AutoCAD automation through Python scripting

Antonio Fernández[0000−0002−4320−2409]1, Lucía Díaz Vilariño[0000−0002−2382−9431]1, and Francesco Bianconi[0000−0003−3371−1928]2

[1] CINTECX, Universidade de Vigo, GeoTech Group,
Campus Universitario, 36310 Vigo, Spain
{antfdez, lucia}@uvigo.es
[2] Department of Engineering, Università degli Studi di Perugia,
Via Goffredo Duranti, 93, 06125 Perugia PG, Italy
bianco@ieee.org

Abstract. This work explores the automation of AutoCAD workflow using Python scripting, with a focus on the different Python libraries available for such tasks. The article analizes four prominent Python packages—`win32com`, `pyautocad`, `PyAutoGUI`, and `ezdxf`—assessing their strengths and weaknesses in the context of AutoCAD automation. To evaluate the performance of these libraries, a Python application was developed for the simple task of prefixing layer names in an AutoCAD drawing, testing each library's effectiveness in accomplishing this task. The findings indicate that each Python library offers distinct advantages depending on the automation requirements, with `ezdxf` standing out in aspects such as versatility, robustness, and documentation quality.

Keywords: AutoCAD · Python · automation · computer aided design · applied programming

1 Introduction

AutoCAD is one of the most widely used platforms in engineering and design fields, offering powerful tools for creating 2D and 3D models. However, for professionals and organizations working with large-scale projects or repetitive tasks, executing commands manually can be time-consuming and error-prone. Automation through scripting presents an effective solution to optimize workflows and improve productivity in AutoCAD environments.

The primary objective of this work is to explore the automation of AutoCAD workflows using Python scripting, focusing on the potential benefits and limitations of various Python libraries. Additionally, our study seeks to offer a clear understanding of the practical applications and trade-offs involved in each tool's usage, guiding users in selecting the most suitable approach for their automation needs.

The rest of the paper is organized as follows. Existing approaches to AutoCAD automation and related literature are reviewed in Sect. 2. In Sect. 3 we provide a comprehensive analysis of four prominent Python packages—`pyautocad`,

`ezdxf`, `win32com`, and `PyAutoGUI`—that can be utilized for automating repetitive tasks, improving efficiency, and extending the capabilities of AutoCAD. The application we developed to test the different Python packages in a real case study and the results we obtained with it are presented in Sect. 4. Finally, Sect. 5 provides the conclusions that can be drawn from our study.

2 Background and related work

Several technologies to automate AutoCAD tasks exist. For years, AutoLISP (a dialect of the programming language Lisp built specifically for use with the full version of AutoCAD and its derivatives) has set the standard for customizing and extending AutoCAD on Windows and Mac OS. However, AutoLISP has a number of shortcomings, such as slow execution of complex or resource-intensive tasks, or limited access to the underlying AutoCAD architecture. Besides, AutoLISP uses a syntax based on parentheses and a functional programming style, which can be difficult for developers who are more familiar with modern, object-oriented languages like C# or Java. For all these reasons, the use of AutoLISP has declined significantly.

Microsoft's Component Object Model (COM) is a binary-interface technology for software components that enables using objects in a language-neutral way between different programming languages, programming contexts, processes and machines. It allows applications on Windows-based platforms to communicate and exchange data. Object Linking and Embedding (OLE) is another proprietary technology developed by Microsoft that allows an editing application to export part of a document to another editing application and then import it with additional content. ActiveX encapsulated both OLE and COM principles into a broader framework for automation and control in Windows environments. ActiveX provides a way to automate AutoCAD using Visual Basic for Applications (VBA). It allows deeper integration than AutoLISP, offering access to more of AutoCAD's features and the ability to create custom forms, i.e. graphical user interfaces (GUIs) such as dialogs, buttons and controls for automation scripts. It is easier for those familiar with Visual Basic but is limited in performance and flexibility compared to compiled solutions.

ObjectARX, the AutoCAD Runtime Extension programming environment, is a powerful application programming interface (API) based on C++ that offers low-level access to AutoCAD's core functions. It's ideal for performance-intensive tasks and highly customized solutions. While it provides the greatest control and efficiency, it requires a strong knowledge of C++ and is more complex to set up than AutoLISP or VBA.

