Software Lab
Computational Engineering Science
Pusher Mechanism

Aaron Albert Floerke, Arseniy Kholod, Xinyang Song, Yanliang Zhu

Supervisor: Dr. rer. nat. Markus Towara*

18th November 2024

*Informatik 12: Software and Tools for Computational Engineering, RWTH Aachen University, info@stce.rwth-aachen.de

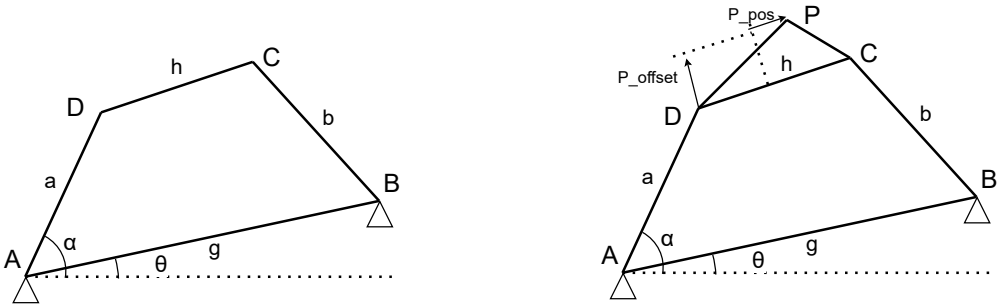
Preface

The topic "Pusher Mechanism" was issued as a final project by the Department of Informatics 12: Software and Tools for Computational Engineering, RWTH Aachen University, for the software developing practice in the Computational Engineering Science B.Sc. program. The work was conducted under the supervision of Dr. rer. nat. Markus Towara.

The topic involves the development of software to model and visualize a planar four-bar linkage extended with a coupler, and to find a suitable linkage for moving a box along a conveyor line under certain constraints, outlined later in this work. These tasks require a foundational knowledge of planar geometry, software engineering, project management and programming. This knowledge was gained by the students of Computational Engineering Science B.Sc. program during the first four semesters of study. Therefore this project is well-suited to the study program for fifth-semester students.

1 Introduction

The four-bar linkage is an important mechanism that has a broad application domain, for example, pumpjacks, robotic arms and automotive engineering. Another possible application for the four-bar linkage is as part of a conveyor line to move products in factories. In this work, we will implement software for the visualization and animation of the extended four-bar linkage with coupler, illustrated in Figures 1a and 1b. This implementation will be used to find a linkage to solve a real-world task, illustrated in Figure 3, which involves moving a box along a conveyor line.



(a) Planar four-bar linkage

(b) Planar four-bar linkage with coupler P

Figure 1: Four-bar linkage

To provide a comprehensive overview, this report is structured as follows. In Section 2, we discuss and analyze user requirements, derive system requirements, and provide a theory foundation for the geometry of the linkage. Section 3 covers the selection of implementation environment based on the task and team's expertise, followed by the design of UML class diagrams to be used during implementation. After that, in Section 4, we implement the four-bar linkage model, the graphical user interface (GUI), and the test cases. In Section 5, we assess the correctness of the implementation by testing all possible motion types of the linkage. Section 6 provides a solution to the problem of moving a box along a conveyor under certain constraints. Finally, in Sections 7 and 8, we describe the software documentation and the project management.

2 Analysis

2.1 User Requirements

User requirements are a list of customer expectations that the software must fulfill after development. These requirements are normally high-level expectations, that need to be translated by the developers into technical system requirements.

To better understand the user requirements listed below, we provide a UML use case diagram in Figure 2. The diagram illustrates the user-software interaction. After the user starts the GUI, the software generates the default visualization of the four-bar linkage. The user also has an option to enter new geometric parameters and obtain the updated visualization. Moreover, if the user enables animation mode, the four-bar linkage model gradually increases the input angle and the GUI updates the visualization every 25 milliseconds with new geometric data to simulate the motion of the linkage.

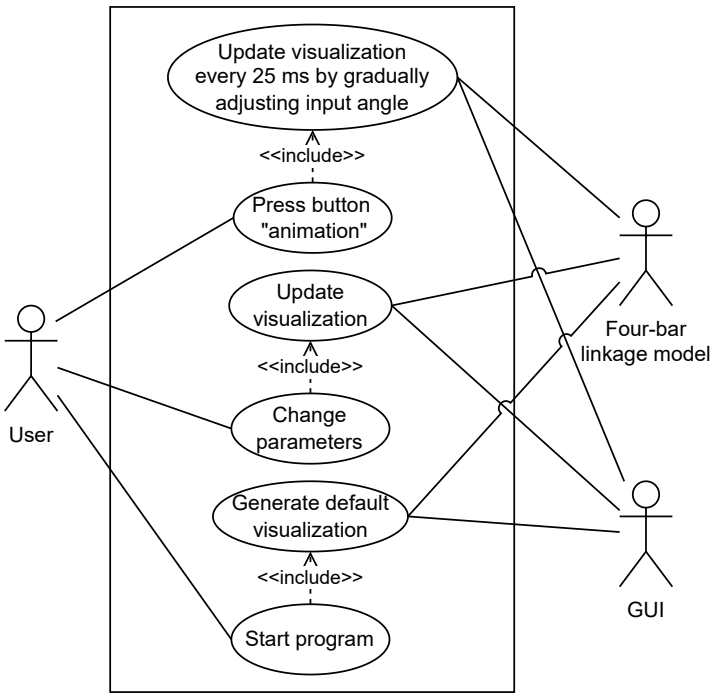


Figure 2: UML use case diagram

The following list contains all the user requirements provided by our supervisor, supplemented with our explanation for each.

- *Requirement: Implement all motion types of a planar four-bar linkage extended with a coupler.*

  Figure 1b illustrates the four-bar linkage with a coupler, which will be implemented in this work. As explained in [1], there are 27 distinct motion cases of the planar four-bar linkage. All of the motion cases and corresponding geometry will be explained in the next sections.

- *Requirement: Implement a graphical user interface (GUI) to display and animate the four-bar linkage, and to input geometric parameters.*

  The GUI should provide the visualization of the four-bar linkage with a coupler, as well as the animation of its motion. User should have an option to configure the linkage parameters using the GUI. We have identified ten free geometric parameters (degrees of freedom), that the GUI should support:

  - The lengths of the four bars (AB, BC, CD, AD).
  - The angle θ between the fixed bar AB and the horizontal line.
  - The input angle α between the bar AD and the horizontal line.
  - The position of the coupler relative to the middle of the floating link CD, expressed by P_{pos} and P_{offset}.
  - Position of the joint A on xy-plane (x- and y-coordinates).

For animation, the input angle α is no longer a free parameter, because it will be dynamically adjusted by the system to simulate the motion of the linkage.

Also note that, for simplicity, we will fix the joint A at $(0, 0)$ for analysis and visualization in the GUI. To change the position of A, we will translate the entire linkage accordingly. The configurable position of A will be important to solve an optimization problem discussed in the next point.

