Figure 1 Logo Delta dynamics.

PROJECT S3

Product report

Improving and Expanding the Delta Robot

ESE-2a-n – 4th of January 2024

Thom Peters(2104909) & Ruben Konings(2108385)
t.peters13@student.han.nl & rl.konings@student.han.nl

1. Version revision

Version	Date	Editor	Note
001	12-12-2023	Thom Peters	Added layout
002	15-12-2023	Thom Peters and Ruben Konings	Start
003	19-12-2023	Thom Peters and Ruben Konings	Added pictures and figures
004	28-12-2023	Thom Peters and Ruben Konings	First commit
010	29-12-2023	Thom Peters	Added conclusion and appendixes
100	29-12-2023	Thom Peters and Ruben Konings	First version released

2. Table of contents

3. Introduction

Our project revolves around upgrading a Delta Picker robot, originally built by a student. The goal is straightforward: to fix the existing problems with the robot and to develop a user-friendly interface for the software already in use.

The Delta Picker, despite its innovative design, has shown some operational challenges that affect its performance. Our first task is to diagnose and resolve these issues. This will involve a close examination of the robot's mechanical and electronic components to find and fix any faults.

Along with the hardware fixes, we are also focusing on making the robot easier to use. This involves designing a new Graphical User Interface (GUI) that works with the current software. The aim is for this interface to be easy to understand and use, even for those without deep technical knowledge.

This project is about enhancing the existing Delta Picker. By addressing its current shortcomings and adding a more user-friendly interface, we aim to not only improve its functionality but also make it a more accessible and efficient tool in the world of robotics.

4. Preliminary investigation

4.1 Problem definition

For this project, there is already an existing delta robot. This robot must be adapted in such a way that it does not cause any problems. Functions must also be added so that the robot can be controlled. The robot must also be able to be controlled from a user interface.

The problem with the delta robot is that the coordinates change as the robot moves. A deviation can be observed and measured, which initially is a deviation in the Z-direction.

A user interface must be designed to control and direct the robot.

4.2 The Delta X2 Robot

In the initial stage of our project, we conducted a thorough preliminary research, which was essential in shaping our approach and strategies. This research focused on three key areas: the stepper motor, a comparative study of another delta robot to which we had access, and the design of the Printed Circuit Board (PCB).

4.3 Research on the stepper motor

The stepper motor is a crucial component of the delta robot, as it provides precision and control in the robot's movements. Our research focused on several types of stepper motors, their operational characteristics, and how these can influence the performance of the delta robot. We looked at aspects such as torque, step angle, and energy efficiency. Additionally, we investigated how different control methods, like micro stepping, can enhance the accuracy and smoothness of the robot's movements. This insight helped us in selecting the right stepper motor for our project and developing an optimized control strategy.

4.4 Comparative study of another delta robot

We had the unique opportunity to study another delta robot used in a similar application. This allowed us to make direct comparisons between our robot and an existing, functioning model. We specifically looked at mechanical construction, movement efficiency, and how the stepper motors were integrated and controlled. This comparative study provided us with valuable insights into best practices, common pitfalls, and potential improvements that we could apply to our own design.

4.5 Research on PCB design

The PCB design plays a vital role in the overall functionality of the delta robot. Our research in this area included studying various PCB designs, component selection, and layout for optimal performance and reliability. We focused on minimizing noise, maximizing efficiency in energy transfer, and ensuring stable and reliable communication between the various parts of the robot. This research was crucial to designing a PCB that not only met our current requirements but was also flexible enough to accommodate future upgrades and modifications.

This preliminary research laid a solid foundation for our project, enabling us to enter the design and development phase with confidence and informed knowledge. The combination of theoretical understanding and practical insights allowed us to take an informed approach in building and optimizing our delta robot.

5. Functional design

5.1 Functional specifications

In consultation with the client, the following list of functional specifications has been drawn up based on the SMART criteria (Unknown, SMART criteria, 2022). The specifications are prioritized in the second column according to the MoSCoW method (Unknown, MoSCoW method, 2022). (Arends, 2022)

SMART functional design		
#	**MoSCoW**	**Description**
F1	**M**	**The Delta robot must work properly**
F1.1	M	This requirement mandates that the Delta robot should consistently and accurately perform its designated tasks. It holds critical importance, especially in applications like manufacturing, assembly, or precision operations.
F1.2	M	Ensuring the robot maintains constant coordinates is essential for precision over time.
F1.2.1	M	• Accurate mathematical calculations are pivotal to maintaining positional stability.
F1.2.2	M	• Avoiding motor and arm flex ensures precise and reliable operation.
F1.2.3	M	• Unrestricted joint movement is crucial for achieving smooth and precise motion.
F1.2.4	M	• Software improvement of the speed of the program or efficiency
F1.2.5	M	• Improve the communication and the speed
F2	**M**	**The Delta robot must work with the Qube**
F2.1	M	This requirement stipulates that the Delta robot must seamlessly integrate with the Qube platform, guaranteeing compatibility.
F2.2	M	Compatibility with the Qube's software is paramount to ensure effective control and monitoring within the Qube ecosystem.
F3	**S**	**The Delta robot must work with plug and play**
F3.1	S	This requirement necessitates straightforward and user-friendly setup and configuration, emphasizing ease of use.
F3.2	S	Uniform control methods enhance the user experience, simplifying the management of multiple robots concurrently.
F3.2.1	S	• Sharing a single ethernet cable reduces clutter and streamlines physical setup.
F4	**M**	**There must be a User Interface on a PC to control the Delta robot**
F4.1	M	This requirement underscores the importance of a user-friendly PC interface to enable operators to interact with and control the robots effectively.
F4.2	M	Individual operation allows for independent control of each robot when necessary.
F4.3	M	The jogging function facilitates manual control for precise positioning, supporting movement in the Y-axes, X-axes, and Z-axes.

F4.4	M	Automated movement programs using XYZ coordinates streamline repetitive tasks.
F4.5	M	The gripper's capability to open and close is essential for tasks involving object manipulation.
F5	**W**	**The Delta robot should work with a pendant**
F5.1	W	This suggests that the robot should offer compatibility with a pendant control device, providing an intuitive control option.
F5.2	W	Consistency with the PC UI controls enhances user familiarity and simplifies transitioning between control methods.
F6	**S**	**Consideration of advanced safety features, such as collision detection and emergency stop mechanisms**
F6.1	S	Advanced safety features add an extra layer of protection for the Delta robot.
F6.1.1	S	• Collision Detection: When a collision risk is identified, the robot can stop or slow down, preventing damage to itself, nearby objects, and ensuring the safety of human operators.
F6.1.2	S	• Emergency Stop Mechanisms: These mechanisms provide a quick and efficient way to halt robot operations in emergency situations.
F7	**W**	**Avoidance of implementing features that significantly increase project complexity without providing clear benefits**
F7.1	W	The focus should be on features that directly contribute to the system's efficiency, safety, or usability. Any proposed features should undergo a cost-benefit analysis to assess their impact on project timelines, budgets, and overall feasibility.
F7.2	W	The project team should maintain a balance between feature-rich functionality and simplicity to deliver a product that meets its intended purpose without unnecessary complexity.

Figure 2 Functional specifications.

5.2 Technical specifications

The following table shows the technical specifications of the project.

SMART technical specifications		
#	**MoSCoW**	**Description**
T1	**M**	For communication from the computer to the delta robot, the EtherCAT protocol is used.
T2	**M**	The development environment of the computer is TwinCAT.
T3	**M**	Programming is done in the C++ language.

