Engineering with Computers (2006) 21: 237–246
DOI 10.1007/s00366-005-0008-4

ORIGINAL ARTICLE

Harry H. Cheng · Dung T. Trang

Object-oriented interactive mechanism design and analysis

Received: 17 September 2004 / Accepted: 3 February 2005 / Published online: 15 December 2005

Abstract We have developed a Ch Mechanism Toolkit for analysis and design of mechanisms. The toolkit was developed using Ch, an embeddable C/C++ interpreter with high-level extensions. The toolkit consists of animation program QuickAnimation™ and a collection of classes for design and analysis of commonly used mechanisms. The Ch Mechanism Toolkit allows users to write simple programs to solve complicated planar mechanism problems. The toolkit can handle mechanisms ranging from the simple fourbar linkage to various sixbar linkages. The Ch Mechanism Toolkit can also be used to design cam-follower systems. It is an effective tool for engineering practice as well as for teaching and learning mechanism design and analysis. This paper describes the design and implementation of the Ch Mechanism Toolkit and its applications.

Keywords Mechanisms · Fourbar linkage · Ch · C/C++ interpreter

1 Introduction

Computational methods for analysis and design of mechanical systems have become increasingly popular in engineering practice. Procedures that were often tedious in the past can now be completed easily with the aid of computers. For example, general purpose software packages such as Pro/ENGINEER, Automated Dynamic Analysis of Mechanical Systems (ADAMS), Dynamic Analysis and Design System (DADS), and Working Model [1] were developed to solve complicated engineering design and analysis problems. Other software packages includes Linkage INteractive Computer Analysis and Graphical Enhanced Synthesis Package (LINCAGES) [2, 3], Watt by Heron [4], and Simulation and Analysis of Mechanisms (SAM) by Artas [5], which are available for synthesis and analysis of planar mechanisms. The software package Synthetica [6] can be used for synthesis of spatial mechanisms. However, the numerical aspects and software implementation of mechanism analysis and design cannot be easily appreciated through the graphical user interface of these software packages. The analytical sequences of the algorithm are not transparent to users in these menu-driven software packages. Therefore, they are not ideal for applications such as teaching and learning of mechanism design. In addition to basic principles, it is increasingly important to understand the computational aspects of the subject. These software packages cannot readily incorporate numerical algorithms such as optimization into the system. On the other hand, general-purpose mathematical and dynamic simulation packages, such as Autolev [7] based on Kane's dynamics formulation, are not convenient for rapid design and analysis of mechanisms. Nevertheless, the C code generated from Autolev can readily run in Ch without any modification [8].

We have developed a Ch Mechanism Toolkit [9] for mechanism design and analysis. The toolkit is developed in C/C++ interpreter Ch [10–12]. Ch, conforming to the international C standard with extensions, contains all salient features of MATLAB for numerical and script computing. A Ch Control System Toolkit is available for design and analysis of control systems [13, 14]. The Ch Mechanism Toolkit [9] is significantly different from other software packages. The toolkit is modular and contains many objects as building blocks. These small building blocs are easy to develop and maintain. The users are able to examine the available source code and sample programs provided within the toolkit. The Ch Mechanism Toolkit [9] has been used to teach the undergraduate course *Computer-Aided Mechanism*

Submitted to Engineering with Computers

H. H. Cheng (✉) · D. T. Trang
Integration Engineering Laboratory, Department of Mechanical and Aeronautical Engineering, University of California, Davis, CA, 95616 USA
E-mail: hhcheng@ucdavis.edu
Tel.: +1-530-7525020
Fax: +1-530-7524158

Design at the University of California, Davis [15]. The toolkit provided students with the opportunity to study and understand the algorithms and their software implementation. Furthermore, students are able to examine sample programs and modify them accordingly to solve their own mechanism design and analysis problems. Through this learning-by-example process, students can better understand the principles and numerical aspects of the subject. Additionally, students can use the toolkit's high-level building blocks to develop their own software programs for solving complicated engineering analysis and design problems. For example, a package for design and analysis of Whitworth quick return mechanism was developed by students as a project for the class [16, 17]. The Ch Mechanism Toolkit is open architecture. Based on this toolkit, a Web-based mechanism design and analysis module has been developed. The user can design and analyze mechanisms interactively through a Web browser without any computer programming [18, 19]. There are also open source programs available for kinematic synthesis [20] based on Ch and its Mechanism Toolkit. The Ch Mechanism Toolkit can also be integrated into CAD environments such as Pro/ENGINEER, ADADMS, and DADS using their C Application Programming Interface (API) and Embedded Ch [21].

This paper describes the design and implementation of the Ch Mechanism Toolkit along with its various features. The convenience and simplicity of the toolkit are illustrated through application examples. Like modules for other mechanisms, the software for design and analysis of fourbar linkages is open source. The presentation of this paper will focus primarily on fourbar linkages. However, the ideas and concepts are applicable to other mechanisms as well.

2 Features of Ch Mechanism Toolkit

The Ch Mechanism Toolkit is a useful tool for design and analysis of planar mechanisms. Features of the toolkit make it convenient for engineering practice, and teaching and learning. This section briefly describes a few features of the Ch Mechanism Toolkit.

2.1 Analysis of various linkages

The Ch Mechanism Toolkit is capable of handling many kinds of planar mechanisms. Analysis can be performed on simple mechanisms such as the fourbar linkage as well as on more complex ones such as sixbar linkages. This toolkit supports the analysis of the fourbar, crank-slider, geared-fivebar, Stephenson I & III, and Watt I & II linkages. Furthermore, the Ch Mechanism Toolkit can be used to design cam-follower systems for both flat faced and roller followers.