- *Requirement: Find the four-bar linkage with a coupler to solve the following optimization problem:*
  - Push the box with size 80×60 from $x = 220$ to $x = 0$
  - Do not cross the area of the labeling machine (area where $x < 80$ and $y > 70$).
  - Pass above the points $(120, 80)$ and $(220, 80)$

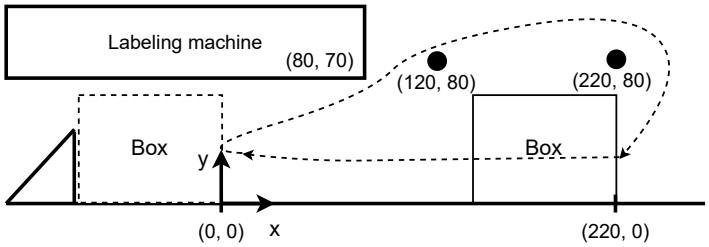


Figure 3: Optimization problem

Figure 3 illustrates the optimization problem. To solve the problem we need to find the geometrical parameters of the linkage, so that the coupler P will move the box along the conveyor line, avoiding all obstacles, and then return for the next box. The example trajectory of P is depicted as the dotted line.

The minimum requirement is to solve this problem manually by experimenting to find a suitable linkage. Ideally, we would develop an algorithm to find suitable parameters efficiently.

2.2 System Requirements

After discussing the user requirements, which provide only a high-level overview of the software, we need to derive more technical system requirements. The system requirements are subdivided into functional requirements, which describe what system should do, and non-functional requirements, which specify how the system should meet the functional requirements.

We begin with the functional requirements.

- *Four-bar linkage model:*
  - The model calculates joints' coordinates based on the input parameters.
  - It implements all 27 motion types of the four-bar linkage with a coupler.
  - It validates the user's input.
  - It ensures stable operation without crashes.

The four-bar linkage model is a back-end part of the software, which implements the geometry and all motion types of the four-bar linkage. Its role is to provide correct data for visualization and animation of the linkage in the GUI. The model validates the input data and notifies the GUI, prompting the user to change an invalid input.

- *Tests:*
  - Implement test cases for all motion types of the four-bar linkage.
  - Provide reference data.

  To guarantee the back-end's accuracy, we must implement test cases for each motion type and provide appropriate reference data.

- *Graphical User Interface (GUI):*
  - The GUI uses the four-bar linkage model (back-end) to get the coordinates of the joints for visualization and animation.
  - It provides a visualization of the four-bar linkage.
  - It contains sliders for user to input geometric parameters.
  - It updates the visualization according to the user's input.
  - It includes an animation mode for smooth motion visualization of the four-bar linkage.
  - It provides tracing of the trajectory for the coupler.

  The GUI is the front-end interface for the user to generate the visualization and animation of the linkage based on the user's input. It uses data obtained from the four-bar linkage model. Therefore, the user must be able to input parameters in the GUI. To solve the optimization problem, we also need to trace the motion of the coupler.

- *Documentation*
  - The four-bar linkage model, tests, and GUI are documented in detail.

  High-quality documentation is essential for code to be reusable and maintainable.

Since we have defined the functional system requirements, we define now the non-functional system requirements.

- *Performance:*
  - The four-bar linkage model must provide smooth animations.
  - The GUI animations should run at a minimum of 30 frames per second.

  The crucial factor for usability of the software is its performance. In our case, performance is measured by the number of frames per second (fps) during the animation. This criterion also depends on the machine used to run the code, so we assess the fps using a standard laptop with an AMD Ryzen 5 4500U chip.

2.3 Geometry

After analyzing the user requirements and deriving system requirements, we focus on the geometry of the given linkage. This will be implemented as part of the four-bar linkage model and used to create visualization and animation later in the GUI.

To visualize the linkage, we need to calculate the positions of all the joints based on the input parameters. The input parameters are:

- The lengths of main four bars AB, BC, CD, AD, that we denoted as g, b, h, a.
- The input angle α.
- The angle θ between the horizontal line and AB.

- The position of coupler P defined with respect to midpoint of CD and denoted as P_{pos} and P_{offset}.

  Note that these two values also can be negative, take a look at the direction of corresponding arrows in Figure 4.

All the mentioned parameters are depicted in Figure 4.

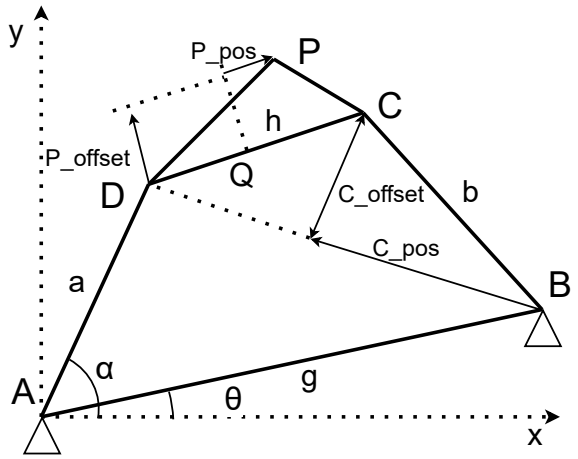


Figure 4: Planar four-bar linkage with coupler P

After recalling all the input parameters we use them to find out the positions of the joints one by one.

- Joint A.

  As previously mentioned, we position joint A at $(0, 0)$ for simplicity. To place it at different coordinates, the whole linkage should be translated by adding the new coordinates of A to each joint's coordinates.

- Joint B.

  The position of joint B can be defined using g, the length of bar AB, and angle θ: $B_x = g\cos(\theta)$, $B_y = g\sin(\theta)$.

- Joint D.

  The position of joint D is also easy to determine using a, the length of AD, and input angle α: $D_x = a\cos(\alpha)$, $D_y = a\sin(\alpha)$.

- Joint C.

  The most complex problem is to determine the position of joint C. The main idea is to derive the position of C by considering the area of triangle $\triangle BCD$ using different methods.

  To determine the position of C, we consider new basis vectors.

  Define a vector $\overrightarrow{BD} = (BD_x, BD_y) = \overrightarrow{D} - \overrightarrow{B} = (D_x - B_x, D_y - B_y)$. Assume first that this vector has non-zero length $|\overrightarrow{BD}| = \sqrt{(BD_x)^2 + (BD_y)^2}$.

  The corresponding unit vector along $\overrightarrow{BD}$ is $\overrightarrow{e}_{BD} = (e_{BD\_x}, e_{BD\_y}) = \frac{\overrightarrow{BD}}{|\overrightarrow{BD}|}$.

  The orthogonal direction is defined by a unit vector $\overrightarrow{n}_{BD} = (-e_{BD\_y}, e_{BD\_x})$.

We use the vectors $\overrightarrow{e}_{BD}$ and $\overrightarrow{n}_{BD}$ as a new orthonormal basis.

