THE PARAMETER IDENTIFICATION OF A NOVEL SPEED REDUCER

Xiaohui Song

Department of Mechanical Engineering
McGill University, Montréal

March 2002

A Thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of
Master of Engineering

ABSTRACT

Many a mechanical application involves power transmission from a high-speed motor to a low-speed load. However, existing speed-reduction mechanisms are usually a major sink of energy and information in mechanical transmissions. Energy and positioning information are lost through: (a) friction between sliding components; (b) compliance; and (c) backlash. A novel transmission for speed reduction, Speed-o-Cam, is currently under research at McGill University's Centre for Intelligent Machines (CIM). The transmission is based on the layout of *pure-rolling* indexing cam mechanisms, and hence, eliminates backlash and friction. Besides zero backlash and low friction losses, Speed-o-Cam also offers the possibility of high stiffness, another essential attribute for high-accuracy applications.

This thesis focuses on the aspects of both model development and mechanical-parameter identification of a spherical prototype of Speed-o-Cam. Our main interest lies in identifying the mechanism stiffness. In order to conduct experiments on the prototype, a testbed was designed and fabricated. A mathematical model of the testbed is first formulated. Based on this model and the results of experiments, the parameters of the Speed-o-Cam prototype are identified. In the process, the stiffness and damping parameters of the couplings of the testbed are also identified.

Power efficiency is an important indicator of speed reducing mechanisms. For the Speed-o-Cam prototype, this indicator is also estimated experimentally.

Résumé

Beaucoup d'applications en mécanique nécessitent des systèmes efficaces de transmission. Cependant, les réducteurs de vitesse actuels engendrent généralement des pertes d'énergie et d'information. Ces pertes sont principalement dues au frottement entre les pièces en mouvement, aux déformations des composants flexibles et au jeu. Le *Centre de recherche sur les machines intelligentes* de l'Université McGill met au point à l'heure actuelle une transmission novatrice, appelée Speed-o-Cam, ayant la fonction de réduction de vitesse. Cette transmission est inspirée des systèmes de mouvement intermittent à cames et roulements *sans frottement.* Par conséquent, le jeux et le frottement sont élimiñes dans le mécanisme. En outre, la transmission Speed-o-Cam offre la possibilité dón grande raideur, une propriété essentielle pour les applications de haute précision.

Cette thèse porte sur deux aspects d'un prototype sphérique de Speed-o-Cam: le développement d'un modèle mathématique et l'identification de ses paramètres. Nous nous intéressons particulièrement ā lìdentification de la raideur du mécanisme.Afin de mener des expér: ences sur le prototype du mécanisme, un banc d'essais a été conçu et réalisé. Dans un premier temps, un modèle mathématique du banc d'essais est établi. Ensuite, les paramétres mécaniques du prototype sont identifiés expérimentalement. Enfin, les paramétres de rigidité et d'amortissement du banc d'essais sont aussi identifiés.

Le rendement est un indicateur important du prototype de Speed-o-Cam, qui est estimé par une série d'expériences.

ACKNOWLEDGEMENTS

I am indebted to my thesis supervisor, Professor Jorge Angeles, for his continued supervision, guidance, suggestions and invaluable support throughout the course of my research. I admire his deep insight into many scientific subjects and his enthusiasm and vigour for research. He has played a very active role in all aspects of my graduate education; I truly appreciate his time and effort.

I would like to sincerely thank Dr. A. Hemami, who shared with me his wide practical knowledge and experience while Professor Angeles was away on sabbatical. Dr. Hemami's help and support have been most valuable.

I would like to thank the support by NSERC (Natural Science and Engineering Research Council, of Canada), under the Strategic Project No. STR192750-1996, and our industrial partners in the Speed-o-Cam Project, Alta Precision Inc., of Ville d'Anjou, Quebec, and Placage Unique Inc., of Rigaud, Quebec for their assistance in producing the Speed-o-Cam prototypes.

I would like to thank Professor Yixin Shao and Machinist Damon Kiperchuk, of the Department of Civil Engineering and Applied Mechanics, for their friendly help when I was conducting part of my experiments in the Solid Mechanics Laboratory.

The two years that I spent at the Centre for Intelligent Machines (CIM) were very joyful and productive. I would also like to thank the staff, secretaries, and my colleagues for their help, support and making this stay very enjoyable.

I am grateful to my parents for their endless encouragement and support, which gave

me love and confidence in completing my thesis. Special thanks are due to my sister for her love, patience and understanding throughout the hard times during my research.

Last but not least, I would like to express my deepest gratitude to the persons who directly or indirectly contributed to my thesis, but are not mentioned here.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

A speed reducer is an integral part of most mechanical transmissions. Power delivered from motors is usually available at a relatively high speed, and relatively low torque. In many applications, however, such as in electrical appliances, machine tools, conveyors, robotic joints and automobiles, power is required at a relatively low speed and high torque. Speed reducers used in mechanical transmissions between the motor and the load provide means of reducing the speed while increasing the torque.

Speed-reduction mechanisms have been built and investigated extensively since the eighteenth century (Euler, 1754). These mechanisms have also been the topic of continuous research for the last fifty years (Buckingham, 1963; Dudley, 1966), during which various mechanisms have emerged to improve speed reducers, most of them being gear-based. One of the most common mechanisms is the spur-gear train, which is the simplest to design and manufacture. This mechanism is usually employed in gearboxes with variable-speed reductions. Others, like planetary, or epicyclic gear trains, which consist of one or more central 'sun' gears with 'planet' gears, translating and rotating around the 'sun' gear, have more complicated layouts. This class of gear trains, accompanied by a clutch, can be found useful in automobile transmissions. Another device, making use of the concept of flexible gears, is the harmonic drive, which has found widespread acceptance since its invention by C.Walton Musser

in 1955 (U.S.patent 2959065). Because of their compact size and high reduction ratio, harmonic drives are often favoured for electromechanical systems with space and weight constraints (Yuen, 1996). Typical examples are the joint transmissions in robotic manipulators, which require relatively low speed and high torque.

Although gear-based mechanisms are widely used for high-rate speed reduction in various industrial environments. These have numerous drawbacks, two of the most significant of them being friction and backlash, which constitute two of the largest sinks of energy and sources of noise in conventional gear-based transmissions. Friction comes mainly from the sliding action of the involute gear teeth, the amount of slip increasing with the applied torque. If slippage is high, not only is the friction loss high, but also the speed regulation, efficiency and the life of the system decrease. Moreover, backlash makes the gear teeth lose contact. The impacts between teeth upon meshing can be damaging to both the machine on which the gears are mounted and the humans using the machine. Harmonic drives, on the other hand, eliminate backlash. However, transmission compliance results from gear-tooth interaction and wave-generator deformation due to high radial forces, and is present in all harmonic drives (Tuttle and Seering 1993). Compliance, in addition, is a source of nonlinearity, which leads to a nonlinear torque transmission between the motor and the load, an otherwise linear system.

Upon considering the foregoing alternatives to gears, along with their drawbacks, a methodology for designing planar, spherical and spatial cam mechanisms was developed by González-Palacios and Angeles (1993) in order to replace gear trains with cam mechanisms. These mechanisms are intended to outperform gear-based transmissions by virtue of:

- *high speed-reduction ratios*; they provide speed reductions of $N : 1$, where N is an integer as high as 10, in one single stage;

- *low friction losses*, because the transmissions have pure-rolling contact;
- *virtually zero backlash*, since the mechanisms can be configured to provide positive motion;
- *high stiffness*, by virtue of the convexity of the driving cam; and
- *low cost* to manufacture, because all contact surfaces can be machined with general-purpose CNC machine tools.

The main thrust of the thesis is the study of the dynamic response of these mechanisms. In particular, investigations are carried out to identify the relevant mechanical parameter of a spherical prototype, namely, its stiffness. In the process, the stiffness and damping parameters of the couplings of the testbed are also identified.

1.1. Background

In this thesis we will investigate the dynamic behaviour and identify the mechanical parameters of a spherical speed reducer prototype. The necessary background is given in this chapter.

1.1.1. Mechanical Power Transmission Mechanical power is generated by a motor and is consumed by a load. Between the motor and the load we have the transmission elements. Mechanical power transmission refers to "the business of moving energy or power from the place where it is generated to the place where it is used by mechanical devices" (Patton, 1980). Energy is transferred through belts, gears, sprockets, shafts, power takeoffs, hydraulic valves, pneumatic hoses, and other mechanical devices. Mechanical power transmission between shafts can be accomplished in a variety of ways. In addition to gears, *flexible elements* such as belts and chains are in common use. These permit power to be transmitted between shafts that are separated by a considerable distance, thus providing the design engineer with greater

flexibility in the layout of driving and driven elements.

Belts are relatively quiet in operation. Except for timing belts, slippage between belt and pulleys causes speed ratios to be nonuniform. This slippage characteristic is sometimes used to advantage by permitting the pulleys to be moved closer together in order to disengage the drive, as in some snowblowers and self-propelled lawn mowers. This may save substantial cost, weight, and the bulk of providing a separate clutch. The flexibility and inherent damping in belts—and, to a lesser extent, in chains—serves to reduce the transmission of shock and vibration.

The design of chains illustrates the general proposition that if a component of desired characteristics is not already available, an engineer should consider the possibility of inventing something new. If little power is required, a "beaded chain," similar to the pull cord on a plain light fixture, can be used. A stronger type of flexible chain incorporates parallel steel cables bonded to the sides of plastic cylindrical "buttons" that simulate the rollers of conventional roller chains (Juvinall and Marshek, 1991).

1.1.2. Speed-o-Cam A cam mechanism is a mechanical device that transmits force or torque from cam to follower through a specific motion program by *higher-pair* contact. The nature of contact in a higher pair is along a line or a point, while that in a lower pair is along a surface (Denavit and Hartenberg, 1964).

Common applications of cam mechanisms include quick-return and indexing motions. Quick-return mechanisms are extensively used in manufacturing processes such as metal-cutting, metal-forming, pick-and-place and material-handing operations.

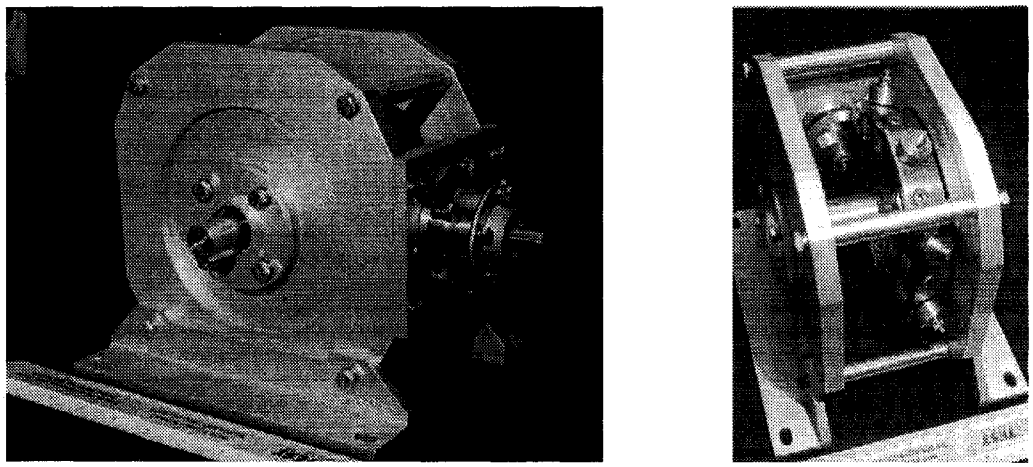
The high-speed reducers of interest, that we term *Speed-o-Cam*, are based on the layout of *pure-rolling* indexing cam mechanisms (ICM), i.e., a cam mechanism whose follower bears a *periodic* geometry, repeating itself N times. The mechanism is thus said to have N stages or *indexing steps*. An ICM is designed so as to allow the production of a periodic, nonreversing speed of the follower when the cam rotates at a constant angular speed (González-Palacios and Angeles, 1993).

While Speed-o-Cam stems from the concept of ICM, its distinguishing feature is the

type of periodic speed produced on the follower, namely, a constant speed that is a multiple or a submultiple of the cam speed. As a matter of fact, when Speed-o-Cam is used to actually reduce speed, just as gear transmissions, the speed-reduction ratio is $1/N$, for an N-stage layout. When Speed-o-Cam is used as a speed amplifier, the speed-amplification ratio is N.

The CIM Robotic Mechanical System Laboratory has produced four prototypes of Speed-o-Cam. Three prototypes are planar speed-reducing mechanisms, used to couple shafts of parallel axes. Another prototype is a spherical speed-reducing mechanism, which is intended for the coupling of intersecting shafts. Planar Speed-o-Cam is intended to replace spur and helical gears, while its spherical counterpart is to replace bevel gears. The R&D of Speed-o-Cam is reported in (González-Palacios and Angeles, 1999; González-Palacios and Angeles, 2000).

The subject of this thesis is specifically the spherical Speed-o-Cam, the first prototype being shown in Fig. 1.1.



(a) (b)

Figure 1.1: Spherical Speed-o-Cam prototype: (a) external view; (b) inner view

1.1.3. Dynamic Systems Nearly all observed phenomena in our daily lives or in engineering systems exhibit important dynamic features. Specific examples may

arise in (a) a physical system, such as a travelling space vehicle, a home-heating system, or in the mining of a mineral deposit; (b) a social system, such as the movement within an organizational hierarchy, the evolution of a tribal class system, or the behaviour of an economic structure; or (c) a life system, such as that of genetic transference, ecological decay, or population growth. While these examples illustrate the pervasiveness of dynamic situations and indicate the potential value of developing means for representing and analyzing dynamic behaviour, it must be emphasized that the general concept of dynamics transcends the particular origin or setting of the process or system.

Many dynamic systems can be understood and analyzed intuitively, without resorting to mathematical modelling and without developing a general theory. Indeed, we often deal quite effectively with many simple dynamic situations in our daily lives. However, in order to approach unfamiliar complex situations efficiently, it is necessary to proceed systematically.

With this view, the term *dynamics* soon takes on somewhat of a dual meaning. It is, first, as stated earlier, a term for the evolutionary phenomena in the world about us, and, second, a term for that part of science that is used for the representation and analysis of such phenomena. In the most profound sense, the term refers simultaneously to both aspects, the real and the abstract, besides the interplay between them. Although there are endless examples of interesting dynamic situations arising in a spectrum of areas, the number of corresponding general forms for mathematical representation is relatively small. Most commonly, dynamic systems are represented mathematically in terms of either differential or difference equations. These equations provide the structure for representing time linkages among variables.

The use of either differential or difference equations to represent dynamic behaviour corresponds, respectively, to whether the behaviour is viewed as occurring in continuous or discrete time. Continuous time corresponds to our usual conception, where time is regarded as a continuous variable and is often viewed as flowing smoothly past us. Mathematically, continuous time of this sort is quantified in terms of the

continuum of real numbers. An arbitrary value instant of continuous time is usually denoted by the letter t. Dynamic behaviour viewed in continuous time is usually described by differential equations, which relate the derivatives of a dynamic variable to its current value.

Discrete time consists of a denumerable sequence of instants rather than a continuum thereof. In terms of applications, it is convenient to introduce this kind of time when events and consequences either occur or are accounted for only at discrete time periods, such as daily, monthly, or yearly. When developing a population model, for example, it may be convenient to work with yearly population changes rather than with continuous-time changes. Discrete time is usually labelled by simply indexing, the discrete instants starting at a convenient reference time. Thus, if time corresponds to integers 0, 1, 2, and so forth, then an arbitrary instant is usually denoted by the letter k. Accordingly, dynamic behaviour viewed in discrete time is usually described by difference equations.

In loose terms a *system* is an object in which variables of different kinds interact and produce observable signals. The observable signals that are of interest to us are usually called *outputs*. The system is also affected by external stimuli, i.e., external signals that can be manipulated by will, and which are called *inputs*. Other stimuli are called disturbances and can be divided into those that are directly measured and those that are only observed through their influence on the output. The distinction between inputs and measured disturbances is often less important for the modelling process (Ljung, 1987).

Once dynamic phenomena and systems experiencing them are well understood, the definition of a dynamic system is readily available. A system is called *dynamic* if its present output depends on its past input; if its current output depends only on the current input, the system is known as *static*. The output of a static system remains constant if the input does not change. The output changes only when the input changes. In a dynamic system, the output changes with time if it is not in a state of equilibrium (Ogata, 1998). Dynamic systems are described by either partial or

ordinary differential equations; sometimes they are described by difference equations (case of discrete-time systems), integral equations and even by integro differential equations. Associated with a dynamical system is the notion of *state* of the system. The state of a dynamic system at a certain instant t_0 is the information pertaining to the system that completely describes the effect of the whole past excitation history up to and including time t_0. The behaviour of a dynamic system at some instant $t > t_0$ is then uniquely specified if two items are given, namely, 1) the time history of the excitations between t_0 and t, and 2) the state of the system at t_0. The state of a system at a time t_0 often turns out to be the familiar *initial conditions* of elementary differential equations.

1.1.4. System Identification The problem of system identification is generally referred to as the determination of the parameters of a mathematical model for a system or process by observing its input-output relationships (Hsia, 1977). System identification is a fundamental problem in science, medicine, engineering, and so on. Humankind has always sought knowledge of a physical system beyond that which is directly observable. The underlying theme is the relationship between the internal structure of a system and the observed output. The hidden features of the system are to be extracted from the experimental data.

Historically, system identification has been motivated by the need to design better control systems. In most practical systems, such as industrial processes, there is seldom sufficient a priori information about a system and its environment to design an effective control strategy. Very frequently, we are faced with the necessity of experimentally determining some important physical parameters such as heat transfer coefficient, chemical reaction rate, damping factor, and so on. The need for highly accurate system models has been intensified by the development of optimal and adaptive control theories. Other engineering applications for system identification of dynamic systems include communication channel probing, and system and fault testing.

The system identification problem can be classified into two categories:

(*i*) *The complete identification problem*, whereby we do not know anything about the system, such as whether it is linear or nonlinear, memoryless or with memory, and so on. Obviously, this is an extremely difficult problem to solve. Usually some kind of assumptions have to be made before any meaningful solution can be attempted. This type of problem is also referred to as a *black box* problem.

(*ii*) *Partial identification problem*, whereby, some basic features of the system, such as linearity, bandwidth, and so on, are assumed to be known. However, we may not know the specific *order* of the system, or the values of the associated coefficients. A situation of this kind is also called a *gray box* problem and is, of course, easier to deal with than the black box problem.

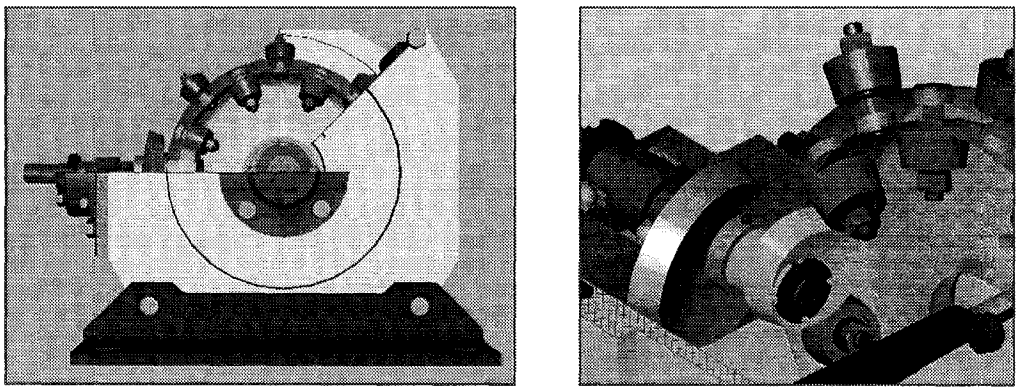
Fortunately, the majority of engineering systems and industrial processes we encounter in practice are of type (*ii*). In many cases, we know a good deal about the structure of the system, so that it is possible to derive a specific mathematical model of the system dynamics. Consequently, only a set of parameters in the model equation are left to be determined. Thus, the modelling problem is reduced to that of *parameter identification* (Hsia, 1977).

Since a majority of system identification problems can be either formulated as or reduced to a parameter identification problem, the treatment of the latter is considered to be of the greatest importance. In principle, we can precisely determine the unknown parameters in a mathematical model where the exact measurements of the input-output data are given. In reality, however, the input-output data are corrupted by measurement noise. The determination of system parameters is essentially a statistical-estimation problem, where we seek to specify a mathematical model that fits the observation data.

1.2. Motivation

Low friction and backlash are crucial in high-precision applications. Based on the layout of pure-rolling indexing cam mechanisms, Speed-o-Cam overcomes these problems associated with gear transmissions and other mechanisms used to reduce speed, because it eliminates friction, backlash and compliance. As well as zero backlash and low friction losses, Speed-o-Cam also exhibits high stiffness, another essential attribute for high-torque applications.

The main contribution of this thesis is a study of the dynamic behaviour of the spherical Speed-o-Cam. A computer-generated model of the spherical Speed-o-Cam is shown in Fig. 1.2, where the cams straddle the inside and outside surfaces of a webbed 'hoop', which has drive pins projecting radially from its inner and outer diameter surfaces. One cam drives the inner pins, the other the outer pins. The two cams are offset to guarantee multi-point contact. The centre of the hoop shares a common centre of rotation with the output shaft, to which it is connected by a web.



