TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

ACTA TECHNICA NAPOCENSIS

Series: Applied Mathematics, Mechanics, and Engineering
Vol. 68, Issue I, March, 2025

PYTHON IN THE PLANAR FOUR-BAR LINKAGE MECHANISM SIMULATION

Tiberiu Alexandru ANTAL

Abstract: *The Python programming language together with the object-oriented programming paradigm are used to write an application using the Matplot library in Python. Using the quiver graphical concept a vector representation of the mechanism is given while the trajectory results in a format that can be used later in trajectory classifications of the four-bar linkage mechanism.*

Key words: *four-bar linkage, quiver, Java, Python, simulation, trajectory.*

1. INTRODUCTION

The classical mechanics is the branch of physics that studies the behavior of physical bodies under the influence of forces, displacements, and constraints. Mechanisms are a subset of mechanics focused on the design and analysis of machines and moving systems. Mechanics provides the foundation, while mechanisms are practical implementations [1], [2]. The use of high level programming languages in the study, simulation, and implementation of mechanisms has evolved over time. Each stage of development has been influenced by computational capabilities of the period, and the existing programming paradigms. During the *Early Era* (1950s–1970s) the focus was on numerical methods and basic simulations (no GUIs) for mechanism analysis using the procedural paradigm with the FORTAN (1957) programming language. This is followed by the *CAD/CAE Tools Era* (1980s–1990s) where tools for computational design integrated with graphical visualization (e.g. SolidWorks, AutoCAD) were created and the new high level languages as C (1972), C++ (1985) and Pascal (1970) used the structured, modular and object oriented paradigms to create reusable simulation software applied in mechanism synthesis and early robotics. The *Modern Era* (2000s–Present) is focusing on high-performance computing, automation, and integration with AI/ML using the object-oriented, functional, and scripting paradigms in advance robotics and AI integration [3], [4]. Some of the new key languages that emerged to cover the mechanisms fields are: Python, Java, Julia and Rust. Of these languages, the easiest for a novice to approach is Python. Python's syntax is simple, making it accessible for engineers and researchers. Libraries like SymPy, PyDy, NumPy, and Matplotlib provide tools for symbolic calculations, numerical simulations, and visualization. Python integrates well with CAD tools and other simulation software through APIs. This is why Python's versatility and extensive ecosystem make it an excellent choice for simulating mechanisms, robots and gears. However, each language has its strengths and weaknesses depending on the use case, for Python the code execution is slower because it is an interpreted language. From an educational point of view, the language forms skills that are not found in other languages, and when switching to other languages, this can be an inconvenience in understanding and applying other languages, as follows: Python doesn’t require explicit variable type declarations (like Java); the dynamic typing specific to the

Received: 29.01.25; **Similarities**: 05.03.25: **Reviewed**: 20.02./19.02.25: **Accepted**:06.03.25.

language can lead to runtime errors that static typing (like Java) would catch during compilation; the Object-Oriented Programming (OOP) paradigm is supported but not enforced, and this can lead to inconsistent practices for beginners.

2. PYTHON and ANACONDA

The Python programming language can be installed from the official Python website found at *python.org*. Designed as a high-level, general-purpose programming language it is known as the core used for many types of software development tools, including scripting, data analysis, machine learning, and extensible with third-party libraries (like NumPy, Pandas, TensorFlow). The module [5] in Python is a file containing Python code (functions, classes, and variables) that can be reused in other Python programs. The modules can be build-in (preinstalled with Python as `math`, `os`, `sys`, `random`), user-defined (creates by the programmer and stored as `.py` files) and third-party (developed by others, available via the Python Package Index - PyPI and installed using a package manager like `pip`). Modules help organize and reuse Python code. Anaconda [6] is a distribution of Python designed for data science, machine learning, and scientific computing. It simplifies the setup and management of Python environments and includes tools for working with data-heavy applications. A package manager is a tool that automates the process of installing, updating, configuring, and managing software packages, including libraries, dependencies, and tools. The Conda [7] package manager that comes with the Anaconda distribution allows creation of Anaconda environments that are isolated being self-contained spaces where you can install and manage Python packages, dependencies, and configurations without affecting other environments or the system Python installations.

