University of Moratuwa, Sri Lanka

Faculty of Engineering

Department of Electronics and Telecommunication Engineering
Semester 4 (Intake 2021)

EN2160 - Electronic Design Realization

Industrial End Effector - Design Details

Group B

Kodikara U.S.S	210293K
Gunawardane E.R.N.H.	210200C
Sehara G.M.M.	210583B
Kodithuwakku J.N.	210294N

This report is submitted as a partial fulfillment for the module EN2160 - Electronic Design Realization, Department of Electronic and Telecommunication Engineering, University of Moratuwa.

Contents

1 Comprehensive Design Details

1.1 Microcontroller Unit

The end-effector utilizes an ATmega2560 microcontroller to primarily gather readings from four sensors and perform calculations to determine the object's distance and orientation (yaw and pitch). The ATmega2560's capability to handle complex tasks and its compatibility with extensive sensor libraries make it an ideal choice for this application.

1.1.1 Distance and Orientation Measurement:

In the design of the end-effector, the ATmega2560 microcontroller plays a pivotal role in integrating sensor data and performing complex calculations to determine object distance and orientation. This microcontroller excels in managing multiple time-of-flight sensors through the I2C protocol, utilizing the Adafruit library for seamless sensor interfacing despite its extensive functionality. Here's a detailed breakdown of its functionalities and implementation:

- Sensor Integration via I2C Protocol:
  - The ATmega2560 communicates with four time-of-flight sensors using the I2C protocol, enabling efficient data transmission between the microcontroller and sensors. This protocol is chosen for its simplicity and suitability for connecting multiple sensors over short distances, ensuring reliable and synchronized data acquisition.
- Distance Measurement and Averaging:
  - The The microcontroller collects distance readings from four distinct points on the object using the time-of-flight sensors. These readings are averaged to provide a precise measurement of the object's distance from the end-effector. This averaging process enhances accuracy by mitigating potential outliers or sensor variations.
- Calculation of Pitch and Yaw:
  - Using the averaged distance data, the ATmega2560 computes the object's orientation in terms of pitch and yaw angles. This calculation involves employing inverse tangent equations to translate distance variations into angular measurements, crucial for determining the object's spatial orientation relative to the end-effector.
- Utilization of Adafruit Library for Sensor Interface:
  - The implementation leverages the Adafruit library due to its comprehensive support for sensor interfacing and data handling. Despite its heavyweight nature, the library simplifies the integration of complex sensors with the ATmega2560, streamlining development efforts and ensuring robust sensor data acquisition.
- Data Transfer via Modbus Protocol:
  - After completing distance and orientation calculations, the microcontroller transfers the processed data to a Modbus converter. The Modbus protocol facilitates efficient and standardized communication between the microcontroller and external systems or devices, ensuring seamless integration into larger control or monitoring frameworks.

1.1.2 Programming the Microcontroller

Programming the ATmega microcontroller using the Unified Program and Debug Interface (UPDI) is a strategically advantageous approach tailored for this application. The UPDI method offers several benefits that are highly pertinent:

- **Efficiency in Firmware Management:** UPDI facilitates efficient uploading of firmware into the ATmega microcontroller, ensuring smooth and reliable operation of the end-effector system.
- **Simplicity and Effectiveness:** This programming interface is known for its straightforward implementation and effective debugging capabilities, making it easier to manage and maintain the firmware.
- **Streamlined Development Process:** UPDI streamlines the development process by enabling quick updates and modifications to the microcontroller's firmware, crucial for adapting to changing project requirements or improving functionality.
- **Future-Proofing and Flexibility:** By using UPDI, the system becomes future-proofed, allowing for potential enhancements or adjustments in operational protocols without major hardware changes.

