Software and Tools for Computational Engineering

RWTH AACHEN UNIVERSITY

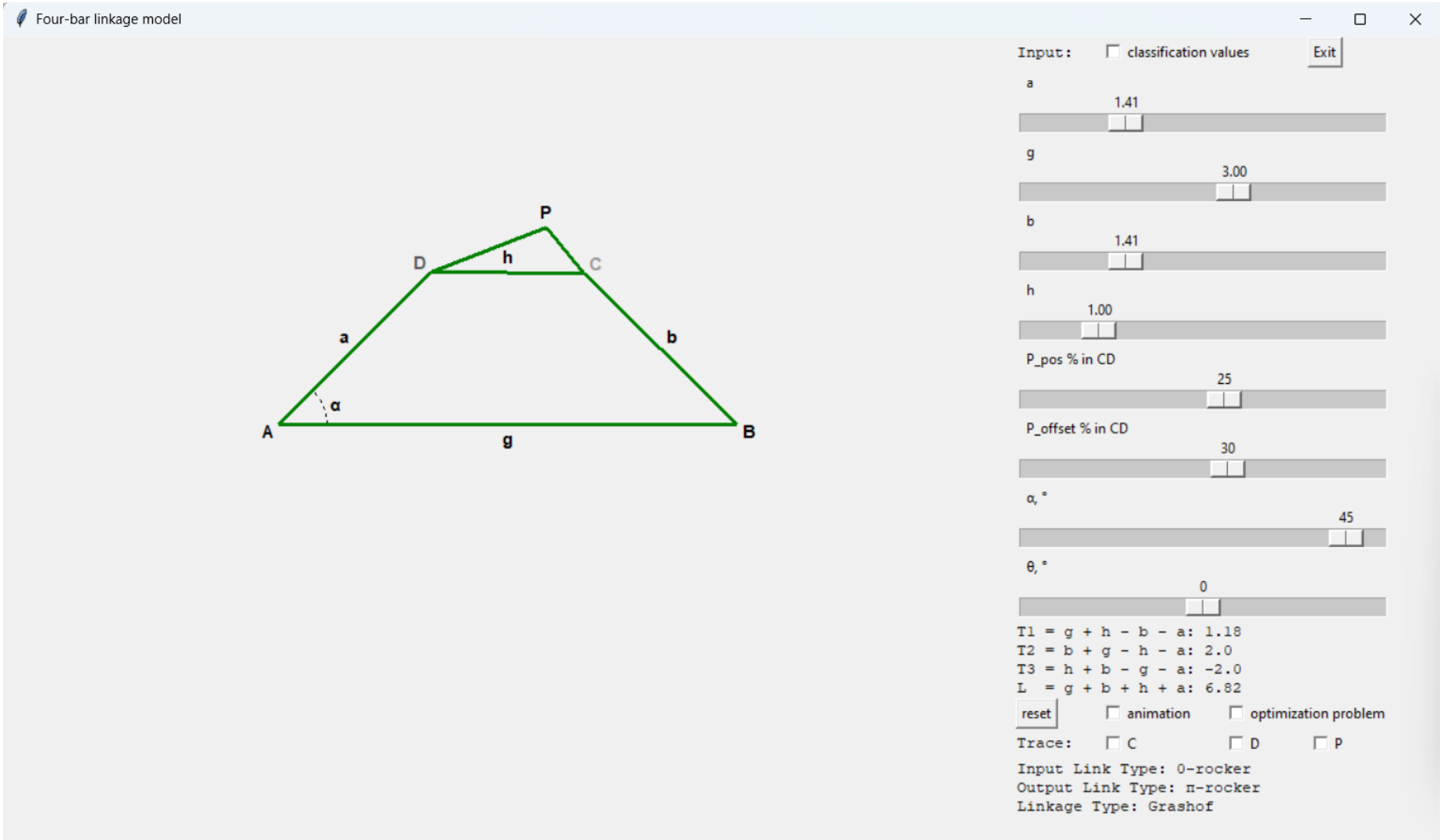
Software Lab Computational Engineering Science

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


Four-bar linkage model
P
D
h
C
a
b
α
A
g
B
Input:
classification values
Exit
a
1.41
g
3.00
b
1.41
h
1.00
P_pos % in CD
25
P_offset % in CD
30
α, °
45
θ, °
0
T1 = g + h - b - a: 1.18
T2 = b + g - h - a: 2.0
T3 = h + b - g - a: -2.0
L = g + b + h + a: 6.82
reset
animation
optimization problem
Trace:
C
D
P
Input Link Type: 0-rocker
Output Link Type: π-rocker
Linkage Type: Grashof

- 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

$$s + l <= p + q \tag{5-1}$$

and all three mobile links will rock if

$$s + l > p + q \tag{5-2}$$

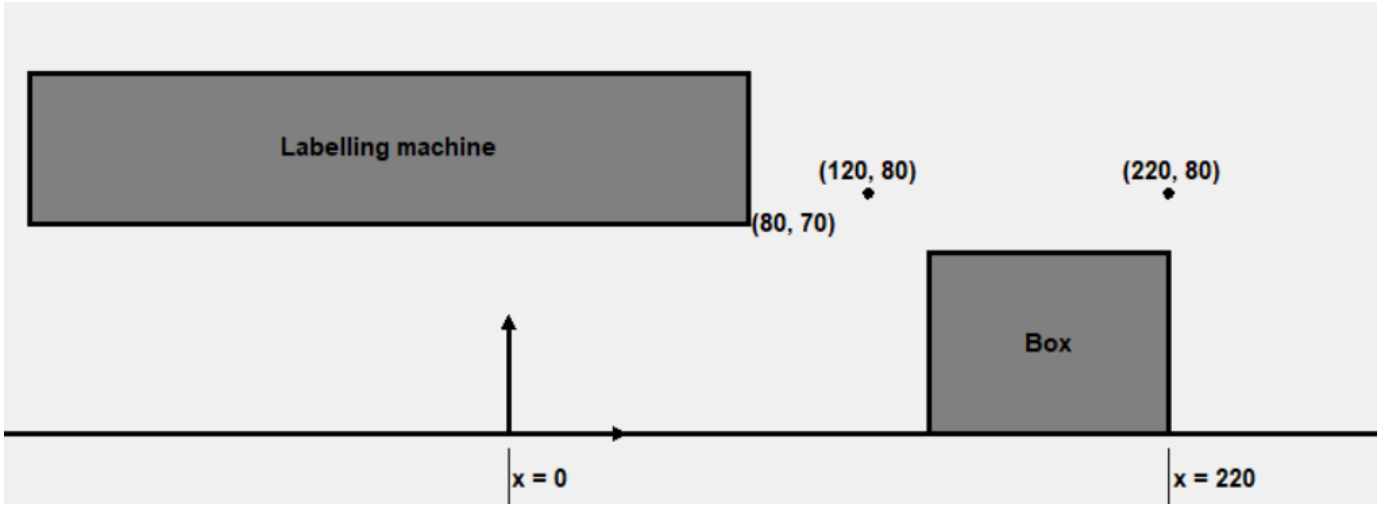
The inequality 5-1 is **Grashof's criterion**.

