Introduction

Patient controlled analgesia (PCA) describes a method of pain management where patients are given the ability to self-administer safe, fixed doses of analgesics via a PCA device. PCA devices (e.g. infusion pumps) are one of the most common methods of providing post-operative analgesia during patient recovery. Although the history of PCA is relatively short (in use since the 1970s), a substantial number of studies have been published concerning best practices for PCA use for a variety of clinical settings and medications. With the advent of digital computers, PCA devices have become even more user friendly and can provide physicians and researchers with records of a patient's use of the PCA pump. However, few—if any—studies exist discussing the retrieval or analysis of the data that these machines record. This report explores the potential benefits of looking at such data and the impact such investigation could have on patient pain management. Also, this document details current progress in the development of software to aid in PCA report visualization and analysis.

Traditional Pain Management

Post-operative pain management via intramuscular (IM) opioids has been well documented with the traditional approach of nurses or physicians giving injections as needed resulting in 50% of patients receiving inadequate relief of pain [1]. The amount of opioid needed for effective analgesia varies from patient to patient and is identified by titrating opioid dosage until reaching a minimum effective analgesic concentration (MEAC) at which the patient has sufficient pain relief [2]. Plasma opioid concentrations are then measured and attempts are made to maintain opioid levels [2]. Maintaining MEAC, however, is difficult when using traditional IM injection due to variation in both dose frequency and the rate of opioid diffusion through muscle. PCA pumps are an effective tool for maintaining MEAC as they offer on-demand, intravenous (IV) injection of analgesics which reduces the time between injection and pain relief.

Modern Use of PCA

In 1971, Sechzer developed the first prototype PCA pump and, since then, such devices have become commonplace in post-operative pain management [3]. Modern PCA machines can be used with a variety of injectable drugs and typically offer two modes of operation: (1) fixed-dose on patient demand and (2) a combination of a continuous infusion and fixed-dose available on patient demand. Both modes of operation require the physician/nurse to specify an initial loading dose, demand dose, lockout interval, background infusion rate, a 1-hour dosage limit, and a 4-hour dosage limit to ensure patient safety [4]. Taking into account the necessary safety considerations, these devices allow for the efficient maintenance of MEAC, and in turn minimize pain, with little risk to the patient. Additionally, these devices internally record their activity (e.g. successful deliveries, requested deliveries, timestamps, dosage, etc…) and performance reports can be extracted from each pump.