(a) (b)

Figure 1.2: (a) View of ensemble and (b) close-up of cam-follower meshing of the spherical Speed-o-Cam prototype

We will focus on the aspects of both model development and parameter identification. In the process, the testbed is designed and fabricated; then, a series of experiments

are conducted. The transfer function of the system is derived and, finally, the stiffness parameter and the power efficiency of the Speed-o-Cam prototype are obtained, which allow us to quantify how this device performs in those transmission aspects described above.

We decided to conduct dynamic tests for stiffness identification, instead of static tests, because by conducting static tests, we can only obtain the stiffness parameters; however, by conducting the dynamic tests, we can obtain not only the stiffness parameters, but also the damping parameters of the mechanical components of our testbed.

1.3. Scope and Organization of the Thesis

The thesis begins with a description of the Speed-o-Cam testbed. Then, a mathematical model of the testbed is derived. Once the model is created, the procedure to identify its mechanical parameters is described. The results of experiments then lead to model validation studies and the identification of the stiffness of Speed-o-Cam.

Chapter 2 describes the Speed-o-Cam testbed, with all its components (mechanical and electric), control software and human-machine interface.

In Chapter 3 the mathematical model of the Speed-o-Cam testbed is formulated. The transfer function of the whole system is derived and then used in conjunction with the results of dynamic experiments.

Chapter 4 describes the experiments conducted to estimate the parameters which will be matched with the mathematical model of Chapter 3. Model validation studies and a refinement of the proposed transfer function are created. An analysis of the experimental data and comparison studies are included.

In Chapter 5 the efficiency of the Speed-o-Cam prototype is estimated. In the process, the power losses of two couplings are also obtained.

Chapter 6 summarizes the work accomplished in this thesis and suggests further research work.

CHAPTER 2

Testbed Calibration

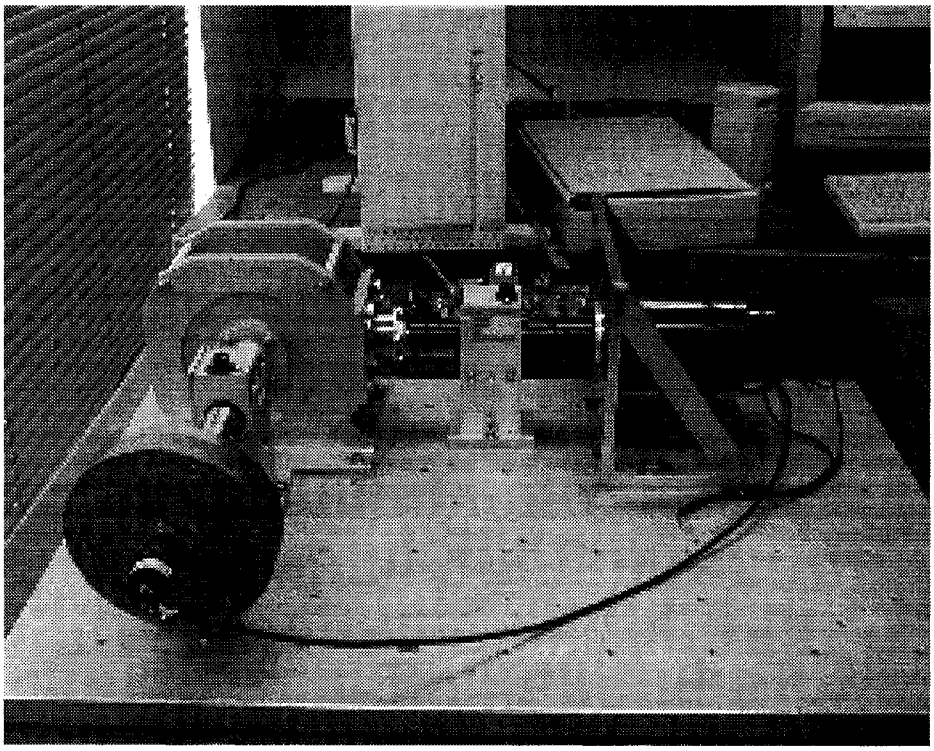


Figure 2.1: Speed-o-Cam testbed

This chapter gives an overview of the Speed-o-Cam testbed hardware and software that were set up to operate the Speed-o-Cam transmission and acquire the measured data.

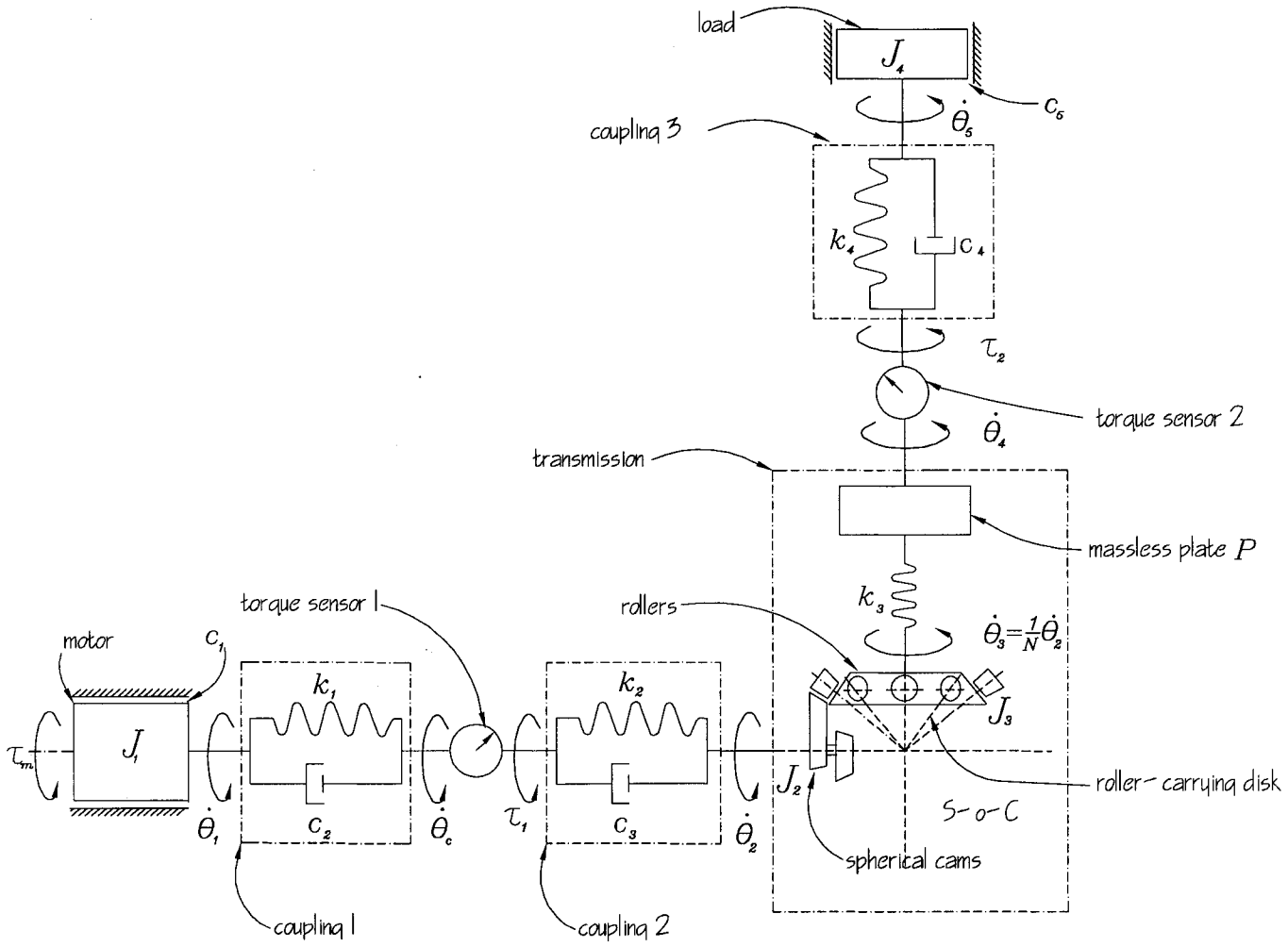


Figure 2.2: Iconic model of the Speed-o-Cam prototype testbed

2.1. Hardware

The Speed-o-Cam testbed, shown in Fig. 2.1 with the iconic model displayed in Fig. 2.2, is composed of several mechanical parts, namely,

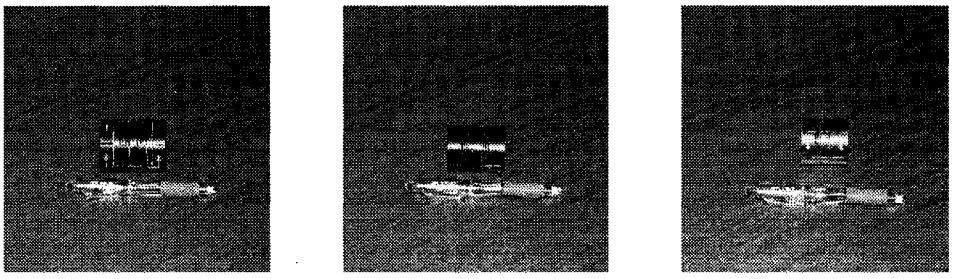
- *three couplings*: Flexible shaft couplings are used in the testbed, which are connected the motor, torque sensors, Speed-o-Cam transmission and load shaft.
- *the spherical Speed-o-Cam prototype*: The spherical Speed-o-Cam prototype is to be test here, which reduces the high speed delivered by the motor to a low speed at the load end.

- *the load shaft*: The load shaft is used to couple the spherical Speed-o-Cam prototype and the load.
- *the load*: A steel rotor is used as inertial load in our experiments.

Moreover, the electric hardware comprises the DC motor, the DC driver, the D/A converter and the data-acquisition system. The details of every component are described below.

2.1.1. Mechanical Hardware Setup

2.1.1.1. **Coupling Specifications** We used three couplings in our testbed. The first one is the DTO125-C14mm Flexible Shaft Coupling[1], which is connected to the DC motor with the input shaft torque sensor. The second one is the DTO125-C12mm Flexible Shaft Coupling, which is used to connect the input shaft torque sensor with the input shaft of the Speed-o-Cam transmission. The last one is the DTO125-C12mm Flexible Shaft Coupling that connects the output shaft torque sensor with the load shaft. All three couplings are shown in Fig. 2.3 with the parameter specifications recorded in Table 2.1.



(a) (b) (c)

Figure 2.3: Flexible shaft couplings: (a) DTO125-C14mm; (b) DTO125-C12mm; (c) BTO150-C14mm

2.1.1.2. **The Load Specifications** In order to conduct the tests of Speed-o-Cam, we introduce one inertial load, with the specifications described in Table 2.2.

[1]As per Electromate Industrial Sales Ltd. catalogue of Stocking Distributor Performance Motion Control Products, 2000.

Coupling	**Peak Torque** Nm	**Stiffness** Nm/deg
DTO125-C14mm Flexible Shaft Coupling	8.47	1.24
DTO125-C12mm Flexible Shaft Coupling	13.56	0.95
BTO150-C14mm Flexible Shaft Coupling	13.56	1.92

Table 2.1: Parameter specifications of the couplings

Mass kg	**Moment of Inertia** $kgmm^2$
4.998	14,752

Table 2.2: Parameter specifications of the load

2.1.2. Electrical Hardware Setup The electrical hardware setup involves using I/O devices, and the host computer. A PC with Pentium III 500MHz processor is used to drive the DC motor and acquire the measured data. The D/A converter card receives digital voltage signals from the computer and sends analog signals via the DC driver to drive the DC motor. Moreover, the A/D converter receives analog voltage signals from the data-acquisition system and converts analog voltage signals into digital data. The latter are then sent to the host computer. The electrical connection from the Speed-o-Cam testbed to the host computer is shown in Fig. 2.4

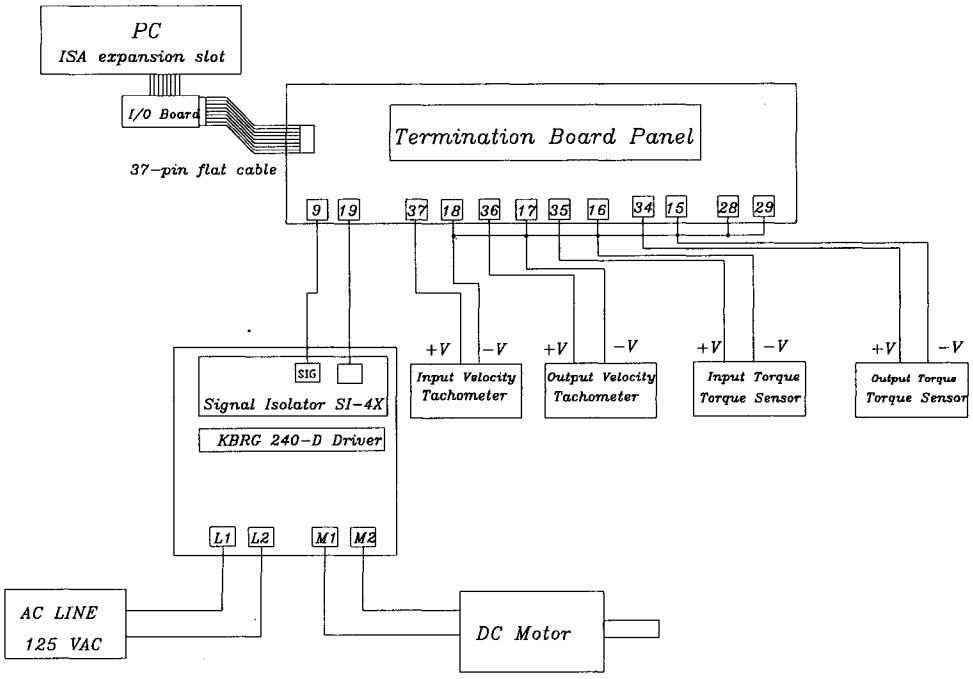


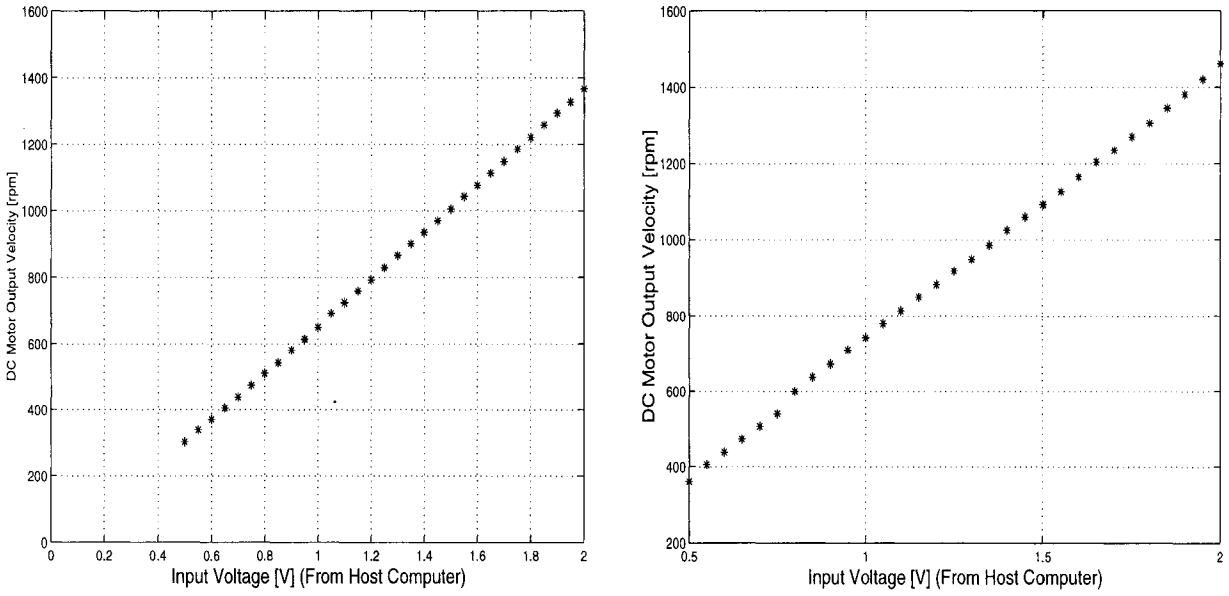
Figure 2.4: Connection scheme from the testbed to the host computer

2.1.2.1. **Motor Specifications** We used an *HS*-250-9 *DC Servo Actuator* for the Speed-o-Cam testbed, which drives the Speed-o-Cam transmission with its load. The specifications of this motor are recorded in Table 2.3.

Rating	Value
Rated Voltage	85V
Rated Current	5.40A
Rated Speed	3000rpm
Efficiency	82.01%
Viscous Damping Constant	44.73Nm/rpm
Moment of Inertia	3.04Nm/(rad/s^2)
Maximum Output Speed	4000rpm

Table 2.3: DC motor specifications

2.1.2.1.1. **Motor calibration** The purpose of the calibration is to identify the constant gain and bias that relate the input voltage from the host computer with the output velocity of the DC motor. The relation thus resulting is intended to eliminate the amplification effect of the DC driver, which is composed of the servo-amplifier and the D/A converter, whose values are unknown at the moment. The motor calibration was conducted with the aid of a multi-voltmeter and the motor tachometer.
The DC motor was run with different input voltages and the output velocities were measured with the motor tachometer. Plots of data obtained from the calibration are shown Fig. 2.5 both with and without mechanism and load.



(a) (b)

Figure 2.5: DC motor calibration for constant-velocity profiles: (a) with mechanism and load; (b) without mechanism and load

The relations for constant velocity profiles with and without mechanism and load are given below, respectively, with voltage measured in V and motor output velocity

in rpm.

$$\text{MOTOR OUTPUT VELOCITY} = 721.8443 * \text{input voltage} - 12.4679 \quad (2.1)$$

$$\text{MOTOR OUTPUT VELOCITY} = 709.3619 * \text{input voltage} - 56.0805 \quad (2.2)$$

2.1.2.2. **DC Driver Setup** The DC driver that we used is the KBRG-240D DC driver, as shown in Fig. 2.6.

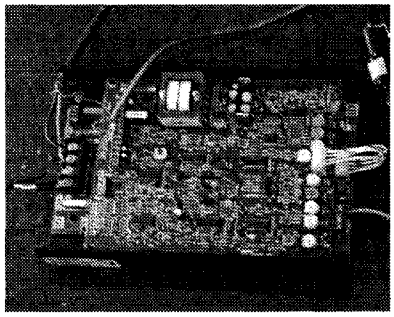


Figure 2.6: DC driver

The KBRG-240D DC driver is a full-wave regenerative control, capable of operating a DC motor in a bidirectional mode. It offers excellent controllability, which closely approximates the performance of servo-type drives. Its ratings are listed in Table 2.4.

Model	Input Voltage (VAC)	Max.AC Current (RMS)	Output Voltage (VDC)	Max. DC Output Current (ADC)	Max. Horsepower HP (KW)
KBRG-240D 8802	115	16	$0 - \pm 90$	11	1(0.75)
	230	16	$0 - \pm 180$	11	2(1.5)

Table 2.4: DC driver electrical ratings

The jumper settings of the DC driver, very important parameters, are shown in Table 2.5.

Jumper Number	Position	Function
J1A and J1B	115	select the input AC line voltage as 115V
J3	90V	to match our motor armature voltage (85V)
J7	Spd	select control-speed mode
J6	NTCL	don't stop the motor if it is overloaded after a delay

Table 2.5: DC driver jumper settings

2.1.2.3. **Digital/Analog Card Setup** The D/A card used for interfacing with the host computer is the ADIO 1600 D/A. This is a 100kHz multifunction analog/digital I/O card, as shown in Fig. 2.7

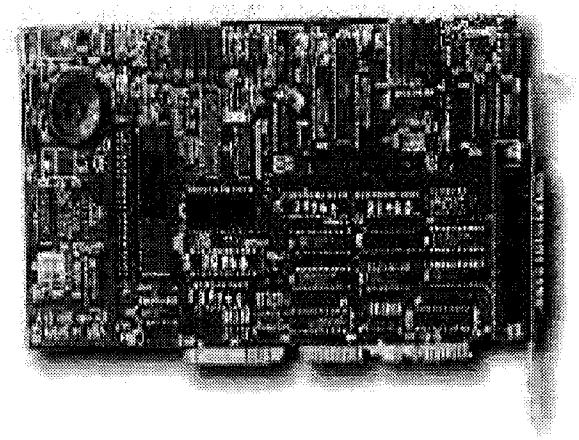


Figure 2.7: ADIO 1600 Analog/Digital I/O card

For our needs, the jumpers are set as follows:

- 4 analog inputs: +/− 10V, differential (bipolar range for negative speed and positive speed).
- 1 analog output: +/− 10V, (bipolar range 10V) for DC driver speed voltage control.
- 1 digital output: 0 low and −5 high to enable the DC driver.
- We have to set the memory port address. In general, the address 0x300 is for prototype card.

- Setting analog inputs
  - Move jumper JP4 to bipolar position.
  - Move jumpers JP8 and JP9 to differential position.
- Setting analog outputs
  - Select bipolar position
  - Select 10V position
  - Select "off" position

After setting the jumpers on this card, we need to set different channels for analog inputs and outputs, as shown in Table 2.6.