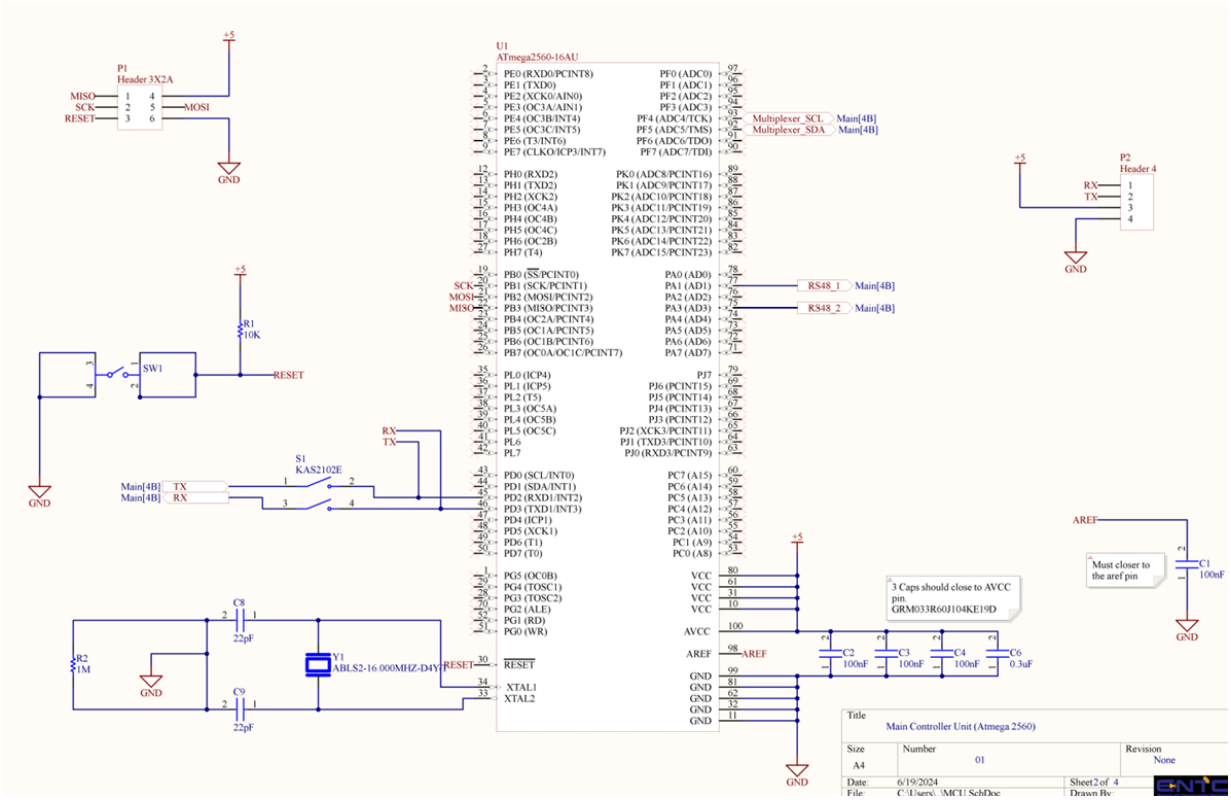


Figure 1: Schematic of microcontroller connections

1.2 Multiplexer

In optimizing the integration of multiple time-of-flight (ToF) sensors with our ATmega2560 microcontroller, we implemented the PCA9546A 1 to 4 multiplexer for streamlined I2C communication.

1.2.1 Why a multiplexer?

- Efficient Management of Sensor Connections:
  - The PCA9546A allows us to connect up to four sensors to a single pair of SDA and SCL lines from the microcontroller.
  - This consolidates multiple I2C connections into a single interface, simplifying the overall wiring layout and reducing the number of required microcontroller pins.
- Minimization of Complexity:
  - By using the multiplexer, we streamline the process of interfacing multiple sensors with the microcontroller.
  - It eliminates the need for additional I2C buses and complex wiring configurations, which can otherwise complicate the design and increase potential points of failure.
- Reduction of Potential Signal Interference:
  - Centralizing sensor connections through the multiplexer helps minimize signal interference and crosstalk between different I2C devices.
  - This ensures that data communication between the microcontroller and each sensor remains reliable and unaffected by external electrical noise.
- Enhanced Reliability of Data Acquisition:
  - The PCA9546A's ability to selectively enable communication with each sensor via its independent channels ensures reliable data acquisition.
  - It provides precise control over which sensor is actively communicating with the microcontroller at any given time, optimizing data throughput and system responsiveness.

