Code Report - Amanda Shack

Short Project and Mandate Description

> **Automated Jet Tracking to steer a laser beam into a flow of liquid containing biological crystals.**
>
> - Has multiple simultaneous data inputs
> - Must show live data feed on graphs
> - Must be reliable and maintainable on the long term (Research Center)
>
> **Amanda requires help for the following topics:**
>
> - High Refresh rate of the plots crashes the plots.
> - Help with the structure of the code
> - General guidance for PyQt Application Development

What we should talk about

> - Wrote a report with general comments and code that you could use
> - Talk about workflow / QtDesigner / IDE
> - Talk about the compartmentalization in PyQt and applications in general.
> - Go over the code I did to compartmentalize the code
> -

General comments on the project:

- **Project not easy to set up (Problem!)**
  - I ran `pip install .` to use the `setup.py` =>

    ```
    ERROR: Could not build wheels for cf-units which use PEP 517 and cannot be installed directly
    ```

  - I ran `python setup.py`
  - I ran `pip install -r requirements.txt` and `pip install -r requirements-dev.txt`
  - Whatever I did, I always had missing packages.
  - At the end:
    - `python testscreen.py` => missing `zmq` (which is not even used)
    - `python jettracking.py` => no module named `MySQLdb`
    - Had to comment the `zmq` and `import IMS` for the application to start

- **Removing unnecessary files and `imports` statements**
  - e.g `context.py` `/psana` `jet_tracking`(argparser)
  - Or at least put them in a `sketch` folder

- **Python Package Format ?**

- From what I understood, you wanted this to be a `PyPi` package, so the people using this can just run `pip install jet-tracking` and then `python -m jet-tracking` and be presented with the UI ?
- If that's the case :
  - I'd create a `__main__.py` where I would launch the `main.py`

- **Put the documentation** link inside the `README.rst` : http://pswww.slac.stanford.edu/swdoc/releases/jet_tracking/

- **Project Structure Redesign**
  - Adding folders structure (suggestion)

```
.
├── .github
│
├── conda-receipe
│
├── docs
    ├── conf.py
    └── index.rst
│
├── jet_tracking
    ├── __init__.py
    ├── __main.__.py
    ├── gui
    |    ├── windows
    |    |     ├──mainWindow.ui
    |    |     └──mainWindow.py
    |    ├── views
    |    |     ├──jetSteeringUi.ui
    |    |     └──jetSteering.py
    |    └── widgets
    |    |      ├──jetSteeringUi.ui
    |    |      └──jetSteering.py
    ├── tools
    |    ├── num_gen.py
    |    └── signals.py
    ├── logs
    |    ├── log0.log
    |    └── log1.log
    ├── _data
    |    └── data1.yml
    ├──
    └── index.html
│
└── setup.py
└── Versioneer.py
└── requirements.txt
└── README.rst
```

- **Setup a proper logging system (rather than prints)**

- In your `main.py` you would usually add

```
import logging.config
from logging.handlers import RotatingFileHandler
import os

log = logging.getLogger(__name__)


@staticmethod
def init_logging():
    logger = logging.getLogger()
    logger.setLevel(logging.NOTSET)

    # Console Handler
    handler = logging.StreamHandler()
    handler.setLevel(logging.INFO)
    formatter = logging.Formatter(
        "%(asctime)s\t (%(name)-25.25s) (thread:%(thread)d) (line:%(lineno)5d)\t[%(levelname)-5.5s] %(message)s")
    handler.setFormatter(formatter)
    logger.addHandler(handler)

    # Log Error File Handler
    os.makedirs(application_path + "{0}log".format(os.sep), exist_ok=True)
    handler = RotatingFileHandler(application_path + "{0}log{0}opt-id.log".format(os.sep), maxBytes=2.3 * 1024 * 1024, backupCount=5)
    handler.setLevel(logging.ERROR)
    formatter = logging.Formatter("%(asctime)s\t (%(name)-30.30s) (thread:%(thread)d) (line:%(lineno)5d)\t[%(levelname)-5.5s] %(message)s")
    handler.setFormatter(formatter)
    logger.addHandler(handler)

@staticmethod
def handle_exception(exc_type, exc_value, exc_traceback):
    if issubclass(exc_type, KeyboardInterrupt):
        sys.__excepthook__(exc_type, exc_value, exc_traceback)
        return

    log.error("Uncaught exception", exc_info=(exc_type, exc_value, exc_traceback))
```

