Board Drawer
“Really Ghost Board”

Документация

Создатели:
Бабенко Егор
Емельянов Алексей
Иванов Глеб

Этот проект был задуман как бюджетная замена смарт-доскам с функцией рисования. Идея заключается в том, чтобы параллельно с выводом презентации с проектора с помощью камеры определять след от лазерной указки, после чего по этому следу рисовать маску и накладывать ее на входной видеопоток проектора.
Данная версия проекта захватывает видео с камеры или видеофайл, на котором обнаруживает след от лазерной указки и выводит видеопоток с маской в окно интерфейса.

1.Установка и запуск программы

Для пользователя

Для запуска приложения, пользователю нужно скачать файл для своей платформы по ссылке приведенной ниже. Далее запустить скаченный файл на своей платформе и пользоваться приложением, предварительно запустив камеру и проектор к своему устройству.

Ссылка на скачивание - https://github.com/JooudDoo/BoardDrawerRepo/releases

Для разработчика

Для запуска приложения, разработчику нужно скачать проект по ссылке приведенной ниже.

Ссылка на скачивание - https://github.com/JooudDoo/BoardDrawerRepo

Или клонировать проект следующей командой

$ git clone https://github.com/JooudDoo/BoardDrawerRepo

Перед запуском нужно, что бы на устройстве был установлен python3*. Далее откройте проект в IDE и установите зависимости используя следующие команду.

$ pip -r install requirements.txt

После установки зависимостей нужно запустить проект следующий командой.

$ python src/main.py

После данной команды проект будет запущен.

