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`

Contents

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.

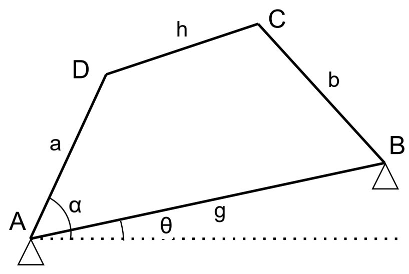
[1]https://docs.python.org/3/library/tkinter.html

1 Introduction

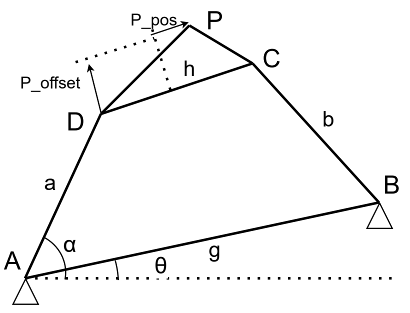
The industrial revolution in the 18th and 19th centuries brought about significant advancements in manufacturing processes, one of which was the challenge of transporting products efficiently between various workstations in factories. A key solution to this challenge was the development of mechanical 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

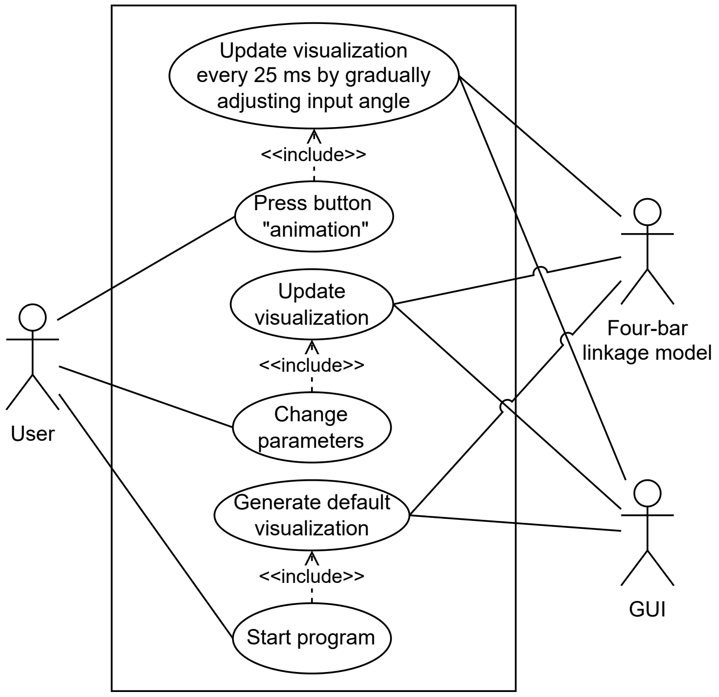


Figure 2: UML use case diagram

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 α. 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 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, expressed by P_{pos} and P_{offset}.

During the animation process, the input angle α 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 80×60 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 $(220, 80)$

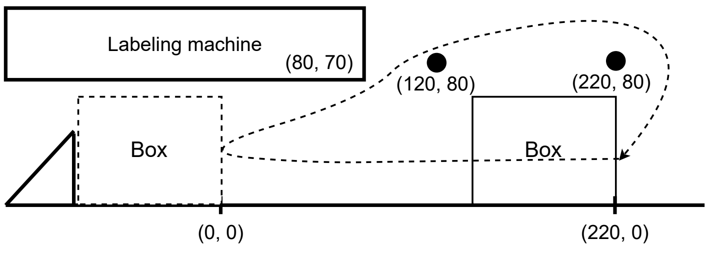


Figure 3: Optimization problem

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 below should be corrected using LLM

2.3 Geometry

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

Now our aim is to calculate the positions of all the joints based on the input parameters, namely lengths of main four bars AB, BC, CD, AD, that we denoted as g, b, h, a, input angle α, angle θ between the horizontal line and AB, and the position of coupler P defined with respect to middle point of CD and denoted as P_{pos} and P_{offset}, note that these two values can be also negative, take a look at the direction of corresponding arrows at the picture. All the input parameters are depicted in Figure 4.

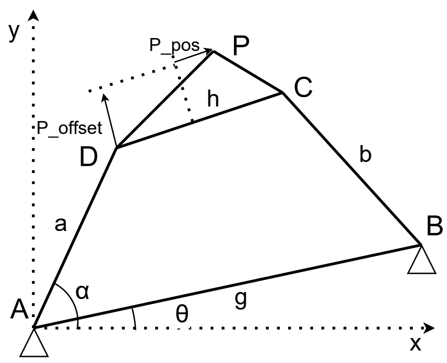


Figure 4: Planar four-bar linkage with coupler P

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

- Joint A.

