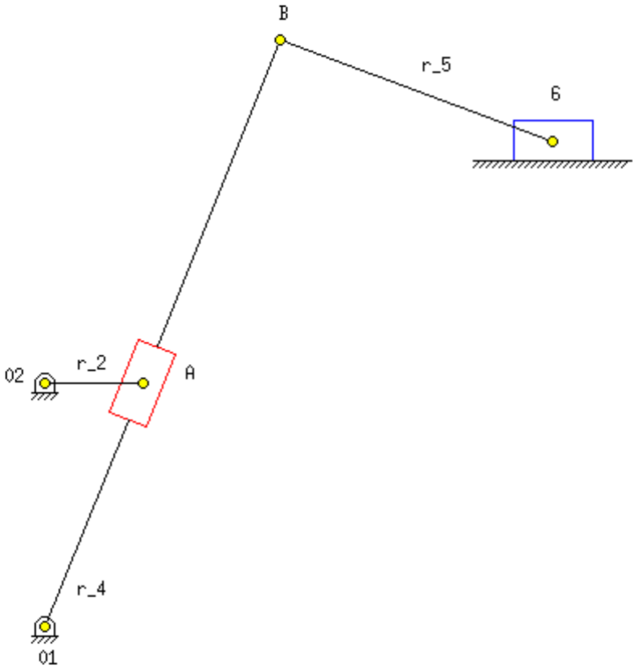
Computer-Aided Design and Analysis of the Whitworth Quick Return Mechanism



Matt Campbell
Stephen S. Nestinger
Computer-Aided Mechanism Design, Project
Department of Mechanincal and Aeronautical Engineering
University of California
Davis, CA 95616

March 2004

Table of Contents

Section **Page**

Abstract

This report discusses the design and implementation of a software package for the computer-aided design and analysis of the Whitworth Quick Return Mechanism. The kinematic analysis of the Whitworth Quick Return Mechanism is discussed with details on how the position, velocity, and accerleration were calculated. The dynamic anaylsis of the Whitworth Quick Return Mechanism follows. A description of the software package is given as well as the package API. A Whitworth Quick Return Mechanism class was created that allows users to easily calculate the position, velocity, acceleration, and forces at each linkage point given an input torque. The class also allows users to find the required input torque given an input force. The class utilizes the available xlinkage software to display an animation of the Whitworth Quick Return Mechanism.

1 Introduction

The implementation of a Whitworth Quick Return Mechanism can be useful for applications requiring slow initial force and a quick reset operation. The design of such a Whitworth Quick Return Mechanism can be tedious to do by hand. High-end commercial computer applications are available that can help in designing a Whitworth Quick Return Mechanism but these applications are usually large and come with other packages that are not essential to the specified task. These commercial applications are also expensive for the general user. For students earning a degree in mechanical engineering, these black-box commercial software packages are suitable for explaining some basic principals and concepts with traditional graphic methods. In order to fully comprehend the subject matter, students must utilize numerical and analytical methods to solve complicated engineering problems. A general purpose Whitworth Quick Return Mechanism software package is required that allows the general public to be able to quickly design and implement a Whitworth Quick Return Mechanism. Such a package would require a simple user front end and easy to understand API. Users should be allowed to fully integrate the software package into their own code with the ability to either choose to specify an input torque or required output force. Students most benefit from an open software package as compared to a black-box software package. Students are able to go through and examine the available source code and modify it to solve similar problems. By learning from examples, students will better understand the principles and numerical aspects of the subject matter.

Utilizing the C/C++ interpreter, Ch, a Whitworth Quick Return Mechanism software package has been created to facilitate in the design and analysis of a Whitworth Quick Return Mechanism [1]. The package contains the CQuickReturn class giving users the ability calculate the position, velocity, and acceleration of each linkage. The fundamental methods used to analyze a Whitworth Quick Return Mechanism can be found in [2]. Users can also plot the output motion of any linkage. Utilizing the xlinkage software available in the Ch Mechanism toolkit, the class provides a function to create an animation file that can be displayed by xlinkage showing the movements of the Whitworth Quick Return Mechanism over time [3].

2 Kinematic Analysis of the Whitworth Quick Return Mechanism

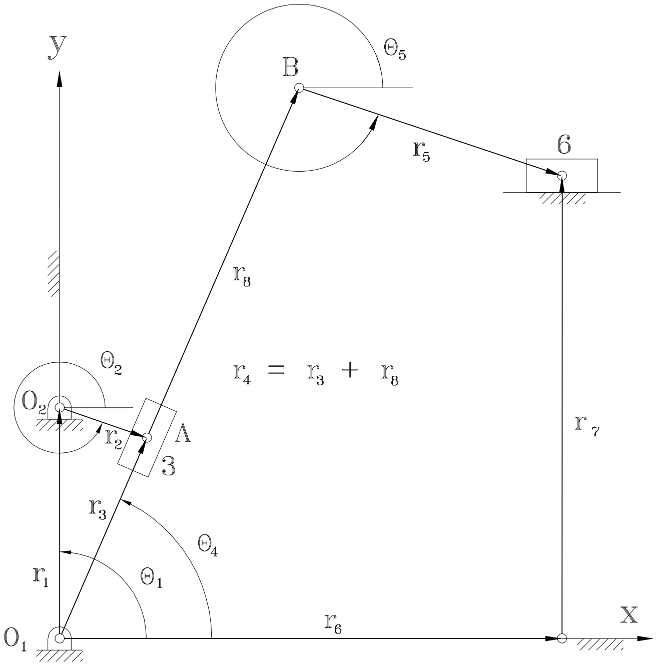


Figure 1. Vector representation of the Whitworth Quick Return Mechanism.

Looking at Figure 1 the Whitworth Quick Return Mechanism can be broken up into multiple vectors and two loops. Utilizing these two loops, the following sections will go through the kinematic analysis of the Whitworth Quick Return Mechanism.

2.1 Position Analysis

For the Whitworth Quick Return Mechanism shown in Figure 1, the displacement analysis can be formulated by the following loop-closer equations

$$\mathbf{r_1} + \mathbf{r_2} = \mathbf{r_3} \tag{1a}$$

$$\mathbf{r_3} + \mathbf{r_8} + \mathbf{r_5} = \mathbf{r_6} + \mathbf{r_7} \tag{1b}$$

Using complex numbers, Equations 1 become

$$r_1 e^{i\theta_1} + r_2 e^{i\theta_2} = r_3 e^{i\theta_3} \tag{2a}$$

$$r_3 e^{i\theta_3} + r_8 e^{i\theta_8} + r_5 e^{i\theta_5} = r_6 e^{i\theta_6} + r_7 e^{i\theta_7} \tag{2b}$$

where the link lengths r_1, r_2, r_5, r_7, and angular positions θ_1, θ_6, and θ_7 are constants. Angular position θ_2 is an independent variable; angular positions θ_3, θ_8, θ_4, and θ_5 are dependent variables. Looking at Figure 1, it can be seen that $\theta_8 = \theta_3 = \theta_4$ and $r_4 = r_3 + r_8$. Substituting and rearranging Equations 2 to have all of the unknowns on the left hand side and all of the knowns on the right hand side gives

$$r_3 e^{i\theta_4} = r_1 e^{i\theta_1} + r_2 e^{i\theta_2} \tag{3a}$$

$$r_4 e^{i\theta_4} + r_5 e^{i\theta_5} - r_6 e^{i\theta_6} = r_7 e^{i\theta_7} \tag{3b}$$

Looking at Equations 3, it can be seen that Equation 3a has 2 unknowns and Equation 3b has 3 unknowns. Utilizing Euler's equation, $e^{i\theta} = \cos\theta + i\sin\theta$, Equation 3a can be broken up into two equations, one comprising of the real numbers and the other comprising of the imaginary numbers.

$$r_3 \cos\theta_4 = r_1 \cos\theta_1 + r_2 \cos\theta_2 \tag{4a}$$

$$r_3 \sin\theta_4 = r_1 \sin\theta_1 + r_2 \sin\theta_2 \tag{4b}$$

Squaring Equations 4, adding them together, and simplifying gives

$$r_3 = \sqrt{(r_1 \cos\theta_1 + r_2 \cos\theta_2)^2 + (r_1 \sin\theta_1 + r_2 \sin\theta_2)^2} \tag{5}$$

Dividing Equation 4b by Equation 4a and simplifying gives

$$\theta_4 = \arctan\left(\frac{r_1 \sin\theta_1 + r_2 \sin\theta_2}{r_1 \cos\theta_1 + r_2 \cos\theta_2}\right) \tag{6}$$

Knowing θ_4, Equation 3b now only has 2 unknowns and becomes

$$r_6 e^{i\theta_6} - r_5 e^{i\theta_5} = r_4 e^{i\theta_4} - r_7^{i\theta_7} \tag{7}$$

Since the right hand side of Equation 7 is constant, we let $re^{i\theta} = r_4 e^{i\theta_4} - r_7 e^{i\theta_7}$ and use it in the rest of the calculations. Breaking Equation 7 up into real and imaginary parts gives

$$r_6 \cos\theta_6 - r_5 \cos\theta_5 = r\cos\theta \tag{8a}$$

$$r_6 \sin\theta_6 - r_5 \sin\theta_5 = r\sin\theta \tag{8b}$$

Solving Equations 8 for r_6 gives

$$r_6 = \frac{r\cos\theta + r_5 \cos\theta_5}{\cos\theta_6} \tag{9a}$$

$$r_6 = \frac{r\sin\theta + r_5 \sin\theta_5}{\sin\theta_6} \tag{9b}$$

where Equation 9a is used when $\cos\theta_6 > 0$ and Equation 9b is used when $\cos\theta_6 = 0$. Substituting Equation 9a into Equation 8b gives

$$\sin(\theta_5 - \theta_6) = \frac{r\cos\theta\sin\theta_6 - r\sin\theta\cos\theta_6}{r_5} \tag{10}$$

Solving for θ_5 we find

$$\theta_{5a} = \theta_6 + \arcsin\left(\frac{r\cos\theta\sin\theta_6 - r\sin\theta\cos\theta_6}{r_5}\right) \tag{11a}$$

$$\theta_{5b} = \theta_6 + \pi - \arcsin\left(\frac{r\sin\theta\cos\theta_6 - r\cos\theta\sin\theta_6}{r_5}\right) \tag{11b}$$

Knowing all of the angular positions and the length of r_6, we can find the position of the output slider, link 6, using

$$\mathbf{P_6} = \mathbf{r_4} + \mathbf{r_5} \tag{12}$$

2.2 Velocity Analysis

For the Whitworth Quick Return Mechanism shown in Figure 1, the velocity analysis can be formulated by taking the time derivative of the loop-closer equations. Taking the time derivative of Equations 2 gives

$$\dot{r}_3 e^{i\theta_4} + r_3 i\omega_4 e^{i\theta_4} = \dot{r}_1 e^{i\theta_1} + r_1 i\omega_1 e^{i\theta_1} + \dot{r}_2 e^{i\theta_2} + r_2 i\omega_2 e^{i\theta_2} \tag{13a}$$

$$\dot{r}_6 e^{i\theta 6} + r_6 i\omega_6 e^{i\theta_6} = \dot{r}_4 e^{i\theta_4} + r_4 i\omega_4 e^{i\theta_4} + \dot{r}_5 e^{i\theta_5} + r_5 i\omega_5 e^{i\theta_5} - \dot{r}_7 e^{i\theta_7} - r_7 i\omega_7 e^{i\theta_7} \tag{13b}$$

Equations 13 can be simplified since $\dot{r}_1 = \dot{r}_2 = \dot{r}_4 = \dot{r}_5 = 0$, the links are assumed to be rigid members that may not elongate, $\omega_1 = 0$, link 1 is a rigid link that is unable to rotate, $\omega_6 = \omega_7 = 0$ and $\theta_6 = 0$ as links 6 and 7 are assumed to be non-rotating imaginary members, and $\dot{r}_7 = 0$ because the output slider 6 is assumed to remain on the ground at all times. Applying these simplifications, we have

$$\dot{r}_3 e^{i\theta_4} + r_3 i\omega_4 e^{i\theta_4} = r_2 i\omega_2 e^{i\theta_2} \tag{14a}$$

$$\dot{r}_6 = r_4 i\omega_4 e^{i\theta_4} + r_5 i\omega_5 e^{i\theta_5} \tag{14b}$$

Equation 14a has 2 unknowns while Equation 14b has 3 unknowns. Breaking up Equation 14a into imaginary and real parts, solving each for $\dot{r}_3$, setting them equal to each other and solving it for ω_4 gives

$$\omega_4 = \frac{r_2\omega_2 \cos\theta_2 \cos\theta_4 + r_2\omega_2 \sin\theta_2 \sin\theta_4}{r_3} \tag{15}$$

$\dot{r}_3$ can be found by plugging the found ω_4 into either the imaginary or real equation of Equation 14a. With the known ω_4, Equation 14b now only has 2 unknowns, $\dot{r}_6$ and ω_5. Breaking it up into its real and imaginary parts we have

$$\dot{r}_6 = -r_4\omega_4 \sin\theta_4 - r_5\omega_5 \sin\theta_5 \tag{16a}$$

$$0 = r_4\omega_4 \sin\theta_4 + r_5\omega_5 \sin\theta_5 \tag{16b}$$

Solving Equation 16b for ω_5, we find

$$\omega_5 = \frac{-r_4\omega_4 \cos\theta_4}{r_5 \cos\theta_5} \tag{17}$$

$\dot{r}_6$ can now be found by plugging the found ω_5 and ω_4 into Equation 16a. We can find the velocity of the output slider, link 6, using

$$\mathbf{V_6} = \dot{\mathbf{r}}_\mathbf{6} + \dot{\mathbf{r}}_\mathbf{7} \tag{18}$$

Breaking Equation 18 into its $\mathbf{X}$ and $\mathbf{Y}$ components, we find

$$V_{6x} = \dot{r}_6 \tag{19a}$$

$$V_{6y} = 0 \tag{19b}$$

2.3 Acceleration Analysis

For the Whitworth Quick Return Mechanism shown in Figure 1, the acceleration analysis can be formulated by taking the second time derivative of the loop-closer equations or by taking the first time derivative of Equations 14 giving

$$\ddot{r}_3 e^{i\theta_4} + 2\dot{r}_3 i\omega_4 e^{i\theta_4} - r_3\omega_4^2 e^{i\theta_4} + r_3 i\alpha_4 e^{i\theta_4} = \dot{r}_2 i\omega_2 e^{i\theta_2} + r_2 i\alpha_2 e^{i\theta_2} - r_2\omega_2^2 e^{i\theta_2} \tag{20a}$$

$$\ddot{r}_6 = \dot{r}_4 i\omega_4 e^{i\theta_4} + r_4 i\alpha_4 e^{i\theta_4} - r_4\omega_4^2 e^{i\theta_4} + \dot{r}_5 i\omega_5 e^{i\theta_5} + r_5 i\alpha_5 e^{i\theta_5} - r_5\omega_5^2 e^{i\theta_5} \tag{20b}$$

Equations 20 can be simplified since $\dot{r_2} = \dot{r_4} = \dot{r_5} = 0$ as links 2, 4 and 5 are assumed to be a rigid links that are unable to elongate. Breaking Equation 20a into real and imaginary parts gives

$$\ddot{r_3}\cos\theta_4 - 2\dot{r_3}\omega_4\sin\theta_4 - r_3\omega_4^2\cos\theta_4 - r_3\alpha_4\sin\theta_4 = -r_2\alpha_2\sin\theta_2 - r_2\omega_2^2\cos\theta_2 \tag{21a}$$

$$\ddot{r_3}\sin\theta_4 + 2\dot{r_3}\omega_4\cos\theta_4 - r_3\omega_4^2\sin\theta_4 + r_3\alpha_4\cos\theta_4 = r_2\alpha_2\cos\theta_2 - r_2\omega_2^2\sin\theta_2 \tag{21b}$$

Solving Equations 21 for r_3, setting them equal to each other and solving for α_4 gives

$$\alpha_4 = \frac{r_2}{r_3}\left\{-\omega_2^2\cos\left(\theta_2 - \theta_4\right) + \alpha_2\sin\left(\theta_2 - \theta_4\right)\right\} - 2\frac{\dot{r_3}}{r_3}\omega_4 \tag{22}$$

$\ddot{r_3}$ can now be found by plugging the found α_4 into Equation 21a. With the known α_4, Equation 20b now only has 2 unknowns, $\ddot{r_6}$ and α_5. Breaking it up into real and imaginary parts gives

$$\ddot{r_6} = -r_4\alpha_4\sin\theta_4 - r_4\omega_4^2\cos\theta_4 - r_5\alpha_5\sin\theta_5 - r_5\omega_5^2\cos\theta_5 \tag{23a}$$

$$0 = r_4\alpha_4\cos\theta_4 - r_4\omega_4^2\sin\theta_4 + r_5\alpha_5\cos\theta_5 - r_5\omega_5^2\sin\theta_5 \tag{23b}$$

Solving Equation 23b for α_5 we find

$$\alpha_5 = \frac{r_4\left(\omega_4^2\sin\theta_4 - \alpha_4\cos\theta_4\right) + r_5\omega_5^2\sin\theta_5}{r_5\cos\theta_5} \tag{24}$$

We can now find $\ddot{r_6}$ by plugging α_5 into Equation 23a. The acceleration of the output slider, link 6, can now be found with

$$\mathbf{a_6} = \ddot{\mathbf{r_6}} + \ddot{\mathbf{r_7}} \tag{25}$$

Breaking Equation 25 into its $\mathbf{X}$ and $\mathbf{Y}$ components, we find

$$a_{6x} = \ddot{r_6} \tag{26a}$$

$$a_{6y} = 0 \tag{26b}$$

3 Dynamic Analysis of the Whitworth Quick Return Mechanism

Utilizing the previous analysis of position, velocity, and acceleration along with the inertia properties, such as mass and mass moment of inertia of each moving body, we are now able to perform force analysis on the Whitworth Quick Return Mechanism. This is done by pulling apart the Whitworth Quick Return Mechanism and determining the static force equations of each member. This model will neglect friction forces. When the force equations for all members have been found, a matrix equation can be formulated and the required torque input for a wanted force output of the output slider, link 6, can be found.

3.1 Forces on Each Member

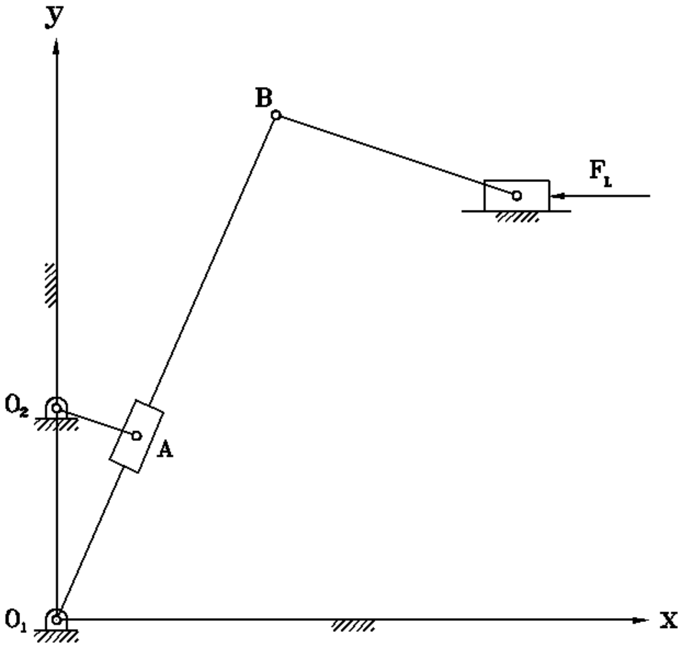


Figure 2. External Forces Acting on the Whitworth Quick Return Mechanism.

For the Whitworth Quick Return Mechanism shown in Figure 2, dynamic formulations can be derived to calculate the required input torque T_s and joint reaction forces. A free body diagram is given for each link.

Three static equilibrium equations can be written in therms of forces in $\mathbf{X}$ and $\mathbf{Y}$ directions and moment about the center of gravity for links 2, 4, and 5. The static equilibrium equations for links 3 and 6 are different since they are sliders. The equilibrium equations for each link is given.

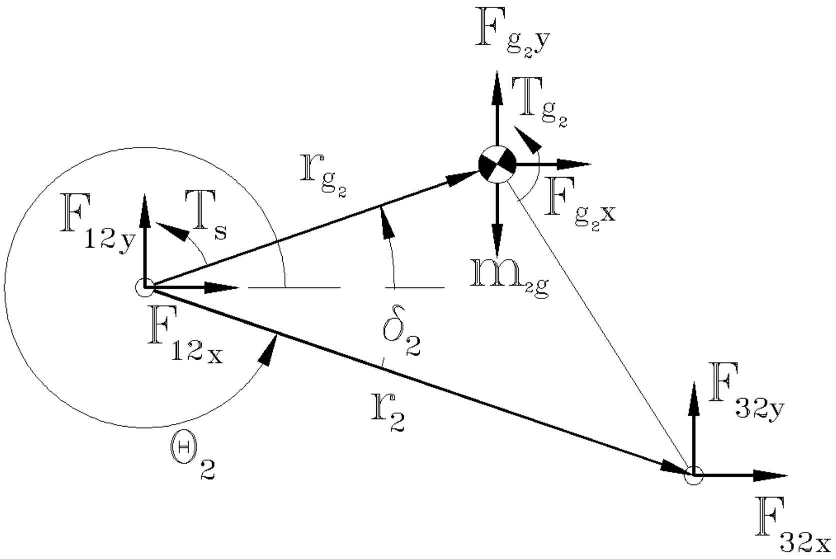


Figure 3. Free Body Diagram of Link 2.

For link 2, we get

$$F_{12x} + F_{32x} + F_{g2x} = 0 \tag{27a}$$

$$-m_2 g + F_{12y} + F_{32y} + F_{g2y} = 0 \tag{27b}$$

$$T_s + (-\mathbf{r}_{g2}) \times \mathbf{F}_{12} + (\mathbf{r}_2 - \mathbf{r}_{g2}) \times \mathbf{F}_{32} + T_{g2} = 0 \tag{27c}$$

where $\mathbf{F}_{12}$ and $\mathbf{F}_{32}$ are the joint forces acting on link 2 from the ground and link3, F_{g2} and T_{g2} are the inertia force and inertia moment of link 2, m_2 is the mass of link 2, $\mathbf{r}_{g2} = r_{g2} e^{i(\theta_2+\delta_2)}$ is the position vector of the center of gravity of link 2 from joint O_2, and T_s is the driving torque.

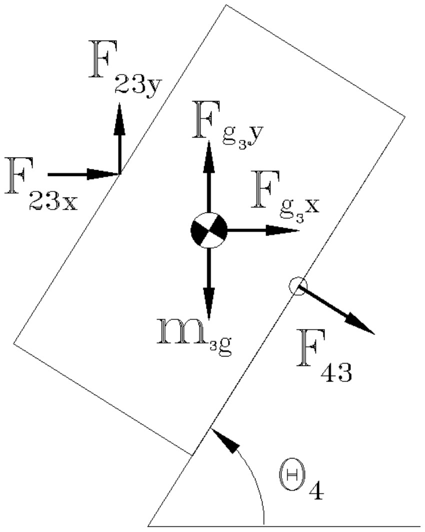


Figure 4. Free Body Diagram of Link 3.

