COMPUTER-ASSISTED SYNTHESIS OF WRAPPING CAM MECHANISMS

by
Kurnia Dias Iskandar

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in Mechanical Engineering

APPROVED:

[illegible]
Charles F. Reinholtz, Chairman

Paul H. Tidwell
Paul H. Tidwell

[illegible]
Eugene F. Brown

June 28, 1996

Blacksburg, Virginia

Key words: Wrapping Cam, Cam Synthesis, Cam Design, Force Synthesis, Kinematics

c. 2

LD
5655
V855
1996
I853
c. 2

COMPUTER-ASSISTED SYNTHESIS OF WRAPPING CAM MECHANISMS

by

Kurnia Dias Iskandar

Charles F. Reinholtz, Chairman

Mechanical Engineering

ABSTRACT

A wrapping cam mechanism consists of a cam wrapped by a belt or flexible band. An analytical method for synthesizing wrapping cam mechanisms has recently been developed. This thesis extends the previous work and describes the development of an interactive wrapping cam synthesis package based on analytical methods. The software presented in this thesis generates cam profiles for eight configurations of wrapping cam-pulley or wrapping cam-sprocket mechanisms. This software package has a graphical user interface to give intuitive interactions. The modularity of the software package increases the flexibility for future program extension. This thesis also discusses the extended synthesis methods for a wrapping cam-link mechanism.

Thank you and dedication
to my family and others
who had given so much support:

A.M. Satari
Vestawati Satari
Achmad Ika Wardhana
Nurlaila Rahman
and
Sri Kusumaning Diah

ACKNOWLEDGMENT

I would like to thank Dr. Charles Reinholtz, Dr. Paul Tidwell, and Dr. Eugene Brown for being members of the advisor committee in my master program at Virginia Polytechnic Institute and State University (VPI & SU). Their help and patience contribute to the success of this thesis work. It has been an honor to know them and being one of their students.

The contribution of PT PAL-INDONESIA, an Indonesian ship building company, also cannot be forgotten. Its scholarship had given me a chance to pursue a higher and a better education. I cannot thank enough for this, and I am very grateful.

TABLE OF CONTENTS:

LIST OF FIGURES

1. INTRODUCTION

There is a need for software that can improve the efficiency of cam design process [Dvorak, 1988]. Even though there are some cam design software available in the market such as Cam Designer by Delta Engineering Corp., no package is available for the design of wrapping cam mechanisms. There are two main reasons why a software for wrapping cam design is not yet available. The analytical methods for wrapping cam synthesis have only been recently developed, and adding these methods into the existing proprietary cam synthesis packages is very difficult.

Nautilus, Inc. has been using wrapping cam mechanisms in their line of exercise equipment. Originally, the design of wrapping cams used in these mechanisms was done using graphical methods. Only recently has a closed-form method for wrapping cam synthesis been developed [Tidwell et al., 1994] [Tidwell, 1995]. The research work presented here extends Tidwell's method of wrapping cam synthesis and implements it in an interactive software package.

An important consideration in designing a software package is the utility of the program for either novice or advance users, e.g. intuitive interface between the user and program and ease of program updating and maintenance. A graphical user interface (GUI) is one way to provide an intuitive interface to the program. The efficiency in program updating and maintenance can be improved by making the program as a network of modules. One programming language that supports the concept of modularity is Object-Oriented Programming (OOP) language. To write and to compile the program's code, Borland C++ version 4.0 is used because it supports object-oriented programming. To conform with OOP, Borland's ObjectWindows Library 2.0 is implemented in the development of the GUI [Swan et al., 1994].

The organization of this thesis is as the following. Chapter 2 is a literature review. Chapter 3 discusses the analytical methods for wrapping cam mechanisms. Chapter 4 presents a brief discussion on C++ and object-oriented programming languages, which are used in developing the package. Chapter 5 describes the objectives of this research. Chapter 6 gives an overview of the wrapping cam synthesis software package. Chapter 7 discusses the graphical user interface (GUI) of this software package. Chapter 8 discusses the object-oriented programming approach in the modelling of a wrapping cam design. Chapter 9 and 10 cover the synthesis and analysis modules of this package. Chapter 11 discusses the conclusion and contributions this work gives to the study of wrapping cam mechanisms. The program source code is listed in the Appendix, with the purpose to help the user in modifying the program if necessary.

2. LITERATURE REVIEW

A wrapping cam mechanism is often used to produce either force functions. The mechanism is composed of a planar disk cam wrapped by either a chain or a belt (Figure 2.1). Unlike traditional cam mechanisms, there is no sliding between the cam and the follower. There is only rolling contact, therefore this joint is also called a gear joint. This type of joints is called a gear joint. The cam and the sprocket center of rotations are single degree of freedom joints, i.e., revolute joints. Tidwell uses the nomenclature 'G' and 'R' for a gear joint and a revolute joint [Tidwell, 1995]. The conventional cam synthesis methods for oscillating or reciprocating followers can not be applied to this mechanism.

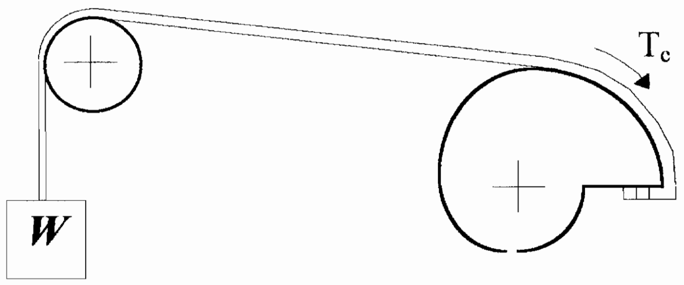


Figure 2.1. A Wrapping Cam Mechanism

Nautilus Inc. uses the wrapping cams in their line of exercise machine products. For quite some times, the graphical methods were the only way to generate the surface of a wrapping cam. Recently, an analytical method for wrapping cam synthesis has been developed [Tidwell et al., 1994] [Tidwell, 1995]. This method uses the loop-closure method and vector-complex notation. Tidwell's work covers two types of wrapping cam mechanisms, GGRR and GRRR. A GGRR mechanism contains a cam wrapped by belt or chains with a pulley or a sprocket at its end (Figure 2.1). In a GRRR mechanism, the belt or chains is connected to a link with a revolute joint (Figure 2.2). The letter 'G' stands for a gear joint, and 'R' stands for a revolute joint. The GGRR wrapping cam has gear-gear-revolute-revolute joints, while the GRRR has gear-revolute-revolute-revolute joints.

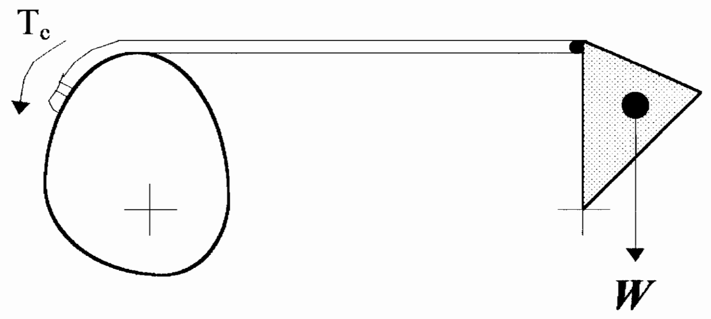


Figure 2.2. A Wrapping Cam with Link Mechanism

This analytical method has been implemented in software using the Matlab and MathCad packages. Matlab and MathCad are general purpose mathematical packages. In Matlab and MathCad programs, the user specifies the variables and functions then asks the program to compute the dependent variables. Both Matlab and MathCad have graphical function capability to plot the function's output. This graphics capability is very useful for mechanism synthesis. Matlab and MathCad are easy to use and readily available for various platforms (e.g. PC, Macintosh, etc). These advantages contribute to the popularity of these software packages among engineers and students.

Mathematical software packages are not very suitable, in many cases, for interactive synthesis of wrapping cam mechanisms. MathCad is a slow and clumsy when it has to calculate many points on the cam surface. The mathematical software packages were designed solely to solve mathematical problems, therefore they do not provide intuitive interaction with the user. The need to have interactive synthesis program for cam design has been recognized for quite sometimes. One reason to have cam synthesis software is to accelerate the design process [Dvorak, 1988]. Interactive synthesis software will allow the users to perform what-if analysis, and puts their intuitive understanding of mechanical design on a firmer foundation [Deitz, 1995]. Even though there are many software packages available for cam kinematic and synthesis analysis, they are only suitable for standard cam mechanisms. Those software packages are only for designing a cam with roller follower, flat face follower, and knife edge follower [Payne, 1994] [Cleghorn and Podhorodeski, 1988] [Dvorak, 1988]. The wrapping cam synthesis problem has different design parameters such as chain's or belt's thickness, sprocket's radius, center distance between cam and sprocket, and torque as the function of cam's angle of rotation. Also, since the analytical wrapping cam synthesis method was only recently developed, it has yet to be implemented in a synthesis package. Adding wrapping cam synthesis to existing packages can be very difficult, eventhough the end-user might has access to the program's source code.

For the reasons described above, this thesis work has the objective of developing an interactive wrapping cams synthesis software package. This package implements the force synthesis case for wrapping cam mechanisms. The force synthesis means that the mechanism supposes to reproduce the user specified external force function. An interactive package requires fast user feedback on designs and some form of visualization, e.g. a graphical user interface. To simplify the program design and to provide versatility for the end-user, this package was designed as a network of modules. These modules are the graphics module including graphical user interface (GUI), the kinematic analysis module, and the wrapping cam synthesis module. One programming language that supports the concept of modularity is C++ / object-oriented programming language which was used to write and to compile the program. The user interacts with this program through GUI. The kinematic analysis module is used to obtain the torque curve or strength curve given the dimensions of the wrapping cam mechanism. The kinematic analysis is helpful to practicing engineers who want to examine the motion characteristics of an existing wrapping cam mechanism and to check a synthesized mechanism. The synthesis module will generate a cam profile that satisfies user specification.

3. WRAPPING CAM SYNTHESIS

3.1. GGRR Wrapping Cam Synthesis

Tidwell's method of wrapping cam synthesis uses loop-closure method expressed in complex vector form [Tidwell et al., 1994] [Tidwell, 1995]. Using vector summation in complex form, a vector-loop is traversed from a point on the cam surface through the sprocket and back, as shown in Fig. 3.1. Using a conjugate geometry method, the cam becomes a reference (fixed) object. The sprocket, the center distance, and the belt rotate relative to the cam. A reference line on the cam is used for angle measurements.

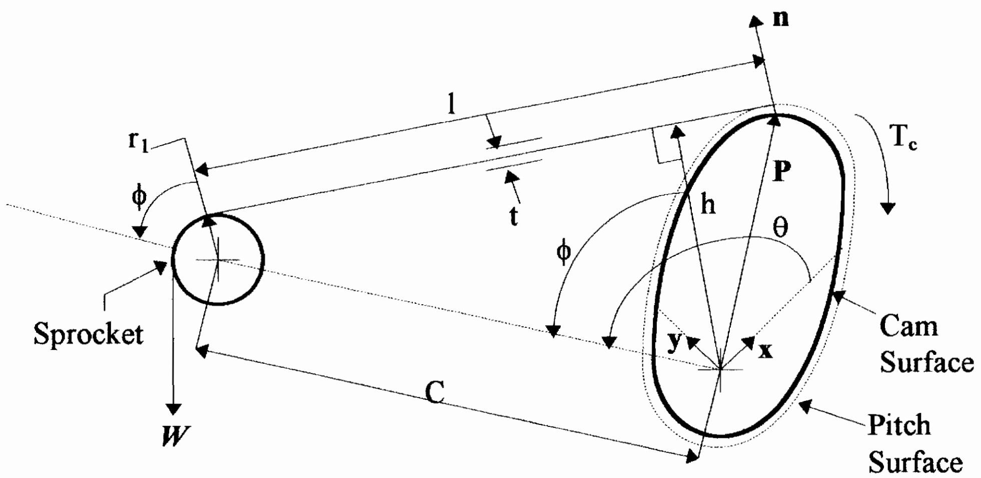


Figure 3.1. Variables of Constant Tension GGRR Wrapping Cam

Figure 3.2 shows a GGRR wrapping cam with a weight hanging at end of the belt. This weight produces a constant tension in the belt with magnitude W. This type of configuration is also called a positive wrapping cam. For the constant tension (positive) GGRR wrapping cam, given the center distance between the cam and sprocket (C), the sprocket radius (r_1), the belt or chain thickness (t), and the weight (W), the equation to locate a point on the cam surface as function of the rotation of the sprocket relative to the cam can be found as follows:

$$\vec{P} = Ce^{i\theta} + r_1 e^{i(\theta-\phi)} + l e^{i(\theta-\phi-\pi/2)} - \frac{t}{2} e^{i(\theta-\phi)}$$

$$\vec{P} = Ce^{i\theta} + (r_1 - \frac{t}{2} - il)e^{i(\theta-\phi)} \tag{3.1}$$

The length l and the angle ϕ must be determined before solving equation 3.1.

Torque acting on the cam is specified as a function of sprocket's angle of rotation around the cam, $T_C(\theta)$, and the weight 'W' hanging at the other end creates a tension on the chain or belt. Thus, the moment arm of the weight can be obtained by:

$$h(\theta) = \frac{T_c(\theta)}{W} \tag{3.2}$$

To eliminate the dependency on the weight W, $T_C(\theta)$ is defined such that

$$T_c(\theta) = c_1 t(\theta), \tag{3.3}$$

where c_1 is an arbitrary constant and $t(\theta)$ is a nondimensional torque function. If W moves a distance L from its initial position, then W does work of magnitude WL.

$$WL = \int_{\theta_{min}}^{\theta_{max}} c_1 t(\theta) d\theta,$$

then solved it for c_1:

$$c_1 = \frac{WL}{\int_{\theta_{min}}^{\theta_{max}} t(\theta) d\theta} \tag{3.4}$$

Then substitute $T_c(\theta) = \frac{WL}{\int_{\theta_{min}}^{\theta_{max}} t(\theta) d\theta} t(\theta)$ into equation 3.2 to obtain:

$$h(\theta) = \frac{L}{\int_{\theta_{min}}^{\theta_{max}} t(\theta)d\theta} t(\theta) \tag{3.5}$$

The angle ϕ, between the line of center distance and the moment arm, in Figure 3.1 can be found as:

$$\phi = \cos^{-1}\left(\frac{(h - r_1)}{C}\right), \tag{3.6}$$

and taking the derivative of ϕ with respect to θ gives:

$$\frac{d\phi}{d\theta} = \frac{-1}{\sqrt{C^2 - (h - r_1)^2}} \frac{dh}{d\theta} \tag{3.7}$$

Length l is the distance along the belt between the point of contact with sprocket to the point of contact on the cam pitch surface, as shown in Figure 3.1. According to [Chakraborty and Dhande, 1977] the condition of contact at the cam surface must be $\frac{d\vec{P}}{d\theta} \cdot \hat{n} = 0$, where $\hat{n}$ is a unit normal vector at the point of contact. From Figure 3.1., the unit normal vector is equal to

$$\hat{n} = e^{i(\theta-\phi)} \tag{3.8}$$

and the tangent vector $\frac{d\vec{P}}{d\theta}$ is equal to

$$\frac{d\vec{P}}{d\theta} = iCe^{i\theta} - i\frac{dl}{d\theta}e^{i(\theta-\phi)} + i\left(r_1 - \frac{t}{2} - il\right)\left(1 - \frac{d\phi}{d\theta}\right)e^{i(\theta-\phi)} \tag{3.9}$$

Now taking the dot product between equations 3.8 and 3.9 produces

$$1 = \frac{C \sin(\phi)}{\left(1 - \frac{d\phi}{d\theta}\right)} \qquad (3.10)$$

From Figure 3.1, sin(ϕ) is:

$$\sin(\phi) = \frac{\sqrt{C^2 - (h - r_1)^2}}{C} \qquad (3.11)$$

Then substitute equations 3.7 and 3.11 into equation 3.10 to obtain

$$1 = \frac{C^2 - (h - r_1)^2}{\sqrt{C^2 - (h - r_1)^2} + \frac{dh}{d\theta}}, \qquad (3.12)$$

where the derivative of h with respect to θ is

$$\frac{dh}{d\theta} = \frac{L}{\int_{\theta_{min}}^{\theta_{max}} t(\theta) d\theta} \frac{dt}{d\theta} \qquad (3.13)$$

The equations 3.1 through 3.13 are sufficient for the synthesis of constant tension GGRR wrapping cams.

For wrapping cams with crossed-chains and constant tension, as shown in Figure 3.2 below, there are some slight modification to the above equations. The cam rotates in the opposite direction of the sprocket rotation.

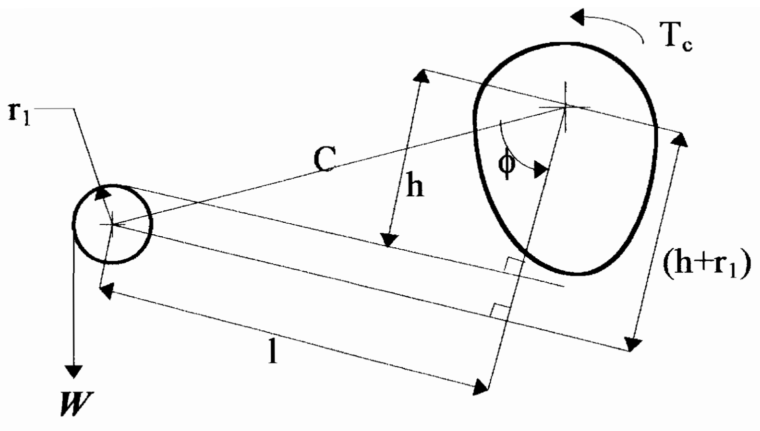


Figure 3.2. Variables of Crossed-Chain GGRR Wrapping Cam

For a wrapping cam with a crossed-chains, ϕ is given by $\phi = \cos^{-1}\left(\frac{(h + r_1)}{C}\right)$. To find the distance between the contact point on the sprocket and the contact point on the cam's surface, equation 3.12 for wrapping cam with crossed-chain becomes:

$$l = \frac{C^2 - (h + r_1)^2}{\sqrt{C^2 - (h + r_1)^2} + \frac{dh}{d\theta}} \tag{3.14}$$

Another configuration of wrapping cam is one that has a weight hanging from a sprocket at a radius r_2. This input sprocket is rigidly attached to the cam (Figure 3.3). The mechanism will have a contant cam torque and will produce a nonlinear chain or belt force which in turn will produce a nonlinear torque, T_s, in the output sprocket with radius r_1. This type of wrapping cam configuration is also called a negative wrapping cam. Let ψ be the rotation of the output sprocket. The mechanism to be synthesized should produce the specified torque as a function of sprocket rotation, $T_s(\psi)$.

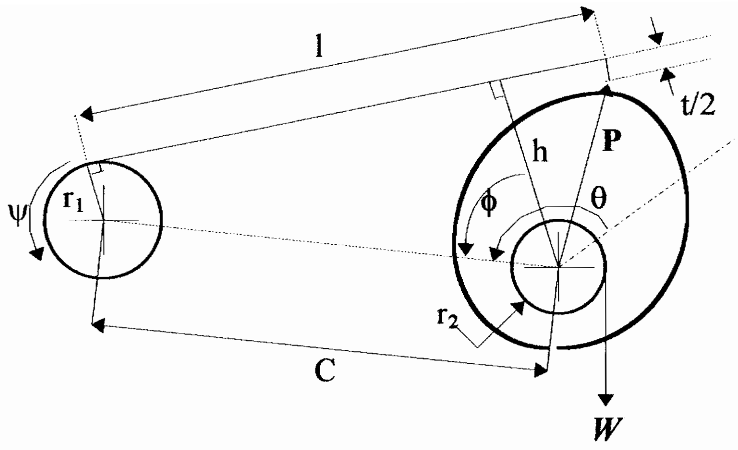


Figure 3.3. Negative GGRR Wrapping Cam

The equations for this constant cam torque (negative) GGRR synthesis case are slightly different than the constant tension wrapping cam cases. There is an additional variable ψ as function of θ in the synthesis equations. Using the chain rule on equations 3.7 and 3.10 should obtain equations 3.15 and 3.16 below:

$$\frac{d\phi}{d\psi}\frac{d\psi}{d\theta} = \frac{-1}{\sqrt{C^2-(h-r_1)^2}}\frac{dh}{d\psi}\frac{d\psi}{d\theta}$$

$$\frac{d\phi}{d\psi} = \frac{-1}{\sqrt{C^2-(h-r_1)^2}}\frac{dh}{d\psi} \tag{3.15}$$

$$l = \frac{C\sin(\phi)}{\left(1-\dfrac{d\phi}{d\psi}\dfrac{d\psi}{d\theta}\right)} \tag{3.16}$$

The rotation velocities of the sprocket and the cam should be equal thus,

$$h\frac{d\theta}{dt} = -r_1\frac{d\psi}{dt},$$

and

$$\frac{d\psi}{d\theta} = \frac{-h}{r_1} \tag{3.17}$$

And the cam's moment arm, h is

$$h = \frac{k}{t(\psi)} \text{ where } k \text{ is a constant equal to } \frac{r_1 r_2}{L}\int_0^{\psi_{max}} t(\psi)d\psi, \tag{3.18}$$

then taking derivative of h with respect to ψ to obtain

$$\frac{dh}{d\psi} = \frac{-k}{t^2(\psi)}\frac{dt}{d\psi} \tag{3.19}$$

Substitute equations 3.17, 3.18, and 3.19 into equation 3.12 to yield the distance between the contact point on the output sprocket and the contact point on the cam surface as

$$l = \frac{C^2 - \left(\frac{k}{t(\psi)} - r_1\right)^2}{\sqrt{C^2 - \left(\frac{k}{t(\psi)} - r_1\right)^2 + \frac{k^2}{t^3(\psi)r_1}\frac{dt(\psi)}{d\psi}}} \tag{3.20}$$

The cam angle of rotation (assuming $\theta_0 = 0$ and $\psi_0 = 0$ at the same place) is

$$\theta = -\frac{r_1}{k}\int_0^{\psi} t(\psi)d\psi \tag{3.21}$$

and the point on the cam surface can be found using equation 3.1.

3.2. GRRR Wrapping Cam Synthesis

Another type of wrapping cam, the GRRR mechanism is also discussed by [Tidwell, 1995]. There is a minor error in the equation for GRRR synthesis derived by Tidwell

[Tidwell, 1995]. Tidwell's GRRR synthesis method does not put into consideration the effect of the link on the design. This thesis presents the proper synthesis for GRRR wrapping cam with one end of the belt attached to a link. The center of mass and the weight of this link causes tension in the belt that wrappes around the cam. The user specifies the torque on the cam as function of the cam rotation. This type of GRRR wrapping cam can be used for counter balance mechanism. Figure 3.4 shows the variables used in deriving the loop-closure method of GRRR synthesis.

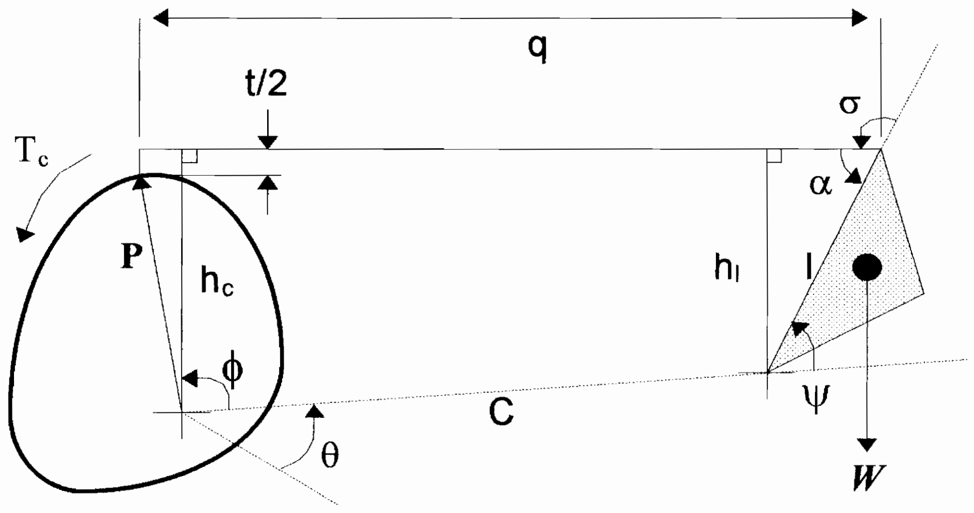


Figure 3.4. Variables of GRRR Wrapping Cam

Some initial parameter values are needed to perform the synthesis of GRRR wrapping cam. They are the distance between the pivot points of the cam and the link also called the center distance (C), the length of the link attached to the belt (l), the link's center of mass and weight, the link's initial position angle (ψ_0), and the thickness of the belt (t). In analyzing this mechanism, a conjugate geometry method is used. The cam is fixed, while the link, the belt, and the center distance rotate relative to it. A reference line on the cam is used for angle measurements. By traversing a vector loop, the equation for points on the cam surface in complex form is obtained as

$$\vec{P} = Ce^{i\theta} + le^{i(\theta+\psi)} + qe^{i(\theta+\psi+\sigma)} + \frac{t}{2}e^{i(\theta+\psi+\sigma+\pi/2)} \tag{3.22}$$

Using equation 3.22, each point on the cam surface is calculated for a given cam rotation θ. First the other variables in equation 3.22 must be calculated.

The condition of contact on the cam surface is $\frac{d\vec{P}}{d\theta} \cdot \hat{n} = 0$. Therefore, the distance (q) measured at the belt's center between the points of contact on the cam surface and the link is determined by calculating the dot product between

$$\frac{d\vec{P}}{d\theta} = Ce^{i(\theta+\pi/2)} + l\left(1 + \frac{d\psi}{d\theta}\right)e^{i(\theta+\psi+\pi/2)} + \left(\frac{dq}{d\theta} - \frac{t}{2}\left(1 + \frac{d\psi}{d\theta} + \frac{d\sigma}{d\theta}\right)\right)e^{i(\theta+\psi+\sigma)}$$
$$+q\left(1 + \frac{d\psi}{d\theta} + \frac{d\sigma}{d\theta}\right)e^{i(\theta+\psi+\sigma+\pi/2)}$$

and

$$\hat{n} = e^{i(\theta+\psi+\sigma-\pi/2)},$$

which yields:

$$C\cos(\psi + \sigma - \pi) + l\left(1 + \frac{d\psi}{d\theta}\right)\cos(\sigma - \pi) + \phi + q\left(1 + \frac{d\psi}{d\theta} + \frac{d\sigma}{d\theta}\right)\cos(-\pi) = 0$$

then solving for q,

$$q = \frac{-C\cos(\psi + \sigma) + l(1 + \frac{d\psi}{d\theta})\cos\sigma}{(1 + \frac{d\psi}{d\theta} + \frac{d\sigma}{d\theta})} \tag{3.23}$$

Based on the conservation of energy, the work produced by the cam is equal to the work done raising the weight of the link: $\int_{\psi_{min}}^{\psi_{max}} T_l d\psi = \int_0^{\theta_{max}} T_c d\theta$. T_l is the torque acting on the link, while T_c is the torque produced at the cam. In this analysis, the link acts as the input member while the cam acts as the output member.

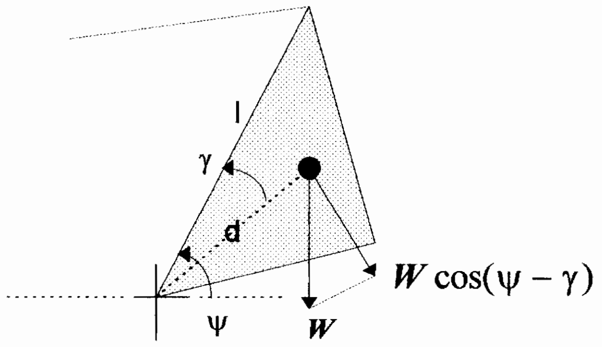


Figure 3.5. Torque Produced by the Link

Assuming the weight 'W' produces the torque on the link T_l and the user specifies the torque on the cam $T_c(\theta)$, then the link's angle of rotation ψ can be calculated as follows. Start with a torque balance on the link of Figure 3.5.

$$T_l = d \cdot W \cos(\psi - \gamma), \tag{3.24}$$

where d is the distance from the link's pivot point to the link's center of mass, W is the weight of the link, and γ is the offset angle.