To determine the area of $\triangle BCD$ we use Heron's formula:

$A_{\triangle BCD} = \sqrt{p(p-b)(p-h)(p-|\overrightarrow{BD}|)}$, where $p = \frac{b+h+|\overrightarrow{BD}|}{2}$ is a semi-perimeter of $\triangle BCD$.

On the other hand the area of $\triangle BCD$ can be determined using length of BD and the perpendicular from C to BD denoted by C_{offset} (see Figure 4): $A_{\triangle BCD} = \frac{|\overrightarrow{BD}|C_{offset}}{2}$. So we can determine $C_{offset} = 2\frac{A_{\triangle BCD}}{|\overrightarrow{BD}|}$

At this point, we know the distance from joint C to BD along $\overrightarrow{n}_{BD}$. To determine the position of C with respect to B, we also need the distance from C to B along $\overrightarrow{e}_{BD}$. The projection length of $\overrightarrow{BC}$ onto direction of $\overrightarrow{e}_{BD}$ is given by $|C_{pos}| = \sqrt{b^2 - C_{offset}^2}$ using Pythagorean theorem. The remaining question is to determine sign of this projection. This can be done using angle $\angle CBD$ and the Law of Cosines:

$\cos(\angle CBD) = \frac{b^2+|\overrightarrow{BD}|^2-h^2}{2b|\overrightarrow{BD}|}$

Then the projection of $\overrightarrow{BC}$ onto direction of $\overrightarrow{e}_{BD}$ is given by $C_{pos} = sign(\cos(\angle CBD))\sqrt{b^2 - C_{offset}^2}$

After determining C_{pos} and C_{offset}, we can find the possible positions of C:

$\overrightarrow{C}_1 = (C_{1\_x}, C_{1\_y}) = \overrightarrow{B} + C_{pos}\overrightarrow{e}_{BD} + C_{offset}\overrightarrow{n}_{BD}$

$\overrightarrow{C}_2 = (C_{2\_x}, C_{2\_y}) = \overrightarrow{B} + C_{pos}\overrightarrow{e}_{BD} - C_{offset}\overrightarrow{n}_{BD}$

There are two possible positions of C, because the normal vector to $\overrightarrow{BD}$ is not unique and can also have the opposite direction. For a static structure, we can choose any of them, so we take $C = C_2$ as the default. For the animation case, that will be discussed later, there are rules for choosing between C_1 and C_2.

In the discussion above we made the assumption that the length of $\overrightarrow{BD}$ is not zero. However, for $b = h$ and a specific value of input angle α, the length can become zero. In this case, the joints A, B, C, D are on the same line, so we define a unit vector $\overrightarrow{e} = sign(\overrightarrow{BA} \cdot \overrightarrow{BC})\frac{\overrightarrow{BA}}{g}$, where $\cdot$ is a scalar product. Then, the position of C is determined uniquely by $\overrightarrow{C} = \overrightarrow{B} + b\overrightarrow{e}$

- Coupler P.

  Since we know the positions of C and D, it is easy to determine the position of P.

  Define the midpoint of CD as $\overrightarrow{Q} = \frac{\overrightarrow{C}+\overrightarrow{D}}{2}$.

  The unit vector along DC is given by $\overrightarrow{e}_{DC} = (e_{DC\_x}, e_{DC\_y}) = \frac{\overrightarrow{C}-\overrightarrow{D}}{h}$. The corresponding normal vector is $\overrightarrow{n}_{DC} = (-e_{DC\_y}, e_{DC\_x})$.

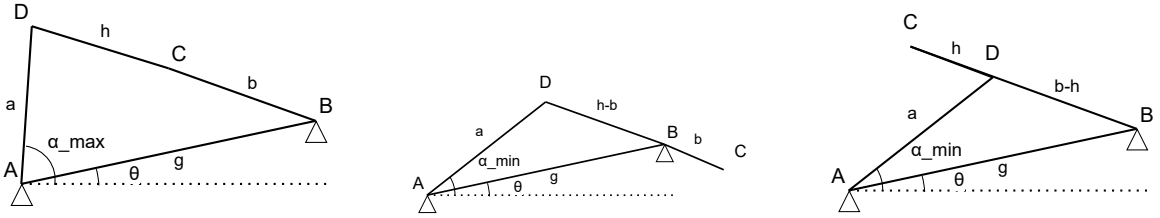
  Then, the position of the coupler P is determined by $\overrightarrow{P} = \overrightarrow{Q} + P_{pos}\overrightarrow{e}_{DC} + P_{offset}\overrightarrow{n}_{DC}$.

  Unlike the position of C, the position of P is determined uniquely, because P_{offset} can be specified as negative by the user, automatically changing the direction of $\overrightarrow{n}_{DC}$.

2.4 Parameter Validation

Not every set of the input parameters is feasible. In this section we will derive constraints on the input angle α and the links' lengths to ensure that the linkage exists.

Figures 5a, 5b, 5c illustrate the cases of the maximum and minimum input angle α. The coupler P has no influence on the limits of the input angle therefore it is not shown in the figures. The limits of α are defined by the cases when the four-bar linkage folds into a triangle.



(a) Maximum input angle α. (b) Minimum input angle α. (c) Minimum input angle α.

Figure 5: Constraints on the input angle α.

- Maximum input angle α.

  Figure 5a displays the upper limit of the angle α, that can be derived using the Law of Cosine for triangle $\triangle ABD$: $(h+b)^2 = a^2 + g^2 - 2ag\cos(\alpha_{max} - \theta)$.

  If this cosine $\cos(\alpha_{max} - \theta) = \frac{a^2+g^2-(h+b)^2}{2ag}$ has an absolute value less than one, there exists an upper limit for the input angle α: $\alpha_{max} = \arccos(\frac{a^2+g^2-(h+b)^2}{2ag}) + \theta$.

- Minimum input angle α.

  Figures 5b, 5c show two different cases for minimum input angle α. The both cases could be described using the Law of Cosine for triangle $\triangle ADB$: $(h-b)^2 = a^2 + g^2 - 2ag\cos(\alpha_{min} - \theta)$.

  If this cosine $\cos(\alpha_{min} - \theta) = \frac{a^2+g^2-(h-b)^2}{2ag}$ has an absolute value less than one, there exists a lower limit of the input angle α: $\alpha_{min} = \arccos(\frac{a^2+g^2-(h-b)^2}{2ag}) + \theta$.

- Special cases.

  If both cosines $\cos(\alpha_{min} - \theta)$ and $\cos(\alpha_{max} - \theta)$ have an absolute value greater or equal one, then there is no limits to the input angle α.

  If only the maximum limit exists, then the input angle α also has a lower boundary, that is symmetric with respect to the ground link AB: $\alpha \in [\alpha_{max}, 2\theta - \alpha_{max}]$.

  In the opposite case, when only the minimum input angle α exists, then the input angle is also bounded from above symmetrically with respect to the ground link AB: $\alpha \in [2\theta + 2\pi - \alpha_{min}, \alpha_{min}]$.