For link 3, we get

$$F_{23x} + F_{43}\cos\phi + F_{g3x} = 0 \tag{28a}$$

$$-m_3 g + F_{23y} + F_{43}\sin\phi + F_{g3y} = 0 \tag{28b}$$

where $\phi = \theta_4 - \frac{\pi}{2}$, $\mathbf{F}_{23}$ and $\mathbf{F}_{43}$ are the joint forces acting on link 3 from links 2 and 4, and m_3 is the mass of link 3. There are no torques on link 3 since it is assumed to be a point mass.

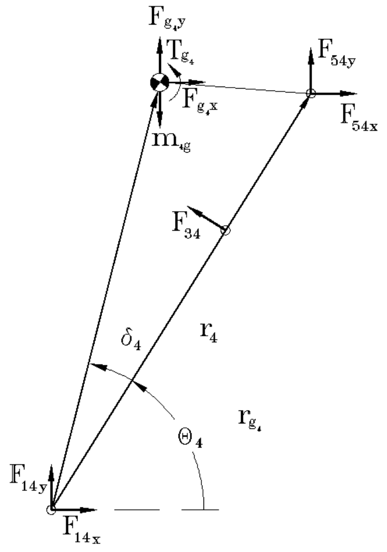


Figure 5. Free Body Diagram of Link 4.

For link 4, we get

$$F_{14x} + F_{34}\cos\phi + F_{54x} + F_{g4x} = 0 \tag{29a}$$

$$-m_4 g + F_{14y} + F_{34}\sin\phi + F_{54y} + F_{g4y} = 0 \tag{29b}$$

$$(-\mathbf{r}_{g_4}) \times \mathbf{F}_{14} + (\mathbf{r}_3 - \mathbf{r}_{g_4}) \times \mathbf{F_{34}} + (\mathbf{r}_4 - \mathbf{r}_{g_4}) \times \mathbf{F_{54}} + T_{g_2} = 0 \tag{29c}$$

where $\phi = \theta_4 - \frac{\pi}{2}$, $\mathbf{F}_{14}$, $\mathbf{F}_{34}$, and $\mathbf{F}_{54}$ are the joint forces acting on link 4 from links 1, 3, and 5, F_{g_4} and T_{g_4} are the inertia force and inertia moment of link 4, m_4 is the mass of link 4, and $\mathbf{r}_{g_4} = r_{g_4} e^{i(\theta_4 + \delta_4)}$ is the position vector of the center of gravity of link 4 from joint O_1. Since link 3 is a slider and the dynamic analysis is neglecting friction, the force $\mathbf{F}_{34}$ will always be normal to link 4. Therefore it is not necessary to break this force into **x** and **y** components and instead use the angle ϕ for this purpose when forming the dynamics equations.

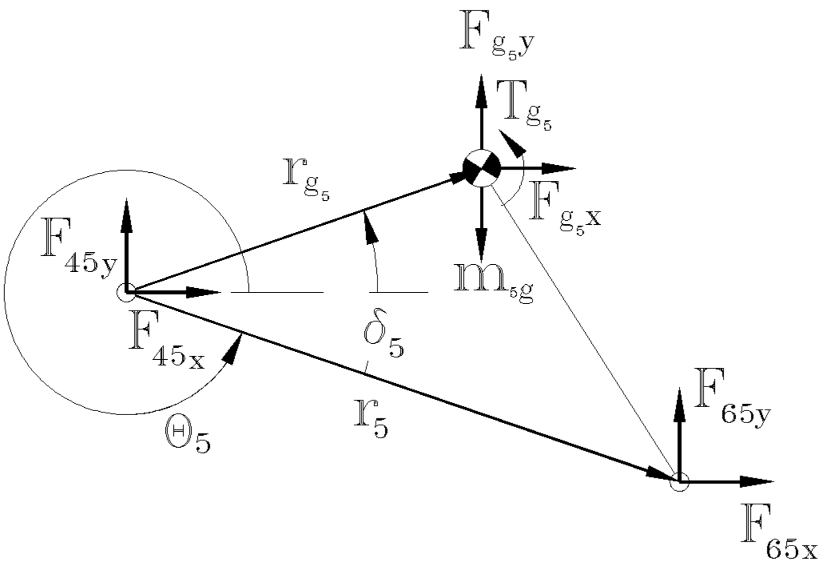


Figure 6. Free Body Diagram of Link 5.

For link 5, we get

$$F_{45x} + F_{65x} + F_{g5x} = 0 \tag{30a}$$

$$-m_5 g + F_{45y} + F_{65y} + F_{g6y} = 0 \tag{30b}$$

$$(-\mathbf{r}_{g5}) \times \mathbf{F}_{45} + (\mathbf{r}_5 - \mathbf{r}_{g5}) \times \mathbf{F_{65}} + T_{g5} = 0 \tag{30c}$$

where $\mathbf{F}_{45}$ and $\mathbf{F}_{65}$ are the joint forces acting on link 5 from links 4 and 6, F_{g5} and T_{g5} are the inertia force and inertia moment of link 5, m_5 is the mass of link 5, and $\mathbf{r}_{g5} = r_{g5} e^{i(\theta_5 + \delta_5)}$ is the position vector of the center of gravity of link 5 from joint B.

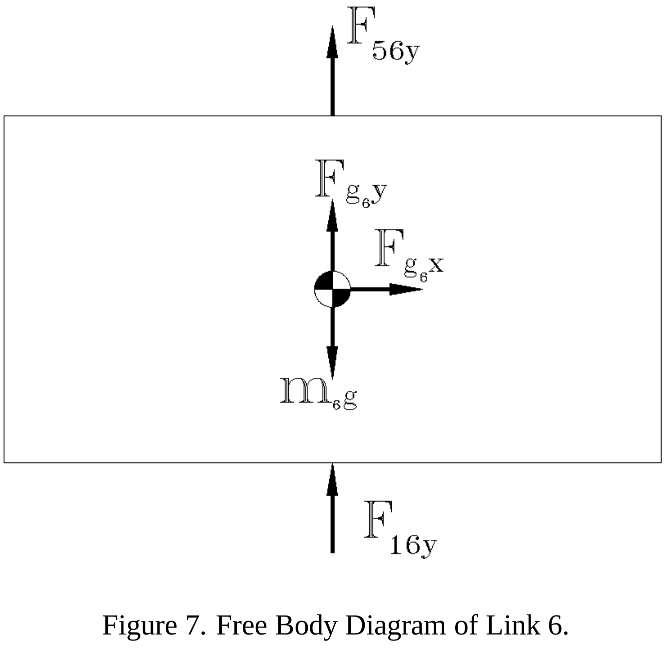


Figure 7. Free Body Diagram of Link 6.

For link 6, we get

$$F_{16x} + F_{56x} + F_{g6x} + F_L = 0 \tag{31a}$$

$$-m_6 g + F_{16y} + F_{56y} + F_{g6y} = 0 \tag{31b}$$

where $\mathbf{F}_{16}$ and $\mathbf{F}_{56}$ are the joint forces acting on link 6 from the ground and link 5, F_L is the output force on link 6 due to the input torque T_s, and m_6 is the mass of link 6. There are no torques on link 6 since it is assumed to be a point mass.

Equations 27c, 29c, and 30c can be expressed explicitly as

$$\begin{aligned}
&T_s - r_{g_2}\cos(\theta_2+\delta_2)F_{12y} + r_{g_2}\sin(\theta_2+\delta_2)F_{12x} \\
&+ \left[r_2\cos\theta_2 - r_{g_2}\cos(\theta_2+\delta_2)\right]F_{32y} - \left[r_2\sin\theta_2 - r_{g_2}\sin(\theta_2+\delta_2)\right]F_{32x} + T_{g_2} &= 0 \qquad \text{(32a)} \\
&-r_{g_4}\cos(\theta_4+\delta_4)F_{14y} + r_{g_4}\sin(\theta_4+\delta_4)F_{14x} \\
&+ \left[r_3\cos\theta_4 - r_{g_4}\cos(\theta_4+\delta_4)\right]\sin(\phi)F_{34} - \left[r_3\sin\theta_4 - r_{g_4}\sin(\theta_4+\delta_4)\right]\cos(\phi)F_{34} \\
&+ \left[r_4\cos\theta_4 - r_{g_4}\cos(\theta_4+\delta_4)\right]F_{54y} - \left[r_4\sin\theta_4 - r_{g_4}\sin(\theta_4+\delta_4)\right]F_{54x} + T_{g_4} &= 0 \qquad \text{(32b)} \\
&-r_{g_5}\cos(\theta_5+\delta_5)F_{45y} + r_{g_5}\sin(\theta_5+\delta_5)F_{45x} \\
&+ \left[r_5\cos\theta_2 - r_{g_5}\cos(\theta_5+\delta_5)\right]F_{65y} - \left[r_5\sin\theta_2 - r_{g_5}\sin(\theta_5+\delta_5)\right]F_{65x} + T_{g_5} &= 0 \qquad \text{(32c)}
\end{aligned}$$

Note that $F_{ijx} = -F_{jix}$ and $F_{ijy} = -F_{jiy}$. Equations 27-31 can be rewritten as 13 linear equations in terms of 14 unknowns F_{12x}, F_{12y}, F_{23x}, F_{23y}, F_{14x}, F_{14y}, F_{34}, F_{45x}, F_{45y}, F_{56x}, F_{56y}, F_{16x}, F_{16y} and T_s (13 joint reaction forces and one input torque). Since this model assumes a fictionless contact at all joints, the ground can not exert a horizontal force on the output slider (link 6), therefore $F_{16x} = 0$. This reduces the number of needed equations to 13 and the problem can thus be solved. The equations can now be collectively expressed as the symbolic matrix eqaution

$$\mathbf{Ax} = \mathbf{b} \tag{33}$$

where

$\mathbf{x} = (F_{12x}, F_{12y}, F_{23x}, F_{23y}, F_{14x}, F_{14y}, F_{34}, F_{45x}, F_{45y}, F_{56x}, F_{56y}, F_{16y}, T_s)^T$

is a vector consisting of the unknown forces and the input torque,

$\mathbf{b} = (F_{g_2x}, F_{g_2y} - m_2 g, T_{g_2}, F_{g_3x}, F_{g_3y} - m_3 g, F_{g_4x}, F_{g_4y} - m_4 g, T_{g_4},$
$F_{g_5x}, F_{g_5y} - m_5 g, T_{g_5}, F_{g_6x} + F_L, -m_6 g)^T$

is a vector that contains the external load, inertia forces, and inertia torques, and $\mathbf{A}$ is the $13x13$ square matrix

$$\begin{bmatrix}
-1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a & b & c & d & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\
0 & 0 & -1 & 0 & 0 & 0 & e & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & f & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & g & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & h & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & i & j & k & l & m & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & n & p & q & r & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0
\end{bmatrix}$$

where

$$
\begin{aligned}
a &= -r_{g_2}\sin\left(\theta_2+\delta_2\right)\\
b &= +r_{g_2}\cos\theta 2+\delta_2)\\
c &= -\left[r_2\sin\theta_2-r_{g_2}\sin\left(\theta_2+\delta_2\right)\right]\\
d &= +\left[r_2\cos\theta_2-r_{g_2}\cos\left(\theta_2+\delta_2\right)\right]\\
e &= \cos\left(\theta_4-\frac{\pi}{2}\right)\\
f &= \sin\left(\theta_4-\frac{\pi}{2}\right)\\
g &= -\cos\left(\theta_4-\frac{\pi}{2}\right)\\
h &= -\sin\left(\theta_4-\frac{\pi}{2}\right)\\
i &= -r_{g_4}\sin\left(\theta_4+\delta_4\right)\\
j &= +r_{g_4}\cos\left(\theta_4+\delta_4\right)\\
k &= \left[r_3\sin\theta_4-r_{g_4}\sin\left(\theta_4+\delta_4\right)\right]\left[\cos\left(\theta_4-\frac{\pi}{2}\right)\right]-\left[r_3\cos\theta_4-r_{g_4}\cos\left(\theta_4+\delta_4\right)\right]\left[\sin\left(\theta_4-\frac{\pi}{2}\right)\right]\\
l &= -\left[r_4\sin\theta_4-r_{g_4}\sin\left(\theta_4+\delta_4\right)\right]\\
m &= +\left[r_4\cos\theta_4-r_{g_4}\cos\left(\theta_4+\delta_4\right)\right]\\
n &= -r_{g_5}\sin\left(\theta_5+\delta_5\right)\\
p &= +r_{g_5}\cos\left(\theta_5+\delta_5\right)\\
q &= -\left[r_5\sin\theta_5-r_{g_5}\sin\left(\theta_5+\delta_5\right)\right]\\
r &= +\left[r_5\cos\theta_5-r_{g_5}\cos\left(\theta_5+\delta_5\right)\right]
\end{aligned}
$$

formed by using the angular position of each link and link parameters.

4 Description of the Software Package CQuickReturn

The CQuickReturn software package allows users to quickly analyze a Whitworth Quick Return Mechanism. It can be used by engineers or as teaching guide to students learning about computer aided design and analysis or mechanisms. The software utilizes the programming paradigm Ch which is a free C/C++ interpreter.

4.1 Getting Started with the Software Package CQuickReturn

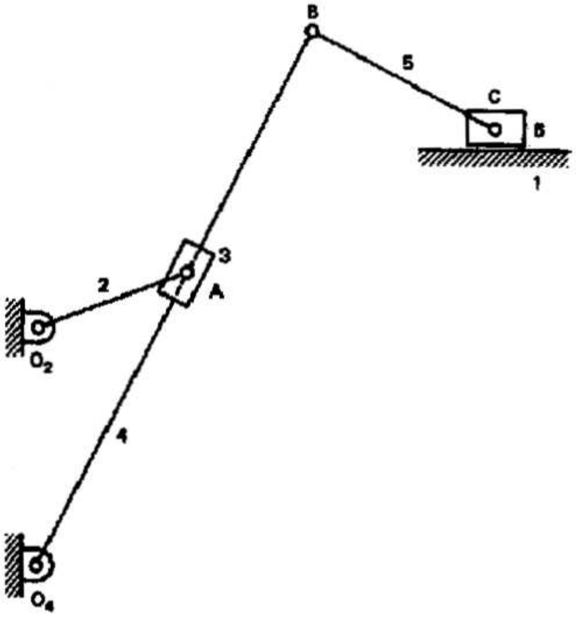


Figure 8. Example configuration of a Whitworth Quick Return Mechanism.

An example program, Program 1, is given to illustrate the basic features of the CQuickReturn software package. Figure 8 shows the configuration used for the example. The link lengths are given as $r_1 = 2.5cm$, $r_2 = 1.0cm$, $r_4 = 6.5cm$, and $r_5 = 3.0cm$. The output slider, link 6, is located $5.0cm$ above the lowest ground pin, O_1. The phase angle for the ground link is $\theta_1 = 90°$. This is a Whitworth quick return mechanism. The velocity profile of the output slider will be plotted and an animation of the mechanism will be created.

Program 1: A sample program for using the CQuickReturn software package.

```
#include <quickreturn.h>
int main(void)
{
   bool unit = SI;
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;
   double theta1 = M_PI/2, theta2 = -30*M_PI_180;
   double omega2 = -15;
   class CQuickReturn mechanism;
   class CPlot plot;
   mechanism.uscUnit(unit);
   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.setNumPoints(360);
   mechanism.plotSliderVel(&plot);
   mechanism.setNumPoints(50);
   mechanism.animation();
   return 0;
}
```

The first line of the program

```
#include <quickreturn.h>
```

includes the header file **quickreturn.h** which defines the **CQuickReturn** class, macros, and prototypes of member functions. Like all C/C++ programs, the program is started with the **main()** function. The next four lines

```
    bool unit = SI;
    double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;
    double theta1 = M_PI/2, theta2 = -30*M_PI_180;
    double omega2 = -15;
```

define the unit type, link lengths, ground and input link angles (in rad), and input link angular velocity (in rad/s) of the quick return mechanism. The lines

```
    class CQuickReturn mechanism;
    class CPlot plot;
```

constructs an object of the **CQuickReturn** class for the calculations and the **CPlot** class to display the output. The following three lines

```
    mechanism.uscUnit(unit);
    mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
    mechanism.setAngVel(omega2);
```

set the units, dimensions of each link, phase angle for link 1, and velocity of the input link as defined above. One line

```
    mechanism.setNumPoints(360);
```

is needed to set the number of points to be used for plotting. The line

```
    mechanism.plotSliderVel(&plot);
```

plots the velocity profile of the output slider, link 6. Figure 9 shows the velocity profile of the output slider after Program 1 has been executed.

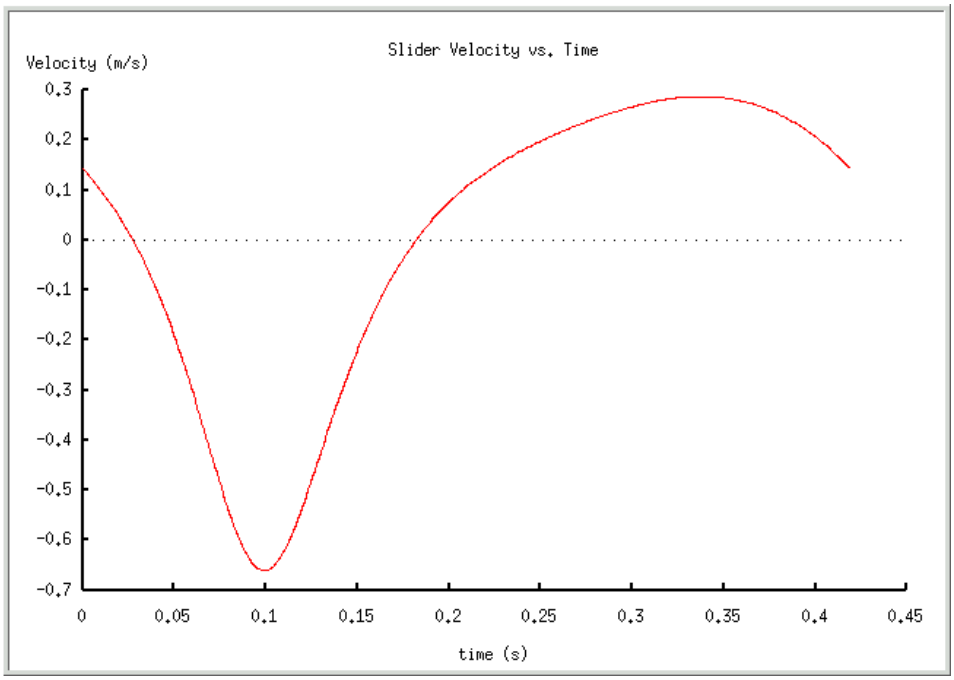


Figure 9. Program 1 output slider velocity plot.

The last two lines

```
mechanism.setNumPoints(50);
mechanism.animation();
```

reset the number of points and create a qanimate animation file that can be played by running *qanimate* with the file **animation.qnm** as its argument. Fewer points are used because the animation doesn't need as many as a plot to create decent output and it keeps the resulting file size down. The animation output when Program 1 is executed is shown in Figure 10. The menu bar in the qanimate window contains two menus, File and Options, and a series of buttons which manipulate the mechanism. The File menu allows users to quit the program and the Options menu allows users to change various display settings. The Next and Prev buttons control the mechanism's position, and the All button displays all mechanism positions at once. The Fast and Slow buttons change the speed of animation while the Go and Stop buttons start and stop animation.

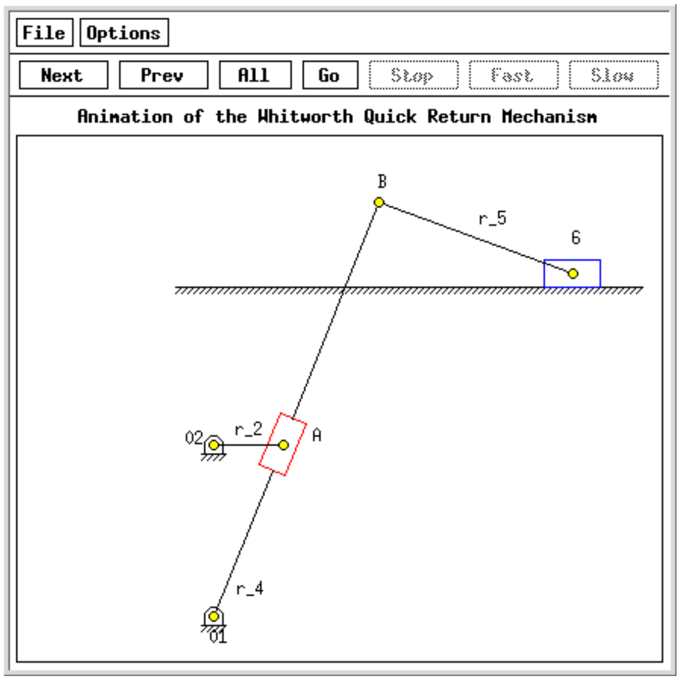


Figure 10. Program 1 animation output.

4.2 Solving Complex Equations

Complex numbers are used for analysis and design of the Whitworth quick return mechanism. A complex equation can be represented in a general form of

$$R_1e^{i\phi_1} + R_2e^{i\phi_2} = z_3 \tag{34}$$

where z_3 can be expressed in either Cartesian coordinates $x_3 + iy_3$ as complex(x3, y3), or polar coordinates $R_3e^{i\phi_3}$ as polar(R3, phi3). Many analysis and design problems for planar mechanisms can be formulated in this form. Because a complex equation can be partitioned into real and imaginary parts, two unknowns out of four parameters R_1, ϕ_1, R_2, and ϕ_2 can be solved in this equation.

Function complexsolve() in ch can be conveniently used to solve Equation 34. Detailed use of the function can be found in the Ch Mechanism Toolkit User's Guide [3].

5 Conclusion

This report presented a software package for the analysis and design of a Whitworth Quick Return Mechanism. The CQuickReturn class can be used to calculate or plot the position, velocity, and acceleration of the mechanism. The CQuickReturn class also provides a function to create an Xlinkage animation file display the changes in configuration of the mechanism overtime. This package is well suitable for rapid prototyping, distance learning, and as a teaching aid.

6 Acknowledgments

We would like to thank professor Harry Cheng for providing this project.

7 Reference

1. Cheng, H. H., 2004. *The Ch Language Environment User's Guide*. URL `http://www.softintegration.com`.
2. Erdman, A. G., and Sandor, G. N., 1997. Mechanism design: Analysis and synthesis, volume 1, 3rd edition.
3. Cheng, H. H., 2004. *The Ch Mechanism Toolkit User's Guide*. URL `http://www.softintegration.com`.

8 Appendix A: CQuickReturn API

---

quickreturn.h

The headerfile **quickreturn.h** contains the definition of the Whitworth Quick Return Mechanism **CQuickReturn** class, defined macros used with the **CQuickReturn** class, and definitions of the **CQuickReturn** class member functions.

The Whitworth Quick Return Mechanism class **CQuickReturn** is suitable for rapid integration into any standard user code. It gives users the ability to easily compute the position, velocity, acceleration, and forces of a Whitworth Quick Return Mechanism making it suitable for rapid prototyping and as a teaching aid.

