Intravision, Robotic Imaging Platform

P. Vriend, K. Niu

June 2024

University of Twente, Biomedical Technology, thesis

Abstract

Image-guided robotic systems are crucial in modern medical surgery, relying heavily on advanced image processing software for decision-making. Traditional platforms like ROS (Robot Operating System) and 3D Slicer have provided robust tools for developing such systems but often come with significant complexity and resource demands, impeding their real-time application and flexibility. This thesis addresses these challenges by proposing a lightweight, modular image-processing software platform tailored for robotic interventions. The platform aims to deliver efficient real-time performance with minimal computational overhead, high accuracy, and seamless integration with various robotic systems. Key features include a user-friendly graphical user interface (GUI) with specialized modules for data plotting, volumetric rendering, and real-time data acquisition. The architecture emphasizes modularity, scalability, and the application of design patterns to ensure maintainability and extendability. The platform's effectiveness is validated through comprehensive testing, demonstrating its capability to handle high-resolution images, live data, and diverse file formats efficiently. This work lays the foundation for future enhancements in real-time data processing and integration with robotic systems, potentially improving the precision and efficacy of image-guided surgeries.

introduction

Image-guided robotic systems have become a pivotal technology in medical surgery. These systems rely on advanced image processing software to interpret visual data and make informed decisions. Historically, platforms like ROS (Robot Operating System) and 3d slicer have been widely used for developing such applications. These frameworks provide a set of tools and libraries that facilitate the development of sophisticated image processing and robotic control algorithms. Despite their capabilities, the complexity and resource demands of these platforms can be prohibitive for real-time applications, particularly in scenarios requiring lightweight and highly efficient processing.

Current software architectures for image-guided robotic systems often have several critical limitations. Firstly, their heavy reliance on extensive libraries and frameworks leads to high computational overhead, which can worsen real-time performance. Secondly, the monolithic nature of many existing platforms decreases flexibility and scalability, making it challenging to adapt to different robotic systems or to integrate new algorithms without substantial reconfiguration of the code. Additionally, the high latency and resource consumption associated with these architectures can limit their effectiveness in dynamic environments, such as small-scale robots or embedded systems.

For the development of an optimized image-processing platform for robotic intervention, several key user requirements must be addressed. Users need a software solution that is lightweight yet powerful, capable of processing high-resolution images in real-time with minimal latency. The platform should be modular and expandable, allowing easy integration with diverse robotic systems and existing algorithms. It should also support loading data both in real time and from files. Furthermore, the solution must be resource-efficient, ensuring that it can operate effectively on devices with limited computational power without compromising performance or accuracy.

The gap in current software lies in the lack of a unified platform that combines real-time or regular data processing with robotic devices, specifically with high efficiency and scalability. Existing solutions are often complex and require heavy computational resources, which is not optimal for dynamic robotic environments. Additionally, the complexity of these solutions introduces a learning curve for using the software. This gap highlights the necessity for a new approach that prioritizes efficiency, flexibility, and scalability.

This thesis proposes the development of a lightweight image-processing software platform specifically tailored for robotic intervention applications. The solution will focus on designing and implementing efficient algorithms optimized for real-time performance, minimizing computational resource usage while maintaining high accuracy. By developing modular software components, the platform will facilitate easy integration with a wide range of robotic systems. Furthermore the software architecture and design patters used in the project will enable future expansion.

methods

modules and classes

The project comprises a lightweight graphical user interface (GUI) with multiple specialized modules. Currently, the primary modules include:

- Matplotlib Module: Facilitates the plotting of 2D data.

- VTK Module: Enables the rendering of volumetric data.
- Live Data Module: Establishes connections for real-time data acquisition.

Some of these modules share overlapping functionalities, necessitating the use of abstract modules and widgets. These abstract components implement common methods such as data loading and GUI setup, ensuring consistency and reusability across different modules. This facilitates creating new modules and ensures the absence of duplicate code. In addition to the primary modules, the project includes several utility components:

- A Custom Mathematical Library: Provide specialized mathematical functions.
- Connection Classes: Facilitate the establishment of IPv4 and serial connections.
- Popup Windows: Communicates errors and information to the user.
- Settings Classes: Updates and reads from the settings.json file.
- Data Manager class: Ensures proper file reading and data storage.
- Sidebar Widget: A button that toggles the module on, revealing the module components and its loaded files, or off, concealing them.