3. ENVIRONEMENT SETUP for PYTHON

A programming IDE (Integrated Development Environment) is a software tool to help developers write/edit, translate to machine language, debug, and manage more efficiently the code development process. Microsoft Visual Studio Code (VS Code) is a free, cross platform, highly extensible, customizable, source-code editor developed by Microsoft that can be downloaded from the following link: https://code.visualstudio.com/. Once the Python Extensions in VS Code are added Python environments can be managed and Python code can be executed and debugged from the editor. As VS Code also supports the Jupyter Notebook the *.ipynb* extension files can be used directly in the editor. A Jupyter Notebook is an open-source web-based application that allows users to create and share documents containing live code, equations, visualizations, and explanatory text. The advantage of Jupiter Notebook is that a full paper or report can be created containing the text, the mathematical formulae and the code (in Python) that implements the mathematics as well together with any results produce by the code (text, graphics or simulations) in a single document that can be shared or use for educational purposes easily. One document contains everything, the problem description, the solution (mathematics and code), and the results. For this purpose the Jupyter Notebook is using the cell which is a container where we write content such as code, text, or visualizations. Cells are the building blocks of Jupyter Notebooks, allowing users to interactively develop and document their code. There are three main types of cells in Jupyter Notebook:

1. **Code Cell**: to execute programming code; when we run the cell, the code is executed, and the output is displayed directly below the cell.
2. **Markdown Cell**: to write formatted text using Markdown (a lightweight markup language for creating formatted text using a plain-text editor that is described at https://www.markdownguide.org/) to add headings, lists, links, code snippets, or mathematical equations using LaTeX.
3. **Raw Cell**: a piece of content that is not executed or rendered being kept as-is without processing.

The Cell has two modes:

1. **Edit Mode**: to edit the cell's content, activated by pressing Enter or clicking inside a cell.
2. **Command Mode**: to manage cells (add, delete, run, etc.) without editing their content, activated by pressing Esc.

Some common Cell operations are:

1. **Run a Cell**: Press Shift+Enter to execute a cell.
2. **Add a New Cell**: Use the + button in the toolbar or press B (below) or A (above) in command mode.
3. **Change Cell Type**:
   a. From the toolbar dropdown.
   b. Shortcut: M for Markdown or Y for Code in command mode.

In Figure 1, VS Code is installed with most Python extension with Jupyter Notebook and the current Cell is in Edit Mode (Enter). This allows viewing how the Markdown language is used to provide explanation related to the problem and the code.

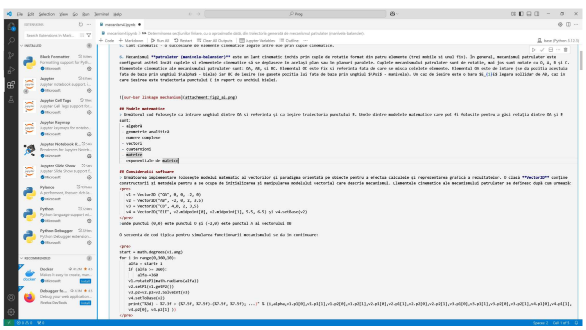

Fig. 1. - VS Code with Extension View and in Edit Mode Cell.

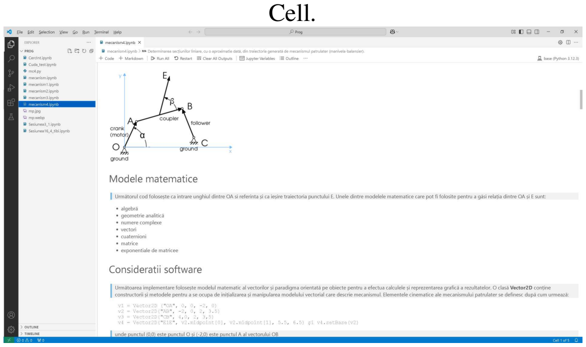

Fig. 2. - VS Code with Explorer View and in Stop Editing Mode Cell.

In Figure 2, VS Code the Jupyter Notebook is in Stop Editing Cell Mode (Ctrl+Alt+Enter), and the document reveals it's formatted view based on Markdown.

4. JAVA vs PYTON

In [16] the Java programming language is used to analyze the four-bar linkage mechanism and a library is provided for this. This paper continues with a Python implementation and Java vs Python comparison to clarify why the Python approach is better if ML should be applied to the computed trajectories.

Table 1

A brief Java ([13], [14]) vs Python comparison.

