

# ImSwitch: Generalizing microscope control in Python

Xavier Casas Moreno<sup>1</sup>, Staffan Al-Kadhimi<sup>1</sup>, Jonatan Alvelid<sup>1</sup>,  
Andreas Bodén<sup>1</sup>, and Ilaria Testa<sup>1</sup>

<sup>1</sup> SciLifeLab, KTH Royal Institute of Technology

DOI: [10.21105/joss.03394](https://doi.org/10.21105/joss.03394)

## Software

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

Editor: [Kevin M. Moerman](#) ↗

## Reviewers:

- [@MakerTobey](#)
- [@uellue](#)
- [@beniroquai](#)
- [@untzag](#)

Submitted: 30 April 2021

Published: 22 June 2021

## License

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

## Summary

The constant development of novel microscopy methods with an increased number of dedicated hardware devices poses significant challenges to software development. On the one hand, software should control complex instruments, provide flexibility to adapt between different microscope modalities, and be open and resilient to modification and extension by users and developers. On the other hand, the community needs software that can satisfy the requirements of the users, such as a user-friendly interface and robustness of the code. In this context, we present ImSwitch, based on the model-view-presenter (MVP) design pattern (Potel, 1996), with an architecture that uses polymorphism to provide a generalized solution to microscope control. Consequently, ImSwitch makes it possible to adapt between different modalities and aims at satisfying the needs of both users and developers. We have also included a scripting module for microscope automation applications and a structure to efficiently share data between different modules, such as hardware control and image processing.

## Statement of need

As described in Chhetri et al. (2020), there is a need to have a generalized solution for microscope control, since the usual approach in microscopy labs is to create their own custom-built software. A complete list is available in Stuurman et al. (2021). ImSwitch is designed to be compatible with many different microscope modalities and customizable to the specific design of individual custom-built microscopes, all while using the same code. We would like to involve the community in further developing ImSwitch in this direction, believing that it is possible to integrate current state-of-the-art solutions into one unified software.

