University of Southern Denmark

Robot Systems Engineering

Start Date: 28. Februar 2023

Due Date: 26. May 2023

Group Number: 5

Instructor: Norbert Krüger

Basic Robot Automation

The Gripper

Submitted By:

Alan Rashid, 07-04-1998, *alras22@student.sdu.dk*

Anas Butt Hussain, 15-12-2003, *anhus22@student.sdu.dk*

Jakub Hubert Rudowski, 01-02-2001, *jarud22@student.sdu.dk*

Patrick Ørsted Povey Andersen, 10-01-2000, *paand22@student.sdu.dk*

Peter Jøker Trachsel, 03-09-2001, *petra22@student.sdu.dk*

Robin Hansen, 08-11-1995, *rohan22@student.sdu.dk*

Thomas Korsgaard Vilholm, 18-08-1997, *thvil22@student.sdu.dk*

Contents

1 Introduction

The goal of this project is to program and design a robot gripper capable of sorting three bricks of different sizes. The project integrates programming, interface/GUI development, and hardware design to create a fully functional robotic system that automates the brick sorting process.

The interface development component entails establishing connections between the robot and the Atmega644 board. This interface seeks to enable seamless communication and control over the robot's movements and tasks.

Furthermore, the project focuses on developing assembly code for the ATmega644 board to enable gripper opening and closing functionality. Precise movements are essential for successfully handling and placing the bricks.

By utilizing the capabilities of the UR5 robot and the ATmega644 board, the project seeks to optimize the sorting process. All programming and connections between the GUI, ATmega644 and the UR5 robot are done in C++.

There were 5 requirements/tasks to fulfill in this project. They were the following:

- Task 1: Develop a 3D model of a gripper that can be printed or otherwise constrcuted.
- Task 2: Use the electronics given to control the gripper.
- Task 3: Develop C++ program to send commands and control the UR5 robot, use Modbus TCP and RS323 protocols for communication, and to develop a GUI.
- Task 4: Develop Software for the AtMega644PA board to control the gripper, and to send/receive information through the RS232 protocol.
- Task 5: Test and document performance of the gripper.

The requirements/tasks, were distributed as follows:

- Task 1: Robin, Peter.
- Task 2: Robin, Thomas, Alan.
- Task 3: Jakub, Patrick, Anas.
- Task 4: Robin, Thomas.
- Task 5: Thomas, Alan, Patrick.

2 Project Description

The primary objective of this project is to develop a comprehensive solution that incorporates programming, design, and graphical user interface (GUI) elements to achieve efficient and accurate sorting of three bricks with varying lengths using a UR5 robot arm. The solution encompasses the following key components:

1. Robot Arm Programming: Implementation of C++ enables the UR5 robot arm to perform the sorting task. This includes incorporating a timer in the programming code to accurately determine the length of each brick.
2. Length Determination and Sorting: Utilizing the implemented timer and programming logic, the robot arm identifies the length of each brick. Once the length is determined, the robot arm proceeds to pick up the brick and place it in the corresponding box. This sorting process ensures that each brick is correctly sorted based on its length.
3. Gripper Design and Functionality: Designing a gripper mechanism that allows the robot arm to securely grip and release the bricks during the sorting process. The gripper design should facilitate precise handling of the bricks to ensure accurate placement in the designated boxes.
4. Graphical User Interface (GUI) Development: Creating a user-friendly GUI using C++ to send commands to the UR5 robot arm. Connection is made via. the MODBUS library. With this GUI, commands can be The GUI should include essential buttons, such as a ”Pick” button to initiate brick retrieval from a predefined location and a ”Place” button to select the desired box for placement. The GUI is responsible for visualizing brick count.

By addressing these objectives, the project aims to develop a fully functional robot arm using the UR5 robot, incorporating programming, design, and GUI elements.

3 Graphical User Interface

The graphical user interface (GUI) is made for allowing simple and straightforward control of the robot arm system and the gripper. The GUI design prioritizes simplicity and ease of use, enabling users to initiate actions with minimal effort. This section includes an insight in how the graphical user interface works and the connection between Atmega and the UR5 robot with the computer. The code referred to in this section is attached in the appendix, it will be referred to exactly as the name of the file.

3.1 Design of the Interface

The GUI has been programmed in C++ by using the wxWidgets library (see [6]), which can be used to write GUI applications across multiple platforms. The design of the GUI had many different iterations throughout the projects timeline. At first the GUI only consisted of a simple wxFrame (see sections 3.1.1 and 3.1.2) with a wxPanel (see section 3.1.3) and a wxButton (see section 3.1.4) called `Pick and Place`. When this button was pressed the robot would sort the brick into its designated location (see fig. 1). But this was not automated and therefore inconvenient, because the button had to be pressed every time it had sorted a brick.

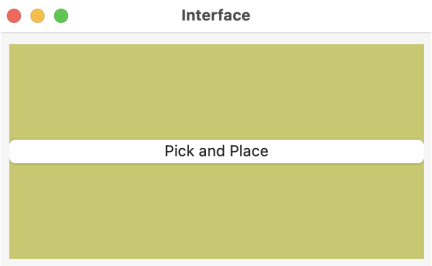


Figure 1: A screenshot of the first GUI iteration

The next iteration would then make the sorting process automated when the button was pressed. But the GUI would then also need a new wxPanel and wxButton labeled `Stop` (see fig. 2), that could be pressed to stop the process. This would give the user control over the operation without closing down the program.

Throughout various tests of the second iteration, the button `Open gripper` was made, so that the user would be able to open up the gripper fingers, in case of a failure when sorting the bricks. When the fingers were not fully open when picking up a brick, the system would sort the brick incorrectly, because the sorting system is time based (see

section 6.1).