2.2 Graphical outputs

The Ch Mechanism Toolkit supports graphical presentations in the form of plots and animations. The toolkit can output linkage analysis results graphically using the Ch plotting class **CPlot**. Class **CPlot** allows for high-level generation of two- and three-dimensional plotting. The users can readily customize the plots generated by the plotting class. For example, properties such as plot titles and axis labels can be set by calling member functions of class **CPlot**. Furthermore, rather than displaying the plots on the computer screen, they can be saved to files of different formats, such as a postscript file, PNG, and GIF.

The users are also able to simulate the motion of the various linkages and cam-follower systems available in the Ch Mechanism Toolkit. Each mechanism class contains an **animation()** function to perform this task. This member function utilizes the QuickAnimation™ software module to generate the desired animation. Figure 1 shows the drawing primitives available in QuickAnimation™. These basic primitives are used to create the mechanical drawing primitives shown in Fig. 2, which are used for generating linkage animations. Member function **animation()** utilizes these primitives to draw the linkages for each frame of animation. Similar to the plotting features of the toolkit, the animation data may also be saved to a file, with extension `.qnm`. Using the animation data, the QuickAnimation™ software module can perform animation at a later time.

3 Design and implementation of Ch Mechanism Toolkit

Object-oriented programming refers to the use of C++ style classes, which consist of a set of variables and

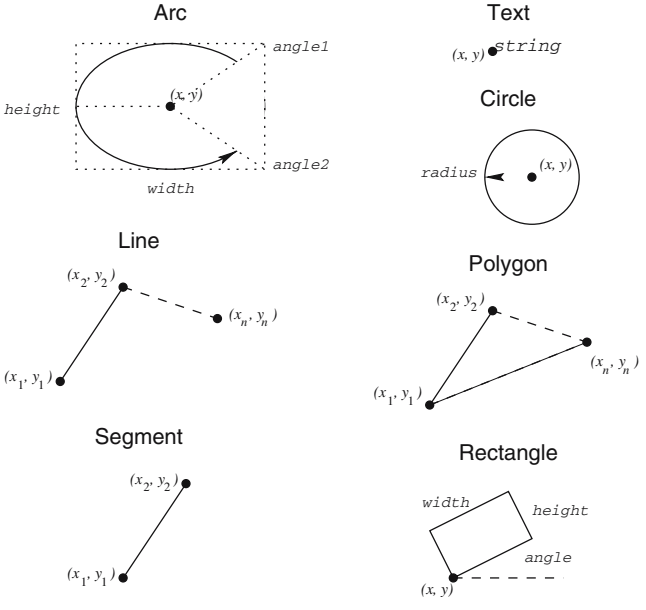

Fig. 1 Graphical representation of general drawing primitives

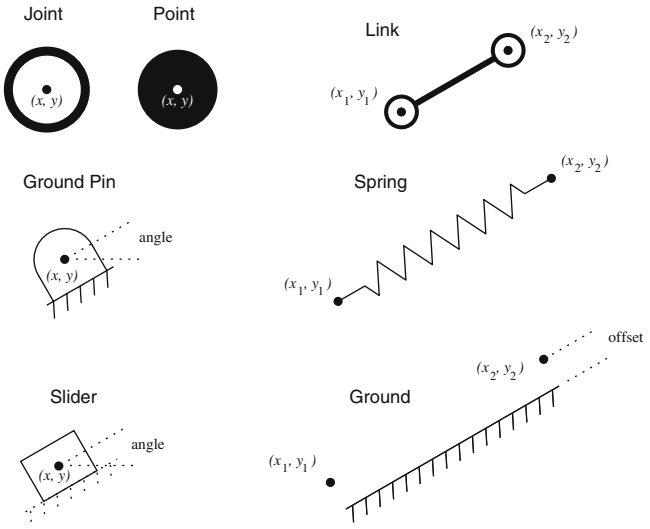

Fig. 2 Graphical representation of mechanical drawing primitives

functions. The attributes and operations for a class are usually referred to as data members and member functions, respectively. Data members are typically private for data encapsulation, meaning that they can only be accessed by member functions associated with the class. Member functions may either be private or public. Private member functions are considered utility functions to their public counterpart. Users have access only to public member functions of a class. They would use these public member functions to indirectly access the private data members and member functions of the class.

The Ch Mechanism Toolkit consists of many C++ style classes for analyzing various planar mechanisms. For example, classes **CFourbar**, **CCrankSlider**, and **CGearedFivebar** are used to handle the analysis and design of fourbar, crank-slider, and geared-fivebar linkages, respectively. Along with these classes, other classes are available for analysis of various sixbar linkages as well as cam-follower mechanisms. As with typical C++ classes, the classes of the Ch Mechanism Toolkit is comprised of private data members and private/public member functions. Private data members are used to store parameters that uniquely specifies the configuration of a certain mechanism, such as link dimensions. The member functions use the values stored in the data members to perform various calculations. For example, all classes of the Ch Mechanism Toolkit contain member functions for calculating the angular positions, velocities, and accelerations of individual link members for a given mechanism configuration. Furthermore, if a coupler is attached to one of the links, kinematic analysis can be performed on the coupler point as well.