1.2.2 Hardware Components

The PCA9546A is a versatile I2C multiplexer designed to enable communication with multiple I2C devices using a single microcontroller interface. It features four independent channels that can be selected via control registers, effectively routing data and clock signals to different I2C buses connected to various sensors. This flexibility makes it ideal for applications requiring efficient management of multiple I2C devices without the need for additional microcontroller pins or complex wiring.

How it is connected

- atmega2560 to multiplexer
  - The main SDA (data) and SCL (clock) lines from the ATmega2560 microcontroller are connected directly to the PCA9546A multiplexer.
- multiplexer to the ToF sensors
  - The PCA9546A multiplexer features four independent channels, each capable of connecting to a separate I2C bus.
  - Each ToF sensor module is connected to one of these channels on the multiplexer, establishing a dedicated communication path between the microcontroller and each sensor.

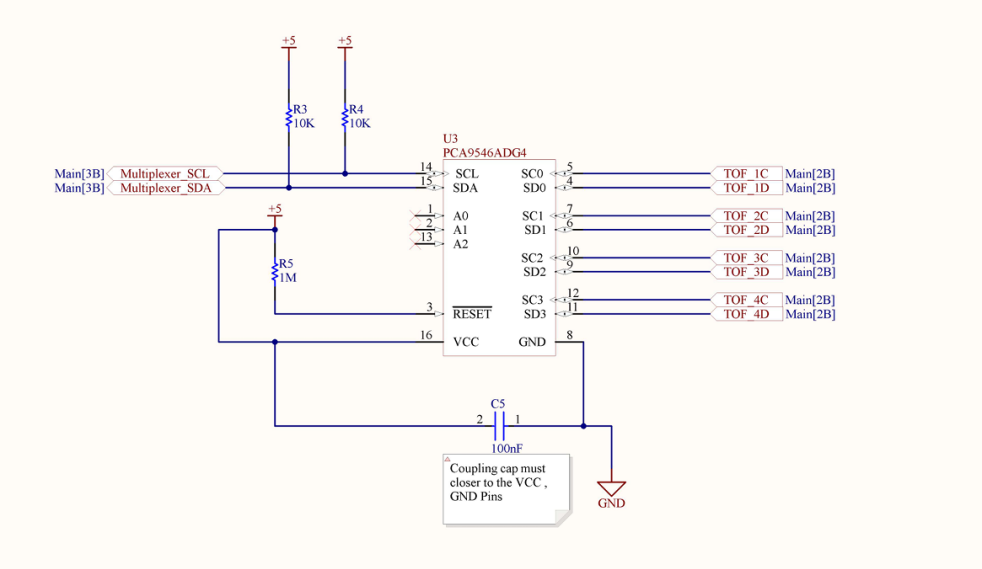


Figure 2: Schematic of PCA9546A connections

1.3 Time of Flight Sensors

1.3.1 Implementation of the Time of Flight Sensors

For the end-effector application, selecting VL53L0X Time-of-Flight (ToF) sensors was crucial for achieving accurate distance measurements from four distinct points on objects. These sensors, renowned for their precision and ease of integration, utilize the I2C communication protocol for seamless interfacing with our ATmega2560 microcontroller. By directly connecting these sensors to a PCA9546A multiplexer, we efficiently manage communication channels while maintaining high measurement accuracy.

1.3.2 Why VL53L0X?

he VL53L0X ToF sensor module is a compact and highly accurate device designed specifically for distance measurement applications. It operates by emitting short pulses of infrared light and measuring the time it takes for these pulses to reflect off a target surface and return to the sensor. This time-of-flight measurement directly correlates to the distance between the sensor and the object being measured. The sensor employs advanced algorithms within its internal processing unit to calculate distances with exceptional precision, achieving resolutions down to millimeters. This method ensures consistent and reliable performance across diverse environmental conditions, making the VL53L0X ideal for applications requiring precise and robust distance measurements. Some advantages of the implementation of VL53L0X are,

- **High Accuracy:** Provides precise distance measurements with resolutions as fine as millimeters, ensuring reliable data for critical applications.
- **Compact Size:** Small form factor makes it suitable for integration into compact or space-constrained devices and systems.
- **Wide Measurement Range:** Offers a wide range of measurement distances, accommodating diverse application needs from short to long distances.
- **Easy Integration:** Supports simple integration into existing systems with straightforward I2C communication protocol and user-friendly interfaces.

