^{-1}\tau_{ek} \tag{4.25}$$

$$\tau_{ek+1} = \tau_{ek} \tag{4.26}$$

or in the matrix format

$$\begin{bmatrix} \mathbf{x}_{k+1} \\ \tau_{ek+1} \end{bmatrix} = \begin{bmatrix} \Phi & -\Gamma\mathbf{M}_k^{-1} \\ 0 & I \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ \tau_{ek} \end{bmatrix} + \begin{bmatrix} \Gamma\mathbf{M}_k^{-1} \\ 0 \end{bmatrix} \tau_{mdk} \tag{4.27}$$

A Kalman filter can be constructed using this model. The outputs of the Kalman filter will be $\hat{\mathbf{q}}$, $\hat{\dot{\mathbf{q}}}$, and $\hat{\tau}_e$. The rest of the computation takes the same procedure as in the first method in Section 4.3.

4.5 Industrial Application of Impedance Sensing: Automatic Referencing of Material Surface in CNC Router Operation

As an illustration of the practical utility of the mechanical impedance technology that is developed in this thesis, the current section will present a commercial application. The developed impedance sensing approach is applied here to a 3D CNC router machine. For our treatment, a CNC machine is considered as a special-purpose robot, and the concepts developed in the thesis are directly applicable.

4.5.1 Background and rationale

CNC router machines are widely used in manufacturing applications, and particularly, in industries where cutting of wood and plastics is involved. In many applications, setting the position of the material surface with respect to the cutter, which is the reference point of cutting depth for the Z drive (see Figure 4.3), is very difficult and time consuming. Especially, when a cutting tool is broken, the corresponding position of the material surface, based on the new length of the tool, should be determined and set properly in order to maintain the required cutting depth. Inaccurate setting of the cutter reference position in machining operations would result in degraded product quality and increased wastage. Automatic and accurate sensing of the material surface will eleviate the problem and also will increase productivity. Major difficulties in automatic sensing of material surface are due to the variability of the length of the cutting tools and the fragility of the cutter blades.

Existing methods of automatic referencing of a material surface are based on microswitches or similar sensors. Microswitches are embedded in a sensor box. The sensor box has to be placed on the material surface for automatic detection. The machine tool is moved down slowly by the Z-drive, and the microswitches are monitored by the controller simultaneously. Upon contact with the surface, the switches are activated and the controller instructs the machine to

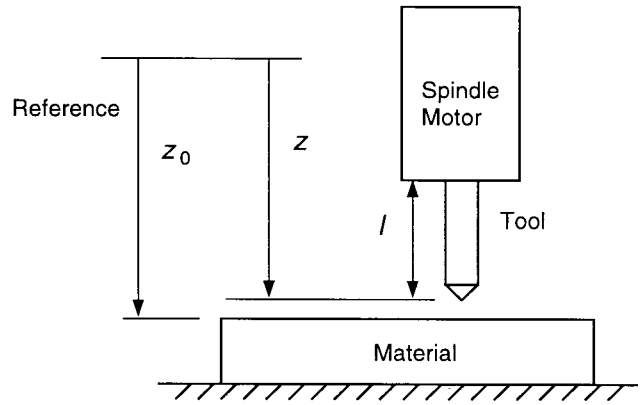


Figure 4.3: Locating the reference position of material surface, for the Z drive of a CNC router.

stop. Such methods have the following drawbacks in practice:

- Additional cost of the sensors and the sensor box, and input channels of the computer system;
- Additional wiring for the sensor box that could cause operational hazards; for example, if the operator forgets to remove the sensor box during operation, the wires could get tangled with the spindle;
- The area of the material surface should be able to at least support the sensor box, and hence, cannot be reduced arbitrarily.
- Potential damage to the tool if the control method is not appropriate.

The mechanical impedance method, as developed in the present work, has significant advantages in determining the location of a material surface. Before discussing these advantages, let us consider the principles associated with the sensing procedure. In the impedance control mode, the Z-drive together with the cutter head is moved down to the material surface. The Z-drive is controlled at a desired level of mechanical impedance that is safe for the cutter head. The Z-drive system together with the material surface is modelled as an equivalent

lumped-parameter system with masses, springs, and dampers. At the cutter-material interface, the cutter head is subjected to "mobility" of both the drive system and the material, generating a net velocity of $v_1 + v_2$, as illustrated in Figure 4.4. Hence the equivalent cutter impedance is,

$$\begin{aligned} z &= \frac{f}{v} = \frac{f}{v_1 + v_2} = \frac{1}{\frac{v_1}{f} + \frac{v_2}{f}} \\ &= \frac{1}{\frac{1}{z_1} + \frac{1}{z_2}} \end{aligned}$$

It follows that,

$$z = \frac{z_1 z_2}{z_1 + z_2} \tag{4.28}$$

where z_1 is the mechanical impedance of the Z-drive with respect to the cutter, and z_2 is the impedance of the material. Note that v_1 and v_2 are the velocity of the Z-drive and the material surface, respectively, when an interactive force f ($f = f_1 = f_2 = f_e$) is present between the cutter and the material. If z is too high, the cutter could be damaged in the process of automatic referencing the material surface. To avoid this problem, both z_1 and z_2 have to be kept low. Since z_1 is implemented by impedance control, it cannot be set at an arbitrarily low level, due to mechanical limits of the drive system and the disturbances that may be present. Therefore, if z_2 is too high, the combined impedance z also could be too high. A simple and convenient way to overcome this difficulty would be to use a passive damping pad, or two-ply paper on hard material surface. This will reduce the overall impedance to a desirable level.

Suppose that the Z-drive may be modelled as a combined system of lumped mass, damper, and Coulomb friction. The dynamic force from the actuator on contact with the material is (see Figure 4.5),

$$f = m_1\ddot{x}_1 + f_c\text{sgn}(\dot{x}_1) + b_1\dot{x}_1 + m_2\ddot{x}_2 + b_2\dot{x}_2 + k_2x_2 \tag{4.29}$$

where variables marked with subscript "$_1$" are parameters of the model of the Z-drive, and those with subscript "$_2$" are parameters of the model of the material. If the Z-drive is considered stiff,

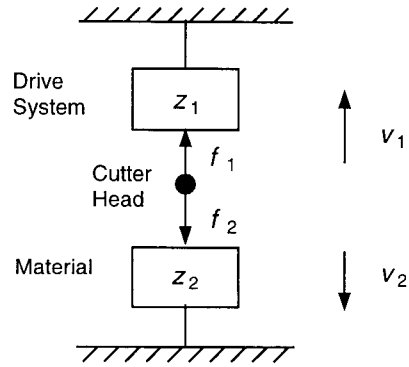


Figure 4.4: Combination of material impedance and Z-drive impedance.

we have,

$$x_1 = x_2 = x \tag{4.30}$$

Then the net force f_e applied to the material can be written as,

$$f_e = f - m_1\ddot{x} - f_c\text{sgn}(\dot{x}) - b_1\dot{x} = m_2\ddot{x} + b_2\dot{x} + k_2x \tag{4.31}$$

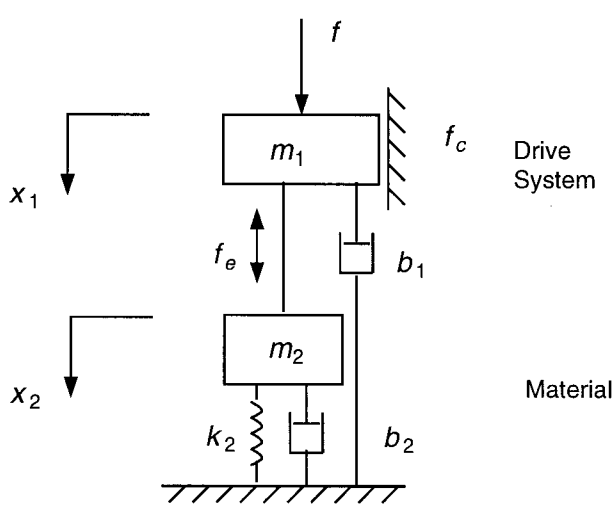


Figure 4.5: The model of interaction between Z-drive and material.

The motion impedance is monitored during motion. If it matches the specified impedance, according to some given criteria, the appropriate actions are taken. For example, if the

impedance reaches a specified threshold value, the motion is commanded to stop. Then the position of the material surface is taken as the stop position, or the stop position compensated by a constant displacement if an additional layer of paper is used. No extra electric hardware and wiring would be needed, in this approach.

4.5.2 Modelling and impedance computation

The Z-drive system consists of a DC servomotor, two guide rails, a ball screw transmission system, a holding mechanism and a spindle motor for the cutter. The system has the following parameters:

Back emf constant k_E	12.8 V/krpm
Torque constant k_τ	17.3 oz.in/A
Resistance R	0.72 Ω
Inductance L	2.5 mH
Rotor inertia J	0.035 oz.in.s^2
Electrical time constant t_e	0.0035 s
Mechanical time constant t_m	0.012 s
Transmission ratio G	0.2 in/rev
Mass of moving parts w	22 lb

Linear current amplifiers are employed. The ratio of mechanical torque/control-voltage k_v, gravity torque τ_g, Coulomb friction torque τ_c, and the viscous damping coefficient b are estimated using the following observed data in operation:

1. When the Z-drive is commanded to move upward at a constant speed of 5 in/s, the required control signal is $3.25V$;

2. When the Z-drive is commanded to move downward at a constant speed of 5 in/s, the

required control signal is $-2.25V$;

3. When the Z-drive is commanded to move upward at a constant speed of 0.5 in/s, the required control signal is $1.5V$;

4. When the Z-drive is commanded to move downward at a constant speed of 0.5 in/s, the required control signal is $-0.5V$.

Since the differences of the Coulomb frictions and the viscous damping coefficients of up and down motions are small, we assume that they are identical for both up and down motions. Then, the simplified friction model given in Chapter 2 yields the following equation:

$$\tau_m - \mathrm{sgn}(v)\tau_c - bv - \tau_g = 0 \tag{4.32}$$

where τ_m is the motor torque and v is the velocity of the positioning drive. The motor torque is measured by the control voltage u. Let the gain of the power amplifier be k_a and the torque constant be k_τ. Then, we have

$$\tau_m = k_a k_\tau u = k_v u \tag{4.33}$$

Therefore, equation (4.32) becomes

$$k_v u - \mathrm{sgn}(v)\tau_c - bv - \tau_g = 0 \tag{4.34}$$

Based on the observed data, we have following four equations:

$$3.25k_v - \tau_c - 5.0b - \tau_g = 0 \tag{4.35}$$

$$-1.5k_v + \tau_c + 5.0b - \tau_g = 0 \tag{4.36}$$

$$2.25k_v - \tau_c - 0.5b - \tau_g = 0 \tag{4.37}$$

$$-0.5k_v + \tau_c + 0.5b - \tau_g = 0 \tag{4.38}$$

Equation (4.35) minus equation (4.37), and equation (4.36) minus equation (4.38) yield the same result in both directions, for viscous damping,

$$k_v - 4.5b = 0; \tag{4.39}$$

Equation (4.35) plus equation (4.36), and equation (4.37) plus equation (4.38) yield the same result in both directions, for gravity torque,

$$1.75k_v - 2\tau_g = 0; \tag{4.40}$$

Now, τ_g may be determined by using the principle of virtual work; thus,

$$\tau_g\omega - vw = 0 \tag{4.41}$$

where ω is the angular velocity of the motor shaft, v is the linear velocity of the Z-drive, and w is the weight of the Z-drive. Then,

$$\begin{aligned} \tau_g &= \frac{v}{\omega}w \\ &= Gw \\ &= \frac{0.2}{2\pi}(22 \times 16) \\ &= 11.2(\text{oz-in}). \end{aligned} \tag{4.42}$$

Substitution of τ_g into equation (4.40) yields,

$$\begin{aligned} k_v &= \frac{2\tau_g}{1.75} \\ &= 12.8(\text{oz.in/V}), \end{aligned} \tag{4.43}$$

then,

$$k_a = \frac{k_v}{k_\tau} = \frac{12.8}{17.3} = 0.74(\text{A/V}), \tag{4.44}$$

Substitution of k_v into equation (4.39) yields,

$$\begin{aligned} b &= \frac{k_v}{4.5} \\ &= 2.8. \end{aligned} \tag{4.45}$$

From any of the equations in (4.35), (4.36), (4.37), and (4.38), we can calculate τ_c as,

$$\tau_c = 16.3(\text{oz.in}). \tag{4.46}$$

This is verified by direct measurement of friction in the experiment. The torque applied at the motor shaft to start moving the Z-drive downward is 16oz $\times$ 0.37inch. Hence, the resisting friction torque is the sum of the applied torque and the gravity torque; i.e., $16 \times 0.37 + \tau_g = 17.15$ oz.in. This is very close to but somewhat higher than the calculated τ_c, due to higher stiction.

By incorporating into equation (4.32), the external force applied to the material and the inertia force of the Z-drive, the system dynamic model becomes,

$$\tau_m - \text{sgn}(v)\tau_c - bv - \tau_g - Gf_e = Gma \tag{4.47}$$

Here, f_e is the force applied to the material, m is the mass of moving parts, a is the drive acceleration. Then,

$$\begin{aligned} f_e &= \frac{1}{G}(\tau_m - \text{sgn}(v)\tau_c - bv - \tau_g) - ma \\ &= \frac{1}{G}(k_v u - \text{sgn}(v)\tau_c - bv - \tau_g) - ma \end{aligned} \tag{4.48}$$

The impedance at the cutter head is computed in the time domain by,

$$z_e = \frac{f_e}{v} \tag{4.49}$$

Here, k_v, G, τ_c, b, τ_g and m are known, u is directly observed, and v should be obtained through a Kalman filter using position data. Acceleration a can be estimated as well from the Kalman

filter. But if the result is too noisy, the desired acceleration a_d can be used in place of the actual acceleration.

The nature of the present application stipulates that the motion must be carried out at very low speed. With high stiction of the mechanical system, the motion would be most likely in the unstable region of Figure 2.2 in Chapter 2. Due to the uncertainty of stiction, the system speed would be inconsistent. Impedance estimation using parameter identification that is based on a linear model is not reliable, and hence cannot be adopted in the present application.

4.5.3 Experimental data and pattern recognition

Figure 4.6 shows the signals of position, estimated velocity, estimated acceleration, and control voltage in an experiment of automatic material surface referencing. A 0.5-inch end mill cutter is used, and the object material is wood. The moving speed is kept at 0.06 in/s with some fluctuations due to stiction. The estimated acceleration is almost zero. Figure 4.7 shows the mechanical impedance at the cutter head. Before the cutter touches the material, the average value of the estimated impedance is necessarily very close to zero. The impedance goes up rapidly as the cutter head touches the material.

In theory, if the measured impedance matches the impedance of the material, the automatic referencing is achieved. However, this method is very difficult and unreliable due to noise and unstable motion. Hence, the following two criteria are employed jointly, in the present application: 1) thresholding; and 2) slope matching (see Figure 4.8). In practice, data in the beginning of motion is discarded due to high unstable stiction and inconsistent acceleration. As a result, the present method requires a minimum initial distance between the cutter and the material; typically 0.2-0.5 inches.

The system status is measured by using the summation of squared error between real-time

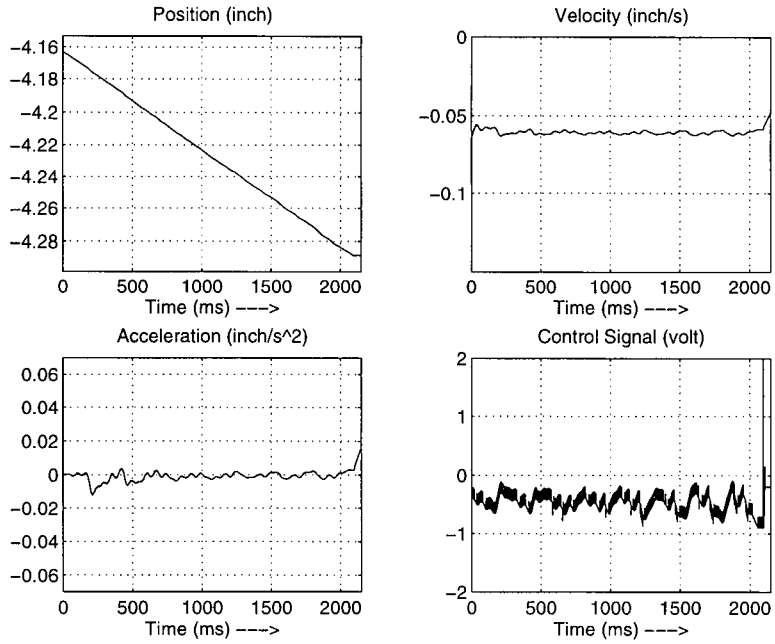


Figure 4.6: Experimental data for the Z-drive in automatic material surface referencing (0.5-inch end mill and object material: wood). (a). position, (b). estimated velocity, (c). estimated acceleration, and (d). control effort.

data and the given pattern; thus,

$$\Upsilon = \sum_{i=n}^{N+n} (z_i - z_{p_i})^2, n = 100, 101, \cdots \text{(first 100 points are ignored)} \tag{4.50}$$

where Υ denotes the sum of squared error; N denotes the length of the pattern; n denotes the data point number in sensing and control signals; and z_i and z_p denote real-time impedance and pattern impedance, respectively. When Υ is less than a specified value Υ_d,

$$\Upsilon < \Upsilon_d \tag{4.51}$$

the cutter is considered to touch the material surface and should be stopped.

Another experiment is carried out using a quarter inch ball mill to touch an aluminum plate with a layer of 2-ply paper. In this case, the contact process was found to be much shorter than that of wood about 0.2 seconds for aluminum. At speed of 0.05 in/s, the process lasted for

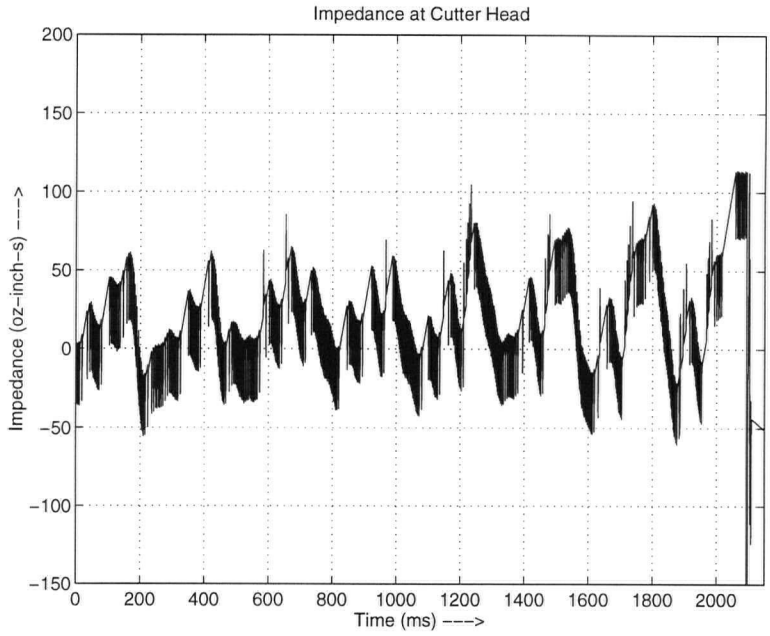


Figure 4.7: Mechanical impedance at cutter end (0.5-inch end mill and object material: wood).

0.01 inch (the thickness of the 2-ply paper was about 0.008 inch). This procedure left a light mark from the blades on the first layer of paper, but the second layer was not reached. This confirms that the blades were not damaged in any manner during the process (the layer of paper was not even cut through). Figures 4.9 and 4.10 show the process signals and the mechanical impedance at the cutter.

4.6 Summary

The general method of extending the impedance estimation technologies to multiple DOF systems is to use differential kinematics to convert measured joint signals into forces/torques and velocities at the tool tip, expressed in world coordinates, and then compute the diagonal impedance matrix (along canonical motion directions) in world coordinates. In this chapter, two time domain impedance estimation methods were extended to the case of multiple DOF

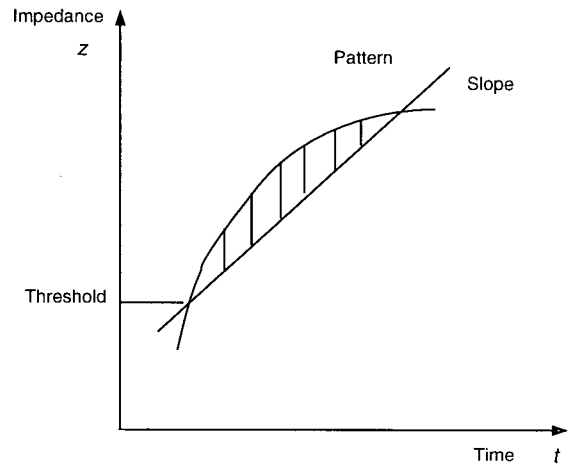


Figure 4.8: Pattern matching in automatic material surface referencing using the mechanical impedance approach.

system. Feedback linearization was used prior to employing a Kalman filter for this purpose in the second method, where external forces are estimated instead of direct computation as in the first method.

The developed impedance sensing technology was implemented in CNC router machine. Specifically, the developed method of impedance sensing was implemented in a commercial CNC router machine, for sensing the position of a material surface for the system referencing for subsequent processing. The approach was found to be very reliable and convenient to use. The cutter blades were not damaged when the processing material was wood, plastic, or aluminum. For very hard material such as steel, a passive pad or a layer of 2-ply paper should be used to separate the cutter head from the processing material (thereby reducing the material impedance). The accuracy of detection was found to be high in all the experiments; for example, the positioning error in the experiments was equal to or less than 0.001 inches for aluminum and plastics, and less than 0.002 inches for wood and other soft materials. These results are quite acceptable in CNC router operations.

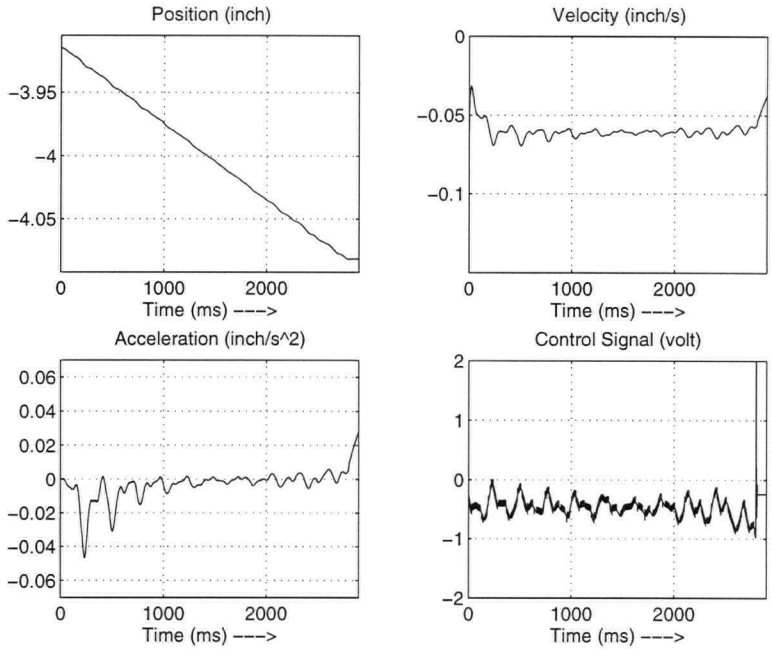


Figure 4.9: Data for the Z-drive in automatic material surface detection (0.25-inch ball mill and aluminum with 2-ply paper). (a). position, (b). estimated velocity, (c). estimated acceleration, (d). control effort.

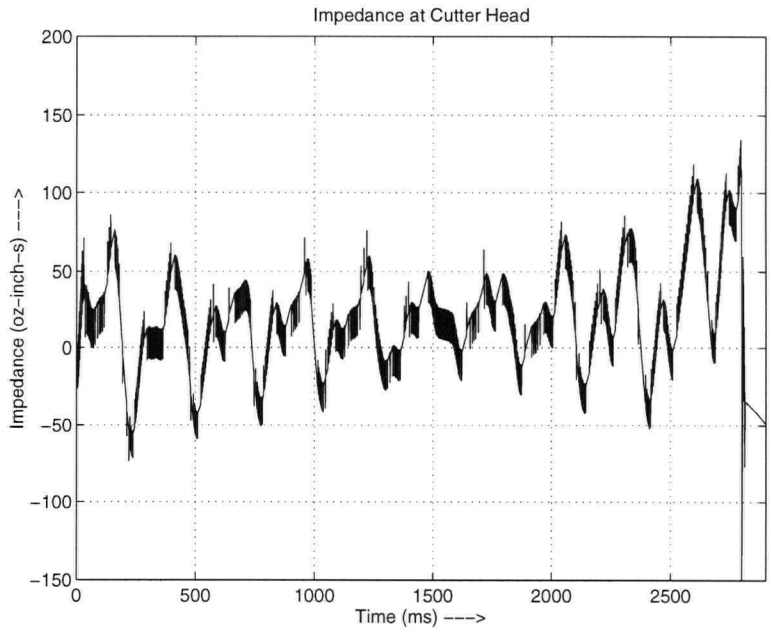


Figure 4.10: Mechanical impedance at cutter end (0.25-inch ball mill and aluminum with 2-ply paper).

Chapter 5

Sensor-Based Dual-Mode Hierarchical Control System

This chapter represents a control system that is able to incorporate the sensing estimation technologies developed in the present research and described in the previous chapters. The concepts of integrated gross motion and fine manipulation will be introduced. Then a dual-mode control system will be developed and implemented, which uses position control and impedance control for the tasks of gross motion and fine manipulation, respectively. Section 5.2 will discuss a hierarchical system for implementing the dual-mode robot control. Section 5.3 will give details of low-level robot control within the dual-mode system. Sections 5.4 and 5.5 will address the middle-level and high-level processes of path planning, information preprocessing and intelligent task control. Section 5.6 will summarize the chapter.

5.1 Introduction

Feedback of sensory information is essential for high-level task control in flexible manipulation. This feature enables the robot system to change the motion plan and control policy based on feedback data. Its implementation will require a robot control system that is capable of receiving and properly using feedback information from process sensors. Conventional robot controllers do not possess this capability. It is natural to organize a robot control system in a hierarchy of functional levels, where sensory information feedback should be allowed from the bottom level to upper levels, and this information should be provided in a compatible form at each level of the hierarchy.

A main objective of the robotic system that is developed in the present research will be to

employ mechanical impedance at the process interface for properly controlling robotic tasks of processing inhomogeneous and flexible natural objects. There are two general types of robotic actions; namely, *gross motion* and *fine manipulation* [15] that are involved in mechanical processing of such objects.

Gross motion. Gross motion is involved in tasks where a robot moves in a significant span of its workspace, usually at high speeds. Accuracy is typically not critical in these operations, except perhaps at a finite set of waypoints and the end point of the trajectory. However, task speed, stability, optimization of dynamic interaction, and obstacle avoidance are of importance here. Gross motion is usually intended for moving the robot end-effector to the vicinity of the destination without interacting with the environment. Hence motion control algorithms are the primary control schemes that are applicable in gross motion.

Fine manipulation. In fine manipulation a robot interacts with its environment to perform actions on an object that is constrained in some manner by the environment. These actions are usually carried out in small movements of medium to low speeds, but at high precision and with the application of desired levels of forces to the object. In fine manipulation, if the robot speed is low, the coupled Coriolis and centrifugal forces of the robot links are less important than the forces that are applied to the object by the end effector in interacting with the environment. Impedance control is desirable in compliant tasks of this type where force-motion relations play an important role in accomplishing the objective.

In a large number of manipulation tasks, both gross motion and fine manipulation are necessary. Consequently, a robot control system preferably should include schemes for fine manipulation as well as gross motion. These schemes may be partitioned as separate control modules, where switching between modules will be necessary depending on the robotic action. The mode switching should be controlled by a high-level intelligent task controller according to the task specifications.

5.2 Hierarchies in Function Level

Figure 5.1 shows a hierarchical robot control system. Note that this structure is constructed by associating the hierarchical levels to the system functions. In other word, the hierarchical level is determined by the function level. Although hardware-oriented implementations in the past have adopted similar hierarchical models, a new software-oriented implementation is developed in the present work using a model of this type that is based on the functional requirements of the system at different activity levels. Observe the two tracks of the system in Figure 5.1, with one track going up from the sensors (Information Track) and the other track coming down into the actuators (Commanding Track).

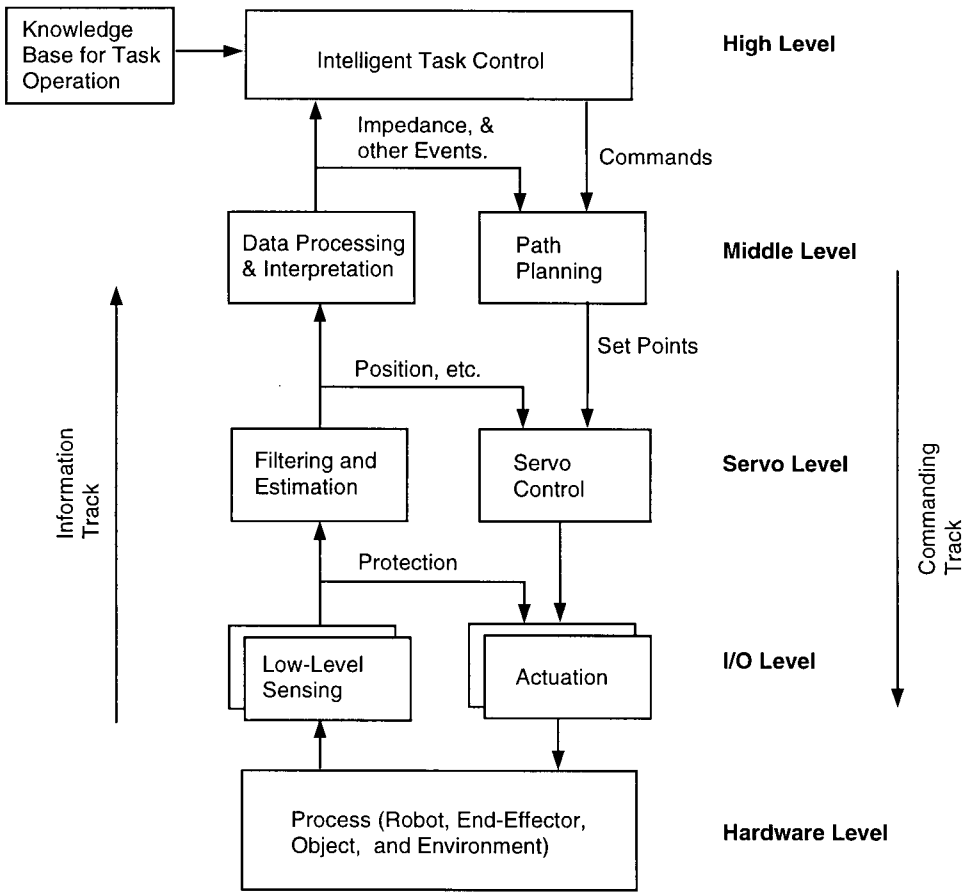


Figure 5.1: A sensor-based hierarchical robot control system.

The lowest layer consists of hardware devices. Necessary sensors, transducers, and actuators should be provided within this level. Resolution, accuracy, and speed (bandwidth) of the sensors and actuators are crucial in this layer, and should be considered for a given set of system specifications. The hardware system of Puma 560 controller which occupies this layer, has been discussed in detail in Chapter 2.

The I/O layer operates the basic I/O hardware, samples the data, and controls the actuators. This layer provides the functionality of sensing and actuation for higher layers. The functions of sensing and actuation are directly connected through a protection module that shuts the system down in emergency without high-level interaction.

