# 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 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[1] 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.

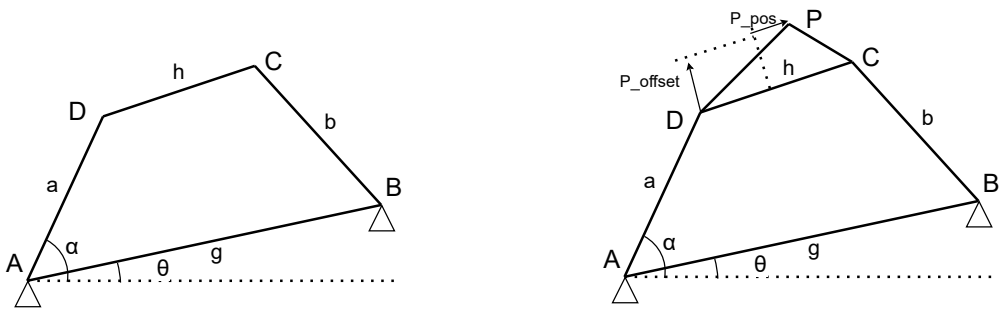
[1] https://docs.python.org/3/library/tkinter.html

# 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 (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 outline the overall vision for how a system should function and what features it must provide to meet user needs. These high-level expectations are the foundation for developers to

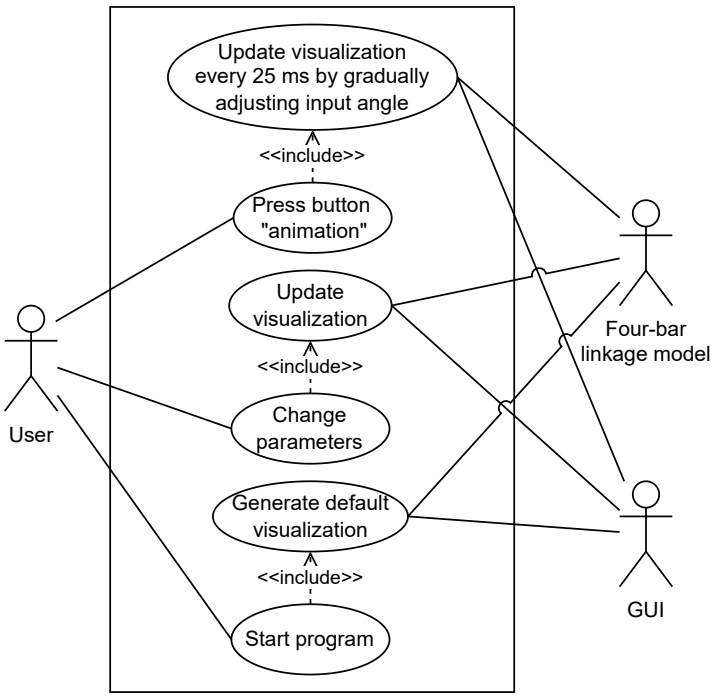



Figure 2: UML use case diagram

derive more detailed and technical system requirements, which define constraints and specifications the software must fulfill.

To enhance understanding of the user requirements described later, we provide a UML use case diagram of user-software interactions in Figure 2. The diagram illustrates the main workflow: after starting the program, the user is presented with a default visualization of the four-bar linkage generated by the GUI and the linkage model. The user can modify this visualization by specifying input parameters through the GUI. Additionally, the four-bar linkage can be animated, with the input angle gradually increasing and the visualization updating every 25 ms upon the user's explicit request.

The following list presents all the user requirements provided by our supervisor, along with our explanations and interpretations of each concept.