By performing the integration of $\int_{\psi_{min}}^{\psi_{max}} T_l d\psi = \int_0^{\theta_{max}} T_c d\theta$, the link's angle of rotation ψ for a given θ is equal to:

$$\psi = \sin^{-1}\left(\frac{1}{d \cdot w}\int_0^{\theta} T_c d\theta + \sin(\psi_{min} - \gamma)\right) + \gamma \tag{3.25}$$

where ψ_{min} is the initial position of the link when the cam angle of rotation θ is equal to zero.

From the equation 3.25 the first and second derivatives of ψ with respect to θ can be obtained as

$$\frac{d\psi}{d\theta} = \frac{T_c(\theta)}{d \cdot w \cdot \cos(\psi - \gamma)} \tag{3.26}$$

$$\frac{d^2\psi}{d\theta^2} = \frac{1}{\cos(\psi - \gamma)} \left\{ \frac{1}{d \cdot w} \frac{dT_c}{d\theta} + \sin(\psi - \gamma) \left(\frac{d\psi}{d\theta} \right)^2 \right\} \tag{3.27}$$

Another variable to be determined before solving equation 3.22 is the angle between the belt and the link (σ).

$$\sigma = \pi - \alpha \tag{3.28}$$

To obtain the angle α, the method of instantaneous centers is used. A line is extended from the belt's center-line until it intersects with an extension of the center distance (Figure 3.6). The angle α is measured from the belt center line to the link. From similar triangles, the length IC is obtained:

$$\frac{IC - C}{IC} = \frac{h_c}{h_1}$$

$$IC = \frac{C}{\left(1 - \frac{h_c}{h_1}\right)} \tag{3.29}$$

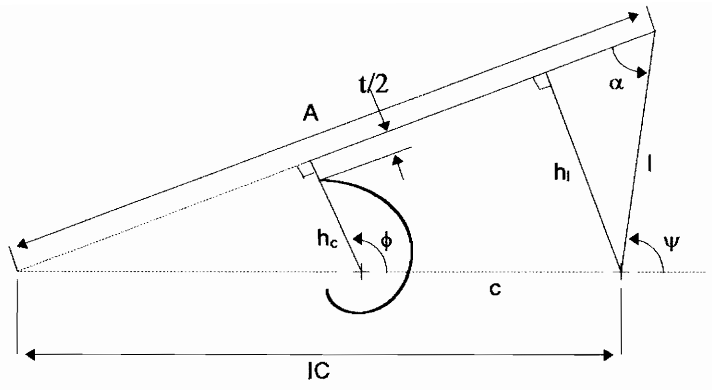


Figure 3.6. Instantaneous Centers Method

The torque acting on the cam T_c is equal to $T_c = F \cdot h_c$, where F is the amount of tension in the belt and h_c is the moment arm of the cam. Meanwhile, the torque acting on the link is equal to $T_l = F \cdot h_l$

$$F = \frac{T_c}{h_c} = \frac{T_l}{h_l},$$

thus $\dfrac{T_c}{T_l} = \dfrac{h_c}{h_l}$ (3.30)

Knowing the work done by the torque on cam is equal to the work done by the torque on the link $\int_{\psi_{min}}^{\psi_{max}} T_l d\psi = \int_0^{\theta_{max}} T_c d\theta$, then $\dfrac{T_c}{T_l} = \dfrac{d\psi}{d\theta}$, thus

$$\frac{h_c}{h_l} = \frac{d\psi}{d\theta}, \tag{3.31}$$

and solving for h_c,

$$h_c = h_l \cdot \frac{d\psi}{d\theta} \text{ or } h_c = l \cdot \sin\alpha \cdot \frac{d\psi}{d\theta}$$

By substituting equation 3.31 into equation 3.29 the value of IC can be obtained as:

$$IC = \frac{C}{\left(1 - \frac{d\psi}{d\theta}\right)}$$

Taking the derivative of IC with respect to θ

$$\frac{dIC}{d\theta} = \frac{C \cdot \frac{d^2\psi}{d\theta^2}}{\left(1 - \frac{d\psi}{d\theta}\right)^2}$$

From the geometry of Figure 3.6, the length A is equal to

$$A^2 = (IC)^2 + 1^2 - 2 \cdot 1 \cdot (IC) \cdot \cos(\pi - \psi)$$

$$\frac{dA}{d\theta} = \frac{1}{2}\left\{2\frac{dIC}{d\theta} - 2 \cdot 1 \cdot \frac{dIC}{d\theta} \cdot \cos(\pi - \psi) - 2 \cdot 1 \cdot IC \cdot \sin(\pi - \psi) \cdot \left(\frac{d\psi}{d\theta}\right)^2\right\}$$

And the angle α is equal to

$$\alpha = \cos^{-1}\left(\frac{A^2 + 1^2 - (IC)^2}{2 \cdot 1 \cdot A}\right) \tag{3.32}$$

$$\frac{d\alpha}{d\theta} = \frac{-1}{\sin\alpha}\left\{\frac{\left(2 \cdot A \cdot \frac{dA}{d\theta} - 2 \cdot IC \cdot \frac{dIC}{d\theta}\right)(2 \cdot 1 \cdot A) - (A^2 + 1^2 - IC^2)\left(2 \cdot 1 \cdot \frac{dA}{d\theta}\right)}{(2 \cdot 1 \cdot A)^2}\right\}$$

Also the angle $\alpha = \psi + \frac{\pi}{2} - \phi$, then substitute it into equation 3.28 to obtain:

$$\sigma = \phi - \psi - \frac{\pi}{2}$$

$$\frac{d\sigma}{d\theta} = \frac{d\phi}{d\theta} - \frac{d\psi}{d\theta}, \tag{3.33}$$

where

$$\phi = \cos^{-1}\left(\frac{h_c - l \cdot \sin\alpha}{C}\right) \tag{3.34}$$

$$\frac{d\phi}{d\theta} = \frac{l \cdot \cos\alpha \cdot \dfrac{d\alpha}{d\theta} - \dfrac{dh_c}{d\theta}}{\sqrt{C^2 - (h_c - l \cdot \sin\alpha)^2}},$$

where

$$\frac{dh_c}{d\theta} = l \cdot \sin\alpha \cdot \frac{d^2\psi}{d\theta^2} + l \cdot \cos\alpha \cdot \frac{d\alpha}{d\theta} \cdot \frac{d\psi}{d\theta}$$

After finding q, ψ, and σ values for given θ and $T_c(\theta)$, the points on the cam surface can be calculated by equation 3.22 for the range of link angles ψ.

4. C++/OBJECT-ORIENTED PARADIGM

Object-Oriented Programming (OOP) is a new programming paradigm that evolved in 1980s [Prata 1995]. One programming language that supports OOP is C++, an extension of the C programming language. This research used Borland C++ Ver. 4.0. Borland C++ Ver. 4.0 is compatible with Windows 3.x, Windows NT, and DOS systems.

Unlike traditional procedural programming languages which emphasize the algorithms, OOP emphasizes the data. The data is designed in such a form that it will be as close as possible to the essential features of a problem [Prata, 1995]. In the OOP, the data and operations that perform on the data are enclosed in a construct called **class**.

A class is a derive data type. Derived data type is data made up of one or more basic data types such as integers, floats, and characters. A class has attributes and methods. Attributes are characteristics of the objects that belong to the class, e.g. a box has width, length, height, and volume. Methods are functions through which the object interacts with its user, e.g. the user can set the box size. Therefore, class supposes to represent 'real-world' object. And an object can be described by its shape and functionality. For example, a class for a storage box (Figure 4.1) has height, width, length, volume, and type of item as its attributes. *set_size(length,width,height)*, *set_item(type)*, *get_item(j)*, *add_item(j)*, and *get_volume( )* are methods to set and to access the information about this class.

C++ has important characteristics that supports the object-oriented programming, such as information hiding (encapsulation), polymorphism, and inheritance. The concept of abstraction facilitates the development of classes. A class is thus also called abstract data type.

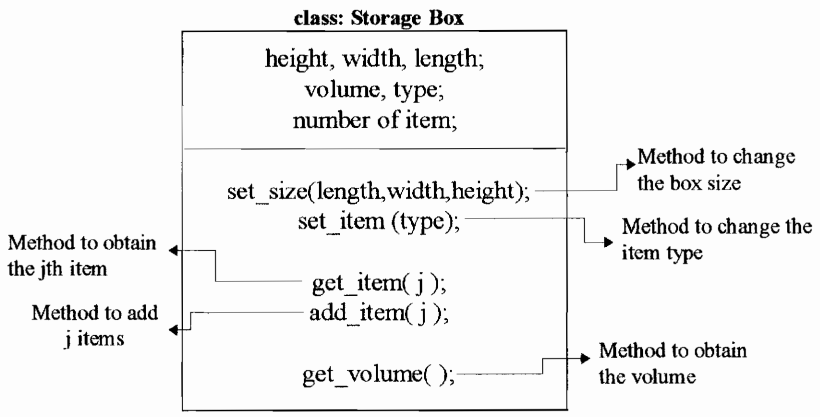


Figure 4.1. Class: Storage Box

Johnsonbaugh and Kalin [Johnsonbaugh and Kalin, 1995] define a data type as abstract if the high-level operations appropriate to the data type are isolated from the low-level implementation details associated with the data type. Encapsulation protects the classes from unauthorized access. One type of polymorphism allows multiple definition of operators and functions depending upon the context of the message received by the object. For example, the functions *rect(w, l, pt)* and *rect(upleft, loright)* of **class rectangular,** both can be used to define a rectangular object, however each of them receive different parameters. Another type of polymorphism is one where a function or method behaves differently depending on the type of object which calls it. For example the function *draw( )* in Figure 4.2 will draw the shape according to what object it relates to, i.e. whether rectangular or circular object. Inheritance eases the work for the user who wants to create new code from old code. It is quite similar to a derived data type, only in this case the user creates a derived class (Figure 4.2). **class Shape** is the base or parent class that describes a general property of shape object. From this parent class, two specific classes are derived, rectangular and circle. These two classes have different dimension variables however they share the same property belonging to **class Shape**. All these characteristics enable C++ to create a collection of objects. This collection, or rather a network, of objects is the foundation of object-oriented programming. The program output is the result of interaction between different objects in that program.

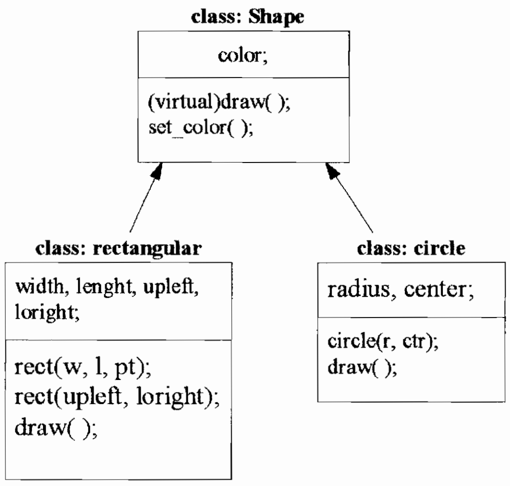


Figure 4.2. Example of Inheritance

Due to its characteristics, i.e. abstraction, encapsulation, polymorphism, and inheritance, OOP supports the concept of modularity better than procedural programming languages. Chin-Purcell and Riley [Chin-Purcell and Riley, 1989] point out some advantages of OOP in supporting a flexible and interactive program for mechanism synthesis. They developed a linkage synthesis program called *Macintosh Lincages* using OOP for Macintosh system. OOP can facilitate the development of more intuitive engineering programs, because the user interacts with the program through a set of objects rather than a set of alphanumeric functions [Deitz, 1995]. The possibility of incorporating parameters other than variable dimensions into the OOP data models is important in mechanical engineering applications. For example a mechanical part can also have stock number information, manufacturing information, material information, in addition to its dimensional shape. Engineers can thus perform product modelling instead of only geometric modelling.

5. OBJECTIVE

The research work presented here has objective to extend Tidwell's analytical synthesis methods for wrapping cam mechanisms and to implement it in an interactive wrapping cam synthesis package. This program must be user friendly and easy to maintain. The criteria of user friendliness is that a novice user can use this program without having to know all the fundamentals behind it, but still maintaining the features needed by the advance user. To provide the advance users with ease of maintainance, this package will be designed as a network of modules. It is expected that the program can be used to perform preliminary design of wrapping cam, which allows quick evaluation of a design without excessive details.

6. PROGRAM OVERVIEW

The program is written and compile in Borland C++ version 4.0. Borland's ObjectWindows Library (OWL) 2.0 is used to develop the graphical user interface (GUI). OWL classes help to simplify the design process of Windows applications.

The package consists of three main modules namely, the graphic module, the kinematic analysis module, and the synthesis module. The user will interact with the program through a graphical user interface (GUI). The GUI in this program has several windows, they are the main window, the edit-view window, the plot-view window, the analysis pop-up window, and the synthesis pop-up window. The main window, with a menu bar attached to it, will be displayed at all time. The edit-view window is a text editing window on which the user could type in the data to be used as input for cam synthesis or cam analysis or to view an output file. The plot-view window displays the scatter plot (or point plot) of the input data. The analysis pop-up window is a modal dialog window where the user types in the required parameter values to perform cam analysis on the given cam design. The synthesis pop-up window is a modal dialog window where the user types in the required parameter values to perform cam synthesis from the given strength curve or function curve. The program structure of this package is shown below. Some of the legends used in Figure 6.1 are:

- PGGRR represents the positive GGRR synthesis dialog box.
- NGGRR represents the negative GGRR synthesis dialog box.
- ANLSYS represents the GGRR analysis dialog box.

- NEWFILE represents the new input file dialog box.
- Solid line means that the user has direct interaction with the object.
- Dash line means that the user has no direct interaction with the object or the object is hidden from the user.

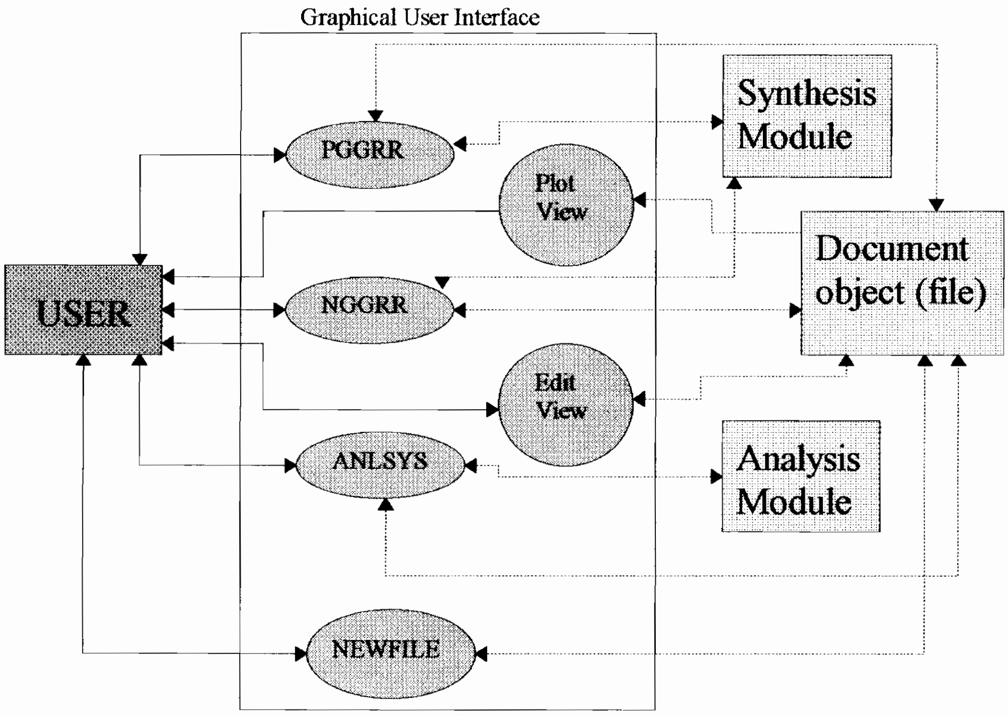


Figure 6.1. The Program Structure

As shown by the above diagram, the user is generally unaware of the processes in the synthesis and the analysis modules. By hidding these modules from direct interaction with the user, any modification in the modules can be done without affecting the rest of the program code. For example, a spline curve fitting method can replace the polynomial curve fitting method and this modification will only affect the synthesis and/or analysis modules. The user will not see the changes in the synthesis and/or analysis modules, because the user only sees the graphical user interface on the screen. This form of information hiding can add flexibility to the program.

Any cam design process involves synthesis and analysis of the cam. In the cam synthesis process, it starts with a known force function or position function data and ends with a cam profile that satisfies those data. The analysis process starts with a given wrapping cam configuration and ends with force functions or position functions as the output. The flow chart below shows the design procedures of the wrapping cam program, as shown in Figure 6.2.

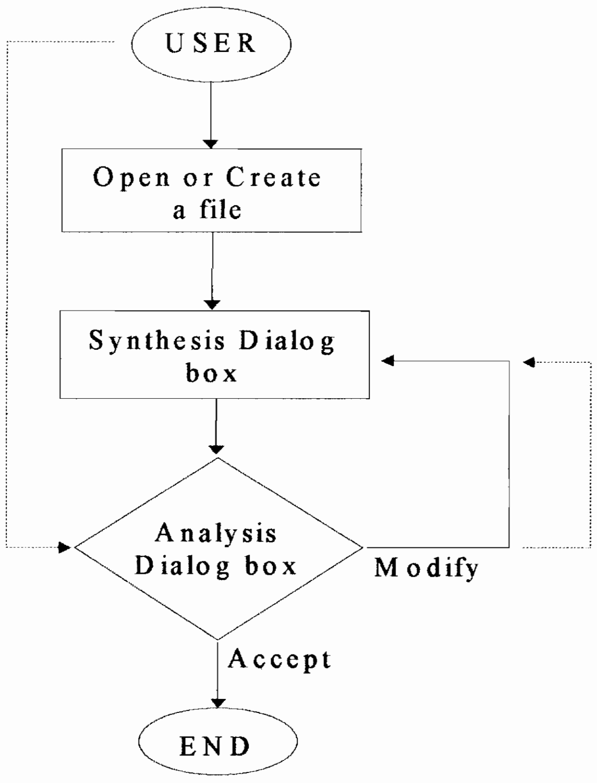


Figure 6.2. Design Process of Wrapping Cam Program

The solid line indicates the steps for a wrapping cam synthesis process. The dash line indicates the steps for a wrapping cam analysis process. Looking at the flow chart, the wrapping cam synthesis procedure in this program will be as follows:

a) Either open the existing force function or position function data file or create a new file.
b) Perform the synthesis by selecting the type of GGRR wrapping cam to be designed.
c) Either open the synthesis output file or directly perform the analysis on the results from the synthesis process.
d) Check the results, and if it is not acceptable then either modify the input file or perform another synthesis with new parameter values.

Meanwhile, to perform the analysis process on the wrapping cam mechanism, the procedures are as the following:

a) Either open the existing wrapping cam file or create a new file.
b) Perform the analysis on the cam.
c) Either open the output file or directly perform the synthesis on the result of the analysis process.
d) Check the result, and if it is not acceptable then either modify the input file or perform another analysis with new parameter values.

The user will begin with a default setting of some design parameters. For example, a wrapping cam with constant chain tension will have the following inputs and parameters:

strength curve, sprocket radius, center distance, chain or belt thickness, and weight stroke (i.e. how far the weight moves from original position). Except for the strength curve diagram, these design parameters and their values will be displayed in the synthesis pop-up window. The strength curve diagram will be displayed in the plot-view window. The synthesis pop-up window is used to change any design parameters within the specified range.

There are several configuration of wrapping cams, for example the positive (constant tension) cam (Figure 6.3), the positive cam with crossed-chain/belt (Figure 6.4), the positive spring mounted (Figure 6.5), the positive spring mounted with crossed-chain/belt (Figure 6.6), the negative cam (constant cam torque) (Figure 6.7), the negative with crossed-chain/belt (Figure 6.8), the positive linkage mounted (Figure 6.9), and the positive linkage mounted with crossed-chain (Figure 6.10). In general, these wrapping cams can be arranged into groups based on where the input force or torque is located, the type of device (e.g. weight, spring, or linkage) connected to the cam, and either crossed or straight connection.

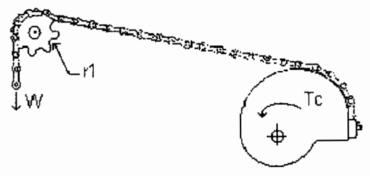


Figure 6.3. Constant Chain Tension (Positive Cam)

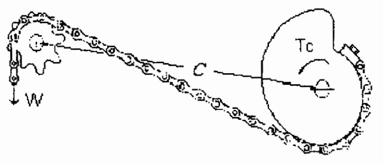


Figure 6.4. Positive Cam with Crossed-Chain

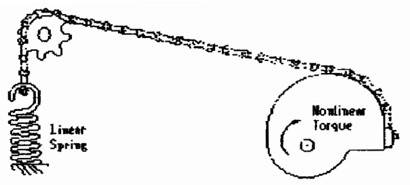


Figure 6.5. Positive Cam with Spring Mounted

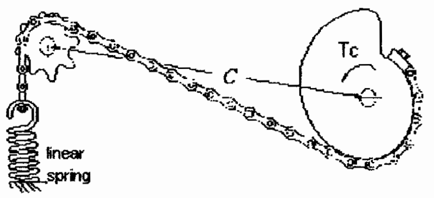


Figure 6.6. Positive Cam with Spring and Crossed-Chain

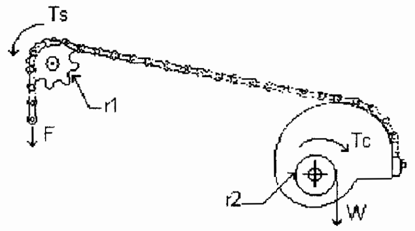


Figure 6.7. Constant Cam Torque (Negative Cam)

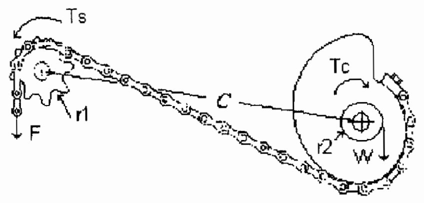


Figure 6.8. Negative Cam with Crossed-Chain

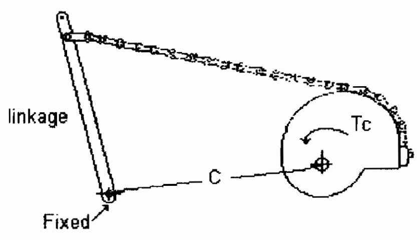


Figure 6.9. Positive Cam with Linkage

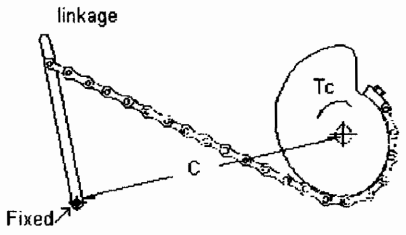


Figure 6.10. Positive Cam with Linkage and Crossed-Chain

Each configuration will require a sligthly different synthesis approach. Nevertheless, the design parameters or data are quite similar for all these configurations. The plot-view window displays the plot of wrapping cam profile. The synthesis pop-up window or the edit-view window shows the corresponding design parameters of the selected cam configuration. To see what type of strength curve a cam configuration gives, the user selects the analysis pop-up window option from the menu and types in all the required parameter values.

To provide the software that encompasses all wrapping cam configurations at this time is not practical. The wrapping cam configurations currently available in the program are the positive and negative GGRR cams. These are the configurations most commonly used in industry. This thesis also covers the object-oriented modelling of wrapping cam data used in the program package. The intent is to aid the user in developing new wrapping cam mechanisms. In order to provide a facility of adding other wrapping cam configurations, this software package is also designed to be an open-ended program. By providing reusable code, the programmer can use the existing code for most of the features of a new synthesis case. Modularity in the source code also adds flexibility to the program.

A typical GGRR wrapping cam design problem has this following inputs, i.e., the belt or chains thickness, the degree of polynomial fit on the force function data, the number of calculated points on the cam surface, the weight stroke, the center distance, the pulley's radius or the number of sprocket teeth and chain pitch, the name of input file that contains force function data, and the name of output file that contains wrapping cam data. Figure 6.11 shows the plot of a force function data used for this example.

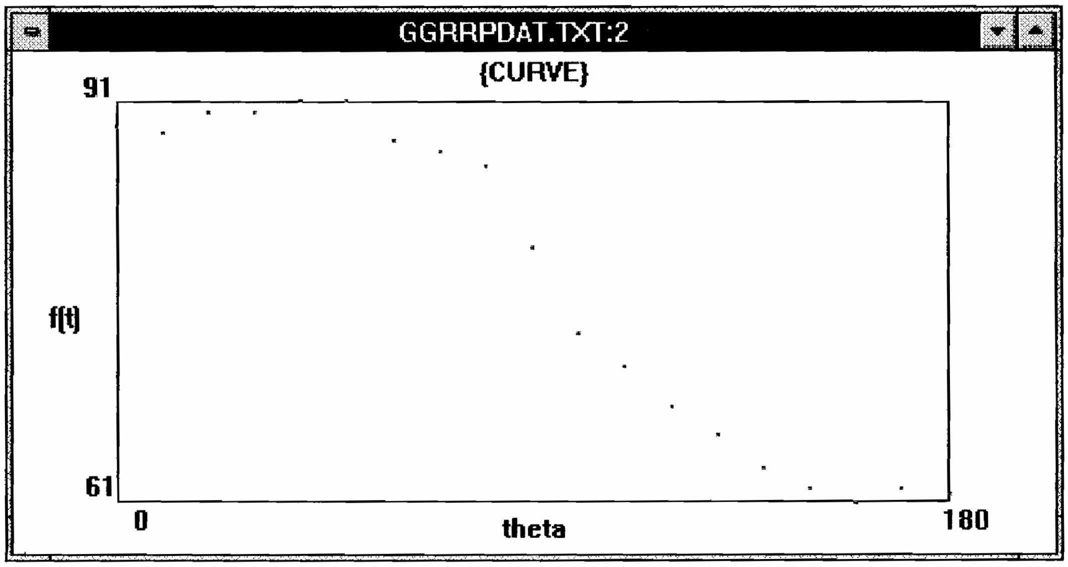


Figure 6.11. An Example of Force Function Data

This force function data is an experimental data used in Tidwell's work [Tidwell, 1995]. A proper comparison between the analysis and the actual data can thus be made. The ASCII text format of this input file is shown in Figure 6.12 below.

```
{CURVE}
19

 0      89
10      88.5
20      90
30      90
40      91
50      91
60      88
70      87
80      86
90      80
100     73.5
110     71
120     68
130     66
140     63.5
```

```
150    62
160    61
170    62
180    61.5
```

Figure 6.12. The Input Data in ASCII Text

To perform the synthesis, the user then calls one of the synthesis dialog boxes. Some of the parameters have default values, and the user can either accept or modify them. The user only needs to supply a name of the output file for this synthesis. Using the example data in Figure 6.12 as the input data, the synthesis module generates the cam surface. The result of the synthesis is stored as an ASCII text file, as shown in Figure 6.13. The plot of the cam surface is shown in Figure 6.14.

```
{GGRRP|S}
26

0.862808     -5.59456
2.10413      -5.17735
3.03107      -4.72976
3.7262       -4.27647
4.24452      -3.83395
4.62604      -3.41245
4.90189      -3.01635
5.09705      -2.64392
5.23081      -2.28742
5.31627      -1.93408
5.35955      -1.56819
5.3596       -1.17415
5.30904      -0.739842
5.19624      -0.259748
5.00829      0.263191
4.73429      0.817215
4.36837      1.38304
3.91181      1.93646
3.37406      2.45157
2.77272      2.90415
2.13278      3.27456
```

```
1.48567      3.55009
0.868349     3.7269
0.323113     3.81183
-0.101949    3.82477
-0.351313    3.80225

5.6607       278.767
5.58859      292.117
5.61765      302.654
5.6721       311.066
5.71972      317.909
5.74848      323.585
5.7556       328.394
5.74197      332.584
5.70908      336.38
5.65715      340.008
5.58427      343.691
5.4867       347.643
5.36034      352.067
5.20273      357.138
5.0152       3.00819
4.8043       9.79368
4.58208      17.568
4.36487      26.3368
4.17067      36.002
4.01523      46.3264
3.90788      56.923
3.84842      67.2913
3.82673      76.8844
3.8255       85.1549
3.82612      91.5269
3.81845      95.2789

1
15
0.5
0.4685
17.75
16
6
```

Figure 6.13. A Wrapping Cam Data in ASCII Text

The output of a synthesis process, i.e., the wrapping cam data, can then be used as input data for the analysis module. The analysis module generates the moment arm versus cam angle of rotation.