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

- 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_1T_2	T_1T_3	*a*	*b*
1	+	+	+	+	+	crank	rocker
2	0	+	+	0	0	crank	π–rocker
3	-	+	+	-	-	π–rocker	π–rocker
4	+	0	+	0	+	crank	0–rocker
5	0	0	+	0	0	crank	crank
6	-	0	+	0	-	crank	crank
7	+	-	+	-	+	π–rocker	0–rocker
8	0	-	+	0	0	crank	crank
9	-	-	+	+	-	crank	crank
10	+	+	0	+	0	crank	π–rocker
11	0	+	0	0	0	crank	π–rocker
12	-	+	0	-	0	π–rocker	π–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	π–rocker	crank
17	0	-	0	0	0	crank	crank
18	-	-	0	+	0	crank	crank
19	+	+	-	+	-	0–rocker	π–rocker
20	0	+	-	0	0	0–rocker	π–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 80×60 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.

- **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`
- **5. Frameworks:**
  - Pdoc: Used for generating project documentation, helping the team understand and maintain the code better.
  - Makefile: For build management.

Design

Class Model(s)

API Documentation

class GUI
- linkage
- tk
- width
- height
- model_frame
- model_animation
- 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

▸ View Source

class GUI: ▸ View Source

- linkage
- tk
- width
- height
- model_frame
- model_animation
- toolbar_frame
- def trace_C(self): ▸ View Source
- def trace_D(self): ▸ View Source
- def trace_P(self): ▸ View Source
- 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): ▸ View Source
- def init_linkage_display(self): ▸ View Source

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_lims()
- calculate_Point_Position()
- calculate_C_Position()
- calculate_P_Position()
- animation_alpha()
- switch_C2_C1()

built with pdoc

four_bar_linkage

▸ View Source

class FourBarLinkage: ▸ View Source

FourBarLinkage(
    AB,
    BC,
    CD,
    DA,
    alpha,
    theta,
    coupler_position,
    coupler_offset,
    timeinterval,
    alpha_velocity
) ▸ View Source

- AB
- BC
- CD
- DA
- alpha
- theta
- alpha_rad
- theta_rad
- coupler_position
- coupler_offset
- t
- alpha_velocity
- C_mode
- def init_default_values(self): ▸ View Source
- def run(self): ▸ View Source

- **1. Class Definition (FourBarLinkage)**
- **2. Initialization (__init__)**
- **3. Geometry Parameter Check:**
  - Geometric Validity (check_Parameter)
  - Linkage Type Identification (find_Linkage_Type)
- **4. Position and Angle Calculation:**
  - Position Calculation (run): Update parameters at each iteration or operation.
  - Angle Limits (calculate_alpha_lims)
  - Point Position Calculation (calculate_Point_Position)
  - Intersection Point Selection (calculate_C_Position)
- **5. Animation Control:**
  - Angle Update (animation_alpha): Update input angle ‘alpha‘ for animation.
  - State Switching (switch_C2_C1)

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.