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

  The planar four-bar linkage extended with a coupler, illustrated in Figure 1b, may appear to be a straightforward geometric structure. However, its motion is more complex than it seems. While the coupler $P$ does not influence the primary motion constraints, the lengths of the four main bars define the limitations of the input angle $\alpha$. These constraints can result in the input angle being unlimited, symmetrically limited, or asymmetrically limited relative to the ground link $AB$. Consequently, the links $AD$ and $BC$ can function as either cranks, capable of full rotation, or rockers, which only partially rotate. In fact, as explained in [1], 27 distinct motion types for such mechanisms have been identified. A deeper analysis of these motion types will be conducted in the theoretical section of our study.

- *Requirement: Implement a graphical user interface (GUI) to display four-bar linkage animation and customize its geometric parameters.*

The GUI should enable users to visualize the linkage, provide smooth animations, and adjust various geometric parameters of the system. We have identified eight key parameters (degrees of freedom, shown in Figure 1b) 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, expressed by $P_{pos}$ and $P_{offset}$.

During the animation process, the input angle $\alpha$ is no longer a free parameter, as it is dynamically determined by the system to ensure smooth motion.

Additionally, while the position of point $A$ could be considered an independent parameter (technically two parameters in 2D), for simplicity, we assume it is fixed at $(0, 0)$ during geometric analysis. If point $A$ needs to be positioned elsewhere, the entire linkage can simply be translated accordingly. This fixed-point assumption will simplify the design phase but can be revisited for solving optimization problems later.

- *Requirement: Determine suitable parameters for the four-bar linkage to solve the following optimization problem:*
  - Push box with size $80 \times 60$ from $x = 220$ to $x = 0$
  - Do not cross the area of the labeling machine (Area with $x < 80$ and $y > 70$).
  - Pass above points $(120, 80)$ and $(220, 80)$

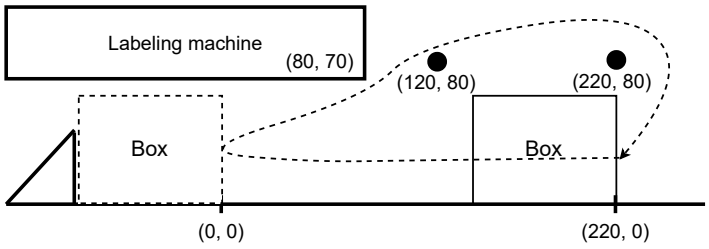



Figure 3: Optimization problem

The optimization problem is illustrated in Figure 3, which shows a conveyor line with the box to be moved by the coupler $P$ while avoiding the forbidden areas. The coupler $P$ must move the box, following a trajectory such as the dotted curve depicted in the figure. After pushing the box to the desired location, the coupler should return to its starting position, navigating around obstacles to repeat the task.

At a minimum, the problem can be addressed manually by experimenting with different parameters to find a feasible solution. Ideally, however, an algorithm could be developed to automate the optimization process and identify the best parameters efficiently.

## 2.2 System Requirements

After reviewing and analyzing the user requirements, which provide a high-level perspective of the problem and its solution, we need to derive more detailed technical requirements and specifications for

the software. The system requirements are categorized into two types: *functional requirements*, which define what the software must do, and *non-functional requirements*, which outline how the system should operate and perform.

We begin with the functional requirements, organized into several subtopics and accompanied by brief explanations for each.

- *Four-bar linkage model:*
  - The model implements the geometry of the four-bar linkage, calculating joint coordinates based on input parameters.
  - It simulates all 27 motion types of the four-bar linkage with a coupler.
  - It ensures stable operation without crashes, regardless of the input parameters.
  - It validates input data and sends error messages to the GUI for user feedback.

  The four-bar linkage model acts as the backend of the system, accurately implementing the geometry and all motion types of the linkage. Its primary role is to provide reliable data for visualization in the GUI while ensuring error-free operation. By validating input parameters and communicating issues through the GUI, the system allows users to address errors efficiently without needing to restart.

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

  To ensure the backend's accuracy, test cases must be implemented for each motion type, supported by reference data to validate the results.