Figure 3 Technical specifications (M = Must, S = Should, C = Could, W = Would).

5.3 Graphical User Interface (GUI)

To access the Graphical User Interface (GUI), launch a web browser and enter the URL http://localhost:1010/ once the TwinCAT program is running. This will bring up the GUI for interaction.

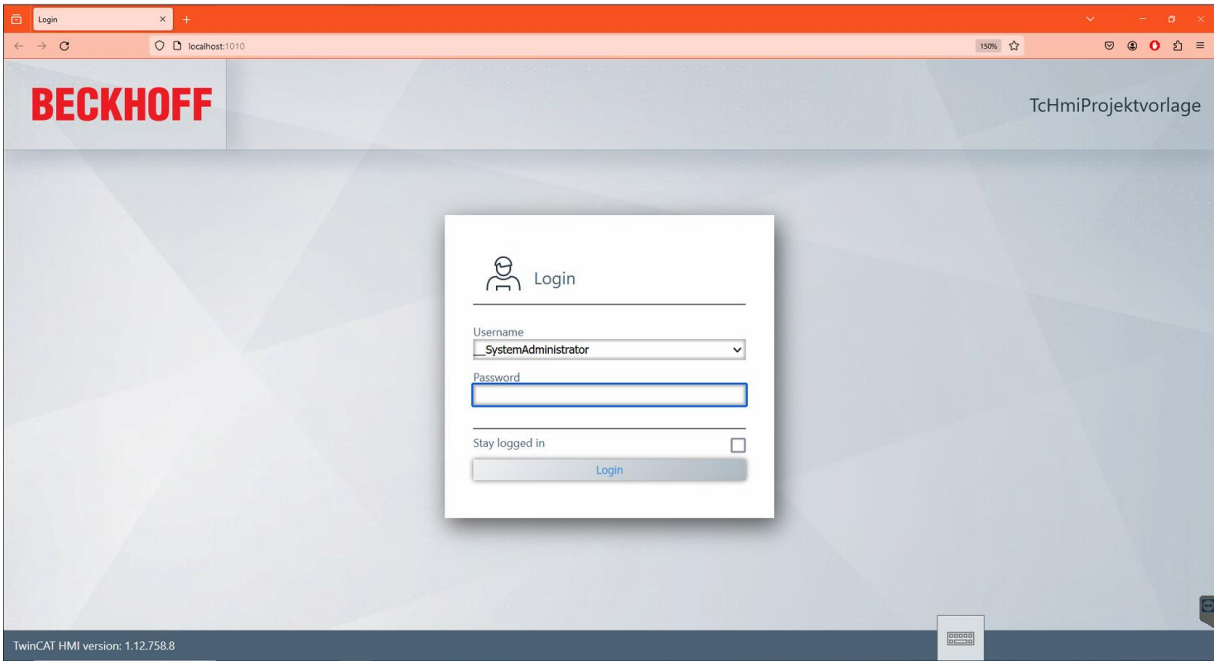

Figure 4 GUI Lock screen.

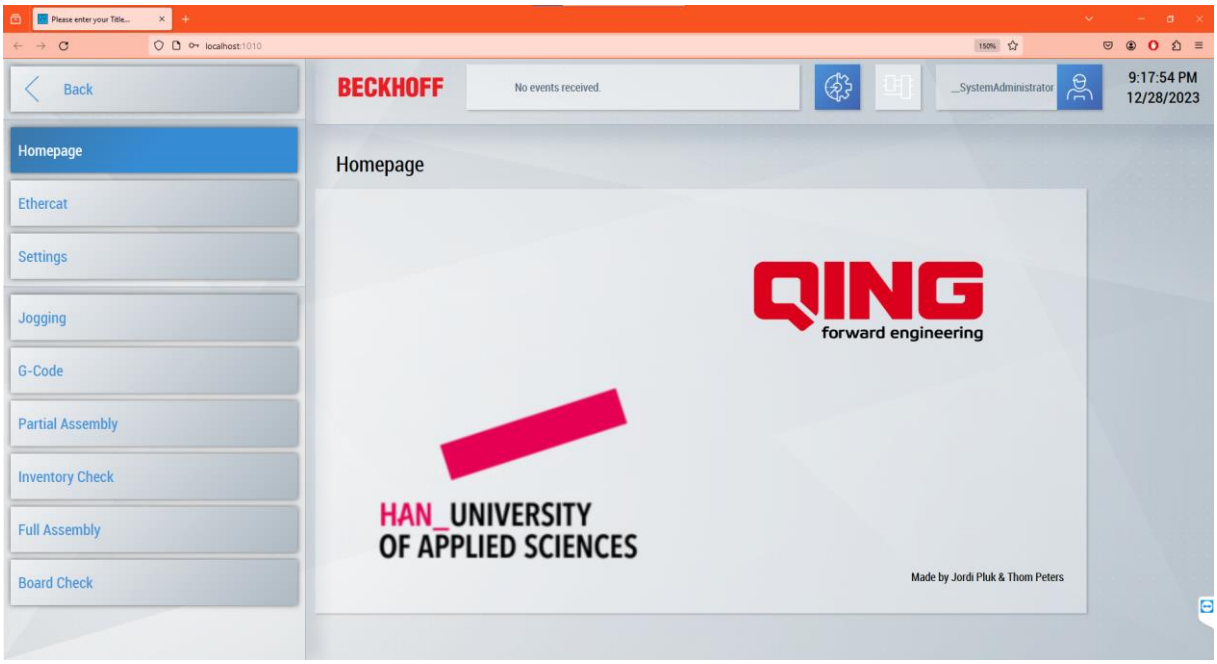

Figure 5 GUI Home screen.

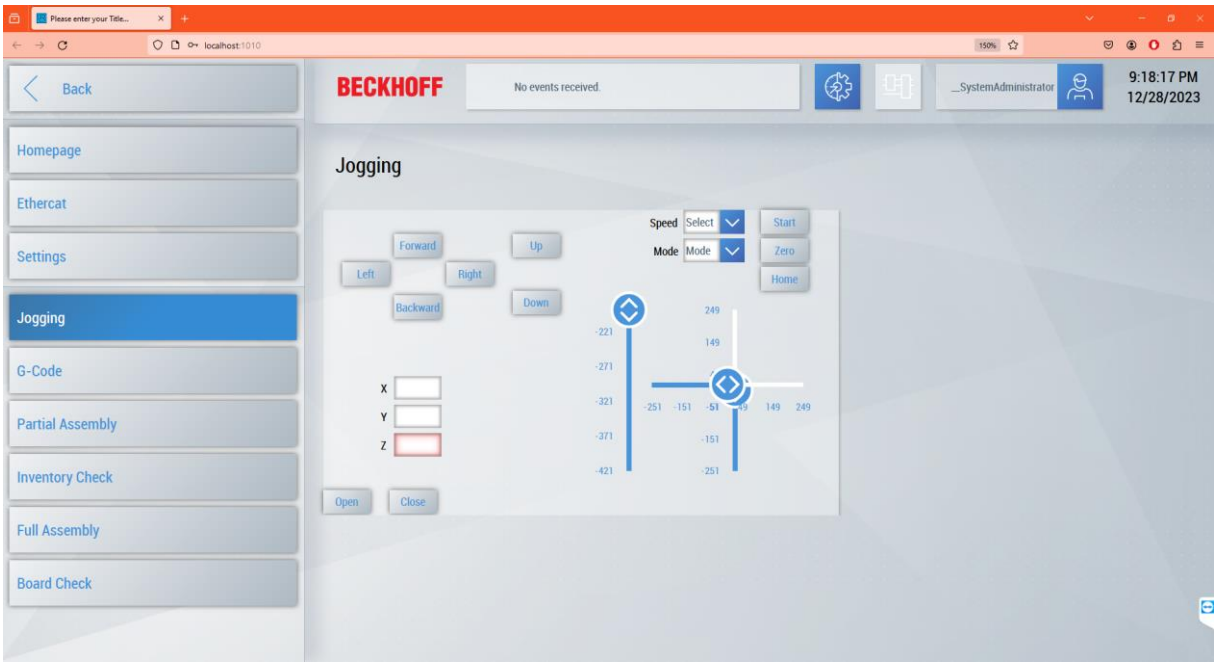

Figure 6 GUI Jogging-screen.