We decided that the GUI will display a slider for input angle α, ensuring that only valid values within the boundaries can be entered by the user. A similar limitation can be derived for the output angle $\angle ABC$.

There is also a case when the four-bar linkage does not exist for any input angle α. This occurs when one of the bars is longer than the sum of other three, so that the quadrilateral $\square ABCD$ does not exist. Denote the longest bar as $l = \max(a, b, g, h)$, the shortest bar as $s = \min(a, b, g, h)$, and two remaining bars as p, l. Then the condition for linkage to exist is $l - p - q - s \leq 0$. This condition will be checked by the four-bar linkage model, which will notify the user through the GUI in case of a problem.

2.5 Classification

After deriving the expressions for constraints on the input angle α, we can classify the motion of four-bar linkage.

Depending on the constraints on the input angle α and the output angle $\angle ABC$, the input link AD and the output link BC can be classified into four types according to [1].

- *Crank.*

  The link can rotate fully, with neither a minimum, nor a maximum for the input angle α (or the output angle $\angle ABC$).

- *Rocker.*

  The link can rotate partially, with both a minimum and a maximum for the input angle α (or the output angle $\angle ABC$).

- *0-rocker.*

  The link can rotate partially, with no minimum but with maximum for the input angle α (or the output angle $\angle ABC$).

- *π-rocker.*

  The link can rotate partially, with no maximum but with minimum for the input angle α (or the output angle $\angle ABC$).

2.6 27 Motion Types

The classification of the input and output links implies, that there are different linkage types. For example, the input and output can independently be classified as cranks or rockers, leading to different linkage types. The study in [1] identifies 27 different combinations. These combinations can be described by signs of characteristic values: $T_1 = g+h-a-b$, $T_2 = b+g-h-a$ and $T_3 = b+h-g-a$. Since each characteristic value can be positive, negative, or zero, there are 27 possible combinations and motion types respectively.

Initially, we discussed to analyze the motion in each of 27 cases separately. However, we decided to implement the general algorithms that handles all cases.

2.7 Animation

At the beginning of the animation, we start with a static state of the linkage. By default we set C_2 as the chosen position for joint C. The animation is implemented as a discrete process by gradually incrementing the input angle using the following key variables:

- dt: Time interval of the animation.
- $\dot{\alpha}$: constant angular velocity of the input angle α.

This is our basic idea of iterative animation:

- Calculate the limit values of α described above.
- Update α using the following formula: $\alpha = \alpha \pm \dot{\alpha} dt$.

  The sign here depends on the `direction` parameter. It determines whether α changes clockwise (`direction = 1`) or counterclockwise (`direction = 0`).

- Use the updated α value to calculate other parameters, such as the positions of the joints.
- If the input angle α is limited, then change the `direction` parameter at the limits to rotate the input link backwards.
- If the input angle is not bounded, ensure that α always remains within the range $[0°, 360°]$. If the input angle exceeds $360°$ or falls below $0°$, we will reset it to the corresponding boundary value.

- Switch between C_1 and C_2 to ensure continuity of animation.

  As we mentioned earlier, for the static configuration, we choose C_2 as the default position for joint C. However, during the motion, we need to switch between C_1 and C_2 to maintain continuous velocity due to the principle of inertia. This means that at every limit of the input angle we switch between C_1 and C_2.

  Note, that we need to switch even at the boundary cases, when the absolute values of $\cos(\alpha_{max} - \theta)$ or $\cos(\alpha_{min} - \theta)$ are equal to one, indicating that the input angle has reached its extreme positions.

- Consider floating point precision.

  To avoid issues caused by floating point inaccuracies when checking if the cosine value of α approaches -1 or 1, we incorporate floating point tolerance of 10^{-10}.

We also created a UML statechart diagram for the animation algorithm that can be found in Figure 9 in the Appendix.

3 Design

After completing the analysis part we begin to design the software that will be implemented later. Our design section consists of two parts, the selection of development infrastructure and the creation of class diagrams to be a basis for implementation.

3.1 Third-Party Software and Development Infrastructure

Since we now understand the problem that we need to solve, we can choose an implementation environment.

We decided to implement our project using Python. This decision is based on several key factors:

- Python has a numpy[1] library that deals well with vector operations. The four-bar linkage model requires solving geometrical problem, so the built-in vector and matrix operations are highly desired to simplify the code.
- Python has a standard GUI library tkinter[2].
- Most of the team members have more experience with Python than with C++.

There is a list of the software and tools we used for implementation:

- Operating systems: Xubuntu and Windows.
- Programming language: Python.
- Integrated development environment (IDE): Spyder[3] and PyCharm[4].
- Package manager: Anaconda[5].
- Libraries: tkinter[2], numpy[1], math[6], unittest[7].

[1] https://numpy.org/
[2] https://docs.python.org/3/library/tkinter.html
[3] https://www.spyder-ide.org/
[4] https://www.jetbrains.com/pycharm/
[5] https://www.anaconda.com/
[6] https://docs.python.org/3/library/math.html
[7] https://docs.python.org/3/library/unittest.html

- Version control system: GitHub repository[8].
- Documentation: Pdoc[9]

3.2 Class Models

In the project we need to store a large amount of variables for geometric representation and visualization of the four-bar linkage, as well as for corresponding tools like buttons und sliders. The number of variables is too large to pass it into different functions as arguments. Therefore, we decided to implement the four-bar linkage model and the GUI as two separate classes, so every member function will have an access to the data stored in the class without passing it as an argument.

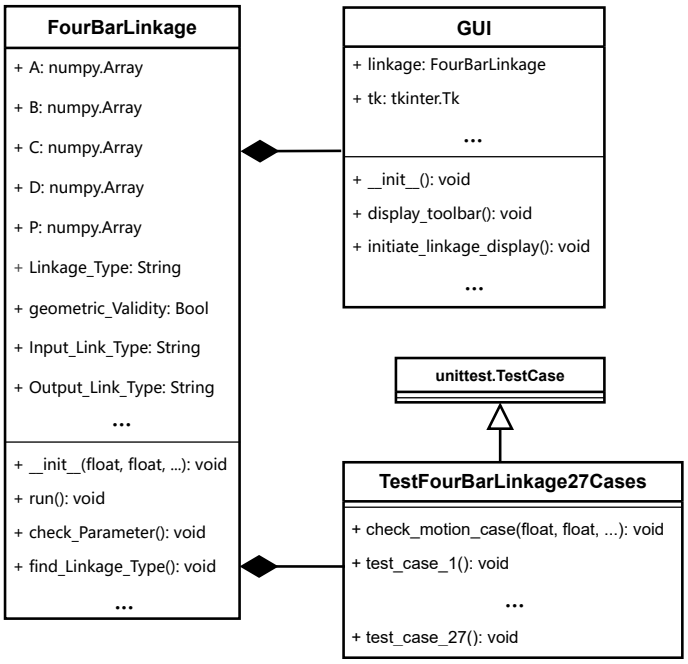


Figure 6: UML class diagram

Figure 6 shows a shortened version of the class diagram. This diagram illustrates all the relationships between classes and the most important attributes. The actual number of attributes is too high to include it in this document, the comprehensive class diagram can be found in our GitHub repository[10].