The servo control layer computes and transmits the drive control signals to the actuators. Timing is very important in this layer since it performs servo control of the hardware devices. Basic functions of the layer include filtering of sensed data, estimation of system parameters and state variables, computation of motion error, and generation of feedback control signals. Conditioning of the sensory information is performed in the left track (Information Track) and generation of the servo control signal is carried out in the right track (Commanding Track). This layer runs at a high frequency; typically, 500Hz or 1KHz.

The middle layer contains two tracks as well. In the information side, a data preprocessing capability is provided to analyze and interpret the low-level signals and transmit appropriate high-level information (e.g., report on system operation status) to the upper level of intelligent task control. At the actuation/commanding side, a path planner performs command interpretation, and sets desired motion trajectories which are transmitted to the servo systems. A closed loop at this level capacitates system monitoring. Motion paths can be changed at this level according to the real-time data and the criteria set by the intelligent task controller.

The top layer is responsible for intelligent task control. The intelligent task control distinguishes itself from traditional task control by two primary features: (1) it adopts a closed loop method that enables sensor-based online task planning, and (2) it employs knowledge-based

decision making. In this layer, the most appropriate commands are decided upon online, based on the system status and employing a knowledge base. Online feedback is quite useful in task programming, mainly to handle uncertainties and unexpected variations. For example, in the robotic fish slicing task, first the backbone position of a salmon can be set as an unknown variable, and subsequently assigned a value once the system detects the bone during real-time operation. The task controller should be able to handle the unknown variables properly. More details on these various considerations will be given later.

In the left track where information feedback goes from the bottom to the top, the information is represented in an appropriate format at each level. The structure that is designed in the present research contains three levels of feedback except for the I/O level protection. Sensor-based control can be performed at each level, which is an important feature that enables flexible operation. In the salmon-slicing task, for instance, the servo level controls the cutter motion. The middle level monitors the process, computes impedance information, and stops operation or changes the motion paths according to online data. The top level makes knowledge-based decisions on tasks. For example, in our salmon slicing task, when the cutter hits the backbone, the cutting impedance becomes very high, and as a result a "stop path" command is sent to the path planner, and the high-level intelligent system determines a new path by interacting with the path planner.

This architecture inherits the characteristics of a general hierarchical system [31]. There are three basic issues that are important here; namely, intelligence, accuracy, and bandwidth. We will briefly address these issues now.

Intelligence. Generally, the higher the hierarchical level, the more intelligent the necessary control actions. The low-level servo system requires little intelligence, as it performs closed loop control using a specified algorithm. The high-level intelligent task controller interprets the input information and performs decision making related to task-level actions; for example, those pertaining to process performance, efficiency, and product quality, by using a knowledge

base and system information.

Accuracy. The lower the hierarchical level, the more accurate the information and control actions. The instantaneous accuracy of the system is governed by the low-level servo system, while the high-level system modules contribute intelligence to the overall system and govern its performance. Usually, decisions at high levels may be made by using qualitative and incomplete information. Furthermore. the system should be able to learn and improve its performance with time. An intelligent system should be able to handle these realities.

Bandwidth. In the present context, bandwidth represents the speed or frequency of operation. The lower the hierarchical level, the higher the operation bandwidth. In particular, the servo system requires a high control bandwidth. The servo control bandwidth is determined by factors such as sampling rate of sensor data and the time needed to compute the control action, and are limited by the hardware capabilities, while the high-level control bandwidth is usually determined by the task requirements and knowledge-based decision making. Task control bandwidth can be significantly lower than the hardware control bandwidth. In particular, the time requirements for task-level decision making may be lengthy and varied, while the duration of a servo loop control cycle is more deterministic. In the implementation of a real-time intelligent system, there are conflicts between the real-time requirements (to meet deadlines) and the intelligent system requirements (to search for a good solution).

The execution of multiple tasks requires parallel computation. Communication between the layers of a hierarchical system must be asynchronous since different layers run at different bandwidths. These issues will be discussed in Chapter 7 in the context of an implementation model (ROACS).

5.3 Low-Level Control Algorithms

The servo system provides the capability to execute high-level commands [31]. Two modes, position control and impedance control, are designed to meet task requirements. Control modes should be selectable in run time for different jobs.

5.3.1 Position control of gross motion

Position control is relatively simple, reliable, and less computationally intensive. It is appropriate for control of gross motion of a robot at high speed. Several methods of position control will be discussed here and corresponding control characteristics will be compared.

The simplest method that is considered is the independent-joint proportional-derivative (PD) control with only a position reference. The commercial Unimate MARK II controller employs this control method [21]. Some modifications and compensations, for example, by augmenting the control signal with a DC offset, are incorporated as well. The PD control law can be expressed by,

$$\tau = \mathbf{K}_p(\mathbf{q}_d - \mathbf{q}) - \mathbf{K}_d\dot{\mathbf{q}} \tag{5.1}$$

where τ denotes a vector of the control torque of the robot joints, $\mathbf{q}_d$, and $\mathbf{q}$ denote vectors of desired joint positions and actual joint positions, respectively, and $\dot{\mathbf{q}}$ denotes the velocity vector of the robot joints. Note that $\mathbf{K}_p$, and $\mathbf{K}_d$ are diagonal matrices representing proportional gains and derivative gains, respectively. This method can be improved by using the velocity references $\dot{\mathbf{q}}_d$ and the feed-forward of gravity torques $\mathbf{G}(\mathbf{q}_d)$.

$$\tau = \mathbf{K}_p(\mathbf{q}_d - \mathbf{q}) + \mathbf{K}_d(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}_d) \tag{5.2}$$

This control method which employs simple computation yields reasonably good control performance [3], and as a result, it is used in many robot control systems.

To consider the fully coupled dynamics of a robot manipulator in control, equation (5.2) is further enhanced as,

$$\tau = \mathbf{K}_p(\mathbf{q}_d - \mathbf{q}) + \mathbf{K}_d(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}) + \mathbf{M}(\mathbf{q}_d)\ddot{\mathbf{q}}_d + \mathbf{C}(\mathbf{q}_d, \dot{\mathbf{q}}_d)\dot{\mathbf{q}}_d + \mathbf{G}(\mathbf{q}_d) \tag{5.3}$$

where $\mathbf{M}(\mathbf{q}_d)\ddot{\mathbf{q}}_d$ is the inertial torque term in robot dynamic equation (4.8), and

$$\mathbf{C}(\mathbf{q}_d, \dot{\mathbf{q}}_d)\dot{\mathbf{q}}_d = \mathbf{D}(\mathbf{q}_d)[\dot{\mathbf{q}}_d\dot{\mathbf{q}}_d] + \mathbf{E}(\mathbf{q}_d)[\dot{\mathbf{q}}_d^2]$$

is a combination of the Coriolis torque term and the centrifugal torque term of equation (4.8). Note that the feed-forward torque is computed using the reference values of position, velocity and acceleration. Due to the tracking error, this cannot compensate for dynamic torque completely, as the reference signals (not the actual values) are used in the computation.

Now modify the feed-forward terms in equation (5.3) by using actual position $\mathbf{q}$, velocity $\dot{\mathbf{q}}$, and desired acceleration $\ddot{\mathbf{q}}_d$. We obtain,

$$\tau = \mathbf{K}_p(\mathbf{q}_d - \mathbf{q}) + \mathbf{K}_d(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}) + \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_d + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \tag{5.4}$$

Substituting the robot dynamic equation (4.8) into equation (5.4), we obtain,

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) = \mathbf{K}_p(\mathbf{q}_d - \mathbf{q}) + \mathbf{K}_d(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}) + \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}_d + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \tag{5.5}$$

By canceling the like terms on the two sides, the equation (5.5) can be written as,

$$\mathbf{M}(\mathbf{q})(\ddot{\mathbf{q}}_d - \ddot{\mathbf{q}}) + \mathbf{K}_d(\dot{\mathbf{q}}_d - \dot{\mathbf{q}}) + \mathbf{K}_p(\mathbf{q}_d - \mathbf{q}) = \mathbf{0} \tag{5.6}$$

Let,

$$\mathbf{K}_d = \mathbf{M}(\mathbf{q})\mathbf{K}'_d, \tag{5.7}$$

$$\mathbf{K}_p = \mathbf{M}(\mathbf{q})\mathbf{K}'_p, \tag{5.8}$$

where $\mathbf{M}(\mathbf{q})$ is assumed positive definite (which is the usual case) and, the position error

$$\mathbf{e} = \mathbf{q}_d - \mathbf{q}. \tag{5.9}$$

Then, equation (5.6) becomes,

$$\ddot{\mathbf{e}} + \mathbf{K}'_d\dot{\mathbf{e}} + \mathbf{K}'_p\mathbf{e} = \mathbf{0}. \tag{5.10}$$

By selecting the gains $\mathbf{K}'_p$ and $\mathbf{K}'_d$ to satisfy the requirements of system performance as determined by equation (5.10), the control law (5.4) becomes

$$\tau = \mathbf{M}(\mathbf{q})(\ddot{\mathbf{q}}_d + \mathbf{K}'_p(\mathbf{q}_d - \mathbf{q}) + \mathbf{K}'_d(\dot{\mathbf{q}}_d - \dot{\mathbf{q}})) + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) \tag{5.11}$$

This is known as the computed torque method. Note that the robot coupled dynamics is fully compensated for, in this method. Research in reference [3] has shown that computed torque method is robust with respect to modelling error.

5.3.2 Impedance control of fine manipulation

Impedance control [93] has been proposed by Hogan [54, 55, 56] in the context of robotic tasks. In this approach, the relationship between the force applied by a manipulator end-effector and the motion of the end-effector is controlled according to some task specifications. This control strategy can be applied to tasks that need a force to generate motion according to a suitable relation, which is the case in many applications such as assembly and cutting. This type of control will be particularly useful in fine manipulation or mechanical processing of an object using a robot, as considered in the present research.

The fundamental idea underlying impedance control of robots is that environmental constraints generally impose a relationship between the motion of the end-effector of a manipulator and the contact force that is exerted by the processed/manipulated object and its environment on the end-effector. Hogan points out that this relationship may be modeled by a generalized impedance consisting of some inertial, damping, and stiffness characteristics. For analytical representation, the combined system of manipulator-object-environment should be well behaved, much like an electrical circuit of matched impedances. More recently, impedance control has been studied by many researchers, within the context of robotics. Lee *et al.* (1994)

[64] studied a stiffness control method, which is a special case of impedance control, for a coupled tendon-driven robot hand. Erlic and Lu (1993) [39] presented an impedance control method without using velocity measurement. McCormick and Schwartz (1993) [68] presented an overview investigation on impedance control for robot manipulators, where they studied the stability of impedance control. It should be mentioned that the stability of an unconstrained manipulator does not imply stability of the same manipulator when it is in contact with a constraining environment. Such unstable behavior is probably caused by the dynamic interactions and unknowns of the environment, and improper task control commands. When a robot is interacting with a dynamic environment, the overall system changes, and hence the stability condition also changes. Then, a high-level monitoring and intelligent control may be needed to realize stable and accurate performance of the overall robotic system in conjunction with its environment.

It is also clear that impedance control gives only implicit impedance requirements for the interface between a manipulator and its environment. The controller may attempt to accommodate such interaction without using an accurate interpretation of the object itself. Explicit mechanical impedance may be required for some manipulation tasks. For example, when a robotic cutter is processing an object consisting of regions with vastly differing material properties, it is important to know the nature of the region that is being cut, in order to either apply a suitable cutting force or to alter the manipulation strategy (e.g., avoid cutting into the backbone). Then, more sophisticated, model-based control may be required. However, impedance control at low level is also useful for fine manipulation, since the proper behaviour between force and motion would be important in achieving a desired performance.

Suppose that the desired impedance at the process interface between the cutter and the object is given by,

$$\mathbf{M}_d\ddot{\mathbf{x}} + \mathbf{B}_d\dot{\mathbf{x}}_e + \mathbf{K}_d\mathbf{x}_e = \mathbf{f}_e \tag{5.12}$$

with $\mathbf{x}_e = \mathbf{x} - \mathbf{x}_d$, where $\mathbf{x}$ represents position and orientation of the end-effector; $\mathbf{x}_d$ denotes the desired position and orientation of the end-effector; and $\mathbf{f}_e$ denotes a generalized force that is exerted on the cutter by the process environment with the associated motion $\mathbf{x}$. $\mathbf{M}_d$, $\mathbf{B}_d$, and $\mathbf{K}_d$ denote the desired mass, viscous damping coefficient, and stiffness matrices of the controled system. In real-time fine manipulation, a desired impedance can be assigned online by task specifications.

By using robot kinematics, we have,

$$\mathbf{x} = \mathbf{f}(\mathbf{q}) \tag{5.13}$$

where $\mathbf{q}$ is a vector of joint angles. Then, the instantaneous motion is represented by

$$\dot{\mathbf{x}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{5.14}$$

where $\mathbf{J}$ is the Jacobian matrix of the manipulator. The torque that is reflected to the robot joints by the external force $\mathbf{f}_e$ can be expressed by

$$\tau_e = \mathbf{J}^T(\mathbf{q})\mathbf{f}_e. \tag{5.15}$$

which may be derived by the principle of virtual work.

Now consider the system dynamic equation,

$$\begin{aligned} \tau_m + \tau_e &= \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{f}(\dot{\mathbf{q}}) \\ &= \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{H}(\mathbf{q}, \dot{\mathbf{q}}) \end{aligned} \tag{5.16}$$

where $\mathbf{M}, \mathbf{C}, \mathbf{G}, \mathbf{f}$ represent the inertia matrix, the Coriolis and centrifugal coefficient matrix, gravity torque vector, and the friction torque vector. The function $\mathbf{H}$ is defined as,

$$\mathbf{H}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + \mathbf{G}(\mathbf{q}) + \mathbf{f}(\dot{\mathbf{q}})$$

Substitution of equation (5.15) into (5.17) yields,

$$\tau_m + \mathbf{J}^T(\mathbf{q})\mathbf{f}_e = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{H}(\mathbf{q}, \dot{\mathbf{q}}) \tag{5.17}$$

which can be transformed into the end-effector motion vector $\mathbf{x}$, as

$$\mathbf{J}(\mathbf{q})^{-T}\tau_m + \mathbf{f}_e = \mathbf{M}_x(\mathbf{q})\ddot{\mathbf{x}} + \mathbf{H}_x(\mathbf{q}, \dot{\mathbf{q}}) \tag{5.18}$$

where the transformed quantities are,

$$\mathbf{J}^{-T} = (\mathbf{J}^T)^{-1} \tag{5.19}$$

$$\mathbf{M}_x(\mathbf{q}) = \mathbf{J}^{-T}(\mathbf{q})\mathbf{M}(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q}) \tag{5.20}$$

$$\mathbf{H}_x(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{J}^{-T}(\mathbf{q})\mathbf{H}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{M}_x(\mathbf{q})\dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}}. \tag{5.21}$$

By substituting $\ddot{\mathbf{x}}$ from equation (5.12) into equation (5.18), we obtain the control law,

$$\begin{aligned} \tau_m &= \mathbf{J}^T(\mathbf{q})[\mathbf{H}_x(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{M}_x(\mathbf{q})\mathbf{M}_d^{-1}(\mathbf{B}_d\dot{\mathbf{x}}_e \\ &\quad +\mathbf{K}_d\mathbf{x}_e) + (\mathbf{M}_x(\mathbf{q})\mathbf{M}_d^{-1} - \mathbf{I})\mathbf{f}_e] \\ &= \mathbf{H}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{M}(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q})\dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}} \\ &\quad -\mathbf{M}(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q})\mathbf{M}_d^{-1}(\mathbf{B}_d\dot{\mathbf{x}}_e + \mathbf{K}_d\mathbf{x}_e) \\ &\quad +(\mathbf{M}(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q})\mathbf{M}_d^{-1} - \mathbf{J}^T(\mathbf{q}))\mathbf{f}_e \end{aligned} \tag{5.22}$$

If we set $M_d = M_x$, then the control law becomes,

$$\tau_m = \mathbf{H}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{M}(\mathbf{q})\mathbf{J}^{-1}(\mathbf{q})\dot{\mathbf{J}}(\mathbf{q})\dot{\mathbf{q}} - \mathbf{J}^T(\mathbf{q})(\mathbf{B}_d\dot{\mathbf{x}}_e + \mathbf{K}_d\mathbf{x}_e) \tag{5.23}$$

If $\dot{\mathbf{q}}$ is very small, which is typically the case in fine manipulation, we have,

$$\tau_m = \mathbf{H}(\mathbf{q}, \dot{\mathbf{q}}) - \mathbf{J}^T(\mathbf{q})(\mathbf{B}_d\dot{\mathbf{x}}_e + \mathbf{K}_d\mathbf{x}_e) \tag{5.24}$$

Equation (5.24) represents a relatively less complex control law that is useful in fine manipulation.

5.3.3 Separation of gross motion and fine manipulation

The main purpose of separating the robot task into various subtasks of different dynamic characteristics would be to take advantage of different control algorithms that are suitable

for specific situations, thereby not having to use one complex control strategy for the overall task. Position control is simple, reliable, and less computationally intensive. It is suitable for gross motion at high speeds and requiring moderate levels of accuracy. However it can make the robot very stiff under constraints, and hence not particularly suitable for contact manipulation. Force control is at the other end of the spectrum for contact manipulation, and is suitable when the end-effector is operating against a hard constraint; for example in grinding and polishing of hard surface. Impedance control incorporates advantages of both motion control and force control, and is particularly suitable for tasks that incorporate somewhat softer constraints; for example, cutting or processing of biological material. However, it generally imposes considerable computational overheads into the controller, and can be quite difficult to implement. As we see from equation (5.24), for low speeds, which is usually the case in fine manipulation, the impedance control algorithm can be simplified. Since it is clear that a single control algorithm is not suitable for both gross motion and fine manipulation, a multi-mode (dual-mode) control scheme is designed in the present work.

There has been some discussion of the separation of gross motion and fine manipulation [15]. In a non-redundant robot manipulator, all the joints should be controlled, irrespective of whether it executes a gross motion or a fine manipulation, to achieve an arbitrary position and orientation in Cartesian space while interactive with the process interface through suitable force characteristics. The separation of the mode of operation (control) here is based on task requirements, and it is a knowledge-based approach. Switching between operating modes can be explicitly specified in the task specifications, or implicitly given by the task control commands. For example, a point to point motion in free space would employ a motion control algorithm, while an interactive process such as material cutting or parts assembly would adopt impedance control. The switching between the control modes is considered safe in a static state, if both the control modes are stable, because there are no dynamic interactions between

the two control modes. The control signal is continuous at the switch point if,

$$\mathbf{q}_d = \mathbf{q}, \dot{\mathbf{q}}_d = \dot{\mathbf{q}} = \mathbf{0} \text{ and } \ddot{\mathbf{q}}_d = \ddot{\mathbf{q}} = \mathbf{0} \tag{5.25}$$

Then, the joint torque τ would be continuous during switching between the motion control and impedance control, and is given by,

$$\tau_{motion} = \tau_{impedance} = \mathbf{H}(\mathbf{q}, \mathbf{0}) = \mathbf{G}(\mathbf{q}). \tag{5.26}$$

It should be cautioned, however, that under substantial dynamic interactions between the robot and its process environment, such a continuous transition is not guaranteed if the switching is not quasi-static. In particular, it is desirable to bring the manipulator to a rest before initiating a material processing task under impedance control.

5.4 Path Planning and Interpolation

5.4.1 Path planning

For robotic applications such as automated fish cutting, it is not always feasible to generate the cutting trajectory either in advance or off-line, because a change of trajectory may be required due to an unexpected environmental change (e.g., encounter of a bone, deformation of the material) or unknown disturbance (e.g., a shift in the object position). In such situations, the motion commands should be generated online according to the real-time environmental conditions, as determined through sensing or other input. Online trajectory planning would also avoid problems related to computer memory, which might occur if all the joint commands for the entire path were to be determined in advance and stored in the memory. In order to save the computational time that is needed for trajectory generation in the present control system, the motion commands are generated at longer sampling periods in an upper level of the hierarchy, and are then interpolated in the joint coordinates at the level of joint servo control.

Planning of trajectories in the joint space is rather simple; specifically, the longest time period of joint motion is computed, assuming the joint operates at the maximum joint speed, and then all six joints are planned using this time. The planned motion should be synchronous for all joints. Typically, a trapezoidal velocity profile is used in this procedure.

Trajectory planning in the Cartesian space is much more complicated, particularly when continuous path motions, not point-to-point motions, are required for the associated robotic task. Online generation of motion commands for the manipulator trajectory is achieved as follows: first an appropriate tool frame (tool tip, or end-effector) motion of the robot manipulator is generated in some manner with respect to the world coordinate frame; for example, by using trapezoidal velocity profiles. Since the planning of position and orientation can be considered separately, due to kinematic decoupling brought about by the spherical wrist, the maximum linear and angular accelerations are used, and so the planning of position and orientation could result in two different time instants. Hence, the longest time is selected for synchronous planning.

The motion of the tool frame is generated in two steps: (1) pre-calculation of trajectory parameters such as time instants of the trapezoidal velocity profiles, and acceleration components, and (2) online path generation at each sampling time.

The motion commands (position and orientation) of the tool tip are converted into those of the wrist center, and then into those with respect to the robot base frame instead of the world coordinate frame. Finally they are converted into joint coordinate commands using inverse kinematics algorithm and to the joint velocity commands using the inverse of the Jacobian, J^{-1}. In the neighborhood of a kinematic singularity and at the singular configuration itself, the damped least-squares method of Jacobian inverse (or generalized inverse) is adopted instead of the classical Jacobian inverse [16]. In this manner, the trajectory can be planned through the wrist singular configuration. The joint motion commands should be further interpolated for each sampling time at the joint servo level.

The effectiveness of trajectory planning depends directly on factors such as the inverse kinematics algorithm that is used, singularity problems, computational load, and trajectory generation in the Cartesian space. Closed-form solutions for the inverse kinematics of PUMA robot have been developed by many researchers. Paul and Zhang's solution [75], which is given in Chapter 2, is used in the current research. In addition to finding solutions of inverse kinematics, it is also important to select an appropriate solution among several available solutions, especially among flip and no-flip solutions, at consecutive sampling time instants of a robotic task. Otherwise, performance instabilities would result as the controller attempts large jumps of joint motion over consecutive sampling times.

Whenever possible, the singular configurations of a robot should be avoided through trajectory planning. A PUMA type manipulator has three kinds of kinematic singularities; shoulder singularity, elbow singularity and wrist singularity [59]. The singular configurations of a PUMA 560 manipulator are determined by the equations [59],

$$a_2C_2 + a_3C_3 + d_4S_{23} = 0 \tag{5.27}$$

$$a_3S_3 - d_4C_3 = 0 \tag{5.28}$$

$$S_5 = 0 \tag{5.29}$$

where a_2, a_3, d_4 are the parameters of the robot as explained in Chapter 2. The manipulator home position and the parameters are illustrated in Figure 2.5. The symbols C_i and S_i adopt the same convention that was introduced in Chapter 2; specifically, they denote $\cos(\theta_i)$ and $\sin(\theta_i)$, respectively, where θ_i is the angular position of the ith joint of the robot.

Equation (5.27) corresponds to shoulder singularities of the PUMA manipulator. This singularity arises when the x_1 coordinate of the wrist center o_4 is zero no matter what the trunk rotation θ_1 is. In these configurations, the velocity component in the z_1 direction cannot be generated, and it follows that z_1 is the degenerate direction in these configurations. If, in these

configurations, the offset d_3 equals zero, then the wrist center will intersect the axis of the trunk rotation z_0. In this singular configuration, the inverse kinematics will have infinitely many solutions, as is well known.

Equation (5.28) represents the elbow singularity. This singularity occurs when the wrist center is extended (but not fully extended) or retracted (but not fully retracted). The exact locations of these singularities are the configurations with $\theta_3 = 87.308$ degrees or -92.692 degrees, for the following parameter values: forearm offset $d_4 = 0.4318$m and link length $a_3 = 0.0203$m. However, the retracted configuration does not occur within the workspace due to the motion limitation of joint 3. The degenerate direction of these singularities is that of x_1.

Equation (5.29) corresponds to the well-known wrist singularity. This singularity arises whenever the joint axes z_3 and z_5 are collinear. In this singularity, not only the mobility is reduced, but also the number of solutions to the inverse kinematics becomes infinite.

We can avoid the shoulder singularity and elbow singularity in the trajectory planning for a specific manipulator task by identifying these singular configurations in advance. However, we have to live with the wrist singularity since it can occur anywhere inside the workspace. If the motion is planned across the wrist singularity ($\theta_5 = 0$), the selection between the two solutions of θ_4 is important because there would be an angle jump (about π rad) in θ_4 if a wrong selection is made. In the trajectory planning process, we use the following simple logic to choose a continuous solution for θ_4 based on its previous value :

$$\text{If } |\theta_4(t_{i-1}) - \theta_4(t_i)| \approx \pi, \text{ then select the other solution.}$$

5.4.2 Interpolation

The set points are interpolated further at servo control frequency. A cubic spline function is used to guarantee continuous position and velocity. Given the two set points at time j and

$(j+1)$, we have,

$$f_j = q_d(jT_x) \tag{5.30}$$

$$v_j = \dot{q}_d(jT_x) \tag{5.31}$$

$$f_{j+1} = q_d((j+1)T_x) \tag{5.32}$$

$$v_{j+1} = \dot{q}_d((j+1)T_x) \tag{5.33}$$

Here, T_x denotes the path planning period, while f and v are position q and velocity $\dot{q}$, respectively.

The problem now is to find a cubic spline function $P_j(t)$, where $jT_x \leq t \leq (j+1)T_x$, that satisfies the boundary conditions:

$$P_j(jT_x) = f_j \tag{5.34}$$

$$\dot{P}_j(jT_x) = v_j \tag{5.35}$$

$$P_{j+1}((j+1)T_x) = f_{j+1} \tag{5.36}$$

$$\dot{P}_j((j+1)T_x) = v_{j+1} \tag{5.37}$$

Let

$$P_j(t) = a_{j0} + a_{j1}(t-t_j) + a_{j2}(t-t_j)^2 + a_{j3}(t-t_j)^3, \tag{5.38}$$

where $t_j = jT_x$, and a_{jn} are the parameters of the cubic spline. We have,

$$f_j = P_j(t_j) = a_{j0} \tag{5.39}$$

$$v_j = \dot{P}_j(t_j) = a_{j1} \tag{5.40}$$

$$f_{j+1} = P_j((i+1)T_x) = a_{j0} + a_{j1}T_x + a_{j2}T_x^2 + a_{j3}T_x^3 \tag{5.41}$$

$$v_{j+1} = \dot{P}_j((j+1)T_x) = a_{j1} + 2a_{j2}T_x + 3a_{3j}T_x^2 \tag{5.42}$$

By solving these equations, we obtain,

$$a_{j0} = f_j \tag{5.43}$$

$$a_{j1} = v_j \tag{5.44}$$

$$a_{j2} = \frac{3}{T_x^2}(f_{j+1} - f_j) - \frac{1}{T_x}(v_{j+1} + 2v_j) \tag{5.45}$$

$$a_{j3} = \frac{2}{T_x^3}(f_j - f_{j+1}) + \frac{1}{T_x^2}(v_{j+1} + v_j) \tag{5.46}$$

The points between time j and time $(j+1)$ can be computed by,

$$f_t = P_j(t) = a_{j0} + a_{j1}(t - t_j) + a_{j2}(t - t_j)^2 + a_{j3}(t - t_j)^3 \tag{5.47}$$

$$v_t = \dot{P}_j(t) = a_{j1} + 2a_{j2}(t - t_j) + 3a_{j3}(t - t_j)^2 \tag{5.48}$$

using the parameters in equations (5.43) through (5.46), where T_x, f_j, f_{j+1}, v_j and v_{j+1} are known variables.

Figures 5.2 and 5.3 show an example trajectory with linear interpolation and cubic interpolation. It is clear that, in cubic interpolation, the path is smoother and the velocity is continuous.

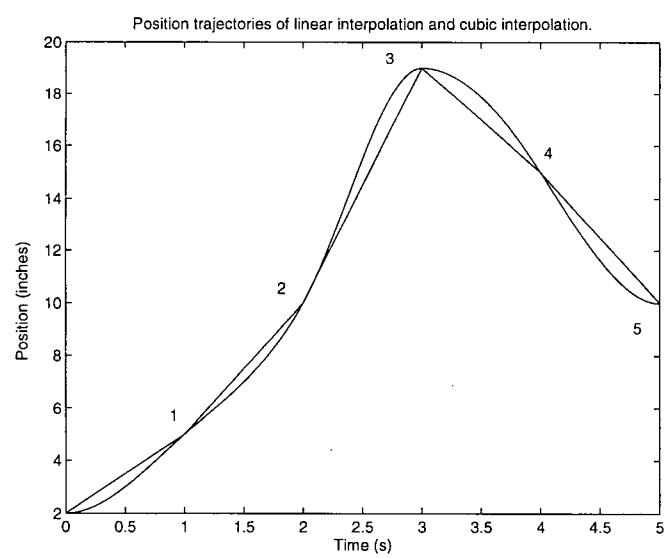


Figure 5.2: Interpolation of position between specified points at 1, 2, 3, 4, 5 seconds.

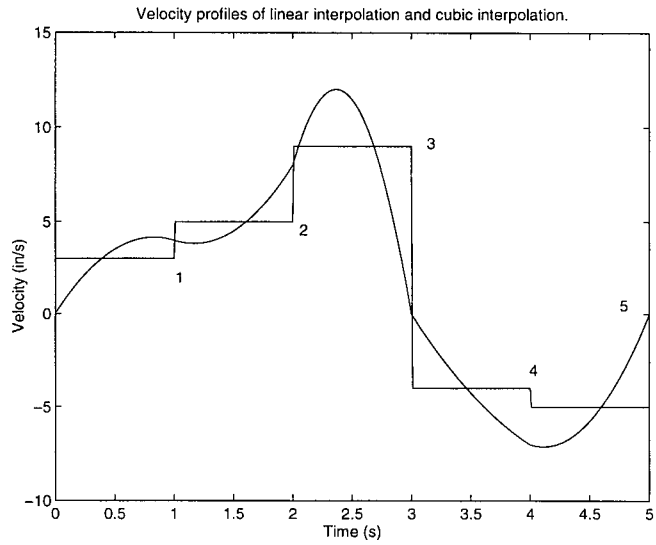


Figure 5.3: Interpolation of velocity between given points at 1, 2, 3, 4, 5 seconds.

5.5 Information Preprocessing and Intelligent Task Control

For the purpose of task control that is based on mechanical impedance, it is required to interpret the "measured" impedance signals online. Since mechanical impedance is task dependent, associated interpretation also will depend on the specific application. Furthermore, the interpretation is generally subjective, based not only on experimental data and theory but also on expert opinion concerning the specific application and process requirements. In view of this, an approach that utilizes fuzzy logic [34] would be suitable in interpreting impedance measurements.

Two approaches are presented now. One approach is model-based, which uses a subjective interpretation of the estimated impedance parameters. The other uses measured impedance profiles themselves, which are subjectively interpreted according to the shape and magnitude of the profile, with regard to available knowledge and experimental data. The first approach is outlined below, and the second approach is given subsequently.

5.5.1 Method based on impedance parameters

For a specific task, a knowledge base may be constructed in order to match linguistic representation (expert opinion) of object properties to the estimated impedance parameters: inertial m_e, dissipative b_e, and elastic k_e. Fuzzy logic can play an important role here. Usually each parameter represents a property of the task interface of the end-effector and the object. It can be fuzzified to describe the property in linguistic terms, as interpreted by an expert, with proper fuzzy membership. In meat cutting, for example, dissipative (resistive) impedance is known to be dominant. As a result, a relatively finer (high) fuzzy resolution in dissipative impedance would be desirable. Inertial and stiffness parameters will provide additional information to help identify the object properties. In material cutting, the stiffness effects would be more important than the inertial effects.

