

# Software Lab Computational Engineering Science

Group 12, Pusher Mechanism

Aaron Floerke, Arseniy Kholod, Xinyang Song and Yanliang Zhu

Informatik 12: Software and Tools for Computational Engineering (STCE)  
RWTH Aachen University

# Contents

## Preface

## Analysis

- User Requirements

- System Requirements

- Theory Base

## Design

- Class Model(s)

## Implementation

- Backend

- Software Tests

- Frontend

## Results

- 27 movement types

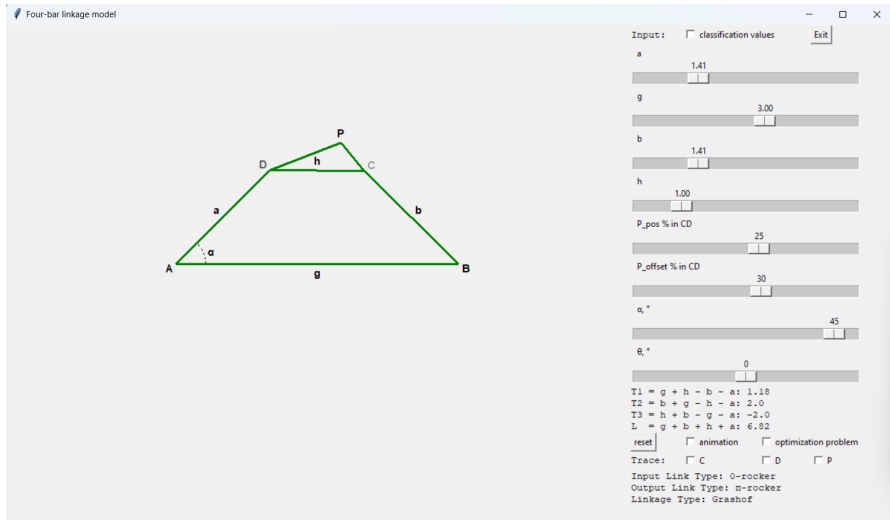
- Optimization problem

## Live Software Demo

## Documentation

## Project Management

## Summary and Conclusion



- Implement 27 motion types of the four-bar linkage with one bar fixed:

- Classification values:

- $T_1 = g + h - b - a$

- $T_2 = b + g - h - a$

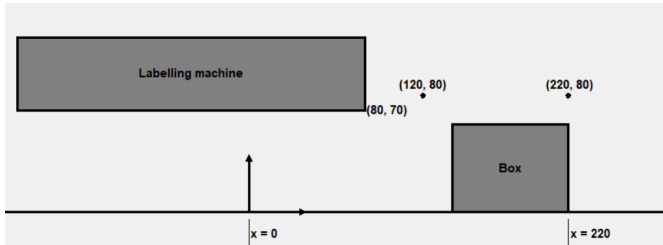
- $T_3 = h + b - g - a$

- Implement GUI with motion animation and the ability to choose geometrical parameters:

- Length of the bars
- Position of the coupler
- Input angle
- Angle relative to the horizon
- Classification values as alternative input

No.	$T_1$	$T_2$	$T_3$	$T_1 T_2$	$T_1 T_3$	$a$	$b$
1	+	+	+	+	+	crank	rocker
2	0	+	+	0	0	crank	$\pi$ -rocker
3	-	+	+	-	-	$\pi$ -rocker	$\pi$ -rocker
4	+	0	+	0	+	crank	0-rocker
5	0	0	+	0	0	crank	crank
6	-	0	+	0	-	crank	crank
7	+	-	+	-	+	$\pi$ -rocker	0-rocker
8	0	-	+	0	0	crank	crank
9	-	-	+	+	-	crank	crank
10	+	+	0	+	0	crank	$\pi$ -rocker
11	0	+	0	0	0	crank	$\pi$ -rocker
12	-	+	0	-	0	$\pi$ -rocker	$\pi$ -rocker
13	+	0	0	0	0	crank	crank
14	0	0	0	0	0	crank	crank
15	-	0	0	0	0	crank	crank
16	+	-	0	-	0	$\pi$ -rocker	crank
17	0	-	0	0	0	crank	crank
18	-	-	0	+	0	crank	crank
19	+	+	-	+	-	0-rocker	$\pi$ -rocker
20	0	+	-	0	0	0-rocker	$\pi$ -rocker
21	-	+	-	-	+	rocker	rocker
22	+	0	-	0	-	0-rocker	crank
23	0	0	-	0	0	0-rocker	crank
24	-	0	-	0	+	0-rocker	0-rocker
25	+	-	-	-	-	rocker	crank
26	0	-	-	0	0	0-rocker	crank
27	-	-	-	+	+	0-rocker	0-rocker

Figure from "Classification, geometrical and kinematic analysis of four-bar linkages" 10.15308/Sinteza-2018-261-266 by Ivana Cvetkovic et al.