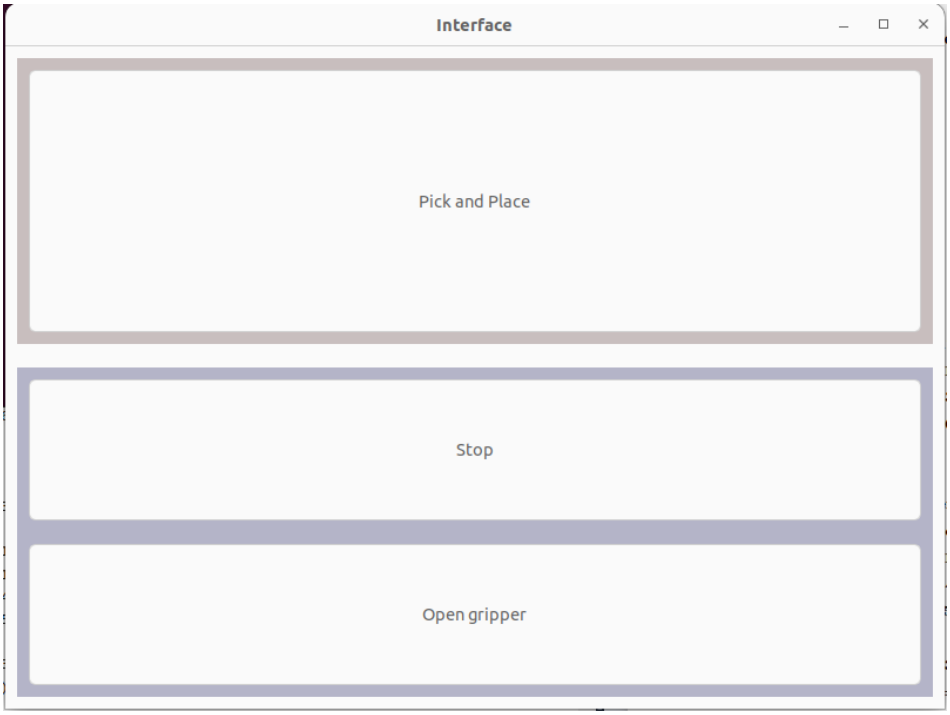


Figure 2: A screenshot of the second GUI iteration

Now with these improvements the user had simple and straightforward control of the robot arm and the gripper with the GUI. But there was a lack of visual data for how many of the different size bricks had been sorted and if there had been any failures. The count of the bricks and failures was added so that the user would have to use minimal effort to keep count and track of the sorted bricks (see fig. 3).

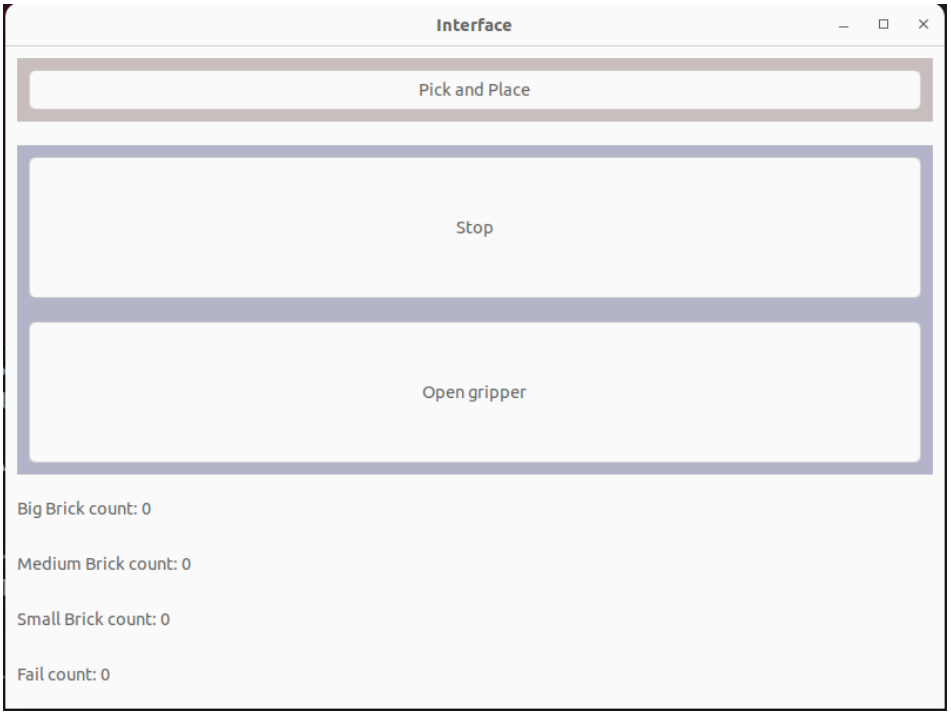


Figure 3: A screenshot of the completed GUI

3.1.1 wxWidgets Application

Making a wxWidgets application required creating a class from wxApp which is a class in the wxWidgets framework that serves as the foundation of your application. The Main application window was made by overriding a function called `OnInit()` and implementing it in the main.cpp file (see main.cpp, line 5-11). OnInit is called at the beginning of the application's execution as part of the initialization process.

3.1.2 wxFrame

The wxFrame is a class that represents the main frame of the GUI. To customize its behavior and add functionality to it, a derived class was made from wxFrame `MyFrame` (See frame.h line 9-47). To add objects such as a wxPanel or a wxButton, an instance of the derived wxFrame class `MyFrame` was made (see frame.cpp line 4-43).