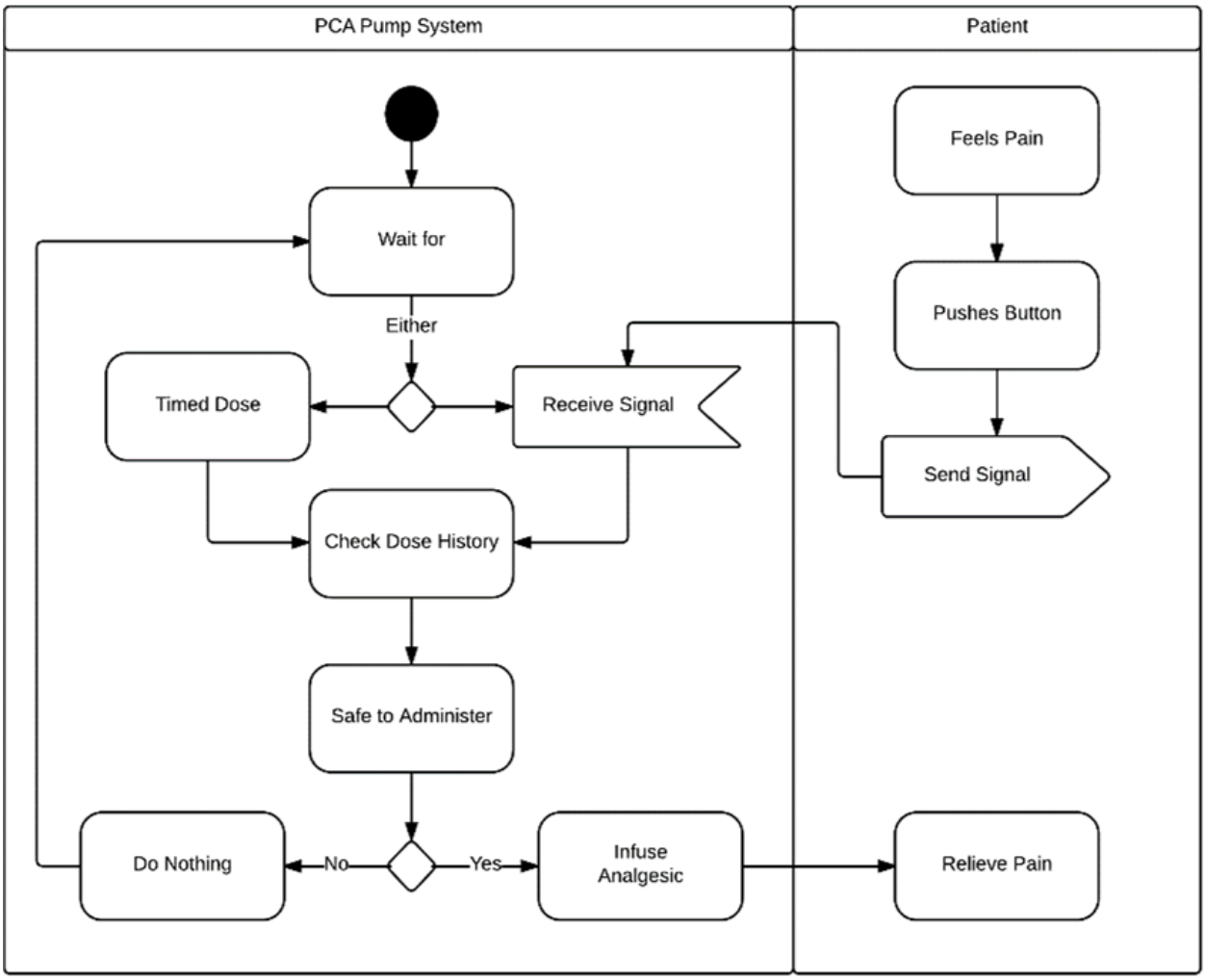

Figure 1: PCA pump UML system diagram

Lack of Quantitative Analysis

With the widespread use of PCA machines across the country, it is interesting that little investigation has been conducted into the use of PCA data for trend analysis and the assessment of patient care. Numerous clinical studies have been conducted to qualitatively assess patient pain management using various opioid dosing regimens . However, our investigation into existing literature concerning PCA data analysis has revealed only a single study by Lin et al. which attempts to predict PCA analgesic consumption using cluster analysis .

Statement of Objectives and Goals

The objective of this project is to begin filling the gap between PCA data analysis and patient care by creating software which will facilitate the visualization, interpretation, and analysis of PCA reports by physicians and researchers. This software will automate the generation and presentation of interactive graphs and statistics about the use of a PCA unit. To automate this process, two main goals need to be achieved: First, PCA log data needs to be deciphered, parsed, and reformatted to enable automated information extraction. Ideally, this parsing and reformatting process will also be automated to the point that an individual need simply open a PCA log in the application software. Second, the front-end software needs offer a clean, simple, and intuitive user interface that presents information that will be valuable to physicians and researchers.

Design Considerations

Security

Given that the purpose of this software is to analyze patient PCA activity, precautions have to be taken to protect PHI. Initially, we had hoped to use web-based APIs to develop this application since they offer a variety of tools for data visualization and have a very clean, crisp look. However, due to concerns about information security, this application has been designed to work independently of the web and instead utilizes the Qt framework for UI development. This decision trades off style for independent operation and security.

Usability

Within software engineering, usability is a broad term which encompasses how well software can be used by the specified consumer to achieve objectives with efficiency and effectiveness. Learnable and intuitive interfaces is an important part of all systems (not just software) and should be constructed with the personas of the users in mind. In this case, the end users take the form of physicians and research scientists who are interested in analyzing use and trends of patient-controlled medication delivery systems.

Automation

To minimize the workload placed on the user, it is necessary to efficiently automate both the reading of PCA logs as well as the processing of log data and the generation of information. One potential end goal of automation would be to allow the user to import a PCA log via a file browser and then, upon file selection, automatically populate the UI elements with graphs, statistics, etc…

The information needs to be extracted from PCA logs

Infusion Activity

Infusion activity encompasses both patient requests for medication and the delivery of medication by the pump. This time series information is presented in the form of an interactive bar graph which can be resized and scrolled to visualize the changes in patterns of patient requests and medication deliveries over time.

Distributions

Distributions of patient requests for medication, the delivery of medication, and the average time delays between patient requests offer insight into analgesic consumption across a group of patients and are useful for discerning trends.