Along with the approaches mentioned above, Python has increasingly garnered attention for automating AutoCAD tasks programmatically in various domains of graphical engineering and communication due to several advantages. Its modern and user-friendly syntax makes it easier to learn, especially for those with prior programming experience. Python also offers extensive libraries for integration with other tools and technologies, as well as greater versatility in han-

dling complex tasks, data analysis, and machine learning. Additionally, Python's cross-platform compatibility and active community further enhance its appeal for AutoCAD automation projects.

Our literature survey revealed that the published works on Python-based AutoCAD automation can be classified according to several criteria. The first is whether we need AutoCAD to be actually installed on our PC or not. Most automating tools work through some kind of application programming interface (such as the cited COM) that link to AutoCAD in some way, hence a running version of the AutoCAD software is required in this case. Other methods operate on AutoCAD-compatible files (such as `DXF`) and do not need the software to be installed. Secondly, if we look at task types, we can distinguish between processing existing drawing files and creating drawing files from scratch through a fully-automated design process. A third classification criterion relates to the field of application. Python-based AutoCAD automation has been successfully implemented in sectors such as mechanical parts design, production facility layout, and university teaching.

Li et al. [1] explored the automation of AutoCAD operations using the `win32com` library, highlighting its effectiveness in streamlining tasks. They demonstrate its application by automating the processing of a batch of drawing files, where the `EXTMIN` and `EXTMAX` variables are read to determine the dimensions of each drawing. The drawings are then saved to a different directory, with their file names modified to include the calculated dimensions. The integration of machine learning into CAD automation has been investigated by Yet et al. [2], who presented a method for generating 3D models of mechanical parts from orthographic views. They employed neural networks to minimize human intervention during the 3D model construction process. The proposed approach is demonstrated by generating simple 3D models from 2D `DXF` drawings, showcasing the potential of machine learning to automate the conversion of 2D orthographic drawings into accurate 3D representations.

Rathod et al. [3,4] developed a Python-based application to automate the creation of compound punching and blanking die drawings. The application features a GUI that allows users to input design parameters easily. By eliminating the need for designers to manually consult manuals, apply formulas, and perform calculations, the tool significantly reduces design time. Similarly, Patil et al. [5] proposed the creation of 2D manufacturing drawings using the `pyautocad` package to automate the design process of rigid flange couplings. They developed a dedicated GUI that facilitates parametric design and modification. Additionally, a knowledge-based engineering (KBE) approach was integrated, enabling the use of a design database and an expert system to optimize the selection of design parameters.

In the realm of industrial facilities, Plotnikov et al. [6] developed a configurator tool to automate the design of a power substation with AutoCAD. They proved the effectiveness of their approach by generating all project documentation (wiring diagrams, lists of elements, equipment layout diagrams, etc.) using a modular structure based on standardized electrical schematics. The authors

utilized the ObjectARX programming environment, as it provided the necessary computational efficiency to reduce execution time to a reasonable level, making it suitable for the complex design and simulation tasks involved in their application.

Some authors investigated the use of AutoCAD automation in engineering teaching. Wang and Zhang [7] proposed a method to prevent cheating in engineering colleges when students use AutoCAD. They developed a feature information recording plug-in using ObjectARX to track and detect cheating behavior. The plug-in records various system details, including motherboard and hard disk serial numbers, whenever commands like `SAVEAS` or `QSAVE` are executed. Moreover, Díaz-Vilariño et al. [8] introduced an innovative self-assessment method for evaluating layouts in industrial plant design, specifically in the engineering education environment. The approach compares a graph of spaces and relationships extracted automatically from AutoCAD drawings created by students with an objective graph representing the optimal solution. This method allows students to evaluate how closely their designs adhere to project specifications, enabling early error detection.

3 Materials and methods

To achieve the objectives of our study, we conducted a detailed review of four Python packages and implemented a simple yet illustrative case study using each. The case study consists of processing an AutoCAD drawing by adding a prefix to the names of its layers. It is common for architects and engineers to perform this task manually when starting their projects with a file downloaded from the cadastral system. Adding a prefix to layer names, for example `CADASTRE_`, is very useful to easily identify the original drawing layers and distinguish them from the layers added in subsequent stages of the project's design. In the following sections, we introduce the analysis of the packages along with simplified code snippets that illustrate the implementation of the case study using each package.