As an example in this approach, the fuzzy sets may be assigned the resolution given in Table 5.1.

Table 5.1: Fuzzy resolutions of impedance parameters.

Parameter	Fuzzy set
b_e	very high (vh), high (hi), medium (md), low (lo)
m_e	high (hi), low (lo)
k_e	high (hi), medium (md), low (lo)

Here, the parameter b_e represents the dissipative resistance of cutting, which may be related to non-elastic deformability and disintegratability of an object such as meat. A bone will have a very high dissipation level, firm meat will have a medium dissipation, and fat and soft (or aged) meat may have a low dissipation. The parameter k_e represents the elastic nature of the material; for example, whether the meat deforms as it is cut and then regains its original shape when the force is released. Common sense tells that skin is more elastic than firm meat, but fat

and soft or aged meat typically have even lower stiffness properties. The inertial parameter m_e may not be very significant in local processing tasks where fine manipulation dominates over gross dynamics. This is, for example, the case in meat cutting. But the inertial parameter may be useful in determining problems of grasping and fixturing (e.g., a loose holding mechanism). These three impedance parameters can be represented as fuzzy quantities [46] which can be represented by suitable membership functions. For example,

$$b_{fuzzy} = \{vh(0.0), hi(0.0), md(0.4), lo(0.6)\},$$

where b_{fuzzy} is the fuzzified form of b_e.

A rule base of identifying different regions in a carcass, for mechanical processing, is given in Figure 5.4. Here, "$\rightarrow$" denotes "membership" of a fuzzy set; "$\longrightarrow$" denotes "implication"; "$\cap$" denotes "and"; b, k, and m denote fuzzified impedance parameters, and the conclusion set is crisp. For example, the second rule in Figure 5.4 states that "*if b is* **medium** *and k is* **medium**, *then this region is* **meat**, *with the rule having a level of belief of 0.8 (80%)*". With a rule base of this type and an identified context $\{b,\ m,\ k\}$, a fuzzy decision making procedure [46] can be applied to reason for object properties during mechanical processing.

$b \rightarrow vh \longrightarrow region \rightarrow bone$ **with** 0.9;
$b \rightarrow md \cap k \rightarrow md \longrightarrow region \rightarrow meat$ **with** 0.8;
$b \rightarrow lo \cap k \rightarrow md \longrightarrow region \rightarrow meat$ **with** 0.9;
$b \rightarrow md \cap k \rightarrow lo \longrightarrow region \rightarrow fat$ **with** 0.75;
$\cdots, \cdots$

Figure 5.4: Material interpretation rule base

5.5.2 Method based on impedance profile interpretation

In the second approach, the impedance curves that are computed directly in the time domain, are interpreted using expert opinion and available experimental data. First, a set of experiments

is conducted where an adequate number of samples of various material (e.g., bone, skin, meat of various texture, and fat) that would be encountered in a task, are processed and impedance data are collected. Using this data, membership functions are established for the impedance characteristics of each material class. A rule base is also developed, through expert opinion and experience, for identifying various regions of the processing task. For example, the rule base may only consider various material regions and transition regions. In the case of a carcass of meat, we may consider a rule base of the form

If BONE is VL and MEAT is VH and FAT is VL then Region is Meat.

If BONE is MD and MEAT is MD and FAT is VL then Region is Trans.

etc.

Then, in applying this knowledge base, $BONE$ denotes a particular impedance reading that is fuzzified with respect to the membership function of bone, $MEAT$ represents the same reading but fuzzified with respect to the membership function of meat, and so on. The compositional rule of inference [31, 38] is applied in the usual manner, then, to obtain the inferences. Note that the first rule above gives a clear decision on meat region, whereas the second rule identifies a transition region between bone and meat. In this manner, an impedance profile is interpreted and the process can be controlled on that basis.

In task control, criteria based on material properties may be used in the form,

$$IF\ E1\ AND\ E2\ AND\ \ldots\ AND\ En\ THEN\ A$$

where, $E1$, $E2$, ..., En are conditional expressions. For example, $R{\rightarrow}meat > 0.8$ becomes true if the membership grade of the fuzzy membership function $R{\rightarrow}meat$, is higher than 0.8. In the given rule, A is a control action that is task specific.

The next chapter in its entirety will be devoted to intelligent task control and related novel design concepts that have been developed in the present research.

5.6 Summary

A five layer hierarchical system was designed in this chapter. It consists of a hardware layer, sensing and actuation layer, servo control layer, behaviour generation and monitoring layer, and a high-level task control layer.

Two control modes were designed for servo control system; namely, position control for gross motion and impedance control for fine manipulation. Computed torque method with gravity and friction compensation was employed for position control. The control mode was selectable through task control.

An online trajectory generation algorithm was proposed for trajectory planning of both position and orientation of the end-effector of a robot. The online trajectory generation algorithm that was developed, would allow immediate planning of a different trajectory whenever a change in the environmental condition is identified. Both rapid joint motion and linear Cartesian-space motion can be planned in this manner. A cubic spline method was applied to servo-level interpolation.

Information from the bottom level of the hierarchical system needs to be further processed and interpreted using fuzzy logic for supervisory control and intelligent task control at upper levels. Characteristics of the process are extracted in this supervisory control layer. Two methods were presented for process interpretation using impedance information.

Chapter 6

Intelligent Task Control

Mechanical impedance at the process interface is an important piece of information for task planning in robotic manipulation or processing of inhomogeneous and flexible materials. It contains useful properties of a mechanical process and of the object that is being processed. These properties are useful in identifying the processing regions in task planning. Then, while processing the object, the impedance can be sensed online and used in properly controlling the task. Knowledge-based decision making is involved here. This chapter will address high-level intelligent task control using online impedance estimation and interpretation for a processing task where robot motion is planned not only based on motion information but also on impedance characteristics at the interface of the processed object and the end-effector. High-level robot tasks will be represented in a descriptive manner rather than specifying a working procedure for the robot manipulator. Knowledge representation and reasoning in the task control level will be discussed. A real-time task language (RTTL) will be presented as well.

Section 6.1 will introduce the concept of intelligent task control. Section 6.2 will discuss the problems in traditional task control and languages. This discussion will lead to the introduction of intelligent task control in Section 6.3. Section 6.4 will address the knowledge representation and reasoning in intelligent task control. Section 6.5 will present the real-time task language (RTTL) which is developed for flexible manipulation in the present research. Section 6.6 will conclude the chapter.

6.1 Introduction

Traditionally, robotic tasks are designed to meet specifications of working procedures, based on position control. There is no feedback of operation information to the task planning level. However, there are many jobs that cannot be pre-designed by motion control only. These jobs require online feedback of operation information for real-time task planning. Mechanical impedance [26] at the processing interface between the object and the end-effector, in dynamic interaction with the restraining environment, is particularly important in processing inhomogeneous and flexible objects consisting of natural material. For example, in fish processing, pre-designed paths may no longer be valid during the actual operation due to environmental uncertainties and material deformation. In this case, tasks should be controlled by using complementary online information as well.

Let us revisit the example task in Section 1.7, Figure 1.5, which is repeated in Figure 6.1. The figure schematically shows a task of slicing to produce boneless steaks of salmon. In this task, first the fish should be sliced into portions of required size by cutting it through to its backbone; specifically, from a_1 to c_1, a_2 to c_2, ... , and a_n to c_n. Then the sliced steaks should be removed from the backbone by the lateral cutting operations: free end to c_n, c_n to c_{n-1}, c_{n-1} to c_{n-2}, ..., and c_1 to c_0. In both sets of operations, the robot controller needs to identify the material type such as meat, skin, and bone. Position control alone is not sufficient to perform the task. Here, cutting impedance may be determined online to sense the various material regions. Some basic problems of this type of robotic operations are identified below, and some corrective procedures are indicated:

- Due to the nature of the material — soft, inhomogeneous, etc. — tasks that are designed for exact position control might not be feasible. The material could be easily deformed during processing. Also, the object may shift against the environmental restraints (holding device). Hence, the pre-designed paths of end-effector movement may not be valid. In

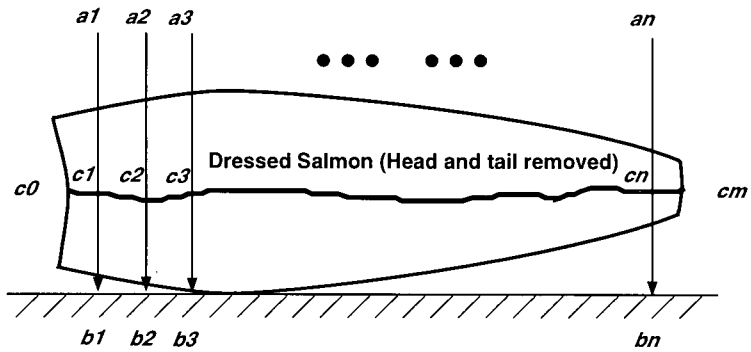


Figure 6.1: Illustration of a robotic salmon slicing task.

fine manipulation against hard constraints (e.g., bone, holding mechanisms) a slight error in motion can cause large and damaging forces. Physical characteristics of the robot and the end-effector (e.g., friction and backlash) may not be properly and quickly sensed and compensated for through motion sensing and control alone.

- The task can be decomposed into some key operations such as slicing and separation. The motion between key operations (e.g., retraction) may not be crucial. Tasks can be designed to describe the key operations only, allowing the robot's control system to determine the motion between key operations.

- Information feedback is essential for task execution, since impedance is sensed and used to detect the backbone and stop the motion accordingly. Furthermore, the position where the cutter reaches a primary bone should be recorded for subsequent path planning.

- Tasks should have some logical priorities; for example, the steak separation task can be performed only after the slicing is completed.

Clearly, traditional task representations are not suitable in the robotic processing of inhomogeneous flexible materials. Open-loop task control cannot meet task requirements in online processing of such materials; for example, in food processing applications. This chapter will

develop, design, and implement an intelligent task control system based on both position and impedance information, and using fuzzy logic and AI technologies for knowledge representation and knowledge-based decision making. The implementation will be done for an industrial machining environment.

6.2 Traditional Task Representation and Control

6.2.1 Task representation in CNC machining

Computer-numerical-control (CNC) machining is the utilization of computer technology for instructing a machine tool as to how and according to what dimensional specifications a machined part should be produced. A majority of CNC controllers follow the G-code programming format standards. Some light-duty machines may support the HP graphics language (HPGL) as well.

A G-code program is a list of motion and control commands that are needed to produce the part to be machined. Some examples of G code commands are given below:

- $G00$ – *Rapid traverse positioning.* This G-code instructs the CNC controller to move all the specified axes as fast as possible to the destination. The command $G00$ should be followed by the coordinates of the destination location in the format:
  $G00\ X1.0\ Y1.0\ Z-1.0.$
  This example would move the end-effector along X, Y, and Z directions to the specified location (1.0, 1.0, -1.0). Most controllers just move each axis individually at the maximum feedrate, independent of the other axes.

- $G01$ – *Linear motion at feedrate.* This G-code instructs the CNC controller to move all specified axes simultaneously at a designated feedrate so that the motion along each axis is completed at the same time. The command $G01$ should be followed by the destination

location and the desired feedrate, in the format:

$G01\ X1.0\ Y1.0\ Z-1.0\ F10.$

In this example, the X, Y, and Z degrees of freedom are moved to the specified location at the speed specified by the F word linearly.

- *$G02$ and $G03$ – circular motion at feedrate in the clockwise ($G02$) or counter-clockwise ($G03$) direction.* The commands $G02$ and $G03$ instruct the CNC controller to move the degrees of freedom simultaneously so as to perform an interpolated circular motion at a designated feedrate so that the motion along each axis is completed simultaneously. For example, the G-code

  $G02\ X1.0\ Y1.0\ I1.0\ J0.0\ F10$

  would move the X, Y degrees of freedom to the specified location along an arc, whose center has X, Y offsets that are specified by the I and J words from the current position, in the clockwise direction, at the speed specified by the F word.

The graphics language HPGL specifies the motion of a pen (of a plotter) or a cutter (e.g., in engraving machines). The basic commands are:

- PD – moves the pen down.
- PU – moves the pen up.
- PA – moves the pen to an absolute position as specified by the parameters given after the command PA. For example, PA 100, 50 would move the pen/end-effector to the position $X = 100$ and $Y = 50$.
- PR – moves the pen to an X, Y position relative to the current position by the relative coordinates as specified by the parameters given after PR. For example PR 50, -10 would move the pen/end-effector 50 units along the +X direction and 10 units along the -Y direction.

- PQ – queries the current position of the pen/end-effector.

Note that in these conventional approaches, tasks are represented, whether in G-code or in HPGL, by pre-defined motions. All parameters are constant, and no online information is used. Clearly, this type of representation is unable to specify a task that cannot be completely defined in commands of exact position control. In particular, if the task requires impedance information measured online in control of its motion paths, the position specifications alone will not be adequate.

6.2.2 Advanced robot control language

Robot manipulators may be considered a more flexible and general category of CNC machines which usually have more than three degrees of freedom (DOF), and can have both prismatic (Cartesian) and revolute (polar) joints. Existing robot languages also adopt the open-loop method which is based on exact position control.

Advanced Control Language (ACL) [1] is a robot task language used in the Scorbot robot [89] controller. It is much more complicated than a general CNC language, and is able to handle sophisticated tasks. The main improvements in this robot control language in comparison to a CNC language are the implementation of variables and program flow control. Some basic commands are given below:

- $MOVE\ pos$ – moves the machine to a position that is specified by pos;
- $HERE\ pos$ – gets the current position and stores it in the variable pos;
- $SET\ var1 = var2$ – assigns the value $var2$ to the variable $var1$;

However, since there is no flexibility in the motion commands themselves, the program flow control does not allow accurate task planning of incompletely known or "qualitative" position

control where impedance information can play a major role. To date, robot languages designed for general purpose task planning are based on position information and associated motion. In a limited sense, force or impedance control may be designed for some special machines to carry out special tasks, without the benefit of a general task language. To represent force and impedance based flexible motion in general, an intelligent task control system with flexible motion commands should be available, which is not the case in the present generation of robot control systems.

6.3 Sensor-Based Intelligent Task Control

Definition: *Intelligent task control pertains to selection of the most appropriate task in a task space at each control time instant according to the current operating conditions of the system, and to properly execute that task in the presence of non-precise and qualitative task representations and unknown dynamic interactions, where knowledge-based reasoning would be important.* There are three basic requirements here:

1. It must be real-time and the solution must be made online.

2. Feedback is necessary for task control. A control decision has to be made based on the real-time performance of the system.

3. Artificial intelligence and knowledge-based reasoning would be required to achieve the "best" solution, in the presence of uncertainties and qualitative task specifications.

Traditional task control adopts an open loop, pre-programmed method. This is not adequate in flexible manipulation where the processing procedure may not be precise, and uncertainties and qualitative requirements may be involved in the task description. A good example is the salmon-slicing task, where the backbone position is uncertain and the "quality" of a steak cannot be specified in terms of precise position coordinates. The uncertainties may be reduced

by online feedback of information. The backbone position in our example can be determined from impedance information, as the cutter reaches the bone. A knowledge-based decision making procedure may be needed here to determine what parts of the fish body should be excluded in portioning a steak.

The most important feature of intelligent task control is that it is able to handle uncertainties and the "qualitative" nature of a task description. This feature affords great flexibility in task execution. If this feature is not available, the tasks can be only programmed in a fixed procedure of precise operations, which may not be physically realizable. Another feature of intelligent task control is that it performs tasks in an intelligent manner based on a priori knowledge of operation and also using online feedback of information to upper levels (e.g., task level) of the control system.

6.3.1 Architecture

Figure 6.2 shows a suitable architecture for an intelligent task control system. In this system, there are four basic segments: task base, database, knowledge base (rule base) and inference engine which will be referred to as the task engine in the reminder of the thesis.

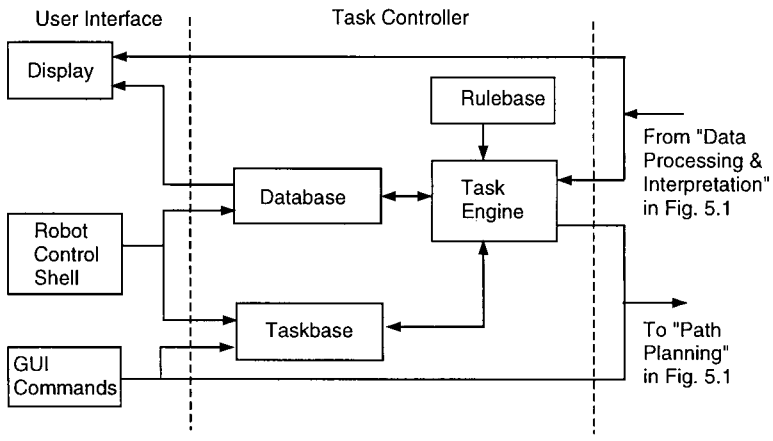


Figure 6.2: Architecture of an intelligent task control system.

The task engine searches the task base and makes decisions (i.e., selects the most suitable

job) according to the knowledge base and information base. The knowledge base and the database should be updated using feedback information and new knowledge from the user and other external sources. Tasks should be represented in a suitable way to facilitate the decision-making process.

6.3.2 Task representation in intelligent task control

Some principles and concepts are proposed here to improve and enhance the traditional task planning that uses position only.

Descriptive task representation: A traditional task program has to precisely specify all the details of motion of a robot manipulator. It should provide a complete working procedure. However, in practice, only some key operations of a task need to be specified in detail, and it may not be necessary to specify, for example, the paths and velocities of intermediate motions. Also, it may not be necessary to give the commands of a task program in the order that they are executed. Instead, a task should be written in the order that it is described. An intelligent task controller and planner of a robot is responsible for selecting the most suitable command for execution in real-time according to the specific conditions of operation.

In the example of salmon slicing, only the slicing and steak separation operations need to be specified. Other motions can be carried out by the robot system automatically, by following some rules, corresponding to product requirements, and using simple point-to-point motion.

Motion restrictions and task set: There are two supplements to the descriptive task representation: motion restrictions and task set. Motion restrictions should be given in the task program; thus the default motions can be planned to avoid restricted areas or obstacles. The task set is designed to enforce the execution order of some consequent motion commands. This enables the use of traditional task representation in intelligent task control. The first motion command in a task set is evaluated in an entire task space for execution of the task set. It is

user's responsibility to ensure that the execution of the task set is smooth.

Online information feedback: Many tasks require not only position control but also, force and impedance feedback. Normally, there are uncertainties in task description, either in position or in force and impedance. Fortunately, a large number of position uncertainties in task representation can be specified in terms of impedance and force and, furthermore, some uncertainties in force and impedance can be specified in terms of position. The traditional methods of robot operation are based on exact position control, but cannot explicitly accommodate situations where processing may be properly specified in terms of the requirements based on mechanical impedance of processing, and where the position requirements are somewhat qualitative and uncertain. Online information of both position and impedance would be necessary for a robot to properly perform tasks in such situations. Variables are required to record online information for use in subsequent motion planning.

Priority of tasks: As the tasks are not presented in their execution order, priorities are very important for a task description. For example, an emergency stop should have the highest priority once it is added to the task base. The task controller should select the most appropriate command in the task space using the priority specifications together with other appropriate selection criteria.

6.4 Knowledge Representation and Reasoning

Formal schemes of knowledge representation and reasoning are given in this section. Incorporating these schemes, a generalized method of system design is presented, which will help in implementing the system.

6.4.1 Knowledge representation

Knowledge representation is important in intelligent task control. The task should be represented in a generalized format, which will facilitate the task engine to efficiently search in the task space. Note that search is a basic function of intelligent task control. A general way to represent task commands may be given as,

$$task(cmd, prio, argc, argv); \tag{6.1}$$

where cmd is the name of a command, $prio$ is the priority level of the command; $argc$ is the number of arguments following the command; and $argv$ is a vector pointing to arguments. In this manner, commands with different numbers of arguments can be represented in the same format. For example, consider the command of priority level 2,

$$movei\ A, B, C, E$$

where $movei$ is the command name, and A, B, C and E are arguments. This command can be represented as

$$task(movei, 2, 4, [A, B, C, E]);$$

The advantage of this latter representation is that it may correspond to a variety of commands and will make the system expandable. The arguments A, B, C, and E may also be specified in different formats. Data representation is made general so as to accommodate different types of information, in the format

$$var(name, type, data); \tag{6.2}$$

where $name$ is the variable name, for example, A, B, C, E; and $type$ represents the data type of the variable. Here A, B, and C specify the type of position, and E is a conditional expression. Also, $data$ is the data structure of the variable.

6.4.2 Task engine

Unlike in traditional task control, where the task controller simply translates the task program sequentially into motion commands for the machine, the task engine will search for the most appropriate job and explain it in terms of motion commands including flexible motion commands to the machine. A formal representation of the reasoning process of a task engine is illustrated below:

$$
\begin{array}{l}
task \longleftarrow \\
\quad bestjob(J), \\
\quad\quad !, \\
\quad\quad\quad execute(J), \\
\quad\quad\quad\quad remove(J), \\
\quad\quad\quad\quad\quad task. \\
task \longleftarrow \\
\quad wait(t), \\
\quad\quad task.
\end{array}
$$

Here the goal $task$ has two clauses. The first clause is for successful execution. Whenever a predicate fails in this clause, the task engine moves to the second clause which will make the task engine sleep for a time interval t.

The first predicate in $task$ is $bestjob()$, where the most appropriate job J is selected. If this fails, which means there are no jobs in the task queue that are executable, the second clause of $task$ will make the system sleep for time t. A cut ! follows $bestjob()$, which prevents any backtracing to this point; because J is the best choice, there is no need to trace back to $bestjob()$. Subsequently, the job J is executed by $execute()$. If this step fails, which means the system does not accept this command due to the system status, the second $task$ clause is called to wait

for a period of time t, during which the system may change its status. An example would be to make the system wait for finishing the trajectory path in the set point buffer, and to make space for accepting new paths. If the command is executed successfully, it should be removed from the task base. This procedure will continue as long as the task control is running.

A simple job search may be based on the following criteria:

- priority of a command;
- executability of a command — all variables in the command should be assigned;
- the order of the commands in the task program.

In case of a task set that enforces the execution order, the whole set is evaluated by the first motion command in the set. More complicated and more intelligent searching algorithms can be employed according to task needs.

The task controller shown in Figure 6.2 consists of a task engine, a data base, and a task base. The task engine searches the task base and selects the most appropriate command for execution. If the execution succeeds, the command is removed from the task base; otherwise, it is retained in the task-base. The next time, the task engine should search for the best command again, because the system status may have changed, and some higher priority jobs may be ready to run if the variables involved in these jobs are assigned. Operation data are stored in the database which is manipulated by the task engine to retrieve and store information. The rule-base provides knowledge of how to identify the material characteristics and other relevant information of the process using impedance and other process information.

6.5 A Real-Time Task Language (RTTL)

In the present research, a new real-time task language (RTTL) is designed following the principles discussed in Section 6.3.2, on task representation in intelligent task control. Some

important commands of this language will be discussed here.

A major advantage of the RTTL is that it provides a number of task commands in order to accommodate online information feedback. For example, two basic operations are defined here for using impedance in motion control: *movei* and *mover*.

The command *movei* A, B, C, E moves the robot end-effector from position A to the destination B or stops when impedance expression E becomes true, and records the corresponding stop point as C. Figure 6.3 illustrates this motion command, where the cutter moves from A towards B, and the impedance expression is

$$E = R{\rightarrow}bone > 0.7.$$

The impedance will increase during the cutting process starting from A as C is approached. Consequently, the membership value of the impedance will become closer and closer to the specified limit in the expression and finally the membership function of $R{\rightarrow}bone$ will be higher than 0.7. Then, the robot will be instructed to stop, and the corresponding stop point will be stored in variable C.

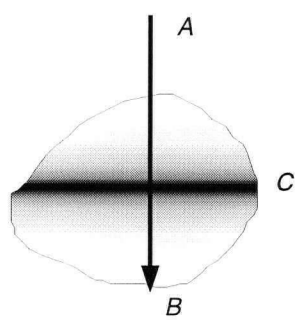


Figure 6.3: Illustration of the execution of the procedure *movei*.

In Figure 6.1, the slicing operation can be designed to move the cutter from a_1 to b_1, a_2 to b_2, ..., until a_n to b_n. Each slicing operation should stop at the back bone locations c_1, c_2, ..., c_n. These positions should be returned to the task controller for path planning of the procedure of steak separation.

The command *mover* A, B, O, D, E moves the robot end-effector from position A to the destination B, but retracts the cutter by a distance D along a specified orientation O (of decreasing impedance in fish processing) when the impedance condition E becomes true. This process is shown in Figure 6.4, where the motion is initially designed for movement from A to B. However, when the cutter hits the bone at point 1 and the condition E becomes true, the initial path is modified by retracting the cutter from position 1 to point 2 (through orientation O, and distance D), and the new path would be from 2 to B. Such modification of the path may happen again within the new path; for example, at point 3, and an appropriate adjustment should be made there as well.

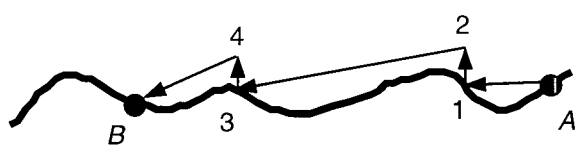


Figure 6.4: Illustration of the execution of the procedure *mover*.

In the fish slicing task, the capability of flexible motion through the use of impedance sensing and feedback will prevent the cutter from cutting into the backbone, thereby improving the efficiency of the process and the quality of the product. Note that *mover* may also be implemented using *movei* at a high level, but for improved efficiency, it has been designed as a basic operation.

Several other commands are defined below:

command cmd – commands the robot server to do the logic action specified by *cmd*; for example, *cmd* =AC_ON would instruct the controller to turn on the AC power of the robot system.

set $var1$=$var2$ – assigns $var2$ to $var1$. Here $var1$ and $var2$ should be of the same type. Variables are usually assigned prior to the motion commands. This means *set* should have a higher priority than the motion commands. Variable types should be identified by their names.

The name convention used in RTTL is explained in Chapter 8 (Refer to Table 8.1 for details).

query type, var – queries the current data of type *type*, and stores in variable *var*. The data type of *var* should be compatible with that of *type*. Here *type* could be the current *position*, or the set point *setpoint*, or the velocity *velocity*. This command is mostly used for information feedback.

display var – displays the value of the variable *var* according to its type. The variable type is encapsulated in the variable name.

wait – informs the path planner that no command should be accepted before the controller completes the previous paths. If the command that follows is a non-motion command, which means the path planner is not involved in executing this command, then another *wait* should be placed before the non-motion command to block it from being executed too early.

stopat exp – sets a stop condition for the data analyzer. When the condition of *exp* is satisfied, the data analyzer triggers a proxy informing the path planner to abandon all pending paths and to plan a decelerating path for stopping the robot. Here, *exp* may be set as impedance information.

movew var1, var, vel – moves the robot end-effector from a confirmed variable or constant *var1* to a destination specified by a confirmed variable or constant *var* at a cruising speed of *vel* in a world coordinates.

movej var1, var, vel – identical to *movew* except that *movej* is defined in joint coordinates, and *var1* and *var* are joint angles.

movetow var, vel – moves from the current position to the destination *var* at a maximum cruising speed of *vel*. This command is defined in world coordinates.

movetoj var, vel – identical to *movetow* except that it is defined in the joint space.

movetowr var, vel – moves the end-effector through a distance specified by *var* in the X, Y, Z directions. The maximum cruising speed is specified by *vel*.

movetojr var, vel – moves the robot joints through a distance/angle specified in *var*. The

maximum angular speed is given by *vel*.

movestop acc – stops the robot at a deceleration specified by *acc*. All pending paths in the set point buffer are abandoned.

delay var – delays the operation by the *var* number of task control periods.

save filename – saves operation data in a file named *filename*. The save status should be turned off by using *save off*.

store filename – stores all variables into a file named *filename* in a format as in *set* command, so that the file can be loaded next time by the command *load*.

The following example illustrates how the task of salmon slicing is implemented:

```
# Variable assignment
... ...
# Slicing
movei a1, b1, c1, e;
movei a2, b2, c2, e;
movei a3, b3, c3, e;
... ...
movei an, bn, cn, e;
# Separation
mover cm, cn, 0, d, e1;
mover cn, cn_1, 0, d, e1;
... ...
mover c2, c1, 0, d, e1;
mover c1, c0, 0, d, e1;
# End of job
```

Here, e and e_1 are conditional expressions for identifying the fish backbone using interpreted

impedance information. Also, $a_1, a_2, ..., a_n$; $b_1, b_2, ..., b_n$ and $c_1, c_2, ..., c_n$ are position variables, O is an orientation variable, and d is a constant. After completing the slicing operations, the variables $c_1, c_2, ..., c_n$ would be assigned and used in the steak separation process.

6.6 Summary

A new method for robotic task control was developed. General principles of task representation were discussed. Online information, particularly, the processing impedance was used in the new task representation. The robot controller was able to recognize different material regions and act in appropriate ways according to the task requirements. Such an approach is essential in processing inhomogeneous and flexible materials of natural objects such as fish and other food items.

A formal representation paradigm was developed for knowledge representation and reasoning in the intelligent task control system. A real-time task language (RTTL) was defined as a useful tool for intelligent task control. As an illustrative example, a salmon-slicing task was represented in the developed RTTL.

Chapter 7

ROACS: A Real-Time Open Architecture Control System

For the purpose of increased flexibility with respect to the implementation of user-developed control schemes; to particularly, impedance sensing and control, and adaptability to a variety of task requirements, a multi-mode, intelligent, hierarchical control system is developed in the present research. The hierarchical structure of the control system, as shown in Figure 5.1, is found to be different from the traditional (commercial) designs of a robot controller. In particular, the control system developed here allows feedback at all levels of the hierarchy and provide real-time intelligent decision-making. This implementation of a control system of this type is facilitated by a novel design of real-time open architecture control system (ROACS). This chapter will discuss the basic requirements and the architecture of a ROACS, and will develop a model for it. In this model, the hierarchical system can be implemented easily: different levels of functions can run at different bandwidths, and synchronization and communication between the function levels will be well modeled.

Organization of the present chapter is as follows: Section 7.1 will give a brief introduction to the importance of the ROACS model in flexible manipulation. Section 7.2 will discuss the conflicts that arise within the requirements of a real-time intelligent system. Section 7.3 will explore a client-server architecture that ensures open-architecture implementation, programmability, and synchronization. Section 7.4 will describe how the system will operate, and Section 7.5 will summarize the basic characteristics of ROACS.

7.1 Introduction

The functional hierarchy of a robot control system, as discussed in Chapter 5, could be realized in several ways. Limited by the computational power, the early implementations of hierarchical control system relied primarily on the hardware architecture. The functions of each level of the hierarchy were traditionally performed by different computational resources. For example the commercial controller of Puma 560 robot [83] has two levels of operation: at the lower level, six separate servo boards control each of the six joints of the robot; and at the upper level, a host computer performs path planning and communication with the six servo systems for trajectory commands. This two-level structure is shown in Figure 7.1. Note that the two layers are hardware specific.