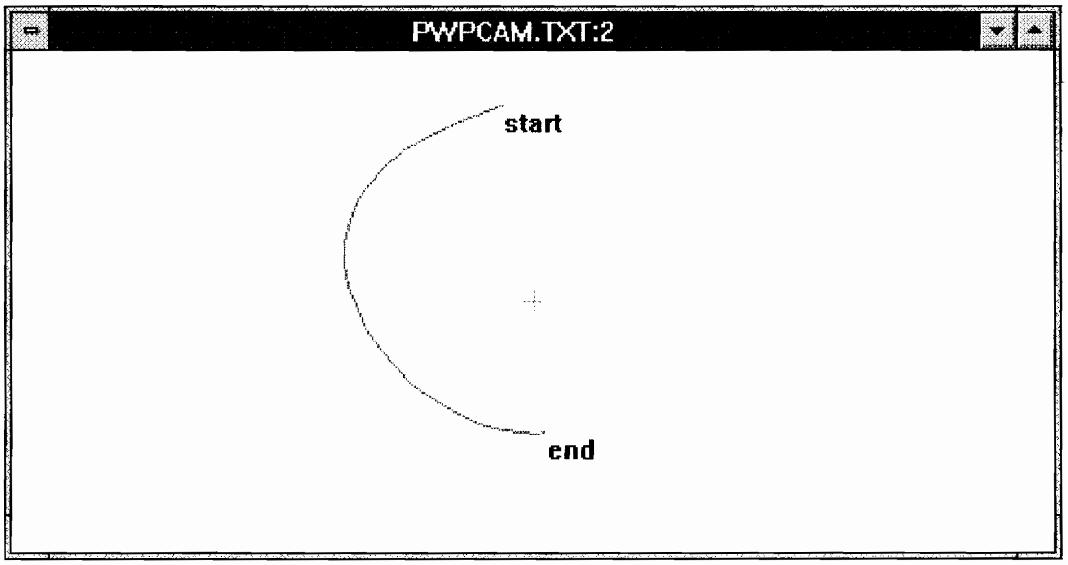


Figure 6.14. Plot of Cam Surface

An output from the analysis module is stored in ASCII text file, as shown in Figure 6.15. The plot of moment arms versus the cam rotations is shown in Figure 6.16.

```
{A_GGRRP|S}
25
1.29976     5.89096
8.36818     5.80333
15.4303     5.80481
22.4886     5.85324
29.5455     5.91557
36.6024     5.96698
43.6607     5.99007
50.721     5.97405
57.7839     5.91386
64.8493     5.80936
71.9172     5.66449
78.9871     5.4864
```

```
86.0586     5.28465
93.131      5.07035
100.204     4.8553
107.276     4.65121
114.348     4.46878
121.418     4.31692
128.486     4.20191
135.552     4.12653
142.616     4.08924
149.678     4.08333
156.738     4.09613
163.797     4.10809
170.857     4.09198
5
```

Figure 6.15. Moment Arms vs. Cam Rotations in ASCII Text

The profile of this curve conforms with the profile of the input data, however the force or torque magnitude is not the same. To obtain the force or torque magnitude for a given cam rotation, a tension in the belt or chains due to weight must be determined.

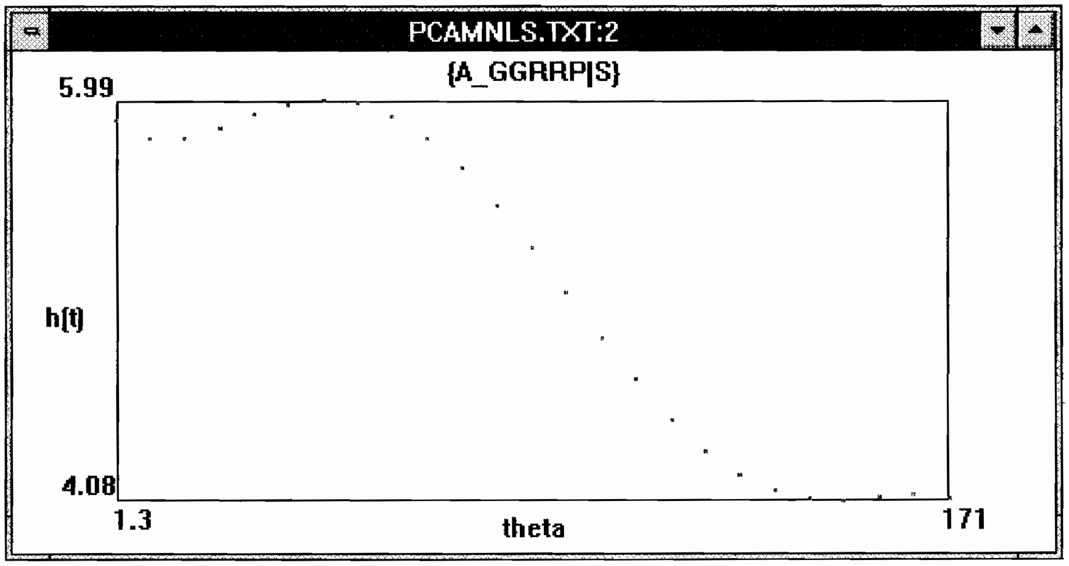


Figure 6.16. Plot of Moment Arms vs. Cam Rotations

7. GRAPHICAL USER INTERFACE (GUI)

The graphical user interface (GUI) in this program was written using Borland ObjectWindows Library (OWL) version 2.0. OWL are classes that encapsulate functions of Windows Application Program Interface (API). Using OWL assures the program's GUI will conform with the object-oriented programming paradigm [Swan et al, 1994].

The program is built on a multiple document interface (MDI) model to provide opening of multiple files. The user may like to open different files at one time to either modify them or just to compare the contents.

To provide both graphical and editing view, this program is built on document/view model. There are two types of views, text edit view and plot view, and one type of document, Borland OWL's TFileDocument, which is an ASCII text file document. TFileDocument is an OWL document class. At this moment, the user must use text edit-view mode to create new input data. The graphical view of the data cannot be modified directly; modifications can only be done through edit-view window.

The window layout in this program follows the standard windows layout with title bar, menu bar, control bars, and status bar. To see what each button represents, move the mouse's pointer onto the buttons, and a hint shows up in the status bar describing the function of the button. The menu in the main window consists of File, Edit, Search, Synthesis, Analysis, Window, and Help. Under File, Edit, and Search menus, submenus contain choices for executing standard operations, e.g. File|Open, Edit|Cut, etc. To perform a GGRR wrapping cam synthesis, the user must select one of the

Synthesis|GGRR menus. There are positive and negative wrapping cams, and their synthesis menus are Synthesis|GGRR|Positive and Synthesis|GGRR|Negative, respectively. The analysis of GGRR wrapping cam is under Analysis|GGRR menu. At the beginning of the program, the main window is displayed with menu and tools or buttons attached to it. The menu in the main window at the start of the program has only two choices which are File|Open and File|New, the rest of the menu choices are grayed. Graying the rest of the menu and buttons reduces the chance of choosing a wrong operation. If the user selects either File|Open or File|New commands, the grayed menus become clear and selectable. The layout of the main window is shown below

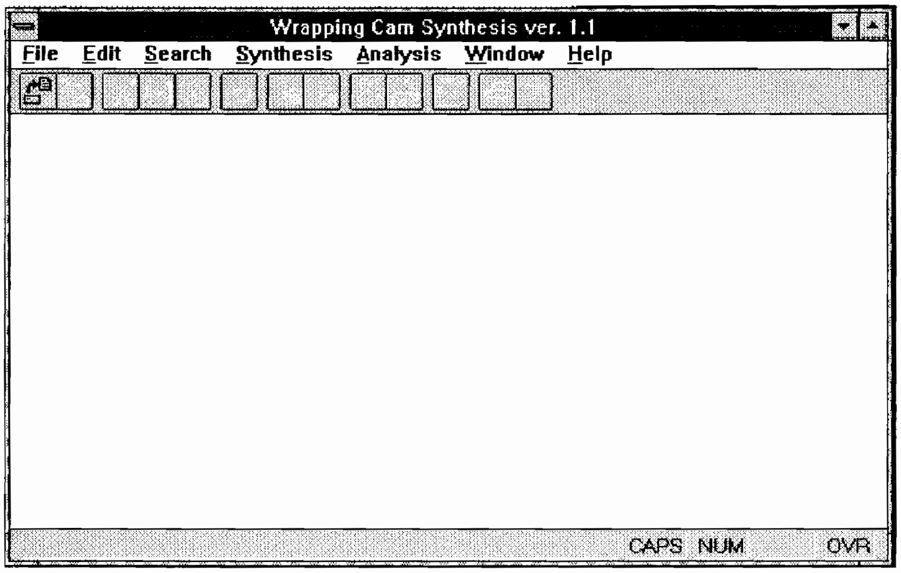

Wrapping Cam Synthesis ver. 1.1
File Edit Search Synthesis Analysis Window Help
CAPS NUM OVR

Figure 7.1. Main Window Layout

This program implements several types of Borland OWL defined Windows. They are the **myMDIClient** derived from **TMDIClient** class, the **myMDIChild** derived from **TMDIChild** class, and the **TGraphView** derived from **TWindowView** class. The user interacts with the program through a type of window called a dialog box. The following subsection covers in some details the Windows interface of this program.

7.1. Windows Interface

The Windows interface function as a wrapper that enclosed the actual operations of the program. The program's Windows interface follows the rules of Borland OWL applications. The whole graphical user interface is enclosed by a **TApplication** derived class. In this program this derived class is called **myWPCamApp** class. The functions of controlling multiple document interface (MDI) is given to **myMDIClient** class. When a file is opened or created, a new window is also created. This new window is an instance of **myMDIChild** class. What special about this window is how it interprets the view of the file. Depending upon the view type chosen, the view on this window can be either an edit-view window or a plot-view window. The diagram (Figure 7.2) below shows the program's encapsulation.

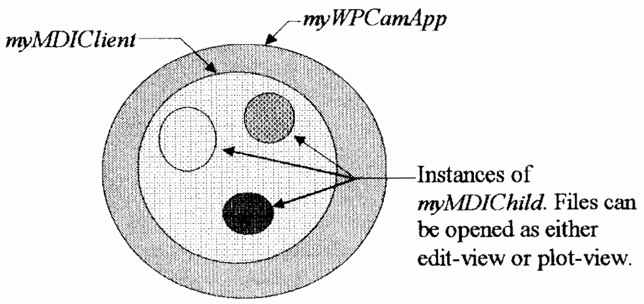
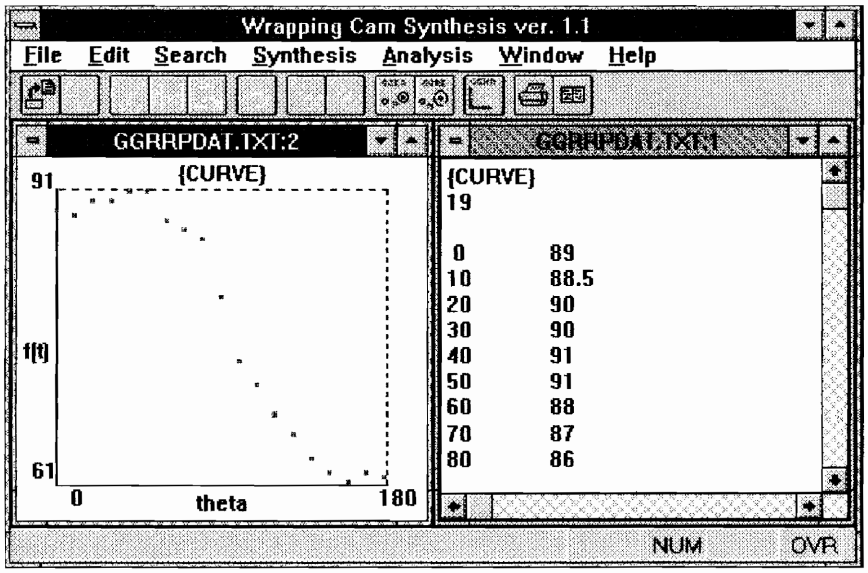

myWPCamApp
myMDIClient
Instances of *myMDIChild*. Files can be opened as either edit-view or plot-view.

Figure 7.2. Encapsulation of Windows Interface

To view the data in a file as text, the edit-view window is used. Currently, the user can only modify the file if it is opened as edit-view window. The plot-view window only provides a graphical view of the data. The user can not edit the data in this view. Because this program implements the document/view model, a single file can be open in multiple views. To open the file in different views, the user selects Window|New Window and the type of view. The figure below shows a file in both the edit view and the plot view windows, as shown in Figure 7.3.


Wrapping Cam Synthesis ver. 1.1
File Edit Search Synthesis Analysis Window Help
GGRRPDAT.TXT:2
{CURVE}
91
f(t)
61
0
theta
180
GGRRPDAT.TXT:1
NUM
OVR

```
{CURVE}
19

0       89
10      88.5
20      90
30      90
40      91
50      91
60      88
70      87
80      86
```

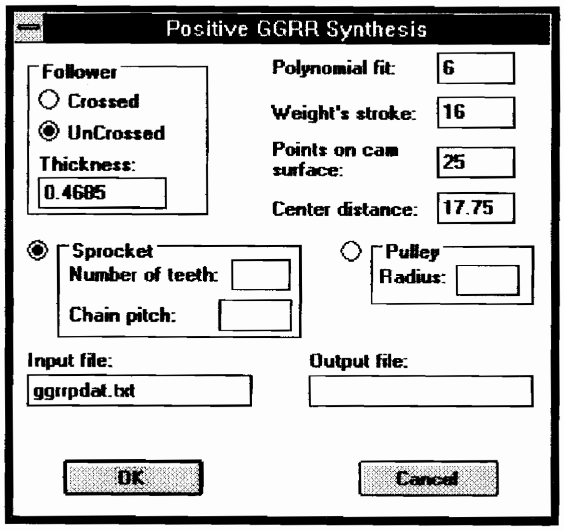
Figure 7.3. Both Plot and Edit View Windows Opened

The user interacts with the synthesis and the analysis modules through dialog boxes. Modal dialog box means that other operations are blocked while a dialog box is still active. The reason to use modal dialog box is to ensure that all design parameters have been given their appropriate values before performing any design calculation. The dialog box for the GGRR wrapping cam synthesis has the following formats:

1. The user types in the parameter values inside the edit control boxes.

2. Some of the edit control boxes are initialized with default values. The user can either accept or change them.
3. A group box is used to group the related parameters, e.g. the sprocket group box has sprocket-teeth and sprocket-pitch edit controls as its members.
4. Because GGRR wrapping cam can have either a sprocket or a pulley as one of its members, the synthesis dialog box has edit control boxes for both of them. To tell the synthesis module which one (sprocket or pulley) is selected, the user should click on the radio button next to its group box.
5. There are two push buttons on the bottom of dialog box. To accept all parameter values click the **OK** button, and then the synthesis module will execute the calculations. To discard all inputted values click the **Cancel** button, and the dialog box will be closed and no calculations performed.

The GGRR synthesis dialog boxes are represented by **GGRRPDialog** class and **GGRRNDialog** class. **GGRRPDialog** dialog box is the interface for positive GGRR wrapping cam synthesis, while **GGRRNDialog** dialog box is the interface for negative GGRR wrapping cam synthesis. The positive and negative GGRR wrapping cam synthesis dialog boxes are shown below (Figure 7.4 and 7.5).



Positive GGRR Synthesis

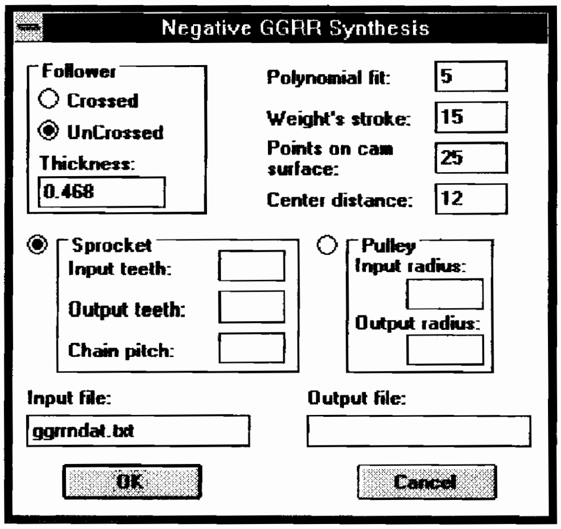
Follower
Crossed [✓ UnCrossed]
Thickness: 0.4685

Polynomial fit: 6
Weight's stroke: 16
Points on cam surface: 25
Center distance: 17.75

[✓ Sprocket]
Number of teeth:
Chain pitch:

Pulley
Radius:

Input file: ggrrpdat.txt
Output file:



Negative GGRR Synthesis

Follower
Crossed [✓ UnCrossed]
Thickness: 0.468

Polynomial fit: 5
Weight's stroke: 15
Points on cam surface: 25
Center distance: 12

[✓ Sprocket]
Input teeth:
Output teeth:
Chain pitch:

Pulley
Input radius:
Output radius:

Input file: ggrrndat.txt
Output file:

Figure 7.4. Positive GGRR Dialog Box **Figure 7.5. Negative GGRR Dialog Box**

The GGRR analysis dialog box also uses the edit control boxes where the user types in the parameter values. The analysis dialog box has fewer parameters than the synthesis box. The parameters are degree of the polynomial curve fit for cam's points, the number of data points to be calculated, the name of the input file, and the name of the output file. The polynomial curve fit and number of data points are initialized to some default values. To accept these and to initiate analysis calculations, the user must click the **OK** push button. Clicking the **Cancel** push button will discard and close the analysis dialog box. **AnalysisGGRRDialog** class represents the GGRR analysis dialog box. The analysis dialog box has the layout shown in Figure 7.6.

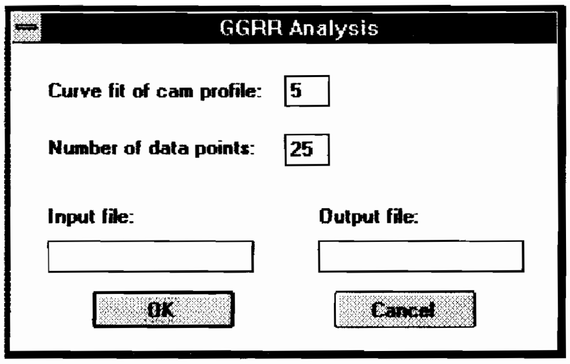


Figure 7.6. GGRR Analysis Dialog Box

The synthesis and analysis dialog boxes require an input file to exist before they can perform either synsthesis or analysis calculations. The program has a dialog box that can help the user create a new input file. The user first selects File|New Input File command from the menu. A dialog box will pop-up. The user has choices of what type of input file is to be created (e.g. curve, positive GGRR with sprocket, negative GGRR with sprocket, etc). To accept the selection, click the **OK** button and the program will create the file with the appropriate layout. The user can then open this file and type in the necessary values. Figure 7.7 shows the layout of dialog box for creating a new input file.

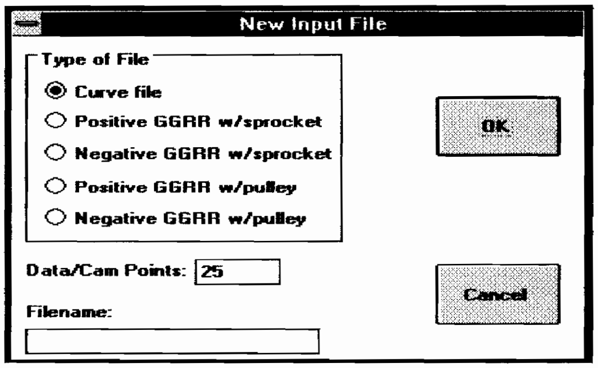


Figure 7.7. New Input File Dialog Box

The dialog box does not perform calculations, it only provides an interface between the user and the program. The synthesis and analysis modules are the ones that actually perform the calculations. The synthesis and analysis modules get their parameter values by calling the transfer-buffers of the dialog boxes. A transfer-buffer is a structure type data that is used by a dialog box to stored data.

An important part of any program is the data. The input and output data of this program are stored as files. In order for a program to properly read and write a file, a file structure must be defined. The next subsection discusses how such a file structure used in this program.

7.2. File Structure

As mentioned earlier, there is only one type of document used in this program, namely, TFileDocument. To have a uniform file output, every file must have a file structure. There are three different file structures used by this program, they are the input curve file, the output file from synthesis calculation, and the output file from analysis calculation. For the input curve file, the file structure is as follows:

```
{identification statement}
[number of data points]
x0      y0
x1      y1
...     ...
```

The output file from synthesis calculation has the following file structure:

{identification statement}
[*number of data points*]

x_0 y_0
x_1 x_1
… …
x_n x_n

cartesian coordinate form of points on cam surface

r_0 θ_0
r_1 θ_1
… …
r_n θ_n

polar coordinate form of points on cam surface

[*follower index*]
[*# of output sprocket teeth*] for {…|P} replaced with [*output pulley radius*]
[*pin separation*] for {…|P} there is no pin separation
[*follower thickness*]
[*center distance*]
[*weight stroke*]
[*degree of polynomial curve fit*]

The output file from the analysis calculation has the following file structure:

{identification statement}
[*number of data points*]
x_0 y_0
x_1 y_1
… …
x_n y_n
[*degree of polynomial curve fit*]

The program also uses nine (9) distinguishable identification statements that appears on the first line of the text file. These identification statements are enclosed within **{..}**, and the words inside the braces give a description what kind of file this is. The input curve file will have a **{curve}** statement in its first line of code. The words inside the braces can be in lower or upper case. This identification is used by the plotting function to properly produce the graphical output on the screen. Since the GGRR wrapping cam can have either a sprocket or a pulley as one of its links, this identification scheme also helps the analysis module to perform the appropriate analysis calculations. The statements **{GGRRP|P}** and **{GGRRP|S}** define the file as being the output file of the synthesis module for GGRR wrapping cam in the **positive** cam configuration. If the synthesis output is a **negative** GGRR wrapping cam, then the file has either **{GGRRN|P}** or **{GGRRN|S}** as the first statement in the file. The last letters **'P'** and **'S'** indicate whether this wrapping cam uses a **pulley** or a **sprocket**. The output from the analysis calculations will have identification statements such as **{A_GGRRP|P}** and **{A_GGRRN|P}** at first line of its output file.

In addition to being an identification of the type of output file, these identification statements are also important when a module tries to read in the data from the file. For example the data in the curve file, a file with **{curve}** in its header, is used by the synthesis module as its input. The analysis module will only read a file with **{GGRRP|P}** or **{GGRRN|P}** or **{GGRRP|S}** or **{GGRRN|S}** in its header. One of these files must exist in order for these modules to perform their operations properly. The next chapters, i.e. chapters 9 and 10, cover the synthesis and analysis modules in more details.

8. OBJECT-ORIENTED APPROACH TO WRAPPING CAM MODELLING

One important issue in software package development is ease of program maintainance. Program maintainance includes adding and/or modifying functions and data in the program's source code. To provide ease of maintainance, this package has been developed as a network of modules. The object-oriented approach or paradigm supports the concept of modularity [Zdonik and Maier, 1990].

The first step in software development using the object-oriented approach is to identify the essential objects in the program. Because an object is an instance of a class, class definitions are needed. In this wrapping cam software package, the essential objects are obviously the wrapping cam mechanisms.

The next step is to develop those classes that represent wrapping cam synthesis and analysis. There are many possible configuration of wrapping cams, but the data used in each configuration is quite similar. It is impossible and impractical to include all wrapping cam configurations in the software at this time. Fortunately, the characteristics of object-oriented programming (OOP) provide a method for creating a new class without the difficulty of writing a whole new program. Data abstraction is used to create a class of object. A new class can be created that shares common features with other classes through inheritance.

In order to take full advantage of the inheritance characteristic of OOP, the class definition must be in such a form that its abstraction is as general as possible. The user can use this already defined class as basis or 'parent' class to create new class. To aid in the software development, a hierarchical diagram is used as a tool for creating classes of wrapping cams (Figure 8.1).

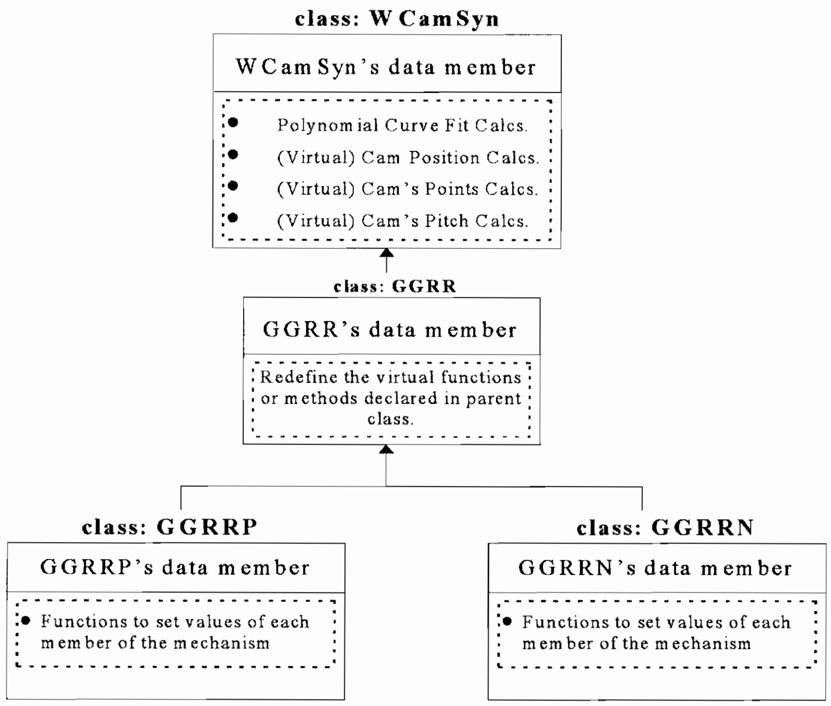

class: WCamSyn
WCamSyn's data member
Polynomial Curve Fit Calcs.
(Virtual) Cam Position Calcs.
(Virtual) Cam's Points Calcs.
(Virtual) Cam's Pitch Calcs.
class: GGRR
GGRR's data member
Redefine the virtual functions or methods declared in parent class.
class: GGRRP
GGRRP's data member
Functions to set values of each member of the mechanism
class: GGRRN
GGRRN's data member
Functions to set values of each member of the mechanism

Figure 8.1. Hierarchical Diagram of GGRR Classes

A wrapping cam is a subset of general cam mechanisms. The definition of cam is the starting point for wrapping cam modeling. The fundamentals of cam design, which are synthesis and analysis [Mabie and Reinholtz, 1987], are the starting point for data abstraction. To have a proper wrapping cam abstract data type, all factors involved in cam design must be considered. Those factors are as follows:

1. Wrapping cams can have many possible configurations. Nevertheless, some wrapping cams have quite similar data. Data sharing between different configuration brings about the implementation of inheritance. Different wrapping cams can actually be

created from a single class; i.e, in OOP this class is called the base class or parent class.