Summary Statistics

Summary statistics such as mean and standard deviation concerning patient requests, medication deliveries, and time between requests will be displayed both for the group of patients as well as each individual patient. This information offers further insight into PCA use by each individual and enables comparison within the whole group. Additionally, total numbers of requests, deliveries, and resting periods will also be included.

Cross-platform Support

Ultimately, this software will be released to the public as a tool for use by physicians and researchers. To make this process as painless as possible, we have elected to develop this application using the open-source Qt framework which is compatible with Windows, OS X, and Linux systems.

Application Development

Development Software, APIs, and Dependencies

Enthought Canopy

Canopy is a python IDE which places numerous python packages at your fingertips. Similar to Anaconda, Canopy offers a convenient package installer and allows for the quick installation of any libraries that you need. Furthermore, Canopy now offers a debugger with breakpoints and line-by-line stepping and execution. Canopy is free for students with an academic (.edu) email address. Application development thus far has been carried out using Canopy.

Qt

Qt (pronounced cute) is a cross-platform framework for developing software applications and supports Linux, OS X, and Windows along with select mobile platforms. Written in C++, Qt offers the ability to write code on one operating system and then simply copy, compile, and run the same code on another system. The Qt libraries were selected for the development of this application to facilitate the eventual open sharing of the software.

PySide

PySide is a python wrapper for the Qt development libraries. As such, PySide allow for the development of complex UIs via python code using Qt's C++ libraries. PySide can be easily installed via Canopy's package installer. Click here for a direct and semi-comprehensive PySide tutorial.

Qt Creator

Qt Creator is a cross-platform integrated development environment (IDE) which offers numerous tools to speed up application development. Of particularly interest is the fact that Qt Creator offers the ability to "drag-and-drop" UI objects to interactively create an interface without having to write every line of code. However, one expectation with this IDE is that you utilize Qt's C++ libraries directly and code functionality in C++. While this prevents us from being able to completely code the application within the IDE, we can still use the IDE to construct the layout of our application, convert our UI file into a python-friendly format, and then use the convenient python wrapper (PySide) to code the functionality of our application. While this method might seem convoluted, it has facilitated the quick overhaul of UI layout without having to significantly alter code. Additional information concerning UI file creation and conversion can be found in future sections under the "UI Design" heading.

PyQtGraph

PyQtGraph is an open-source, pure python graphics library that can be used to create interactive plots and graphs. PyQtGraph was selected not only for its interactivity (compare with matplotlib which offers only static plot types) but also because it interfaces easily with the Qt GraphicsView framework. Additional information concerning the interfacing process can be found under the "UI Design" and "Functionality" headings.

Other Dependencies

Several python libraries were used in the development of this application. Although most are standard and dependencies are automatically installed when installing some of the main packages (e.g. pyqtgraph), all libraries have been listed here:

- PySide
- pyqtgraph
- sys
- os

- pandas
- numpy
- scipy
- functools

Software Installation

Enthought Canopy

Canopy can be downloaded for free from the following link using an academic email address. The installer should take care of the setup.

Qt / Qt Creator

Both the Qt libraries and Qt Creator can be installed from the following link.

PySide

PySide can be easily installed via Canopy's package manager. Simply search for pyside and install.

PyQtGraph

PyQtGraph can be installed via pip from within the "Canopy Command Prompt." Simply run python and type "pip install pyqtgraph" Alternate installation methods are listed here.

UI Design

The following section details the development of the front-end user interface layout along with the conversion process required to utilize Qt Creator UI files in python code.

UI Layout

Early discussion of user interface layout resulted in the construction of a "four quadrant" type design which provides the user with the ability to import logs, visualize medication requests/deliveries across time, look at the distributions of patient behavior, and quickly obtain summary statistics regarding PCA use. This general design was retained throughout the development process although additional features were added to improve usability.