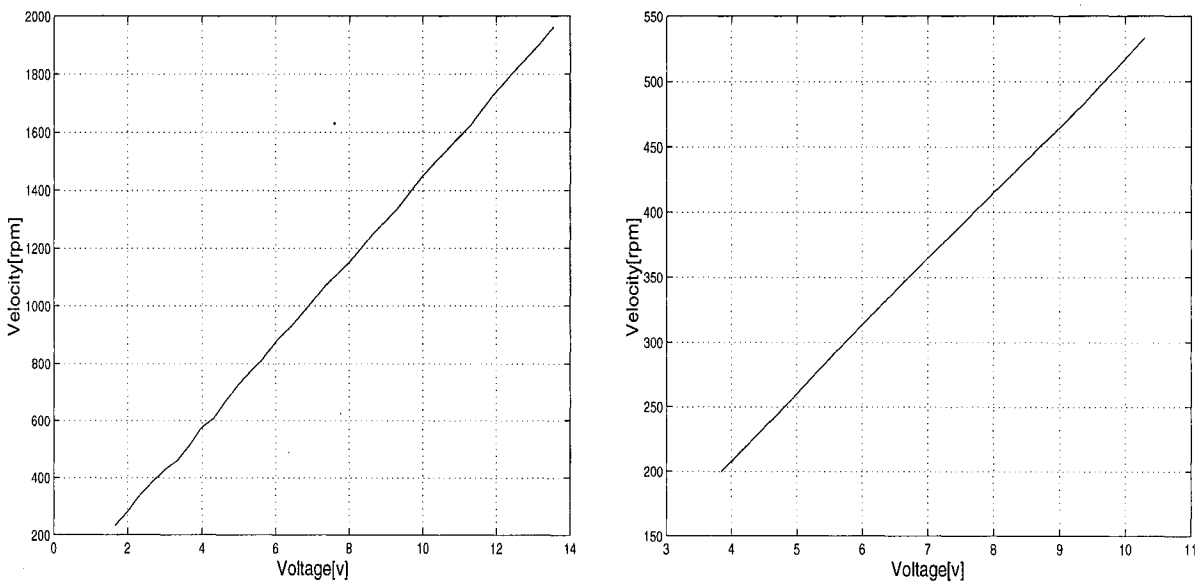
Pin Number	Name	Function
19	L.L GND	Low Level Ground
9	DAC0 out	D/A Channel 0 Output
7	COM	Power Common (Logic GND)
23	OP0	Digital Output Channel 0
37	CH0 HI	Channel 0 Analog high input
18	CH0 LO	Channel 0 Analog low input
36	CH1 HI	Channel 1 Analog high input
17	CH1 LO	Channel 1 Analog low input
35	CH2 HI	Channel 2 Analog high input
16	CH2 LO	Channel 2 Analog low input
34	CH3 HI	Channel 3 Analog high input
15	CH3 LO	Channel 3 Analog low input
28	L.L GND	Low Level Ground
29	L.L GND	Low Level Ground

Table 2.6: Pin assignments of termination board panel

The analog inputs are configured as differential channels. A differential input uses two input channels and the signal corresponds to the voltage difference between these two channels. Pins numbers 9 and 19, to which we will connect the DC driver control connector, allow us to start and stop the DC motor.

2.1.2.4. **Data-Acquisition System** The data-acquisition system includes: the DC motor tachometer, termed *input velocity tachometer*; an input shaft torque sensor, termed *input torque sensor*; an output shaft tachometer, referred to as *output velocity tachometer*; and an output shaft torque sensor, called *output torque sensor*. Since the outputs of two tachometers and the torque sensors are voltages, for obtaining the respective velocities and the torque, the voltage readouts at the A/D converter channel have to be divided by the sensitivity constant of each sensor.

In order to find the precise sensitivity constants, the calibrations were performed. Tachometer calibration was conducted with the aid of a multi-voltmeter and a hand-held mechanical tachometer. The DC motor was run with different velocities. The motor velocities and the output voltages from the tachometers were measured with the hand-held tachometer and the multi-voltmeter, respectively. Plots of data obtained from the calibration are shown in Fig. 2.8.



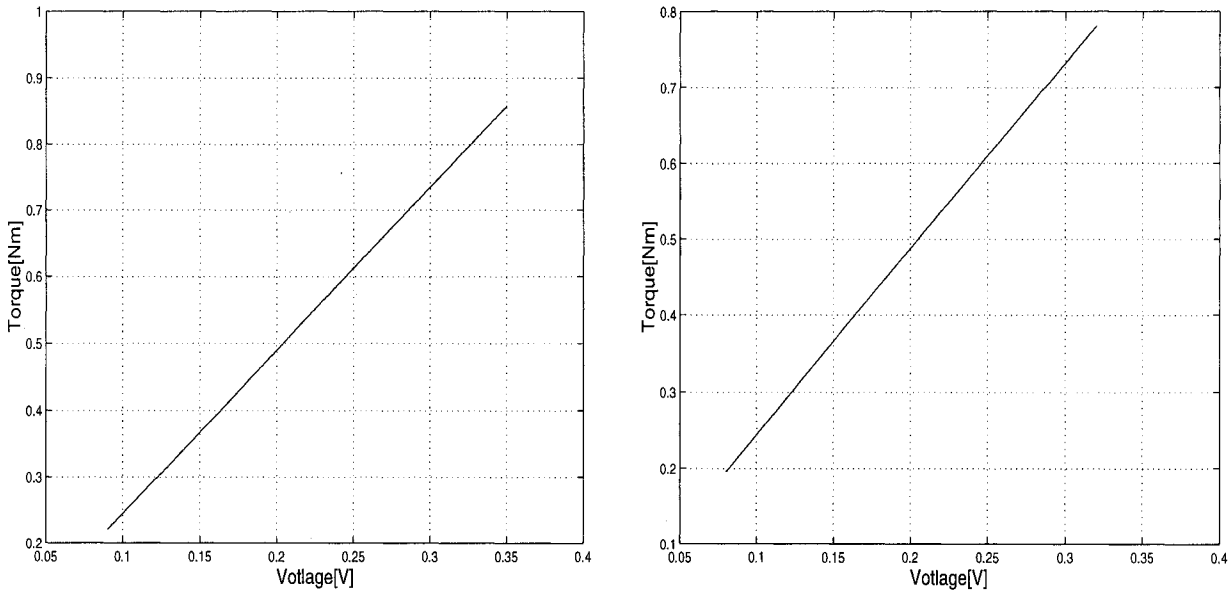
(a) **(b)**

Figure 2.8: Tachometer calibrations: (a) the DC motor tachometer; (b) the output shaft tachometer

Furthermore, torque sensor calibration was conducted with the aid of the multi-voltmeters and the DC motor tachometer. The DC motor was run with different velocities measured with the DC motor tachometer, and the DC motor input voltages and currents were measured with a multi-voltmeter. The DC motor output torques were calculated based on the formula below

$$\tau_m = \frac{UI\eta}{\omega_m} \tag{2.3}$$

where I and U are DC motor input current and voltage, η is the efficiency of the motor, and ω_m is DC motor output velocity. Moreover, the output voltage from torque sensors was measured with another multi-voltmeter. Plots of the data obtained from the calibration are shown in Fig. 2.9.



(a) (b)

Figure 2.9: Tachometer calibrations: (a) the input shaft torque sensor; (b) the output shaft torque sensor

Finally, the sensitivity constants of tachometers and torque sensors are obtained and

recorded in Table 2.7.

Sensor	Sensitivity Constant
Input velocity tachometer	6.9[V/krpm]
Output velocity tachometer	19.3[V/krpm]
Input Torque sensor	2.45[mV/Nm]
Output Torque sensor	2.44[mV/Nm]

Table 2.7: Sensitivity constants for the sensors

2.2. Software

The control and data-acquisition C++ program, called *Simple*, is included in Appendix A. *Simple* can control the DC motor and acquire data for analysis. Using this program, the DC motor can be controlled with arbitrary velocity profiles, preferably of the constant types.

2.2.1. Human-Machine Interface The interface is produced by C++ code to create a graphical X-Windows environment, as shown in Fig. 2.10. X-Windows offers the user various options to perform operations such as sending signals to operate the DC motor. X-Windows provides push-button options activated by the use of a computer mouse. X-Windows comprises five menu boxes, each menu box supplied with command buttons. The menu boxes occupy the top section of X-Windows, which are

- File: There are `New, Open, Save, Save As` and `Exit` commands in this menu box, which permit one to load or save measurement data and exit the program.
- Edit: This menu box includes `Undo, Cut, Copy` and `Paste` commands, which permit one to edit one's input signal data.
- View: In this menu box, one can select to display or hide the `toolbar` and `Status Bar` when the program is run.

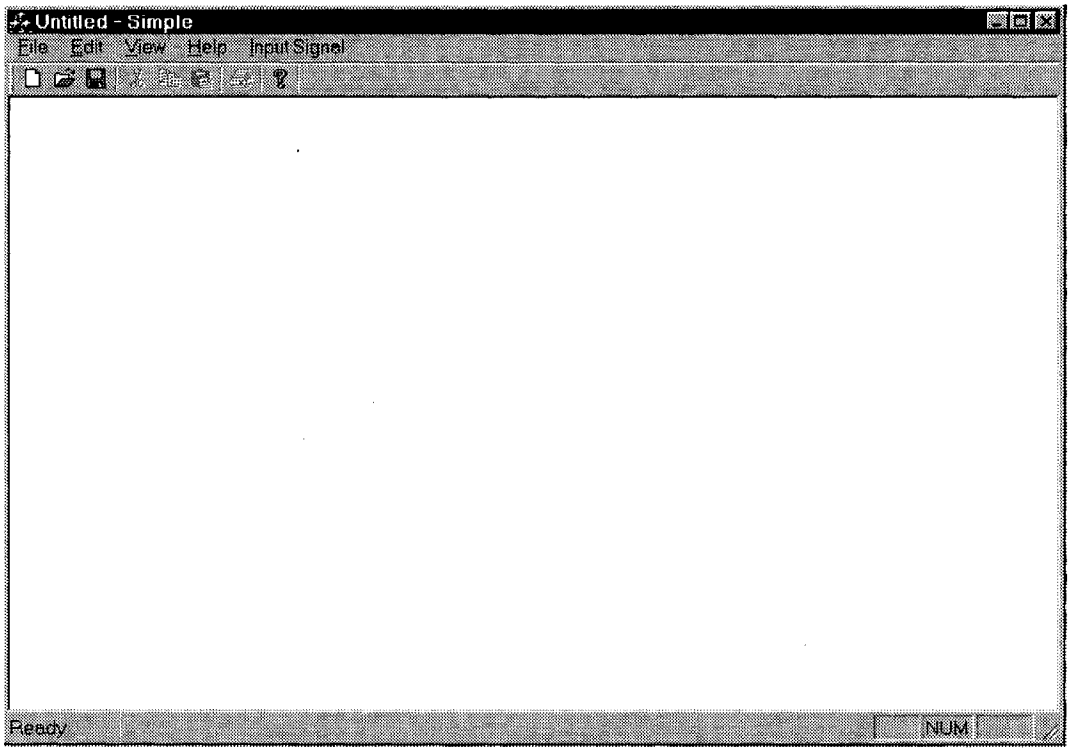


Figure 2.10: The human-machine interface under the X-Windows environment

- Help: This is a very useful menu box. When the mouse is placed on each button, the explanation of this program is displayed.
- Input Signal: This is the most important menu box in the program, which one can input constant voltage signal to control the DC motor with constant velocity.

CHAPTER 3

Modelling of the Testbed

In this chapter, the Newton-Euler and the Lagrange formulations are used in developing the mathematical model for the testbed of the Speed-o-Cam prototype; the iconic model of the testbed is displayed in Fig. 2.2. Based on this model, the transfer function of output velocity to input velocity is derived.

3.1. Iconic Model of the Testbed

In the iconic model of the prototype, as illustrated in Fig. 2.2, all springs and dashpots are torsional elements that are assumed to operate within their linear range.

3.1.1. List of Symbols

k_1 : torsional stiffness of coupling 1

k_2 : torsional stiffness of coupling 2

k_3 : stiffness of the roller pins of the transmission

k_4 : torsional stiffness of coupling 3

c_1 : coefficient of viscous damping of the motor

c_2 : coefficient of viscous damping of coupling 1

c_3 : coefficient of viscous damping of coupling 2

c_4 : coefficient of viscous damping of coupling 3

c_5 : coefficient of viscous damping of the load shaft

J_1 : moment of inertia of the motor

J_2 : moment of inertia of the cam shaft

J_3 : moment of inertia of the roller-carrying disk

J_4 : moment of inertia of the load

$\dot{\theta}_1$: angular velocity of the motor

$\dot{\theta}_c$: angular velocity of the high-speed shaft between coupling 1 and coupling 2

$\dot{\theta}_2$: angular velocity of the high-speed shaft

$\dot{\theta}_3$: angular velocity of the low-speed shaft
of the the transmission under rigid-pin conditions

$\dot{\theta}_4$: angular velocity of the low-speed shaft due to roller-pin flexibility

$\dot{\theta}_5$: angular velocity of the load

τ_m: motor-supplied torque

τ_1: the torque of the high-speed shaft

τ_2: the torque of the low-speed shaft

N : integer giving the speed-reduction ratio as $N : 1$

3.2. Main Assumptions

We assume that all shafts, connectors, cams and load are rigid bodies. Moreover, the rollers are rigidly attached to the roller-carrying disk. However, in practice, the rollers are not rigidly attached to this disk by means of the roller-pins. Thus, the velocities of the high-speed shaft and low-speed shaft do not precisely obey the relation of the speed reduction $N : 1$. In order to find the velocity of the low-speed shaft $\dot{\theta}_4$, and the stiffness of the roller-pins k_3, we model the roller-carrying disk as a massless plate P, and lump its mass in the rotor of moment of inertia J_3, as shown in Fig. 2.2.

3.3. Derivation of the Mathematical Model

We formulate the governing equations using a Lagrangian approach (Seely, 1964). If we let $\boldsymbol{q}$ be the vector of independent generalized coordinates, then the Langrange

equation of the system is given below:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\boldsymbol{q}}}\right) - \frac{\partial L}{\partial \boldsymbol{q}} = \frac{\partial}{\partial \dot{\boldsymbol{q}}}(\Pi - \Delta) \tag{3.1}$$

where

L : the Lagrangian of the system, given by $L = T - V$

T : the total kinetic energy of the system

V : the total potential energy of the system

Π : the power supplied to the system by motor sources

Δ : the dissipation function associated with all dashpots in the system

The total kinetic energy is obtained by adding the individual energies of the various bodies comprising the system. The obvious choice of generalized coordinates is the set of angular displacements θ_i, for $i = 2, ..., 5$ and c, the total kinetic energy of the system hence being

$$T = \frac{1}{2}J_1\dot{\theta}_1^2 + \frac{1}{2}J_2\dot{\theta}_2^2 + \frac{1}{2}J_3\dot{\theta}_3^2 + \frac{1}{2}J_4\dot{\theta}_5^2 \tag{3.2}$$

Furthermore, the system has elastic elements, while gravity does not intervene. The potential energy is, therefore,

$$V = \frac{1}{2}k_1(\theta_1 - \theta_c)^2 + \frac{1}{2}k_2(\theta_c - \theta_2)^2 + \frac{1}{2}k_3(\theta_3 - \theta_4)^2 + \frac{1}{2}k_4(\theta_4 - \theta_5)^2 \tag{3.3}$$

Since all the input power comes via the motor-supplied torque τ, which is delivered at an angular velocity $\dot{\theta}_1$, we have

$$\Pi = \tau\,\dot{\theta}_1 \tag{3.4}$$

With regard to viscous damping only, the dissipation function is

$$\Delta = \frac{1}{2}c_1\dot{\theta}_1^2 + \frac{1}{2}c_2(\dot{\theta}_1 - \dot{\theta}_c)^2 + \frac{1}{2}c_3(\dot{\theta}_c - \dot{\theta}_2)^2\frac{1}{2}c_4(\dot{\theta}_4 - \dot{\theta}_5)^2 + \frac{1}{2}c_5\dot{\theta}_5^2 \tag{3.5}$$

The speed ratio being $1/N$, under the assumption that $\theta_3 = 0$, when $\theta_2 = 0$, we have,

$$\theta_3 = \frac{1}{N}\theta_2 \tag{3.6}$$

It is then apparent that out of the five generalized coordinates θ_i, for $i = 2, ..., 5$ and c, only four are independent, and hence, the system has four degrees of freedom. Substituting eqs.(3.2)–(3.6) into eq.(3.1), the Langrange equation becomes

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{C}\dot{\boldsymbol{q}} + \boldsymbol{K}\boldsymbol{q} = \boldsymbol{f}(t) \tag{3.7}$$

where $\boldsymbol{q}$ is the 4-dimensional vector of generalized coordinates and $\boldsymbol{f}(t)$ is the 4-dimensional vector of generalized force, namely,

$$\boldsymbol{q} = \begin{bmatrix} \theta_c \\ \theta_2 \\ \theta_4 \\ \theta_5 \end{bmatrix}, \qquad \boldsymbol{f}(t) = \begin{bmatrix} k_1\theta_1 + c_2\dot{\theta}_1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Moreover, $\boldsymbol{M}$, $\boldsymbol{C}$ and $\boldsymbol{K}$ are the 4×4 mass, damping and stiffness matrices, respectively, as given below:

$$\boldsymbol{M} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & J_2 + J_3/N^2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & J_4 \end{bmatrix}$$

$$\boldsymbol{C} = \begin{bmatrix} c_2 + c_3 & -c_3 & 0 & 0 \\ -c_3 & c_3 & 0 & 0 \\ 0 & 0 & c_4 & -c_4 \\ 0 & 0 & -c_4 & c_4 + c_5 \end{bmatrix}$$

$$\boldsymbol{K} = \begin{bmatrix} k_1 + k_2 & -k_2 & 0 & 0 \\ -k_2 & k_2 + k_3/N^2 & -k_3/N & 0 \\ 0 & -k_3/N & k_3 + k_4 & -k_4 \\ 0 & 0 & -k_4 & k_4 \end{bmatrix}$$

3.4. Derivation of the Transfer Function Model

Transfer function models are widely used in many single-input single-output (SISO) systems. The transfer function is a compact representation of the differential equation describing the ratio of the output variable to the input variable, these variables being expressed as polynomial functions of s, after Laplace-transforming the input-output differential equation (Savant, 1964).