CQuickReturn

The **CQuickReturn** class can be used in the analysis and design of a Whitworth Quick Return Mechanism. The member functions of the **CQuickReturn** class allows for the calculation of the position, velocity, acceleration, and forces of a given Whitworth Quick Return Mechanism.

Public Data

None

Public Member Functions

Functions	Descriptions
˜CQuickReturn()	Class destructor.
animation()	Create a qanimate file to animate the mechanism.
CQuickReturn()	Class constructor. Creates an instance of the class and initializes all private data members.
displayPosition()	Create a qanimate file to display the configuration of the mechanism.
getAngAccel()	Get the angular acceleration of a link.
getAngPos()	Get the angular position of a link.
getAngVel()	Get the angular velocity of a link.
getForces()	Get the forces acting on the mechanism.
getPointAccel()	Get the point acceleration of a link.
getPointPos()	Get the point position of a link.
getPointVel()	Get the point velocity of a link.
getRequiredTorque()	Get the required torque for the mechanism.
plotAngAccel()	Plot the angular acceleration of a link versus time.
plotAngPos()	Plot the angular position of a link versus time.
plotAngVel()	Plot the angular velocity of a link versus time.
plotCGaccel()	Plot the acceleration of the CQ of a link versus time.
plotForce()	Plot all of the forces versus time.
plotSliderAccel()	Plot the acceleration of the output slider versus time.
plotSliderPos()	Plot the position of the output slider versus time.

plotSliderVel()	Plot the velocity of the output slider versus time.
plotTorque()	Plot the required input torque versus time.
setAngVel()	Set angular velocity of link 2.
setForce()	Set the load force on the mechanism.
setGravityCenter()	Set the center of gravity parameters of the links.
setInertia()	Set the inertial parameters of the links.
setLinks()	Set lengths of links.
setMass()	Set the mass of the links.
setNumPoints()	Set the number of points for plotting and animating.
sliderAccel()	Get the output slider's acceleration.
sliderPos()	Get the output slider's position.
sliderRange()	Get the poition range of the slider.
sliderVel()	Get the output slider's velocity.
uscUnit()	Set to output in USC units or SI units.

Constants

The following macros are defined for the **CQuickReturn** class.

Macros	Descriptions
ALL_MAG_PLOTS	Identifier for all of the force magnitude plots.
ALL_FORCE_PLOTS	Identifier for all of the force plots.
F12X	Identifier for the force plot of F_{12x}.
F12Y	Identifier for the force plot of F_{12y}.
F14X	Identifier for the force plot of F_{14x}.
F14Y	Identifier for the force plot of F_{14y}.
F16Y	Identifier for the force plot of F_{16y}.
F23X	Identifier for the force plot of F_{23x}.
F23Y	Identifier for the force plot of F_{23y}.
F45X	Identifier for the force plot of F_{45x}.
F45Y	Identifier for the force plot of F_{45y}.
F56X	Identifier for the force plot of F_{56x}.
F56Y	Identifier for the force plot of F_{56y}.
MAG_F12	Identifier for the force plot of the magnitude of F_{12}.
MAG_F14	Identifier for the force plot of the magnitude of F_{14}.
MAG_F23	Identifier for the force plot of the magnitude of F_{23}.
MAG_F34	Identifier for the force plot of the magnitude of F_{34}.
MAG_F56	Identifier for the force plot of the magnitude of F_{56}.
MAG_F45	Identifier for the force plot of the magnitude of F_{45}.
QR_LINK_2	Identifier for link 2.
QR_LINK_4	Identifier for link 4.
QR_LINK_5	Identifier for link 5.
QR_LINK_2_CG	Identifier for the CG of link 2.
QR_LINK_4_CG	Identifier for the CG of link 4.
QR_LINK_5_CG	Identifier for the CG of link 5.
QR_POINT_A	Identifier for point A.
QR_POINT_B	Identifier for point B.

CQuickReturn::˜**CQuickReturn**

Synopsis
#include <**quickreturn.h**>
˜CQuickReturn();

Purpose
Reserved for future use.

Return Value
None

Parameters
None

Description
None

Example None

Output None

CQuickReturn::**animation**

Synopsis
#include <**quickreturn.h**>
void animation(. . . /* [**int** *outputtype*, **string_t** *filename*] */);

Syntax
animation();
animation(*outputtype*);
animation(*outputtype*, *filename*);

Purpose
Creates a qanimate animation file of the Whitworth quick return mechanism.

Return Value
None

Parameters

outputtype	optional arguement for setting the output type of the animation.
filename	optional arguement to give an animation output file a name, should have *.qnm* extention, necessary when output is file, optional when output is a stream.

Description

Creates a qanimate animation file of the Whitworth quick return mechanism. This functions calculates the position of all of the points of the mechanism as the angle of link 2 is changed from 0 to $2 * \pi$. It then writes the required primative descriptions to a file that can then be run with qanimate. The *outputtype* can be one of the following three options:

QANIMATE_OUTPUTTYPE_DISPLAY - This will write the output to a temp file that will be erased after the animation window is closed. This is the default for *outputtype*. The *filename* arguement is not necessary.

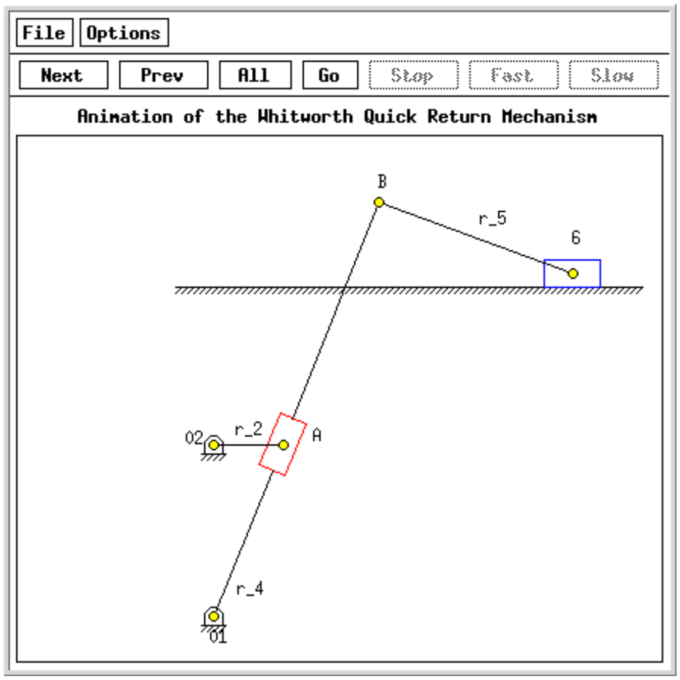
QANIMATE_OUTPUTTYPE_STREAM - This writes the file to stdout and is used for streaming over the internet. The *filename* arguement is optional.

QANIMATE_OUTPUTTYPE_FILE - This creates a file with the name *filename*. It can be run in with the command *qanimate filename* within Ch.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   int numpoints = 25;
   bool unit = SI;

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.uscUnit(unit);
   mechanism.setNumPoints(numpoints);
   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.animation(QANIMATE_OUTPUTTYPE_DISPLAY);

   return 0;
}
```

Output



CQuickReturn::**CQuickReturn**

Synopsis
#include <**quickreturn.h**>
CQuickReturn();

Purpose
Class constructor of the **CQuickReturn** class.

Return Value
None

Parameters
None

Description
Constructs on object of **CQuickReturn** class type. This function also initailizes all of the private data memebers.

Example None

Output None

CQuickReturn::**displayPosition**

Synopsis

#include <quickreturn.h>
void displayPosition(**double** *theta2*, . . . /* [**int** *outputtype*, **string_t** *filename*] */);

Syntax
displayPosition(*theta2*);
displayPosition(*theta2*, *outputtype*);
displayPosition(*theta2*, *outputtype*, *filename*);

Purpose
Creates a qanimate animation file that displays a static position of the Whitworth quick return mechanism.

Return Value
None

Parameters

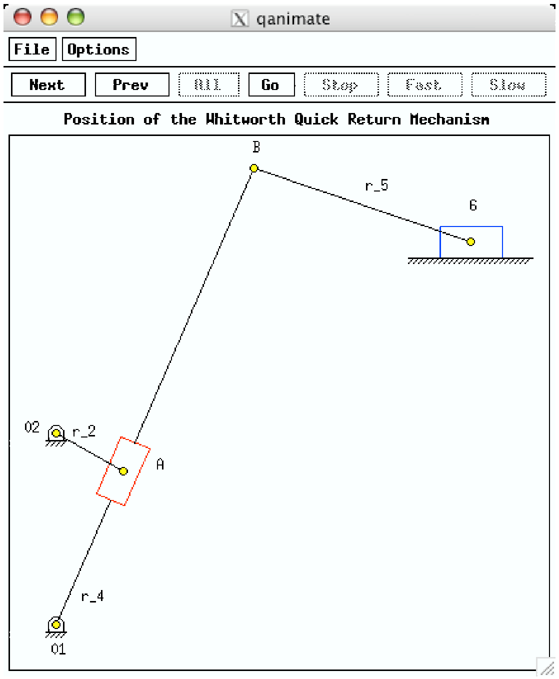
theta2	angle of input link used when displaying mechanism
outputtype	optional arguement for setting the output type of the animation.
filename	optional arguement to give an animation output file a name, should have *.qnm* extention, necessary when output is file, optional when output is a stream.

Description
Creates a qanimate animation file of the Whitworth quick return mechanism that shows the mechanism statically with the input link at the angle specified by the user. It then writes the required primative descriptions to a file that can then be run with qanimate. The *outputtype* can be one of the following three options:
QANIMATE_OUTPUTTYPE_DISPLAY - This will write the output to a temp file that will be erased after the animation window is closed. This is the default for *outputtype*. The *filename* arguement is not necessary.
QANIMATE_OUTPUTTYPE_STREAM - This writes the file to stdout and is used for streaming over the internet. The *filename* arguement is optional.
QANIMATE_OUTPUTTYPE_FILE - This creates a file with the name *filename*. It can be run in with the command *qanimate filename* within Ch.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta2 = -30*M_PI_180;                                            //rad
   int numpoints = 25;
   bool unit = SI;

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.uscUnit(unit);
   mechanism.setNumPoints(numpoints);
   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.animation(theta2, QANIMATE_OUTPUTTYPE_DISPLAY);

   return 0;
}
```

Output



Position of the Whitworth Quick Return Mechanism

CQuickReturn::getAngAccel

Synopsis

#include <**quickreturn.h**>
double getAngAccel(**double** *theta2*, **int** *link*);

Purpose

Acquires the angular acceleration of any link.

Return Value

Returns the wanted angular acceleration.

Parameters

theta2 The angle of link 2 used to calculate the instantaneous angular acceleration of a link.
link enumerated value identifying which link to calculate the angular accerleration of.

Description

This function calculates the angular acceleration of any link and returns the calculated value.

Example

```
#include <stdio.h>
#include <quickreturn.h>

int main(void)
{
    /* Set up required parameters */
```

```
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   double omega2 = -15.0;                                                    //rad/sec
   bool unit = SI;
   double angularaccel= 0;

    /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.uscUnit(unit);

   angularaccel = mechanism.getAngAccel(theta2, QR_LINK_5);

   printf("The angular acceleration of link 5 = %f\n", angularaccel);

   return 0;
}
```

Output

```
The angular acceleration of link 5 = 37.836011
```

CQuickReturn::**getAngPos**

Synopsis

#include <**quickreturn.h**>
double getAngPos(**double** *theta2*, **int** *link*);

Purpose

Acquires the angular position of any link.

Return Value

Returns the wanted angular position.

Parameters

theta2	The angle of link 2 used to calculate the instantaneous angular position of a link.
link	enumerated value identifying which link to calculate the angular position of.

Description

This function calculates the angular position of any link and returns the calculated value.

Example

```
#include <stdio.h>
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
```

```
   bool unit = SI;
   double angularpos = 0;

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.uscUnit(unit);

   angularpos = mechanism.getAngPos(theta2, QR_LINK_5);

   printf("The angular position of link 5 = %f\n", angularpos);

   return 0;
}
```

Output

```
The angular position of link 5 = -0.352274
```

CQuickReturn::getAngVel

Synopsis
#include <quickreturn.h>
double getAngVel(double *theta2*, **int** *link*);

Purpose
Acquires the angular velocity of any link.

Return Value
Returns the wanted angular velocity.

Parameters

theta2	The angle of link 2 used to calculate the instantaneous angular velocity of a link.
link	enumerated value identifying which link to calculate the angular velocity of.

Description
This function calculates the angular velocity of any link and returns the calculated value.

Example

```
#include <stdio.h>
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   double omega2 = -15.0;                                                    //rad/sec
   bool unit = SI;
   double angularvel = 0;

   /* Create CQuickReturn Object */
```

```
   CQuickReturn mechanism;

   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.uscUnit(unit);

   angularvel = mechanism.getAngVel(theta2, QR_LINK_5);

   printf("The angular velocity of link 5 = %f\n", angularvel);

   return 0;
}
```

Output

```
The angular velocity of link 5 = 1.773780
```

CQuickReturn::getForces

Synopsis
#include <quickreturn.h>
void getForces(**double** *theta2*, **arraydouble** *forces*);

Purpose
Acquires the forces acting on the mechanism.

Return Value
None

Parameters

theta2 The angle of link 2 used to calculate the forces on all the links.
forces An array of 12 elements to store the calculated forces

Description
This function calculates the forces acting on the mechanism and stored the calculated values in the array *forces*.

Example

```
#include <stdio.h>
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;           //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   double omega2 = -15.0;                                                    //rad/sec
   double rg2 = 0.0125, rg4 = 0.0275, rg5 = 0.0250;                          //meters
   double delta2 = 30*M_PI/180, delta4 = 15*M_PI/180, delta5 = 30*M_PI/180;  //rad
   double ig2 = 0.012, ig4 = 0.119, ig5 = 0.038;                             //kg*m^2
   double m2 = 0.8, m3 = 0.3, m4 = 2.4, m5 = 1.4, m6 = 0.3;                  //kg
   double fl = -100;                                                         //N
```

```
   int numpoints = 360;
   bool unit = SI;
   array double forces[12] = {0};

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.setGravityCenter(rg2, rg4, rg5, delta2, delta4, delta5);
   mechanism.setInertia(ig2, ig4, ig5);
   mechanism.setMass(m2, m3, m4, m5, m6);
   mechanism.setForce(fl);
   mechanism.setNumPoints(numpoints);
   mechanism.uscUnit(unit);
   mechanism.getForces(theta2, forces);

   printf("Force Name    Value (N)\n");
   printf("F12x          %f\n", forces[0]);
   printf("F12y          %f\n", forces[1]);
   printf("F23x          %f\n", forces[2]);
   printf("F23y          %f\n", forces[3]);
   printf("F14x          %f\n", forces[4]);
   printf("F14y          %f\n", forces[5]);
   printf("F34           %f\n", forces[6]);
   printf("F45x          %f\n", forces[7]);
   printf("F45y          %f\n", forces[8]);
   printf("F56x          %f\n", forces[9]);
   printf("F56y          %f\n", forces[10]);
   printf("F16y          %f\n", forces[11]);

   return 0;
}
```

Output

```
Force Name    Value (N)
F12x          -162.639205
F12y          73.672264
F23x          -160.690648
F23y          66.949264
F14x          70.111066
F14y          58.739047
F34           -172.342129
F45x          93.816762
F45y          -86.189132
F56x          99.042701
F56y          -102.746351
F16y          105.689351
```

CQuickReturn::getPointAccel

Synopsis

#include <quickreturn.h>
double complex getPointAccel(**double** *theta2*, **int** *point*);

Purpose
Acquires the acceleration of any point.

Return Value
Returns the wanted acceleration.

Parameters
theta2	The angle of link 2 used to calculate the acceleration of a point.
point	An enumerated value identifying which point to calculate the acceleration of.

Description
This function calculates the acceleration of any point and returns the calculated value.

Example

```
#include <stdio.h>
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   double omega2 = -15.0;                                                    //rad/sec
   bool unit = SI;
   double complex pointaccel = 0;

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.uscUnit(unit);

   pointaccel = mechanism.getPointAccel(theta2, QR_POINT_A);

   printf("The acceleration of point A is %f\n", pointaccel);

   return 0;
}
```

Output

```
The acceleration of point A is complex(-2.250000,0.000000)
```

CQuickReturn::getPointPos

Synopsis
#include <quickreturn.h>
double complex getPointPos(**double** *theta2*, **int** *point*);

Purpose
Acquires the position of any point.

Return Value
Returns the wanted position.

Parameters
theta2 The angle of link 2 used to calculate the position of a point.
point An enumerated value identifying which point to calculate the position of.

Description
This function calculates the position of any point and returns the calculated value.

Example

```
#include <stdio.h>
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   bool unit = SI;
   double complex pointpos = 0;

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.uscUnit(unit);

   pointpos = mechanism.getPointPos(theta2, QR_POINT_A);

   printf("The position of point A is %f\n", pointpos);

   return 0;
}
```

Output

```
The position of point A is complex(0.010000,0.025000)
```

CQuickReturn::getPointVel

Synopsis
#include <quickreturn.h>
double complex getPointVel(double *theta2*, **int** *point*);

Purpose
Acquires the velocity of any point.

Return Value

Returns the wanted velocity.

Parameters

theta2 The angle of link 2 used to calculate the velocity of a point.
point An enumerated value identifying which point to calculate the velocity of.

Description

This function calculates the velocity of any point and returns the calculated value.

Example

```
#include <stdio.h>
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   double omega2 = -15.0;                                                    //rad/sec
   bool unit = SI;
   double complex pointvel = 0;

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.uscUnit(unit);

   pointvel = mechanism.getPointPos(theta2, QR_POINT_A);

   printf("The velocity of point A is %f\n", pointvel);

   return 0;
}
```

Output

```
The velocity of point A is complex(0.010000,0.025000)
```

CQuickReturn::getRequiredTorque

Synopsis

#include <quickreturn.h>
double getRequiredTorque(double *theta2*);

Purpose

Calculates the required input torque at a given angle *theta2*.

Return Value

Returns the calculated required input torque.

Parameters

theta2 The angle of link 2 used to calculate the required input torque.

Description

Calculates the required input torque at a given angle of link 2 and returns the calculated value.

Example

```
#include <stdio.h>
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   double omega2 = -15.0;                                                    //rad/sec
   double rg2 = 0.0125, rg4 = 0.0275, rg5 = 0.0250;                          //meters
   double delta2 = 30*M_PI/180, delta4 = 15*M_PI/180, delta5 = 30*M_PI/180;  //rad
   double ig2 = 0.012, ig4 = 0.119, ig5 = 0.038;                             //kg*mˆ2
   double m2 = 0.8, m3 = 0.3, m4 = 2.4, m5 = 1.4, m6 = 0.3;                  //kg
   double fl = -100;                                                         //N
   int numpoints = 360;
   bool unit = SI;
   double requiredtorque = 0;

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.setGravityCenter(rg2, rg4, rg5, delta2, delta4, delta5);
   mechanism.setInertia(ig2, ig4, ig5);
   mechanism.setMass(m2, m3, m4, m5, m6);
   mechanism.setForce(fl);
   mechanism.setNumPoints(numpoints);
   mechanism.uscUnit(unit);

   requiredtorque = mechanism.getRequiredTorque(theta2);

   printf("The required torque is %f\n", requiredtorque);

   return 0;
}
```

Output

```
The required torque is -0.006734
```

CQuickReturn::**plotAngAccel**

Synopsis

#include <quickreturn.h>

void plotAngAccel(**CPlot** **plot*);

Purpose

Plots the angular acceleration of links 4 and 5 over time.

Return Value

None

Parameters

&plot pointer to the plot object declared in calling program for displaying output.

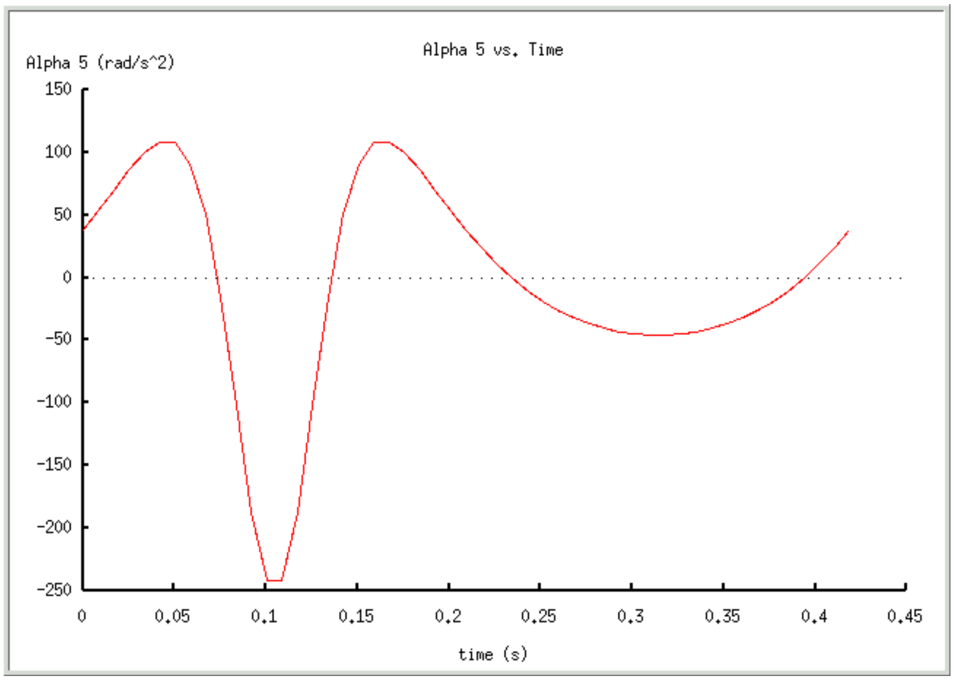
Description

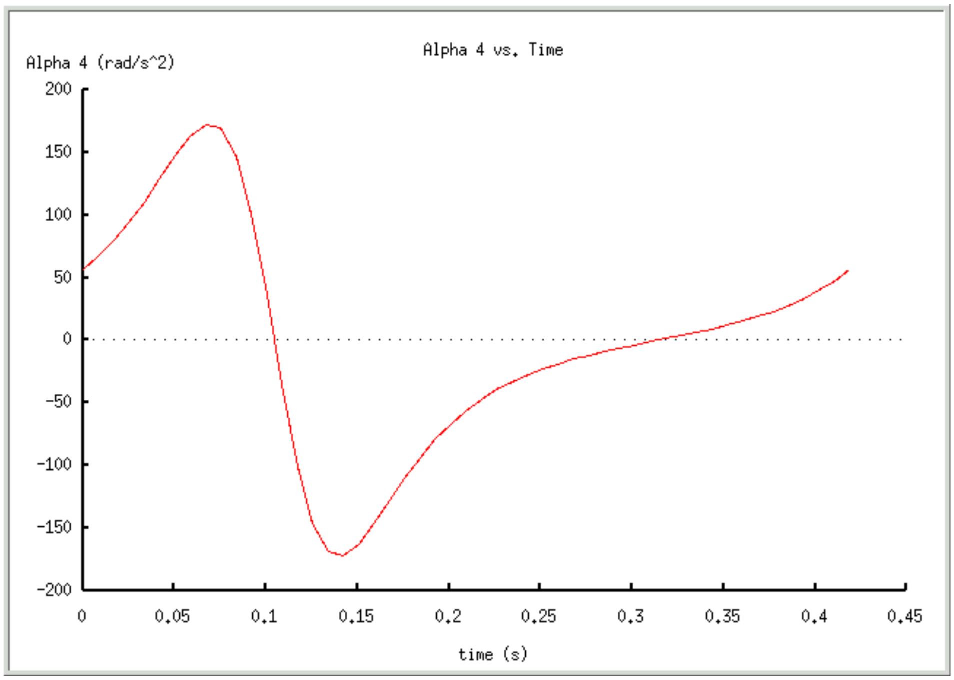
Calculates the angular acceleration of links 4 and 5 as the angle of link 2 changes over time. It then creates plots of the angular accerleration of links 4 and 5 over time.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   double omega2 = -15.0;                                                    //rad/sec
   int numpoints = 360;
   bool unit = SI;

