

# Software Lab

## Computational Engineering Science

### Pusher Mechanism

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

Supervisor: Dr. rer. nat. Markus Towara\*

18<sup>th</sup> November 2024



---

\*Informatik 12: Software and Tools for Computational Engineering, RWTH Aachen University,  
[info@stce.rwth-aachen.de](mailto:info@stce.rwth-aachen.de)

# Preface

The topic "Pusher Mechanism" was assigned as a final project by the Department of Informatik 12: Software and Tools for Computational Engineering, RWTH Aachen University, for the Software Lab course in the Computational Engineering Science B.Sc. program. This work was carried out under the supervision of Dr. rer. nat. Markus Towara.

This project involves developing an enhanced version of the well-known planar four-bar linkage, a mechanical system widely used in applications like conveyor systems, oil well pumps, and robotic arms. By adding an extra joint, the extended mechanism offers more degrees of freedom, making it better suited for specific tasks. In this work, we designed and implemented this extended four-bar linkage to find a suitable mechanism for moving a box along a conveyor while avoiding obstacles.

In the first phase of the project, we analyzed the user requirements provided by our supervisor and broke them down into system requirements. This was followed by a theoretical analysis of the mechanism's geometry. Based on this analysis, we selected Python as the implementation environment due to its suitability for the task and the team's expertise.

The implementation consists of three interconnected components. First, the backend was developed to handle the geometry of the linkage, calculating the coordinates of all joints based on input parameters to ensure accurate modeling.

The second component is the frontend, a graphical user interface (GUI) created with the Tkinter<sup>1</sup> library. It enables users to visualize the linkage's movement, modify its parameters, and display essential information about the mechanism.

Additionally, a well-documented testing process was carried out to ensure the correctness and reliability of both the backend and frontend. This testing verified the system's performance across various scenarios, ensuring its accuracy and robustness.

With our implementation, we successfully addressed the optimization problem of moving a box along a conveyor while avoiding obstacles. The addition of an extra joint to the four-bar linkage provided the necessary degrees of freedom, enabling precise trajectory design. This solution met the system and user requirements and demonstrated the mechanism's effectiveness in achieving task-specific motion.

Furthermore, detailed documentation of the software and project management processes was created to enhance maintainability and offer a clear understanding of the project's structure. This documentation ensures that future developers or users can efficiently modify and extend the system. Overall, this work demonstrates the successful combination of theoretical analysis, design, and practical implementation in creating a functional pusher mechanism.

---

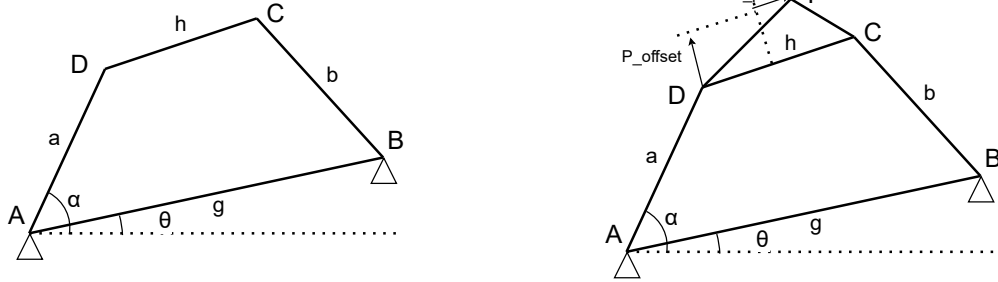
<sup>1</sup><https://docs.python.org/3/library/tkinter.html>

# 1 Introduction

The industrial revolution in the 18th and 19th centuries brought about significant advancements in manufacturing processes, one of which was the challenge of transporting products efficiently between various workstations in factories. A key solution to this challenge was the development of mechanical systems like the four-bar linkage, which can be used as a pusher mechanism to move products along production lines or between different conveyor systems. Despite its simple structure, the four-bar linkage has proven to be an effective mechanism in various industrial applications.

Over time, the four-bar linkage model has expanded beyond basic conveyor systems and has found applications in more complex systems, such as pumpjacks, robotic arms, and automotive engineering. Its simplicity and efficiency continue to make it relevant in modern mechanical design.