2. Any cam synthesis requires as input the type of output motion or force the cam should produce [Liang and Quinn, 1991]; hence, the wrapping cam synthesis class needs to have a function or method for reading input curves. Since every wrapping cam synthesis requires this , the base class should contain such a method.
3. The final output of wrapping cam synthesis is the cam's profile. The synthesis base class should have a method to represent this output, because this output data is used by all wrapping cams.
4. Even though several wrapping cam configurations use similar methods, implementation of them for a particular cam is likely to be different. The OOP concept of polymorphism and virtual functions provide a way to redefine the actual implementation of a method.
5. A new wrapping cam class is derived from the base class by either adding or redefining data and/or methods. The crossing of the follower (belt or chain) in GGRR wrapping cam, however, does not produce another derived wrapping cam class, because it does not requires new data and the method implementation does not change significantly.
6. Portability of the program's input and output must also be considered. This program's input and output are ASCII text files only. This ensure that other programs can also use these files.

7. The members of a wrapping cam mechanism have dimensional values such as the sprocket's radius, the follower thickness, etc. The wrapping cam class definition must includes these parameter as its data members.

The first class to be defined is **WCamSyn** class which acts as the base class for other wrapping cam synthesis classes. **WCamSyn** class has a function to perform a curve fitting method on the force synthesis or function generation data points, which is required prior to any calculation of wrapping cam synthesis. The other member functions are declared to be virtual functions.

There are two synthesis classes in this program, one represents positive GGRR wrapping cam and the other represents negative GGRR wrapping cam. The reason for having two different synthesis classes for GGRR wrapping cam is because each mechanism requires different data members (attributes) and implementation of methods. However, both positive and negative GGRR classes use similar equations to determine points on the cam surface. Both classes also implement the least-square method in their polynomial curve fitting. Because of these similarities, the positive and negative GGRR wrapping cam objects share the same base or parent classes. And from these base classes the specific GGRR wrapping cam classes are derived. The calculations for cam position, cam's points, and cam pitch are declared as virtual functions in the **WCamSyn** class, so that a different wrapping cam class can use this base class. For example a GRRR wrapping cam synthesis may use the polynomial curve fitting but have different synthesis method, therefore it will require different calculations to find the cam positions and the cam's points. Meanwhile to facilitate a mean of communication between the user and these classes, this program uses the graphical user interface (GUI). The GUI of this package is discussed in previous chapter 7.

9. SYNTHESIS MODULE

The synthesis module is represented by the **GGRRP** class and the **GGRRN** class. These two classes are derived from the **GGRR** class. The **GGRRP** class is the synthesis module for the positive GGRR wrapping cam mechanism. The **GGRRN** class is the synthesis module for the negative GGRR wrapping cam mechanism. The **GGRR** class has the **WCamSyn** class as its base class. The synthesis module consists of several methods and data members. These methods and data members directly affect the behavior of the synthesis module itself. Since every cam design or synthesis requires the force functions or position functions to be specified before hand, a class with a method to process these input data can act as the base class from which the other synthesis classes will be built. This base class must also have methods to represent the synthesis results. The cam profile is represented by both real-imaginary (Cartesian) coordinate and polar coordinate forms. The Cartesian coordinate form is used by the plotting functions, since the program does not plot in polar form. The polar coordinate form is used in analysis calculations to obtain the output curve.

For simplicity of the calculations, the data points from input motion curve are interpolated using polynomial equation with a maximum degree of ten (10). Degree of polynomial is set at maximum of ten to avoid excessive oscillation in the data interpolation. The interpolation calculation is done using the least square method. The function ***void** poly_coef (**int** n, **int** d, **double*** x, **double*** y)* is a member of the **WCamSyn** class that performs least square curve fitting prior to the synthesis calculation.

The **WCamSyn** class also has several pure virtual functions. A virtual function does nothing for the base class, and it has to be redefined by the derived class. The functions ***void** cam_motion(**int** n), point2d* point_vector(), and point2d* pitch_point()* are pure virtual functions. Different wrapping cam configuration uses different methods to calculate points on the cam surface. By declaring these methods as virtual functions, the derived class only needs to modify the virtual functions and still uses the other non-virtual functions. To convert from Cartesian coordinates to polar coordinates, this class also has *point2d* polar_coor (**int** n, point2d* cartesian)*, which reads in the Cartesian points and returns points in the polar form. The **WCamSyn** base class is declared as follows

```
class WCamSyn{
   public:
            WCamSyn(){polar = NULL;}
            ~WCamSyn(){delete []polar;}
   //Member function to calculate the polynomial function
   //coefficients by least square method. Input: number of
   //data points(n), degree of polynomial(d), abscisca (x), and
   //ordinate (y). This function also calculates the xmin
   //and xmax; the lower and upper boundary of input data.
            void poly_coef(int n, int d, double *x, double *y);
   //Member function to convert Cartesian to polar coordinates.
            point2d* polar_coor(int n, point2d* cartesian);
   //a pure virtual function to find the cam range of motion.
   //Input: number of the cam position(n).
            virtual void cam_motion (int n) = 0;
   //a pure virtual function to calculate cam surface.
            virtual point2d* point_vector() = 0;
   //a pure virtual function to calculate cam pitch.
            virtual point2d* pitch_point() = 0;
   protected:
            ArrayDb coef;
            point2d* polar;
            double xmin, xmax;
            int degree, numData;
};
```

In the **GGRR** class, virtual functions or methods of **WCamSyn** are redefined. The function ***void** cam_motion **(int** n)* determines the cam positions for the GGRR wrapping cam based on the number of cam points to be calculated. The function *point2d* point_vector ( )* calculates and returns the points on the cam surface in Cartesian coordinates. And the *point2d* pitch_point ( )* calculates and returns the points on the cam pitch surface in Cartesian coordinate. The **GGRR** class is declared as follows

```
class GGRR : public WCamSyn{
    public:
            GGRR(); //default constructor
            ~GGRR();        //default destructor

    //Redefined pure virtual functions
            void cam_motion(int n);
            point2d* point_vector(); //surface in Cartesian coordinates
            point2d* pitch_point();  //pitch surface in Cartesian coordinates

    protected:
            double t, C, r1;
            double *phi, *theta, *q;
            point2d* camPts;
            point2d* camPch;
            int numPts;
};
```

The functions or methods in the **GGRR** class are used by both positive and negative GGRR wrapping cam synthesis; that is why the **GGRRP** class and the **GGRRN** class are derived from it. The main difference between positive and negative GGRR wrapping cams is the location of the applied weight. In positive GGRR wrapping cam, the weight is attached to the end of the follower (chains or belt) that wraps around the output sprocket. In negative GGRR wrapping cam, the weight is attached to a sprocket or a pulley, which is also rigidly attached to the cam. Because of this, *GGRRN* class requires a different approach to find cam positions, moment arms of the cam, and other parameters required in the calculations.

The configuration of positive GGRR wrapping cam requires the following members: an output sprocket or pulley, a chain or belt, and a cam. The function ***void*** *sprocket_radius(**int** N, **double** p)* is the **GGRRP** member function that reads in the number of teeth and pitch of the output sprocket and calculates the radius of output sprocket. If a pulley is used instead of a sprocket, ***void*** *pulley_radius(**double** rad)* is used to read in the pulley's radius. The function ***void*** *torque_moment_arm(**double** stroke)* is member function that calculates the cam's moment arms and reads in the weight stroke. The follower index, the center distance, and the follower thickness are read by the function ***void*** *follower_length_thick(**int** index, **double** center, **double** thick)*. This member function also calculates other intermediate parameters used in the synthesis equations. The **GGRRP** class is declared as follows

```
class GGRRP : public GGRR{
    public:
            GGRRP();
            ~GGRRP();
    //Member functions

            void sprocket_radius(int N, double p);        //sprocket radius
            void pulley_radius(double rad);
            void torque_moment_arm(double stroke);
            void follower_length_thick(int index, double center, double thick);

    protected:
            Sprocket gear1;
```

```
                double L;
                double *h;
                int sign;
        };
```

The negative GGRR wrapping cam mechanisms have different attributes and implementation of methods. Therefore, its synthesis requires slightly different calculations to find the cam moment arm and other intermediate parameters. This mechanism also has a sprocket or pulley attached to the cam onto which a weight is hanging. The function ***void** sprocket_radius (**int** N1, **int** N2, **double** p)* accepts the number of teeth on the output and input sprockets and the sprocket pitch, and also calculates their radius. If the mechanism uses pulleys instead of sprockets, then the funcion ***void** pulley_radius(**double** rad1, **double** rad2)* should be used. The **GGRRN** class declaration is as follows

```
        class GGRRN : public GGRR{
          public:
                GGRRN();      //default constructor
                ~GGRRN();     //default destructor

          //Member functions
          //Get radius of sprocket. Input: number of teeth for output
          //sprocket(N1), number of teeth for input sprocket(N2), and
          //chain's pin separation(p).
                void sprocket_radius(int N1, int N2, double p);
          //Get radius of pulleys. Input: radius of output pulley(rad1)
          //and radius of input pulley(rad2).
                void pulley_radius(double rad1, double rad2);
          //Function to calculate torque's moment arm. Input: the weight's stroke.
                void torque_moment_arm(double stroke);
          //Function to calculate follower length. Input: cam's branching (index =1
          // or index = -1), center distance (center), and follower thickness(thick).
                void follower_length_thick(int index, double center, double thick);

          //Redefine the virtual function of wcamsyn base class.
                void cam_motion(int n);
          protected:
                    Sprocket gear1;
                    Sprocket gear2;
                    double L, r2, k;
                    double *h, *S;
                    int sign;
     };
```

The objects of these classes builds the actual synthesis module. In this program, the user does not interact directly with these classes, but instead the user interacts through their dialog boxes and transfer-buffers. The dialog boxes for the synthesis module is discussed in chapter 7.1. The output from the synthesis module is a text file that describes the cam surface in both Cartesian (rectangular) and polar coordinates, and gives other related results such as follower index, number of sprocket teeth, etc. This output file of the synthesis can be

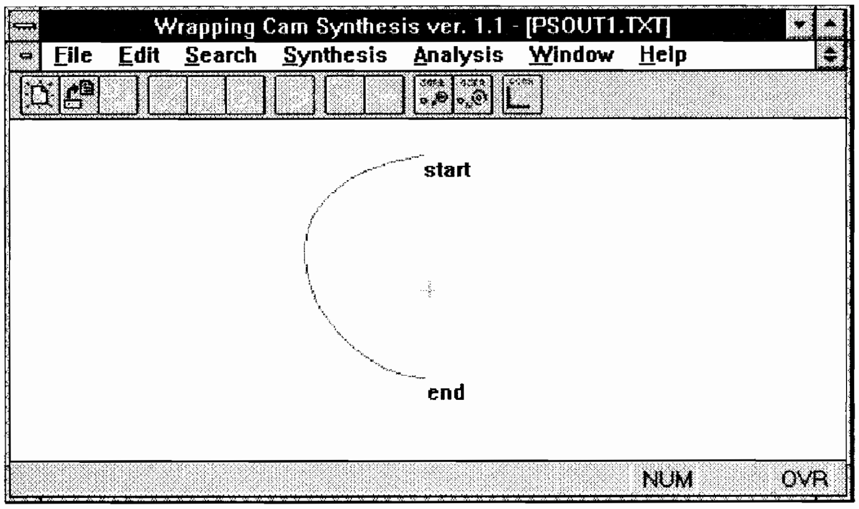

Wrapping Cam Synthesis ver. 1.1 - [PSOUT1.TXT]
File Edit Search Synthesis Analysis Window Help
start
end
NUM
OVR

Figure 9.1. Graphical Plot of Cam Profile

viewed as either text edit-view or graphical plot-view. The graphical plot-view can give a better view on the shape of the cam. And from the cam profile plot, a judgement can be made whether this cam conforms with the user specified force synthesis or function generation data. In the plot example above (Figure 9.1), the cam is plotted with the cross-mark indicates the cam's center, the word **start** indicates the starting point of contact with the follower, and the word **end** indicates the ending point of contact with the follower. The input file for the above example of wrapping cam synthesis is a force function based on

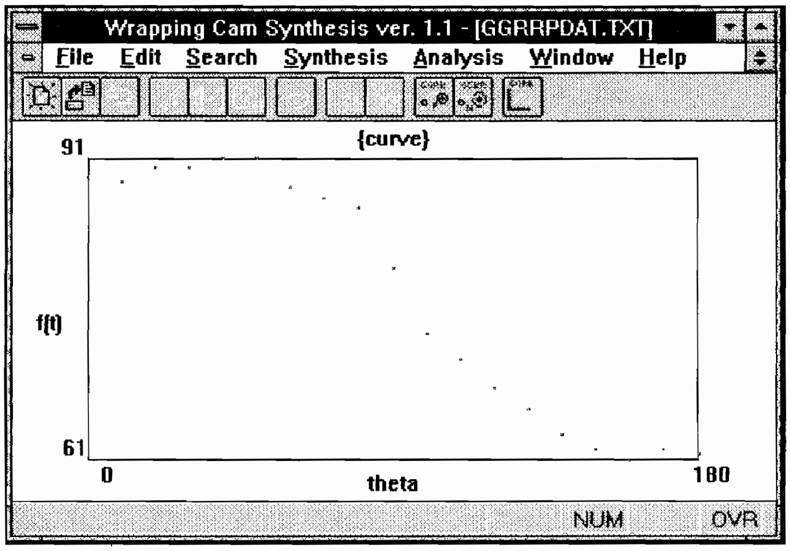

Wrapping Cam Synthesis ver. 1.1 - [GGRRPDAT.TXT]
File Edit Search Synthesis Analysis Window Help
{curve}
91
f(t)
61
0
theta
180
NUM
OVR

Figure 9.2. Plot of Force Function

the experimental data from Tidwell's work (Figure 9.2). The force is plotted with respect to the cam rotation angle theta. Before it is used as input data, this data points are interpolated by a polynomial function.

10. ANALYSIS MODULE

The kinematic analysis of the GGRR wrapping cam is the same for both positive GGRR and negative GGRR wrapping cams. The analysis module is represented by the **GGRRAnls** class. The analysis module calculates the moment arms versus the cam rotation angles. Even though the analysis equations for positive and negative GGRR cams are similar, the actual relationship between the moment arms and the cam rotation angles is not. The moment arms (h) and the cam rotation angle (θ) in the positive GGRR cam has a directional relationship. In the negative GGRR cam, the moment arms and the cam rotation angle has an inverse relationship. Therefore in order to have the correct data for plotting the analysis output of a negative GGRR cam, the moment arms (h) is inverted from 'h' to '1/h'.

Data points on the cam surface in GGRR analysis are fitted by a polynomial curve. A least-square method is employed to fit a curve onto the data. At first, the instance or object of **GGRRAnls** reads the data required for analysis. This data is polar coordinates of cam surface, the number of points on the cam surface, the follower index (uncrossed or crossed), the radius of output sprocket or pulley, the follower thickness, and the center distance. The function ***void** wrapping_cam_data (point2d* pol, **int** num, **int** index, **double** r, **double** t, **double** c)* is the member function of the **GGRRAnls** class that reads these parameters. The function *point2d* kinematic_analysis(**int** num, **int** d)* is the member function that calculates and returns data points of the output force or position function curve. And the function ***int** number_of_data ()* returns the number of data calculated. The **GGRRAnls** class is declared as follows

```
class GGRRAnls : public WCamAnls {
   public:
   //Redefine the virtual functions
   //The input: d = degree of polynomial curve fit for cam surface.
   //        num = number of data points to be calculated.
   //return output force synthesis or position function curve
           point2d* kinematic_analysis(int num, int d);

           GGRRAnls();   //default constructor
           ~GGRRAnls();  //default destructor
   //Member function to read in wrapping cam data
   //The inputs are: pol = polar coordinate
   //         num = number of cam surface data points
   //        index = the branching sign (1 = uncrossed, -1 = crossed)
   //            r = radius of output sprocket
   //            t = follower's thickness
   //            c = center distance
   //----------------------------------------------------------------------
           void wrapping_cam_data(point2d* pol,int num,int index,double r,
                  double t,double c); //read in all needed data for analysis

   //Member function to return number of data calculated
           int number_of_data(){return m;}

   protected:
   //Variables: curve = strength curve data points
   //             x = follower's position on cam surface
   //             y = radius of polar coordinate of cam suface
   //          sign = branching index (1=uncrossed, -1=crossed)
   //        org_num = number of actual cam's data points
   //            m = number of data points to be calculated
   //           tau = follower's angle position
   //         theta = cam's angle of rotation
   //             h = moment arm
   //        radius = output sprocket's radius
   //       thickness = follower's thickness
   //----------------------------------------------------------------------
           point2d* curve;
           ArrayDb x;
```

```
        ArrayDb y;
        int sign, org_num, m;
        double* h, *theta, *tau;
        double radius, center, thickness, range;
};
```

The user does not interact directly with this module, but instead through a dialog box with edit control boxes. The object or instance of the **GGRRAnls** class calls the transfer buffer of its dialog box to retrieve the parameter values. When the object successfully performs the analysis, it writes the output into a text file. The output file of the analysis calculations describes the strength curve or position function curve in Cartesian coordinates and the degree of polynomial curve used to approximate the cam surface. The result of the analysis can also be viewed as the plot-view, Figure 10.1 shows an example of a curve plot between the cam's moment arm versus the cam angle of rotation.

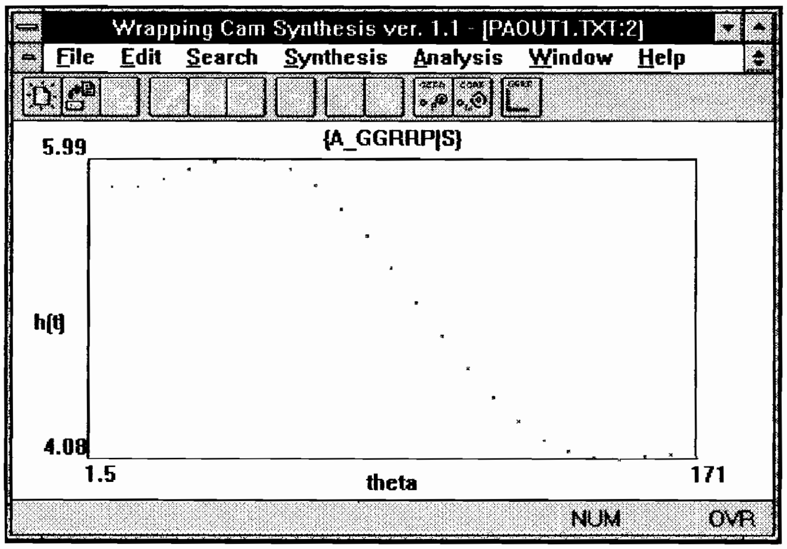


Figure 10.1. Plot of Moment Arms vs. Cam Rotations

11. CONCLUSION AND CONTRIBUTION

The program has been successfully tested against earlier MathCad and MatLab results. The current implementation has several advantages. The program developed in this work can be useful for either novice or advanced users. Using a graphical user interface (i.e. dialog boxes and other visual interfaces), the novice user can easily navigate through the program. This program is a single, stand-alone program which treats eight (8) cases (i.e. positive-uncrossed-sprocket GGRR, positive-uncrossed-pulley GGRR, positive-crossed-sprocket GGRR, positive-crossed-pulley GGRR, negative-uncrossed-sprocket GGRR, negative-uncrossed-pulley GGRR, negative-crossed-sprocket GGRR, and negative-crossed-pulley GGRR). The MathCad wrapping cam program, for example, is not a single, stand-alone program. In order to perform different wrapping cam cases the user needs to call different MathCad files. Compared to the MathCad models, this program also performs the calculation much faster. The newly developed program provides a convenient interface and gives the ability to evaluate many more alternative in the preliminary design of wrapping cams.

The GRRR wrapping cam is not implemented in this program, in part because the practical application of this type of mechanism has yet to be found. However, it is important to study this mechanism for future reference. This paper also presents the correct analytical approach for the synthesis of GRRR wrapping cam, where the weight of a link produces the tension in the belt.

This program also shows the versatility of the object-oriented programming paradigm. The encapsulation of data and methods inside an object makes the interaction between the user and the program more secure. The user does not have direct access to the synthesis and analysis modules other than through the program's graphical user interface. This gives an intuitive interface to the program which can ease the of use of this package by a novice user. Because this program is modular, any future expansion to it is possible. By implementing the object-oriented programming paradigm (OOP), this program becomes very modular which improves the flexibility to expand its scope. The advance user can add other types of wrapping cams modules to this program, as long as the code is compatible with Borland C++ version 4.0. The modularity concept also improves the effectiveness in the debugging process. It is hoped that the implementation of OOP in this research work will help encourage engineers and students to learn the object-oriented programming paradigm, because OOP appears to be the future of programming language [Simms, 1996].

All these features make this program suitable as industry-ready software. At this moment, this program runs on 16-bit IBM compatible PC with MS-Windows version 3.x. It can, however, be compiled and run on different computer platforms such as a 32-bit computer with Microsoft Windows 95 and Windows NT.

12. REFERENCES

1. Dvorak, P., 1988, "Software Speeds Cam Design," Machine Design, Vol. 60 (October 6), pp. 137-138.

2. Tidwell, P.H., Naveed Bandukwala, Sanjay G. Dhande, Charles F. Reinholtz, and Greg Webb, 1994, "Synthesis of Wrapping Cams," Journal of Mechanical Design, Vol. 116 No. 2, pp. 634-638.

3. Tidwell, Paul H., 1995, "Wrapping Cam Mechanisms," *Ph.D. Dissertation*, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, January.

4. Swan, T., Robert Arnson, and Marco Cantù, 1994, *ObjectWindows 2.0 Programming*, Random House, Inc.

5. Deitz, D., 1995, "Kinematic Analysis Programs Reduce Overdesign," Mechanical Engineering, Vol. 117 No. 6, June, pp. 80-81.

6. Payne, Stephen R., 1994, "A CAD-Interactive Software Package for the Synthesis of Planar Disk Cams", *Master of Science Thesis*, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, February.

7. Cleghorn, W.L. and R.P. Podhorodeski, 1988, "Disk Cam Design Using a Microcomputer," International Journal of Mechanical Engineering Education, Vol. 16 No. 4, pp. 235-250.

8. Chakraborty, J. and S.G. Dhande, 1977, *Kinematics and Geometry of Planar and Spatial Cam Mechanisms*, J. Wiley & Sons, New York.

9. Prata, S., 1995, *C++ Primer Plus*, Second Edition, Waite Group Press.

10. Johnsonbaugh, R. and Martin Kalin, 1995, *Object-Oriented Programming in C++*, Prentice-Hall, Inc.

11. Chin-Purcell, K. and D. Riley, 1989, "An Interactive, Object Oriented Approach to Mechanism Synthesis," Computers in Engineering, Proceedings of the 1989 ASME International Computers in Engineering Conference and Exposition, July 30 - August 3, Vol. 1 pp. 225-232.

12. Deitz, D., 1995, "Programming From a Clean Slate," Mechanical Engineering, Vol. 117 No. 4, April, pp. 84-86.

13. Zdonik, Stanley B. and David Maier, 1990, *Readings in Object-Oriented Database Systems*, Morgan Kaufmann Publishers, Inc., pp. 84-91.

14. Mabie, H.H. and Charles F. Reinholtz, 1987, *Mechanisms and Dynamics of Machinery*, Fourth Edition, John Wiley & Sons, Inc.

15. Liang, Z. and C. Jack Quinn, 1991, "Accurate Design of a Cam Profile on the CAD System," Journal of Manufacturing Systems, Vol. 10 No. 6, pp. 501-508.

16. Simms, A., 1996, "Visualizing the future," Engineer's Forum -- Virginia Tech's student Engineering Magazine, Vol. 15 No. 2, pp.8-9.

APPENDIX

Appendix A: Installing & Running The Program

This appendix discusses all necessary files needed to run the Wrapping Cam Synthesis version 1.1 package. Some Borland C++ DLL files are not included in this documentation, however they are very important to run this software package. These Borland C++ DLL files are **BIDS40.DLL**, **OWL200.DLL**, and **BC40RTL.DLL**. The executable code to run this program is **wrpcam11.exe**. This version of the software package is a Windows application program, therefore it must be runned under the Microsoft Window 3.x.

To install this program under the Microsoft Windows, the user should follow these procedures:

1. Open File Manager from Windows, the select the drive the contains the program diskette.
2. Copy wrpcam11.exe, BIDS40.DLL, OWL200.DLL, BC40RTL.DLL, and other existing data files into the HDD.
3. After copying all necessary files, close File Manager. Select File|New command from the menu. If you want to make a new group for this program then select the group item. If you want to add a new program item into the existing group then select the program item.

Appendix B: Program Source Codes

The program source codes are grouped according to what roles they play in supporting this program. The first group is the source codes for the graphical user interface. The second group is the codes for the wrapping cam classes. And the third group is functions used to support the synthesis and analysis modules.