- ▶ Solve an optimization problem:
  - ▶ Push box with size  $80 \times 60$  from  $x = 220$  to  $x = 0$
  - ▶ Do not cross the area of the labelling machine (Area with  $x < 80$  and  $y > 70$ ).
  - ▶ Pass above points  $(120, 80)$  and  $(220, 80)$

### ► **Four-bar linkage model:**

- System simulates all the motion types of the four-bar linkage.
- System does not crash with any input of geometrical configuration.

### ► **Tests:**

- Implement test cases for geometry.
- Implement test cases with bad input to test system stability.

### ► **Graphical User Interface:**

- GUI provides the four-bar linkage visualization and motion animation.
- User can input geometrical data by moving a point on a slide bar.
- GUI is coupled with the four-bar linkage model to use implemented motion cases for animation.
- GUI provides tracing for trajectories of the points.
- GUI classifies of the linkage.

### ► **Optimization problem:**

- It should be possible to find a solution (manually) for the optimization problem using the four-bar linkage model.
- GUI visualizes the solution.

### ► **Performance:**

- The four-bar linkage model is fast enough to provide smooth GUI animations.
- GUI animations are not slower than 30 frames per second.

### ► **Usability:**

- Every essential part of the four-bar linkage model is well documented.
- GUI is easy to operate and all functionalities are self-explanatory.
- GUI source code is well documented.

- $s$  = length of shortest bar
- $l$  = length of longest bar
- $p, q$  = lengths of intermediate bar

**Grashof's theorem** states that a four-bar mechanism has *at least* one revolving link if

$$s + l \leq p + q \quad (5-1)$$

and all three mobile links will rock if


$$s + l > p + q \quad (5-2)$$

The inequality 5-1 is **Grashof's criterion**.

<https://www.cs.cmu.edu/~rapidproto/mechanisms/chpt5.html>




- ▶ The right figure is our kinematic analysis of the four-bar linkage.
- ▶ Ultimately, we chose the geometric simulation as it is more convenient to implement.



(1)  $\vec{AD} + \vec{DC} = \vec{AB} + \vec{BC}$   
 $\vec{AD} \Rightarrow$  input linkage  $\Rightarrow \psi_1'$  known as constant  $\Rightarrow \psi_1'' = 0$   
 angle to x-axis is positive when anti-clockwise

(2) static analysis

Attention: All initialization,  $AD$ ,  $DC$ ,  $BC$ ,  $AB$  and  $\psi_1$  are already known



$\psi_1$  negative

(2)  $AD \cos \psi_1 + DC \cos \psi_2 = BC \cos \psi_3 + AB$  x-axis  
 (3)  $AD \sin \psi_1 + DC \sin \psi_2 = BC \sin \psi_3$  y-axis

$\psi_1 \Rightarrow$  known, can be seen as constant

	$\psi_1$	$\psi_2$
(2)	x	x
(3)	x	x

$\Rightarrow \psi_2, \psi_3 \checkmark$

(3) derivate (2) (3) with  $t \Rightarrow$  angular velocity

(4)  $-DC \sin \psi_2 \cdot \dot{\psi}_2 + BC \sin \psi_3 \cdot \dot{\psi}_3 = AD \sin \psi_1 \cdot \dot{\psi}_1$   
 (5)  $DC \cos \psi_2 \cdot \dot{\psi}_2 - BC \cos \psi_3 \cdot \dot{\psi}_3 = AD \cos \psi_1 \cdot \dot{\psi}_1$

written as matrix form  $\Rightarrow$  for calculating in computer

$[A] \cdot [\dot{\psi}] = [B] \cdot [\dot{q}]$

$[A] = \begin{bmatrix} -DC \sin \psi_2 & BC \sin \psi_3 \\ DC \cos \psi_2 & -BC \cos \psi_3 \end{bmatrix}$   $[\dot{\psi}] = \begin{bmatrix} \dot{\psi}_2 \\ \dot{\psi}_3 \end{bmatrix}^T$   
 $[B] = \begin{bmatrix} AD \sin \psi_1 & 0 \\ 0 & -AD \cos \psi_1 \end{bmatrix}$   $[\dot{q}] = \begin{bmatrix} \dot{\psi}_1 \\ \dot{\psi}_1 \end{bmatrix}^T$