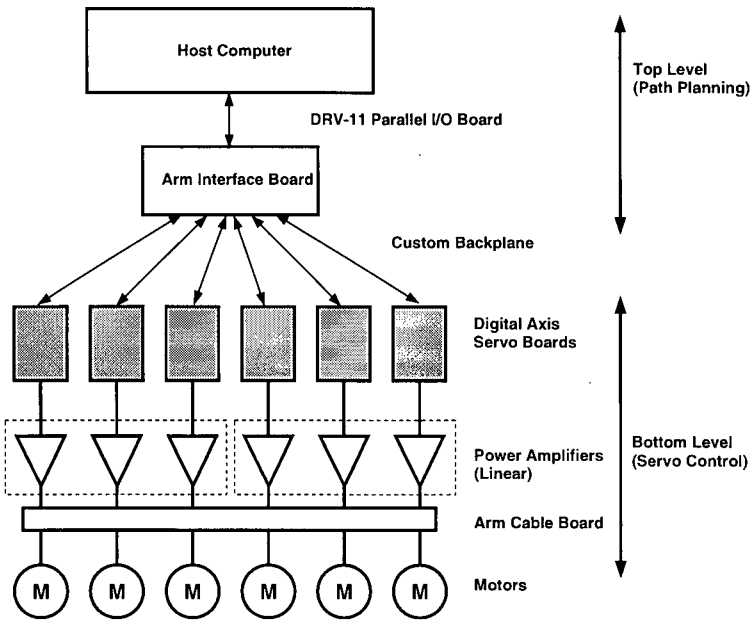


Figure 7.1: Illustration of a commercial Unimate controller.

Although this system is seemingly complicated in its hardware structure, the overall function is quite simple and is inadequate for advanced sensing and control requirements of flexible automation. The joint servo systems control the robot joints separately as individual systems.

The coupled dynamics of the joints (equation (4.8)) is totally ignored. Even the popular computed-torque method cannot be implemented using this system. The overall system does not have any knowledge-based reasoning and decision making capability. Even worse, the sensor information of the system is not open to the user; hence, a sensor-based intelligent method has no place in such a system.

Generally, robot control is limited by its computational complexity and strict real-time constraints. Performance of a robot controller is greatly determined by the specific architecture of the controller. Fulfilling of heavy computational needs, while satisfying high control bandwidth requirements, is a dilemma in real-time control. Hierarchical structures are used primarily to isolate and distribute the computational complexity (and associated functional complexity), in which dedicated digital signal processing (DSP) boards are usually employed for high-bandwidth direct control in the lower layers of the hierarchy. However, conventional, hierarchical structures have several drawbacks. For example, this type of control structure is fixed by hardware; communication between the host computer (high level) and the DSP boards, and among different DSP boards themselves, is quite limited in terms of information quantity and flexibility; control algorithms are fixed or limited by the capacity of the DSP boards; and the cost of a DSP board can be excessive. On the other hand, the practical need for open-architecture controllers is steadily increasing, in real-time control applications. This is particularly true in hierarchical intelligent control, for example, as utilized in robotic systems.

The dramatic increase of the computational power of computers, and the availability of fast local area networks have made it possible to use a single computer or a local area network to perform all the functions of a hierarchical system, such as the one developed in the present research. This essentially moves the complexity of the system from the hardware structure to the software structure. Therefore, flexibility will be significantly increased, since it would be much easier to modify the system so as to adapt for different applications, or variability within the same application, which is a critical requirement in soft automation [29].

7.2 Real-Time Intelligent System

Robot control is real-time in nature and for complex and flexible tasks, knowledge-based decision making and associated "intelligent system" characteristics will be needed. A robot control system has to deal with multiple events in a time-coordinated manner, for example, multiple joint control, sensing and handling of contact reactions, path planning, and task planning all require real-time response. Besides the real-time requirement, a robot system distinguishes itself from other systems by the need for varying degrees of intelligence. In practice, there exists a conflict between the real-time requirements and implementation of artificial intelligence [70]. This should be resolved in order to make them coexist and function efficiently within a system.

7.2.1 Real-time system

By definition, a real-time system should respond in a timely and predictable way to unpredictable external events [81]. The two requirements are:

- Meeting deadlines. Once an event occurs, an action has to be taken within a predetermined time limit. For example, servo control has to be completed within one sampling period. Missing a deadline is considered a severe software fault.

- Simultaneity or simultaneous processing: even if more than one event occur simultaneously, all deadlines for all these events should be met. This means that a real-time system should possess inherent parallelism. One major approach to achieving this is through multi-tasking.

Real-time systems can be further classified into soft and hard real-time systems. In a hard real-time system, lateness cannot be allowed because there could be a catastrophic failure if a deadline is missed. In the fish slicing application, for example, if the cutter hits a major bone,

and timely action is not taken by the controller to stop and retract the cutter, the quality of the entire product could be considerably degraded, with serious negative consequences. Another example is in process control, where the sampling and control period should not be missed, or the system accuracy will suffer and the process may become unstable. Other common applications in the hard real-time domain include nuclear power plant control, medical monitoring, and aircraft control.

When a deadline is missed, a soft real-time system may allow the resulting low performance with an increased cost for lateness. The task planning of a robot system is a soft real-time system. If the motion commands cannot be planned on time, the robot will stop between the motions (low productivity), the overall motion may be jerky.

A real-time system does not necessarily mean a fast system, even though the speed is critical in many real-time applications. A database querying system can respond quickly compared to human time scales, but if we use the same system in a critical application that requires responses in the time scale of milliseconds; it will not be fast enough. A real-time system has to meet the deadlines in a predictable way, the deadlines themselves may be of different time scales.

7.2.2 Real-time system with intelligent response

Early real-time systems operated in relatively simple, well characterized environments. Commercial robot controllers usually possess only the position control capabilities, which respond to a given set of task commands. The robot work environment is considered simply as an empty, free-run space. Environmental interaction, unless clearly specified, may cause errors or failure of the system. In general, a conventional robot is an open-loop system at the task level, as illustrated in Figure 7.2.

An intelligent robot system should be able to handle the non-predictable real-world (some

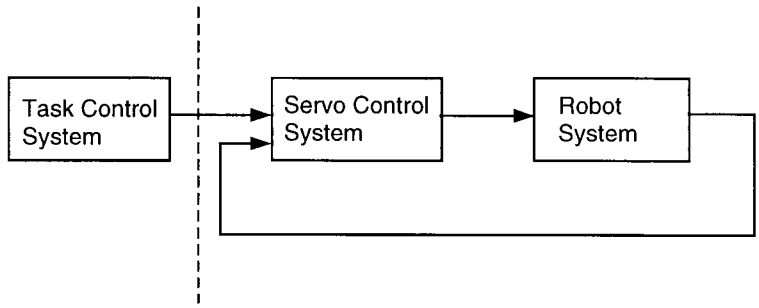


Figure 7.2: Open-loop task control in a commercial robot.

assumptions may be made to limit the complexity of the real world), and should involve sensor-based control even at the task level. The control system has to make decisions according to unpredictable sensory information in real-time (See Figure 7.3).

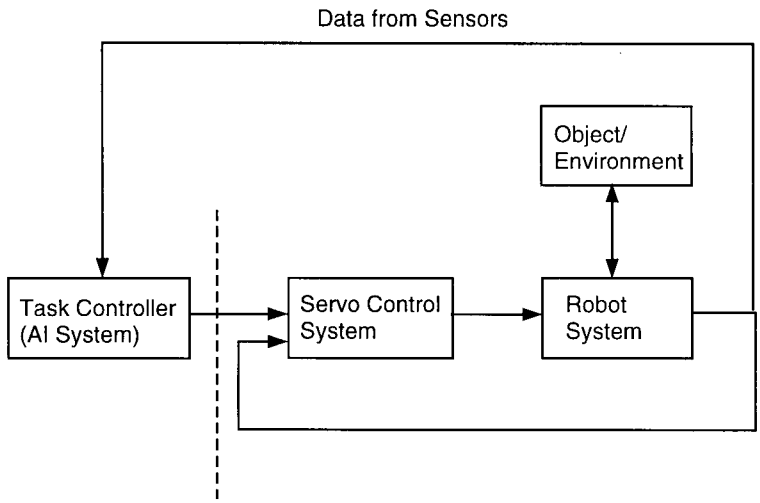


Figure 7.3: Sensor-based, AI-integrated, robot task control.

7.2.3 Artificial intelligence

Artificial intelligence (AI) attempts to computationally model various "intelligent" capabilities of human beings. Many of the AI problems are solved by search, which can be viewed as a basic function of AI. Such AI capabilities may be incorporated into robotic task control. In this

context, jobs in a task space are evaluated and, the best one is chosen and sent for execution. A search method is also used to confirm whether the variables of a job are evaluated.

The difficulty of building real-time AI systems is that they require reasoning until an acceptable solution is found and, the time required for obtaining a solution is not fixed and usually unpredictable. Also, they are required to work continuously over extended periods of time. These needs of an AI system conflict with real-time deadline requirements. There are three basic methods to integrate an AI system into real-time control.

- Embedding AI in real-time. This method forces AI computation to meet deadlines just like other real-time tasks, either by guaranteeing that the worst computation time meets the deadline or by interrupting lengthy computation and taking a "so-far optimal" or a default solution. Direct fuzzy logic control usually takes this approach.

- Embedding real-time in AI. This method employs typical AI methods to solve problems, but under some circumstances these techniques might be short-circuited in favour of a real-time reflexive action. Intelligent monitoring usually takes this approach where AI plays a major role while real-time response is usually not critical, but interrupts AI procedures for emergency handling.

- Cooperating AI and real-time. This method keeps the AI and real-time subsystems separate, but employs an interface of communication and scheduling between the two subsystems, so as to cooperate their individual functions.

Since real-time servo control and high-level intelligent task control can be considered as two separate subsystems, it is natural to use the cooperative AI-RT method in ROACS which is developed in the present research.

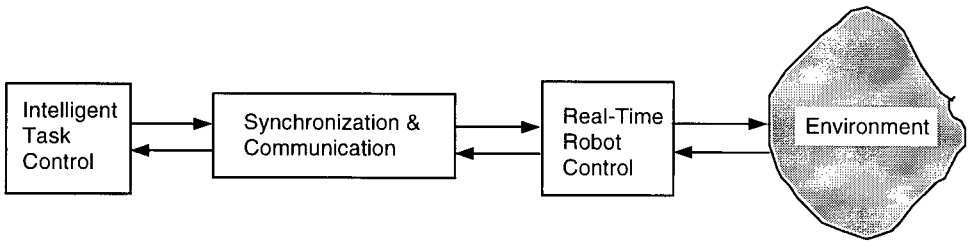


Figure 7.4: A cooperative real-time intelligent system.

7.2.4 Synchronization requirement

An important synchronization requirement of ROACS is that the intelligent task control should be able to change the robot path in real-time. This feature has been ignored in some systems, where a path planner is employed to pass trajectory set points to a servo controller with a single direction first-in-first-out pipe (FIFO) [88]. In ROACS which is developed in the present research, when encountering unpredictable external events, the task controller is able to change the set points of the servo system immediately, thereby handling unexpected events, such as, avoiding catastrophic collisions.

7.3 Client-Server Architecture

Generally, a robotic system consists of a set of relatively independent, yet functionally related components such as robotic arms, robot vision systems, and end-effectors. A robot controller should be able to make these devices work autonomously, while cooperating with other physical components and high-level modules such as a robot-human interface and intelligent decision makers, when performing a task. For example, the low-level direct control of a manipulator should not be affected by a failure within the user interface. In general, the requirements of a robot controller may be summarized as follows:

- **Autonomy:** Robot components should be able to work autonomously. Corresponding controllers should not malfunction in relation to their processing ability, even in the

presence of failures in other parts of the system.

- **Modifiability:** A robot control system should be easily modifiable; for example, through adding or removing components, without having to make substantial changes to the system architecture.
- **Programmability:** Functions of robot components should be changeable through programming, to ensure the flexibility of process automation.
- **Communication:** A well-organized communication structure should be available, through which various functions of the system components can be properly planned and scheduled in order to carry out a task.

The robot control system, as developed in the present work, is intended to have these derivable characteristics. Accordingly, the ROACS is given a client-server architecture as its implementation model (see Figure 7.5), with emphases on real-time performance of servers and the presence of intelligence in clients. As we will show in the sequel, this structure has some other significant advantages as well.

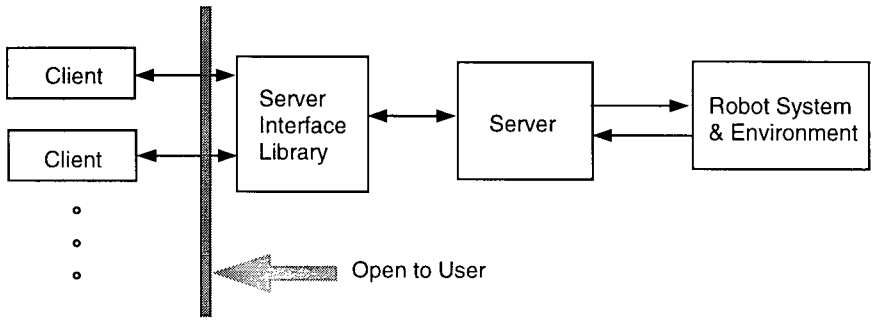


Figure 7.5: A client-server system for robot control.

7.3.1 Client-server systems

A server is a processing element, which is basically a computer process that is running independently, to handle a robotic device with specific time constraints. A client is a process that requires information from servers. Usually clients give commands as well to the servers.

A client-server architecture separates system functions into two parts: the server accepts and executes commands, and provides system information; while a client requests and receives data for analysis, display, etc., and may send commands to the server according to the results of a data analysis. The server part of the architecture is more suitable for real-time control and the client part is better suited for functioning as an intelligent system. This aspect will be further discussed when addressing the structure of ROACS.

A server can be properly designed to accommodate (i.e., open to) customized client systems. This will make the entire system quite open and flexible, and suitable for high-level coordination in a large system which may contain multiple servers. The modularity of a client-server system ensures the reliability of the server, which is critical to hardware control, and protects it from malfunction due to a failure in the client system. With an open-architecture server, the system functions can be easily modified and programmed within the client software.

7.3.2 The structure of ROACS

Figure 7.6 shows the architecture of ROACS. Note that both the server and the clients have a communication interface, that is assisted by a set of library functions. The server has three components: servo control system, path planner, and data analyzer.

The Control Server, which is called an active server, is indispensable in real-time applications. This is quite different from a passive server as usually in database applications, where the server becomes simply a data storage and retrieval tool and, the client software undertakes

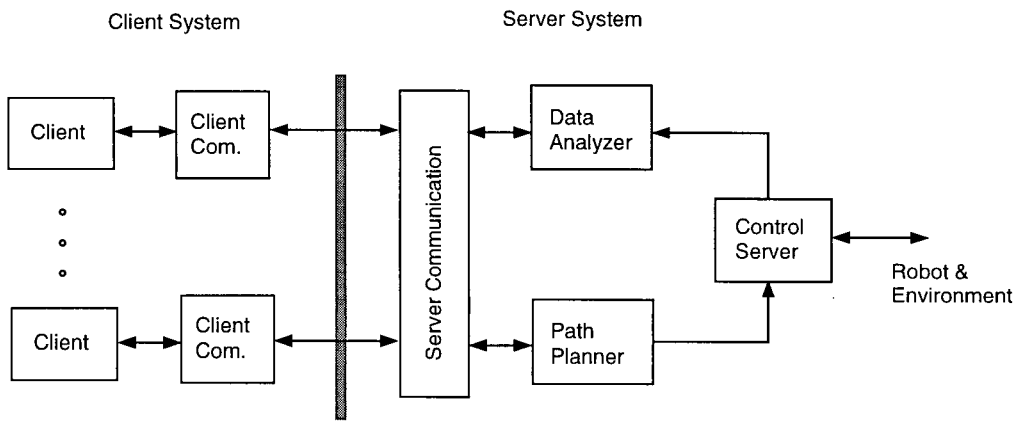


Figure 7.6: A system diagram of ROACS which uses a client-server architecture.

all the operation functions. The active servers enable themselves to assume periodic or event-driven tasks without client intervention. Here, the control server in ROACS carries out periodic sampling and servo control at high bandwidth.

The Path Planner generates trajectory set points for the robot, according to motion commands which it receives from the clients. The Path Planner is a passive server which can be triggered by commands from clients and other servers. The Data Analyzer is also a passive server. It runs when new data is available, or when a client requests for services of data change and retrieval. The basic functions of the Data Analyzer are to communicate data between a client and the Control Server, and to perform data analysis such as interpretation of mechanical impedance information.

While the server system provides several basic operations, the functionality of the overall system mainly depends on client software. Functionality of the system can be enhanced by adding new clients. One of the client modules that is available in ROACS is the intelligent task control module and associated graphic user interface (see Figure 7.7).

The client system consists of four components: task controller, information updating module, display, and task shell/command parser. The task control module and the information-updating module assume periodic tasks, and transmit system information to the display module.

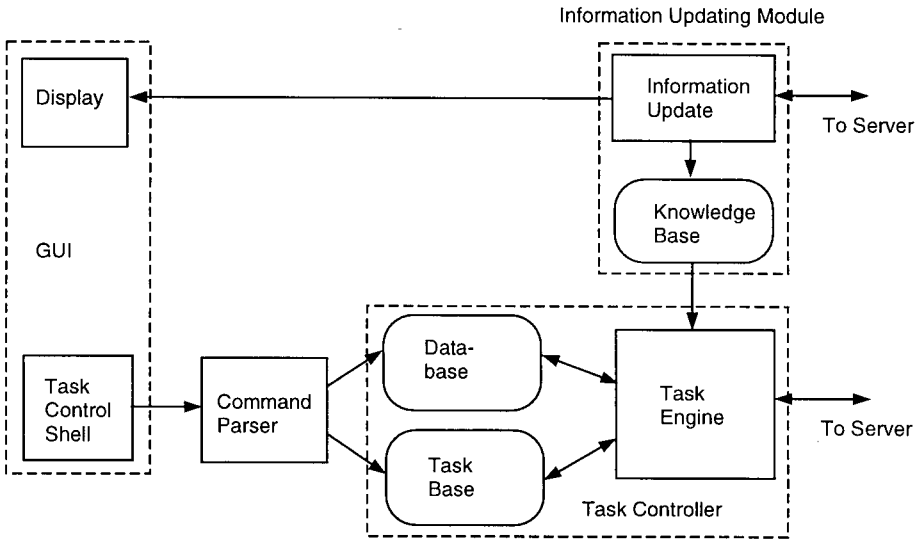


Figure 7.7: A client module for intelligent task control and its graphic user interface.

The task shell and the command parser will stack the commands that are sent to the task base from input sources.

7.4 Scheduling, Synchronization and Communication

In ROACS, there are at least four processes: control server, path planner, data analyzer, a client system, and possibly some threads in the client system. Scheduling and synchronization are essential for proper operation of the entire system. Communication between processes is critical to process synchronization and data transfer. Now we will address these issues.

7.4.1 Process scheduling

ROACS may be scheduled by a customized scheduler or by a real-time operating system (RTOS). The basic requirements are:

- Priority driven. When two or more processes are ready to run, the process with the highest priority should be selected.

- Preemptive. If a higher priority process becomes ready, the running process which has a lower priority should be interrupted in deference to the process with the higher priority.

These properties will ensure that the most critical missions will be executed timely, while the high-level intelligent modules that can consume considerable computing power will not block the low-level and routine control process.

Clearly, in ROACS, the control server should be assigned the highest priority. The control server does sampling and servo control at high frequency, usually 500Hz to 1KHz. It also determines the system state and conducts emergency protection functions. This process should never be interrupted, or blocked by other servers or clients.

The Control Server has three states:

$$\mathbf{S} = \{S_r, S_b, S_s\} \tag{7.1}$$

where S_r denotes the running state, S_b denotes the timer-blocked state, and S_s denotes the stopped state. State transitions are shown in Figure 7.8.

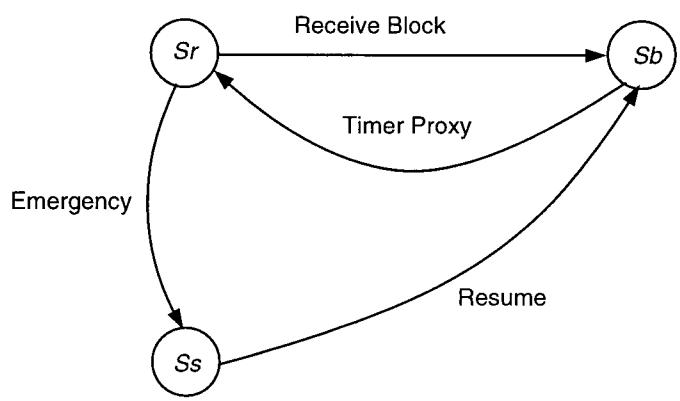


Figure 7.8: State transitions of the Control Server.

Normally the process switches between the running (sampling and control) state and the timer-blocked state. Specifically, when the process completes its periodic task (sampling and control), its state changes from S_r to S_b, and in turn the process is triggered from the S_b state

by a timer to carry out a wake up and run operation. When an emergency occurs, or a stop command is received, the Control Server moves to the stopped-state S_s. This state can be resumed by a command once the emergency situation has been handled.

$$T = \begin{cases} S_r \rightarrow S_b, & \text{when periodic sampling and control is done;} \\ S_b \rightarrow S_r, & \text{when periodic timing is triggered;} \\ S_r \rightarrow S_s, & \text{when an emergency occurs;} \\ S_s \rightarrow S_b, & \text{when the system is resumed to work.} \end{cases} \tag{7.2}$$

The Path Planner and the Data Analyzer take the second highest priority, which is lower than that of the Control Server and higher than that of the clients. The processes may be interrupted by the Control Server.

The Path Planner has two blocked states; *wait* state and buffer-blocked state $\{S_w, S_b\}$, in addition to a running state S_r. The associated transition is shown in Figure 7.9. When a

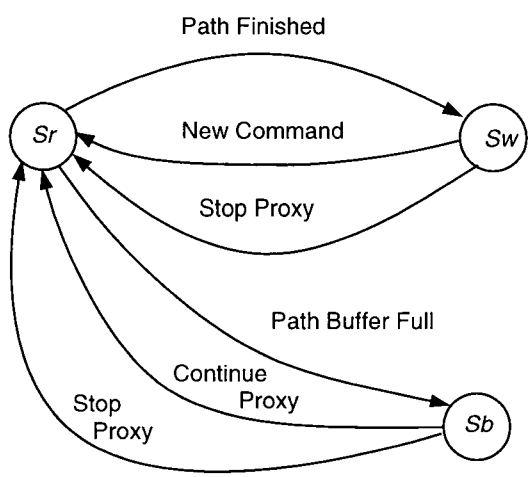


Figure 7.9: State transitions of the Path Planner.

trajectory path is completed, the Path Planner will be blocked in waiting for a new command $\{S_w\}$. A new path command will trigger the process to plan a new trajectory, or a stop-proxy that is sent from the Data Analyzer to stop the current path, will activate the process to remove

the existing path and plan a stop path (deceleration). In case a trajectory is too long and cannot be planned in one shot, the process will be blocked when the path buffer is full $\{S_b\}$. A continue proxy, which is sent from the trajectory set-point consumer (the Control Server) when the set-point buffer is consumed to half size, will trigger the process to continue the path planning. The stop proxy will do the same for the blocked state $\{S_b\}$ as it does for the waiting state $\{S_w\}$. The transitions are as follows:

$$T = \begin{cases} S_r \rightarrow S_w, & \text{when a path is finished;} \\ S_w \rightarrow S_r, & \text{when a new command or a stop-proxy is received;} \\ S_r \rightarrow S_b, & \text{when the set-point buffer is full and path is not finished;} \\ S_b \rightarrow S_r, & \text{when a continue proxy or a stop proxy is received;} \end{cases} \tag{7.3}$$

The Data Analyzer has a relatively simple pair of states: running and blocked $\{S_r, S_b\}$. The transitions are shown in Figure 7.10.

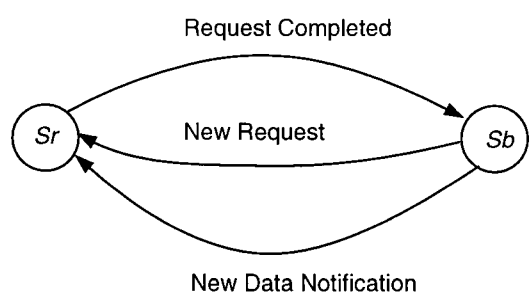


Figure 7.10: State transitions of the Data Analyzer.

Here we have,

$$T = \begin{cases} S_r \rightarrow S_b, & \text{when a request or data processing is completed;} \\ S_b \rightarrow S_r, & \text{when a new request or a new-data notification is received.} \end{cases} \tag{7.4}$$

The client processes take the lowest priority.

7.4.2 Communication and synchronization

Communication and synchronization are highly implementation dependent. The ROACS is designed with the portable operating system interface (POSIX) standard [43] for real-time systems. The user interface of inter-process communication (IPC) and synchronization, of which the implementations are usually operating system dependent, are well defined in the POSIX standards, and hence are portable to other systems.

Since the Control Server operates at a high frequency in the range 500Hz – 1000Hz, inter-process communication should not take more than 100μs in each cycle. This communication is intended mainly for reading set points and passing the sampled data to the Data Analyzer. The shared memory method is used for inter-process communication between servers, as it is the most efficient way to share data between processes. A shared memory object is a name-based memory area that is mapped to different process address spaces, and hence can be accessed by different processes. However, the free-form, low-level nature of shared memory may also corrupt the data and hang a program very easily. In particular, when using shared memory there will not be implicit synchronization between processes. Exclusive data access has to be guaranteed in the system design in order to protect data from corruption.

The shared memory between the Control Server and the Path Planner is a set-point buffer. The Path Planner writes trajectory setpoints into the buffer, and the control server retrieves these setpoints. There are two important issues pertaining to process synchronization: the buffer length dilemma and exclusive access.

- Setpoint length dilemma. In the extreme case where the buffer length is 1, the Path Planner will calculate only 1 point every time. Such a short setpoint buffer allows for swift path changes, which is an advantage. The drawback is that the control server may lose the set point in case the planner fails to update the path on time. This happens when a path preparation takes longer time than the trajectory update period, or too much

competition exists at the priority level of the path planner. This arrangement creates an overhead for the system, and the system may lose control easily when an emergency occurs.

Because of the use of shared memory, planned points can be easily replaced by those for a new path, e.g., a path to stop. If the setpoint buffer is too long, the computations for the planned path will be wasted when a path change is requested/commanded. The recommended buffer length is 20 to 100 points.

- Data protection. A system method such as semaphore, which blocks a process, may not apply to ROACS. Obviously, the Control Server should never be blocked (the block time may exceed its sampling period). A circular buffer is designed with a writing pointer $pt1$, and a reading pointer $pt2$ with the condition

$$pt1 \neq pt2 \tag{7.5}$$

It means that if $pt1 = pt2$, the planner cannot overwrite the existing data, since it has not been used yet. Note that $pt2 - 1$ is the last point that the planner will write. In this manner, a single setpoint will never be accessed at the same time by both the Path Planner and the Control Server, although the entire circular buffer is simultaneously accessible. Here, simultaneous access means that a process can be interrupted in the middle of the buffer operation, say, during a buffer write, while the interrupting process is also able to access the buffer.

The Path Planner will stop when the setpoint buffer becomes full, and will reject any new commands. A continue-path proxy from the Control Server can trigger the process to continue path planning.

The Control Server and the Data Analyzer employ another shared memory for data transfer. It uses block transfer instead of circular buffer. When the control server stores data, a pointer

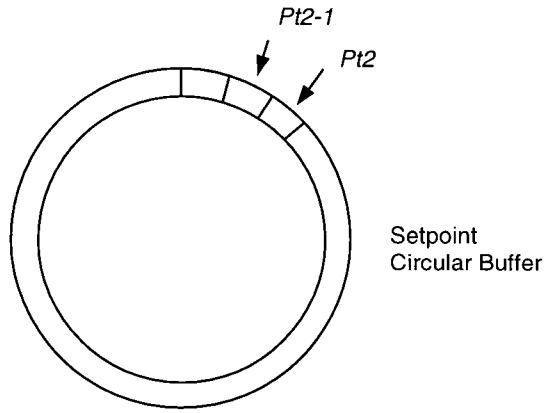


Figure 7.11: Illustration of a circular setpoint buffer.

that points to the stack size should increase. The Data Analyzer consumes the entire data block and resets the pointer.

The Data Analyzer issues a stop-path proxy to the Path Planner when a change of path is needed. The servers and the clients may adopt the priority-driven message queue or the message passing method for communication depending on the implementation platform. Implementation of the ROACS model in the real-time operating system QNX will be detailed in the next chapter.

7.5 Summary

This chapter discussed the real-time open architecture control system (ROACS) that is developed in the present research. First, the system implementation requirements were given and then the challenges of real-time intelligent system were discussed. A real-time response should possess the capability to interrupt lengthy and time-variant AI processes. Priority and the preemptiveness are two basic properties of real-time scheduling, which have been incorporated into ROACS.

A client-server architecture was adopted as an implementation model for ROACS, which is well suited for cooperative real-time intelligent systems. Here if the task engine is interrupted, the best solution so-far would be used. This model has the advantages of open architecture,

flexibility, programmability, and efficient communication. These characteristics make it a good implementation model for a hierarchical system whose objective is sensor based robot control.

A detailed architecture for ROACS was proposed. It consisted of the Control Server, the Path Planner, the Data Analyzer, and client software. Synchronization and communication between the Control Server and the Path Planner; the Control Server and the Data Analyzer; and between the servers and the clients were developed.

Chapter 8

Prototyping of ROACS

This chapter will address the issues in the implementation of the real-time open-architecture control system (ROACS) that is developed in the context of the present research, and was modeled in the previous chapter. A brief introduction will be given to outline the main characteristics of ROACS with respect to its implementation. By satisfying these characteristics, the implementation platform will be chosen first, where several popular real-time operating systems such as VxWorks, Windows NT, and QNX, will be discussed and compared. On this basis, QNX operating system will be selected as the implementation platform for ROACS. Next, the details of the implementation, such as the design of a multi-tasking program for client-server system, and message passing between processes, will be presented.

The chapter will also provide details of commands at different hierarchical levels. The data structures given in this chapter are very useful in developing client software, which need to communicate with the developed servers. The real-time task language (RTTL) is also very useful in programming different applications using the developed robot system.

The organization of the chapter will be as follows: Section 8.1 will briefly present the characteristics of ROACS. Section 8.2 will discuss and compare the features of several real-time operating systems. Sections 8.3 and 8.4 will present the implementation details of the servers and the client software, respectively. Section 8.5 will describe an application of the developed system. Section 8.6 will conclude the chapter.

8.1 Introduction

ROACS is a new architecture for cooperative intelligent real-time systems, and has been developed in the present research. As natural requirements of a cooperative real-time intelligent system of this type, the system should have the following properties:

- Real-time characteristics. ROACS is a multi-tasking system. Each task has an assigned priority, and can be pre-empted by higher priority tasks. This will guarantee first execution of mission-critical tasks that have high priority.

- Separation of the management of real-time and AI attributes. A real-time subsystem that is dedicated to low-level control and a separate intelligent subsystem that is dedicated to high-level task control are present in a decentralized manner. This separation resolves conflicts between real-time and intelligent systems, that result from varying performance requirements. A client-server structure plays an essential role in this separation.

- Network-transparent communication. Communication and synchronization between the real-time subsystem and the intelligent subsystem result in an overall integrated system that behaves as a real-time intelligent system. Network-transparent communication may scale ROACS up as a network controller.

This chapter will discuss the implementation of the platform of ROACS, according to the requirements identified above. Also, detailed realization of the ROACS model will be given.

