Задание: и 27.10.23, 3.11.23 4 занятия.

1. Выбрать объект для управления (желательно связанный с тематикой ВКР). Объекты у каждого не должны повторяться. Например джойстик или пульт мобильного, летающего или манипуляционного робота.

2. Продумать наполнение интерфейса. Количество интерактивных объектов не менее 3 (кнопки, уровни и т.п.)

3. Запрограммировать форму интерфейса и оснастить его минимальной графикой (фон, текст и иконки на интерактивных элементах).

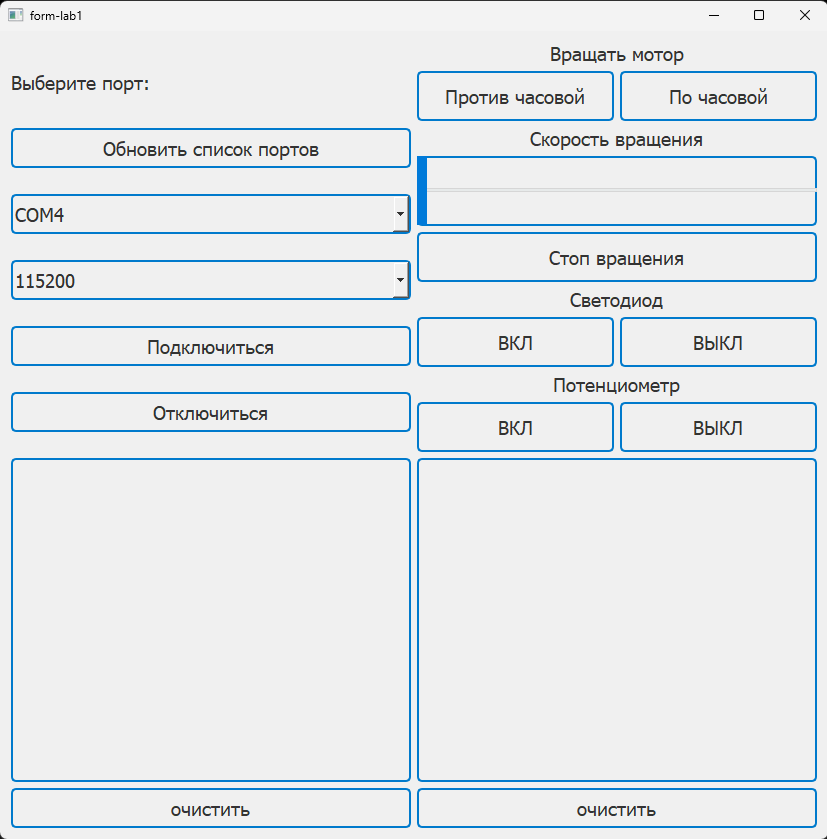
4. Связать работу по активации интерактивных элементов с объектом вкр (желательно) или с имитацией логических пробников (световой индикацией подтверждающей действие).

5. Сгенерировать файл проекта в виде пускового файла. И загрузить его в ответ на это задание. Также задание, скриншоты программы разработанного приложения, и листинг кода включить в файл ворд (желательно pdf) и загрузить в ответ на задание.

Ссылка на гит с проектом:

https://github.com/nameunique/DaGOI\_lab1/tree/main

Скриншот программы:



Листинги:

Скетч ардуино:

const int PIN\_STEP = 23;

const int PIN\_DIR = 22;

const int PIN\_EN = 21;

const int LED = 13;

const int potens = 36;

int currentSpeed = 5;

bool clockwise = true;

bool rotating = false;

bool led\_on = true;

bool is\_on = false;

bool able\_get\_potens\_val = false;

String commandBuffer = "";

void setup() {

pinMode(PIN\_STEP, OUTPUT);

pinMode(PIN\_DIR, OUTPUT);

pinMode(PIN\_EN, OUTPUT);

pinMode(LED, OUTPUT);

pinMode(potens, INPUT);

digitalWrite(PIN\_STEP, 1);

digitalWrite(PIN\_DIR, 0);

