Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет «Высшая школа экономики».
Институт когнитивных нейронаук

Kinematic 4

Система кинематического анализа движения кисти.

Документация разработчика

2025

Аннотация

Настоящий документ является **документацией разработчика** по эксплуатации система кинематического анализа движения кисти - Kinematic 4 [кинема́тик фор]. В данном руководстве приводится следующая информация:

- Технологии и инструменты
- Краткое описание возможностей
- Архитектура системы
- Исходный код
- Термины и сокращения

Содержание

1 ВВЕДЕНИЕ

Настоящая программа является десктопным приложением и предназначена для выполнения автоматизированного анализа кинематических данных о движении руки человека. Система работает с данными, сбор которых осуществляется в рамках лабораторных экспериментов, в ходе которых испытуемого просят схватить объект, который ставят перед ним, и поставить на заданное место. Движение повторяется 64 раза (меняется сам объект, его ориентация, ориентация пластинки, на которую надо поставить объект). Система кинематического анализа с частотой 250Гц записывает 3D координаты "трекеров", которые крепятся на испытуемом и на объекте. В эксперименте используется 7 трекеров, наклеенных на следующие точки:

1) Ногтевая пластина большого пальца;
2) Ногтевая пластина указательного пальца;
3) Наружная сторона кости предплечья (дистальный отдел, область головки лучевой кости) - вспомогательные данные для анализа целостного движения руки;
4) Внутренняя сторона кости предплечья (дистальный отдел, область шиповидного отростка) - вспомогательные данные для анализа целостного движения руки;
5) Трекер наклеенный на недвижимую часть очков;
6) Трекер наклеенный на движимую часть очков;
7) Объект, который испытуемый должен схватить.

В результате записи имеются 3D координаты этих трекеров, изменяемые во времени на протяжении всего эксперимента (записаны в открытом формате). Основной задачей программы является обработка данных о движении руки и объекта с указанных трекеров, включая выявление ключевых моментов:

- Начало эксперимента (Start);
- Начало движения (поднятие руки) (Hand lifting);
- Начало раскрытия пальцев (GA opening);
- Максимальная апертура раскрытие пальцев (GA max);
- Подъем объекта (Object lifting);
- Опускание объекта (Object placed);

Также программа фиксирует значений следующих показателей в рамках эксперимента для каждого момента времени:

- Открытие очков - расстояние между трекерами 5 и 6;
- Положение руки по оси Z - изменения координаты средней точки между трекерами 3 и 4 по оси Z;

- Апертура захвата - расстояние между трекерами 1 и 2;
- Положение объекта по оси Z - изменения координаты трекера 7 по оси Z;

На основе значений указанных моментов времени система осуществляет расчет следующих метрик, интерпретируемых пользователями:

- Hand lifting - время между началом эксперимента и началом движения (поднятия руки);
- GA opening - время между началом эксперимента и началом раскрытия пальцев;
- Object lifting - время между началом эксперимента и подъемом объекта;
- GA max - время между началом эксперимента и максимальной апертурой раскрытия пальцев;
- Total movement time - время между началом эксперимента и опусканием объекта;
- Reaction time - равняется параметру Hand lifting;
- Time of max GA - значение разности параметров *GA max* - Hand lifting;
- Time to reach - значение разности параметров *GA max* - Object lifting;
- GA% - значение процентного соотношения параметров *Time of max GA*/*Time to reach*;
- Time of object movement - значение разности параметров *Total movement time* - Object lifting.

1.1 Технологии и инструменты

Языки программирования:

- Python 3.9.0

Фреймворки:

- PyQT 6

Библиотеки:

Системные библиотеки Python:

- import **sys** – предоставляет доступ к объектам и функциям, связанным с интерпретатором Python, например, для обработки аргументов командной строки.

Работа с данными:

- import **pandas** as pd – мощная библиотека для работы с табличными данными, их обработки и анализа.

Визуализация:

- import **matplotlib.pyplot** as plt – инструмент для построения графиков и визуализации данных.

- from **matplotlib.backends.backend_qt5agg** import **FigureCanvasQTAgg** as FigureCanvas – используется для встраивания графиков Matplotlib в приложения на базе PyQt.
- from **matplotlib.backends.backend_qt5agg** import **NavigationToolbar2QT** as NavigationToolbar – добавляет навигационную панель для работы с графиками в PyQt-приложении.
- from **matplotlib.figure** import **Figure** – создание объектов графиков для визуализации.

Графический интерфейс (GUI):

- from **PyQt6.QtWidgets** import **QApplication, QInputDialog, QAbstractItemView, QTableView, QMainWindow, QVBoxLayout, QComboBox, QAbstractItemView, QFileDialog, QMessageBox** – модули для создания элементов пользовательского интерфейса, таких как окна, таблицы, выпадающие списки, диалоги и сообщения.
- from **PyQt6.QtCore** import **Qt, QAbstractTableModel, QModelIndex, pyqtSignal** – обеспечивает базовые классы и сигналы для работы с моделью данных, взаимодействием и событиями в интерфейсе.
- from **PyQt6.uic** import **loadUi** – позволяет загружать и использовать пользовательские интерфейсы, созданные в Qt Designer.

Научные вычисления и обработка сигналов:

- import **numpy** as np – библиотека для работы с многомерными массивами данных и выполнения математических операций.
- from **scipy.io** import **loadmat** – модуль для работы с файлами формата .mat, используемого в MATLAB.
- from **scipy.signal** import **find_peaks** – позволяет находить пики в данных (например, в временных рядах или сигналах).
- from **scipy.ndimage** import **gaussian_filter1d** – реализует одномерное гауссово сглаживание данных.

Работа с файлами и Excel:

- from **tkinter** import **filedialog** – предоставляет простые диалоговые окна для выбора и открытия файлов.
- from **openpyxl** import **load_workbook** – модуль для чтения и записи Excel-файлов формата .xlsx.
- from **openpyxl.styles** import **Font, PatternFill, Border, Side** – стилизация ячеек Excel-файлов (шрифты, цвета, границы и т.д.).

- from **openpyxl.utils.dataframe** import **dataframe_to_rows** – преобразование данных из DataFrame (Pandas) в строки Excel.

1.2 Краткое описание возможностей

Главная страница и элементы интерфейса

При запуске программы открывается главная страница с интуитивно понятным интерфейсом. Основные элементы:

1) Шапка программы – содержит кнопки, предоставляющие доступ к следующему функционалу:

1) "Импорт" – для загрузки данных в файле *mat* формата;
2) "Экспорт" – для сохранения обработанных данных в формате *xlsx* (Excel);
3) "Настройки" – для конфигурации параметров расчетных алгоритмов.

2) Просмотр параметров в табличном виде – отображает рассчитанные параметры загруженных данных в табличном формате;

3) Визуализация данных – программа предоставляет возможность построения графиков по выбранным данным;

4) Фильтры визуализации – предоставляет чекбоксы для выбора параметров, которые должны быть отображены на графике.

Функции управления графиком (панель инструментов)

На графике предусмотрены следующие инструменты:

1) Reset to original view – возврат к исходному виду графика;

2) Back to previous view и **Forward to next view** – перемещение между сохраненными состояниями отображения;

3) Move tool – интерактивное перемещение графика;

4) Zoom to rectangle – увеличение выделенной области графика;

5) Configure subplots – настройка расположения отдельных графиков и их элементов;

6) Edit axis, curve and image parameters – редактирование параметров осей, кривых и изображения;

7) Save the figure – сохранение графика в файл.