3.1.3 wxPanel

The wxPanel class represented a graphical container in the wxFrame `MyFrame` and are often a child window to the frame (see frame.cpp line 8). To customize its behavior and to add methods, a class derived from wxPanel (MyPanel) (See panel.h line 10-49) was made. The wxPanel was used to contain/organize controls such as wxButtons and text (see frame.cpp line 12). In the GUI the wxPanel was used to contain wxButtons (see frame.cpp line 12). The GUI all so uses an instance of the `MyPanel` class for

functions like the click event from the wxButtons to manipulate the robot's gripper and control the robots movements(see panel.cpp line 78-88).

3.1.4 wxButton

The wxButton class represented a clickable button. When clicked, it generated an event (such as a wxCommandEvent) (see panel.cpp line 180-253) that could be manipulated to perform a specific action or series of instructions.

3.2 UR5 Robot Code

The code which is responsible for UR5 robot movement can be programmed from the UR5 Teach Pendant (see fig. 4). In the program there are two subprograms: `"Before Start"` and `"Robot Program"`.

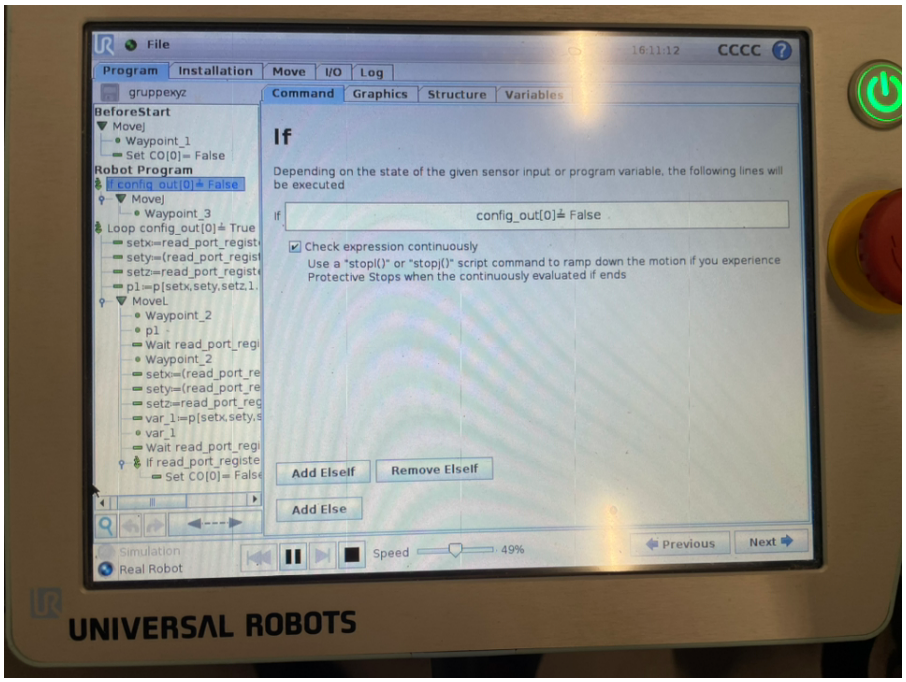


Figure 4: A picture of the Teach Pendant on which the robot code is written

The ”before start” program is responsible for moving the UR5 robot to a hard-coded location so that it is easier to set and configure the gripper. The program starts by moving the arm to a predefined start-position which stays there until `Config_Out` is set to 1. When the config is set to true, the program reads ports: 128, 129, 130 in the setx, sety and setz variables. A problem was found during coding, it was found that when writing coordinates in C++ code, the maxiumum value that could be sent to the UR5 robot was 255. A number was added to compensate for that in these variables. The coordinates written from the C++ in registers are also divided by a thousand. The

reason for this was that the coordinates had to be in meters. These coordinates were combined into a point which was named `p1`, and this point is the pick-up location point. A `Waypoint_2` has been made to make the gripper first hover over the pick-up place. The gripper then moves to p1 and waits for the register 135 to be equal to one (look section 3.3). It again hovers and waits for the next register. This time moving to the drop-off location according to the length of the respective brick (look section 3.3). This location is saved in `var_1` point. The arm moves to this point and waits for register 136 to be equal 1, which occurs when the gripper is fully open.

3.3 Connection between hardware and software

The connection was made by using two libraries, serialib (see [4]) and libmodbus (see [3]). The two libraries combined, were integrated into the back-end, so signals could be initialized fulfilling the goal of the robotic system.

3.3.1 Serialib

The serialib library (in the code serialib.h, serialib.cpp from [4]) was a helper tool to connect to the ATmega board, which is in control of the gripper mechanism. The goal of the ATmega class in the code (atmega.h and atmega.cpp) was to connect the board and read/write values from it. The assembly program could then send and receive information in form of a number. These numbers act as instructions for the gripper (section 5.1).

The ATmega class connects the whole Interface with the ATmega board by using the following functions. Simply described here is what they do:

Function:	Functionality:
ATmega()	Constructor for the ATmega class. Works as a device creator in the C++ program.
_openDevice()	Connects the device by using the serialib library. The two inputs in the function are a USB port in which the ATmega board is connected to and a baudrate.
write_close_Grip()	Signalizes that the gripper has to close. It uses the serialib library function writeBytes() to send 0xf0 which is 240 in decimal. When the ATmega board receives this number the gripper will close (see section 5.1).
write_open_Grip()	It has the same purpose as write_close_Grip() but it sends 0x0f which is 15 in decimal. The ATmega then knows it has to open the gripper (see section 5.1).
read_Value()	The assembly code in the ATmega board is programmed in a way which shows when the motor experiences a current load. This value is received in the C++ Inteface by using serialib library function readBytes(). The function works as a gate for the UR5 robot and C++ code to know when the gripper has gripped an object, in this case a brick. The number which ATmega board sends is 69 when it detects an object was gripped.
stop_Value()	Functionality is the same as for read_Value(), but it receives 42 in decimal when the gripper is fully open.
getValue()	Function accesses the value variable.
getTime()	Function accesses the time variable.