8.2 Real-Time Operating System

For the prototyping of ROACS, a real-time operating system is required. Due to its complexity, it is unlikely that ROACS could be built on a bare hardware system. Therefore, two parts are needed to make ROACS as a real-time system: (1) a real-time operating system (RTOS);

and (2) a real-time application program. The operating system is the base of the application program. If an application is built on a non-RTOS, then the real-time performance of the system would be compromised [81]. Therefore, the selection of a proper RTOS is crucial in the current development, and deserves some discussion. In the present research, the following real-time operating systems were considered:

Tornado/VxWorks is a well-known RTOS. It runs on a variety of processors. It supports POSIX.1b. Note that VxWorks is intended for use with embedded real-time systems. It is not self-hosted; applications can only be developed using cross-development hosts. Tornado/VxWorks is quite expensive compared to other RTOS. Also, VxWorks does not provide memory protection, hence it is not ideal for modular systems, such as ROACS.

Windows NT has been adopted by some vendors to develop real-time systems. However, it is not a real-time operating system although it has some real-time features [81]. There are many reasons why NT is not suitable as a RTOS, most critical of which is the Deferred Procedure Call (DPC) problem. In Windows NT, the interrupt functions are placed in a DPC instead of an interrupt service routine (ISR), and all DPCs are put in a queue with the same priority. The following problem rises here: mission critical DPCs can be blocked by other DPCs (e.g., mouse, keyboard, hard disk services) that are placed ahead in the queue. With real-time extension enhancement, interrupt latency can be limited to $50\mu s$ which is quite large compared to the usual 3-4μs in RTOS. However, Windows NT is widely used with DSP boards and other RTOS for non-mission critical tasks and in user interfaces, while DSP and RTOS run real-time tasks. It is clear that this type of system will increase the cost and complexity of development.

QNX is a real-time operating system [79]. This is the RTOS which we have selected for prototyping ROACS. To fully appreciate the utilization of important QNX features in the implementation of ROACS, QNX operating system will be discussed in detail in the following subsections.

8.2.1 QNX real-time operating system

The basic design of QNX operating system (OS) incorporates a microkernel which is different from an OS with a monolithic design, such as Windows NT and UNIX, where the system-level functions like hardware drivers must be part of the kernel to run. The microkernel of QNX OS is extremely small. It contains a minimum number of features which implement the basic system calls. These include message passing along with other interprocess communication and support for interrupt handling. Other functions of the operating system are provided optionally and modularly through processes that communicate mostly through message passing. These processes are practically no difference from a user-written program. QNX OS meets the real-time requirements that have been discussed before; specifically:

High real-time performance. QNX OS supports multitasking (multithreads with QNX/ Neutrino) programming. It is POSIX compliant (POSIX.1b, .1c, .1d, .2). With a context-switch speed of 1.1 microseconds and interrupt latency of 4.4 microseconds on a Pentium 100MHz, QNX delivers system services significantly faster than a traditional kernel architecture, while maintaining all the inherent advantages of a true microkernel design.

Robustness and reliability. In QNX OS, processes run in their own protected memory space. Device drivers as well as other user applications run in the user space; they cannot overwrite any areas outside their own address space. As a result, kernel faults are extremely rare under QNX OS, and also, a faulty process will not affect other processes; for instance, in ROACS, the critical control server should not be affected by a failure in the graphic user interface (GUI). Since, typically in other operating systems, the drivers run in the kernel mode, user applications that call drivers may destroy the system memory space and will lead to a system crash.

Complete scalability. The microkernel design of QNX is modular and hence scalable. This can be achieved by adding or removing function modules. While QNX OS fits in a deeply

embedded system quite well, it also can be scaled up for use in a factory-wide control system.

Network transparency. QNX provides network-transparent communication; e.g., message passing, pipes, signals, and message queues over network. This turns the entire local area network (LAN) into a single logic machine. As a result, applications can be distributed across a network without changing a single line of code. This feature allows ROACS to run its real-time subsystem and the intelligent subsystem in different machines within a QNX network.

With rapid growth of computation power and the sharp drop in price of hardware, personal computers (PCs), especially Pentium machines, are very quickly moving into the domain of real-time industrial applications. Prototyping of ROACS in Pentium machines with QNX real-time operating system exploits the advantages of a high-performance operating system, low cost, and easy-maintenance hardware.

8.3 Implementation of Servers

The structure of the server system of ROACS has been implemented in its original form using QNX. Communication is done through shared memory and QNX message passing.

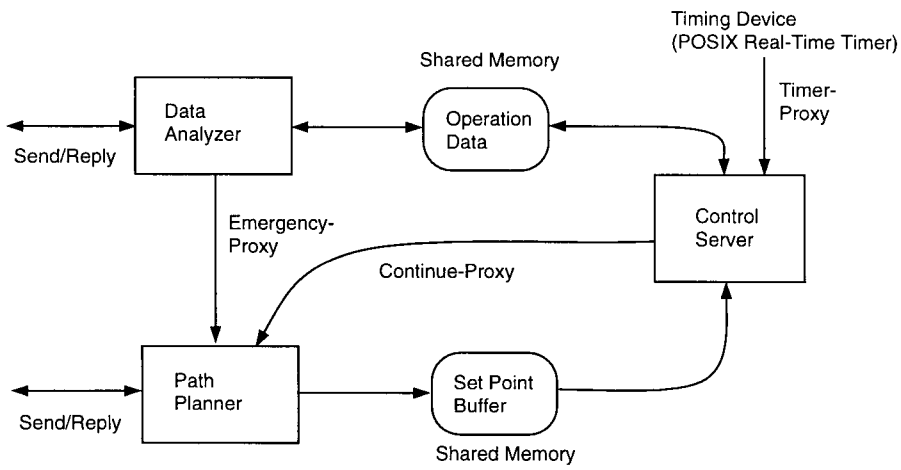


Figure 8.1: Architecture and message passing of the ROACS server.

8.3.1 Control server

The control server is an active server that is attached to a timing device (hardware timer or software timer). A POSIX real-time timer is used in the prototype of ROACS. Timing control is done through timer proxy triggering at the sampling times (see Figure 8.1). The process is usually blocked by a "Receive" of the timer proxy. The basic functions of the control server are listed below: (see Figure 8.2)

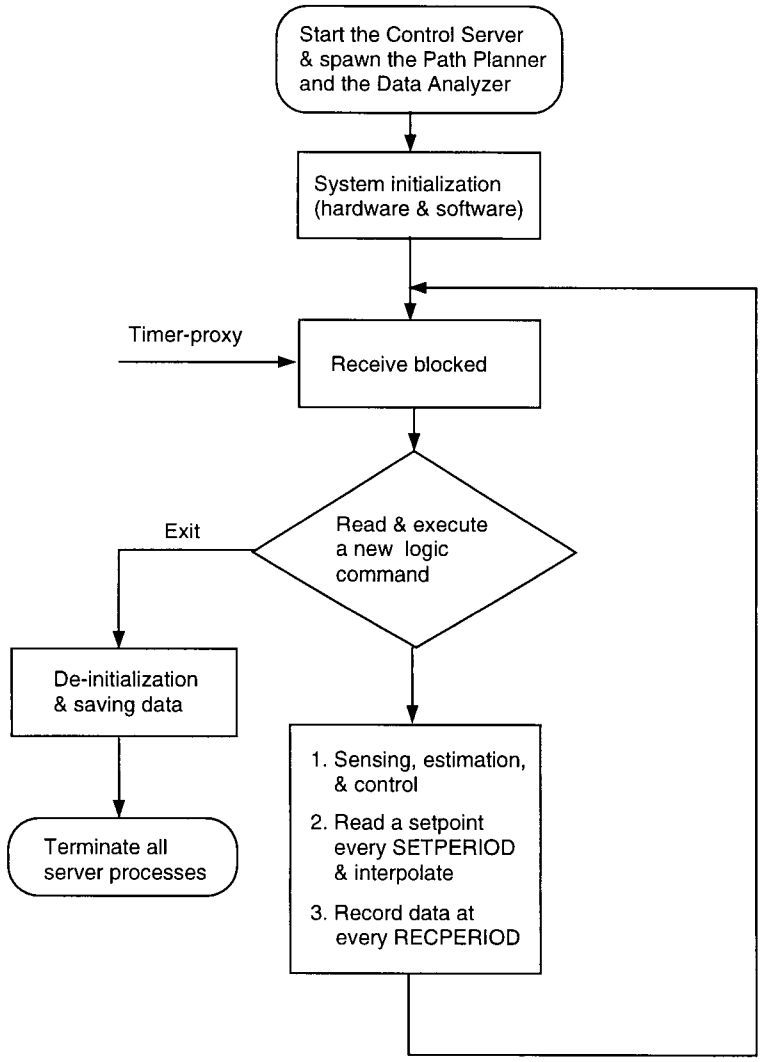


Figure 8.2: Flow chart of the Control Server process.

- Initialization of system parameters. Most of the parameters are read from a plain text initialization file (e.g., *puma.ini*) in the format,

  KEYWORD: parameters; # comment

- Initialization of system hardware. Encoder, analog to digital conversion (A/D), and digital to analog conversion (D/A) cards are initialized, and reset.

- Low-level sensing, estimation, and control. Robot joint positions and motor currents are read from hardware, and velocity, acceleration and external force variables are estimated (see algorithms in Chapter 3). The control algorithm is applied and the control signals are sent to the power amplifiers through D/A conversion for driving the robot.

- Read the desired trajectory setpoints at every trajectory planning period (SETPERIOD) and interpolate in every control loop (see algorithms in Chapter 5).

- Record data into the shared memory at every recording time (RECPERIOD).

- Checking the trajectory following error, and protection. If a trajectory following error is detected that is larger than a given threshold value, the system should be stopped immediately (a fatal error). The detection executes the following check:

  $$\textit{If } |e_i| > e_{i_{max}}, \textit{ then joint } i \textit{ is faulty}$$

- Execute the logic operations.

The shared memory that is used for the setpoint buffer has the following data structure:

```
typedef struct {
        char    startFlag;
        BYTE    pt1; // write point
```

```
BYTE    pt2; // read point
SetPoint setpt[SETLENGTH];
} ShmSetPoint;
```

where *setpt* is the circular buffer with each unit containing the desired position, velocity and acceleration of all six joints; $pt1$ is the writing pointer; $pt2$ is the reading pointer; and $startFlag$ is used to pass logic commands from the Path Planner to the Control Server. Implemented logic commands are:

- AC_ON – turn on AC power of the robot system;
- AC_OFF – turn off AC power of the robot system;
- DC_ON – turn on DC power supply of the power amplifiers;
- DC_OFF – turn off DC power supply of the power amplifiers;
- BRAKE_RELEASE – release the joint brakes;
- BRAKE_HOLD – hold the joint brakes;
- END_EFFECTOR_ON – turn on the end-effector;
- END_EFFECTOR_OFF – turn off the end-effector;
- START – start the robot system, which is equivalent to the combination of command AC_ON, DC_ON, and BRAKE_RELEASE;
- STOP – temporarily stop the robot, which is equivalent to the combination of command DC_OFF, BRAKE_HOLD, plus resetting DAC;
- EXIT – stop the robot system, quit the Control Server, and terminate the Path Planner and the Data Analyzer.

- WAIT – inform the path planner not to receive any new commands before the setpoint buffer is drained.

8.3.2 Path planner

The Path Planner is triggered by any message that is sent to it. The messages can be commands from client software or proxies from the Control Server and the Data Analyzer. The commands take the following data structure:

```
typedef struct {
        char startFlag;
        char coordFlag;
        double r1[3][3];
        double p1[6];
        double r2[3][3];
        double p2[6];
        double tv; // tool velocity
        double ta; // tool acceleration
        double av; // angular velocity
        double aa; // angular acceleration
        } Command;
```

Here $startFlag$ is a flag used for specifying commands (up to 256 commands), and $coordFlag$ is a flag indicating the coordinate system in which the points are represented. There are two primary types of robot coordinate systems: world coordinates and joint coordinates. Note that $r1$ and $p1$ represent the first point in the command, and $r2$ and $p2$ represent the second point in the command. In world coordinates, r is the orientation matrix, and the first three members of the p array represent the position in the Cartesian space. In joint coordinates, p represents all

six angles of the robot joints (for a six DOF robot). Note that tv, ta, av, and aa are used for specifying velocity and acceleration in a motion command.

Figure 8.3 shows a flow chart of the Path Planner. The Path Planner runs until either a

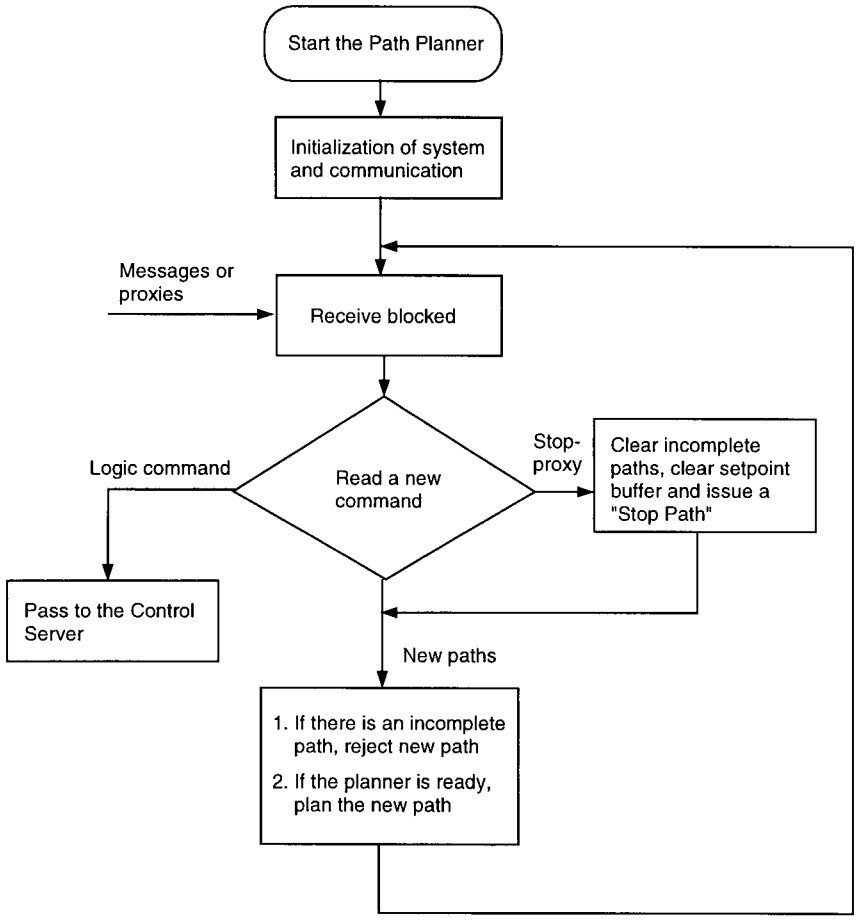


Figure 8.3: Flow chart of the Path Planner process.

trajectory path is fully planned or the setpoint buffer is full. In the former situation, the Path Planner is ready to accept a new path, while in the latter situation, the Path Planner has a path yet to be planned completely, and will not accept a new path, but will return the error message "BUF_FULL" to the sender. However, emergency logic commands will be accepted at any time even when the buffer is full.

Planning of the incomplete path can be continued by a server notification which tells the path planner that there is enough space in the setpoint buffer, and the planning process should be continued. The trade off of the length of the setpoint buffer (SETLENGTH) has been discussed in the previous chapter. If the setpoint buffer is too short, there will be an overhead of notification, and may cause missing the deadline of trajectory planning. If it is too long, there will be a computation overhead in the case of sensor-based task control; i.e., the planned path would be discarded when a new path is required to be planned according to the operation information.

Because the Control Server and the Path Planner are located in two separate processes, significant flexibility advantages are achieved; for example, they can run at different bandwidths. The Path Planner does not have to run at every control period. Also, the Path Planner functions as a communicator to client software on behalf of the control server; thus, the Control Server would not be blocked. For example, the Send/Reply will block a process for 2ms, for communication between two processes at a common node, and 10ms for processes at different nodes that are linked with Arcnet. This would be an excessively long block for the Control Server which runs every 1-2ms, if the Path Planner is not present.

The basic functions of the Path Planner are:

- Establish communication handles with the Control Server and the Data Analyzer. This includes mapping the shared memory to its own address space and obtaining proxy identifications.
- Accept commands and pass logic commands that are a subset of commands to the Control Server.
- Plan a path or reject a path according to the current status.

Except for transferring the logic commands that are listed in the Control Server, the Path Planner will compute the robot path for the following commands:

- MOVEW. This command moves the robot end-effector from the specified first point $\{r_1, p_1\}$ in world coordinates to the second point $\{r_2, p_2\}$. A linear path should be planned between the two points.

- MOVEJ. This command moves the robot end-effector from the specified first point $\{p_1\}$ in joint coordinates to the second point $\{p_2\}$, with the joints moving synchronously.

- MOVETOW. This command is identical to MOVEW, except MOVETOW moves the robot end-effector from the last point in the previous path rather than from $\{r_1, p_1\}$, to $\{r_2, p_2\}$.

- MOVETOJ. This command is identical to MOVEJ, except MOVETOJ moves the robot end-effector from the last point in the previous path (in joint coordinates) rather than from $\{p_1\}$, to $\{p_2\}$.

- MOVETOWR. This is a relative motion command which moves the robot end-effector from the last point in the previous path through a distance in world coordinates that is specified by $\{p_2\}$. Only the position can be changed, and the orientation is kept the same.

- MOVETOJR. This is a relative motion command which moves the robot end-effector from the last point in the previous path by an increment $\{p_2\}$. The last setpoint is read first, and the destination is computed by adding the increment of each joint. Joint motion is planned.

- MOVESTOP. This command clears the incomplete path and the set point buffer, and plans a short stop path based on the maximum acceleration that is given for stop. No position parameter is required, but the final stop point is returned to the calling party. This enables position feedback in task-level planning.

- GET_BUFLEN. This command returns the current length of the setpoint buffer. Zero length means no motion.
- MOVETOJBUTTON. This command moves the joints that are specified by the flags MOTION+ and MOTION- to either the maximum or the minimum limits. Other joints will remain stationary. This command works with MOVESTOP for continuous jogging.

These functions provide basic operation of the robot. Complicated tasks should be planned in task control level using these commands based on the operation information. This will be discussed later in the context of task control level.

8.3.3 Data analyzer

There are two basic functions of the Data Analyzer: data communication and data analysis. Data communication has special importance in the controller of ROACS, since data feedback to the task control level is a designed feature of the present system, and furthermore access to operation data is essential in an open architecture controller. Again, the control server would not handle the communication in a single sampling period; thus, the Data Analyzer plays an important role in data communication between the Control Server and client processes.

Another important function that is available in ROACS is data analysis, which primarily carries out impedance estimation and interpretation. The Data Analyzer runs on client commands or server notification of new record data. When the operation data meet some specified criteria; e.g., an impedance threshold value, a proxy will be issued to the Path Planner to stop the robot.

The communication between the Data Analyzer and the Control Server is based on a shared memory that has the following data structure:

```
typedef struct {
        // Server data
```

```
        BYTE pt;
        RecordData data[RECLENGTH];
        Flags flags;
        // New data flag
        char newData;
        double kp[CHNUM];
        double kd[CHNUM];
        } ShmRData;
```

Here, the shared memory has two parts. The first part consists of the data records portion *data* and the control server operation flags *flags*. A pointer *pt* indicates the current number of records. When the data have been consumed; for example, retrieved by a client process, the pointer will be reset to zero. The second part of the shared memory is dedicated to control parameter updating. Here, *newData* is the flag which indicates that a new parameter set is available. Note that *kp* and *kd* are proportional and derivative (PD) control parameters, respectively. The communication port that is open to clients for parameter updating adopts a message structure:

```
typedef struct {
        char newData;
        double kp[CHNUM];
        double kd[CHNUM];
        } Msg;
```

Except for parameter update, *newData* in the message structure is also a command for data operations. The basic commands are listed below.

- GET_LAST_DATA. This command requests the Data Analyzer to send the last record of operation data to the calling party.

- GET_DATA. This command requests the Data Analyzer to send all records available thus far to the calling party, and then resets the record pointer.
- SET_PID. This command replaces the PID parameters in the shared memory with the ones sent by the message. A new data flag in the shared memory should be set to notify the Control Server.
- GET_PID. This command requests the Data Analyzer to send the current PD parameters to the calling process.
- GET_FLAGS. This command requests the Data Analyzer to send current operation flags, which are stored by the Control Server in the shared memory, to the calling process.
- SET_SAFETY. This command sets the safety check flag; hence, the Control Server will monitor the trajectory-following error and the motor current for safety, and will shutdown the system if a fatal error occurs.
- UNSET_SAFETY. This command removes the safety check flag. This is used in some experiments where no safety check is required; for example, in step response experiments.
- STOPAT. This command sets the stop condition for the robot. When the condition is satisfied in execution, the Data Analyzer will trigger a proxy to the Path Planner which will stop the robot by changing motion path. The task level command *stopat* is implemented using this command.

When a client module is monitoring the system operation, it should use GET_LAST_DATA and GET_FLAGS to check the operation status. If it is required to save operation data, then the appropriate command would be GET_DATA.

Convenient functions have been developed for the client software to access the servers. The details are given in Appendix B.

8.4 Implementation of Intelligent Task Control

Intelligent task control and graphics user interface function within the software of a client, which runs at a much lower bandwidth than the server does. There are three parts to the system; namely, the task controller, the information updating module, and the GUI. Figure 7.7 gave a detailed structure of the client system.

8.4.1 Task control module

The task engine is attached to a POSIX real-time timer, which runs at every TASKPERIOD. This period should be larger than 20ms in an Arcnet-based QNX local network, in order to reduce the communication overhead by considering a 10ms send block in communication between different nodes. The task engine follows the formal representation that is given in Chapter 6; specifically,

1. Search for the best job and always keep a best so-far solution. If there is no solution available, wait for the next cycle; there might be an information update and, as a result, some commands could become executable.

2. Execute the best job. If the task engine fails to send the job to a server (e.g., servers are busy for a pending path), the job should be kept in the task base.

3. Remove the executed job.

To dynamically manage the task base, a double-chain data structure has been implemented for task representation. The data structure is as follows:

```
typedef struct _task {
        char cmd[10];
        int prio;
```

```
        int argc;
        char *argv[6];
        struct _task *prev;
        struct _task *next;
        } Task;
```

Clearly, it has the basic structure of the representation in formula 6.1), $task(cmd, prio, argc, argv)$. In addition, two pointers $\{prev,\ next\}$ are used to link the task base in bi-direction. Insertion and deletion of tasks can be easily manipulated by using pointers.

Variables take a similar structure. It also has a double-chain structure based on the representation in formula (6.2), $var(name, type, data)$; thus,

```
typedef struct _variable {
        char name[10];
        int type;
        double p[6];
        double r[3][3];
        struct _variable *prev;
        struct _variable *next;
        } Var;
```

where *name*, and *type* denote the variable name and the variable type, respectively. The convention of names and types is given in Table 8.1. Note that $\{p,\ r\}$ are the variables associated with variable quantities, and $\{prev,\ next\}$ are pointers.

The ROACS system, which is developed in this research, uses the following name conventions:

Commands. The *move* commands suffixed with w for world coordinate motion, with j for joint space motion, with *to* for destination move, and with r for relative motion.

Variables. Information of the variables is encapsulated. The variable names contain information on their types. Table 8.1 gives the basic types of variables.

Table 8.1: Variable name convention.

Variable Name	Type Code	Key String	Representation
Position in world coord.	1		r[3][3], p[3]
Position in Joint coord.	10	J	p[6]
Velocity	2	VEL	r[0][0]
Impedance	3	IMP	r[0][0]

8.4.2 Task command shell

A task shell is implemented in the prototype of ROACS. A command parser is developed to interpret the real-time task language (RTTL). There are two types of commands: (1) RTTL commands and (2) parser level commands. The RTTL commands are stored first in the task base and then run by the task engine, where the best commands are always selected first. The parser commands are executed in the interpretation level only, and can be considered as shell commands. The commands are:

- A task name. Information of the named task will be displayed.
- A variable name. Information of the named variable will be displayed.
- *task*. This command displays the entire task base upon call.
- *var*. This command displays the entire variable database upon call.
- *clear*. This command clears the task base and the variable base, but will load some constant parameters from the file "task.ini".
- *run*. This command tells the task controller to start.

- *step*($F10$). Step through the task base.
- *pause*. Pause the task engine, which can be reactivated by *step* or *run*.
- *load* filename. This command loads the task file *filename* into the shell.

The implemented RTTL commands are listed in Table 8.2.

8.4.3 Information update and GUI

Information from the low-level operation of a process should be fed back to the task control level in order to fully enable sensor-based control technologies. In ROACS, this is accomplished by the information updating module. This module runs at the same frequency as the task engine, so as to synchronize sensing and control at the task level. Information feedback is done through communication to the Data Analyzer. Data is resampled at the task control bandwidth. However, it can save all run time data if needed for off line data analysis.

The graphics user interface (GUI) of the robot control system is shown in Figure 8.4. There are six real-time trend windows for displaying the signals. Each of the six windows can display: desired position *dpos*, position *pos*, velocity *vel*, current *crt*, control signal *ctr*, and impedance *imp* as options for each joint. Options can be selected online, in an option dialog box.

The scroll bars and arrow buttons under the trends allow direct and interactive motion control of robot joints, online, by clicking the arrow buttons. Joint positions are displayed in both desired and actual values. These are the basic jogging functions for robot operation. More and complicated operations may be carried out through a task shell. Robot tasks can be programmed using RTTL and loaded and run in the task shell.

An HTML-based online help window is also implemented. A typical display of the help window is shown in Figure 8.5.

In addition, a Kalman filter interface (see Figure 8.6) has been created for research in data

Table 8.2: Implemented RTTL commands.

Command	Arguments	Comments
delay	*num*	*num* TASKPERIOD delay.
exit		Terminates servers and exits the system.
wait		Informs the Path Planner not to accept new commands until the pending paths are completed.
set	$var1$=$var2$	Assigns $var2$ to $var1$. $var2$ could be a constant.
command	*cmd*	*cmd* is any server-acceptable logic commands.
query	*type var*	Defined *type*: "setpoint", "setpointJ", and "positionJ".
display	*var*	Displays the variable online.
movew, movej	$var1$, $var2$, *vel*	Moves from coordinate $var1$ to coordinate $var2$ at speed *vel*.
movetow, movetoj	$var2$, *vel*	A relative move from current coordinate to destination $var2$ at speed *vel*.
movetowr, movetojr	$\Delta var2$, *vel*	A relative move from current coordinate by an increment $\Delta var2$ at speed *vel*.
moveiw, moveij	$var1$, $var2$, $var3$, *vel*, *exp*	A flexible motion from coordinate $var1$ to coordinate $var2$ at speed *vel*. $var3$ stores the stop point if the condition *exp* becomes true and the robot is asked to stop; otherwise, $var2$ is assigned.
mover	a, b, d, O, *exp*	A flexible motion from a to b. Once the condition *exp* becomes true, the motion should be shifted through a distance d along orientation O.

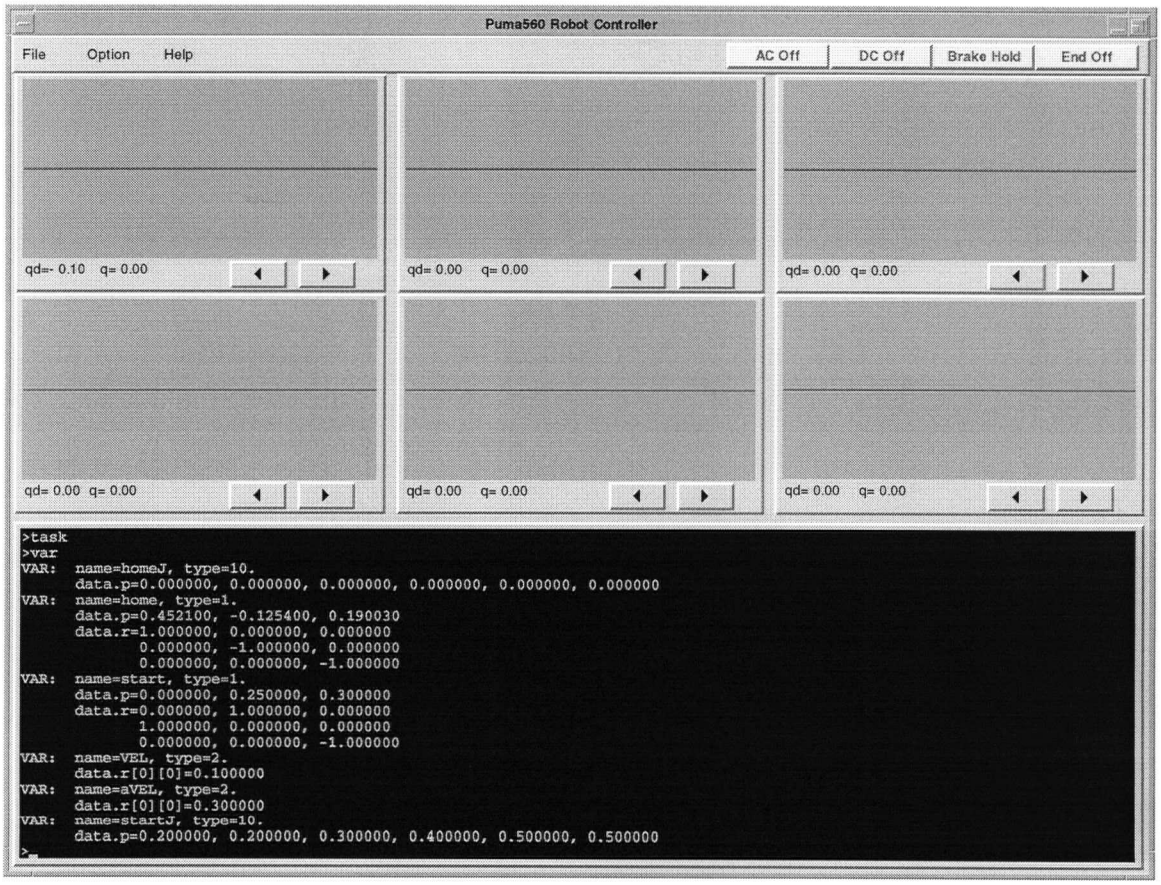


Figure 8.4: The graphics user interface of ROACS.

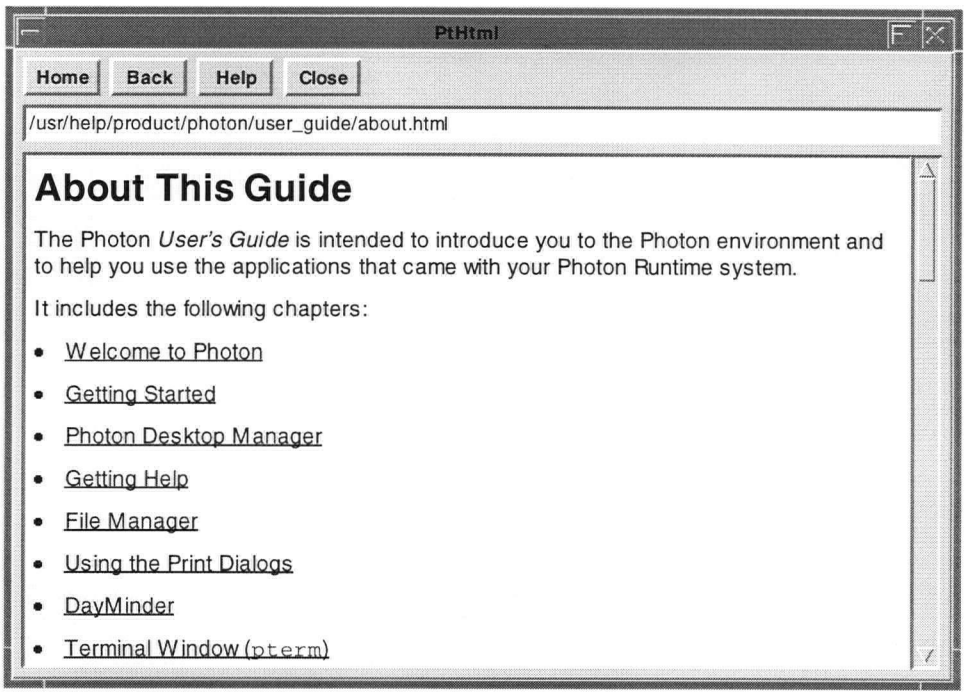


Figure 8.5: A typical display of the help window.

analysis using Kalman filter techniques. It has different display options. Parameters of the Kalman filter can be changed through this interface.