Upon launching the application, a data manager class is instantiated. This class serves as a central repository for all loaded or incoming data, ensuring that data is accessible across different modules. The shared instance of the data manager class allows seamless data transfer between modules.

code architecture

Figure 1 illustrates the high-level architecture of the code. The diagram focuses on abstract modules to maintain clarity and brevity. In practice, these abstract classes serve as parent classes, with specific subclasses extending their functionality to implement detailed operations.

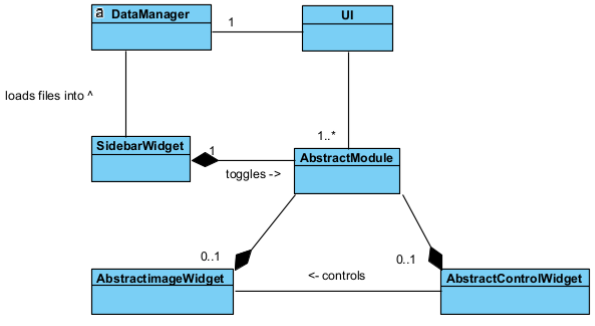


Figure 1: High level architecture of the code

libraries and frameworks

To realise the lightweight GUI, Python 3 is used alongside the library PySide2. PySide2 is a set of Python bindings for the Qt application framework, developed by The Qt Company. It allows Python developers to utilize the functionalities of Qt, a C++ framework used for developing cross-platform applications with a native look and feel. Applications created with PySide2 can run on various operating systems, including Windows, Linux, macOS, iOS and Android without the need of changing source code. Furthermore the MatPlotLib and VTK library are used to integrate 2d and 3d images into the GUI. To establish connections, the python libraries "socket" and "pyserial" are used. Additionally, the libraries numpy, pandas and scipy are used for handling data.

design patterns

For consistent code structure, design patterns are applied. The following patterns can be considered when looking at the code structure:

- Composite Pattern: Modules aggregate multiple widgets, providing a unified interface for interaction.
- Strategy Pattern: Each module employs different strategies for tasks such as plotting data, toggling windows, and interacting with data.
- Model-View-Controller (MVC) Pattern: The GUI acts as the View, DataManager functions as the Model, and each Module serves as the Controller.
- Decorator Pattern: Classes that extend abstract classes add specific functionalities, thereby enhancing the base classes.
- Template Method Pattern: Abstract classes define methods like setup() and plot() that act as templates, allowing subclasses to override specific steps.
- Adapter Pattern: The image widget and control widget adapt VTK and Matplotlib widgets for integration into the GUI.

Figure 2 illustrates these design patterns in more detail. The figure shows that abstract classes have some functions shared across modules, as well as methods that are overridden by specific modules. Additionally, the implemented VTK module acts as a mediator between the VTK image and control widgets, linking the control panel buttons' callbacks to actions in the image widget. Implementing additional modules, such as the MatPlotLib module works in a similar fashion. A module does not necessarily need to include both a control and an image widget. For instance, the connection module solely features a control panel for establishing connections. This approach enhances flexibility in designing new modules.

Algorithms and data flow

The system is designed to accommodate multiple file formats across various visualization modules. Here, a detailed description of the file handling and visualization processes for Matplotlib, VTK, and live data plotting is displayed:

Matplotlib Integration

For Matplotlib, supported file formats include CSV, XLSX, and TXT. The Data Manager module parses the file extension and loads the data into a Pandas DataFrame. This DataFrame serves as the basis for plotting within the image widget.

VTK Integration

VTK supports a wide array of file extensions, that is: "vtk", "vtu", "vtp", "vti", "stl", "obj", "ply", "jpg", "jpeg", "png", "tif", "dicom", "nii", "nii.gz", "mhd", and "DCM". The Data Manager handles these formats by employing the appropriate reader for each file type. This reader is returned and utilized for further processing within the application.