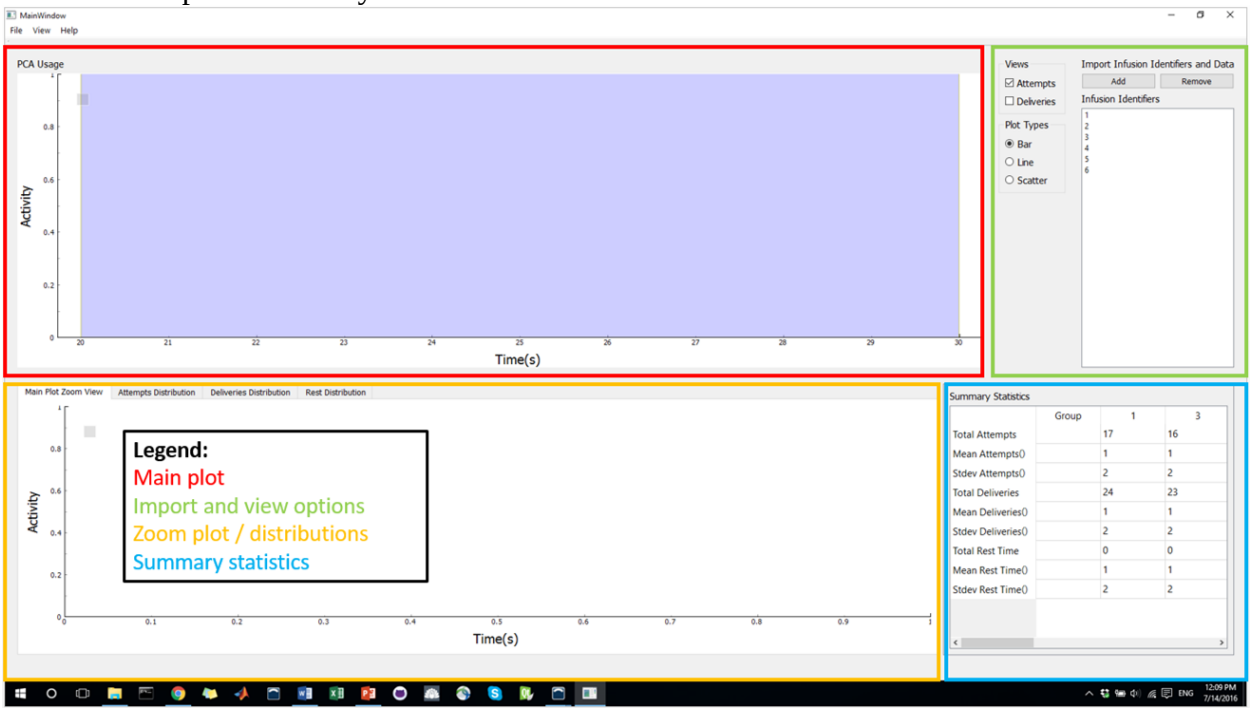

Figure 2: Screen shot of the current application with the four quadrants highlighted.

UI Components

The following subsections describe the components/elements of the current UI and offer a visual. Components have been manually filled with dummy data to facilitate debugging. This dummy code will have to be replaced with data structures created by preprocessing code.

Menu Bar

The "menu bar" offers *File*, *View*, and *Help* tabs which enable the user to perform various functions. Under the *File* tab are **Open** and **Exit** buttons which perform their intuitive functions. The *View* tab is current inactive with the goal of adding options to look at both the *Zoom Plot*, and *Distribution Plots* without having to click on the actual tabs. Finally, the *Help* tab (current inactive) will eventually link to a user manual which will open when this button is clicked.

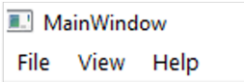

Figure 3: Screen shot of the menu bar.

Log Import and View Options

This subsection of the application provides a variety of features listed below:

1. Add
    a. This button has the same functionality as the *Open* menu bar option and allows the user to import a log.
2. Remove
    a. This button removes the currently select identifier from the infusion identifiers list box.
3. Infusion Identifiers List Box
    a. This list box contains a list of the infusion identifiers extracted from the log and allows the user to select specific infusions.
4. Views Group Box
    a. This box offers the user the ability to view medication requests, medication deliveries, or both on the main plot.
5. Plot Types Group Box
    a. This box offers the user the ability to change the type of graph format for the main plot. These currently include bar, line, and scatter plot formats. However, given the data structure currently used to store the attempts and deliveries information, the line and scatter plot formats have become obsolete and should be removed.

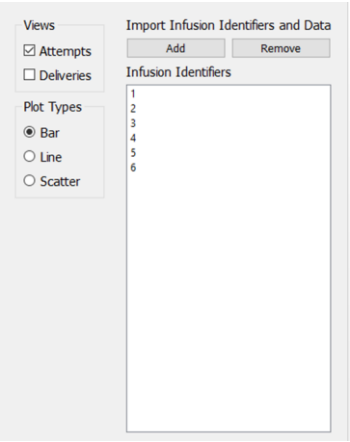

Figure 4: Screen shot of the log import and view options.