3.1 `win32com`

The `win32com` package [9] enables interaction with applications through the COM interface. Therefore, it is only available on Windows platforms. The core of this package is the `win32com.client.Dispatch` class, which, when instantiated, associates a COM object with a specific application. Once this object is created, methods can be invoked, and properties can be accessed within the application. It is important to point out that `win32com` relies on the third-party `pywin32` library. For a detailed understanding of how Python applications can be integrated on Windows, the book by Hammond and Robinson [10] is a valuable resource. A primary challenge in automating AutoCAD with `win32com` is runtime errors, often caused by AutoCAD being busy with other tasks and unable to respond. To handle these errors, it is common to use error-checking strategies, such as checking the state of AutoCAD before making a call, using `try-except` blocks

to catch the exceptions, and retrying commands after a delay. However, these remedies complicate the code and slow down script execution.

```
from win32com.client.dynamic import Dispatch

prefix = 'CADASTRE_'
acad = Dispatch('AutoCAD.Application')
acad.Visible = True
doc = acad.Documents.Open('input_file.dwg')
for layer in doc.Layers:
    name = layer.Name
    if name not in ('0', 'Defpoints'):
        layer.Name = prefix + name
doc.SaveAs('output_file.dwg')
```

3.2 pyautocad

`pyautocad` [11] is a library aimed at simplifying the writing of ActiveX Automation scripts for AutoCAD with Python. `pyautocad` offers a more straightforward approach than `win32com`, which is more difficult to learn. However, it does not provide full access to AutoCAD's COM API, which may limit its capabilities for complex tasks. Regarding the maintenance of `pyautocad`, the project repository has not been updated in several years, which could pose compatibility issues with future AutoCAD versions if the COM API changes.

```
from pyautocad import Autocad

prefix = 'CADASTRE_'
acad = Autocad(create_if_not_exists=True)
for layer in acad.doc.Layers:
    name = layer.Name
    if name not in ('0', 'Defpoints'):
        layer.Name = prefix + name
```

3.3 PyAutoGUI

`PyAutoGUI` [12] is a multi-platform library that enables a Python script to control the mouse and keyboard, making it a versatile tool for automating tasks in any software, including AutoCAD. However, its reliance on pixel- and dialog-based interactions can lead to issues with resolution changes or UI modifications, making it less reliable than API-based solutions. Moreover, the script only simulates input actions, such as moving the mouse, clicking, typing text, or pressing keys, but it does not have the ability to receive information or react to events within the application. As a consequence, it is recommended to introduce delays between commands, on the one hand to prevent too rapid inputs that might overwhelm AutoCAD or cause it to miss some actions, and on the other to allow AutoCAD to handle each task before the next one is triggered. This, of course, makes the execution slower.

```
import pyautogui as pgui
import os, time

def typewrite(keyboard_input, delay=1):
    pgui.typewrite(keyboard_input)
    pgui.keyDown('enter')
    pgui.keyUp('enter')
    time.sleep(delay)
prefix = 'CADASTRE_'
names = ['BUSQUEDA', 'CONSTRUCCION', 'PARCELA', 'REFCATASTRAL',
         'SUBPARCELA', 'TXTCONSTRU', 'TXTSUBPA']
os.startfile('start input_file.dwg'); time.sleep(3)
for name in names:
    typewrite('-LAYER')
    typewrite('Rename')
    typewrite(name)
    typewrite(prefix + name)
    pgui.hotkey('escape')
typewrite('SAVEAS')
typewrite('output_file.dwg')
pgui.hotkey('alt', 's')
```

3.4 ezdxf

`ezdxf` [13] is a Python package to create new `DXF` documents and read/modify/write existing `DXF` documents. This package offers several advantages, including ease of use and the ability to work across different platforms without the need to have AutoCAD installed. However, its functionality is limited to handling `DXF` files and does not support direct AutoCAD interaction, unlike packages such as `pyautocad` or `win32com`. In terms of performance, `ezdxf` is generally faster for handling `DXF` files due to its lightweight nature. Regarding documentation, `ezdxf` stands out with its comprehensive and well-maintained resources, making it easier for users to implement.

```
import ezdxf

prefix = 'CADASTRE_'
doc = ezdxf.readfile('input_file.dxf')
clayer_name = doc.header['$CLAYER']
doc.header['$CLAYER'] = '0'
names = [layer.dxf.name for layer in doc.layers]
for name in names:
    if name not in ('0', 'Defpoints'):
        layer = doc.layers.get(name)
        layer.rename(prefix + name)
doc.header['$CLAYER'] = prefix + clayer_name
doc.saveas('output_file.dxf')
```