Aspect	Java	Python
Typing	Statically typed (variable types must be declared).	Dynamically typed (variable types are inferred).
Syntax	Strict and detailed; requires semicolons and braces.	Simple and concise; no semicolons or braces required.
Compilation	Compiled into bytecode and run on the Java Virtual Machine (JVM) - faster.	Interpreted, although Python code is compiled into bytecode before execution - slower.
Performance	Faster for large-scale applications due to static typing and optimization.	Slower compared to Java due to dynamic typing but sufficient for most tasks.
Learning	Requires more effort due to its strict syntax and broadness.	Easier for beginners because of its simplicity and readability.
Paradigms	Primarily Object-Oriented, with support for generic, procedural and functional programming.	Multi-paradigm: supports object-oriented, procedural, and functional programming.
Readability	Detailed, requires additional repetitive code.	High readability, concise syntax, and clean code structure.
Community and Libraries	Large community with extensive libraries for enterprise applications.	Large community with extensive libraries for data science, **machine learning**, and more.

As the application provided to solve the problem in Python is Object-Oriented (OO) a comparison between the two languages from this point of view would also be welcomed. OO Programming (OOP) is a paradigm that organizes code around data using the concept of objects and their interactions. Both Java and Python have language constructs to cover the

OOP paradigm, however they have notable differences in syntax, implementation, and features. The following table (Table 2) is covering the key differences related to OOP in Java vs. Python.

Table 2

Key differences: OOP Java ([9], [10]) vs Python.

Feature	Java	Python
Class Definition	Must explicitly define the class and its members.	Similar but more flexible and requires less code.
Inheritance	Supports single and multilevel inheritance but not multiple inheritance (uses interfaces).	Supports single, multilevel, and multiple inheritance directly.
Access Modifiers	Uses public, protected, and private for encapsulation.	No strict access modifiers; uses naming conventions (e.g., _ for protected).
Constructors	Special method with the same name as the class.	Uses __init__ as the constructor method.
Abstract Classes	Supported via the abstract keyword.	Supported using the abc module.
Interfaces	Separate keyword (interface), multiple interfaces can be implemented.	Achieved using abstract base classes or multiple inheritance.
Method Overloading	Supported through different parameter lists.	Not natively supported but can be simulated with default or variable-length arguments.
Method Overriding	Explicitly defined using @Override annotation.	Implicitly allowed without special syntax.
Polymorphism	Enforced strictly with defined types.	Dynamically typed, allowing more flexibility.

5. THE FOUR-BLINKAGE MECHANISM OOP IMPLENENTATION in PYTON

Based on the mathematical model from [16] the following code is using the Python quiver concept to draw and simulate de mechanism. Many researches [8], [10], [11] and public codes [9] are available on this subject; some are made in Python [8], while others are in Java [14], [16] or AutoLISP [15]. This research is based on quiver which is plot type that displays vectors as arrows. It is often used in scientific computing and data visualization to represent vector fields. The Matplotlib library provides a function called quiver() to create quiver plots. The simplest signature of the function is:

```
quiver([X, Y], U, V, [C])
```

where, X, Y - are the coordinates of the origins; U, V - the components of the vector field, representing the direction and magnitude of the arrows and C (Optional) - a color map to represent additional data (e.g., magnitude).

The computation and the simulation are based on the `Vector2D` class that implements all the required methods. The constructor of the `Vector2D` class is:

```
def __init__(self, id, x1:float, y1:float,
x2:float, y2:float):
 self.id = id ; self.p1 = np.array([x1,
y1], dtype=np.float64)
 self.p2 = np.array([x2, y2],
dtype=np.float64); self.midpoint()
 self.direction = None; self.mag =
self.Mag(); self.ang = self.Ang()
 self.q = plt.quiver(self.p1[0],
self.p1[1], -self.p1[0] + self.p2[0],  -
self.p1[1] + self.p2[1], angles="xy",
scale_units="xy", scale=1)
```

The instantiations of the vectors that make up the mechanism are:

```
v1 = Vector2D("a", 0, 0, -2, 0)
v2 = Vector2D("b", -2, 0, 2, 3.5)
v3 = Vector2D("c", 4,0, 2, 3.5)
v4 = Vector2D("d", v2.midpoint[0],
v2.midpoint[1], 5.5, 6.5)
v4.setBase(v2)
```

This will create the mechanism representation from Figure 3.