   /* Create CQuickReturn and Plot Objects */
   CQuickReturn mechanism;
   CPlot plot;

   mechanism.uscUnit(unit);
   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.setNumPoints(numpoints);
   mechanism.plotAngAccel(&plot);

   return 0;
}
```

Output





CQuickReturn::plotAngPos

Synopsis

#include <quickreturn.h>

void plotAngPos(CPlot **plot*);

Purpose

Plots the angular position links 4 and 5 over time.

Return Value

None

Parameters

&plot pointer to the plot object declared in calling program for displaying output.

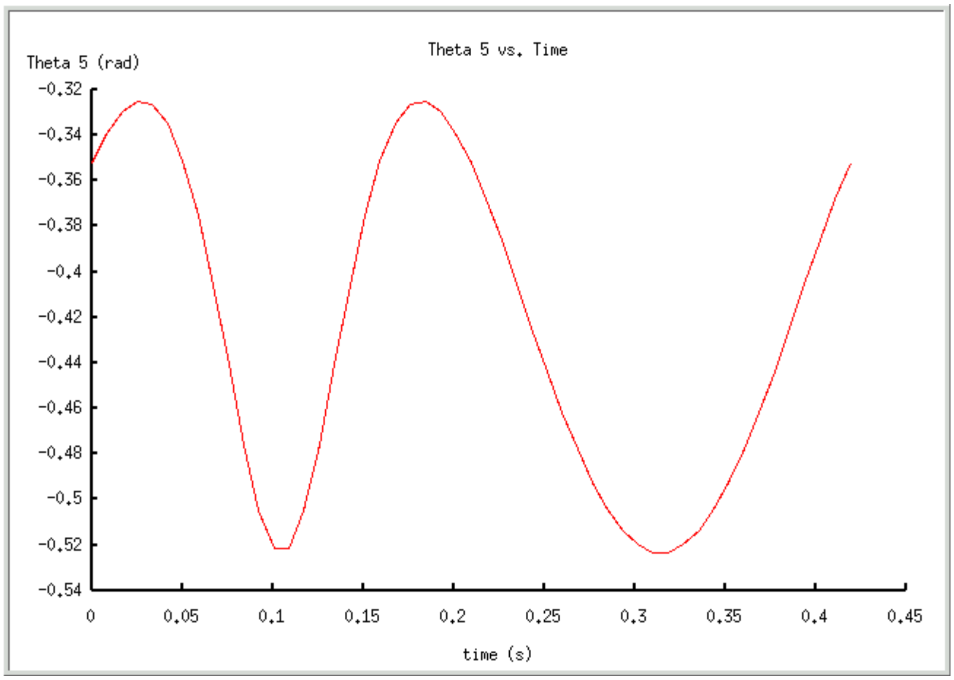
Description

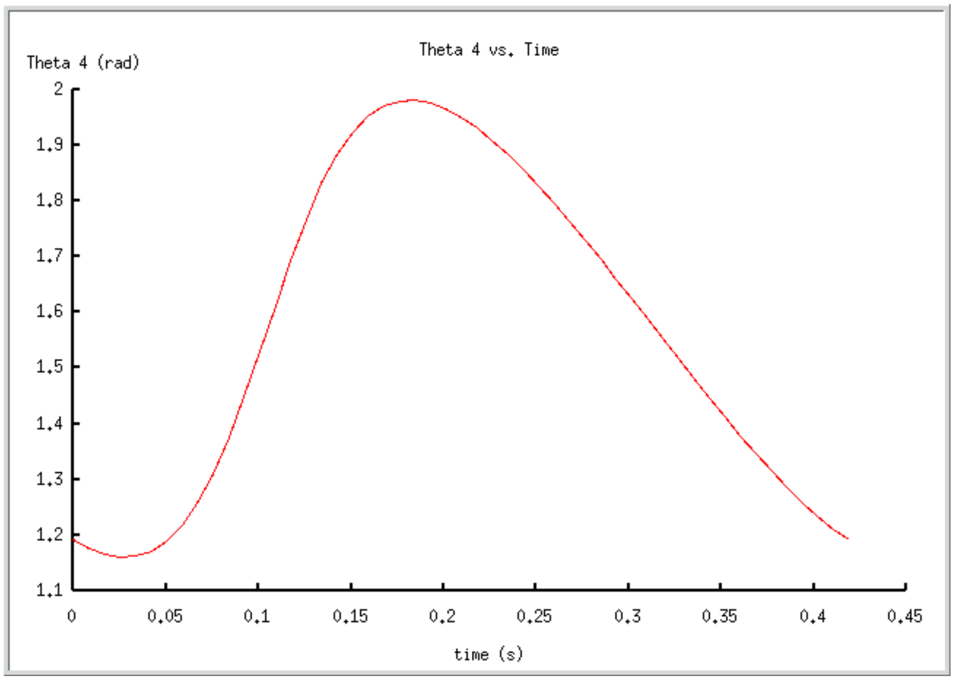
Calculates the angular position of links 4 and 5 as the angle of link 2 changes over time. It then creates plots of the position of links 4 and 5 over time.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                     //rad
   double theta2 = 0.0;                                                        //rad
   int numpoints = 360;
   bool unit = SI;

   /* Create CQuickReturn and Plot Objects */
   CQuickReturn mechanism;
   CPlot plot;

   mechanism.uscUnit(unit);
   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setNumPoints(numpoints);
   mechanism.plotAngPos(&plot);

   return 0;
}
```

Output





CQuickReturn::plotAngVel

Synopsis

#include <quickreturn.h>

void plotAngVel(CPlot **plot*);

Purpose

Plots the angular velocity of links 4 and 5 over time.

Return Value

None

Parameters

&plot pointer to the plot object declared in calling program for displaying output.

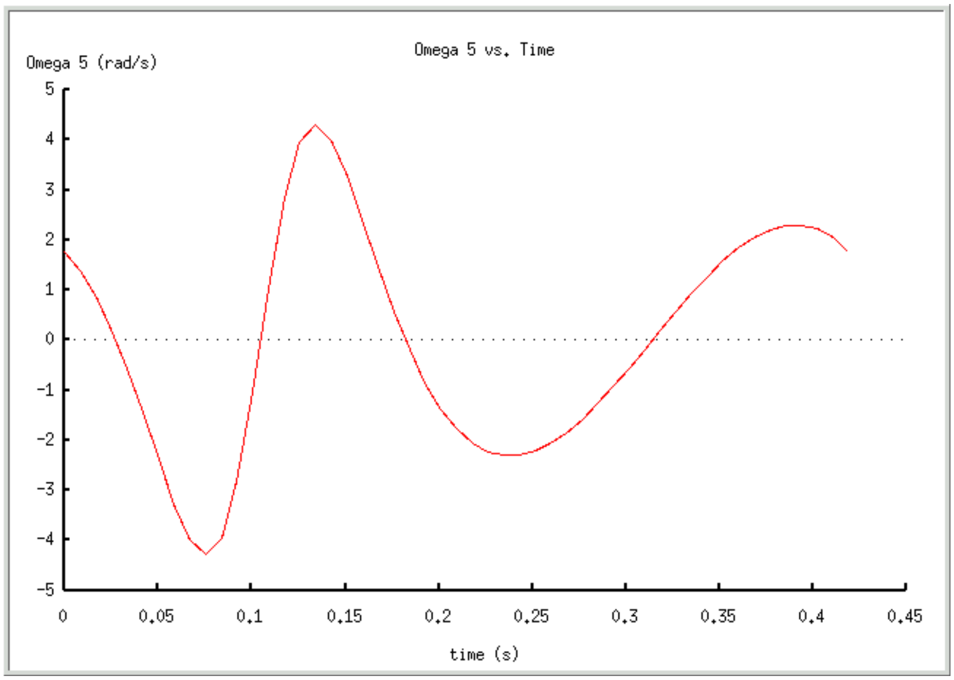
Description

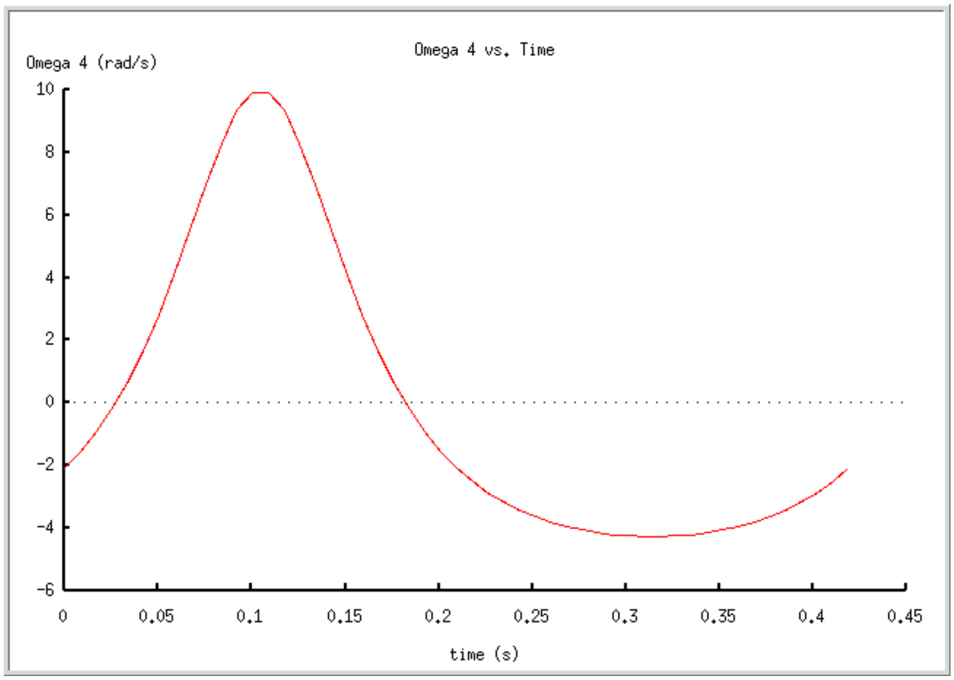
Calculates the angular velocity of links 4 and 5 as the angle of link 2 changes over time. It then creates plots of the angular velocity of links 4 and 5 over time.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   double omega2 = -15.0;                                                    //rad/sec
   int numpoints = 360;
   bool unit = SI;

   /* Create CQuickReturn and Plot Objects */
   CQuickReturn mechanism;
   CPlot plot;

   mechanism.uscUnit(unit);
   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.setNumPoints(numpoints);
   mechanism.plotAngVel(&plot);

   return 0;
}
```

Output





CQuickReturn::plotCGaccel

Synopsis
#include <quickreturn.h>
void plotCGaccel(CPlot **plot*);

Purpose

Plots the CG acceleration of links 2, 4, and 5.

Return Value

None

Parameters

&plot pointer to the plot object declared in calling program for displaying output.

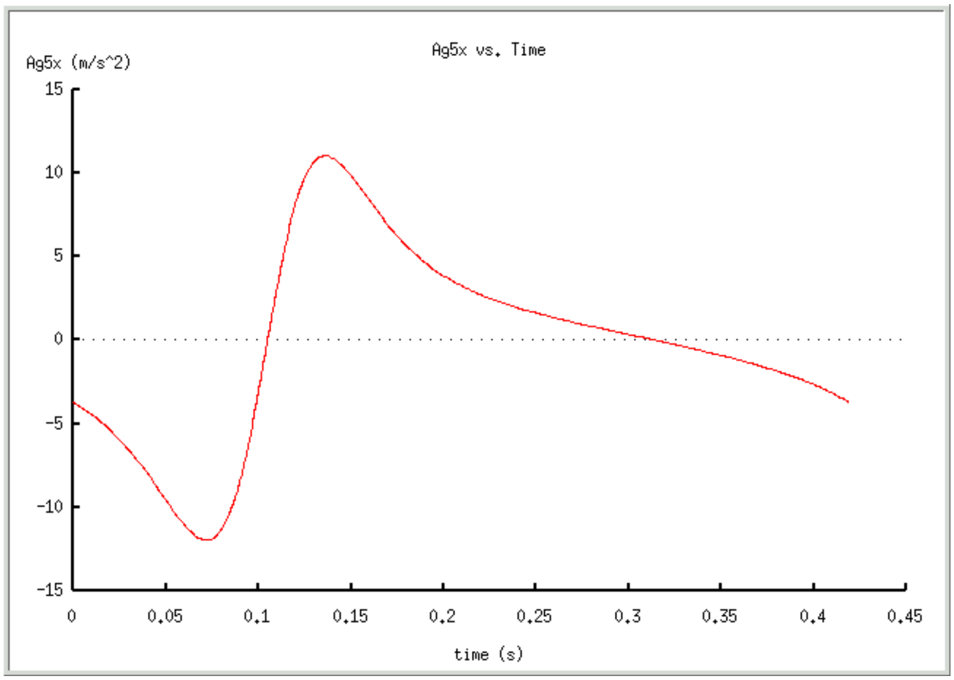
Description

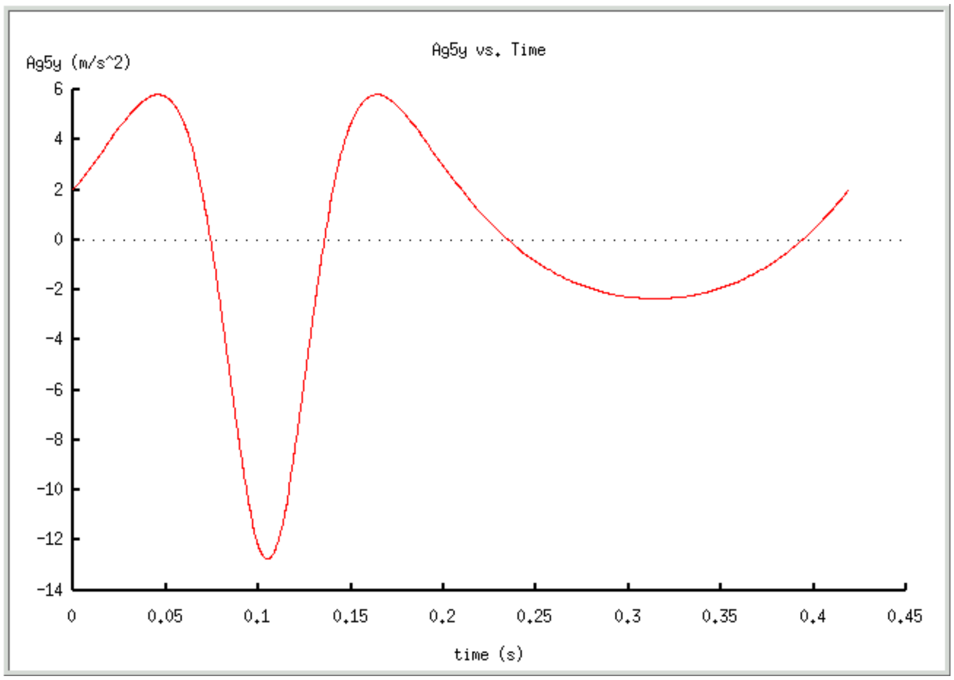
Calculates the CG acceleration of links 2, 4, and 5 as the angle of link 2 changes over time. It then creates plots of the CG acceleration of links 2, 4, and 5 over time.

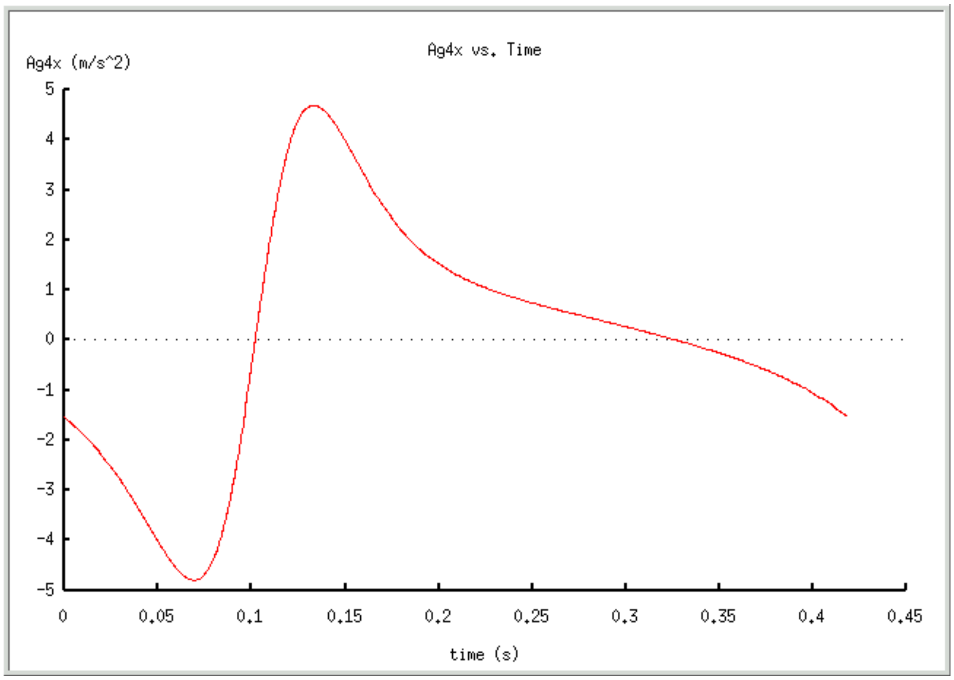
Example

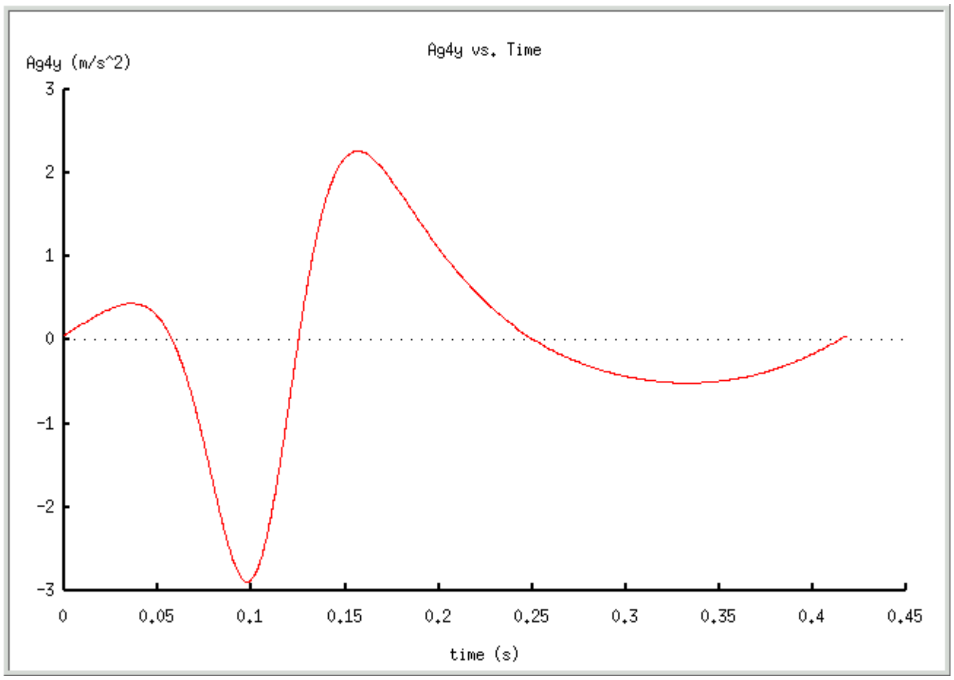
```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   double omega2 = -15.0;                                                    //rad/sec
   double rg2 = 0.0125, rg4 = 0.0275, rg5 = 0.0250;                          //meters
   double delta2 = 30*M_PI/180, delta4 = 15*M_PI/180, delta5 = 30*M_PI/180;  //rad
   int numpoints = 360;
   bool unit = SI;