Live Data Plotting

Incomming data from a device is expected in a format where each data point consists of two columns (x and y values) separated by commas, with each data set terminated by a newline. Upon reception, the data is parsed and stored in a Pandas DataFrame. This data is then appended to an animation in Matplotlib. To optimize performance, only the last 100 data points are retained and visualized within the animation.

Visualization in Image Widget

For Matplotlib, data is accessed directly from the Pandas DataFrame for plotting within the image widget. For live visualisation, the animation data is continuously updated by appending new DataFrame entries and maintaining a rolling window of the last 100 data points.

Visualization within the VTK image widget involves initial checks to determine the data type being plotted, such as vtkPolyData or unstructured grids. This step is crucial as it dictates the creation of the appropriate mapper and actor necessary for rendering. Once established, the mapper is linked to the actor, which is then rendered within the widget.

DICOM volume plotting presents unique challenges due to the necessity of loading multiple files. To address this, the Data Manager supports directory opening, associating the directory with a dicomImageReader. This reader facilitates proper rendering within the image widget, adjusting opacity and color based on scalar values derived from the DICOM files.

Data Interaction

The VTK and Matplotlib modules offer diverse methods for interacting with plotted data. The Matplotlib widget control panel features functionalities such as a 'Convolve' button, facilitating the calculation and visualization of the convolution between two datasets. Additionally, it includes a 'Limit' button to constrain data within specified minimum and maximum values. Moreover, it incorporates a 'Filter' button alongside a dialog for configuring Butterworth filter settings.

The VTK control widget provides options to adjust mesh opacity and color. Furthermore, it allows for a camera reset to return to the original mesh orientation.

Test Strategies

The application underwent comprehensive testing employing several strategies to ensure its functionality, reliability, and usability.

Unit, integration, and functional tests were conducted to verify that data could be loaded into the program and interacted with, such as through applying filters and setting mesh colors. Data compatibility and integration were assessed using both the VTK and MatPlotLib modules. A variety of data sets were used, including incorrectly formatted files

Additionally, performance testing was performed. This involved configuring multiple servers to send data at increasing sampling rates. Large datasets, such as extensive DICOM directories, were loaded to evaluate GPU usage and performance.

Finally, the application was tested across multiple versions of Python, including all dependent packages, to ensure compatibility and stability.

results

The final result of this thesis is the application itself, which is available from the RAM gitlab.

To test the performance of the application, big data sets were loaded into the VTK and Matplotlib module. The result of this can be seen in figure 3. While the image widgets showed a slower response time, the user interface was not affected.

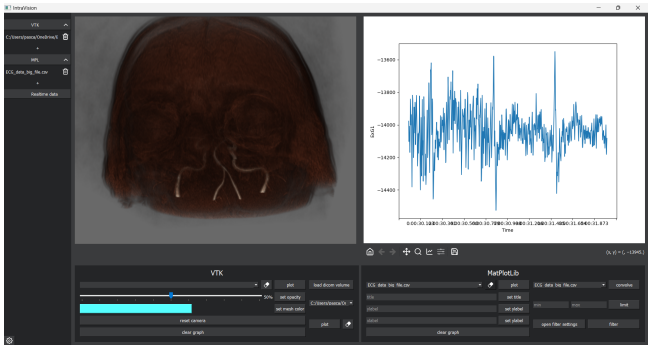
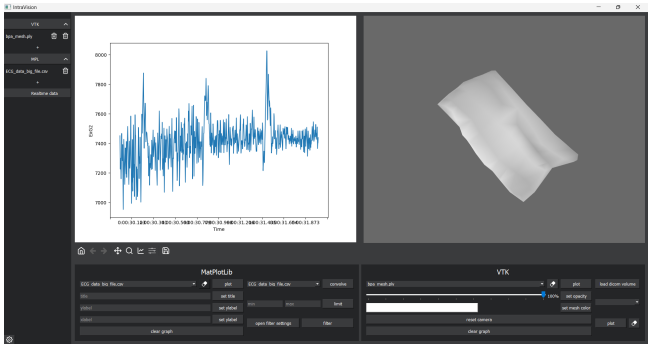
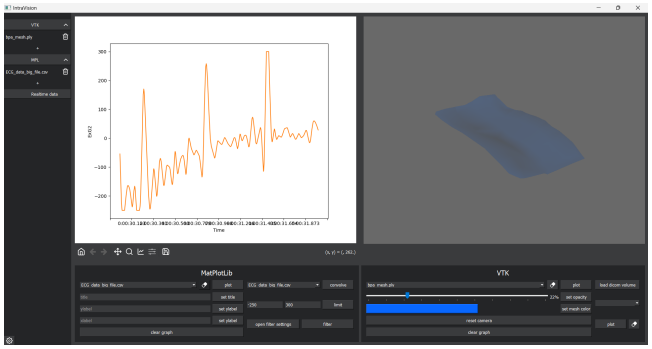