$\Rightarrow$  Function (4), (5)  $\Rightarrow$  2 Function  
 2 parameter  $\Rightarrow$  solve  $\dot{\psi}_2$  and  $\dot{\psi}_3$

### ▶ 1. Operating System:

- ▶ Xubuntu/Windows

### ▶ 2. Developing Environment:

- ▶ Programming Language: Python.
- ▶ IDE: Spyder/Pycharm.
- ▶ Package Manager: Anaconda.

### ▶ 3. Libraries:

- ▶ Frontend: tkinter, math, numpy
- ▶ Backend: math, numpy

### ▶ 4. Version Control System:

- ▶ GitHub: Remote code repositories for team collaboration, code reviews, and version control.  
[https://github.com/einsflash/Project\\_Pusher\\_Mechanism](https://github.com/einsflash/Project_Pusher_Mechanism)

### ▶ 5. Frameworks:

- ▶ Pdoc: Used for generating project documentation, helping the team understand and maintain the code better.

### API Documentation

class GUI  
linkage  
tk  
width  
height  
model\_frame  
toolbar\_frame  
trace\_C()  
trace\_D()  
trace\_P()  
positions\_C  
positions\_D  
positions\_P  
x\_axis  
y\_axis  
A\_x  
A\_y  
pin\_box\_to\_coupler  
prev\_coupler\_position  
prev\_box\_position  
init\_toolbar()  
init\_linkage\_display()  
display\_classification\_values()  
display\_bars\_values()  
display\_information()  
scaling\_factor()  
calculate\_normalities()  
update\_parameter\_a()  
update\_parameter\_g()  
update\_parameter\_b()  
update\_parameter\_h()  
update\_parameter\_p\_pos()  
update\_parameter\_p\_off()  
update\_parameter\_alpha()

### gui

```
# class GUI:
    linkage
    tk
    width
    height
    model_frame
    toolbar_frame
    trace_C()
    trace_D()
    trace_P()
    positions_C
    positions_D
    positions_P
    x_axis
    y_axis
    A_x
    A_y
    pin_box_to_coupler
    prev_coupler_position
    prev_box_position
    init_toolbar()
    init_linkage_display()
    display_classification_values()
    display_bars_values()
    display_information()
    scaling_factor()
    calculate_normalities()
    update_parameter_a()
    update_parameter_g()
    update_parameter_b()
    update_parameter_h()
    update_parameter_p_pos()
    update_parameter_p_off()
    update_parameter_alpha()
    def trace_C(self):
    def trace_D(self):
    def trace_P(self):
    positions_C
    positions_D
    positions_P
    x_axis
    y_axis
    A_x
    A_y
    pin_box_to_coupler
    prev_coupler_position
    prev_box_position
    def init_toolbar(self):
    def init_linkage_display(self):
```

### API Documentation

class FourBarLinkage  
FourBarLinkage()  
AB  
BC  
CD  
DA  
alpha  
theta  
alpha\_rad  
theta\_rad  
coupler\_position  
coupler\_offset  
t  
alpha\_velocity  
C\_mode  
init\_default\_values()  
run()  
check\_Parameter()  
find\_Linkage\_Type()  
calculate\_Classification\_Value()  
calculate\_Edge\_Value()  
calculate\_alpha\_lim()  
calculate\_Point\_Position()  
calculate\_C\_Position()  
calculate\_P\_Position()  
animation\_alpha()  
switch\_C2\_C1()

built with pdoc

### four\_bar\_linkage

class FourBarLinkage:

```
FourBarLinkage(
    AB,
    BC,
    CD,
    DA,
    alpha,
    theta,
    coupler_position,
    coupler_offset,
    timeinterval,
    alpha_velocity
)
```

```
AB
BC
CD
DA
alpha
theta
alpha_rad
theta_rad
```

```
coupler_position
coupler_offset
```

```
t
alpha_velocity
```

```
C_mode
```

```
def init_default_values(self):
```

```
def run(self):
```

