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.

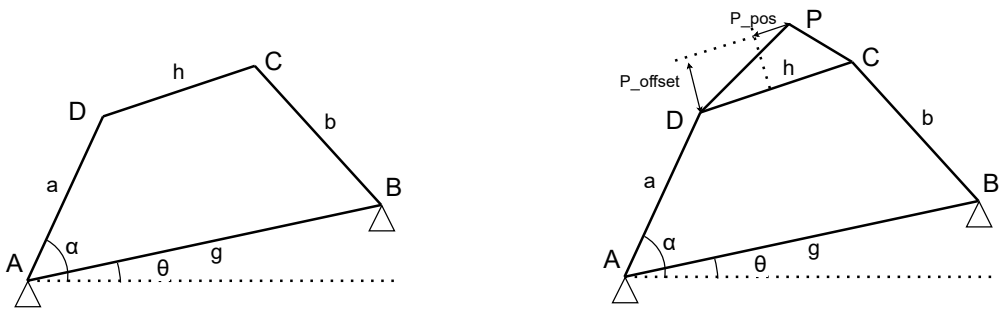
[1] https://docs.python.org/3/library/tkinter.html

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

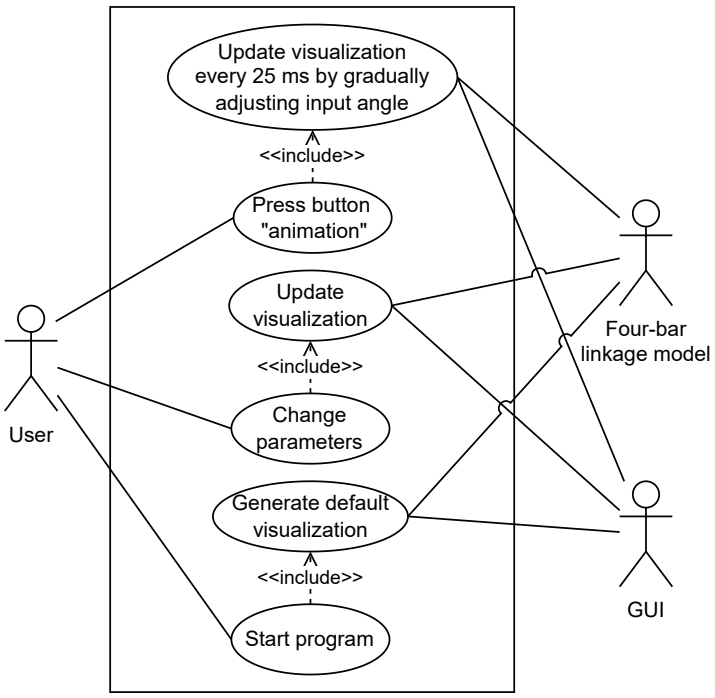


Figure 2: UML use case diagram

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) 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.

- Find and provide suitable parameters for the pusher mechanism to meet requirements of 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 $(220, 80)$

2.2 System Requirements

Functional

- **Four-bar linkage model**:
  - System simulates all 27 motion types of the four-bar linkage with coupler.
  - System does not crash with any parameter input.
  - System validates input data.
- **Tests**:
  - Implement test cases to cover all motion cases.
  - Provide reference data to compare results with.
- **Graphical User Interface**:
  - GUI is coupled with the four-bar linkage model to use implemented motion cases for animation.
  - GUI provides the four-bar linkage visualization.
  - GUI has slide bars to input geometrical data.
  - GUI reacts to new input data by changing the four-bar linkage visualization accordingly.
  - GUI provides animation of the four-bar linkage motion.
  - GUI provides tracing of the coupler to show its trajectory.
- **Optimization problem**:

  - GUI visualizes the solution.

Non-functional

- **Performance**:
  - The four-bar linkage model is fast enough to provide smooth GUI animations.
  - GUI animations are not slower than 30 frames per second.
- **Usability**:
  - GUI and four bar linkage implementation are well-documented.

2.3 Theory

explain geometry

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

```
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[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]

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.

[2] Adam Ries. *Rechenung auff der Linihen und Federn.* Annaberg, 1522.

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/`