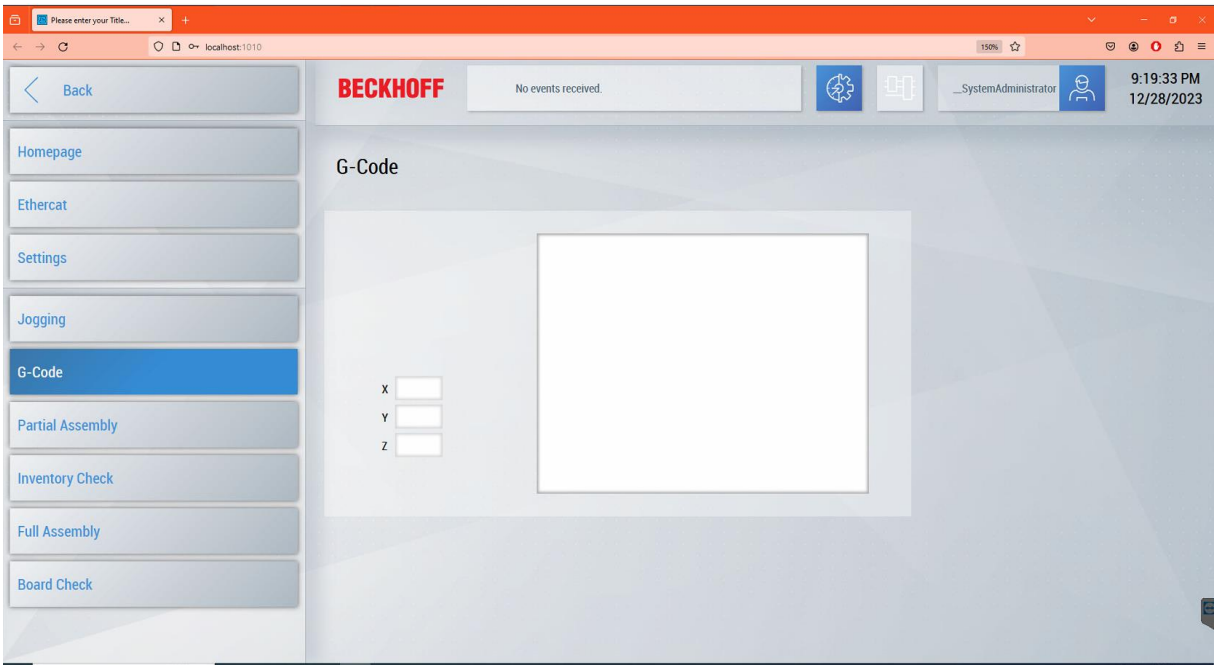

Figure 7 GUI GCode-screen.

6. Technical design

6.1 Hardware

In our project report, we delve into the intricate setup of the motor and control systems of our delta picker. Central to our design is the integration of motors, each connected to stepper drives, which are in turn connected to an Arduino Uno. This setup forms the backbone of our robotic system, ensuring precise control and movement.

Motor and Arm Design

The motors are integral to the functionality of the delta picker, dictating the movement and precision of the robot's arms. Attached to the shafts of these motors are 3D-printed upper arms, each designed to move independently. This design allows for precise control over the height and positioning of the delta picker's hand, enabling it to perform tasks with high accuracy.

Electrical Design and PCB Implementation

A significant advancement in our project was the transition from standard wiring to a more sophisticated Printed Circuit Board (PCB) system. This transition was made to streamline the electrical connections and enhance the overall efficiency and reliability of the robot. The PCB is designed to manage an input of 24 volts with a maximum of 10 amperes. This power input is also directed into the PCB, which plays a crucial role in power management and distribution.

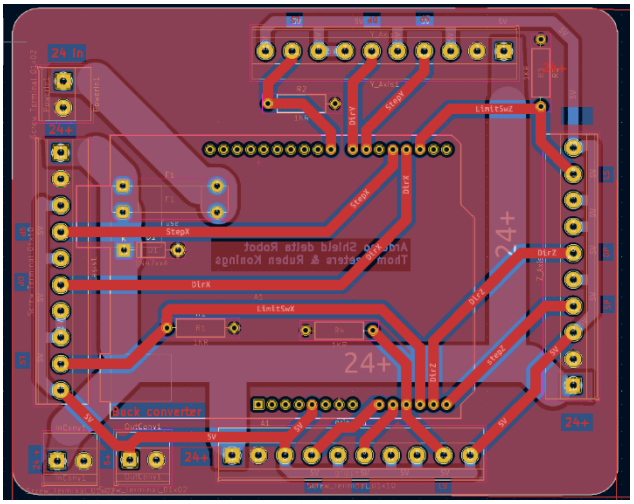

Figure 8 PCB design.

Within the PCB, the voltage is regulated, converting the 24 volts down to 5 volts, necessary for powering the control buttons used for the homing process of the delta picker. This voltage regulation is vital for ensuring that the different components of the robot receive the appropriate power levels for optimal operation. Additionally, the 24 volts are maintained for the motors, ensuring they have sufficient power for all movements and functions.

This sophisticated electrical design, and the implementation of the PCB not only enhanced the functionality of our delta picker but also made it more robust and dependable. The integration of these systems highlights our project's commitment to technical excellence and innovation in robotic design.

6.2 Software

6.2.1 Graphical User Interface (GUI)

The software aspect of the delta picker robot project encompasses a sophisticated Graphical User Interface (GUI) and a versatile G-Code mode, both designed to provide an intuitive and comprehensive control experience for the user. Developed in C++ using the TwinCAT environment and Visual Studio IDE, the software is tailored to meet the demands of precision in robotic operations and troubleshooting.

6.2.2 GUI development

The GUI is engineered with a focus on user-friendliness and precision. It features a jogging control panel for manual robot movement along X, Y, and Z axes, along with directional buttons for intuitive navigation. Users can enter specific coordinates for precise control, and the GUI offers a graphical representation of the robot's range of motion. Speed and mode selections are adjustable via dropdown menus, offering dynamic control over the robot's operation.

As a debugging tool, the GUI facilitates an efficient feedback loop between software commands and the hardware's response. This is especially crucial in identifying and addressing movement inaccuracies. The design of the GUI underscores the importance of a seamless user experience, allowing for simple yet robust control over the complex mechanics of the robot.