  As was already mentioned for simplicity we put joint A in the point $(0, 0)$. If for some reasons it should be placed in another place, the whole linkage should be just translated, by simply adding to each joint coordinates the coordinates of A.

- Joint B.

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

- Joint D.

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

- Joint C.

  The most difficult geometrical problem is to determine the position of joint C. For this purpose we consider a triangle $\triangle BCD$. The main idea is to derive position of C by looking at the area of the triangle using different approaches.

  Define a vector $\overrightarrow{BD} = (BD_x, BD_y) = \overrightarrow{D} - \overrightarrow{B} = (D_x - B_x, D_y - B_y)$. Assume firstly that this vector is not zero-vector with length $|\overrightarrow{BD}| = \sqrt{(BD_x)^2 + (BD_y)^2}$.

  Then a unit vector along $\overrightarrow{BD}$ is $\overrightarrow{e}_{BD} = (e_{BD-x}, e_{BD-y}) = \overrightarrow{BD}/|\overrightarrow{BD}|$.

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

  The area of $\triangle BCD$ can be determined by Heron's formula:

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

  At the other hand the area of $\triangle BCD$ can be determined using length of BD and the distance from C to BD denoted as C_{offset}: $A_{\triangle BCD} = |\overrightarrow{BD}| \cdot C_{offset}/2$. So we can determine $C_{offset} = 2 \cdot A_{\triangle BCD}/|\overrightarrow{BD}|$

  Now we now the distance of joint C to BD along $\overrightarrow{n}_{BD}$. But we want to determine the position of C with respect to B. For this purpose we need also to determine the distance in the orthogonal direction $\overrightarrow{e}_{BD}$, to do that we project $\overrightarrow{BC}$ onto $\overrightarrow{BD}$. The projection length is $|C_{pos}| = \sqrt{b^2 - C_{offset}^2}$. The question is which sign do C_{pos} have. It can be determined using angle $\angle CBD$. With cosine rule:

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

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

  As long as C_{pos} and C_{offset} are determined in the orthogonal directions, we can find out the position of the C with respect to B just summing it up multiplied by the corresponding unite vectors. The problem arises, when trying to find the position of the C, because in contrast to vector $\overrightarrow{e}_{BD}$, that is uniquely defined, vector $\overrightarrow{n}_{BD}$ could be also with the opposite direction $-\overrightarrow{n}_{BD}$, which implies two possible positions of C, that are symmetric with respect to BD:

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

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

  The question is how to choose between C_1 and C_2. For the static case it does not play a role, because both positions are possible, so we decided to take $C = C_2$ as a default one. For the animation, that is discussed a bit later, the selection between C_1 and C_2 is not so straight forward.

  In the discussion above we made an assumption, that $|\overrightarrow{BD}|$ is not zero, but it can be the case, when $b = h$ and specific input angle α is given. In this case joints A, B, C, D are at the same line, so we define a unite vector $\overrightarrow{e} = sign(< \overrightarrow{BA}, \overrightarrow{BC} >) \cdot \overrightarrow{BA}/g$, where $< \cdot, \cdot >$ is a scalar multiplication. Then position of C is determined uniquely by $\overrightarrow{C} = \overrightarrow{B} + b \cdot \overrightarrow{e}$

- Coupler P.

  As long as we know the positions of C and D, it is trivial to determine the position of P.

  Determine the middle point of CD as $\overrightarrow{Q} = (\overrightarrow{C} + \overrightarrow{D})/2$.

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

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

One could say, that like for the case of joint C, here is also two different positions of P, because $\overrightarrow{n}_{DC}$ can be determined in the opposite direction, but the difference is, that unlike C_{offset}, P_{offset} can be specified negative by user, so the problem of the opposite direction of the normal vector is not a problem any more.

2.4 Animation

3 Design

3.1 Principal Components and Third-Party Software

libraries that you built on explained briefly and references to further information

3.2 Class Models

UML Class diagram(s) and description; should link into overall design through reference of application programming interfaces (API) of third-party software

4 Implementation

4.1 Development Infrastructure

programming language, compiler, run time libraries, target platform (hardware, operating system)

4.2 Source Code

overview of source code structure (file names, directories); build instructions; references into source code documentation e.g, doxygen[2]; short (!) code listings

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

if helpful (must come with detailed explanation)

4.3 Software Tests

e.g, googletest[3]

[2]https://github.com/doxygen/doxygen
[3]https://github.com/google/googletest

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[4]

8 Acknowledgment

Use of ChatGPT for grammatical correctness.

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[5] and make[6]

A.2 Testing

e.g, `make test`

A.3 Running

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

[4] `https://git.rwth-aachen.de`
[5] `https://cmake.org/`
[6] `https://www.gnu.org/software/make/`