## a. Elements in Class

### ▶ 1. Input Parameters:

- ▶ AB, BC, CD, DA
- ▶ alpha, theta, alpha\_rad, theta\_rad
- ▶ coupler\_position, coupler\_offset
- ▶ t, alpha\_velocity, C\_mode

### ▶ 2. Animation-Related Attributes:

- ▶ switch\_C2\_C1\_180, switch\_C2\_C1\_360
- ▶ C2\_C1\_switched\_last\_time, direction

### ▶ 3. Geometry Validity & Type Check:

- ▶ Linkage\_Type, geometric Validity
- ▶ Input\_Link\_Type, Output\_Link\_Type

### ▶ 4. Angle Limits:

- ▶ alpha\_lims, alpha\_rad\_lims, alpha\_limited

### ▶ 5. Position of Points:

- ▶ A, B, C, C1, C2, D, P

### ▶ 6. Classification Values:

- ▶ T1, T2, T3, L

## b. Two Modes for User Parameters

- ▶ `calculate_Classification_Value(self)`
- ▶ `calculate_Edge_Value(self)`

## c. Display Linkage Motion Type

- ▶ `find_Linkage_Type(self)`
  - ▶ Use a Python dictionary to store the type data.

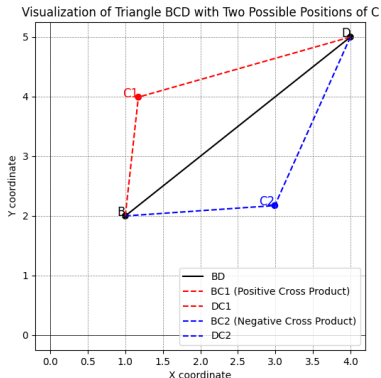
## d. Check Parameter

- ▶ `check_Parameter(self)`
  - ▶ Check linkage type.
  - ▶ `geometric_Verify`

## e. Update Parameters Every Run

- ▶ `run(self)`
  - ▶ `self.calculate_Classification_Value()`
  - ▶ `self.check_Parameter()`
  - ▶ If `geometric_Verify` is False, exit.
  - ▶ `self.find_Linkage_Type()`
  - ▶ `self.calculate_alpha_lims()`
  - ▶ `self.calculate_Point_Position()`

- ▶ A and B are fixed points.
- ▶ D can be determined using angle  $\alpha$  and length AD.
- ▶ Given positions of B and D, and all side lengths of triangle BCD, point C has two possible locations  $C_1$  and  $C_2$ .
- ▶ Calculating point  $C_1$  and  $C_2$ .
- ▶ Choose C as  $C_2$  per default and switch to  $C_1$  if needed.



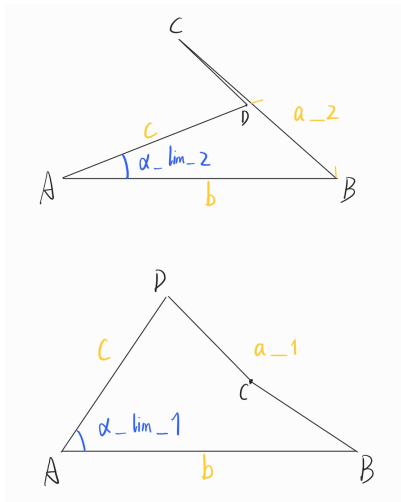
Picture from "test for calculating point C.py"

► **Cosine Law Formula:**

►  $\cos(\alpha_{\text{lims1}}) = \frac{b^2 + c^2 - a_1^2}{2bc}$

►  $\cos(\alpha_{\text{lims2}}) = \frac{b^2 + c^2 - a_2^2}{2bc}$

- Determine whether to switch between  $C_1$  and  $C_2$  to ensure animation continuity:
- Switch when  $\alpha$  reaches its limits.
  - Switch in special cases, when  $\alpha$  limits are exactly  $0^\circ$ ,  $360^\circ$  and/or  $180^\circ$ .



## ► Basic Concept:

- Change  $\alpha$  according to the defined limits.
- Reverse direction at boundaries.
- Switch between configurations ( $C_1$  and  $C_2$ ) to ensure continuity.

## ► Direction Control:

- $\text{direction} = 0$ : *Increasing  $\alpha$*
- $\text{direction} = 1$ : *Decreasing  $\alpha$*

## ► Updating $\alpha$ :

- Update by  $\alpha_{\text{velocity}} * t$ .
- Reverse direction when reaching limit values.