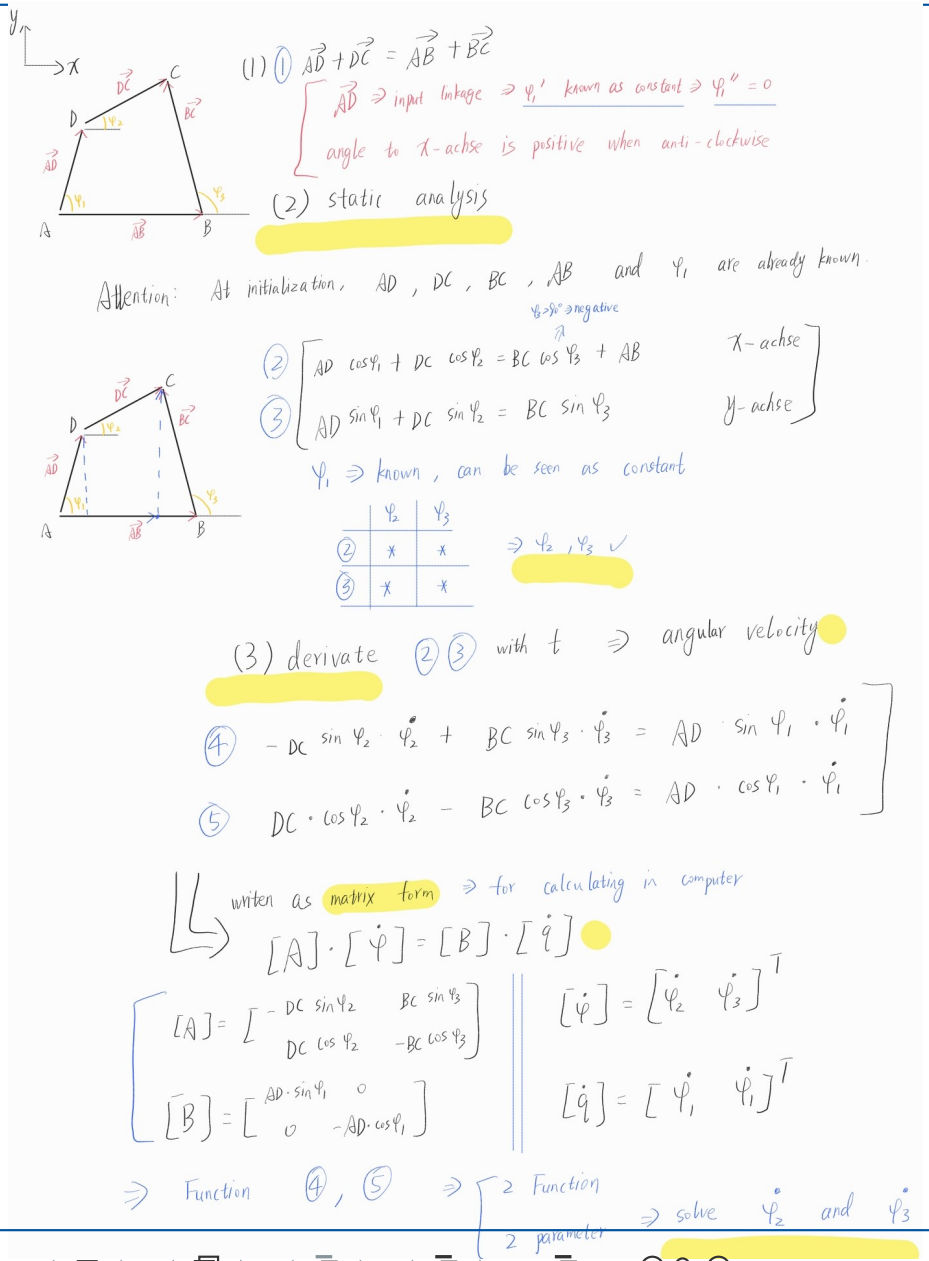
d. Check Parameter

- `check_Parameter(self)`
  - Check linkage type.
  - `geometric_Validity`

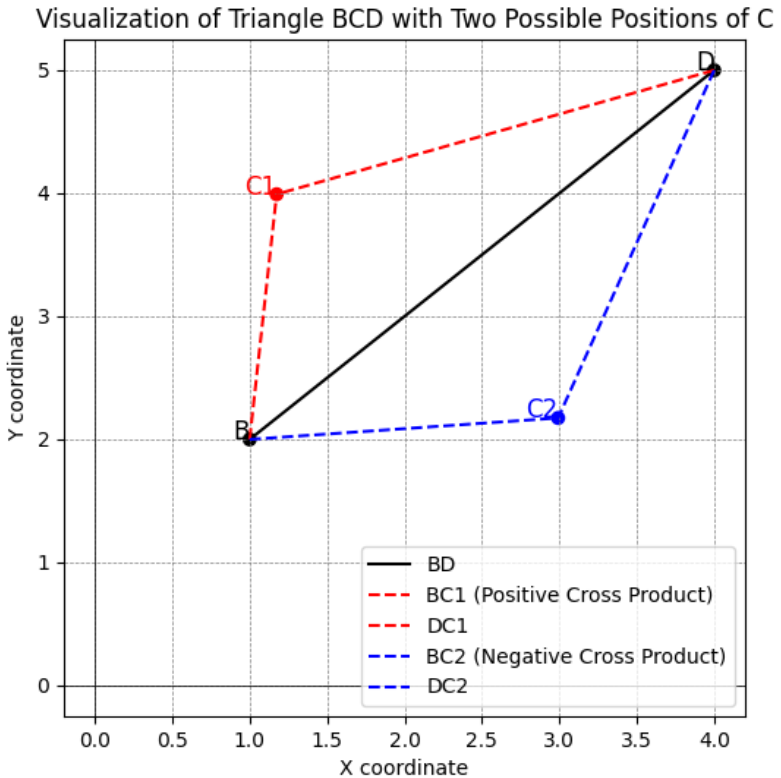
e. Update Parameters Every Run

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

- 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.

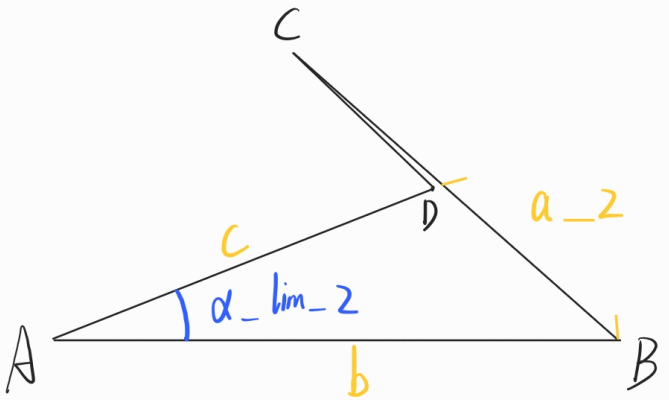


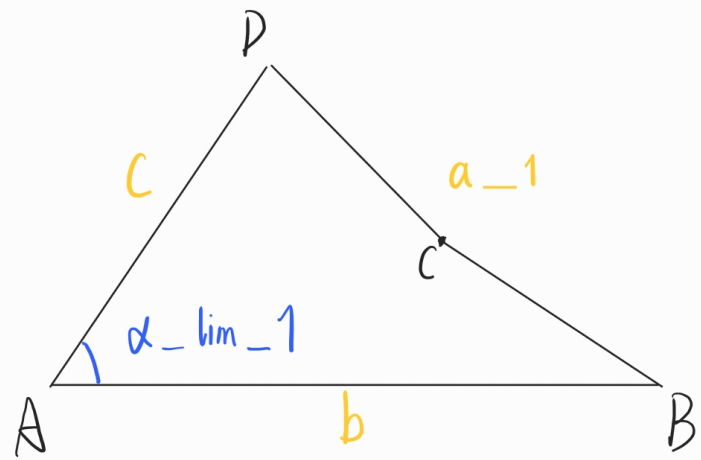
- A and B are fixed points.
- D can be determined using angle `alpha` and length `AD`.
- Given positions of B and D, and all side lengths of triangle BCD, point C has two possible locations.
- Calculating point C.