2 АРХИТЕКТУРА СИСТЕМЫ

Классы и их назначение

PandasModel – Этот класс реализует интерфейс для отображения данных в таблице на основе Pandas DataFrame.

- Основные методы включают подсчёт строк/столбцов, получение данных для отображения и настройку заголовков столбцов.

MainWindow – Главный класс приложения, который реализует пользовательский интерфейс и логику обработки данных.

- Содержит методы для настройки приложения, работы с данными, расчёта параметров, визуализации и экспорта.

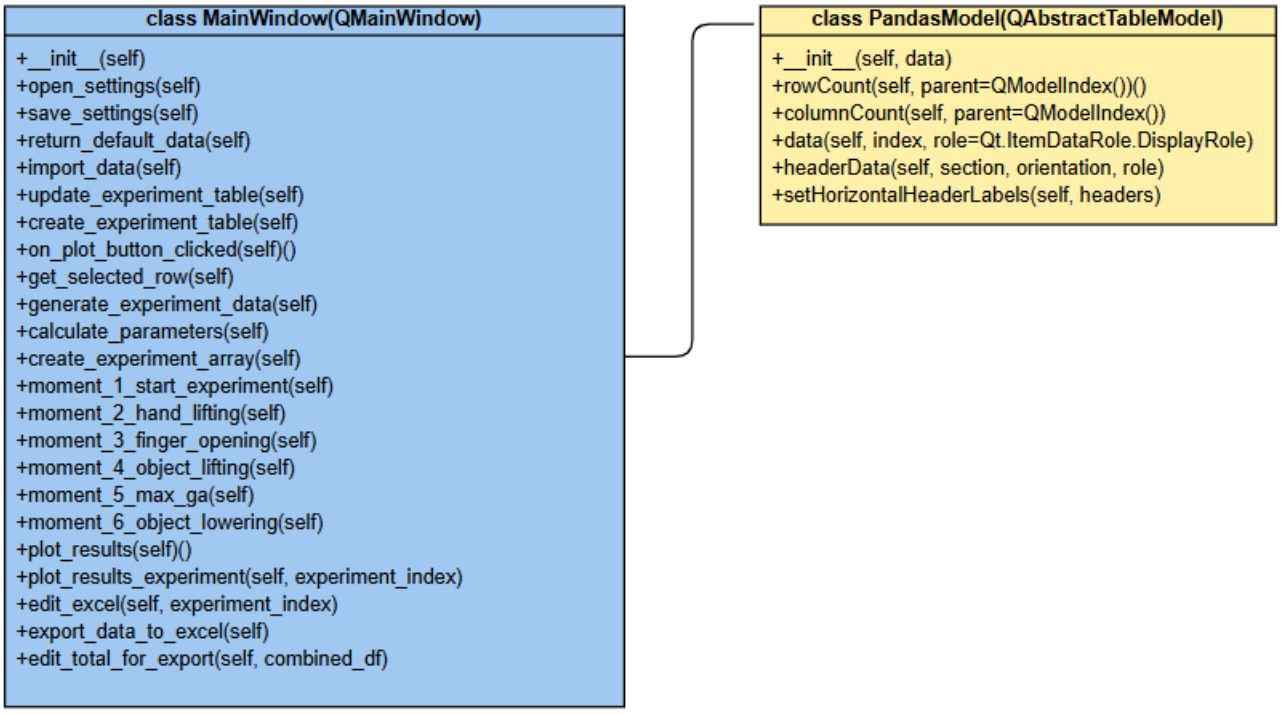

Рисунок 2.1 – UML схема классов программы

Список функций

Описание логики работы функций представлено с формате комментариев к исходному коду в разделе 3 (ИСХОДНЫЙ КОД).

1.**Функции для работы с настройками:**

- open_settings: открывает окно с настройками программы.
- save_settings: сохраняет текущие настройки пользователя.
- return_default_data: возвращает настройки и данные по умолчанию.

2.**Функции для работы с данными:**

- import_data: импортирует данные из файлов .mat или других поддерживаемых форматов.
- update_experiment_table: обновляет таблицу экспериментов на основе загруженных данных.
- create_experiment_table: создаёт таблицу для отображения данных экспериментов.

3.**Функции для расчёта параметров:**

- generate_experiment_data: формирует массив экспериментов, хранящий ключевые события по каждому эксперименту.
- create_experiment_array: подготовливает массив отдельных экспериментов для визуального отображения графиков (разделение на эксперименты).
- calculate_parameters: выполняет расчёты ключевых параметров для каждого эксперимента.

4.**Функции алгоритмов для определения моментов:**

- moment_1_start_experiment: определяет момент начала эксперимента.
- moment_2_hand_lifting: определяет момент подъёма руки.
- moment_3_finger_opening: определяет момент открытия пальцев.
- moment_4_object_lifting: определяет момент подъёма объекта.
- moment_5_max_ga: определяет момент достижения максимального значения апертуры захвата.
- moment_6_object_lowering: определяет момент опускания объекта.

5.**Функции для визуализации данных:**

- plot_results: строит общий график для всех экспериментов.
- plot_results_experiment: строит график для выбранного эксперимента.

6.**Функции для экспорта данных:**

- edit_excel: редактирует экспортируемый файл Excel, добавляя рассчитанные параметры.
- export_data_to_excel: выполняет экспорт данных в Excel.
- edit_total_for_export: объединяет данные перед экспортом.

3 ИСХОДНЫЙ КОД

3.1 Функции для работы с настройками

- open_settings: открывает окно с настройками программы.

```
def open_settings(self):
    self.settings_window = QMainWindow()
    # loadUi('settings.ui', self.settings_window)

    ui_file = self.resource_path("settings.ui")
    loadUi(ui_file, self.settings_window)

    # Установка текущих значений в настройки
    self.settings_window.lineEdit.setText(str(self.tl))  # Расстояние между пиками
    self.settings_window.lineEdit_2.setText(str(self.hand_lifting))  # Порог скорости
руки
    self.settings_window.lineEdit_3.setText(str(self.finger_opening))  # Порог
открытия пальца
    self.settings_window.lineEdit_5.setText(str(self.object_lifting))  # Порог
подъема объекта
    self.settings_window.lineEdit_4.setText(str(self.object_lowering))  # Порог
опускания объекта
    self.settings_window.lineEdit_6.setText(str(self.max_FG))  # Максимум FG
    self.settings_window.lineEdit_7.setText(str(self.min_FG))  # Минимум FG

    self.settings_window.show()
    self.settings_window.pushButton_4.clicked.connect(self.save_settings)
    self.settings_window.pushButton_5.clicked.connect(self.return_default_data)
```