Figure 3: A performance test by loading a big dicom directory that displays the circle of Willis and ECG data, obtained from BMT module 3, that holds several thousands of points.

A second test was performed to confirm the usability of the buttons to interact with the data. Again, data sets were loaded into the VTK and Matplotlib module and plotted. Then, a 4th order bandpass filter was applied with a cutoff of 0.2-40hz and sampling frequency of 1000. Additionally a limit was set to -250 to 300. For the vtk mesh, the opacity was lowered and the color was set to blue. The result of this can be seen in figure 4



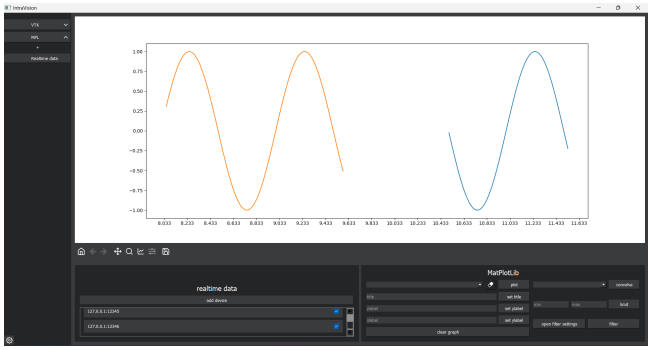
(a)



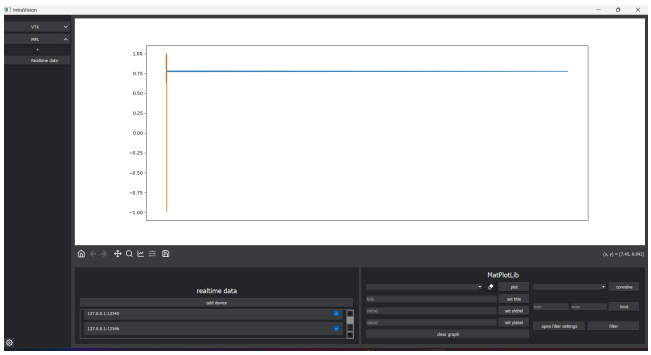
(b)

Figure 4: Test for interacting with data in the Matplotlib module and vtk module. a) Data that has not yet been altered. b) Data after filters and limits have been applied and the opacity and color have been altered.

Finally a test was performed to confirm the functionality of live data visualisation. A script was designed to send sign wave data over a socket at a predefined sample rate. Two of these servers that send the data were initialized and connected with. At a sampling rate of 100 hz, no issuess were observed, but at a sampling rate of 1000 hz data clipping was present. This can be seen in figure 5



(a)



(b)

Figure 5: Result of testing live data visualization. a) The two sign waves are plotted in real time, with no issues. b) Clipping and data loss is observed due to a to high sample rate.

discussion

This thesis presents a software platform enabling data visualization with seamless integration of various file formats and medical devices. It supports 2D and 3D figures, allowing interaction by altering visual and numerical aspects, such as changing mesh colors or applying filters. Testing revealed no major problems, although some issues were identified. Future integration of the Robotic Operating System (ROS) could enhance communication with robotic devices. Currently, the program effectively visualizes data from files and medical devices, demonstrating sufficient functionality and reliability for practical applications.