1.3.3 Hardware Connections

- Direct Connection to Multiplexer Channels:
  - Each VL53L0X sensor module is connected directly to one of the output channels of the PCA9546A multiplexer.

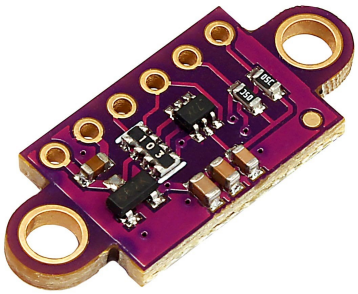


Figure 3: VL53L0X Sensor Module

1.4 Modbus Transmitter

In the development of an industrial end effector, particularly for a vacuum gripper, precise and reliable transmission of sensor data is crucial. Our project employs four Time of Flight (ToF) sensors to measure pitch and yaw angles, with the data transmitted to the robot control panel using the Modbus protocol. This section details the transmission component of the project, covering hardware and data transmission steps.

1.4.1 Implementation of Modbus Communication

Modbus is a serial communication protocol designed for industrial applications, facilitating communication between electronic devices. Its simplicity, reliability, and support for both serial and TCP/IP communications make it an excellent choice for our project.

1.4.2 Why Choose Modbus?

- **Standardization:** Widely used in industrial environments.
- **Distance and Reliability:** Supports long-distance communication with robust error checking.
- **Integration:** Easily integrates with various industrial equipment and control systems.

1.4.3 Hardware Components

MAX485 TTL to RS485 Converter Module
To transmit data from the ToF sensors to the robot control panel, we use the MAX485 TTL to RS485 MAX485CSA Converter Module. This module converts TTL level signals from the sensors to RS485, suitable for long-distance communication.

How It Works

The MAX485 module converts the TTL signals from the sensors to differential RS485 signals, enabling reliable data transmission over long distances in noisy industrial environments.

How to Connect

- atmega2560 to MAX485 Module:
  - Connect the appropriate GPIO pins of the atmega2560 (TX and RX pins, usually pins 18 and 19 for serial communication) to the input pins of the MAX485 module (DI and RO pins respectively).
  - Connect the DE (Data Enable) and RE (Receive Enable) pins of the MAX485 module to a digital pin on the Arduino Mega to control the direction of data flow.
- MAX485 Module to RS485 Bus:
  - Connect the RS485 output pins (A and B) of the MAX485 module to the RS485 input of the robot control panel.
- Power Supply:
  - Ensure proper power supply connections to the MAX485 module from the Arduino Mega or an external source.

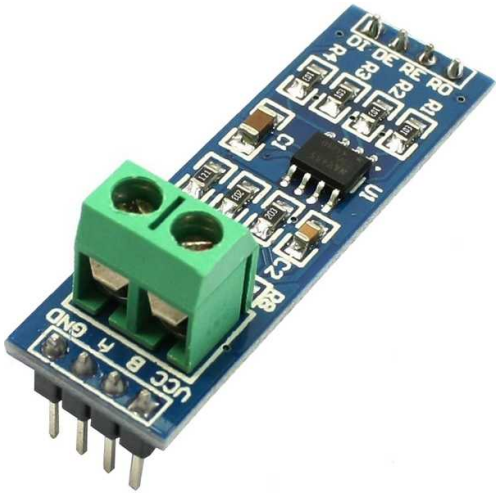


Figure 4: MAX485 TTL to RS485 Converter Module

1.5 Testing and Demonstration

In the end effector, we utilize the Modbus protocol to communicate with the robot arm. However, for testing and debugging purposes, we need to display the calculated pitch and yaw angles from our end effector. Since these values are transmitted using the Modbus protocol, they cannot be directly observed on the computer interface. Therefore, we employ an RS485 to USB converter module along with Modbus Poll software. This setup is essential solely for demonstration and testing purposes.