6.2.3 G-Code mode integration

The G-Code mode is a significant enhancement to the GUI, providing an interface for advanced users to input G-Code commands directly. This mode enables:

- **Linear Movements**: Using commands like **G1 X100 Y100 Z-250**.
- **Delays**: With commands such as **G4 3** for a pause of 3 seconds.
- **Homing**: Using **G28** to return the robot to its home position.
- **Gripper Control**: With **M04** and **M05** for closing and opening the gripper, respectively.
- **Speed Adjustment**: Via **M204 75**, to set the speed to a specified unit.

This mode offers the flexibility for users to programmatically customize the robot's movement and automate tasks, bridging traditional CNC programming with modern robotics control.

6.2.4 Collaborative effort and user experience

Developed collaboratively, the software reflects a harmonious blend of usability and functionality. It caters to a spectrum of users, from those seeking straightforward control to experts requiring detailed command sequences. The GUI's interaction with the hardware is continuously refined to ensure a responsive and cohesive experience.

6.2.5 Conclusion and outlook

The software component is critical to the delta picker robot's functionality, offering a robust interface for control and diagnostics. The project's commitment to innovation is evident in the GUI's design and the inclusion of G-Code mode. Future developments aim to enhance features such as syntax highlighting, G-Code validation, and expanding the command set.

The ongoing advancements in the software are expected to elevate the performance of the delta picker robot, promising improved precision and an enriched user experience. As the project progresses, the software will remain a centrepiece, evolving to meet the challenges and user needs in the field of robotics control systems.

7. Realisation

7.1 Debugging

7.1.1 The problem

In the initial stages of our project, we were faced with a perplexing issue with our delta picker robot. It had a mind of its own, refusing to go to the exact same spot twice even though we had not changed the instructions. Precision is the name of the game in the robot world, especially for the tasks we envisioned it performing in industrial settings. To get to the root of this, we went back to basics: we fastened a pencil to the robot's 'hand' to trace its path on paper. This simple tactic was our initial strategy to check the robot's path fidelity. However, it was not foolproof. We noticed errors, particularly along the vertical (Z) axis — sometimes the pencil would jab into the paper rather than glide over it, indicating an undesired movement.

7.1.2 Stepper motors

The hardware side of things brought its own set of challenges. The stepper motors, which are vital for the robot's movement, were occasionally losing their connections, leading to erratic behaviour. In response, we took a deep dive into improving the Printed Circuit Board (PCB) — the central hub of the robot's electrical system. We hand-picked components with a proven record for durability and focused on perfecting our assembly techniques. Despite these improvements, the robot's positioning remained unpredictable. This suggested that we might be dealing with more complex issues, potentially related to the mechanical structure or the software algorithms that drive the robot's movements.

7.1.3 Comparative Analysis

To broaden our understanding, we examined a commercially available delta picker that was known for its reliability. One striking difference was the use of springs in the arm joints of the commercial model, which helped maintain a steady hand during movements. Inspired by this, we contemplated a redesign of our robot's arms to incorporate a similar feature while ensuring they did not clash at higher elevations. We also spotted a weakness in the plexiglass base — it tended to flex, affecting the robot's precision. Strengthening this base became a priority. Additionally, the issue of the arms slipping on the motor shafts was a red flag; gearing systems were a promising solution for providing the necessary grip and torque.

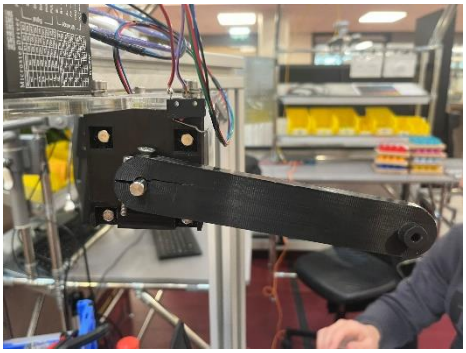

Figure 9 Deltapicker arm.

7.1.4 Recommendations

With the insights gained from our firsthand experience and comparative analysis, we have identified several promising avenues for enhancing our robot's performance. One area we have only scratched the surface of is the stepper motors' micro stepping capabilities. Fine-tuning these settings is key to solving our precision woes, but we've yet to explore this fully due to time constraints. Our groundwork has set the stage for future teams to delve into this and other identified areas, such as arm mechanics and control software, to further the evolution of delta picker robots in the industry.

7.2 Hardware

7.2.1 Printed Circuit Board (PCB)

Our approach to the PCB design has been intentionally straightforward, with an emphasis on making troubleshooting as painless as possible. The choice to use through-hole technology caters well to our prototyping phase, where manual tweaks are common.

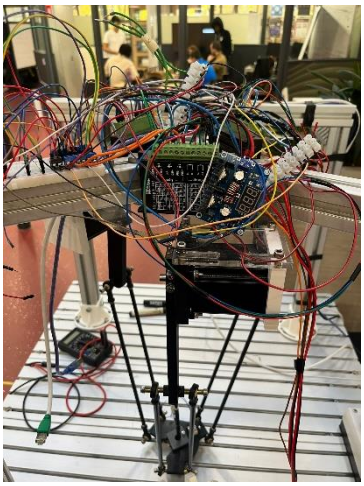

Figure 10 Deltapicker cable management before PCB.

The compact design of the PCB was a strategic decision, reducing the space and complexity that often accompanies larger boards. This consideration significantly eases the process of identifying and rectifying issues that may arise during development and testing.

The meticulously organized wiring is another aspect of our design philosophy, aimed at preventing signal interference and electromagnetic disturbances. Every connection is routed thoughtfully to support this goal. We have also gone to great lengths to label each component and connector clearly. This diligence ensures that anyone who steps in for troubleshooting can quickly identify the parts and understand the layout.

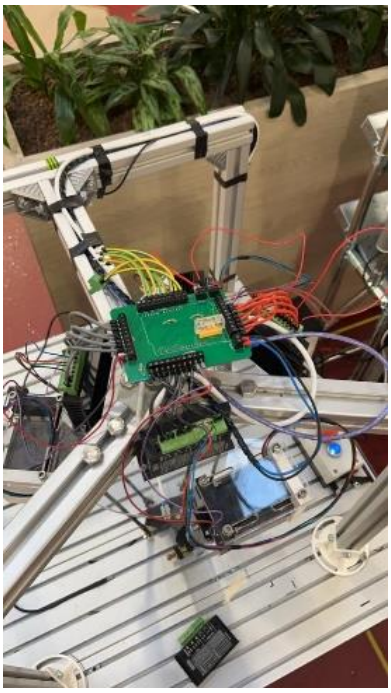

Figure 11 Deltapicker cable management after PCB.

Modularity in design has allowed us to break down the circuitry into manageable blocks, a choice that pays dividends when isolating and addressing specific issues. Furthermore, we have integrated dedicated test points and ports throughout the board, which serve as convenient access points for our diagnostic tools, like oscilloscopes and multimeters.

Considering the compact size of our PCB, power management was another critical design focus. We implemented robust filtering and decoupling strategies to prevent power fluctuations that could otherwise lead to complex and confusing hardware behaviour.

The PCB design, with its through-hole parts and intentional layout, is a testament to our focus on creating a user-friendly, adaptable, and educational platform. It is designed to evolve smoothly from a prototype to a potential small-scale production without losing the ease of manual adjustments that are so crucial during the development phase.

In sum, our PCB design marries simplicity with functionality, striking a balance that serves the dual purpose of facilitating development while preparing for future scalability. It is a careful blend of old-school accessibility with the foresight of modern design requirements, ensuring that our delta picker robot can meet the demands of both the present and the future.