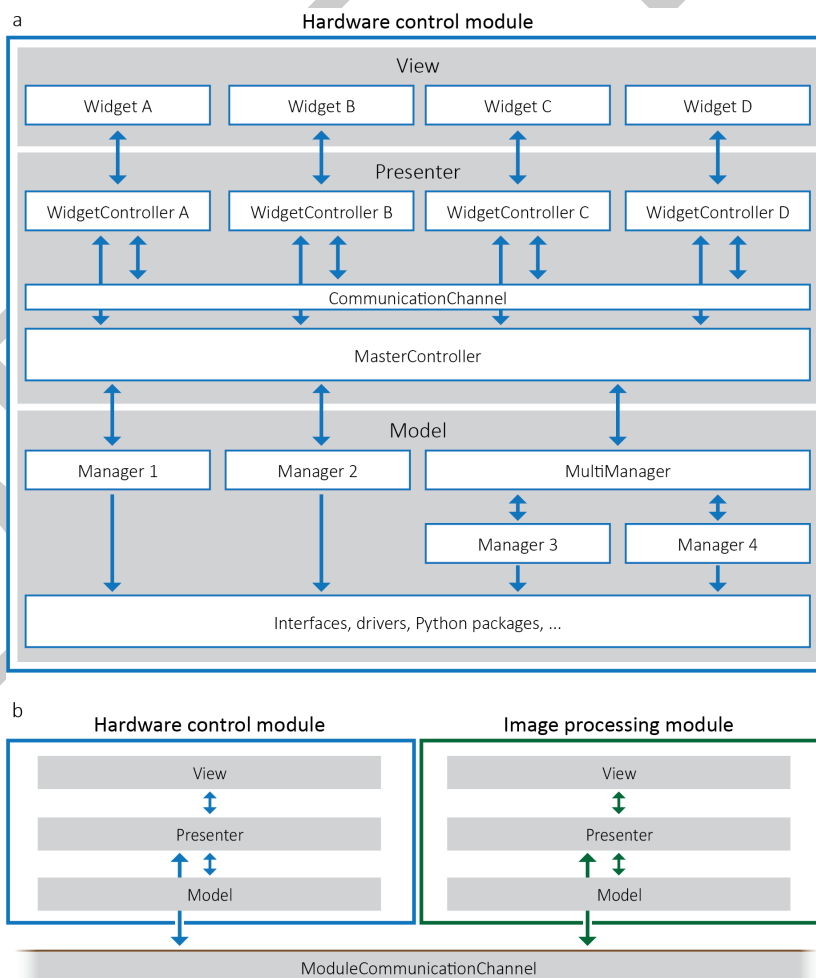
## Architecture and implementation

ImSwitch consists of software modules, e.g., hardware control and image processing, that exchange data through the ModuleCommunicationChannel (Figure 1). Modules can also expose an internal Application Programming Interface (API) to the scripting platform, allowing user-defined scripts to interact with them without requiring the user to modify the code itself to meet their demands. Each of the modules follows the MVP architecture, which has three primary layers: model, view, and presenter. The hardware control module has the following structure:

- The **model** layer represents the backend of the program. It has direct access to the hardware through a device sub-layer, which contains the interfaces of the devices, drivers, and Python packages. To provide polymorphism, we have added another sub-layer in

the model that consists of what we name managers and multi-managers. The managers are objects that will specify how to control the devices, and which packages should be used. For example, a microscope setup with two cameras and three point detectors will make use of a multi-manager called `DetectorsManager`, which in turn will manage two instances of `CameraManager` and three instances of `PointDetectorManager`. Both managers implement the same functionality through identical function calls but with different implementations.

- The **view** layer is the Graphical User Interface (GUI). The GUI for each component of the hardware control module is represented by a view element called a `Widget`. The view layer is linked to the presenter layer by linking each `Widget` with a `WidgetController`. The presenters (`WidgetControllers`) manipulate the views (`Widgets`) but not vice versa.
- We implemented the **presenter** layer using two sub-layers: the first sub-layer contains `WidgetControllers` that mainly read the input of the user through the view and communicate it to the `MasterController`; the second sub-layer has the `MasterController` that directly interacts with the managers. The `CommunicationChannel` serves as a platform to share demands and information between `WidgetControllers`.



**Figure 1:** Architecture of ImSwitch

We use PyQt ([PyQT, 2012](#)) for the GUI layout, thread management, and event management; Napari ([napari contributors, 2019](#)) for image visualization, which can easily integrate multiple

56 image layers and supports plugins for image processing; and HDF5 ([The HDF Group, 2000-](#)  
57 [2010](#)) for storing image data and metadata, including all parameters of each experiment and  
58 acquisition. The user can readily load metadata from stored files into ImSwitch, allowing  
59 experiment acquisition from templates.

60 Central to the concept of allowing modularity is the use of JSON configuration files to de-  
61 fine different microscopes. They contain information about the hardware devices and their  
62 connections, and the GUI tools to be loaded.

63 As a proof of concept ImSwitch has been implemented to control the various micro-  
64 scopes of our lab: point-scanning STED/confocal ([Alvelid & Testa, 2019](#)) and parallelized  
65 RESOLFT/confocal (MoNaLISA) ([Masullo et al., 2018](#)). These microscopes were previously  
66 controlled with a combination of purpose-built software that were microscope-specific  
67 or closed-source software. The reader can find more details about the implementations  
68 and documentation of ImSwitch at [imswitch.readthedocs.io](#). We have implemented the  
69 image processing module for reconstructing MoNaLISA images using our pre-existing shared  
70 library. Other microscope modalities requiring designated reconstruction or processing could  
71 implement separate modules, using the ModuleCommunicationChannel to retrieve the data,  
72 and readily load them into ImSwitch.

## 73 Comparison to other software

74  $\mu$ Manager ([Edelstein et al., 2010](#)) is a free and open-source software for microscope con-  
75 trol written in Java and C++, with pervasive driver support to different devices and widely  
76 used by microscopists. However,  $\mu$ Manager does not adapt fully to complex microscopes  
77 that require scanning and triggering or other intricate operations. Moreover, software mod-  
78 ification and extension are difficult due to the programming languages and the architecture.  
79 [Python bindings](#) were recently made available for [MMCore](#), which is the device control layer  
80 of  $\mu$ Manager, making it compatible with ImSwitch implementation (through the model device  
81 sub-layer). Pycro-Manager ([Pinkard et al., 2021](#)) provides access to  $\mu$ Manager from Python  
82 and combines it with image processing and multiple applications. However, custom plugins  
83 must still be written for the  $\mu$ Manager Java layer, making it complicated to use in complex  
84 devices without a pre-existing specialized implementation. This limits the user-friendliness for  
85 microscope users.