We find the transfer function of the system at hand by choosing θ_1 as the input variable and θ_5 as the output variable. To this end, we first take the Laplace transform of eqs.(3.7), with zero initial conditions. The Laplace transforms in terms of $\theta_i(t)$ are

$$k_1\Theta_1 + c_2\Theta_1 s = (c_2 + c_3)\Theta_c s - c_3\Theta_2 s + (k_1 + k_2)\Theta_c - k_2\Theta_2 \tag{3.8a}$$

$$\begin{aligned} 0 = {} & (J_2 + 1/N^2 J_3)\Theta_2 s^2 - c_3\Theta_c s + c_3\Theta_2 s - k_2\Theta_c \\ & + (k_2 + 1/N^2 k_3)\Theta_2 - 1/N\Theta_4 k_3 \end{aligned} \tag{3.8b}$$

$$0 = c_4\Theta_4 s - c_4\Theta_5 s - 1/N k_3\Theta_2 + (k_3 + k_4)\Theta_4 - k_4\Theta_5 \tag{3.8c}$$

$$0 = J_4\Theta_5 s^2 - c_4\Theta_4 s + (c_4 + c_5)\Theta_5 s - k_4\Theta_4 + k_4\Theta_5 \tag{3.8d}$$

We derive the desired transfer function, namely,

$$G(s) \equiv \frac{\Theta_5(s)}{\Theta_1(s)} = \frac{P(s)}{Q(s)} \tag{3.9}$$

where

$$\begin{aligned} P(s) &= k_3N(-c_2c_3c_4s^3 + (k_1c_3c_4 + c_2c_3k_4 + c_2k_2c_4)s^2 \\ &\quad + (k_1c_3k_4 + k_1k_2c_4 + c_2k_2k_4)s + k_1k_2k_4) \qquad (3.10a) \\ Q(s) &= Bs^6 + Cs^5 + Ds^4 + Es^3 + Fs^2 + Gs + H \qquad (3.10b) \end{aligned}$$

with coefficients B, C, D, E, F, G and H given by

$$B = c_3J_2N^2c_4J_4 + J_3c_4c_2J_4 + c_3J_3c_4J_4 + J_2N^2c_4c_2J_4 \qquad (3.10c)$$

$$\begin{aligned} C &= J_2N^2k_4c_2J_4 + J_3k_4c_2J_4 + c_3J_3k_3J_4 + k_2J_3c_4J_4 + k_2J_2N^2c_4J_4 \\ &\quad + c_3J_2N^2c_4c_5 + c_3J_3k_4J_4 + J_3c_4k_1J_4 + J_3k_3c_2J_4 + J_2N^2c_4c_2c_5 \\ &\quad + J_2N^2k_3c_2J_4 + J_2N^2c_4k_1J_4 + c_2c_3N^2c_4J_4 + c_3J_2N^2k_4J_4 \\ &\quad + c_3J_2N^2k_3J_4 + J_3c_4c_2c_5 + c_3J_3c_4c_5 \qquad (3.10d) \end{aligned}$$

$$\begin{aligned} D &= J_3k_3c_2c_5 + J_3k_4k_1J_4 + J_2N^2k_4c_2c_5 + J_2N^2k_3k_1J_4 + k_2N^2c_4c_2J_4 \\ &\quad + k_3c_4c_2J_4 + J_2N^2c_4k_1c_5 + J_2N^2k_3c_2c_4 + J_2N^2k_3c_2c_5 \\ &\quad + J_3c_4k_1c_5 + J_3k_3c_2c_4 + J_3k_4c_2c_5 + J_3k_3k_1J_4 + k_2J_2N^2k_4J_4 \\ &\quad + c_3J_2N^2k_3c_5 + c_3J_3k_3c_4 + c_3J_3k_3c_5 + k_2J_3k_3J_4 + k_2J_2N^2c_4c_5 \\ &\quad + c_2c_3N^2c_4c_5 + c_2c_3N^2k_4J_4 + k_2J_3c_4c_5 + c_3J_3k_4c_5 + k_2J_3k_4J_4 \\ &\quad + k_1c_3N^2c_4J_4 + k_2J_2N^2k_3J_4 + c_2c_3N^2k_3J_4 + c_3J_2N^2k_3c_4 \\ &\quad + c_3k_3c_4J_4 + c_3J_2N^2k_4c_5 + J_2N^2k_4k_1J_4 \qquad (3.10e) \end{aligned}$$

$$
\begin{aligned}
E &= J_3k_4k_1c_5 + k_3c_4c_2c_5 + J_2N^2k_4k_1c_5 + k_3c_4k_1J_4 + k_2N^2c_4c_2c_5 \\
&\quad + N^2k_4c_2k_2J_4 + k_2N^2c_4k_1J_4 + k_3k_4c_2J_4 + J_2N^2k_3c_2k_4 \\
&\quad + J_2N^2k_3k_1c_5 + J_3k_3k_1c_5 + N^2k_3c_2k_2J_4 + J_3k_3c_2k_4 + J_3k_3k_1c_4 \\
&\quad + k_1c_3N^2k_4J_4 + k_2J_2N^2k_4c_5 + k_2J_3k_3c_4 + k_2J_3k_3c_5 + k_2k_3c_4J_4 \\
&\quad + c3J_3k_3k_4 + c_3J_2N^2k_3k_4 + c_3k_3k_4J_4 + k_2J_3k_4c_5 + c_3k_3c_4c_5 \\
&\quad + k_2J_2N^2k_3c_4 + k_2J_2N^2k_3c_5 + c_2c_3N^2k_3c_4 + J_2N^2k_3k_1c_4 \\
&\quad + k_1c_3N^2k_3J_4 + c_2c_3N^2k_4c_5 + c_2c_3N^2k_3c_5 + k_1c_3N^2c_4c_5 \qquad (3.10f) \\
F &= J_2N^2k_3k_1k_4 + J_3k_3k_1k_4 + k_2k_3k_4J_4 + k_2k_3c_4c_5 \\
&\quad + k_1c_3N^2k_3c_4 + N^2k_3c_2k_2c_4 + c_2c_3N^2k_3k_4 + k_2J_3k_3k_4 \\
&\quad + N^2k_3k_1k_2J_4 + k_3k_4c_2c_5 + N^2k_4c_2k_2c_5 + k_1c_3N^2k_4c_5 \\
&\quad + N^2k_4k_1k_2J_4 + k_3c_4k_1c_5 + k_1c_3N^2k_3c_5 + N^2k_3c_2k_2c_5 \\
&\quad + k_2J_2N^2k_3k_4 + k_3k_4k_1J_4 + k_2N^2c_4k_1c_5 + c_3k_3k_4c_5 \qquad (3.10g) \\
G &= N^2k_3c_2k_2k_4 + k_2k_3k_4c_5 + N^2k_3k_1k_2c_4 + k_3k_4k_1c_5 \\
&\quad + k_1c_3N^2k_3k_4 + N^2k_4k_1k_2c_5 + N^2k_3k_1k_2c_5 \qquad (3.10h) \\
H &= k_1k_2k_3k_4N^2 \qquad (3.10i)
\end{aligned}
$$

3.5. Inertial Properties of the Testbed Elements

In the dynamic analysis of the Speed-o-Cam testbed, to be undertaken in Chapter 4, the inertial properties of the rotating parts are required. For a uniform solid, these properties are the volume, the centroid coordinates, and the inertia tensor. We assume that all rotating parts, the cam shafts and the roller-carrying disk have undergone a full static and dynamic balancing, and hence, the only relevant property for our analysis is the moment of inertia about the axis of rotation.

Pro/ENGINEER Model Analysis (MA) is a useful tool, which includes facilities to

calculate the volumetric properties of general solids. The cam input shaft and the roller-carrying disk were generated in Pro/ENGINEER, as shown in Fig. 3.1

(a) (b)

Figure 3.1: Solid models of (a) the cam shaft; and (b)the roller-carrying disk

The inertial-parameter values, as reported by Pro/Engineer, are recorded in Table 3.1.

	Mass (kg)	Moment of Inertia ($kgmm^2$)
Cam Shaft	0.5676	128.75022
Roller-Carrying Disk	1.9414	4778.4172

Table 3.1: Moments of inertia of the input cam shaft and the roller-carrying disk with output shaft

CHAPTER 4

Dynamic Analysis of the Testbed

4.1. System Identification

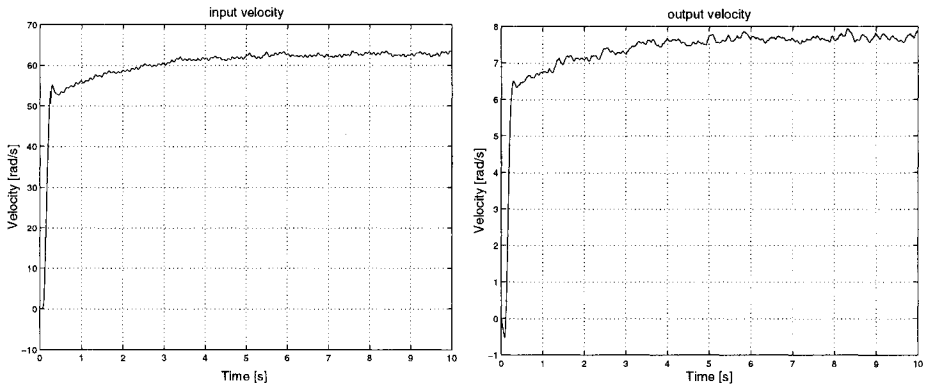
The system identification problem consists in estimating the parameters of a system model based on observed input-output data. The procedure to determine these parameters involves three basic ingredients:

- Design an experiment and collect input-output data from measurements.
- Examine the data. Polish the data so as to remove trends and outliers, and select useful portions of the original data, possibly applying filters to enhance important frequency ranges.
- Compute the best model parameters according to the input-output data.

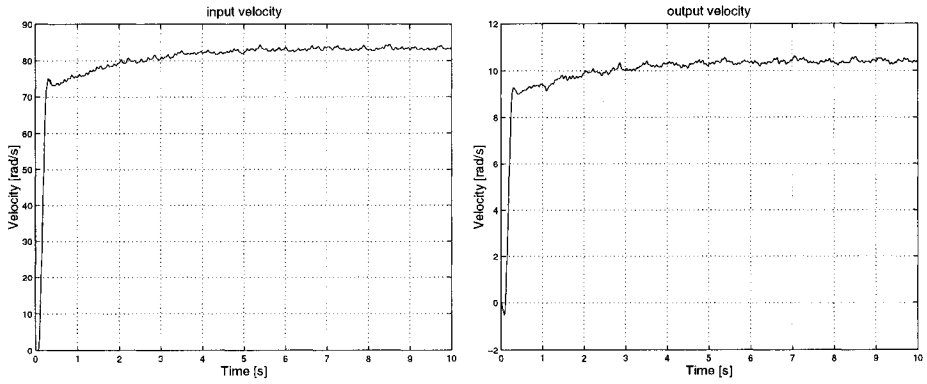
In the previous chapters, the testbed and the data acquisition program were described. In order to identify the testbed model parameters, we will begin with input-output data analysis; to this end, the numerical results of the testbed transfer function are obtained.

4.1.1. Input-Output Data Analysis Good and simple insight into a system's dynamic properties is obtained by looking at its step response or impulse response (Wolovich, 1994). In our case, the step response of the system is used for this purpose; that is, we study the output as a result of a step input. Since practically we only can change the supply voltage to the motor, a true step in the input velocity

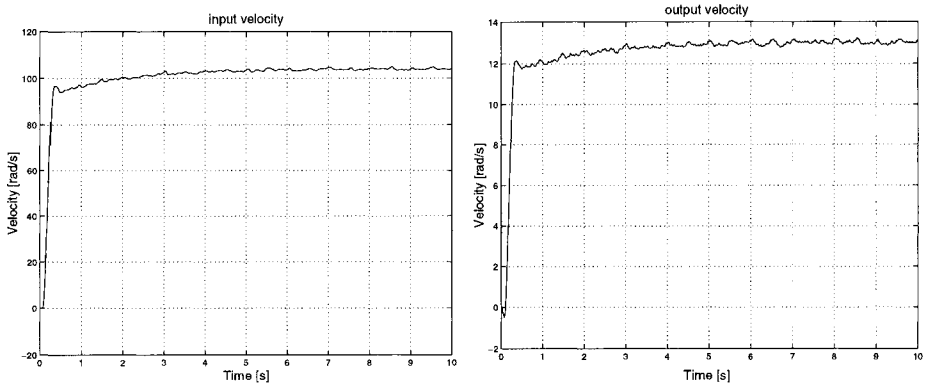
is not possible. However, as made apparent from the Fig. 2.5, the motor shows a a predominantly static behaviour. We can thus reliably accept that a voltage step to the motor leads to a step input in the angular velocity of the motor shaft. In our experiments, constant voltage input signals are sent from the host computer to the motor, so as to produce motor velocities of 600rpm, 800rpm, 1000rpm, 1200rpm and 1400rpm. $\dot{\theta}_1$, the input variable, is measured from the *input velocity tachometer*, and $\dot{\theta}_5$, the output variable, is measured from the *output velocity tachometer*. We collected data for a time span of 10s with sampling rate of 10ms, thus acquiring 10000 sets of input-output data, which were then saved in a file for each experiment; these were then analyzed with MATLAB. Moreover, for each motor speed, we conducted five experiments. The best data group with least measurement noise is selected out of five experiments by means of the fast Fourier transform (Kahaner, Moler and Nash, 1977). Furthermore, in addition to the decisions required for the transfer function selection and validation, the data may need to be processed carefully. Hence, we removed the outliers to refine data and filtered the high-frequency noise by means of Butterworth filter. The plots for the best groups of data sets of input and output angular velocities with different velocities are obtained, finally, as shown in Fig. 4.1.



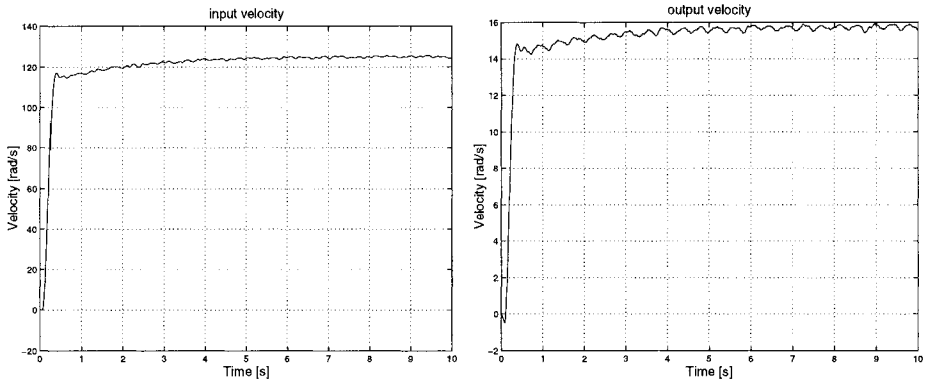
(a)


input velocity
Velocity [rad/s]
Time [s]
output velocity
Velocity [rad/s]
Time [s]

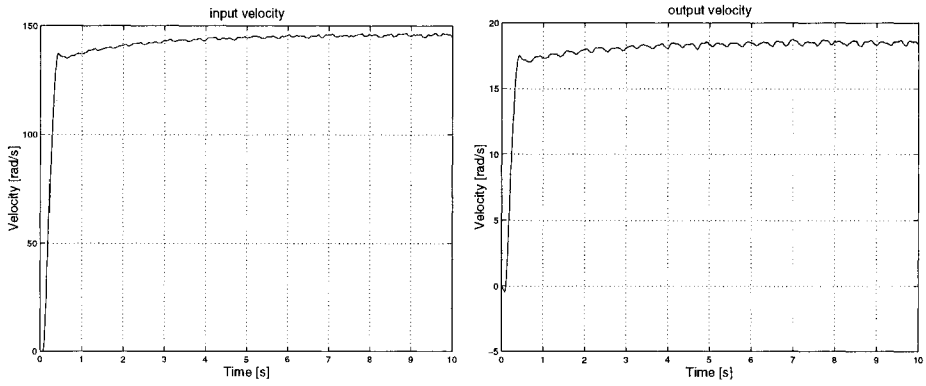
(b)


input velocity
Velocity [rad/s]
Time [s]
output velocity
Velocity [rad/s]
Time [s]

(c)


input velocity
Velocity [rad/s]
Time [s]
output velocity
Velocity [rad/s]
Time [s]

(d)



(e)

Figure 4.1: The signals recorded of input and output velocities for input velocity values of: (a) 600rpm; (b) 800rpm; (c) 1000rpm; (d) 1200rpm; (e) 1400rpm

Five groups of input-output data sets where the setting speed varied from 600rpm to 1400rpm were acquired. We did the system identification analysis in the time-domain not the frequency-domain, because the capability limit of our DC driver card, which can not produce a sinusoidal voltage signal to the DC motor. We analyzed these data by means of the System Identification toolbox in MATLAB (Ljung 2000). The transfer function, termed the *experimental transfer function* (ETF), was obtained in the form

$$H(s) = \frac{N(s)}{D(s)}$$

where

$$\begin{aligned} N(s) &= -72.47s^5 - 16360s^4 - 1.685 \times 10^7 s^3 + 8.027 \times 10^9 s^2 + 1.239 \times 10^{12} s \\ &\quad +7.022 \times 10^{13} \\ D(s) &= s^6 + 2056s^5 + 1.557 \times 10^6 s^4 + 5.165 \times 10^8 s^3 + 1.058 \times 10^{11} s^2 + 1.443 \times 10^{13} s \\ &\quad +5.608 \times 10^{14} \end{aligned}$$

The associated Bode plots and the plot of pole-zero map are shown in Fig. 4.2 and Fig. 4.3.

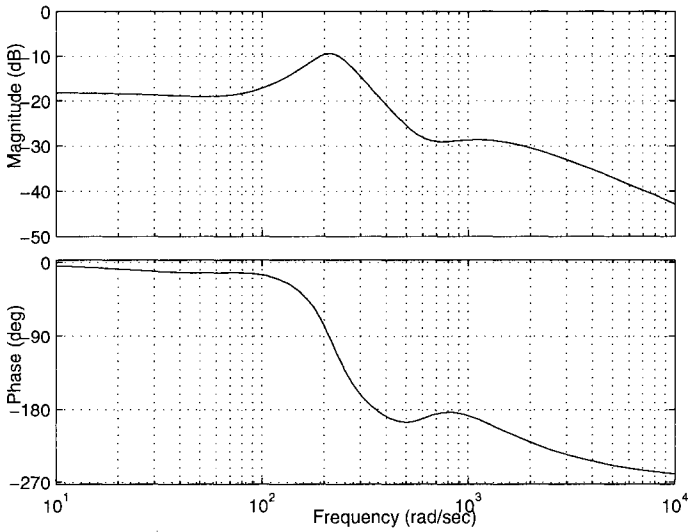


Figure 4.2: The Bode plots of the system

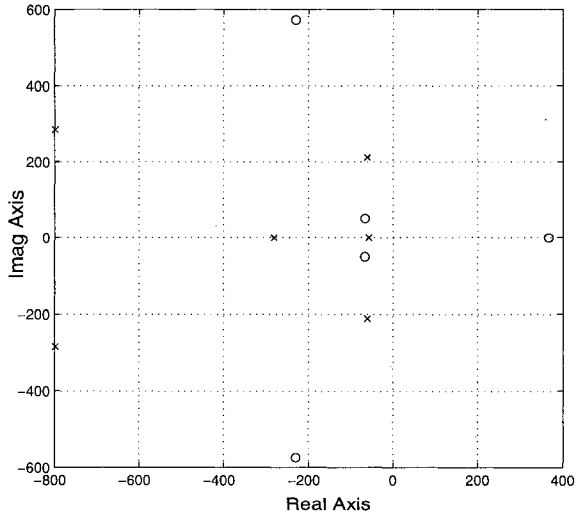


Figure 4.3: The pole-zero map of the system

To verify this transfer function, we use the *Final-Value Theorem* (Cannon, 1967) with the step time response function, thus obtaining

$$\lim_{s\to 0}[s\frac{H(s)}{s}] = \frac{7.022 \times 10^{13}}{5.608 \times 10^{14}} = 0.1252$$

Apparently, this value is very close to $1/N$, the speed reduction ratio of 1/8, thus validating the above transfer function.

To obtain the natural frequencies and damping ratios of the testbed, we factored the sixth order polynomial of the denominator of ETF into three quadratic polynomials, with real coefficients, upon pairing the complex-conjugate roots (D'Azzo and Houpis, 1988), thus obtaining

$$\begin{aligned} M_1 &= s^2 + 338.55s + 16104.4250 \\ M_2 &= s^2 + 120.56s + 48463.2713 \\ M_3 &= s^2 + 1596.88s + 718452.3737 \end{aligned}$$

The natural frequencies and damping ratios of the system are shown in Table 4.1.

Mode	Natural frequencies	Damping ratios
The first mode	19.92Hz	1.3338
The second mode	35.0547Hz	0.2738
The third mode	134.9707Hz	0.94198

Table 4.1: The natural frequencies and damping ratios of the system

4.1.2. Approximating the Numerator Polynomial of ETF With the *model transfer function* (MTF), derived in Chapter 3, we have two expressions for the transfer function of the Speed-o-Cam testbed. The numerator of the MTF is a third-degree polynomial, while the numerator of the ETF is a fifth-degree polynomial. In order to match the coefficients of the two expressions to identify the mechanical parameters of the prototype, the Chebyshev algorithm (Henrici 1964) was employed

to approximate the fifth-degree polynomial of the numerator of the ETF by a third-degree polynomial.

The Chebyshev algorithm is described below: If we let $Q(x)$ be a polynomial of degree n with leading coefficient 1,

$$Q(x) = x^n + a_{n-1}x^{n-1} + ... + a_0$$

then the approximate polynomial $P(x)$ of degree $n-1$ is chosen so that the maximum value of error $|Q(x)-P(x)|$ is minimized; $P(x)$is given by

$$P(x) = Q(x) - L(x) \tag{4.1}$$

where

$$L(x) = \frac{1}{2^{n-1}}T_n(x)$$

with T_n denoting the nth Chebyshev polynomial.

The original fifth-degree numerator polynomial of the ETF is

$$\begin{aligned} Q(s) = & -72.47s^5 - 16360s^4 - 1.685 \times 10^7 s^3 + 8.027 \times 10^9 s^2 + 1.239 \times 10^{12} s \\ & + 7.022 \times 10^{13} \end{aligned}$$

Using eq.(4.1) and taking the interval [−1, 1], we obtain

$$\begin{aligned} P_4(s) = & -16360s^4 - 1.685009059 \times 10^7 s^3 + 8.027 \times 10^9 s^2 + 1.23 \times 10^{12} s \\ & + 7.022 \times 10^{13} \end{aligned}$$

Thus approximating the original polynomial with a fourth-degree polynomial. By using the approximating process of eq.(4.1) recursively in the same interval, finally, the third-degree polynomial approximation is obtained as

$$\begin{aligned} P_3(s) = & -1.685009059 \times 10^8 s^3 + 8.026983640 \times 10^9 s^2 + 1.23 \times 10^{12} s \\ & + 7.022 \times 10^{13} \end{aligned} \tag{4.2}$$

The ETF becomes, therefore,

$$H(s) = \frac{P_3(s)}{D(s)} \tag{4.3}$$

Where:

$$\begin{aligned} P_3(s) &= -1.685009059 \times 10^8 s^3 + 8.026983640 \times 10^9 s^2 + 1.23 \times 10^{12} s \\ &\quad +7.022 \times 10^{13} \\ D(s) &= s^6 + 2056 s^5 + 1.557 \times 10^6 s^4 + 5.165 \times 10^8 s^3 + 1.058 \times 10^{11} s^2 \\ &\quad +1.443 \times 10^{13} s + 5.608 \times 10^{14} \end{aligned}$$

Moreover, to verify this new transfer function by means of *Final-Value Theorem*

$$\lim_{s \to 0}[s\frac{H(s)}{s}] = \frac{7.022 \times 10^{13}}{5.608 \times 10^{14}} = 0.1252$$

The validity of this transfer function does not change. The coefficients of the two transfer functions can now be matched.