   /* Create CQuickReturn and Plot Objects */
   CQuickReturn mechanism;
   CPlot plot;

   mechanism.uscUnit(unit);
   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.setGravityCenter(rg2, rg4, rg5, delta2, delta4, delta5);
   mechanism.setNumPoints(numpoints);
   mechanism.plotCGaccel(&plot);

   return 0;
}
```

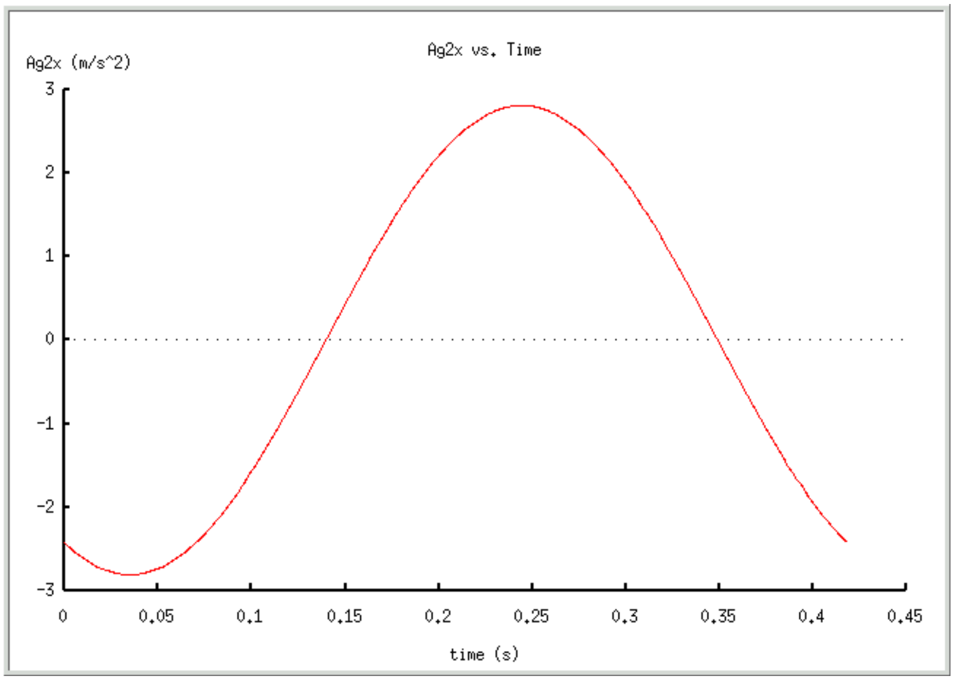
Output

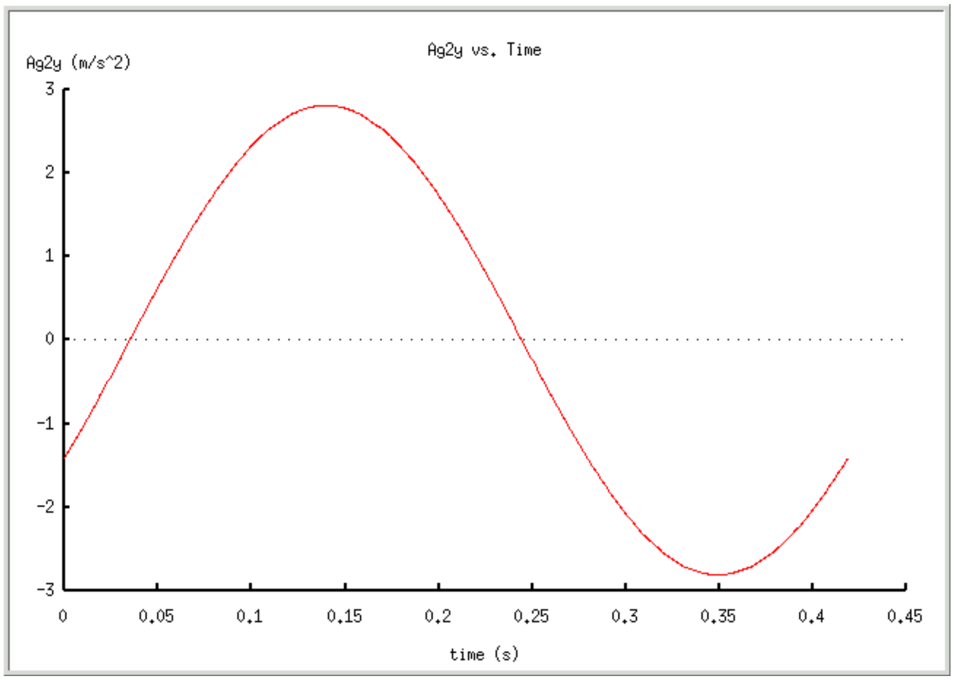

Ag5x vs. Time
Ag5x (m/s^2)
15
10
5
0
-5
-10
-15
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
time (s)


Ag5y vs. Time
Ag5y (m/s^2)
6
4
2
0
-2
-4
-6
-8
-10
-12
-14
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
time (s)


Ag4x vs. Time
Ag4x (m/s^2)
5
4
3
2
1
0
-1
-2
-3
-4
-5
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
time (s)


Ag4y vs. Time
Ag4y (m/s^2)
3
2
1
0
-1
-2
-3
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
time (s)





CQuickReturn::plotForce

Synopsis
#include <quickreturn.h>
void plotForce(int *plot*, **CPlot** **plot*);

Purpose

Plots all of the forces acting on the mechanism over time.

Return Value

None

Parameters

plot	An enumerated which determins what force plots to create.
&plot	pointer to the plot object declared in calling program for displaying output.

Parameter Discussion

plot is an integer that represents which force plots to create. The enumerated values of *F12X*, *F12Y*, *MAG_F12*, and so on have been defined in the header file **quickreturn.h**. *plot* is bitwise checked to see which force plots are wanted. For more option, see the MACRO definitions in the **quickreturn.h** section. *&plot* is a pointer to the CPlot class variable defined in the calling program. All the plotting member functions use this parameter.

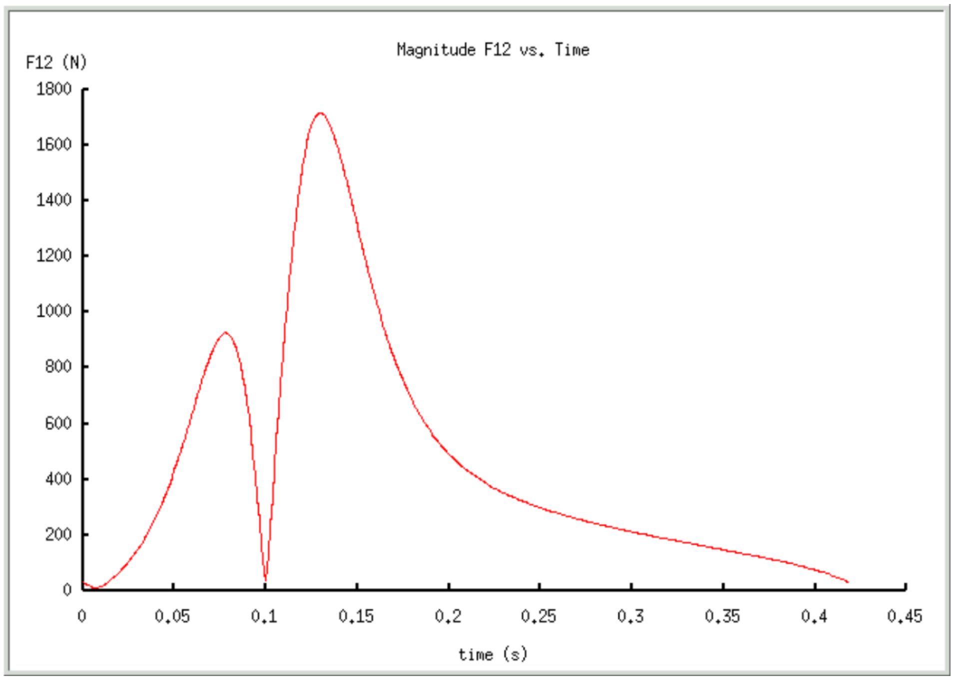
Description

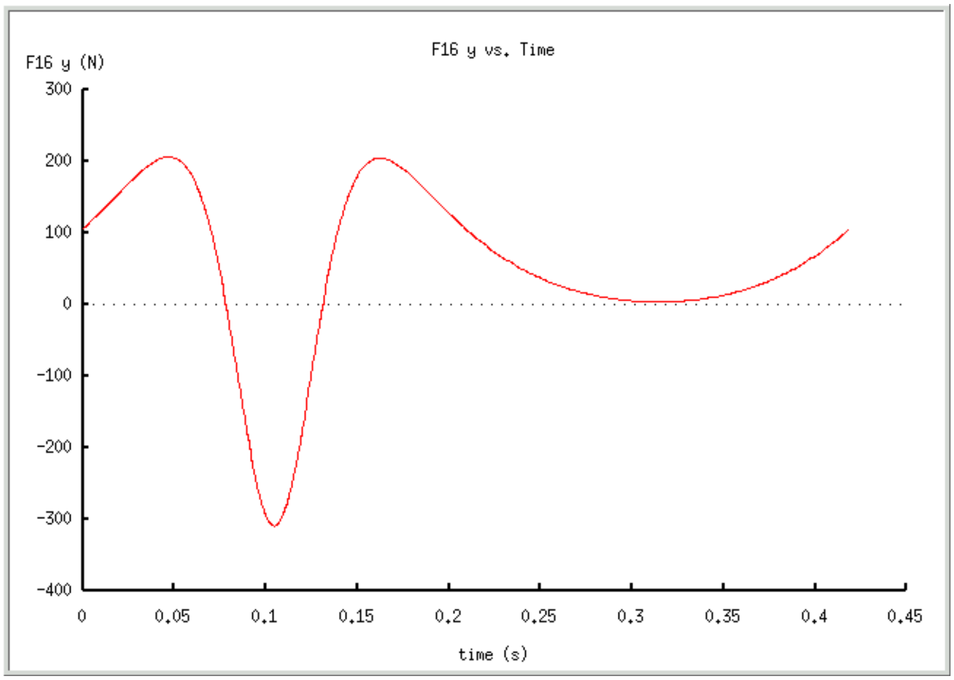
Calculates all of the forces acting on the mechanism as the angle of link 2 changes over time. It then creates plots of the forces over time.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   double omega2 = -15.0;                                                    //rad/sec
   double rg2 = 0.0125, rg4 = 0.0275, rg5 = 0.0250;                          //meters
   double delta2 = 30*M_PI/180, delta4 = 15*M_PI/180, delta5 = 30*M_PI/180;  //rad
   double ig2 = 0.012, ig4 = 0.119, ig5 = 0.038;                             //kg*m^2
   double m2 = 0.8, m3 = 0.3, m4 = 2.4, m5 = 1.4, m6 = 0.3;                  //kg
   double fl = -100;                                                         //N
   int numpoints = 360;
   bool unit = SI;

   /* Create CQuickReturn and Plot Objects */
   CQuickReturn mechanism;
   CPlot plot;

   mechanism.uscUnit(unit);
   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.setGravityCenter(rg2, rg4, rg5, delta2, delta4, delta5);
   mechanism.setInertia(ig2, ig4, ig5);
   mechanism.setMass(m2, m3, m4, m5, m6);
   mechanism.setForce(fl);
   mechanism.setNumPoints(numpoints);
   mechanism.plotForce(MAG_F12 | F16Y, &plot);

   return 0;
}
```

Output





CQuickReturn::plotSliderAccel

Synopsis

#include <quickreturn.h>

void plotSliderAccel(**CPlot** **plot*);

Purpose

Plots the acceleration of the output slider over time.

Return Value

None

Parameters

&plot pointer to the plot object declared in calling program for displaying output.

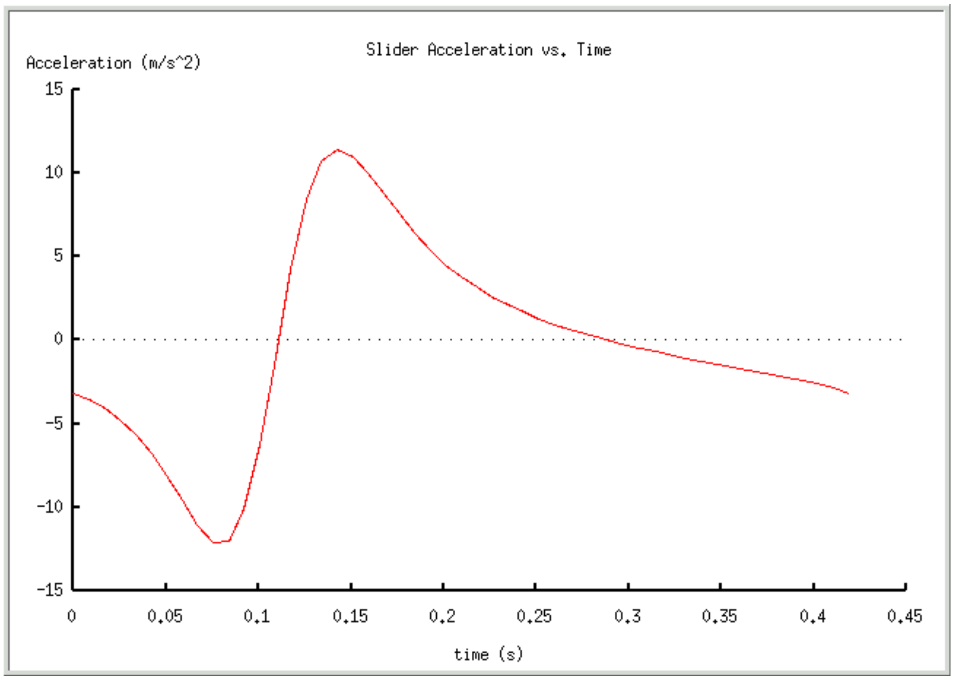
Description

Calculates the acceleration of the output silder as the angle of link 2 changes with time. It then creates a plot of the acceleration of the output slider over time.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                    //rad
   double theta2 = 0.0;                                                       //rad
   double omega2 = -15.0;                                                     //rad/sec
   int numpoints = 360;
   bool unit = SI;

   /* Create CQuickReturn and Plot Objects */
   CQuickReturn mechanism;
   CPlot plot;

   mechanism.uscUnit(unit);
   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.plotSliderAccel(&plot);

   return 0;
}
```

Output



CQuickReturn::plotSliderPos

Synopsis

#include <quickreturn.h>
void plotSliderPos(CPlot **plot*);

Purpose

Plots the Position of the output slider over time.

Return Value

None

Parameters

&plot pointer to the plot object declared in calling program for displaying output.

Description

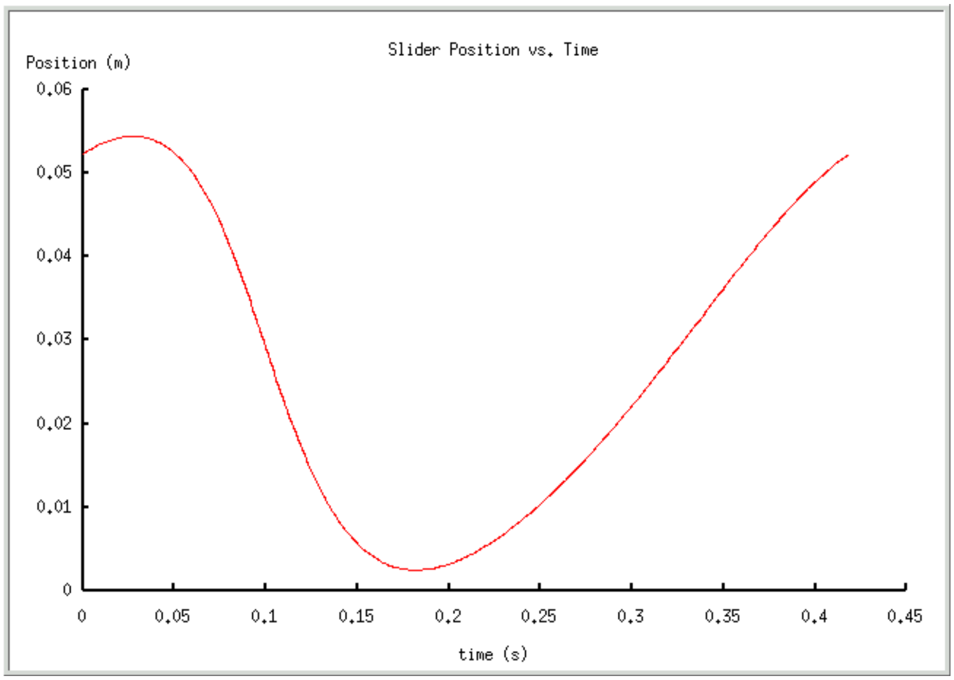
Calculates the position of the output silder as the angle of link 2 changes with time. It then creates a plot of the position of the output slider over time.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                    //rad
   double theta2 = 0.0;                                                       //rad
   int numpoints = 360;
```

```
   bool unit = SI;

   /* Create CQuickReturn and Plot Objects */
   CQuickReturn mechanism;
   CPlot plot;

   mechanism.uscUnit(unit);
   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setNumPoints(numpoints);
   mechanism.plotSliderPos(&plot);

   return 0;
}
```

Output



CQuickReturn::plotSliderVel

Synopsis

#include <quickreturn.h>
void plotSliderVel(CPlot **plot*);

Purpose

Plots the velocity of the output slider over time.

Return Value

None

Parameters

&plot pointer to the plot object declared in calling program for displaying output.

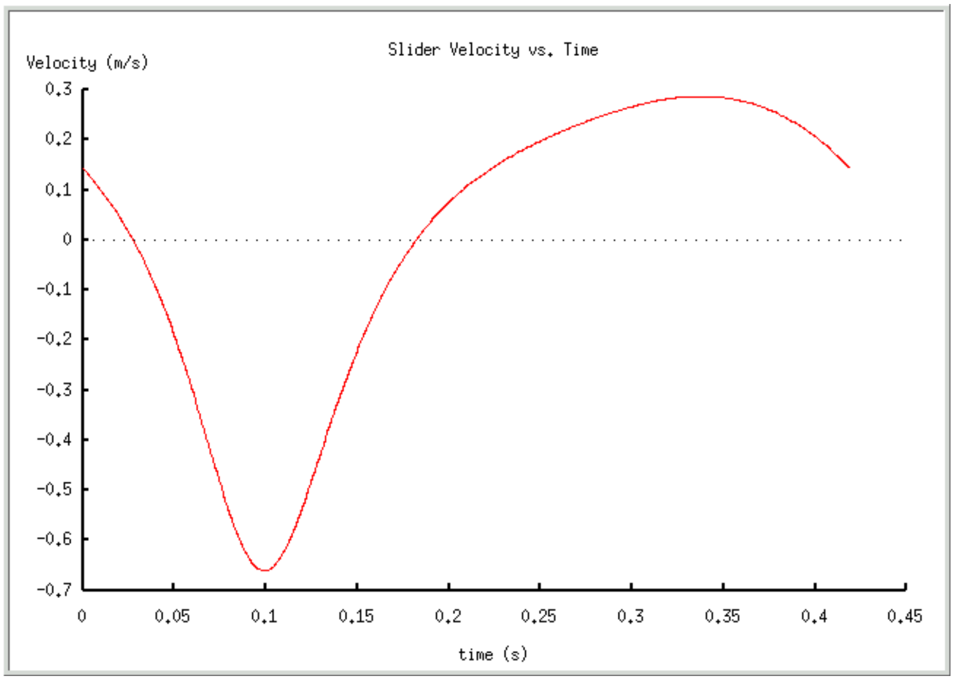
Description

Calculates the velocity of the output silder as the angle of link 2 changes with time. It then creates a plot of the velocity of the output slider over time.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;             //meters
   double theta1 = M_PI/2;                                                    //rad
   double theta2 = 0.0;                                                       //rad
   double omega2 = -15.0;                                                     //rad/sec
   int numpoints = 360;
   bool unit = SI;

   /* Create CQuickReturn and Plot Objects */
   CQuickReturn mechanism;
   CPlot plot;

   mechanism.uscUnit(unit);
   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.setNumPoints(numpoints);
   mechanism.plotSliderVel(&plot);

   return 0;
}
```

Output



CQuickReturn::plotTorque

Synopsis
#include <quickreturn.h>
void plotTorque(**CPlot** **plot*);

Purpose
Plots the required input torque over time.

Return Value
None

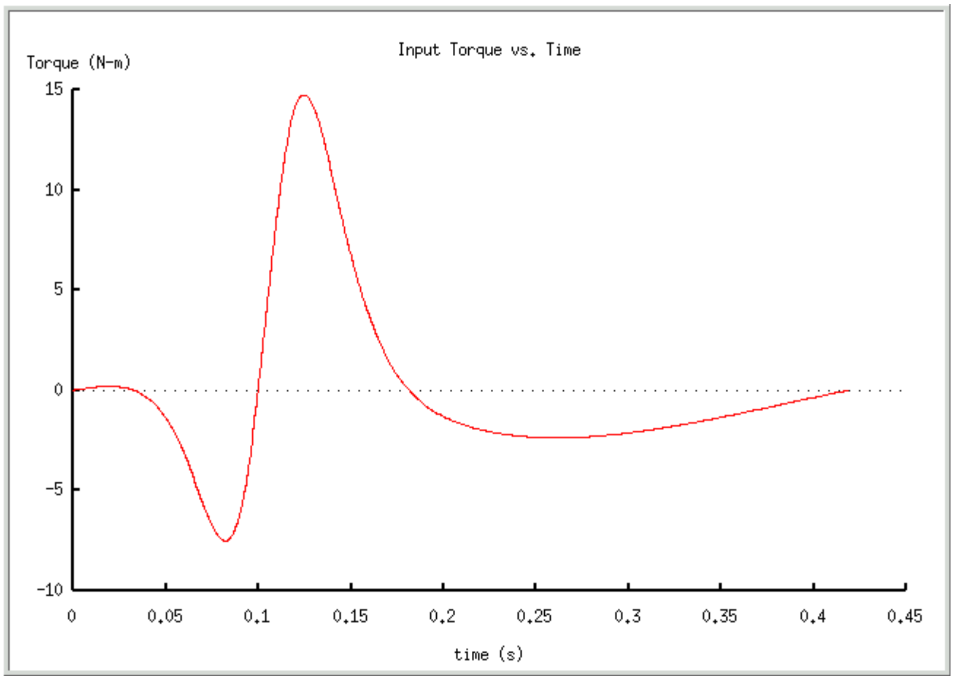
Parameters
&plot pointer to the plot object declared in calling program for displaying output.

Description
Calculates the required input torque for link 2 as the angle of link2 changes over time. It then creates a plot of the torque over time.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                    //rad
   double theta2 = 0.0;                                                       //rad
   double omega2 = -15.0;                                                     //rad/sec
   double rg2 = 0.0125, rg4 = 0.0275, rg5 = 0.0250;                           //meters
   double delta2 = 30*M_PI/180, delta4 = 15*M_PI/180, delta5 = 30*M_PI/180;  //rad
   double ig2 = 0.012, ig4 = 0.119, ig5 = 0.038;                              //kg*mˆ2
   double m2 = 0.8, m3 = 0.3, m4 = 2.4, m5 = 1.4, m6 = 0.3;                  //kg
   double fl = -100;                                                          //N
   int numpoints = 360;
   bool unit = SI;

   /* Create CQuickReturn and Plot Objects */
   CQuickReturn mechanism;
   CPlot plot;

   mechanism.uscUnit(unit);
   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.setGravityCenter(rg2, rg4, rg5, delta2, delta4, delta5);
   mechanism.setInertia(ig2, ig4, ig5);
   mechanism.setMass(m2, m3, m4, m5, m6);
   mechanism.setForce(fl);
   mechanism.setNumPoints(numpoints);
   mechanism.plotTorque(&plot);

   return 0;
}
```

Output



CQuickReturn::setAngVel

Synopsis
#include <**quickreturn.h**>
void setAngVel(**double** *omega2*);

Purpose
Sets the angular velocity of link 2.

Return Value
None

Parameters
omega2 The angular velocity of link 2.

Description
Sets the angular velocity of link 2. This is used in velocity calculations.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double omega2 = -15.0;                                              //rad/sec

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;
```

```
   mechanism.setAngVel(omega2);

   return 0;
}
```

Output None

CQuickReturn::**setForce**

Synopsis

#include <**quickreturn.h**>

void setForce(**double** *fl*);

Purpose

Sets the load force.

Return Value

None

Parameters

fl	The load force on the mechanism.

Description

Sets the load force that acts on the output slider.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double fl = -100;                                                  //N

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setForce(fl);


   return 0;
}
```

Output None

CQuickReturn::**setGravityCenter**

Synopsis

#include <**quickreturn.h**>

void setGravityCenter(**double** *rg2*, **double** *rg4*,**double** *rg5*,**double** *delta2*,**double** *delta4*,**double** *delta5*);

Purpose

Sets the gravity center parameters of all of the links.

Return Value
None

Parameters

rg2	Length of the CG phasor of link 2.
rg4	Length of the CG phasor of link 4.
rg5	Length of the CG phasor of link 5.
delta2	Angle of the CG phasor of link 2.
delta4	Angle of the CG phasor of link 4.
delta5	Angle of the CG phasor of link 5.