8.5 Salmon Slicing Using RTTL

To illustrate the application of RAOCS which has been developed in the present research, the specific example domain that was introduced earlier in the thesis is performed now using this system.

A view of the laboratory prototype for robotic fish slicing is shown in Figure 8.7. The setup includes a PUMA 560 robot manipulator fitted with an electric meat carver, a low-level controller which has been developed in this research using an industrial Pentium 90MHz PC board that is located in a passive ISA bus chassis which also houses the A/D and D/A cards and the optical encoder cards, an optional second PC computer Pentium 200MHz, and some

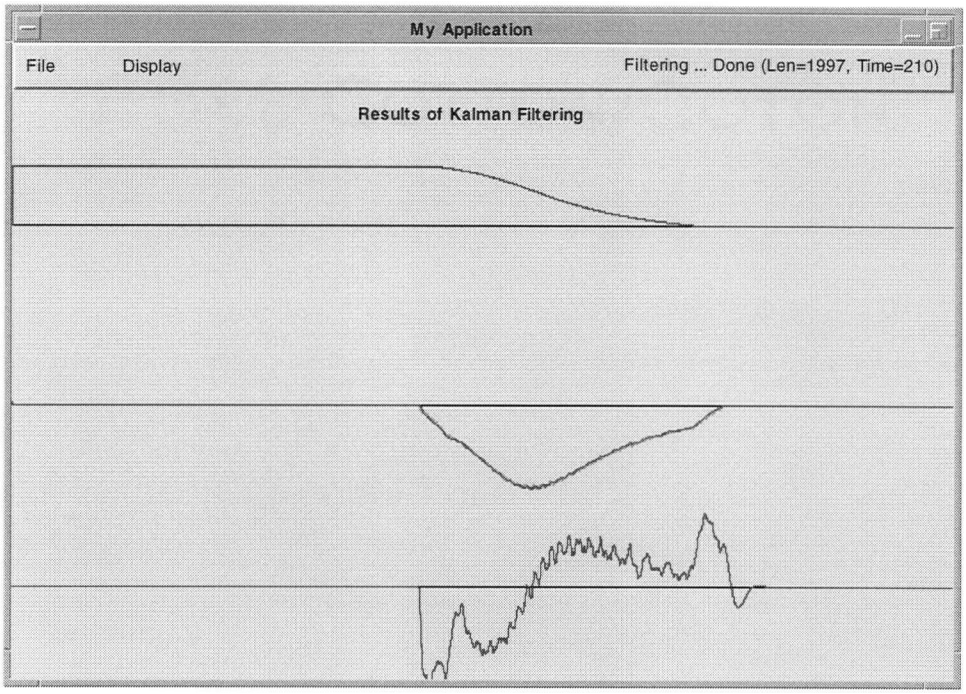


Figure 8.6: The user interface of Kalman filter.

signal conditioning hardware. The ROACS system in its present form can run either in a single node system (Pentium 90MHz only) or in a QNX network (both computers). In salmon slicing experiment, both nodes are used; the Pentium 90MHz run as the server and the second Pentium is used to run the task control system and the graphics user interface.

Figure 8.8 shows a close-up view of the robotic cutting module: an electric cutter that is mounted at the wrist of the PUMA 560 robot. In slicing a salmon for high-end products such as boneless steaks, one requirement would be to avoid cutting through the backbone. Here, mechanical impedance of the cutting process is estimated from the sensed currents of the robot actuators, and is used in task control. Object features of the process; for example, firmness of the fish, can be extracted as well from impedance information. Typically, the magnitude of impedance that is associated with cutting through a bone is significantly high compared to that for cutting through skin, fat or meat. Furthermore, the nature of the impedance; specifically,

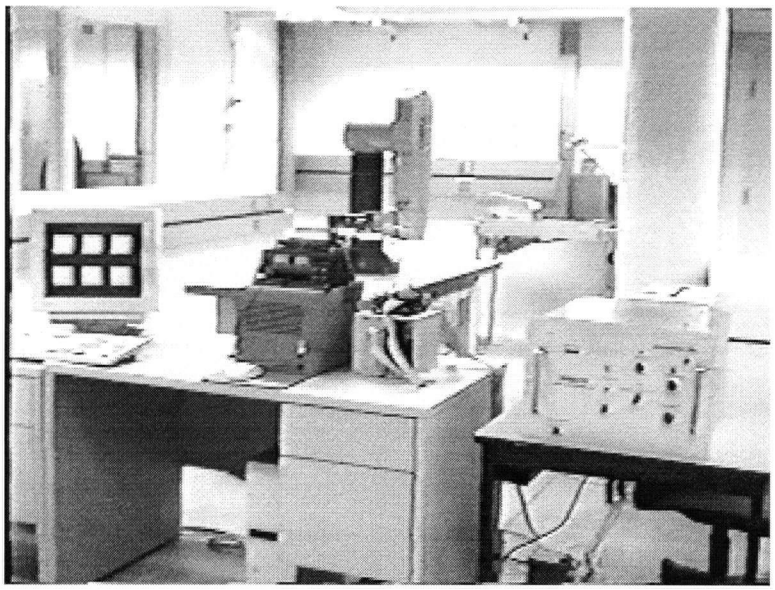


Figure 8.7: Laboratory view of the robotic fish cutting system controlled by ROACS.

whether dominated by dissipative, elastic, or inertial components, also would indicate the nature of the processed material. In particular, cutting through a bone is associated with comparatively high dissipative impedance. This online data will provide processing information for the task planner; for instance, to withdraw the cutter or change the cutting direction, when a main bone is reached.

A lateral cutting process removes the fish slices from the backbone (See Figure 8.9). This process uses the recorded end positions of the previous process of slicing. Again, position control only is not adequate for proper performance. The cutter (position and orientation) is adjusted according to the sensed mechanical impedance at the cutter-object interface, in order to carry out lateral removal of the steaks without cutting into the backbone.

Figure 8.10 shows a set of salmon steaks that have been produced by our impedance-controlled robotic fish cutter, under control of ROACS.

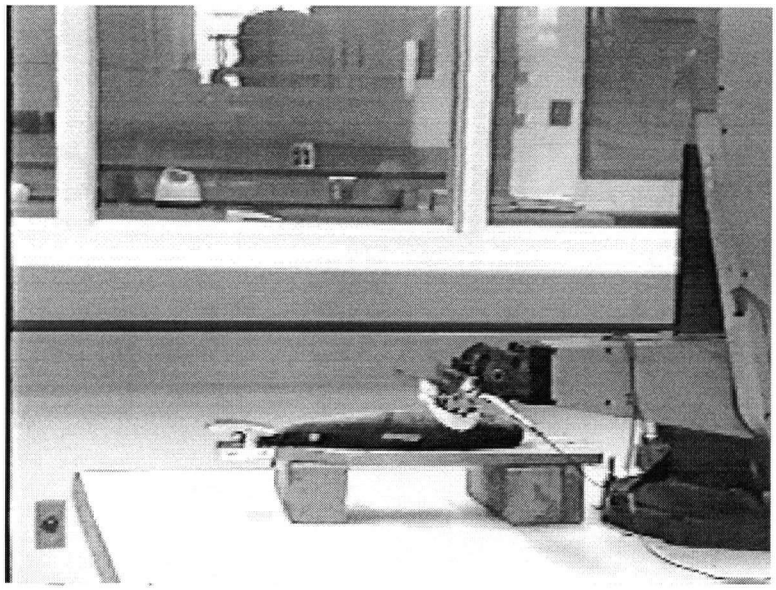


Figure 8.8: A close-up view of the robotic salmon slicing unit.

8.6 Summary

Implementation of ROACS was discussed in this chapter. The QNX real-time operating system was selected for the development platform. The microkernel architecture and the network-transparent message passing function of QNX make it very suitable for intelligent robot control.

The prototype of ROACS which has been developed in the course of the present research, was described in this chapter. This system has a client-server architecture, where the Control Server is attached to a POSIX real-time timer and self-activated for periodic sampling and control tasks. The Path Planner and the Data Analyzer cooperate with the Control Server and communicate with client software.

Client software was implemented in Photon windowing system, which contains an intelligent task controller, an information updating module, a task command shell, and a graphic display of joint data. The command shell allows the operation of a robot programmable.

The developed system has proved its satisfactory real-time performance, and adequate

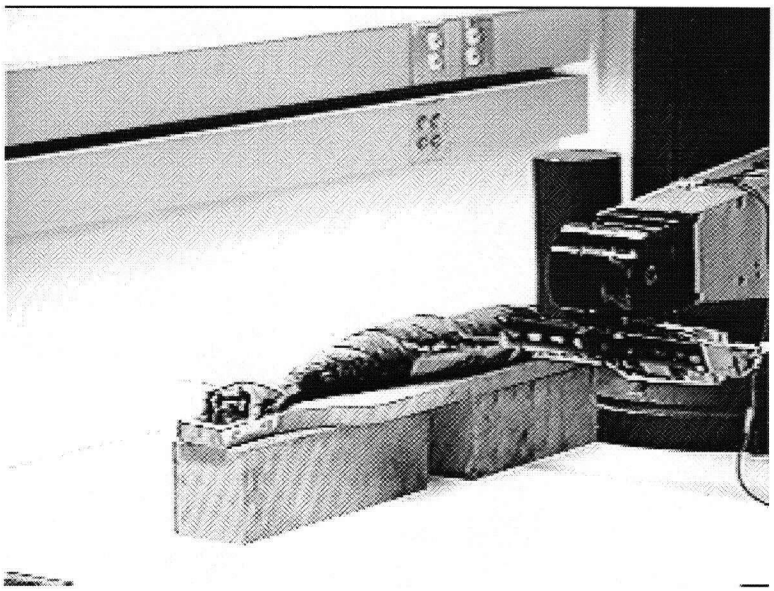


Figure 8.9: Separation process of salmon steaks.

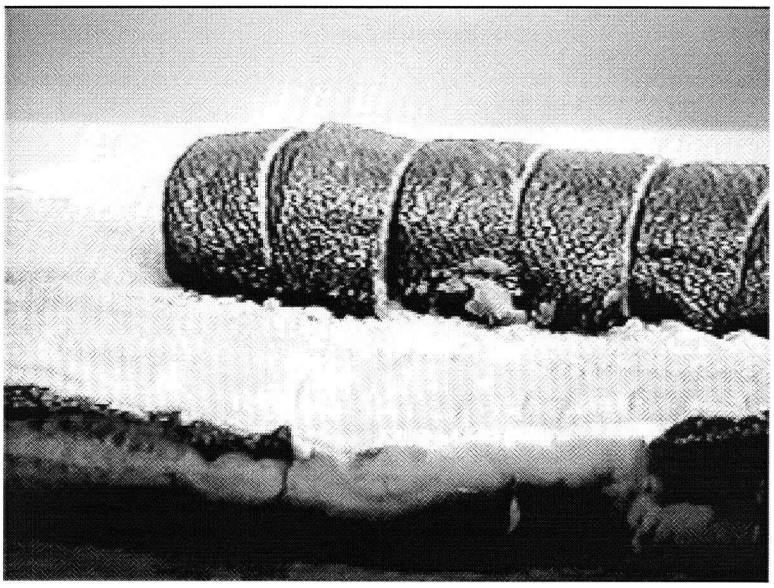


Figure 8.10: Sliced boneless salmon steaks.

flexibility in actual operation during laboratory experiments. Operation data of the robot system can be fed back to the task level. Therefore, intelligent control of robotic systems for application in automated processes that are rely on sensor information can be implemented efficiently in ROACS. The ROACS system was designed and demonstrated to provide high flexibility of operation, improved performance, and convenient enhancement by the user.

Chapter 9

Conclusion

In this concluding chapter, the main contributions of the research will be outlined, their significance will be indicated, and some limitations of the accomplishments and possible directions of further work will be pointed out. The organization of the chapter will be as follows: Section 9.1 will list the major contributions of the present research. Section 9.2 will discuss the rationale and significance of the research, where the work will be briefly justified and the importance of the work will be indicated. Section 9.3 will address the limitations of the control system (ROACS) which has been developed in the present work, and recommendations will be made for future work, for overcoming these limitations.

9.1 Major Contributions

The main objectives of the present research have been to investigate, develop, and implement technologies of flexible robotic manipulation, along with the capability to utilize mechanical impedance information at the process interface. These objectives have been accomplished. A real-time open-architecture robot control system (ROACS) has been designed, developed and implemented. To demonstrate the utility and performance of this system in flexible manipulation, a salmon slicing task was developed and tested in laboratory. This application is particularly suitable for testing the control system as it requires sensor-based task control, and benefits from online sensing and control of mechanical impedance at the cutting interface.

The controller of commercial robots are by and large "closed" to the user, lacks adequate flexibility and intelligence, and primarily incorporate motion sensing and control. It is in this

context of practical shortcomings of industrial robots that important contributions have been made through the present research. These contributions are outlined below in several categories.

1. Analytical formulation of robot dynamics and control in a form that is particularly suitable for achieving the identified objectives of the research.

2. Modelling and online identification of mechanical impedance at the robot process interface.

3. Design and development of an open-architecture, flexible, hierarchical, multi-mode, intelligent control system for robots in the applications of industrial automation.

4. Implementation of a customized, open-system controller for a PUMA 560 robot.

5. Prototyping and practical demonstration of an advanced, real-time, open-architecture control system (ROACS) for robotic applications.

These main contributions are highlighted as follows:

The conceptual design and development of flexible manipulation that is based on mechanical impedance information, has been accomplished. Flexible manipulation, which handles task uncertainties by means of feedback of sensor information at the task control level, has shown significant advantages over the traditional, position-based, hard manipulation in processing flexible and inhomogeneous natural materials. Position uncertainties, which may occur due to inaccurate measurement or estimation, material deformation, slipping, and improper handling during processing, can be reduced by using run-time impedance information at the process interface.

The first requirement of implementing impedance-based flexible robot manipulation is to obtain the impedance information online using normal operation data. In accomplishing this, an appropriate analytical formulation for a multi-degree-of-freedom robot has been made. A

control strategy of linearizing feedback along with task separation of gross motion and fine manipulation, has been formulated. A model for mechanical impedance at the process interface of robot has been developed and interpreted using inertial, elastic, and dissipative modes. Three approaches have been developed for online estimation of the mechanical impedance at the robotic process interface, based on the analytical formulation of robot dynamics and control. The technologies of sensing and estimation of mechanical impedance at the process interface of a robot by detecting only the joint motion signals and actuator efforts have been developed, implemented, and demonstrated in the research. This approach of internal sensing, which does not incorporate explicit force sensors, is reliable and cost-effective. Further advantages can be found in that the internal-sensing approach is easy to implement and use; in particular, the payload of the robot is not compromised because heavy sensors are not added for the purpose.

A sensor-based, five-layer, hierarchical control system has been developed to accommodate the new technologies of flexible manipulation. In this system, the servo modules occupy the bottom level and the intelligent task control is placed at the top. A distinctive feature of this architecture is the feasibility of information feedback at every level. Data can be processed and interpreted (abstracted) in appropriate formats for use in different levels. In the middle level, paths of robot motion are planned online according to the operation status of the particular task. Approaches have been developed for knowledge-based ("intelligent") interpretation of impedance information to be used in process monitoring and control. Task uncertainties at the top-level task controller can be reduced by online recognition of material characteristics using knowledge-based decision making, and feedback of this high-level information into the control system.

The intelligent task control, at the top level of the hierarchy, is a novel design for a robot task control system. Robot tasks are represented in a descriptive manner. Unknown variables are incorporated to handle uncertainties. Task priorities are employed to maintain a priority-driven control procedure instead of an order-driven one. A general method of task representation and

reasoning has been designed. An associated real-time task language has been designed and implemented.

As a general implementation model, a real-time open-architecture control system, or ROACS, has been developed and prototyped for facilitate the implementation of the technologies developed in the present research. ROACS adopts a co-operative real-time intelligent control model. Real-time servo control system has been developed as active servers and the high-level intelligent task controller has been designed as a set of client modules.

The implementation of the ROACS model in the present research has been based on the QNX real-time operating system. To accommodate a flexible software design, the system hardware has been designed in an open mode which incorporates software triggered general I/O boards. The commercial robot controller of PUMA 560 has been customized for this purpose. The hardware simplicity and software flexibility make the system open for further modification and advanced development.

The Puma 560 robot incorporating the ROACS control system, which has been developed in the present research, is operational in the Industrial Automation Laboratory. The system has two computer nodes: the servo node and the task and GUI node. An application of salmon slicing of boneless steaks has been developed and successfully carried out using the developed system.

9.2 Rationale and Significance of the Research

The present research was motivated by several shortcomings of industrial robots. Notable are the following:

1. The low-level controller of a typical industrial robot is neither accessible nor directly modifiable or programmable by the user. Hence, user-developed advanced (and task specific) control schemes cannot be implemented.

2. Motion control using independent feedback of joint position is the commonly used control approach of an industrial robot. This approach has serious limitations, particularly in fine manipulation and mechanical processing tasks which may involve small motions and large forces, and deformable and inhomogeneous objects.

3. The robot control system typically does not have the capability for high-level interpretation (abstraction) of low-level sensory information, knowledge-based evolution of the task performance, information feedback at high level, and general "intelligent" control.

These shortcomings have made industrial robots somewhat ineffective in performing advanced processing tasks in flexible manipulation systems.

Flexible manipulation is based on online sensing of process information, interpretation of such information, intelligent decision making using process knowledge, and improvement of performance through high-level feedback of process information. A system that has these capabilities are particularly suitable for robotic processing of flexible, inhomogeneous, and natural objects. Tasks that involve such objects normally possess position uncertainties in task specifications due to inaccurate measurement or estimation of process information, material deformation, poor handling of the object and unknown disturbances. The position specifications are essential in traditional gross/hard manipulation. In flexible manipulation, however, task uncertainties can be resolved by using intelligent decision making based on knowledge about the process itself along with the online operation information. Mechanical impedance at the process interface of the robot end-effector and the processed object is found very useful in task description in fine/flexible manipulation; hence can be used in lieu of position in task specifications and control schemes. The technologies developed in the present research represent a significant step forward in this context. In particular, they will directly accommodate accurate robot manipulation of flexible, inhomogeneous, and natural objects, and will improve the quality of robot manipulation and CNC machining of such processes through monitoring

and high-level intelligent control.

The development of impedance-based flexible manipulation which incorporates impedance sensing and interpretation for task monitoring and intelligent control, is a significant contribution of the research. The internal sensing methods which use only joint motion signals and actuator efforts, were shown to be reliable, robust, environment-friendly, and cost-effective. The analytical formulation of the robot dynamics, control, and process impedance that has been given in the research was key to accomplishing online sensing of impedance and control in fine manipulation. The development of a hierarchical sensing and control structure with feedback at all layers, has made the practical implementation of the robot control system possible. A novel design of intelligent task control system that was accomplished in the present research easily accommodates the online feedback of run-time information and intelligent decision making. The basic idea in the prototyping of a real-time open architecture control system (ROACS), as carried out in the research, is to move the complexity of the system design to software, and to make the system hardware as simple and general as possible. In this manner the system will be portable to other platforms. The modular design of the software, as incorporated here, is suitable for handling the complexity of a general hierarchical system, and makes the system open to further development and performance enhancement.

9.3 Limitations and Future Work

A practical problem of the current robot system is the high friction in the joints of PUMA 560. Friction, especially nonlinear stiction, hinders accurate estimation of the process force, although the change in force is more important in the control of some classes of robotic tasks. The accurate estimation of the force that is present at the end-effector is useful in monitoring the process load in such applications as robot manipulation, CNC machining, and excavator operations. Issues of mechanical design of a robot; particularly the drive system including

the transmission mechanism and associated gear ratios, have an important influence in the accurate sensing of process forces. For flexible manipulation using internal sensing methods, a robot should be designed to have a low gear ratio, low backlash, and low friction; preferably, a direct-drive system, such that it will be sensitive to external forces and associated sensory information.

Further improvement of the algorithms for impedance estimation would be desirable. The method of identification of the impedance parameters should be more completely extended to the case of multiple DOF systems. In particular, since the impedance matrix at the process interface may be coupled in general, either the transformation that will decouple the impedances should be generated or, alternatively, coupling impedances should be estimated. The sequential search method of intelligent control that is used in the current system is not suitable for handling large task-bases and databases. The simple sequential search has been implemented in the present system only to verify the basic concepts of intelligent task control. More efficient search methods should be developed and employed in practical applications. Another future task is to generalize the ROACS system to handle not only impedance events, but also other robotic processing events.

Bibliography

[1] *ACL: Advanced Control Language Reference Guide and ATS: Advanced Terminal Software,* (1991). Eshed Robotec Inc., Princeton, NJ.

[2] Abidi, M.A., and R.C. Gonzalez (eds.) (1992). *Data Fusion in Robotics and Machine Intelligence.* Academic Press, San Diego, CA.

[3] An, C.H., C.G. Atkeson, and J.M. Hollerbach (1988). *Model-Based Control of a Robot Manipulator.* The MIT Press: Cambridge, Massachusetts.

[4] Armstrong-Helouvry, B., and B. Amin (1994). PID control in the presence of static friction: exact and describing function analysis. *Proc. 1994 American Control Conference*, Baltimore, MD, pp. 597-601.

[5] Armstrong, B., O. Khatib, and J. Burdick (1986). The explicit dynamic model and inertial parameters of the Puma 560 arm. *Proc. IEEE Int. Conf. Robotics & Automation*, Vol. 1, pp 510-518, San Francisco, CA.

[6] Asada, H., and J.E. Slotine (1986). *Robot Analysis and Control.* John Wiley & Sons, Inc., NY.

[7] Astrom, K.J., and B. Wittenmark (1990). *Computer Controlled Systems: Theory and Design.* Prentice Hall, Inc., Englewood Cliffs, NJ.

[8] Bench-Capon, T.J.M., (1990). *Knowledge Representation : An Approach to Artificial Intelligence.* Academic Press, San Diego, CA.

[9] Bicchi, A., J.K. Salisbury, and D.L. Brock (1993). Contact sensing from force measurements. *Int. J. Robotics Research*, Vol. 12, No. 3, pp. 249-262.

[10] Brady, M., J.M. Hollerbach, T.J. Johnson, T. Lozano-Perez, and M.T. Mason (eds) (1984). *Robot Motion: Planning and Control.* The MIT Press, Cambridge, MA.

[11] Brady, M. (ed) (1989). *Robotics Science.* The MIT Press, Cambridge, MA.

[12] Burg, T.C., D.M. Dawson, J. Hu, and P. Vedagarbha (1994). Velocity tracking control for a separately excited DC motor without velocity measurements. *Proc. 1994 American Control Conference*, Baltimore, MD, pp. 1051-1056.

[13] Cai, L., and G. Song (1994). Robust position/force control of robot manipulators during contact tasks. *Proc. 1994 American Control Conference*, Baltimore, MD, pp. 216-220.

[14] Carter, J.P. (1994). Smell/taste: odors in virtual reality. *IEEE Proc. 1994 Intl. Conference on Systems, Man, and Cybernetics*, Vol. 2, pp. 1781-1784.

[15] Chen, Y., M. Caron, V.J. Modi, C.W. de Silva, and A.K. Misra (1998). Dynamic modeling of a variable geometry manipulator with separation of fine and gross manipulations. *Proc. 1998 World Automation Congress,* Vol. 6, pp. 467-474, Anchorage, Alaska.

[16] Chiaverini, S., L. Sciavicco, and B. Siciliano, (1990). *Advanced Robot Control.* Springer-Verlag, Berlin.

[17] Chin, P.C., and K.J. Waldron (1993). Hybrid velocity/force control of manipulators using PID controller and feed forward compensation. *Proc. 2nd IEEE Conference on Control Applications*, Vancouver, BC, pp. 403-408.

[18] Chryssolouris, G., M. Domroese, and P. Beaulieu (1992). Sensor synthesis for control of manufacturing processes. *ASME J. Engineering for Industry*, Vol. 114, May, pp. 158-174.

[19] Chui, C.K., and G. Chen (1991). *Kalman Filtering with Real-Time Applications.* Springer-Verlag, Berlin, Germany.

[20] Colhour, T.G., and S.S. Nair (1994). Accurate estimation of friction. *Proc. 1994 American Control Conference*, Baltimore, MD, pp. 1188-1189.

[21] Corke, P.I. (1994). *The Unimation Puma Servo System.* Technical Report MTM-226, CSIRO Division of Manufacturing Technology, Preston, Australia.

[22] Corke, P.I., and B. Armstrong-Helouvry (1994). A search for consensus among model parameters reported for the Puma 560 robot. *Proc. IEEE Int. Conf. On Robotics & Automation*, pp. 1608-1613.

[23] Craig, J.J. (1986). *Introduction to Robotics: Mechanics and Control.* Addison-Wesley Publishing Company, Reading, MA.

[24] Dawson, D.M., F.L. Lewis, and J.F. Dorsey (1992). Robust force control of a robot manipulator. *Int. J. Robotics Research*, Vol. 11, No. 4, pp. 312-319.

[25] De Silva, C.W (1975). A technique to model the simply supported Timoshenko beam in the design of mechanical vibrating systems. *Int. J. Mechanical Sciences*, Vol. 17, pp. 389-393.

[26] De Silva, C.W. (1989). *Control Sensors and Actuators.* Prentice Hall, Inc., Englewood Cliffs, NJ.

[27] De Silva, C.W., and A.G.J., MacFarlane (1989). *Knowledge-Based Control with Application to Robots.* Springer-Verlag, Berlin, Germany.

[28] De Silva, C.W. (1992). Research laboratory for fish processing automation. *Int. J. of Robotics and Computer-Integrated Manufacturing,* Vol. 9, pp. 49-60.

[29] De Silva, C.W. (1993). Soft automation of industrial processes. *Int. J. Engineering Application of Artificial Intelligence*, Vol. 6, No. 2, pp. 87-90.

[30] De Silva, C.W. (1994). Automation intelligence. *Int. J. Engineering Applications of Artificial Intelligence*, Vol. 7, No.5, pp. 471-477.

[31] De Silva, C.W. (1995). *Intelligent Control: Fuzzy Logic Applications.* CRC Press, Boca Raton, FL.

[32] De Silva, C.W. (1999). *Vibration: Fundamentals and Practice.* CRC Press, Boca Raton, FL.

[33] De Silva, C.W., and J. Gu (1994). An intelligent system for dynamic sharing of workcell components in process automation. *Int. J. Engineering Applications of Artificial Intelligence,* Vol. 7, No.5, pp. 571-586.

[34] De Silva, C.W., and J. Gu (1995). On-line sensing and modeling of mechanical impedance in robotic food processing. *Proc. IEEE Conference on Systems Man, and Cybernetics*, Vancouver, Canada, Vol. 2, pp. 1693-1698.

[35] De Silva, C.W., and J. Gu (1996). Use of mechanical impedance in sensing object-end effector interactions. *Proc. IEEE Int. Symposium on Industrial Electronics,* Warsaw, Poland, Vol.1, pp. 60-65.

[36] De Silva, C.W., R.G. Gosine, Q.M. Wu, N. Wickramarachchi, and A. Beatty (1993). Flexible automation of fish processing. *Int. J. Engineering Applications of Artificial Intelligence,* Vol. 6, No. 2, pp. 165-178.

[37] De Wit, C.C., H. Olsson, K.J. Astrom, and P. Lischinsky (1993). Dynamic friction models and control design. *Proc. 1993 American Control Conference*, Vol. 2, pp. 1920-1926.

[38] Dubois, D., and H. Prade (1992). Combination of fuzzy information in the framework of possibility theory. Chapter 12 of book: Abidi, M.A., and R.C. Gonzalez (eds) (1992), *Data Fusion in Robotics and Machine Intelligence,* Academic Press, San Diego, CA.

[39] Erlic, M., and W.S. Lu (1993). Impedance control without using velocity measurements. *Proc. 2nd IEEE Conference on Control Applications*, Vancouver, BC, pp. 47-51.

[40] Fisher, W.D. and M.S. Mujtaba (1992). Hybrid position/force control: a correct formulation. *Int. J. Robotics Research*, Vol. 11, No. 4, pp. 299-311.

[41] Fong, Y.C., (1994). *Biomechanics I.* Kluwer Academic Press, Boston, MA.

[42] Fu, K.S., R.C. Gonzalez, and C.S.G. Lee (1987). *Robotics : Control, Sensing, Vision, and Intelligence.* McGraw-Hill, New York, NY.

[43] Gallmeister, B.O. (1995). *Programming for the Real World: POSIX.4.* O'Reilly & Assoc., Inc. CA.

[44] Grewal, M.S., and A.P. Andrews (1993). *Kalman Filtering: Theory and Practice.* Prentice-Hall, Inc., Englewood Cliffs, NJ.

[45] Gu, J.H. (1994). *Knowledge Based Restructuring of Flexible Production Systems.* M.A.Sc Thesis, Department of Mechanical Engineering, University of British Columbia, Vancouver, Canada.

[46] Gu, J.H. and C.W. de Silva (1994). Fuzzy decision making in variable-structure control systems. *Proc. International Mechanical Engineering Congress, ASME*, Chicago, IL, DSC-Vol. 55-1, pp. 483-492.

[47] Gu, J.H., and C.W. de Silva (1996). Interpretation of mechanical impedance profiles for intelligent control of robotic meat processing. *Proc. IEEE IECON.* Taipei, Taiwan, pp. 507-512.

[48] Gu, J.H., and C.W. de Silva (1997). Client/server architecture for implementing real-time robot control and industrial processes. *Proc. 2nd Asian Control Conference*, Vol III, Seoul, Korea, pp. 259-262.

[49] Gu, J.H., and C.W. de Silva (1998). Use of mechanical impedance in robotic task control. *Proc. 1998 World Automation Congress,* Vol. 6, Anchorage, Alaska, pp. 539-544.

[50] Gu, Y.L., and N.K. Loh (1988). Dynamic modeling and control by utilizing an imaginary robot model. *IEEE J. of Robotics & Automation,* Vol. 4, N0. 5, pp. 532-540.

[51] Haessig, D.A. Jr., and B. Friedland (1991). On the modeling and simulation of friction. *ASME J. Dynamic Systems, Measurement, and Control*, Vol. 113, Sep., pp. 354-362.

[52] Hager, G.D. (1990). *Task-Directed Sensor Fusion and Planning: A Computational Approach.* Kluwer Academic Publishers, Boston, MA.

[53] Hill, G., E. Horstkotte, and J. Teichrow (1991). *Fuzzy-C Development System User's Manual.* Togai Infralogic, Inc., CA, 1988-1991, Release 2.3.0.

[54] Hogan, N. (1985a). Impedance control: an approach to manipulation: part I – theory. *ASME J. Dynamic Systems, Measurement, and Control*, Vol. 107, Mar., pp. 1-7.

[55] Hogan, N. (1985b). Impedance control: an approach to manipulation: part II – implementation, *ASME J. Dynamic Systems, Measurement, and Control*, Vol. 107, Mar., pp. 8-16.

[56] Hogan, N. (1985c). Impedance control: an approach to manipulation: part III – applications. *ASME J. Dynamic Systems, Measurement, and Control*, Vol. 107, Mar., pp. 17-24.