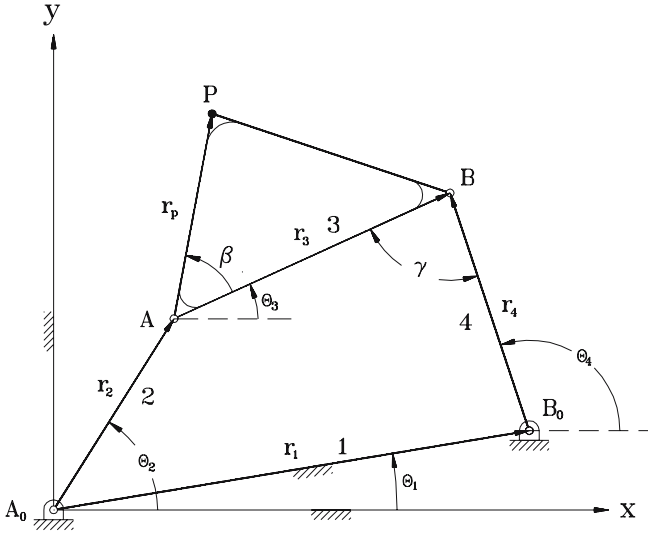

Fig. 3 Fourbar linkage

3.1 Data members

As described earlier, the private data members of a Ch Mechanism toolkit class consists of parameters that defines a unique configuration of a planar linkage. For example, consider the fourbar linkage shown in Fig. 3. Link lengths r_1, r_2, r_3, and r_4, and phase angle θ_1 for link 1 are some parameters that can be used to define a fourbar linkage. These values are used to perform various analysis on the fourbar linkage, such as calculating the angular position, velocity, and acceleration of the individual links. Table 1 is the list of private data members for class **CFourbar**. Note that the variable names are prefixed by `m_` to indicate that they are private members. Among the list of data members are

Table 1 Private data members of class **CFourbar**

Data member	Description
`double m_r[1:4]`	Link lengths
`double m_theta1`	Phase angle for ground link
`double m_rp, m_beta`	Coupler point parameters
`double m_inputlimitmin[2]`	Minimum value for theta 2
`double m_inputlimitmax[2]`	Maximum value for theta 2
`double m_outputlimitmin[2]`	Minimum value for theta 4
`double m_outputlimitmax[2]`	Maximum value for theta 4
`double m_rg[1:4]`	Distance to center of gravity
`double m_delta[1:4]`	Phase angle to center of gravity
`double m_mass[1:4]`	Mass of links
`double m_inertia[1:4]`	Inertia of links
`int m_numpoints`	Number of plotting points/animation frames
`double m_omega2`	Constant angular velocity for link 2
`int m_trace`	Trace option for animation
`bool m_uscunit`	Option for SI or US customary unit

Table 2 Private member functions of class **CFourbar**

Private member function	Description
`void m_createAnimationData()`	Generate animation data
`void m_createAnimationDataForRC()`	Generate animation data for rocker-crank mechanisms
`void m_drawAnimation()`	Draw animation
`void m_forcesForGeneral()`	Force analysis
`void m_forcesForRC()`	Force analysis for rocker-crank mechanisms
`int m_grashofTest()`	Perform Grashof test
`void m_initialize()`	Initialize data members
`double m_newTheta()`	Angle increment function for animation

`m_rp` and `m_beta`, which can be used to specify a coupler, if one is attached to the fourbar. Other members, such as `m_mass` and `m_inertia` refer to the mass and inertia values of the individual links. These parameters are essential in performing dynamic analysis.

3.2 Member functions

Most classes in the Ch Mechanism Toolkit contain both private and public member functions. Private member functions are often viewed as utility functions. They are usually used by public member functions to perform a specific task and to simplify the programming code.

3.2.1 Private member functions

The private member function of class **CFourbar** are listed in Table 2. Most of these functions are used for generating animation data and dynamics analysis. Member function **m_grashofTest()** determines whether or not the fourbar is a Grashof linkage, and **m_initialize()** initializes the private data members with default values.

3.2.2 Public member functions

Public member functions for various classes of the Ch Mechanism Toolkit can be divided into two categories: *setup* and *analysis* functions. The *setup* functions are those used to assign values to the private data members. They allow users to indicate the fourbar linkage to be analyzed. The rest of the public member functions are used to perform various analysis on the fourbar defined by the *setup* functions.

Table 3 lists all the *setup* functions for class **CFourbar**. The most commonly used function among those listed is member function **setLinks()**. The link lengths are the parameters that define a fourbar linkage. For example, the function prototype for member function **setLinks()** is as follows.

```
int setLinks(double r1, double r2, double r3, double r4,
             double theta1);
```

Calling this member function would assign the values of `r1`, `r2`, `r3`, and `r4` to data member `m_r`, which is an array with four elements representing the fourbar link lengths. The value of `theta1` would be stored in data member `m_theta1`. The other *setup* functions behave in a similar manner. That is, they are used to assign values to the respective data members. Note that users do not need to call all the setup functions when using class **CFourbar**. Only those applicable to the desired analysis are required. For example, if only the angular positions of the links are desired, member function **setCouplerPoint()** does not need to be called, since the coupler point is irrelevant in the results.

Table 3 Setup functions of class **CFourbar**

Setup function	Description
`void setCouplerPoint()`	Define coupler point
`void setGravityCenter()`	Define center of gravity of links
`void setInertia()`	Set inertia values of links
`void setAngularVel()`	Set constant angular velocity of input link, θ_2
`int setLinks()`	Set link lengths and θ_1
`void setMass()`	Set mass values
`void setNumPoints()`	Set number of plotting points/animation frames

Analysis functions available for class **CFourbar** are listed in Table 4. These functions can perform a wide range of tasks, ranging from kinematic analysis to dynamic analysis of the fourbar. Asides from numerical outputs, graphical outputs in the forms of plots and animations are available. These graphical outputs provide users with visual interpretations to enhance their understanding of the mechanism. For example, given the angular position of one link, member function **angularPos()** is used to calculate the angular position of the remaining links. On the other hand, given angles of two moving links, member function **getAngle()** calculates the angle of the remaining moving link. Member function **displayPosition()** displays a configuration of a fourbar linkage, whereas member function **displayPositions()** displays multiple configurations of a fourbar linkage. The animation feature allows users to simulate the motion of various linkages so that they can gain a better understanding of the behavior of the mechanism.

