UNIVERSITY OF CALIFORNIA,
IRVINE

A CAD Add-in for Synthesis of RSSR Function Generators

PROJECT REPORT

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Mechanical and Aerospace Engineering

by

Julius Klein

Faculty Advisor:
Professor J. Michael McCarthy

2005

Contents

List of Tables

List of Figures

Nomenclature

Roman symbols

a	A real number.
$\mathbf{v}$	A three-dimensional vector.
$\mathbf{P}$	A three-dimensional point.
$[M]$	A matrix.
$\hat{z}$	The skew symmetric matrix that represents the three-dimensional vector $\mathbf{z}$.
$\bar{\mathbf{v}}$	The homogeneous coordinates of a three-dimensional vector.
$\bar{\mathbf{P}}$	The homogeneous coordinates of a three-dimensional point.
$\{\mathrm{F}\}$	A coordinate frame.
$\times$	The cross product of vectors.
$\cdot$	The dot product of vectors.
$\|\mathbf{v}\|$	The norm of a vector.

Acknowledgments

In first place I would like to express my gratitude with a special thanks to Pete Balsells. Without his generosity this project would not exist. The Balsells Fellowship has made many people's dreams come true. This program is alive because of the effort of Pete, the Generalitat de Catalunya, the University of California and the coordination of Professor Roger Rangel.

In second place I would like to thank the National Science Foundation for supporting this project.

I would also like to thank Dr. McCarthy for his support and guidance and Dr. Jabbari and Dr. Reinkensmeyer for being on my committee.

A big thanks to the components of the Robotics Lab and specially Dr. Haijun Su for sharing his knowledge and tricks regarding Visual Basic programming and Dr. Alba Perez for the hours we spent discussing robot synthesis.

In the other side of the Atlantic Ocean, there is a place where I used to live. I want to thank my friends in Catalunya for their support and understanding.

Thank you to my parents for their understanding and their love. It's not easy to accept that the bird has to leave the nest some day. I love you to!

Abstract

A CAD Add-in for Synthesis of RSSR Function Generators

by

Julius Klein

Master of Science in Mechanical and Aerospace Engineering

University of California, Irvine, 2005

Professor J. Michael McCarthy, Chair

In this project we consider the implementation of a CAD Add-in for the synthesis of an RSSR function generator. A CAD add-in is software that uses the Application Programmers Interface (API) of a computer-aided design system to integrate a stand-alone algorithm into the host program. Our implementation uses the API of SolidWorks, but the same strategy can be used to integrate linkage synthesis routines into any CAD system. We demonstrate the formulation of an Add-in by integrating the synthesis algorithm for an RSSR function generator to be used as a steering linkage.

Chapter 1 describes the kinematics of the RSSR mechanism. The creation of an add-in is introduced in Chapter 2. Chapter 3 describes the RSSR function generator. Chapter 4 illustrates how geometric data is retrieved from the CAD software. Chapter 5 explains the modeling of the RSSR linkage. Chapter 6 illustrates the creation of a toolbar and menu item. Conclusions and future work are found in Chapter 7. A complete example is presented in Chapter 8.

Chapter 1

Kinematics Overview

1.1 Introduction

In this chapter we formulate the synthesis of an RSSR function generator for use as a vehicle steering linkage design tool integrated into a standard computer-aided-design package in the form of a CAD Add-in.

The spatial four bar linkage constructed from two fixed revolute, or hinged, joints and two floating spherical, or ball, joints is called an RSSR linkage, Fig. 1.1.

Our implementation of the RSSR function generator inverts the linkage to transform the angular information into positions of the coupler of an RRSS linkage. The RR crank is specified and we design the SS crank to provide the desired coordination. See Hunt (1978) [1] and McCarthy (2000) [2] for the details of SS crank synthesis.

Figure 1.1: RSSR mechanism.

1.2 Literature Review

The synthesis and analysis of an RSSR steering mechanism is presented in Simionescu et al. (2000) [3]. Other work on steering linkages can be found in Simionescu and Beale (2002) [4] who study the Ackermann steering linkage, and Dvali and Aleksishvili (1971) [5] who consider a spherical four-bar linkage to design an automobile steering gear. A general study of the RSSR mechanism is presented by Molian (1973) [6] as well as Duffy and Gilmartin (1978) [7].

1.3 Kinematics Review

In this section we will review the main concepts of linkage kinematics. For more detailed explanations, please refer to Suh et al. (1978) [8], Sandor (1984) [9], McCarthy (1990) [10], Murray et al. (1994) [11]), Tsai (1999) [12], Dukkipati (2001) [13] and Waldron (2003) [14].

1.3.1 Frame Coordinate Systems

Kinematics is the branch of mechanics that studies the geometric properties of the motion of points, and is not concerned with forces caused or affected by the motion.

A linkage or mechanism is a machine composed by rigid members joined together. The joints constrain the relative movement between links.

When studying the motion of mechanisms, we study the position of a body relative to another. To do so, we attach coordinate frames to each body. Points on one body can be located with respect to any existing frame in the mechanism using coordinate transformations. Usually we set a fixed reference frame, called world or ground frame, and we refer the other bodie's frames to it.

Consider a frame $\{M\}$ defined in a rigid body (Fig. 1.2). Let $\{F\}$ be a fixed frame or ground frame. Let $\mathbf{P}$ be a point in the body. Let $\mathbf{p}_m = (p_{x1}, p_{y1}, p_{z1})^T$ be the vector coordinates of $\mathbf{P}$ expressed in $\{M\}$. The coordinate transformation from $\{M\}$ to $\{F\}$ is

$$\mathbf{p}_f = [R]\mathbf{p}_m + \mathbf{d}, \tag{1.1}$$

where $\mathbf{p}_f = (p_{x2}, p_{y2}, p_{z2})^T$ are the vector coordinates of $\mathbf{P}$ in $\{F\}$, $[R]$ and $\mathbf{d}$ are the rotation matrix and translation vector of $\{M\}$ with respect to $\{F\}$, respectively.

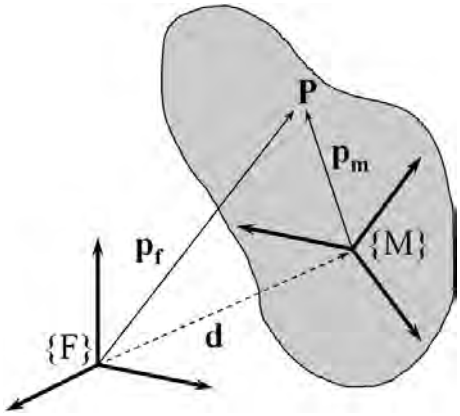

Figure 1.2: Frame coordinate transformation.

Rigid body rotations must satisfy the following conditions:

- The distance between points in the same body is preserved.

- The orientation of the reference frame is preserved .

These conditions lead to two fundamental properties for rotation matrices, $[R]$. First, the columns of $[R]$ must be orthonormal so that distance between two points is preserved. This property implies that

$$[R][R]^T = I, \tag{1.2}$$

which leads to the statement that the inverse of the rotation matrix is the same as its transpose. Second, to preserve the orientation of the body frame, the rotation matrix must be right-handed. This is expressed as

$$det([R]) = +1. \tag{1.3}$$

If the determinant is -1, the transformation is called a *reflection*.

The rotation matrix about a generic axis $\mathbf{v}$ and rotation angle θ can be calculated using

$$[R(\mathbf{v}, \theta)] = I + \hat{v}\sin\theta + \hat{v}^2(1 - \cos\theta), \tag{1.4}$$

where I is the three dimensional identity matrix and $\hat{v}$ is the skew symmetric matrix corresponding to the vector $\mathbf{v} = (v_x, v_y, v_z)^T$, defined as

$$\hat{v} = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}. \tag{1.5}$$

In a rotation about the x-axis, $\mathbf{v} = (1, 0, 0)^T$ and the rotation matrix is

$$[R_x(\theta)] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}. \tag{1.6}$$

Similarly, the rotation about the y-axis has the form of

$$[R_y(\theta)] = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}. \tag{1.7}$$

The rotation about the z-axis results in

$$[R_z(\theta)] = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{1.8}$$

1.3.2 Homogeneous Transformations

In 1955, J. Denavit and R. S. Hartenberg [15] first proposed the use of matrices to represent the articulations of a mechanism. These matrices, called *Homogeneous Transforms*, describe the coordinate transformation from $\{M\}$ to $\{F\}$. The coordinate transformation described by Eqn. (1.1) is now expressed as

$$\bar{\mathbf{p}}_f = [T]\bar{\mathbf{p}}_m, \tag{1.9}$$

where $\bar{\mathbf{p}} = (p_x, p_y, p_z, 1)^T$ and $[T]$ is the 4×4 homogeneous representation of the displacement described in Eqn. (1.1), and is expressed as

$$[T] = \begin{bmatrix} [R] & \mathbf{d} \\ 0 & 1 \end{bmatrix}. \tag{1.10}$$

Recall that $[R]$ is a 3×3 rotation matrix and $\mathbf{d}$ is a three-dimensional column vector, therefore the zero located in the lower left position represents the three-dimensional row zero vector.

Homogeneous transformation matrices also preserve distances between rotated points and are right handed. The inverse of a 4×4 homogeneous transformation matrix is built as

$$\begin{aligned} [T]^{-1} &= \begin{bmatrix} [R]^{-1} & -[R]^{-1}\mathbf{d} \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} [R]^T & -[R]^T\mathbf{d} \\ 0 & 1 \end{bmatrix}. \end{aligned} \tag{1.11}$$

It can be easily proved that $det([T]) = 1$ and $[T][T]^{-1} = I$, where I is the 4×4 identity matrix.

We will denote $[Z(\theta, d)]$ as the screw displacement matrix corresponding to a rotation about the z-axis and a translation d in the direction of the z-axis. Similarly

we will denote $[X(\theta, d)]$ when the rotation is about the x-axis and the translation d is in the direction of the x-axis.

For serial chains, we can define the vector coordinates of the end-effector with respect to a ground frame by computing the composite transformation matrix as the product of consecutive matrix transformations associated to each link (Fig. 1.3). We must take into account that the product of transformation matrices is not commutative. When the displacement is referenced to the coordinate axes of a moving frame, the composite transformation is the result of the postmultiplication of the matrix. Otherwise, when referenced to the coordinate axes of a fixed frame, the matrix is premultiplied.

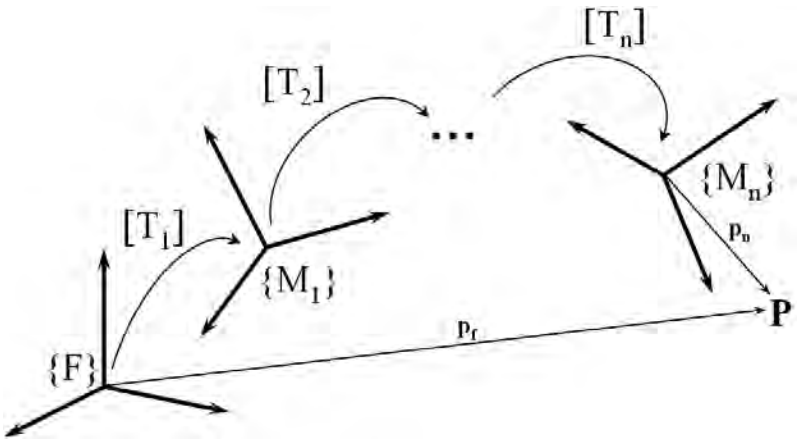

Figure 1.3: Composite frame coordinate transformation.

The vector coordinates of the end-effector in the fixed frame are determined using

$$\begin{aligned} \bar{\mathbf{p}}_f &= [T]\bar{\mathbf{p}}_n \\ &= [T_1][T_2]...[T_n]\bar{\mathbf{p}}_n, \quad i = 1, ..., n. \end{aligned} \tag{1.12}$$

The kinematics equation of a serial chain defines the set of all positions reachable by the end-effector. This set of positions is also known as the *workspace*. We write the kinematics equation as

$$[T(\vec{\theta})] = [G][K(\vec{\theta})][H], \tag{1.13}$$

where $[G]$ locates the first link relative to the base (world frame) and $[H]$ locates the end-effector relative to the last link (tool frame), both fixed. The matrix $[K(\vec{\theta})]$ locates the last link relative to the first one and is expressed in terms of the joint parameters, $\vec{\theta}$. Following the Denavit and Hartenberg convention, $[K(\vec{\theta})]$ is expressed as

$$[K(\vec{\theta})] = [Z(\theta_1, d_1)][X(\alpha_{12}, a_{12})][Z(\theta_2, d_2)] \dots [X(\alpha_{n-1,n}, a_{n-1,n})][Z(\theta_n, d_n)]. \quad (1.14)$$

1.3.3 Relative Transformations

We can describe the kinematics equation with respect to a reference position $[T_0]$. The displacement $[T_i]$ relative to this position is

$$\begin{aligned}
[T_{0i}] &= [T_i][T_0]^{-1} \\
&= ([G][Z(\theta_{1i}, d_{1i})] \dots [Z(\theta_{ni}, d_{ni})][H])([G][Z(\theta_{10}, d_{10})] \dots [Z(\theta_{n0}, d_{n0})][H])^{-1} \\
&= [T(\Delta\theta_1, \mathsf{S}_1)] \dots [T(\Delta\theta_n, \mathsf{S}_n)], \qquad (1.15)
\end{aligned}$$

where

$$\begin{aligned}
[T(\Delta\theta_1, \mathsf{S}_1)] &= [G][Z(\theta_{1i}, d_{1i})][Z(\theta_{10}, d_{10})]^{-1}[G]^{-1}, \\
[T(\Delta\theta_2, \mathsf{S}_2)] &= ([G][Z(\theta_{10}, d_{10})][X(\alpha_{12}, a_{12})][Z(\theta_{2i}, d_{2i})]) \\
&\qquad\qquad ([G][Z(\theta_{10}, d_{10})][X(\alpha_{12}, a_{12})][Z(\theta_{20}, d_{20})])^{-1}, \\
&\vdots \\
[T(\Delta\theta_n, \mathsf{S}_n)] &= ([G][[Z(\theta_{10}, d_{10})] \dots)[Z(\theta_n, d_n)][Z(\theta_{n0}, d_{n0})]^{-1}([G][[Z(\theta_{10}, d_{10})] \dots)^{-1}. \qquad (1.16)
\end{aligned}$$

The displacements $[T(\Delta\theta_i, \mathsf{S}_i)]$ are the relative rotations and translations along the joint axes of the chain from the chosen reference configuration.

1.4 Kinematics of the RSSR Linkage

The RSSR linkage is often used to drive a wheel in which one of the revolute joints is known as the king-pin. The king-pin is angled to provide specific performance

of the vehicle during turning movements. We assume the other revolute axis has been specified as well, and our goal is to locate the two spherical joints such that the rotation around these axes are coordinated. This is known as the synthesis of an RSSR function generator.

In order to formulate this problem, we assume the two fixed revolute joints are given, where the axis of the input R-joint passes through the point $\mathbf{O}$ in the direction $\mathbf{z}_1$, and the axis of the output R-joint passes through $\mathbf{C}$ in the direction $\mathbf{z}_2$.

In addition, we assume that the ball joint of the input crank is given in a reference position, that is $\mathbf{S}_1$.

Finally, we assume that we are given a direction $\mathbf{x}_2$ along the line $\mathbf{P}_{\text{ref}} - \mathbf{C}$, where $\mathbf{P}_{\text{ref}}$ is a reference point used to measure the output angle.

We define the ground frame F located at $\mathbf{O}$ and let this frame be fixed. Its its z-axis in the direction $\mathbf{z}_1$ and the x-axis is normal to $\mathbf{z}_1$ and lies in the plane generated by the vector $\mathbf{z}_1$ and a vector in the direction of the line $\mathbf{O} - \mathbf{C}$. Figure 1.4 shows the reference frame in 2 views: a front view and a special view in the direction of $\mathbf{z}_1$.

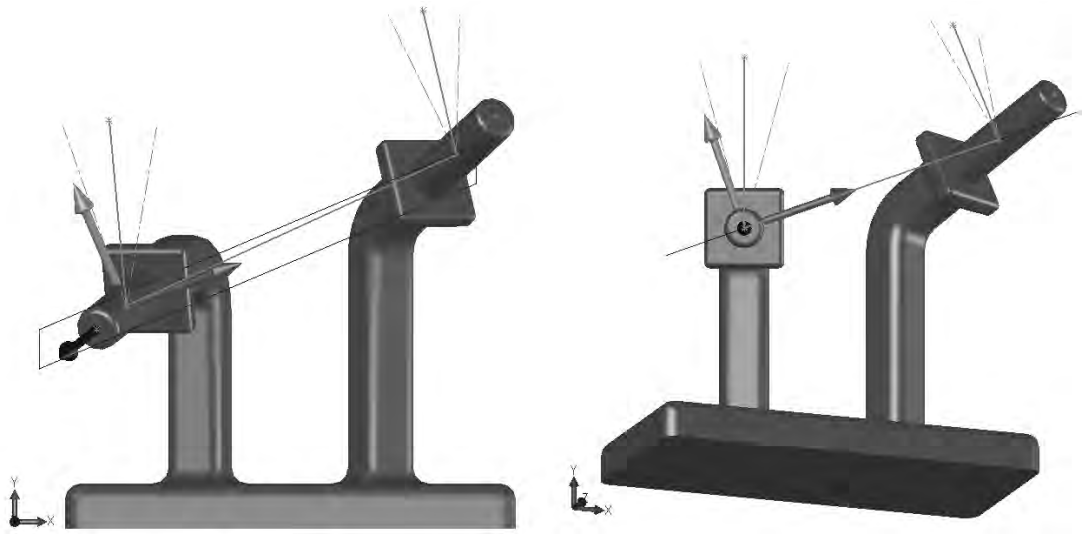

Figure 1.4: Reference frame.

This allows us to define a frame $\{B_1\}$ with its origin $\mathbf{O}$, with $\mathbf{z}_1$ as its z-axis and the unit vector $\mathbf{x}_1$ in the direction $\mathbf{S}_1 - \mathbf{O}$ as its x-axis. This frame rotates about

$\mathbf{z}_1$ an angle α, measured with respect to the x-axis in $\{F\}$, (Fig. 1.5).

We also define a frame $\{B_2\}$ located at $\mathbf{C}$, with its x-axis in the direction $\mathbf{x}_2$ and its z-axis in the direction $\mathbf{z}_2$.

The connecting rod joins the ball joint point $\mathbf{S}_1$ and the ball joint $\mathbf{S}_2$, which our algorithm will compute. The distance from $\mathbf{O}$ to $\mathbf{S}_1$ is r_1 and from $\mathbf{C}$ to $\mathbf{S}_2$ is r_2. The length of the connecting rod is r_3.

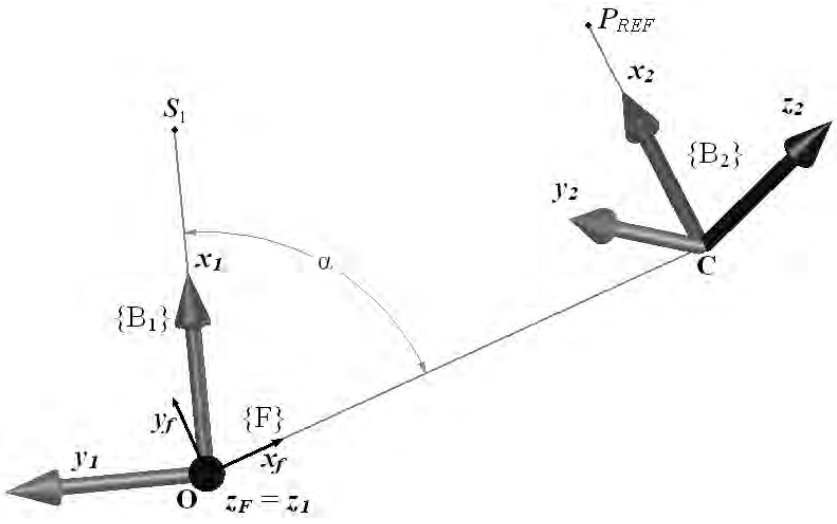

Figure 1.5: Defining frames.

We now derive the synthesis equations that allow us to determine the coordinates of $\mathbf{S}_2$ given the set of specified coordinate crank angles. In our method, θ is measured relative to the line $\mathbf{S}_1 - \mathbf{O}$, therefore, we can define $\theta_i = \alpha + \alpha_i$, (Fig. 1.6). Similarly, we define the output angle ϕ to be relative to the line $\mathbf{P}_{ref} - \mathbf{C}$.

We note $\mathbf{S}_{i,j}$ as the j-th position of $\mathbf{S}_i$ ($i = 1$ corresponds to the input crank's ball joint, and $i = 2$ to the output crank). For $\theta = 0$, the input crank's ball joint point is located at $\mathbf{S}_{1,0}$.