Main Plot

The main plot offers a visual depiction of an individual's requests and deliveries across time. The user can scroll through the plot and zoom in on sections. The plot also contains a legend to aid in differentiating between request and deliveries. Finally, a blue overlay on the plot allows the user to select a subsection of the main graph to enlarge in the *zoom plot* tab below.

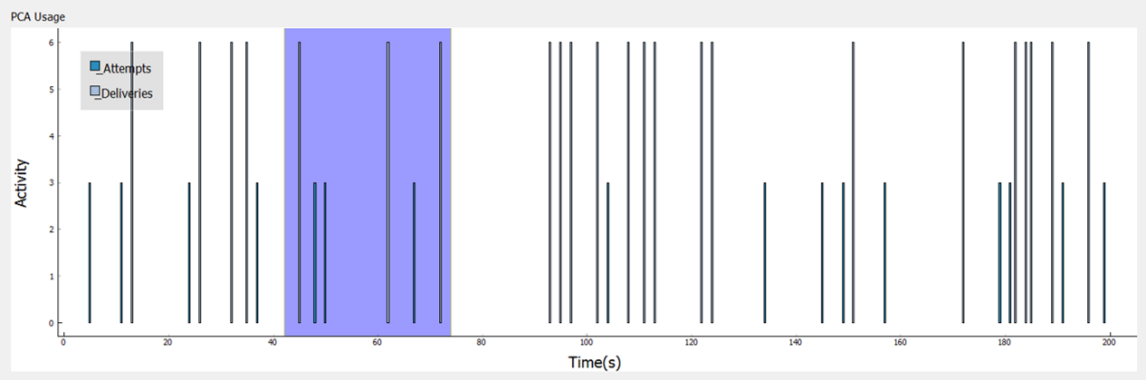

Figure 5: Main plot showing example attempt (request) and delivery data.

Zoom Plot

The zoom plot offers a zoomed-in view of the region of the main plot that is covered by the blue linear region. The zoom plot offers the same features as the main plot and is located in the set of tabs below the main plot.

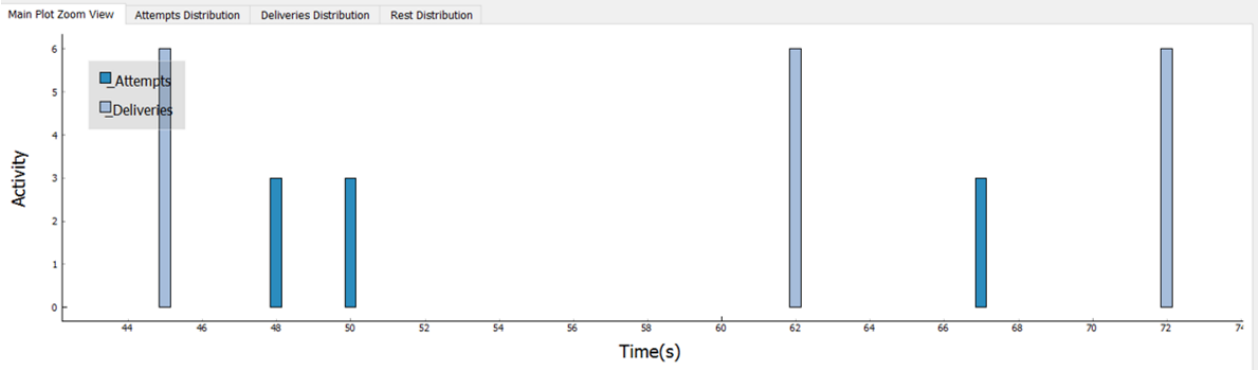

Figure 6: Zoom plot depicting a subsection of the main plot.

Distribution Plots

The distributions plot tabs are located in the same set of tabs as the *zoom plot* and offer distributions of patient request for medication, delivery of medication, and the time between patient requests for

medication (aka rest time). Note: The “rest time” distribution will be styled identical to either the request or delivery distributions. As such, it is not shown below to conserve space.