B.1. The Graphical User Interface (GUI) Classes

Class Name:	Header File:	Resource File:	About The Class:
AnalysisGGRRDialog	anlggrrd.h	anlggrrd.cpp	Dialog box for GGRR analysis
GGRRNDialog	ggrrndlg.h	ggrrndlg.cpp	Dialog box for negative GGRR synthesis
GGRRPDialog	ggrrpdlg.h	ggrrpdlg.cpp	Dialog box for positive GGRR synthesis
myMDIChild	mymdichl.h	mymdichl.cpp	MDI child windows
myMDIClient	mymdicln.h	mymdicln.cpp	MDI client window
TGraphView	tgraphvw.h	tgraphvw.cpp	Plot view window
myWPCamApp	wpcamapp.h	wpcamapp.cpp	Windows application
NewInputFileDialog	nwnptfdg.h	nwnptfdg.cpp	Dialog box for new input file
TCam	tcam.h	tcam.cpp	Cam object
T2Curve	t2curve.h	t2curve.cpp	Curve object

```
#if !defined(__anlggrrd_h)          // Sentry, use file only if it's not already included.
#define __anlggrrd_h

/*  Project wpcamgui
    Virginia Tech.
    Copyright © 1993. All Rights Reserved.

    SUBSYSTEM:    wpcamgui.exe Application
    FILE:         anlggrrd.h
    AUTHOR:       Kurnia Dias Iskandar


    OVERVIEW
    ========
    Class definition for AnalysisGGRRDialog (TDialog).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\dialog.h>
#include <owl\edit.h>
#include <owl\validate.h>

#include "myguiapp.rh"           // Definition of all resources.

// Declare a transfer buffer structure for this dialog box.
struct TransAnalysisGGRR{
        TransAnalysisGGRR();

    char InputFile[13];
    char OutputFile[13];
    char CurveFit[7];
    char DataPoints[7];
};

//{{TDialog = AnalysisGGRRDialog}}
class AnalysisGGRRDialog : public TDialog {
public:
    AnalysisGGRRDialog (TWindow* parent, TransAnalysisGGRR& transfer);
    virtual ~AnalysisGGRRDialog ();

//{{AnalysisGGRRDialogVIRTUAL_BEGIN}}
public:
   virtual void SetupWindow ();
//{{AnalysisGGRRDialogVIRTUAL_END}}
};   //{{AnalysisGGRRDialog}}


#endif                              // __anlggrrd_h sentry.
```

```
/*  Project wpcamgui
    Virginia Tech.
    Copyright © 1993. All Rights Reserved.

    SUBSYSTEM:    wpcamgui.exe Application
    FILE:         anlggrrd.cpp
    AUTHOR:       Kurnia Dias Iskandar


    OVERVIEW
    ========
    Source file for implementation of AnalysisGGRRDialog (TDialog).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include "anlggrrd.h"

// Constructor for transfer buffer structure.
TransAnalysisGGRR::TransAnalysisGGRR(){
        InputFile[0] = '\0';
        OutputFile[0] = '\0';
        strcpy(CurveFit, "5");
        strcpy(DataPoints, "25");
}

//{{AnalysisGGRRDialog Implementation}}
AnalysisGGRRDialog::AnalysisGGRRDialog (TWindow* parent, TransAnalysisGGRR&
transfer):
        TDialog(parent, IDD_ANALYSISGGRR)
{
        new TEdit(this, IDC_INPUTFILE, sizeof(transfer.InputFile));
        new TEdit(this, IDC_OUTPUTFILE, sizeof(transfer.OutputFile));
        new TEdit(this, IDC_CURVEFIT, sizeof(transfer.CurveFit))
                ->SetValidator(new TRangeValidator(1, 10));
        new TEdit(this, IDC_DATAPOINTS, sizeof(transfer.DataPoints))
                ->SetValidator(new TFilterValidator("0-9"));

        SetTransferBuffer(&transfer);        // Sending transfer buffer to its dialog box.
        // INSERT>> Your constructor code here.
}
```

```
AnalysisGGRRDialog::~AnalysisGGRRDialog ()
{
        Destroy();

        // INSERT>> Your destructor code here.
}


void AnalysisGGRRDialog::SetupWindow ()
{
  TDialog::SetupWindow();

  // INSERT>> Your code here.

}
```

```
#if !defined(__ggrrndlg_h)          // Sentry, use file only if it's not already included.
#define __ggrrndlg_h

/*  Project wpcamgui
    Virginia Tech.
    Copyright © 1993. All Rights Reserved.

    SUBSYSTEM:   wpcamgui.exe Application
    FILE:        ggrrndlg.h
    AUTHOR:      Kurnia Dias Iskandar


    OVERVIEW
    ========
    Class definition for GGRRNDialog (TDialog).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\dialog.h>
#include <owl\radiobut.h>
#include <owl\edit.h>
#include <owl\validate.h>

#include "myguiapp.rh"          // Definition of all resources.

//
// Definition of transfer buffer structure for GGRRNDialog box.
// Its member function and member data.
//
struct TransGGRRN {
        TransGGRRN();

        char InputFile[13];
   char OutputFile[13];
   char CamPosition[6];
   char CurveFit[6];
   char CenterDistance[14];
   char Stroke[14];
   char InSprocketTeeth[6];
   char OutSprocketTeeth[6];
   char PinSeparation[14];
```

```
  char InPulleyRadius[14];
  char OutPulleyRadius[14];
  char FollowerThick[14];
  UINT Crossed;
  UINT UnCrossed;
  UINT Sprocket;
  UINT Pulley;
};


//{{TDialog = GGRRNDialog}}
class GGRRNDialog : public TDialog {
public:
   GGRRNDialog (TWindow* parent, TransGGRRN& transfer);
   virtual ~GGRRNDialog ();

//{{GGRRNDialogVIRTUAL_BEGIN}}
public:
   virtual void SetupWindow ();
//{{GGRRNDialogVIRTUAL_END}}
};   //{{GGRRNDialog}}


#endif                                  // __ggrrndlg_h sentry.
```

```
/*  Project wpcamgui
   Virginia Tech.
   Copyright © 1993. All Rights Reserved.

   SUBSYSTEM:    wpcamgui.exe Application
   FILE:         ggrrndlg.cpp
   AUTHOR:       Kurnia Dias Iskandar


   OVERVIEW
   ========
   Source file for implementation of GGRRNDialog (TDialog).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include "ggrrndlg.h"

// Constructor for TransGGRRN transfer buffer.
TransGGRRN::TransGGRRN(){
        InputFile[0] = '\0';
        OutputFile[0] = '\0';
        strcpy(CamPosition, "25");
        strcpy(CurveFit, "5");
        strcpy(CenterDistance, "12");
        strcpy(Stroke, "15");
        InSprocketTeeth[0] = '\0';
        OutSprocketTeeth[0] = '\0';
        PinSeparation[0] = '\0';
        InPulleyRadius[0] = '\0';
        OutPulleyRadius[0] = '\0';
        strcpy(FollowerThick, "0.468");
        Crossed = BF_UNCHECKED;
        UnCrossed = BF_CHECKED;
        Sprocket = BF_CHECKED;
        Pulley = BF_UNCHECKED;
}

//{{GGRRNDialog Implementation}}


GGRRNDialog::GGRRNDialog (TWindow* parent, TransGGRRN& transfer):
        TDialog(parent, IDD_GGRRNDIALOG)
{
        new TEdit(this, IDC_INPUTFILE, sizeof(transfer.InputFile));
        new TEdit(this, IDC_OUTPUTFILE, sizeof(transfer.OutputFile));
        new TEdit(this, IDC_CAMPOSITION, sizeof(transfer.CamPosition))
                ->SetValidator(new TFilterValidator("0-9"));
        new TEdit(this, IDC_CURVEFIT, sizeof(transfer.CurveFit))
                ->SetValidator(new TRangeValidator(1, 10));
        new TEdit(this, IDC_CENTERDISTANCE, sizeof(transfer.CenterDistance))
                ->SetValidator(new TFilterValidator(".0-9"));
        new TEdit(this, IDC_STROKE, sizeof(transfer.Stroke))
                ->SetValidator(new TFilterValidator(".0-9"));
        new TEdit(this, IDC_INSPROCKETTEETH, sizeof(transfer.InSprocketTeeth))
                ->SetValidator(new TFilterValidator("0-9"));
        new TEdit(this, IDC_OUTSPROCKETTEETH,
sizeof(transfer.OutSprocketTeeth))
                ->SetValidator(new TFilterValidator("0-9"));
        new TEdit(this, IDC_PINSEPARATION, sizeof(transfer.PinSeparation))
                ->SetValidator(new TFilterValidator(".0-9"));
        new TEdit(this, IDC_INPULLEYRADIUS, sizeof(transfer.InPulleyRadius))
                ->SetValidator(new TFilterValidator(".0-9"));
        new TEdit(this, IDC_OUTPULLEYRADIUS, sizeof(transfer.OutPulleyRadius))
                ->SetValidator(new TFilterValidator(".0-9"));
        new TEdit(this, IDC_FOLLOWERTHICK, sizeof(transfer.FollowerThick))
                ->SetValidator(new TFilterValidator(".0-9"));
        new TRadioButton(this, IDC_CROSSED);
        new TRadioButton(this, IDC_UNCROSSED);
        new TRadioButton(this, IDC_SPROCKET);
        new TRadioButton(this, IDC_PULLEY);
        // INSERT>> Your constructor code here.

        SetTransferBuffer(&transfer);          //Sending transfer buffer to dialog.
}


GGRRNDialog::~GGRRNDialog ()
{
        Destroy();

        // INSERT>> Your destructor code here.

}
```

```
void GGRRNDialog::SetupWindow ()
{
   TDialog::SetupWindow();

   // INSERT>> Your code here.

}
```

```
#if !defined(__ggrrpdlg_h)           // Sentry, use file only if it's not already included.
#define __ggrrpdlg_h

/*  Project wpcamgui
    Virginia Tech.
    Copyright © 1993. All Rights Reserved.

    SUBSYSTEM:    wpcamgui.exe Application
    FILE:         ggrrpdlg.h
    AUTHOR:       Kurnia Dias Iskandar


    OVERVIEW
    ========
    Class definition for GGRRPDialog (TDialog).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\dialog.h>
#include <owl\radiobut.h>
#include <owl\edit.h>
#include <owl\validate.h>

#include "myguiapp.rh"            // Definition of all resources.

//
// transfer buffer structure for GGRRPDialog has the following
// members and constructor
//
struct TransGGRRP {
        TransGGRRP();       //constructor

   char InputFile[13];
   char OutputFile[13];
   char CamPosition[6];
   char CurveFit[6];
   char CenterDistance[14];
   char Stroke[14];
   char SprocketTeeth[6];
   char PinSeparation[14];
   char PulleyRadius[14];
```

```
  char FollowerThick[14];
  UINT Crossed;
  UINT UnCrossed;
  UINT Sprocket;
  UINT Pulley;
};

//{{TDialog = GGRRPDialog}}
class GGRRPDialog : public TDialog {
public:
        GGRRPDialog (TWindow* parent, TransGGRRP& transfer);
        virtual ~GGRRPDialog ();


//{{GGRRPDialogVIRTUAL_BEGIN}}
public:
        virtual void SetupWindow ();
//{{GGRRPDialogVIRTUAL_END}}
};   //{{GGRRPDialog}}


#endif                              // __ggrrpdlg_h sentry.
```

```
/*  Project wpcamgui
    Virginia Tech.
    Copyright © 1993. All Rights Reserved.

    SUBSYSTEM:    wpcamgui.exe Application
    FILE:         ggrrpdlg.cpp
    AUTHOR:       Kurnia Dias Iskandar


    OVERVIEW
    ========
    Source file for implementation of GGRRPDialog (TDialog).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include "ggrrpdlg.h"

// Constructor for transfer buffer.
TransGGRRP::TransGGRRP() {
        InputFile[0] = '\0';
        OutputFile[0] = '\0';
        strcpy(CamPosition, "25");
        strcpy(CurveFit, "6");
        strcpy(CenterDistance, "17.75");
        strcpy(Stroke, "16");
        SprocketTeeth[0] = '\0';
        PinSeparation[0] = '\0';
        PulleyRadius[0] = '\0';
        strcpy(FollowerThick, "0.4685");
        Crossed = BF_UNCHECKED;
        UnCrossed = BF_CHECKED;
        Sprocket = BF_CHECKED;
        Pulley = BF_UNCHECKED;
}


//{{GGRRPDialog Implementation}}
GGRRPDialog::GGRRPDialog (TWindow* parent, TransGGRRP& transfer):
         TDialog(parent, IDD_GGRRPDIALOG)
{
         new TEdit(this, IDC_INPUTFILE, sizeof(transfer.InputFile));
        new TEdit(this, IDC_OUTPUTFILE, sizeof(transfer.OutputFile));
        new TEdit(this, IDC_CAMPOSITION, sizeof(transfer.CamPosition))
                ->SetValidator(new TFilterValidator("0-9"));
        new TEdit(this, IDC_CURVEFIT, sizeof(transfer.CurveFit))
                ->SetValidator(new TRangeValidator(1, 10));
        new TEdit(this, IDC_CENTERDISTANCE, sizeof(transfer.CenterDistance))
                ->SetValidator(new TFilterValidator(".0-9"));
        new TEdit(this, IDC_STROKE, sizeof(transfer.Stroke))
                ->SetValidator(new TFilterValidator(".0-9"));
        new TEdit(this, IDC_SPROCKETTEETH, sizeof(transfer.SprocketTeeth))
                ->SetValidator(new TFilterValidator("0-9"));
        new TEdit(this, IDC_PINSEPARATION, sizeof(transfer.PinSeparation))
                ->SetValidator(new TFilterValidator(".0-9"));
        new TEdit(this, IDC_PULLEYRADIUS, sizeof(transfer.PulleyRadius))
                ->SetValidator(new TFilterValidator(".0-9"));
        new TEdit(this, IDC_FOLLOWERTHICK, sizeof(transfer.FollowerThick))
                ->SetValidator(new TFilterValidator(".0-9"));
        new TRadioButton(this, IDC_CROSSED);
        new TRadioButton(this, IDC_UNCROSSED);
        new TRadioButton(this, IDC_SPROCKET);
        new TRadioButton(this, IDC_PULLEY);

        // INSERT>> Your constructor code here.

        SetTransferBuffer(&transfer);        // Sending transfer buffer to controls.
}


GGRRPDialog::~GGRRPDialog ()
{
        Destroy();

        // INSERT>> Your destructor code here.

}


void GGRRPDialog::SetupWindow ()
{
        TDialog::SetupWindow();

        // INSERT>> Your code here.
}
```

```
#if !defined(__mymdichl_h)          // Sentry, use file only if it's not already included.
#define __mymdichl_h

/*  Project mywpcam1
    VPI & SU
    Copyright © 1993. All Rights Reserved.

    SUBSYSTEM:    mywpcam1.exe Application
    FILE:         mymdichl.h
    AUTHOR:       Kurnia D. Iskandar


    OVERVIEW
    ========
    Class definition for myMDIChild (TMDIChild).
*/


#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\editfile.h>
#include <owl\listbox.h>

#include "wpcamapp.rh"          // Definition of all resources.

#include <iomanip.h>          // Header file for i/o manipulation.

#include <ggrrp.h>                                  // Definition for GGRR Positive
synthesis.
#include <ggrrn.h>                                  // Definition for GGRR Negative
synthesis.
#include <ggrranls.h>                       // Definition for GGRR Kinematic analysis.
#include <point2d.h>                                // Definition for 2D cartesian
coordinate.

#include <ggrrpdlg.h>                 // Definition for GGRRPositive dialog.
#include <ggrrndlg.h>                       // Definition for GGRRNegative dialog.
#include <anlggrrd.h>                       // Definition for AnalysisGGRR dialog.

//{{TMDIChild = myMDIChild}}
class myMDIChild : public TMDIChild {
private:
        GGRRP syn1;
        GGRRN syn2;
   GGRRAnls analysis1;
        point2d* a;
   point2d* b;
   point2d* data1;
   point2d* data2;
   point2d* data3;
   int num, n, d, N1, N2, index;
   double* xval, *yval, *S, dist, p, l, t, r1, r2;
        Sprocket gear;
        char type[11];

protected:
        TransGGRRP transggrrp;
        TransGGRRN transggrrn;
        TransAnalysisGGRR transanlysggrr;

public:
   myMDIChild (TMDIClient &parent, const char far *title=0, TWindow *clientWnd=0,
BOOL shrinkToClient = FALSE, TModule* module = 0);
   virtual ~myMDIChild ();

//{{myMDIChildVIRTUAL_BEGIN}}
public:
        virtual void Paint (TDC& dc, BOOL erase, TRect& rect);
        virtual void CleanupWindow();
//{{myMDIChildVIRTUAL_END}}
//{{myMDIChildRSP_TBL_BEGIN}}
protected:
   void EvGetMinMaxInfo (MINMAXINFO far& minmaxinfo);
   void CmAnalysisGGRR ();
   void CmGGRRNegative ();
   void CmGGRRPositive ();
//{{myMDIChildRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(myMDIChild);
};   //{{myMDIChild}}


#endif                              // __mymdichl_h sentry.
```

```
/*  Project mywpcam1
    VPI & SU
    Copyright © 1993. All Rights Reserved.

    SUBSYSTEM:    mywpcam1.exe Application
    FILE:         mymdichl.cpp
    AUTHOR:       Kurnia D. Iskandar


    OVERVIEW
    ========
    Source file for implementation of myMDIChild (TMDIChild).
*/


#include <owl\owlpch.h>
#pragma hdrstop

#include "wpcamapp.h"
#include "mymdichl.h"

#include <stdio.h>

//{{myMDIChild Implementation}}


//
// Build a response table for all messages/commands handled
// by myMDIChild derived from TMDIChild.
//
DEFINE_RESPONSE_TABLE1(myMDIChild, TMDIChild)
//{{myMDIChildRSP_TBL_BEGIN}}
    EV_WM_GETMINMAXINFO,
    EV_COMMAND(CM_ANALYSISGGRR, CmAnalysisGGRR),
    EV_COMMAND(CM_GGRRNEGATIVE, CmGGRRNegative),
        EV_COMMAND(CM_GGRRPOSITIVE, CmGGRRPositive),
//{{myMDIChildRSP_TBL_END}}
END_RESPONSE_TABLE;


//////////////////////////////////////////////////////////
// myMDIChild
// ==========
// Construction/Destruction handling.
myMDIChild::myMDIChild (TMDIClient &parent, const char far *title, TWindow
*clientWnd, BOOL shrinkToClient, TModule *module)
        : TMDIChild (parent, title, clientWnd, shrinkToClient, module)
{
        // INSERT>> Your constructor code here.
        xval = NULL;
        yval = NULL;
        S = NULL;
        a = NULL;
        b = NULL;
        data1 = NULL;
        data2 = NULL;
        data3 = NULL;
        type[0] = '\0';
}


myMDIChild::~myMDIChild ()
{
        Destroy();

        // INSERT>> Your destructor code here.
        delete[] data1;
        delete[] data2;
        delete[] data3;
        delete[] S;
        delete[] xval;
        delete[] yval;
}


//
// Paint routine for Window, Printer, and PrintPreview for an TEdit client.
//
void myMDIChild::Paint (TDC& dc, BOOL, TRect& rect)
{
        myWPCamApp *theApp = TYPESAFE_DOWNCAST(GetApplication(),
myWPCamApp);
        if (theApp) {
                // Only paint if we're printing and we have something to paint, otherwise
do nothing.
                if (theApp->Printing && theApp->Printer && !rect.IsEmpty()) {
                    // Use pageSize to get the size of the window to render into.
For a Window it's the client area,
                    // for a printer it's the printer DC dimensions and for print
preview it's the layout window.
                    TSize   pageSize(rect.right - rect.left, rect.bottom -
rect.top);

            }
        }
}


void myMDIChild::EvGetMinMaxInfo (MINMAXINFO far& minmaxinfo)
{
        myWPCamApp *theApp = TYPESAFE_DOWNCAST(GetApplication(),
myWPCamApp);
        if (theApp) {
            if (theApp->Printing) {
                    minmaxinfo.ptMaxSize = TPoint(32000, 32000);
                    minmaxinfo.ptMaxTrackSize = TPoint(32000, 32000);
                    return;
            }
        }
        TMDIChild::EvGetMinMaxInfo(minmaxinfo);
}

void myMDIChild::CmAnalysisGGRR ()
{
        // INSERT>> Your code here.
        if (AnalysisGGRRDialog(this, transanlysggrr).Execute() == IDOK){
            // Create input and output file stream objects.
            ifstream fin(transanlysggrr.InputFile, ios::in|ios::nocreate);
            ofstream fout(transanlysggrr.OutputFile, ios::out|ios::app|ios::trunc);

            if (!fin.good() || !fout.good()){
                MessageBox("Cannot Open/Create File", "File Error");
            }else{

                // Read in the data from input file.
                fin>>type;      // type of cam file
                fin>>n;         // number of points of cam profile
                data1 = new point2d[n];
                data2 = new point2d[n];
```

```
                    for (int i=0; i<n; i++)
                            fin>>data1[i].x>>data1[i].y;  // read in cartesian form of
cam profile
                    for (i=0; i<n; i++)
                            fin>>data2[i].x>>data2[i].y;  // read in polar form of cam
profile
                    fin>>index;     // read in the follower index (crossed or uncrossed)

                    if (strcmpi(type,"{GGRRP|S}") == 0)
                    {
                            fin>>N1>>p;  // read in the sprocket teeth and sprocket
pitch
                            r1 = gear.Radius(p,N1);
                    }else if (strcmpi(type,"{GGRRN|S}") == 0)
                    {
                            fin>>N1>>N2>>p;  // read in the output and input sprocket
teeth
                                                                        // and sprocket
pitch
                            r1 = gear.Radius(p,N1);
                    }else if (strcmpi(type,"{GGRRP|P}") == 0)
                    {
                            fin>>r1;          // read in the pulley radius
                    }else if (strcmpi(type,"{GGRRN|P}") == 0)
                    {
                            fin>>r1>>r2;  // read in the output and input pulley radius
                    }
                    fin>>t>>dist>>l;
                    fin.close();

                    // Convert type char to numeric type (int or double).
                    d = atoi(transanlysggrr.CurveFit);
                    num = atoi(transanlysggrr.DataPoints);

                    // Perform kinematic analysis by calling the GGRRAnls object.
                    analysis1.wrapping_cam_data(data2,n,index,r1,t,dist);
                    data3 = analysis1.kinematic_analysis(num, d);

                    // Create an output file stream object.
                    fout.setf(ios::left, ios::adjustfield);
                    if (strcmpi(type,"{GGRRP|S}") == 0)
                    {
```

```
                                fout<<"{A_GGRRP|S}"<<"\n";
                        }else if (strcmpi(type,"{GGRRP|P}") == 0)
                        {
                                fout<<"{A_GGRRP|P}"<<"\n";
                        }else if (strcmpi(type,"{GGRRN|S}") == 0)
                        {
                                fout<<"{A_GGRRN|S}"<<"\n";
                        }else if (strcmpi(type,"{GGRRN|P}") == 0)
                        {
                                fout<<"{A_GGRRN|P}"<<"\n";
                        }

                        // write out the number of data points calculated
                        fout<<analysis1.number_of_data()<<"\n";

                        // write out the analysis data points
                        for (i=0; i<num; i++)

        fout<<setprecision(6)<<setw(15)<<data3[i].x<<setw(15)<<data3[i].y<<"\n";

                        fout<<d;        // write out the polynomial curve fit used
                        fout.close();

                }

        }
}


void myMDIChild::CmGGRRNegative ()
{
        // INSERT>> Your code here.
        if (GGRRNDialog(this, transggrrn).Execute() == IDOK){
                // Create input and output file stream objects.
                ifstream fin(transggrrn.InputFile, ios::in|ios::nocreate);
                ofstream fout(transggrrn.OutputFile, ios::out|ios::app|ios::trunc);

                if (!fin.good() || !fout.good()){
                        MessageBox("Cannot Open/Create File", "File Error");
                }else{

                        // Read in from the input file.
                        fin>>type;
                        fin>>n;
                        xval = new double[n];
                        yval = new double[n];
                        for (int i=0; i<n; i++){
                                fin>>xval[i];
                                fin>>yval[i];
                        }
                        fin.close();

                        if (transggrrn.Crossed == BF_CHECKED &&
transggrrn.UnCrossed == BF_UNCHECKED){
                                index = -1;
                        }else if (transggrrn.Crossed == BF_UNCHECKED &&
transggrrn.UnCrossed == BF_CHECKED){
                                index = 1;
                        }

                        // Convert type char to double or int and assigned to corresponding
variables.
                        d = atoi(transggrrn.CurveFit);
                        l = atof(transggrrn.Stroke);
                        num = atoi(transggrrn.CamPosition);
                        dist = atof(transggrrn.CenterDistance);
                        t = atof(transggrrn.FollowerThick);

                        if (transggrrn.Sprocket == BF_CHECKED && transggrrn.Pulley
== BF_UNCHECKED){
                                N1 = atoi(transggrrn.OutSprocketTeeth);
                                N2 = atoi(transggrrn.InSprocketTeeth);
                                p = atof(transggrrn.PinSeparation);
                                syn2.sprocket_radius(N1,N2,p);
                                r1 = gear.Radius(p, N1);
                        }else if (transggrrn.Sprocket == BF_UNCHECKED &&
transggrrp.Pulley == BF_CHECKED){
                                r1 = atof(transggrrn.OutPulleyRadius);
                                r2 = atof(transggrrn.InPulleyRadius);
                                syn2.pulley_radius(r1, r2);
                        }

                        S = new double[n];
                        for (int j=0; j<n; j++) S[j] = r1*xval[j];        // Calculate chain
displacement.
```

```
                    // Perform the actual synthesis by calling GGRRN object.
                    syn2.poly_coef(n,d,S,yval);
                    syn2.cam_motion(num);
                    syn2.torque_moment_arm(l);
                    syn2.follower_length_thick(index,dist,t);
                    a = syn2.point_vector();
                    b = syn2.polar_coor(num,a);

                    // Define a file output stream.
                    fout.setf(ios::left, ios::adjustfield);
                    if (transggrrn.Sprocket == BF_CHECKED)
                    {
                            fout<<"{GGRRN|S}"<<"\n";
                    }else if (transggrrn.Pulley == BF_CHECKED)
                    {
                            fout<<"{GGRRN|P}"<<"\n";
                    }
                    fout<<(num+1)<<"\n";          // write out the number points of cam
profile
                    fout<<"\n";
                    // write out the cam profile in cartesian form
                    for (j=0; j<num+1; j++)

    fout<<setprecision(6)<<setw(15)<<a[j].x<<setw(15)<<a[j].y<<"\n";
                    fout<<"\n";
                    // write out the cam profile in polar form
                    for (j=0; j<num+1; j++)

    fout<<setprecision(6)<<setw(15)<<b[j].x<<setw(15)<<b[j].y<<"\n";

                    fout<<"\n"<<index<<"\n";      // write out the follower index

    //(crossed or uncrossed)
                    if (transggrrn.Sprocket == BF_CHECKED)
                    {
                            // write out the output sprocket teeth and
                            // the input sprocket (attached to cam) teeth
                            fout<<N1<<"\n"<<N2<<"\n"<<p<<"\n";
                    }else if (transggrrn.Pulley == BF_CHECKED)
                    {
                            // write out the output pulley radius and
                            // and the input pulley (attached to cam) radius
                            fout<<r1<<"\n"<<r2<<"\n";
```

```
                        }
                        fout<<t<<"\n"<<dist<<"\n"<<l;          // write out the follower
thickness,

        // center distance, and weight's stroke
                        fout<<"\n"<<d;          // write out the polynomial curve fit used
                        fout.close();
                }
        }
}


void myMDIChild::CmGGRRPositive ()
{
        // INSERT>> Your code here.
        if (GGRRPDialog(this, transggrrp).Execute() == IDOK){
                // Create input and output file stream objects.
                ifstream fin(transggrrp.InputFile, ios::in|ios::nocreate);
                ofstream fout(transggrrp.OutputFile, ios::out|ios::app|ios::trunc);

                if (!fin.good() || !fout.good()){
                        MessageBox("Cannot Open/Create File", "File Error !");
                }else{

                        //Read in the strength curve
                        fin>>type;
                        fin>>n; //Read in number of data points.
                        xval = new double[n];
                        yval = new double[n];
                        for (int i=0; i<n; i++){
                                fin>>xval[i];
                                fin>>yval[i];
                        }
                        fin.close();

                        if (transggrrp.Crossed == BF_CHECKED &&
transggrrp.UnCrossed == BF_UNCHECKED){
                                index = -1;
                        }else if (transggrrp.Crossed == BF_UNCHECKED &&
transggrrp.UnCrossed == BF_CHECKED){
                                index = 1;
                        }
```

```
                    //Convert type char to double or int and assigned to corresponding
variables.
                    d = atoi(transggrrp.CurveFit);
                    l = atof(transggrrp.Stroke);
                    num = atoi(transggrrp.CamPosition);
                    dist = atof(transggrrp.CenterDistance);
                    t = atof(transggrrp.FollowerThick);

                    if (transggrrp.Sprocket == BF_CHECKED && transggrrp.Pulley
== BF_UNCHECKED){
                            N1 = atoi(transggrrp.SprocketTeeth);
                            p = atof(transggrrp.PinSeparation);
                            syn1.sprocket_radius(N1,p);
                    }else if (transggrrp.Sprocket == BF_UNCHECKED &&
transggrrp.Pulley == BF_CHECKED){
                            r1 = atof(transggrrp.PulleyRadius);
                            syn1.pulley_radius(r1);
                    }

                    // Perform the actual synthesis by calling GGRRP object.
                    syn1.poly_coef(n,d,xval,yval);
                    syn1.cam_motion(num);
                    syn1.torque_moment_arm(l);
                    syn1.follower_length_thick(index,dist,t);
                    a = syn1.point_vector();
                    b = syn1.polar_coor(num,a);

                    // Define a file output stream.
                    fout.setf(ios::left, ios::adjustfield);
                    if (transggrrp.Sprocket == BF_CHECKED)
                    {
                            fout<<"{GGRRP|S}"<<"\n";
                    }else if (transggrrp.Pulley == BF_CHECKED)
                    {
                            fout<<"{GGRRP|P}"<<"\n";
                    }
                    fout<<(num+1)<<"\n";          // write out number of points of cam
profile
                    fout<<"\n";
                    // write out the cam profile in cartesian form
                    for (int j=0; j<num+1; j++)

    fout<<setprecision(6)<<setw(15)<<a[j].x<<setw(15)<<a[j].y<<"\n";
```

```
                        fout<<"\n";
                        // write out the cam profile in polar form
                        for (j=0; j<num+1; j++)

        fout<<setprecision(6)<<setw(15)<<b[j].x<<setw(15)<<b[j].y<<"\n";
                        fout<<"\n"<<index<<"\n";     // write out follower index

        //(crossed or uncrossed)
                        if (transggrrp.Sprocket == BF_CHECKED)
                        {
                                fout<<N1<<"\n"<<p<<"\n";  // write out sprocket teeth
and sprocket pitch
                        }else if (transggrrp.Pulley == BF_CHECKED)
                        {
                                fout<<r1<<"\n";         // write out pulley's radius
                        }
                        fout<<t<<"\n"<<dist<<"\n"<<l; // write out follower thickness,

        // center distance and weight's stroke.
                        fout<<"\n"<<d;          // write out the polynomial curve fit used
                        fout.close();
                }
        }
}

void myMDIChild::CleanupWindow ()
{
        TMDIChild::CleanupWindow();

        // INSERT>> Your code here.
        data1 = NULL;
        data2 = NULL;
        data3 = NULL;
        S = NULL;
        xval = NULL;
        yval = NULL;
}
```

```
#if !defined(__mymdicln_h)          // Sentry, use file only if it's not already included.
#define __mymdicln_h

/*  Project mywpcam1
    VPI & SU
    Copyright © 1993. All Rights Reserved.

    SUBSYSTEM:    mywpcam1.exe Application
    FILE:         mymdicln.h
    AUTHOR:       Kurnia D. Iskandar


    OVERVIEW
    ========
    Class definition for myMDIClient (TMDIClient).
*/


#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\opensave.h>
#include <nwnptfdg.h>

#include "wpcamapp.rh"          // Definition of all resources.

//{{TMDIClient = myMDIClient}}
class myMDIClient : public TMDIClient {
protected:
        TransBuffNewFile transbuffnewfile;

public:
        int ChildCount;              // Number of child window created.

        myMDIClient ();
        virtual ~myMDIClient ();

        void OpenFile (const char *fileName = 0);

private:
    void LoadTextFile ();

//{{myMDIClientVIRTUAL_BEGIN}}
protected:
   virtual void SetupWindow ();
//{{myMDIClientVIRTUAL_END}}

//{{myMDIClientRSP_TBL_BEGIN}}
protected:
         void CmNewInputFile ();
   void CmFilePrint ();
         void CmFilePrintSetup ();
   void CmFilePrintPreview ();
   void CmPrintEnable (TCommandEnabler &tce);
//{{myMDIClientRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(myMDIClient);
};   //{{myMDIClient}}


#endif                                  // __mymdicln_h sentry.
```

```
/*  Project mywpcam1
   VPI & SU
   Copyright © 1993. All Rights Reserved.

   SUBSYSTEM:    mywpcam1.exe Application
   FILE:         mymdicln.cpp
   AUTHOR:       Kurnia D. Iskandar


   OVERVIEW
   ========
   Source file for implementation of myMDIClient (TMDIClient).
*/


#include <owl\owlpch.h>
#pragma hdrstop


#include "wpcamapp.h"
#include "mymdicln.h"
#include "mymdichl.h"
#include "apxprint.h"
#include "apxprev.h"

//{{myMDIClient Implementation}}


//
// Build a response table for all messages/commands handled
// by myMDIClient derived from TMDIClient.
//
DEFINE_RESPONSE_TABLE1(myMDIClient, TMDIClient)
//{{myMDIClientRSP_TBL_BEGIN}}
        EV_COMMAND(CM_NEWINPUTFILE, CmNewInputFile),
        EV_COMMAND(CM_FILEPRINT, CmFilePrint),
   EV_COMMAND(CM_FILEPRINTERSETUP, CmFilePrintSetup),
   EV_COMMAND(CM_FILEPRINTPREVIEW, CmFilePrintPreview),
   EV_COMMAND_ENABLE(CM_FILEPRINT, CmPrintEnable),
   EV_COMMAND_ENABLE(CM_FILEPRINTERSETUP, CmPrintEnable),
   EV_COMMAND_ENABLE(CM_FILEPRINTPREVIEW, CmPrintEnable),
//{{myMDIClientRSP_TBL_END}}
END_RESPONSE_TABLE;
```

```
//////////////////////////////////////////////////////
// myMDIClient
// ==========
// Construction/Destruction handling.
 myMDIClient::myMDIClient ()
 : TMDIClient ()
{
   // Change the window's background color
   SetBkgndColor(RGB(0xff, 0xff, 0xff));

   ChildCount = 0;

   // INSERT>> Your constructor code here.

}


 myMDIClient::~myMDIClient ()
{
   Destroy();

   // INSERT>> Your destructor code here.

}


//////////////////////////////////////////////////////
// myMDIClient
// ==========
// MDIClient site initialization.
void myMDIClient::SetupWindow ()
{
   // Default SetUpWindow processing.
   TMDIClient::SetupWindow ();