The back-end class `FourBarLinkage` implements the geometry and animation of the four-bar linkage with a coupler. The number of member variables is large, but the most important for visualization are the coordinates of joints A, B, C, D, and P. There are also member variables used for classification (`Linkage_Type`, `Input_Link_Type`, `Output_Link_Type`) and for input validation (`geometric_Validity`) to notify GUI about invalid parameters. The main member functions are the constructor `__init__` to set geometrical parameters, `run` to compute the joints' coordinates,

[8] `https://github.com/einsflash/Project_Pusher_Mechanism`
[9] https://pdoc.dev/docs/pdoc.html
[10] `https://github.com/einsflash/Project_Pusher_Mechanism/blob/main/src/class_diagram.pdf`

`check_Parameter` to validate user input, and `find_Linkage_Type` to classify the linkage. There are many other attributes used for internal purposes, but they are less important for a general overview.

The front-end class `GUI` is used to create the graphical user interface to visualize linkage, provide its animation and get geometrical parameters from the user. This class has also a large number of attributes, but only several of them are important for an overview. The member variables include an instance of the `FourBarLinkage` class, which is used to obtain joint coordinates for visualization and animation. The `GUI` class also contains an instance of `tkinter.Tk`, the basic class for creating GUI using tkinter. The main member functions are the constructor `__init__` to initiate the GUI, `display_toolbar` to set up the toolbar of buttons, sliders, and other elements, and `initiate_linkage_display` to set up the visualization of the linkage.

The test class `TestFourBarLinkage27Cases` inherits from `unittest.TestCase` and is used to test all motion types of the linkage. This class has only member functions that implement test cases for each motion types.

4 Implementation

4.1 Back-End

e.g. validation of input parameters

4.2 Front-End

After considering the back-end part of the software that implements the geometry of the four-bar linkage extended with a coupler, we focus on the GUI. The GUI will use the back-end to visualize the linkage and simulate its animation, enabling the user to adjust geometrical parameters of the linkage. The GUI is implemented using tkinter[2]. We will not go into too many details about the implementation to keep the description concise.

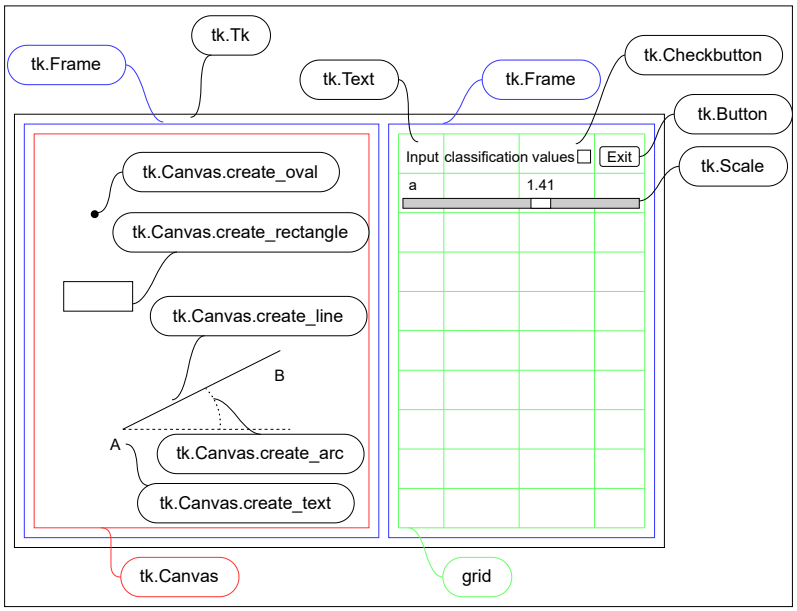


Figure 7: The main components of the GUI.

The main components of the GUI are illustrated in Figure 7. The components of the tkinter are

widgets that can be displayed in the GUI. The basic object of the GUI is `tkinter.Tk`. All other widgets are placed in the GUI using `tkinter.Frame` widget, which creates a grid and allows the explicit specification of widget positions.

The GUI is divided into two parts by two `tkinter.Frame` objects. The left one is used to generate the visualization of the linkage. It holds a singe widget, `tkinter.Canvas`, which can display all the basic geometric figures and text. The positions of these figures are specified using pixel-based x-y coordinates. The right `tkinter.Frame` is used to display the toolbar. The widget's position in the toolbar is specified using the grid. The toolbar is used to enter new geometrical parameters using the sliders `tkinter.Scale`. The toolbar also allows the user to change the visualization mode, e.g. enabling animation or tracing, using `tkinter.Button` and `tkinter.Checkbutton` objects.

To start the GUI, we call `tkinter.Tk.mainloop`.

The GUI needs the coordinates of the joints to visualize the linkage on `tkinter.Canvas`. To achieve that, the GUI contains the `FourBarLinkage` object as a member variable. To calculate the joints' coordinates, the GUI calls the back-end's member function `FourBarLinkage.run` and extracts the positions using the back-end's member variables `FourBarLinkage.A`, `FourBarLinkage.B`, and so on. These coordinates are scaled to fit the GUI's screen resolution, and then each link of the four-bar linkage is visualized using the `tkinter.Canvas.create_line` function.

For animation, we use `tkinter.Tk.after(25, func)`, which calls the `func` function after 25 milliseconds, allowing to update the visualization of the linkage in a loop.

To respond to the new geometric parameters entered by the user using the slider `tkinter.Scale`, each slider is linked to a specific function, called after each change of the value. For example, the slider used to set a is linked to the function `update_parameter_a`:

```
1 def update_parameter_a(self, val):
2         self.linkage.DA = float(val)
3         self.delete_tracing()
4         self.refresh()
```

This function reads the new value of a stored in the `val` variable, and updates the geometric parameter AD in the linkage accordingly. It also calls the `delete_tracing` function to remove no longer relevant trajectory visualization for C, D, and P. Finally, the `refresh` function is called to update the linkage visualization.

The same logic is applied to all toolbar widgets to update the visualization of the linkage based on the user's input. Note that we do not create new figures in `tkinter.Canvas` to generate new visualization, but instead update already existing figures by moving or stretching them for efficiency.

Another important feature of the GUI is different input and visualization modes. The user can choose to enter classification values (T_1, T_2, T_3, see Section 2.6) instead of the links' lengths or choose to display the optimization problem instead of the basic linkage visualization. The important building block to implement these features is to hide or show objects on the display. Common tkinter widgets can be hidden by removing them from the grid. For example, for the a slider: `self.slider_a.grid_remove()`, and shown by adding them back: `self.slider_a.grid()`. The same is done for geometric figures in the `tkinter.Canvas` object using the `itemconfigure` function and switching the state of the figure between `normal` and `hidden`.