Real-time data

The application supports multiple methods for connecting with devices to receive real-time data. One method involves establishing an internet connection by providing an IPv4 address and a port number. Upon connection, a new thread is initiated to read and optionally plot the data using Matplotlib.

Another communication method is through a serial connection. The program automatically scans for connected devices and displays them as available ports. The user can select a port and enter the device's baud rate to establish the connection. Similar to the internet connection, a new thread is started to read and plot the incoming data.

Concurrency is a critical consideration when dealing with multiple threads. When multiple devices are connected, their threads attempt to access the image widget and data manager object simultaneously, which can lead to memory leaks. To address this, a thread-

ing lock is implemented to ensure that only one thread can access these objects at any time. This implementation prevents concurrent access issues, maintaining the application's stability and performance.

Current limitations

While the application is functional, several limitations have been identified that impact its performance and usability. Despite the application's general efficiency, large datasets from medical files can significantly slow down the graphical user interface (GUI). This issue is evident in both the Matplotlib and VTK widgets, where plotting extensive data causes notable delays. The handling of large volumes of data results in a bottleneck, as the current implementation struggles to maintain responsiveness when visualizing high-density information. Optimizing data processing and rendering techniques will be crucial for enhancing performance, especially in medical applications where large datasets are common.

When plotting live data, users encounter limitations due to the inability to apply filters or limits in real time. The current system only allows for data alteration processes to be applied to existing data already present within the system, rather than to incoming data streams. This restricts the flexibility and functionality of the application when dealing with live data, as users cannot dynamically adjust visualizations based on real-time data inputs. Addressing this limitation would involve developing mechanisms to process and filter live data as it is received, thereby enhancing the application's real-time data visualization capabilities. Additionally, the application crashes when a connection is broken by an external source.

Furthermore, the application requires live data to follow a specific format, which poses additional constraints. Currently, live data must consist of only two data points for x and y values, separated by a comma, with multiple points being separated by new lines. This rigid format can be restrictive and may not accommodate the diverse data structures encountered in various medical and scientific applications. Expanding the flexibility of data input formats would allow the application to handle a broader range of data types and sources, improving its versatility and user-friendliness.

These limitations highlight areas for future improvement to enhance the application's ability to manage large medical datasets and live data more effectively. By addressing the issues related to data handling efficiency, real-time data processing, and input format flexibility, the application can be significantly improved to meet the demands of complex medical and scientific data visualization tasks.

Future Improvements

As discussed in , the current application does not support the application of filters to live data streams. Addressing this limitation would significantly enhance the usability and functionality of the software. A potential solution is to implement filtering and limiting mechanisms that apply each time data is read from the connection. This would involve modifying the data reading process to include checks for any active filters or limits before the data is visualized.

To achieve this, boolean fields could indicate whether the incoming data should be filtered. These fields would be dynamically updated based on user settings, allowing for real-time adjustments to the data visualization. Filter settings would be managed through the settings.json file, which the application would read to determine the necessary filtering parameters.

However, implementing this feature presents some challenges, particularly in maintaining performance. Continuously reading and applying filter settings from a JSON file at high refresh rates could introduce significant overhead, potentially slowing down the data processing pipeline. This is especially critical in scenarios where data is received at high frequencies, such as real-time monitoring of medical devices. One approach to mitigate these performance issues is to cache the filter settings in memory, reducing the need to repeatedly access the JSON file.

Another potential enhancement is to implement a more flexible data handling framework that supports a variety of data formats and structures. This would involve developing more sophisticated parsing and processing capabilities that can adapt to different data formats. By broadening the range of supported data formats, the application could become more versatile and capable of handling real-time data from sources more effectively.

Finally, integrating advanced data processing technologies, such as parallel processing and GPU acceleration, could further improve the application's performance. Leveraging these technologies would allow the application to handle larger datasets and more complex filtering operations without compromising speed and responsiveness.