Picture from "test for calculating point C.py"

- **Cosine Law Formula:**
  - $$\cos(\alpha_{\text{lims1}}) = \frac{b^2 + c^2 - a_1^2}{2bc}$$
- Determine whether to switch point C to ensure animation continuity based on limits and thresholds:
  - `self.switch_C2_C1_180`
  - `self.switch_C2_C1_360`

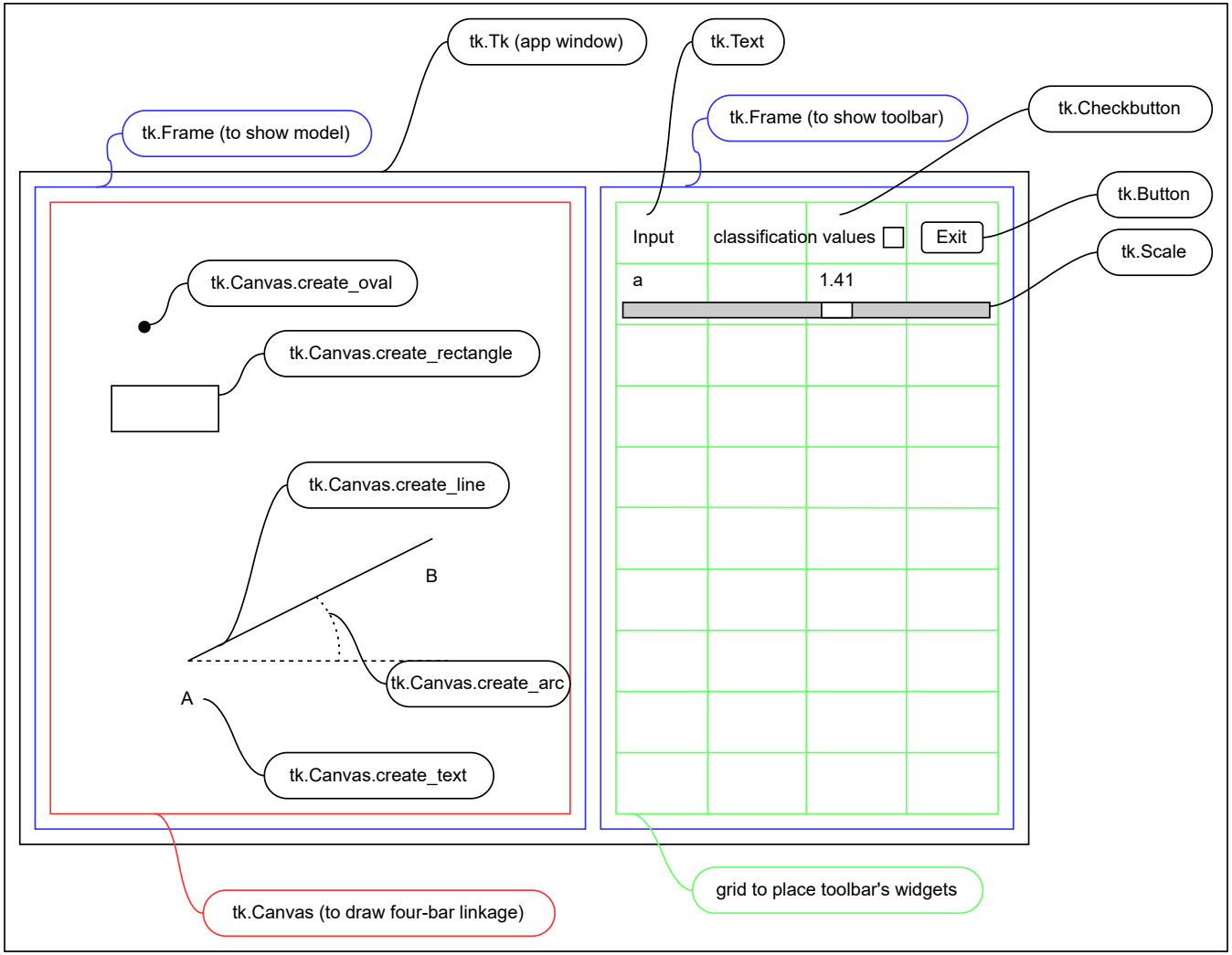




- **Basic Concept:**
  - Change `alpha` according to the defined limits.
  - Reverse direction at boundaries.
  - Switch between configurations (`C1`, `C2`) to ensure continuity.
- **Direction Control:**
  - `direction = 0`: *Increasing `alpha`*
  - `direction = 1`: *Decreasing `alpha`*
- **Updating `alpha`:**
  - Update by `alpha_velocity * t`.
  - Reverse direction when reaching limit values.
- **Special Switching Conditions:**
  - Switch at **180°** and **360°** based on specific conditions:
    - `self.switch_C2_C1_180`
    - `self.switch_C2_C1_360`
  - Handle floating-point precision issues (10^{-12}).
- **Configuration Tracking:**
  - Avoid redundant switching between `C1` and `C2`.

```
def animation_alpha(self):
    # Update alpha based on direction
    if self.direction == 0: # Increasing alpha
        self.alpha += self.alpha_velocity * self.t
    elif self.direction == 1: # Decreasing alpha
        self.alpha -= self.alpha_velocity * self.t
    ...
    self.alpha_rad = math.radians(self.alpha)
    ...
    # Handle alpha limits when limited
    if self.alpha_limited:
        if self.alpha >= self.alpha_lims[1]:
            self.alpha = self.alpha_lims[1]
            self.alpha_rad = self.alpha_rad_lims[1]
            self.direction = 1
            self._switch_C2_C1_if_needed()
        elif self.alpha <= self.alpha_lims[0]:
            self.alpha = self.alpha_lims[0]
            self.alpha_rad = self.alpha_rad_lims[0]
            self.direction = 0
            self._switch_C2_C1_if_needed()
    ...
    # Handle switch at 180 degrees and 360 degrees
    self._handle_special_switches()
    ...
    # Maintain alpha between 0 and 360 if not limited
    if not self.alpha_limited:
        self.alpha %= 360.0
        self.alpha_rad = math.radians(self.alpha)
    ...
    # Reset switch status if not switched this time
    if C2_C1_switched_pre_last_time:
        self.C2_C1_switched_last_time = False
```