Serial.begin(115200);

delay(1000);

}

void executeCommand(String command) {

command.trim(); // Удалить начальные и конечные пробелы

if (!command.isEmpty()) { // Проверка на непустую строку

if (command.startsWith("s")) {

int newSpeed = command.substring(1).toInt();

currentSpeed = newSpeed;

} else if (command.startsWith("mode")) {

int mode = command.substring(4).toInt();

if (mode == 0) {

rotating = false; // Сброс флага вращения

is\_on = false;

} else if (mode == 10) {

rotating = true;

clockwise = false;

is\_on = true;

} else if (mode == 11) {

rotating = true;

clockwise = true;

is\_on = true;

}

} else if (command.startsWith("led")) {

int isledon = command.substring(3).toInt();

if (isledon == 0)

led\_on = false;

else

led\_on = true;

} else if (command.startsWith("p")) {

int ispon = command.substring(1).toInt();

if (ispon == 0)

able\_get\_potens\_val = false;

else

able\_get\_potens\_val = true;

}

// // Выводим команду в монитор порта

// Serial.print("Get from serial: ");

// Serial.println(command);

}

}

void loop() {

while (Serial.available() > 0) {

char incomingChar = Serial.read();

if (incomingChar == '#') {

executeCommand(commandBuffer);

commandBuffer = ""; // Очистить буфер после выполнения команды

} else {

commandBuffer += incomingChar;

}

}

digitalWrite(PIN\_EN, is\_on ? LOW : HIGH);

if (rotating) {

digitalWrite(PIN\_DIR, clockwise ? HIGH : LOW);

digitalWrite(PIN\_STEP, HIGH);

delay(currentSpeed);

digitalWrite(PIN\_STEP, LOW);

delay(currentSpeed);

}

digitalWrite(LED, led\_on ? LOW : HIGH);

if (able\_get\_potens\_val) {

int analogValue = analogRead(potens);

Serial.print("p" + String(analogValue) + "#");

}

}

Скрипт main.py

import sys

import os

from PyQt5 import uic

from PyQt5.QtWidgets import QApplication, QWidget

from PyQt5.QtCore import Qt, QThread, pyqtSignal

from datetime import datetime

import serial.tools.list\_ports

import serial

current\_file\_path = os.path.abspath(\_\_file\_\_)

folder\_path = os.path.dirname(current\_file\_path)

class SerialThread(QThread):

    data\_received = pyqtSignal(str)

    def \_\_init\_\_(self, port, baudrate):

        super(SerialThread, self).\_\_init\_\_()

        self.port = port

        self.baudrate = baudrate

        self.serial\_port = None

    def run(self):

        try:

            self.serial\_port = serial.Serial(self.port, self.baudrate, timeout=1)

            self.serial\_port.flushInput()

            self.serial\_port.flushOutput()

        except Exception as e:

            self.data\_received.emit(f"Ошибка при подключении к COM-порту: {str(e)}")

            return

        current\_message = ""

        while self.serial\_port.is\_open:

            data = self.serial\_port.read(1).decode('utf-8', errors='ignore')

            if data:

                current\_message += data

                if current\_message.startswith('p'):

                    if(current\_message.endswith("#")):

                        current\_message = current\_message[1:-1]

                        potentiometer\_value = int(current\_message)

                        self.data\_received.emit(f"Значение потенциометра: {potentiometer\_value}")

                        current\_message = ""

                        self.serial\_port.flushInput()

                else:

                    current\_message = ""

                    self.serial\_port.flushInput()

    def send\_message(self, message):

        if self.serial\_port is not None and self.serial\_port.is\_open:

            try:

                self.serial\_port.write(message.encode('utf-8'))

            except Exception as e:

                self.data\_received.emit(f"Ошибка при отправке данных: {str(e)}")

    def stop(self):

        if self.serial\_port is not None:

            self.serial\_port.close()

        self.terminate()

class MainWindowLocal(QWidget):

    def \_\_init\_\_(self):

        super(MainWindowLocal, self).\_\_init\_\_()

        uic.loadUi(folder\_path + "/mainform.ui", self)

        self.serial\_thread = None

        #region Start settings

        self.refreshSerialPorts()

        baudrates = ["9600", "19200", "38400", "57600", "115200"]

        self.cbBaudrate.addItems(baudrates)

        self.cbBaudrate.setCurrentIndex(len(baudrates) - 1)