[57] Howe, R.D., and M.R. Cutkosky (1992). Touch sensing for robotic manipulation and recognition. Book: Khatib, O., J.J. Craig, and T. Lozano-Perez (eds), *The Robotics Review 2,* The MIT Press, Cambridge, MA.

[58] Jackson, L.B. (1989). *Digital Filters and Signal Processing.* Kluwer Academic Press, Boston, MA.

[59] Kang, C.G., J.H. Gu, and C.W. de Silva (1998). Efficient trajectory planning for position and orientation of a Puma robot. *Proc. 1998 World Automation Congress*, Vol. 6, Anchorage, Alaska, pp. 467-474.

[60] Khatib, O., J.J. Craig, and T. Lozano-Perez (eds) (1989). *The Robotics Review 1.* The MIT Press, Cambridge, MA.

[61] Khatib, O., J.J. Craig, and T. Lozano-Perez (eds) (1992). *The Robotics Review 2.* The MIT Press, Cambridge, MA.

[62] Klafter, R.D., T.A. Chmielewski, and M. Negin (1989). *Robotic Engineering: An Integrated Approach.* Prentice-Hall, Inc, Englewood Cliffs, NJ.

[63] Kok, R. and V. Harading (1996). Client/server strategies come to the plant floor. *Manufacturing & Process Automation*, Vol.11, No. 4, pp. 12-13.

[64] Lee, Y.T., H.R. Choi, W.K. Chung, and Y. Youm (1994). Stiffness control of a coupled tendon-driven robot hand. *IEEE J. Control Systems,* Vol. 14, No. 5, pp. 10-14.

[65] Lewis, F.L. (1986). *Optimal Estimation.* John Wiley and Sons, New York, NY.

[66] Lewis, F.L. (1992). *Applied Optimal Control and Estimation: Digital Design and Implementation.* Prentice Hall, Englewood Cliffs, NJ.

[67] Maquin, D., B. Gaddouna, and J. Ragot (1994). Estimation of unknown inputs in linear systems. *Proc. 1994 American Control Conference*, Baltimore, MD, pp. 1195-1197.

[68] McCormick, W., and H.M. Schwartz (1993). An investigation of impedance control for robot manipulators. *Int. J. Robotics Research*, Vol. 12, No. 5, pp. 473-489.

[69] Meng, Q.H.M., and W.S. Lu (1994). A unified approach to stable adaptive force/position control of robot manipulators. *Proc. 1994 American Control Conference*, Baltimore, MD, pp. 200-201.

[70] Musliner, D.J., J.A. Hendler, A.K. Agrawala, E.H. Durfee, J.K. Strosnider, and C.J. Paul (1994). The challenges of real-time AI. *U. Maryland Technical Report*, CS-TR-3290, UMIACS-TR-94-69, June, 1994.

[71] Nagata, F., K. Watanabe, K. Sato, and K. Izumi (1998). Position based impedance control using fuzzy environment models. *Proc. 37th SICE Annual Conference,* pp. 837-842.

[72] Nicosia, S., and P. Tomei (1990). Robot control by using only position measurements. *IEEE Trans. Automatic Control*, Vol. 35, pp. 1058-1061.

[73] Pagilla, P., and M. Tomizuka (1994). Hybrid force/position control of two arms carrying one object. *Proc. 1994 American Control Conference*, Baltimore, MD, pp. 195-199.

[74] Pau, L. (1989). Knowledge representation approaches in sensor fusion. *Automatica*, Vol. 25, pp. 207-214.

[75] Paul, R.P., and H. Zhang (1986). Computationally efficient kinematics for manipulators with spherical wrists based on the homogeneous transformation representation. *Int. J. Robotics Research*, Vol. 5, No. 2, pp. 32-44.

[76] Raibert, M.H., and J.J. Craig (1981). Hybrid position/force control of manipulators. *ASME J. Dynamic Systems, Measurement, and Control*, Vol. 103, Jun., pp. 126-133.

[77] Redfield, C.L., R. Noren, D.C. Varner, and K.C. Golas (1994). Hearing: speech recognition. *IEEE Proc. 1994 Int. Conference on Systems, Man, and Cybernetics*, Vol. 2, pp. 1774-1776.

[78] Southward, S.C., C.J. Radcliffe, and C.R. MacCluer (1991). Robust nonlinear stick-slip friction compensation. *ASME J. Dynamic Systems, Measurement, and Control*, Vol. 113, Dec., pp. 639-645.

[79] *System Architecture* (1994). QNX Software System Ltd., Kanata, Canada.

[80] Tafazoli, S., C.W. de Silva, and P.D. Lawrence (1998). Tracking control of an electro-hydraulic manipulator in the presence of friction. *IEEE Trans. on Control Systems Technology*, Vol. 6, No. 3, May, pp. 401-411.

[81] Timmerman, M., and J. Monfret (1997). Windows NT as real-time OS. *Real-Time Magzine*, Q2.

[82] Tou, J.T., and J.G. Balchen (eds) (1990). *Highly Redundant Sensing in Robotics Systems.* Springer-Verlag, Berlin, Germany.

[83] *Unimate Puma Mark II Robot 500 Series Equipment Manual for VAL II and VAL Plus 398U1* (1985). Unimation Co.

[84] *User's Manual: DDA06* (1987). Keithley MetraByte Corporation, Taunton, MA.

[85] *User's Manual: M5312* (1987). Keithley MetraByte Corporation, Taunton, MA.

[86] *User's Manual: PCA6157 CPU Board* (1995). Industrial Computer Resources, San Diego, CA.

[87] *User's Manual: PCL-814B Modulized DAS Card* (1994). Advantech Co., Ltd.

[88] *User's Manual: PMDI Coordinate Motion Library* (1996). Precision Microdynamics Inc.

[89] *User's Manual: Scorbot-ER V* (1991). Eshed Robotec Inc., Princeton, NJ.

[90] Whitney, D.E. (1987). Historical perspective and state of the art in robot force control. *Int. J. Robotics Research*, Vol. 6, No. 1, pp. 3-14.

[91] Yang, Y.P., and J.S. Chu (1993). Adaptive velocity control of DC motors with Coulomb friction identification. *ASME J. Dynamic Systems, Measurement, and Control*, Vol. 115, Mar., pp. 95-102.

[92] Yao, B., S.P. Chan, and D.W. Wang (1994). Variable structure adaptive motion and force control of robot manipulators. *Automatica*, Vol. 30, No. 9, pp. 1473-1477.

[93] Yoshikawa, T. (1990). *Foundations of Robotics: Analysis and Control.* The MIT Press, Cambridge, MA.

Appendix A

User Manual of ROACS

ROACS is a real-time open-architecture robot control system. A client-server architecture has been used for its implementation. Figure A.1 illustrates the physical structure of the system. As shown in the figure, Node 2 runs the low-level control system which is designed as a set of servers. Node 1 usually runs the intelligent task controller with a graphics user interface, which is designed as a client module. The user interface can be remotely displayed on any node in the network; for example Node 3. This appendix provides a manual for ROACS. In particular, Section A.1 and Section A.2 will give the user instructions for the server and client segments, respectively.

A.1 ROACS Servers

A.1.1 Command line

Both the computer system, including Node 1 and Node 2, and the customized robot controller should be powered on in order to run the Puma 560 system with ROACS. Note that the servo controller is assigned to Node 2 in the QNX network of ROACS in the Industrial Automation Laboratory (IAL). To start the ROACS servers, type the following command in a QNX shell on either Node 1 or Node 2,

```
%> onnode 2 Puma560
```

This command will load in the Control Server, which is named *Puma560*. Once loaded, the Control Server will spawn the Path Planner and the Data Server. The communication between

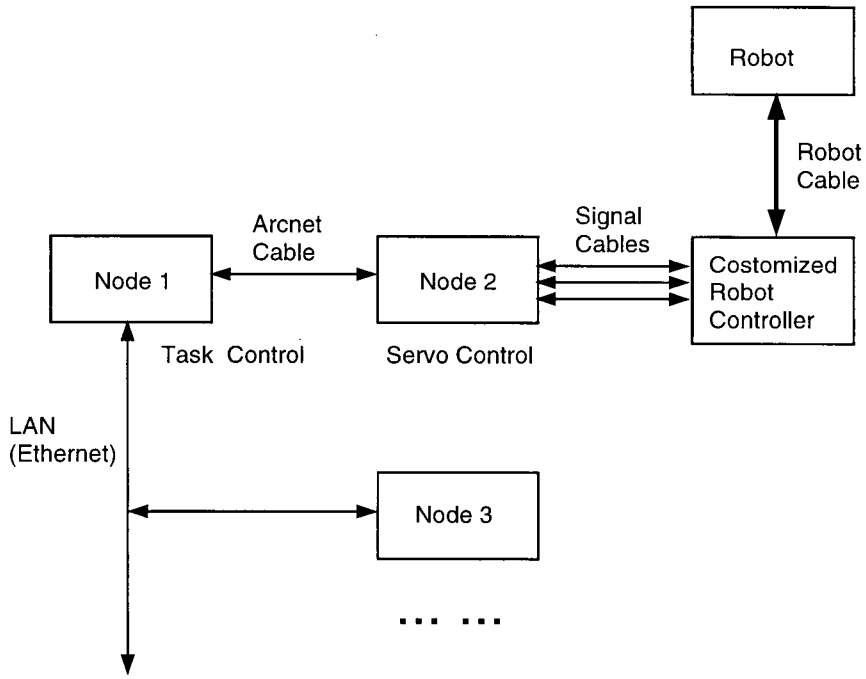


Figure A.1: Physical structure of ROACS.

the servers will be established as well, during this procedure.

A.1.2 System initialization

The ROACS servers display some important information of the system during the initialization process. Check this information to make sure that you have set appropriate parameters for the controller. A typical run-time display should look like,

```
Ticksize=1
System CTRPERIOD=2ms, ticks=2
System SETPERIOD=10ms, RECPERIOD=8ms
Kp=1000.0, 1300.0, 600.0, 800.0, 800.0, 800.0
Kd=10.0, 13.0, 5.0, 6.0, 6.0, 6.0
param->followError=0.02, 0.02, 0.02, 0.03, 0.03, 0.03
```

```
param->initpos=0.0045, 0.0297, 1.5655, -0.0001, 0.0000, -0.0001
hardware=1
PathPlan: PATH_GEN_SAMP_TIME=0.010s
Puma560: pid_pathplan=3205, pid_dataserv=3214,
Puma560: proxy_pathplan=3218, proxy_dataserv=3219
DataServ: recording period=0.008
DataServ: pid_controller=3161
DataServ: pid_pathplanner=3205, pid_controller=3161,
DataServ: proxy_stop=3224, proxy_dataserv=3219
PathPlan: pid_controller=3161, proxy_pathplanner=3218,
PathPlan: pid_dataserv=3214, proxy_stop=3224.
PathPlan: Original aStopMax=2.0
```

Here, the system parameters are displayed in the order of the execution of the programs, but can be grouped into the following four parts:

- Control period - In this part, the system displays the *ticksize* in milliseconds, which is automatically optimized for the given control period by the ROACS control server; the number of ticks per control period; and the control period, which is the nearest value to the given control period and fits the selected tick-size. The periods for path planning and data analysis are given based on the selected servo control period, since they should be integer-multipliers of the servo control period. In the above example, the *ticksize* is 1 millisecond, the control period is 2 milliseconds — 2 ticks per control period, and the periods of path planning (PATH_GEN_SAMP_TIME) and the data analysis (recording period) are 0.01 seconds and 0.008 seconds, respectively.

- Communication handles - The Control Server *Puma560* will spawn the Path Planner *PathPlan* and the Data Analyzer *DataServ* once *Puma560* is loaded in. The communication between the servers is established at that moment. The process IDs and the proxy IDs will be displayed from all three processes. On successful establishment of communication, the process IDs that are displayed from different processes should be consistent. As seen from the example, the process IDs of the controller *Puma560* (pid_controller), the path planner *PathPlan* (pid_pathplanner), and the data analyzer *DataServ* (pid_dataserv) are 3161, 3205, and 3214, respectively. These numbers are consistent in all reports. The continue-proxy (proxy_pathplan) is 3218, which is the same as what are reported by both *Puma560* and *PathPlan*. The stop-proxy (proxy_stop) holds the ID number of 3224. This is reported by both *DataServ* and *PathPlan*. Another proxy is the notification of new data from *Puma560* to *DataServ*, which has the ID number 3219.

- Hardware information - A flag of hardware presence and the parameters of the initial positions of the robot should be displayed. The hardware flag indicates if the robot system is present: *hardware*=1 for yes and *hardware*=0 for no. The initial positions of the robot joints are displayed in *initPos*, which were saved when the system exit of the last operation (The robot is assumed the same configuration after operation, which is the usual case). This data are read from a data file named *puma.dat*. In this example, the initial joint positions are {0.0045, 0.0297, 1.5655, -0.0001, 0.0000, -0.0001}.

- Control parameters - *Kp* and *Kd* are the parameters of the servo PD controller. For ROACS, the *Kp* parameters are set as {1000.0, 1300.0, 600.0, 800.0, 800.0, 800.0}, and the *Kd* parameters are set as {10.0, 13.0, 5.0, 6.0, 6.0, 6.0}, for the six joints. The default acceleration for jogging is set as *aMaxStop* initially, which can be changed during operation, through task programming. In this example,

aMaxStop=2.0 rad/s^2. For safety protection, the limits of trajectory-following errors *followError* are also given. Whenever a trajectory-following error exceeds its limit, the system will shutdown. The limits are given as {0.02, 0.02, 0.02, 0.03, 0.03, 0.03} in radians.

A.1.3 Setup parameters

The parameters of system initialization are read from a text file named *puma.ini*. This file, in the present system, is shown below:

```
# This file contains initialization parameters.
# Minimum sampling rate is 0.2ms, default sampling rate is 1.0ms.
# <-(0.2, 0.3, 0.4, 0.5, 1.0, 2.0, 3.0, .....).
CTRPERIOD 2.0
SETPERIOD 10.0
RECPERIOD 8.0
# Kp and Kd for PD control
Kp  =  {1000.0; 1300.0; 600.0; 800.0; 800.0; 800.0}
Kd  =  {10.0; 13.0; 5.0; 6.0; 6.0; 6.0}
# Limits of the joint positions
Limit_posP = { 4.0; 4.36; 4.36; 2.62; 1.74; 4.64}
Limit_posN = { -4.0; -1.22; -1.22; -2.62; -1.74; -4.64}
# Limits of the joint currents
Limit_crt = { 4.9; 4.9; 4.9; 4.0; 4.0; 4.0}
# Torque constant of joint motors
Torque_const = { 9.76; 19.4; 8.95; 2.4; 2.0; 2.13}
# Friction (stiction) in positive directions
```

```
Friction_s = { 0.2; 0.2; 0.1; 0.3; 0.2; 0.2}
# Friction (stiction) in negative directions
Friction_s_ = { -0.2; -0.2; -0.1; -0.3; -0.3; -0.3}
# Friction (Columb) in positive directions
Friction_c = { 0.2; 0.2; 0.1; 0.3; 0.2; 0.2}
# Friction (Columb) in negative directions
Friction_c_ = { -0.2; -0.2; -0.1; -0.3; -0.2; -0.2}
# Limits of following errors (in radius)
FOLLOW_ERROR ={ 0.02; 0.02; 0.02; 0.03; 0.03; 0.03}
# Hardware presence
HARDWARE yes
#end
```

Here, each line defines an item in the format,

Keyword value #comment

where the *Keyword* can be one of the following terms:

- CTRPERIOD: The sampling and servo control period in milliseconds. The minimum control period is 0.2ms. The default is 1.0ms. The exact time can be rounded off to fit the tick-size of the system.
- SETPERIOD: The path planning period. It has to be an integer multiplier (1, 2, 3, ...) of CTRPERIOD.
- RECPERIOD: The period of data recording and data analysis. It has to be an integer multiplier (1, 2, 3, ...) of CTRPERIOD.
- KP: The proportional gains of the PD servo controller. The *value* should be a vector of six elements for joints 1 to 6 given in braces.

- KD: the derivative gains of the PD controller. The *value* should be a vector of six elements for joints 1 to 6 given in braces.

- Limit_posP: Joint limits in the positive directions. The *value* should be a vector of six elements for joints 1 to 6 given in braces.

- Limit_posN: Joint limits in the negative directions. The *value* should be a vector of six elements for joints 1 to 6 given in braces.

- Limit_crt: Limits of the joint motor currents. The *value* should be a vector of six elements for joints 1 to 6 given in braces.

- Torque_const: Torque constants of the joint motors. The *value* should be a vector of six elements for joints 1 to 6 given in braces.

- Friction_s: The stiction of the joints in the positive directions. The *value* should be a vector of six elements for joints 1 to 6 given in braces.

- Friction_s_: The stiction of the joints in the negative directions. The *value* should be a vector of six elements for joints 1 to 6 given in braces.

- Friction_c: The Coulomb friction of the joints in the positive directions. The *value* should be a vector of six elements for joints 1 to 6 given in braces.

- Friction_c_: The Coulomb friction of the joints in the negative directions. The *value* should be a vector of six elements for joints 1 to 6 given in braces.

- FOLLOW_ERROR: The limits of the trajectory-following error that is allowed for joint motion. If any one of the trajectory-following error values exceeds the corresponding limit, the system will take it as a fatal error and will shutdown the machine. The *value* should be a vector of six elements for joints 1 to 6 given in braces. Note that, for rigid

motion, the error limits should be set small, and for impedance-based operation, the limits should be set relatively large.

- HARDWARE: The logic flag to inform the system if the control hardware is present. In case of a test of high level intelligent control, the system may run without hardware.

A.1.4 Communication handles

The communication handles of the ROACS servers are the registered global names. The names of the Control Server, the Path Planner, and the Data Analyzer are "/robot/control", "/robot/pathplan", and "/robot/dataserv", respectively. The client software should set communication by allocating the servers using their names.

A.2 Intelligent Task Controller

A.2.1 Command line

The client module that has been developed with ROACS servers is *Client*. It is a Photon program which requires Photon environment for execution. To run *Client*, click on the *Client* icon in the Photon desk top manager, or type the following shell command:

```
%> Client
```

Once loaded, *Client* will allocate ROACS servers, and establish communication between the client software and the ROACS servers. *Client* will quit if such communication cannot be established.

The *Client* program can run at any node in the QNX network, as long as it can establish a direct connection to the servo node (Node 2). All QNX machines in a local area network (LAN) are communication transparent. Communication between machines that are connected by different LANs should be forwarded by a gateway that is cross connected in multiple LANs.

All Ethernet message packets are forwardable, which results in network transparency of all Ethernet connected QNX machines. If the servo node is fast enough; e.g., Pentium II 400MHz, *Client* may run on the same node with the servers.

The *Client* program will bring up a graphics user interface. The initialization procedure will be carried out quietly. Communication to the ROACS servers is performed by allocating the servers in the network.

A.2.2 Graphics display

The graphics user interface of the ROACS system is shown in Figure A.2.

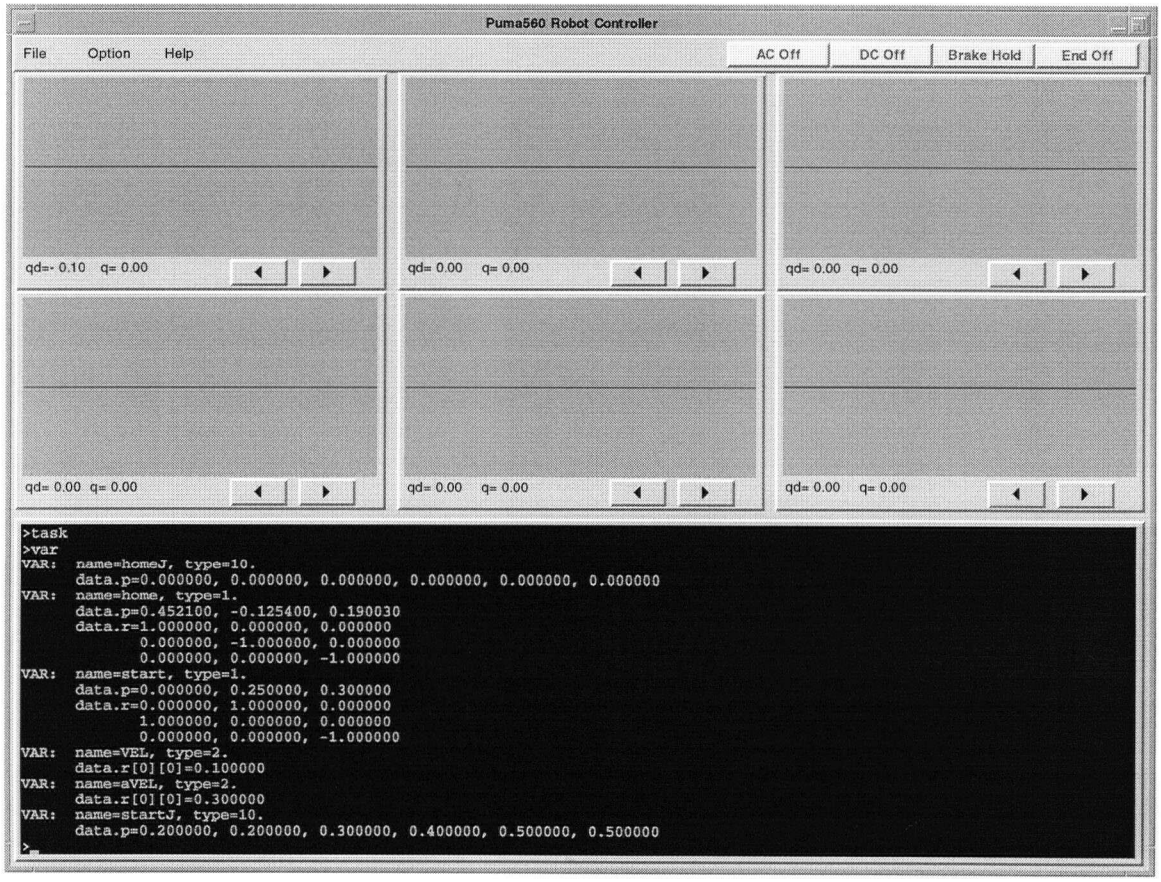


Figure A.2: The graphics user interface of ROACS.

There are six real-time trend windows for displaying the signals. Each of the six trend windows can display: desired position *dpos* (red), actual position *pos* (green), velocity *vel* (cyan), current *crt* (yellow), control signal *ctr* (blue), and impedance *imp* (brown) as options for each joint. Options can be selected on line, in an option dialog box. The flags at the top right corner of the display indicate the system status (on/off) for AC power, DC power, the robot brakes, and the end-effector.

There are three menus at the top left corner: "File", "Option", and "Help". The "File" menu has two items: "Load Task" and "Exit System". Selection of "Exit System" in this menu will quit the entire ROACS system. The item "Load Task" will allow the user to use a file-selection box for loading a task file into the task base. The file-selection box is shown in Figure A.3.

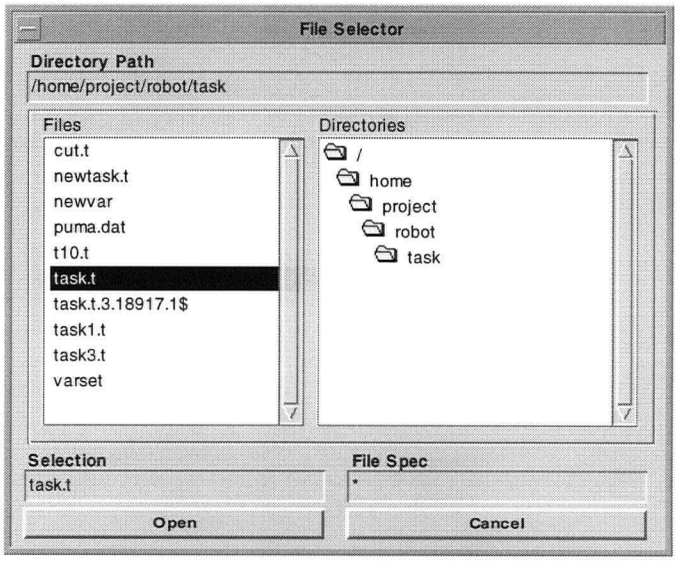


Figure A.3: The file selection box in ROACS.

The "Option" menu also provides two items: "RtTrend Options" and "Joint Control Option". The item "RtTrend Options" brings up a dialog box that contains two sliders for selection of the numbers of trends and the display-sampling rate (see Figure A.4). The item "Joint Control Option" also brings up a dialog box that contains a slider for changing the jogging speed (see Figure A.5).

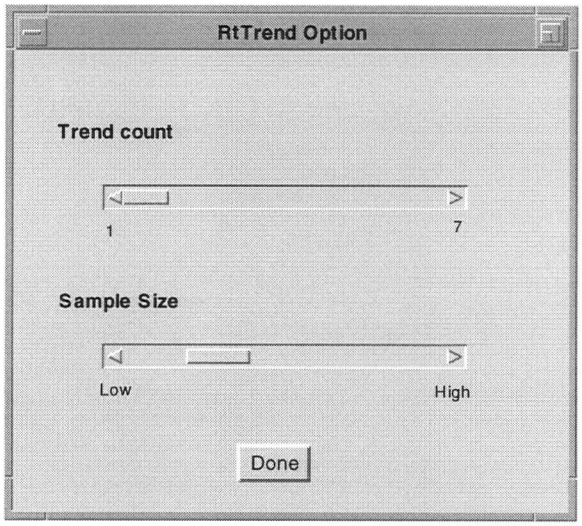


Figure A.4: The RtTrend option box in ROACS.

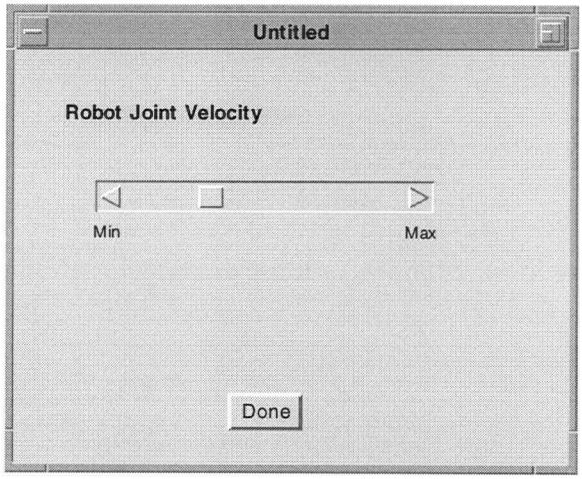


Figure A.5: The joint-control option box in ROACS.

The "Help" menu provides on-line help by the item "HTML On-line Help" and the information about the ROACS system itself by the item "About". The "about" dialog box is shown in Figure A.6.

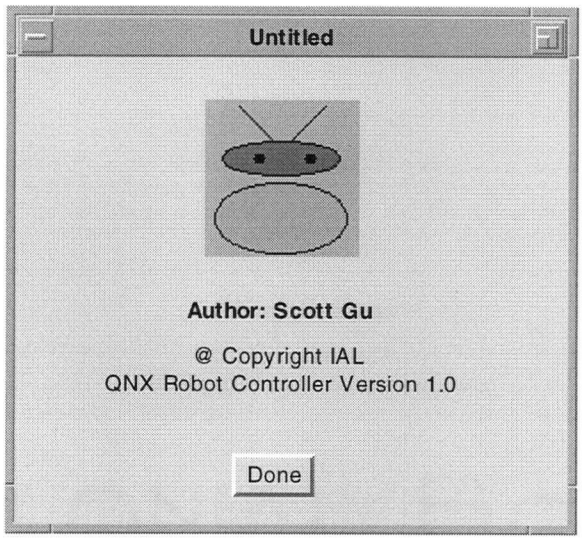


Figure A.6: A view of the *About* dialog.

The on-line help is an HTML browser; thus, all on-line help files should be written in the HTML format. The help window is shown in Figure A.7. The buttons in the help window have the same functions as in the other HTML browsers: "Home" to go to a fixed home link; "Back" to trace back a link; "Help" to show help information of the on-line helper; and "Close" to close the help window.

The arrow buttons under the trend windows (see Figure A.2) allow direct and interactive motion control of the robot joints. A single click on a button will move the corresponding joint through a small angle, which is 0.02 radians in the present system. Holding down a button will move the corresponding joint continuously, upon releasing the button. Joint positions, both reference and actual values, are displayed as text. More and complicated operations may be carried out through the task shell, where robot tasks can be programmed using RTTL.

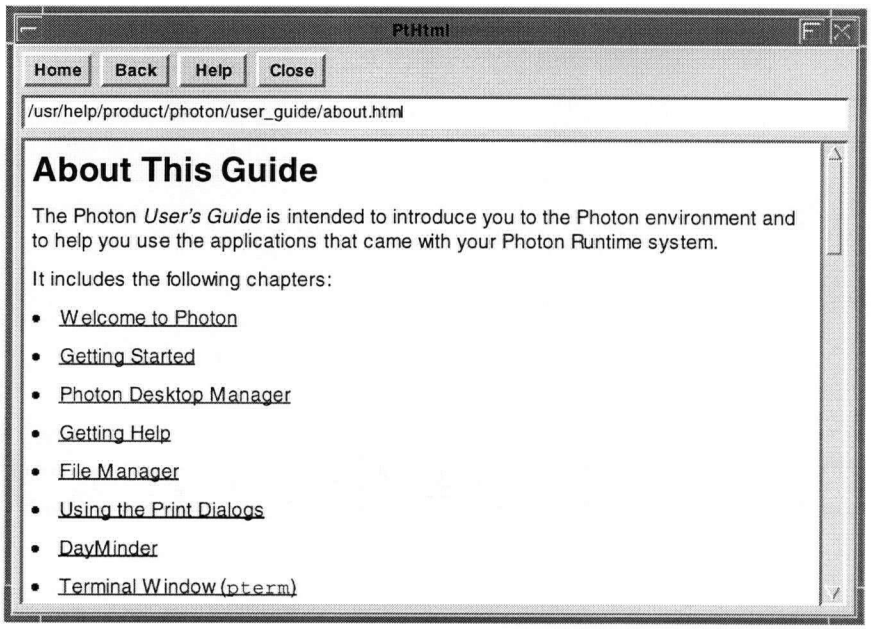


Figure A.7: A typical display of the help window.

A.2.3 Task shell commands and programming

The task shell, as seen from Figure A.2, is text based. Commands can be typed in at the prompt, or loaded in from a task program. A task program is a plain text file containing the RTTL commands. To load a file, user may type the following command at the shell prompt (another option is to use the "File" menu):

```
>load filename.t;
```

where *filename.t* is the file to be loaded. The extension "*.t*" is not necessary, but incorporated here to stand for "task".

The task engine has three modes: "run", "step", and "pause". It is initialized to mode "pause". The command *run* will start the task engine for continuous operation. The command line is,

```
>run;
```

In this mode, the task engine will repeatedly search for the most appropriate task in the task base and will send it to the ROACS servers during every task-control period, which is 100 milliseconds in the present system. The command *step* will run the task engine in a step mode. The command line is,

```
>step;
```

In the *step* mode, the task engine performs only one step of task searching and delivery (to the ROACS servers). Then it waits for the next *step* command. The steps that follow may be simplified to the single keyboard strike *F10*. The third command of mode-control is *pause*, which has the command line,

```
>pause;
```

The *pause* command will turn the task engine to the *pause* mode from the *run* or the *step* mode.