## ► Switching Between $C_1$ and $C_2$ :

- Switch at  $\alpha$  limits.
- Switch at  $0^\circ$ ,  $360^\circ$  and/or  $180^\circ$  if it corresponds to  $\alpha$  limits.
- Handle floating-point precision issues ( $10^{-12}$ ).

## ► Configuration Tracking:

- Avoid redundant switching between  $C_1$  and  $C_2$ .



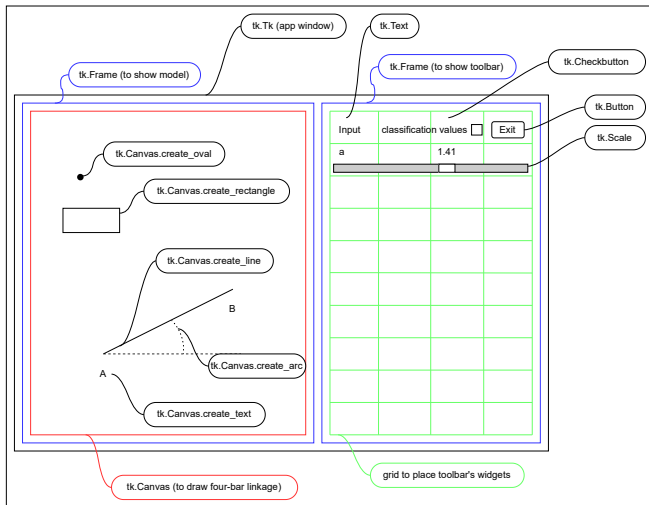
```

# Update alpha based on current direction
if self.direction == 0: # Increasing alpha
    self.alpha += self.alpha_velocity * self.t
    self.alpha_rad = math.radians(self.alpha)

# Check if alpha exceeds the upper limit
if self.alpha_limited and self.alpha >= self.alpha_lims[1]:
    # Set alpha to the upper limit
    self.alpha = self.alpha_lims[1]
    self.alpha_rad = self.alpha_rad_lims[1]
    # Switch direction to decreasing
    self.direction = 1
    # switch C1 and C2 if didnt switch last time
    if not self.C2_C1_switched_last_time:
        self.switch_C2_C1()
        self.C2_C1_switched_last_time = True

# switch by 0 and 360 degrees if needed
if self.switch_C2_C1_360 and ((self.alpha - self.theta >= -10**-12 and \
    self.alpha - self.theta - self.alpha_velocity * self.t <= 10**-12) or \
    (self.alpha - self.theta >= 360.0 and \
    self.alpha - self.theta - self.alpha_velocity * self.t <= 360.0)):
    # switch C1 and C2 if didnt switch last time
    if not self.C2_C1_switched_last_time:
        self.switch_C2_C1()
        self.C2_C1_switched_last_time = True
# same for switch by 180 degrees if needed
# alpha is not limited values have to stay from 0 to 360
if not self.alpha_limited and self.alpha >= 360.0:
    self.alpha = self.alpha - 360.0
    self.alpha_rad = math.radians(self.alpha)
  
```

- ▶ Tests classify motion of four-bar linkage based on values  $T_1$ ,  $T_2$ , and  $T_3$
- ▶ Verifies correct behavior for Crank, Rocker, and intermediate states
- ▶ Covers all possible combinations of positive, negative, and zero values
- ▶ Ensures accurate classification of input and output links
- ▶ Test cases include Crank-Rocker, Double Crank, Double Rocker, and Rocker-Crank scenarios
- ▶ Each test checks specific link motion configurations
- ▶ Automated with `unittest` framework for reproducibility and consistency



- Initiate all tkinter objects inside GUI class and generate app window:  
`GUI().tk.mainloop()`

- ▶ To display different modes, some objects have to be hidden or shown.
- ▶ For objects in tk.Canvas use itemconfigure:
  - ▶ Hide:  
`self.model_animation.itemconfigure(self.model_animation.AB_line, state='hidden')`
  - ▶ Show:  
`self.model_animation.itemconfigure(self.model_animation.AB_line, state='normal')`
- ▶ For widgets like tk.Scale or tk.Text:
  - ▶ Hide: `self.slider_T1.grid_remove()`
  - ▶ Show: `self.slider_T1.grid()`

- In Backend check if parameters are valid:  $V = l - s - p - q \leq 0$
- If  $V > 0$ :