The ATmega class also detects which brick the ”fingers” grip. The detection method used is a C++ timer with chrono library. It determines the wait time of the gripper from closing the fingers to getting a hold of the brick. Experiments were made and the seconds for each brick were set, by analyzing the observations (look fig. 9). The timer starts when the function `write_close_Grip()` is used and ends when `read_Value()` signalizes that the brick is fully gripped. The UR5 robot would not work if the whole system was not connected, that is why the libmodbus C++ library was a necessity for the integration of software and hardware.

3.3.2 Libmodbus

The libmodbus library allows C++ code to access UR5 robots' data via. an Ethernet cable. Functions from the library are used in the MyPanel class. Look at the example

function below:

```
void MyPanel::setX(uint16_t val)
{

    modbus_t *ur5 = modbus_new_tcp("192.168.100.11", 502);
    modbus_connect(ur5);
    int reg = modbus_write_register(ur5, 128, val);
    if (reg == -1)
    {
        std::cout << ("Modbus: Counldn't set x!") << std::endl;
    }
}
```

Figure 5: Custom made function by using libmodbus library

As seen from fig. 5, libmobus functions `modbus_new_tcp()`, `modbus_connect()` and `modbus_write_register()` were used to access and write registers in the UR5 robot (For register look under sec 3.3). These registers were used to manipulate robots functionality. As for an example if the robot detected the largest brick, the code set the registers to drop it off in a area which was set for that type of brick. As seen in the C++ code (look panel.h and panel.cpp), there were several custom made functions which use libmodbus. The three main functions were `OnClick()`, `setPickupLocation()` and `setDropoffLocation()`. The `setPickupLocation()` set four registers of the UR5 robot so that the robot moved to the PickUp position. While the `setDropoffLocation()` set the height of the drop to 200, which is 0.2 meters in z-height for the robot. Then used the Atmega class function `getTime()` to find out the type of the gripped brick. The seconds from the timer decided where this brick should be put (look panel.cpp lines [28 - 60]), if the gripper was fully closed then the `_failCount` incremented and the UR5 robot stopped executing its sorting task. The last function was `OnClick()` which was a wxWidgets function in the MyPanel class which corresponded to the clicking of the buttons.

3.3.3 Functionality of Interface Buttons

There are several micro tasks to be made by the robot. It includes moving, picking and dropping the brick, and again returning to the pick up place. These instructions are coded in the `"Pick and Place"` button. When the button is pressed (look lines 185 - 248 in panel.cpp, code is described by usage of comments). The main goal of `"Pick and Place"` button was to set the Pickup location which was always the same. Then to signal the gripper to close and wait for it to grip a brick, the `Wait_Close` was then

set to signal UR5 robot that robot can move to the drop off location based on the timer. Program gave the signal for the gripper to open and the brick fell into the designated area for its type. When the gripper was fully open, the program set `Wait_Open` and the UR5 robot repeated its task encoded in the `"Pick and Place"` button.
On the other hand this task could be manually stopped by pressing the `"Stop"` button. The functionality of the `"Stop"` button was to change the `_stopFlag` to a true statement (`_stopFlag` is false by default). The `"Open gripper"` button was made in case of the gripper to be stuck in a closing position. Whole process intended that the gripping mechanism must be adequate to what the robot task was. In the next section the design of the gripper and functionality is discussed.

4 Gripper

The design of the gripper focuses on its closing mechanism, which is essential to its function. The closing mechanism consists of a motor-driven bolt attached to the shaft. The motor is connected to a static finger, and when activated, the bolt rotates inside a nut placed in a moving finger (see figure Figure 6).

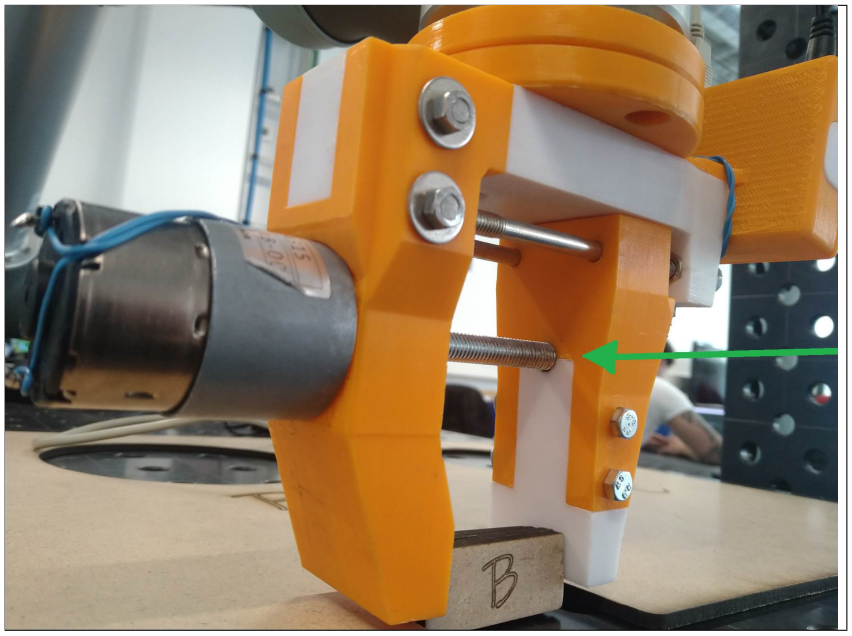


Figure 6: Close up of the gripper highlighting the closing mechanism. The green arrow indicates where the nut pulling the moving finger is. The nut is of course not actually visible, since it is inside the moving finger, which is the finger opposite to the finger that has the motor attached to it

The gripper is designed to grab blocks, lift them, and place them at a specified location. To achieve this, the gripper features two fingers. One finger is rigidly attached to the base, while the other finger is movable. The movement of the movable finger is controlled by a motor-driven threaded rod and nut system. The motor rotates a threaded rod, which engages with the nut embedded in the finger. As the rod spins, the nut moves closer or farther away from the fixed finger, depending on the direction of the motor's rotation.