- *Graphical User Interface (GUI):*
  - The GUI incorporates the four-bar linkage model (backend), utilizing its geometry and motion cases for visualization and animation.
  - It provides a visualization of the four-bar linkage.
  - It contains sliders to allow users to input and adjust geometric parameters.
  - It updates the visualization in real-time when new geometric parameters are provided.
  - It includes an animation mode for smooth motion visualization of the four-bar linkage.
  - It provides coupler tracing to display coupler's trajectory.

  The GUI serves as the user's primary interaction point (frontend), incorporating all functionalities relevant to user needs. Its main purpose is to visually represent the four-bar linkage using joint coordinates obtained from the backend. Users can adjust geometric parameters via sliders, with the visualization updating instantly to reflect changes. The GUI also includes an animation mode, ensuring smooth movement of the linkage. Additionally, the tracing of the coupler $P$ during animation is crucial for solving the optimization problem, providing valuable insights into its trajectory.

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

  Clear and comprehensive documentation is essential for maintaining software. To ensure the system remains reusable for future developers, we document all components in detail.

After discussing the functional system requirements, which outline what the system must do, we now turn to the non-functional requirements, which describe how the system should do it.

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

  Performance is a critical factor for usability, as users expect quick responses without noticeable delays. To meet this requirement, the four-bar linkage model and GUI should be optimized to ensure smooth animations at 30 frames per second on standard computers (e.g., with an AMD Ryzen 4500U chip). This ensures that the system remains free of lag, providing a smooth user experience.

all above needs to be redacted without chatgpt

## 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.

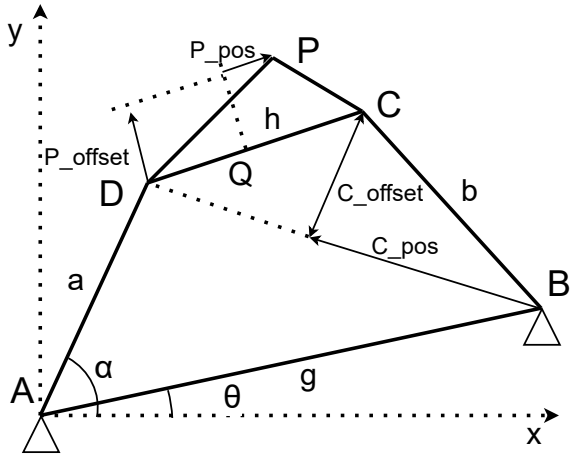



Figure 4: Planar four-bar linkage with coupler $P$

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 $\overrightarrow{e}_{BD} = (e_{BD\_x}, e_{BD\_y}) = \frac{\overrightarrow{BD}}{|\overrightarrow{BD}|}$.

  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:

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

  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 $\alpha$, 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 $\overrightarrow{e} = sign(\overrightarrow{BA} \cdot \overrightarrow{BC}) \frac{\overrightarrow{BA}}{g}$, 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 Classification

crank, rocker

## 2.5 27 movement cases

## 2.6 Animation

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

- `[t]`: Time interval of the animation.
- `[alpha_velocity]`: Angular velocity of the input angle $\alpha$.

This is our basic idea of iterative animation:

- *Basic Idea:*

  1. Calculate the limit values of $\alpha$ based on the geometry.
  2. Incrementally update $\alpha$ using the following formula:

     $$\alpha = \alpha \pm \texttt{alpha\_velocity} \times \texttt{t}$$

     The sign here depends on the `[direction]` parameter. It determines whether $\alpha$ changes clockwise (`direction = 1`) or counterclockwise (`direction = 0`).
  3. Use the updated $\alpha$ value to calculate other parameters, such as the positions of the joints. Ensure that $\alpha$ always remains within the range $[0°, 360°]$. If it exceeds 360° or falls below 0°, we will reset it to the corresponding boundary value.
  4. Implement a switching algorithm to choose point $C$ between $C_1$ and $C_2$ when necessary. This can ensure continuity of animation.