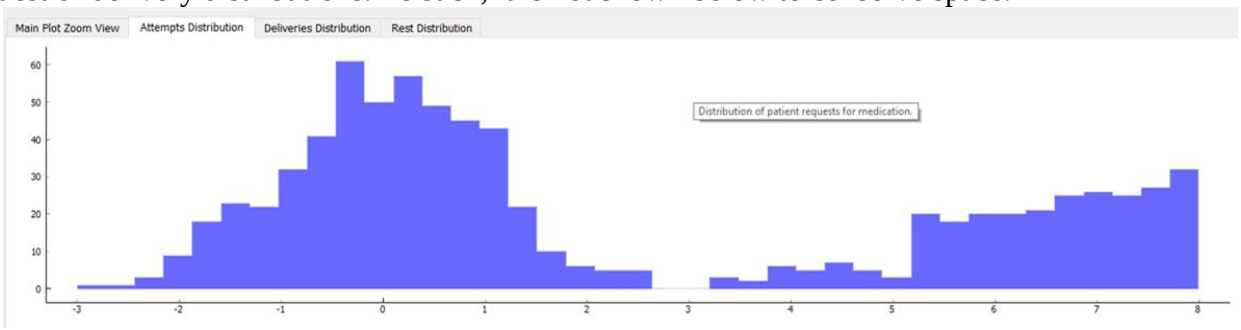

Figure 7: Example medication requests (attempts) distribution.

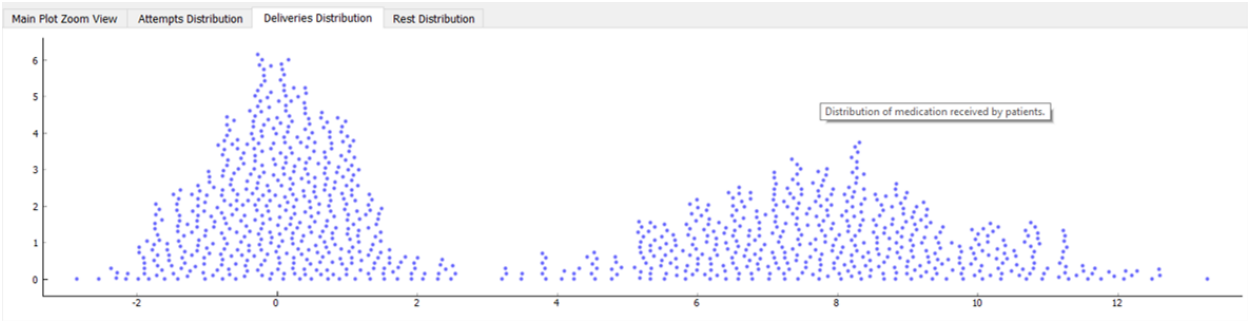

Figure 8: Example medication deliveries distribution in another style.

Summary Statistics

This table offers summary statistics regarding the requests, deliveries, and resting time for 1) all patients in the list box and 2) each individual. Currently, the data found within the table is dummy code that will need to be replaced actual statistics and properly placed within the table.

Summary Statistics

	Group	1	3
Total Attempts		17	16
Mean Attempts()		1	1
Stdev Attempts()		2	2
Total Deliveries		24	23
Mean Deliveries()		1	1
Stdev Deliveries()		2	2
Total Rest Time		0	0
Mean Rest Time()		1	1
Stdev Rest Time()		2	2

Figure 9: Example summary statistics table. Dummy data has been filled into the columns.

Interfacing with PyQtGraph

PySide widgets created in Qt Creator can be easily setup to work with PyQtGraph objects. To create a widget that works with PyQtGraph:

1. Select your desired widget, right-click, and select “Promote to…”
2. Next, select the base class as “QGraphicsView” and fill in the *Promoted class name* field with “PlotWidget” and the *Header file* field with “pyqtgraph.”
3. Then, press ADD.