Additionally, the gripper includes an enclosure for the ATMega644PA chip, which connects to the motor. The gripper is primarily constructed using PLA material, commonly used in Ultimaker S3 printers. The printed flanges are used to connect the gripper to the robot, while non-printed components such as bolts, nuts, and washers are used for assembly.

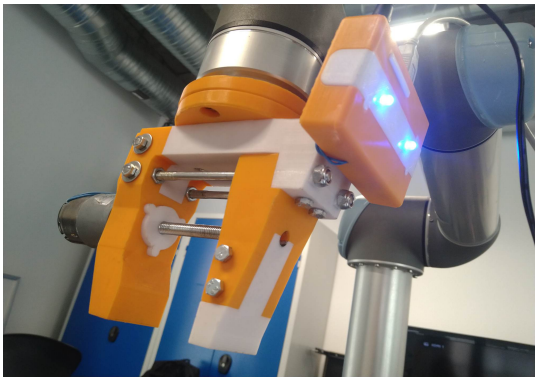
4.1 Design-Comments

The gripper design was created using Autodesk Inventor software. The overall infill of the gripper is set to 30

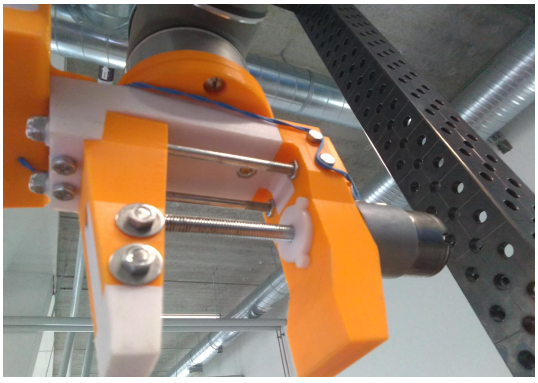
The design deviated from the original plan, which aimed to incorporate a pinion and rack system with a higher translation to motor revolution ratio. However, due to a reading error with the current sensor, the design was changed to the bolt and nut mechanism.

Despite the grippers design deviation and sub optimal specifications, like its unnecessarily slow operating speed, it is still considered successful for the project's requirements, being the automatic sorting of bricks of different sizes.

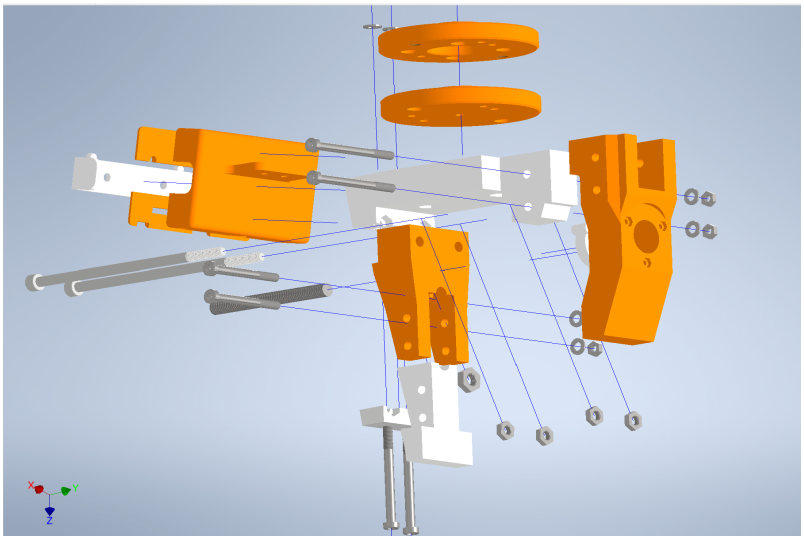
Some more pictures and an exploded view can be seen below (see figure fig. 7).



(a) The gripper featuring the ATMega board casing: Printer method



(b) The gripper seen from the opposite side



(c) An exploded view of the gripper in AutoDesk Inventor

Figure 7: Gripper showcase

5 ATMega-Board

In the ATMega-Board, there are various ICs and peripherals, the relevant ones are listed below:

- ATMega644PA chip ([5] Datasheet)
- H-bridge IC (IFX9201SG) ([1] Datasheet)
- Current sensor IC (INA240A3PWR) ([2] Datasheet)
- USART port

5.1 Program

Fundamentally, described in a stepwise fashion, the board has been made to work in the following manner:

- Initialize ATMega644PA chip.
- Standby mode: Wait for a receive complete interrupt.
- On interrupt, read data register and stay in ”receive mode” until correct close/open command has been received.
- Wait for the motor to stabilize it’s speed.
- Constantly read the current sensor output connected to the motor.
- When threshold voltage for closing/opening is reached, send ”closing done”/”opening done” messsage over the USART port. This ”closing done”/”opening done” message is then used as described in subsection 3.3.1.
- Re-enter ”Standby mode” where the motor is turned off.

The chip was programmed in the Microchip Studio IDE, in the ”Assembly” programming language. A detailed documentation for both the code and the board, is found in the Appendix.