- save_settings: сохраняет текущие настройки пользователя.

```
def save_settings(self):
    # Сохранение настроек при закрытии окна
    try:
        self.tl = float(self.settings_window.lineEdit.text().replace(',', '.'))  #
расстояние между пиками
        self.hand_lifting = float(self.settings_window.lineEdit_2.text().replace(',',
'.'))  # Порог скорости руки
        self.finger_opening =
float(self.settings_window.lineEdit_3.text().replace(',', '.'))  # Порог открытия
пальца
        self.object_lifting =
float(self.settings_window.lineEdit_5.text().replace(',', '.'))  # Порог подъема
объекта
        self.object_lowering =
float(self.settings_window.lineEdit_4.text().replace(',', '.'))  # Порог опускания
объекта
        self.max_FG = float(self.settings_window.lineEdit_6.text().replace(',', '.'))
# Максимум FG
        self.min_FG = float(self.settings_window.lineEdit_7.text().replace(',', '.'))
# Минимум FG

        self.update_experiment_table()
    except ValueError:
        QMessageBox.warning(self, "Ошибка", "Введены некорректные значения в
настройках")
```

- return_default_data: возвращает настройки и данные по умолчанию.

```
def return_default_data(self):

    # Возвращаем значения по умолчанию
    self.settings window.lineEdit.setText(str(self.default values['tl']))  #
Расстояние между пиками
    self.settings_window.lineEdit_2.setText(str(self.default_values['hand_lifting']))
# Порог скорости руки

self.settings_window.lineEdit_3.setText(str(self.default_values['finger_opening']))
# Порог открытия пальца

self.settings_window.lineEdit_5.setText(str(self.default_values['object_lifting']))
# Порог подъема объекта

self.settings_window.lineEdit_4.setText(str(self.default_values['object_lowering']))
# Порог опускания объекта
    self.settings_window.lineEdit_6.setText(str(self.default_values['max_FG']))  #
Максимум FG
    self.settings_window.lineEdit_7.setText(str(self.default_values['min_FG']))  #
Минимум FG

    # Обновляем текущие значения
    self.tl = self.default_values['tl']
    self.hand_lifting = self.default_values['hand_lifting']
    self.finger_opening = self.default_values['finger_opening']
    self.object_lifting = self.default_values['object_lifting']
    self.object_lowering = self.default_values['object_lowering']
    self.max_FG = self.default_values['max_FG']
    self.min_FG = self.default_values['min_FG']
```

3.2 Функции для работы с данными

- import_data: импортирует данные из файлов .mat или других поддерживаемых форматов.

```
def import_data(self):
    self.file_path, _ = QFileDialog.getOpenFileName(self, "MAT files", "*.mat")

    if not self.file_path:
        QMessageBox.warning(self, "Предупреждение", "Файл не выбран.")
        return -1
    try:
        self.data = loadmat(self.file_path)
        self.file_loaded = True

        if not self.data:
            QMessageBox.warning(self, "Ошибка", "Некорректный файл (проверьте
содержимой файла)")
    except FileNotFoundError:
        QMessageBox.warning(self, "Ошибка", "Файл не найден. Пожалуйста, проверьту
путь к файлу.")
        return -1
    except Exception as e:
        QMessageBox.warning(self, "Ошибка", f"Произошла ошибка при загрузке файла:
{e}")
        return - 1


    # Сохраняем путь к файлу
    self.current_file_path = self.file_path
```

```
    self.calculate_parameters()
    self.moment_1_start_experiment()
    self.moment_2_hand_lifting()
    self.moment_3_finger_opening()
    self.moment_4_object_lifting()
    self.moment_5_max_ga()
    self.moment_6_object_lowering()
    self.create_experiment_array()

    # Создание таблицы с данными экспериментов
    self.experiment_table = None  # Инициализация таблицы
    self.create_experiment_table()  # Создаем таблицу после загрузки данных
```

- update_experiment_table: обновляет таблицу экспериментов на основе загруженных данных.

```
def update_experiment_table(self):
    """Обновляет данные в отчете, используя путь к загруженному файлу."""
    try:
        # Загружаем данные из сохраненного пути
        self.data = loadmat(self.current_file_path)
        self.calculate_parameters()
        self.moment_1_start_experiment()
        self.moment_2_hand_lifting()
        self.moment_3_finger_opening()
        self.moment_4_object_lifting()
        self.moment_5_max_ga()
        self.moment_6_object_lowering()
        self.create_experiment_array()
        # Создание таблицы с данными экспериментов
        self.experiment_table = None  # Инициализация таблицы
        self.create_experiment_table()  # Создаем таблицу после загрузки данных
        if not hasattr(self, 'experiment_data') or not self.experiment_data:
            self.figure.clear()
            self.canvas.draw()
            QMessageBox.warning(self, "Ошибка", "Отсутствуют данные, соответствующие
указанным настройкам (измените настройки параметров на настройки по умолчанию)")

        else:
            QMessageBox.information(self, "Уведомление", "Данные успешно обновлены!",
QMessageBox.StandardButton.Ok)
    except Exception as e:
        QMessageBox.warning(self, "Ошибка", f"Произошла ошибка при обновлении данных:
{e}")
```

- create_experiment_table: создаёт таблицу для отображения данных экспериментов.

```
def create_experiment_table(self):
    """Создает таблицу экспериментов в интерфейсе."""
    self.generate_experiment_data()  # Генерируем данные для таблицы

    # Создаем Dataframe
    df = pd.DataFrame(self.experiment_data)

    # Натсраиваем модель для табличного воспроизведения
    self.model = PandasModel(df)
    self.tableView.setModel(self.model)

    # заголовки таблицы
    self.model.setHorizontalHeaderLabels(["№", "Start Experiment (s)", "Hand Lifting
```

```
(s)", "GA Opening", "Object Lifting (s)", "GA max", "Object placed"])

    #Установка стиля для заголовка
    header = self.tableView.horizontalHeader()
    header.setStyleSheet("QHeaderView::section { background-color: lightgrey; border-
bottom: 1px solid black; }")
```

3.3 Функции для расчета параметров

- generate_experiment_data: формирует массив экспериментов, хранящий ключевые события по каждому эксперименту.

```
def generate_experiment_data(self):
    """Формирует данные для каждого эксперимента."""
    # Инициализация списка для хранения данных по каждому эксперименту
    self.experiment_data = []

    # Проходим по массиву экспериментов (начальная и конечная точки каждого)
    for idx, (start_point, end_point) in enumerate(self.experiment_array):
        # Находим время начала эксперимента (первое значение 1 в столбце 0 между
start_point и end_point)
        start_experiment_time = int(np.where(self.TF3[start_point:end_point, 0] ==
1)[0][0] + start_point) / 250

        # Находим время поднятия руки (первое значение 1 в столбце 2 между
start_point и end_point)
        hand_lifting_time = int(np.where(self.TF3[start_point:end_point, 2] ==
1)[0][0] + start_point) / 250

        # Находим время поднятия объекта (первое значение 1 в столбце 4 между
start_point и end_point)
        object_lifting_indices = np.where(self.TF3[start_point:end_point, 4] == 1)[0]
        if object_lifting_indices.size > 0:
            # Если значения найдены, берем первое и преобразуем в секунды
            object_lifting_time = int(object_lifting_indices[0] + start_point) / 250
        else:
            # Если значения отсутствуют, записываем NaN
            object_lifting_time = np.nan

        # Находим момент открытия пальцев (GA Opening, первое значение 1 в столбце 3
между start_point и end_point)
        ga_opening = int(np.where(self.TF3[start_point:end_point, 3] == 1)[0][0] +
start_point) / 250

        # Находим момент максимального открытия пальцев (GA Max, аналогично GA
Opening)
        ga_max = int(np.where(self.TF3[start_point:end_point, 3] == 1)[0][0] +
start_point) / 250

        # Находим момент опускания объекта (Object Placed, первое значение 1 в
столбце 6 между start_point и end_point)
        object_placed_indices = np.where(self.TF3[start_point:end_point, 6] == 1)[0]
        if object_placed_indices.size > 0:
            # Если значения найдены, берем первое и преобразуем в секунды
            object_placed = int(object_placed_indices[0] + start_point) / 250
        else:
            # Если значения отсутствуют, записываем NaN
            object_placed = np.nan

        # Формируем запись для текущего эксперимента с указанием всех значимых
моментов
```

```
        self.experiment_data.append({
            'Experiment No': idx + 1,  # Номер эксперимента (индекс + 1)
            'Start Experiment': start_experiment_time,  # Время начала эксперимента
            'Hand Lifted Time': hand_lifting_time,  # Время поднятия руки
            'QA Opening': ga_opening,  # Время открытия пальцев (GA Opening)
            'Object Lifted Time': object_lifting_time,  # Время поднятия объекта
            'GA Max': ga_max,  # Момент максимального открытия пальцев
            'Object Placed': object_placed  # Время опускания объекта
        })
```