        #endregion

        #region Events

        self.btRefreshSerialPorts.clicked.connect(self.refreshSerialPorts)

        self.btClearLogs1.clicked.connect(self.clearLog1)

        self.btClearLogs2.clicked.connect(self.clearLog2)

        self.btConnect.clicked.connect(self.connect\_serial)

        self.btDisconnect.clicked.connect(self.disconnect\_serial)

        self.btLedOn.clicked.connect(self.sendLedOn)

        self.btLedOff.clicked.connect(self.sendLedOff)

        self.btPotensOn.clicked.connect(self.sendPotensOn)

        self.btPotensOff.clicked.connect(self.sendPotensOff)

        self.btTurnMotorLeft.clicked.connect(self.TurnMotorLeft)

        self.btTurnMotorRight.clicked.connect(self.TurnMotorRight)

        self.btStopTurningMotor.clicked.connect(self.StopMotorTurning)

        self.sldSpeedMotor.valueChanged.connect(self.SliderSpeedChange\_Handler)

        #endregion

    def refreshSerialPorts(self):

        self.cbSerial.clear()

        ports = serial.tools.list\_ports.comports()

        for port in ports:

            self.cbSerial.addItem(port.device)

    def clearLog1(self):

        self.tbLogs1.clear()

    def clearLog2(self):

        self.tbLogs2.clear()

    def writeToLog1(self, message):

        self.tbLogs1.append(message)

    def writeToLog2(self, message):

        self.tbLogs2.append(message)

    def connect\_serial(self):

        if self.serial\_thread is not None and self.serial\_thread.isRunning():

            self.writeToLog1("Уже подключено к COM-порту.")

            return

        port\_name = self.cbSerial.currentText()

        baudrate = int(self.cbBaudrate.currentText())

        self.serial\_thread = SerialThread(port\_name, baudrate)

        self.serial\_thread.data\_received.connect(self.serial\_data\_receive\_handler)

        self.serial\_thread.start()

        self.writeToLog1(f"Подключено к {port\_name} с баудрейтом {baudrate}.")

    def disconnect\_serial(self):

        if self.serial\_thread is not None and self.serial\_thread.isRunning():

            self.serial\_thread.stop()

            self.writeToLog1("Отключено от COM-порта.")

            self.serial\_thread = None

    def send\_message\_to\_serial(self, mess):

        if self.serial\_thread is not None and self.serial\_thread.isRunning():

            self.serial\_thread.send\_message(mess)

            self.writeToLog1(f"Отправлено: {mess}")

    def serial\_data\_receive\_handler(self, messange):

        self.writeToLog2(messange)

    def sendLedOn(self):

        self.send\_message\_to\_serial("led1#")

    def sendLedOff(self):

        self.send\_message\_to\_serial("led0#")

    def sendPotensOn(self):

        self.send\_message\_to\_serial("p1#")

    def sendPotensOff(self):

        self.send\_message\_to\_serial("p0#")

    def TurnMotorLeft(self):

        self.send\_message\_to\_serial("mode10#")

    def TurnMotorRight(self):

        self.send\_message\_to\_serial("mode11#")

    def StopMotorTurning(self):

        self.send\_message\_to\_serial("mode0#")

    def SliderSpeedChange\_Handler(self):

        value = self.sldSpeedMotor.value()

        self.send\_message\_to\_serial(f"s{value}#")

def main():

    app = QApplication(sys.argv)

    mainWindow1 = MainWindowLocal()

    # mainWindow1.showFullScreen()

    mainWindow1.show()

    sys.exit(app.exec\_())

if \_\_name\_\_ == '\_\_main\_\_':

    main()