Using Eqn. (1.9) we define the location of $\mathbf{S}_1$ and $\mathbf{S}_2$ in two positions relative to $\mathbf{S}_{1,0}$, $\mathbf{P}_{ref}$ with

$$\bar{\mathbf{S}}_{1,i} = r_1[T_{1,i}]\bar{\mathbf{x}}_1, \quad i = 1, 2 \tag{1.17}$$

$$\bar{\mathbf{S}}_{2,i} = r_2[T_{2,i}]\bar{\mathbf{x}}_2, \quad i = 1, 2, \tag{1.18}$$

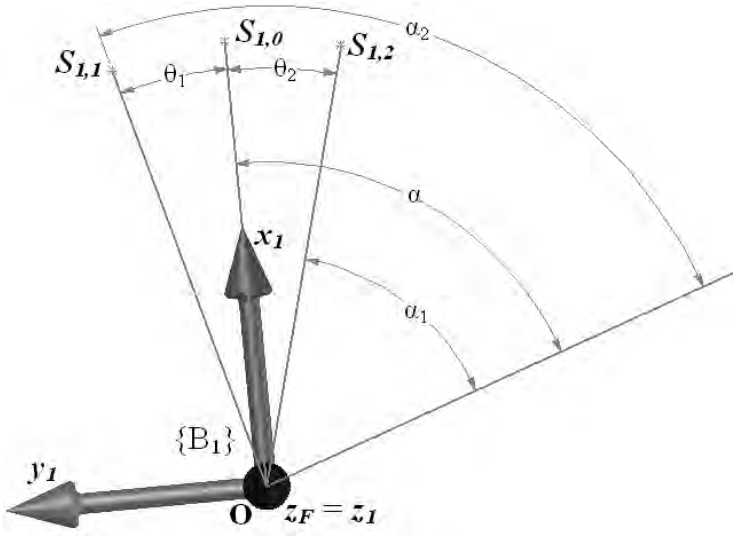

Figure 1.6: Defining relative angles.

where $[T_1]$, $[T_2]$ are the screw displacement matrices corresponding to

$$[T_1] = \begin{bmatrix} [R_1] & \mathbf{O} \\ 0 & 1 \end{bmatrix}$$
$$[T_2] = \begin{bmatrix} [R_2] & \mathbf{C} \\ 0 & 1 \end{bmatrix}, \tag{1.19}$$

where $[R_1]$ is a rotation by an angle θ about $\mathbf{z}_1$, and $[R_2]$ a rotation by an angle ϕ about $\mathbf{z}_2$, both determined using Eqn. (1.4), resulting in

$$[R_1(\theta_1)] = I + \hat{z}_1 \sin\theta_1 + \hat{z}_1{}^2(1 - \cos\theta_1),$$
$$[R_2(\phi_1)] = I + \hat{z}_2 \sin\phi_1 + \hat{z}_2{}^2(1 - \cos\phi_1). \tag{1.20}$$

Given r_1, $\mathbf{x}_1$, $\mathbf{z}_1$ and θ, we determine $\mathbf{S}_{1,i}$ and $\mathbf{S}_{2,i}$ by using Eqn. (1.17), which expands as

$$\mathbf{S}_{1,i} = \mathbf{O} + r_1 \cos\theta_i \mathbf{x}_1 + r_1 \sin\theta_i \mathbf{y}_1 \quad i = 1, 2. \tag{1.21}$$

Notice that $\mathbf{O}$ is the origin of the frame defined by $\mathbf{x}_1$,$\mathbf{y}_1$ and $\mathbf{z}_1$.

Similarly, we obtain the position of the second ball joint using

$$\mathbf{S}_{2,i} = \mathbf{C} + r_2 \cos\phi_i \mathbf{x}_2 + r_2 \sin\phi_i \mathbf{y}_2 \quad i = 1, 2. \tag{1.22}$$

where $\mathbf{C}$ is the origin of the frame defined by $\mathbf{x}_2$,$\mathbf{y}_2$ and $\mathbf{z}_2$.

Since the length of SS dyad (the distance between $\mathbf{S}_1$ and $\mathbf{S}_2$) remains constant for all positions, we can define a *constraint equation* for each i-th position. For the two given relative positions, we obtain the following system of constraint equations

$$\begin{aligned}(\mathbf{S}_{1,1} - \mathbf{S}_{2,1}) \cdot (\mathbf{S}_{1,1} - \mathbf{S}_{2,1}) &= r_3^2 \\ (\mathbf{S}_{1,2} - \mathbf{S}_{2,2}) \cdot (\mathbf{S}_{1,2} - \mathbf{S}_{2,2}) &= r_3^2,\end{aligned} \tag{1.23}$$

where r_3 is unknown. We reduce the number of unknowns by subtracting the elements in Eqn. (1.23) to obtain the *design equation*, defined as

$$(\mathbf{S}_{1,1} - \mathbf{S}_{2,1}) \cdot (\mathbf{S}_{1,1} - \mathbf{S}_{2,1}) - (\mathbf{S}_{1,2} - \mathbf{S}_{2,2}) \cdot (\mathbf{S}_{1,2} - \mathbf{S}_{2,2}) = 0. \tag{1.24}$$

Solving the design equation for r_2 we can fully define the location of $\mathbf{S}_2$ for any position.

1.5 Steering Linkage Synthesis

For the steering linkage design case, the three positions coincide with the straight trajectory of a wheel (0), right turn (1) and left turn (2). The left and right turn are defined with a relative angle with respect to the straight position.

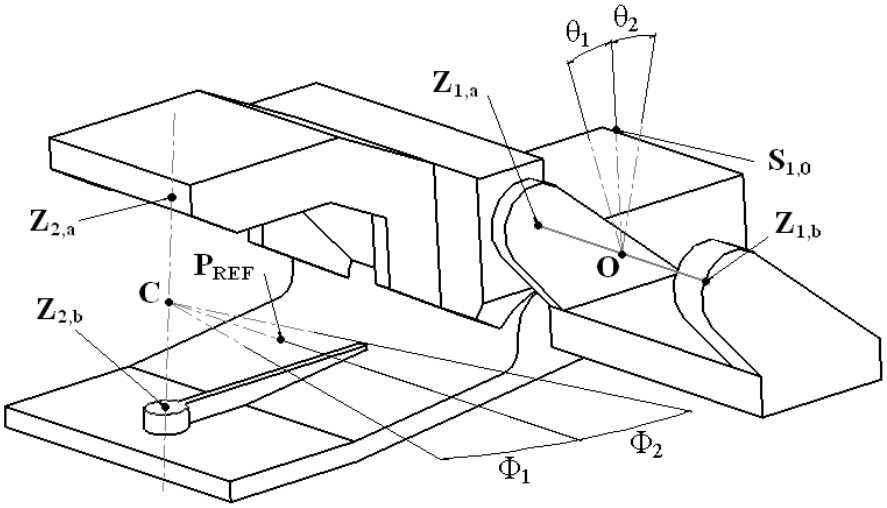

Figure 1.7: Specified geometry.

Our function generator can solve for any relative positions between the two rotating axes. It can also solve for any desired relation of angles between the input and output angle. This allows the designer to specify a different turning behavior for the wheels during a right turn or a left turn. The location of the input data points and angles is shown in the modeled supporting part in Fig. 1.7. This figure shows the support for left RSSR steering linkage, the right mechanism is a mirror image of this model part.

We define the input and output axes ($\mathbf{z}_1$,$\mathbf{z}_2$) with a pair of points for each axis (see Table 1.1).

Table 1.1: Axes definition.

vector	point 1	point 2
$\mathbf{z}_1$	$\mathbf{Z}_{1,a}$	$\mathbf{Z}_{1,b}$
$\mathbf{z}_2$	$\mathbf{Z}_{2,a}$	$\mathbf{Z}_{2,b}$

We start defining the normalized frame vectors $\mathbf{x}_1$, $\mathbf{y}_1$ and $\mathbf{z}_1$ for the input crank, given by

$$\begin{aligned}\mathbf{x}_1 &= (\mathbf{S}_{1,0} - \mathbf{Z}_{1,a}) - \mathbf{z}_1(\mathbf{S}_{1,0} - \mathbf{Z}_{1,a}) \cdot \mathbf{z}_1,\\ \mathbf{y}_1 &= \mathbf{z}_1 \times \mathbf{x}_1,\\ \mathbf{z}_1 &= \mathbf{Z}_{1,a} - \mathbf{Z}_{1,b}.\end{aligned} \tag{1.25}$$

Notice that we compute $\mathbf{z}_1$ first, then $\mathbf{x}_1$ and $\mathbf{y}_1$. These vectors must be normalized.

Point O is calculated using

$$\mathbf{O} = \mathbf{Z}_{1,a} + \mathbf{z}_1((\mathbf{S}_{1,0} - \mathbf{Z}_{1,a}) \cdot \mathbf{z}_1). \tag{1.26}$$

The length of input crank, r_1, is the value we will later use to resize our library model part. It is calculated as

$$r_1 = \frac{\|(\mathbf{Z}_{1,b} - \mathbf{Z}_{1,a}) \times (\mathbf{Z}_{1,a} - \mathbf{S}_{1,0})\|}{\|(\mathbf{Z}_{1,b} - \mathbf{Z}_{1,a})\|}. \tag{1.27}$$

Solving Eqn. (1.27) we determine the two relative positions of the input ball joint $\mathbf{S}_1$ using Eqn. (1.21). These two relative positions are referred to the case $\theta = 0$ and $\phi = 0$.

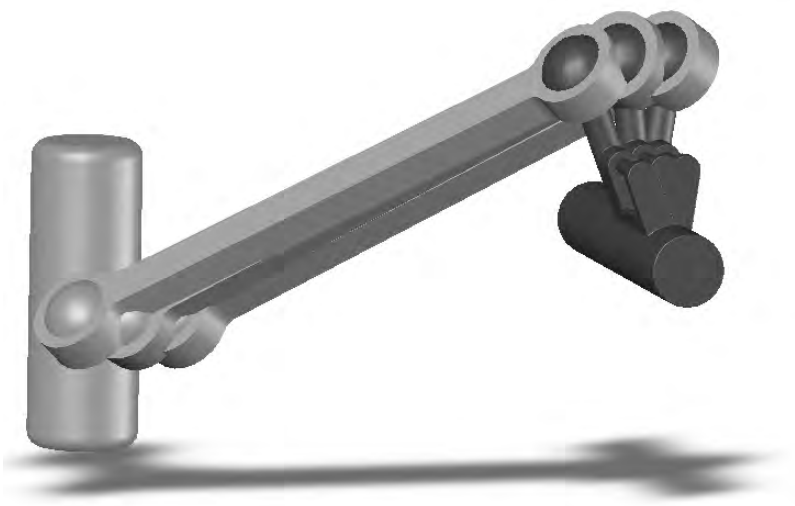

Figure 1.8: Three positions superimposed.

To determine $\mathbf{S}_2$ we need to solve r_2. To do so, we start defining the frame of the output crank. The rotation axis $\mathbf{z}_2$ is defined by the two points $\mathbf{Z}_{2,b}$ and $\mathbf{Z}_{2,a}$, after normalizing. Furthermore, $\mathbf{x}_2$ is determined using

$$\mathbf{x}_2 = (\mathbf{P}_{ref} - \mathbf{Z}_{2,a}) - \mathbf{z}_2(\mathbf{P}_{ref} - \mathbf{Z}_{2,a}) \cdot \mathbf{z}_2, \tag{1.28}$$

and frame vector $\mathbf{y}_2$ is orthonormal to $\mathbf{z}_2$ and $\mathbf{x}_2$. We calculate $\mathbf{C}$, the origin of this second frame using

$$\mathbf{C} = \mathbf{Z}_{2,a} + \mathbf{z}_2((\mathbf{P}_{ref} - \mathbf{Z}_{2,a}) \cdot \mathbf{z}_2). \tag{1.29}$$

Solving Eqn. (1.22) for the two relative positions and substituting the result into Eqn. (1.24) we finally determine r_2. Once the positions are defined, we calculate r_3 using

$$r_3 = \sqrt{(\mathbf{S}_1 - \mathbf{S}_2) \cdot (\mathbf{S}_1 - \mathbf{S}_2)}. \tag{1.30}$$

Note that since the length of the SS dyad is set to be constant, we can use any of the three positions in Eqn. (1.30).

The three positions of the RSSR linkage are shown superimposed in Fig. 1.8.

Chapter 2

Creating an Add-in

2.1 Introduction

The mechanism synthesis process usually takes place on the drafting board. This process is difficult and time consuming. The modeling process takes place in a computerized environment. The purpose of this project is to develop a tool that will make designers' work more efficient. Our goal is to transport the synthesis stage from the drafting board to the same environment where the modeling takes place. This is done by creating an add-in and making it accessible from toolbar icon or a menu in the modeling software.

In this chapter we describe the key elements for a CAD add-in user interface. We start by making a distinction between Macros and Add-ins, we continue noting the interaction elements needed to communicate between the add-in and the host software. We also describe the add-in's graphical user interface.

2.2 Macros vs Add-ins

A macro is a set of keystrokes and instructions that are recorded, saved, and assigned to a short key code. Macros can usually be created without writing any programming code. When the key code is typed, the recorded keystrokes and instructions execute (play back). A macro does not add new functionality but it can simplify day-to-day operations, which otherwise become tedious.

An add-in is a small program which runs in conjunction with another application that enhances the functionality of that program. In order for the add-in to run, the main application must be running as well. Usually, all add-in products are fully integrated into the host software providing users with access to a specific toolbar, pull down menus and property sheets.

In our case study, we are creating a CAD software add-in. Our program adds new synthesizing functionality to the CAD software environment and is accessible from a toolbar or icon within the host software.

2.3 Add-in for a CAD software

The integration of specialized numerical routines into CAD software uses the Software Development Kit (SDK) provided with the software. The SolidWorks SDK contains all of the functionality needed to develop SolidWorks API applications. Wizards for creating SolidWorks add-ins are also included. This API SDK is available on the SolidWorks installation CD under the API folder or online at the SW API Support site [16]. Without the SDK we won't have access to the code that allows the connection between the add-in and the host. For our implementation the synthesis algorithm is written in Visual Basic, using the Windows Application (COM) template from Microsoft's Visual Studio.NET 2003. This project is composed by the files described in Table 2.1. The architecture of the RSSR function generator add-in is shown in Fig. 2.1.

The API contains the functions that the programmer can call from Visual Basic 6.0, Visual Basic for Applications (VBA), Visual Basic .NET, Visual C .NET, Visual

Table 2.1: Project files

Filename	description
RSSR.exe	Is the main add-in program
RSSR_server.dll	RSSR synthesis routine
SldWorks.dll	SolidWorks Type Library
SwConst.dll	Predefined constants used in SolidWorks
getRSSR.dll	Toolbar/Icon callback
InputRod.sldprt	Model of the input crank
OutputRod.sldprt	Model of the output crank
TieRod.sldprt	Model of the connecting rod
RSSR.sldasm	Assembly of a generic RSSR mechanism

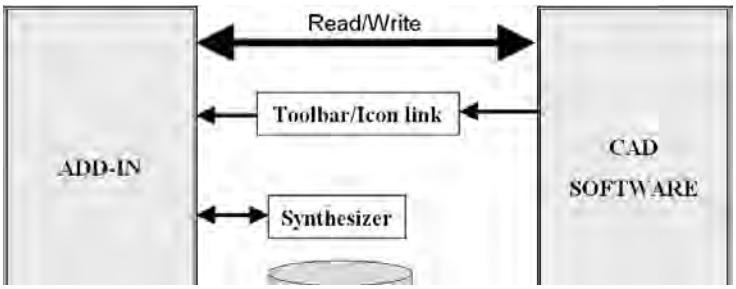

Figure 2.1: Add-in architecture.

C++ 6.0 , Visual C++ .NET, or SolidWorks macro files. The SolidWorks API software must be installed so that the Type Libraries are registered in the Windows Registry, this enables the programming software (Visual Studio.NET) to present a programmer with a list of type libraries on the current computer. By adding a reference to the SW Type Library we are allowing a COM client, such as the host CAD software, to connect to our add-in object. Note that SolidWorks acts as an out-of-process COM server, meaning that it runs in a separate process from the add-in.

The add-in application is compiled as a COM server Windows Application (*exe* file extension). This COM application can be executed from a menu toolbar/icon in the CAD software or in stand-alone mode. In both cases, the add-in is equally able to communicate with the CAD software. To call the application from a menu

toolbar/icon from the CAD software we must add a linking module from the host to our project. The linking module for this project is called *getRSSR.dll* and is described in Chapter 6.

The goal of our add-in is to add a solid model assembly of an RSSR mechanism that satisfies the geometric requirements of the designer/user. The steps followed by the add-in are summarized as:

1. Load user interface.
2. Establish a connection with the CAD software.
3. Retrieve data from current assembly.
4. Synthesize RSSR Linkage.
5. Create solid model of the steering linkage.

2.4 Interaction with the Host CAD Software

The add-in must include a reference variable to handle data transfer to/from the CAD software. This variable must be defined as

```
swApp = CreateObject("SldWorks.Application")
```

where *SldWorks* is the highest-level object in the API. it provides direct and indirect access to all other objects exposed in this particular API. For SolidWorks, the main reference library is called *SW 2004 Type Library* located in the Installation CD. This type library defines the object structure characteristic of the CAD software and the appropriate function calls (*methods*) allowed for each specific type of object. These functions provide direct access to SolidWorks functionality such as creating a line, inserting an existing part into a part document, or verifying the parameters of a surface. For example: if we consider the object type *SketchSegment*, one of the applicable methods for this object type is *GetLength(SketchSegment)*, which determines the length of the selected segment. It is key to the success of any project

to know the object's hierarchy structure characteristic to each application. This project works mainly with three object types: parts, assemblies and sketches. The characteristic object structure that applies to this particular project is shown in Fig. 2.2.

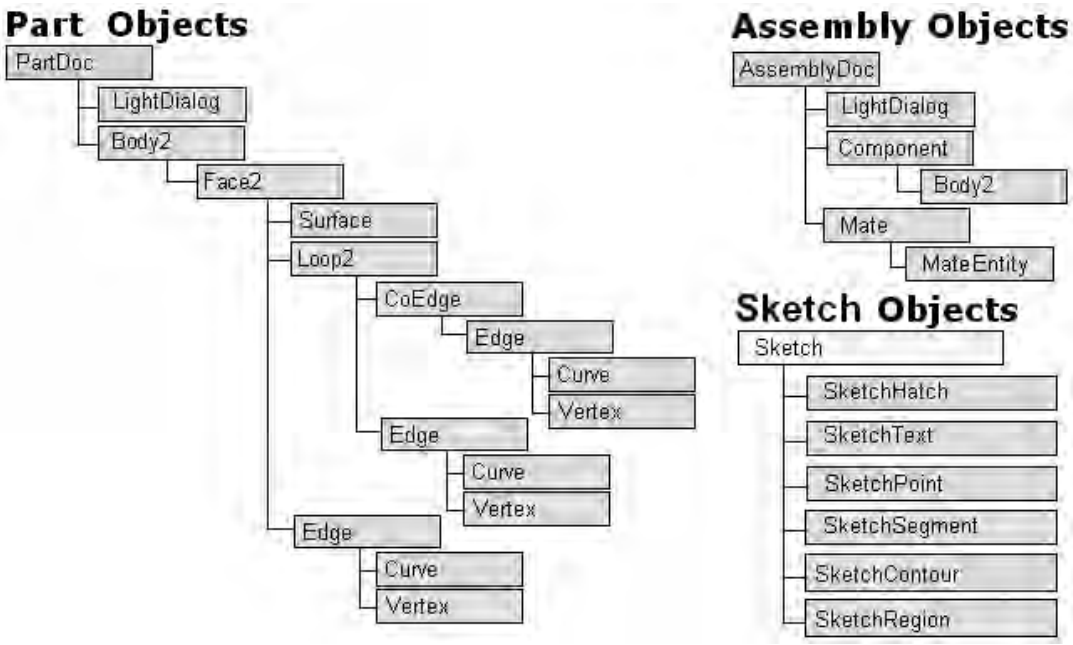

Figure 2.2: Object hierarchy.

If help would be needed when programming the add-in, the best help resource is the API Help Menu in SolidWorks. This help menu includes documented information about every object available in the SolidWorks API, including its associated properties and methods. For more experienced programmers, the easiest way to obtain help when programming is to record a sequence of actions using the SolidWorks Macro Recording tool. This tool converts the chain of actions into Visual Basic calls that correspond to the actions performed in the user interface. This code can later be used to fit the add-in's specific needs.

2.5 Graphical User Interface

The tool we provide for the designer is in the form of a graphical user interface (GUI). Our RSSR synthesis add-in GUI is shown in Fig. 2.3. The functionality of

the interface can be summarized in the following:

- GET: retrieves the coordinates of a selected point directly.
- HELP: retrieves the coordinates of a selected entity with some guidance using a help menu interface (Fig. 2.4).
- DEFAULT VALUES: loads a default set of points and angles.
- CLEAR: erases all the values appearing in the text boxes.
- CALCULATE: synthesizes the steering mechanism given the input data.
- DRAW IN SOLIDWORKS: adds the corresponding synthesized solid model of the RSSR steering mechanism to the active assembly.
- CLOSE: quits the add-in and returns to the current CAD software session.