In this work, we aim to analyze the theoretical principles behind the four-bar linkage, design and implement an extended version of the mechanism with an additional joint, named coupler (*P*) (see Figures 1a and 1b), and apply it to solve the problem of moving a box along a production line while avoiding obstacles. By enhancing the classic four-bar linkage, we seek to provide a more flexible solution suited to complex real-world tasks.



(a) Planar four-bar linkage

(b) Planar four-bar linkage with coupler  $P$

Figure 1: Four-bar linkage

The structure of this paper is as follows: In Section 2, we analyze the user requirements, derive the system requirements, and provide an overview of the theoretical analysis of the mechanism's geometry. Section 3 covers the selection of the implementation environment, taking into account the system requirements and the team's expertise, as well as the preparation of UML class models for the implementation phase. Section 4 describes the implementation of the four-bar linkage, the graphical user interface, and the software testing process, while Section 5 provides software documentation to ensure its maintainability. In Section 6, we use the developed software to determine the appropriate mechanism parameters to move the box along the conveyor. Finally, in Sections 7, we discuss our 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

all above  
needs to be  
redacted  
without  
chatgpt

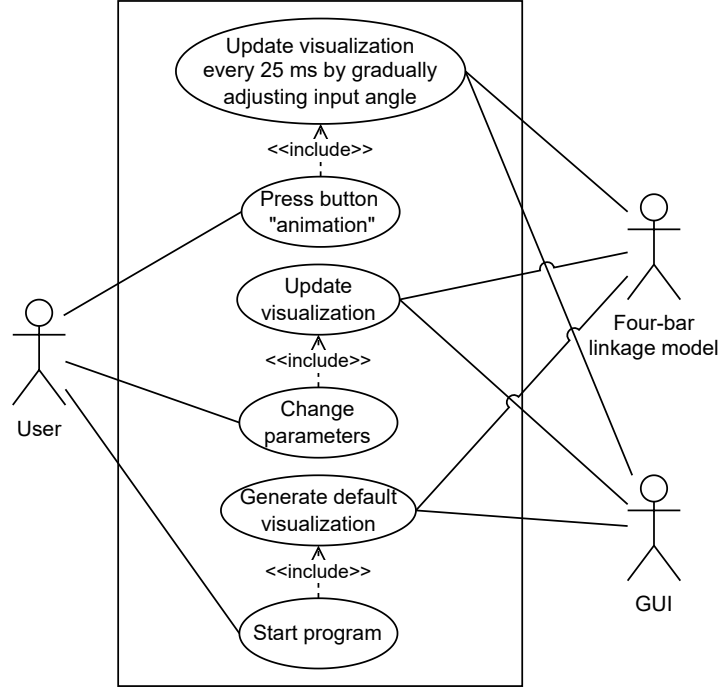


Figure 2: UML use case diagram

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.

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  $\theta$  between the fixed bar  $AB$  and the horizontal line.
- The input angle  $\alpha$  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  $\alpha$  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  $80 \times 60$  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  $(220, 80)$

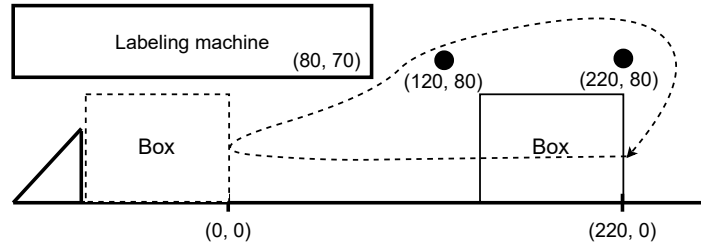


Figure 3: Optimization problem

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 backend 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 backend’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 (backend) 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 frontend 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  $\alpha$ .
- The angle  $\theta$  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.

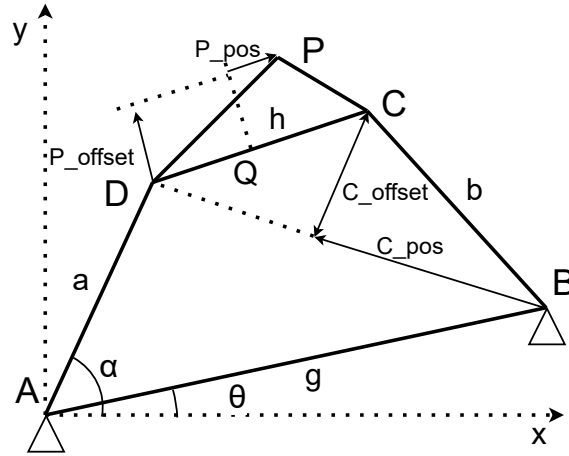


Figure 4: Planar four-bar linkage with coupler  $P$

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  $\theta$ :  $B_x = g \cos(\theta)$ ,  $B_y = g \sin(\theta)$ .