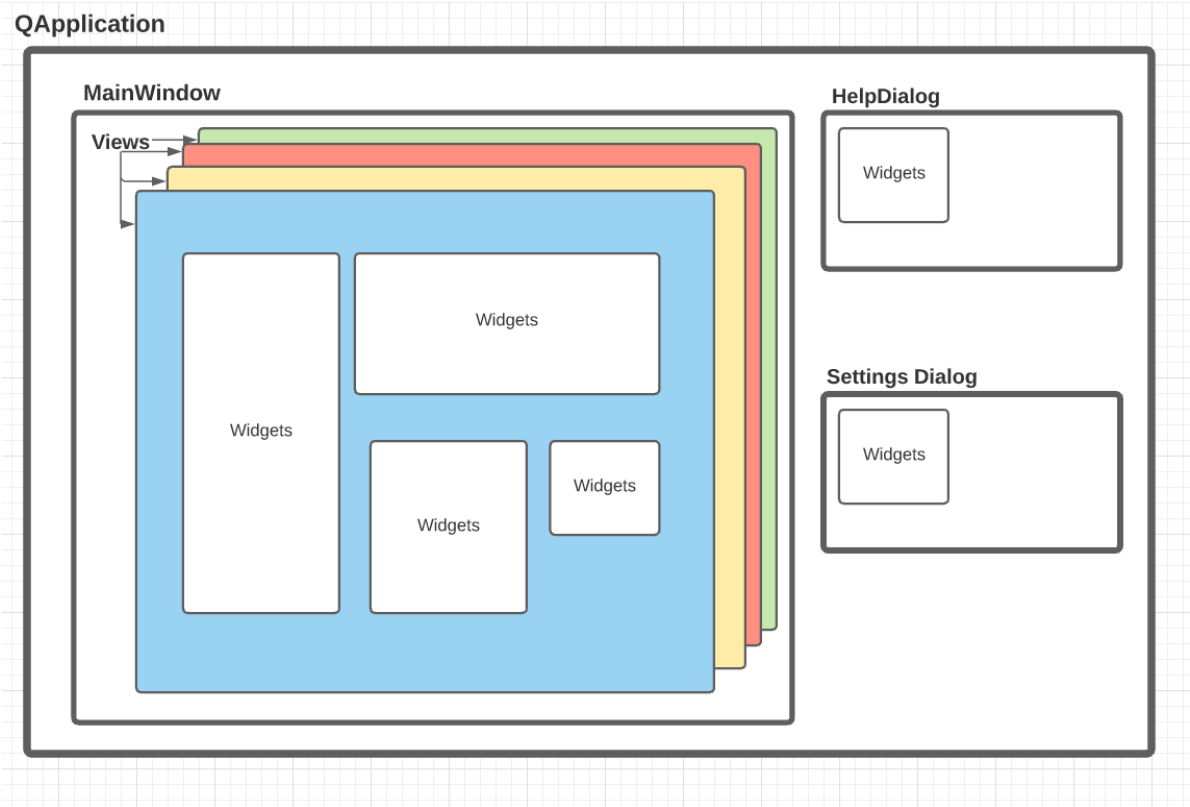
- In the following files, if you want to access your *magnificent* logger

```
import logging
log = logging.getLogger(__name__)
```

- More consistent PEP8 check and consistent naming convention will help a lot. (more on the pyqt convention when showing QtDesigner)