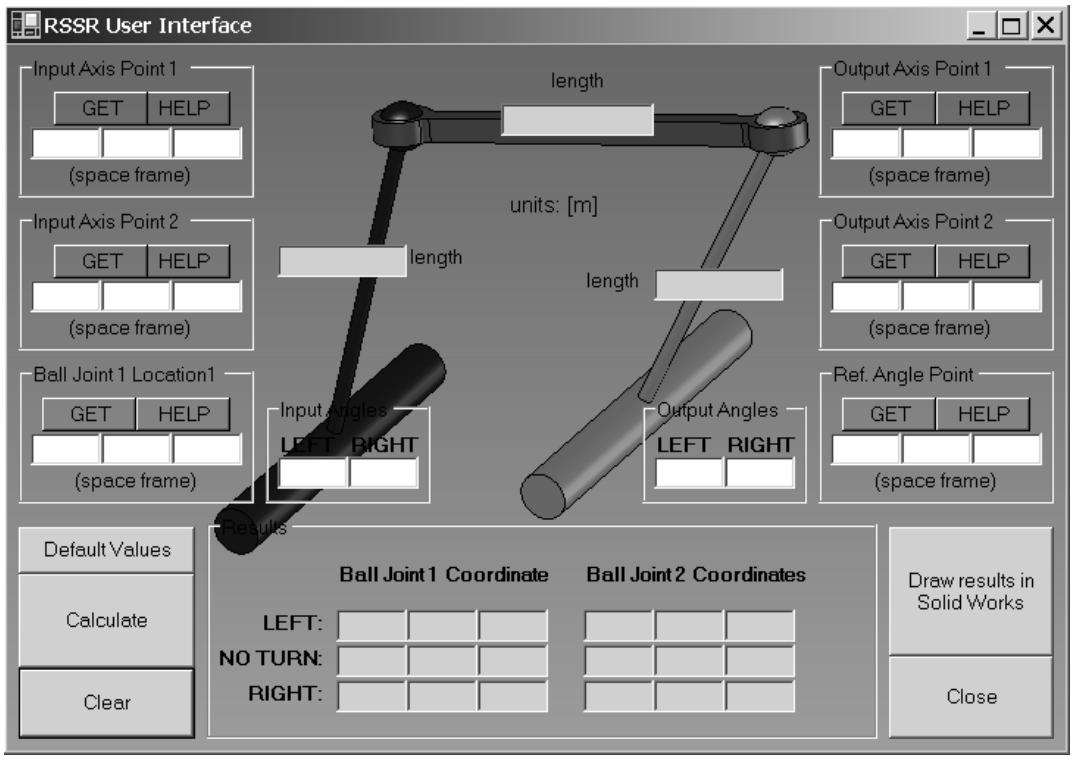

Figure 2.3: RSSR steering linkage GUI.

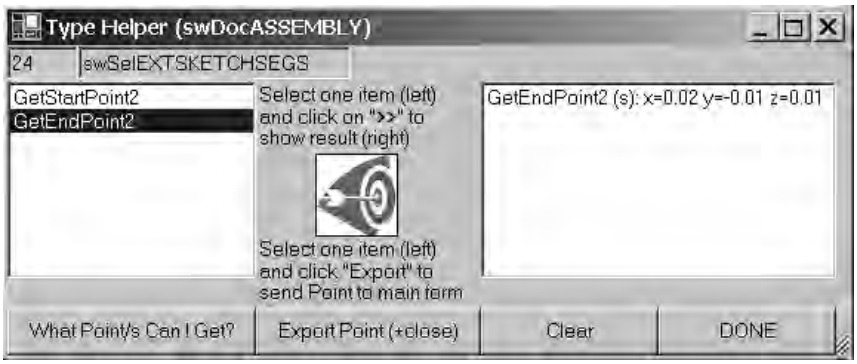

Figure 2.4: Help menu.

More detailed specifications on the process of retrieving the specified geometry and the creation of the solid model can be found in the following chapters.

The code corresponding to this GUI can be found in Appendix A.1.

Chapter 3

Implementing an RSSR Function Generator

3.1 Introduction

In this chapter will we will explain how the RSSR function generator is implemented into the add-in. The first step is to create the code that will define all the functions, routines and methods involved in our project. Then we compile the RSSR function generator into a Dynamic Link Library (dll) file. The third step is to add this dll file to the main add-in project as reference. Then we must create a dll object handler within the add-in to be able to access the functionality of the function generator.

3.2 Creating *RSSR_server.dll*

The RSSR function generator has been implemented and compiled independently from the add-in. We have created *RSSR_server.dll*, an RSSR Function Generator

Dynamic Link Library. The full source code of *RSSR_server.dll* can be found in Appendix A.2. For simplicity purposes, this library file has been written in C++. The same result could have been obtained by using Visual Basic or Visual Basic.NET.

This dll file must enable external access to its functions and variables. The main public functions that enable the add-in to enter data to the dll and retrieve the resulting data are sorted in Table 3.1.

Table 3.1: Read/Write commands.

Write data	Read data
put_data2($\mathbf{Z}_1$, $\mathbf{Z}_2$, $\mathbf{S}_1$, $\mathbf{P}_{REF}$)	
put_theta2(θ)	HCalculate2($\mathbf{S}_1$,$\mathbf{S}_2$)
put_phi2(ϕ)	

3.3 Calling *RSSR_server.dll* from the Add-in

To be able to use the RSSR function generator package from the add-in, *RSSR_server.dll* must be added to the main reference list. By establishing a reference to this file, we are enabling access to the class functions defined in the dll file. The communication between the dll and the add-in is summarized in Fig. 3.1.

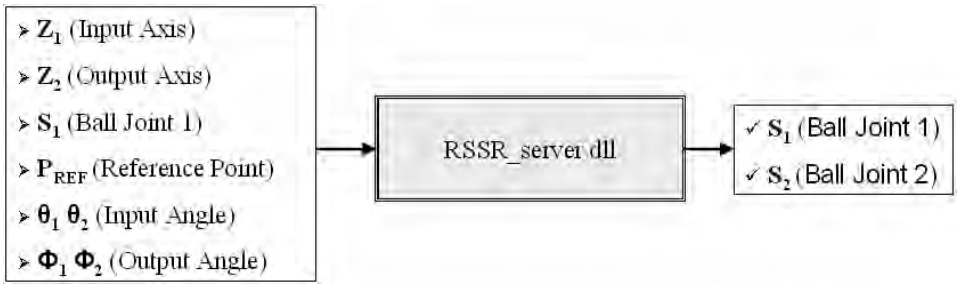

Figure 3.1: *RSSR_server.dll* diagram.

The add-in calls the commands that enable writing data in the form of coordinate points and angles, and the commands that enable reading the resulting data (coordinate points).

To use the dll file from the add-in we must define a dll object handler *rssrObj* defined as

```
Dim rssrObj As RSSR_Server.CNAT_RSSR
```

To call the commands in Table 3.1 we use the following notation

```
rssrObj.name_of_function(parameters)
```

Once the location of $\mathbf{S}_1$ and $\mathbf{S}_2$ is determined (in the three positions) we use the equations from Chapter I to calculate r_1, r_2 and r_3.

Chapter 4

Selecting User Specified Geometry from the CAD Software

4.1 Introduction

Usually CAD software has three different types of documents: parts, drawings and assemblies. Our add-in is intended to generate assemblies.

The data used to synthesize the steering linkage is the form of three-dimensional points retrieved directly from the solid model. The coordinates of these points must be referenced to a global world.

4.2 Sketch Frame, Model Frame and World Frame

A sketch is typically used in part or assembly documents to generate a solid body or such subsequent features as a cut or boss. A sketch is also used in drawing documents whenever the user constructs individual lines, arcs, and so on, on a drawing sheet

or in a drawing view. But our add-in works only with assemblies. If the user calls the add-in from a part or drawing document, an error message is generated.

In order to design our linkage we extract data from sketch points, vertices, edges and sketch segments in an assembly drawing. This is a small subset of the 109 selectable objects in SolidWorks 2004. Any object selected by the user contains not only geometrical information, but also has information concerning the layer in which it was drawn or the material of the body it belongs to, among others. Since this application focuses on lines and points included in sketches, it is important to know exactly what the object hierarchy structure looks like for sketches, parts and assemblies (Fig. 2.2).

This application retrieves data from points one at a time. If more than one geometrical entity is selected, only the first one will be taken into account. In this add-in, lines are defined from the coordinates of its end and start points. If the user knows the exact location of each point, data can be retrieved directly by selecting a desired point in first place and then selecting the add-in command labeled *GET*. The coordinates with respect to a fixed coordinate system in the assembly space frame will be loaded into the corresponding fields. We will call this fixed reference frame *World Frame*, $\{W\}$. For more complicated selections, or less experienced users, a help interface is available (Fig. 2.4).

Let's consider the case where the user wants to retrieve the coordinates of the end point of the selected line. After selecting the line and later selecting the help command, a list of points associated to the geometrical entity is generated. Having selected a line, the listed available points will consist in the starting point and the ending point. From this list, the user can choose the desired point he wants to retrieve, in this case the end point. Before exporting the data, the user can preview the (x, y, z) coordinates in the reference coordinate system $\{W\}$ in a preview window.

It's crucial to know how the coordinates of the elements are treated by the CAD software. When the CAD software retrieves data from a sketch element (for example: edges) it gathers the (x, y, z) coordinates of the starting point and the end point of the edge with respect to the planar sketch in which it was drawn. For each sketch

plane, there is a homogeneous transformation from the sketch frame $\{S\}$ to the model frame $\{M\}$, unique for each part. At the same time, the body is located in an assembly with a reference frame, the world frame $\{W\}$. Therefore, there exists a homogeneous transformation from the body frame to the spatial assembly frame. In this add-in, the synthesis calculations are based on a fixed coordinate system (World Frame) characteristic to the assembly. SolidWorks stores these transformation as a homogeneous transformation matrix. Note that for this particular CAD software, transformation matrix $[T]$ is stored as a homogeneous matrix of 16 elements, ordered as

$$[T] = \left[\begin{array}{ccc|c} a & b & c & n \\ d & e & f & o \\ g & h & i & p \\ \hline j & k & l & m \end{array}\right], \tag{4.1}$$

where the first 9 (a to i) are elements of a 3x3 rotational sub-matrix, the next 3 (j,k,l) define a translation vector, the next 1 (m) is a scaling factor. The last 3 elements (n,o,p) are unused in this context. This matrix is the transposed form of the homogeneous transformation described in Eqn. (1.10).

The 3×3 rotational sub-matrix represents 3 axis sets: row 1 for x-axis components of rotation, row 2 for y-axis components of rotation, and row 3 for z-axis components of rotations. The 3 axes are constrained to be orthogonal and unified so that they produce a pure rotational transformation. Reflections can also be added to these axes by setting the components to negative. The rotation sub-matrix coupled with the lower-left translation vector and the lower-right corner scaling factor creates an affine transformation, one that preserves lines and parallelism: maps parallel lines to parallel lines.

If the 3 axis sets of the 3x3 rotational sub-matrix are not orthogonal or unified, then they are automatically corrected according to the following rules:

- If any axis is 0, or any two axes are parallel, or all axes are coplanar, then an identity matrix replaces the rotational sub-matrix

- All axes are corrected to be of unit length

- The axes are built to be orthogonal to each other in the prioritized order of z, x, y (x is orthogonal to z, y is orthogonal to z and x)

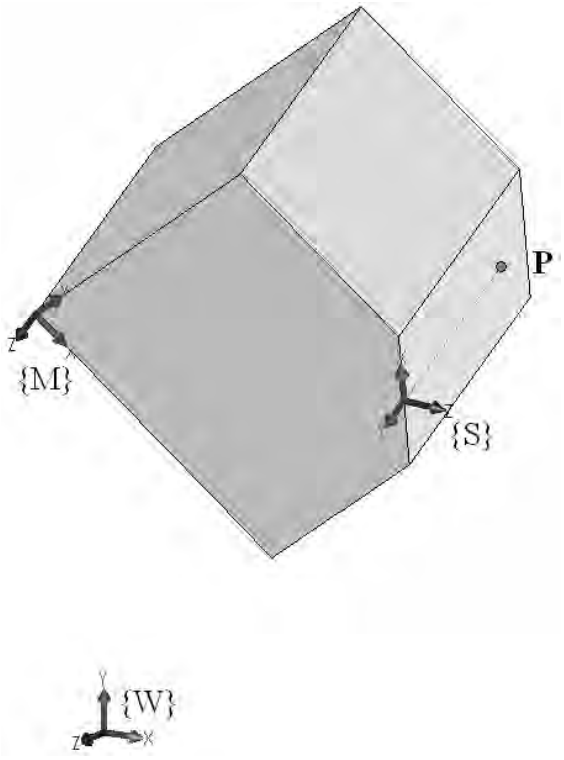

Figure 4.1: Three different coordinates frames.

The relative transformation from the sketch frame to the fixed space frame (Fig. 4.1) is defined as

$$\mathbf{P}_{\{W\}} = [T_{\{M\}\rightarrow\{W\}}][T_{\{S\}\rightarrow\{M\}}]\mathbf{P}_{\{S\}}, \tag{4.2}$$

where $\mathbf{P}_{\{W\}}$ are the coordinates of a point in the fixed world frame characteristic of the assembly space, $T_{\{M\}\rightarrow\{W\}}$ is the transformation from the model frame to the fixed world frame, $T_{\{S\}\rightarrow\{M\}}$ is the transformation from the sketch frame to the model frame and $\mathbf{P}_{\{S\}}$ are the coordinates of a point in the sketch frame. Note that whatever the user configured units are, the coordinates of points are always retrieved in meters by the API.

To handle these transformations, the API provides a set of helpful tools. One of these tools is *SldWorks.MathTransform*, which provides a simplified interface

for manipulating transformation matrix data. The *SldWorks.MathUtility* tool provides access to the SolidWorks math objects used for creating new transform matrix objects, new *MathPoint* objects and *MathVector* objects. The methods most commonly used to create our add-in are:

- *swComp.Transform2* retrieves the homogeneous transformation from the fixed assembly frame to the model or body frame of the selected component (part). This method can only be used when the selected object is referenced to the root component.

- *swSketch.ModelToSketchTransform* this property contains the model-to-sketch transform for the sketch member. Multiplying a point located in model space coordinates by the resulting matrix, we obtain the coordinates of the point in terms of the sketch frame. Note that to transform coordinates from $\{S\}$ to $\{M\}$ we use the inverse form of this property.

- *Transform2.Inverse* creates a new *MathTransform* object by inverting the values in an already existing transform object.

- *swMathPt.MultiplyTransform(swCompXform)* transforms the coordinates of a point, *swMathPt*, using a specified transformation matrix, *swCompXform*. The *MathPoint* object is multiplied by a *MathTransform* object . The matrix is rotated, scaled, and then translated.

  For more details on applied geometry for computer graphics, see Marshall (1999) [17].

4.3 Example of Coordinate Transformation

In this example we show the use of the API commands used to retrieve the coordinates of a point with respect to three frames.

The model consists of a cube with a chamfered edge. We have added a sketch line on the chamfer plane. The origin of the cube's frame or *model frame*, $\{M\}$ is located at the lower left vertex, as shown in Fig. 4.1. The origin of the *sketch*

frame, $\{S\}$ is located on the chamfer plane. The origin of the assembly frame or *world frame* is $\{W\}$. The coordinate frame origins are automatically set by the CAD software.

Once the model is complete, it has been imported into an assembly. The position and orientation of the cube within the assembly is quite random.

The coordinates of the selected point, $\mathbf{P}$ with respect to $\{S\}$ are $(0, -0.02, 0)$.

The following macro is written in Visual Basic for Applications (VBA) using the Macro Editing tool provided by the CAD software.

VBA Code for the Macro: *swTtransformsMacro.bas*

```
Sub main()

Dim swApp                       As SldWorks.SldWorks
Dim swModel                     As SldWorks.ModelDoc2
Dim swComp                      As SldWorks.Component2
Dim swSelMgr                    As SldWorks.SelectionMgr
Dim swSketchPoint               As SldWorks.SketchPoint
Dim swSketch                    As SldWorks.Sketch
Dim swMathPt                    As SldWorks.MathPoint
Dim swMathUtil                  As SldWorks.MathUtility

Dim swCompXform                 As SldWorks.MathTransform
Dim swCompXformInv              As SldWorks.MathTransform
Dim swModel2Sketch              As SldWorks.MathTransform
Dim swModel2SketchInv           As SldWorks.MathTransform
Dim swTotalTrans                As SldWorks.MathTransform
Dim swTotalTransInv             As SldWorks.MathTransform

Dim swSelectType                As Integer
Dim swXYZPoint(2)               As Double
Dim vXform                      As Variant
Dim vPt                         As Variant
Dim swPointSketchFrame          As Variant
Dim swPointBodyFrame            As Variant
Dim swPointWorldFrame           As Variant
Dim OutputFormat                As String
```

```
OutputFormat = "0.0000"

'Connect the program to SolidWorks
Set swApp = CreateObject("SldWorks.Application")
'Get the active document (should be an assembly)
Set swModel = swApp.ActiveDoc
'Prepare the MathUtility
Set swMathUtil = swApp.GetMathUtility

'Get the SelectionMgr
Set swSelMgr = swModel.SelectionManager

'type of object selected (swSelectType=25)
swSelectType = swSelMgr.GetSelectedObjectType2(1)

'Get the selected sketch point
Set swSketchPoint = swSelMgr.GetSelectedObject5(1)

'Obtain the sketch where the point belongs to
Set swSketch = swSketchPoint.GetSketch

'Obtain the Component (part, model or body) where the point belongs to
Set swComp = swSelMgr.GetSelectedObjectsComponent2(1)

swXYZPoint(0) = swSketchPoint.X
swXYZPoint(1) = swSketchPoint.Y
swXYZPoint(2) = swSketchPoint.Z

Set swMathPt = swMathUtil.CreatePoint(swXYZPoint)

'****************************
'*    swPointSketchFrame    *
'* (X,Y,Z) IN SKETCH FRAME  *
'****************************

swPointSketchFrame = swXYZPoint

'Retrieve the transformation of the selected swComp(body)
'first we retrieve the transformation from world frame to model frame
Set swCompXform = swComp.Transform2

'The inverse of the previous XForm goes from model frame to world frame
Set swCompXformInv = swCompXform.Inverse
```

```
Set swModel2Sketch = swSketch.ModelToSketchTransform
Set swModel2SketchInv = swModel2Sketch.Inverse
Set swMathPt = swMathUtil.CreatePoint(swPointSketchFrame)
'Calculate the cords with respect to the model frame {M}
Set swMathPt = swMathPt.MultiplyTransform(swModel2SketchInv)

'****************************
'*    swPointBodyFrame      *
'* (X,Y,Z) IN MODEL FRAME   *
'****************************

swPointBodyFrame = swMathPt.ArrayData

Set swTotalTrans = swModel2SketchInv.Multiply(swCompXform)
Set swTotalTransInv = swTotalTrans.Inverse
Set swMathPt = swMathUtil.CreatePoint(swPointSketchFrame)

'Calculate the cords with respect to the world frame {W}
Set swMathPt = swMathPt.MultiplyTransform(swTotalTrans)

'****************************
'*    swPointWorldFrame     *
'* (X,Y,Z) IN WORLD FRAME   *
'****************************

swPointWorldFrame = swMathPt.ArrayData

'Print out the results

vPt = swPointSketchFrame

Debug.Print "-------------------------------------------"
Debug.Print "POINT IN SKETCH FRAME"
Debug.Print "{{" & Format(vPt(0), OutputFormat) & "},
Debug.Print " {" & Format(vPt(1), OutputFormat) & "},"
Debug.Print " {" & Format(vPt(2), OutputFormat) & "}}"
Debug.Print ""
Debug.Print ""

vPt = swPointBodyFrame

Debug.Print "POINT IN MODEL(BODY) FRAME"
Debug.Print "{{" & Format(vPt(0), OutputFormat) & "},
```

```
Debug.Print " {" & Format(vPt(1), OutputFormat) & "},"
Debug.Print " {" & Format(vPt(2), OutputFormat) & "}}"
Debug.Print ""

vPt = swPointWorldFrame

Debug.Print "POINT IN WORLD FRAME"
Debug.Print "{{" & Format(vPt(0), OutputFormat) & "},
Debug.Print " {" & Format(vPt(1), OutputFormat) & "},"
Debug.Print " {" & Format(vPt(2), OutputFormat) & "}}"
Debug.Print ""

Debug.Print "-------------------------------------------"
Debug.Print ""

vXform = swCompXform.ArrayData

Debug.Print "-------------------------------------------"
Debug.Print ""
Debug.Print "swCompXform = "
PrintXForm (vXform)

vXform = swCompXformInv.ArrayData

Debug.Print "swCompXformInv = "
PrintXForm (vXform)

vXform = swModel2Sketch.ArrayData

Debug.Print "swModel2Sketch = "
PrintXForm (vXform)

vXform = swModel2SketchInv.ArrayData

Debug.Print "swModel2SketchInv = "
PrintXForm (vXform)

vXform = swTotalTrans.ArrayData

Debug.Print "swTotalTrans = "
PrintXForm (vXform)

vXform = swTotalTransInv.ArrayData
```

```
Debug.Print "swTotalTransInv = "
PrintXForm (vXform)
End Sub
'-----------------------------------------------------------------------
Sub PrintXForm(number As Variant)
Debug.Print "{"
Debug.Print "{" & number(0) & ", " & number(1) & ", " & number(2) & ", " & number(13) & "},"
Debug.Print "{" & number(3) & ", " & number(4) & ", " & number(5) & ", " & number(14) & "},"
Debug.Print "{" & number(6) & ", " & number(7) & ", " & number(8) & ", " & number(15) & "},"
Debug.Print "{" & number(9) & ", " & number(10) & ", " & number(11) & ", " & number(12) & "}}"
Debug.Print ""
End Sub
```

And we obtain

$$[swCompXform] = \begin{bmatrix} 0.6017 & -0.7266 & -0.3316 & 0 \\ 0.7973 & 0.5715 & 0.1943 & 0 \\ 0.0484 & -0.3813 & 0.9232 & 0 \\ -0.0168 & 0.0373 & -0.0170 & 1 \end{bmatrix}$$

$$[swCompXformInv] = \begin{bmatrix} 0.6017 & 0.7973 & 0.0484 & 0 \\ -0.7266 & 0.5715 & -0.3813 & 0 \\ -0.3316 & 0.1943 & 0.9232 & 0 \\ 0.0316 & -0.0046 & 0.0307 & 1 \end{bmatrix}$$

$$[swModel2Sketch] = \begin{bmatrix} -0.7071 & 0.0000 & 0.7071 & 0 \\ 0.7071 & 0.0000 & 0.7071 & 0 \\ 0.0000 & 1.0000 & 0.0000 & 0 \\ 0.0000 & 0.0000 & -0.0354 & 1 \end{bmatrix}$$

$$[swModel2SketchInv] = \begin{bmatrix} -0.7071 & 0.7071 & 0.0000 & 0 \\ 0.0000 & 0.0000 & 1.0000 & 0 \\ 0.7071 & 0.7071 & 0.0000 & 0 \\ 0.0250 & 0.0250 & 0.0000 & 1 \end{bmatrix}$$

$$[swTotalTrans] = \begin{bmatrix} 0.1383 & 0.9179 & 0.3719 & 0 \\ 0.0484 & -0.3813 & 0.9232 & 0 \\ 0.9892 & -0.1097 & -0.0971 & 0 \\ 0.0181 & 0.0334 & -0.0204 & 1 \end{bmatrix}$$

$$[swTotalTransInv] = \begin{bmatrix} 0.1383 & 0.0484 & 0.9892 & 0 \\ 0.9179 & -0.3813 & -0.1097 & 0 \\ 0.3719 & 0.9232 & -0.0971 & 0 \\ -0.0256 & 0.0307 & -0.0163 & 1 \end{bmatrix}.$$

These transformation matrices are written in the form of Eqn. (4.1), where the translation term is in the fourth row. To express the matrices according to the Denavit and Hartenberg convention, we must transpose the previous matrices.