7.3 Software

The software section elucidates the developed code that drives the delta picker robot. We provide well-commented code snippets with syntax highlighting for readability. These snippets, no longer than 20 lines each, demonstrate key functionalities of the most relevant subsystems. The full code listings are available in the appendices.

7.3.1 Motor Movement Algorithms

The algorithms for motor movement are essential for calculating the precise angles required for the robot's arms to reach specific coordinates.

```
// AngleCalculation.cpp
double CalculateAngles(double X00, double Y00, double Z00)
{
  double Y1 = -0.5 * 0.57735 * Rb; // f/2 * tan(30 deg)
  Y00 -= 0.5 * 0.57735 * Rh;       // shift center to edge

  double a = (pow(X00, 2) + pow(Y00, 2) + pow(Z00, 2) + pow(B, 2) -
pow(F, 2) - pow(Y1, 2)) / (2.0 * Z00); // z = a + b*y
  double b = (Y1 - Y00) / Z00;

  double d = -(a + b * Y1) * (a + b * Y1) + B * (b * b * B + B); //
discriminant

  double Yj = (Y1 - a * b - sqrt(d)) / (b * b + 1); // choosing outer
povar
  double Zj = a + b * Yj;

  double Angle = atan(-Zj / (Y1 - Yj)) * 180 / 3.14159 + ((Yj > Y1) ? 180
: 0);
  double ReverseAngle = 90.000 - Angle;

  return ReverseAngle;
}
```

This function calculates the inverse kinematics for a single motor, determining the angle necessary to reach a point in 3D space.

```
// AngleCalculation.cpp
ThetaResults CoordinatesToDelta(double X0, double Y0, double Z0)
{
  ThetaResults Results;

  Results.ThetaX = CalculateAngles(X0, Y0, Z0);
  //                      Rotate coordinates +120 degrees when looking
top down for Y motor
  Results.ThetaY = CalculateAngles((X0 * cos120 + Y0 * sin120), (Y0 *
cos120 - X0 * sin120), Z0);
  //                      Rotate coordinates -120 degrees when looking
top down for Z motor
  Results.ThetaZ = CalculateAngles((X0 * cos120 - Y0 * sin120), (Y0 *
cos120 + X0 * sin120), Z0);

  return Results;
}
```

This function adjusts the angles for all three motors, allowing the robot to move its end effector to any point within its working envelope.

7.3.2 Sensor data processing

Sensor data is crucial for calibrating the robot's movements and ensuring proper interaction with the environment.

```
// DeltaPickerFunctions.cpp
void DeltaPickerCalibrate(void)
{
  int SWXEnabled = 0;
  int SWYEnabled = 0;
  int SWZEnabled = 0;

  MotorX.setPinsInverted(true, true, true);
  MotorY.setPinsInverted(true, true, true);
  MotorZ.setPinsInverted(true, true, true);

  bool LimitSwitchX = false, LimitSwitchY = false, LimitSwitchZ = false;
  while (!(SWXEnabled && SWYEnabled && SWZEnabled))
  {
    // Read Limit switches
    LimitSwitchX |= digitalRead(LIMIT_SWITCH_PIN_X); // X Motor
    LimitSwitchY |= digitalRead(LIMIT_SWITCH_PIN_Y); // Y Motor
    LimitSwitchZ |= digitalRead(LIMIT_SWITCH_PIN_Z); // Z Motor

    if (LimitSwitchX)
    {
      SWXEnabled = 1;
    }
    if (LimitSwitchY)
    {
      SWYEnabled = 1;
    }
    if (LimitSwitchZ)
    {
      SWZEnabled = 1;
    }
    Serial.println("Moving");
    MotorX.move(-500);
    MotorY.move(-500);
    MotorZ.move(-500);
    if (!SWXEnabled)
      MotorX.run();
    if (!SWYEnabled)
      MotorY.run();
    if (!SWZEnabled)
      MotorZ.run();
  }

  // put motors on 90 degrees        Exactly 420 steps! DONT CHANGE
  MotorX.move(420);
  MotorY.move(420);
  MotorZ.move(420);
  while (MotorX.distanceToGo() && MotorY.distanceToGo() &&
MotorZ.distanceToGo())
  {

    MotorX.run();
    MotorY.run();
    MotorZ.run();
  }
}
```

During calibration, the robot uses limit switches to find the 'home' position, which establishes a known starting point for accurate movements.

```
// main.cpp
while (1)
  {
    EtherCATCycleUpdate(); // Get new EtherCAT data
    bool DeltaPickerEnabled = EtherCATGetBIT(DeltaPickerEnableByte,
DeltaPickerEnableBit);

    if (DeltaPickerEnabled == false)
    {
      SECONDDeltaPickerCalibrate();
      Serial.println("Recalibration done");
      EtherCATCycleUpdate(); // Get new EtherCAT data
      delay(2000);
    }

    if (DeltaPickerEnabled == true) // If system is enabled
    {
      SetSpeed = EtherCATGetByte(DeltaPickerSpeedByte) & 0x7F;
      DeltaPickerUpdateMaxSpeed(SetSpeed);                  // Without
enable 0-100 speed
      DeltaPickerGetDesiredCoordinates(DesiredCoordinates); // get
desired coordinates from deltapicker

      if (DesiredCoordinates != CurrentCoordinates &&
DesiredCoordinateInScope(DesiredCoordinates))
      {
        // Move accel stepper
        MoveToCoordinates(CurrentCoordinates, DesiredCoordinates);
        CurrentCoordinates.x = DesiredCoordinates.x;
        CurrentCoordinates.y = DesiredCoordinates.y;
        CurrentCoordinates.z = DesiredCoordinates.z;
      }

      else
      {
        delay(10); // Wait 10 mili second before rechecking
      }
      // AdjustGripper();
    }

    else
    {
      // No enable
      delay(20); // Wait a second before rechecking if enable is out
    }
    UpdateEtherCAT_Deltapicker(CurrentCoordinates, SetSpeed,
DeltaPickerEnabled);
  }
```

Calibration is key to the robot's consistent performance, and the EtherCAT update loop maintains communication and control over the robot's actions.