Four-bar linkage model

Invalid setup, change geometrical values

Input: ☐ classification values

a 1.51

g 4.01

b 1.41

h 1.00

P\_pos % in CD 25

P\_offset % in CD 30

$\alpha$ , ° 0

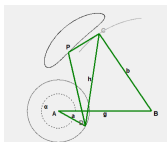
$\theta$ , ° 0

T1 = g + h - b - a: 2.09  
T2 = b + g - h - a: 2.91  
T3 = h + b - g - a: -3.11  
L = g + b + h + a: 7.93

☐ animation ☐ optimization problem

Trace: ☐ C ☐ D ☐ P

Input Link Type: 0-rocker  
Output Link Type: n-rocker  
Linkage Type: Grashof



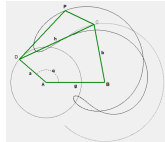
$$T_{1,2,3} = 1.0, 1.0, 1.0$$



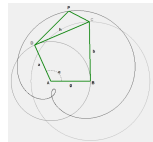
$$T_{1,2,3} = 0.0, 1.0, 1.0$$



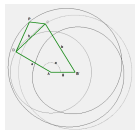
$$T_{1,2,3} = -1.0, 1.0, 1.0$$



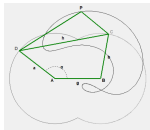
$$T_{1,2,3} = 1.0, 0.0, 1.0$$



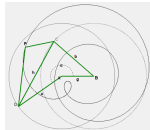
$$T_{1,2,3} = 0.0, 0.0, 1.0$$



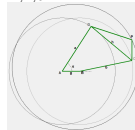
$$T_{1,2,3} = -1.0, 0.0, 1.0$$



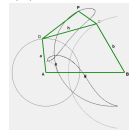
$$T_{1,2,3} = 1.0, -1.0, 1.0$$



$$T_{1,2,3} = 0.0, -1.0, 1.0$$



$$T_{1,2,3} = -1.0, -1.0, 1.0$$



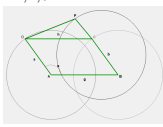
$$T_{1,2,3} = 1.0, 1.0, 0.0$$



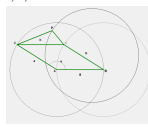
$$T_{1,2,3} = 0.0, 1.0, 0.0$$



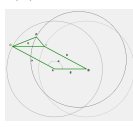
$$T_{1,2,3} = -1.0, 1.0, 0.0$$



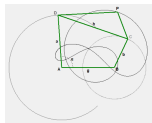
$$T_{1,2,3} = 1.0, 0.0, 0.0$$



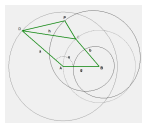
$$T_{1,2,3} = 0.0, 0.0, 0.0$$



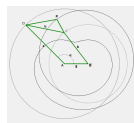
$$T_{1,2,3} = -1.0, 0.0, 0.0$$



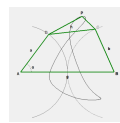
$$T_{1,2,3} = 1.0, -1.0, 0.0$$



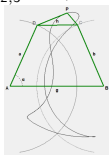
$$T_{1,2,3} = 0.0, -1.0, 0.0$$



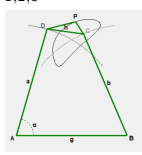
$$T_{1,2,3} = -1.0, -1.0, 0.0$$



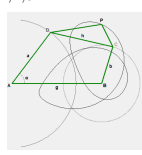
$$T_{1,2,3} = 1.0, 1.0, -1.0$$



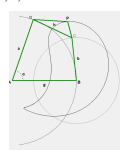
$$T_{1,2,3} = 0.0, 1.0, -1.0$$



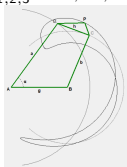
$$T_{1,2,3} = -1.0, 1.0, -1.0$$



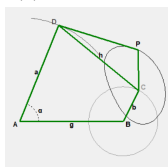
$$T_{1,2,3} = 1.0, 0.0, -1.0$$



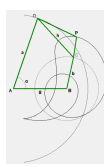
$$T_{1,2,3} = 0.0, 0.0, -1.0$$



$$T_{1,2,3} = -1.0, 0.0, -1.0$$



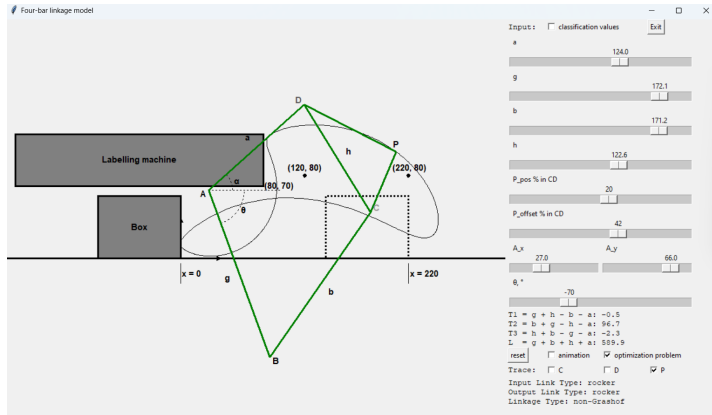
$$T_{1,2,3} = 1.0, -1.0, -1.0$$



$$T_{1,2,3} = 0.0, -1.0, -1.0$$



$$T_{1,2,3} = -1.0, -1.0, -1.0$$



- ▶ 9 degrees of freedom (all lengths in cm):
  - ▶ Length of four bars:  $a = 124.0$ ,  $b = 171.2$ ,  $g = 172.1$ ,  $h = 122.6$ .
  - ▶ Coupler position:  $P_{pos} = 20.0\%$ ,  $P_{offset} = 42.0\%$  of  $h$ .
  - ▶ Position of point A:  $A_x = 27.0$ ,  $A_y = 66.0$ .
  - ▶ Angle of ground bar relative to horizon:  $\theta = -70.0^\circ$



1. Changing the input of sidebar.
2. Start the animation.
3. Test different motion types.
4. Enable points tracing.
5. Solve the optimization problem.

## Comprehensive Documentation of the FourBarLinkage Class in Python

October 15, 2024