We use Eqn. (1.9) and Eqn. (1.12) to obtain the vector coordinates of the selected point, $\mathbf{P}$, with respect to the three different frames. Using the previous transformation matrices we obtain

$$\mathbf{P}_{\{S\}} = [I]\mathbf{P}_{\{S\}} \tag{4.3}$$

$$\mathbf{P}_{\{M\}} = [swModel2SketchInv]\mathbf{P}_{\{S\}} \tag{4.4}$$

$$\mathbf{P}_{\{W\}} = [swCompXform][swModel2SketchInv]\mathbf{P}_{\{S\}} \tag{4.5}$$

The resulting point coordinates for $\mathbf{P}$ are listed in Table 4.1.

Table 4.1: Point coordinates with respect to the three frames.

Point	x	y	z
$\mathbf{P}_{\{S\}}$	0.0000	-0.0200	0.0000
$\mathbf{P}_{\{M\}}$	0.0250	0.0250	-0.0200
$\mathbf{P}_{\{W\}}$	0.0172	0.0410	-0.0389

Chapter 5

Generating a Solid Model of the Synthesized RSSR Mechanism

5.1 Introduction

In this chapter we explain the process of converting the numerical result of the synthesis routine into a solid model.

The input and output crank and the connecting rod are modified from the generic RSSR assembly library we have created. Before importing the RSSR mechanism into the current assembly, the add-in resizes the three parts of the assembly using the values of r_1, r_2 and r_3 (see Chapter I).

5.2 Modifying Library Parts

After the RSSR linkage is synthesized, the user selects the command labeled *Draw in SolidWorks* to create a solid model of the linkage and import it into the current

user's assembly.

The synthesized RSSR mechanism is not created from scratch. We have created a generic RSSR assembly. We access this library assembly to resize to our convenience and import it into the user's main assembly. To resize the parts we must access the dimension object that handles the length of the feature we want to resize, (Fig. 5.1). This is the sequence of commands used to modify r_1 (see Chapter I).

```
'1.- Resize Input Ball Joint Location distance to 'Length1'
boolstatus = swModel.Extension.SelectByID("SJointPoint",...
   "SKETCH", 0, 0, 0, False, 0, Nothing)
boolstatus = swModel.Extension.SelectByID("D1@SJointPoint@...
   InputRod.SLDPRT", "DIMENSION", 0, 0, 0, False, 0, Nothing)

swModel.Parameter("D1@SJointPoint").SystemValue = Length1

swModel.EditRebuild()
write(Chr(10) + "R1 has been resized to   " & Length1 & " m")

swModel.ViewZoomtofit2()
swModel.EditRebuild()
swApp.CloseDoc("InputRod.SLDPRT")

'swFileName = "steering.SLDASM"
swModel = swApp.ActivateDoc(swFileName)
swModel = swApp.ActiveDoc
```

The complete sequence of commands used to import the RSSR assembly is found in Appendix A.1.7.

After the RSSR assembled mechanism is resized, it is imported into the user's current assembly. Usually there will be only two RSSR mechanisms in one assembly, one corresponding to the left wheel and the other to the right one. This add-in can

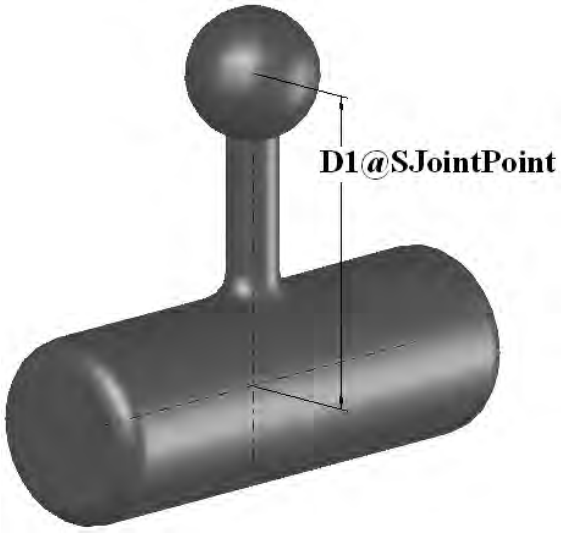

Figure 5.1: Resizing a dimension.

repeat this process and import as much as 12 different synthesized RSSR mechanisms per assembly. Using a different configuration for each synthesized RSSR assembly is the only way to have two or more assemblies with different dimensions in the same assembly (Tickoo 2003 [18]).

The last step is to determine the proper location of the imported assembly. This is done by mating planes, lines and points of the RSSR assembly to the lines and points of the user's assembly, where the data was retrieved from. The mates used to relocate the RSSR mechanism are listed in Table 5.1.

Table 5.1: Mates used to locate the imported RSSR mechanism.

Element 1	Element 2	Relation
axis of rotation (input crank)	$\mathbf{z}_1$	collinear
plane $\perp$ rotation axis (input crank)	$\mathbf{S}_1$	coincident
axis of rotation (output crank)	$\mathbf{z}_2$	collinear
plane $\perp$ rotation axis (output crank)	$\mathbf{P}_{REF}$	coincident
ball joint 1 (input crank)	connecting rod end 1	coincident
ball joint 2 (output crank)	connecting rod end 2	coincident

The mate that sets the spherical joint between the connecting rod and the two cranks is already defined in the original RSSR assembly library model.

The process of creating, importing and locating the synthesized steering linkage can be summarized with the following steps:

1. Look for previously imported RSSR sub-assemblies.
2. Open the RSSR library sub-assembly and assign a new configuration number.
3. Resize parts using computed values for r_1, r_2 and r_3.
4. Import resized library assembly into the main user's assembly.
5. Relocate input axis.
6. Relocate the rotation plane of the input crank.
7. Relocate the output axis.
8. Relocate the rotation plane of the output crank.
9. Zoom view to fit the steering linkage.

The final result after rendering the solid model of the assembly of the input crank (1), output crank (2) and connecting rod (3) is shown in Fig. 5.2. The complete steering mechanism is shown in Fig. 5.3.

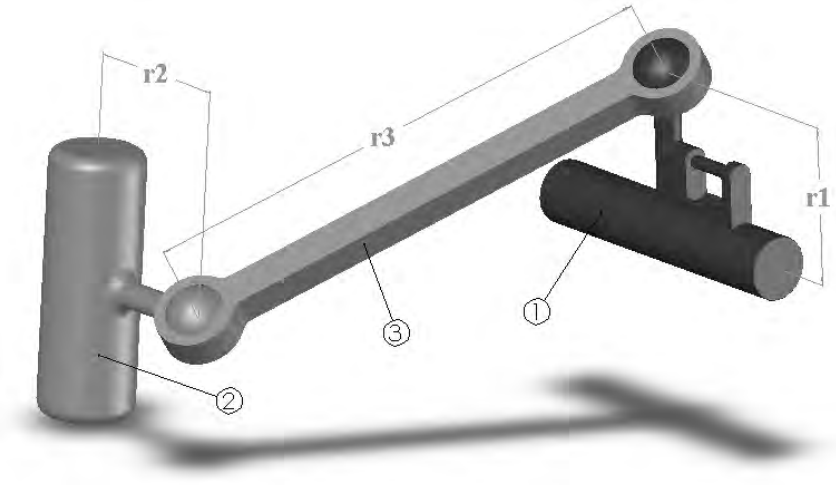

Figure 5.2: Synthesized RSSR mechanism (left side).

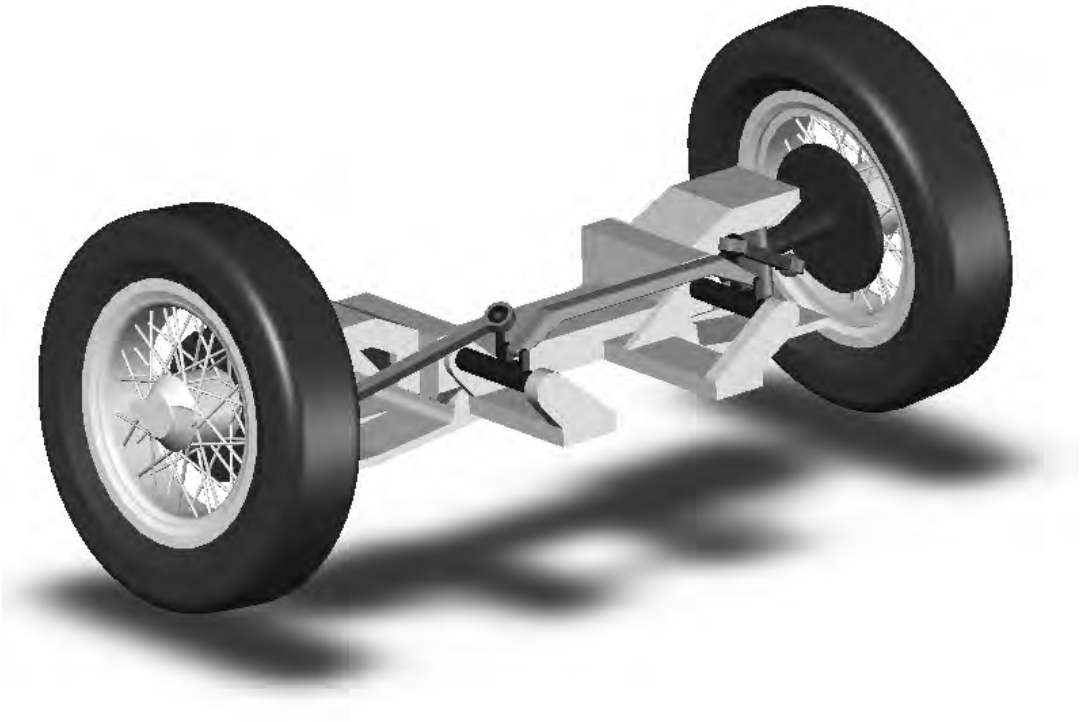

Figure 5.3: Complete steering linkage device with coupler.

Chapter 6

Creating a Toolbar/Icon to Access the Add-in from the CAD Software

6.1 Introduction

In this chapter we describe the creation of a custom-made menu item or toolbar icon in the host software.

Although our tool can synthesize RSSR chains in a stand-alone mode, the final purpose of the add-in is to generate a solid model of the solution within the modeling software.

6.2 Creating a Toolbar/Icon

After having compiled our synthesizer we obtain the file *RSSR.exe*, a stand-alone application capable of communicating with SolidWorks, synthesizing the desired RSSR mechanism and importing a solid model representation of the linkage. To make the synthesizing application embedded into the host CAD software we must create a linking object that will call the application from a menu or an icon in the toolbar. We call this linking object *getRSSR.dll*. This dll program has been written in VBA (Visual Basic for Applications) using the SolidWorks API online support guidance (see [16]).

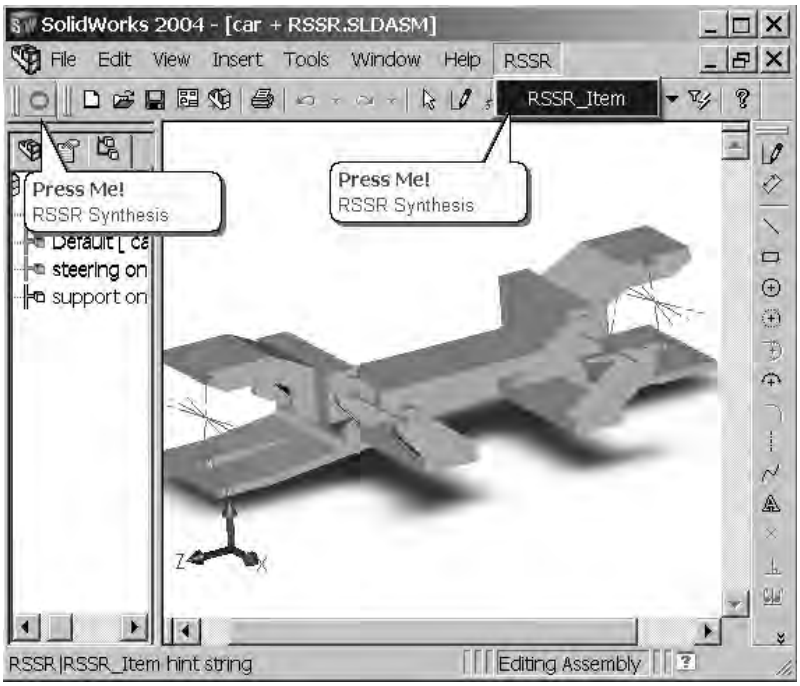

Figure 6.1: Menu toolbar and icon.

To convert a dll into a SW Addin, the dll must have a class module that implements the exposed type libraries for add-in use, *swpublished.tlb*, along with *sldworks.tlb* (see Chapter 2). The class module requires at least two procedures

`SwAddin_ConnectToSW`

`SwAddin_DisconnectFromSW`

The complete code for *getRSSR.dll* is presented in Appendix A.3.

To create a menu in the host application's environment, the add-in must call the command *AddMenuItem2* and pass the callback method associated with the menu item. It can also call *AddToolbar4* to create a Windows-style toolbar that contains a set of application-defined buttons and pass the callback method associated with the toolbar button, as shown in Fig. 6.1.

The linking application *getRSSR.dll* is compiled as a Dynamic Link Library to allow the following events:

- Implement a co-creatable object that supports *SwAddin*.
- Define event handlers as needed.
- Register the Add-in CLSID (Class ID) in the computer's registry.

The CLSID is an identification tag associated with an ActiveX or OLE2.0 object created by a specific component or server. CLSID values appear in the Registry and must be unique for each type of object that the server can create. Applications that support Microsoft's COM architecture register their objects as class IDs. To register the compiled project as a SolidWorks add-in, the recently created dll must be opened using the main CAD executable file *solidworks.exe*. This registration process is only done once. The host software will register the add-in in the Windows Registry, under

```
HKEY_LOCAL_MACHINE\SOFTWARE\Solidworks\AddIns
```

To make the registered add-in visible, it must be activated from the CAD software's add-in tool as shown in Fig. 6.2.

When the toolbar or the menu are created, we must define what their function is, i.e. their callback function. Any toolbar or menu item has four states, listed in Table 6.1.

When the user selects the menu/toolbar, the callback function launches *RSSR.exe*. This event is programmed as

```
Public Sub RSSR_Item()
    Dim retval As Boolean
    retval = Shell("C:\temp\RSSR.exe", 1)
End Sub

Public Sub ToolbarFunc()
Dim retval As Boolean
    retval = Shell("C:\temp\RSSR.exe", 1)
End Sub
```

To remove the add-in from the host software the user must disable the add-in from the manager (Fig. 6.2) and then remove the Windows Registry entry.

Table 6.1: Menu/Toolbar states.

value	state information for the menu/toolbar item
0	Disabled and unchecked
1	Enabled and unchecked (default)
2	Disabled and checked
3	Enabled and checked

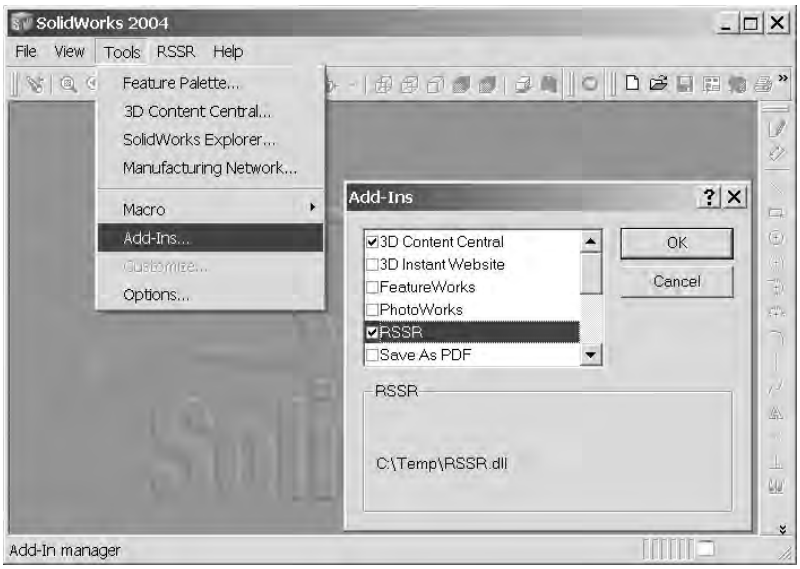

Figure 6.2: Add-in manager.

Chapter 7

Summary

7.1 Conclusions

We have successfully created an algorithm for an RSSR function generator written in Visual Basic.NET and integrated into the SolidWorks CAD program as an Add-in. This Add-in establishes the connection between the host software and our program by reading data from the host which is needed by our routine, then generating solid models and exporting them to the host. The construction of the Add-in was relatively easy using Visual Basic. NET, however, determining the SolidWorks functions and object hierarchy was rather difficult. Because these functions are CAD software specific, it seems difficult to generalize this process.

7.2 Future Work

This research provides the foundation for new results in several different directions:

- Integrate new synthesis routines into CAD environments and develop more

industry specific synthesis applications. This would allow the designers to evaluate the design candidate using existing tools such as Ansys, Cosmos, etc.

- Improving the data acquisition routine by considering a wider range of objects. An improved version of this add-in should include the possibility of selecting a line object instead of a set of points.

- Applying analysis techniques to identify interference with other parts. Although a solution to the synthesis of an RSSR linkage may exist, its location within an assembly might be restricted by the neighboring parts.

- Improving the library model of the RSSR assembly making each part more realistic. In the actual model, there is no part connecting the turning wheels to the output crank.

- Optimizing the resizing process described in Chapter 5. Our program must open each one of the three parts to resize them individually. This process might be optimized by hiding the resizing process, not making it visible to the eyes of the user.

- Including an animation module to our add-in. This would allow the user to turn the steering wheel and see how the wheels respond. Most CAD softwares include animation packages.

Chapter 8

Steering Linkage Synthesis Example

8.1 Introduction

In this chapter we describe the use of our add-in for a particular case and we present the corresponding numerical results of the synthesis of the steering linkage.

8.2 Using the Add-in

The first step is for the user to model the support part where the steering mechanism will be added. Let's assume the support part file created in SolidWorks is called *FrontSupport.sldprt*. The location of the rotation axis of the input and output cranks must be defined by a line type object (SW types: *Vertex*, *Edge*, *SketchLine*, *SketchSpline)* or by two points (SW types: *SketchPoint* or *Vertex*).

Then the support part model must be inserted in an assembly file (for example:

car.sldasm).

The third step is to launch the add-in from the toolbar or icon in the CAD software. The add-in will only work if it has been launched from the assembly mode (the active file would be *car.sldasm*)).

The next step is for the user to use the graphical user interface to enter the data points required to synthesize the steering linkage. To do so, the user must first select a point object in the assembly. Once the point is selected, the user must select the GUI command labeled *GET*. The coordinates of the point will be loaded into the corresponding text box. This process must be repeated for each point (6 in total).

The pair or relative angles are entered directly using the keyboard.

The fifth step is to synthesize the mechanism by selecting the button labeled *Calculate*. The three positions of the ball joints and the link lengths are loaded in the corresponding text boxes.