If the back-end detects invalid input data, the GUI temporarily hides the linkage visualization and displays an error.

We also decided to visualize the optimization problem, illustrated in Figure 3, and its solution within the GUI. This feature was added using `tkinter.Checkbutton`, so when the user activates it, the default visualization of the GUI is replaced with the optimization problem, showing the linkage, the box to be moved and all the obstacles.

The Appendix includes screenshots of the GUI. Figures 10a, 10b, and 10c illustrate the default mode, the classification values input mode with an invalid setup message, and the optimization problem visualization, respectively.

The overview of the GUI class can be found in the documentation[11].
The running instructions are provided in Appendix B.

4.3 Software Tests

5 Validation

To validate the correctness of the implementation, we tested all 27 possible motion types described in [1]. The main aspect of interest is the absence of jumps in the coordinates and velocities of the joints, ensuring smooth animation.

Figure 11 is a set of screenshots of trajectory tracing for all 27 motion types. As described in Section 2.6, the motion type depends on the sign of the classification values T_1, T_2 and T_3. We decided to test the motion types with classification values of -1, 0 and 1. This validation method helped us to find and fix bugs in the implementation. And we can now conclude that the implementation works for all of motion types.

6 Optimization Problem

After we implemented the four-bar linkage model, the GUI, and validated their accuracy, we proceeded to solve the optimization problem illustrated in Figure 3.

To solve the problem, we implemented the optimization problem mode in the GUI, such that when the user activates the animation, the coupler P moves the box along a conveyor line around all obstacles. The box is moved by the coupler only while it remains in contact along the negative x-direction. With this approach and experience with all 27 motion types, we were able to find a suitable linkage that fulfills all the constraints described in Section 2.1.

As was discussed in Section 2.1, the four-bar linkage with a coupler generally has ten free parameters. However, to animate the motion, the input angle α cannot be chosen freely anymore. This means, the optimization problem has a degree of freedom of nine. The following list contains all of the freely selectable geometrical parameters that uniquely define the linkage solving the optimization problem:

- The four bar lengths: $|AD| = a = 124.0$ cm, $|BC| = b = 171.2$ cm, $|AB| = g = 172.1$ cm, $|CD| = h = 122.6$ cm.
- The coupler position relative to $|CD| = h$: $P_{pos} = 20.0\%$, $P_{offset} = 42.0\%$.
- The position of joint A: $A_x = 27.0$ cm, $A_y = 66.0$ cm relative to the x-y coordinate system denoted in Figure 3.
- The angle between the ground link AB and horizontal line: $\theta = -70.0°$

Figure 10c illustrates the solution of the optimization problem. The trajectory of coupler P is denoted in the figure. Using the chosen linkage, the coupler P moves the box to the target position and then returns for the next one, avoiding all obstacles.

7 Documentation

8 Project Management

Effective team management was essential for the success of this project. In the following section, we provide an overview of the task allocation, team collaboration, and actual progress.

[11] `https://htmlpreview.github.io/?https://github.com/einsflash/Project_Pusher_Mechanism/blob/main/src/doc_for_gui/gui.html`

At the beginning of the project, we used a Gantt chart to outline the preliminary project management. The tasks and timelines were organized and distributed across three months, as illustrated in Figure 8.

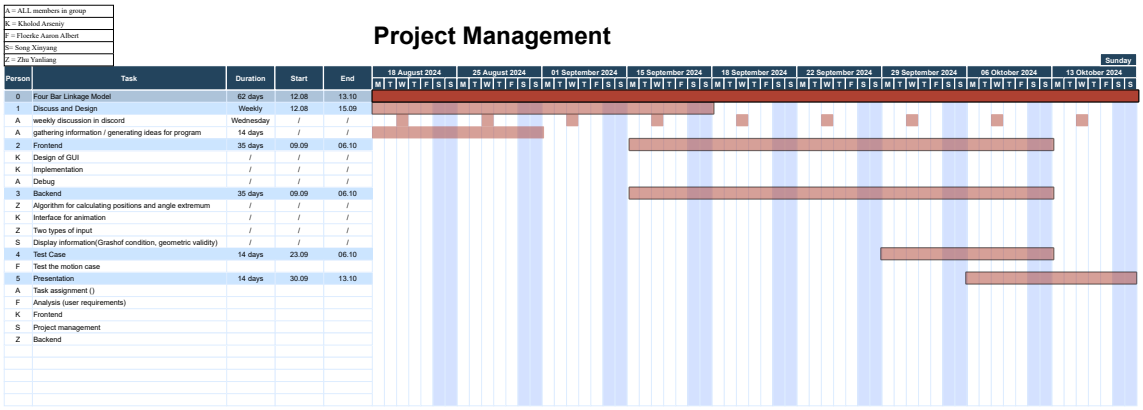


Figure 8: Gantt Chart for Project Management

The front-end and back-end development of the four-bar linkage was carried out using the agile methodology, Scrum, which employs an incremental and iterative approach to ensure project completion. The team meetings were held at least once every two weeks for planning the next and reviewing the previous tasks. We implemented features step by step and conducted testing accordingly as planned.

The following list is the specific task allocation for each group member:

- Arseniy Kholod: The leader of the team, responsible for front-end development and task assignment.
- Yanliang Zhu: Responsible for back-end development.
- Aaron Albert Floerke: Responsible for testing.
- Xinyang Song: Responsible for code documentation.

Below is an overview of team collaboration and progress:

- The first goal was to implement a basic static visualization of the four-bar linkage. Arseniy used tkinter to create the GUI. Zhu added functions such as joints' position calculation and validation of the input parameters to the back-end.
- In mid-August, Aaron introduced Python's unit testing to test 27 different motion types.
- By mid-September, we completed the function for animation. However, Zhu encountered issues with switching between C_1 and C_2 in the back-end in specific cases.
- Arseniy raised an issue on GitHub and solved the switching problem. Afterward, he added several new features to the front-end, such as tracing of joints.
- In early October, our team solved the optimization problem. After the presentation in October, we began preparing the final report.

9 Conclusions

Over the last several months, we successfully completed the "Pusher Mechanism" project. This included analyzing user requirements, deriving system requirements, establishing a theoretical foundation for the four-bar linkage, designing and implementing the software and test cases, validating the system's accuracy and solving the defined optimization problem.

The most challenging part of this project was the project management. Organizing the team, efficiently distributing tasks, and integrating all software components together were the critical parts. But despite the challenges, we are proud to successfully manage them.

While we are satisfied with the results, there is room for improvement. Although we developed software to validate the solution of the optimization problem, we did not implement an automated algorithm to efficiently determine such solutions. Developing such an algorithm would be a logical next step for this project.

References

[1] Ivana Cvetkovic, Misa Stojicevic, Branislav Popkonstantinović, and Dragan Cvetković. Classification, geometrical and kinematic analysis of four-bar linkages. pages 261–266, 01 2018.