86 Microscope ([Pinto et al., 2021](#)) is a recent alternative for providing a resilient device layer  
87 in Python for controlling microscope devices. It is possible to script microscope experiments  
88 and use Cockpit ([Phillips et al., 2021](#)) as a GUI. However, the GUI is still limited to simpler  
89 microscopes. The most substantial focus of Microscope is to provide a standard interface to  
90 control microscope device types. Therefore, it could be integrated with ImSwitch to work  
91 towards software generalization by increasing compatibility and including even more devices.

## 92 Acknowledgements

93 We would like to acknowledge the authors of [Tempesta](#), a further development of Tormenta  
94 ([Barabas & Masullo, 2016](#)), for their contributions to earlier software and inspiration for a user-  
95 friendly interface. We are also thankful to Francesca Pennacchietti for testing and providing  
96 feedback.

97 We thank the Swedish Foundation for Strategic Research (SSF) funding FFL15-0031 for  
98 supporting the project.

## References

- Alvelid, J., & Testa, I. (2019). Stable stimulated emission depletion imaging of extended sample regions. *Journal of Physics D: Applied Physics*, 53. <https://doi.org/10.1088/1361-6463/ab4c13>
- Barabas, F., & Masullo, L. A. (2016). Note: Tormenta: An open source Python-powered control software for camera based optical microscopy. *Review of Scientific Instruments*, 87(126103). <https://doi.org/10.1063/1.4972392>
- Chhetri, R., Preibisch, S., & Stuurman, N. (2020). *Software for microscopy workshop white paper*. <http://arxiv.org/abs/2005.00082>
- Edelstein, A., Amodaj, N., Hoover, K., Vale, R., & Stuurman, N. (2010). Computer control of microscopes using µmanager. *Current Protocols in Molecular Biology*, 92(1), 14.20.1–14.20.17. <https://doi.org/10.1002/0471142727.mb1420s92>
- Masullo, L. A., Bodén, A., Pennacchietti, F., Coceano, G., Ratz, M., & Testa, I. (2018). Enhanced photon collection enables four dimensional fluorescence nanoscopy of living systems. *Nat Commun*, 9(3281). <https://doi.org/10.1038/s41467-018-05799-w>
- napari contributors. (2019). *napari: a multi-dimensional image viewer for python*. <https://doi.org/doi:10.5281/zenodo.3555620>
- Phillips, M. A., Pinto, D. M. S., Hall, N., Mateos-Langerak, J., Parton, R. M., Titlow, J., Stoychev, D. V., Parks, T., Pinto, T. S., Sedat, J. W., Booth, M. J., Davis, I., & Dobbie, I. M. (2021). Microscope-cockpit: Python-based bespoke microscopy for bio-medical science. *bioRxiv*. <https://doi.org/10.1101/2021.01.18.427178>
- Pinkard, H., Stuurman, N., Ivanov, I. E., Anthony, N. M., Ouyang, W., Li, B., Yang, B., Tsuchida, M. A., Chhun, B., Zhang, G., Mei, R., Anderson, M., Shepherd, D. P., Hunt-Isaak, I., Dunn, R. L., Jahr, W., Kato, S., Royer, L. A., Thiagarajah, J. R., ... Waller, L. (2021). Pycro-manager: Open-source software for customized and reproducible microscope control. *Nature Methods*, 18(3), 226–228. <https://doi.org/10.1038/s41592-021-01087-6>
- Pinto, D. M. S., Phillips, M. A., Hall, N., Mateos-Langerak, J., Stoychev, D., Pinto, T. S., Booth, M. J., Davis, I., & Dobbie, I. M. (2021). Python-microscope: High performance control of arbitrarily complex and scalable bespoke microscopes. *bioRxiv*. <https://doi.org/10.1101/2021.01.18.427171>
- Potel, M. (1996). *MVP: Model-view-presenter the taligent programming model for c++ and java*.
- PyQT. (2012). *PyQt reference guide*. <http://www.riverbankcomputing.com/static/Docs/PyQt4/html/index.html>
- Stuurman, N., Preibisch, S., Babcock, H., & Chhetri, R. (2021). *uScopeControl* (Version 0.1). Zenodo. <https://doi.org/10.5281/zenodo.4433237>
- The HDF Group. (2000-2010). *Hierarchical data format version 5*. <http://www.hdfgroup.org/HDF5>