The last step is to add a model of the synthesized RSSR mechanism into *car.sldasm*). This is done by selecting the command button labeled *Draw in SolidWorks*. When this button is selected, SolidWorks opens the three components (*sldprt* files) of the RSSR mechanism and resizes them. After resizing, it imports the assembly of the synthesized linkage (*RSSR.sldasm*) into *car.sldasm*) and it locates it according to the given specifications using mates.

After the RSSR mechanism has been imported into the assembly, the user can add animation features to check the behavior of the synthesized linkage. If the user wants to add wheels driven by the output crank, mates need to be added so that the wheel turns according to the steering linkage.

This process needs to be repeated for each RSSR mechanism, i.e.: in a standard vehicle there exists a left and right steering mechanism. The left and right linkages are connected by a rod (not generated by our add-in) and this rod is driven by the steering wheel. In this case, the synthesis process must be repeated twice.

Table 8.1: Specified geometry, left and right side.

	left side			right side		
Point	x	y	z	x	y	z
$\mathbf{Z}_{1,a}$	-0.8603	0.6826	1.2646	1.8769	-1.0895	-0.7850
$\mathbf{Z}_{1,b}$	-0.7281	0.7862	0.9015	2.4769	-1.0895	-0.7850
$\mathbf{S}_{1,0}$	-0.8034	1.6072	1.1082	2.1769	-0.7895	-0.7850
$\mathbf{Z}_{2,a}$	0.9181	1.6265	0.8974	1.4269	-0.9395	-1.5000
$\mathbf{Z}_{2,b}$	0.7368	1.4575	0.5834	1.4269	-1.4395	-1.5500
$\mathbf{P}_{REF}$	0.5879	2.0962	0.3255	1.8269	-1.1895	-1.5250

8.3 Numerical Results

In this example the left side steering mechanism has its input crank's axis horizontal and parallel to the wheel in the straight position. The rotation axis of the output crank is tilted. For simplicity purposes, the corresponding input/output pairs of angles are equal ($\theta_1 = \phi_1$, $\theta_2 = \phi_2$), we recall that the designer can choose any combination of angles. The input data used for left side mechanism in this example is described in Table 8.1 and Table 8.2. The location of the corresponding input data points and angles is shown in the modeled supporting part in Fig. 1.7. A complete model of the steering mechanism support is shown in Fig. 8.1. Note that the right side is a mirror image of the left side.

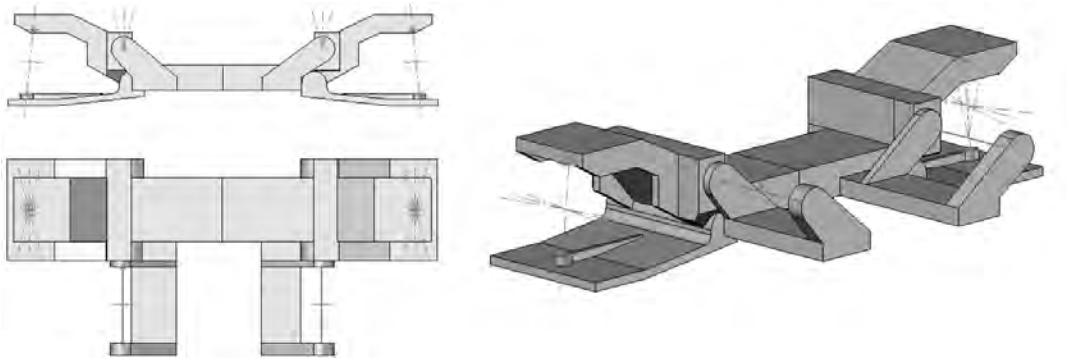

Figure 8.1: Supporting part (views: FRONT, TOP, ISOMETRIC).

The coordinates of the ball joints of the left side mechanism are listed in Table 8.3. The lengths of the three components (see Fig. 5.2) are shown Table 8.4. The resulting RSSR steering linkage for the left wheel modeled with the CAD software

Table 8.2: Specified angles.

	left side		right side	
Angle	Pos. 1	Pos. 2	Pos. 1	Pos. 2
θ	-15^o	15^o	-15^o	15^o
ϕ	-15^o	15^o	-15^o	15^o

Table 8.3: Resulting ball joints positions.

	left side			right side		
Point	x	y	z	x	y	z
$\mathbf{S}_{1,0}$	2.1769	-0.7895	0.7850	2.1769	-0.7895	-0.7850
$\mathbf{S}_{1,1}$	2.1769	-0.7997	0.8626	2.1769	-0.7997	-0.7074
$\mathbf{S}_{1,2}$	2.1769	-0.7997	0.7074	2.1769	-0.7997	-0.8626
$\mathbf{S}_{2,0}$	1.7451	-1.1895	1.5250	1.7451	-1.1895	-1.5250
$\mathbf{S}_{2,1}$	1.7343	-1.1813	1.6070	1.7343	-1.1977	-1.4430
$\mathbf{S}_{2,2}$	1.7343	-1.1977	1.4430	1.7343	-1.1813	-1.6070

is shown in Fig. 8.2. The complete steering mechanism is shown for three positions in Fig. 8.3. A modeled car using the steering mechanism is shown in Fig. 8.4.

Table 8.4: Link lengths.

	left side	right side
r_1	0.8500	0.8500
r_2	0.5805	0.5805
r_3	1.6476	1.6476

Table 8.5: Frame center and vector coordinates, left and right side.

	left side			right side		
Point	x	y	z	x	y	z
O	2.1769	-1.0895	0.7850	2.1769	-1.0895	-0.7850
$\mathbf{x}_1$	0.0000	1.0000	0.0000	0.0000	1.0000	0.0000
$\mathbf{y}_1$	0.0000	0.0000	1.0000	0.0000	0.0000	1.0000
$\mathbf{z}_1$	1.0000	0.0000	0.0000	1.0000	0.0000	0.0000
C	1.4269	-1.1895	1.5250	1.4269	-1.1895	-1.5250
$\mathbf{x}_2$	1.0000	0.0000	0.0000	1.0000	0.0000	0.0000
$\mathbf{y}_2$	0.0000	0.0995	0.9950	0.0000	0.0995	0.9950
$\mathbf{z}_2$	0.0000	-0.9950	0.0995	0.0000	-0.9950	0.0995

Figure 8.2: Left wheel steering mechanism.

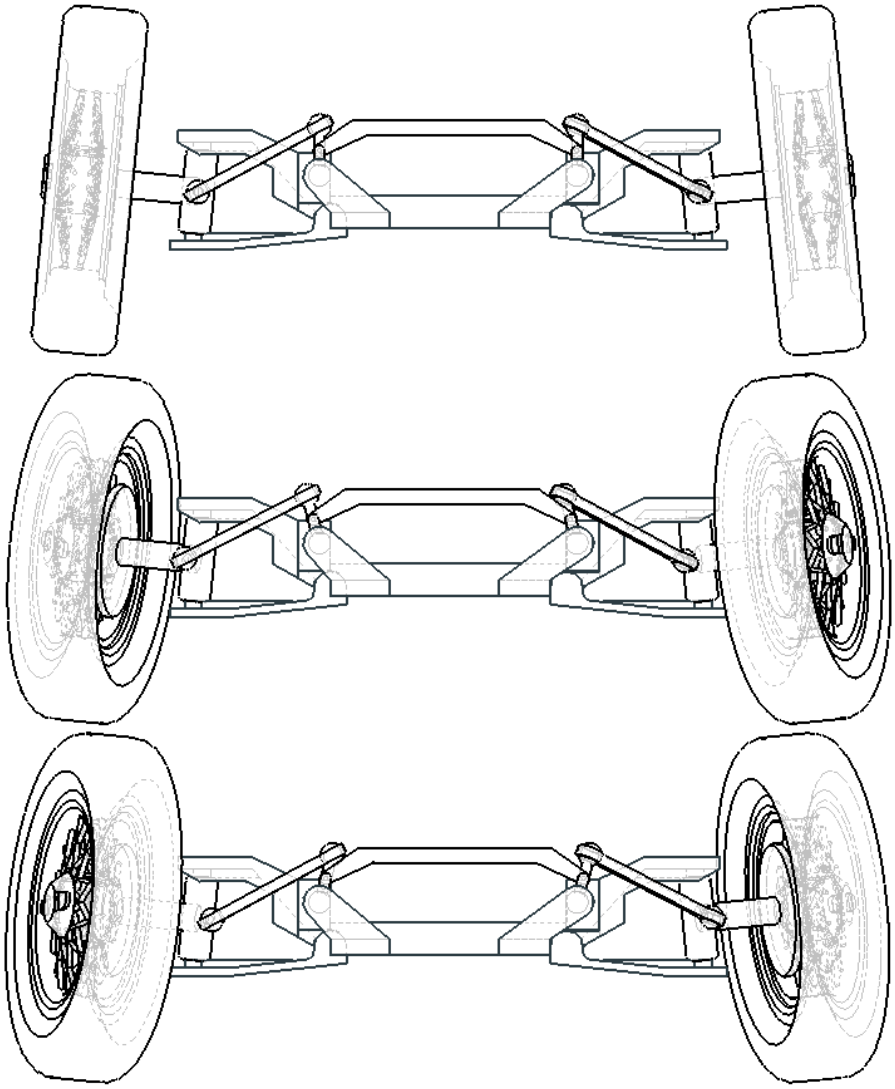

Figure 8.3: Three positions of the synthesized steering linkage.

Figure 8.4: Car model with synthesized steering mechanism.

Bibliography

[1] Hunt, K. H., 1978. *Kinematic Geometry of Mechanisms.* Clarendon Press, Oxford.

[2] McCarthy, J. M., 2000. *Geometric Design of Linkages.* Springer Verlag, New York.

[3] Simionescu, P. A., Smith, M. R., and Tempea, I., 2000. "Synthesis and analysis of the two loop translational input steering mechanism". *Mechanism and Machine Theory,* **35**(7), July, pp. 927–943.

[4] Simionescu, P. A., and Beale, D., 2002. "Optimum synthesis of the four-bar function generator in its symmetric embodiment: the ackermann steering linkage". *Mechanism and Machine Theory,* **37**(12), December, pp. 1487–1504.

[5] Dvali, R. R., and Aleksishvili, N. I., 1971. "Design for an automobile steering gear as a spherical four-bar linkage". *Journal of Mechanisms,* **6**(2), May, pp. 167–175.

[6] Molian, S., 1973. "Kinematics and dynamics of the rssr mechanism". *Mechanism and Machine Theory,* **8**(2), Summer, pp. 271–282.

[7] Duffy, J., and Gilmartin, M. J., 1978. "Displacement analysis of the generalized rssr mechanism". *Mechanism and Machine Theory,* **13**(5), Summer, pp. 533–541.

[8] Suh, C. H., and Radcliffe, C. W., 1978. *Kinematics and Mechanism Design.* John Wiley and Sons, Inc., New York.

[9] Sandor, G. N., and Erdman, A. G., 1984. *Advanced Mechanism Design: Analysis and Synthesis Vol. II.* Prentice-Hall, Inc., New Jersey.

[10] McCarthy, J. M., 1990. *Introduction to Theoretical Kinematics.* The MIT Press, Massachusetts.

[11] Murray, R. M., Sastry, S. S., and Li, Z., 1994. *A Mathematical Introduction to Robotic Manipulation.* CRC Press, Inc., Boca Raton, Florida.

[12] Tsai, L. W., 1999. *Robot Analysis: the Mechanics of Serial and Parallel Manipulators.* John Wiley and Sons, Inc., New York.

[13] Dukkipati, R. V., 2001. *Spatial mechanisms : analysis and synthesis.* CRC Press, Boca Raton.

[14] Waldron, K. J., and Kinzel, G. L., 2003. *Kinematics, Dynamics, and Design of Machinery, 2nd Edition.* John Wiley and Sons, Inc., New York.

[15] Denavit, J., and Hartenberg, R. S., 1955. "A kinematic notation for lower pair mechanisms based on matrices". *Journal of Applied Mechanics,* **22**(7), June, pp. 215–221.

[16] SolidWorks, 2005. Api support. on the WWW, April. URL `http://www.solidworks.com/pages/services/APISupport.html`.

[17] Marshall, D. L., 1999. *Applied Geometry For Computer Graphics.* Springer-Verlag, UK.

[18] Tickoo, S., 2003. *SolidWorks for Designers, Release 2003.*

Appendix A

Add-In Codes

A.1 *RSSR.exe*

A.1.1 Public Class MainForm

```
Inherits System.Windows.Forms.Form
Dim rssrObj As RSSR_Server.CNAT_RSSR            'RSSR calculations
server
Public WithEvents swApp As SldWorks.SldWorks    'SolidWorks session

Dim swModel As SldWorks.ModelDoc2
Dim swComp As SldWorks.Component2
Dim SelMgr As SldWorks.SelectionMgr

Dim x1(2) As Double            ' x-coordinates of S Joint 1
(innerball)
Dim y1(2) As Double            ' y-coordinates of S Joint 1
Dim z1(2) As Double            ' z-coordinates of S Joint 1
Dim x2(2) As Double            ' x-coordinates of S Joint 2
(outerball)
Dim y2(2) As Double            ' y-coordinates of S Joint 2
Dim z2(2) As Double            ' z-coordinates of S Joint 2
Dim theta(1) As Double         ' input angle specifications
Dim phi(1) As Double           ' output angle specifications
Dim InputAxisPt1(2) As Double  ' coordinates of upper point of axis
1 (R Joint 1)
Dim InputAxisPt2(2) As Double  ' coordinates of lower point of axis
1 (R Joint 1)
Dim RefPt(2) As Double         ' coordinates of kingpin point on
axis 2
Dim OutputAxisPt1(2) As Double ' coordinates of upper point of axis
2 (R Joint 2)
Dim OutputAxisPt2(2) As Double ' coordinates of lower point of axis
2 (R Joint 2)
Dim innerball(2) As Double     ' coordinates of outerball (S Joint
2) at theta=0
Dim Length1 As Double          ' distance [m] between input axis
and ball joint 1
Dim Length2 As Double          ' distance [m] between output axis
and ball joint 2
Dim Length3 As Double          ' length [m] of the rod that
connects Ball 1 and Ball 2
Dim i As Integer               ' index for loop routines
'---------------
```

```
Public swObj As Object = Nothing
Public swPointBodyFrame(2) As Double
Public swPointSpaceFrame As Object = Nothing
```

A.1.2 Private Sub GetButton1_Click

```
Private Sub GetButton1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles GetButton1.Click

'call sub function
GetPointCoords()

'Fill in the text boxes

'in space frame
TextBoxI1x.Text() = CStr(swPointSpaceFrame(0))
TextBoxI1y.Text() = CStr(swPointSpaceFrame(1))
TextBoxI1z.Text() = CStr(swPointSpaceFrame(2))

End Sub
```

A.1.3 Public Sub GetButton1help_Click

```
Public Sub GetButton1help_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles GetButton1help.Click
'call sub function
GetPointCoordsHelp()

'Fill in the text boxes

'in space frame
TextBoxI1x.Text() = CStr(swPointSpaceFrame(0))
TextBoxI1y.Text() = CStr(swPointSpaceFrame(1))
TextBoxI1z.Text() = CStr(swPointSpaceFrame(2))

End Sub
```

A.1.4 Public Function GetPointCoords

```
Public Function GetPointCoords() As Double
Dim swObj As Object 'general SW object
Dim swSelMgr As SldWorks.SelectionMgr
Dim OpenForm As PointImporter

swModel = swApp.ActiveDoc
swSelMgr = swModel.SelectionManager
swObj = swSelMgr.GetSelectedObject5(1)

OpenForm = New PointImporter
OpenForm.swB2S(swObj, swApp)
swPointSpaceFrame = OpenForm.swPointSpaceFrame
swPointBodyFrame = OpenForm.swPointBodyFrame
OpenForm = Nothing
End Function
```

A.1.5 Public Function GetPointCoordsHelp

```
Public Function GetPointCoordsHelp() As Double

Dim OpenForm As PointImporter
OpenForm = New PointImporter
OpenForm.ShowDialog()
swPointBodyFrame = OpenForm.swPointBodyFrame
swPointSpaceFrame = OpenForm.swPointSpaceFrame
OpenForm = Nothing

End Function
```

A.1.6 Public Sub ButtonCalc_Click

```
Public Sub ButtonCalc_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles ButtonCalc.Click
' convert angle specs in degrees to radians
theta(0) = CDbl(TextBoxIAng1.Text()) * 3.14159 / 180
theta(1) = CDbl(TextBoxIAng2.Text()) * 3.14159 / 180
phi(0) = CDbl(TextBoxOAng1.Text()) * 3.14159 / 180
phi(1) = CDbl(TextBoxOAng2.Text()) * 3.14159 / 180

' get inputs from text boxes, convert to doubles and assign to
variable names
InputAxisPt1(0) = CDbl(TextBoxI1x.Text())
InputAxisPt1(1) = CDbl(TextBoxI1y.Text())
InputAxisPt1(2) = CDbl(TextBoxI1z.Text())
InputAxisPt2(0) = CDbl(TextBoxI2x.Text())
InputAxisPt2(1) = CDbl(TextBoxI2y.Text())
InputAxisPt2(2) = CDbl(TextBoxI2z.Text())

RefPt(0) = CDbl(TextBoxRefPtx.Text())
RefPt(1) = CDbl(TextBoxRefPty.Text())
RefPt(2) = CDbl(TextBoxRefPtz.Text())

OutputAxisPt1(0) = CDbl(TextBoxO1x.Text())
OutputAxisPt1(1) = CDbl(TextBoxO1y.Text())
OutputAxisPt1(2) = CDbl(TextBoxO1z.Text())
OutputAxisPt2(0) = CDbl(TextBoxO2x.Text())
OutputAxisPt2(1) = CDbl(TextBoxO2y.Text())
OutputAxisPt2(2) = CDbl(TextBoxO2z.Text())

innerball(0) = CDbl(TextBoxBJx.Text())
innerball(1) = CDbl(TextBoxBJy.Text())
innerball(2) = CDbl(TextBoxBJz.Text())

rssrObj = New RSSR_Server.CNAT_RSSR

' call the rssr_server program to do the calculations
rssrObj.put_data2(InputAxisPt1(0), InputAxisPt2(0),
OutputAxisPt1(0), OutputAxisPt2(0), innerball(0), RefPt(0))
rssrObj.put_theta2(theta(0))
rssrObj.put_phi2(phi(0))
rssrObj.HCalculate2(x1(0), y1(0), z1(0), x2(0), y2(0), z2(0))

'calculate distance from Input Axis to Ball Joint 1
Dim p0(2) As Object 'copy of S Joint 1 (outerball)
Dim p1(2) As Object 'copy of InputAxisPt1(3)
Dim p2(2) As Object 'copy of InputAxisPt2(3)
Dim ControlParam As Double

p0(0) = innerball(0) 'ball joint 1 location
p0(1) = innerball(1)
p0(2) = innerball(2)

p1(0) = InputAxisPt1(0)
p1(1) = InputAxisPt1(1)
p1(2) = InputAxisPt1(2)

p2(0) = InputAxisPt2(0)
p2(1) = InputAxisPt2(1)
p2(2) = InputAxisPt2(2)
ControlParam = NormVB(SubsVB(p2, p1))

If NormVB(SubsVB(p2, p1)) = 0 Then  'Norm = 0 means something is
wrong
Dim Msg, Style, Title, Response As Object
Msg = "Input axis has norm = 0, review your selection"
Style = vbOKOnly
Title = "Error"
```

```
Call MsgBox(Msg, Style, Title) ' Display error message
Response = MsgBox(Msg, Style, Title)
'GoTo
Else
Length1 = NormVB(CrossVB(SubsVB(p2, p1), SubsVB(p1, p0))) /
NormVB(SubsVB(p2, p1))
System.Diagnostics.Debug.Write(Chr(10) &
"Distance(SJoint1,IntputAxis) = " & CStr(Format(Length1,
"0.0000")))
End If

'calculate distance from Output Axis to Ball Joint 2

p0(0) = x2(0) 'ball joint 2 location
p0(1) = y2(0)
p0(2) = z2(0)

p1(0) = OutputAxisPt1(0)
p1(1) = OutputAxisPt1(1)
p1(2) = OutputAxisPt1(2)

p2(0) = OutputAxisPt2(0)
p2(1) = OutputAxisPt2(1)
p2(2) = OutputAxisPt2(2)
ControlParam = NormVB(SubsVB(p2, p1))

If NormVB(SubsVB(p2, p1)) = 0 Then  'Norm = 0 means something is
wrong
Dim Response As Object

Response = MsgBox("Output axis has norm = 0, review your
selection", vbOKOnly, "Error") ' Display error message
'GoTo
Else
Length2 = NormVB(CrossVB(SubsVB(p2, p1), SubsVB(p1, p0))) /
NormVB(SubsVB(p2, p1))
System.Diagnostics.Debug.Write(Chr(10) &
"Distance(SJoint2,OutputAxis) = " & CStr(Format(Length2, "0.0000"))
& "m")

End If

'calculate length of the rod 'Length3'
Length3 = (((x1(0) - x2(0)) ^ 2 + (y1(0) - y2(0)) ^ 2 + (z1(0) -
z2(0)) ^ 2) ^ 0.5)
System.Diagnostics.Debug.Write(Chr(10) & "TieRod length = " &
CStr(Format(Length3, "0.0000")) & "m")

' display the results in the output area
TextBoxBJ1StartX.Text() = CStr(x1(0))
TextBoxBJ1StartY.Text() = CStr(y1(0))
TextBoxBJ1StartZ.Text() = CStr(z1(0))

TextBoxBJ1MidX.Text() = CStr(x1(1))
TextBoxBJ1MidY.Text() = CStr(y1(1))
TextBoxBJ1MidZ.Text() = CStr(z1(1))

TextBoxBJ1EndX.Text() = CStr(x1(2))
TextBoxBJ1EndY.Text() = CStr(y1(2))
TextBoxBJ1EndZ.Text() = CStr(z1(2))

TextBoxBJ2StartX.Text() = CStr(x2(0))
TextBoxBJ2StartY.Text() = CStr(y2(0))
TextBoxBJ2StartZ.Text() = CStr(z2(0))

TextBoxBJ2MidX.Text() = CStr(x2(1))
TextBoxBJ2MidY.Text() = CStr(y2(1))
```

```
TextBoxBJ2MidZ.Text() = CStr(z2(1))

TextBoxBJ2EndX.Text() = CStr(x2(2))
TextBoxBJ2EndY.Text() = CStr(y2(2))
TextBoxBJ2EndZ.Text() = CStr(z2(2))

Length1TxtBox.Text() = Length1
Length2TxtBox.Text() = Length2
Length3TxtBox.Text() = Length3

End Sub
```

