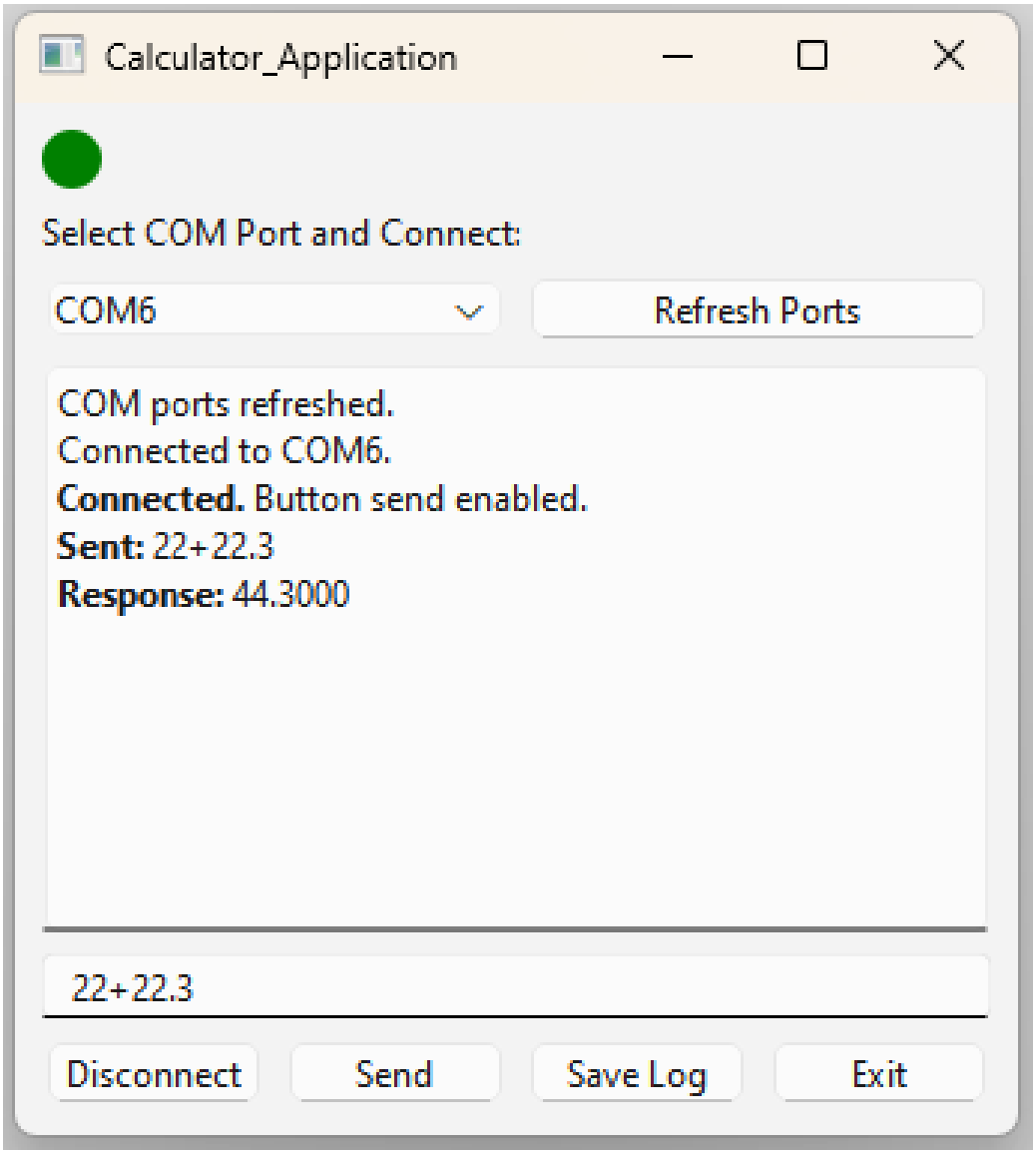
Erarbeitungs-/Reflexionsphase

Einleitung

In der zweiten Phase der Entwicklung des Projekts, welches die serielle Kommunikation zwischen einer PC-Anwendung und einem Mikrocontroller implementiert, wurden die in der Konzeptionsphase geplanten Schritte umgesetzt. Die Erarbeitungs-/Reflexionsphase beinhaltet die Entwicklung und Testung aller Module sowie die Überprüfung der Interaktion zwischen den Komponenten. Ziel war es, eine funktionale erste Version der Software bereitzustellen, die alle Grundfunktionen korrekt ausführt.