- Joint  $D$ .

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

- 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  $\vec{e}_{BD} = (e_{BD-x}, e_{BD-y}) = \frac{\overrightarrow{BD}}{|\overrightarrow{BD}|}$ .

The orthogonal direction is defined by a unit vector  $\vec{n}_{BD} = (-e_{BD-y}, e_{BD-x})$ .

We use the vectors  $\vec{e}_{BD}$  and  $\vec{n}_{BD}$  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  $\vec{n}_{BD}$ . To determine the position of  $C$  with respect to  $B$ , we also need the distance from  $C$  to  $B$  along  $\vec{e}_{BD}$ . The projection length of  $\overrightarrow{BC}$  onto direction of  $\vec{e}_{BD}$  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  $\vec{e}_{BD}$  is given by  $C_{pos} = \text{sign}(\cos(\angle CBD))\sqrt{b^2 - C_{offset}^2}$ .

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

$$\vec{C}_1 = (C_{1-x}, C_{1-y}) = \vec{B} + C_{pos}\vec{e}_{BD} + C_{offset}\vec{n}_{BD}$$

$$\vec{C}_2 = (C_{2-x}, C_{2-y}) = \vec{B} + C_{pos}\vec{e}_{BD} - C_{offset}\vec{n}_{BD}$$

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  $\alpha$ , 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  $\vec{e} = \text{sign}(\overrightarrow{BA} \cdot \overrightarrow{BC}) \frac{\overrightarrow{BA}}{|\overrightarrow{BA}|}$ , where  $\cdot$  is a scalar product. Then, the position of  $C$  is determined uniquely by  $\vec{C} = \vec{B} + b\vec{e}$ .

- 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  $\vec{Q} = \frac{\vec{C} + \vec{D}}{2}$ .

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

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

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  $\vec{n}_{DC}$ .



## 2.4 Parameter Validation

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

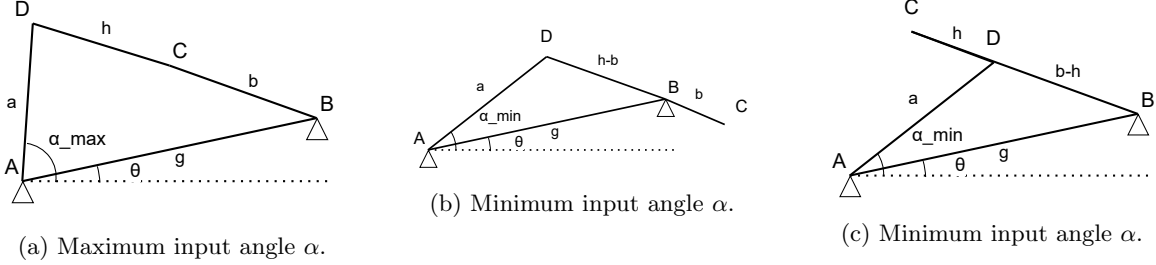


Figure 5: Constraints on the input angle  $\alpha$ .

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

- Maximum input angle  $\alpha$ .

Figure 5a displays the upper limit of the angle  $\alpha$ , 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  $\alpha$ :  $\alpha_{max} = \arccos\left(\frac{a^2 + g^2 - (h+b)^2}{2ag}\right) + \theta$ .

- Minimum input angle  $\alpha$ .

Figures 5b, 5c show two different cases for minimum input angle  $\alpha$ . 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  $\alpha$ :  $\alpha_{min} = \arccos\left(\frac{a^2 + g^2 - (h-b)^2}{2ag}\right) + \theta$ .

- 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  $\alpha$ .

If only the maximum limit exists, then the input angle  $\alpha$  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  $\alpha$  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  $\alpha$ , 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  $\alpha$ . 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  $\alpha$ , we can classify the motion of four-bar linkage.