Integration with robotic devices

Over the past decade, significant developments have emerged in the field of image-guided robot-assisted interventions. An article by Unger et al describes several methodologies through which robotic devices, in conjunction with imaging devices, can be utilized to enhance medical procedures. [1] Robotic devices offer high precision, which is particularly beneficial for interventions such as needle biopsies and pedicle screw insertion. [2] [3]

The platform discussed in this thesis can be used to interact with both medical images and robotic devices. One way of communicating with robots is through the Robotic Operating System (ROS). ROS includes software libraries and tools designed to be easily integrated with various systems. Connolly et al. developed a module that integrates the open-source 3D Slicer software with ROS, demonstrating its potential for creating practical, image-guided robotic systems [4]. Despite its capabilities, a potential limitation of this module is the incompatibility of some devices with either ROS or 3D Slicer.

Another article by Frank et al. Describes the ROS-IGTL-Bridge, a ROS node that links other ROS nodes to medical imaging software. [5] It does this by establishing a TCP/IP socket with the imaging software and forwarding data received by the robotic system in a formatted matter. Such a bridge can be useful for connect-

ing robotic systems to the imaging platform described in this thesis

conclusion

This thesis presents a novel lightweight imaging platform tailored for robotic interventions, addressing the limitations of existing software frameworks like ROS and 3D Slicer. These traditional platforms, while powerful and suitable for large-scale clinical applications, often struggle with the demands of small-scale robotic systems due to their complexity and resource intensity. The platform developed in this thesis is designed with a modular architecture that prioritizes efficiency, scalability, and ease of integration with a variety of devices, including medical sensors and imaging systems.

The proposed platform features a user-friendly graphical user interface (GUI) composed of specialized modules for 2D data plotting, volumetric rendering, and real-time data acquisition. The use of abstract modules and widgets ensures consistency and re-usability, minimizing code duplication and facilitating the addition of new functionalities. The software leverages Python 3 and libraries such as PySide2, Matplotlib, and VTK to achieve a cross-platform, visually cohesive application. By employing design patterns like Composite, Strategy, MVC, Decorator, Template Method, and Adapter, the platform ensures a robust and maintainable code structure.

Comprehensive testing demonstrated the platform's ability to efficiently process and visualize large datasets and real-time data from various medical devices, revealing no major issues but identifying areas for future improvement. Performance optimization, especially in handling extensive medical datasets and live data, remains a priority. Future work will focus on enhancing real-time data filtering, expanding the flexibility of data input formats, and integrating advanced data processing technologies such as parallel processing and GPU acceleration.

Furthermore, the integration with robotic devices through frameworks like ROS shows great promise for enhancing image-guided robotic systems. The platform's modular design allows for easy adaptation and expansion, paving the way for future developments in medical robotics. Potential enhancements include integrating the Robotic Operating System (ROS) for improved communication with robotic devices and implementing more sophisticated data handling frameworks to support a broader range of data types and structures.

In summary, this thesis lays a solid foundation for a scalable, efficient, and versatile image-processing platform tailored for robotic interventions. By addressing the limitations of existing systems and providing a flexible and modular architecture, this platform has the potential to significantly improve the precision and efficacy of image-guided surgeries, ultimately contributing to advancements in medical robotics and patient care.

References

[1] Unger M, Berger J, Melzer A. Robot-assisted image-guided interventions. Front Robot AI. 2021 Jul;8:664622.

[2] Smakic A, Rathmann N, Kostrzewa M, Schönberg SO, Weiß C, Diehl SJ. Performance of a robotic assistance device in computed tomography-guided percutaneous diagnostic and therapeutic procedures. Cardiovasc Radiol. 2018 Apr;41(4):639-44.

[3] Lefranc M, Peltier J. Accuracy of thoracolumbar transpedicular and vertebral body percutaneous screw placement: coupling the Rosa® Spine robot with intraoperative flat-panel CT guidance–a cadaver study. J Robot Surg. 2015 Dec;9(4):331-8.

[4] Connolly L, Deguet A, Leonard S, Tokuda J, Ungi T, Krieger A, et al. Bridging 3D Slicer and ROS2 for image-guided robotic interventions. Sensors (Basel). 2022 Jul;22(14):5336.

[5] Frank T, Krieger A, Leonard S, Patel NA, Tokuda J. ROS-IGTL-Bridge: an open network interface for image-guided therapy using the ROS environment. Int J Comput Assist Radiol Surg. 2017 Aug;12(8):1451-60.