GitHub Link

Umsetzung der PC-Anwendung



- **Verbindungsmanagement:**
  - Die PC-Anwendung wurde in C++ unter Verwendung des Qt-Frameworks entwickelt.
  - Ein Hauptmerkmal ist die Verbindung zu einem Mikrocontroller über eine serielle Schnittstelle. Ein Button ermöglicht die Verbindung und Trennung durch den Aufruf der Funktion: „toggleConnection()“

```
void MainWindow::toggleConnection()
{
    if (serial->isOpen()) // Verbindung trennen wenn verbunden ···

    QString selectedPort = portSelector->currentText();
    if (selectedPort.isEmpty())
    {
        QMessageBox::warning(this, "Connection Error", "No COM port selected.");
        return;
    }

    serial->setPortName(selectedPort);
    serial->setBaudRate(QSerialPort::Baud9600);
    serial->setDataBits(QSerialPort::Data8);
    serial->setParity(QSerialPort::NoParity);
    serial->setStopBits(QSerialPort::OneStop);
    serial->setFlowControl(QSerialPort::NoFlowControl);

    if (serial->open(QIODevice::ReadWrite)) // Verbindung herstellen
    {
        manualDisconnection = false;
        logOutput->append("Connected to " + selectedPort + ".");
        updateConnectionStatus(true);
    }
    else
    {
        QMessageBox::warning(this, "Connection Error", "Could not open the selected COM port.");
    }
}
```

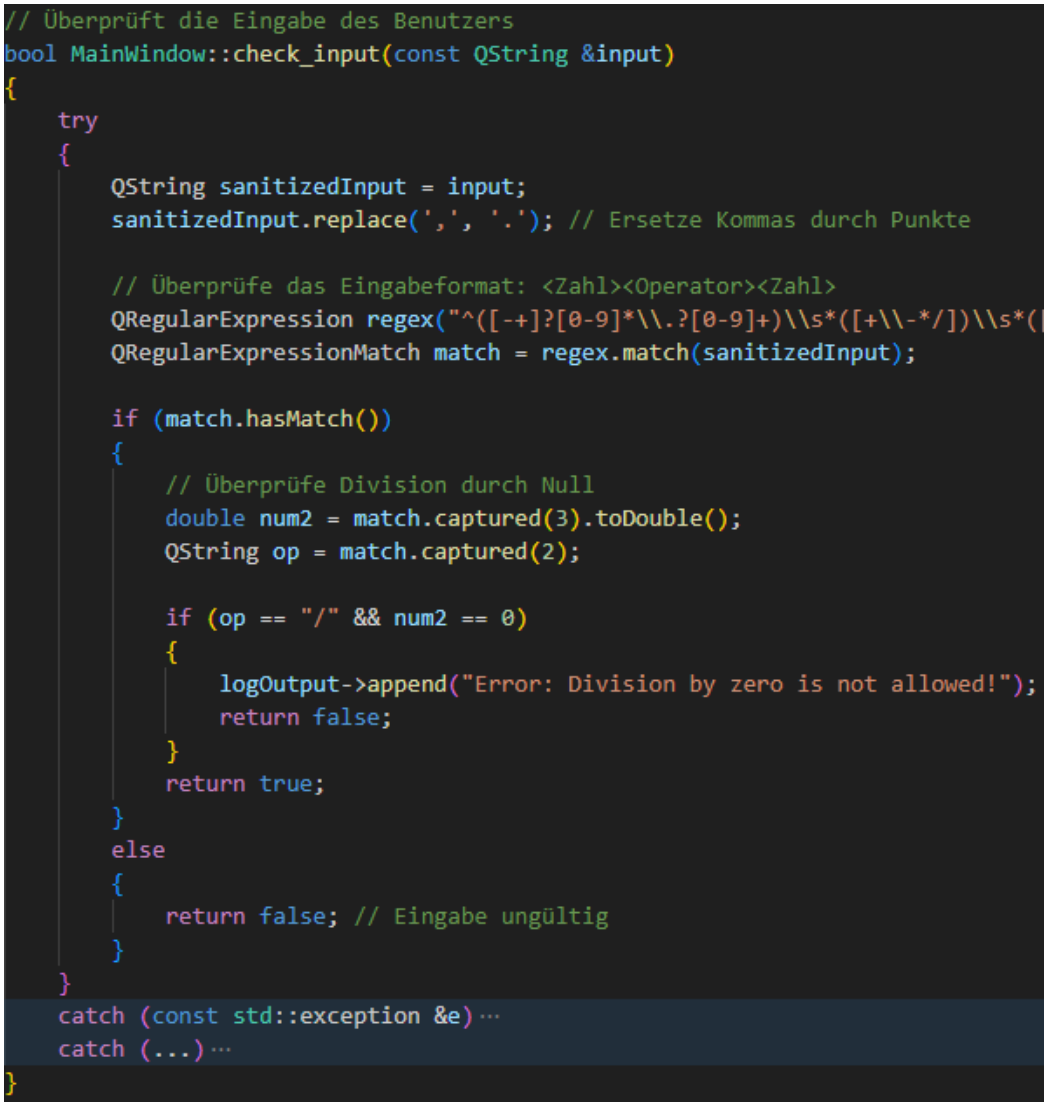
  - Es wurde ein Mechanismus implementiert, der die Verbindung alle 100 Millisekunden überprüft. Der Status wird durch eine LED-Anzeige (rot=nicht verbunden/grün=verbunden) visualisiert. Funktion: „run()“ des Objekts Connection Checker

```
// Die Hauptfunktion des Threads zur Überwachung der Verbindung
void ConnectionChecker::run()
{
    while (running)
    {
        QThread::msleep(highPriority ? 50 : 100);
        bool isOpen = false;

        try
        {
            MainWindow *mainWindow = static_cast<MainWindow *>(parent());
            if (!mainWindow->isProcessing())
            {
                isOpen = serial->isOpen() && serial->isWritable();
                if (isOpen) ···
            }
            else
            {
                isOpen = true; // Während der Verarbeitung Verbindung als offen annehmen
            }
        }
        catch (...)
        {
            isOpen = false; // Fehlerhafte Verbindung als getrennt markieren
        }

        emit connectionStatusChanged(isOpen); // Sende den Status
    }
}
```

