




1 pyRTC: An open-source Python solution for kHz 2 real-time control of adaptive optics systems

3 **Jacob Taylor** ^{1,2}, **Robin Swanson**^{2,3}, and **Suresh Sivanandam**^{1,2}

4 **1** David A. Dunlap Department of Astronomy and Astrophysics, University of Toronto, 50 St George St,
5 Toronto, ON M5S 3H4, Canada **2** Dunlap Institute for Astronomy and Astrophysics, University of
6 Toronto, 50 St George St, Toronto, ON M5S 3H4, Canada **3** Department of Computer Science,
7 University of Toronto, 40 St George St, Toronto, ON M5S 2E4, Canada

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Arfon Smith](#) 

Reviewers:

- [@joseph-long](#)
- [@joao-aveiro](#)
- [@sefffal](#)

Submitted: 01 February 2024

Published: unpublished

License

Authors of papers retain copyright
and release the work under a
Creative Commons Attribution 4.0
International License ([CC BY 4.0](#)).

8 Summary

9 Adaptive optics (AO) is a technology that rapidly detects and corrects optical aberrations to
10 significantly improve the resolution of an optical system. AO has been applied to imaging
11 problems in fields such as astronomy, ophthalmology, and microscopy, as well as various military
12 and industrial applications. AO systems operate using a so-called 'real-time controller' (RTC),
13 a term used by the community to describe software responsible for converting optical aberration
14 measurements into corresponding corrections. In astronomical contexts, RTCs typically control
15 100-1000 degrees of freedom at speeds between 500-2000 Hz.

16 pyRTC is an open-source, community-driven Python package for real-time control of AO
17 systems, built with the following core goals:

- **Customizable High-Performance AO Pipeline:** Provide an efficient RTC pipeline with potential for full user customization.
- **Abstraction of Core AO System Components:** Facilitate support for a broad range of AO system architectures.
- **Open Library of API Examples:** Provide a library of examples for common hardware APIs used by the community to save time implementing basic hardware interactions.
- **Real-Time Monitoring and Interface Flexibility:** Support real-time access to intermediate data products, text-based user interaction, and straightforward integration with user-built GUIs.
- **Cross-Platform Compatibility:** Ensure broad usability across different operating systems.

28 In this publication, we present a pre-alpha version of the pyRTC package to the community
29 and invite them to try it out on their hardware, provide feedback, and contribute to the code
30 base or hardware API library.

31 Statement of Need

32 Hardware providers for AO system components (cameras, deformable mirrors, etc...) currently
33 provide API support for only three programming languages: C/C++, Python, and MATLAB.
34 High-performance RTCs have been developed in C/C++ (e.g., CACAO [[@CACAO](#)], DAO,
35 DARC[[@DARC](#)], HEART[[@HEART](#)]), while off-the-shelf MATLAB controllers are available for
36 purchase. Off-the-shelf RTC solutions can be costly, and they lack customizability and
37 transparency. The community-led C++ solutions, known for their performance, can be
38 complex to understand and implement, leaving AO researchers with limited options: expensive
39 RTCs, investing in software expertise for C++ solutions, or creating custom low-performance
40 RTCs.

41 pyRTC is an open-source RTC software for AO that aims to be the highest performance
42 free AO control software available in Python, maintaining sufficient user-friendliness for the

43 average AO researcher. pyRTC abstracts the variable hardware components in an AO system
44 into high-level control objects and provides an architecture for combining those objects into a
45 high-performance pipeline. Traditional performance limitations in Python, due to the Global
46 Interpreter Lock (GIL), are circumvented by running each pipeline operation as an independent
47 subprocess, communicating via shared memory and TCP sockets. This architecture allows for
48 soft real-time monitoring of intermediate data products, efficient CPU usage, compatibility
49 with custom GUIs, and the use of the Python interpreter as a simple RTC interface.

50 While extreme AO applications may still require custom C++ solutions, pyRTC is envisioned
51 for a wider range of applications, including:

- 52 ▪ Moderate performance adaptive optics applications (approximately 1 kHz speed for about
53 100 modes).
- 54 ▪ Lab environments dependent on student labor.
- 55 ▪ Test systems for hardware/software at on-sky speeds.
- 56 ▪ Any neural network/AI-based controller built in Python.

57 Features and Implementation

58 pyRTC is structured in order to minimize the amount of additional coding required to integrate
59 pyRTC into a new hardware environment. The way pyRTC accomplishes this is by defining
60 abstract superclasses for AO components, namely:

- 61 ▪ `Loop.py`: Responsible for the AO integrator logic, relies on data products from the
62 slopes process class.
- 63
- 64 ▪ `ScienceCamera.py`, Responsible for the PSF logic, connects to the PSF camera and
65 produces data products for further processing
- 66 ▪ `SlopesProcess.py`, Responsible for the slopes computations, relies on data products from
67 the `WavefrontSensor` class.
- 68 ▪ `WavefrontCorrector.py`, Responsible for the controlling the Deformable Mirror, receives
69 commands from `Loop` class or elsewhere.
- 70 ▪ `WavefrontSensor.py`, Responsible for the WFS logic, connects to the WFS camera and
71 produces data products for the `SlopesProcess` class.

72 These superclasses are then overridden by the user defined hardware class which interfaces with
73 the hardware's API. We have provided examples in the `pyRTC/hardware` folder. Ideally, users
74 will contribute their hardware examples and the repository will serve as a library of examples
75 for new users to follow. For some of the components (e.g., `SlopesProcess`), users can choose
76 to override the classes if they require specific computations or they can use the classes default
77 functionality. We intend to expand the scope of the default functionality as new use cases
78 emerge.

79 Once the hardware classes have been established, a communication interface implemented in
80 `Pipeline.py` allows the user to initialize their AO loop as either a set of independent processes
81 which communicate via TCP (for performance, to get around the GIL), or within a single
82 program (for simplicity). In either case, pyRTC has been written to be entirely initialized
83 using a config YAML file. This includes the functions which will be included in the main RTC
84 pipeline. Therefore, once the core hardware compatibility has been written, all of the real-time
85 manipulation of the system is to be done via iPython interface, or via config file changes.

86 Shared Memory and Live Viewing

87 pyRTC is built using shared memory objects provided by the `multiprocessing` python package.
88 Therefore, all data products shared between pyRTC components are available for soft real-time
89 viewing and analysis. pyRTC comes with a real-time viewing script called `pyRTCview.py` which
90 utilizes the `pyQT5` package to produce a live feed of a specific shared memory object. For
91 example, to view the images produced by the `WavefrontSensor` class run:

92 `python pyRTCView.py wfs`

93 We hope to expand this viewer into an example GUI in the future.

94

95 **Acknowledgements**

96 **References**

DRAFT