4.2. Mechanical Parameter Identification

We identify the mechanical parameters by matching the coefficients of the two transfer functions. The mechanical parameters that we want to identify are k_1, k_2, k_3, k_4, c_2, c_3, c_4 and c_5, which occur in the eleven-coefficient matching equations. Furthermore, it is apparent that out of eleven equations, only ten are independent and hence, the equations that we will use are

$$-1.685 \times 10^8 = -k_3 c_2 c_3 c_4 N \tag{4.4a}$$

$$8.027 \times 10^9 = k_3 N(k_1 c_3 c_4 + c_2 c_3 k_4 + c_2 k_2 c_4) \tag{4.4b}$$

$$0.123 \times 10^{13} = k_3 N(k_1 c_3 k_4 + k_1 k_2 c_4 + c_2 k_2 k_4) \tag{4.4c}$$

$$1 = c_3 J_2 N^2 c_4 J_4 + J_3 c_4 c_2 J_4 + c_3 J_3 c_4 J_4 + J_2 N^2 c_4 c_2 J_4 \tag{4.4d}$$

$$
\begin{aligned}
2056 &= J_2N^2k_4c_2J_4 + J_3k_4c_2J_4 + c_3J_3k_3J_4 + k_2J_3c_4J_4 \\
&\quad + c_3J_2N^2c_4c_5 + c_3J_3k_4J_4 + J_3c_4k_1J_4 + J_3k_3c_2J_4 \\
&\quad + J_2N^2k_3c_2J_4 + J_2N^2c_4k_1J_4 + c_2c_3N^2c_4J_4 \\
&\quad + c_3J_3c_4c_5 + c_3J_2N^2k_3J_4 + J_3c_4c_2c_5 + k_2J_2N^2c_4J_4 \\
&\quad + J_2N^2c_4c_2c_5 + c_3J_2N^2k_4J_4 \qquad (4.4e) \\
1.557 \times 10^6 &= J_3k_3c_2c_5 + J_3k_4k_1J_4 + J_2N^2k_4c_2c_5 + J_2N^2k_3k_1J_4 \\
&\quad + k_2N^2c_4c_2J_4 + k_3c_4c_2J_4 + J_2N^2c_4k_1c_5 + J_2N^2k_3c_2c_4 \\
&\quad + J_2N^2k_3c_2c_5 + J_2N^2k_4k_1J_4 + J_3c_4k_1c_5 + J_3k_3c_2c_4 \\
&\quad + J_3k_3k_1J_4 + k_2J_2N^2k_4J_4 + c_3J_2N^2k_3c_4 + c_3J_2N^2k_3c_5 \\
&\quad + c_3J_3k_3c_4 + c_3J_3k_3c_5 + k_2J_3k_3J_4 + k_2J_2N^2c_4c_5 \\
&\quad + c_3J_2N^2k_4c_5 + c_2c_3N^2c_4c_5 + c_2c_3N^2k_4J_4 + k_2J_3c_4c_5 \\
&\quad + c_3J_3k_4c_5 + k_2J_3k_4J_4 + c_3k_3c_4J_4 + k_1c_3N^2c_4J_4 \\
&\quad + c_2c_3N^2k_3J_4 + J_3k_4c_2c_5 + k_2J_2N^2k_3J_4 \qquad (4.4f) \\
5.165 \times 10^8 &= J_3k_4k_1c_5 + k_3c_4c_2c_5 + J_2N^2k_4k_1c_5 + k_3c_4k_1J_4 \\
&\quad + N^2k_4c_2k_2J_4 + k_2N^2c_4k_1J_4 + k_3k_4c_2J_4 \\
&\quad + J_2N^2k_3k_1c_5 + J_3k_3k_1c_5 + N^2k_3c_2k_2J_4 + J_3k_3c_2k_4 \\
&\quad + k_1c_3N^2k_3J_4 + k_1c_3N^2k_4J_4 + k_2J_2N^2k_4c_5 + k_2J_3k_3c_4 \\
&\quad + k_2k_3c_4J_4 + c_2c_3N^2k_4c_5 + c3J_3k_3k_4 + c_3J_2N^2k_3k_4 \\
&\quad + k_2J_3k_4c_5 + c_3k_3c_4c_5 + k_1c_3N^2c_4c_5 + k_2J_2N^2k_3c_4 \\
&\quad + c_2c_3N^2k_3c_4 + c_2c_3N^2k_3c_5 + J_2N^2k_3k_1c_4 + k_2N^2c_4c_2c_5 \\
&\quad + J_2N^2k_3c_2k_4 + J_3k_3k_1c_4 + k_2J_3k_3c_5 + c_3k_3k_4J_4 \\
&\quad + k_2J_2N^2k_3c_5 \qquad (4.4g)
\end{aligned}
$$

$$
\begin{aligned}
1.058 \times 10^{11} &= J_2N^2k_3k_1k_4 + J_3k_3k_1k_4 + k_2k_3k_4J_4 + k_2k_3c_4c_5 \\
&\quad + k_1c_3N^2k_3c_4 + N^2k_3c_2k_2c_4 + c_2c_3N^2k_3k_4 + k_2J_3k_3k_4 \\
&\quad + N^2k_3k_1k_2J_4 + k_3k_4c_2c_5 + N^2k_4c_2k_2c_5 + k_1c_3N^2k_4c_5 \\
&\quad + N^2k_4k_1k_2J_4 + k_3c_4k_1c_5 + k_1c_3N^2k_3c_5 + N^2k_3c_2k_2c_5 \\
&\quad + k_2J_2N^2k_3k_4 + k_3k_4k_1J_4 + k_2N^2c_4k_1c_5 + c_3k_3k_4c_5 \qquad (4.4h) \\
1.443 \times 10^{13} &= N^2k_3c_2k_2k_4 + k_2k_3k_4c_5 + N^2k_3k_1k_2c_4 + k_3k_4k_1c_5 \\
&\quad + k_1c_3N^2k_3k_4 + N^2k_4k_1k_2c_5 + N^2k_3k_1k_2c_5 \qquad (4.4i) \\
5.608 \times 10^{14} &= k_1k_2k_3k_4N^2 \qquad (4.4j)
\end{aligned}
$$

We thus have ten nonlinear equations with eight unknowns, the system being *overdetermined.* We will solve this overdetermined system by means of least squares (Lawson and Hanson 1974).

In order to obtain reliable solutions with minimum errors, we convert the above eight equations into dimensionless form (Lu, 1979); we do this by introducing two basic parameters. These parameters are the *characteristic stiffness* k_{char} and the *characteristic damping* c_{char}, so that,

$$k_i = \kappa_i k_{char},\ i = 1,\ 2,\ 3,\ 4 \qquad (4.5)$$

$$c_n = \gamma_n c_{char},\ n = 2,\ 3,\ 4,\ 5 \qquad (4.6)$$

where

$$
\begin{aligned}
k_{char} &= \frac{T}{\omega\tau} \\
c_{char} &= \sqrt{J_{mr}k_{char}}
\end{aligned}
$$

J_{mr} : moment of inertia of the motor rotor

k_{char} : torsional stiffness of the motor shaft

T : rated motor torque

ω : rated motor speed

τ : mechanical time constant of the motor

c_{char} : coefficient of viscous damping of the motor

κ_i : dimensionless torsional stiffness, $i = 1, 2, 3, 4$

γ_n : dimensionless coefficient of viscous damping, $n = 2, 3, 4, 5$

The motor specifications described in Table 4.2,

Parameters	Value
T	980.8Nmm
ω	18000deg/s
τ	0.007sec
J_{mr}	3050.6Nmms^2/deg
k_{char}	7.8Nmm/deg
c_{char}	154Nmms/deg

Table 4.2: Unit parameters specifications

Therefore, eqs.(4.4a)–(4.4j) become, in dimensionless form

$$\phi_1 = \gamma_2\gamma_3\gamma_4\kappa_3 - 0.7394 \tag{4.7a}$$

$$\phi_2 = \kappa_1\gamma_3\gamma_4\kappa_3 + \gamma_2\gamma_3\kappa_4\kappa_3 + \gamma_2\kappa_2\gamma_4\kappa_3 - 695.3967 \tag{4.7b}$$

$$\phi_3 = \kappa_1\gamma_3\kappa_4\kappa_3 + \kappa_1\kappa_2\gamma_4\kappa_3 + \gamma_2\kappa_2\kappa_4\kappa_3 - 2103829.767 \tag{4.7c}$$

$$\phi_4 = \gamma_3\gamma_4 + \gamma_4\gamma_2 - 0.2196 \times 10^{-12} \tag{4.7d}$$

$$\begin{aligned}\phi_5 = {} & \kappa_4\gamma_2 + \gamma_3\kappa_3 + \kappa_2\gamma_4 + 0.2061\gamma_3\gamma_4\gamma_5 + \gamma_3\kappa_4 + \gamma_4\kappa_1 + \kappa_3\gamma_2 \\ & + 0.2061\gamma_4\gamma_2\gamma_5 + 14.9475\gamma_2\gamma_3\gamma_4 - 0.8912 \times 10^{-8}\end{aligned} \tag{4.7e}$$

$$\begin{aligned}\phi_6 &= \kappa_4\gamma_2\gamma_5 + 72.5224\kappa_2\gamma_4\gamma_2 + 72.5224\gamma_2\gamma_3\kappa_3 + 2.1332\gamma_3\kappa_3\gamma_4\gamma_3\kappa_4\gamma_5 \\ &\quad + 72.5224\gamma_2\gamma_3\kappa_4 + \gamma_4\kappa_1\gamma_5 + 2.1332\kappa_3\gamma_2\gamma_4 + \kappa_3\gamma_2\gamma_5 + 4.8518\kappa_3\kappa_1 \\ &\quad + 4.8518\kappa_2\kappa_4 + 4.8518\kappa_2\kappa_3 + 4.8518\kappa_4\kappa_1 + 72.5224\kappa_1\gamma_3\gamma_4 \\ &\quad + 14.9475\gamma_2\gamma_3\gamma_4\gamma_5 + \gamma_3\kappa_3\gamma_5 + \kappa_2\gamma_4\gamma_5 - 0.6465 \times 10^{-3} \qquad (4.7f)\end{aligned}$$

$$\begin{aligned}\phi_7 &= \kappa_2\gamma_4\kappa_1 + 0.0294\kappa_2\kappa_3\gamma_4 + 0.0138\kappa_2\kappa_4\gamma_5 + 0.2061\gamma_2\gamma_3\gamma_4\kappa_3 \\ &\quad + 0.0032\gamma_3\kappa_3\gamma_4\gamma_5 + 0.2061\gamma_2\gamma_3\kappa_4\gamma_5 + 0.2061\kappa_1\gamma_3\gamma_4\gamma_5 \\ &\quad + 0.2061\kappa_2\gamma_4\gamma_2\gamma_5 + 0.0138\kappa_3\kappa_1\gamma_5 + \kappa_4\gamma_2\kappa_2 + \kappa_3\gamma_2\kappa_2 \\ &\quad + 0.0294\gamma_3\kappa_3\kappa_4 + 0.0138\kappa_4\kappa_1\gamma_5 + 0.0294\kappa_3\gamma_4\kappa_1 \\ &\quad + 0.0032\kappa_3\gamma_4\gamma_2\gamma_5 + 0.0138\kappa_2\kappa_3\gamma_5 + 0.2061\gamma_2\gamma_3\kappa_3\gamma_5 \\ &\quad + \kappa_1\gamma_3\kappa_3 + 0.0294\kappa_3\kappa_4\gamma_2 + \kappa_1\gamma_3\kappa_4 - 0.0584 \qquad (4.7g)\end{aligned}$$

$$\begin{aligned}\phi_8 &= \kappa_3\kappa_1\kappa_4 + \kappa_2\kappa_3\kappa_4 + 0.1095\kappa_2\kappa_3\gamma_4\gamma_5 + 7.0072\kappa_1\gamma_3\gamma_4\kappa_3 \\ &\quad + 7.0072\gamma_2\gamma_3\kappa_4\kappa_3 + 33.9976\kappa_3\kappa_1\kappa_2 + 0.1095\kappa_3\kappa_4\gamma_2\gamma_5 \\ &\quad + 7.0072\kappa_1\gamma_3\kappa_4\gamma_5 + 7.0072\kappa_2\gamma_4\kappa_1\gamma_5 + 33.9976\kappa_4\kappa_1\kappa_2 \\ &\quad + 7.0072\kappa_1\gamma_3\kappa_3\gamma_5 + 7.0072\kappa_3\gamma_2\kappa_2\gamma_5 + 0.1095\gamma_3\kappa_3\kappa_4\gamma_5 \\ &\quad + 7.0072\gamma_2\kappa_2\gamma_4\kappa_3 + 7.0072\kappa_4\gamma_2\kappa_2\gamma_5 + 0.1095\kappa_3\gamma_4\kappa_1\gamma_5 \\ &\quad - 8028.2194 \qquad (4.7h)\end{aligned}$$

$$\begin{aligned}\phi_9 &= \gamma_2\kappa_2\kappa_4\kappa_3 + 0.0156\kappa_2\kappa_3\kappa_4\gamma_5 + \kappa_1\kappa_2\gamma_4\kappa_3 + 0.0156\kappa_3\kappa_4\kappa_1\gamma_5 \\ &\quad + \kappa_1\gamma_3\kappa_4\kappa_3 + \kappa_4\kappa_1\kappa_2\gamma_5 + \kappa_3\kappa_1\kappa_2\gamma_5 - 3085189.384 \qquad (4.7i)\end{aligned}$$

$$\phi_{10} = \kappa_1\kappa_2\kappa_3\kappa_4 - 2139332168 \qquad (4.7j)$$

Furthermore, all κ_1, κ_2, κ_3, κ_4, γ_2, γ_3, γ_4 and γ_5 are positive quantities, i.e,

$$h_1 \equiv \kappa_1 > 0 \tag{4.8a}$$

$$h_2 \equiv \kappa_2 > 0 \tag{4.8b}$$

$$h_3 \equiv \kappa_3 > 0 \tag{4.8c}$$

$$h_4 \equiv \kappa_4 > 0 \tag{4.8d}$$

$$h_5 \equiv \gamma_2 > 0 \tag{4.8e}$$

$$h_6 \equiv \gamma_3 > 0 \tag{4.8f}$$

$$h_7 \equiv \gamma_4 > 0 \tag{4.8g}$$

$$h_8 \equiv \gamma_5 > 0 \tag{4.8h}$$

The problem can be formulated as an inequality-constrained nonlinear least-square problem, which consists in finding the least-square error f in the approximation of an overdetermined system of nonlinear equations, $\boldsymbol{\phi}(\mathrm{x}) = \mathbf{0}$, i.e. (Teng and Angeles 2001),

$$f(\mathbf{x}) = \frac{1}{2}\boldsymbol{\phi}^T\mathbf{W}\boldsymbol{\phi} \quad \rightarrow \quad \min_{\mathbf{x}} \tag{4.9}$$

subject to the linear inequality constraints

$$\mathbf{h}(\mathbf{x}) > \mathbf{0} \tag{4.10}$$

where $\boldsymbol{\phi}$ is an 10-dimensional vector with components $\phi_i(\mathbf{x}), i = 1, \ldots, 10$, eqs.(4.7a)–(4.7j), while $\mathbf{x}$ is a 8-dimensional vector, where $x_i, i = 1, \ldots, 8$, denote the eight unknowns κ_1, κ_2, κ_3, κ_4, γ_2, γ_3, γ_4, γ_5. Moreover, $\mathbf{h}$ is a 8-dimensional vector of linear inequality constraints with its elements $h_i(\mathbf{x}), i = 1, \ldots, 8$, as defined in eqs.(4.8a)–(4.8h), the 10×10 weighting matrix $\mathbf{W}$ being diagonal, i.e.,

$$\mathbf{W} = \mathrm{diag}(w_1, w_2 ... w_{10})$$

Where

$$w_i = \frac{1}{\|\phi_i(\mathbf{x})\|^2_{\mathbf{x}=\mathbf{x}_0}}$$

with $i = 1, \ldots, 10$ and $\mathbf{x}_0$ is the initial guesses. The values of $\mathbf{x}_0$ and w_i are listed in Tables 4.3 and 4.4.

	Initial guesses		**Initial guesses**
κ_1	158	γ_2	2
κ_2	121	γ_3	2
κ_3	120000	γ_4	6
κ_4	300	γ_5	9

Table 4.3: Initial guesses

w_1	1.06048×10^{-15}
w_2	1.88352×10^{-17}
w_3	4.39456×10^{-16}
w_4	2.86344×10^{-04}
w_5	2.28412×10^{-15}
w_6	6.60810×10^{-16}
w_7	1.93624×10^{-13}
w_8	8.17289×10^{-16}
w_9	6.14479×10^{-16}
w_{10}	2.75912×10^{-15}

Table 4.4: Numerical values of the weights

The solutions of eq.(4.9) produced with MATLAB are recorded in Table 4.5.

	Dimensionless mechanical parameters
κ_1	154.9487
κ_2	121.0256
κ_3	117440
κ_4	299.8718
γ_2	0.0012
γ_3	0.0011
γ_4	4.6669
γ_5	8.9994
f_min	4.3392×10^{-7}

Table 4.5: Dimensionless mechanical parameters

To verify that the results of Table 4.5 are indeed optimum, we evaluted the gradient of the objective function at the solution obtained (Luenberger, 1968). The values are

$$\nabla f = \mathbf{\Phi}^T \mathbf{W} \boldsymbol{\phi} = \begin{bmatrix} 0.856394 \\ -0.009569 \\ -0.013391 \\ 0.911531 \\ 0.331273 \\ 0.499969 \\ 0.013964 \\ 0.003965 \end{bmatrix}$$

Where $\mathbf{\Phi}$ is the gradient of $\boldsymbol{\phi}$ with respect to $\mathbf{x}$. Although ∇f is not exactly zero, its components are reasonably "small", which indicates an acceptable solution.
By using eqs.(4.5) and (4.6), the mechanical parameters are obtained, as shown in Table 4.6.

Mechanical parameters	Value
k_1	1.21Nm/deg
k_2	0.94Nm/deg
k_3	916.03Nm/deg
k_4	2.34Nm/deg
c_2	1.85×10^{-4}Nms/deg
c_3	1.69×10^{-4}Nms/deg
c_4	0.72Nms/deg
c_5	1.39Nms/deg

Table 4.6: The mechanical parameters

4.2.1. Analysis of the Results Parameters k_1, k_2 and k_4 are the torsional stiffnesses of couplings 1, 2 and 3 respectively. We conducted the torsional stiffness test of the three couplings in the Solid Mechanics Laboratory of the Department of Civil Engineering and Applied Mechanics. The test machine with the test setup is shown in Fig. 4.4

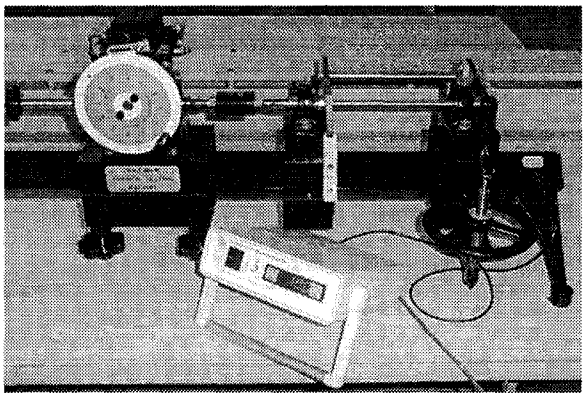


Figure 4.4: Torsional stiffness test machine

The measured stiffness and identified Stiffnesses of couplings 1, 2 and 3 with the in corresponding errors are shown in Tables 4.7

Coupling	**Measured Stiffness** Nm/deg	**Identified Stiffness** Nm/deg	**errors**
Coupling 1	1.24	1.21	2.42%
Coupling 2	0.95	0.94	1.05%
Coupling 3	1.92	2.34	21.87%

Table 4.7: The stiffness of the couplings

It is apparent that the errors between measured and identified stiffness values of couplings 1, 2 are quite small, while the error of coupling 3 is ten times as large, although still within reasonable values. The reason for the large error associated with coupling 3 is that we assumed the steel connector and the load shaft to be rigid bodies; we only considered the stiffness of coupling 3 in the equivalent output shaft of our testbed. Therefore, some errors were brought in. Moreover, in order to verify the reliability and sensitivity of the mechanical parameters, we changed the weighting factors to

$$w_i = \frac{1}{\|\phi_i(\mathbf{x})\|_{\mathbf{x}=\mathbf{x}_0}}$$

with $i = 1, \ldots, 10$ and defined $\mathbf{x}_0$ as the same initial guess. The values of the new weights w_i are recorded in Tables 4.8

w_1	3.2565×10^{-8}	w_6	2.5706×10^{-8}
w_2	4.3399×10^{-9}	w_7	4.4003×10^{-7}
w_3	2.0963×10^{-8}	w_8	2.85882×10^{-8}
w_4	1.6900×10^{-2}	w_9	2.4788×10^{-8}
w_5	4.7792×10^{-7}	w_{10}	5.2527×10^{-8}

Table 4.8: Numerical values of the new weights

The solutions reported by MATALB are recorded in Table 4.9.