Table 4 Analysis functions of class **CFourbar**

Analysis function	Description
`int angularAccel()`	Angular acceleration analysis
`int angularPos()`	Angular position analysis
`int angularVel()`	Angular velocity analysis
`int animation()`	Simulate motion of fourbar
`void couplerCurve()`	Calculate coordinates of coupler curve
`double complex couplerPointAccel()`	Calculate coupler point acceleration
`void couplerPointPos()`	Calculate coupler point position
`double complex couplerPointVel()`	Calculate coupler point velocity
`int displayPosition()`	Display a configuration of fourbar linkage
`int displayPositions()`	Display multiple configurations of fourbar linkage
`int getAngle()`	Calculate a joint angle
`int getJointLimits()`	Calculate joint limits for input and output links
`int grashof()`	Determine the type of fourbar linkage
`void forceTorque()`	Calculate joint forces and output torque
`void plotCouplerCurve()`	Plot coupler curve
`void plotTransAngles()`	Plot transmission angles
`int printJointLimits()`	Print joint limits for input and output links
`int synthesis()`	Synthesis for fourbar linkage
`void transAngle()`	Calculate a transmission angle
`void transAngles()`	Calculate transmission angles
`void uscUnit()`	Specify SI or US customary unit

Member functions in Table 5 are used for analysis and plotting of position, velocity, acceleration, and force when the input link 2 rotates with a constant angular velocity.

Table 5 Analysis functions of class **CFourbar** for a constant angular velocity of input link 2

Analysis function	Description
`int angularAccels()`	Angular acceleration analysis
`int angularPoss()`	Angular position analysis
`int angularVels()`	Angular velocity analysis
`int forceTorques()`	Calculate joint forces and output torques
`void plotAngularAccels()`	Plot angular acceleration
`void plotAngularPoss()`	Plot angular position
`void plotAngularVels()`	Plot angular velocity
`void plotForceTorques()`	Plot joint forces and output torques

4 Program structure

Figure 4 shows the general form of programs utilizing the Ch Mechanism Toolkit. The program structure is quite simple and straightforward. The first step in writing a program for mechanism analysis is to include relevant header files. For example, header file **fourbar.h** is included in any program written for fourbar analysis. The next step is to declare the necessary variables, including variables for parameter values as well as results. In the variable declaration, an object of the desired class should be instantiated. In Fig. 4, the line

instantiates an object of class **CFourbar** called `fourbar`. Once this is done, member functions for specifying and analyzing the fourbar can be called. To correctly perform the analysis, however, all the necessary *setup* function(s) should be called prior to the *analysis* function(s). Otherwise, the results from the analysis may be incorrect. Default values for specifying a fourbar linkage are used in the calculations if the user does not specify them.

5 Application examples using Ch Mechanism Toolkit

In this section, three examples are presented to illustrate features and applications of the Ch Mechanism Toolkit.

5.1 Example 1

Problem statement: The link lengths of a fourbar linkage in Fig. 3 are given as follows: $r_1 = 12$ cm, $r_2 = 4$ cm, $r_3 = 12$ cm, and $r_4 = 7$ cm. The phase angle for the ground link is $\theta_1 = 10°$. The coupler point P is defined

```
/* header file(s) */
#include <fourbar.h>

/* main() function */
int main() {
    /* variables declaration */
    ...
    CFourbar fourbar;  // instantiate object class

    /* setup functions */
    setLinks();
    ...

    /* analysis function(s) */
    fourbar.angularPos();
    ...

    /* display results */
    printf();
    ...

    return 0;
}
```