- *Animation Parameters:* We design these parameters for animation:

- `[direction]`: Indicates whether $\alpha$ changes clockwise (`1`) or counterclockwise (`0`).
- `[C_mode]`: Determines which $C$ ($C_1$ or $C_2$) is chosen for the animation.
- `[C2_C1_switched_last_time]`: This avoids redundant switching of point $C$ during updates.
- `[switch_C2_C1_360]` and `[switch_C2_C1_180]`: These two parameters are responsible for situations when $\alpha$ reaches its limit values (close to $0°$, $180°$, or $360°$). Switching point $C$ at these values ensures continuity in the animation. Because the limit values often correspond to collinear configurations of points $B$, $C$, and $D$.
- `[alpha_limited]`: Indicates whether $\alpha$ is restricted or not.

- *Floating Point Precision:* To avoid issues caused by floating point inaccuracies when checking if the cosine value of $\alpha$ approaches $-1$ or 1, we use floating point tolerance of $10^{-10}$. When the angle between $AD$ and $AB$ is very close to $0°$, $180°$, or $360°$, the program needs to switch between $C_1$ and $C_2$ for continuity 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[2] 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[3].
- 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[4] and PyCharm[5].
- Package manager: Anaconda[6].
- Libraries: tkinter[3], numpy[2], math[7], unittest[8].
- Version control system: GitHub repository[9].
- Documentation: Pdoc[10]

[2] https://numpy.org/
[3] https://docs.python.org/3/library/tkinter.html
[4] https://www.spyder-ide.org/
[5] https://www.jetbrains.com/pycharm/
[6] https://www.anaconda.com/
[7] https://docs.python.org/3/library/math.html
[8] https://docs.python.org/3/library/unittest.html
[9] `https://github.com/einsflash/Project_Pusher_Mechanism`
[10] https://pdoc.dev/docs/pdoc.html

## 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.

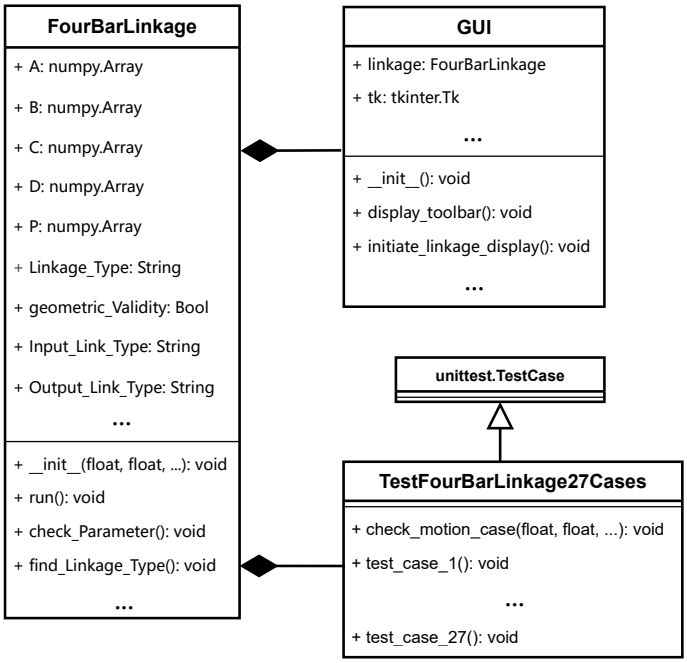



Figure 5: UML class diagram

Figure 5 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[11].

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_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, `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 vi-

[11] `https://github.com/einsflash/Project_Pusher_Mechanism/blob/main/src/class_diagram.pdf`

sualization 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[12]; 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[13]

# 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[14]

# 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.

[12] `https://github.com/doxygen/doxygen`
[13] `https://github.com/google/googletest`
[14] `https://git.rwth-aachen.de`

# A User Documentation

## A.1 Building

e.g, using cmake[15] and make[16]

## A.2 Testing

e.g, `make test`

## A.3 Running

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

[15] `https://cmake.org/`
[16] `https://www.gnu.org/software/make/`