- **Eingabevalidierung:**
    - Die Funktion check_input() überprüft die Benutzereingaben. Erlaubte Formate sind z. B. a+b, a-b, a*b und a/b.
    - Negative Zahlen und Dezimalwerte werden unterstützt. Es wird überprüft, ob durch null geteilt wird.



```
// Überprüft die Eingabe des Benutzers
bool MainWindow::check_input(const QString &input)
{
    try
    {
        QString sanitizedInput = input;
        sanitizedInput.replace(',', '.'); // Ersetze Kommas durch Punkte

        // Überprüfe das Eingabeformat: <Zahl><Operator><Zahl>
        QRegularExpression regex("^([-+]?[0-9]*\\.?[0-9]+)\\s*([+\\-*/])\\s*(
        QRegularExpressionMatch match = regex.match(sanitizedInput);

        if (match.hasMatch())
        {
            // Überprüfe Division durch Null
            double num2 = match.captured(3).toDouble();
            QString op = match.captured(2);

            if (op == "/" && num2 == 0)
            {
                logOutput->append("Error: Division by zero is not allowed!");
                return false;
            }
            return true;
        }
        else
        {
            return false; // Eingabe ungültig
        }
    }
    catch (const std::exception &e)···
    catch (...)···
}
```

- **Kommunikation und Interaktion:**
  - Mit dem „Send-Button“ wird die eingegebene Rechenoperation nach Überprüfung auf Richtigkeit an den Mikrocontroller gesendet.
  - Die Rückmeldung des Mikrocontrollers wird in einem Textfeld angezeigt.

```
// Die Eingabe an den µC Senden
void MainWindow::sendCalculation()
{
    //logOutput->append("<b>Debug:</b> sendCalculation() aufgerufen.");

    QString calculation = inputField->text();
    if (calculation.isEmpty())
    {
        logOutput->append("<b>Error:</b> Input field is empty!");
        return;
    }

    //logOutput->append("<b>Debug:</b> Eingabe erhalten: " + calculation);

    // Eingabe validieren
    if (!check_input(calculation))
    {
        logOutput->append("<b>Error:</b> Invalid input format! Use a+b | a-b | a*b | a/b.");
        return;
    }

    calculation.replace(',', '.');
    calculation += '\n'; // **Newline für Arduino!**
    //logOutput->append("<b>Debug:</b> Sende-Befehl formatiert: " + calculation);

    if (serial->isOpen() && serial->isWritable())···
    else···
}
```

  - Der Button: „Save Log“ ermöglicht das Speichern aller Interaktionen in einer Textdatei.

```
// Speichert die Kommunikationhistorie in eine Text-Datei
void MainWindow::saveLog()
{
    QString fileName = QFileDialog::getSaveFileName(this, "Save Log", "", "Text Fi
    if (fileName.isEmpty())
        return;

    QFile file(fileName);
    if (file.open(QIODevice::WriteOnly | QIODevice::Text))
    {
        QTextStream out(&file);
        out << logOutput->toPlainText();
        file.close();
        logOutput->append("Log saved successfully.");
    }
    else
    {
        logOutput->append("Error: Could not save the log.");
    }
}
```

- **Fehlerbehandlung:**
    - Fehlermeldungen, wie bei Verbindungsverlust, werden in einem separaten Logfeld und als Dialog angezeigt

```
void MainWindow::writeErrorLog(const QString &message)
{
    QFile logFile(QDir::currentPath() + "/error_log.txt");
    if (logFile.open(QIODevice::Append | QIODevice::Text))
    {
        QTextStream out(&logFile);
        out << QDateTime::currentDateTime().toString("yyyy-
        logFile.close();
    }
}
```