}


//////////////////////////////////////////////////////
// myMDIClient
// ==========
// Menu File Print command
void myMDIClient::CmFilePrint ()
{
   //
   // Create Printer object if not already created.
   //
   myWPCamApp *theApp = TYPESAFE_DOWNCAST(GetApplication(),
myWPCamApp);
   if (theApp) {
      if (!theApp->Printer)
         theApp->Printer = new TPrinter;

      //
      // Create Printout window and set characteristics.
      //
      APXPrintOut printout(theApp->Printer, Title, GetActiveMDIChild(), TRUE);

      theApp->Printing = TRUE;

      //
      // Bring up the Print dialog and print the document.
      //
      theApp->Printer->Print(GetActiveMDIChild()->GetClientWindow(), printout,
TRUE);

      theApp->Printing = FALSE;
   }
}


//////////////////////////////////////////////////////////
// myMDIClient
// ==========
// Menu File Print Setup command
void myMDIClient::CmFilePrintSetup ()
{
   myWPCamApp *theApp = TYPESAFE_DOWNCAST(GetApplication(),
myWPCamApp);
   if (theApp) {
      if (!theApp->Printer)
         theApp->Printer = new TPrinter;

      //
      // Bring up the Print Setup dialog.
      //
      theApp->Printer->Setup(this);
   }
}


/////////////////////////////////////////////////////////
// myMDIClient
// ==========
// Menu File Print Preview command
void myMDIClient::CmFilePrintPreview ()
{
   myWPCamApp *theApp = TYPESAFE_DOWNCAST(GetApplication(),
myWPCamApp);
   if (theApp) {
      if (!theApp->Printer)
         theApp->Printer = new TPrinter;

      theApp->Printing = TRUE;

      PreviewWindow *prevW = new PreviewWindow(Parent, theApp->Printer,
GetActiveMDIChild(), "Print Preview", new TLayoutWindow(0));
      prevW->Create();

      GetApplication()->BeginModal(GetApplication()->MainWindow);

      // We must destroy the preview window explicitly.  Otherwise, the window will not
be destroyed until
      // it's parent the MainWindow is destroyed.
      prevW->Destroy();
      delete prevW;

      theApp->Printing = FALSE;
   }
}


/////////////////////////////////////////////////////////
// myMDIClient
// ==========
// Menu enabler used by Print, Print Setup and Print Preview.
void myMDIClient::CmPrintEnable (TCommandEnabler &tce)
```

```
{
        if (GetActiveMDIChild()) {
     myWPCamApp *theApp = TYPESAFE_DOWNCAST(GetApplication(),
myWPCamApp);
     if (theApp) {
        // If we have a Printer already created just test if all is okay.
        // Otherwise create a Printer object and make sure the printer
        // really exists and then delete the Printer object.
        if (!theApp->Printer) {
                                        theApp->Printer = new TPrinter;

           tce.Enable(theApp->Printer->GetSetup().Error == 0);
                              } else
                                        tce.Enable(theApp->Printer->GetSetup().Error ==
0);
                  }
        } else
                  tce.Enable(FALSE);
}

//
// A new input file command
//
void myMDIClient::CmNewInputFile ()
{
        // INSERT >> Your code here.
        int num;

        if (NewInputFileDialog(this, transbuffnewfile).Execute() == IDOK)
        {
                ofstream fout(transbuffnewfile.Filename, ios::out|ios::app|ios::trunc);

                if (!fout.good())
                {
                        MessageBox("Cannot Open/Create File", "File Error");
                }
                else
                {
                        num = atoi(transbuffnewfile.NumPoints);
                        if (transbuffnewfile.CurveFile == BF_CHECKED)
                        {
                                fout<< "{curve}" << "\n";
                                fout<< num << "\n\n";
        for (int i = 1; i <= num; i++)
                fout<< "x["<<i<<"]     "<<"y["<<i<<"]\n";
}else if (transbuffnewfile.PosGGRRSprocket == BF_CHECKED)
{
        fout<< "{GGRRP|S}\n";
        fout<< num << "\n\n";
        for ( int i = 1; i <= num; i++)
                fout<< "x["<<i<<"]     "<<"y["<<i<<"]\n";
        fout <<"\n";
        for ( i = 1; i <= num; i++)
                fout<< "r["<<i<<"]     "<<"theta["<<i<<"]\n";
        fout <<"\n";
        fout << "[follower index (-1 or 1)]\n";
        fout << "[sprocket teeth]\n";
        fout << "[sprocket pitch]\n";
        fout << "[follower thickness]\n";
        fout << "[center distance]\n";
        fout << "[weight stroke]\n";
}else if (transbuffnewfile.NegGGRRSprocket == BF_CHECKED)
{
        fout<< "{GGRRN|S}\n";
        fout<< num << "\n\n";
        for ( int i = 1; i <= num; i++)
                fout<< "x["<<i<<"]     "<<"y["<<i<<"]\n";
        fout <<"\n";
        for ( i = 1; i <= num; i++)
                fout<< "r["<<i<<"]     "<<"theta["<<i<<"]\n";
        fout << "\n";
        fout << "[follower index (-1 or 1)]\n";
        fout << "[output sprocket teeth]\n";
        fout << "[input sprocket teeth]\n";
        fout << "[sprocket pitch]\n";
        fout << "[follower thickness]\n";
        fout << "[center distance]\n";
        fout << "[weight stroke]\n";
}else if (transbuffnewfile.PosGGRRPulley == BF_CHECKED)
{
        fout<< "{GGRRP|P}\n";
        fout<< num << "\n\n";
        for ( int i = 1; i <= num; i++)
                fout<< "x["<<i<<"]     "<<"y["<<i<<"]\n";
        fout <<"\n";
        for ( i = 1; i <= num; i++)
```

```
                        fout<< "r["<<i<<"]     "<<"theta["<<i<<"]\n";
                fout << "\n";
                fout << "[follower index (-1 or 1)]\n";
                fout << "[pulley radius]\n";
                fout << "[follower thickness]\n";
                fout << "[center distance]\n";
                fout << "[weight stroke]\n";
        }else if (transbuffnewfile.NegGGRRPulley == BF_CHECKED)
        {
                fout<< "{GGRRN|P}\n";
                fout<< num << "\n\n";
                for ( int i = 1; i <= num; i++)
                        fout<< "x["<<i<<"]     "<<"y["<<i<<"]\n";
                fout <<"\n";
                for ( i = 1; i <= num; i++)
                        fout<< "r["<<i<<"]     "<<"theta["<<i<<"]\n";
                fout << "\n";
                fout << "[follower index (-1 or 1)]\n";
                fout << "[output pulley radius]\n";
                fout << "[input pulley radius]\n";
                fout << "[follower thickness]\n";
                fout << "[center distance]\n";
                fout << "[weight stroke]\n";
        }
    }
  }
}
```

```
#if !defined(__tgraphvw_h)          // Sentry, use file only if it's not already included.
#define __tgraphvw_h

/*  Project wrpcam11
        VPI & SU
        Copyright © 1993. All Rights Reserved.

        SUBSYSTEM:    wrpcam11.exe Application
        FILE:         tgraphvw.h
        AUTHOR:       Kurnia D. Iskandar


        OVERVIEW
        ========
        Class definition for TGraphView (TWindowView).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\docview.h>
#include <owl\filedoc.h>

#include "wpcamapp.rh"          // Definition of all resources.

//{{TWindowView = TGraphView}}
class TGraphView : public TWindowView {
private:
        char label[7];
        TPoint* points;
        int xMinPost, yMinPost, xMaxPost, yMaxPost;

protected:
        char type[11];
        char* File;
        int num;
        double *x, *y;
        double xmin, ymin, xmax, ymax;

public:
        TGraphView (TFileDocument& doc, TWindow* parent = 0);
        virtual ~TGraphView ();
//{{TGraphViewVIRTUAL_BEGIN}}
public:
        virtual char far* GetClassName();
        virtual void GetWindowClass(WNDCLASS& wndClass);
        virtual void Paint(TDC& dc, BOOL erase, TRect& rect);
        virtual void SetupWindow ();
        virtual void CleanupWindow ();

//{{TGraphViewVIRTUAL_END}}

//{{TGraphViewRSP_TBL_BEGIN}}
protected:
//{{TGraphViewRSP_TBL_END}}
DECLARE_RESPONSE_TABLE(TGraphView);
};    //{{TGraphView}}


#endif                              // __tgraphvw_h sentry.
```

```
/* Project wrpcam11
        VPI & SU
        Copyright © 1993. All Rights Reserved.

        SUBSYSTEM:    wrpcam11.exe Application
        FILE:         tgraphvw.cpp
        AUTHOR:       Kurnia D. Iskandar


        OVERVIEW
        ========
        Source file for implementation of TGraphView (TWindowView).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include "tgraphvw.h"
#include <fstream.h>
#include <sort.h>
#include <math.h>
#include <tcam.h>
#include <t2curve.h>

//
// Build a response table for all messages/commands handled
// by the application.
//
DEFINE_RESPONSE_TABLE1(TGraphView, TWindowView)
//{{TGraphViewRSP_TBL_BEGIN}}
//{{TGraphViewRSP_TBL_END}}
END_RESPONSE_TABLE;

//{{TGraphView Implementation}}


TGraphView::TGraphView (TFileDocument& doc, TWindow* parent):
        TWindowView(doc, parent)
{

        // INSERT>> Your constructor code here.
        File = new char[strlen(doc.GetTitle())+1];
        strcpy(File,doc.GetTitle());
        x = NULL;
        y = NULL;
        num = 0;
        xmin = 0.0;
        ymin = 0.0;
        xmax = 0.0;
        ymin = 0.0;
        points = NULL;

        type[0] = '\0';
}


TGraphView::~TGraphView ()
{
        // INSERT>> Your destructor code here.
        delete []points;
        delete []x;
        delete []y;
        delete []File;

        Destroy();

}

char far* TGraphView::GetClassName()
{
        // define a specific WNDCLASS
        return "TGraphView";
}

void TGraphView::GetWindowClass(WNDCLASS& wndClass)
{
        TWindowView::GetWindowClass(wndClass);

        wndClass.style = CS_HREDRAW|CS_VREDRAW;

}

void TGraphView::Paint(TDC& dc, BOOL erase, TRect& rect)
{
        TWindowView::Paint(dc,erase,rect);
```

```
	// INSERT>> Your code here
	if (strcmp(File, "Untitled") != 0)
	{
		// draw the data as point or scatter plot
		if (strcmpi(type,"{CURVE}") == 0)
		{
			T2Curve my_curve;
			my_curve.set_curve_xy (x, y, num);
			my_curve.set_label(type,"theta","f(t)");
			my_curve.draw_curve(dc, rect);

		}else
		if (strcmpi(type,"{GGRRP|S}") == 0 || strcmpi(type,"{GGRRP|P}") == 0
			|| strcmpi(type,"{GGRRN|S}") == 0 || strcmpi(type,"{GGRRN|P}")
== 0)
		{
			TCam my_cam;
			TPoint label1, label2;

			my_cam.set_cam_xy(x, y, num);
			my_cam.draw_cam(dc, rect);
			label1 = my_cam.get_start_pts();
			label2 = my_cam.get_end_pts();
			dc.SetTextAlign(TA_TOP|TA_LEFT);
			dc.TextOut(label1.x+1,label1.y+1,"start");
			dc.SetTextAlign(TA_TOP|TA_LEFT);
			dc.TextOut(label2.x+1,label2.y+1,"end");

		}else
		if (strcmpi(type,"{A_GGRRP|S}") == 0 || strcmpi(type,"{A_GGRRP|P}")
== 0)
		{
			T2Curve my_curve;
			my_curve.set_curve_xy(x, y, num);
			my_curve.set_label(type,"theta","h(t)");
			my_curve.draw_curve(dc, rect);

		}else
		if (strcmpi(type,"{A_GGRRN|S}") == 0 || strcmpi(type,"{A_GGRRN|P}")
== 0)
		{
			T2Curve my_curve;
```

```
                        my_curve.set_curve_xy(x, y, num);
                        my_curve.set_label(type,"theta","1/h(t)");
                        my_curve.draw_curve(dc, rect);

                }

        if (FlashWindow(TRUE))     Invalidate();
        }
}

void TGraphView::SetupWindow ()
{
        TWindowView::SetupWindow();

        // INSERT>> Your code here.

        if (strcmp(File, "Untitled") != 0)
        {
                ifstream fin(File, ios::in|ios::nocreate);
                fin >> type;
                fin >> num;  // read in number of data points

                x = new double[num];
                y = new double[num];
                points = new TPoint[num];

                for (int i = 0; i < num; i++){
                        fin >> x[i] >> y[i];
                }
                fin.close();

                if (strcmpi(type,"{A_GGRRN|S}") == 0 || strcmpi(type,"{A_GGRRN|P}")
== 0)
                {
                        for ( i = 0; i < num; i++)
                                y[i] = 1.0/y[i];
                }

                xmin = DMin(x,num);
                ymin = DMin(y,num);
                xmax = DMax(x,num);
                ymax = DMax(y,num);
        }

}

void TGraphView::CleanupWindow ()
{
        TWindowView::CleanupWindow();

        // INSERT>> Your code here.
        points = NULL;
        x = NULL;
        y = NULL;
        File = NULL;
}
```

```
#if !defined(__wpcamapp_h)          // Sentry, use file only if it's not already included.
#define __wpcamapp_h

/*  Project mywpcam1
    VPI & SU
    Copyright © 1993. All Rights Reserved.

    SUBSYSTEM:    mywpcam1.exe Application
    FILE:         wpcamapp.h
    AUTHOR:       Kurnia D. Iskandar


    OVERVIEW
    ========
    Class definition for myWPCamApp (TApplication).
*/


#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\statusba.h>
#include <owl\controlb.h>
#include <owl\buttonga.h>
#include <owl\editview.h>
#include <owl\listview.h>
#include <owl\docmanag.h>
#include <owl\filedoc.h>
#include <owl\printer.h>

#include "wpcamapp.rh"          // Definition of all resources.
#include <tgraphvw.h>

//{{TApplication = myWPCamApp}}
class myWPCamApp : public TApplication {
private:
    int nChild;

private:
    void SetupSpeedBar (TDecoratedMDIFrame *frame);

public:
    myWPCamApp ();
```

```
   virtual ~myWPCamApp ();

   // Public data members used by the print menu commands and Paint routine in
MDIChild.
   TPrinter      *Printer;                    // Printer support.
   BOOL           Printing;                   // Printing in progress.

//{{myWPCamAppVIRTUAL_BEGIN}}
public:
   virtual void InitMainWindow();
//{{myWPCamAppVIRTUAL_END}}

//{{myWPCamAppRSP_TBL_BEGIN}}
protected:
         void EvNewView (TView& view);
   void EvCloseView (TView& view);
   void CmHelpAbout ();
   void EvWinIniChange (char far* section);
//{{myWPCamAppRSP_TBL_END}}
  DECLARE_RESPONSE_TABLE(myWPCamApp);
};   //{{myWPCamApp}}


#endif                              // __wpcamapp_h sentry.
```

```
/*  Project mywpcam1
    VPI & SU
    Copyright © 1993. All Rights Reserved.

    SUBSYSTEM:    mywpcam1.exe Application
    FILE:         wpcamapp.cpp
    AUTHOR:       Kurnia D. Iskandar


    OVERVIEW
    ========
    Source file for implementation of myWPCamApp (TApplication).
*/


#include <owl\owlpch.h>
#pragma hdrstop


#include <wpcamapp.h>
#include <mymdicln.h>
#include <mymdichl.h>
#include <myabtdlg.h>                    // Definition of about dialog.

//{{myWPCamApp Implementation}}

//{{DOC_VIEW}}
DEFINE_DOC_TEMPLATE_CLASS(TFileDocument, TEditView, DocType1);
DEFINE_DOC_TEMPLATE_CLASS(TFileDocument, TGraphView, DocType2);
//{{DOC_VIEW_END}}

//{{DOC_MANAGER}}
DocType1 __dvt1("As Text Edit(*.txt)", "*.txt", 0, "TXT",dtAutoDelete|dtUpdateDir);
DocType2 __dvt2("As Plot View(*.txt)", "*.txt", 0, "TXT",dtAutoDelete|dtUpdateDir);
//{{DOC_MANAGER_END}}

//
// Build a response table for all messages/commands handled
// by the application.
//
DEFINE_RESPONSE_TABLE1(myWPCamApp, TApplication)
//{{myWPCamAppRSP_TBL_BEGIN}}
        EV_OWLVIEW(dnCreate, EvNewView),
        EV_OWLVIEW(dnClose,  EvCloseView),
        EV_COMMAND(CM_HELPABOUT, CmHelpAbout),
        EV_WM_WININICHANGE,
//{{myWPCamAppRSP_TBL_END}}
END_RESPONSE_TABLE;


//////////////////////////////////////////////////////////
// myWPCamApp
// =====
//
myWPCamApp::myWPCamApp () : TApplication("Wrapping Cam Synthesis ver. 1.1")
{

        Printer = 0;
        Printing = FALSE;

        DocManager = new TDocManager(dmMDI | dmMenu);

        // INSERT>> Your constructor code here.
        nChild = 0;

}


myWPCamApp::~myWPCamApp ()
{
        if (Printer)
                delete Printer;

        // INSERT>> Your destructor code here.

}


void myWPCamApp::SetupSpeedBar (TDecoratedMDIFrame *frame)
{
        //
        // Create default toolbar New and associate toolbar buttons with commands.
        //
        TControlBar* cb = new TControlBar(frame);
        cb->Insert(*new TButtonGadget(CM_MDIFILENEW, CM_NEWINPUTFILE));
        cb->Insert(*new TButtonGadget(CM_MDIFILEOPEN, CM_MDIFILEOPEN));
```

```
	cb->Insert(*new TButtonGadget(CM_FILESAVE, CM_FILESAVE));
	cb->Insert(*new TSeparatorGadget(6));
	cb->Insert(*new TButtonGadget(CM_EDITCUT, CM_EDITCUT));
	cb->Insert(*new TButtonGadget(CM_EDITCOPY, CM_EDITCOPY));
	cb->Insert(*new TButtonGadget(CM_EDITPASTE, CM_EDITPASTE));
	cb->Insert(*new TSeparatorGadget(6));
	cb->Insert(*new TButtonGadget(CM_EDITUNDO, CM_EDITUNDO));
	cb->Insert(*new TSeparatorGadget(6));
	cb->Insert(*new TButtonGadget(CM_EDITFIND, CM_EDITFIND));
	cb->Insert(*new TButtonGadget(CM_EDITFINDNEXT,
CM_EDITFINDNEXT));
	cb->Insert(*new TSeparatorGadget(6));
	cb->Insert(*new TButtonGadget(CM_GGRRPOSITIVE,
CM_GGRRPOSITIVE));
	cb->Insert(*new TButtonGadget(CM_GGRRNEGATIVE,
CM_GGRRNEGATIVE));
	cb->Insert(*new TSeparatorGadget(6));
	cb->Insert(*new TButtonGadget(CM_ANALYSISGGRR,
CM_ANALYSISGGRR));
	//cb->Insert(*new TSeparatorGadget(6));
	//cb->Insert(*new TButtonGadget(CM_FILEPRINT, CM_FILEPRINT));
	//cb->Insert(*new TButtonGadget(CM_FILEPRINTPREVIEW,
CM_FILEPRINTPREVIEW));

	// Add fly-over help hints.
	cb->SetHintMode(TGadgetWindow::EnterHints);

	frame->Insert(*cb, TDecoratedFrame::Top);
}


//////////////////////////////////////////////////////////
// myWPCamApp
// =====
// Application intialization.
//
void myWPCamApp::InitMainWindow ()
{
	TDecoratedMDIFrame* frame = new TDecoratedMDIFrame(Name,
MDI_MENU, *(new myMDIClient), TRUE);

	nCmdShow = (nCmdShow != SW_SHOWMINNOACTIVE) ?
SW_SHOWNORMAL : nCmdShow;
```

```
        //
        // Assign ICON w/ this application.
        //
        frame->SetIcon(this, IDI_WPCAM);

        //
        // Menu associated with window and accelerator table associated with table.
        //
        frame->AssignMenu(MDI_MENU);

        //
        // Associate with the accelerator table.
        //
        frame->Attr.AccelTable = MDI_MENU;


        SetupSpeedBar(frame);

        TStatusBar *sb = new TStatusBar(frame, TGadget::Recessed,

       TStatusBar::CapsLock      |

       TStatusBar::NumLock       |

       TStatusBar::ScrollLock    |

       TStatusBar::Overtype);
        frame->Insert(*sb, TDecoratedFrame::Bottom);

        MainWindow = frame;

}


/////////////////////////////////////////////////////////
// myWPCamApp
// =====
// Response Table handlers:
//
void myWPCamApp::EvNewView (TView& view)
{
	TMDIClient *mdiClient = TYPESAFE_DOWNCAST(MainWindow-
>GetClientWindow(), TMDIClient);
	if (mdiClient) {
		myMDIChild* child = new myMDIChild(*mdiClient, 0,
view.GetWindow());

		// Associate ICON w/ this child window.
		child->SetIcon(this, IDI_DOC);

		child->Create();
	}
}


void myWPCamApp::EvCloseView (TView&)
{
}


//////////////////////////////////////////////////////////
// myWPCamApp
// ==========
// Menu Help About mywpcam1.exe command
void myWPCamApp::CmHelpAbout ()
{
	//
	// Show the modal dialog.
	//
	myAboutDlg(MainWindow).Execute();
}

void myWPCamApp::EvWinIniChange (char far* section)
{
	if (lstrcmp(section, "windows") == 0) {
		// If the device changed in the WIN.INI file then the printer
		// might have changed.  If we have a TPrinter (Printer) then
		// check and make sure it's identical to the current device
		// entry in WIN.INI.
      if (Printer) {
         char printDBuffer[255];
         LPSTR printDevice = printDBuffer;
         LPSTR devName = 0;
         LPSTR driverName = 0;
```

```
        LPSTR outputName = 0;

        if (::GetProfileString("windows", "device", "", printDevice, sizeof(printDevice))) {
          // The string which should come back is something like:
          //
          //     HP LaserJet III,hppcl5a,LPT1:
          //
          // Where the format is:
          //
          //     devName,driverName,outputName
          //
          devName = printDevice;
          while (*printDevice) {
            if (*printDevice == ',') {
              *printDevice++ = 0;
              if (!driverName)
                driverName = printDevice;
              else
                outputName = printDevice;
            } else
              printDevice = AnsiNext(printDevice);
          }

          if ((Printer->GetSetup().Error != 0)                             ||
            (lstrcmp(devName, Printer->GetSetup().GetDeviceName()) != 0)   ||
            (lstrcmp(driverName, Printer->GetSetup().GetDriverName()) != 0) ||
            (lstrcmp(outputName, Printer->GetSetup().GetOutputName()) != 0)) {

            // New printer installed so get the new printer device now.
            delete Printer;
            Printer = new TPrinter;
          }
        } else {
          // No printer installed (GetProfileString failed).
          delete Printer;
          Printer = new TPrinter;
        }
      }
    }
}


int OwlMain (int , char* [])
```

```
{
   myWPCamApp     App;
   int            result;

   result = App.Run();

   return result;
}
```

```
#if !defined(__nwnptfdg_h)           // Sentry, use file only if it's not already included.
#define __nwnptfdg_h

/*  Project wrpcam11
   VPI & SU
   Copyright © 1993. All Rights Reserved.

   SUBSYSTEM:    wrpcam11.exe Application
   FILE:         nwnptfdg.h
   AUTHOR:       Kurnia D. Iskandar


   OVERVIEW
   ========
   Class definition for NewInputFileDialog (TDialog).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include <owl\dialog.h>
#include <owl\radiobut.h>
#include <owl\edit.h>
#include <owl\validate.h>

#include "wpcamapp.rh"          // Definition of all resources.

//
// transfer buffer structure for this dialog
//
struct TransBuffNewFile
{
        TransBuffNewFile();

  char Filename[13];
        char    NumPoints[5];
        UINT  CurveFile;
        UINT  PosGGRRSprocket;
        UINT  NegGGRRSprocket;
        UINT  PosGGRRPulley;
        UINT  NegGGRRPulley;
};
```

```
//{{TDialog = NewInputFileDialog}}
class NewInputFileDialog : public TDialog {
public:
   NewInputFileDialog (TWindow* parent, TransBuffNewFile& transfer);
   virtual ~NewInputFileDialog ();
};   //{{NewInputFileDialog}}


#endif                              // __nwnptfdg_h sentry.
```

```
/*  Project wrpcam11
   VPI & SU
   Copyright © 1993. All Rights Reserved.

   SUBSYSTEM:    wrpcam11.exe Application
   FILE:         nwnptfdg.cpp
   AUTHOR:       Kurnia D. Iskandar


   OVERVIEW
   ========
   Source file for implementation of NewInputFileDialog (TDialog).
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include "nwnptfdg.h"

//
// Constructor of transfer buffer
//
TransBuffNewFile::TransBuffNewFile ()
{
        Filename[0] = '\0';
        strcpy(NumPoints, "25");
        CurveFile = BF_CHECKED;
        PosGGRRSprocket = BF_UNCHECKED;
        NegGGRRSprocket = BF_UNCHECKED;
        PosGGRRPulley = BF_UNCHECKED;
        NegGGRRPulley = BF_UNCHECKED;
}


//{{NewInputFileDialog Implementation}}


NewInputFileDialog::NewInputFileDialog (TWindow* parent, TransBuffNewFile&
transfer):
        TDialog(parent, IDD_NEWINPUTFILEDLG)
{
        // INSERT>> Your constructor code here.
        new TEdit(this, IDC_FILENAME, sizeof(transfer.Filename));
        new TEdit(this, IDC_NUMBERPOINTS, sizeof(transfer.NumPoints))
                ->SetValidator(new TFilterValidator("0-9"));
        new TRadioButton(this, IDC_CURVEFILE);
        new TRadioButton(this, IDC_PGGRRSPROCKET);
        new TRadioButton(this, IDC_NGGRRSPROCKET);
        new TRadioButton(this, IDC_PGGRRPULLEY);
        new TRadioButton(this, IDC_NGGRRPULLEY);

        SetTransferBuffer(&transfer);
}


NewInputFileDialog::~NewInputFileDialog ()
{
        Destroy();

        // INSERT>> Your destructor code here.

}
```

```
#if !defined(__tcam_h)              // Sentry, use file only if it's not already included.
#define __tcam_h

/*  Project wrpcam11
        VPI & SU
        Copyright © 1993. All Rights Reserved.

        SUBSYSTEM:    wrpcam11.exe Application
        FILE:         tcam.h
        AUTHOR:       Kurnia D. Iskandar


        OVERVIEW
        ========
        Class definition for TCam.
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include <sort.h>

//
// Class TCam declaration
//
class TCam {
 private:
        double* cam_x, *cam_y;       // x-y cartesian coordinate form
        double *cam_r;          // cam radius
        TPoint* cam_points;  // points in client window coordinate form
        int num_pts;    // number of cam's points
 public:
        TCam ();
        TCam (double* x, double* y, int n);
        ~TCam ();
        void set_cam_xy(double* x, double* y, int n);
        TPoint get_start_pts ();
        TPoint get_end_pts ();
        virtual void draw_cam (TDC& dc, TRect& rect);
};
#endif
```

```
/*  Project wrpcam11
        VPI & SU
        Copyright © 1993. All Rights Reserved.

        SUBSYSTEM:    wrpcam11.exe Application
        FILE:         tcam.cpp
        AUTHOR:       Kurnia D. Iskandar


        OVERVIEW
        ========
        Source file for implementation of TCam.
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include <tcam.h>
#include <math.h>

//
// Class TCam definition
//
TCam::TCam () {
        cam_x = NULL;
        cam_y = NULL;
        cam_points = NULL;
        cam_r = NULL;
        num_pts = 0;
}
TCam::TCam (double* x, double* y, int n) {
        num_pts = n;
        cam_x = new double[num_pts];
        cam_y = new double[num_pts];
        cam_r = new double[num_pts];
        for (int i = 0; i < num_pts; i++) {
                cam_x[i] = x[i];
                cam_y[i] = y[i];
                cam_r[i] = sqrt(cam_x[i]*cam_x[i] + cam_y[i]*cam_y[i]);
        }
}
TCam::~TCam () {
        delete [] cam_x;
        delete [] cam_y;
        delete [] cam_points;
        delete [] cam_r;
}
void TCam::set_cam_xy(double* x, double* y, int n) {
        num_pts = n;
        cam_x = new double[num_pts];
        cam_y = new double[num_pts];
        cam_r = new double[num_pts];
        for (int i = 0; i < num_pts; i++) {
                cam_x[i] = x[i];
                cam_y[i] = y[i];
                cam_r[i] = sqrt(cam_x[i]*cam_x[i] + cam_y[i]*cam_y[i]);
        }
}
TPoint TCam::get_start_pts () {
        return cam_points[0];
}
TPoint TCam::get_end_pts () {
        return cam_points[num_pts-1];
}
void TCam::draw_cam(TDC& dc, TRect& rect)
{
        TPoint center;
        TPoint upleft;
        TPoint loright;
        int width, height;
        double rmax;

        rmax = DMax(cam_r,num_pts);

        // the center of the cam
        center.x = 0.5 * rect.right;
        center.y = 0.5 * rect.bottom;

        // set up the width & height of the focus box
        if (rect.bottom < rect.right)
                width = height = 0.80 * rect.bottom;
        else if (rect.bottom > rect.right)
                width = height = 0.80 * rect.right;
        else
        {
                width = 0.80 * rect.right;
        height = 0.80 * rect.bottom;
}

//
// rectangular frame coordinates
//
upleft.x = center.x - 0.5 * width;
upleft.y = center.y - 0.5 * height;
loright.x = center.x + 0.5 * width;
loright.y = center.y + 0.5 * height;

//
// rotate the focus box to properly align the cam
//
if (cam_y[0] < 0.0) {
        TPoint dummy;

        dummy = upleft;
        upleft = loright;
        loright = dummy;
}
//
// draw the enclosing box (focus box)
//
//TPen pen1(TColor::Black, 1, PS_SOLID);
//dc.SelectObject(pen1);
//dc.Rectangle(upleft, loright);

// mark the center of the cam
TPen pen3(TColor::LtGreen, 1, PS_SOLID);
dc.SelectObject(pen3);
dc.MoveTo(center.x-5,center.y);
dc.LineTo(center.x+5,center.y);
dc.MoveTo(center.x,center.y-5);
dc.LineTo(center.x,center.y+5);

// make scatter or point plot of the cam profile
// in polar form
TPen pen2(TColor::LtRed, 1, PS_SOLID);
dc.SelectObject(pen2);

cam_points = new TPoint[num_pts];
for (int j = 0; j < num_pts; j++) {
```

```
            if (cam_x[j] > 0.0 && cam_y[j] > 0.0) {
                    cam_points[j].x = (cam_x[j]/rmax)*(loright.x-center.x)+center.x;
                    cam_points[j].y = (cam_y[j]/rmax)*(upleft.y-center.y)+center.y;
            }else
            if (cam_x[j] < 0.0 && cam_y[j] > 0.0) {
                    cam_points[j].x = (cam_x[j]/-rmax)*(upleft.x-center.x)+center.x;
                    cam_points[j].y = (cam_y[j]/rmax)*(upleft.y-center.y)+center.y;
            }else
            if (cam_x[j] < 0.0 && cam_y[j] < 0.0) {
                    cam_points[j].x = (cam_x[j]/-rmax)*(upleft.x-center.x)+center.x;
                    cam_points[j].y = (cam_y[j]/-rmax)*(loright.y-center.y)+center.y;
            }else
            if (cam_x[j] > 0.0 && cam_y[j] < 0.0) {
                    cam_points[j].x = (cam_x[j]/rmax)*(loright.x-center.x)+center.x;
                    cam_points[j].y = (cam_y[j]/-rmax)*(loright.y-center.y)+center.y;
            }
            //dc.Rectangle(cam_points[j].x-1,cam_points[j].y-
1,cam_points[j].x+1,cam_points[j].y+1);
        }//end for-loop

        dc.MoveTo (cam_points[0]);
        for (j = 1; j < num_pts; j++){
            dc.LineTo (cam_points[j]);
            dc.MoveTo (cam_points[j]);
        }// end for-loop
}
```

```
#ifndef _t2curve_h
#define _t2curve_h

/* Project wrpcam11
        VPI & SU
        Copyright © 1993. All Rights Reserved.

        SUBSYSTEM:    wrpcam11.exe Application
        FILE:         t2curve.h
        AUTHOR:       Kurnia D. Iskandar


        OVERVIEW
        ========
        Class definition for T2Curve.
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include <sort.h>

//
// Class T2Curve declaration
//
class T2Curve {
        private:
                double *curve_x, *curve_y;
                TPoint* curve_points;
                int num_pts;
                char* main, *x_label, *y_label;
        public:
                T2Curve ();
                T2Curve (double *x, double *y, int n);
                ~T2Curve ();
                void set_curve_xy (double *x, double *y, int n);
                void set_label (const char* label1,const char* label2,const char* label3);
                virtual void draw_curve(TDC& dc, TRect& rect);
};
#endif
```

```
/*  Project wrpcam11
        VPI & SU
        Copyright © 1993. All Rights Reserved.

        SUBSYSTEM:    wrpcam11.exe Application
        FILE:         t2curve.cpp
        AUTHOR:       Kurnia D. Iskandar


        OVERVIEW
        ========
        Source file for implementation of T2Curve.
*/

#include <owl\owlpch.h>
#pragma hdrstop

#include <t2curve.h>
#include <stdio.h>

//
// Class T2Curve definition
//
T2Curve::T2Curve (){
        curve_x = NULL;
        curve_y = NULL;
        curve_points = NULL;
}
T2Curve::T2Curve (double *x, double *y, int n) {
        num_pts = n;
        curve_x = new double[num_pts];
        curve_y = new double[num_pts];
        for (int i = 0; i < num_pts; i++) {
                curve_x[i] = x[i];
                curve_y[i] = y[i];
        }
}
T2Curve::~T2Curve () {
        delete [] curve_x;
        delete [] curve_y;
        delete [] curve_points;
}
void T2Curve::set_curve_xy (double *x, double *y, int n) {
        num_pts = n;
        curve_x = new double[num_pts];
        curve_y = new double[num_pts];
        for (int i = 0; i < num_pts; i++){
                curve_x[i] = x[i];
                curve_y[i] = y[i];
        }
}
void T2Curve::set_label (const char* label1, const char* label2,
const char* label3) {
        main = new char[strlen(label1)+1];
        x_label = new char[strlen(label2)+1];
        y_label = new char[strlen(label3)+1];
        strcpy(main, label1);
        strcpy(x_label, label2);
        strcpy(y_label, label3);
}
void T2Curve::draw_curve(TDC& dc, TRect& rect) {
        char label[7];
        TPoint center;
        TPoint upleft;
        TPoint loright;
        int xMinPost, yMinPost, xMaxPost, yMaxPost;
        int width, height;
        double xmin, ymin, xmax, ymax;

        xmin = DMin(curve_x,num_pts);
        ymin = DMin(curve_y,num_pts);
        xmax = DMax(curve_x,num_pts);
        ymax = DMax(curve_y,num_pts);

        center.x = 0.5 * rect.right;
        center.y = 0.5 * rect.bottom;

        width = 0.80 * rect.right;
        height = 0.80 * rect.bottom;

        //
        // rectangular frame coordinates
        //
        upleft.x = center.x - 0.5 * width;
        upleft.y = center.y - 0.5 * height;
        loright.x = center.x + 0.5 * width;
loright.y = center.y + 0.5 * height;

xMinPost = upleft.x;
yMinPost = loright.y;
xMaxPost = loright.x;
yMaxPost = upleft.y;

//
// draw the label
//
sprintf(label,"%7.3g",xmin);
dc.SetTextAlign(TA_TOP|TA_CENTER);
dc.TextOut(xMinPost,yMinPost+1,label);

sprintf(label,"%7.3g",ymin);
dc.SetTextAlign(TA_RIGHT|TA_BOTTOM);
dc.TextOut(xMinPost,yMinPost,label);

sprintf(label,"%7.3g",ymax);
dc.SetTextAlign(TA_RIGHT|TA_BOTTOM);
dc.TextOut(xMinPost-1,yMaxPost+1,label);

sprintf(label,"%7.3g",xmax);
dc.SetTextAlign(TA_TOP|TA_CENTER);
dc.TextOut(xMaxPost,yMinPost+1,label);

dc.SetTextAlign(TA_TOP|TA_CENTER);
dc.TextOut(0.5*rect.right,0.01*rect.bottom,main);
dc.TextOut(0.5*rect.right,0.92*rect.bottom,x_label);
dc.SetTextAlign(TA_LEFT|TA_CENTER);
dc.TextOut(0.05*rect.right,0.5*rect.bottom,y_label);

//
// draw the enclosing box (frame box)
//
TPen pen1(TColor::Black, 1, PS_SOLID);
dc.SelectObject(pen1);
dc.Rectangle(upleft,loright);

// make scatter or point plot
TPen pen2(TColor::LtRed, 1, PS_SOLID);
dc.SelectObject(pen2);
curve_points = new TPoint[num_pts];
```

```
          for (int j = 0; j < num_pts; j++)
          {
                    curve_points[j].x = ((curve_x[j] - xmin)/(xmax - xmin))
                                                                                          *(xMaxPost-
xMinPost)+xMinPost;
                    curve_points[j].y = ((curve_y[j] - ymin)/(ymax - ymin))
                                                                                          *(yMaxPost-
yMinPost)+yMinPost;
                    dc.Rectangle(curve_points[j].x-1,curve_points[j].y-1,
                              curve_points[j].x+1,curve_points[j].y+1);
          }
}
```