- create_experiment_array: подготовливает массив отдельных экспериментов для визуального отображения графиков (разделение на эксперименты).

```
def create_experiment_array(self):
    self.experiment_array = []

    # Находим индексы всех строк, где в первом столбце массива TF3 стоит 1 (начало эксперимента)
    start_indices = np.where(self.TF3[:, 0] == 1)[0]
    # Проходим по каждому индексу начала эксперимента
    for start_point in start_indices:
        adjusted_start = max(0, start_point - 200)  # Уменьшаем начало на 200 кадров, не выходя за пределы массива
        end_point = start_point + 4000  # Конечная точка эксперимента (начало + 4000 кадров)
        # Добавляем кортеж (начало, конец) в массив экспериментов
        self.experiment_array.append((adjusted_start, end_point))
```

- calculate_parameters: выполняет расчёты ключевых параметров для каждого эксперимента.

```
def calculate_parameters(self):

    frame = self.data['frame']
    glasses = self.data['glasses']
    index = self.data['index']
    thumb = self.data['thumb']
    inside = self.data['inside']
    outside = self.data['outside']
    object_ = self.data['object']

    nrows = len(frame)
    FG = np.zeros((nrows, 3))
    GA = np.zeros((nrows, 3))
    Wrist = np.zeros((nrows, 3))

    for t in range(nrows):
        FG[t, 0] = np.sqrt((frame[t, 2] - glasses[t, 2]) ** 2 +
                           (frame[t, 3] - glasses[t, 3]) ** 2 +
                           (frame[t, 4] - glasses[t, 4]) ** 2)
        GA[t, 0] = np.sqrt((index[t, 2] - thumb[t, 2]) ** 2 +
                           (index[t, 3] - thumb[t, 3]) ** 2 +
                           (index[t, 4] - thumb[t, 4]) ** 2)

    Wrist[:, 0] = (inside[:, 2] + outside[:, 2]) / 2
    Wrist[:, 1] = (inside[:, 3] + outside[:, 3]) / 2
    Wrist[:, 2] = (inside[:, 4] + outside[:, 4]) / 2

    self.FG = np.zeros((nrows, 3))
    self.GA = np.zeros((nrows, 3))
```

```
    self.Wrist = np.zeros((nrows, 2))

    self.FG[:, 0] = FG[:, 0]
    self.FG[:, 1] = gaussian_filter1d(FG[:, 0], sigma=10)
    self.GA[:, 0] = GA[:, 0]
    self.GA[:, 1] = gaussian_filter1d(GA[:, 0], sigma=10)
    self.Wrist[:, 0] = gaussian_filter1d(Wrist[:, 1], sigma=10)
    self.Inside_Y_Smooth = gaussian_filter1d(inside[:, 3], sigma=10)
    self.Obj_smooth = gaussian_filter1d(object_[:, 3], sigma=30)

    self.FG_speed = np.diff(self.FG[:, 1]) * 100
    self.FG_speed = np.append(self.FG_speed, 0)
    self.Wr_speed = np.diff(self.Wrist[:, 0]) * 100
    self.Wr_speed = np.append(self.Wr_speed, 0)
    self.Obj_speed = np.diff(self.Obj_smooth) * 100
    self.Obj_speed = np.append(self.Obj_speed, 0)

    self.FG_speed = gaussian_filter1d(self.FG_speed, sigma=10)
    self.Obj_speed = gaussian_filter1d(self.Obj_speed, sigma=10)

    self.frame = frame
    self.TF3 = np.zeros((nrows, 8))
```

3.4 Функции для алгоритмов для определения моментов

- moment_1_start_experiment: определяет момент начала эксперимента.

```
def moment_1_start_experiment(self):
    # Устанавливаем пороговое значение расстояния между пиками
    tl = self.tl

    # Задаем ограничения для значений FG (Frame-Glasses)
    setting_max_FG = self.max_FG  # Максимальное значение FG
    setting_min_FG = self.min_FG  # Минимальное значение FG

    # Предварительно находим пики в данных скорости FG
    # prominence=0.02 задает минимальную высоту "холмов", distance=tl — минимальное
расстояние между пиками
    all_peaks, _ = find_peaks(self.FG_speed, prominence=0.02, distance=tl)

    # Инициализируем список для хранения валидных пиков
    valid_peaks = []

    # Проходим по всем найденным пикам
    for peak in all_peaks:
        fg_value = self.FG[peak, 1]  # Извлекаем значение FG для текущего пика
        # Проверяем, лежит ли значение FG в заданных пределах
        if setting_min_FG <= fg_value <= setting_max_FG:
            valid_peaks.append(peak)  # Если пик валиден, добавляем его в список
            print(f"Пик принят, индекс: {peak} FG = {fg_value}")
        else:
            print(f"Пик отклонен, индекс: {peak} FG = {fg_value}")

    # Проверяем, найден ли хотя бы один валидный пик
    if not valid_peaks:
        print("Не найдено ни одного подходящего пика.")  # Сообщаем об отсутствии
валидных пиков
        return  # Завершаем выполнение функции

    # Обрабатываем каждый валидный пик
    for peak in valid_peaks:
        temp2 = peak  # Устанавливаем начальный индекс для поиска
```

```
        # Поиск первого нулевого значения скорости после текущего пика
        while temp2 < len(self.FG_speed) and self.FG_speed[temp2] > 0:
            temp2 += 1

        # Если найдено валидное значение, обновляем массив TF3
        if temp2 < len(self.FG_speed):  # Проверяем, чтобы индекс не вышел за пределы
массива
            self.TF3[temp2, 0] = 1  # Помечаем точку как момент начала эксперимента
            # print(f"Точка начала эксперимента найдена на индексе: {temp2}")
```