7.3.3 User interface controls

User inputs from the GUI are converted into actions by the robot, ensuring the user's commands are executed correctly.

```
// MainController.cpp
// Jogging Code
```

```
#ifndef demo //Demo defenition in MainController.h
    if (m_JoggingControlI.JoggingStart == true)
    {
        m_DeltaPickerControl.EnableDeltaPicker = true;
        m_DeltaPickerControl.SpeedControl = 75;

        m_DeltaPickerControl.Xcoordinate = 0;
        m_DeltaPickerControl.Ycoordinate = 0;
        m_DeltaPickerControl.Zcoordinate = 0;
    }

    if (m_JoggingControlI.JoggingHome == true)
    {
        m_DeltaPickerControl.EnableDeltaPicker = false;
        m_DeltaPickerControl.SpeedControl = 75;

        m_DeltaPickerControl.Xcoordinate = 0;
        m_DeltaPickerControl.Ycoordinate = 0;
        m_DeltaPickerControl.Zcoordinate = 0;
        //m_DeltaPickerControl.EnableDeltaPicker = true;
    }

    m_JoggingControlO.CurrentCoordinatesX =
m_DeltaPickerControl.Xcoordinate;
    m_JoggingControlO.CurrentCoordinatesY =
m_DeltaPickerControl.Ycoordinate;
    m_JoggingControlO.CurrentCoordinatesZ =
m_DeltaPickerControl.Zcoordinate;
    m_DeltaPickerControl.SpeedControl = m_JoggingControlI.JoggingSpeed;

    if (m_JoggingControlI.JoggingZero == true)
    {
        m_DeltaPickerControl.Xcoordinate = 0;
        m_DeltaPickerControl.Ycoordinate = 0;
        m_DeltaPickerControl.Zcoordinate = -220;
    }

    if (m_JoggingControlI.GripperOpen == true)
    {
        m_DeltaPickerControl.PickerState = Open;
    }

    if (m_JoggingControlI.GripperClose == true)
    {
        m_DeltaPickerControl.PickerState = Closed;
    }

    // Button interpeter
    if (m_JoggingControlI.ModeSelect == 1)
    {
        if (m_JoggingControlI.JoggingLeft == true)
        {
            m_DeltaPickerControl.Xcoordinate += 10;
        }

        if (m_JoggingControlI.JoggingRight == true)
        {
            m_DeltaPickerControl.Xcoordinate -= 10;
        }

        if (m_JoggingControlI.JoggingForward == true)
```

```
        {
            m_DeltaPickerControl.Ycoordinate += 10;
        }

        if (m_JoggingControlI.JoggingBackward == true)
        {
            m_DeltaPickerControl.Ycoordinate -= 10;
        }

        if (m_JoggingControlI.JoggingUp == true)
        {
            m_DeltaPickerControl.Zcoordinate += 10;
        }

        if (m_JoggingControlI.JoggingDown == true)
        {
            m_DeltaPickerControl.Zcoordinate -= 10;
        }
    }

    // Text input interpeter
    if (m_JoggingControlI.ModeSelect == 2)
    {
        m_DeltaPickerControl.Xcoordinate = m_JoggingControlI.JoggingX;
        m_DeltaPickerControl.Ycoordinate = m_JoggingControlI.JoggingY;
        m_DeltaPickerControl.Zcoordinate = m_JoggingControlI.JoggingZ;
    }

    // Axes interpeter
    if (m_JoggingControlI.ModeSelect == 3)
    {
        m_DeltaPickerControl.Xcoordinate =
m_JoggingControlI.JoggingAxesX;
        m_DeltaPickerControl.Ycoordinate =
m_JoggingControlI.JoggingAxesY;
        m_DeltaPickerControl.Zcoordinate =
m_JoggingControlI.JoggingAxesZ;
    }

    // G-Code interpeter
    if (m_JoggingControlI.ModeSelect == 4)
    {

        char buf[301];
        memset(buf, 0, 300);
        buf[300] = '\0';

        memcpy(buf, m_Gcode.GCodeText, 300);//m_Gcode.GCodeText, 300);

        std::string abc(buf, 300);
        //m_Trace.Log(tlAlways, "%s", abc.c_str());

        GcodeConversion(abc);
    }
```

This segment shows how the system manages jogging inputs, interpreting user commands to move the robot incrementally in the chosen direction.

```
// MainController.cpp
std::string CMainController::extractCoordinateValue(const std::string&
line, char coordinate) {
```

```
    std::size_t coordIndex = line.find(coordinate);
    if (coordIndex != std::string::npos) {
        coordIndex++; // Move past the coordinate character (e.g., 'X')
        std::size_t spaceIndex = line.find(' ', coordIndex);
        if (spaceIndex != std::string::npos) {
            return line.substr(coordIndex, spaceIndex - coordIndex);
        }
        else {
            return line.substr(coordIndex); // In case it's the last
coordinate
        }
    }
    return ""; // Coordinate not found
}
```

```
// MainController.cpp
std::string CMainController::extractValue(const std::string& line, char
delimiter) {
    size_t spacePos = line.find(delimiter);
    if (spacePos != std::string::npos) {
        return line.substr(spacePos + 1);
    }
    return "";
}
```

```
// MainController.cpp
void CMainController::splitGCode(size_t start, std::string &gcode,
std::vector<GCodeLine> & relevantLines) {
    size_t end = gcode.find("\\n",start);
    //m_Trace.Log(tlAlways, "end %d",end  );
    if (end != std::string::npos) {
        std::string line = gcode.substr(start, end - start);
        //m_Trace.Log(tlAlways, "%s", line);

        // Check if the line is not empty before processing
        if (!line.empty()) {


            std::string commandType = "";

            if (line.find("G1") != std::string::npos) {
                commandType = "G1";
                m_Trace.Log(tlAlways, "G1");
            }
            else if (line.find("G4") != std::string::npos) {
                commandType = "G4";
                m_Trace.Log(tlAlways, "G4");
            }
            else if (line.find("G28") != std::string::npos) {
                commandType = "G28";
                m_Trace.Log(tlAlways, "G28");
            }
            else if (line.find("M04") != std::string::npos) {
                commandType = "M04";
                m_Trace.Log(tlAlways, "M04");
            }
            else if (line.find("M05") != std::string::npos) {
                commandType = "M05";
                m_Trace.Log(tlAlways, "M05");
```

```
            }
            else if (line.find("M204") != std::string::npos) {
                commandType = "M204";
                m_Trace.Log(tlAlways, "M204");
            }

            relevantLines.push_back({ line, commandType });
        }

        splitGCode(end + 1, gcode, relevantLines); // Move to the next
line
    }
    else if (start < gcode.length()) {
        // Process the remaining part of the string if it's not empty
        std::string line = gcode.substr(start);
        if (!line.empty()) {
            relevantLines.push_back({ line, "" }); // Last part
        }
    }
};
```

```
// MainController.cpp
// Recursive lambda to process G-code lines
void CMainController::processGCodeLines(size_t index,
std::vector<GCodeLine>& relevantLines) {
    if (index < relevantLines.size()) {
        const auto& gcodeLine = relevantLines[index];

        // Processing logic here...
        if (gcodeLine.commandType == "G1") {
            m_DeltaPickerControl.EnableDeltaPicker = true;
            std::string xValue = extractCoordinateValue(gcodeLine.line,
'X');
            std::string yValue = extractCoordinateValue(gcodeLine.line,
'Y');
            std::string zValue = extractCoordinateValue(gcodeLine.line,
'Z');

            m_DeltaPickerControl.Xcoordinate = std::atoi(xValue.c_str());
            m_DeltaPickerControl.Ycoordinate = std::atoi(yValue.c_str());
            m_DeltaPickerControl.Zcoordinate = std::atoi(zValue.c_str());
        }
        else if (gcodeLine.commandType == "G4") {
            m_Trace.Log(tlAlways, "G4 - Delay");
            m_DeltaPickerControl.EnableDeltaPicker = true;
        }
        else if (gcodeLine.commandType == "G28") {
            m_DeltaPickerControl.EnableDeltaPicker = false;
            m_DeltaPickerControl.SpeedControl = 75;

            m_DeltaPickerControl.Xcoordinate = 0;
            m_DeltaPickerControl.Ycoordinate = 0;
            m_DeltaPickerControl.Zcoordinate = 0;
        }
        else if (gcodeLine.commandType == "M04") {
            m_DeltaPickerControl.EnableDeltaPicker = true;
            m_DeltaPickerControl.EnableGripper = Closed;
        }
        else if (gcodeLine.commandType == "M05") {
            m_DeltaPickerControl.EnableDeltaPicker = true;
```

```
            m_DeltaPickerControl.EnableGripper = Open;
        }
        else if (gcodeLine.commandType == "M204") {
            m_DeltaPickerControl.EnableDeltaPicker = true;
            std::string speedValue = extractValue(gcodeLine.line, ' ');
            m_DeltaPickerControl.SpeedControl =
std::atoi(speedValue.c_str());
        }

        processGCodeLines(index + 1,relevantLines);
    }
}
```

```
// MainController.cpp
void CMainController::GcodeConversion(std::string gcode) {
    std::vector<GCodeLine> relevantLines;
    size_t start = 0;

    splitGCode(start,gcode,relevantLines);
    processGCodeLines(0,relevantLines);
}
```

The G-code interpreter takes a string of G-code and translates it into movements and actions, such as moving to coordinates or operating the gripper.