	Dimensionless mechanical parameters
κ_1	270.1154
κ_2	97.6923
κ_3	123360
κ_4	298.4308
γ_2	0.0509
γ_3	0.0542
γ_4	0.4239
γ_5	0.7312
$f_{\min}$	1.5628

Table 4.9: Dimensionless mechanical parameters

The mechanical parameters derived from Table 4.9 are shown in Table 4.10.

Mechanical parameters	**Value**
k_1	2.11Nm/deg
k_2	0.76Nm/deg
k_3	962.23Nm/deg
k_4	2.33Nm/deg
c_2	7.83×10^{-3}Nms/deg
c_3	8.35×10^{-3}Nms/deg
c_4	6.53×10^{-2}Nms/deg
c_5	0.11Nms/deg

Table 4.10: The mechanical parameters

Comparing the mechanical parameters recorded in Tables 4.6 and 4.10, we can find that the values of the torsional stiffness are more reliable than the values of the coefficient of viscous damping, especially, the torsional stiffness of the Speed-o-Cam prototype k_3, in which we are interested. The values of the coefficient of viscous damping are too sensitive to the weighting matrix. The reason of this phenomenon, which cause some errors in our results, is that we obtained the transfer function of the testbed by using a time-domain analysis, rather than frequency-domain analysis.

Therefore, some useful information may have been lost, while some errors are brought in (Rohrs, Melsa and Schultz, 1993).

CHAPTER 5

Efficiency of the Speed-o-Cam Prototype

Efficiency is an important factor in continuously operated drive trains. Over a period of years, losses of a even low percentage of the transmitted power can be expensive. Low efficiency may bring about additional penalties, since friction losses introduce heat, which must be dissipated. The usual definition of efficiency η is

$$\eta = \frac{\text{power output}}{\text{power input}}$$

The Speed-o-Cam testbed shown in Fig. 5.1 is used for the determination of the Speed-o-Cam efficiency.

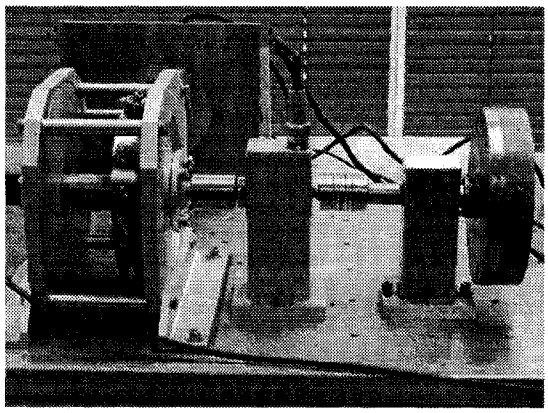


Figure 5.1: Speed-o-Cam power efficiency testbed

The iconic model of the testbed shown in Fig. 2.2 is reproduced in Fig. 5.2 for quick reference.

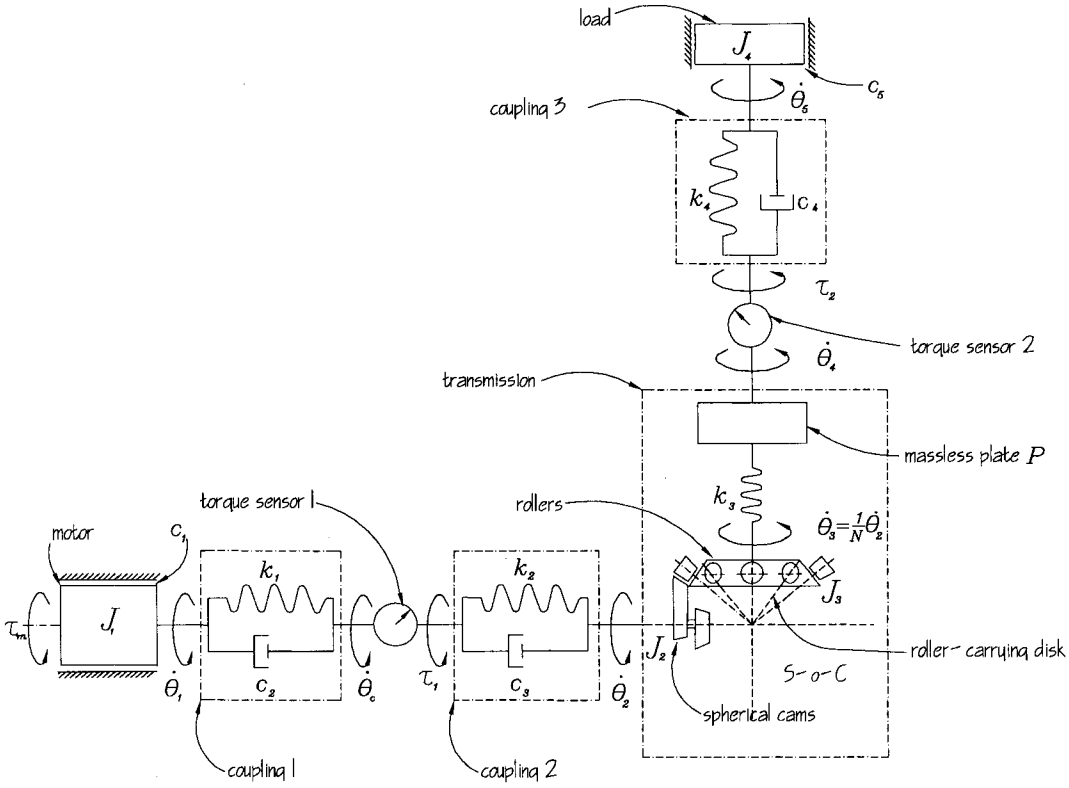


Figure 5.2: Iconic model of Speed-o-Cam power efficiency testbed

In order to conduct the efficiency experiments, we measured $\dot{\theta}_1$ and τ_1 with an *input velocity tachometer* and an *input torque sensor*. Moreover $\dot{\theta}_4$ and τ_2 are the output variables, measured with an *output velocity tachometer* and an *output torque sensor*. The efficiency is computed as

$$\eta = \frac{P_o}{P_i} \tag{5.1}$$

where P_i and P_o are the input and output powers, respectively. These values are calculated based on the formulas,

$$P_i = \dot{\theta}_1 \tau_1 \tag{5.2}$$

$$P_o = \dot{\theta}_4 \tau_2 \tag{5.3}$$

We select three experimental speeds, 500rpm, 1000rpm and 1500rpm, to conduct the efficiency experiments. The efficiency η_1, that we call *system efficiency* under those velocities is obtained by using eqs.(5.1)–(5.3). The efficiencies are shown in Table 5.1 together with the experimental values of input velocity, output velocity, input torque and output torque.

speed seting	experimental values				η_1
rpm	$\dot{\theta}_1$ rad/s	$\dot{\theta}_4$ rad/s	τ_1 Nm	τ_2 Nm	
500	52.82309	6.4175	0.6900	4.8994	86.265%
1000	106.4647	13.2392	0.6846	4.8230	87.601%
1500	158.2754	19.5378	0.4128	2.9265	87.512%

Table 5.1: The *system efficiency* with experimental value specifications

5.1. The Efficiencies of the Couplings

Two couplings are involved in the efficiency experiments, namely, coupling 1 and coupling 2, as shown in Fig. 5.2. Coupling 1 is connected to the DC motor with the input shaft torque sensor and coupling 2 is used to connect the input shaft torque sensor with the input shaft of the Speed-o-Cam transmission. Both couplings dissipate a part of the power generated by the motor. Therefore, a series of experiments should be performed to estimate the efficiencies η_{c1} and η_{c2} of the two couplings. These values should be taken into account while calculating the Speed-o-Cam efficiency.
To determine the efficiency of the couplings, we used the same Speed-o-Cam testbed with a different setup, shown in Fig. 5.3.

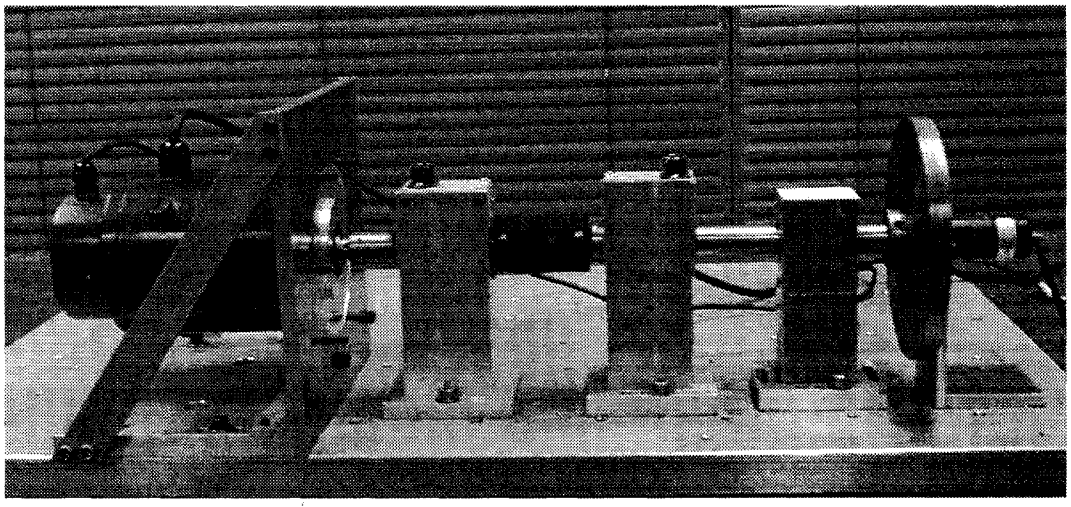


Figure 5.3: The coupling testbed

The iconic model of the coupling testbed is shown in Fig. 5.4

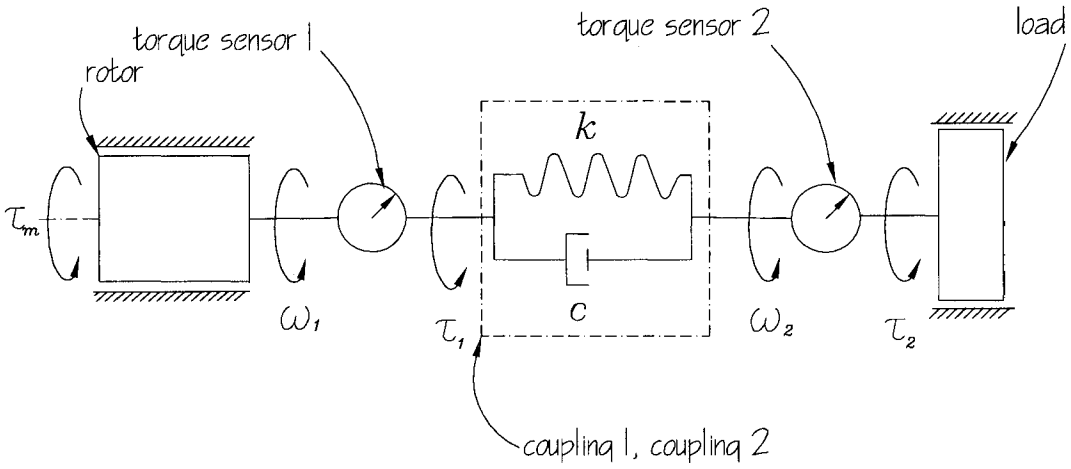


Figure 5.4: Iconic model of the coupling testbed

In Fig. 5.4:

k : torsional stiffness of the test coupling

c : coefficient of viscous damping of the test coupling

ω_1 : angular velocity of the rotor

ω_2 : angular velocity of the load shaft

τ_m: rotor-supplied torque

τ_1: the input torque

τ_2: the output torque

The couplings are tested separately. Each experiment is carried out with three different input velocities, which are the same as those we set in the *system efficiency* experiments. We chose ω_1 and τ_1 as input variables measured from an *input velocity tachometer* and an *input torque sensor.* Moreover ω_2 and τ_2, the output variables, are measured from an *output velocity tachometer* and an *output torque sensor.* The efficiency η_c of each coupling is thus obtained as

$$\eta_c = \frac{P_{co}}{P_{ci}} = \frac{\omega_2 \tau_2}{\omega_1 \tau_1} \tag{5.4}$$

where

P_{ci} : the input power of the test coupling

P_{co} : the output power of the test coupling

ω_1 : input power of motor

ω_2 : output power of coupling

The experimental values of the velocities and torques with the efficiencies of two couplings were obtained as recorded in Tables 5.2 and 5.3.

speed setting	experimental values				η_{c1}
rpm	ω_1 rad/s	ω_2 rad/s	τ_1 Nm	τ_2 Nm	
500	52.4960	51.4783	0.3037	0.3040	98.158%
1000	105.3717	103.9033	0.2712	0.2714	98.679%
1500	157.9762	155.0182	0.2125	0.2121	98.312%

Table 5.2: Efficiency values with the experimental values of coupling 1

speed setting	experimental values				η_{c2}
rpm	ω_1 rad/s	ω_2 rad/s	τ_1 Nm	τ_2 Nm	
500	51.6826	50.5715	0.2821	0.2829	98.127%
1000	105.0463	103.0399	0.2577	0.2580	98.204%
1500	157.9762	155.0182	0.2121	0.2125	98.312%

Table 5.3: Efficiency values with the experimental values of coupling 2

5.2. The Efficiency of the Prototype

Finally, the efficiency of the Speed-o-Cam prototype, which we term η_s, is obtained as,

$$\eta_s = \frac{\eta_1}{\eta_{c1}\eta_{c2}} \tag{5.5}$$

The efficiency of the Speed-o-Cam prototype are recorded in Table 5.4.

experimental speed	η_s
500rpm	89.561%
1000rpm	90.397%
1500rpm	89.965%

Table 5.4: Efficiency of the Speed-o-Cam prototype

The average values of the efficiency of most common power transmission elements with same speed reduction ratio are: Spur gears with cut teeth, including bearings, 98%; bevel gears with cut teeth, including bearings, 94% (Oberg, Jones and Horton,

1988). If lubrication is poor or if the drive operates at lower-than-rated power, the efficiency will be lower. In high-reduction worm drives, losses may exceed 50% of the transmitted power. Traction drives have efficiencies from 65% to 90%, where the mechanical losses include the bearing friction, windage and brush-friction losses (Plint and Martyr 1995).

Comparing the average values of the efficiency of those power transmission elements with the efficiency of the Speed-o-Cam prototype, it can be said that the efficiency of the Speed-o-Cam prototype is on the low side. However, since this is the first spherical Speed-o-Cam prototype, its efficiency of around 90% can still be increased with proper redesign.

CHAPTER 6

Concluding Remarks

6.1. Conclusions

We introduced here a procedure to identify the mechanical parameters of a novel mechanical transmission, Speed-o-Cam, for speed reduction. The morphology of this transmission stems from that of pure-rolling indexing cam mechanisms; hence, Speed-o-Cam is backlash-free and frictionless.

In this thesis, we studied the dynamics of this novel mechanism. In particular, this thesis focuses on the aspects of both model development and mechanical parameter identification, as pertaining to a spherical prototype of Speed-o-Cam. In order to conduct the experiments on the prototype, a testbed was designed and fabricated. Then, the iconic model of the testbed was produced, all the related equations, such as the total kinetic energy and the total potential energy of the system were determined, so as to use a Lagrange formulation to develop the dynamic model. A mathematical model of the testbed was first derived under some main assumptions. Then, the model transfer function (MTF) of the output velocity to the input velocity of the system was derived based on the mathematical model of the testbed. Furthermore, a series of new experimental procedures for system identification were proposed and explained. As a result, the experimental transfer function (ETF) of the testbed was obtained by using the *system identification* toolbox in MATLAB. To this end, by

matching the coefficients of the two transfer functions at hand, the problem of mechanical parameter identification was investigated.

In consequence, investigations were carried out to identify the values of the relevant mechanical parameter of a spherical prototype, namely, its stiffness. In the process, the stiffness and damping parameters of the couplings of the testbed were also identified.

In addition, an important factor of power transmissions, which is the efficiency of the prototype was obtained in this thesis. The efficiency of the prototype was found to be of around 90%, which is still low, but acceptable for power transmissions.

6.2. Recommendations for Future Research

As an extension to the work reported here, several interesting issues remain to be further explored:

(i) Improving the stiffness of the spherical Speed-o-Cam. Since the spherical Speed-o-Cam has zero backlash and low friction losses, it should have a high stiffness. However, the stiffness parameter that finally we obtained is 916.03Nm-/deg, which is not as high as we desired. Hence, we should optimize the roller and the roller-disk design to improve the stiffness.

(ii) Improving the efficiency of the spherical Speed-o-Cam by enhancing the accuracy of manufacturing and assembly .

(iii) Comparing the Speed-o-Cam prototype with a gear-box speed reducer in terms of dynamic performance.

(iv) Investigating the dynamics of cam-roller speed reducers with different speed reduction ratios.

(v) Implementing a feedback control system of the testbed. The testbed is supplied with an open-loop control system and, hence, causes the operational speed to be inexact. We could add a compensating part in the control system by feeding back the input velocity from the *input velocity tachometer*, which will lead to the desired operational speed.

REFERENCES

Branch, M.A. and Grace, A., 1996, *Matlab Optimization Toolbox*, Math Works Inc., Natick, MA.

Buckingham, E., 1963, *Analytical Mechanics of Gears*, Dover Publication, Inc., New York.

Cannon, R.H., 1967, *Dynamics of Physical Systems*, McGraw-Hill Book Company, New York.

D'Azzo, J.J. and Houpis, C.H., *Linear Control System Analysis and Design Conventional and Modern*, McGraw-Hill Book Company, Singapore.

Dudley, D.W., 1991, *Handbook of Practial Gear*, McGraw-Hill Book Company, New York.

Denavit, J. and Hartenberg, R.S., 1964, *Kinematic Synthesis of Linkages*, McGraw-Hill, New York; Toroto.

Euler, L., 1754, *Elements of Algebra*, J. Johnson, London.

González-Palacios, M.A. and Angeles, J., 1993 *Cam Synthesis*, Kluwer Academic

Publishers, Dordrecht, Boston.

González-Palacios, M.A. and Angeles, J., 1999, "The Design of a Novel Mechanical Transmission for Speed Reduction," *ASME J. of Mechanical Design*, Vol. 121, no. 4, pp. 538-543.

González-Palacios, M.A. and Angeles, J., Sept. 10-13, 2000, "The novel design of a pure-rolling transmission to convert rotational into translational motion," *Proc. 2000 ASME Design Engineering Technical Conferences* Baltimore.

Henrici, P., 1964, *Elements of Numerical Analysis*, John Wiley & Sons, Inc. New York.

Hsia, T.C., 1977, *System Identification*, D.C. Heath and Company Lexington, Massachusettsm, Toronto.

Juvinall, R.C. and Marshek, K.M., 1991, *Fundamentals of Machine Component Design*, John Wiley & Sons. Inc New York.

Kahaner, D., Moler, C. and Nash, S., 1977, *Numerical Methods and Software*, Prentice Hall, Englewood Cliffs, New Jersey.

Lawson, C.L. and Hanson, R.J., 1974 *Solving Least Squares Problems*, Prentice-Hall, Inc., Englewood Cliffs, N.J.

Ljung, L., 1987, *System Identification Theory For the User*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Ljung, L., 2000, *System Identification Toolbox User's Guide*, Math Works, Inc., Natick, MA.

Lu, P.C., 1979, *Fluid Mechanics*, The Iowa State University Press, Ames, Iowa.

Luenberger, D.G., 1968, *Optimization by Vector Space Methods*, John Wiley & Sons, Inc. New York.

Oberg, E., Jones, F.D. and Horton, H.L., 1988, *Machinery's Handbook*, Industrial Press Inc., New York.

Ogata, K., 1998, *System Dynamics*, Prentice Hall, Upper Saddle River, New Jersey.

Patton, W.J., 1980, *Mechanical Power Transmission*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Plint, M. and Martyr, A.J., 1995, *Engine Testing Theory and Practice*, Butterworth-Heinemann, Oxford.

Rohrs, C.E., Melsa, J.L. and Schultz, D.G., 1993 *Linear Control Systems*, McGraw-Hill, Inc. New York.

Savant, C.J. Jr., 1964, *Control System Design*, McGraw-Hill Book Company, New York.

Seely, S., 1964, *Dynamic Systems Analysis*, Reinhold Publishing Corporation, New York.

Teng, C.P. and Angeles, J., 2001, *The Solution of Nonlinear Programming Problems with the Orthogonal Decomposition Algorithm –A Tutorial*, Department of Mechanical Engineering, McGill University, Montreal.

Tuttle, T. and Seering, W., 1993 "Kinematic Error, Compliance, and Friction in a Harmonic Drive Gear Transmission." *ASME Advances in Design Automation* Vol 65-1.

Thomson, W.T., 1981, *Theory of Vibration with Applications*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Wolovich, W.A., 1994, *Automatic Control Systems–Basic Analysis and Design*, Holt, Rinehart and Winston, Inc.

Yuen, W.K., 1996, *Dynamics and Control of a High-Rate Speed Reduction Cam Mechanism*, Thesis, Department of Mechanical Engineering, McGill University, Montreal.