Depending on the constraints on the input angle  $\alpha$  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  $\alpha$  (or the output angle  $\angle ABC$ ).
- *Rocker.*  
The link can rotate partially, with both a minimum and a maximum for the input angle  $\alpha$  (or the output angle  $\angle ABC$ ).
- *0-rocker.*  
The link can rotate partially, with no minimum but with maximum for the input angle  $\alpha$  (or the output angle  $\angle ABC$ ).
- *$\pi$ -rocker.*  
The link can rotate partially, with no maximum but with minimum for the input angle  $\alpha$  (or the output angle  $\angle ABC$ ).

## 2.6 27 Motion Cases

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  $\alpha$ .

This is our basic idea of iterative animation:

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

The sign here depends on the **direction** parameter. It determines whether  $\alpha$  changes clockwise (**direction** = 1) or counterclockwise (**direction** = 0).

- Use the updated  $\alpha$  value to calculate other parameters, such as the positions of the joints.

- If the input angle  $\alpha$  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  $\alpha$  always remains within the range  $[0^\circ, 360^\circ]$ . If the input angle exceeds  $360^\circ$  or falls below  $0^\circ$ , 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  $\alpha$  approaches  $-1$  or  $1$ , we incorporate floating point tolerance of  $10^{-10}$ .

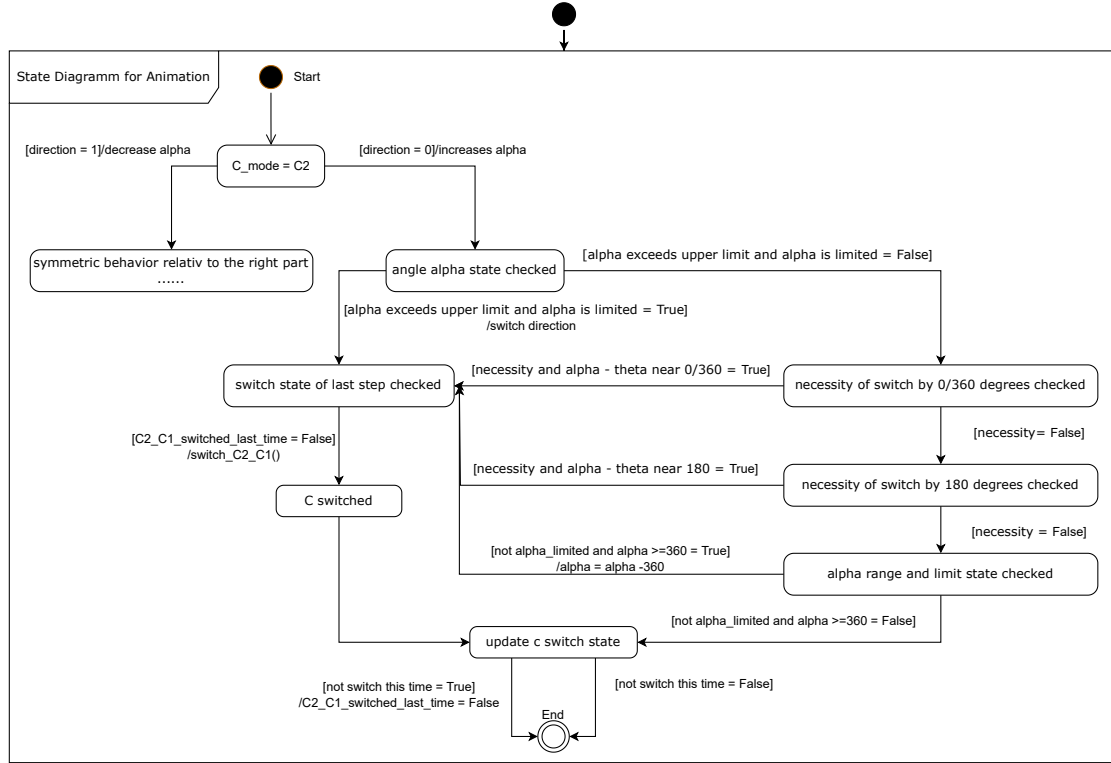


Figure 6: State Diagramm of Animation

### 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`<sup>2</sup> 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`<sup>3</sup>.
- 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<sup>4</sup> and PyCharm<sup>5</sup>.
- Package manager: Anaconda<sup>6</sup>.
- Libraries: `tkinter`<sup>3</sup>, `numpy`<sup>2</sup>, `math`<sup>7</sup>, `unittest`<sup>8</sup>.
- Version control system: GitHub repository<sup>9</sup>.
- Documentation: Pdoc<sup>10</sup>

#### 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 7 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<sup>11</sup>.