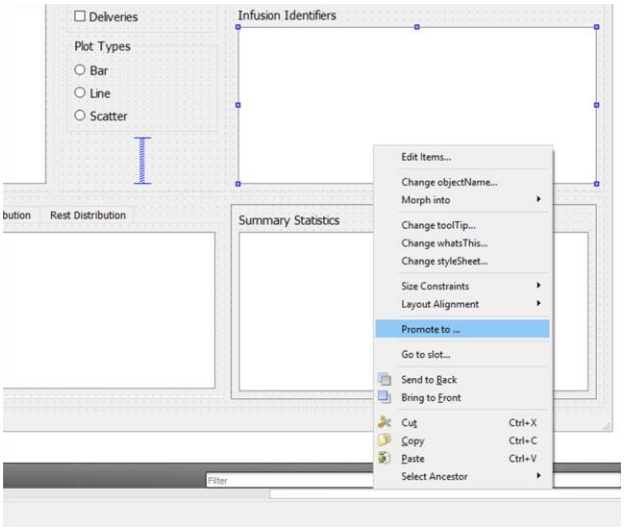

Figure 10: Step 1

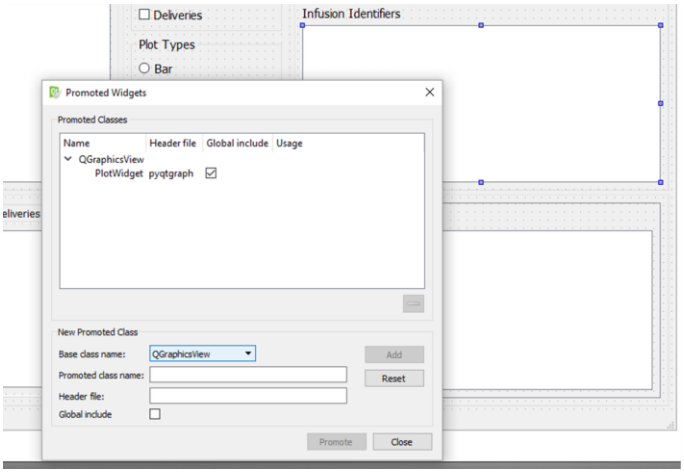

Figure 11: Step 2

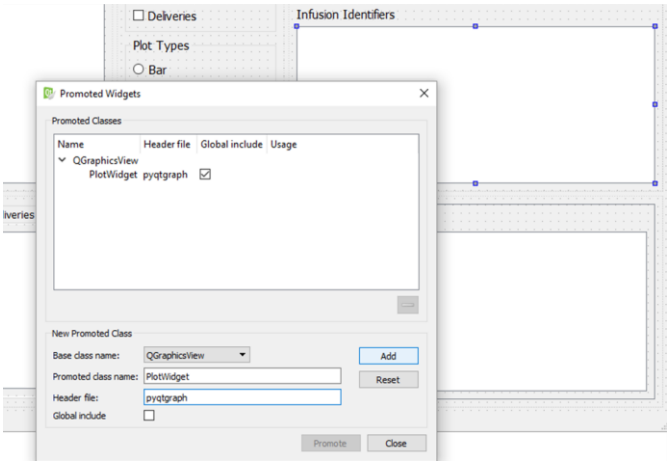

Figure 12: Step 3

This widget will now behave as a pyqtgraph plot widget.

Conversion of *.ui File to Python Code

Once you have a Qt Creator project and are ready to convert your UI file to a python file, do the following:

1. Navigate to your UI file and save the path in your cut buffer.

2. Open the command prompt and navigate to the UI file via the saved path.
3. Type the following command: "pyside-uic mainwindow.ui -o mainwindow.py"
4. A python file (mainwindow.py) will be created that you can import into your main class.

UI Functionality

The functional code for all UI widgets (buttons, windows, etc…) extracted from the imported "mainwindow.py" file are contained in a separate, single-class file titled "MainWindow"
All class functions have been defined to interact with the widgets and code functionality. The remaining subsections detail the functions tied to each widget as well as some of the details of pyqtgraph functionality. Furthermore, information about the data processing and storage can also be found below.

UI Functions

This section describes the functions that implement the behavior of various buttons and windows found within the interface. Some of the functions listed below require several arguments. However, for brevity, they have not been included in these document headers. The required arguments can be found in the function definitions within the python notebook.

__init__()

This function is the constructor for the MainWindow class and initializes some mainwindow properties and—currently—generates the dummy debugging data that is place in the widgets. Several important functions defined further down are also called within this constructor.

createActions()

This function creates "actions" that provide functionality for the tabs within the menu bar. However, it is important to note that these actions are not assigned to the menu bar options within this function. That occurs in the assignWidgets() function below.

addToolTips()

This function creates and assigns the various tooltips that can be seen when hovering over an object in the UI.

assignWidgets()

This function connects widgets extracted from the "mainwindow.py" file with code that either edits widget properties or provides functionality (e.g. state changes, button clicked, region change).

updateZoomPlot()

This function updates the zoom plot when the linear region on the main plot is resized or moved.

updateZoomRegion()

This function updates the zoom region (the blue linear region contained on the main plot) when the zoom plot is shifted/zoomed by the user.

updateMainPlot()

This function updates various components of the main plot and relies on information from the other widgets to properly display attempts/deliveries. It identifies the selected dataset to plot; determines whether the user wants to display attempts, deliveries, or both; selects the desired plot format, plots the data, and updates the plot legend.

updateStatsTable()

