**Декомпозиция проекта "Виртуальный лазерный станок"**

**1. Общая архитектура и разделение на модули**

Проект состоит из **двух частей**, взаимодействующих по **TCP/UDP**:

1. **Серверная часть (станок)** – эмулирует работу лазерного станка, выполняя команды управления.
2. **Клиентская часть (GUI + логика)** – предоставляет интерфейс для управления станком и обработки изображений.

**1.1 Серверная часть (станок)**

* **Имитация работы шаговых двигателей и лазера.**
* **Сетевое взаимодействие (TCP/UDP)** – обработка команд от клиента, передача текущего состояния.
* **Контроль ограничений:**
  + Поле обработки **500×500** единиц.
  + Проверка скорости и координат.
  + Лазер можно включать/выключать только в пределах рабочего поля.
* **Логирование событий:** регистрация действий (включение лазера, перемещения, ошибки).

**1.2 Клиентская часть (GUI + логика)**

* **Графический интерфейс (Qt/PyQt)** – управление станком, ввод параметров.
* **Визуализация движения лазера** (текущая и предыдущая позиции, сетка, история пути).
* **Обработка ЧБ-изображений** – анализ пикселей, преобразование в команды для станка.
* **Сетевое взаимодействие (TCP/UDP клиент)** – отправка команд, получение состояния.

**1.3 Вспомогательные модули**

* **bresenham.py** – алгоритм Брезенхэма для пошагового движения лазера.
* **network.py** – базовые классы для работы с сетью (сокеты, сериализация).
* **utils.py** – логирование, вспомогательные функции, чтение конфигов.

**2. Серверная часть – "Станок"**

**2.1 Компоненты сервера**

1. **Модель состояния (MachineState)**
   * Координаты: x, y.
   * Скорость: speed.
   * Состояние лазера: laser\_on.
   * Ограничения: max\_x = 500, max\_y = 500.
2. **Обработчик команд (MachineController)**
   * turn\_on(), turn\_off() – управление лазером.
   * move\_to(x, y, speed) – пошаговое перемещение.
   * get\_status() – получение текущего состояния.
   * Проверка границ (x, y ≤ 500).
3. **Механизм оповещений**
   * Сообщение клиенту о начале и завершении движения.
   * Регулярная отправка текущих координат.
   * Логирование событий.

**2.2 Алгоритм движения**

* Использование **алгоритма Брезенхэма** для пошагового перемещения.
* Эмуляция "реального времени" через **задержки** (time.sleep(step\_time)).
* Параллельное выполнение движения и приема команд.

**2.3 Сетевое взаимодействие (сервер TCP/UDP)**

* Принимает команды: {"cmd": "move", "x": 100, "y": 200, "speed": 5}.
* Отправляет статусы: {"status": "ok", "x": ..., "y": ..., "laser\_on": ...}.
* Использует многопоточное или асинхронное выполнение (asyncio).

**2.4 Логирование**

* Логирование всех команд, событий движения и ошибок.

**3. Клиентская часть – "Интерфейс управления"**

**3.1 Графический интерфейс (PyQt)**

1. **Главное окно (QMainWindow)**
   * Поля для ввода координат (QSpinBox, QLineEdit).
   * Поле скорости (QSpinBox).
   * Кнопки управления (QPushButton):
     + "Поехать в точку"
     + "Включить/Выключить лазер"
     + "Загрузить изображение"
     + "Сканировать"
2. **Окно просмотра (QWidget)**
   * Отображение **сетки, текущего положения лазера, пройденного пути**.
   * **Зумирование и панорамирование** (как в графических редакторах).
   * Обработка **кликов** (QMouseEvent) – отправка команд move\_to(x, y).

**3.2 Сетевое взаимодействие**

* **TCP/UDP клиент** – отправка команд на сервер.
* **Регулярное обновление состояния станка** (получение текущих координат).
* **Обработка ошибок соединения**.

**3.3 Загрузка и сканирование изображений**

1. **Открытие изображения (QFileDialog)**
   * Преобразование в **ч/б** (QImage → QPixmap).
2. **Анализ пикселей**
   * Определение **чёрных точек** (где лазер включается).
   * Построчная обработка – создание списка команд.
3. **Отправка в станок**
   * По каждой активной точке move\_to(x, y), turn\_on(), turn\_off().
   * Оптимизация – **пропуск пустых областей**.

**3.4 Хранение истории движения**

* **Сохранение последних N перемещений**.
* **Кнопка сброса истории**.

**4. Этапы разработки**

**4.1 Неделя 1**

✅ **Сервер**

* Базовая структура.
* Приём команд (TCP/UDP).
* Реализация turn\_on(), turn\_off(), move\_to().

✅ **Клиент (GUI)**

* Прототип интерфейса PyQt.
* Поля ввода, кнопки, тестовые события.

**4.2 Неделя 2**

✅ **Сервер**

* **Алгоритм Брезенхэма** (реальное пошаговое движение).
* Проверка границ, логирование.

✅ **Клиент**

* Отрисовка сетки, положения лазера (QPainter).
* Масштабирование, обработка кликов.
* **Связь с сервером**.

**4.3 Неделя 3**

✅ **Сервер**

* **Логирование, обработка ошибок**.
* **Периодическое обновление координат**.

✅ **Клиент**

* Обработка состояния станка.
* **Загрузка изображений, анализ пикселей**.
* **Генерация команд из изображения**.

**4.4 Неделя 4**

✅ **Финальная отладка и тестирование**

* Проверка всех функций.
* **Стабилизация сетевого взаимодействия**.
* **Документация**.

**5. Дополнительные улучшения**

* **Режим "калибровки"** (обнуление координат).
* **Оптимизация работы с сетью** (асинхронность).
* **Настраиваемый размер поля** (если потребуется).
* **Логирование в файл или БД**.
* **Развитие алгоритма сканирования** (учёт полутонов).

**Итог**

Этот план позволяет разрабатывать проект **поэтапно**, начиная с простого функционала и добавляя расширенные возможности. В нём **чётко разделены сервер и клиент**, что облегчит тестирование и масштабирование. 🚀