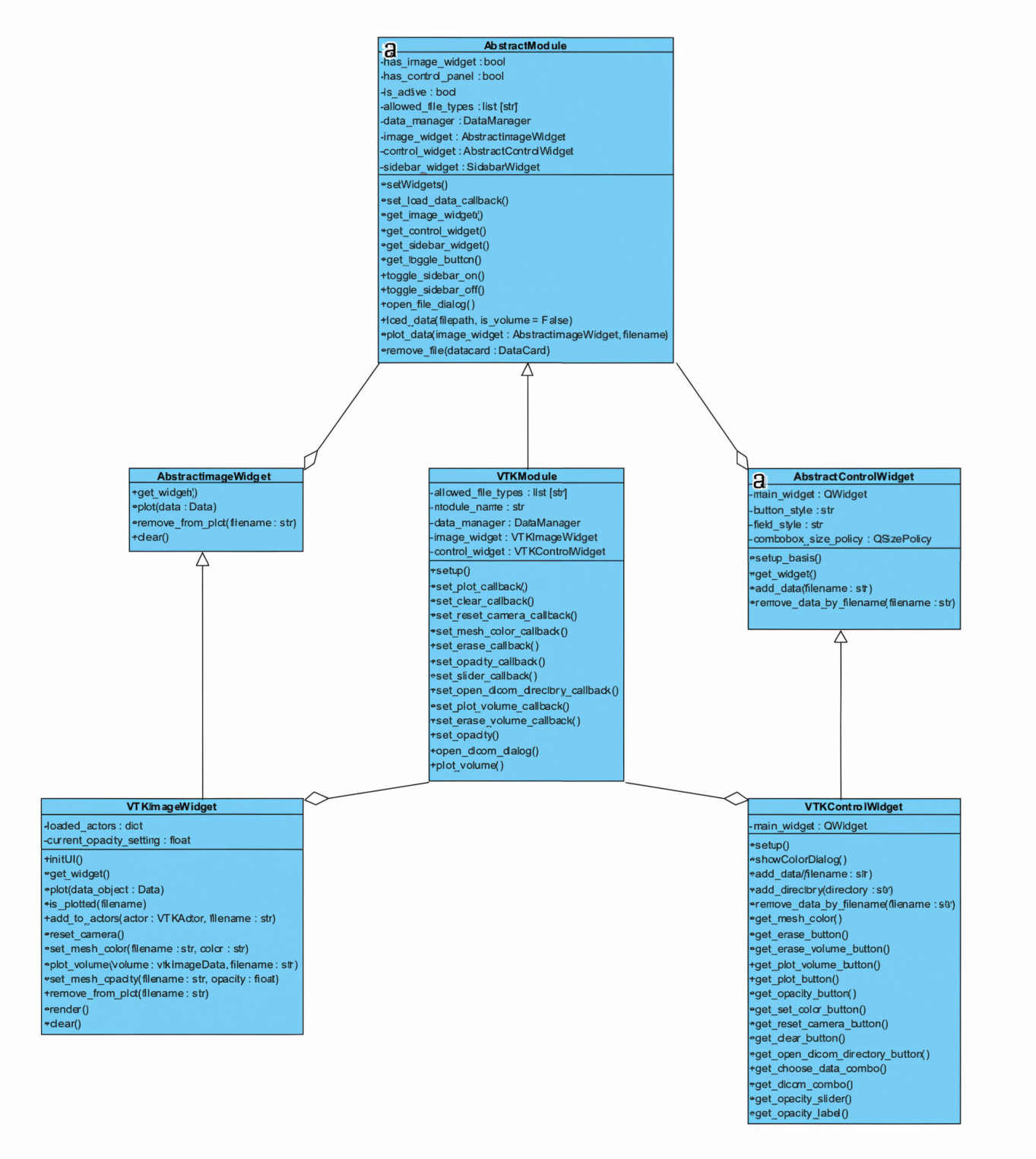


Figure 2: Detailed class diagram of the implementation of the VTK module.