While the numerical simulation of the mechanism is done by running the following for loop:

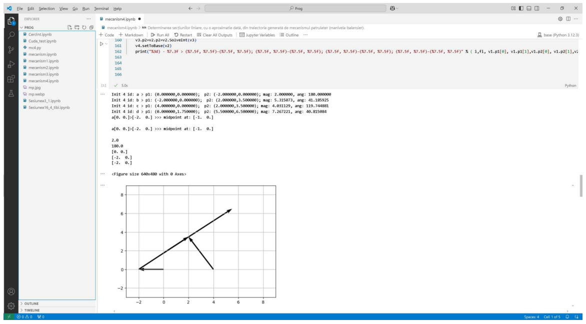

Fig. 3. - Quiver representation of the four-bar linkage mechanism in Python.

```
for i in range(0,360,10):
 fi = start+ i
 if (fi >= 360):
  fi-=360
 v1.rotateP1(math.radians(fi))
 v2.setP1(v1.getP2())
 v3.p2=v2.p2=v2.SolveInt(v3)
 v4.setToBase(v2)
```

The graphical simulation is based on the following animation function named `ani(fi)` that removes the previous drawn vectors and plots them for a new position given by `fi` parameter and draws the trajectory of the mechanism from `xdata,ydata`.

```
def ani(fi):
 global v1,v2,v3,v4
 v1.remove();v2.remove();v3.remove();
 v4.remove();v1.rotateP1(math.radians(fi))
 v2.setP1(v1.getP2());
v3.p2=v2.p2=v2.SolveInt(v3);v4.setToBase(v
2); v1.draw(plt);v2.draw(plt);
v3.draw(plt); v4.draw(plt);
 xdata.append(v4.p2[0]);
ydata.append(v4.p2[1])
 line.set_data(xdata,ydata); return line,
```

While the simulation code, running in different Cell is :

```
figure, ax =
plt.subplots(figsize=(10,5), dpi=100)
ax.set_aspect(1)
ax.set_xlim(-3, 11);ax.set_ylim(-3, 11)
plt.grid(); v1.draw(plt); v2.draw(plt)
;v3.draw(plt); v4.draw(plt);
v4.setBase(v2)
line, = ax.plot([], [])
line.set_data([], []) ;xdata, ydata =
[], []
from IPython.display import HTML
anim = FuncAnimation(figure,
  func = ani, frames =
np.arange(math.degrees(v1.ang),
math.degrees(v1.ang)+361, 10),
  interval = 100, repeat = False, blit =
True )
HTML(anim.to_html5_video())
```

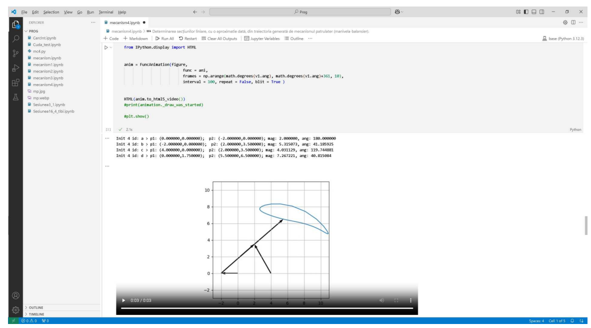

Fig. 4. - Movie of the simulation of the four-bar linkage mechanism in Python.

As visible in Figure 4 the trajectory generated by the mechanism can be a subject to classification. AI can be used to classify sections from the generated trajectories and to provide intelligently selected geometrical dimensions to achieve certain subsection as lines, circles or other curves. This approach can be used in the early development stages of different mechanisms, like flexible surgical instruments [17], to special, reconfigurable architectures [18], [19].

6. REFERENCES

[1] Plitea, N.; Hesselbach, J.; Pisla, D.; Raatz, A.; Vaida, C.; Wrege, J.; Burisch, A. *Innovative Development of Parallel Robots and Microrobots*. Acta Teh. Napoc. Ser. Appl. Math. Mec. 2006, 49, 5–26.

[2] Vaida, C.; Pisla, D.; Schadlbauer, J.; Husty, M.; Plitea, N. *Kinematic analysis of an innovative medical parallel robot using study parameters*. In New Trends in Medical and Service Robots. Mechanisms and Machine Science; Wenger, P., Chevallereau, C., Pisla, D., Bleuler, H., Rodić, A., Eds.; Springer: Cham, Switzerland, 2016; Volume 39.