APPENDIX A

The Control and Data-Acquisition C++ Program

```
========================================================================
       MICROSOFT FOUNDATION CLASS LIBRARY : Simple
========================================================================


AppWizard has created this Simple application for you.  This application
not only demonstrates the basics of using the Microsoft Foundation classes
but is also a starting point for writing your application.

This file contains a summary of what you will find in each of the files that
make up your Simple application.

Simple.dsp
    This file (the project file) contains information at the project level and
    is used to build a single project or subproject. Other users can share the
    project (.dsp) file, but they should export the makefiles locally.

Simple.h
    This is the main header file for the application.  It includes other
    project specific headers (including Resource.h) and declares the
    CSimpleApp application class.

Simple.cpp
    This is the main application source file that contains the application
    class CSimpleApp.

Simple.rc
    This is a listing of all of the Microsoft Windows resources that the
    program uses.  It includes the icons, bitmaps, and cursors that are stored
    in the RES subdirectory.  This file can be directly edited in Microsoft
    Visual C++.

Simple.clw
    This file contains information used by ClassWizard to edit existing
    classes or add new classes.  ClassWizard also uses this file to store
    information needed to create and edit message maps and dialog data
    maps and to create prototype member functions.

res\Simple.ico
    This is an icon file, which is used as the application's icon.  This
    icon is included by the main resource file Simple.rc.

res\Simple.rc2
    This file contains resources that are not edited by Microsoft
    Visual C++.  You should place all resources not editable by
    the resource editor in this file.

Simple.reg
    This is an example .REG file that shows you the kind of registration
    settings the framework will set for you.  You can use this as a .REG
    file to go along with your application or just delete it and rely
    on the default RegisterShellFileTypes registration.


/////////////////////////////////////////////////////////////////////////////

For the main frame window:

MainFrm.h, MainFrm.cpp
    These files contain the frame class CMainFrame, which is derived from
    CFrameWnd and controls all SDI frame features.

res\Toolbar.bmp
    This bitmap file is used to create tiled images for the toolbar.
    The initial toolbar and status bar are constructed in the CMainFrame
    class. Edit this toolbar bitmap using the resource editor, and
    update the IDR_MAINFRAME TOOLBAR array in Simple.rc to add
    toolbar buttons.
/////////////////////////////////////////////////////////////////////////////

AppWizard creates one document type and one view:

SimpleDoc.h, SimpleDoc.cpp - the document
    These files contain your CSimpleDoc class.  Edit these files to
    add your special document data and to implement file saving and loading
    (via CSimpleDoc::Serialize).

SimpleView.h, SimpleView.cpp - the view of the document
    These files contain your CSimpleView class.
```

CSimpleView objects are used to view CSimpleDoc objects.

///
Other standard files:

StdAfx.h, StdAfx.cpp
    These files are used to build a precompiled header (PCH) file
    named Simple.pch and a precompiled types file named StdAfx.obj.

Resource.h
    This is the standard header file, which defines new resource IDs.
    Microsoft Visual C++ reads and updates this file.

///
Other notes:

AppWizard uses "TODO:" to indicate parts of the source code you
should add to or customize.

If your application uses MFC in a shared DLL, and your application is
in a language other than the operating system's current language, you
will need to copy the corresponding localized resources MFC42XXX.DLL
from the Microsoft Visual C++ CD-ROM onto the system or system32 directory,
and rename it to be MFCLOC.DLL. ("XXX" stands for the language abbreviation.
For example, MFC42DEU.DLL contains resources translated to German.) If you
don't do this, some of the UI elements of your application will remain in the
language of the operating system.

///

```
// AmplitudeDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Simple.h"
#include "AmplitudeDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAmplitudeDlg dialog


CAmplitudeDlg::CAmplitudeDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CAmplitudeDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CAmplitudeDlg)
    m_amplitude = 0.0f;
    //}}AFX_DATA_INIT
}


void CAmplitudeDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAmplitudeDlg)
    DDX_Text(pDX, IDC_AMPLITUDE, m_amplitude);
    DDV_MinMaxFloat(pDX, m_amplitude, 0.2f, 2.3f);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CAmplitudeDlg, CDialog)
    //{{AFX_MSG_MAP(CAmplitudeDlg)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAmplitudeDlg message handlers
```

```
// FrequencyDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Simple.h"
#include "FrequencyDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CFrequencyDlg dialog


CFrequencyDlg::CFrequencyDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CFrequencyDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CFrequencyDlg)
    m_frequency = 0.0f;
    //}}AFX_DATA_INIT
}


void CFrequencyDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CFrequencyDlg)
    DDX_Text(pDX, IDC_FREQUENCY, m_frequency);
    DDV_MinMaxFloat(pDX, m_frequency, 5.e-002f, 5.f);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CFrequencyDlg, CDialog)
    //{{AFX_MSG_MAP(CFrequencyDlg)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFrequencyDlg message handlers
```

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Simple.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMainFrame

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    ON_WM_CREATE()
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
};

/////////////////////////////////////////////////////////////////////////////
// CMainFrame construction/destruction

CMainFrame::CMainFrame()

    // TODO: add member initialization code here

}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
        return -1;

    if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP
        | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
        !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
    {
        TRACE0("Failed to create toolbar\n");
        return -1;      // fail to create
    }

    if (!m_wndStatusBar.Create(this) ||
        !m_wndStatusBar.SetIndicators(indicators,
          sizeof(indicators)/sizeof(UINT)))
    {
        TRACE0("Failed to create status bar\n");
        return -1;      // fail to create
    }

    // TODO: Delete these three lines if you don't want the toolbar to
    //  be dockable
    m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
    EnableDocking(CBRS_ALIGN_ANY);
    DockControlBar(&m_wndToolBar);

    return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
```

```
    if( !CFrameWnd::PreCreateWindow(cs) )
        return FALSE;
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return TRUE;
}

/////////////////////////////////////////////////////////////////////////////
// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
    CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
    CFrameWnd::Dump(dc);
}

#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CMainFrame message handlers
```

```
// OffsetDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Simple.h"
#include "OffsetDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// COffsetDlg dialog


COffsetDlg::COffsetDlg(CWnd* pParent /*=NULL*/)
    : CDialog(COffsetDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(COffsetDlg)
    m_offset = 0.0f;
    //}}AFX_DATA_INIT
}


void COffsetDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(COffsetDlg)
    DDX_Text(pDX, IDC_OFFSET, m_offset);
    DDV_MinMaxFloat(pDX, m_offset, -1.5f, 1.5f);
    //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(COffsetDlg, CDialog)
    //{{AFX_MSG_MAP(COffsetDlg)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// COffsetDlg message handlers
```

```
// SamplingRateDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Simple.h"
#include "SamplingRateDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSamplingRateDlg dialog


CSamplingRateDlg::CSamplingRateDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CSamplingRateDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CSamplingRateDlg)
    //}}AFX_DATA_INIT
}




BEGIN_MESSAGE_MAP(CSamplingRateDlg, CDialog)
    //{{AFX_MSG_MAP(CSamplingRateDlg)
        // NOTE: the ClassWizard will add message map macros here
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSamplingRateDlg message handlers
```

```
// Simple.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "Simple.h"

#include "MainFrm.h"
#include "SimpleDoc.h"
#include "SimpleView.h"

#include "TextureTimer.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSimpleApp

BEGIN_MESSAGE_MAP(CSimpleApp, CWinApp)
    //{{AFX_MSG_MAP(CSimpleApp)
    ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
    // Standard file based document commands
    ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSimpleApp construction

CSimpleApp::CSimpleApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance


/////////////////////////////////////////////////////////////////////////////
// The one and only CSimpleApp object

CSimpleApp theApp;

    // and object to handle the multimedia timer, hopefully, it'll work
TextureTimer MMTimer;

/////////////////////////////////////////////////////////////////////////////
// CSimpleApp initialization

BOOL CSimpleApp::InitInstance()
{
    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    //  of your final executable, you should remove from the following
    //  the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();         // Call this when using MFC in a shared DLL
#else
    Enable3dControlsStatic();   // Call this when linking to MFC statically
#endif

    // Change the registry key under which our settings are stored.
    // TODO: You should modify this string to be something appropriate
    // such as the name of your company or organization.
    SetRegistryKey(_T("Local AppWizard-Generated Applications"));

    LoadStdProfileSettings(1);  // Load standard INI file options (including MRU)

    // Register the application's document templates.  Document templates
    //  serve as the connection between documents, frame windows and views.

    CSingleDocTemplate* pDocTemplate;
    pDocTemplate = new CSingleDocTemplate(
        IDR_MAINFRAME,
```

```
        RUNTIME_CLASS(CSimpleDoc),
        RUNTIME_CLASS(CMainFrame),       // main SDI frame window
        RUNTIME_CLASS(CSimpleView));
    AddDocTemplate(pDocTemplate);

    // Enable DDE Execute open
    EnableShellOpen();
    RegisterShellFileTypes(TRUE);

    // Parse command line for standard shell commands, DDE, file open
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);

    // Dispatch commands specified on the command line
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;

    // The one and only window has been initialized, so show and update it.
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();

    // Enable drag/drop open
    m_pMainWnd->DragAcceptFiles();

    return TRUE;
}


/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
    CAboutDlg();

// Dialog Data
    //{{AFX_DATA(CAboutDlg)
    enum { IDD = IDD_ABOUTBOX };
    //}}AFX_DATA

    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAboutDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:
    //{{AFX_MSG(CAboutDlg)
        // No message handlers
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
    //{{AFX_DATA_INIT(CAboutDlg)
    //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CSimpleApp::OnAppAbout()
{
    CAboutDlg aboutDlg;
    aboutDlg.DoModal();
}
```

```
/////////////////////////////////////////////////////////////////////////////
// CSimpleApp message handlers
```

```
// SimpleDoc.cpp : implementation of the CSimpleDoc class
//

#include "stdafx.h"
#include "Simple.h"

#include "SimpleDoc.h"

#include "TextureTimer.h"  //support for the data to be stored

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSimpleDoc

IMPLEMENT_DYNCREATE(CSimpleDoc, CDocument)

BEGIN_MESSAGE_MAP(CSimpleDoc, CDocument)
    //{{AFX_MSG_MAP(CSimpleDoc)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSimpleDoc construction/destruction

CSimpleDoc::CSimpleDoc()
{
        FileName[0]='\0';
        m_pFile = NULL;
}

CSimpleDoc::~CSimpleDoc()


BOOL CSimpleDoc::OnNewDocument()
{
//  if (!CDocument::OnNewDocument())
//      return FALSE;

    // TODO: add reinitialization code here
    // (SDI documents will reuse this document)

    CloseFile();
    BOOL bRet = CDocument::OnNewDocument();
    if (bRet)       // do normal stuff
        ((CEditView*)m_viewList.GetHead())->SetWindowText(NULL);

    //JCMH
    CHAR tmp[200];
    strcpy(tmp,"untitled");
    SaveSpeedOCAMData(tmp,1);
    OnOpenDocument("untitled");
    //

    return bRet;
}

/////////////////////////////////////////////////////////////////////////////
// CSimpleDoc serialization
#define SIZE_ARRAY  10000   // max size of storage
#define CHANNELS    5
float data_sensors[SIZE_ARRAY][CHANNELS];
UINT  tot_samples;


void CSimpleDoc::Serialize(CArchive& ar)

  if (ar.IsStoring())
  {
//      // TODO: add storing code here
    }// if is storing
    else
    {
```

```
        // TODO: add loading code here
    }
}

/////////////////////////////////////////////////////////////////////////////
// CSimpleDoc diagnostics

#ifdef _DEBUG
void CSimpleDoc::AssertValid() const
{
    CDocument::AssertValid();
}

void CSimpleDoc::Dump(CDumpContext& dc) const
{
    CDocument::Dump(dc);
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CSimpleDoc commands

// CFile::Open mode flags
const OPENREAD   = CFile::modeRead       | CFile::shareDenyNone;
const OPENWRITE  = CFile::modeReadWrite  | CFile::shareDenyWrite;
const OPENCREATE = CFile::modeCreate     | CFile::modeReadWrite  |
   CFile::shareDenyWrite;

// ******************************************************* //
//============== OnOpenDocument ==========================
//  Open New doc. Close old one in case this is an SDI app//
// =======================================================


BOOL CSimpleDoc::OnOpenDocument(LPCTSTR lpszPathName)
{
    CloseFile();  // Required for SDI app only, because MFC re-uses doc

    // Open the file using the ImageGear library
    LPTSTR lpsz = new TCHAR[100];
    strcpy(lpsz, lpszPathName);
    strcpy(FileName, lpszPathName);
//  return TRUE;

   // Open the file
    CFile* pFile = OpenFile(lpszPathName, m_bReadOnly);
    if (!pFile)
        return FALSE;
    if (m_bReadOnly)
    {
      // Doc was opened read-only: tell user
        CString s;
        s.Format("File '%s' is in use.\nIt will be opened read-only",
               lpszPathName);
        AfxMessageBox(s);
    }
    m_pFile = pFile;
   // Now do standard MFC Open, but close file if the open fails
   BOOL bRet = CDocument::OnOpenDocument(lpszPathName);
   if (!bRet)
    CloseFile();

   return bRet;


}

//====================OnSaveDocument=========================
// Save document. Use already-open file, unless saving to a new name.
// Either way, lock the file and set length to zero before saving it.
// =======================================================
BOOL CSimpleDoc::OnSaveDocument(LPCTSTR lpszPathName)
{
    BOOL bReadOnly = m_bReadOnly;
    CFile* pFile   = m_pFile;

    if(m_pFile == NULL)
    {
//JCMH
        LPTSTR lpsz = new TCHAR[100];
```

```
        strcpy(lpsz, lpszPathName);
        strcpy(FileName, lpszPathName);
        // save the image in memory to the file selected
        SaveSpeedOCAMData(FileName,0);
    }


    ASSERT_VALID(pFile);


    // Check for different file names
    CString sFileName = pFile->GetFilePath();
    BOOL bNewFile = (sFileName != lpszPathName);
    if (bNewFile)
    {
        //JCMH
        LPTSTR lpsz = new TCHAR[100];
        strcpy(lpsz, lpszPathName);
        strcpy(FileName, lpszPathName);
        // save the image in memory to the file selected
        SaveSpeedOCAMData(FileName,0);
        return TRUE; // once we have save it retunr;
        //end JCMH
      // saving w/different name: open new file
        pFile = OpenFile(lpszPathName, bReadOnly, TRUE);
        if (!pFile)
            return FALSE;
    }   // document was opened
    ASSERT_VALID(pFile);

    // If can't get write access, can't save.
    // Display message and return FALSE.
    //
    if (bReadOnly)
    {
        CString s;
        s.Format("File '%s' is in use.\nSave with a different name.",
                lpszPathName);
        AfxMessageBox(s);
        if (bNewFile)           // if new file was opened:
            pFile->Close();      // close it
        return FALSE;
    }
    if (bNewFile)
    {       // new file was opened: install it and close the old one
        ASSERT(m_pFile);        // sanity check
        m_pFile->Close();       // close old one
        m_pFile = pFile;        // and replace w/new one
        m_bReadOnly = bReadOnly;// read-only flag too
    }
    // Now do normal Serialize. Lock the file first and set length to zero
    // This is required because I opened with modeNoTruncate. You might
    // want to consider "robust" saving here: that is, save to a temp file
    // before destroying the original file; then if the save succeeds, replace
    // the original file with the new one.    //
//  pFile->LockRange(0,(DWORD)-1);    // will throw exception if fails
//  pFile->SetLength(0);              // otherwise will append
//  BOOL bRet = CDocument::OnSaveDocument(lpszPathName); // normal MFC save
    BOOL bRet=TRUE;
//  pFile->UnlockRange(0,(DWORD)-1); // unlock


//  DoSave(NULL);
//  return TRUE;
            //JCMH
        //m_pFile->Abort();
        m_pFile->Close();
        LPTSTR lpsz = new TCHAR[100];
        strcpy(lpsz, lpszPathName);
        strcpy(FileName, lpszPathName);

        // save the image in memory to the file selected
        SaveSpeedOCAMData(FileName,0);
        OnOpenDocument(lpszPathName);
        return TRUE; // once we have save it retunr;
        //end JCMH


    return bRet;
}
```

```
// ======================== CloseFile ============
// Close the file if it's open. Called from multiple
// places for SDI app
// ===============================================
void CSimpleDoc::CloseFile()
{
    /* if an old image exists, delete it before loading the new one */
    if (m_pFile)
    {
        m_pFile->Close();
        m_pFile = NULL;
    }

}


//========================= OnCloseDocument ====================
// Close document: time to really close the file too. MFC only calls this
// function in a MDI app, not SDI.//
// ============================================================
void CSimpleDoc::OnCloseDocument()
{
    CloseFile(); // close file
    CDocument::OnCloseDocument(); // Warning: must call this last
                                  // because MFC will "delete this"
}


// ========================= OpenFile =================
// Open the document file. Try to open with write access,
// else read-only.
// bCreate says whether to create the file, used when saving
// to a new name.
// Returns the CFile opened, and sets bReadOnly.//
// ================================================
CFile* CSimpleDoc::OpenFile(LPCTSTR lpszPathName,
                            BOOL& bReadOnly, BOOL bCreate)

    CFile* pFile = new CFile;
    ASSERT(pFile);
    bReadOnly = TRUE; // assume read only

    // try opening for write
    CFileException fe;
    if (pFile->Open(lpszPathName, bCreate ? OPENCREATE : OPENWRITE, &fe))
    {
        bReadOnly = FALSE;      // got write access
    }
    else
        if (bCreate || !pFile->Open(lpszPathName, OPENREAD, &fe))
        {
            // can't open for read OR write--yikes! Time to punt
            delete pFile;
            pFile = NULL;
            ReportSaveLoadException(lpszPathName, &fe, FALSE,
                                    AFX_IDP_FAILED_TO_OPEN_DOC);
        }

    if (pFile)
        pFile->SeekToBegin();

    return pFile;
}

// ================= ReleaseFile ========================
// "Release" the file. This means either abort or close.
// In the case of close, I don't really close it, but leave
// file open for duration of user session.//
// ======================================================
void CSimpleDoc::ReleaseFile(CFile* pFile, BOOL bAbort)
{
    if (bAbort)
        CDocument::ReleaseFile(pFile, bAbort);
    else
        if (!m_bReadOnly)
        {
            pFile->Flush(); // write changes to disk, but don't close!
        }
```

```
}

// ============== DoFileSave ======================
// Map "Save" to "Save As" if doc is read-only
// ================================================
BOOL CSimpleDoc::DoFileSave()
{
    return m_bReadOnly ?
      DoSave(NULL) :             // do Save As
      CDocument::DoFileSave();   // save as normal
}
// ================= GetFile ============================
// Override to use my always-open CFile object instead
// of creating and opening a new one.//
// ======================================================
CFile* CSimpleDoc::GetFile(LPCTSTR, UINT, CFileException*)
{
    ASSERT_VALID(m_pFile);
    return m_pFile;
}

//////////////////////////////////////////////////////
// Save data related to the speedocam
//////////////////////////////////////////////////////
void CSimpleDoc::SaveSpeedOCAMData(CHAR* FileName1,BOOL m_or_unt)
{
    if(m_or_unt == 1) // 1 means we have to save the untitle file
        strcat(FileName1,"");
    else
        strcat(FileName1, ".m");
    FILE *file = fopen(FileName1,"w");
    if( file == NULL )
        perror( "Open failed on output file" );
    else
    {
        printf( "Open succeeded on output file\n" );
        fprintf(file,"samprate= %07.5f;\n",DataSampRate);
        fprintf(file,"amplitude= %07.5f;\n",DataAmp);
        fprintf(file,"frequency= %07.5f;\n",DataFreq);
        fprintf(file,"offset= %07.5f;\n",DataOffset);
        fprintf(file,"data=[\n");
        for (unsigned int n=0;n< tot_samples ;n++)
        {
            fprintf(file,"%07.5f %07.5f %07.5f %07.5f %07.5f\n",
                data_sensors[n][0],
                data_sensors[n][1],
                data_sensors[n][2],
                data_sensors[n][3],
                data_sensors[n][4]);
        }
        fprintf(file,"];\n");
        fclose( file );
    }
}
```

```
// SimpleView.cpp : implementation of the CSimpleView class
//

#include "stdafx.h"
#include "Simple.h"

#include "SimpleDoc.h"
#include "SimpleView.h"

#include "FrequencyDlg.h"
#include "AmplitudeDlg.h"
#include "SamplingRateDlg.h"
#include "OffsetDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CSimpleView

IMPLEMENT_DYNCREATE(CSimpleView, CView)

BEGIN_MESSAGE_MAP(CSimpleView, CView)
    //{{AFX_MSG_MAP(CSimpleView)
    ON_COMMAND(ID_INPUTSIGNAL_AMPLITUDE, OnInputsignalAmplitude)
    ON_COMMAND(ID_INPUTSIGNAL_FREQUENCY, OnInputsignalFrequency)
    ON_COMMAND(ID_INPUTSIGNAL_RUN, OnInputsignalRun)
    ON_COMMAND(ID_INPUTSIGNAL_STOP, OnInputsignalStop)
    ON_UPDATE_COMMAND_UI(ID_INPUTSIGNAL_RUN, OnUpdateInputsignalRun)
    ON_UPDATE_COMMAND_UI(ID_INPUTSIGNAL_STOP, OnUpdateInputsignalStop)
    ON_COMMAND(ID_INPUTSIGNAL_OFFSET, OnInputsignalOffset)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
 / CSimpleView construction/destruction