B.2. Wrapping Cam Classes

Class Name:	Header File:	Resource File:	About The Class:
GGRR	ggrr.h	ggrr.cpp	General GGRR synthesis class
GGRRAnls	ggrranls.h	ggrranls.cpp	GGRR analysis class
GGRRN	ggrrn.h	ggrrn.cpp	negative GGRR synthesis class
GGRRP	ggrrp.h	ggrrp.cpp	positive GGRR synthesis class
WCamAnls	wcamanls.h		Abstract base class for wrapping cam analysis
WCamSyn	wcamsyn.h	wcamsyn.cpp	Base class for wrapping cam synthesis

```
//ggrr.h
//Header file for GGRR class of wrapping cam mechanism.
//Written by: Kurnia D. Iskandar
//Date: 11/01/1995
//-----------------------------------------------------
#ifndef _GGRR_H_
#define _GGRR_H_
#include <math.h>
#include <point2d.h>
#include <wcamsyn.h>
//t = follower's thickness.
//C = center distance between cam and sprocket or pulley.
//r1 = radius of output sprocket.
//phi = angle (radian) locating the contact point between follower
//     and sprocket or pulley.
//theta = angle (radian) of rotation of sprocket or pulley
//       around the cam.
//q = length of follower between points of contacts.
//camPts = points on the cam profile surface.
//camPch = points on the cam pitch surface.
//numPts = number of points to be generated by synthesis process.
class GGRR : public WCamSyn{
        public:
                GGRR(); //default constructor
                ~GGRR();     //default destructor

        //Redefined pure virtual functions
                void cam_motion(int n);
                point2d* point_vector(); //surface in cartesian coordinate
                point2d* pitch_point();         //pitch surface in cartesian coordinate

        protected:
                double t, C, r1;
                double *phi, *theta, *q;
                point2d* camPts;
                point2d* camPch;
                int numPts;
};
#endif
```

```
//ggrrp.cpp
//Class method definition for GGRR positive wrapping cam.
//Written by: Kurnia D. Iskandar
//Date: 10/31/1995
//-------------------------------------------------------
#include <ggrrp.h>
//Default constructor
GGRRP::GGRRP(){
        h = NULL; q = NULL;
        phi = NULL; theta = NULL;
        camPts = NULL; camPch = NULL;
        numPts = 0;
}

//Default destructor
GGRRP::~GGRRP(){
        delete []h;
        delete []q;
        delete []phi;
        delete []theta;
        delete []camPts;
        delete []camPch;
}

//Public member function to calculate the gear radius.
//Function reads in number of teeth and chain's pitch.
void GGRRP::sprocket_radius(int N, double p){
        r1 = gear1.Radius(p,N);
}

//Function reads in the pulley's radius
void GGRRP::pulley_radius(double rad){
        r1 = rad;
}

//Public member function to calculate torque moment arm.
//Function reads in the stroke of weight stack.
void GGRRP::torque_moment_arm(double stroke){
        double denum, poly;

        h = new double[numPts];
        L = stroke;
        denum = polyintg(coef, degree, xmin, xmax);
        //cout<<"denum="<<denum;
        //Given the cam range of motion theta, find torque moment arm.
        for (int i=0; i<numPts; i++){
                poly = 0.0;
                for (int j=0; j<=degree; j++)
                        poly += coef[j]*riseto(theta[i],(double)(degree-j));
                h[i] = L * poly/denum;
                //cout<<"h["<<i<<"] = "<<h[i]<<" ";
        }
}

//Function to calculate the follower length. Input:
//branching index (index = 1 or -1), center distance (center), and follower
//thickness(thick).
void GGRRP::follower_length_thick(int index, double center, double thick){
        double hprime, denum, dt;
        double a, b;

        sign = index;
        C = center;
        t = thick;
        denum = polyintg(coef,degree,xmin,xmax);
        //cout<<"denum="<<denum;
        q = new double[numPts];
        phi = new double[numPts];

        for (int i=0; i<numPts; i++){
                a = C*C-(h[i]-sign*r1)*(h[i]-sign*r1);
                //cout<<"a="<<a<<" ";
                b = sqrt(a);
                //cout<<"b="<<b<<" ";
                dt = polyderv(theta[i],coef,degree,1);
                //cout<<"dt="<<dt<<" ";
                hprime = L / denum * dt;
                //cout<<"hprime="<<hprime<<" ";
                q[i] = a / (b + hprime);
                //cout<<"q["<<i<<"]= "<<q[i]<<" ";
                phi[i] = acos((h[i] - sign * r1) / C);
                //cout<<"phi["<<i<<"]="<<phi[i]<<" ";
        }
}
```

```
//ggrranls.h
//Header file of GGRR wrapping cam analysis class.
//Written by: Kurnia D. Iskandar
//Date: 11/20/1995
//-----------------------------------------------------
#ifndef _GGRRANLS_H_
#define _GGRRANLS_H_

#include <math.h>
#include <sort.h>
#include <arraydb.h>
#include <leastqr.h>
#include <wcamanls.h>
#include <riseto.h>
#include <polyderv.h>
#include <point2d.h>

class GGRRAnls : public WCamAnls {
	public:
	//Redefine the virtual functions
	//The input: d = degree of polynomial curve fit for cam surface.
	//				num = number of data points to be calculated.
		point2d* kinematic_analysis(int num, int d); //return output force synthesis
or

//position function curve

		GGRRAnls();   //default constructor
		~GGRRAnls();  //default destructor
	//Member function to read in wrapping cam data
	//The inputs are: pol = polar coordinate
	//		num = number of cam surface data points
	//				index = the branching sign (1 = uncrossed, -
1 = crossed)
	//		r = radius of output sprocket
	//		t = follower's thickness
	//		c = center distance
	//-----------------------------------------------------------------
		void wrapping_cam_data(point2d* pol,int num,int index,double r,
			double t,double c); //read in all needed data for analysis

	//Member function to return number of data calculated
		int number_of_data(){return m;}
```

```
        protected:
        //Variables: curve = strength curve data points
        //                                          x = follower's position on cam surface
        //                  y = radius of polar coordinate of cam suface
        //                sign = branching index (1=uncrossed, -1=crossed)
        //                                        org_num = number of actual cam's data points
        //                 m = number of data points to be calculated
        //                 tau = follower's angle position
        //               theta = cam's angle of rotation
        //                  h = moment arm
        //              radius = output sprocket's radius
        //             thickness = follower's thickness
        //--------------------------------------------------------------------------
                point2d* curve;
                ArrayDb x;
                ArrayDb y;
                int sign, org_num, m;
                double* h, *theta, *tau;
                double radius, center, thickness, range;
};
#endif
```

```
//ggrranls.cpp
//Class method for GGRR analysis.
//Written by: Kurnia D. Iskandar
//Date: 11/25/1995
//----------------------------------------------------
#include <ggrranls.h>

//Default constructor
GGRRAnls::GGRRAnls(){
	curve = NULL; h = NULL; theta = NULL; tau = NULL;
	m = 0;
}

//Default destructor
GGRRAnls::~GGRRAnls(){
	delete [] curve; delete [] h; delete [] theta;
	delete [] tau;
}

//Member functions
void GGRRAnls::wrapping_cam_data(point2d* pol,int num,
int index,double r,double t,double c){
	double* dum;
	double min,max;
	ArrayDb a(num);
	ArrayDb b(num);

	org_num = num;
	sign = index;
	radius = r;
	center = c;
	thickness = t;
	x = a; y = b;

	dum = new double[num];


	for (int j=0; j < num; j++){
		(pol[j].y > pol[num-1].y) ? (dum[j]=pol[j].y-360.0):(dum[j]=pol[j].y);
	}
	min = DMin(dum, num);
	max = DMax(dum, num);
	range = max - min;
```

```
        for (int i=0; i<num; i++){
                x[i] = dum[i]-min;                //unit in degree
                y[i] = pol[i].x;  //radius of polar coordinate data of cam surface
        }

        delete [] dum;
}

//The virtual function
point2d* GGRRAnls::kinematic_analysis(int num, int d){
        double dp, p, step;
        double* sigma;
        ArrayDb coef;

        //Approximate the cam surface with polynomial curve (least square fit)
        coef = leastqr(x,y,org_num,d);

        m = num;   //number of data points to be calculated
        tau = new double[m];  //unit in degree
        step = range/m;

        tau[0] = 0.0;
        for (int i=1; i < m; i++){
                tau[i]=tau[i-1]+step;
        }

        //Calculate the strength curve data; i.e. moment arms and the cam rotation
        //angles.
        h = new double[m];
        theta = new double[m];
        sigma = new double[m];
        curve = new point2d[m];
        for (int j=0; j < m; j++){
                p = 0.0;
                for (int n=0; n <= d; n++) p += coef[n]*riseto(tau[j],(double)(d-n));
                dp = polyderv(tau[j], coef, d, 1);
                sigma[j] = atan(dp/p);
                h[j] = p * cos(sigma[j]) + thickness/2.0;
                theta[j] = tau[j] + sigma[j] + acos((h[j] - sign*radius)/center);

                curve[j].x = theta[j];    //unit in radian
                curve[j].y = h[j];
        }

        delete [] sigma;

        return curve;
}
```

```
//ggrrn.h
//Header file for GGRR wrapping cam with negative cam.
//Written by: Kurnia D. Iskandar
//Date: 11/01/1995
//-----------------------------------------------------
#ifndef _GGRRN_H_
#define _GGRRN_H_
#include <math.h>
#include <polyintg.h>
#include <polyderv.h>
#include <riseto.h>
#include <sprocket.h>
#include <point2d.h>
#include <ggrr.h>

class GGRRN : public GGRR{
	public:
		GGRRN();	//default constructor
		~GGRRN();	//default destructor

	//Member functions
		//Get radius of sprocket. Input: number of teeth for output
		//sprocket(N1), number of teeth for input sprocket(N2), and
		//chain's pin separation(p).
		void sprocket_radius(int N1, int N2, double p);
		//Get radius of pulleys. Input: radius of output pulley(rad1)
		//and radius of input pulley(rad2).
		void pulley_radius(double rad1, double rad2);
		//Function to calculate torque's moment arm. Input: the weight's stroke.
		void torque_moment_arm(double stroke);
		//Function to calculate follower length. Input: cam's branching (index =1 or
		//index = -1), center distance (center), and follower thickness(thick).
		void follower_length_thick(int index, double center, double thick);

	//Redefine the virtual function of wcamsyn base class.
		void cam_motion(int n);

	protected:
	Sprocket gear1;
		Sprocket gear2;
		double L, r2, k;
		double *h, *S;
		int sign;
};
#endif
```

```
//ggrrn.cpp
//Class method definition for GGRR wrapping cam with negative cam.
//Written by: Kurnia D. Iskandar
//Date: 11/01/1995
//-------------------------------------------------------
#include <ggrrn.h>
//Default constructor
GGRRN::GGRRN(){
        h = NULL; q = NULL;
        phi = NULL; theta = NULL;
        camPts = NULL; camPch = NULL; S = NULL;
        numPts = 0;
}

//Default destructor
GGRRN::~GGRRN(){
        delete []h; delete []q;
        delete []phi; delete []theta;
        delete []S; delete []camPts;
        delete []camPch;
}

void GGRRN::cam_motion(int n){
        double step;

        numPts = n+1;
        step = (xmax - xmin)/n;
        S = new double[numPts];
        for (int i=0; i<numPts; i++) S[i] = xmin + i*step;
}

//Get radius of sprocket. Input: number of teeth for output
//sprocket(N1), number of teeth for input sprocket(N2), and
//chain's pin separation(p).
void GGRRN::sprocket_radius(int N1, int N2, double p){
        r1 = gear1.Radius(p, N1);
        r2 = gear2.Radius(p, N2);
}

//Get radius of pulleys. Input: radius of output pulley(rad1)
//and radius of input pulley(rad2).
void GGRRN::pulley_radius(double rad1, double rad2){
        r1 = rad1; r2 = rad2;
}

//Function to calculate torque's moment arm. Input: the weight's stroke.
void GGRRN::torque_moment_arm(double stroke){
        double B, ds, poly;

        L = stroke;
        ds = polyintg(coef, degree, 0.0, xmax);
        k = L / ds;

        theta = new double[numPts];
        h = new double[numPts];

        for (int i=0; i<numPts; i++){
                poly = 0.0;
                for (int j=0; j<=degree; j++) poly += coef[j]*riseto(S[i],(double)(degree-
j));
                h[i] = r2 / (poly * k);
                B = polyintg(coef, degree, 0.0, S[i]);
                theta[i] = k/r2 * B;
        }
}

void GGRRN::follower_length_thick(int index, double center, double thick){
        double a, b, c, poly, ds;

        sign = index;
        C = center;
        t = thick;
        q = new double[numPts];
        phi = new double[numPts];
        for (int i=0; i<numPts; i++){
                a = C*C - (h[i]-sign*r1)*(h[i]-sign*r1);
                b = sqrt(a);
                poly = 0.0;
                for (int j=0; j<=degree; j++) poly += coef[j]*riseto(S[i],(double)(degree-
j));
                ds = polyderv(S[i],coef,degree,1);
                c = r2 * r2 * ds / (k * k * poly * poly * poly);
                q[i] = a / (b - c);
                phi[i] = acos((h[i] - sign*r1)/C);
        }
}
```

```
//ggrrp.h
//Header file for GGRR wrapping cam mechanism with positive cam.
//Written by: Kurnia D. Iskandar
//Date: 10/31/1995
//-----------------------------------------------------------------
//L = stroke of the weight.
//h = torque moment arm of tension in the follower.
//gear1 = a sprocket object.
//sign = -1 for crossed or 1 for uncrossed configuration.
#ifndef _GGRRP_H_
#define _GGRRP_H_
#include <math.h>
#include <ggrr.h>
#include <point2d.h>
#include <sprocket.h>
#include <polyintg.h>
#include <polyderv.h>

class GGRRP : public GGRR{
        public:
                GGRRP();
                ~GGRRP();
        //Member functions

                void sprocket_radius(int N, double p);          //sprocket radius
                void pulley_radius(double rad);
                void torque_moment_arm(double stroke);
                void follower_length_thick(int index, double center, double thick);

        protected:
                Sprocket gear1;
                double L;
                double *h;
                int sign;
};
#endif
```

```
//ggrrp.cpp
//Class method definition for GGRR positive wrapping cam.
//Written by: Kurnia D. Iskandar
//Date: 10/31/1995
//-----------------------------------------------------
#include <ggrrp.h>
//Default constructor
GGRRP::GGRRP(){
        h = NULL; q = NULL;
        phi = NULL; theta = NULL;
        camPts = NULL; camPch = NULL;
        numPts = 0;
}

//Default destructor
GGRRP::~GGRRP(){
        delete []h;
        delete []q;
        delete []phi;
        delete []theta;
        delete []camPts;
        delete []camPch;
}

//Public member function to calculate the gear radius.
//Function reads in number of teeth and chain's pitch.
void GGRRP::sprocket_radius(int N, double p){
        r1 = gear1.Radius(p,N);
}

//Function reads in the pulley's radius
void GGRRP::pulley_radius(double rad){
        r1 = rad;
}

//Public member function to calculate torque moment arm.
//Function reads in the stroke of weight stack.
void GGRRP::torque_moment_arm(double stroke){
        double denum, poly;

        h = new double[numPts];
        L = stroke;
        denum = polyintg(coef, degree, xmin, xmax);
        //cout<<"denum="<<denum;
        //Given the cam range of motion theta, find torque moment arm.
        for (int i=0; i<numPts; i++){
                poly = 0.0;
                for (int j=0; j<=degree; j++)
                        poly += coef[j]*riseto(theta[i],(double)(degree-j));
                h[i] = L * poly/denum;
                //cout<<"h["<<i<<"] = "<<h[i]<<" ";
        }
}

//Function to calculate the follower length. Input:
//branching index (index = 1 or -1), center distance (center), and follower
//thickness(thick).
void GGRRP::follower_length_thick(int index, double center, double thick){
        double hprime, denum, dt;
        double a, b;

        sign = index;
        C = center;
        t = thick;
        denum = polyintg(coef,degree,xmin,xmax);
        //cout<<"denum="<<denum;
        q = new double[numPts];
        phi = new double[numPts];

        for (int i=0; i<numPts; i++){
                a = C*C-(h[i]-sign*r1)*(h[i]-sign*r1);
                //cout<<"a="<<a<<" ";
                b = sqrt(a);
                //cout<<"b="<<b<<" ";
                dt = polyderv(theta[i],coef,degree,1);
                //cout<<"dt="<<dt<<" ";
                hprime = L / denum * dt;
                //cout<<"hprime="<<hprime<<" ";
                q[i] = a / (b + hprime);
                //cout<<"q["<<i<<"]= "<<q[i]<<" ";
                phi[i] = acos((h[i] - sign * r1) / C);
                //cout<<"phi["<<i<<"]="<<phi[i]<<" ";
        }
}
```

```
//wcamanls.h
//Header file for Wrapping Cam Analysis abstract base class.
//Written by: Kurnia D. Iskandar
//Date: 11/14/1995
//------------------------------------------------------
#ifndef _WCAMANLS_H_
#define _WCAMANLS_H_
#include <point2d.h>

class WCamAnls {
	public:

	//Pure virtual functions.
		virtual point2d* kinematic_analysis(int num, int d)=0; //returns data points
};
#endif
```

```
//wcamsyn.h
//Header file for wrapping cam synthesis abstract base class.
//Written by: Kurnia D. Iskandar
//Date: 10/30/1995
//-------------------------------------------------------
#ifndef _WCAMSYN_H_
#define _WCAMSYN_H_
#include <math.h>
#include <arraydb.h>
#include <leastqr.h>
#include <point2d.h>
#include <sort.h>
//coef = an ArrayDb of polynomial coefficients
//degree = a degree of polynomial curve fitting
//numData = number of data points of input curve
//xmin = lower boundary of input data
//xmax = upper boundary of ouput data
//polar = polar coordinate

class WCamSyn{
        public:
                WCamSyn(){polar = NULL;}
                ~WCamSyn(){delete []polar;}
        //Member function to calculate the polynomial function
        //coefficients by least square method. Input: number of
        //data points(n), degree of polynomial(d), abscisca (x), and
        //ordinate (y). This function also calculates the xmin
        //and xmax; the lower and upper boundary of input data.
                void poly_coef(int n, int d, double *x, double *y);
        //Member function to convert cartesian to polar coordinate.
                point2d* polar_coor(int n, point2d* cartesian);
        //a pure virtual function to find the cam range of motion.
        //Input: number of the cam position(n).
                virtual void cam_motion (int n) = 0;
        //a pure virtual function to calculate cam surface.
                virtual point2d* point_vector() = 0;
        //a pure virtual function to calculate cam pitch.
                virtual point2d* pitch_point() = 0;
        protected:
                ArrayDb coef;
                point2d* polar;
                double xmin, xmax;
                int degree, numData;
};
#endif
```

```
//wcamsyn.cpp
//Class method definition for wrapping cam abstract base class.
//Written by: Kurnia D. Iskandar
//Date: 10/31/1995
//-------------------------------------------------------
#include <wcamsyn.h>
void WCamSyn::poly_coef(int n, int d, double *x, double *y){
	double ymax, *xrad;

	numData = n;
	degree = d;
	xrad = new double[numData];
	//Convert degree to radian
	for (int j=0; j<numData; j++) xrad[j] = x[j]/180.0 * M_PI;
	ArrayDb a(xrad,numData);
	xmin = DMin(xrad,numData);//lower boundary
	//cout<<"\nlower boundary = "<<xmin;
	xmax = DMax(xrad,numData);		//upper boundary
	//cout<<"\nupper boundary = "<<xmax;
	ymax = DMax(y,numData);	//max. value for ordinate normalization
	//Normalizing the ordinate of input data
	ArrayDb ynormal(numData);
	for (int i=0; i<numData; i++) ynormal[i]=y[i]/ymax;
	//Curve fit the data by least square a polynomial method
	coef = leastqr(a,ynormal,numData,degree);
	//cout<<"\nThe curve fit coefficients:\n";
	//cout<<coef;

	delete []xrad;
}

//Member function to convert cartesian to polar coordinate.
point2d* WCamSyn::polar_coor(int n, point2d* cartesian){
	double *radius, *angle;
	double a, b;

	radius = new double[n+1];
	angle = new double[n+1];
	for (int i=0; i<n+1; i++){
		a = cartesian[i].x * cartesian[i].x + cartesian[i].y * cartesian[i].y;
		b = cartesian[i].y/cartesian[i].x;
		radius[i] = sqrt(a);
		angle[i] = atan(b)/M_PI * 180.0; //unit in degree
```

```
                if (cartesian[i].x<0.0 && cartesian[i].y>0.0){
                        angle[i] += 180.0;
                }else if(cartesian[i].x<0.0 && cartesian[i].y<0.0){
                        angle[i] += 180.0;
                }else if (cartesian[i].x>0.0 && cartesian[i].y<0.0){
                        angle[i] += 360.0;
                }
        }

        polar = new point2d[n+1];
        for (int j=0; j<n+1; j++){
                polar[j].x = radius[j];
                polar[j].y = angle[j];
        }
        delete []radius;
        delete []angle;

        return polar;
}
```