A.1.7 Private Sub DrawButton_Click

```
Private Sub DrawButton_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles DrawButton.Click

'*****************************************************************
'* Every time the user inserts a new RSSR block,
'* the assembly must change configuration. The 3 parts
'* of the assembly ALSO must change config. For example,
'* the 3rd RSSR block could be RSSR-7 but must have
'* "config3" as the active configuration. TieRod.Sldprt,
'* InputRod.sldprt and OutputRod.sldprt must also have
'* "config3" as the active configuration.
'*
'* This is the only way to have the same sub-assembly several time
'* in the same assembly with different length parts and
orientations
'*****************************************************************

Dim swModel As SldWorks.ModelDoc2 = Nothing      'Active open model
Dim swComponent As SldWorks.Component2 = Nothing
Dim swRootComp As SldWorks.Component2 = Nothing  'Root Component
Dim swChildComp As SldWorks.Component2 = Nothing 'Child Component,
the father is the Root Comp.
Dim swConf As SldWorks.configuration = Nothing   'Active
configuration

Dim Plane As Object = Nothing
Dim varChildComp As Object = Nothing 'variant with ALL the
components in the assembly
Dim boolstatus As Boolean = False
Dim swTitle As String = "noTitle"
Dim swPath As String = "noPath"
Dim swFileName As String = "noName"
Dim ActualChar As String = ""
Dim ConfigXX As String = ""
Dim RSSRcount As Integer = 1 'to know what configuration to use
(only 4 diff. configs are allowed)
Dim i As Integer

swModel = swApp.ActiveDoc

'*****************************************************************
'* First of all we must have a folder named "RSSR" under the folder
where
'* the main user assembly is. After that, we must OPEN the
RSSR.SLDASM to load
'* it in memory and import it later.
'*****************************************************************

'*****************************************************************
'* We can get the entire path+filename but we want to separate them
'*****************************************************************

swPath = swModel.GetPathName
i = swPath.Length

While Not (ActualChar = "\") And Not (i = 0)
    ActualChar = swPath.Chars(i - 1)
    i = i - 1
End While

swFileName = swPath.Substring(i + 1, swPath.Length - (i + 1))
swPath = swPath.Substring(0, i + 1)
System.Diagnostics.Debug.Write(Chr(10) + swFileName + " @ " +
swPath)
'*********at this point we have a path and a separated
filename*********
```

```
'*************************************************************
'* We need to get the name of the current doc.
'* This is usefull for further use of SelectByID
'* GetTitle returns name that appeats on the top part of the
program
'* (inlcuding the extension) We want the name WITHOUT extension
'*************************************************************

swTitle = swModel.GetTitle   '
swTitle = swTitle.Substring(0, swTitle.IndexOf(".")) '

'***************************************
'* we want to know how many RSSR blocks
'* have been already inserted BEFORE
'* we insert a new block
'***************************************

swConf = swModel.GetActiveConfiguration
swRootComp = swConf.GetRootComponent
varChildComp = swRootComp.GetChildren

For i = 0 To UBound(varChildComp)
swChildComp = varChildComp(i)

'*************************************************************
'* We must change the active configuration in the ORIGINAL
'* RSSR.sldasm BEFORE we import it to the main USER ASSEMBLY. If
'* component name starts with RSSRXXXXXX then add 1 to the
RSSRcount
'* if there is a RSSR-1 the next (RSSR-2) will use 'config2'
'*************************************************************

If swChildComp.Name2.Substring(0, 4) = "RSSR" Then
RSSRcount = RSSRcount + 1
ConfigXX = String.Concat("config", RSSRcount)
'ConfigXX is the configuration we need to set up
'as active to import the proper RSSR
End If

'TraverseComponent(swChildComp, nLevel + 1) 'we could look in a
deaper level
System.Diagnostics.Debug.Write(Chr(10) + swChildComp.Name2 + " <" +
swChildComp.ReferencedConfiguration + ">")
Next i

'1.- Open Document RSSR.sldasm to load it in memory

'*************************************************
'* here we change the main model doc to RSSR.sldasm
'* and we change the configuration to the RSSRCount
'*************************************************

swModel = swApp.OpenDoc4(swPath + "RSSR\RSSR.sldasm",
SwConst.swDocumentTypes_e.swDocASSEMBLY, 0, "",
SwConst.swFileLoadError_e.swGenericError)
swModel = swApp.ActivateDoc(swPath + "RSSR\RSSR.sldasm")
' Show "configXXXX" and make it the active configuration
swModel.ShowConfiguration2(String.Concat("config", RSSRcount))
swModel.ViewZoomtofit2()
swModel.EditRebuild()

'2.- import RSSR.sldasm at x=0, y=0, z=0

'*************************************************************
'* For every SESSION, there should be a RSSR subfolder.
'* From there the user can import 9 RSSR in one assembly
'* The imported block must be set up to FLEXIBLE and FULLY RESOLVED
```

```
'*****************************************************************

swModel = swApp.ActivateDoc(swFileName) 'swFileName =
"steering.SLDASM"
swModel = swApp.ActiveDoc
swComponent = swModel.AddComponent2(swPath + "RSSR\RSSR.sldasm",
InputAxisPt1(0), InputAxisPt1(1), InputAxisPt1(2))
swModel.EditRebuild()
swApp.CloseDoc("RSSR.sldasm")   'after adding, we can close the
imported file

'This property sets the active configuration used by this component
swComponent.ReferencedConfiguration = String.Concat("config",
RSSRcount)
System.Diagnostics.Debug.Write(Chr(10) +
String.Concat(swComponent.Name2, "@", swTitle))
boolstatus =
swModel.Extension.SelectByID(String.Concat(swComponent.Name2, "@",
swTitle), "COMPONENT", 0, 0, 0, False, 0, Nothing)
boolstatus =
swModel.CompConfigProperties3(SwConst.swComponentSuppressionState_e
.swComponentFullyResolved,
SwConst.swComponentSolvingOption_e.swComponentFlexibleSolving,
True, True)
swModel.ClearSelection()
swModel.EditRebuild()

'3.- Open the 3 components to show "configXXXX" and make it the
active configuration

'*****************
'* Open InputRod *
'*****************

boolstatus = swModel.Extension.SelectByID(swComponent.Name2 + "@" +
swTitle + "/InputRod-1@RSSR", "COMPONENT", 0, 0, 0, False, 0,
Nothing)
swModel.OpenCompFile()
swModel = swApp.ActivateDoc(swPath + "RSSR\InputRod.SLDPRT")
swModel.ShowConfiguration2(String.Concat("config", RSSRcount))
'swModel.ShowNamedView2("",
SwConst.swStandardViews_e.swIsometricView) 'Isometric
swModel.ViewZoomtofit2()
swModel.EditRebuild()

'1.- Resize Input Ball Joint Location distance to 'Length1'
boolstatus = swModel.Extension.SelectByID("SJointPoint", "SKETCH",
0, 0, 0, False, 0, Nothing)
boolstatus =
swModel.Extension.SelectByID("D1@SJointPoint@InputRod.SLDPRT",
"DIMENSION", 0, 0, 0, False, 0, Nothing)
swModel.Parameter("D1@SJointPoint").SystemValue = Length1
swModel.EditRebuild()
write(Chr(10) + "Length from InputAxis to SBJ1 has been resized to
" & Length1 & " m")
'swModel.ShowNamedView2("",
SwConst.swStandardViews_e.swIsometricView) 'Isometric
swModel.ViewZoomtofit2()
swModel.EditRebuild()
swApp.CloseDoc("InputRod.SLDPRT")

swModel = swApp.ActivateDoc(swFileName) 'swFileName =
"steering.SLDASM"
swModel = swApp.ActiveDoc

'******************
'* Open OutputRod *
'******************
```

```
boolstatus = swModel.Extension.SelectByID(swComponent.Name2 + "@" +
swTitle + "/OutputRod-1@RSSR", "COMPONENT", 0, 0, 0, False, 0,
Nothing)
swModel.OpenCompFile()
swModel = swApp.ActivateDoc(swPath + "RSSR\OutputRod.SLDPRT")
swModel.ShowConfiguration2(String.Concat("config", RSSRcount))
'swModel.ShowNamedView2("",
SwConst.swStandardViews_e.swIsometricView) 'Isometric
swModel.ViewZoomtofit2()
swModel.EditRebuild()

'2.- Resize Output Ball Joint Location distance to 'Length2'
boolstatus = swModel.Extension.SelectByID("SJointPoint", "SKETCH",
0, 0, 0, False, 0, Nothing)
boolstatus =
swModel.Extension.SelectByID("D1@SJointPoint@OutputRod.SLDPRT",
"DIMENSION", -0.00753819604718, 0.4155276003872, -0.03298281800036,
False, 0, Nothing)
swModel.Parameter("D1@SJointPoint").SystemValue = Length2
swModel.EditRebuild()
write(Chr(10) & "Length from OutputAxis to SBJ2 has been resized to
" & Length2 & " m")
'swModel.ShowNamedView2("",
SwConst.swStandardViews_e.swIsometricView) 'Isometric
swModel.ViewZoomtofit2()
swModel.EditRebuild()
swApp.CloseDoc("OutputRod.SLDPRT")

swModel = swApp.ActivateDoc(swFileName) 'swFileName =
"steering.SLDASM"
swModel = swApp.ActiveDoc

'***************
'* Open TieRod *
'***************

boolstatus = swModel.Extension.SelectByID(swComponent.Name2 + "@" +
swTitle + "/TieRod-1@RSSR", "COMPONENT", 0, 0, 0, False, 0,
Nothing)
swModel.OpenCompFile()
swModel = swApp.ActivateDoc(swPath + "RSSR\TieRod.SLDPRT")
swModel.ShowConfiguration2(String.Concat("config", RSSRcount))
'swModel.ShowNamedView2("",
SwConst.swStandardViews_e.swIsometricView) 'Isometric
swModel.ViewZoomtofit2()
swModel.EditRebuild()

'3.- Resize Tie Rod distance between Joints to 'Length3'

boolstatus = swModel.Extension.SelectByID("AxisLength", "SKETCH",
0, 0, 0, False, 0, Nothing)
boolstatus =
swModel.Extension.SelectByID("LengthBetweenSJoints@AxisLength",
"DIMENSION", -0.01390101330837, 0.06195296002655, -
0.04804187843797, False, 0, Nothing)
 swModel.Parameter("LengthBetweenSJoints@AxisLength").SystemValue =
Length3
 swModel.EditRebuild()
 write("Length of TieRod has been resized to   " & Length3 & " m")
 'swModel.ShowNamedView2("",
SwConst.swStandardViews_e.swIsometricView) 'Isometric
 swModel.ViewZoomtofit2()
 swModel.EditRebuild()
 swApp.CloseDoc("TieRod.SLDPRT")

 swModel = swApp.ActivateDoc(swFileName) 'swFileName =
"steering.SLDASM"
```

```
 swModel = swApp.ActiveDoc

'*************************************************************
'* At this point we have inserted a new RSSR-X block (fully
resolved + flexible)
'* But this block is NOT complete!!!!!!!!!!!!!!!!!
'*
'* To completely define this block, the following are needed:
'*
'* - location of InputAxis [2 POINTS]
'* - position of OutputAxis [2 POINTS]
'* - location of SJoint1[POINT]
'* - reference point in output Axis (0 angle ref)
'* - positioning of OutputRod wrt OutputAxis (rotation plane)
'* - location of SJoint2 point
'*************************************************************

'*****************************************************************
'* to change a length it must be changed within the ORIGINAL PART:
'*      InputRod.SLDPRT / OutputRod.SLDPRT / TieRod.SLDPRT
'*               (and the SPECIFIC CONFIGURATION)
'*****************************************************************

'1.- Relocate InputRod Rotation Axis

'****************************************
'* type could be VERTEX or SKETCHPOINT,
'* but what matters are the coords
'****************************************

write(Chr(10) + CStr(InputAxisPt1(0)) + " " + CStr(InputAxisPt1(1))
+ " " + CStr(InputAxisPt1(2)))

boolstatus = swModel.Extension.SelectByID("", "EXTSKETCHPOINT",
InputAxisPt1(0), InputAxisPt1(1), InputAxisPt1(2), False, 0,
Nothing)
If boolstatus = False Then
boolstatus = swModel.Extension.SelectByID("", "VERTEX",
InputAxisPt1(0), InputAxisPt1(1), InputAxisPt1(2), False, 0,
Nothing)
End If
boolstatus = swModel.Extension.SelectByID("Line2@RotationAxis@" +
swComponent.Name2 + "@" + swTitle + "/InputRod-1@RSSR",
"EXTSKETCHSEGMENT", 0, 0, 0, True, 0, Nothing)
swModel.AddMate(0, -1, False, 0, 0)
swModel.ClearSelection()

boolstatus = swModel.Extension.SelectByID("", "EXTSKETCHPOINT",
InputAxisPt2(0), InputAxisPt2(1), InputAxisPt2(2), False, 0,
Nothing)
If boolstatus = False Then
boolstatus = swModel.Extension.SelectByID("", "VERTEX",
InputAxisPt2(0), InputAxisPt2(1), InputAxisPt2(2), False, 0,
Nothing)
End If
boolstatus = swModel.Extension.SelectByID("Line2@RotationAxis@" +
swComponent.Name2 + "@" + swTitle + "/InputRod-1@RSSR",
"EXTSKETCHSEGMENT", 0, 0, 0, True, 0, Nothing)
swModel.AddMate(0, -1, False, 0, 0)
swModel.ClearSelection()

swModel.EditRebuild()

'2.- Position rotation plane1
boolstatus = swModel.Extension.SelectByID("Front@" +
swComponent.Name2 + "@" + swTitle + "/InputRod-1@RSSR", "PLANE", 0,
0, 0, False, 0, Nothing)
boolstatus = swModel.Extension.SelectByID("", "EXTSKETCHPOINT",
```

```
innerball(0), innerball(1), innerball(2), True, 0, Nothing)
If boolstatus = False Then
boolstatus = swModel.Extension.SelectByID("", "VERTEX",
innerball(0), innerball(1), innerball(2), True, 0, Nothing)
End If
swModel.AddMate(0, -1, False, 0, 0)
swModel.ClearSelection()

swModel.EditRebuild()

'3.- Relocate OutputRod Rotation Axis

boolstatus = swModel.Extension.SelectByID("", "EXTSKETCHPOINT",
OutputAxisPt1(0), OutputAxisPt1(1), OutputAxisPt1(2), False, 0,
Nothing)
If boolstatus = False Then
boolstatus = swModel.Extension.SelectByID("", "VERTEX",
OutputAxisPt1(0), OutputAxisPt1(1), OutputAxisPt1(2), False, 0,
Nothing)
End If
boolstatus = swModel.Extension.SelectByID("Line1@RotationAxis@" +
swComponent.Name2 + "@" + swTitle + "/OutputRod-1@RSSR",
"EXTSKETCHSEGMENT", 0, 0, 0, True, 0, Nothing)
swModel.AddMate(0, -1, False, 0, 0)
swModel.ClearSelection()

boolstatus = swModel.Extension.SelectByID("", "EXTSKETCHPOINT",
OutputAxisPt2(0), OutputAxisPt2(1), OutputAxisPt2(2), False, 0,
Nothing)
If boolstatus = False Then
boolstatus = swModel.Extension.SelectByID("", "VERTEX",
OutputAxisPt2(0), OutputAxisPt2(1), OutputAxisPt2(2), False, 0,
Nothing)
End If
boolstatus = swModel.Extension.SelectByID("Line1@RotationAxis@" +
swComponent.Name2 + "@" + swTitle + "/OutputRod-1@RSSR",
"EXTSKETCHSEGMENT", 0, 0, 0, True, 0, Nothing)
swModel.AddMate(0, -1, False, 0, 0)
swModel.ClearSelection()

swModel.EditRebuild()

'4.- Position rotation plane2
write(Chr(10) + "Front@" + swComponent.Name2 + "@" + swTitle +
"/OutputRod-1@RSSR")
boolstatus = swModel.Extension.SelectByID("Front@" +
swComponent.Name2 + "@" + swTitle + "/OutputRod-1@RSSR", "PLANE",
0, 0, 0, False, 0, Nothing)
boolstatus = swModel.Extension.SelectByID("", "EXTSKETCHPOINT",
RefPt(0), RefPt(1), RefPt(2), True, 0, Nothing)
If boolstatus = False Then
boolstatus = swModel.Extension.SelectByID("", "VERTEX", RefPt(0),
RefPt(1), RefPt(2), True, 0, Nothing)
End If
swModel.AddMate(0, -1, False, 0, 0)
swModel.ClearSelection()
swModel.ViewZoomtofit2()
swModel.EditRebuild()

End Sub
```

A.1.8 Other Tools

```
Private Function CrossVB(ByVal u As Object, ByVal v As Object) As
Object
Dim vector(2) As Object
vector(0) = u(1) * v(2) - u(2) * v(1)
vector(1) = u(2) * v(0) - u(0) * v(2)
vector(2) = u(0) * v(1) - u(1) * v(0)
CrossVB = vector
End Function

Private Function SubsVB(ByVal u As Object, ByVal v As Object) As
Object
Dim vector(2) As Object
vector(0) = v(0) - u(0)
vector(1) = v(1) - u(1)
vector(2) = v(2) - u(2)
SubsVB = vector
End Function

Private Function AddVB(ByVal v1 As Object, ByVal v2 As Object) As
Object
Dim vector(2) As Object
vector(0) = v2(0) + v1(0)
vector(1) = v2(1) + v1(1)
vector(2) = v2(2) + v1(2)
AddVB = vector
End Function

Private Function DotVB(ByVal v1 As Object, ByVal v2 As Object) As
Double
Dim i As Integer
For i = LBound(v1) To UBound(v1)
DotVB = DotVB + v1(i) * v2(i)
Next i
End Function

Private Function NormVB(ByVal v1 As Object) As Double
'in .NET you must declare a default value for OPTIONAL objects
Dim vector As Object
vector = v1
Dim i As Integer
NormVB = Math.Sqrt(DotVB(vector, vector))
End Function

Private Function UnitVB(ByVal vector As Object) As Object
Dim norm As Double
norm = Math.Sqrt(vector(0) * vector(0) + vector(1) * vector(1) +
vector(2) * vector(2))
If norm = 0 Then Exit Function
vector(0) = vector(0) / norm
vector(1) = vector(1) / norm
vector(2) = vector(2) / norm
UnitVB = vector
End Function

Private Function write(ByVal text As String)
System.Diagnostics.Debug.Write(Chr(10) + text)
End Function
```

A.1.9 Point Importer

```
Imports SldWorks
Imports SwConst
Imports WindowsApplicationSW1.MainForm

Public Class PointImporter
Inherits System.Windows.Forms.Form

'Global SW variables
Public WithEvents swApp As SldWorks.SldWorks
Dim swModel As SldWorks.ModelDoc2
Dim swSelMgr As SldWorks.SelectionMgr
Dim swMathUtil As SldWorks.MathUtility

'general SW object
Public swObj As Object
Public swPointBodyFrame(2) As Double
Public swPointSpaceFrame(2) As Double

'other variables
Dim swDocumentTypes As Integer 'enum list can be found in
swconst.dll
Dim swSelectType As Integer 'enum list can be found in swconst.dll
```