Fig. 4 Program structure for using Ch Mechanism Toolkit

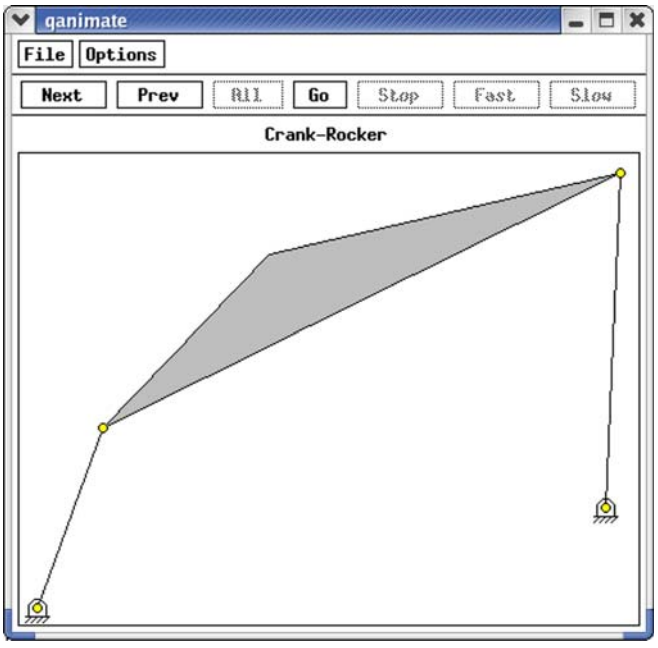

Fig. 5 The configuration for the first branch of the fourbar linkage

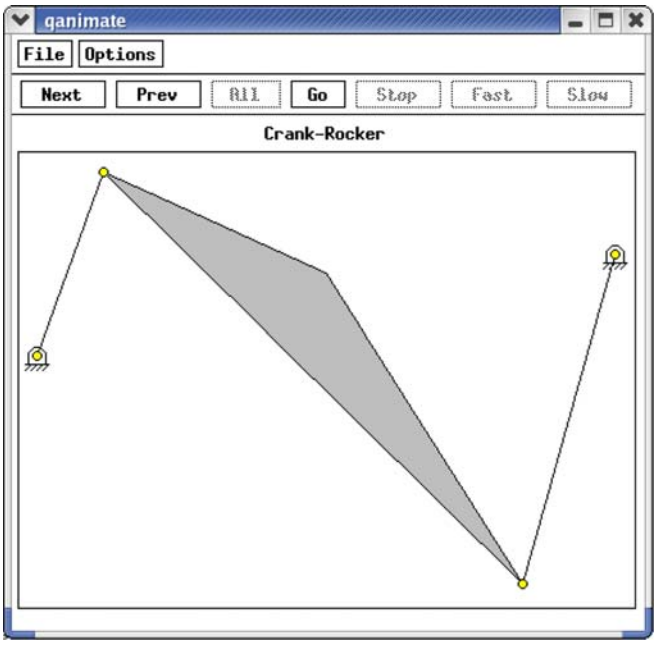

Fig. 7 The configuration for the second branch of the fourbar linkage

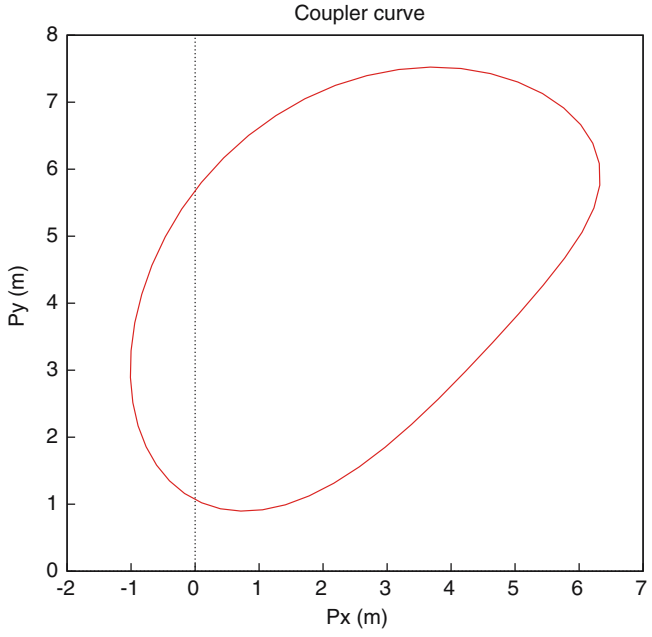

Fig. 6 The coupler curve for the first branch of the fourbar linkage

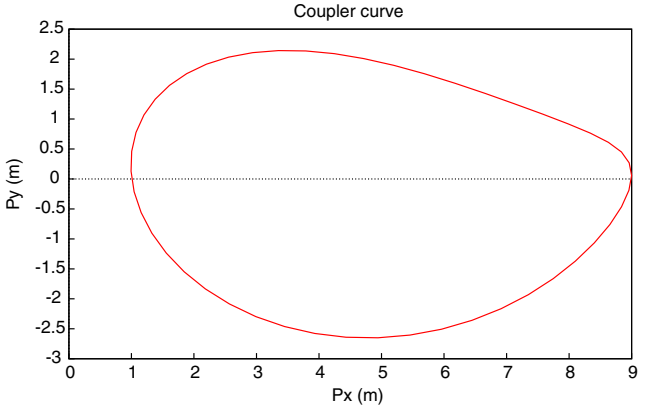

Fig. 8 The coupler curve for the second branch of the fourbar linkage

by the distance $r\_p = 5$ cm and constant angle $\beta = 20°$. Determine the angular positions θ_3 and θ_4 as well as the position for coupler point P when the input angle $\theta_2 = 70°$. Plot the coupler curve for the coupler point P when input link 2 is rotated from θ_{2min} to θ_{2max}. Also displays the current configuration of the fourbar linkage.

The above problem is solved with the Ch Mechanism Toolkit. Using the program structure shown in Fig. 4 as reference, Program 1 is a solution to the given problem. After instantiating an object of class **CFourbar** and declaring the necessary variables, member functions **setLinks()** and **setCouplerPoint()** are called to define the fourbar linkage with the given parameters. The desired results are then obtained through function calls to **angularPos()**, **couplerPointPos()**, **plotCouplerCurve()**, and **displayPosition()**. Figures 5 and 6 from Program 1 are the configuration and coupler curve for the first branch of the fourbar linkage, respectively. Figures 7 and 8 are the configuration and coupler curve for the second branch of the fourbar linkage, respectively. The numerical results of Program 1 are given below.

Program 1 Solution program for Example 1

```
1st set of solutions:
    theta3 =  0.459, theta4 =  1.527, P = complex( 4.822, 7.374)
2nd set of solutions:
    theta3 = -0.777, theta4 = -1.845, P = complex( 5.917, 1.684)
```

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

int main() {
   CFourbar fourbar;
   double r1 = 12, r2 = 4, r3 =12, r4 = 7, theta1 = 10*M_PI/180;
   double rp = 5, beta = 20*M_PI/180;
   double theta_1[1:4], theta_2[1:4];
   double complex p1, p2;  // tow solution of coupler point P
   double theta2 = 70*M_PI/180;
   CPlot plot1, plot2;

   theta_1[1] = theta1;
   theta_1[2] = theta2; // theta2
   theta_2[1] = theta1;
   theta_2[2] = theta2; // theta2
   fourbar.setLinks(r1, r2, r3, r4, theta1);
   fourbar.setCouplerPoint(rp, beta);
   fourbar.angularPos(theta_1, theta_2, FOURBAR_LINK2);
   fourbar.couplerPointPos(theta2, p1, p2);
   plot1.outputType(PLOT_OUTPUTTYPE_FILE, "postscript eps color", "couplercurve1.eps");
   fourbar.plotCouplerCurve(&plot1, 1);
   plot2.outputType(PLOT_OUTPUTTYPE_FILE, "postscript eps color", "couplercurve2.eps");
   fourbar.plotCouplerCurve(&plot2, 2);
   fourbar.displayPosition(theta_1[2], theta_1[3], theta_1[4]);
   fourbar.displayPosition(theta_2[2], theta_2[3], theta_2[4]);