- moment_2_hand_lifting: определяет момент подъёма руки.

```
def moment_2_hand_lifting(self):
    tl = self.hand_lifting  # Пороговое значение скорости руки, определяющее
минимальный средний уровень скорости для фиксации подъема руки.
    # В массиве TF3 ищутся индексы всех точек, где значение в первом столбце равно 1
(начало эксперимента).
    k2 = np.where(self.TF3[:, 0] == 1)[0]

    # Проходим по каждому индексу начала эксперимента.
    for temp in k2:
        temp2 = temp  # Локальная переменная для отслеживания текущей позиции во
времени.
        # Бесконечный цикл для поиска момента начала подъема руки.
        while True:
            temp2 += 1  # Двигаемся вперед по временной оси.

            # Рассчитываем среднюю скорость движения руки на отрезке из 20 кадров.
            Wr_speed_next = np.mean(self.Wr_speed[temp2:temp2 + 20])

            # Проверяем два условия:
            # 1. Текущая скорость руки положительна (рука движется вверх).
            # 2. Средняя скорость на отрезке превышает заданное пороговое значение.
            if self.Wr_speed[temp2] > 0 and Wr_speed_next > tl:
                break  # Если оба условия выполнены, фиксируем момент подъема руки.

        # Устанавливаем значение 1 в третьем столбце TF3, фиксируя момент подъема
руки.
        self.TF3[temp2, 2] = 1
```

- moment_3_finger_opening: определяет момент открытия пальцев.

```
def moment_3_finger_opening(self):
    # Устанавливаем пороговое значение для открытия пальца
    tl = self.finger_opening
    # Задаем диапазон для расчета среднего значения апертуры
    range_ = 100
    # Находим индексы точек начала эксперимента (где TF3[:, 0] == 1)
    k2 = np.where(self.TF3[:, 0] == 1)[0]

    # Проходим по каждой точке начала эксперимента
    for temp in k2:
        temp2 = temp  # Устанавливаем начальный индекс для поиска
        # Рассчитываем среднее значение апертуры до текущей точки
        GA_Av = np.mean(self.GA[temp2 - range :temp2, 1])
        # Ищем первую точку, где значение апертуры превышает заданный порог
        while self.GA[temp2, 1] <= GA_Av * tl:
            temp2 += 1
        # Помечаем момент открытия пальцев в массиве TF3
        self.TF3[temp2, 3] = 1
```

- moment_4_object_lifting: определяет момент подъёма объекта.

```
def moment_4_object_lifting(self):
    # Устанавливаем пороговое значение скорости для подъема объекта
    tl = self.object_lifting
    # Находим индексы точек начала эксперимента (где TF3[:, 0] == 1)
    k2 = np.where(self.TF3[:, 0] == 1)[0]

    # Проходим по каждой точке начала эксперимента
    for temp in k2:
        temp2 = temp  # Устанавливаем начальный индекс для поиска
        while True:
            temp2 += 1  # Переходим к следующему индексу
            # Проверяем, чтобы temp2 не выходил за пределы массива
            if temp2 >= len(self.Obj_speed) - 20:  # Убедитесь, что есть достаточно
элементов для среднего
                print("Индекс temp2 выходит за пределы массива")
                return  # Прерываем выполнение функции

            # Рассчитываем среднюю скорость объекта на следующих 20 кадрах
            Obj_speed_next = np.mean(self.Obj_speed[temp2:temp2 + 20])
            # Проверяем условия скорости для подтверждения подъема объекта
            if self.Obj_speed[temp2] > 0 and Obj_speed_next > tl:
                break  # Если условия выполнены, выходим из цикла

        # Помечаем момент подъема объекта в массиве TF3
        self.TF3[temp2, 4] = 1
```

- moment_5_max_ga: определяет момент достижения максимального значения апертуры захвата.

```
def moment_5_max_ga(self):
    # Поиск всех пиков апертуры захвата (GA)
    # prominence=0.002 — минимальная высота "холмов", distance=100 — минимальное
расстояние между пиками
    GAmax, _ = find_peaks(self.GA[:, 1], prominence=0.002, distance=100)

    # Находим индексы момента подъема руки (TF3[:, 2] == 1) и подъема объекта (TF3[:,
4] == 1)
    k = np.where(self.TF3[:, 2] == 1)[0]
    k3 = np.where(self.TF3[:, 4] == 1)[0]

    # Проходим по индексам начала подъема руки
    for i, temp in enumerate(k):
        # Проверяем, что существует соответствующий момент подъема объекта
        lift_idx = k3[i] if i < len(k3) else None

        # Получаем индексы моментов открытия пальца (TF3[:, 3] == 1)
        open_finger_idx = np.where(self.TF3[:, 3] == 1)[0]

        # Инициализируем список для пиков в пределах 800 кадров от открытия пальца
        peaks_in_range = []

        for x in GAmax:
            if x >= temp:  # Убедиться, что пик находится после начала движения руки
                # Проверяем, что пик находится перед моментом подъема объекта
                if lift_idx is not None and x < lift_idx:
                    # Проверяем, что хотя бы одна точка открытия пальца находится в
пределах 800 кадров
                    if any(abs(x - open_finger) <= 800 for open_finger in
open_finger_idx):
```

```
                        peaks_in_range.append(x)

        # Если найдены пики, выбираем самый высокий
        if peaks_in_range:
            # Находим индекс максимального пика по значению в GA
            max_peak = max(peaks_in_range, key=lambda x: self.GA[x, 1])
            self.TF3[max_peak, 5] = 1  # Помечаем максимум апертуры в массиве TF3
```

- moment_6_object_lowering: определяет момент опускания объекта.

```
def moment_6_object_lowering(self):
    # Устанавливаем пороговое значение скорости опускания предмета
    tl = self.object_lowering
    # Находим индексы момента подъема объекта (где TF3[:, 4] == 1)
    k2 = np.where(self.TF3[:, 4] == 1)[0]

    # Проходим по каждой точке подъема объекта
    for temp in k2:
        temp2 = temp  # Устанавливаем начальный индекс для поиска
        while True:
            temp2 += 1  # Переходим к следующему кадру
            # Рассчитываем среднюю скорость объекта на следующих 75 кадрах
            Obj_speed_next = np.mean(np.abs(self.Obj_speed[temp2:temp2 + 75]))
            # Проверяем условия скорости для подтверждения опускания объекта
            if np.abs(self.Obj_speed[temp2]) <= tl / 5 and Obj_speed_next <= tl:
                break  # Если условия выполнены, выходим из цикла
        # Помечаем момент опускания объекта в массиве TF3
        self.TF3[temp2, 6] = 1
```

3.5 Функции для для визуализации данных

- plot_results: строит общий график для всех экспериментов.

```
def plot_results(self): # Визуализация общего графика
        t = self.frame[:, 1]
        A = self.FG[:, 1]
        B = self.GA[:, 1]
        W = self.Wrist[:, 0]
        Ob = self.Obj_smooth

        Z = self.TF3[:, 0].astype(bool)
        Z2 = self.TF3[:, 2].astype(bool)
        Z3 = self.TF3[:, 3].astype(bool)
        Z4 = self.TF3[:, 4].astype(bool)
        Z5 = self.TF3[:, 5].astype(bool)
        Z6 = self.TF3[:, 6].astype(bool)

        plt.plot(t, A, label='FG')
        plt.plot(t[Z], A[Z], 'b*', label='Start Experiment')
        plt.plot(t, W, label='Wrist_Y')
        plt.plot(t[Z2], W[Z2], 'k*', label='Hand Lifting')
        plt.plot(t, B, label='GA')
        plt.plot(t[Z5], B[Z5], 'm*', label='Max GA')
        plt.plot(t, Ob, label='Object Y')
        plt.plot(t[Z4], Ob[Z4], 'r*', label='Object Lifting')
        plt.plot(t[Z6], Ob[Z6], 'g*', label='Object Lowering')
        plt.plot(t[Z3], B[Z3], 'y*', label='Finger Opening')

        plt.xlabel('Time (s)')
        plt.ylabel('Distance (cm)')
        plt.legend()
        plt.show()
```