A.1.10 Private Sub GetFromSW_Click

```
Private Sub GetFromSW_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles GetFromSW.Click
Dim NewString As String

'reset previous results
CodeTextBox.Text() = "code"
TypeTxtBox.Text() = "swConst type"
WhatToDoList.Items.Clear()
ResultListBox.Items.Clear()

'get the current Active Doc
swModel = swApp.ActiveDoc

While swModel Is Nothing
MsgBox("open a document" + Chr(10) + "and click again", vbOKOnly,
"Error")
Exit Sub
End While

swMathUtil = swApp.GetMathUtility 'load the math utility tool
swSelMgr = swModel.SelectionManager
swDocumentTypes = swModel.GetType 'chack what has been selected
swObj = swSelMgr.GetSelectedObject5(1)

Select Case swDocumentTypes

Case SwConst.swDocumentTypes_e.swDocNONE
NewString = "swDocNONE"
MsgBox("no active document", vbOKOnly, "Error")
Case SwConst.swDocumentTypes_e.swDocPART
NewString = "swDocPART"
Case SwConst.swDocumentTypes_e.swDocASSEMBLY
NewString = "swDocASSEMBLY"
Case SwConst.swDocumentTypes_e.swDocDRAWING
NewString = "swDocDRAWING"
Case SwConst.swDocumentTypes_e.swDocSDM
NewString = "swDocSDM"
Case Else
NewString = "not listed"

End Select
Me.Text = "Type Helper (" + NewString + ")"

swSelectType = swSelMgr.GetSelectedObjectType2(1)
CodeTextBox.Text() = swSelectType

If swSelectType = SwConst.swSelectType_e.swSelNOTHING Then
TypeTxtBox.Text() = "swSelNOTHING"
WhatToDoList.Items.Add("No selection")
Else

Select Case swSelectType
Case SwConst.swSelectType_e.swSelEDGES '1 edge is a FEATURE
TypeTxtBox.Text() = "swSelEDGES"

'WhatToDoList.Items.Add("GetClosestPointOn")
WhatToDoList.Items.Add("GetStartVertex") 'edge -> vertex
WhatToDoList.Items.Add("GetEndVertex") 'edge -> vertex
'WhatToDoList.Items.Add("GetMidPoint")

Case SwConst.swSelectType_e.swSelFACES '2
TypeTxtBox.Text() = "swSelFACES"

WhatToDoList.Items.Add("Select point/vertices please")

Case SwConst.swSelectType_e.swSelVERTICES '3
```

```
TypeTxtBox.Text() = "swSelVERTICES"

WhatToDoList.Items.Add("GetPoint")

Case SwConst.swSelectType_e.swSelDATUMPLANES ’4
TypeTxtBox.Text() = "swSelDATUMPLANES"

Case SwConst.swSelectType_e.swSelDATUMAXES ’5
TypeTxtBox.Text() = "swSelDATUMAXES"

Case SwConst.swSelectType_e.swSelDATUMPOINTS ’6
TypeTxtBox.Text() = "swSelDATUMPOINTS"

Case SwConst.swSelectType_e.swSelSKETCHES ’9
TypeTxtBox.Text() = "swSelSKETCHES"

Case SwConst.swSelectType_e.swSelSKETCHSEGS ’10
TypeTxtBox.Text() = "swSelSKETCHSEGS"

Case SwConst.swSelectType_e.swSelSKETCHPOINTS ’11
TypeTxtBox.Text() = "swSelSKETCHPOINTS"

WhatToDoList.Items.Add("TargetClick")

Case SwConst.swSelectType_e.swSelDIMENSIONS ’14
TypeTxtBox.Text() = "swSelDIMENSIONS"

WhatToDoList.Items.Add("Select edges,lines or points")

Case SwConst.swSelectType_e.swSelCOMPONENTS ’20
TypeTxtBox.Text() = "swSelCOMPONENTS"

WhatToDoList.Items.Add("Select edges,lines or points")

Case SwConst.swSelectType_e.swSelMATES ’21
TypeTxtBox.Text() = "swSelMATES"

WhatToDoList.Items.Add("Select edges,lines or points")

Case SwConst.swSelectType_e.swSelBODYFEATURES ’22
TypeTxtBox.Text() = "swSelBODYFEATURES"

WhatToDoList.Items.Add("Select edges,lines or points")

Case SwConst.swSelectType_e.swSelREFCURVES ’23
TypeTxtBox.Text() = "swSelREFCURVES"

Case SwConst.swSelectType_e.swSelEXTSKETCHSEGS ’24
TypeTxtBox.Text() = "swSelEXTSKETCHSEGS"

WhatToDoList.Items.Add("GetStartPoint2")
WhatToDoList.Items.Add("GetEndPoint2")

Case SwConst.swSelectType_e.swSelEXTSKETCHPOINTS ’25
TypeTxtBox.Text() = "swSelEXTSKETCHPOINTS"

WhatToDoList.Items.Add("GetSketchPoint")

Case SwConst.swSelectType_e.swSelMATEGROUPS ’33
TypeTxtBox.Text() = "swSelMATEGROUPS"

Case SwConst.swSelectType_e.swSelPOINTREFS ’41
TypeTxtBox.Text() = "swSelPOINTREFS"

Case SwConst.swSelectType_e.swSelREFEDGES ’51
TypeTxtBox.Text() = "swSelREFEDGES"

WhatToDoList.Items.Add("GetStartVertex")
```

```
WhatToDoList.Items.Add("GetEndVertex")

Case SwConst.swSelectType_e.swSelMIDPOINTS ’59
TypeTxtBox.Text() = "swSelMIDPOINTS"

Case SwConst.swSelectType_e.swSelCOORDSYS ’61
TypeTxtBox.Text() = "swSelCOORDSYS"

Case SwConst.swSelectType_e.swSelSURFACEBODIES ’75
TypeTxtBox.Text() = "swSelSURFACEBODIES"

Case SwConst.swSelectType_e.swSelSOLIDBODIES ’76
TypeTxtBox.Text() = "swSelSOLIDBODIES"

WhatToDoList.Items.Add("GetType")
’type = Body2.GetType ( ) ’This method gets the type of the body.

Case SwConst.swSelectType_e.swSelCENTERLINES ’103
TypeTxtBox.Text() = "swSelCENTERLINES"

Case Else    ’ Other values.
TypeTxtBox.Text() = "UNKNOWN (OTHER)"

End Select
End If
End Sub
```

A.1.11 Generate List

```
Private Sub PictureBox1_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles PictureBox1.Click
Dim swComp As SldWorks.Component2
Dim TotalSelectedCount As Integer = 0
Dim i As Single = 0

TotalSelectedCount = WhatToDoList.SelectedItems.Count()

If TotalSelectedCount > 0 Then
Dim FunctionString As String = ""
For i = 0 To TotalSelectedCount - 1
FunctionString = WhatToDoList.SelectedItems(i).ToString()

swObj = swSelMgr.GetSelectedObject5(1)

Select Case FunctionString

Case "TargetClick"
swPointSpaceFrame = TargetClick()
swPointBodyFrame = swPointSpaceFrame
ResultListBox.Items.Add(FunctionString + " (s): x=" +
CStr(Format(swPointSpaceFrame(0), "0.00")) + " y=" +
CStr(Format(swPointSpaceFrame(1), "0.00")) + " z=" +
CStr(Format(swPointSpaceFrame(2), "0.00")))

Case "GetClosestPointOn"
ResultListBox.Items.Add(FunctionString + " (s): x=" +
CStr(Format(swPointSpaceFrame(0), "0.00")) + " y=" +
CStr(Format(swPointSpaceFrame(1), "0.00")) + " z=" +
CStr(Format(swPointSpaceFrame(2), "0.00")))

Case "GetStartVertex"

'*****************************************
'* the selected object is an EDGE,         *
'* that 's why we have to specify an       *
'* optional parameter to swB2S to          *
'* treat the object as a VERTEX            *
'*****************************************

swB2S(swGetStartVertex(swObj), ,
SwConst.swSelectType_e.swSelVERTICES)
ResultListBox.Items.Add(FunctionString + " (s): x=" +
CStr(Format(swPointSpaceFrame(0), "0.00")) + " y=" +
CStr(Format(swPointSpaceFrame(1), "0.00")) + " z=" +
CStr(Format(swPointSpaceFrame(2), "0.00")))

Case "GetEndVertex"
swB2S(swGetEndVertex(swObj), ,
SwConst.swSelectType_e.swSelVERTICES)
ResultListBox.Items.Add(FunctionString + " (s): x=" +
CStr(Format(swPointSpaceFrame(0), "0.00")) + " y=" +
CStr(Format(swPointSpaceFrame(1), "0.00")) + " z=" +
CStr(Format(swPointSpaceFrame(2), "0.00")))

Case "GetStartPoint2"

'*****************************************
'* the selected object is a SKETCHSEGMENT, *
'* that 's why we have to specify an       *
'* optional parameter to swB2S to          *
'* treat the object as a SKETCHPOINT       *
'*****************************************

swB2S(swGetStartPoint2(swObj), ,
SwConst.swSelectType_e.swSelEXTSKETCHPOINTS)
```

```
ResultListBox.Items.Add(FunctionString + " (s): x=" +
CStr(Format(swPointSpaceFrame(0), "0.00")) + " y=" +
CStr(Format(swPointSpaceFrame(1), "0.00")) + " z=" +
CStr(Format(swPointSpaceFrame(2), "0.00")))

Case "GetEndPoint2"

swB2S(swGetEndPoint2(swObj), ,
SwConst.swSelectType_e.swSelEXTSKETCHPOINTS)
ResultListBox.Items.Add(FunctionString + " (s): x=" +
CStr(Format(swPointSpaceFrame(0), "0.00")) + " y=" +
CStr(Format(swPointSpaceFrame(1), "0.00")) + " z=" +
CStr(Format(swPointSpaceFrame(2), "0.00")))

Case "GetPoint"
swB2S(swObj)
ResultListBox.Items.Add(FunctionString + " (s): x=" +
CStr(Format(swPointSpaceFrame(0), "0.00")) + " y=" +
CStr(Format(swPointSpaceFrame(1), "0.00")) + " z=" +
CStr(Format(swPointSpaceFrame(2), "0.00")))

Case "GetSketchPoint"
swB2S(swObj)
ResultListBox.Items.Add(FunctionString + " (s): x=" +
CStr(Format(swPointSpaceFrame(0), "0.00")) + " y=" +
CStr(Format(swPointSpaceFrame(1), "0.00")) + " z=" +
CStr(Format(swPointSpaceFrame(2), "0.00")))

End Select
Next i
End If
End Sub
```

A.1.12 Export

```
Private Sub ExportButton_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles ExportButton.Click
Dim swComp As SldWorks.Component2
Dim TotalSelectedCount As Integer = 0
Dim i As Single = 0

TotalSelectedCount = WhatToDoList.SelectedItems.Count()

If TotalSelectedCount > 0 Then
Dim FunctionString As String = ""
For i = 0 To TotalSelectedCount - 1
FunctionString = WhatToDoList.SelectedItems(i).ToString()

Select Case FunctionString

Case "GetStartVertex"
swB2S(swGetStartVertex(swObj), ,
SwConst.swSelectType_e.swSelVERTICES)

Case "GetEndVertex"
swB2S(swGetEndVertex(swObj), ,
SwConst.swSelectType_e.swSelVERTICES)

Case "GetStartPoint2"
swB2S(swGetStartPoint2(swObj), ,
SwConst.swSelectType_e.swSelEXTSKETCHPOINTS)

Case "GetEndPoint2"
swB2S(swGetEndPoint2(swObj), ,
SwConst.swSelectType_e.swSelEXTSKETCHPOINTS)

Case "GetPoint"
swB2S(swGetPoint(swObj))

End Select
Next i
End If
Me.Close()
End Sub
```

A.1.13 Retrieving Points

```
'*********************************************************
'* These functions only filter the input from "whatever"  *
'* to an object than will be treatable by swB2S().        *
'* ------------------------------------------------------ *
'* swB2S() takes a POINT/VERTEX GENERAL OBJECT and        *
'* calculates its coords in several frames:               *
'*           SketchFrame / BodyFrame / SpaceFrame         *
'*********************************************************

Function TargetClick() As Object
'input: NOTHING!!!!!!!!!!!!!!!!!!
'output: object (with 3 coordinates -> x=object(0), y=object(1),
z=object(2))
TargetClick = swSelMgr.GetSelectionPoint(1)
End Function

Private Function swGetPoint(ByVal swObj As Object) As
SldWorks.Vertex
'input: object -> vertex
'output: object (with 3 coordinates -> x=object(0), y=object(1),
z=object(2))
Dim swVertex As SldWorks.Vertex
swVertex = swObj
swGetPoint = swVertex.GetPoint
End Function

Function swGetStartVertex(ByVal swObj As Object) As SldWorks.Vertex
'input: edge
'output: vertex
Dim swEdge As SldWorks.Edge
swEdge = swObj
swGetStartVertex = swEdge.GetStartVertex
End Function

Function swGetEndVertex(ByVal swObj As Object) As SldWorks.Vertex
'input: edge
'output: vertex
Dim swEdge As SldWorks.Edge
swEdge = swObj
swGetEndVertex = swEdge.GetEndVertex
End Function

Function swGetStartPoint2(ByVal swObj As Object) As
SldWorks.SketchPoint
'input: Sketch Line
'output: Sketch Point
Dim swSketchSeg As SldWorks.SketchSegment
swSketchSeg = swObj
Select Case swSketchSeg.GetType
Case SwConst.swSketchSegments_e.swSketchLINE
Dim swSketchLine As SldWorks.SketchLine
swSketchLine = swObj
swGetStartPoint2 = swSketchLine.GetStartPoint2
Case SwConst.swSketchSegments_e.swSketchSPLINE
Dim swSketchSpline As SldWorks.SketchSpline
Dim swSplinePoints As Object
swSketchSpline = swObj
swSplinePoints = swSketchSpline.GetPoints2
'first point is swSplinePoints(0)
'last point is swSplinePoints(UBound(vSplinePt))
swGetStartPoint2 = swSplinePoints(0)
End Select
End Function

Function swGetEndPoint2(ByVal swObj As Object) As
SldWorks.SketchPoint
```

```
'input: Sketch Line
'output: Sketch Point
Dim swSketchSeg As SldWorks.SketchSegment
swSketchSeg = swObj
Select Case swSketchSeg.GetType
Case SwConst.swSketchSegments_e.swSketchLINE
Dim swSketchLine As SldWorks.SketchLine
swSketchLine = swObj
swGetEndPoint2 = swSketchLine.GetEndPoint2
Case SwConst.swSketchSegments_e.swSketchSPLINE
Dim swSketchSpline As SldWorks.SketchSpline
Dim swSplinePoints As Object
swSketchSpline = swObj
swSplinePoints = swSketchSpline.GetPoints2
'first point is swSplinePoints(0)
'last point is swSplinePoints(UBound(vSplinePt))
swGetEndPoint2 = swSplinePoints(UBound(swSplinePoints))
End Select

End Function
```

A.1.14 Coordinate Frame Transformation

```
Public Function swB2S(ByVal swObj As Object, Optional ByVal
swAppImport As SldWorks.SldWorks = Nothing, Optional ByVal
swSelectTypeImport As Integer = -1) As Object ' Body to Space
coordinate transform

Dim swModel As SldWorks.ModelDoc2
Dim swSelMgr As SldWorks.SelectionMgr
Dim swComp As SldWorks.Component2
Dim swMathUtil As SldWorks.MathUtility 'repeated to make external
call easier

If swAppImport Is Nothing Then
'do nothing because we are not externally calling the function
Else
swApp = swAppImport
End If

swModel = swApp.ActiveDoc
swMathUtil = swApp.GetMathUtility
swSelMgr = swModel.SelectionManager
swComp = swSelMgr.GetSelectedObjectsComponent2(1)
System.Diagnostics.Debug.Write(Chr(10) + swComp.Name2)

If swSelectTypeImport = -1 Then
'we are not externally calling the function
swSelectType = swSelMgr.GetSelectedObjectType2(1) 'this might seem
repeated, but it's necessary if calling function from MainForm
Else
swSelectType = swSelectTypeImport
End If

Select Case swSelectType

Case SwConst.swSelectType_e.swSelVERTICES ' 3

Dim swVertex As SldWorks.Vertex
Dim swMathPt As SldWorks.MathPoint
Dim swCompXform As SldWorks.MathTransform
Dim swCompXformInv As SldWorks.MathTransform

swVertex = swObj
swPointBodyFrame = swVertex.GetPoint 'in body frame
!!!!!!!!!!!!!!!!!!!!!!!!!!

'Retrieve the transformation of the selected swComp
swCompXform = swComp.Transform2

'get the inverse transformation and capture the array of values
swCompXformInv = swCompXform.Inverse
swMathPt = swMathUtil.CreatePoint(swPointBodyFrame)
'VBA: must be DOUBLE or LONG, not VARIANT!!!!!!!!!!
swMathPt = swMathPt.MultiplyTransform(swCompXform)

swPointSpaceFrame = swMathPt.ArrayData
'in space frame !!!!!!!!!!!!!!!!!!!!!!!!!!

Case SwConst.swSelectType_e.swSelEXTSKETCHPOINTS ' 25

Dim swSketchPoint As SldWorks.SketchPoint
Dim swSketch As SldWorks.sketch
Dim swMathPt As SldWorks.MathPoint
Dim swCompXform As SldWorks.MathTransform
Dim swCompXformInv As SldWorks.MathTransform
Dim swModel2Sketch As SldWorks.MathTransform
Dim swModel2SketchInv As SldWorks.MathTransform
Dim swTotalTrans As SldWorks.MathTransform
```

```
Dim swXYZPoint(2) As Double
Dim swPointSketchFrame(2) As Double
'Dim vXform As Object 'only used to keep track of Xform matrix

swSketchPoint = swObj
swSketch = swSketchPoint.GetSketch

'Get coordinates in SKETCH FRAME!!!!!
swXYZPoint(0) = swSketchPoint.x
swXYZPoint(1) = swSketchPoint.y
swXYZPoint(2) = swSketchPoint.z
System.Diagnostics.Debug.Write(swXYZPoint(0))

'****************************
'*    swPointSketchFrame    *
'* (X,Y,Z) IN SKETCH FRAME  *
'****************************

swPointSketchFrame = swXYZPoint
'in sketch frame

'Retrieve the transformation of the selected swComp(body)

swCompXform = swComp.Transform2
'from space  frame to swComp(body) frame

swCompXformInv = swCompXform.Inverse
'from swComp(body) frame to space  frame

swModel2Sketch = swSketch.ModelToSketchTransform
swModel2SketchInv = swModel2Sketch.Inverse

swMathPt = swMathUtil.CreatePoint(swPointSketchFrame)
swMathPt = swMathPt.MultiplyTransform(swModel2SketchInv)
'sketch -> swComp(body)

'****************************
'*     swPointBodyFrame     *
'*  (X,Y,Z) IN BODY FRAME   *
'****************************

swPointBodyFrame = swMathPt.ArrayData
'in body frame

swTotalTrans = swModel2SketchInv.Multiply(swCompXform)
swMathPt = swMathUtil.CreatePoint(swPointSketchFrame)
swMathPt = swMathPt.MultiplyTransform(swTotalTrans)

'****************************
'*     swPointSpaceFrame    *
'* (X,Y,Z) IN SPACE FRAME   *
'****************************

swPointSpaceFrame = swMathPt.ArrayData
'in space frame !!!!!!!!!!!!!!!!!!!!!!!!!!!!

Case Else

    'Include more object types here

End Select

End Function
```