- 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()
```

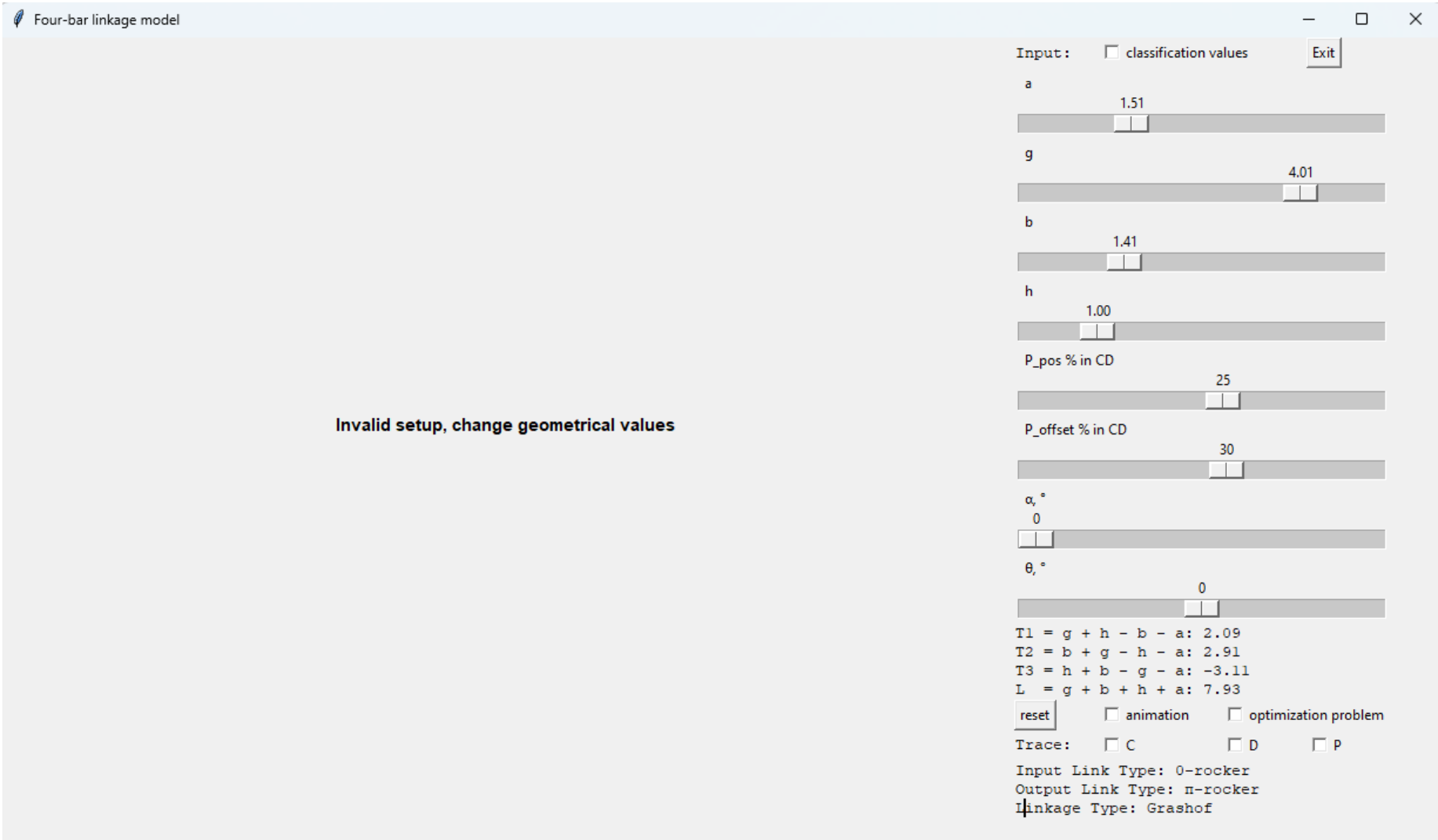
- Update objects in tk.Canvas every animation step using coords and/or itemconfigure for optimization