- plot_results_experiment: строит график для выбранного эксперимента.

```
def plot_results_experiment(self, experiment_index):
    '''Строим график'''
    try:
        # Получаем начальную и конечную точки эксперимента
        start_point, end_point = self.experiment_array[experiment_index]

        # Проверяем, что end_point не выходит за пределы данных
        if end_point > len(self.frame):
            end_point = len(self.frame)

        # Извлекаем данные для данного эксперимента
        t = self.frame[start_point:end_point, 1]  # Время
        A = self.FG[start_point:end_point, 1]  # FG
        B = self.GA[start_point:end_point, 1]  # GA
        W = self.Wrist[start_point:end_point, 0]  # Wrist_Y
        Ob = self.Obj_smooth[start_point:end_point]  # Object Y

        # Извлечение соответствующих точек событий для эксперимента
        Z = self.TF3[start_point:end_point, 0].astype(bool)  # Start Experiment
        Z2 = self.TF3[start_point:end_point, 2].astype(bool)  # Hand Lifting
        Z3 = self.TF3[start_point:end_point, 3].astype(bool)  # Finger Opening
        Z4 = self.TF3[start_point:end_point, 4].astype(bool)  # Object Lifting
        Z5 = self.TF3[start_point:end_point, 5].astype(bool)  # Max GA
        Z6 = self.TF3[start_point:end_point, 6].astype(bool)  # Object Lowering

        # Очищаем фигуру перед построением нового графика
        self.figure.clear()
        ax = self.figure.add_subplot(111)

        # Построение графиков для данного эксперимента
        if self.checkBox_9.isChecked():
            ax.plot(t, A, label='Открытие очков') # FG
        if self.checkBox.isChecked(): # Start Experiment
            ax.plot(t[Z], A[Z], 'b*', label='Начало эксперимента')
        if self.checkBox_4.isChecked(): # Deviation
            ax.plot(t, W, label='Положение кисти (Z)') # Wrist_Y
        if self.checkBox_2.isChecked(): # Hand Lifting
            ax.plot(t[Z2], W[Z2], 'k*', label='Подъем кисти')
        if self.checkBox_10.isChecked():
            ax.plot(t, B, label='Апертура захвата') # GA
        if self.checkBox_3.isChecked(): # Max Aperture (Max GA)
            ax.plot(t[Z5], B[Z5], 'm*', label='Апертура захвата (Max)')
        if self.checkBox_8.isChecked(): # Trajectory
            ax.plot(t, Ob, label='Положение объекта (Z)')
        if self.checkBox_5.isChecked(): # Object Lifting
            ax.plot(t[Z4], Ob[Z4], 'r*', label='Подъем объекта')
        if self.checkBox_6.isChecked(): # Object Lowering
            ax.plot(t[Z6], Ob[Z6], 'g*', label='Опускание объекта')
        if self.checkBox_7.isChecked(): # Finger Opening
            ax.plot(t[Z3], B[Z3], 'y*', label='Размыкание пальцев')
        ax.set_xlabel('Time (s)')
        ax.set_ylabel('Distance (cm)')
        ax.legend()
        ax.set_title(f'Experiment {experiment_index + 1}')

        # Обновляем график
        self.canvas.draw()
    except:
        QMessageBox.warning(self, "Ошибка", "Отсутствуют данные для обновления
```

```
графика")
        return
```

3.6 Функции для экспорта данных

- edit_excel: редактирует экспортируемый файл Excel, добавляя рассчитанные параметры.

```
def edit_excel(self, experiment_index):
    '''Редактирование Excel файла'''
    # Загрузка данных из Excel
    excel_df = pd.read_excel(self.file_path2)

    # Копия DataFrame
    modified_df = excel_df.copy()

    # Переименовывем столбцы
    modified_df = modified_df.rename(columns={'Unnamed: 0': 'Subject',
'Figure':'Object', 'Figure \nOrientation':'Object orientation', 'Plate
orientation':'Plate orientation'})

    # Удаляем ненужные столбцы
    modified_df = modified_df.drop(columns=['Unnamed: 4', 'Unnamed: 5', 'Unnamed:
6'])

    # Добавление новых столбец
    modified_df['Object Degree'] = (modified_df['Object orientation']-1)*90
    modified_df['Plate Degree'] = (modified_df['Plate orientation'] - 1)*90
    modified_df['Rotation'] = modified_df['Object Degree'] - modified_df['Plate
Degree']

    # Формирование столбца Group
    def calculate_group(rotation):
        if rotation == 0:
            return "N1"
        elif rotation == -90:
            return "N4"
        elif rotation == 90:
            return "N2"
        elif rotation == 180:
            return "N3"
        elif rotation == -180:
            return "N3"
        elif rotation == 270:
            return "N4"
        elif rotation == -270:
            return "N2"

    modified_df['Group'] = modified_df['Rotation'].apply(calculate_group)

    modified_df['Subject'] = experiment_index # Изменяем индексы на номер
эксперимента

    print(modified_df.columns)
    return modified_df
```