A.2 RSSR_server.dll

```
// NAT_RSSR.cpp : Implementation of CNAT_RSSR

#include "stdafx.h"
#include "NAT_RSSR.h"
#include ".\nat_rssr.h"


// CNAT_RSSR
CNAT_RSSR::CNAT_RSSR()
{
for(int i=0;i<3;i++)            // initialize coordinates
{
theta[i] = 0;
phi[i] = 0;
pt64[i] = 0;                // coordinates for lower input R Joint
kingPinPt[i] = 0;           // coordinates for kingpin point
ubj1[i] = 0;                // coordinates for upper output R Joint
lbj1[i] = 0;                // coordinates for lower output R Joint
outerball[i] = 0;           // coordinates for outer ball joint
}
for(int i=0;i<3;i++)
{
inputAxisPt1[i] = 0;
inputAxisPt2[i] = 0;
outputAxisPt1[i] = 0;
outputAxisPt2[i] = 0;
innerball[i] = 0;
refPt[i] = 0;
}
}

STDMETHODIMP CNAT_RSSR::HCalculate(DOUBLE* x1,DOUBLE* y1,DOUBLE*
z1,
                                   DOUBLE* x3,DOUBLE* y3,DOUBLE*
z3)
{

double s1[3], s2[3][3];
//S1 is not used in current program, s2 is outerball coordinates
double r1, r2;
// link lengths
double a[3], b[3], c[3];
// coefficients of constraint equation variables
double o2[3];
// coordinate system origin on axis 2
double x2[3],y2[3],z2[3];
// coordinates of outerball;
double temp;
// temporary scalar holder;
double tempArray[3];
// temporary vector holder;
double dotProduct;
// dot product result
double crossProduct[3];
// cross product result
double y;
// y coordinate of innerball
int i,j;
// loop index;

// compute z2: unit vector along axis 2
for(i=0;i<3;i++) tempArray[i] = ubj1[i] - lbj1[i];
dot(tempArray,tempArray,dotProduct);
for(i=0;i<3;i++) z2[i] = tempArray[i] / sqrt(dotProduct);
// z2
```

```
// compute x2: unit vector along r2
for(i=0;i<3;i++) tempArray[i] = outerball[i]-kingPinPt[i];
dot(tempArray,z2,temp);
for(i=0;i<3;i++) tempArray[i] = tempArray[i] - temp*z2[i];
dot(tempArray,tempArray,dotProduct);
for(i=0;i<3;i++) x2[i] = (tempArray[i])/sqrt(dotProduct);
// x2

//for(i=0;i<3;i++) cout << x2[i] << ' ';
//cout << '\n';

// compute y2: unit vector normal to z2 and x2
cross(z2,x2,crossProduct);
dot(crossProduct,crossProduct,dotProduct);
for(i=0;i<3;i++) y2[i] = crossProduct[i]/sqrt(dotProduct);
// y2

// compute o2: coordinate system origin on axis 2
for(i=0;i<3;i++) tempArray[i]=outerball[i]-lbj1[i];
dot(tempArray,z2,dotProduct);
for(i=0;i<3;i++) o2[i] = lbj1[i] + z2[i]*(dotProduct);
//o2

//for(i=0;i<3;i++) cout << o2[i] << ' ';
//cout << '\n';

// solve for r2
for(i=0;i<3;i++) tempArray[i] = outerball[i]-o2[i];
dot(tempArray,tempArray,dotProduct);
r2 = sqrt(dotProduct);

// solve for s2
for(i=0;i<3;i++)
for(j=0;j<3;j++)
s2[i][j] = o2[i] + r2*cos(phi[j])*x2[i] + r2*sin(phi[j])*y2[i];
for(i=0;i<3;i++)
{
x3[i] = s2[0][i];
y3[i] = s2[1][i];
z3[i] = s2[2][i];
}

// solve constraint equation for r1 and y
for(i=0;i<3;i++)
{
a[i] = 2*(pt64[0] - s2[0][i])* cos(theta[i]) + 2*(pt64[2] -
s2[2][i])*sin(theta[i]);
b[i] = -2*s2[1][i];
c[i] = pt64[0]*pt64[0] + pt64[2]*pt64[2] - 2*pt64[0]*s2[0][i] +
s2[0][i]*s2[0][i] + s2[1][i]*s2[1][i] + s2[2][i]*s2[2][i] -
2*pt64[2]*s2[2][i];
}

r1 = ((b[1] - b[0])*(c[2] - c[0]) - (b[2] - b[0])*(c[1] -
c[0]))/((b[2] - b[0])*(a[1] - a[0]) - (b[1] - b[0])*(a[2] - a[0]));

y = (-1*(c[1] - c[0]) - (a[1] - a[0])*r1)/(b[1] - b[0]);

// solve for coordinates of innerball
for(i=0;i<3;i++) x1[i] = (r1*cos(theta[i])) + (pt64[0]);
// x1
for(i=0;i<3;i++) z1[i] = r1*sin(theta[i]) + pt64[2];
// z1
for(i=0;i<3;i++) y1[i] = y;
// y1
return S_OK;
}
```

```
void CNAT_RSSR::cross(double u[],double v[],double crossProduct[])
{
crossProduct[0] = (u[1]*v[2]-u[2]*v[1]);
crossProduct[1] = (u[2]*v[0]-u[0]*v[2]);
crossProduct[2] = (u[0]*v[1]-u[1]*v[0]);
}

void CNAT_RSSR::dot(double u[],double v[],double &dotProduct)
{
dotProduct = (u[0]*v[0])+(u[1]*v[1])+(u[2]*v[2]);
}
void CNAT_RSSR::vcross(double u[],double v[],double crossProduct[])
{
crossProduct[0] = (u[1]*v[2]-u[2]*v[1]);
crossProduct[1] = (u[2]*v[0]-u[0]*v[2]);
crossProduct[2] = (u[0]*v[1]-u[1]*v[0]);
}
void CNAT_RSSR::vsub(double u[],double v[],double umv[])
{
for(int i=0;i<3;i++) umv[i] = u[i] - v[i];
}
void CNAT_RSSR::vadd(double u[],double v[],double uav[])
{
for(int i=0;i<3;i++) uav[i] = u[i] + v[i];
}
void CNAT_RSSR::vscale(double v[],double s, double stimev[])
{
for(int i=0;i<3;i++) stimev[i] = s*v[i];
}
void CNAT_RSSR::vdot(double u[],double v[],double &dotProduct)
{
dotProduct = (u[0]*v[0])+(u[1]*v[1])+(u[2]*v[2]);
}
void CNAT_RSSR::vnormalize(double v[])
{
double dotProduct=0;
vdot(v,v,dotProduct);
dotProduct = sqrt(dotProduct);
v[0] = v[0]/dotProduct;
v[1] = v[1]/dotProduct;
v[2] = v[2]/dotProduct;
}
void CNAT_RSSR::dirVec(double end[], double start[], double dir[])
{
vsub(end,start,dir);
vnormalize(dir);
}

STDMETHODIMP CNAT_RSSR::get_datheta(DOUBLE** pVal)
{
// TODO: Add your implementation code here

return S_OK;
}

STDMETHODIMP CNAT_RSSR::put_datheta(DOUBLE* newVal)
{
// TODO: Add your implementation code here
theta[0] = newVal[0];
theta[1] = newVal[1];
theta[2] = newVal[2];

return S_OK;
}

STDMETHODIMP CNAT_RSSR::put_theta(DOUBLE* newVal)
{
```

```
// TODO: Add your implementation code here
theta[0] = newVal[0];
theta[1] = newVal[1];
theta[2] = newVal[2];

return S_OK;
}

STDMETHODIMP CNAT_RSSR::put_phi(DOUBLE* newVal)
{
// TODO: Add your implementation code here
phi[0] = newVal[0];
phi[1] = newVal[1];
phi[2] = newVal[2];
return S_OK;
}

STDMETHODIMP CNAT_RSSR::put_data(DOUBLE* newData1,DOUBLE*
newData2,DOUBLE* newData3,DOUBLE* newData4,DOUBLE* newData5)
{
// TODO: Add your implementation code here
for(int i=0;i<3;i++)
{
pt64[i] = newData1[i];
// coordinates for lower input R Joint
kingPinPt[i] = newData2[i];
// coordinates for kingpin point
ubj1[i] = newData3[i];
// coordinates for upper output R Joint
lbj1[i] = newData4[i];
// coordinates for lower output R Joint
outerball[i] = newData5[i];
}
return S_OK;
}

STDMETHODIMP CNAT_RSSR::put_theta2(DOUBLE* newTheta)
{
// TODO: Add your implementation code here
theta[0] = newTheta[0];
theta[1] = newTheta[1];
return S_OK;
}

STDMETHODIMP CNAT_RSSR::put_phi2(DOUBLE* newPhi)
{
// TODO: Add your implementation code here
phi[0] = newPhi[0];
phi[1] = newPhi[1];
return S_OK;
}

STDMETHODIMP CNAT_RSSR::put_data2(DOUBLE* newInputAxisPt1, DOUBLE*
newInputAxisPt2, DOUBLE* newOutputAxisPt1, DOUBLE*
newOutputAxisPt2, DOUBLE* newInnerball, DOUBLE* newRefPt)
{
// TODO: Add your implementation code here
for(int i=0;i<3;i++)
{
inputAxisPt1[i] = newInputAxisPt1[i];
inputAxisPt2[i] = newInputAxisPt2[i];
outputAxisPt1[i] = newOutputAxisPt1[i];
outputAxisPt2[i] = newOutputAxisPt2[i];
innerball[i] = newInnerball[i];
refPt[i] = newRefPt[i];
}
return S_OK;
}
```

```
STDMETHODIMP CNAT_RSSR::HCalculate2(DOUBLE* s1x, DOUBLE* s1y,
DOUBLE* s1z, DOUBLE* s2x, DOUBLE* s2y, DOUBLE* s2z)
{
// TODO: Add your implementation code here
// This is where the new functions go.
double s1[3][3],s2[3][3];
//coordinates of two spherical joints
double R1,R2;
//radius of driving and driven links
double o1[3], o2[3];
//origines of the two coordinate sytems
double x1[3],y1[3],z1[3];
// coordinates of innerball;
double x2[3],y2[3],z2[3];
// unit vectors of axis 2 coordinate sys;
double temp;
// temporary scalar holder;
double tempArray[3];
// temporary vector holder;
double tempArray2[3];
// temporary vector holder;
double dotProduct;
// vdot product result
double crossProduct[3];
// vcross product result
double x, y;
double C1, C2, C3, D1, D2, D3;
int i,j;
// loop index;


// compute the coordinate system of driving link (axis2)
dirVec(inputAxisPt2,inputAxisPt1,z1);
// z1

//x1 = vnormalize[(innerball - inputAxisPt1) - ((innerball -
inputAxisPt1).z1)*z1]
vsub(innerball,inputAxisPt1,tempArray); //innerball-inputAxisPt1
vdot(tempArray,z1,dotProduct);
vscale(z1,dotProduct,tempArray2); //tempArray2=z1*((innerball -
inputAxisPt1).z1)
vsub(tempArray,tempArray2,x1);
vnormalize(x1);
// x1

vcross(z1,x1,y1);
// y1

//o1 = inputAxisPt1 + z1*((innerball - inputAxisPt1).z1)
vadd(inputAxisPt1,tempArray2,o1);
// o1

//R1 = Sqrt[(innerball - o1).(innerball - o1)]
vsub(innerball,o1,tempArray);
vdot(tempArray,tempArray,dotProduct);
R1 = sqrt(dotProduct);

// compute the coordinate system for driven link axis (axis2)
dirVec(outputAxisPt2,outputAxisPt1,z2);
// z2

//x2 = vnormalize[((refPt - outputAxisPt1) - ((refPt -
outputAxisPt1).z2)*z2]
vsub(refPt,outputAxisPt1,tempArray);
//tempArray = refPt-outputAxisPt1
vdot(tempArray,z2,dotProduct);
vscale(z2,dotProduct,tempArray2);
```

```
//tempArray2=((refPt - outputAxisPt1).z2)*z2
vsub(tempArray,tempArray2,x2);
vnormalize(x2);
// x2

vcross(z2,x2,y2);
// y2
// o2 = outputAxisPt1 + ((refPt - outputAxisPt1).z2)*z2
vadd(outputAxisPt1,tempArray2,o2);
// o2

s1[0][0] = innerball[0];
s1[0][1] = innerball[1];
s1[0][2] = innerball[2];

for(i=0;i<2;i++)
for(j=0;j<3;j++)
s1[i+1][j] = o1[j] + R1*cos(theta[i])*x1[j] +
R1*sin(theta[i])*y1[j];


C1 = -2*o2[0]*x2[0] - 2*o2[1]*x2[1] - 2*o2[2]*x2[2] +
2*x2[0]*innerball[0] + 2*x2[1]*innerball[1] + 2*x2[2]*innerball[2]
+ cos(phi[0])*(2*o2[0]*x2[0] + 2*o2[1]*x2[1] + 2*o2[2]*x2[2] -
2*x2[0]*s1[1][0] - 2*x2[1]*s1[1][1] - 2*x2[2]*s1[1][2]) +
sin(phi[0])*(2*o2[0]*y2[0] + 2*o2[1]*y2[1] + 2*o2[2]*y2[2] -
2*y2[0]*s1[1][0] - 2*y2[1]*s1[1][1] - 2*y2[2]*s1[1][2]);

C2 = -2*o2[0]*y2[0] - 2*o2[1]*y2[1] - 2*o2[2]*y2[2] +
2*y2[0]*innerball[0] + 2*y2[1]*innerball[1] + 2*y2[2]*innerball[2]
+ sin(phi[0])*(-2*o2[0]*x2[0] - 2*o2[1]*x2[1] - 2*o2[2]*x2[2] +
2*x2[0]*s1[1][0] + 2*x2[1]*s1[1][1] + 2*x2[2]*s1[1][2]) +
cos(phi[0])*(2*o2[0]*y2[0] + 2*o2[1]*y2[1] + 2*o2[2]*y2[2] -
2*y2[0]*s1[1][0] - 2*y2[1]*s1[1][1] - 2*y2[2]*s1[1][2]);

C3 = 2*o2[0]*innerball[0] - innerball[0]*innerball[0] +
2*o2[1]*innerball[1] - innerball[1]*innerball[1] +
2*o2[2]*innerball[2] - innerball[2]*innerball[2] - 2*o2[0]*s1[1][0]
+ s1[1][0]*s1[1][0] - 2*o2[1]*s1[1][1] + s1[1][1]*s1[1][1] -
2*o2[2]*s1[1][2] + s1[1][2]*s1[1][2];

D1 = -2*o2[0]*x2[0] - 2*o2[1]*x2[1] - 2*o2[2]*x2[2] +
2*x2[0]*innerball[0] + 2*x2[1]*innerball[1] + 2*x2[2]*innerball[2]
+ cos(phi[1])*(2*o2[0]*x2[0] + 2*o2[1]*x2[1] + 2*o2[2]*x2[2] -
2*x2[0]*s1[2][0] - 2*x2[1]*s1[2][1] - 2*x2[2]*s1[2][2]) +
sin(phi[1])*(2*o2[0]*y2[0] + 2*o2[1]*y2[1] + 2*o2[2]*y2[2] -
2*y2[0]*s1[2][0] - 2*y2[1]*s1[2][1] - 2*y2[2]*s1[2][2]);

D2 = -2*o2[0]*y2[0] - 2*o2[1]*y2[1] - 2*o2[2]*y2[2] +
2*y2[0]*innerball[0] +  2*y2[1]*innerball[1] + 2*y2[2]*innerball[2]
+ sin(phi[1])*(-2*o2[0]*x2[0] - 2*o2[1]*x2[1] - 2*o2[2]*x2[2] +
2*x2[0]*s1[2][0] + 2*x2[1]*s1[2][1] + 2*x2[2]*s1[2][2]) +
cos(phi[1])*(2*o2[0]*y2[0] + 2*o2[1]*y2[1] + 2*o2[2]*y2[2] -
2*y2[0]*s1[2][0] - 2*y2[1]*s1[2][1] - 2*y2[2]*s1[2][2]);

D3 = 2*o2[0]*innerball[0] - innerball[0]*innerball[0] +
2*o2[1]*innerball[1] - innerball[1]*innerball[1] +
2*o2[2]*innerball[2] - innerball[2]*innerball[2] - 2*o2[0]*s1[2][0]
+ s1[2][0]*s1[2][0] - 2*o2[1]*s1[2][1] + s1[2][1]*s1[2][1] -
2*o2[2]*s1[2][2] + s1[2][2]*s1[2][2];

//C1*x + C2*y + C3 = 0,  D1*x + D2*y + D3 = 0
x = -((C3*D2 - C2*D3)/(-(C2*D1) + C1*D2));
y =  -((C3*D1 - C1*D3)/(C2*D1 - C1*D2));

//s2[0] = o2 + x*x2 + y*y2
vscale(x2,x,tempArray);
vadd(o2,tempArray,tempArray2);
```

```
vscale(y2,y,tempArray);
vadd(tempArray2,tempArray,s2[0]);

//s2[1,2] = o2 + (x*Cos[?] - y*Sin[?])*x2 + (x*Sin[?] +
y*Cos[?])*y2
for(i=1;i<3;i++)  {
vscale(x2,x*cos(phi[i-1])-y*sin(phi[i-1]), tempArray);
vadd(o2,tempArray,s2[i]);
vscale(y2,x*sin(phi[i-1])+y*cos(phi[i-1]),tempArray);
vadd(s2[i],tempArray,s2[i]);
}

for(i=0;i<3;i++)
{
s1x[i] = s1[i][0];            // x coordinates of s1
s1y[i] = s1[i][1];            // y coordinates of s1
s1z[i] = s1[i][2];            // z coordinates of s1
s2x[i] = s2[i][0];            // x coordinates of s2
s2y[i] = s2[i][1];            // y coordinates of s2
s2z[i] = s2[i][2];            // z coordinates of s2
}


return S_OK;
}

// RSSR_Server.cpp : Implementation of DLL Exports.

#include "stdafx.h"
#include "resource.h"

// The module attribute causes DllMain, DllRegisterServer and
DllUnregisterServer to be automatically implemented for you
[ module(dll, uuid = "{F4FCC7AC-314B-413F-8F30-07FC66BFEA00}",
name = "RSSR_Server",
helpstring = "RSSR_Server 1.0 Type Library",
resource_name = "IDR_RSSR_SERVER") ]
class CRSSR_ServerModule
{
public:
// Override CAtlDllModuleT members
};
```

A.3 getRSSR.dll

```
'MAKE FILE: getRSSR.DLL

'Make sure that a reference to the swpublished.tlb type
library exists

'Tell VB that you are going to provide functionality for the
SwAddin interface
Implements SWPublished.SwAddin

Dim iSldWorks                   As SldWorks.SldWorks
Dim iCookie                     As Long
Dim iToolbarID                  As Long

''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
'Implementation methods of the SwAddin interface
Private Function SwAddin_ConnectToSW(ByVal ThisSW As Object,
ByVal Cookie As Long) As Boolean
Dim bRet As Boolean

' store reference to SW session
Set iSldWorks = ThisSW

' store cookie from SW
iCookie = Cookie

'Inform SW about the object that contains the callbacks
bRet = iSldWorks.SetAddinCallbackInfo(App.hInstance, Me,
iCookie)

'Add a menu item on the frame when no documents are present
bRet = iSldWorks.AddMenu(swDocNONE, "RSSR", 3)
bRet = iSldWorks.AddMenuItem2(swDocNONE, iCookie,
"DocNONE_Item@RSSR", -1, "DocNONE_Item", "DocNONE_ItemUpdate",
"RSSR|DocNONE_Item hint string")
bRet = iSldWorks.AddMenuItem2(swDocPART, iCookie,
"DocPART_Item@RSSR", -1, "DocPART_Item", "DocPART_ItemUpdate",
"RSSR|DocPART_Item hint string")
bRet = iSldWorks.AddMenuItem2(swDocASSEMBLY, iCookie,
"RSSR_Item@RSSR", -1, "RSSR_Item", "RSSR_ItemUpdate",
"RSSR|RSSR_Item hint string")
bRet = iSldWorks.AddMenuItem2(swDocDRAWING, iCookie,
"DocDRAWING_Item@RSSR", -1, "DocDRAWING_Item",
"DocDRAWING_ItemUpdate", "RSSR|DocDRAWING_Item hint string")
iToolbarID = iSldWorks.AddToolbar3(iCookie, "RSSR Toolbar",
102, 101, -1, _
swDocTemplateTypeNONE + swDocTemplateTypePART +
swDocTemplateTypeASSEMBLY + swDocTemplateTypeDRAWING)

bRet = iSldWorks.AddToolbarCommand2(iCookie, iToolbarID, 0,
"ToolbarFunc", "ToolbarFuncUpdate", "Press Me!", "Hint in
status bar")

' DO NOT use - toolbar visibility managed by SW
'bRet = iSldWorks.ShowToolbar2(iCookie, iToolbarID)

SwAddin_ConnectToSW = True
End Function

Private Function SwAddin_DisconnectFromSW() As Boolean
Dim bRet            As Boolean

'Remove any UI that was added earlier
bRet = iSldWorks.RemoveMenu(swDocNONE, "RSSR", "")
bRet = iSldWorks.RemoveMenu(swDocPART, "RSSR", "")
bRet = iSldWorks.RemoveMenu(swDocASSEMBLY, "RSSR", "")
```

```
bRet = iSldWorks.RemoveMenu(swDocDRAWING, "RSSR", "")

bRet = iSldWorks.RemoveToolbar2(iCookie, iToolbarID)

Set iSldWorks = Nothing

SwAddin_DisconnectFromSW = True
End Function

,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
'Callback routines for SW
Public Sub DocNONE_Item()
MsgBox "RSSR works with assemblies"
End Sub

Public Function DocNONE_ItemUpdate() As Long
'Return the state information for the menu item
'  0 - Disabled and unchecked
'  1 - Enabled and unchecked (default when update routine does
not exist)
'  2 - Disabled and checked
'  3 - Enabled and checked

DocNONE_ItemUpdate = 1
End Function

Public Sub DocPART_Item()
MsgBox "RSSR works with assemblies"
End Sub

Public Function DocPART_ItemUpdate() As Long
'Return the state information for the menu item
'  0 - Disabled and unchecked
'  1 - Enabled and unchecked (default when update routine does
not exist)
'  2 - Disabled and checked
'  3 - Enabled and checked

DocPART_ItemUpdate = 1
End Function

Public Sub RSSR_Item()
Dim retval As Boolean
retval = Shell("C:\temp\RSSR.exe", 1)
End Sub

Public Function RSSR_ItemUpdate() As Long
'Return the state information for the menu item
'  0 - Disabled and unchecked
'  1 - Enabled and unchecked (default when update routine does
not exist)
'  2 - Disabled and checked
'  3 - Enabled and checked

RSSR_ItemUpdate = 1
End Function

Public Sub DocDRAWING_Item()
MsgBox "RSSR works with assemblies"
End Sub

Public Function DocDRAWING_ItemUpdate() As Long
'Return the state information for the menu item
'  0 - Disabled and unchecked
'  1 - Enabled and unchecked (default when update routine does
not exist)
'  2 - Disabled and checked
'  3 - Enabled and checked
```

```
DocDRAWING_ItemUpdate = 1
End Function

Public Sub ToolbarFunc()
Dim retval As Boolean
retval = Shell("C:\temp\RSSR.exe", 1)
End Sub

Public Function ToolbarFuncUpdate() As Long
' 0 Button is disabled (grey)
' 1 Button is enabled (default state).
' 2 Button is Disabled and "Pushed"
' 3 Button is Enabled and "Pushed

ToolbarFuncUpdate = 1
End Function
```