6 Data Processing

This section focuses on the conducted experiments with the robot to evaluate its performance and optimize its functionality. Two key experiments were performed, targeting different aspects of the robot's capabilities.

6.1 Experiment: Gripping Timer

In this experiment, the robot utilized a gripping timer to determine the size of the handled bricks. The objective was to identify the appropriate duration, measured in seconds, for the gripper to successfully pick up bricks of three different sizes. There was an expectation of approximately 8 seconds to grip the largest brick, 14 seconds for the medium-size brick and 22 seconds for the smallest brick. Fine-tuning the gripping timer resulted in more accurate and efficient handling of the bricks, enabling precise sorting based on size.

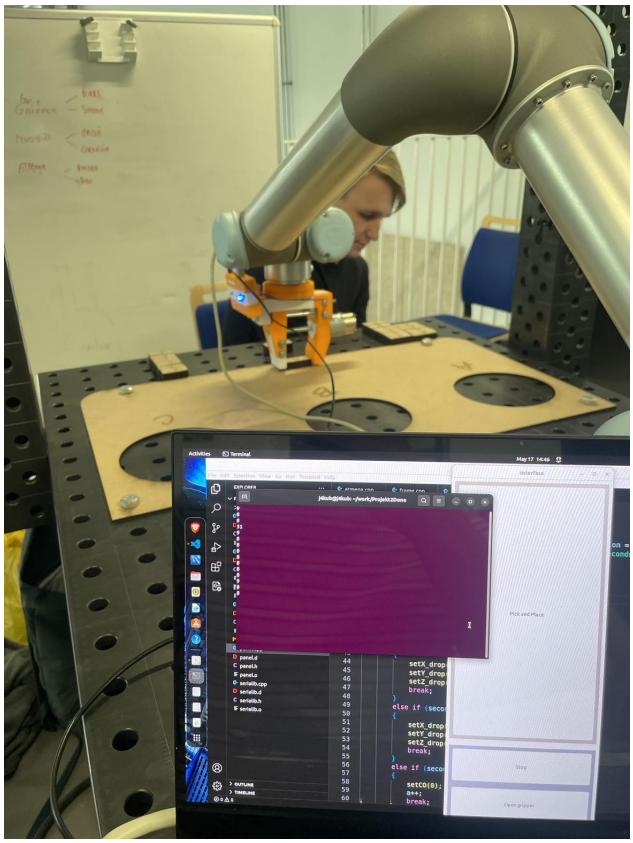


Figure 8: A photo of the experiment process

The experiment was based on quantitative measurements from the data which was obtained by continuously running the program.

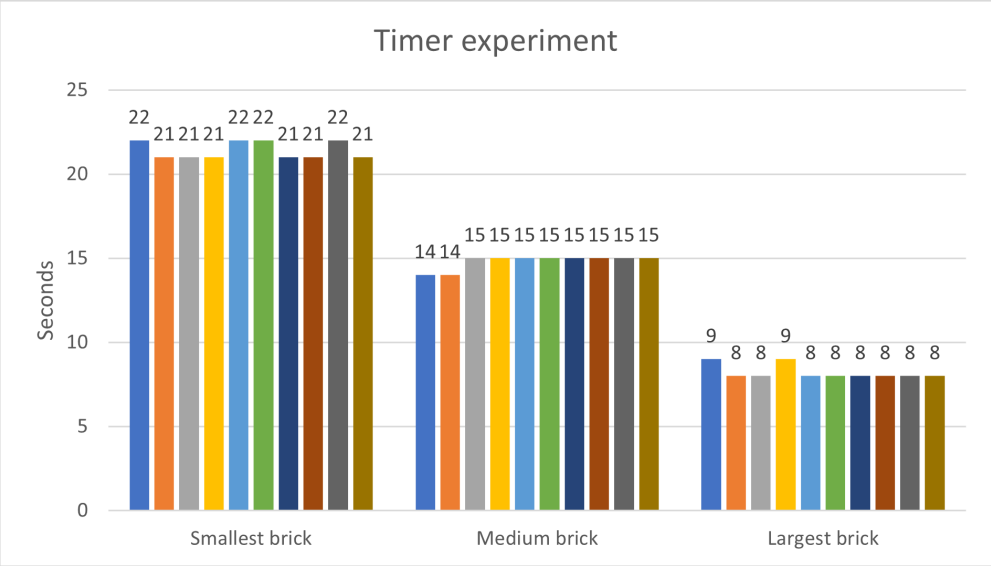


Figure 9: A graph of the results

By analyzing the data from the experiment as seen on the figure above, the time needed for each brick was set (look section 3.3.2 for Drop off function). The data in this experiment was printed directly into the Ubuntu terminal, with the help of `std::cout`, a C++ function. The experiment also showed how the gripper is performing while gripping. The results are also seen in the table above. The smallest brick has the longest wait time at 22 seconds, medium brick at 14 seconds and the largest at 8 seconds. It can be concluded that the results match the expectations.