```
class GUI:
  def __init__(self):
    ...
    self.init_toolbar()
    ...
  def init_toolbar(self):
    ...
    self.enable_animation = tk.IntVar()
    self.animation_button = tk.Checkbutton(self.toolbar_frame, text="animation",
                                           variable=self.enable_animation,
                                           onvalue=1, offvalue=0, command=self.animation)
    self.animation_button.grid(sticky="W", row=10, column=2)
    ...
  def refresh(self):
    ...
    self.linkage.run()
    ...
    self.update_linkage_display()
  def animation(self):
    self.run_animation()
  def run_animation(self):
    if self.enable_animation.get():
      self.linkage.animation_alpha() # alpha = alpha + d_alpha
      self.refresh()
      self.tk.after(25, self.run_animation)
  def update_linkage_display(self):
    ...
    self.model_animation.coords(self.model_animation.AB_line, [A_x, A_y, B_x, B_y])
    ...
```

- 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()

Implementation

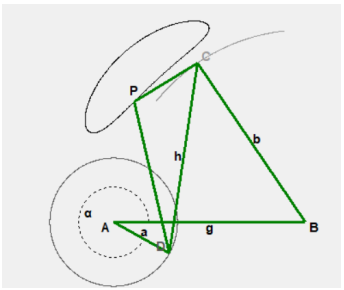
GUI, Invalid Setup Handling



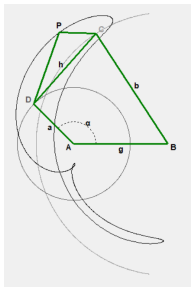
```
class GUI:
  def __init__(self):
    ...
    self.init_linkage_display()
    ...
  ...
  def init_linkage_display(self):
    self.model_animation.invalid_text = self.model_animation.create_text(round(self.model_animation.width/2),
                                                                          round(self.model_animation.height/2),
                                                                          text="Invalid setup, change geometrical values",
                                                                          fill="black", font=('Helvetica 11 bold'))
    self.model_animation.itemconfigure(self.model_animation.invalid_text, state='hidden')