   /**** the first set of solutions ****/
   printf("1st set of solutions: \n");
   printf("    theta3 = %6.3f, theta4 = %6.3f, P = %6.3f \n", theta_1[3], theta_1[4], p1);
   /**** the second set of solutions ****/
   printf("2nd set of solutions: \n");
   printf("    theta3 = %6.3f, theta4 = %6.3f, P = %6.3f \n", theta_2[3], theta_2[4], p2);
   return 0;
}
```

5.2 Example 2

Problem statement: The link lengths of a fourbar linkage in Fig. 3 are given as follows: $r_1 = 12$ cm, $r_2 = 4$ cm, $r_3 = 12$ cm, and $r_4 = 7$ cm. The phase angle for the ground link is $\theta_1 = 10°$, and the constant angular velocity of the input link is $\omega_2 = 5$ rad/sec. Plot the angular positions, velocities, and accelerations of links 3 and 4 with respect to time for the first branch.

Using the plotting features of the Ch Mechanism Toolkit, this problem is solved by Program 2. After links r_1 to r_4 and phase angle θ_1 have been set, member function **setAngularVel()** is called to specify the constant angular velocity ω_2. To indicate the number of data points to plot, member function **setNumPoints()** is required. Member functions **plotAngularPoss()**, **plotAngularVels()**, and **plotAngularAccels()** are then used to generate the desired results. Angular position, velocity, and acceleration plots for links 3 and 4 are shown in Figs. 9, 10, and 11. These plots allow users to better understand how the coupler and output links behave with respect to time and a constant angular velocity applied to the input link.

Another member function, **plotForceTorques()**, can be used to plot the individual joint forces and output torque of the fourbar linkage. This function is similar to the angular position, velocity, and acceleration plotting functions. However, it requires additional parameter values such as the mass and inertia properties of each link.

5.3 Example 3

Problem statement: Simulate the motion of the fourbar linkage defined in Example 1 for its entire range of motion.

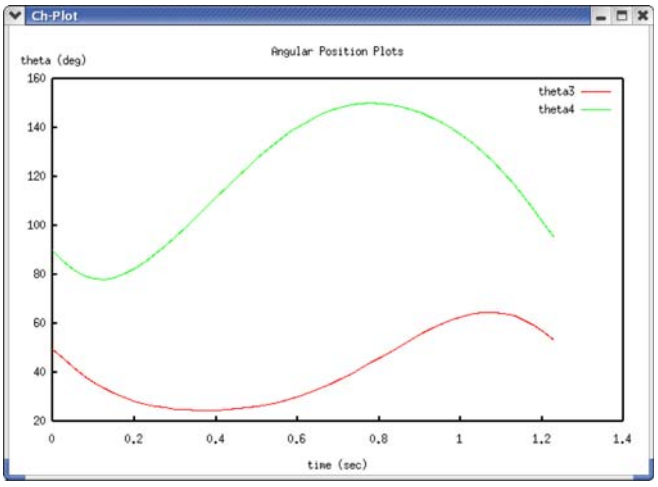

Fig. 9 Angular position plots

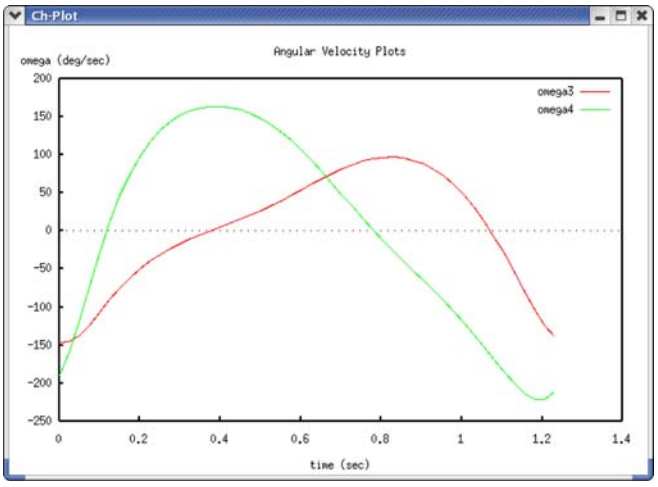

Fig. 10 Angular velocity plots

Program 2 Solution program for Example 2

```
#include <math.h>
#include <fourbar.h>

int main() {
    double r[1:4], theta1;
    double omega2;
    int numpoints = 50;
    CFourbar fourbar;
    CPlot plot1, plot2, plot3;

    /* default specification of the four-bar linkage */
    r[1] = 12; r[2] = 4; r[3] = 12; r[4] = 7;
    theta1 = 10*M_PI/180;
    omega2 = 5;  /*  rad/sec  */

    fourbar.setLinks(r[1], r[2], r[3], r[4], theta1);
    fourbar.setAngularVel(omega2);
    fourbar.setNumPoints(numpoints);
    fourbar.plotAngularPoss(&plot1, 1);
    fourbar.plotAngularVels(&plot2, 1);
    fourbar.plotAngularAccels(&plot3, 1);
    return 0;
}
```

Program 3 Solution program for Example 3