7.3.4 User interface PLC handling

All interactive elements, such as buttons, sliders, and dropdown menus, are meticulously integrated within the system. They are connected to corresponding variables through a mapping process that bridges the TwinCAT Human-Machine Interface (HMI) with the underlying delta picker project written in C++.

The HMI operates on principles like PLC (Programmable Logic Controller) programming, ensuring a robust and reliable user interaction experience. It is essential that these user interface components are managed effectively to ensure accurate and responsive operation.

To achieve this, we have implemented a logic control mechanism like a flip-flop. This mechanism is crucial for monitoring the state of each control element, determining whether a button has been pressed or released. The system performs this check at a rapid pace, every 5 milliseconds, to maintain real-time responsiveness.

Here is a conceptual example of how this works:

```
JoggingStart0 AT %Q* : BOOL := FALSE;
JoggingStartI AT %I* : BOOL := FALSE;

ToggleVar: BOOL := FALSE;
Timer1: TON;
FlipFlopStart: BOOL; // Flip-flops to control the outputs
```

```
// Initialize the timer
Timer1(IN := TRUE, PT := T#3MS);

// Check if JoggingStartI is pressed
IF JoggingStartI THEN
    // Start the timer when JoggingStartI is pressed
    Timer1(IN := TRUE);
```

```
    // Set the flip-flop to TRUE when JoggingStartI is pressed
    FlipFlopStart := TRUE;
END_IF;

// Toggle JoggingStartO based on the flip-flop state
IF FlipFlopStart THEN
    JoggingStartO := TRUE;
ELSE
    JoggingStartO := FALSE;
END_IF;

// Check if the timer has elapsed to reset the flip-flops and turn off
the outputs
IF Timer1.Q THEN
    Timer1(IN := FALSE);
    FlipFlopStart := FALSE;
END_IF;
```

This code mimics the behaviour of a flip-flop used in PLCs to ensure that button presses result in single actions rather than repeated triggers.

8. Testing

8.1 Test scenario 1: Initial Setup and Functionality Check (F1)

Supplies:

- The Deltra robot

Evaluated functional requirements:

- F1

Test performed:

- 28-12-2023, by Thom Peters and Ruben Konings

Steps:

01	**Turn on the Delta Robot and TwinCAT program.**
	Expectation: System starts without errors, and the GUI appears.
	Passed

Table 1 Test 1.

8.2 Test scenario 2: Integration with Qube Platform (F2)

Supplies:

- The Deltra robot
- The Qube Platform

Evaluated functional requirements:

- F2

Test performed:

- 28-12-2023, by Thom Peters and Ruben Konings

Steps:

01	**Connect the Delta Robot to the Qube platform.**
	Expectation: Seamless integration and compatibility.
	Passed

Table 2 Test 2.

8.3 Test scenario 3: Plug and Play Functionality (F3)

- The Deltra robot
- The Qube Platform

Evaluated functional requirements:

- F3

Test performed:

- 28-12-2023, by Thom Peters and Ruben Konings

Steps:

01	**Evaluate the setup and configuration process.**
	Expectation: User-friendly setup with straightforward configuration.
	Passed

Table 3 Test 3.

8.4 Test scenario 4: User Interface Control on PC (F4)

- The Deltra robot
- The Qube Platform

Evaluated functional requirements:

- F4

Test performed:

- 28-12-2023, by Thom Peters and Ruben Konings

Steps:

01	**Interact with the Delta Robot using the PC user interface.**
	Perform tasks using manual jogging and automated movement programs.
	Expectation: Effective control of the robot via PC interface.
	Passed

Table 4 Test 4.

8.5 Test scenario 5: Pendant Compatibility (F5)

- The Deltra robot
- The Qube Platform
- Pendant

Evaluated functional requirements:

- F5

Test performed:

- 28-12-2023, by Thom Peters and Ruben Konings

Steps:

01	**Evaluate the robot with a pendant control device.**
	Expectation: Intuitive control and consistent UI with the PC controls.
	Failed

Table 5 Test 5.

8.6 Test scenario 6: Advanced Safety Features (F6)

- The Deltra robot
- The Qube Platform

Evaluated functional requirements:

- F6

Test performed:

- 28-12-2023, by Thom Peters and Ruben Konings

Steps:

01	**Simulate scenarios to evaluate collision detection and emergency stop.**
	Expectation: Robot responds appropriately to collision risks and can be suddenly stopped in emergencies.
	Passed, except for the collision detection

Table 6 Test 6.

8.7 Test scenario 7: Feature Evaluation (F7)

- The Deltra robot
- The Qube Platform

Evaluated functional requirements:

- F7

Test performed:

- 28-12-2023, by Thom Peters and Ruben Konings

Steps:

01	**Assess the impact of additional features on the project's complexity and benefits.**
	Expectation: Features contribute positively without unnecessary complexity.
	Passed

Table 7 Test 7.

9. Conclusion and recommendations

9.1 Project Overview

This project aimed to enhance the Delta Robot's functionality and usability. Our objectives included improving mechanical components, streamlining the control interface, and integrating advanced software solutions.

9.2 Achievements

- **Mechanical improvements:** We successfully redesigned key components for greater durability and efficiency.
- **Software enhancements:** The development of a more intuitive GUI and the integration of G-Code support marked significant progress in software usability.
- **Comparative analysis:** Our robot now closely matches the capabilities of commercial counterparts in several aspects.

9.3 Challenges

- Precision and control: Despite improvements, achieving high precision in movements remains a challenge.
- Integration issues: Combining hardware and software elements seamlessly was more complex than anticipated.

9.4 Future recommendations

1. **Enhanced sensor integration:** Implementing more sophisticated sensors could improve accuracy and adaptability.
2. **Machine learning integration:** Utilizing AI for task automation could expand the robot's capabilities.
3. **Hardware upgrades:** Continued refinement of the stepper motors and PCB design is crucial for performance enhancement.
4. **Expand applications:** Exploring uses in diverse fields like healthcare, manufacturing, and consumer goods could broaden the robot's impact.
5. **Community involvement:** Open sourcing some software components might accelerate development through community contributions.

9.5 Final thoughts

The advancements made in this project pave the way for the Delta Robot to become a more accessible and versatile tool in robotics, opening doors for broader applications and innovations.

10. Bibliography

Arends, H. (2022). *Productrapport.* Arnhem: Embedded Systems Engineering.