Description
Sets the gravity center parameters of all of the links. These are used for force calculations.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double rg2 = 0.0125, rg4 = 0.0275, rg5 = 0.0250;                          //meters
   double delta2 = 30*M_PI/180, delta4 = 15*M_PI/180, delta5 = 30*M_PI/180;  //rad

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setGravityCenter(rg2, rg4, rg5, delta2, delta4, delta5);

   return 0;
}
```

Output None

CQuickReturn::setInertia

Synopsis
#include <quickreturn.h>
void setInertia(double *ig2*, **double** *ig4*, **double** *ig5*);

Purpose
Sets the inertia properties of all of the links.

Return Value
None

Parameters

ig2	The moment of inertia of link 2.
ig4	The moment of inertia of link 4.
ig5	The moment of inertia of link 5.

Description
Sets the inertia properties of all of the links. These are used in the force calculations.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double ig2 = 0.012, ig4 = 0.119, ig5 = 0.038;                          //kg*m^2

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setInertia(ig2, ig4, ig5);

   return 0;
}
```

Output None

CQuickReturn::setLinks

Synopsis

#include <quickreturn.h>
void setLinks(double *r1*, **double** *r2*,**double** *r4*,**double** *r5*,**double** *r7*,**double** *theta1*);

Purpose

Sets the length of the links and the phase angle of the ground link.

Return Value

None

Parameters

r1	The length of gound.
r2	The length of link 2.
r4	The length of link 4.
r5	The length of link 5.
r7	The vertical height of the output slider with respect to the lowest groundpin.
theta1	The phase angle of the ground phasor.

Description

Sets the length of the links and the phase angle of the ground link. These are used for all of the other calculations.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad

   /* Create CQuickReturn Object */
```

```
   CQuickReturn mechanism;

   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);

   return 0;
}
```

Output None

CQuickReturn::**setMass**

Synopsis
#include <quickreturn.h>
void setMass(**double** *m2*, **double** *m3*, **double** *m4*, **double** *m5*, **double** *m6*);

Purpose
Sets the mass parameters of all of the links.

Return Value
None

Parameters

m2	The mass of link 2.
m3	The mass of link 3.
m4	The mass of link 4.
m5	The mass of link 5.
m6	The mass of link 6.

Description
Sets the mass parameters of all of the links. This is mainly used to calculate the forces acting on the mechanism.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double m2 = 0.8, m3 = 0.3, m4 = 2.4, m5 = 1.4, m6 = 0.3;                  //kg

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setMass(m2, m3, m4, m5, m6);

   return 0;
}
```

Output None

CQuickReturn::**setNumPoints**

Synopsis

#include <quickreturn.h>
void setNumPoints(double *numpoints*);

Purpose

Set the number of points.

Return Value

None

Parameters

numpoints The number of points to use when plotting and creating the animation file.

Description

Set the number of points used when creating plots and creating the animation file.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   int numpoints = 360;

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setNumPoints(numpoints);

   return 0;
}
```

Output None

CQuickReturn::sliderAccel

Synopsis

#include <quickreturn.h>
double sliderAccel(double *theta2*);

Purpose

Calculates the acceleration of the output slider.

Return Value

Returns the accerleration of the output slider.

Parameters

theta2 The angle of link 2 used to calculate the acceleration the output slider.

Description

Calculates the acceleration of the output slider and returns the calculated value.

Example

```
#include <stdio.h>
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   double omega2 = -15.0;                                                    //rad/sec
   bool unit = SI;
   double slideraccel = 0;

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.setAngVel(omega2);
   mechanism.uscUnit(unit);
   slideraccel = mechanism.sliderAccel(unit);

   printf("The acceleration of the output slider is %f\n", slideraccel);

   return 0;
}
```

Output

```
The acceleration of the output slider is -3.190996
```

CQuickReturn::sliderPos

Synopsis
#include <quickreturn.h>
double sliderPos(double *theta2*);

Purpose
Calculates the position of the output slider.

Return Value
The calculated position of the output slider

Parameters
theta2 The angle of link 2 used to calculate the position of the output slider.

Description
Calculates the position of the output slider and returns the calculated value.

Example

```
#include <stdio.h>
#include <quickreturn.h>

int main(void)
{
```

```
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   bool unit = SI;
   double sliderpos = 0;

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.uscUnit(unit);
   sliderpos = mechanism.sliderPos(theta2);

   printf("The position of the output slider is %f\n", sliderpos);

   return 0;
}
```

Output

```
The position of the output slider is 0.052298
```

CQuickReturn::sliderRange

Synopsis

#include <quickreturn.h>
void sliderRange(**double** *&max*, **double** *&min*);

Purpose

Calculates the range of the output slider.

Return Value

None

Parameters

max A variable passed by reference used to store the maximum attained position of the output slider.
min A variable passed by reference used to store the minimum attained position of the output slider.

Description

Calculates the maximum and minimum position of the output slider and saves them into the variabled passed by reference.

Example

```
#include <stdio.h>
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   bool unit = SI;
```

```
   double max = 0, min = 0;

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
   mechanism.uscUnit(unit);
   mechanism.sliderRange(max, min);

   printf("The range of the output slider is from %f to %f\n", min, max);

   return 0;
}
```

Output

```
The range of the output slider is from 0.002444 to 0.054414
```

CQuickReturn::sliderVel

Synopsis

#include <quickreturn.h>
double sliderVel(**double** *theta2*);

Purpose

Calculates the velocity of the output slider.

Return Value

Returns the velocity of the output slider.

Parameters

theta2 The angle of link 2 used to calculate the acceleration the output slider.

Description

Calculates the velocity of the output slider and returns the calculated value.

Example

```
#include <stdio.h>
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   double r1 = .025, r2 = .010, r4 = .065, r5 = .030, r7 = .050;            //meters
   double theta1 = M_PI/2;                                                   //rad
   double theta2 = 0.0;                                                      //rad
   double omega2 = -15.0;                                                    //rad/sec
   bool unit = SI;
   double slidervel = 0;

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.setLinks(r1, r2, r4, r5, r7, theta1);
```

```
   mechanism.setAngVel(omega2);
   mechanism.uscUnit(unit);
   slidervel = mechanism.sliderVel(theta2);

   printf("The velocity of the output slider is %f\n", slidervel);

   return 0;
}
```

Output

```
The velocity of the output slider is 0.143224
```

CQuickReturn::uscUnit

Synopsis

#include <quickreturn.h>
void uscUnit(bool *unit*);

Purpose

Determines what units are used.

Return Value

None

Parameters

unit An enumerated value of either USC or SI.

Description

Determines what units are used. This allows users to input in either SI or USC units. Any output after this function has been used to changed the units will reflect the new units chosen.

Example

```
#include <quickreturn.h>

int main(void)
{
   /* Set up required parameters */
   bool unit = SI;

   /* Create CQuickReturn Object */
   CQuickReturn mechanism;

   mechanism.uscUnit(unit);

   return 0;
}
```

Output

```
!! You’re using SI UNITS !!
```

9 Appendix B: Source Code

9.1 Source Code of Classes and Functions

quickreturn.h

```
#ifndef _QUICKRETURN_H_
#define _QUICKRETURN_H_

#include <linkage.h>
#include <stdio.h>
#include <stdbool.h>
#include <stdarg.h>
#include <float.h>
#include <complex.h>
#include <chplot.h>

enum
{
   SI,             //  SI units, use with uscUnit()
   USC,            //  USC units,  "    "    "
   QR_LINK_2,      //  Link 2
   QR_LINK_4,      //  Link 4
   QR_LINK_5,      //  Link 5
   QR_POINT_A,     //  Point A: point of slider 3
   QR_POINT_B,     //  Point B: end of Link 4
   QR_LINK_2_CG,   //  Center of Mass of Link 2
   QR_LINK_4_CG,   //  Center of Mass of Link 4
   QR_LINK_5_CG    //  Center of Mass of Link 5
};

enum      //  Pick which plots to output for plotForce()
{
   F12X = 1,
   F12Y = 2,
   MAG_F12 = 4,
   ALL_F12 = 7,
   F23X = 8,
   F23Y = 16,
   MAG_F23 = 32,
   ALL_F23 = 56,
   F14X = 64,
   F14Y = 128,
   MAG_F14 = 256,
   ALL_F14 = 448,
   MAG_F34 = 512,
   F45X = 1024,
   F45Y = 2048,
   MAG_F45 = 4096,
   ALL_F45 = 7168,
   F56X = 8192,
   F56Y = 16384,
```

```
   MAG_F56 = 32768,
   ALL_F56 = 57344,
   F16Y = 65536,
   ALL_MAG_PLOTS = 103204,
   ALL_FORCE_PLOTS = 131071
};

/*********************************************
 * CQuickReturn class definition
 ********************************************/

class CQuickReturn
{
   private:
      // Private data members
      double m_delta[1:5];     // phase angle of CG of links
      double m_inertia[1:5];   // inertia of the links
      double m_mass[1:6];      // mass of the links
      double m_omega[1:5];     // angular velocity
      double m_alpha[1:5];     // angular acceleration
      double m_r[1:8];         // lengths of links
      double m_rg[1:5];        // distance to CG of links
      double m_theta[1:7];     // phase angles for the links
      double m_r3_dot;         // first derivative of r7
      double m_r6_dot;         // first derivative of r9
      double m_r3_double_dot;  // second derivative of r7
      double m_r6_double_dot;  // second derivative of r9
      double m_load;           // external force or return slider
      double complex m_v3;     // velocity of slider 3
      double complex m_a3;     // acceleration of slider 3
      double complex m_ag2;    // acceleration of mass center of link 2
      double complex m_ag4;    // acceleration of mass center of link 4
      double complex m_ag5;    // acceleration of mass center of link 5
      int m_numpoints;         // number of points to plot or for animation
      bool m_uscunit;          // unit choice

      // Private function members
      void m_initialize(void);              // initialize private members
      void calcPosition(double theta2);     // calc. ang. pos. and m_r9
      void calcVelocity(double theta2);     // calc. ang. vel. and r9 dot
      void calcAcceleration(double theta2); // calc. ang. accel. and r9 ddot
      void calcForce(double theta2, array double x[13]); // calc. forces

   public:
      // Constructor and Destructor
      CQuickReturn();
      ˜CQuickReturn();

      // Setting information functions
      void setLinks(double r1, r2, r4, r5, r7, theta1);
      void setAngVel(double initomega2);
      void setGravityCenter(double rg2, rg4, rg5, delta2, delta4, delta5);
      void setInertia(double ig2, ig4, ig5);
      void setMass(double m2, m3, m4, m5, m6);
      void setForce(double fl);
      void setNumPoints(int numpoints);
      void uscUnit(bool unit);

      // Output information functions
```

```
      double sliderPos(double theta2);
      double sliderVel(double theta2);
      double sliderAccel(double theta2);
      double getRequiredTorque(double theta2);
      void sliderRange(double& max, double& min);

      // Output display functions
      void displayPosition(double theta2, ...);
      void animation(...);

      // Plot information functions
      void plotSliderPos(CPlot *plot);
      void plotSliderVel(CPlot *plot);
      void plotSliderAccel(CPlot *plot);
      void plotAngPos(CPlot *plot);
      void plotAngVel(CPlot *plot);
      void plotAngAccel(CPlot *plot);
      void plotCGaccel(CPlot *plot);
      void plotForce(int plot_output, CPlot *plot);
      void plotTorque(CPlot *plot);

      // Information for specified point or link
      double getAngPos(double theta2, int link);
      double getAngVel(double theta2, int link);
      double getAngAccel(double theta2, int link);
      double complex getPointPos(double theta2, int point);
      double complex getPointVel(double theta2, int point);
      double complex getPointAccel(double theta2, int point);
      void getForces(double theta2, array double y[12]);
};

#pragma importf <CQuickReturn.chf>

#endif
```

CQuickReturn.chf

```
/***********************************************************
 * File name: CQuickReturn.chf
 *            member functions of class CQuickReturn
 ***********************************************************/
#include <quickreturn.h>

void CQuickReturn::m_initialize(void)
{

   /* defaults */
   m_r[1]  = 0.025;            // length of link 1
   m_r[2]  = 0.010;            // length of link 2
   m_r[4]  = 0.065;            // length of link 4
   m_r[5]  = 0.030;            // length of link 5
   m_r[7]  = 0.050;            // length of link 10

   m_theta[1]  = M_PI/2;       // angle of link 1
   m_theta[6]  = 0;            // angle of link 9
   m_theta[7]  = M_PI/2;       // angle of link 10

   m_omega[2] = -15;           // input angular velocity

   m_mass[2] = 0.8;            // mass of link 2 in kg
   m_mass[3] = 0.3;            // mass of link 3 in kg
   m_mass[4] = 2.4;            // mass of link 4 in kg
   m_mass[5] = 1.4;            // mass of link 5 in kg
   m_mass[6] = 0.3;            // mass of link 6 in kg

   m_inertia[2] = 0.012;       // mass inertia of link 2
   m_inertia[4] = 0.119;       // mass inertia of link 4
   m_inertia[5] = 0.038;       // mass inertia of link 5

   m_rg[2] = 0.0125;           // CG distance of link 2 in m
   m_rg[4] = 0.0275;           // CG distance of link 4 in m
   m_rg[5] = 0.0250;           // CG distance of link 5 in m

   m_delta[2] = 30*M_PI_180;   // phase angle for CG of link 2
   m_delta[4] = 15*M_PI_180;   // phase angle for CG of link 4
   m_delta[5] = 30*M_PI_180;   // phase angle for CG of link 5

   m_load = -100;              // exeternal load on return slider

   m_numpoints = 50;           // number of points for animation

   m_uscunit = 0;              // selection SI units
}

/*******************************************
 * Constructor of class CQuickReturn
 ******************************************/
CQuickReturn::CQuickReturn()
{
   m_initialize();
}
```

```
/******************************************
 * Destructor of class CQuickReturn
 ******************************************/
CQuickReturn::~CQuickReturn()
{}

/*******************************************************************
 * setAngVel()
 *
 * Set angular velocity.
 *
 * Arguments: omega2 ... angular velocity of input link
 *******************************************************************/
void CQuickReturn::setAngVel(double omega2)
{
   m_omega[2] = omega2;
}

/*********************************************************************
 * setGravityCenter()
 *
 * Set the distances and offset angles for center of mass
 * for each link.
 *
 * Arguments: rg2 ... distance to center of mass for link 2
 *            rg4 ...    "     "    "    "    "   "    "  4
 *            rg5 ...    "     "    "    "    "   "    "  5
 *            delta2 ... anglular offset of center of mass for link 2
 *            delta4 ...    "        "   "     "   "    "   "    "  4
 *            delta5 ...    "        "   "     "   "    "   "    "  5
 *********************************************************************/
void CQuickReturn::setGravityCenter(double rg2, rg4, rg5, delta2, delta4, delta5)
{

   m_rg[2] = rg2;      // CG distance of link 2 in m
   m_rg[4] = rg4;       // CG distance of link 4 in m
   m_rg[5] = rg5;       // CG distance of link 5 in m

   m_delta[2] = delta2;      // phase angle for CG of link 2
   m_delta[4] = delta4;      // phase angle for CG of link 4
   m_delta[5] = delta5;      // phase angle for CG of link 5

   if(m_uscunit)
   {
      m_rg[2] *= M_FT2M;     // ft --> m
      m_rg[4] *= M_FT2M;     // ft --> m
      m_rg[5] *= M_FT2M;     // ft --> m
   }
}

/***************************************************
 * setInertia()
 *
 * Set the Moment of Inertia for each link.
 *
 * Arguments: ig2 ... Moment of Inertia for link 2
 *            ig4 ...    "   "     "     "    "  4
 *            ig5 ...    "   "     "     "    "  5
 ***************************************************/
```

```
void  CQuickReturn::setInertia(double ig2, ig4, ig5)
{
   m_inertia[2] = ig2;
   m_inertia[4] = ig4;
   m_inertia[5] = ig5;

   if(m_uscunit)
   {
      m_inertia[2] *= M_LBFTSS2KGMM;     // lb-ft-sˆ2 --> kg-mˆ2
      m_inertia[4] *= M_LBFTSS2KGMM;     // lb-ft-sˆ2 --> kg-mˆ2
      m_inertia[5] *= M_LBFTSS2KGMM;     // lb-ft-sˆ2 --> kg-mˆ2
   }
}

/****************************************************************
 * setLinks()
 *
 * Set the Length for each link and angle between ground pins.
 *
 * Arguments: r1  ... length Link 1
 *            r2  ...    "     "  2
 *            r4  ...    "     "  4
 *            r5  ...    "     "  5
 *            r7  ...    "     "  7
 *            theta 1 ... angle of Link 1
 ****************************************************************/
void CQuickReturn::setLinks(double r1, r2, r4, r5, r7, theta1)
{
   int characteristic;
   // Check geometry input for values that will make the
   // mechanism not work
   if((int)(1000000*r2) >= (int)(1000000*r1))
   {
      printf("r2 must be less than r1.\n"
             "Try resetting the geometry and trying agian.\n\n");
      exit(1);
   }
   if((int)(1000000*r4) < (int)(1000000*r1) + (int)(1000000*r2))
   {
      printf("r4 must be greater than r1 + r2.\n"
             "Try resetting the geometry and trying agian.\n\n");
      exit(1);
   }
   if((int)(1000000*r4) > (int)(1000000*r5) + (int)(1000000*abs(r7)))
   {
      printf("r4 must be less than r5 + r7.\n"
             "Try resetting the geometry and trying agian.\n\n");
      exit(1);
   }

   // If geometry valid assign parameters
   m_r[1] = r1;
   m_r[2] = r2;
   m_r[4] = r4;
   m_r[5] = r5;
   m_r[7] = r7;
   m_theta[1] = theta1;

   if(m_uscunit)
```

```
   {
      m_r[1] *= M_FT2M;     // ft --> m
      m_r[2] *= M_FT2M;     // ft --> m
      m_r[4] *= M_FT2M;     // ft --> m
      m_r[5] *= M_FT2M;     // ft --> m
      m_r[7] *= M_FT2M;     // ft --> m
   }
}

/*******************************************
 * setMass()
 *
 * Set the Mass for each link.
 *
 * Arguments: m2 ... mass of Link 2
 *            m3 ...   "  "    "  3
 *            m4 ...   "  "    "  4
 *            m5 ...   "  "    "  5
 *            m6 ...   "  "    "  6
 ******************************************/
void CQuickReturn::setMass(double m2, m3, m4, m5, m6)
{
   m_mass[2] = m2;
   m_mass[3] = m3;
   m_mass[4] = m4;
   m_mass[5] = m5;
   m_mass[6] = m6;

   if(m_uscunit)
   {
      m_mass[2] *= M_SLUG2KG;     // slug --> kg
      m_mass[3] *= M_SLUG2KG;     // slug --> kg
      m_mass[4] *= M_SLUG2KG;     // slug --> kg
      m_mass[5] *= M_SLUG2KG;     // slug --> kg
      m_mass[6] *= M_SLUG2KG;     // slug --> kg
   }
}

/**********************************************
 * setForce()
 *
 * Set the external force on return slider.
 *
 * Arguments: fl ... exteral load on slider
 *********************************************/
void CQuickReturn::setForce(double fl)
{
   m_load = fl;

   if(m_uscunit)
   {
      m_load *= M_LB2N;    // lb --> N
   }
}

/******************************************************
 * setNumPoints()
 *
 * Set the number of points for calcs & animation.
```

```
 *
 * Arguments: numpoints ... number of points used
 *****************************************************/
void CQuickReturn::setNumPoints(int numpoints)
{
   m_numpoints = numpoints;
}

/*******************************************
 * uscUnit()
 *
 * Set the units preference.
 *
 * Arguments: unit ... true for USC units
 *                     false for SI units
 ******************************************/
void CQuickReturn::uscUnit(bool unit)
{
   m_uscunit = unit;
   if(m_uscunit)
      printf("\n!! You’re using ENGLISH UNITS !!\n\n");
   else
      printf("\n!! You’re using SI UNITS !!\n\n");
}

/**************************************************************
 * calcPosition()
 *
 * Computes the angular positions &
 * unknown lengths for a given angle of the driving link
 * of the quick return mechanism.
 *
 * Arguments: theta2 ... drive link positions
 *************************************************************/
void CQuickReturn::calcPosition(double theta2)
{

   int n1, n2;
   double temp1, temp2;
   double complex z;

   m_theta[2] = theta2;
   // Solve First Loop: r1 + r2 = r3 -> r3 - r2 = r1
   // z = r1
   n1 = 1;
   n2 = 2;
   z = polar(m_r[1], m_theta[1]);
   // find r3, theta4
   complexsolve(n1, n2, -m_r[2], m_theta[2], z, m_r[3], m_theta[4], temp1, temp2);

   // Solve Second Loop: r4 + r5 = r6 + r7 ->  r6 - r5 = r4 - r7
   // z = r4 - r7
   n1 = 1;
   n2 = 4;
   z = polar(m_r[4], m_theta[4]) - polar(m_r[7], m_theta[7]);
   // find r6, theta5
   complexsolve(n1, n2, m_theta[6], -m_r[5], z, m_r[6], m_theta[5], temp1, temp2);

   // Solve r8: r3 + r8 = r4 -> r8 = r4 - r3
```

```
   m_r[8] = m_r[4] - m_r[3];
}

/************************************************
 * calcVelocity()
 *
 * Computes the velocities for
 * the quick return mechanism.
 *
 * Arguments: theta2 ... drive link positions
 ************************************************/
void CQuickReturn::calcVelocity(double theta2)
{
   calcPosition(theta2);

   // omega4
   m_omega[4] = (m_r[2]/m_r[3])*(cos(m_theta[2] - m_theta[4]))*m_omega[2];

   // omega5
   m_omega[5] = -(m_r[4]/m_r[5])*(cos(m_theta[4])/cos(m_theta[5]))*m_omega[4];

   // velocity r7
   m_r3_dot = m_r[2]*m_omega[2]*(sin(m_theta[4]-m_theta[2]));

   // velocity r9
   m_r6_dot = (m_r[4]*m_omega[4]*(sin(m_theta[5]-m_theta[4])))/(cos(m_theta[5]));

   // Velocity of slider 3
   m_v3 = I*polar((m_r[3]*m_omega[4]), m_theta[4]) + polar(m_r3_dot, m_theta[4]);
}

/****************************************************************
 * calcAcceleration()
 *
 * Computes the accelerations for
 * the quick return mechanism.
 *
 * Arguments: theta2 ... drive link positions
 ****************************************************************/
void CQuickReturn::calcAcceleration(double theta2)
{
   double a,b,c,d;
   double complex w,x,y,z;

   calcVelocity(theta2);

   // alpha 4
   a = -m_r[2]*m_omega[2]*m_omega[2]*(sin(m_theta[2] - m_theta[4]));
   b = m_r[2]*m_alpha[2]*(cos(m_theta[2] - m_theta[4]));
   c = -2*m_r3_dot*m_omega[4];
   d = m_r[3];
   m_alpha[4] = (a + b + c)/d;

   // alpha 5
   a = m_r[4]*m_omega[4]*m_omega[4]*(sin(m_theta[4]));
   b = -m_r[4]*m_alpha[4]*(cos(m_theta[4]));
   c = m_r[5]*m_omega[5]*m_omega[5]*(sin(m_theta[5]));
   d = m_r[5]*(cos(m_theta[5]));
   m_alpha[5] = (a + b + c)/d;
```

```
   // acceleration r3
   a = m_r[3]*m_omega[4]*m_omega[4];
   b = -m_r[2]*m_omega[2]*m_omega[2]*(cos(m_theta[2] - m_theta[4]));
   c = -m_r[2]*m_alpha[2]*(sin(m_theta[2] - m_theta[4]));
   m_r3_double_dot = a + b + c;