- export_data_to_excel: выполняет экспорт данных в Excel.

```
def export_data_to_excel(self):
    """Экспортирует данные эксперимента в Excel с выбором пути сохранения."""
    if self.file_loaded == False:
        QMessageBox.warning(self, "Ошибка", "Отсутсвует файл с данными эксперимента")
        return

    experiment_index, ok = QInputDialog.getText(self, "Введите номер эксперимента",
"1. Неоходимо ввести номер эксперимента \n2. Далее выбрать файл шаблона с информацией
по эксперименту и нажать 'Открыть' в диалоговом окне проводника, \na затем в другом
окне выбрать куда сохранить файл. \nНомер эсперимента: ")
    if not ok or not experiment_index:
        QMessageBox.warning(self, "Ошибка", "Номер эксперимента не был введен")
        return

    self.file_path2, _ = QFileDialog.getOpenFileName(self, "Exsel files", "*.xlsx")
    if not self.file_path2:
        QMessageBox.warning(self, "Ошибка", "Не выбран файл с информацией об
эксперименте")
        return

    # Открытие диалога для выбора места сохранения файла
    file_path = filedialog.asksaveasfilename(defaultextension=".xlsx",
                                             filetypes=[("Excel files", "*.xlsx"),
                                                        ("All files", "*.*")])
    if not file_path:
        return  # Если путь не выбран, выходим из функции

    try:
        # Создание DataFrame из данных self.experiment_data
        mat_df = pd.DataFrame(self.experiment_data,
                              columns=['Experiment No', 'Start Experiment', 'Hand
Lifted Time', 'QA Opening', 'Object Lifted Time', 'GA Max', 'Object Placed'])

        mat_df = mat_df.rename(columns={'Start Experiment': 'Start', 'Hand Lifted
Time':'Hand lifting', 'Object Lifted Time':'Object lifting', 'QA Opening':'GA
opening'})
        # Загрузка данных из редактированного Excel файла
        modified_df = self.edit_excel(experiment_index)

        # Объединение mat и Excel данных
        combined_df = pd.concat([modified_df, mat_df.iloc[:, 1:]], axis=1)

        total_df = self.edit_total_for_export(combined_df)

        # Запись DataFrame в файл Excel
        with pd.ExcelWriter(file_path, engine='openpyxl') as writer:
            total_df.to_excel(writer, index=False)

        workbook = load_workbook(file_path)
        worksheet = workbook.active

        header_font = Font(bold=True)
        header_fill = PatternFill(start_color='C0C0C0', end_color='C0C0C0',
fill_type='solid')
        thin_border = Border(left=Side(style='thin'), right=Side(style='thin'),
top=Side(style='thin'), bottom=Side(style='thin'))

        # Проходимся по первой строке (заголовкам)
        for cell in worksheet[1]:
            cell.font = header_font
            cell.fill = header_fill
            cell.border = thin_border
```

```
        # Увеличиваем высоту первой строки
        worksheet.row_dimensions[1].height = 30

        # Добавляем тонкие границы для всех ячеек
        for row in worksheet.iter_rows():
            for cell in row:
                cell.border = thin_border

        # Добавляем жирные линии на строках 6, 8 и 10
        thick_border = Border(right=Side(style='thick'))
        for column_number in [8, 14, 24]:
            for row in worksheet.iter_rows():
                cell = row[column_number - 1]
                cell.border = Border(left=cell.border.left,
right=Side(style='thick'), top = cell.border.top, bottom= cell.border.bottom)

        # Сохраняем файл с применёнными стилями
        workbook.save(file_path)

        # Уведомление об успешном экспорте
        QMessageBox.information(self, "Экспорт завершен", f"Таблица успешно сохранена
в {file_path}")
    except Exception as e:
        # Вывод сообщения об ошибке, если что-то пошло не так
        QMessageBox.warning(self, "Ошибка экспорта", f"Не удалось сохранить файл.
(Необходимо выбрать корректный файл )")
```

- edit_total_for_export: объединяет данные перед экспортом.

```
def edit_total_for_export(self, combined_df):
    '''Собираем страничку Total для экспортируемого файла'''

    combined_df['Hand\n lifting'] = combined_df['Hand lifting'] -
combined_df['Start']
    combined_df['GA\n opening'] = combined_df['GA opening'] - combined_df['Start']
    combined_df['Object\n lifting'] = combined_df['Object lifting'] -
combined_df['Start']
    combined_df['GA max\n'] = combined_df['GA Max'] - combined_df['Start']
    combined_df['Total movement time'] = combined_df['Object Placed'] -
combined_df['Start']
    combined_df['Reaction time'] = combined_df['Hand\n lifting']
    combined_df['Time of max GA'] = combined_df['GA max\n'] - combined_df['Hand\n
lifting']
    combined_df['Time to reach'] = combined_df['Object\n lifting'] -
combined_df['Hand\n lifting']
    combined_df['GA%'] = (combined_df['Time of max GA']/combined_df['Time to reach'])
* 100
    combined_df['Time of object movement'] = combined_df['Total movement time'] -
combined_df['Object\n lifting']

    return combined_df
```

4 ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ

Интерфейс проекта реализован с использованием PyQt6. В основе лежат два файла pril.ui и settings.ui, отвечающие за главное окно и окно настроек.

4.1 Основное окно приложения pril.ui

Главное окно отвечает за импорт и экспорт данных, визуализацию в таблицах и графиках, а также за работу с настройками с помощью привязки графических элементов к методам python.

Основные элементы главного окна:

1) **QMainWindow**: окно главного приложения.
   - window - управляет всеми графическими элементами.
2) **QGroupBox**: группировка элементов.
   - groupBox – блок, где расположены кнопки импорта, экспорта и настроек, а также заголовок приложения.

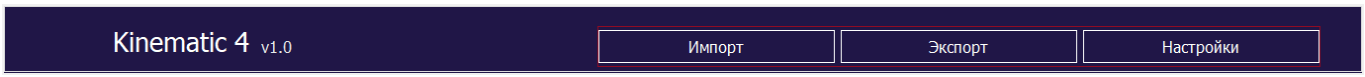

Рисунок 4.1 - QGroupBox

3) **QLabel**: текстовое наполнение.
   - label – отображает заголовок приложения “Kinematic4”.
   - label_2 – отображает версию приложения.

Рисунок 4.2 – Qlabel’s

4) **QHBoxLayout** (horizontalLayout_2): класс управления компоновкой содержащий в себе основные кнопки управления **QPushBitton**.
   - pushButton – кнопка вызова функции импорта данных.
   - pushButton_1 – кнопка открытия настроек.
   - pushButton_3 – кнопка вызова функции экспорта данных.

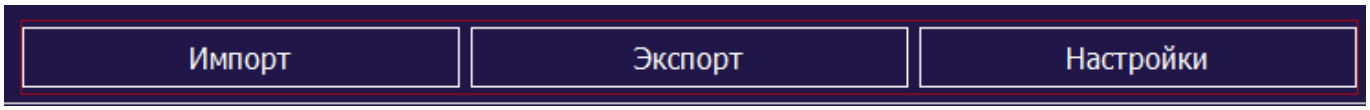

Рисунок 4.3 – QHBoxLayout с кнопками QPushButton

5) **QTableView**: отображение таблицы данных.
   - tableView – пространство отображения загруженных данных испытания в табличном виде.
6) **QGraphicsView**: графическое представление данных.

- graphicsView – пространство отображения графика на основе выбранного эксперимента.

7) **QHBoxLayout** (horizontalLayout_4): класс управления компоновкой содержащий в себе verticalLayout с **QCheckBox** для выбора параметров.

- checkBox - отвечает за начало эксперимента на графика. «Начало эксперимента [FG]»
- checkbox_2 - отвечает за момент подъема руки. «Подъем кисти»
- checkbox_3 – отвечает за момент максимальной апертуры захвата. «Апертура захвата (max) [FG]»
- checkbox_5 – отвечает за момент подъема объекта. «Подъем объекта»
- checkbox_6 – отвечает за момента опускания предмета. «Опускание предмета»
- checkbox_7 – отвечает за момент размыкания пальцев. «Размыкание пальцев [GA]»
- checkbox_9 – отвечает за путь открытия очков. «Открытие очков»
- checkbox_10 – отвечает за путь апертуры захвата. «Апертура захвата»
- checkbox_4 – отвечает за положение кисти. «Положение кисти (Z)»
- checkbox_8- отвечает за положение объекта. «Положение объекта (Z)»

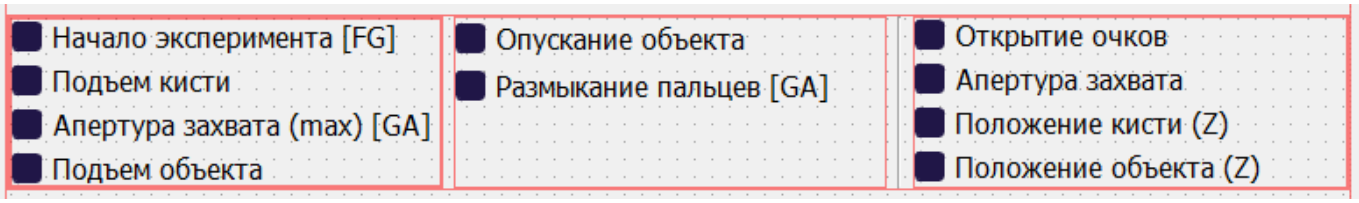

Рисунок 4.4 – QHBoxLayout с чекбоксами QCheckBox

8) **QPushButton**: функциональные кнопки.