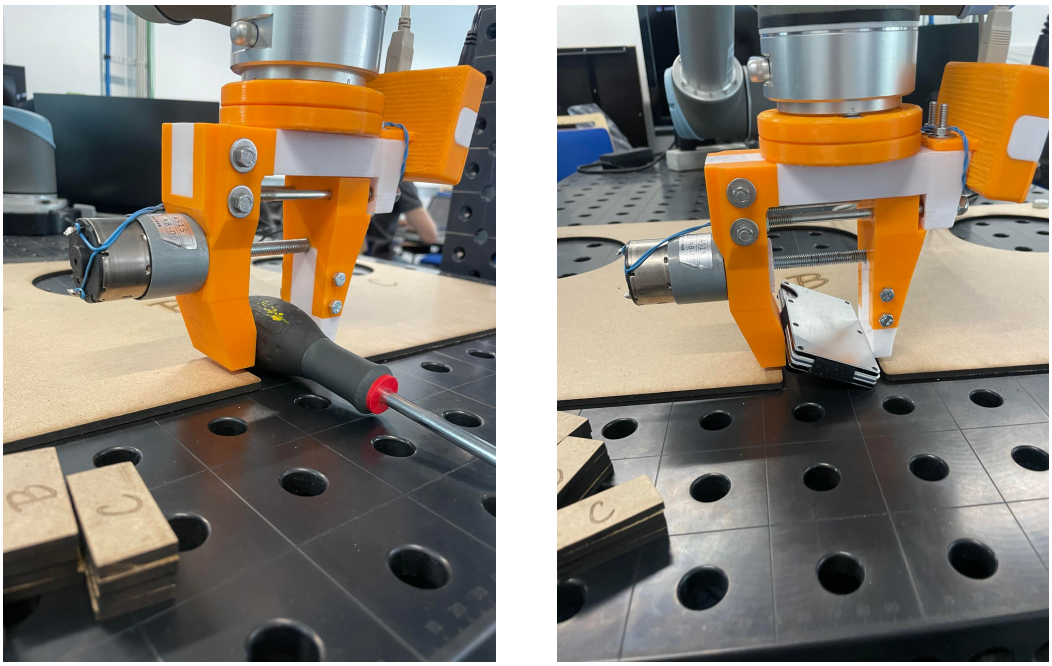
6.2 Experiment: Different Objects

This experiment involved placing various objects for the gripper to pick up, aiming to assess its capacity and limitations. By testing the gripper with different objects, valuable insight was gained regarding its ability to securely hold objects of varying sizes and shapes. The following objects were: `screwdriver`, `wallet`, `metal brick` and `magnetic eraser for whiteboard`. It was expected that the robot was versatile enough to grip these different objects.

The attempt to pick and place a screwdriver was successful. As seen from fig. 10 (a) the gripper picked the screwdriver and placed it in another position. The shape of the

picked object was elliptic yet made out of rubber.

The metallic wallet's shape was rectangular, yet the surface was quite slippery as seen from fig. 10 (b) the wallet was tilted during the pick up process. The gripper succeeded in moving and placing the wallet.



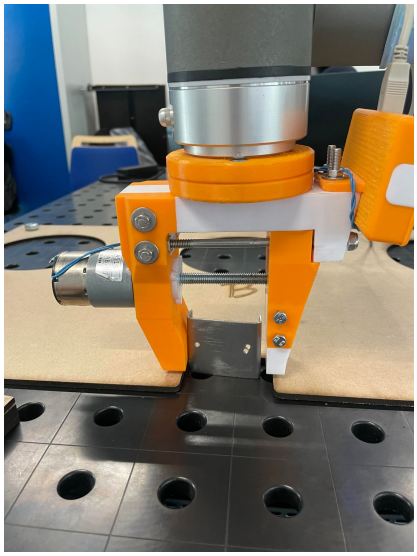
(a) Picture of gripping a screwdriver test

(b) Picture of the wallet gripping test

Figure 10: Gripper test for different objects

As for the metal brick, it has a shape similar to the normal bricks, but the material is different fig. 11 (a). It is made from metal. The gripper successfully picked and placed the metal brick.

The gripper failed to pick up the eraser, it slipped out of the fingers grasp fig. 11 (b). This happened due to the shape of the eraser which is a trapezoid thereby reducing the surface area the gripper is pressing against.
The gripper is capable of gripping a lot of different shaped objects such as elliptic and rectangular. The gripper is not capable of picking up a trapezoid, because of its mechanical limitations.



(a) Picture of gripping a metal brick test

(b) Picture of gripping an eraser test

Figure 11: Gripper test for different objects

6.3 Experiment: Unaligned bricks

The objective for the last experiment was finding out whether it mattered for the brick to be perfectly placed in the pick up spot. The experiment, see fig. 12, was set up so that there was twenty bricks which were perfectly placed and after that 20 bricks were placed with an angle of 25 degrees. For the perfectly placed brick 20 out of 20 were successfully picked and placed, but for the bricks which were unaligned 19 out of 20 were successfully picked and placed. Yet a second attempt was tried with an angle of 30 degrees and every brick failed to be picked up. The limitation for the angle of the brick was around 30 degrees.

Figure 12: A picture of the experiment

7 Discussion

As in any human endeavor, there are always improvements to make. The subsections for this current section simply comprise a listing of some of the improvements that could have been implemented if more time had been available.

7.1 Pinion and Rack system

The pinion and rack system, as mentioned in the ”Design-Comments” subsection 4.1, would look something like this (see figure fig. 13):

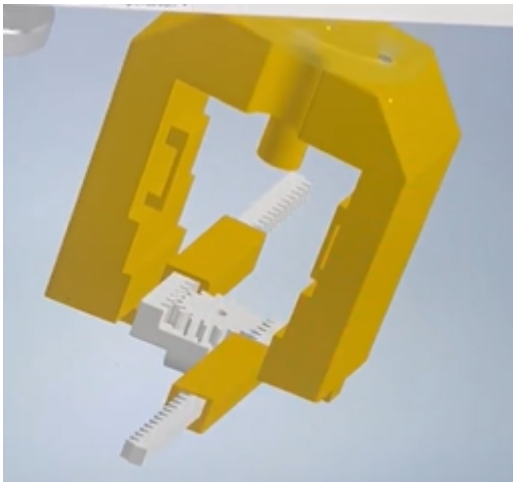


Figure 13: Early concept for the gripper based on the pinion and rack closing mechanism.