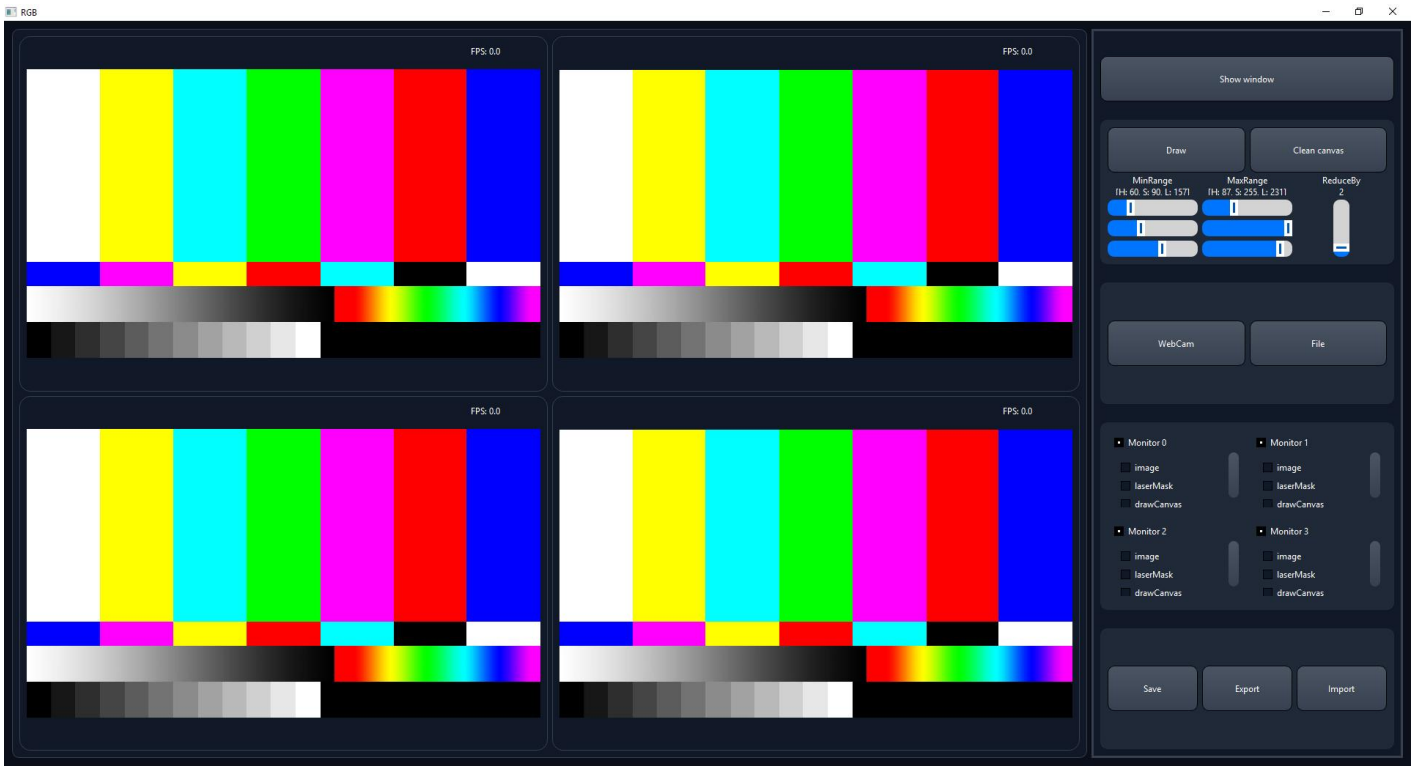
Docker образ

Ссылка на скачивание - https://hub.docker.com/r/jooud/rgb_board

Для самостоятельной сборки Docker образа можно использовать Dockerfile.

$ docker build -t rgb_board .

2.Интерфейс



На изображении представлен Debug-интерфейс для разработчиков. Слева расположены четыре экрана для отображения видеопотоков с различными фильтрами.
Справа находится панель настроек:

- Кнопка Show window - открывает окно с итоговым видеопотоком(с использованием всех фильтров)[1]
- Кнопка Draw/No draw - включает/выключает режим рисования
- Кнопка Clean canvas - очищает слой с рисунком
- Ползунки MinRange и MaxRange для харкатеристик HSL[2] - настраивают верхний и нижний порог обнаружения лазера
- Ползунок ReduceBy - настраивает делитель размеров изображения при детекции лазера
- Кнопка WebCam - устанавливает источником видеопотока нулевую камеру компьютера
- Кнопка File - устанавливает источником видеопотока зацикленный видеофайл[3]

- Окно настроек всех четырех мониторов - включает мониторы и отображение слоев фильтров на них
- Кнопка Save - сохраняет выставленные настройки в стандартный файл настроек(эти настройки будут использоваться при следующем запуске)
- Кнопка Export - сохраняет настройки в указанный JSON файл
- Кнопка Import - загружает настройки в указанный JSON файл

[1]На данный момент не работает
[2]В файле настроек формат может быть изменен на RGB
[3]На данный момент может портить формат видео и увеличивать скорость воспроизведения

3.Структура программы

Файл main.py вызывает функцию runWindow из файла DebugWindow.py

3.1 DebugWindow.py

Функция runWindow создаёт и отображает переменную класса DebugWindow.

3.1.1 Класс DebugWindow

Этот класс при инициализации создаёт переменные классов SettingsManager(указывая ей, что нужно загружать данные о настройках из кэш-файла), VideoHandler, DebugImageProcessor. После этого он создаёт список мониторов, создаёт таймер с помощью метода createTimers(в основном не используется, поскольку в других классах есть свои), создаёт UI с помощью метода setupUI(который использует класс SettingsBar) и устанавливает ассеты методом setupStyles. В конце запускаются созданные таймеры.

3.1.2 Класс SettingsBar

Этот класс создаёт все меню настроек с помощью pyQt, используя переменные классов SettingsManager, VideoHandler, DebugImageProcessor, передаваемые в класс SettingsBar из класса DebugWindow.

3.2 SettingsLoader.py

3.2.1 Класс ColorRangeSettings

Этот класс содержит настройки диапазона детекции лазера и может принимать настройки в формате строки, чисел или JSON(метод loadJSON). Также имеет метод getDict который возвращает класс в формате настройки для класса SettingsManager. Использует классы HSL и RGB из файла ColorContainers.py.

3.2.2 Класс DymmySetting

Стандартный класс настройки формата “настройка: значение”.

3.2.3 Класс SettingsManager

Класс, который хранит в себе все настройки для прграммы(по умолчанию - диапазон обнаружения лазера, формат диапазона RGB/HSL, делитель изображения при детекции) и загружает настройки из стандартного кэш файла. Метод addSetting позволяет добавить дополнительные настройки в список настроек. Методы exportSettingsFromJSON(используется методом updateSettingsInFile для сохранения настроек в кэш-файл) и importSettingsToJSON позволяют сохранять/загружать настройки из JSON файлов.