[3] Tucan, P.; Vaida, C.; Horvath, D.; Caprariu, A.; Burz, A.; Gherman, B.; Iakab, S.; Pisla, D. *Design*

and Experimental Setup of a Robotic Medical Instrument for Brachytherapy in Non-Resectable Liver Tumors. *Cancers* **2022**, *14*, 5841. https://doi.org/10.3390/cancers14235841

[4] Rus, G.; Andras, I.; Vaida, C.; Crisan, N.; Gherman, B.; Radu, C.; Tucan, P.; Iakab, S.; Hajjar, N.A.; Pisla, D. *Artificial Intelligence-Based Hazard Detection in Robotic-Assisted Single-Incision Oncologic Surgery*. *Cancers* **2023**, *15*, 3387. https://doi.org/10.3390/cancers15133387

[5] https://www.python.org/

[6] https://docs.anaconda.com/

[7] https://docs.sympy.org/latest/modules/physics/mechanics/examples/four_bar_linkage_example.html

[8] Sun M. *Design and Kinematic Analysis of Planar Four-Bar Linkages with an Object Oriented Python Program.* AMR 2014;1046:174–6. https://doi.org/10.4028/www.scientific.net/amr.1046.174.

[9] https://github.com/kevin-hannegan/4-Bar-Linkage-Simulator

[10] Wang B, Du X, Ding J, Dong Y, Wang C, Liu X. *The Synthesis of Planar Four-Bar Linkage for Mixed Motion and Function Generation.* Sensors. 2021; 21(10):3504. https://doi.org/10.3390/s21103504.

[11] G. Asaeikheybari, A. S. Lafmejani, A. Kalhor and M. T. Masouleh, *Dimensional synthesis of a four-bar linkage mechanism via a PSO-based Cooperative Neural Network approach*, 2017 Iranian Conference on Electrical Engineering (ICEE), Tehran, Iran, 2017, pp. 906-911, doi: 10.1109/IranianCEE.2017.7985168.

[12] ANTAL, T .A., *Programming AutoCAD using JAWIN from Java in JDeveloper, Acta Technica Napocensis*, Series: Applied Mathemathics and Mechanics, ISSN 1221-5872, 53(3), p.481-486, Cluj-Napoca, 2010.

[13] ANTAL, T. A., *Java - Iniţiere - îndrumător de laborator*, Editura UTPRE, *ediţia a II-a*, Editura RISOPRINT, 2006, p.264, ISBN 973-751-349-5.

[14] ANTAL, T. A., *Elemente de Java cu JDeveloper - îndrumător de laborator*, Editura UTPRES, 2013, p.150, ISBN: 978-973-662-827-6.

[15] ANTAL, T. A., *Mechanism displacement analysis with AutoLisp in AutoCAD. Acta Technica Napocensis*, Series: Applied Mathematics and Mechanics, ISSN 1221-5872, 45, p. 19-24, Cluj-Napoca, 2002.

[16] ANTAL, Tiberiu Alexandru. *About the transformation of a structured code to a library in Java.* ACTA TECHNICA NAPOCENSIS - Series: Applied Mathematics, Mechanics, and Engineering, v. 67, n. 1, mar. 2024. ISSN 2393–2988.

[17] Vaida, C.; Plitea, N.; Pisla, D.; Gherman, B. *Orientation module for surgical instruments—A systematical approach*. Meccanica 2013, 48, 145–158.

[18] Nurahmi, L.; Caro, S.; Wenger, P.; Schadlbauer, J.; Husty, M. *Reconfiguration analysis of a 4-RUU parallel manipulator*. Mech. Mach. Theory 2015, 96, 269–289.

[19] Pisla, D.; Plitea, N.; Videan, A.; Prodan, B.; Gherman, B.; Lese, D. *Kinematics and design of two variants of a reconfigurable parallel robot*. In Proceedings of the ASME/IFToMM International Conference on Reconfigurable Mechanisms and Robots, London, UK, 24 July 2009.

Utilizarea limbajului de programare Python în simularea mecanismului patrulater

Rezumat: Limbajul de programare Python împreună cu paradigma de programare orientată pe obiecte sunt folosite pentru a scrie o aplicație de simulare utilizând biblioteca Matplot din Python. Folosirea implementării Python a conceptului de vector oferă o reprezentare grafică vectorială a mecanismului și produce traiectoria într-un format care poate fi utilizat ulterior în problema clasificării traiectoriilor generate de mecanismul patrulater.

Cuvinte cheie: *legătură cu patru bare, tolbă, Java, Python, simulare, traiectorie.*

ANTAL Tiberiu Alexandru, Professor, dr. eng., Technical University of Cluj-Napoca, Department of Mechanical System Engineering, antaljr@mail.utcluj.ro, 0264-401667, B-dul Muncii, Nr. 103-105, Cluj-Napoca, ROMANIA.