PyQt Specific Analysis

- Usual structure of a long lasting Desktop Software

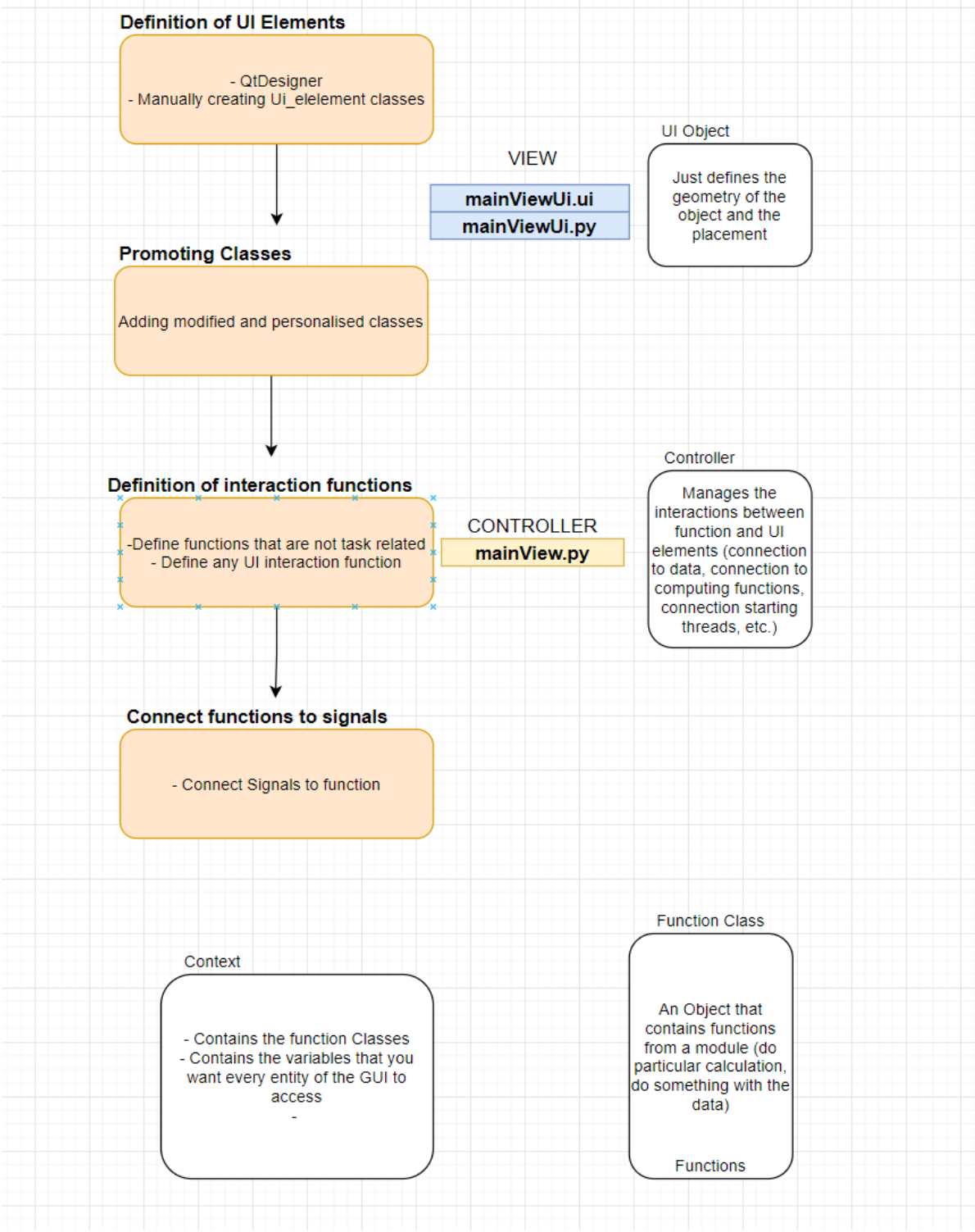


- You want to separate the UI elements from the controller elements
- Setup a `QApplication` `class` and renaming `jettracking.py` to `main.py` (convention)

```
class App(QApplication):
    def __init__(self, sys_argv):
        super(App, self).__init__(sys_argv)
        log.debug("This is the mainThread")

        self.setAttribute(Qt.AA_EnableHighDpiScaling)
        self.setStyle("Fusion")
        self.mainModel = MainModel()
        self.mainWindow = MainWindow(model=self.mainModel)
        self.mainWindow.setWindowTitle("jet-tracker")
        self.mainWindow.show()

def main():
    # Makes the icon in the taskbar as well.
    appID = "opt-id"  # arbitrary string
    if os.name == 'nt':

 ctypes.windll.shell32.SetCurrentProcessExplicitAppUserModelID(appID)
    elif os.name == 'posix':
        pass

    app = App(sys.argv)
    app.setWindowIcon(QIcon(application_path + "
{0}gui{0}misc{0}logo{0}logo3.ico".format(os.sep)))
    sys.exit(app.exec_())

if __name__ == "__main__":
```

```
    main()
```

- Use a "model" that will contain your objects, and pass your model to your instances. Like you did with `Context`



- If you're to start using QtDesigner, th emost efficient way to convert .ui to .py, so that changes are loaded automatically, is this:

```
simulationViewUiPath = os.path.dirname(os.path.realpath(__file__)) +
"\\simulationViewUi.ui"
Ui_simulationView, QtBaseClass = uic.loadUiType(simulationViewUiPath)
```

This will use the `pyuic5` from `PyQt` and convert `.ui` filetypes to `.py` (like your are doing).

- No execute all the connection function in the `__init__()`, I'd try to compartmentalize!:

```
class SimulationView(QWidget, Ui_simulationView):
    SIGNAL_toggled_plot_indicator = "indicator"

    def __init__(self, context=None, controller=None):
        super(SimulationView, self).__init__()
        self.model = model
        self.allPlotsDict = {}
        self.setupUi(self)
        self.connect_buttons()
        self.connect_signals()
        self.connect_checkbox()
        self.create_plots()
        self.initialize_view()
```