B.3. Supporting Functions

Class/Function Name:	Header File:	Resource File:	About The Class:
Array2D	array2d.h	array2d.cpp	2-D array or matrix
ArrayDb	arraydb.h	arraydb.cpp	Dynamic array
DMin & Dmax	sort.h	sort.cpp	Sorting functions
gausswp	gausswp.h	gausswp.cpp	Gaussian elimination with partial pivoting
leastqr	leastqr.h	leastqr.cpp	Least square method for polynomial functions
point2d	point2d.h	point2d.cpp	2-D points
polyderv	polyderv.h	polyderv.cpp	Function to calculate a derivative value of polynomial function
polyintg	polyintg.h	polyintg.cpp	Numerical integration of polynomial function
riseto	riseto.h	riseto.cpp	Modified power function
Sprocket	sprocket.h	sprocket.cpp	Function to calculate sprocket radius from number of teeth and sprocket pitch

```
//array2d.h
//class declaration for 2D array
//Written by:Kurnia D. Iskandar
//Date: 09/22/1995
//------------------------------------------------------
#ifndef _ARRAY2D_H_
#define _ARRAY2D_H_

#include<iostream.h>
#include<stdlib.h>      //exit() function

const int numPtr = 10;//maximum array size is 10 by 10

class Array2D{
        private:
                unsigned int row, column;
        protected:
                double * ptr[numPtr];
        public:
                //default constructor
                Array2D();
                //initial Array2D of n by m, and set each element to val
                Array2D(unsigned int n, unsigned int m, double val=0.0);
                //initial Array2D object to another Array2D object
                Array2D(const Array2D & a);
                //destructor
                ~Array2D();
                //public methods
                unsigned int arsize(){return row;}
                //overloaded operators
                double * operator[](int i);
                const double * operator[](int i) const;
                Array2D & operator=(const Array2D & a);
};

#endif
```

```
//array2d.cpp
//class methods for 2D array
//Written by:Kurnia D. Iskandar
//Date: 09/22/1995
//-------------------------------------------------------
#include<array2d.h>

//default constructor
Array2D::Array2D(){
        for(int i=0; i<numPtr; i++)
                ptr[i] = NULL;
        row = column = 0;
}

//initial Array2D of n by m, and set each element to val
Array2D::Array2D(unsigned int n, unsigned int m, double val){
        int i,j;
        //initialize the pointer array to avoid loss pointers
        for (i=0; i<numPtr; i++)
                ptr[i] = NULL;
        row = n;
        column = m;
        for(i=0; i<row; i++){
                ptr[i] = new double[column]; //allocate memory
                for(j=0; j<column; j++)
                        ptr[i][j] = val;  //set each element to val
        }
}

//initial Array2D object to another Array2D object
Array2D::Array2D(const Array2D & a){
        int i,j;
        //initialize the pointer array to avoid loss pointers
        for (i=0; i<numPtr; i++)
                ptr[i] = NULL;

        row = a.row;
        column = a.column;
        for(i=0; i<row; i++){
                ptr[i] = new double[column]; //allocate memory
                for(j=0; j<column; j++)
                        ptr[i][j] = a.ptr[i][j];   //assign value to it
        }
```

```
}

//destructor
Array2D::~Array2D(){
        for (int i=0; i<numPtr; i++)
                delete []ptr[i];
}

//let user access element by index (assignment allowed)
double * Array2D::operator[](int i){
        //check index before continuing
        if(i<0 || i>=row){
                cerr<<"Error in array limits: "<< i
                        <<" is a bad row index\n";
                exit(1);
        }
        return ptr[i];
}

//let user access element by index (assignment disallowed)
const double * Array2D::operator[](int i) const{
        //check index before continuing
        if(i<0 || i>=row){
                cerr<<"Error in array limits: "<< i
                        <<" is a bad row index\n";
                exit(1);
        }
        return ptr[i];
}

//define class assignment
Array2D & Array2D::operator=(const Array2D & a){
        int i,j;

        if (this == &a) //if object assigned to self,
                return *this;   //don't change anything
        //free memory from old data and point to NULL
        for (i=0; i<numPtr; i++){
                delete [] ptr[i];
                ptr[i] = NULL;
        }

        row = a.row;
        column = a.column;
        for (i=0; i<row; i++){
                ptr[i] = new double[column];
                for(j=0; j<column; j++)
                        ptr[i][j] = a.ptr[i][j];
        }
        return *this;
}
```

```
//arraydb.h
//Define array class with size of row by column
//Based from: Stephen Prata, "C++ Primer Plus", 2nd Edition, 1995.
//Date: 09/20/1995
//-----------------------------------------------------------
#ifndef _ARRAYDB_H_
#define _ARRAYDB_H_
#include <iostream.h>

class ArrayDb {
	private:
		unsigned int size;			//number of array elements

	protected:
		double * arr;				//address of first element

	public:
		ArrayDb();			//default constructor
		//create an ArrayDb of n elements, set each to val
		ArrayDb(unsigned int n, double val = 0.0);
		//create an ArrayDb of n elements, initialized to array pn
		ArrayDb(const double* pn, unsigned int n);
		ArrayDb(const ArrayDb &a);		//copy constructor
		~ArrayDb();
	//destructor
		unsigned int arsize()const{return size;}		//return array size
	//overloaded operators
		double & operator[](int i);
		const double & operator[](int i)const;
		ArrayDb & operator=(const ArrayDb & a);
		friend ostream & operator<<(ostream & os, const ArrayDb & a);
};

#endif
```

```
//arraydb.cpp
//Class method for ArrayDb class
//Based from: Stephen Prata, "C++ Primer Plus", 2nd Edition, 1995.
//Date: 09/20/1995
//-------------------------------------------------------
#include <iostream.h>
#include <stdlib.h>              //exit()prototype
#include <arraydb.h>

//default constructor -- no arguments
ArrayDb::ArrayDb(){
        arr = NULL;
        size = 0;
}

//constructs array of n elements, each set to val
ArrayDb::ArrayDb(unsigned int n, double val){
        size = n;
        arr = new double[size];
        for (int i=0; i<size; i++)
                arr[i] = val;
}

//initialize ArrayDb object to a nonclass array
ArrayDb::ArrayDb(const double *pn, unsigned int n){
        arr = new double[n];
        size = n;
        for (int i=0; i<size; i++)
                arr[i] = pn[i];

}

//initialize ArrayDb object to another ArrayDb object
ArrayDb::ArrayDb(const ArrayDb & a){
        size = a.size;
        arr = new double[size];
        for (int i=0; i<size; i++)
                arr[i] = a.arr[i];
}

//destructor
ArrayDb::~ArrayDb(){
        delete [] arr;
```

```
}

//let user access elements by index (assignment allowed)
double & ArrayDb::operator[](int i){
        //check index before continuing
        if (i<0 || i>=size){
                cerr<<"Error in array limits: "
                                << i << " is a bad index\n";
                exit(1);
        }
        return arr[i];
}

//let user access element by index (assignment disallowed);
const double & ArrayDb::operator[](int i)const{
        //check index before continuing
        if (i<0 || i>=size) {
                cerr<<"Error in array limits: "
                                << i <<" is a bad index\n";
                exit(1);
        }
        return arr[i];
}

//define class assignment
ArrayDb & ArrayDb::operator=(const ArrayDb & a){
        if (this == &a) //if object assigned to self,
                return *this;   //don't change anything
        delete arr;
        size = a.size;
        arr = new double[size];
        for (int i=0; i<size; i++)
                arr[i] = a.arr[i];
        return *this;
}

//quick output, 5 values a line
ostream & operator<<(ostream & os, const ArrayDb & a){
        for (int i=0; i<a.size; i++){
                os << a.arr[i] <<"\n";
                if (i % 5 == 4)
                        os << "\n";
        }
        if (i % 5 !=0)
                os << "\n";
        return os;
}
```

```
//sort.h
//Header file for sorting functions
//Written by: Kurnia D. Iskandar
//Date: 10/29/1995
//--------------------------------------------------------
//a = a type double array of values.
//n = number of element in the array.
//DMin returns a minimum value of type double.
//DMax returns a maximum value of type double.

#ifndef _SORT_H_
#define _SORT_H_

double DMin(const double* a, const int n);
double DMax(const double* a, const int n);
#endif
```

```
//sort.cpp
//File definition for sorting functions
//Written by: Kurnia D. Iskandar
//Date: 10/29/1995
//-------------------------------------------------------
#include <sort.h>
double DMin(const double* a, const int n){
        double min;
        min = a[0];
        for (int i=1; i<n; i++){
                if (a[i] < min) min = a[i];
        }
        return min;
}
double DMax(const double* a, const int n){
        double max;
        max = a[0];
        for (int i=1; i<n; i++){
                if (a[i] > max) max = a[i];
        }
        return max;
}
```

```
//gausswp.h
//header file for gaussian elimination with partial pivoting
//Written by:Kurnia D. Iskandar
//Date: 09/26/1995
//-----------------------------------------------------------
#ifndef _GAUSSWP_H_
#define _GAUSSWP_H_

#include<array2d.h>
#include<arraydb.h>
#include<math.h>
#include<iostream.h>

// X =  n by n polynomial matrix
// Y = right hand matrix
// size = size of the matrix (n = size)

ArrayDb gausswp(Array2D & X , ArrayDb & Y);

#endif
```

```
//gausswp.cpp
//Function definition for gaussian with partial pivoting
//Based on: Stanley I. Grossman and William R. Derrick, "Advanced
//                                  Engineering Mathematics",
//Maximum array size can be solved is 10 by 10.
//To change the array size go to array2d.h file.
//-------------------------------------------------------
#include<gausswp.h>

ArrayDb gausswp(Array2D & X, ArrayDb & Y){
        int i, j, index;
        unsigned int size;
        double large, temp, pivot;

        size = X.arsize();
        //create row echelon form
        for (j=0; j<size; j++){
                if (j != (size-1)){                    //start of step 1-4
                        large = fabs(X[j][j]);                 //step 1
                        index = j;
                        for (i=j; i<size; i++){
                                if (fabs(X[i][j])>large){
                                        large = fabs(X[i][j]);
                                        index = i;
                                }
                        }
                        //swapping row of X matrix; step 2
                        for (i=j; i<size; i++){
                                temp = X[j][i];
                                X[j][i] = X[index][i];
                                X[index][i] = temp;
                        }
                        //swapping row of Y matrix; step 2
                        temp = Y[j];
                        Y[j] = Y[index];
                        Y[index] = temp;
                        //divide the pivot row with its diagonal element; step 3
                        pivot = X[j][j];
                        for (i=j; i<size; i++)
                                X[j][i] /= pivot;
                        Y[j] /= pivot;
                        //add multiple the first row to other rows; step 4
                        for (i=j+1; i<size; i++){
```

```
                    double m = X[i][j];
                    for (int l=j; l<size; l++)
                         X[i][l] += (-1)*m*X[j][l];
                    Y[i] += (-1)*m*Y[j];
               }
          }else if (j == (size-1)){
               double n = X[j][j];
               X[j][j] /= n;
               Y[j] /= n;
          }
     }//finish row echelon form
     //back substitution
     ArrayDb D(size);
     for (i=(size-1); i>=0; i--){
          if (i == (size-1)){
               D[i] = Y[i];
          }else if (i != (size-1)){
               double sum = 0.0;
               for (j=(size-1); j>=(i+1); j--)
                    sum += X[i][j] * D[j];
               D[i] = Y[i] - sum;
          }
     }
     return D;
}//end function definition
```

```
//leastqr.h
//function declaration
//least square method solved using gaussian with partial pivoting
//Written by: Kurnia D. Iskandar
//Date: 09/24/1995
//------------------------------------------------------
#ifndef _LEASTQR_H_
#define _LEASTQR_H_

#include<array2d.h>
#include<arraydb.h>
#include<gausswp.h>
#include<riseto.h>
#include<math.h>
#include<iostream.h>

//Function prototype least square method for polynomial
//functions solved by Gaussian with partial pivoting.
//Function returns an ArrayDb type data. The first element
//corresponds to the variable with the highest power.
// xval = pointer to an array of x values of type ArrayDb.
// yval = pointer to an array of y values of type ArrayDb.
// m = number of points
// n = degree of polynomial
// The highest degree of polynomial is 10; to change it, go
// to array2d.h file and change numPtr value.
ArrayDb leastqr(const ArrayDb & xval, const ArrayDb & yval,
                                        const int m, const int n);

#endif
```

```
//leastqr.cpp
//function definition of least square method solved by gaussian
//with partial pivoting
//information on variables can be found in header file
#include<leastqr.h>

ArrayDb leastqr(const ArrayDb & xval, const ArrayDb & yval,
                                                const int m, const int n){
        Array2D B(n+1, n+1);
        ArrayDb C(n+1);
        ArrayDb A;
        int i, j, k, r;
        double xsum, ysum;
        if (n > 10 || n < 0){
                cerr<<"Error, "<<n<<" is invalid degree polynomial.\n";
                cerr<<"Check numPtr in array2d.h file and n in leastqr.h file.";
                exit(1);
        }
        for (i=0; i<=n; i++){
                for (j=0; j<=n; j++){
                        xsum = 0.0;
                        r = (2 * n) - (i + j);
                        for (k=0; k<m; k++){
                                xsum += riseto(xval[k],double(r));
                        }
                        B[i][j] = xsum;
                }
        }
        for (i=0; i<=n; i++){
                ysum = 0.0;
                for (k=0; k<m; k++){
                        ysum += yval[k] * riseto(xval[k],double(n-i));
                }
                C[i] = ysum;
        }
        //call gaussian function and assigned to A
        A = gausswp(B, C);
        return A;
}
```

```
//point2d.h
//File to declaration for 2D points
#ifndef _POINT2D_H_
#define _POINT2D_H_
#include <iostream.h>

class point2d {
        //no private members
        public:
        //public data
                double x, y;

                point2d(){}     //default constructor
                point2d(const point2d & a); //copy constructor
                ~point2d(){} //destructor
        //Member functions
                point2d & operator=(const point2d & a); //assignment operator
        //Friend function
                friend ostream & operator<<(ostream & os, const point2d & a);
};
#endif
```

```
//point2d.cpp
//Method defintion for 2D points
#include <point2d.h>

//copy constructor
point2d::point2d(const point2d & a){
        x = a.x; y = a.y;
}

//Assignment operator
point2d& point2d::operator=(const point2d & a){
        if (this == &a)
                return *this;
        x = a.x; y = a.y;
        return *this;
}

ostream & operator<<(ostream & os, const point2d & a){
        os<<a.x<<" "<<a.y;
        return os;
}
```

```
//polyderv.h
//Header file for derivative of polynomial function at a given
//value.
//Written by: Kurnia D. Iskandar
//Date: 10/28/1995
//-------------------------------------------------------
//v = a value to be evaluated.
//a = an array of polynomial function coefficients; a[0] corres-
//   ponds to coefficient of variable with highest degree,...,
//   and a[n] corresponds to a constant variable.
//n = degree of polynomial function.
//p = order of derivative of the polynomial function.
//Function returns a type double value.
#ifndef _POLYDERV_H_
#define _POLYDERV_H_
#include<riseto.h>
#include<arraydb.h>

double polyderv(const double v, const ArrayDb &a, int n, int p);

#endif
```

```
//polyderv.cpp
//Function definition
//Written by: Kurnia D. Iskandar
//Date: 10/28/1995
//--------------------------------------------------------
#include <polyderv.h>
double polyderv(const double v, const ArrayDb &a, int n, int p){
        double sum ;
        double d;

        sum = 0.0;
        for (int k=0; k<=(n-p); k++){
                d = 1.0;
                if (k < (n-p)){
                        for (int i=0; i<p; i++) d *= (double)(n-k-i);
                        sum += a[k] * d * riseto(v,(double)(n-k-p));
                }
                else if (k == (n-p)){
                        sum += a[k];
                }
        }
        return sum;
}
```

```
//polyintg.h
//Header file for numerical integration of polynomial function,
//given the min and max values.
//Written by: Kurnia D. Iskandar
//Date: 10/27/1995
//------------------------------------------------------
//a = polynomial coeficients; ArrayDb data type; a[0] corresponds
//   to coefficient for the highest degree variable, a[1] is
//   for the next variable, so on, and a[n] is a constant.
//n = degree of polynomial.
//min = minimum boundary value.
//max = maximum boundary value.
//The function return a type double.

#ifndef _POLYINTG_H_
#define _POLYINTG_H_
#include <arraydb.h>
#include <riseto.h>
double polyintg(const ArrayDb &a, const int n, const double min, const double max);
#endif
```

```
//polyintg.cpp
//Function definition for numerical integration of polynomial function.
#include <polyintg.h>

double polyintg(const ArrayDb &a, const int n, const double min, const double max){
        double sum1;
        double sum2;
        double f;

        sum1 = sum2 = 0.0;
        for (int i=n; i>=0; i--){
                sum1 += a[n-i]*riseto(max,(double)(i+1))/(double)(i+1);
                sum2 += a[n-i]*riseto(min,(double)(i+1))/(double)(i+1);
                //cout<<sum1<<" "<<sum2<<"\n";
        }
        f = sum1 - sum2;
        return f;
}
```

```
//riseto.h
//file declaration for power function
//a is rised to b, and function returns double
//Written by:Kurnia D. Iskandar
//Date: 09/27/1995
//-----------------------------------------------------
#ifndef _RISETO_H_
#define _RISETO_H_

#include<iostream.h>
#include<math.h>

double riseto(double a, double b);

#endif
```

```
//riseto.cpp
//file definition for power function
//Written by:Kurnia D. Iskandar
//Date: 09/27/1995
//-------------------------------------------------------
#include<riseto.h>

double riseto(double a, double b){
	double temp, output;

	if (a == 0.0 && b != 0.0){
		output = 0.0;
	}else if (a == 0.0 && b == 0.0){
		output = 1.0;
	}else if (a < 0.0 && int(b)%2 == 0){
		temp = b * log(-1*a);
		output = exp(temp);
	}else if (a < 0.0 && int(b)%2 != 0){
		temp = b * log(-1*a);
		output = -1 * exp(temp);
	}else {
		temp = b * log(a);
		output = exp(temp);
	}
	return output;
}
```

```
//sprocket.h
//Mechanical sprocket class declaration
//Written by: Kurnia D. Iskandar
//Date: 10/27/1995
//-------------------------------------------------------
#ifndef _SPROCKET_H_
#define _SPROCKET_H_
#include <math.h>
class Sprocket {
	private:
		double radius; //Sprocket radius.
		double pitch;  //Link distance or pin separation for
		                                              //ANSI type 40 chain.
		int teeth;     //Number of sprocket teeth.

		double findRadius(double p, int t);
	public:
		Sprocket(){}   //default constructor
		//Create and initialize an object. Input: pin separation (p)
		//and number of teeth (t).
		Sprocket(double p,int t);
		~Sprocket(){} //default constructor

	//Member functions
		//Return sprocket radius. Input: pin separation(p) and
		//number of teeth(t).
		double Radius(double p, int t); //return the sprocket radius.
		int numTeeth(){return teeth;} //return the number of sprocket teeth.
	//Overloaded operator
};
#endif
```

```
//sprocket.cpp
//Class method definition for sprocket class
//Written by: Kurnia D. Iskandar
//Date: 10/29/1995
//------------------------------------------------------
#include <sprocket.h>

//Create and initialize an object
Sprocket::Sprocket(double p, int t){
	pitch = p; teeth = t;
}

//Private class method definition
double Sprocket::findRadius(double p, int t){
	return radius = p / (2 * sin(M_PI/(double)t));
}

//Public class member definition
double Sprocket::Radius(double p, int t){
	pitch = p; teeth = t;
	return radius = findRadius(pitch, teeth);
}
```

VITA

Name: Kurnia Dias Iskandar

Date and Place of Birth: 6 April 1970, Bogor

Home Country: INDONESIA

Educational Background:

- *Master of Science in Mechanical Engineering,* June 1996, Virginia Polytechnic Institute and State University (VPI & SU), Blacksburg, Virginia, USA.

- *Bachelor of Science in Mechanical Engineering,* May 1993, West Virginia Institute of Technology, Montgomery, West Virginia, USA.

Signature: [illegible]

Kurnia Dias Iskandar