4 Results and discussion

To facilitate the testing process, we have developed an application in Python. The entire code is available in a public repository on GitHub [14]. This application features a graphical user interface that makes it very easy to enter the prefix to be added to the layer names, choose the Python package to be used, select the drawing file to be processed, and launch the activity. By pressing a button on the GUI (see Fig. 1) the user can access the contents of an `HTML` help file that is displayed in the default browser. In order to make things even easier, we have used `PyInstaller` to bundle the script and all its dependencies into a standalone `EXE` file, which can be run from the console or by double clicking on the file icon. This makes it unnecessary for the user to install Python on their computer.

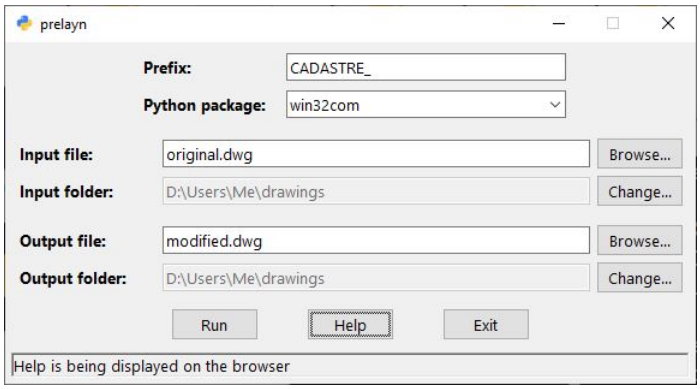

Fig. 1. Graphical user interface of the application developed for this study.

A comparative analysis of execution speed, ease of implementation, quality of documentation, robustness against runtime errors, and versatility is provided. We have ranked the four packages considered according to the above five criteria and represented all the rankings simultaneously using a radar chart for visual interpretation of the results (see Fig. 2). The execution time differs considerably between packages: `ezdxf` and `pyautocad` perform the task within hundredths of a second, `win32com` within seconds, and `PyAutoGUI` within tens of seconds.

The findings indicate that each Python library offers distinct advantages depending on the automation requirements. `PyAutoGUI`, while the most versatile approach, is generally slower and less reliable for complex automation tasks compared to the other tools. `pyautocad` provides a straightforward solution for automating AutoCAD commands and tasks within the application, but may be limited in handling complex drawing manipulations. It should be noted that `pyautocad` is the only tool that does not implement the functionality to open/save a drawing. `win32com` offers a more flexible solution for advanced

automation, enabling full control of the AutoCAD environment, but involves a steeper learning curve. The main drawback of the `win32com` package is the large number of runtime errors that occur when using it. Avoiding them is possible, but at the cost of significantly complicating the code. In addition, `win32com` is very poorly documented, which considerably hampers its widespread adoption. The package `ezdxf` proves to be highly effective for tasks that involve the creation and editing of `DXF` files. This package offers excellent performance and has the advantage of not requiring interaction with the AutoCAD application. As a result, it can be used on any platform, and users do not need an Autodesk license to utilize its functionality.

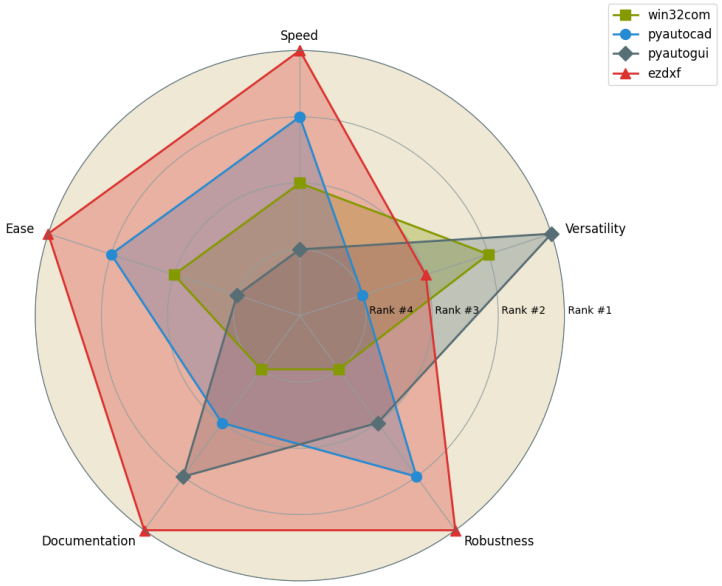

Fig. 2. Comparison of the main characteristics of the considered Python packages.