    ...
  ...
  def update_linkage_display(self):
    if self.linkage.geometric_Validity:
      self.show_linkage()
      if self.enable_optimization_problem.get():
        self.show_optimization_problem()
      self.model_animation.itemconfigure(self.model_animation.invalid_text, state='hidden')
    else:
      self.hide_linkage()
      self.hide_optimization_problem()
      self.model_animation.itemconfigure(self.model_animation.invalid_text, state='normal')
      return
    ...
```

Results

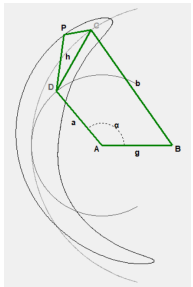
27 movement types



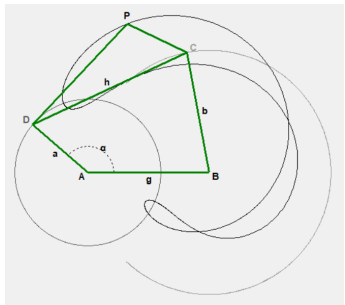
$T_{1,2,3} = 1.0, 1.0, 1.0$



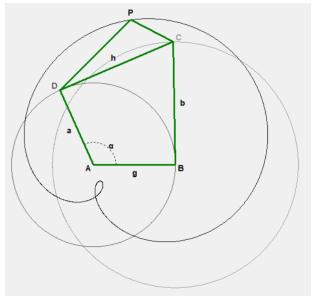
$T_{1,2,3} = 0.0, 1.0, 1.0$



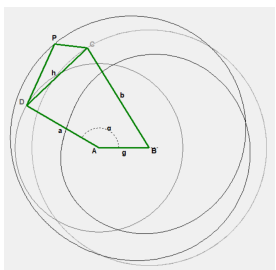
$T_{1,2,3} = -1.0, 1.0, 1.0$



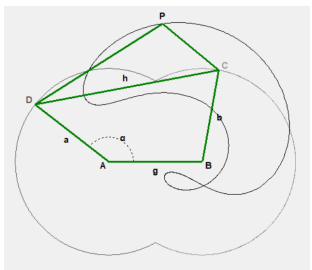
$T_{1,2,3} = 1.0, 0.0, 1.0$



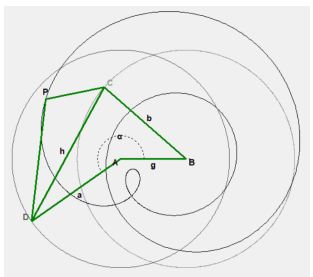
$T_{1,2,3} = 0.0, 0.0, 1.0$



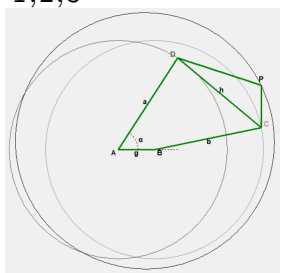
$T_{1,2,3} = -1.0, 0.0, 1.0$



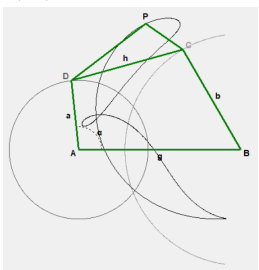
$T_{1,2,3} = 1.0, -1.0, 1.0$



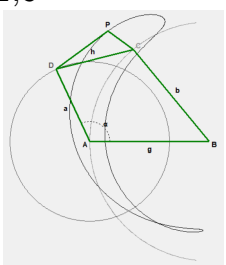
$T_{1,2,3} = 0.0, -1.0, 1.0$



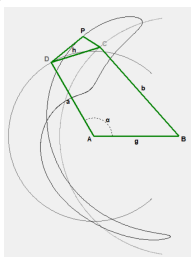
$T_{1,2,3} = -1.0, -1.0, 1.0$



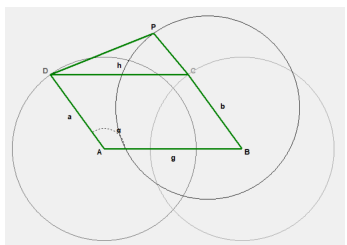
$T_{1,2,3} = 1.0, 1.0, 0.0$



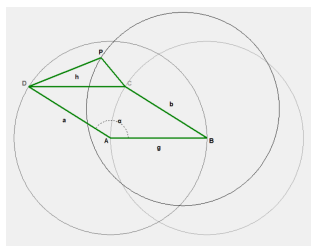
$T_{1,2,3} = 0.0, 1.0, 0.0$



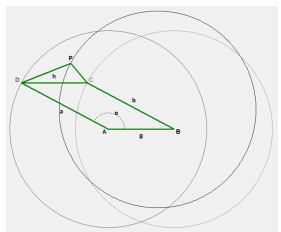
$T_{1,2,3} = -1.0, 1.0, 0.0$



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



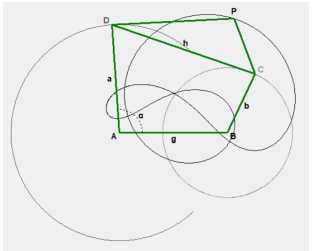
$T_{1,2,3} = 0.0, 0.0, 0.0$



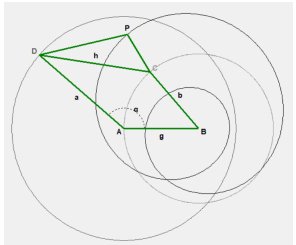
$T_{1,2,3} = -1.0, 0.0, 0.0$

Results

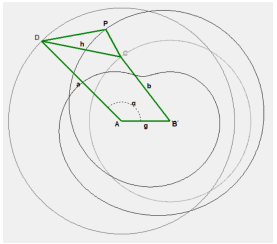
27 movement types



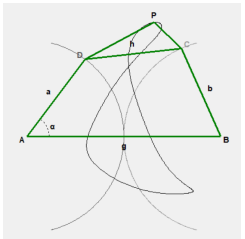
$T_{1,2,3} = 1.0, -1.0, 0.0$