3.3 VideoHandler.py

3.3.1 Класс VideoHandler

Этот класс служит для захвата видео из потока. Метод change_source позволяет изменить поток захвата, а метод getFrame выдаёт текущее изображение из текущего потока.

3.4 ColorContainers.py

Содержит классы RGB и HSL для хранения информации о значениях этих форматов и загрузки этих значений из разных форматов(str, int и т.д).

3.5 CameraHandler.py

Содержит класс CameraHandler, который по факту является классом VideoHandler.

3.6 ImageProcessor.py

Используется для детекции лазера и рисования.

3.6.1 Класс Layer

Список имён слоёв изображений.

3.6.2 Класс LayerInfo

Хранит информацию о слое(имя, функция для его сборки, слои необходимые для сборки). Метод funcWrapper проверяет список зависимостей на наличие слоёв, необходимых для сборки слоёв из этого списка(многоуровневые зависимости) и добавляет необходимые слои в список.

3.6.3 Класс BasicImageProcessor

Используется для обработки изображений. При инициализации создаёт CameraHandler и список слоёв. Методы addLayerInfo и getLayerInfo используются для добавления и получения информации о слоях. Метод _createLayerDepends создаёт список зависимостей для слоя(многоуровневые зависимости добавляются рекурсивным методом). Метод _getLayers производит сборку зависимостей и возвращает готовый слой, который можно применять к видеопотоку или другому слою. Метод setCameraHandler устанавливает поток для захвата.

3.6.4 Класс DebugImageProcessor

Стандартный обработчик изображений. При инициализации устанавливает настройки детектора(метод checkSettings) и стандартный цвет “карандаша”, его позицию и создаёт слой детекции и слой “холста”(методы setupLayers и createCanvas). Холст можно очистить методом clearCanvas. Метод switchDrawMode включает и выключает режим рисования. Для точности отрисовки используются координаты лазера с прошлого кадра и метод getMoments, который по нынешним координатам лазера и сохраненным рисует линию. Для наложения фильтров используется метод createImageFromLayers, который вызывает метод applyFilter,

который в свою очередь использует метод applyMask и накладывает наложения с определённой прозрачностью.

3.7 Assets_Loader.py

Используется для загрузки ассетов.

3.8 Папка UI

Содержит файлы с pyQt элементами интерфейса.

3.8.1 ImageViewer.py

3.8.1.1 Класс FPSMeter

Используется для подсчёта количества кадров в секунду и его отображения.

3.8.1.2 Класс ImView

Для отображения изображения используется очередь обработанных изображений. Для загрузки в нее изображения используется метод adImageToQueue. Для получения изображения из очереди используется метод updateImageFromQueue, который при отсутствии изображений в очереди выжидает небольшой промежуток времени и выводит стандартный экран.

4. Используемые библиотеки

· altgraph - для построения графов.
· autopep8 - автоматически форматирует код Python в соответствии с руководством по стилю PEP 8.
· numpy - это основной пакет для научных вычислений с помощью Python.
· opencv-python - это пакет отвечает за работу с изображением.
· pefile - Обрабатываемые элементы, такие как таблица импорта, доступны с именами в нижнем регистре, чтобы отличить их от имён базовой структуры в верхнем регистре.
· pycodestyle - это инструмент для проверки кода Python на соответствие некоторым соглашениям о стилях в PEP 8.
· PyQt6 - это полный набор привязок Python для Qt v6. Он реализован в виде более чем 35 модулей расширения и позволяет использовать Python в качестве альтернативного языка разработки приложений для C++ на всех поддерживаемых платформах, включая iOS и Android.
· PyQt6-Qt6 - Этот пакет содержит подмножество установки Qt, которое требуется для PyQt6. Обычно он устанавливается автоматически с помощью pip при установке PyQt6.
· PyQt6-sip - Модуль расширения sip обеспечивает поддержку пакета PyQt6.
· six - Оболочка API Python для e621.net