A Animation UML State Diagram

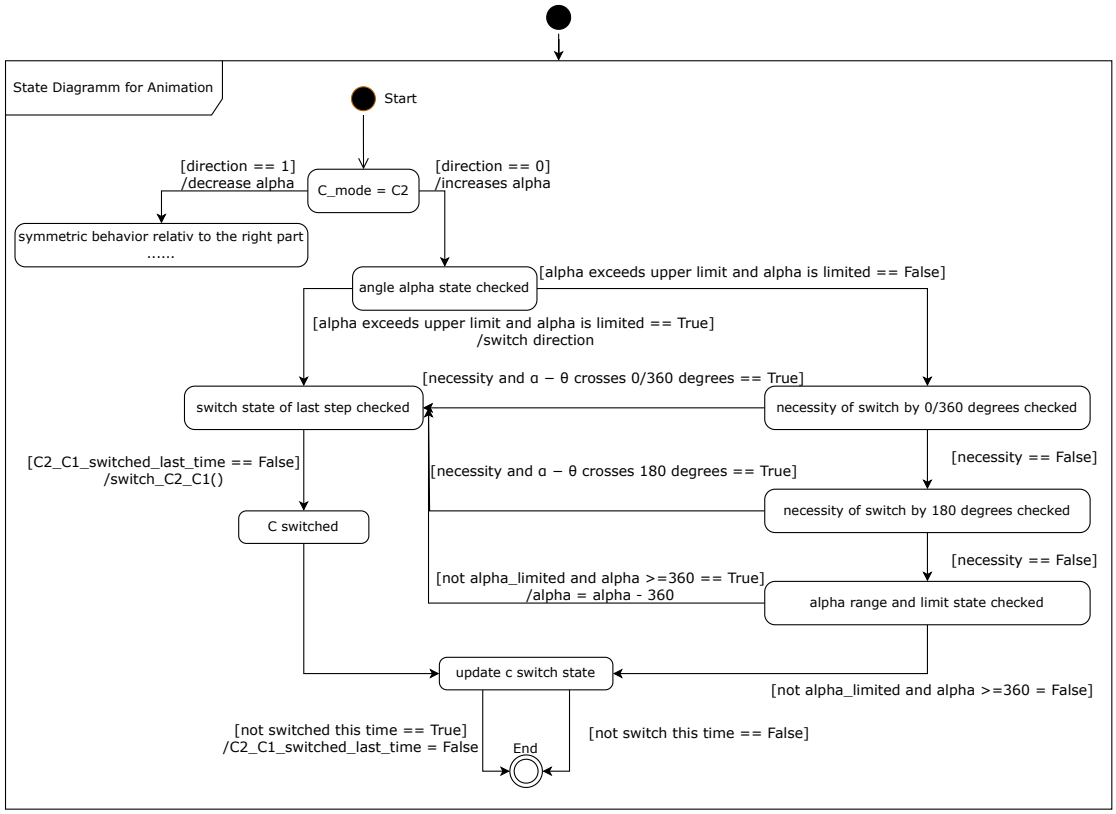


Figure 9: State Diagramm for Animation

B User Documentation

B.1 Installation

Install Python[12] and Python's libraries: tkinter[13], numpy[14], math[15], and unittest[16]. Note that tkinter, math and unittest are pre-installed with Python, while numpy needs to be installed separately.

B.2 Running

Download the gui.py[17] and four_bar_linkage.py[18] files in the same directory. Start the GUI by calling the following.

```
1 $ python ./gui.py
```

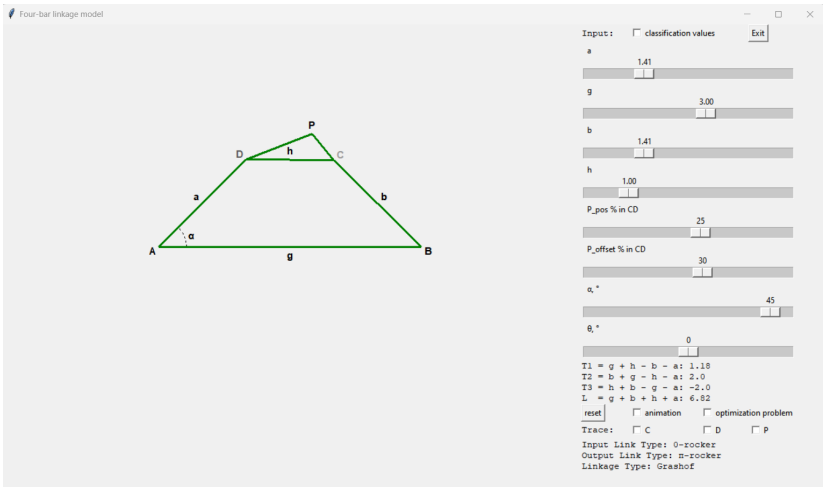
B.3 Testing

Download the following files in the same directory: 27Test.py[19] and motionTestCases.py[20]. Test the software by calling:

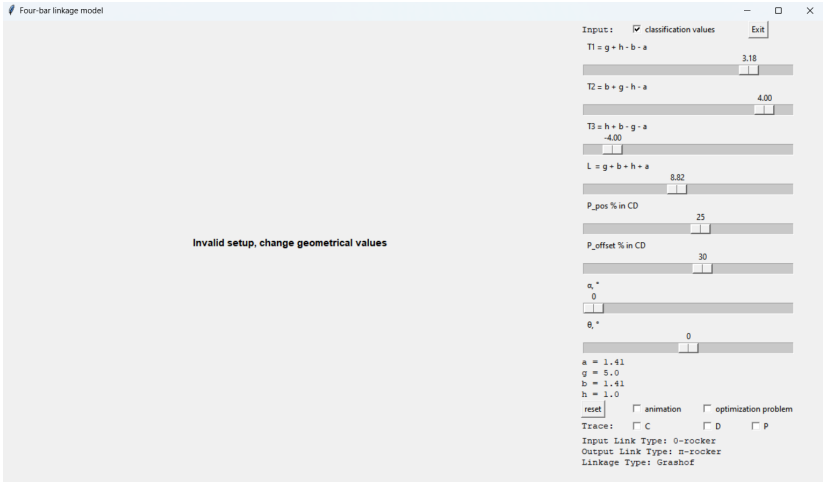
```
1 $ python ./motionTestCases.py
2 $ python ./27Test.py
```