```
#include<fourbar.h>

int main() {
   CFourbar fourbar;
   double r1 = 12, r2 = 4, r3 =12, r4 = 7, theta1 = 10*M_PI/180;
   double rp = 5, beta = 20*M_PI/180;

   fourbar.setLinks(r1, r2, r3, r4, theta1);
   fourbar.setCouplerPoint(rp, beta, TRACE_ON);
   fourbar.animation(1);
   fourbar.animation(2);
   return 0;
}
```

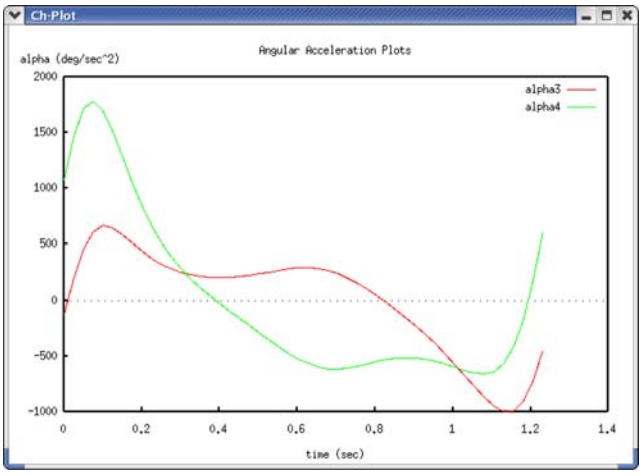

Fig. 11 Angular acceleration plots

Similar to the first example, this problem is easily solved with the Ch Mechanism Toolkit. Program 3 is used to create an animation of the fourbar. Notice that Program 3 is similar to Program 1. The primary difference is that Program 3 calls member function **animation()** instead of **angularPos()**, **couplerPointPos()**, and **displayPosition()** to obtain the desired output. Another difference is that there is an additional argument in member function **setCouplerPoint()**. This third argument, TRACE_ON, is a macro used to specify tracing of the coupler point. Note that the member function **animation()** contains a single integer argument. This number refers to the branch number of the fourbar linkage. Depending on its type, a fourbar linkage may have up to four branches. Since the fourbar defined in the problem statement is a Grashof crank-rocker, it has only two branches. Figure 12 shows one frame of animation for the first branch of the fourbar mechanism, whereas Fig. 13 is an overlay of all the frames of animation. Likewise, Fig. 14 is one frame of animation of the second branch, and Fig. 15 shows all of the animation frames.

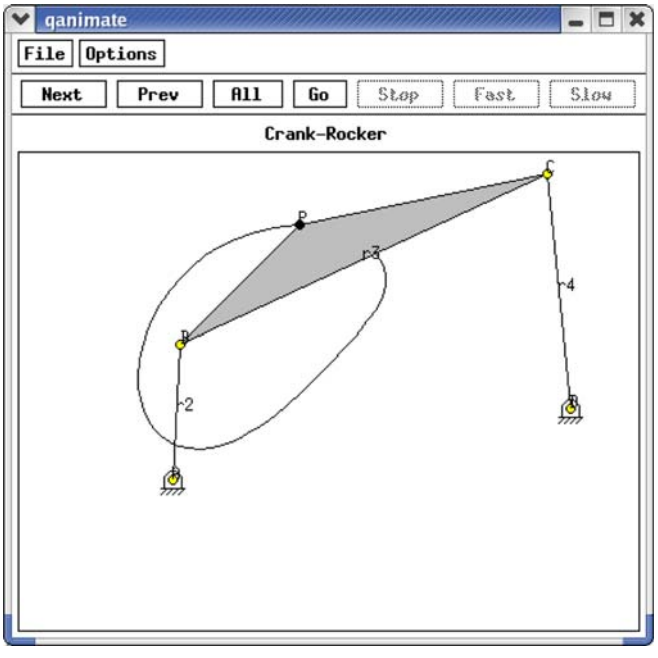

Fig. 12 Single frame of fourbar animation for the first branch

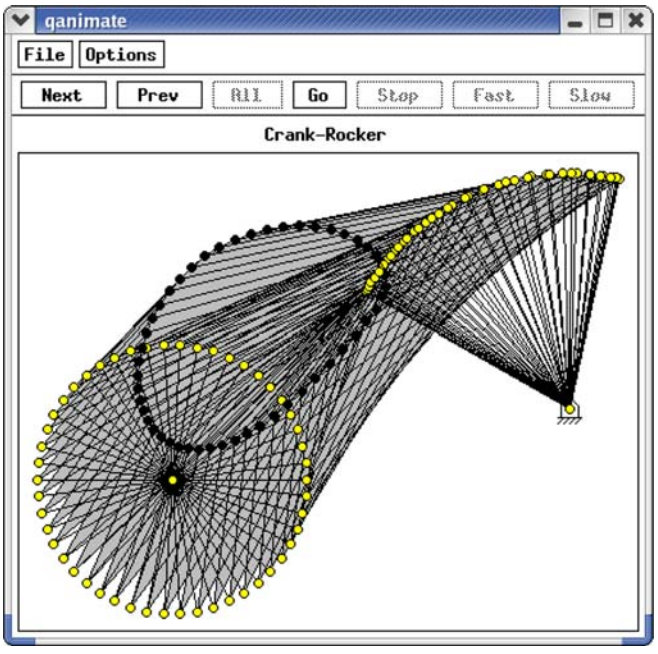

Fig. 13 All frames of fourbar animation for the first branch

6 Conclusion

An object-based mechanism toolkit has been developed. The toolkit consists of animation program QuickAnimation™ and a collection of classes for design and analysis of commonly used planar mechanisms. Written in Ch, a C/C++ interpreter, the Ch Mechanism Toolkit

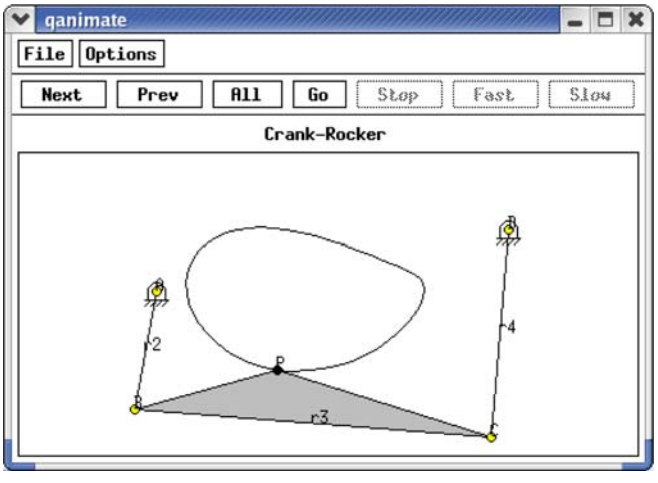

Fig. 14 Single frame of fourbar animation for the second branch

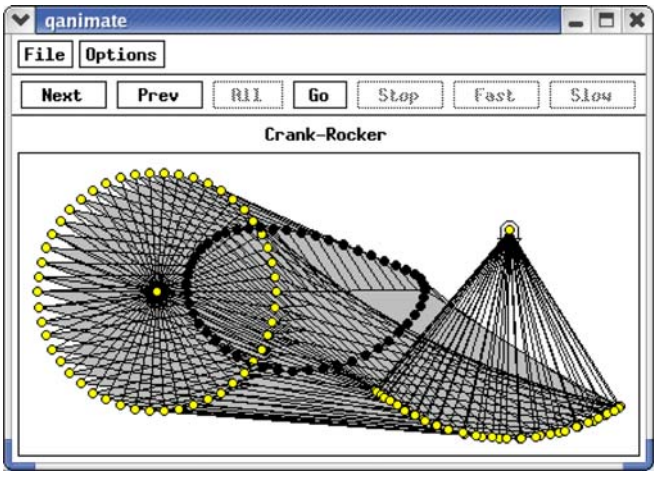

Fig. 15 All frames of fourbar animation for the second branch

is useful for solving practical engineering problems in design and analysis of mechanisms. It can compute the angular positions, velocities, and accelerations of the individual links of mechanisms such as the fourbar, slidercrank, geared-fivebar mechanisms, sixbar linkages, and cam-follower systems. Furthermore, graphical features such as high-level plotting and animation enhance the toolkit's applicability in engineering practice, and teaching and learning of mechanism design and analysis. The object-based design makes the Ch Mechanism Toolkit easy to extend and maintain. A Web-based mechanism design and analysis system based on this Ch Mechanism Toolkit has been developed. The ideas and concepts presented in the Ch Mechanism Toolkit can be applied to solve many other mechanism design and analysis problems.