- pushButton_6 – кнопка «Визуализация данных». Вызывает функцию визуализации данных эксперимента в зависимости от выбранного эксперимента.
- pushButton_7 – кнопка «Применить фильтр». Вызывает функцию обновления графика по выбранным элементам checkbox.

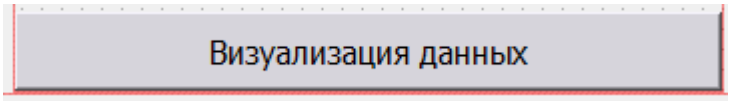

Рисунок 4.5 – PushButton_6

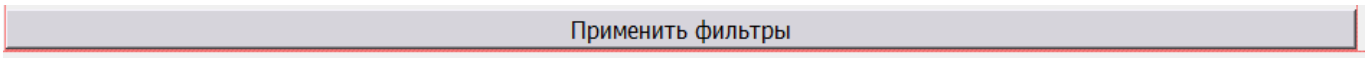

Рисунок 4.6 – PushButton_7

4.2 Окно настроек settings.ui

Окно настроек предоставляет пользователю возможность менять параметры расчета данных с импортируемого файла. Открывается нажатием кнопки «Настройки» из главного окна.

Основные элементы окна настроек

1) **QWidget**: бызовый класс для всех объектов пользовательского интерфейса.
   - ExtandedSettings – управляет всеми графическими элементами данного окна.
2) **QLineEdit**: поля ввода.
   - lineEdit – расстояние между пиками.
   - lineEdit_2 – пороговое значение скорости поднятия руки.
   - lineEdit_3 – пороговое значение открытия пальца.
   - lineEdit_4 – пороговое значение скорости опускания предмета.
   - lineEdit_5 – пороговое значение скорости подъема предмета.
   - lineEdit_6 – максимальное значение Frame-Glasses.
   - lineEdit_7 – минимальное значение Frame-Glasses.

Рисунок 4.7 – QLineEdit

3) **QPushButton**: кнопки управления.
   - pushButton_4 – кнопка (Обновить отчет) обновления таблицы на основе измененных пользователем параметров.
   - pushButton_5 – кнопка (Вернуть значения по умолчанию) сбрасывания параметров к значениям по умолчанию.

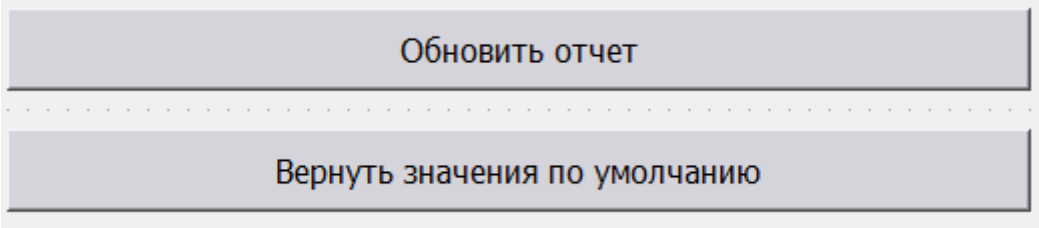

Рисунок 4.8 – кнопки управления

4.3 Взаимодействие между окнами

Главное окно приложения MainWindow загружает интерфейс из файла pril.ui и привязывает к соответствующим методам. Для загрузки вызывается метод LoadUI.

Окно настроек вызывается из главного окна при нажатии на кнопку «Настройки» и открытии файла settings.ui. Для загрузки вызывается метод LoadUI.

5 ТЕРМИНЫ И СОКРАЩЕНИЯ

Термин	**Полная форма**
Испытание	Физический процесс, проводимый в лаборатории, в ходе которого испытуемого просят схватить объект, который ставят перед ним, и поставить на заданное место. Движение повторяется 64 раза (эксперимента).
Эксперимент	Часть *Испытания*, в рамках которой выполняется одно из 64 движений в лаборатории
Кинематика	Раздел механики, изучающий движение тел без учета их массы и действующих на них сил. Основной целью кинематического анализа является нахождение значений и направлений скоростей и ускорений точек механизма, а также угловых скоростей и ускорений его звеньев.
Кинематический анализ	Процесс определение положений звеньев, траекторий отдельных точек механизма, угловых скоростей и ускорений звеньев, линейных скоростей и ускорений отдельных точек механизма
Моторное планирование	Способность понять, спланировать и реализовать незнакомое двигательное действие или последовательность действий.
Интеграция	Процесс объединения различных компонентов или систем с целью создания единого и функционального целого. В контексте разработки программного обеспечения и информационных технологий интеграция подразумевает взаимодействие между различными программами, приложениями или сервисами для обеспечения их взаимной совместимости и эффективной работы вместе.
Десктопное приложение	Программа, которая устанавливается на компьютер пользователя и работает под управлением операционной системы.
ПКМ	Правая кнопка мыши. Операция нажатия правой кнопки мыши для вызова контекстного меню или выполнения других действий, предусмотренных программой.
ЛКМ	Операция нажатия левой кнопки мыши для выбора объектов, запуска функций или выполнения других действий.

Модальное окно	Всплывающее окно, которое требует обязательного взаимодействия пользователя перед выполнением других операций. Например, окно подтверждения или предупреждения.
Контекстное меню	Выпадающий список команд или параметров, отображаемый при нажатии ПКМ на объекте или рабочей области.
Панель инструментов	Группа кнопок или иконок на экране, предоставляющих быстрый доступ к часто используемым функциям или командам.
Пользовательский интерфейс (UI)	Визуальная часть программы, с которой взаимодействует пользователь. Включает кнопки, поля ввода, окна и другие элементы.
Кнопка	Элемент интерфейса, на который можно нажать для выполнения определенной функции (например, "ОК", "Отмена", "Сохранить").
Форма ввода	Экран или окно, предназначенное для ввода данных пользователем. Может содержать текстовые поля, флажки, переключатели и другие элементы.
Флажок (чекбокс)	Элемент интерфейса в виде квадрата, который пользователь может включить или выключить для выбора параметра.
Поле ввода	Элемент интерфейса, где пользователь может ввести текст или данные.
Уведомление	Сообщение на экране, информирующее пользователя о текущем состоянии системы, результатах выполнения действия или ошибке.
Логирование	Процесс записи действий программы, ошибок и других событий для последующего анализа.
Диалоговое окно	Окно, предназначенное для взаимодействия пользователя с программой, например, для ввода данных или выбора опций.
Загрузка	Процесс открытия файла, программы или данных в систему.
Ошибка выполнения (runtime error)	Сообщение о некорректной работе программы, возникающее во время ее использования.

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц) в докум	№ документа	Входящий № сопроводительного докум и дата	Подп	Дата
Изм	измененных	замененных	новых	аннулированных					