### 1 Introduction

The `FourBarLinkage` class in Python models a four-bar linkage mechanism, commonly used in mechanical systems. A four-bar linkage consists of four rigid bars connected by rotary joints, and depending on the bar lengths and angles, it can exhibit different motion types like crank-rocker, double-rocker, or double-crank. This class allows for the simulation of the mechanism and provides users the ability to input parameters such as link lengths, initial angles, and angular velocities.

#### User Inputs

Users can directly input the following parameters:

- `AB, BC, CD, DA`: Lengths of the four bars in the linkage.
- `alpha, theta`: Angles of the input and fixed links (in degrees).
- `alpha.velocity`: Angular velocity of the input link (in degrees per second).

## Unit Test Documentation for Four-Bar Linkage

October 13, 2024

### Overview of the Test Collection

The unit tests for the Four-Bar Linkage system are designed to verify the correct classification of linkage motion types. Each test checks whether the linkage is correctly identified as a specific combination of crank, rocker, and intermediate states based on the values of  $T_1$ ,  $T_2$ , and  $T_3$ . These tests ensure that for various configurations of the four-bar linkage, the input and output links are classified accurately. The classifications are important for understanding the behavior of the system and ensuring correct functionality in different scenarios.

The test cases systematically cover all possible combinations of positive, negative, and zero values for the parameters  $T_1$ ,  $T_2$ , and  $T_3$ , ensuring that every possible motion configuration is tested. The expected outcomes are predefined according to the known behavior of four-bar linkages.

### Explanation of Test Collection

Each collection of test cases verifies a specific motion classification of the four-bar linkage system. The classifications generally fall into the categories of Crank, Rocker, and intermediate states like *n-rocker* or *0-rocker*.

#### Crank-Rocker Classifications

The tests in this group verify the combinations where one of the links acts as a crank while the other behaves as a rocker.

### 4 GUI Implementation

The `FourBarLinkage` class is integrated with a graphical user interface (GUI), allowing users to interact with the linkage system dynamically. Through the GUI, users can adjust parameters like link lengths and angles in real-time and observe the system's behavior.

#### 4.1 `init_linkage_display()` - Initialize Linkage Display

- **Location:** In the GUI class.
- **Description:** Sets up the visual elements needed to display the linkage on the canvas, such as drawing the bars and marking the points A, B, C, D, and P.

#### 4.2 `refresh()` - Refresh GUI

- **Location:** In the GUI class.
- **Description:** Updates the visual representation of the linkage every time a user changes the parameters, ensuring the display remains synchronized with the system's current state.