The most important feature in the pinion and rack configuration when solving for our speed issue, is that the radius of the pinion is proportional to the rack translation per degree of rotation, which delivers some significant flexibility in both design and performance.

7.2 Wattage calculator and Database

Incorporating a Wattage calculator with a Database could give more direct insight into the watt usage of the system. How would this work? This would work simply by calculating the average wattage over a work iteration, multiplying it by the time(in hours) for said work iteration to complete, and storing this calculation into a database with time stamps. This way one can see how much energy was expended during any given time interval.

7.3 Sleep mode

Although the knowledge regarding sleep modes for micro-controllers is limited, it is

known that entering ”sleep mode” saves energy. Therefore, programming a ”sleep mode” into the microcontroller as its ”standby mode” (as mentioned in the ”ATMega-Board” section) could potentially reduce energy consumption, for example.

7.4 MySQL Database

A database could have been set up, to display the number of bricks at a certain position on the GUI, so that it would be easier to see how many bricks are available to be sorted and the results would have been ”remembered” on a new start up of the program. Because it would be possible to send a query to the database on start-up.

7.5 Experiments

There were three experiments made: Timer, Object and Unaligned bricks experiments had successfully tested the limitations of the robot. The timer experiment had an impact on how the bricks are sorted see fig. 9.

The object experiment showed that the gripper was able to handle different shaped objects. However, the gripper encountered a problem when trying to pick up the trapezoidal shaped eraser, the gripper could not apply sufficient grip force due to the reduced surface area. See section 6.2.

The unaligned bricks experiment showed a limitation of 30 degrees when trying to pick up bricks that were purposefully misaligned. See section 6.3.

8 Conclusion

The goal was to make an automated sorting machine, and have the robot know how big the brick is, and then sort accordingly.

The presented ATMega-Board integrated various ICs and peripherals to achieve specific functionalities. It includes an ATMega644PA chip, an H-bridge IC, a current sensor IC, and a USART port. After initialization, the chipp stays in ”standby mode”, awaiting a receive complete interrupt. Upon receiving the interrupt, it reads the data register and remains in receive mode until the correct close/open command is received. After a short duration, when the motor’s speed is stabilized, the current sensor on the motor continuously monitors the load on the motor. When the threshold voltage for closing/opening is reached, a ”closing done” or ”opening done” message is sent via the USART port. Finally, the board re-enters ”standby mode”, turns off the motor, and waits for a new command.

The provided C++ code implements a GUI-based application that allows users to interact with the robot’s gripper. Connection between the UR5 robot and the C++ code were made with the Libmodbus library. The `MyPanel` class handles the gripper’s movement and location settings. The `Atmega` class manages communication with the Atmega device. Additionally, the `MyFrame` class provides the primary interface window, with buttons for controlling the gripper. Finally, the `OnInit()` function initializes and launches the application.

The gripper design centers around a motor-driven bolt attached to the shaft, serving as its closing mechanism. This bolt interacts with a nut placed inside a movable finger, while the motor is connected to a fixed finger. With two fingers, one rigidly attached to the base and the other movable, the gripper employs a motor-driven threaded rod and nut system to control the movement of the movable finger. By rotating the rod, the nut engages with the finger, enabling it to move closer or farther from the fixed finger depending on the motor’s spin direction. The gripper is constructed primarily using PLA material, incorporating printed flanges for easy connection to the robot. It also features an enclosure for the ATMega644PA chip, which interfaces with the motor. The gripper fits the project’s requirements in terms of sturdiness and durability, making it efficient for sorting bricks.

The determination of which it was relied on a timer. The gripping timer experiment showed that the largest brick took approximately 8 seconds to grip. It took 14 seconds to grip the medium-sized brick. Lastly, it took 22 seconds to grip the smallest brick. The experiment perfectly matched the expectations.

The second experiment showed that the gripper is quite versatile. Lots of distinct objects were tested and trialed. The gripper managed to lift a screwdriver, a wallet and a metal brick. The gripper however failed to lift an eraser due to its trapezoid figure.

In the third experiment, attempts were made to pick up the bricks from different angles. Results showed that if the bricks are horizontal to the gripper, the gripper will have no issue picking them up. However, when the bricks were tilted at a 25-degree angle, 19 out of 20 bricks were successfully picked up. A second attempt was made, this time at an angle of 30 degrees. However, every attempt was unsuccessful. From this we find a limitation of 30 degrees for the angle of the brick.

9 Appendix

This Appendix refers to supplementary files sent together with this document inside a ZIP-file named ”Appendix”. The names of the folders and documents inside the zip file referred to in this document are the following:

- C++ code
- Assembly Code
- 644Board-Schematic.pdf

10 Bibliography

[1] Infineon. *IFX9201SG - 6 A H-Bridge with SPI*, 2015. Datasheet.

[2] Texas Instruments. *INA240 High- and Low-Side, Bidirectional, Zero-Drift, Current-Sense Amplifier with Enhanced PWM rejection*, 2016. Datasheet.

[3] libmodbus. Libmodbus library. `https://libmodbus.org/`. Accessed: 09-04-2023.

[4] Lulu's blog. C++ cross-platform rs232 serial communication library. `https://lucidar.me/en/serialib/cross-plateform-rs232-serial-library/`. Accessed: 15-04-2023.

[5] Microchip. *ATmega164A/PA/324A/PA/644A/PA/1284/P - megaAVR® Data Sheet*, 2020. Datasheet.

[6] wxWidgets. wxwidgets library. `https://www.wxwidgets.org`. Accessed: 02-04-2023.