---

<sup>2</sup> <https://numpy.org/>

<sup>3</sup> <https://docs.python.org/3/library/tkinter.html>

<sup>4</sup> <https://www.spyder-ide.org/>

<sup>5</sup> <https://www.jetbrains.com/pycharm/>

<sup>6</sup> <https://www.anaconda.com/>

<sup>7</sup> <https://docs.python.org/3/library/math.html>

<sup>8</sup> <https://docs.python.org/3/library/unittest.html>

<sup>9</sup> [https://github.com/einsflash/Project\\_Pusher\\_Mechanism](https://github.com/einsflash/Project_Pusher_Mechanism)

<sup>10</sup> <https://pdoc.dev/docs/pdoc.html>

<sup>11</sup> [https://github.com/einsflash/Project\\_Pusher\\_Mechanism/blob/main/src/class\\_diagram.pdf](https://github.com/einsflash/Project_Pusher_Mechanism/blob/main/src/class_diagram.pdf)

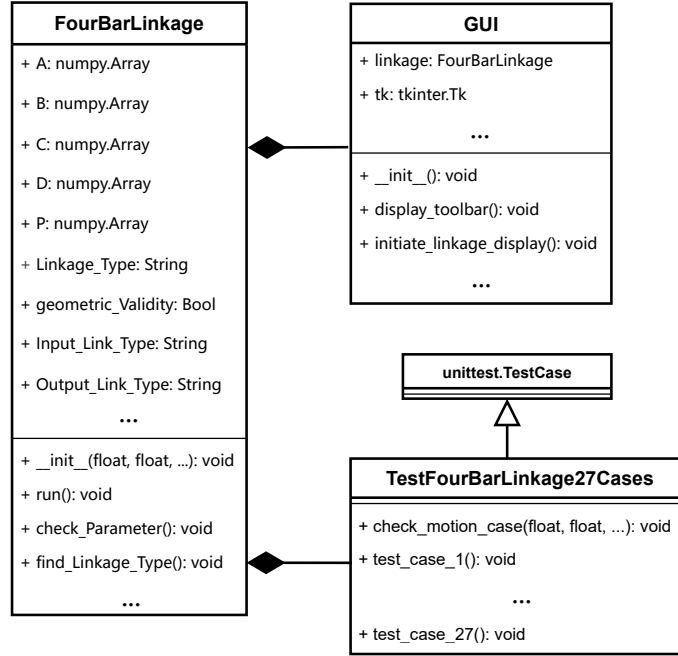


Figure 7: UML class diagram

The backend 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_Veracity`) to notify GUI about invalid parameters. The main member functions are the constructor `__init__` to set geometrical parameters, `run` to compute the joints' coordinates, `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 frontend 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 Backend

e.g. validation of input parameters

### 4.2 Frontend

overview of source code structure (file names, directories); build instructions; references into source code documentation e.g. doxygen<sup>12</sup>; short (!) code listings

```
1 #include<iostream>
2 int main() {
3     std::cout << "Leave me alone world!" << std::endl;
4     return 42;
5 }
```

if helpful (must come with detailed explanation)

### 4.3 Software Tests

e.g. googletest<sup>13</sup>

## 5 Documentation

## 6 Optimization Problem

## 7 Project Management

who did what, when, and why; organization of collaboration, i.e. [online] meetings, software version control (e.g. git<sup>14</sup>)

## 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 User Documentation

### A.1 Building

e.g. using cmake<sup>15</sup> and make<sup>16</sup>

### A.2 Testing

e.g. `make test`

---

<sup>12</sup><https://github.com/doxygen/doxygen>

<sup>13</sup><https://github.com/google/googletest>

<sup>14</sup><https://git.rwth-aachen.de>

<sup>15</sup><https://cmake.org/>

<sup>16</sup><https://www.gnu.org/software/make/>

### A.3 Running

documented sample session(s); e.g, `make run`