References

1. Working Model User's Guide (1989) Knowledge revolution
2. Erdman AG, Gustafson JE (1981) Lincages: linkage interactive computer analysis and graphically enhanced synthesis package. ASME Paper, no. 77-DET-5
3. Erdman AG, Riley DR (1981) Computer-aided linkage design using the lincages package. ASME Paper, no. 81-DET-121
4. WATT 1.6 User's Guide (2002) Heron technologies. [Online]. Available at http://www.heron-technologies.com
5. SAM 5.0 User's Guide (2003) Artas—engineering software. [Online]. Available at http://www.artas.nl
6. Perez A, Su HJ, McCarthy M (2004) Synthetica 2.0: software for the synthesis of constrained serial chains. In: Proceedings of the ASME design engineering technical conferences, no. DETC2004/57524, Salt Lake City, September 2004
7. Autolev. Online Dynamics, Inc. [Online]. Available at http://www.autolev.com
8. Using Ch with Autolev to solve dynamics equations. [Online]. Available at http://iel.ucdavis.edu/projects/autolev
9. Ch Mechanism Toolkit, Softintegration, Inc. [Online]. Available at http://www.softintegration.com/products/toolkit/mechanism/
10. Cheng HH (1993) Scientific computing in the Ch programming language. Sci Program 2(3):49–75
11. — (2006) Ch: a C/C++ interpreter for script computing. C/C++ User's J 24(1):6–12
12. Ch—an Embeddable C/C++ Interpreter. [Online]. Available at http://www.softintegration.com
13. Zhu Y, Chen B, Cheng HH (2003) An object-based software package for interactive control system design and analysis. ASME Trans J Comput Inf Sci Eng 3(4):366–371
14. Yu Q, Chen B, Cheng HH (2004) Web-based control system design and analysis. IEEE Control Syst Mag 24(3):45–57
15. Cheng HH (1994) Pedagogically effective programming environment for teaching mechanism design. Comput Appl Eng Educ 2(1):23–39
16. Cheng HH, Campbell M (2005) Effective teaching of computer integrated mechanism analysis and design. In: Proceedings of the ASME 29th mechanism and robotics conference, no. DETC2005-85565, Long Beach, California, September 2005
17. Design and Analysis of Whitworth Quick Return Mechanism. [Online]. Available at http://iel.ucdavis.edu/projects/mechanism/quickreturn
18. Cheng HH, Trang DT (2004) Web-based mechanism design and analysis. In: Proceedings of the ASME 28th mechanism and robotics conference, no. DETC2004-57594, Salt Lake City, Utah, September 2004
19. Web-Based Mechanism Design and Analysis, Softintegration, Inc. [Online]. Available at http://softintegration.com/webservices/mechanism/
20. Pennestri E. Kinematic Synthesis of Mechanisms. [Online]. Available at http://www.ingegneriameccanica.org/mechanisms.htm
21. Embedded Ch, Softintegration, Inc. [Online]. Available at http://www.softintegration.com/products/sdk/embedded_ch/