5 Conclusions

Automation of AutoCAD through Python scripting presents significant opportunities to boost workflow efficiency, reduce manual errors, and improve productivity. The choice of a Python package depends largely on the specific task and user preferences. `PyAutoGUI` serves as a fast and flexible solution for automating interactions with AutoCAD through mouse and keyboard control. `pyautocad` is ideal for users looking for simplicity and direct interaction with AutoCAD, while `win32com` is best suited for more complex and customizable tasks. `ezdxf` is perfect for file manipulations that do not require interaction with AutoCAD. Due to its good performance in each of the aspects analyzed, we consider the latter

as the most convenient Python tool for automating AutoCAD tasks. Ultimately, the work provides valuable insights that will help engineers and developers in selecting the right Python package for their AutoCAD automation projects, based on the trade-offs between factors such as ease of implementation and versatility, among others.

Acknowledgments. Antonio Fernández would like to thank Diego Campos Juanatey for suggesting various tasks that could be automated in the professional use of AutoCAD. This work was partially supported by human resources grant RYC2020-029193-I funded by MCIN/AEI/10.13039/501100011033 and NextGenerationEU/PRTR, and 'El FSE invierte en tu futuro' programme funded by ESF.

References

1. Li, J., Wang, P., Li, Q., He, Y., Sun, L., Sun, Y., Zhao, P., Jia, L.: Research on application automation operations based on Win32com. In: Frontiers in Artificial Intelligence and Applications. Advances in Artificial Intelligence, Big Data and Algorithms, vol. 373, pp. 113–118. IOS Press (2023)
2. Yet, J.Y., Lau, C.Y., Thang, K.F.: 2D Orthographic Drawings to 3D Conversion using Machine Learning. In: 2022 3rd International Conference on Smart Electronics and Communication (ICOSEC). pp. 1308–1312 (2022)
3. Rathod, V., Jha, P.K., Sawai, N.M.: Optical CAD modelling and designing of compound die using the Python scripting language. International Journal on Interactive Design and Manufacturing 17(2), 981—991 (2023)
4. Rathod, V., Karanjkar, A., Sawai, N.: A Python-based GUI approach for efficient component design of compound die for composite material: 3D and 2D CAD modeling. Journal of Mines, Metals and Fuels 72(10), 1075–1091 (2024)
5. Patil, K., Deshpande, S., Joshi, S., Raul, A.: Developing an integrated design framework using Python scripting for parametric CAD modelling of flange coupling. International Journal on Interactive Design and Manufacturing 18(7), 4451—4462 (2024)
6. Plotnikov, D., Aljeazna, W., Lachin, V., Solomentsev, K.Y.: Using the ObjectARX programming environment for modular information-measuring systems design and simulation. Journal of Physics: Conference Series 1582(1) (2020)
7. Wang, L., Zhang, W.: Development of computer drawing cheating identification system using the nine-screen method. ACM International Conference Proceeding Series pp. 118—125 (2023)
8. Díaz Vilariño, L., González-Crespón, J.L., Alonso-Rodríguez, J.A., Fernández-Álvarez, A.: Una propuesta para la autoevaluación de layouts en la asignatura Oficina Técnica y Proyectos. In: Actas de las VII Jornadas Iberoamericanas de Innovación Educativa en el Ámbito de las TIC y las TAC (InnoEducaTIC 2020). pp. 267–274 (2020), http://hdl.handle.net/10553/76539, in Spanish
9. Golden, T.: PyWin32 documentation. Available online https://timgolden.me.uk/pywin32-docs/index.html (2019)
10. Hammond, M., Robinson, A.: Python Programming on Win32. O'Reilly Media, Sebastopol, CA (2000)
11. Haritonov, R.: pyautocad's documentation. Available online https://pyautocad.readthedocs.io/en/latest/ (2014)

12. Sweigart, A.: PyAutoGUI's documentation. Available online https://pyautogui.readthedocs.io/en/latest/ (2019)
13. Moitzi, M.: Documentation for ezdxf. Available online https://ezdxf.readthedocs.io/en/stable/ (2014)
14. Fernández, A.: Github repository for PREfixing LAYer Names of AutoCAD drawings. Available online https://github.com/tonechas/prelayn (2025)