#### 4.3 `run_animation()` - Run Animation

- **Location:** In the GUI class.
- **Description:** Runs the continuous animation of the four-bar linkage mechanism on the canvas. The points and bars update in real-time as the system moves, based on the input parameters.

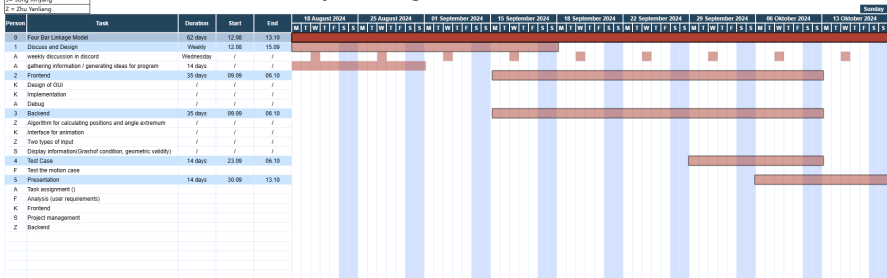
- ▶ **1. Discuss and Design:**
  - ▶ weekly discussion in discord.
  - ▶ gathering information / generating ideas for program.
- ▶ **2. Frontend:**
  - ▶ Design of GUI
  - ▶ Implementation
  - ▶ Debug
- ▶ **3. Backend:**
  - ▶ Algorithm for calculating positions and angle extremum
  - ▶ Interface for animation
  - ▶ Two types of input
  - ▶ Display information(Grashof condition, geometric validity)
- ▶ **4. Test the motion case:**
- ▶ **5. Presentation:**
  - ▶ Analysis (user requirements)
  - ▶ Frontend
  - ▶ Project management
  - ▶ Backend
- ▶ \*The following page outlines the responsibilities of each person.

# Project Management

## Gantt Chart

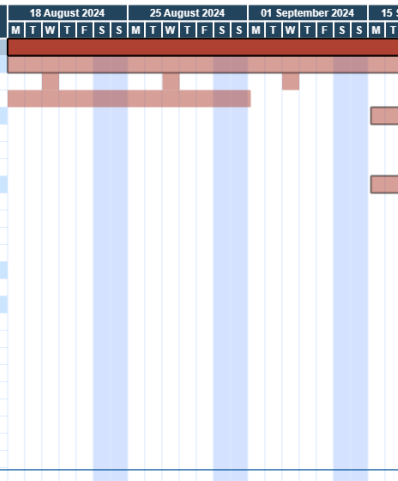
A = ALL members in group
K = Khalid Arseniy
F = Floerke Aaron Albert
S= Song Xinyang
Z = Zhu Yanliang

### Project Management



Person	Task	Duration	Start	End
0	Four Bar Linkage Model	62 days	12.08	13.10
1	Discuss and Design	Weekly	12.08	15.09
A	weekly discussion in discord	Wednesday	/	/
A	gathering information / generating ideas for program	14 days	/	/
2	Frontend	35 days	09.09	06.10
K	Design of GUI	/	/	/
K	Implementation	/	/	/
A	Debug	/	/	/
3	Backend	35 days	09.09	06.10
Z	Algorithm for calculating positions and angle extremum	/	/	/
K	Interface for animation	/	/	/
Z	Two types of input	/	/	/
S	Display information(Grashof condition, geometric validity)	/	/	/
4	Test Case	14 days	23.09	06.10
F	Test the motion case			
5	Presentation	14 days	30.09	13.10
A	Task assignment ()			
F	Analysis (user requirements)			
K	Frontend			
S	Project management			
Z	Backend			

# Project Management



# Summary and Conclusion

- ▶ Analysis:
  - ▶ User requirements
  - ▶ System requirement
  - ▶ Theory base
- ▶ Design:
  - ▶ Development infrastructure: Python and Tkinter
  - ▶ Class models
- ▶ Implementation:
  - ▶ Backend: four bar linkage geometry
  - ▶ Frontend: GUI
  - ▶ Unittest
- ▶ Results:
  - ▶ 27 movement cases
  - ▶ Optimization problem solution
- ▶ Documentation
- ▶ Project management: Gantt Chart.

- ▶ Cvetkovic, Ivana and Stojicevic, Misa and Popkonstantinović, Branislav and Cvetković, Dragan. (2018). Classification, geometrical and kinematic analysis of four-bar linkages. 261-266. 10.15308/Sinteza-2018-261-266.