Onbekend. (2022, Juni 6). *MoSCoW method*. Retrieved from Wikipedia: https://en.wikipedia.org/w/index.php?title=MoSCoW_method&oldid=1091822315

Onbekend. (2022, Mei 25). *SMART criteria*. Retrieved from Wikipedia: https://en.wikipedia.org/w/index.php?title=SMART_criteria&oldid=1089766780

11. Appendices

11.1 Blog

Title: Navigating Ongoing Challenges in Our Delta Robot Project: A New Approach to Debugging

Introduction

Welcome to our latest blog post! We are on a journey filled with continuous learning and adaptation, diving into innovative methods for debugging and enhancing both the hardware and the graphical user interface (GUI) of our delta robot. This post will give you a glimpse into the latest developments and how we are tackling these evolving challenges together.

Continuing the Hardware Debugging Saga

Our focus from the start was pinpointed on identifying and fixing the robot's movement inaccuracies. Despite our combined efforts, the robot was not performing up to the mark. This led us to brainstorm alternative strategies for debugging. We are now delving into the various aspects of the hardware, adopting a more comprehensive approach to uncover potential issues.

Adopting a New Route for Debugging

In our quest for a seamless solution, we decided to embark on a different debugging path. This new direction involves a comprehensive examination of both the mechanical and electronic components of our robot. We are particularly keen on this part, as it requires a deep dive into the interplay between software commands and hardware responses. Our goal is to ensure the system operates flawlessly, with each part working in harmony.

Progress on the GUI Front

While one of us tackles hardware troubleshooting, the other is spearheading the GUI refinement. The aim here is to develop an interface that is not just user-friendly but also capable of handling complex commands with ease. Our combined insights have been invaluable, especially regarding the GUI's interaction with the hardware, helping us ensure a cohesive experience for our users.

Conclusion

As we navigate through these challenges, our project is constantly evolving. Each hurdle we encounter teaches us something new, fuelling our creativity and willingness to try innovative approaches. Currently, our focus is firmly set on a comprehensive debugging strategy and meticulous GUI refinement. We believe these efforts will significantly shape the future of our delta robot, promising enhanced performance and a superior user experience.

Stay tuned for more updates as we continue this exciting journey. Thanks for joining us on this adventure, and we hope our experiences inspire your own projects and challenges!

Thom and Ruben

ESE-2a-n

11.2 User manual

11.2.1 Introduction

Welcome to the Delta Robot User Manual. This guide will help you set up and operate your Delta Robot efficiently.

11.2.2 Setup

Here is how you setup the delta robot before using:

- **Unpacking**: Carefully remove the robot from its packaging.
- **Power connection**: Connect the robot to the power socket of the Qube using the rounded power cable.
- **Communication connection**: Connect the robot to the ethernet cable port of the Qube.
- **Initial inspection**: Check for any visible damage or loose parts.

11.2.3 Operating instructions

Here is how you need to operate the delta robot:

Power on:

1. Turn on the robot using the main power button.
2. Release the emergency stop and press the blue button. A light indicates it's powered on.

Software Interface:

1. Open the robot's GUI on your computer.
2. Start TwinCAT by clicking the stairs icon on the top left, accepting all prompts.
3. Open a web browser and navigate to http://localhost:1010/.

Login:

- Enter the password: L@brobot01.

Jogging:

1. Select a mode in the "Jogging" tab: button, typing, slider, or GCode.
2. Use the chosen mode for operation.
3. Press "Home" for calibration.
4. "Start" moves the robot to coordinates X0 Y0 Z-220.

GCode Operations:

- **Linear Movements:** Use commands like **G1 X100 Y100 Z-250**.
- **Delays**: Create pauses with **G4 3**.
- **Homing**: Return to home position with **G28**.
- **Gripper Control**: Open/close gripper with **M04** and **M05.**
- **Speed Adjustment**: Set speed with **M204 75.**

Basic Operations:

- Use GUI for simple pick and place tasks.

Advanced Features (for experienced users):

- Access additional settings for enhanced precision and custom movement patterns.

11.2.4 Troubleshooting

Common issues and quick fixes:

- **Connectivity issues**: Ensure all cables are securely connected.
- **Movement problems**: Restart the robot and recalibrate if necessary.
- **Software errors**: Restart the software or contact the developers.

11.2.5 Care and Maintenance

Regularly check for loose screws/wires and clean the robot to prevent dust accumulation.

11.2.6 Contact Info

For further assistance, contact t.peters13@student.han.nl or rl.konings@student.han.nl.

11.3 Planning

Can be found in: \DeltaRobotThomandRuben\Documentation.

11.4 Plan of approach

Can be found in: \DeltaRobotThomandRuben\Documentation.

11.5 Poster

Can be found in: \DeltaRobotThomandRuben\Documentation.

11.6 Personal evaluation

11.6.1 Thom Peters

11.6.1.1 Introduction

As part of the team responsible for enhancing the Delta Robot, my journey through this project has been both challenging and enlightening. The focus of our work was on addressing mechanical and software issues and developing a more intuitive Graphical User Interface (GUI).

11.6.1.2 Reflection on personal learning goals

At the outset, my personal learning goals were aligned with acquiring deeper technical skills in robotics, particularly in PCB design and software integration. Working closely with Ruben Konings, my objectives extended to improving the robot's mechanical reliability and user interaction experience.

11.6.1.3 Contribution to the project

My primary contributions included:

- Debugging and enhancing the robot's hardware, particularly focusing on the stepper motors and PCB design.
- Collaborating in the development of the GUI, ensuring it was user-friendly and capable of handling complex commands.
- Conducting a comparative study with a commercially available delta robot, which provided valuable insights for improvements.

11.6.1.4 Reflection on the collaboration process

The collaboration process was marked by a constant exchange of ideas and solutions. The challenges we faced, especially in debugging and integrating the GUI with the robot's hardware, necessitated a collaborative approach. Our complementary skills in hardware and software development were instrumental in achieving the project goals.

11.6.1.5 Conclusion

This project has been a significant learning experience, enhancing my skills in robotics, team collaboration, and problem-solving. The challenges encountered have prepared me for future endeavours in the field of robotics and automation.

11.6.2 Ruben Konings

11.6.2.1 Introduction

My role in enhancing the Delta Robot has been a blend of technical challenges and learning. I focused primarily on hardware issues, specifically in PCB design and motor testing, to improve the robot's overall functionality.

11.6.2.2 Reflection on personal learning goals

The success of this project was largely due to the teamwork, particularly my collaboration with Thom Peeters. Working with Thom, we combined our strengths in hardware and design. This teamwork was essential in seamlessly integrating the PCB and motor testing with the overall system. Our shared efforts in troubleshooting and refining the hardware components exemplified the value of collaboration in solving complex technical challenges. This experience has reinforced the importance of teamwork in achieving project goals in the field of robotics.

11.6.2.3 Contribution to the project

My primary contributions included:

- Debugging and enhancing the robot's hardware, particularly focusing on the stepper motors and PCB design.
- Designing and implementing a robust PCB, tailored specifically for the robot's needs.
- Conducting detailed testing and troubleshooting of the stepper motors, ensuring optimal performance and integration.

11.6.2.4 Reflection on the collaboration process

The project demanded close collaboration, especially in aligning the hardware developments with the overall project objectives. My focus on hardware testing and PCB design required constant communication and problem-solving within the team.

11.6.2.5 Conclusion

Working on this robot project has been a great learning experience for me. I improved a lot in PCB design and testing motors. I also learned how to solve problems better and work with others on technical projects. This project has prepared me well for future work in robotics and technology.

11.7 Code

Can be found in \DeltaRobotThomandRuben\Software and on GitHub.

11.8 Videoclip

Can be found in: \DeltaRobotThomandRuben\Documentation.