   // acceleration r6
   a = -m_r[4]*m_omega[4]*m_omega[4]*(cos(m_theta[4]));
   b = -m_r[4]*m_alpha[4]*(sin(m_theta[4]));
   c = -m_r[5]*m_omega[5]*m_omega[5]*(cos(m_theta[5]));
   d = -m_r[5]*m_alpha[5]*(sin(m_theta[5]));
   m_r6_double_dot = a + b + c + d;

   // Acceleration of slider 3
   w = I*polar((m_r[3]*m_alpha[4]), m_theta[4]);
   x = -polar((m_r[3]*m_omega[4]*m_omega[4]), m_theta[4]);
   y = polar(m_r3_double_dot, m_theta[4]);
   z = I*polar((2*m_r3_dot*m_omega[4]), m_theta[4]);
   m_a3 = w + x + y + z;

   // Acceleration of the Centers of Mass
   x = I*polar((m_rg[2]*m_alpha[2]), (m_theta[2] + m_delta[2]));
   y = polar((m_rg[2]*m_omega[2]*m_omega[2]), (m_theta[2] + m_delta[2]));
   m_ag2 = x - y;

   x = I*polar((m_rg[4]*m_alpha[4]), (m_theta[4] + m_delta[4]));
   y = polar((m_rg[4]*m_omega[4]*m_omega[4]), (m_theta[4] + m_delta[4]));
   m_ag4 = x - y;

   w = I*polar((m_r[4]*m_alpha[4]), m_theta[4]);
   x = polar((m_r[4]*m_omega[4]*m_omega[4]), m_theta[4]);
   y = I*polar((m_rg[5]*m_alpha[5]), (m_theta[5] + m_delta[5]));
   z = polar((m_rg[5]*m_omega[5]*m_omega[5]), (m_theta[5] + m_delta[5]));
   m_ag5 = w - x + y - z;
}

/*******************************************************************************
 * calcForce()
 *
 * Calculate force on links of quick return mechanism at a specific
 * position of the input link.
 *
 * Arguments: theta2 ... drive link positions
 *  x[13] = f12x,f12y,f23x,f23y,f14x,f14y,f34,f45x,f45y,f56x,f56y,f16y,Ts
 *******************************************************************************/
void CQuickReturn::calcForce(double theta2, array double x[13])
{
   array double B[13], A[13][13] = {0};
   double fl = m_load;
   double g = 9.81;     // m/s^2
   double phi;
   double fg2x, fg2y, fg3x, fg3y, fg4x, fg4y, fg5x, fg5y, fg6x, tg2, tg4, tg5;

   if(m_uscunit)
      fl *= M_LBFT2NM;     // lb-ft --> N-m

   calcAcceleration(theta2);
```

```
phi = m_theta[4] - (M_PI/2);

fg2x = -m_mass[2]*real(m_ag2);
fg2y = -m_mass[2]*imag(m_ag2);
tg2 = -m_inertia[2]*m_alpha[2];

fg3x = -m_mass[3]*real(m_a3);
fg3y = -m_mass[3]*imag(m_a3);

fg4x = -m_mass[4]*real(m_ag4);
fg4y = -m_mass[4]*imag(m_ag4);
tg4 = -m_inertia[4]*m_alpha[4];

fg5x = -m_mass[5]*real(m_ag5);
fg5y = -m_mass[5]*imag(m_ag5);
tg5 = -m_inertia[5]*m_alpha[5];

fg6x = -m_mass[6]*m_r6_double_dot;

B[0] = fg2x;
B[1] = fg2y - m_mass[2]*g;
B[2] = tg2;
B[3] = fg3x;
B[4] = fg3y - m_mass[3]*g;
B[5] = fg4x;
B[6] = fg4y - m_mass[4]*g;
B[7] = tg4;
B[8] = fg5x;
B[9] = fg5y - m_mass[5]*g;
B[10] = tg5;
B[11] = fg6x + fl;
B[12] = -m_mass[6]*g;

A[0][0]   = -1;
A[0][2]   = 1;
A[1][1]   = -1;
A[1][3]   = 1;
A[2][0]   = -m_rg[2]*sin(m_theta[2] + m_delta[2]);
A[2][1]   = m_rg[2]*cos(m_theta[2] + m_delta[2]);
A[2][2]   = -(m_r[2]*sin(m_theta[2]) - m_rg[2]*sin(m_theta[2] + m_delta[2]));
A[2][3]   = m_r[2]*cos(m_theta[2]) - m_rg[2]*cos(m_theta[2] + m_delta[2]);
A[2][12]  = -1;
A[3][2]   = -1;
A[3][6]   = cos(phi);
A[4][3]   = -1;
A[4][6]   = sin(phi);
A[5][4]   = -1;
A[5][6]   = -cos(phi);
A[5][8]   = 1;
A[6][5]   = -1;
A[6][6]   = -sin(phi);
A[6][9]   = 1;
A[7][4]   = -m_rg[4]*sin(m_theta[4] + m_delta[4]);
A[7][5]   = m_rg[4]*cos(m_theta[4] + m_delta[4]);
A[7][6]   = ((m_r[3]*sin(m_theta[4]) - m_rg[4]*sin(m_theta[4] + m_delta[4]))
            *(cos(phi)))
            - ((m_r[3]*cos(m_theta[4]) - m_rg[4]*cos(m_theta[4] + m_delta[4]))
            *(sin(phi)));
A[7][7]   = -( m_r[4]*sin(m_theta[4]) - m_rg[4]*sin(m_theta[4] + m_delta[4]));
```

```
   A[7][8]   = m_r[4]*cos(m_theta[4]) - m_rg[4]*cos(m_theta[4] + m_delta[4]);
   A[8][7]   = -1;
   A[8][9]   = 1;
   A[9][8]   = -1;
   A[9][10]  = 1;
   A[10][7]  = -m_rg[5]*sin(m_theta[5] + m_delta[5]);
   A[10][8]  = m_rg[5]*cos(m_theta[5] + m_delta[5]);
   A[10][9]  = -(m_r[5]*sin(m_theta[5]) - m_rg[5]*sin(m_theta[5] + m_delta[5]));
   A[10][10] = m_r[5]*cos(m_theta[5]) - m_rg[5]*cos(m_theta[5] + m_delta[5]);
   A[11][9]  = -1;
   A[12][10] = -1;
   A[12][11] = -1;

   x = inverse(A)*B;
}

/************************************************
 * sliderPos()
 *
 * Return postion of slider for given theta2.
 *
 * Arguments: theta2 ... drive link positions
 *
 * Return value: slider position
 ************************************************/
double CQuickReturn::sliderPos(double theta2)
{
   calcPosition(theta2);

   if(m_uscunit)
   {
      return (m_r[6] /= M_FT2M); // m --> ft
   }
   else
   {
      return m_r[6];
   }
}

/*****************************************************
 * sliderVel()
 *
 * Return Velocity of slider for given theta2.
 *
 * Arguments: theta2 ... drive link positions
 *
 * Return value: slider velocity
 *****************************************************/
double CQuickReturn::sliderVel(double theta2)
{
   calcVelocity(theta2);

   if(m_uscunit)
   {
      return m_r6_dot /= M_FT2M; // m --> ft
   }
   else
   {
      return m_r6_dot;
```

```
   }
}

/**********************************************************
 * sliderAccel()
 *
 * Return Acceleration of slider for given theta2.
 *
 * Arguments: theta2 ... drive link positions
 *
 * Return value: slider acceleration
 **********************************************************/
double CQuickReturn::sliderAccel(double theta2)
{
   calcAcceleration(theta2);

   if(m_uscunit)
   {
      return m_r6_double_dot /= M_FT2M; // m --> ft
   }
   else
   {
      return m_r6_double_dot;
   }
}

/*******************************************************************
 * sliderRange()
 *
 * Calculate the maximum and minimum distances
 * that the slider travels.
 *
 * Arguments: max ... storage for maximum position of the range
 *            min ... storage for minimum position of the range
 *******************************************************************/
void CQuickReturn::sliderRange(double& max, double& min)
{
   double max_pos=-100;
   double min_pos=100;
   double interval = 2*M_PI / (m_numpoints);
   int i;

   for(i=0; i<=m_numpoints; i++)
   {
      calcPosition(i*interval);
      max_pos = (m_r[6] > max_pos) ? m_r[6] : max_pos;
      min_pos = (m_r[6] < min_pos) ? m_r[6] : min_pos;
   }
   max = max_pos;
   min = min_pos;

   if(m_uscunit)
   {
      max /= M_FT2M;    // m --> ft
      min /= M_FT2M;    // m --> ft
   }
}

/**********************************************************
```

```
 * getRequiredTorque()
 *
 * Return Torque needed at input link for given theta2.
 *
 * Arguments: theta2 ... drive link positions
 *
 * Return value: input torque
 ***********************************************************/
double CQuickReturn::getRequiredTorque(double theta2)
{
   array double x[13];

   calcForce(theta2, x);

   if(m_uscunit)
   {
      return x[12] /= 1.355818991; // N-m --> lb-ft
   }
   else
   {
      return x[12];
   }
}

/******************************************************************
 * getAngPos()
 *
 * Return angular position for given theta2 & link number.
 *
 * Arguments: theta2 ... drive link positions
 *            link ... link angle desired
 *
 * Return value: theta for desired link
 ******************************************************************/
double CQuickReturn::getAngPos(double theta2, int link)
{
   double output;
   calcPosition(theta2);

   switch(link)
   {
      case QR_LINK_2:
         output = m_theta[2];
         break;
      case QR_LINK_4:
         output = m_theta[4];
         break;
      case QR_LINK_5:
         output = m_theta[5];
         break;
   }
   return output;
}

/******************************************************************
 * getAngVel()
 *
 * Return angular velocity for given theta2 & link number.
 *
```

```
 * Arguments: theta2 ... drive link positions
 *            link ... link angular velocity desired
 *
 * Return value: omega for desired link
 *****************************************************************/
double CQuickReturn::getAngVel(double theta2, int link)
{
   double output;
   calcVelocity(theta2);

   switch(link)
   {
      case QR_LINK_2:
         output = m_omega[2];
         break;
      case QR_LINK_4:
         output = m_omega[4];
         break;
      case QR_LINK_5:
         output = m_omega[5];
         break;
   }
   return output;
}

/*****************************************************************
 * getAngAccel()
 *
 * Return angular acceleration for given theta2 & link number.
 *
 * Arguments: theta2 ... drive link positions
 *            link ... link angular acceleration desired
 *
 * Return value: alpha for desired link
 *****************************************************************/
double CQuickReturn::getAngAccel(double theta2, int link)
{
   double output;
   calcAcceleration(theta2);

   switch(link)
   {
      case QR_LINK_2:
         output = m_alpha[2];
         break;
      case QR_LINK_4:
         output = m_alpha[4];
         break;
      case QR_LINK_5:
         output = m_alpha[5];
         break;
   }
   return output;
}

/*****************************************************************
 * getPointPos()
 *
 * Return position of a point for given theta2 & point number.
```

```
 *
 * Arguments: theta2 ... drive link positions
 *            point ... point position desired
 *
 * Return value: position for desired point as complex number
 ***************************************************************/
double complex CQuickReturn::getPointPos(double theta2, int point)
{
   double complex output;
   calcPosition(theta2);

   switch(point)
   {
      case QR_POINT_A:
         output = polar(m_r[3], m_theta[4]);
         break;
      case QR_POINT_B:
         output = polar(m_r[4], m_theta[4]);
         break;
      case QR_LINK_2_CG:
         output = polar(m_r[1], m_theta[1]) + polar(m_rg[2], m_theta[2]);
         break;
      case QR_LINK_4_CG:
         output = polar(m_rg[4], m_theta[4]);
         break;
      case QR_LINK_5_CG:
         output = polar(m_r[4], m_theta[4]) + polar(m_rg[5], m_theta[5]);
         break;
   }

   if(m_uscunit)
   {
      output /= 0.304800609;    // m --> ft
   }

   return output;
}

/***************************************************************
 * getPointVel()
 *
 * Return velocity of a piont for given theta2 & point number.
 *
 * Arguments: theta2 ... drive link positions
 *            point ... point velocity desired
 *
 * Return value: velocity for desired point as complex number
 ***************************************************************/
double complex CQuickReturn::getPointVel(double theta2, int point)
{
   double complex output, x, y;
   calcVelocity(theta2);

   switch(point)
   {
      case QR_POINT_A:
         output = I*polar(m_r[3]*m_omega[4], m_theta[4]);
         break;
      case QR_POINT_B:
```

```
         output = I*polar(m_r[4]*m_omega[4], m_theta[4]);
         break;
      case QR_LINK_2_CG:
         output = I*polar(m_rg[2]*m_omega[2], m_theta[2]);
         break;
      case QR_LINK_4_CG:
         output = I*polar(m_rg[4]*m_omega[4], m_theta[4]);
         break;
      case QR_LINK_5_CG:
         x = I*polar(m_r[4]*m_omega[4], m_theta[4]);
         y = I*polar(m_rg[5]*m_omega[5], m_theta[5]);
         output = x + y;
         break;
   }

   if(m_uscunit)
   {
      output /= 0.304800609;    // m --> ft
   }

   return output;
}

/*********************************************************************
 * getPointAccel()
 *
 * Return acceleration of a point for given theta2 & point number.
 *
 * Arguments: theta2 ... drive link positions
 *            point ... point acceleration desired
 *
 * Return value: acceleration for desired point as complex number
 *********************************************************************/
double complex CQuickReturn::getPointAccel(double theta2, int point)
{
   double complex output, x, y;
   calcAcceleration(theta2);

   switch(point)
   {
      case QR_POINT_A:
         x = I*polar((m_r[2]*m_alpha[2]), m_theta[2]);
         y = -polar((m_r[2]*m_omega[2]*m_omega[2]), m_theta[2]);
         output = x + y;
         break;
      case QR_POINT_B:
         x = I*polar((m_r[4]*m_alpha[4]), m_theta[4]);
         y = -polar((m_r[4]*m_omega[4]*m_omega[4]), m_theta[4]);
         output = x + y;
         break;
      case QR_LINK_2_CG:
         output = m_ag2;
         break;
      case QR_LINK_4_CG:
         output = m_ag4;
         break;
      case QR_LINK_5_CG:
         output = m_ag5;
         break;
```

```
   }

   if(m_uscunit)
   {
      output /= 0.304800609;    // m --> ft
   }

   return output;
}

/*****************************************************************************
 * getForces()
 *
 * Calculates internal forces of mechanism at a give angle
 * of the input link.
 *
 * Arguments: theta2 ... drive link positions
 *  y[13] = f12x,f12y,f23x,f23y,f14x,f14y,f34x,f34y,f45x,f45y,f56x,f56y,f16y
 *****************************************************************************/
void CQuickReturn::getForces(double theta2, array double y[12])
{
   int i;
   array double x[13];

   calcForce(theta2, x);

   for(i = 0; i < 12; i++)
   {
      y[i] = x[i];
   }

   if(m_uscunit)
   {
      y /= 4.448221615;  // N --> lb
   }
}

/***************************************************************
 * plotSliderPos()
 *
 * Plot slider position as function of time.
 *
 * Arguments: *plot ... pointer to plot defined
 *                      in the calling program
 ***************************************************************/
void CQuickReturn::plotSliderPos(CPlot *plot)
{
   int i;
   double increment;
   string_t  xlabel, ylabel, title;
   array double time[m_numpoints + 1], Position[m_numpoints + 1];

   increment = 2*M_PI/m_numpoints;
   if(m_omega[2] < 0)
      increment = -increment;

   for(i = 0; i <= m_numpoints; i++)
   {
      time[i] = (i*increment/m_omega[2]);
```

```
      Position[i] = sliderPos(i*increment);
   }

   title = "Slider Position vs. Time"; xlabel = "time (s)";

   if(m_uscunit)
   {
      ylabel = "Position (ft)";
   }
   else
   {
      ylabel = "Position (m)";
   }

   plotxy(time, Position, title, xlabel, ylabel, plot);
   plot->plotting();
}

/***********************************************************
 * plotSliderVel()
 *
 * Plot slider velocity as function of time.
 *
 * Arguments: *plot ... pointer to plot defined
 *                      in the calling program
 ***********************************************************/
void CQuickReturn::plotSliderVel(CPlot *plot)
{
   int i;
   double increment;
   string_t  xlabel, ylabel, title;
   array double time[m_numpoints + 1], Velocity[m_numpoints + 1];

   increment = 2*M_PI/m_numpoints;
   if(m_omega[2] < 0)
      increment = -increment;

   for(i = 0; i <= m_numpoints; i++)
   {
      time[i] = (i*increment/m_omega[2]);
      Velocity[i] = sliderVel(i*increment);
   }
   title = "Slider Velocity vs. Time"; xlabel = "time (s)";

   if(m_uscunit)
   {
      ylabel = "Velocity (ft/s)";
   }
   else
   {
      ylabel = "Velocity (m/s)";
   }

   plotxy(time, Velocity, title, xlabel, ylabel, plot);
   plot->plotting();
}

/***********************************************************
 * plotSliderAccel()
```

```
 *
 * Plot slider acceleraion as function of time.
 *
 * Arguments: *plot ... pointer to plot defined
 *                      in the calling program
 **********************************************************/
void CQuickReturn::plotSliderAccel(CPlot *plot)
{
   int i;
   double increment;
   string_t  xlabel, ylabel, title;
   array double time[m_numpoints + 1], Acceleration[m_numpoints + 1];

   increment = 2*M_PI/m_numpoints;
   if(m_omega[2] < 0)
      increment = -increment;

   for(i = 0; i <= m_numpoints; i++)
   {
      time[i] = (i*increment/m_omega[2]);
      Acceleration[i] = sliderAccel(i*increment);
   }
   title = "Slider Acceleration vs. Time"; xlabel = "time (s)";

   if(m_uscunit)
   {
      ylabel = "Acceleration (ft/s^2)";
   }
   else
   {
      ylabel = "Acceleration (m/s^2)";
   }

   plotxy(time, Acceleration, title, xlabel, ylabel, plot);
   plot->plotting();
}

/***************************************************************
 * plotAngPos()
 *
 * Plot angular position of links 4 & 5 as function of time.
 *
 * Arguments: *plot ... pointer to plot defined
 *                      in the calling program
 ***************************************************************/
void CQuickReturn::plotAngPos(CPlot *plot)
{
   int i;
   double increment;
   string_t  xlabel, ylabel, title;
   array double time[m_numpoints + 1];
   array double Theta4[m_numpoints + 1], Theta5[m_numpoints + 1];

   increment = 2*M_PI/m_numpoints;
   if(m_omega[2] < 0)
      increment = -increment;

   for(i = 0; i <= m_numpoints; i++)
   {
```

```
      calcPosition(i*increment);
      time[i] = (i*increment/m_omega[2]);
      Theta4[i] = m_theta[4];
      Theta5[i] = m_theta[5];
   }
   title = "Theta 4 vs. Time"; xlabel = "time (s)"; ylabel = "Theta 4 (rad)";
   plotxy(time, Theta4, title, xlabel, ylabel, plot);
   plot->plotting();

   title = "Theta 5 vs. Time"; xlabel = "time (s)"; ylabel = "Theta 5 (rad)";
   plotxy(time, Theta5, title, xlabel, ylabel, plot);
   plot->plotting();
}

/***************************************************************
 * plotAngVel()
 *
 * Plot angular velocity of links 4 & 5 as function of time.
 *
 * Arguments: *plot ... pointer to plot defined
 *                      in the calling program
 ***************************************************************/
void CQuickReturn::plotAngVel(CPlot *plot)
{
   int i;
   double increment;
   string_t  xlabel, ylabel, title;
   array double time[m_numpoints + 1];
   array double Omega4[m_numpoints + 1], Omega5[m_numpoints + 1];

   increment = 2*M_PI/m_numpoints;
   if(m_omega[2] < 0)
      increment = -increment;

   for(i = 0; i <= m_numpoints; i++)
   {
      calcVelocity(i*increment);
      time[i] = (i*increment/m_omega[2]);
      Omega4[i] = m_omega[4];
      Omega5[i] = m_omega[5];
   }
   title = "Omega 4 vs. Time"; xlabel = "time (s)"; ylabel = "Omega 4 (rad/s)";
   plotxy(time, Omega4, title, xlabel, ylabel, plot);
   plot->plotting();

   title = "Omega 5 vs. Time"; xlabel = "time (s)"; ylabel = "Omega 5 (rad/s)";
   plotxy(time, Omega5, title, xlabel, ylabel, plot);
   plot->plotting();
}

/********************************************************************
 * plotAngAccel()
 *
 * Plot angular acceleration of links 4 & 5 as function of time.
 *
 * Arguments: *plot ... pointer to plot defined
 *                      in the calling program
 ********************************************************************/
void CQuickReturn::plotAngAccel(CPlot *plot)
```

```
{
   int i;
   double increment;
   string_t  xlabel, ylabel, title;
   array double time[m_numpoints + 1];
   array double Alpha4[m_numpoints + 1], Alpha5[m_numpoints + 1];

   increment = 2*M_PI/m_numpoints;
   if(m_omega[2] < 0)
      increment = -increment;

   for(i = 0; i <= m_numpoints; i++)
   {
      calcAcceleration(i*increment);
      time[i] = (i*increment/m_omega[2]);
      Alpha4[i] = m_alpha[4];
      Alpha5[i] = m_alpha[5];
   }
   title = "Alpha 4 vs. Time"; xlabel = "time (s)"; ylabel = "Alpha 4 (rad/sˆ2)";
   plotxy(time, Alpha4, title, xlabel, ylabel, plot);
   plot->plotting();

   title = "Alpha 5 vs. Time"; xlabel = "time (s)"; ylabel = "Alpha 5 (rad/sˆ2)";
   plotxy(time, Alpha5, title, xlabel, ylabel, plot);
   plot->plotting();
}

/***************************************************************
 * plotCGaccel()
 *
 * Plot acceleration of centers of mass for links 2, 4, & 5
 * as functions of time for X & Y components of acceleration
 * and magnitude of acceleration.
 *
 * Arguments: *plot ... pointer to plot defined
 *                      in the calling program
 ***************************************************************/
void CQuickReturn::plotCGaccel(CPlot *plot)
{
   int i;
   double increment;
   string_t  xlabel, ylabel, title;
   array double time[m_numpoints + 1];
   array double complex ag2[m_numpoints + 1], ag4[m_numpoints + 1];
   array double complex ag5[m_numpoints + 1], mag_ag[m_numpoints + 1];

   increment = 2*M_PI/m_numpoints;
   if(m_omega[2] < 0)
      increment = -increment;

   for(i = 0; i <= m_numpoints; i++)
   {
      calcAcceleration(i*increment);
      time[i] = (i*increment/m_omega[2]);
      ag2[i] = m_ag2;
      ag4[i] = m_ag4;
      ag5[i] = m_ag5;
   }
```