This function calculates the summary statistics indicated by the table headers and populates the summary statistics table with values. Note, the mean and stdev calculations have not yet been implement. Currently, dummy variables are used to populate the table. These dummy values are contained WITHIN this function and need to be replaced with the results of actual calculation.

removeListItem()

This function removes the currently-highlighted item in the listbox. It does NOT remove the data from the main data structure. This should be implemented in future versions.

barChanged()

This function updates the main plot when the state of the *bar* checkbox is altered.

scatterChanged()

This function updates the main plot when the state of the *scatter* checkbox is altered.

lineChanged()

This function updates the main plot when the state of the *line* checkbox is altered.

closeEvent()

This function intercepts the exit event and puts up a message box asking the user whether they want to exit the application.

openFileDialog()

This function is called when the user attempts to import a log file. It opens a local browsing window and lets the user select a log file. Currently, once the user selects a file and presses *Okay*, the file is loaded into a pandas dataframe. It was intended that, in future versions, this function will automatically populate the listbox with infusion IDs, conduct various preprocessing tasks, populate the summary statistics table and distribution plots, and add content to the main plot using default settings. These features/functions have **NOT** yet been implemented.

main()

The following code is located at the bottom of the python notebook and is responsible for running the application. Only a single PySide application can be running at a time. As such, this function checks whether an application instance currently exists and, if so, uses the current instance to run the program.

```
In [3]: def main():
            app = QtGui.QApplication.instance()
            if app is None:
                app = QtGui.QApplication(sys.argv)

            ex = MainWindow()
            sys.exit(app.exec_())

In [4]: if __name__ == '__main__':
            main()

An exception has occurred, use %tb to see the full traceback.

SystemExit: 0


To exit: use 'exit', 'quit', or Ctrl-D.
```

Figure 13: Application main function and boilerplate code to run main.

PyQtGraph Functionality

PyQtGraph code can be found throughout the python notebook and can be identified by the “pg” tag in front of variables and functions. For more information concerning pyqtgraph code, see their documentation here. Additionally, you can open up a set of functional example programs AND view the source code by opening the canopy command prompt, running python, importing pyqtgraph examples, and then running “examples.run()”

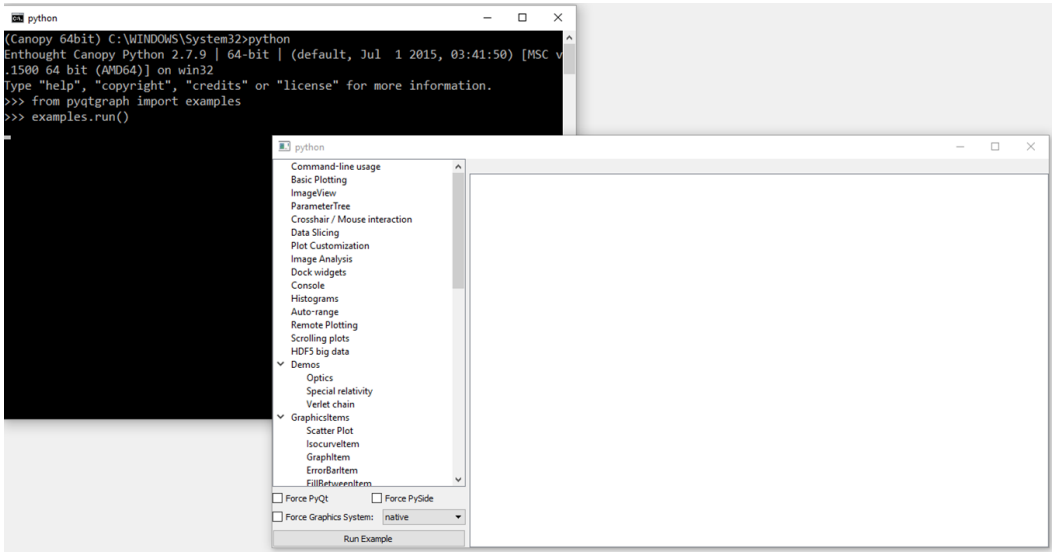

Figure 14: Running the pyqtgraph examples application.

Data Storage

The dummy test data current populating the UI is stored as a dictionary of dataframes. Each dictionary key represents an infusion ID and the associated dataframe contains two columns (x and y) which, in turn, contain time points normalized to the starting point (i.e. zero) and numbers associated with medication attempts (the number 3) and deliveries (the number 6). Since much of the code within this class file has been written with this structure in mind, it is probably wise to retain this data structure format. However, this is entirely up to the future developer and can be changed should a better idea exist.