CSimpleView::CSimpleView()
{
    // TODO: add construction code here

    // initial values of the input signal and sampling rate
    Amplitude = 0;
    freq    =   1;
    SetSampling();
    //SamplingRate = (float)(1.0/200.0); // 200 Hertz
    //MMtimerPeriod = (UINT)(1000.0 * SamplingRate);
    Offset = 1.88;
    run_stop=0; // not running
}

CSimpleView::~CSimpleView()
{
}

BOOL CSimpleView::PreCreateWindow(CREATESTRUCT& cs)
{
    // TODO: Modify the Window class or styles here by modifying
    //  the CREATESTRUCT cs

    return CView::PreCreateWindow(cs);
}

/////////////////////////////////////////////////////////////////////////////
// CSimpleView drawing

void CSimpleView::OnDraw(CDC* pDC)
{
    CSimpleDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here

/////////////////////////////////////////////////////////////////////////////
// CSimpleView diagnostics

#ifdef _DEBUG
```

```
void CSimpleView::AssertValid() const
{
    CView::AssertValid();
}

void CSimpleView::Dump(CDumpContext& dc) const
{
    CView::Dump(dc);
}

CSimpleDoc* CSimpleView::GetDocument() // non-debug version is inline
{
    ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CSimpleDoc)));
    return (CSimpleDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CSimpleView message handlers

void CSimpleView::OnInputsignalAmplitude()
{
    // TODO: Add your command handler code here
//  Beep(500,500);
    CAmplitudeDlg dlg;
    dlg.m_amplitude = Amplitude;
    if(dlg.DoModal()== IDOK)
        Amplitude = dlg.m_amplitude;

}

void CSimpleView::OnInputsignalFrequency()
{
    // TODO: Add your command handler code here

    CFrequencyDlg dlg;
    dlg.m_frequency = freq;
    if(dlg.DoModal() == IDOK)
    {
        freq = dlg.m_frequency;
        SetSampling();
    }
}



void CSimpleView::OnInputsignalRun()
{
    // TODO: Add your command handler code here
    Beep(10000,50);
    run_stop=1; //set flag to one to idicate it's running

    // Create multimedia timer
    CSimpleDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);

    pDoc->DataSampRate = SamplingRate;
    pDoc->DataAmp = Amplitude;
    pDoc->DataFreq = freq;
    pDoc->DataOffset = Offset;
    id_timer2=MMTimer.Create((UINT)MMtimerPeriod,(UINT)0,freq,
                             Amplitude, Offset, SamplingRate);
    if (id_timer2== NULL)
        run_stop=0; //bRtn = FAILURE;

    CClientDC dc(this);
    dc.Rectangle(5, 5,20,20);

}

void CSimpleView::OnInputsignalStop()
{
    // TODO: Add your command handler code here
    Beep(13000,500);

    if(TIMERR_NOERROR == timeKillEvent(id_timer2))
    {
        run_stop=0;
//      DDA08.InitDAC();
        CClientDC dc(this);
```

```
        dc.Rectangle(10, 10,15, 15);
    }


void CSimpleView::OnUpdateInputsignalRun(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
//  Beep(5000,500);
    if(run_stop)
        pCmdUI->Enable( FALSE );// process running, so set unable it
    else
        pCmdUI->Enable( TRUE );
}

void CSimpleView::OnUpdateInputsignalStop(CCmdUI* pCmdUI)
{
    // TODO: Add your command update UI handler code here
//  Beep(5000,500);
    if(run_stop)
        pCmdUI->Enable( TRUE );// process running, so set unable it
    else
        pCmdUI->Enable( FALSE );
}



void CSimpleView::OnInputsignalOffset()
{
    // TODO: Add your command handler code here

    COffsetDlg dlg;
    dlg.m_offset = Offset;
    if (dlg.DoModal() == IDOK)
        Offset = dlg.m_offset;
}

// instead of reading a new sampling rate, we compute it based on the
// value of the frequency of the signal.
void CSimpleView::SetSampling()
{
    float T;
    if(freq <1.0)
        T=(float)(1.0/freq);
    else
        T=1.0;
    SamplingRate = (float)(T/1000.0); // we use this variable to compute the input signal
    MMtimerPeriod = (UINT)T; // the timer will run every T milliseconds
}
```

```
// stdafx.cpp : source file that includes just the standard includes
//  Simple.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

```
// stdafx.cpp : source file that includes just the standard includes
//  Simple.pch will be the pre-compiled header
//  stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

```
// TextureTimer.cpp
//
// Implementation of the class members of TextureTimer class
//
// (C) Xiaohui Song. 1999.
//

#include "stdafx.h"
#include "TextureTimer.h"

#include "f:\novy-2~1\DDA16\DLPortIO\Api\dlportio.h" //to access memory address


// create timer and set data members
UINT TextureTimer::Create(UINT nPeriod, UINT nRes, float f,
                          float amp, float offset, float samprate)
{
    // Set data members
    sr_timer = samprate;
    amp_timer = amp;
    off_amplitude = offset;
    omega = (float)((float)TWOPI * f);
    samp_n = 0;
    max_time = (float)(10.0/f);

    // stop_signal is initialized to 0, and will be set to zero once
    // two cycles have elapsed
    stop_signal =0;

    // initiallize in zero both dig-to-analog channels
    WriteAnalogOutput(0, 0);
    WriteAnalogOutput(1, 0);
  // Create multimedia timer
    id_timer_this = timeSetEvent (nPeriod, nRes, TimeProc,
                        (DWORD) this, TIME_PERIODIC);
    return(id_timer_this);
    // if successful return the IDtimer, if not returN NULL
}

 / callback function: display texture matrix
void CALLBACK TextureTimer::TimeProc(UINT uID, UINT uMsg,
                                     DWORD dwUser,
                                     DWORD dw1, DWORD dw2)
{
    // dwUser contains ptr to Timer object
    TextureTimer * ptimer = (TextureTimer *) dwUser;

    int i;
    float time,input;

    time = ptimer->samp_n * ptimer->sr_timer;
    input = (float)(ptimer->amp_timer * (float)sin(ptimer->omega * time)
                    +ptimer->off_amplitude);

    if(time > ptimer->max_time)
    {
        ptimer->samp_n--;
        tot_samples = ptimer->samp_n;
        input=0;
        //Beep(10,1);
    }


//  WriteAnalogOutput(unsigned char chan, float volt)
    WriteAnalogOutput(0, input);
    for(i=0; i < CHANNELS-1 ;i++)   // read 4 channels
        //ReadAnalogInput(unsigned char chan,unsigned char gain)
        data_sensors[ptimer->samp_n][i]=ReadAnalogInput(i,0);
//      data_sensors[ptimer->samp_n][i]=input;

//  for(i=0; i < CHANNELS-1 ;i++)
//      data_sensors[ptimer->samp_n][i]=input;

    data_sensors[ptimer->samp_n][i]=input;//ReadAnalogInput(i,0);

    ptimer->samp_n++;
}


// update # of samples in a stage
```

```
void TextureTimer::UpdateData(int samps_per_stage)
{
}

// ------------------------------------------
// write the command signal into the DC motor
// ------------------------------------------
void TextureTimer::WriteAnalogOutput(unsigned char chan, float volt)
{
    //-------- check voltage range ----------
    if (volt >= 4.99)   volt = (float)  4.99;
    if (volt <= -4.99)  volt = (float)  -4.99;

   //----------- convert voltage to 12 bit value ------------
//volt=-5.0;
    unsigned short volt_w;
    unsigned char low_w, high_w;
    volt_w = (unsigned short)((volt*(0x7FF))/10.0+(0x7FF));
    low_w  = (volt_w & 0xFF);  // get low byte
    high_w = ((volt_w>>8) & 0x0F); // get high part

    unsigned short offset_add=chan*2;
    DlPortWritePortUchar(CARDBASE+0x08+offset_add,  low_w);
    DlPortWritePortUchar(CARDBASE+0x08+offset_add+1,high_w);
//  unsigned short value = (unsigned short)volt;//volt;
//  volt_w = (unsigned short)(4098.0/20.0*(volt+10.0));
//  DlPortWritePortUchar(CARDBASE+0x08,2048&0xff);
//  DlPortWritePortUchar(CARDBASE+0x09,2048/256);

}
// ------------------------------------------
// read the signal comming from the sensors
// ------------------------------------------
float TextureTimer::ReadAnalogInput(unsigned char chan,unsigned char gain)
{
    unsigned int    timeout = 60000;
    long int        read_volt=0;
    float           volt;


//  DlPortWritePortUchar(CARDBASE+0, 0x04); //used with an external trigger
    DlPortWritePortUchar(CARDBASE+2, (gain << 4 | chan));// write channel  and enable start conversion
    DlPortWritePortUchar(CARDBASE+3, 0x0);  // start conversion


    while( (DlPortReadPortUchar(CARDBASE+2) & 0X80) ) timeout--;// wait for ADC conversion
//  volt = (float) ((unsigned) (DlPortReadPortUchar(CARDBASE+2)>>4));

//  read_volt = (DlPortReadPortUchar(CARDBASE+7));
//  read_volt = (DlPortReadPortUchar(CARDBASE+6));
//  read_volt = (DlPortReadPortUchar(CARDBASE+7)<<4);
//  read_volt |= (DlPortReadPortUchar(CARDBASE+6)>>4);

    read_volt = (DlPortReadPortUchar(CARDBASE+6)>>4);
    read_volt |= (DlPortReadPortUchar(CARDBASE+7)<<4);

    if (timeout == 0)
    {
        return 0xffff;
    }
//  volt = (float)((float)read_volt*20.0/4096.0-10.0);
    return (float)((float)read_volt*20.0/4096.0-10.0);
}
```

```
#if !defined(AFX_AMPLITUDEDLG_H__019C6707_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
#define AFX_AMPLITUDEDLG_H__019C6707_E54B_11D2_8BB4_0060081FFF94__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// AmplitudeDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CAmplitudeDlg dialog

class CAmplitudeDlg : public CDialog
{
// Construction
public:
    CAmplitudeDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CAmplitudeDlg)
    enum { IDD = IDD_AMPLITUDE };
    float   m_amplitude;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CAmplitudeDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CAmplitudeDlg)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_AMPLITUDEDLG_H__019C6707_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
```

```
// DDA-08.h
// Header file that defines constants to be used by an object
// This definitions allow communication between the software
// and the DA_AD
//
// (C) Xiaohui Song. 1999.
//

#ifndef _DA_AD_H
#define _DA_AD_H

//#define   CARDBASE    0x340   // memory address of the DA_AD card
//#define SIZE_ARRAY    10000   // max size of storage

//const int CHANNELS=4;      // number of CHANNELs to read from
#define CARDBASE    0x300   // memory address of the DA_AD card
#define SIZE_ARRAY  2500    // max size of storage
#define CHANNELS    5       // number of CHANNELs to read from

float data_sensors[SIZE_ARRAY][CHANNELS];
UINT  tot_samples;


#endif
```

```
#if !defined(AFX_FREQUENCYDLG_H__019C6706_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
#define AFX_FREQUENCYDLG_H__019C6706_E54B_11D2_8BB4_0060081FFF94__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// FrequencyDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CFrequencyDlg dialog

class CFrequencyDlg : public CDialog
{
// Construction
public:
    CFrequencyDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CFrequencyDlg)
    enum { IDD = IDD_FREQUENCY };
    float   m_frequency;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CFrequencyDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CFrequencyDlg)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_FREQUENCYDLG_H__019C6706_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
```

```
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__019C66F8_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
#define AFX_MAINFRM_H__019C66F8_E54B_11D2_8BB4_0060081FFF94__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMainFrame : public CFrameWnd
{

protected: // create from serialization only
    CMainFrame();
    DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:

// Operations
public:

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CMainFrame)
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CMainFrame();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
    CStatusBar  m_wndStatusBar;
    CToolBar    m_wndToolBar;

// Generated message map functions
protected:
    //{{AFX_MSG(CMainFrame)
    afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code!
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_MAINFRM_H__019C66F8_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
```

```
#if !defined(AFX_OFFSETDLG_H__86BE1090_E619_11D2_8BB4_0060081FFF94__INCLUDED_)
#define AFX_OFFSETDLG_H__86BE1090_E619_11D2_8BB4_0060081FFF94__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// OffsetDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// COffsetDlg dialog

class COffsetDlg : public CDialog
{
// Construction
public:
    COffsetDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(COffsetDlg)
    enum { IDD = IDD_OFFSET };
    float   m_offset;
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(COffsetDlg)
    protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(COffsetDlg)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_OFFSETDLG_H__86BE1090_E619_11D2_8BB4_0060081FFF94__INCLUDED_)
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Simple.rc
//
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME                   128
#define IDR_SIMPLETYPE                  129
#define IDD_FREQUENCY                   130
#define IDD_AMPLITUDE                   131
#define IDD_SAMPLINGRATE                132
#define IDD_OFFSET                      135
#define IDC_FREQUENCY                   1000
#define IDC_AMPLITUDE                   1001
#define IDC_SAMPLINGRATE                1002
#define IDC_OFFSET                      1006
#define ID_INPUTSIGNAL_FREQUENCY        32771
#define ID_INPUTSIGNAL_AMPLITUDE        32772
#define ID_INPUTSIGNAL_SAMPLINGRATE     32773
#define ID_INPUTSIGNAL_RUN              32774
#define ID_INPUTSIGNAL_STOP             32775
#define ID_INPUTSIGNAL_OFFSET           32776

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                     1
#define _APS_NEXT_RESOURCE_VALUE        136
#define _APS_NEXT_COMMAND_VALUE         32777
#define _APS_NEXT_CONTROL_VALUE         1007
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

```
#if !defined(AFX_SAMPLINGRATEDLG_H__019C6708_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
#define AFX_SAMPLINGRATEDLG_H__019C6708_E54B_11D2_8BB4_0060081FFF94__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// SamplingRateDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CSamplingRateDlg dialog

class CSamplingRateDlg : public CDialog
{
// Construction
public:
    CSamplingRateDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
    //{{AFX_DATA(CSamplingRateDlg)
    enum { IDD = IDD_SAMPLINGRATE };
    //}}AFX_DATA


// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CSamplingRateDlg)
    //}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    //{{AFX_MSG(CSamplingRateDlg)
        // NOTE: the ClassWizard will add member functions here
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SAMPLINGRATEDLG_H__019C6708_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
```

```
// Simple.h : main header file for the SIMPLE application
//

#if !defined(AFX_SIMPLE_H__019C66F4_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
#define AFX_SIMPLE_H__019C66F4_E54B_11D2_8BB4_0060081FFF94__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols

// JCMH
#include "TextureTimer.h"

  // a global timer.
extern TextureTimer MMTimer;

/////////////////////////////////////////////////////////////////////////////
// CSimpleApp:
// See Simple.cpp for the implementation of this class
//

class CSimpleApp : public CWinApp
{
public:
    CSimpleApp();

// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CSimpleApp)
    public:
    virtual BOOL InitInstance();
    //}}AFX_VIRTUAL

 / Implementation
    //{{AFX_MSG(CSimpleApp)
    afx_msg void OnAppAbout();
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SIMPLE_H__019C66F4_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
```

```
// SimpleDoc.h : interface of the CSimpleDoc class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_SIMPLEDOC_H__019C66FA_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
#define AFX_SIMPLEDOC_H__019C66FA_E54B_11D2_8BB4_0060081FFF94__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


class CSimpleDoc : public CDocument
{
protected: // create from serialization only
    CSimpleDoc();
    DECLARE_DYNCREATE(CSimpleDoc)

// Attributes
public:

        float   DataAmp;        // input amplitude
        float   DataFreq;       // input frequency
        float   DataOffset;     // input offset
        float   DataSampRate;       // sampling rate
        UINT Data_n;            // number of samples
        CStdioFile DataFile;        // data file
// Operations
public:
    CHAR FileName[200]; //name of the file

    CFile*  m_pFile;        // the file kept open during editing
    BOOL    m_bReadOnly;    // wheter read only access
// operations overriding MFC operations
    virtual BOOL OnOpenDocument(LPCTSTR lpszPathName);
    virtual BOOL OnSaveDocument(LPCTSTR lpszPathName);
    // helpers
    void    CloseFile();
    void    SaveSpeedOCAMData(CHAR* FileName1,BOOL m_or_unt);

    CFile* OpenFile(LPCTSTR lpszPathName, BOOL& bReadOnly,
                    BOOL bCreate=FALSE);

    virtual void OnCloseDocument();
    virtual void ReleaseFile(CFile* pFile, BOOL bAbort);
    virtual BOOL DoFileSave();
    virtual CFile* GetFile(LPCTSTR lpszFileName, UINT nOpenFlags,
                           CFileException* pError);



// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CSimpleDoc)
    public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CSimpleDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CSimpleDoc)
        // NOTE - the ClassWizard will add and remove member functions here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SIMPLEDOC_H__019C66FA_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
```

```
// SimpleView.h : interface of the CSimpleView class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_SIMPLEVIEW_H__019C66FC_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
#define AFX_SIMPLEVIEW_H__019C66FC_E54B_11D2_8BB4_0060081FFF94__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


class CSimpleView : public CView
{
protected: // create from serialization only
    CSimpleView();
    DECLARE_DYNCREATE(CSimpleView)

// Attributes
public:
    CSimpleDoc* GetDocument();

    // variables to be used with the input signal to the DC-motor
    float freq,         // given in Hertz
        Amplitude,      // given in volts
        SamplingRate,   // given in seconds i.e. 0.001sec = 1msec
        Offset;         // offset on the input signal
    UINT MMtimerPeriod;

    BOOL run_stop; // 1 running, 0 stop
    UINT id_timer2; //timeSetEvent, for multimedia timer

// Operations
public:
    void SetSampling();
// Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CSimpleView)
    public:
    virtual void OnDraw(CDC* pDC);  // overridden to draw this view
    virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
    protected:
    //}}AFX_VIRTUAL

// Implementation
public:
    virtual ~CSimpleView();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    //{{AFX_MSG(CSimpleView)
    afx_msg void OnInputsignalAmplitude();
    afx_msg void OnInputsignalFrequency();
    afx_msg void OnInputsignalRun();
    afx_msg void OnInputsignalStop();
    afx_msg void OnUpdateInputsignalRun(CCmdUI* pCmdUI);
    afx_msg void OnUpdateInputsignalStop(CCmdUI* pCmdUI);
    afx_msg void OnInputsignalOffset();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in SimpleView.cpp
inline CSimpleDoc* CSimpleView::GetDocument()
   { return (CSimpleDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_SIMPLEVIEW_H__019C66FC_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
```

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__019C66F6_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
#define AFX_STDAFX_H__019C66F6_E54B_11D2_8BB4_0060081FFF94__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN        // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdisp.h>        // MFC Automation classes
#include <afxdtctl.h>       // MFC support for Internet Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>         // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT


//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__019C66F6_E54B_11D2_8BB4_0060081FFF94__INCLUDED_)
```

```
// TextureTimer.h
// Header file that defines a class that will contain
// the data and functions needed to set and access a
// multimedia timer
//
//
// (C) Xiaohui Song. 1998.
//

#ifndef _CLASS_TEXTURE_TIMER
#define _CLASS_TEXTURE_TIMER

//JCMH: defined to handle the timer
#include <mmsystem.h.>//include for the DisplayTexture type
// declaration of the CArray stuff JCMH
#include <afxtempl.h>

#include <math.h>
#include <conio.h>

#define PI              3.1415927
#define TWOPI           2*PI // to work with rad/sec

#define MMTIMER_PERIOD      1 // in miliseconds

#define CARDBASE    0x300   // memory address of the DA_AD card
#define SIZE_ARRAY  10000   // max size of storage
#define CHANNELS    5       // number of CHANNELs to read from

extern float data_sensors[SIZE_ARRAY][CHANNELS];
extern UINT  tot_samples;


class TextureTimer
{
    // parameter passed to the class
    float sr_timer;     // sampling rate (in sec, i.e. 0.005=5milsec
    float amp_timer;    // amplitude of the signal
    float off_amplitude;        // offset of the signal
    // variables used inside the class
    int samp_n;         // current sample
    float omega;        // 2 * pi * f

    float max_time;

    UINT id_timer_this;

    UINT    stop_signal; // will tell us when the signal
                         // has reached two cycles and w emust send
                         // a zero signal


    static void CALLBACK TimeProc(UINT uID, UINT uMsg, DWORD dwUser,
                                  DWORD dw1, DWORD dw2);

    static void WriteAnalogOutput(unsigned char chan, float volt);
    static float ReadAnalogInput(unsigned char chan,unsigned char gain);

public:
    TextureTimer()
    {
        samp_n=0;
    }
    ~TextureTimer(){samp_n=0;}
    UINT Create(UINT nPeriod, UINT nRes, float f, float amp,
        float offset, float samprate);

    void UpdateData(int sampstage);
};

#endif
```

Document Log:

Manuscript Version 1 — March, 2002

Typeset by $\mathcal{AMS}$-LaTeX — 28 April 2002

XIAOHUI SONG

CENTRE FOR INTELLIGENT MACHINES, MCGILL UNIVERSITY, 3480 UNIVERSITY ST., MONTRÉAL (QUÉBEC) H3A 2A7, CANADA, *Tel.* : (514) 398-5856

E-mail address: songwell@cim.mcgill.ca

Typeset by $\mathcal{AMS}$-LaTeX