Umsetzung der Mikrocontroller-Anwendung

- **Hauptschleife:**
    - Die Hauptlogik der Anwendung

```
void loop() {
  // auf serielle Eingabe warten
  while (Serial.available() > 0) {
    char receivedChar = Serial.read();
    if (receivedChar == '\n' || receivedChar == '\r') {
      if (receivedMessage.length() > 0) {
        String Result = GetResult(receivedMessage);
        Serial.println(Result);
        receivedMessage = "";  // Reset für die nächste
      }
    } else if (receivedChar == "") {
      String Result = GetResult(receivedMessage) + "";
      Serial.println(Result);
      receivedMessage = "";  // Zurücksetzen der empfang
    } else {
      receivedMessage += receivedChar;  // Anhängen von
    }
  }
}
```

- **Initialisierung:**
    - Konfiguration der seriellen Schnittstelle zur Kommunikation mit der PC-Anwendung.

```
void setup() {
  Serial.begin(9600);
}
```

- **Eingangsverarbeitung und Berechnung:**
    - Die empfangenen Daten werden in drei Variablen (Zahl1, Operator, Zahl2) aufgeteilt.
    - Bei Dezimalzahlen wird das Komma in einen Punkt umgewandelt.
    - Die arithmetische Operation wird basierend auf dem Operator ausgeführt.
    - Division durch null wird abgefangen und entsprechend behandelt.

```
String GetResult(String input_string) {
  char operation;
  double num1, num2;
  double result;
  int operator_index = -1;
  int string_len = input_string.length();  //Abfrage der Länge der Zeichenkette
  //Suche nach dem Index der Operation in der Zeichenfolge des arithmetischen Ausdrucks
  for (int i = 1; i <= string_len; i++) {
    operation = input_string[i];
    if ((operation == '+') || (operation == '-') || (operation == '*') || (operation == '/')) {…
    }
  }
  //Wenn der Operator nicht gefunden wurde, wird eine Fehlermeldung angezeigt
  if (operator_index == -1) { return "Error: operation not found"; }

  //Die Ausdrücke trennen und die Zahlen von String in Double umwandeln
  String s_num1 = input_string.substring(0, operator_index);
  String s_num2 = input_string.substring(operator_index + 1, string_len);
  num1 = s_num1.toDouble();
  num2 = s_num2.toDouble();

  //Die eigentliche Berechnung durchführen
  switch (operation) {
    case '+':…
    case '-':…
    case '*':…
    case '/':…
  }
  String s_result = String(result, 4);  //Long Double in String umwandeln mit 4 Nachkommastellen
  return s_result;
}
```

- **Ausgabe:**
    - Das Ergebnis wird als Zeichenkette formatiert und an die PC-Anwendung zurückgesendet.

```
String Result = GetResult(receivedMessage) + "";
Serial.println(Result);
```

Codequalität und Testung

- **Kommentierung:** Der gesamte Code wurde mit klaren Kommentaren versehen, um die Verständlichkeit zu erhöhen.
- **Lesbarkeit:** Einheitliche Namenskonventionen und einheitliches Codelayout wurden eingehalten.
- **Testen:**
    - Jedes Modul wurde einzeln getestet.
    - Die Interaktion zwischen PC- und Mikrocontroller-Anwendung wurde durch verschiedene Szenarien überprüft (z. B. fehlerhafte Eingaben, Verbindungsverlust, gültige Berechnungen).
    - Das System wurde erfolgreich auf die Behandlung von Sonderfällen geprüft.

Herausforderungen und Lösungen

- **Verbindungsverlust erkennen:**
    - Herausforderung: Verbindungsverluste korrekt zu erkennen und zu melden.
    - Lösung: Ein Hintergrundthread überprüft den Verbindungsstatus. Ein Meldungsdialog erscheint nur einmal pro Verbindungsverlust.
- **Flexibilität bei Endzeichen:**
    - Herausforderung: Unterschiedliche Endzeichen (\n, \r, \r\n) zu unterstützen.
    - Lösung: Eine Konfigurationsoption wurde implementiert, mit der im Code für die PC-Anwendung und Mikrokontroller-Anwendung das Endzeichen fest ist.
- **Eingabeverarbeitung:**
    - Herausforderung: Benutzerfreundlichkeit und Fehlertoleranz bei Eingaben sicherzustellen.
    - Lösung: Die Funktion check_input() wurde robust gestaltet, um alle gängigen Eingabeformate zu akzeptieren und Benutzer bei Fehlern zu informieren.