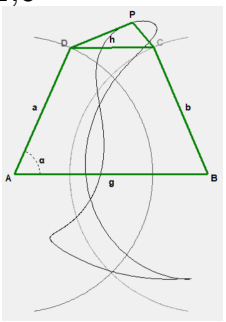
$T_{1,2,3} = 0.0, -1.0, 0.0$



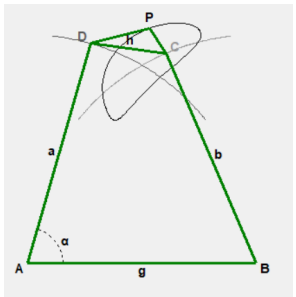
$T_{1,2,3} = -1.0, -1.0, 0.0$



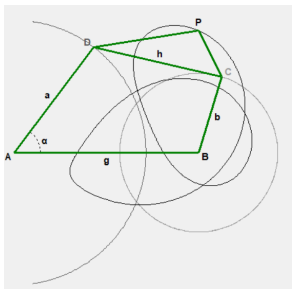
$T_{1,2,3} = 1.0, 1.0, -1.0$



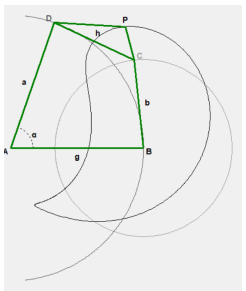
$T_{1,2,3} = 0.0, 1.0, -1.0$



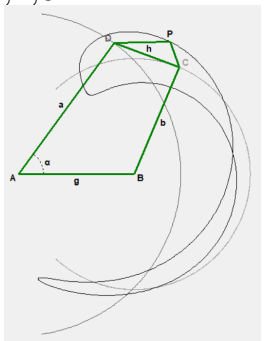
$T_{1,2,3} = -1.0, 1.0, -1.0$



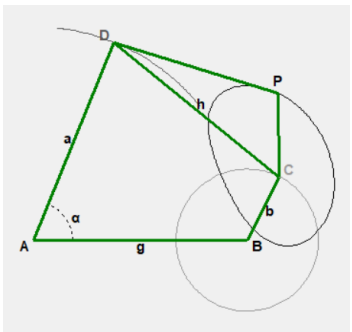
$T_{1,2,3} = 1.0, 0.0, -1.0$



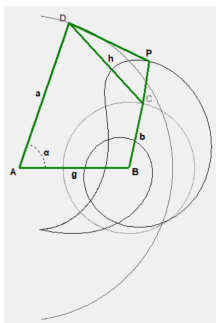
$T_{1,2,3} = 0.0, 0.0, -1.0$



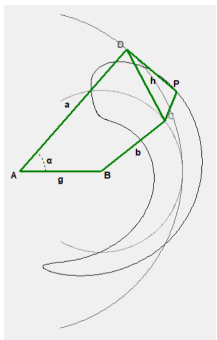
$T_{1,2,3} = -1.0, 0.0, -1.0$



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



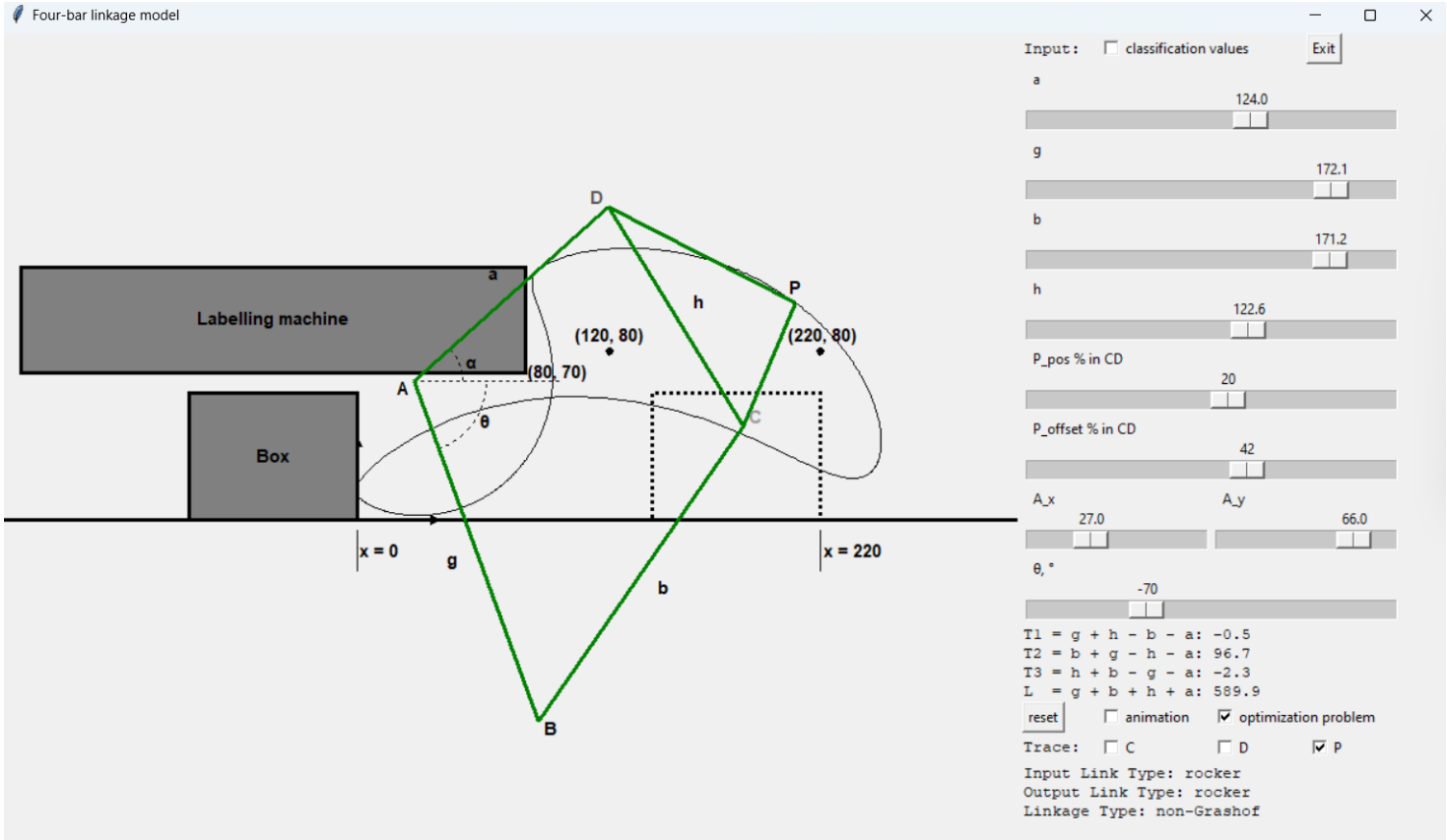
$T_{1,2,3} = 0.0, -1.0, -1.0$



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

Results

Optimization problem



- 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: $\theta = -70.0°$

Documentation for Frontend(GUI)

Documentation for Backend

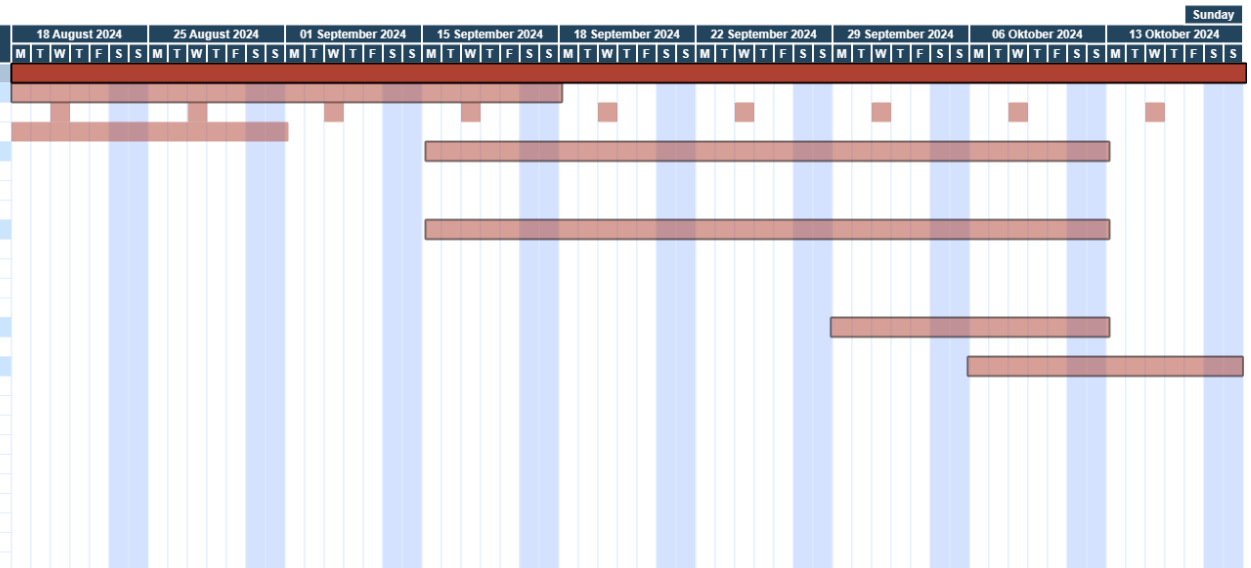
- **1.Discuss and Design:**
  - weekly discussion in discord.
  - gathering information / generating ideas for program.
- **2.Frontend:**
  - Design of GUI
  - Implementation
  - Debug
- **3.Backend:**
  - Algorithm for calculating positions and angle extremum
  - Interface for animation
  - Two types 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.

Gantt Chart

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

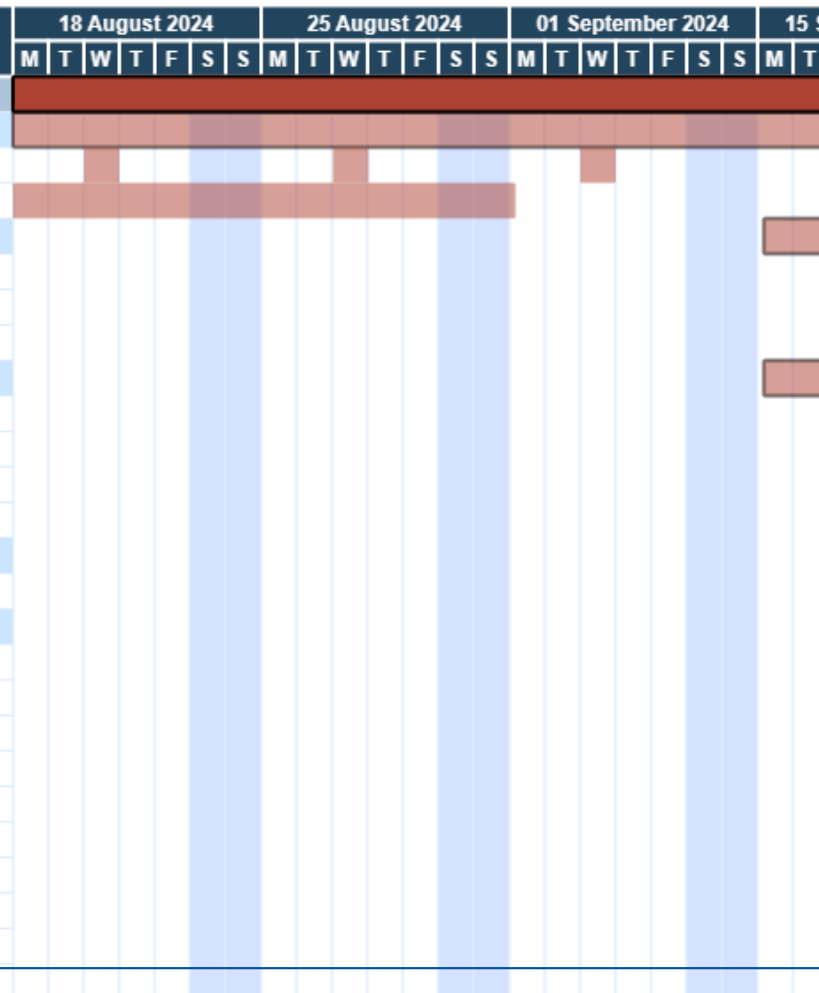
Project Management

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			



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

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			



Live Software Demo

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

Summary and Conclusion

Literature

- 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.