C Screenshots

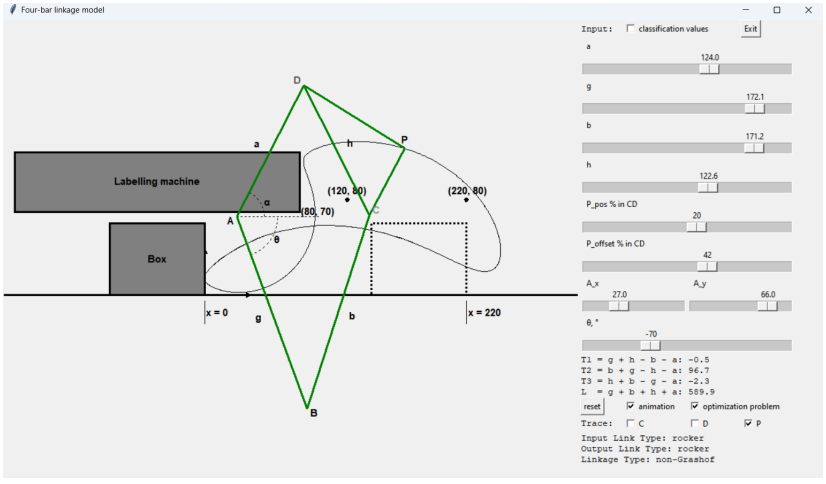
[12] https://www.python.org/downloads/
[13] https://docs.python.org/3/library/tkinter.html
[14] https://numpy.org/
[15] https://docs.python.org/3/library/math.html
[16] https://docs.python.org/3/library/unittest.html
[17] https://github.com/einsflash/Project_Pusher_Mechanism/blob/main/src/gui.py
[18] https://github.com/einsflash/Project_Pusher_Mechanism/blob/main/src/four_bar_linkage.py
[19] https://github.com/einsflash/Project_Pusher_Mechanism/blob/main/src/27Test.py
[20] https://github.com/einsflash/Project_Pusher_Mechanism/blob/main/src/motionTestCases.py



(a) GUI, default.

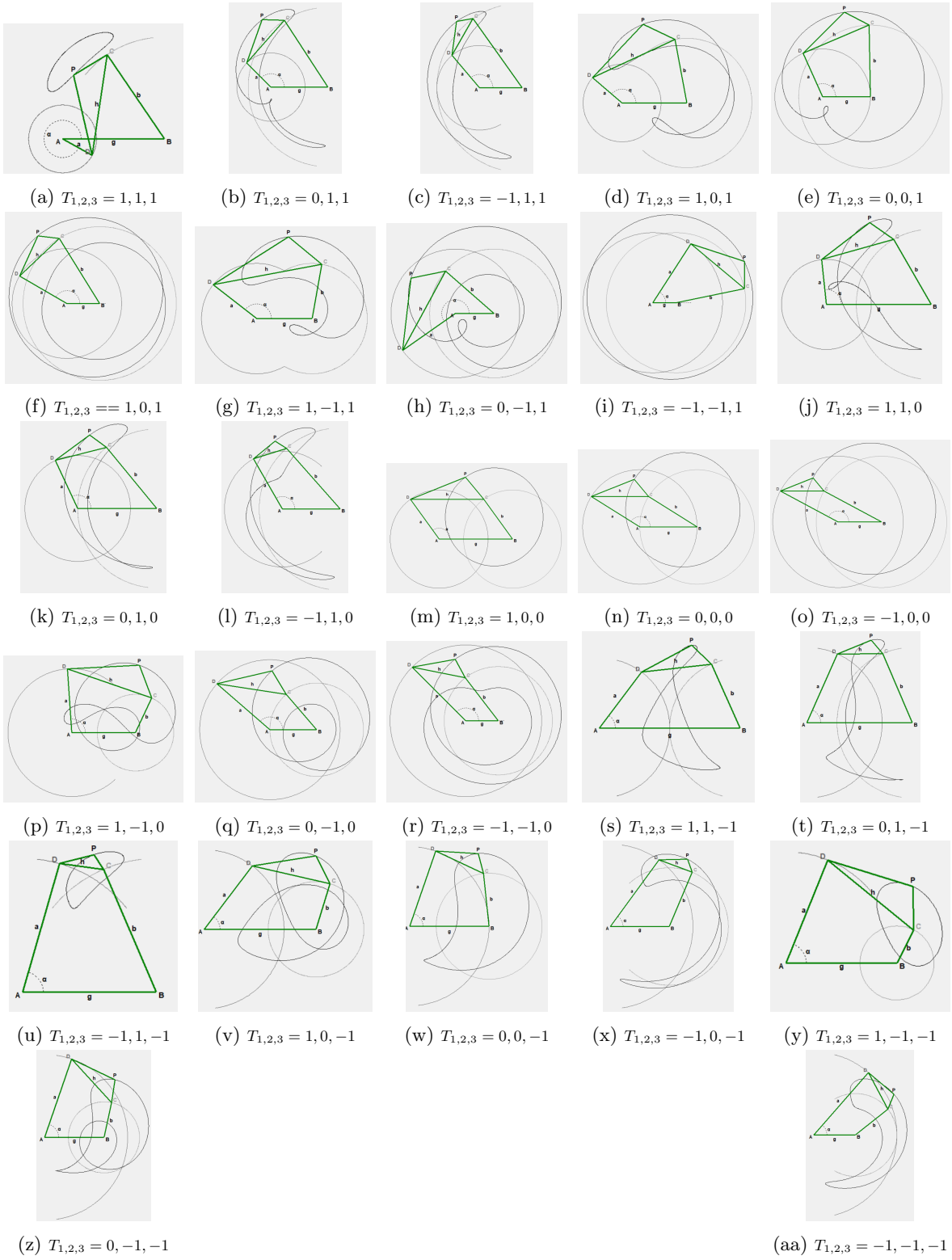


(b) GUI, invalid setup and classification values as the input.



(c) GUI, animation, tracing and optimization problem.

Figure 10: GUI screenshots



(a) $T_{1,2,3} = 1, 1, 1$ (b) $T_{1,2,3} = 0, 1, 1$ (c) $T_{1,2,3} = -1, 1, 1$ (d) $T_{1,2,3} = 1, 0, 1$ (e) $T_{1,2,3} = 0, 0, 1$

(f) $T_{1,2,3} == 1, 0, 1$ (g) $T_{1,2,3} = 1, -1, 1$ (h) $T_{1,2,3} = 0, -1, 1$ (i) $T_{1,2,3} = -1, -1, 1$ (j) $T_{1,2,3} = 1, 1, 0$

(k) $T_{1,2,3} = 0, 1, 0$ (l) $T_{1,2,3} = -1, 1, 0$ (m) $T_{1,2,3} = 1, 0, 0$ (n) $T_{1,2,3} = 0, 0, 0$ (o) $T_{1,2,3} = -1, 0, 0$

(p) $T_{1,2,3} = 1, -1, 0$ (q) $T_{1,2,3} = 0, -1, 0$ (r) $T_{1,2,3} = -1, -1, 0$ (s) $T_{1,2,3} = 1, 1, -1$ (t) $T_{1,2,3} = 0, 1, -1$

(u) $T_{1,2,3} = -1, 1, -1$ (v) $T_{1,2,3} = 1, 0, -1$ (w) $T_{1,2,3} = 0, 0, -1$ (x) $T_{1,2,3} = -1, 0, -1$ (y) $T_{1,2,3} = 1, -1, -1$

(z) $T_{1,2,3} = 0, -1, -1$ (aa) $T_{1,2,3} = -1, -1, -1$

Figure 11: 27 motion types.