```
if(m_uscunit)
{
   ag2 /= 0.304800609;  // m --> ft
   ag4 /= 0.304800609;  // m --> ft
   ag5 /= 0.304800609;  // m --> ft
}

title = "Ag2x vs. Time"; xlabel = "time (s)";
if(m_uscunit)
   ylabel = "Ag2x (ft/sˆ2)";
else
   ylabel = "Ag2x (m/sˆ2)";
plotxy(time, real(ag2), title, xlabel, ylabel,plot);
plot->plotting();

title = "Ag2y vs. Time"; xlabel = "time (s)";
if(m_uscunit)
   ylabel = "Ag2y (ft/sˆ2)";
else
   ylabel = "Ag2y (m/sˆ2)";
plotxy(time, imag(ag2), title, xlabel, ylabel, plot);
plot->plotting();

title = "Magnitude of Ag2 vs. Time"; xlabel = "time (s)";
if(m_uscunit)
   ylabel = "Ag2 (ft/sˆ2)";
else
   ylabel = "Ag2 (m/sˆ2)";
mag_ag = abs(ag2);
plotxy(time, mag_ag, title, xlabel, ylabel, plot);
plot->plotting();

title = "Ag4x vs. Time"; xlabel = "time (s)";
if(m_uscunit)
   ylabel = "Ag4x (ft/sˆ2)";
else
   ylabel = "Ag4x (m/sˆ2)";
plotxy(time, real(ag4), title, xlabel, ylabel, plot);
plot->plotting();

title = "Ag4y vs. Time"; xlabel = "time (s)";
if(m_uscunit)
   ylabel = "Ag4y (ft/sˆ2)";
else
   ylabel = "Ag4y (m/sˆ2)";
plotxy(time, imag(ag4), title, xlabel, ylabel, plot);
plot->plotting();

title = "Magnitude of Ag4 vs. Time"; xlabel = "time (s)";
if(m_uscunit)
   ylabel = "Ag4 (ft/sˆ2)";
else
   ylabel = "Ag4 (m/sˆ2)";
mag_ag = abs(ag4);
plotxy(time, mag_ag, title, xlabel, ylabel, plot);
plot->plotting();

title = "Ag5x vs. Time"; xlabel = "time (s)";
if(m_uscunit)
```

```
      ylabel = "Ag5x (ft/sˆ2)";
   else
      ylabel = "Ag5x (m/sˆ2)";
   plotxy(time, real(ag5), title, xlabel, ylabel, plot);
   plot->plotting();

   title = "Ag5y vs. Time"; xlabel = "time (s)";
   if(m_uscunit)
      ylabel = "Ag5y (ft/sˆ2)";
   else
      ylabel = "Ag5y (m/sˆ2)";
   plotxy(time, imag(ag5), title, xlabel, ylabel, plot);
   plot->plotting();

   title = "Magnitude of Ag5 vs. Time"; xlabel = "time (s)";
   if(m_uscunit)
      ylabel = "Ag5 (ft/sˆ2)";
   else
      ylabel = "Ag5 (m/sˆ2)";
   mag_ag = abs(ag5);
   plotxy(time, mag_ag, title, xlabel, ylabel, plot);
   plot->plotting();
}

/***************************************************************
 * plotForce()
 *
 * Plot specified internal force of quick return
 * mechanism as function of time.
 *
 * Arguments: plot_output  ... specifies which forces to plot
 *            *plot        ... pointer to plot defined
 *                             in the calling program
 ***************************************************************/
void CQuickReturn::plotForce(int plot_output, CPlot *plot)
{
   int i;
   double increment;
   string_t  xlabel, ylabel, title;
   array double time[m_numpoints + 1], Forces[12][m_numpoints + 1];
   array double F_array[13], (*F_plot)[:];
   array double complex F_mag[1][m_numpoints + 1];

   increment = 2*M_PI/m_numpoints;
   if(m_omega[2] < 0)
      increment = -increment;

   for(i = 0; i <= m_numpoints; i++)
   {
      // time for constant omega
      time[i] = (i*increment/m_omega[2]);
      calcForce(i*increment, F_array);
      Forces[0][i] = F_array[0];
      Forces[1][i] = F_array[1];
      Forces[2][i] = F_array[2];
      Forces[3][i] = F_array[3];
      Forces[4][i] = F_array[4];
      Forces[5][i] = F_array[5];
      Forces[6][i] = F_array[6];
```

```
      Forces[7][i] = F_array[7];
      Forces[8][i] = F_array[8];
      Forces[9][i] = F_array[9];
      Forces[10][i] = F_array[10];
      Forces[11][i] = F_array[11];
   }

   if(m_uscunit)
      Forces /= 4.448221615; // N --> lb

// F12 Plots
   F_plot = (array double [:][:])(double [1][m_numpoints + 1])&Forces[0][0];
   real(F_mag) = F_plot;
   if(plot_output & F12X)
   {
      title = "F12 x vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F12 x (lbs)";
      else
         ylabel = "F12 x (N)";
      plotxy(time, F_plot, title, xlabel, ylabel, plot);
      plot->plotting();
   }

   F_plot = (array double [:][:])(double [1][m_numpoints + 1])&Forces[1][0];
   imag(F_mag) = F_plot;
   if(plot_output & F12Y)
   {
      title = "F12 y vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F12 y (lbs)";
      else
         ylabel = "F12 y (N)";
      plotxy(time, F_plot, title, xlabel, ylabel, plot);
      plot->plotting();
   }

   if(plot_output & MAG_F12)
   {
      title = "Magnitude F12 vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F12 (lbs)";
      else
         ylabel = "F12 (N)";
      plotxy(time, abs(F_mag), title, xlabel, ylabel, plot);
      plot->plotting();
   }

// F23 Plots
   F_plot = (array double [:][:])(double [1][m_numpoints + 1])&Forces[2][0];
   real(F_mag) = F_plot;
   if(plot_output & F23X)
   {
      title = "F23 x vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F23 x (lbs)";
      else
         ylabel = "F23 x (N)";
      plotxy(time, F_plot, title, xlabel, ylabel, plot);
```

```
      plot->plotting();
   }

   F_plot = (array double [:][:])(double [1][m_numpoints + 1])&Forces[3][0];
   imag(F_mag) = F_plot;
   if(plot_output & F23Y)
   {
      title = "F23 y vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F23 y (lbs)";
      else
         ylabel = "F23 y (N)";
      plotxy(time, F_plot, title, xlabel, ylabel, plot);
      plot->plotting();
   }

   if(plot_output & MAG_F23)
   {
      title = "Magnitude F23 vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F23 (lbs)";
      else
         ylabel = "F23 (N)";
      plotxy(time, abs(F_mag), title, xlabel, ylabel, plot);
      plot->plotting();
   }

// F14 Plots
   F_plot = (array double [:][:])(double [1][m_numpoints + 1])&Forces[4][0];
   real(F_mag) = F_plot;
   if(plot_output & F14X)
   {
      title = "F14 x vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F14 x (lbs)";
      else
         ylabel = "F14 x (N)";
      plotxy(time, F_plot, title, xlabel, ylabel, plot);
      plot->plotting();
   }

   F_plot = (array double [:][:])(double [1][m_numpoints + 1])&Forces[5][0];
   imag(F_mag) = F_plot;
   if(plot_output & F14Y)
   {
      title = "F14 y vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F14 y (lbs)";
      else
         ylabel = "F14 y (N)";
      plotxy(time, F_plot, title, xlabel, ylabel, plot);
      plot->plotting();
   }

   if(plot_output & MAG_F14)
   {
      title = "Magnitude F14 vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F14 (lbs)";
```

```
      else
         ylabel = "F14 (N)";
      plotxy(time, abs(F_mag), title, xlabel, ylabel, plot);
      plot->plotting();
   }

// F34 Plots
   F_plot = (array double [:][:])(double [1][m_numpoints + 1])&Forces[6][0];
   if(plot_output & MAG_F34)
   {
      title = "Magnitude F34 vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F34 (lbs)";
      else
         ylabel = "F34 (N)";
      plotxy(time, F_plot, title, xlabel, ylabel, plot);
      plot->plotting();
   }

// F45 Plots
   F_plot = (array double [:][:])(double [1][m_numpoints + 1])&Forces[7][0];
   real(F_mag) = F_plot;
   if(plot_output & F45X)
   {
      title = "F45 x vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F45 x (lbs)";
      else
         ylabel = "F45 x (N)";
      plotxy(time, F_plot, title, xlabel, ylabel, plot);
      plot->plotting();
   }

   F_plot = (array double [:][:])(double [1][m_numpoints + 1])&Forces[8][0];
   imag(F_mag) = F_plot;
   if(plot_output & F45Y)
   {
      title = "F45 y vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F45 y (lbs)";
      else
         ylabel = "F45 y (N)";
      plotxy(time, F_plot, title, xlabel, ylabel, plot);
      plot->plotting();
   }

   if(plot_output & MAG_F45)
   {
      title = "Magnitude F45 vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F45 (lbs)";
      else
         ylabel = "F45 (N)";
      plotxy(time, abs(F_mag), title, xlabel, ylabel, plot);
      plot->plotting();
   }

// F56 Plots
   F_plot = (array double [:][:])(double [1][m_numpoints + 1])&Forces[9][0];
```

```
   real(F_mag) = F_plot;
   if(plot_output & F56X)
   {
      title = "F56 x vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F56 x (lbs)";
      else
         ylabel = "F56 x (N)";
      plotxy(time, F_plot, title, xlabel, ylabel, plot);
      plot->plotting();
   }

   F_plot = (array double [:][:])(double [1][m_numpoints + 1])&Forces[10][0];
   imag(F_mag) = F_plot;
   if(plot_output & F56Y)
   {
      title = "F56 y vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F56 y (lbs)";
      else
         ylabel = "F56 y (N)";
      plotxy(time, F_plot, title, xlabel, ylabel, plot);
      plot->plotting();
   }

   if(plot_output & MAG_F56)
   {
      title = "Magnitude F56 vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F56 (lbs)";
      else
         ylabel = "F56 (N)";
      plotxy(time, abs(F_mag), title, xlabel, ylabel, plot);
      plot->plotting();
   }

// F16 Plots
   F_plot = (array double [:][:])(double [1][m_numpoints + 1])&Forces[11][0];
   if(plot_output & F16Y)
   {
      title = "F16 y vs. Time"; xlabel = "time (s)";
      if(m_uscunit)
         ylabel = "F16 y (lbs)";
      else
         ylabel = "F16 y (N)";
      plotxy(time, F_plot, title, xlabel, ylabel, plot);
      plot->plotting();
   }
}

/****************************************************************
 * plotTorque()
 *
 * Plot input torque to the quick return mechanism
 * as function of time.
 *
 * Arguments: *plot ... pointer to plot defined
 *                      in the calling program
 ****************************************************************/
```

```
void CQuickReturn::plotTorque(CPlot *plot)
{
   int i;
   double increment;
   string_t  xlabel, ylabel, title;
   array double time[m_numpoints + 1], Ts[m_numpoints + 1];

   increment = 2*M_PI/m_numpoints;
   if(m_omega[2] < 0)
      increment = -increment;

   for(i = 0; i <= m_numpoints; i++)
   {
      // time for constant omega
      time[i] = (i*increment/m_omega[2]);
      Ts[i] = getRequiredTorque(i*increment);
   }

   title = "Input Torque vs. Time"; xlabel = "time (s)";
   if(m_uscunit)
      ylabel = "Torque (ft-lb)";
   else
      ylabel = "Torque (N-m)";
   plotxy(time, Ts, title, xlabel, ylabel, plot);
   plot->plotting();
}

/***************************************************************
 * displayPosition()
 *
 * This function will write a file that can be run with
 * qanimate to display the configuration of the mechanism.
 *
 * Arguments: theta2 ... determines the configuration
 ***************************************************************/
void CQuickReturn::displayPosition(double theta2, ...)
{
   complex R[1:8];
   double sliderlength = m_r[4] / 8;
   double sliderwidth = sliderlength / 2;
   char *QnmFileName;
   FILE *positionpipe;
   int outputType = QANIMATE_OUTPUTTYPE_DISPLAY; // default display
   va_list ap;
   int vacount;

   va_start(ap, theta2);
   vacount = va_count(ap);
   if(vacount > 0)
      outputType = va_arg(ap, int);
   if(outputType == QANIMATE_OUTPUTTYPE_STREAM)
      positionpipe = stdout;
   else
   {
      if(outputType == QANIMATE_OUTPUTTYPE_FILE)
         QnmFileName = va_arg(ap, char*);
      else
         QnmFileName = tempnam("C:/Windows/temp", NULL);
      // Try to open output file
```

```
   if(!(positionpipe = fopen(QnmFileName, "w")))
   {
      fprintf(stderr, "displayPosition(): unable to open output file ’%s’\n", QnmFileName);
      return;
   }
}
va_end(ap);

// Call calcPosition() to calculate everything
calcPosition(theta2);

// Create complex to make it easier to use
R[1] = polar(m_r[1], m_theta[1]);
R[2] = R[1] + polar(m_r[2], m_theta[2]);
R[4] = polar(m_r[4], m_theta[4]);
R[5] = R[4] + polar(m_r[5], m_theta[5]);
R[3] = polar(m_r[3]-sliderlength/2, m_theta[4])
       + polar(sliderwidth/2, m_theta[4]-M_PI/2);
R[7] = polar(m_r[3]-sliderlength/2, m_theta[4]);
R[8] = polar(m_r[3]+sliderlength/2, m_theta[4]);

/* display position */
fprintf(positionpipe, "#qanimate position data\n");
fprintf(positionpipe, "title \"Position of the Whitworth"
                      " Quick Return Mechanism\"\n\n");
fprintf(positionpipe, "fixture\n");
fprintf(positionpipe, "groundpin 0 0 %f %f\n\n",
        real(R[1]), imag(R[1]));
fprintf(positionpipe, "joint 0 0 \\\n");
fprintf(positionpipe, "ground %f %f %f %f\n\n",
        real(R[5])-sliderlength, imag(R[5])-sliderwidth/2,
        real(R[5])+sliderlength, imag(R[5])-sliderwidth/2);
fprintf(positionpipe, "link %f %f %f %f \\\n",
        real(R[1]), imag(R[1]),
        real(R[2]), imag(R[2]));
fprintf(positionpipe, "line 0 0 %f %f \\\n",
        real(R[7]), imag(R[7]));
fprintf(positionpipe, "line %f %f %f %f \\\n",
        real(R[8]), imag(R[8]),
        real(R[4]), imag(R[4]));
fprintf(positionpipe, "link %f %f %f %f \\\n",
        real(R[4]), imag(R[4]),
        real(R[5]), imag(R[5]));
fprintf(positionpipe, "rectangle %f %f %f %f angle %f pen red \\\n",
        real(R[3]), imag(R[3]), sliderlength,
        sliderwidth, M_RAD2DEG(m_theta[4]));
fprintf(positionpipe, "rectangle %f %f %f %f angle %f pen blue \\\n",
        real(R[5])-sliderlength/2, imag(R[5])-sliderwidth/2,
        sliderlength, sliderwidth, 0.0);
fprintf(positionpipe, "text %f %f \"01\" \\\n",
        -sliderwidth/8, -sliderwidth);
fprintf(positionpipe, "text %f %f \"02\" \\\n",
        (abs(m_theta[2]) > M_PI/2 && abs(m_theta[2]) < 3*M_PI/2)
           ?(real(R[1])+sliderwidth):(real(R[1])-sliderwidth),imag(R[1]));
fprintf(positionpipe, "text %f %f \"A\" \\\n",
        real(polar(m_r[3]+sliderlength/4,m_theta[4])
           +polar(sliderwidth, -M_PI/2+m_theta[4])),
        imag(polar(m_r[3]+sliderlength/4,m_theta[4])
           +polar(sliderwidth, -M_PI/2+m_theta[4])));
```

```
   fprintf(positionpipe, "text %f %f \"B\" \\\n",
           real(R[4]), imag(R[4])+sliderwidth/2);
   fprintf(positionpipe, "text %f %f \"r_5\" \\\n",
           real(R[4]+polar(m_r[5]/2, m_theta[5])
              +polar(sliderwidth/2, M_PI/2+m_theta[5])),
           imag(R[4]+polar(m_r[5]/2, m_theta[5])
              +polar(sliderwidth/2, M_PI/2+m_theta[5])));
   fprintf(positionpipe, "text %f %f \"6\" \\\n",
           real(R[5]), imag(R[5])+sliderwidth);
   fprintf(positionpipe, "text %f %f \"r_4\" \\\n",
           real(polar(sliderwidth, m_theta[4])
              +polar(sliderwidth/2, m_theta[4]-M_PI/2)),
           imag(polar(sliderwidth, m_theta[4])
              +polar(sliderwidth/2, m_theta[4]-M_PI/2)));
   fprintf(positionpipe, "text %f %f \"r_2\"\n\n",
           real(R[1]+polar(m_r[2]/3, m_theta[2])
              +polar(sliderwidth/3, M_PI/2-m_theta[2])),
           imag(R[1]+polar(m_r[2]/3, m_theta[2])
              +polar(sliderwidth/3, M_PI/2-m_theta[2])));

   if(outputType == QANIMATE_OUTPUTTYPE_FILE)
   {
      fclose(positionpipe);
   }
   else if(outputType == QANIMATE_OUTPUTTYPE_DISPLAY)
   {
      fclose(positionpipe);
      #ifndef _DARWIN_
         qanimate $QnmFileName
      #endif // DARWIN
      remove(QnmFileName);
      free(QnmFileName);
   }
}

/****************************************************************
 * animation()
 *
 * This function will write a file that can be run with
 * qanimate to display an animation of the mechanism as
 * the second link is rotated.
 ****************************************************************/
void CQuickReturn::animation(...)
{
   complex R[1:8];
   double interval = 2*M_PI / (m_numpoints);
   double sliderlength = m_r[4] / 8;
   double sliderwidth = sliderlength / 2;
   double max, min;
   int i;
   char *QnmFileName;
   int outputType = QANIMATE_OUTPUTTYPE_DISPLAY; // default display
   FILE *animationpipe;
   va_list ap;
   int vacount;

   va_start(ap, VA_NOARG);
   vacount = va_count(ap);
   if(vacount > 0)
```

```
   outputType = va_arg(ap, int);
if(outputType == QANIMATE_OUTPUTTYPE_STREAM)
   animationpipe = stdout;
else
{
   if(outputType == QANIMATE_OUTPUTTYPE_FILE)
      QnmFileName = va_arg(ap, char*);
   else
      QnmFileName = tempnam("C:/Windows/temp", NULL);
   // Try to open output file
   if(!(animationpipe = fopen(QnmFileName, "w")))
   {
      fprintf(stderr, "animation(): unable to open output file ’%s’\n", QnmFileName);
      return;
   }
}
va_end(ap);

sliderRange(max, min);
R[1] = polar(m_r[1], m_theta[1]);

/* Write header part */
fprintf(animationpipe, "#qanimate animation data\n");
fprintf(animationpipe, "title \"Animation of the Whitworth Quick"
                       " Return Mechanism\"\n\n");
fprintf(animationpipe, "fixture\n");
fprintf(animationpipe, "groundpin 0 0 %f %f\n\n",
        real(R[1]), imag(R[1]));
fprintf(animationpipe, "ground %f %f %f %f \n\n"
        ,min-sliderlength
        ,m_r[7]-sliderwidth/2
        ,max+sliderlength
        ,m_r[7]-sliderwidth/2);
fprintf(animationpipe, "animate restart\n");

for(i=0; i < m_numpoints; i++)
{
   // Call calcPosition() to calculate everything
   calcPosition(i*interval*m_omega[2]/abs(m_omega[2]));

   // Create complex to make it easier to use
   R[2] = R[1] + polar(m_r[2], m_theta[2]);
   R[4] = polar(m_r[4], m_theta[4]);
   R[5] = R[4] + polar(m_r[5], m_theta[5]);
   R[3] = polar(m_r[3]-sliderlength/2, m_theta[4])
          + polar(sliderwidth/2, m_theta[4]-M_PI/2);
   R[7] = polar(m_r[3]-sliderlength/2, m_theta[4]);
   R[8] = polar(m_r[3]+sliderlength/2, m_theta[4]);

   /* Write animation part */
   fprintf(animationpipe, "link %f %f %f %f \\\n",
           real(R[1]), imag(R[1]),
           real(R[2]), imag(R[2]));
   fprintf(animationpipe, "line 0 0 %f %f \\\n",
           real(R[7]), imag(R[7]));
   fprintf(animationpipe, "line %f %f %f %f \\\n",
           real(R[8]), imag(R[8]),
           real(R[4]), imag(R[4]));
   fprintf(animationpipe, "link %f %f %f %f \\\n",
```

```
              real(R[4]), imag(R[4]),
              real(R[5]), imag(R[5]));
      fprintf(animationpipe, "rectangle %f %f %f %f angle %f pen red \\\n",
              real(R[3]), imag(R[3]), sliderlength, sliderwidth,
              M_RAD2DEG(m_theta[4]));
      fprintf(animationpipe, "rectangle %f %f %f %f angle %f pen blue \\\n",
              real(R[5])-sliderlength/2, imag(R[5])-sliderwidth/2,
              sliderlength, sliderwidth, 0.0);
      fprintf(animationpipe, "joint 0 0 \\\n");
      fprintf(animationpipe, "stopped \\\n");
      fprintf(animationpipe, "text %f %f \"01\" \\\n",
              -sliderwidth/8, -sliderwidth);
      fprintf(animationpipe, "text %f %f \"02\" \\\n",
              (abs(m_theta[2]) > M_PI/2 && abs(m_theta[2]) < 3*M_PI/2)
                 ?(real(R[1])+sliderwidth):(real(R[1])-sliderwidth),imag(R[1]));
      fprintf(animationpipe, "text %f %f \"A\" \\\n",
              real(polar(m_r[3]+sliderlength/4,m_theta[4])
                 +polar(sliderwidth, -M_PI/2+m_theta[4])),
              imag(polar(m_r[3]+sliderlength/4,m_theta[4])
                 +polar(sliderwidth, -M_PI/2+m_theta[4])));
      fprintf(animationpipe, "text %f %f \"B\" \\\n",
              real(R[4]), imag(R[4])+sliderwidth/2);
      fprintf(animationpipe, "text %f %f \"r_5\" \\\n",
              real(R[4]+polar(m_r[5]/2, m_theta[5])
                 +polar(sliderwidth/2, M_PI/2+m_theta[5])),
              imag(R[4]+polar(m_r[5]/2, m_theta[5])
                 +polar(sliderwidth/2, M_PI/2+m_theta[5])));
      fprintf(animationpipe, "text %f %f \"6\" \\\n",
              real(R[5]), imag(R[5])+sliderwidth);
      fprintf(animationpipe, "text %f %f \"r_4\" \\\n",
              real(polar(sliderwidth, m_theta[4])
                 +polar(sliderwidth/2, m_theta[4]-M_PI/2)),
              imag(polar(sliderwidth, m_theta[4])
                 +polar(sliderwidth/2, m_theta[4]-M_PI/2)));
      fprintf(animationpipe, "text %f %f \"r_2\"\n\n",
              real(R[1]+polar(m_r[2]/3, m_theta[2])
                 +polar(sliderwidth/3, M_PI/2-m_theta[2])),
              imag(R[1]+polar(m_r[2]/3, m_theta[2])
                 +polar(sliderwidth/3, M_PI/2-m_theta[2])));
   }

   if(outputType == QANIMATE_OUTPUTTYPE_FILE)
   {
      fclose(animationpipe);
   }
   else if(outputType == QANIMATE_OUTPUTTYPE_DISPLAY)
   {
      fclose(animationpipe);
      #ifndef _DARWIN_
         qanimate $QnmFileName
      #endif // DARWIN
      remove(QnmFileName);
      free(QnmFileName);
   }
}
```

Index