Operation data can be saved in a file in plain text format, which can be read by other data processing programs such as Excel and Matlab. The associated command is,

```
>save filename;
```

This command will save the operation data, which are resampled at period RECPERIOD, in files *filename.n* with the extension *n*, indicating the channel numbers 1 to 6. For example, *filename.1* would contain the operation data about joint 1. To stop saving data, type,

```
>save off;
```

Note that, *save off* should not be missed in a user's program if a *save* command is present. Failing to do so may result in disk overflow. Shell variables can also be saved in a plain text file by using a different command, which has the command line,

```
>store filename;
```

The file *filename* stores all the shell variables in the format of *set*. This file can be reloaded subsequently to continue the interrupted task.

The command *exit*,

```
>exit;
```

will quit the entire system. ROACS will save the joint positions in the data file *puma.dat*, which will be used in the homing procedure of the subsequent operation.

Appendix B

Functions of Server Interface Library

The ROACS server-interface library *robotlib* provides convenient functions for users to build their own customized client modules, or to enhance the ROACS intelligent task-control module *Client* by adding new client modules. The functions in the *robotlib* library handle the details of communication between the clients and the servers. This communication uses the message passing methods of QNX.

Data structures are defined in *server.h* and functions are defined in *robotlib.h*. The functions of *robotlib* are itemized below:

- **initServer**

  initialize the communication links to the ROACS servers.

  Synopsys:

  #include <server.h>

  #include <iolevel.h>

  int initServer();

  Description:

  The function *initServer()* sets up the communication handles to the Path Planner and the Data Analyzer. This function allocates the servers by their global names over the entire QNX network; thus the communication links are network transparent. The developed clients can run on the same node as the servers, or any other node in the network.

  Returns:

  RET_OK on success. On error, a value of (-1) is returned, and the *errno* is set. The

calling process will exit if the communication cannot be established.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner, Data_Analyzer

- **robotAC_ON**

turn on AC power of the robot control system.

Synopsys:

#include <server.h>

#include <iolevel.h>

int robotAC_ON();

Description:

The function *robotAC_ON()* sends the command *AC_ON* to the Path Planner, which in turn transfers the command to the Control Server. Once received, the Control Server will call the *iolevel* (see Appendix C) function *AC_power_on()* to turn on the system AC power.

Returns:

RET_OK on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner, Control_Server

- **robotAC_OFF**

turn off AC power of the robot control system.

Synopsys:

#include <server.h>

#include <iolevel.h>

int robotAC_OFF();

Description:

The function *robotAC_OFF()* sends the command *AC_OFF* to the Path Planner, which then transfers the command to the Control Server. Once received, the Control Server will call the *iolevel* function *AC_power_off()* to turn off the AC power of the system.

Returns:

RET_OK on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner, Control_Server

- **robotDC_ON**

turn on DC power of the robot arm.

Synopsys:

#include <server.h>

#include <iolevel.h>

int robotDC_ON();

Description:

The function *robotDC_ON()* sends the command *DC_ON* to the Path Planner, which then transfers the command to the Control Server. Once received, the Control Server will

call the *iolevel* function *DC_power_on()* to turn on the power of the robot arm.

Returns:

RET_OK on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner, Control_Server

- **robotDC_OFF**

turn off DC power of the robot arm.

Synopsys:

#include <server.h>

#include <iolevel.h>

int robotDC_OFF();

Description:

The function *robotDC_OFF()* sends the command *DC_OFF* to the Path Planner, which then transfers the command to the Control Server. Once received, the Control Server will call the *iolevel* function *DC_power_off()* to turn off the power of the robot arm.

Returns:

RET_OK on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner, Control_Server

- **robotBRAKE_RELEASE**

  release the brakes of the robot arm.

  Synopsys:

  #include <server.h>

  #include <iolevel.h>

  int robotBRAKE_RELEASE();

  Description:

  The function *robotBRAKE_RELEASE()* sends the command *BRAKE_RELEASE* to the Path Planner, which then transfers the command to the Control Server. Once received, the Control Server will call the *iolevel* function *brake_release()* to release the brakes of the robot arm.

  Returns:

  RET_OK on success. On error, a value of (-1) is returned, and the *errno* is set.

  Errors:

  ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

  Class:

  Path_Planner, Control_Server

- **robotBRAKE_HOLD**

  hold the brakes of the robot arm.

  Synopsys:

  #include <server.h>

  #include <iolevel.h>

  int robotBRAKE_HOLD();

  Description:

The function *robotBRAKE_HOLD()* sends the command *BRAKE_HOLD* to the Path Planner, which then transfers the command to the Control Server. Once received, the Control Server will call the *iolevel* function *brake_hold()* to hold the brakes of the robot arm.

Returns:

RET_OK on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner, Control_Server

- **robotSTART**

start to run the robot.

Synopsys:

#include <server.h>

#include <iolevel.h>

int robotSTART();

Description:

The function *robotSTART()* sends the command *START* to the Path Planner, which then transfers the command to the Control Server. Once received, the Control Server will call the *iolevel* functions *AC_power_on(), DC_power_on,* and *brake_release()* to power on the system, release the brakes, and start sampling and control, respectively.

Returns:

RET_OK on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner, Control_Server

- **robotSTOP**

stop the robot.

Synopsys:

#include <server.h>

#include <iolevel.h>

int robotSTOP();

Description:

The function *robotSTOP()* sends the command *STOP* to the Path Planner, which then transfers the command to the Control Server. Once received, the Control Server will reset DAC ports to zero, and call the *iolevel* functions *DC_power_off()*, and *brake_hold()* to stop the robot control system.

Returns:

RET_OK on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner, Control_Server

- **robotEXIT**

stop the robot and exit the system.

Synopsys:

```
#include <server.h>
#include <iolevel.h>
int robotEXIT();
```

Description:

The function *robotEXIT()* sends the command *EXIT* to the Path Planner, which then transfers the command to the Control Server. Once received, the Control Server will carry out the same steps as for the command *STOP* and, in addition, shutdown the AC power. After these actions, the command will set a flag to terminate the ROACS servers and will record the current joint positions in the file *puma.dat.*

Returns:

RET_OK on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner, Control_Server

- **robotEND_EFFECTOR_ON**

turn on the end effector of the robot system.

Synopsys:

```
#include <server.h>
#include <iolevel.h>
int robotEND_EFFECTOR_ON();
```

Description:

The function *robotEND_EFFECTOR_ON()* sends the command *END_EFFECTOR_ON*

to the Path Planner, which then transfers the command to the Control Server. Once received, the Control Server will call the *iolevel* function *end_effector_on()* to turn on the robot end-effector.

Returns:

RET_OK on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner, Control_Server

- **robotEND_EFFECTOR_OFF**

turn off the end effector of the robot system.

Synopsys:

#include <server.h>

#include <iolevel.h>

int robotEND_EFFECTOR_OFF();

Description:

The function *robotEND_EFFECTOR_OFF()* sends the command *END_EFFECTOR_OFF* to the Path Planner, which then transfers the command to the Control Server. Once received, the Control Server will call the *iolevel* function *end_effector_off()* to turn off the robot end-effector.

Returns:

RET_OK on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of

the servers.

Class:

Path_Planner, Control_Server

- **robotWAIT**

  make the Path Planner wait until the execution of the pending paths is fully completed.

  Synopsys:

  #include <server.h>

  #include <iolevel.h>

  int robotWAIT();

  Description:

  The function *robotWAIT()* sends the command *WAIT* to the Path Planner. The Path Planner will reject any new paths until the execution of the previous paths is completed.

  Returns:

  RET_OK on success. On error, a value of (-1) is returned, and the *errno* is set.

  Errors:

  ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

  Class:

  Path_Planner

- **moveWorld**

  perform a linear motion in the world coordinate.

  Synopsys:

  #include <server.h>

  #include <iolevel.h>

int moveWorld(double r1[][3], double p1[], double r2[][3], double p2[], double tv, double ta, double av, double aa);

Description:

The function *moveWorld()* moves the robot end-effector from the first point, which is defined by $[r1, p1]$, to the second point, which is defined by $[r2, p2]$, in world coordinates. The maximum tool velocity and acceleration are given by tv and ta, respectively. The maximum angular velocity and angular acceleration of the joints are given by av and aa, respectively. Here, $r1$ and $r2$ define the orientation of the robot end effector in world coordinates, as 3×3 matrices. Also, $p1$ and $p2$ define the positions of the end effector in world coordinates, as arrays of the X, Y, and Z coordinates.

Returns:

RET_OK – Success.

BUF_FULL – The setpoint buffer is full; thus this path is rejected.

PATH_TOO_SHORT – The commanded path is too short.

(-1) – Failure in communication, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner

- **movetoWorld**

perform a destination linear motion in world coordinates.

Synopsys:

#include <server.h>

#include <iolevel.h>

int movetoWorld(double r2[][3], double p2[], double tv, double ta, double av, double aa);

Description:

The function *movetoWorld()* moves the robot end effector from the current position (including orientation) to the second point, which is defined by $[r2, p2]$, in world coordinates. The maximum tool velocity and acceleration are given by tv and ta, respectively. The angular velocity and angular acceleration of the joints are given by av and aa, respectively. Here $r2$ defines the orientation of the robot end effector in world coordinates, as a 3×3 matrix. Also, $p2$ defines the position of the end effector in world coordinates as an array of the X, Y, and Z coordinates.

Returns:

RET_OK – Success.

BUF_FULL – The setpoint buffer is full; thus this path is rejected.

PATH_TOO_SHORT – The commanded path is too short to be planned.

(-1) – Failure in communication, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner

- **movetoWorldR**

perform a relative-destination linear motion in world coordinates.

Synopsys:

#include <server.h>

#include <iolevel.h>

int movetoWorldR(double delta_p2[], double tv, double ta, double av, double aa);

Description:

The function *movetoWorldR()* moves the robot end effector from the current position (including orientation) to the second position, while maintaining the same orientation. The offset of the second point is defined by *delta_p2* in the X, Y, and Z directions. The maximum tool velocity and acceleration are given by *tv* and *ta*, respectively. The angular velocity and angular acceleration of the joints are given by *av* and *aa*, respectively.

Returns:

RET_OK – Success.

BUF_FULL – The setpoint buffer is full; thus this path is rejected.

PATH_TOO_SHORT – The commanded path is too short to be planned.

(-1) – Failure in communication; the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner

- **moveJoint**

perform a point-to-point (PTP) motion in the joint coordinates.

Synopsys:

#include <server.h>

#include <iolevel.h>

int moveJoint(double p1[], double p2[], double tv, double ta, double av, double aa);

Description:

The function *moveJoint()* moves the robot joints from the first position, which is defined

by $p1$, to the second position, which is defined by $p2$, in the joint coordinates. The maximum tool velocity and acceleration are given by tv and ta, respectively. The angular velocity and angular acceleration of the joints are given by av and aa, respectively. Here $p1$ and $p2$ define the positions of the joints, as arrays of 6 (CHNUM) elements.

Returns:

RET_OK – Success.

BUF_FULL – The setpoint buffer is full; thus this path is rejected.

PATH_TOO_SHORT – The commanded path is too short to be planned.

(-1) – Failure in communication; the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner

- **movetoJoint**

perform a destination point-to-point (PTP) motion in the joint coordinates.

Synopsys:

#include <server.h>

#include <iolevel.h>

int movetoJoint(double p2[], double tv, double ta, double av, double aa);

Description:

The function *movetoJoint()* moves the robot joints from the current position to the a position, which is defined by $p2$ in the joint coordinates. The maximum tool velocity and acceleration are given by tv and ta, respectively. The angular velocity and angular acceleration of the joints are given by av and aa, respectively. Here $p2$ defines the

positions of the joints, as an array of 6 (CHNUM) elements.

Returns:

RET_OK – Success.

BUF_FULL – The setpoint buffer is full; thus this path is rejected.

PATH_TOO_SHORT – The commanded path is too short to be planned.

(-1) – Failure in communication; the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner

- **movetoJointRcommand**

perform a relative destination point-to-point (PTP) motion in the joint coordinates.

Synopsys:

#include <server.h>

#include <iolevel.h>

int movetoJointRcommand(double delta_p2[], double tv, double ta, double av, double aa);

Description:

The function *movetoJointRcommand()* moves the robot joints from the current position to a destination which has the offset *delta_p2* away from the current position, in the joint coordinates. The maximum tool velocity and acceleration are given by *tv* and *ta*, respectively. The angular velocity and angular acceleration of the joints are given by *av* and *aa*, respectively. Here *delta_p2* defines the offset of each joint relative to the current position.

Returns:

RET_OK – Success.

BUF_FULL – The setpoint buffer is full; thus this path is rejected.

PATH_TOO_SHORT – The commanded path is too short to be planned.

(-1) – Failure in communication; the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner

- **movetoJointR**

perform a single-joint motion.

Synopsys:

#include <server.h>

#include <iolevel.h>

int movetoJointR(int ch, double deltap, double tv, double ta, double av, double aa);

Description:

The function *movetoJointR()* performs a single-joint motion through an angle of *deltap*. Also, *ch* defines the joint number (0-5). The maximum tool velocity and acceleration are given by *tv* and *ta*, respectively. The angular velocity and angular acceleration of the joints are given by *av* and *aa*, respectively. This function is a special case of the function *movetoJointRcommand()* where the motion of all the joints can be defined. This function has a special use in jogging motions, where only one joint moves at a time.

Returns:

RET_OK – Success.

BUF_FULL – The setpoint buffer is full; thus this path is rejected.

PATH_TOO_SHORT – The commanded path is too short to be planned.

(-1) – Failure in communication; the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner

- **movetoJointButton**

perform a continuous single-joint motion.

Synopsys:

#include <server.h>

#include <iolevel.h>

int movetoJointButton(int ch, double flag, double tv, double ta, double av, double aa);

Description:

The function *movetoJointButton()* performs a continuous single-joint motion at the given velocity in the direction *flag*, where *flag* takes a value of either 1 or (-1) indicating the motion direction to be positive or negative. Also, *ch* defines the joint number (0-5). The maximum tool velocity and acceleration are given by *tv* and *ta*, respectively. The angular velocity and angular acceleration of the joints are given by *av* and *aa*, respectively. This function is provided for continuous jogging of the robot joints.

Returns:

RET_OK – Success.

BUF_FULL – The setpoint buffer is full; thus this path is rejected.

PATH_TOO_SHORT – The commanded path is too short to be planned.

(-1) – Failure in communication; the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner

- **moveStop**

stop robot motion, abort pending paths.

Synopsys:

#include <server.h>

#include <iolevel.h>

int moveStop(double aa);

Description:

The function *moveStop()* stops robot motion at the maximum acceleration of *aa*. This function can be used with *movetoJointButton()* to perform continuous jogging. This function is also important for performing impedance-based actions.

Returns:

RET_OK – Success.

(-1) – Failure in communication; the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner

- **getSetptLength1**

get the length of the set point buffer from the Path Planner.

Synopsys:

#include <server.h>

#include <iolevel.h>

int getSetptLength1();

int getSetptLength();

Description:

The function *getSetptLength1()* obtains the length of the setpoints from the Path Planner. The buffer length does not indicate whether there is pending path. However, when *getSetptLength1()* returns a zero, the buffer will be empty, and the robot will remain stationary. (If there are pending paths, the buffer should not be empty; the Control Server triggers the Path Planner to plan a new path when the buffer is half). The function *getSetptLength()* is similar, but obtains the information from the Data Analyzer.

Returns:

The length of the set point buffer. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Path_Planner, Data_Analyzer

- **getOperData**

get a block of operational data.

Synopsys:

#include <server.h>

#include <iolevel.h>

int getOperData(RecordData *retRec).

Description:

The function *getOperData()* gets a block of operation data. The data block, which is pointed to by *retRec*, may have up to RECLENGTH points. The data structure is shown below:

```
typedef struct {
 clock_t time;
 float dpos[CHNUM];
 float pos[CHNUM];
 float vel[CHNUM];
 float crt[CHNUM];
 float imp[CHNUM];
 float ctr[CHNUM];
 float acc[CHNUM];
 float torq[CHNUM];
 } RecordData;
```

where *time* is the time, as counted from the start of the ROACS servers; *dpos* denotes the desired position; *pos* denotes the actual position; *vel* denotes the joint velocity, *crt* denotes the motor current of a joint; *imp* denotes the single-DOF impedance of a joint; *ctr* denotes the control signal; *acc* denotes the acceleration; and *torq* denotes the joint torque.

This function gets all recorded data in the data buffer, and resets the buffer pointer to zero after taking the data.

Returns:

The number of records. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Data_Analyzer

- **getCrtOperData**

  get the last record of the operational data.

  Synopsys:

  #include <server.h>

  #include <iolevel.h>

  int getCrtOperData(RecordData *retRec);

  Description:

  The function *getCrtOperData()* gets the last record of data in the data buffer. The record is pointed to by *retRec*, which has the data structure of *RecordData*. This function does not reset the data buffer.

  Returns:

  The number of records (1). On error, a value of (-1) is returned, and the *errno* is set.

  Errors:

  ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

  Class:

  Data_Analyzer

- **getCrtAllDataJoint**

  get position, velocity, motor current, and impedance of all joints.

Synopsys:

#include <server.h>

#include <iolevel.h>

int getCrtAllDataJoint(double p[], double v[], double crt[], double imp[]);

Description:

The function *getCrtAllDataJoint()* reads the last record of the position *p*, velocity *v*, motor current *crt*, and impedance *imp* from the Data Analyzer. Note that *p, v, crt, imp* should be declared befor calling this function. This function does not reset the data buffer.

Returns:

1 on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Data_Analyzer

- **getCrtPosJoint**

get the current position of all joints.

Synopsys:

#include <server.h>

#include <iolevel.h>

int getCrtPosJoint(double p[]);

Description:

The function *getCrtPosJoint()* reads the last record of the position *p* from the Data Analyzer. Note that *p* should be declared before this function call. This function does

not reset the data buffer.

Returns:

1 on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Data_Analyzer

- **getCrtPosWorld**

get the current position of the end effector in world coordinates.

Synopsys:

#include <server.h>

#include <iolevel.h>

int getCrtPosWorld(double p[], double r[][3]);

Description:

The function *getCrtPosWorld()* reads the last record of the position *p* and the orientation *r* of the end effector of the robot in world coordinates, from the Data Analyzer. Note that *p[3], r[3][3]* should be declared before this function call. This function does not reset the data buffer.

Returns:

1 on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Data_Analyzer

- **getCrtSetptJoint**

  get the current setpoint of all joints.

  Synopsys:

  #include <server.h>

  #include <iolevel.h>

  int getCrtSetptJoint(double p[]);

  Description:

  The function *getCrtSetptJoint()* reads the last record of the set point p of all joints of the robot from the Data Analyzer. Note that p should be declared before this function call. This function does not reset the data buffer.

  Returns:

  1 on success. On error, a value of (-1) is returned, and the *errno* is set.

  Errors:

  ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

  Class:

  Data_Analyzer

- **getCrtSetptWorld**

  get the current setpoint of the end-effector in world coordinates.

  Synopsys:

  #include <server.h>

  #include <iolevel.h>

  int getCrtSetptWorld(double p[], double r[][3]);

  Description:

The function *getCrtSetptWorld()* reads the last record of the position *p* and the orientation *r* of the end effector in world coordinates, from the Data Analyzer. Note that *p[3], r[3][3]* should be declared befor calling this function. This function does not reset the data buffer.

Returns:

1 on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Data_Analyzer

- **setSafetyCheck**

enable/disable safety check in the Control Server.

Synopsys:

#include <server.h>

#include <iolevel.h>

int setSafetyCheck(int fl);

Description:

The function *setSafetyCheck()* enables the safety check in the Control Server by setting the flag *fl* to 1; and disables the safety check by setting *fl* to 0. The flag is passed through the Data Server.

Returns:

1 on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Data_Analyzer, Control_Server

- **getSafetyCheck**

get the safety-check flag from the Control Server.

Synopsys:

#include <server.h>

#include <iolevel.h>

int getSafetyCheck( );

Description:

The function *getSafetyCheck()* reads the safety status of the robot.

Returns:

-10 for no hardware failure and 0-5 for respective joint failure. On communication error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Data_Analyzer, Control_Server

- **getPowerStatus**

get the server operation flags from the Control Server.

Synopsys:

#include <server.h>

#include <iolevel.h>

int getPowerStatus(char *ac, char *dc, char *brake, char *end_effector);

Description:

The function *getPowerStatus()* reads the server flags through the Data Analyzer. Note that *ac, dc, brake, end_effector* are the flags for AC power, DC power, the robot brake, and the robot end-effector, respectively. A flag that has the value 1 means positive (power on, brake released, end effector on), and that has the value 0 means negative (power off, brake held, end effector off).

Returns:

1 on success. On error, a value of (-1) is returned, and the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Data_Analyzer, Control_Server

- **getLimitHit**

check if a joint limit is hit.

Synopsys:

#include <server.h>

#include <iolevel.h>

int getLimitHit();

Description:

The function *getLimitHit()* reads the server flags on joint-limit-hit through the Data Analyzer.

Returns:

0 for no joint limit error, 1-6 for the respective joint (1-6) failure, (-1) for communication error; the *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Data_Analyzer, Control_Server

- **getSetptLength**

get the length of the setpoints.

Synopsys:

#include <server.h>

#include <iolevel.h>

int getSetptLength();

Description:

The function *getSetptLength()* obtains the length of the set points. This information is derived from the Control Server.

Returns:

The length of the set points. (-1) on error; *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Data_Analyzer, Control_Server

- **robotStopped**

check if the robot is stationary.

Synopsys:

#include <server.h>

#include <iolevel.h>

int robotStopped();

Description:

The function *robotStopped()* checks if the robot is stationary by checking if the length of the setpoints is zero.

Returns:

1 if robot is stationary, 0 for no determination of the robot status.

Class:

Data_Analyzer

- **setCtrlParam**

set PD control parameters.

Synopsys:

#include <server.h>

#include <iolevel.h>

int setCtrlParam(double kp[], double kd[]);

Description:

The function *setCtrlParam()* sets the PD control parameters to the Control Server. Note that *kp*, and *kd* are the proportional and derivative gains, respectively, of the low-level PD controller.

Returns:

RET_OK on success; a value of (-1) on error and *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Data_Analyzer, Control_Server

- **getCtrlParam**

  read PD control parameters from the shared memory.

  Synopsys:

  #include <server.h>

  #include <iolevel.h>

  int getCtrlParam(double kp[], double kd[]);

  Description:

  The function *getCtrlParam()* reads the PD control parameters from the shared memory. Note that *kp* and *kd* are pointers to the arrays of PD parameters.

  Returns:

  RET_OK on success; a value of (-1) on error and *errno* is set.

  Errors:

  ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

  Class:

  Data_Analyzer, Control_Server

- **stopAt**

  set criteria for stopping the robot.

  Synopsys:

  #include <server.h>

  #include <iolevel.h>

  int stopAt(double exp[]);

  Description:

  The function *stopAt()* sets impedance conditions to the Data Server. The Data Server will trigger a stop-proxy to the Path Planner if one of the impedance threshold values,

which is expressed by *exp* (six dof), is reached.

Returns:

RET_OK on success; a value of (-1) on error and *errno* is set.

Errors:

ESRCH – A QNX error; no such process. The communication is broken due to exit of the servers.

Class:

Data_Analyzer, Control_Server

Appendix C

Functions of Hardware Interface Library

The hardware interface library is designed for the user to access low-level hardware. It is quite useful for user to implement low-level control algorithms using the setup described in the user manual. The library consists of three parts of I/O functions: functions of the M5312 encoder cards, functions of the DDA06 DAC card, and functions of the PCL814B multiple function card. The data structure used in the library functions is given in the header file "iolevel.h", which should be included in the user's source code that calls the library functions.

To use the functions in this library, user's program must be linked for the privilege level 1, and the process must be run by the super-user. User should check Watcom C compiler manual for more information on privilege levels.

Functions are classified into three parts: PCL814B functions, DDA06 functions, and M5312 functions. This will help the user for identifying the relations between the hardware and the library functions.

- **initSystem**

  initialize the hardware system which includes two M5312 encoder cards, one DDA06 DAC card, and one PCL814B multiple function card.

  Synopsys:

  #include <iolevel.h>

  int initSystem(double ctrperiod);

  Description:

  The function *initSystem()* initializes the I/O cards. The period of sampling and control

should be passed to this function in milliseconds by *ctrperiod.* This function uses the predefined base addresses for the I/O cards, which are defined in *iolevel.h.* In case of change of the addresses, to initialize the I/O boards properly, the user should either change the address definitions in *iolevel.h*, or use the functions *initPCL814B(), initM5312(),* and *initDDA06()* instead.

Returns:

1 on success. On error, -1 is returned, and the *errno* is set.

Errors:

INITPCL814B_FAIL: Failure on initalization of PCL814B board.

INITDDA06_FAIL: Failure on initialization of DDA06 board.

INITM5312_1_FAIL: Failure on initialization of the first M5312 board.

INITM5312_2_FAIL: Failure on initialization of the second M5312 board.

Class:

PCL814B, DDA06, M5312

- **initPCL814B**

initialize the PCL814B multiple function card.

Synopsys:

#include <iolevel.h>

int initPCL814B(double ctrperiod, int b_addr);

Description:

The function *initPCL814B()* initializes the PCL814B card. Sampling period *ctrperiod* in milliseconds and the base address *b_addr* of the card should be passed to the function. This function should be called prior any other PCL814B function calls.

Returns:

1 on success. 0 on error (hardware failure).

Class:

PCL814B

- **readPCL814B**

read potentiometer voltages and motor currents from PCL814B multiple function card.

Synopsys:

#include <iolevel.h>

void readPCL814B(double *crt, double *pot, int chnum);

Description:

The function *readPCL814B()* reads motor currents and potentiometer voltages of *chnum* channels from the PCL814B card. The returned data are stored in arrays pointed to by *crt* and *pot*. Note that the value of *chnum* can be up to 6.

Class:

PCL814B

- **PCL814BDO**

sent data to digital output port of PCL814B multiple function board.

Synopsis:

#include <iolevel.h>

void PCL814BDO(char v, int offset, int b_addr);

Description:

The function *PCL814BDO()* sends the byte *v* to the PCL814B output port. The base address of the board is given by *b_addr*, and the offset of the port is given by *offset*. This function is intended for internal use, and the user may not need it.

Class:

PCL814B

- **PCL814BAD**

  performs low-level A/D conversion in PCL814B card.

  Synopsis:

  #include <iolevel.h>

  int PCL814BAD(double *fData, int b_addr);

  Description:

  The function *PCL814BAD()* triggers the PCL814B multiple function board to perform the A/D conversions for all 16 channels. The base address of the board is passed by *b_addr* and the returned data is stored in an array pointed to by *fData*. The ranges of the analog signals are set to ±5V, except for the channels 5 and 6, for which the ranges are set to ±0.625V. This is because the latter two channels are used for measuring the currents of joints 5 and 6 where no amplifiers are employed.

  Returns:

  1 on success, and 0 on error.

  Class:

  PCL814B

- **AC_power_on**

  turn on the AC power of the control system.

  Synopsis:

  #include <iolevel.h>

  void AC_power_on();

  Description:

  The function *AC_power_on()* turns on the AC power of the system. Note that a delay should follow this function, prior to executing other commands. This delay should be longer than or equal to the settling time of the electric system (power supply etc.).

Class:

PCL814B

- **AC_power_off**

  turn off the AC power of the control system.

  Synopsis:

  #include <iolevel.h>

  void AC_power_off();

  Description:

  The function *AC_power_off()* turns off the AC power of the system. This function should be called after shutting down the control system.

  Class:

  PCL814B

- **end_effector_on**

  turn on the robot end-effector (cutter).

  Synopsis:

  #include <iolevel.h>

  void end_effector_on();

  Description:

  The function *end_effector_on()* turns on the robot end-effector.

  Class:

  PCL814B

- **end_effector_off**

  turn off the robot end-effector (cutter).

  Synopsis:

  #include <iolevel.h>

void end_effector_off();

Description:

The function *end_effector_off()* turns off the robot end-effector.

Class:

PCL814B

- **brake_release**

  release the robot brakes.

  Synopsis:

  #include <iolevel.h>

  void brake_release();

  Description:

  The function *brake_release()* releases the robot brakes.

  Class:

  PCL814B

- **brake_hold**

  hold the robot brakes.

  Synopsis:

  #include <iolevel.h>

  void brake_hold();

  Description:

  The function *brake_hold()* holds the robot brakes. Note that this function should be called after a *DC_power_off()* call.

  Class:

  PCL814B

- **DC_power_on**

turn on the DC power of the control system.

Synopsis:

#include <iolevel.h>

void DC_power_on();

Description:

The function *DC_power_on()* turns on the DC power of the system. The power amplifiers will start to operate upon calling this function.

Class:

PCL814B

- **DC_power_off**

  turn off the DC power of the control system.

  Synopsis:

  #include <iolevel.h>

  void DC_power_off();

  Description:

  The function *DC_power_off()* turns off the DC power of the system. This function should be followed by a *brake_hold()* call to avoid a collapse of the robot arm.

  Class:

  PCL814B

- **initDDA06**

  initialize the DDA06 D/A board.

  Synopsis:

  #include <iolevel.h>

  int initDDA06(int b_addr);

  Description:

The function *initDDA06()* initializes the DDA06 board that has the base address *b_addr*. This function will set the output of all the channels to zero.

Returns:

1 on success and 0 on hardware failure.

Class:

DDA06

- **DDA06DA**

  send analog signals to power amplifiers.

  Synopsis:

  #include <iolevel.h>

  void DDA06DA(int ddaPort, double *v1, int b_addr);

  Description:

  The function *DDA06DA()* converts the digital signals that are pointed to by *v1*, to analog outputs. The D/A board has the base address *b_addr*, and the port is given by *ddaPort*. The range of the signals is ±10V. Digital numbers given by *v1* will be truncated if they are outside the range.

  Class:

  DDA06

- **outputDA**

  send a single control signal to a specific channel.

  Synopsis:

  #include <iolevel.h>

  void outputDA(int ch, double *v);

  Description:

  The function *outputDA()* sends a control signal given by *v* to channel *ch*. Here *v* is in the

range of ±10V. Also, *ch* is a number between 0 to 5, and stands for the control channels 1 to 6.

Class:

DDA06

- **stop**

  stop the robot motion.

  Synopsis:

  #include <iolevel.h>

  void stop();

  Description:

  The function *stop()* resets the control output, turns the amplifier power off, and holds the robot arm stationary. To resume robot control, *DC_power_on()* and *brake_release()* should be called.

  Class:

  PCL814B, DDA06

- **initM5312**

  initialize M5312 encoder cards.

  Synopsis:

  #include <iolevel.h>

  int initM5312(int b_addr);

  Description:

  The function *initM5312()* initializes an M5312 encoder board that has the base address *b_addr*. To initialize both the encoder cards, this function should be called twice with the corresponding base addresses. This function should be called prior to a call to any other M5312 functions.

Returns:

1 on success and 0 on hardware failure.

Class:

M5312

- **resetM5312**

  reset all encoder counts to zero.

  Synopsis:

  #include <iolevel.h>

  int resetM5312();

  Description:

  The function *resetM5312()* resets the encoder counts to zero for all channels of both M5312 boards. This function calls *initM5312()*, but the base addresses are predefined in *iolevel.h.*

  Returns:

  1 on success and (-1) on error. Also *errno* will be set.

  Errors:

  INITM5312_1_FAIL: No. 1 board is failure;

  INITM5312_2_FAIL: No. 2 board is failure.

  Class:

  M5312

- **readJointPosition**

  read robot joint position by channels.

  Synopsis:

  #include <iolevel.h>

  double readJointPosition(int jointnum);

Description:

The function *readJointPosition()* reads the position of the joint *jointnum*. The joint number *jointnum* can range from 0 to 7. Channels 0 to 5 are used for the robot joints 1 to 6.

Returns:

The joint angles in radians (for channels 0-5). Two extra encoder channels can also be read, but the return value will be the raw count.

Class:

M5312