1.5.1 USB to RS485 Communication Module

Functionality
The USB to RS485 Communication Module is essential for converting Modbus protocol signals from the RS485 bus into a format compatible with USB, enabling data display and interaction on a computer.

Operational Mechanism

- Signal Conversion:
  - Converts Modbus protocol signals from RS485 to USB format, facilitating direct communication between the RS485 bus and a computer.
- Display Compatibility:
  - Enables the display and interaction of received data from RS485 devices on a computer interface via USB port of the computer.
- Diagnostic and Debugging Tool:
  - Used extensively for monitoring and debugging data transmissions between sensors, the end effector, and the computer; ensuring clear and accurate display of data received via Modbus protocol on a computer screen.

Connection Procedure:

- Connect the RS485 side of the 'USB to RS485 module' to the 'MAX485 TTL to RS485 Converter Module'.
- Plug the USB side of the module into a computer, establishing a direct interface for data display and interaction.

Figure 5: USB to RS485 Communication Module

1.5.2 Modbus poll software

Functionality
Modbus Poll software serves as a Modbus master simulator designed for monitoring and testing Modbus systems. It facilitates visualization and interaction with data exchanged over the RS485 bus.

Operational Mechanism

- Data Monitoring:
  - Allows users to send Modbus queries to devices connected on the RS485 bus.
- Real-time Display:
  - Displays data received from connected devices, enabling real-time monitoring and testing of Modbus communications.

- Diagnostic and Debugging Tool:
  - Essential for verifying data accuracy, integrity, and transmission reliability before it reaches the robot control panel, ensuring the data transmitted by sensors is accurate and consistent.

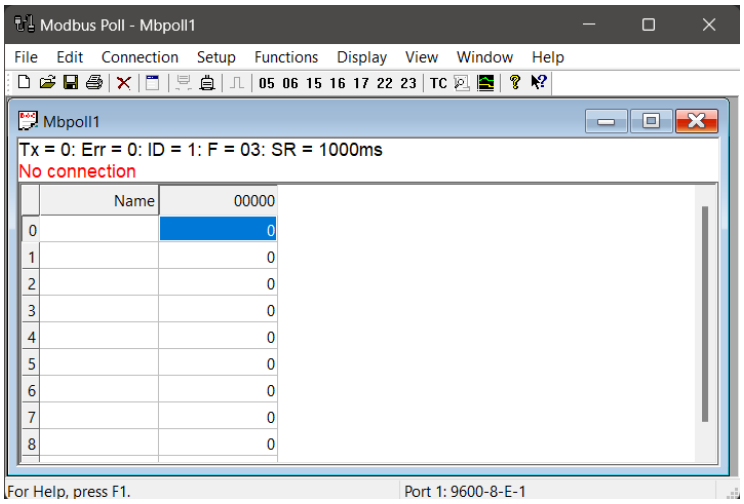


Figure 6: Modbus Poll Software

2 Final Code

```
#include <avr/io.h>
#include "delay.h"
#include "uart.h"
#include "TWI.h"
#include "ToF.h"
#include "multi_distance.h"

// // below values should be configured
float dv = 41; // vertical distance between the ToF sensors
float dhu = 76; // horizontal distance between the upper ToF sensors
float dhd = 76; // horizontal distance between the lower ToF sensors

// // below values are read and calculated
float relativeDistance;
float dlu; // left upper ToF sensor distance
float dru; // right upper ToF sensor distance
float dld; // left down ToF sensor distance
float drd; // right down ToF sensor distance
float thetaP; // pitch angle (angle around the horizontal axis)
float thetaY; // yaw angle (angle around the vertical axis)

int main(){
  init_uart(); // Intializing the Serial communication with UART protocol
  i2c_init(); // Intializing TWI
  setID(); // Intializing the multiplexer and the ToF sensors
  init_rs485(); // Initialize RS485 pins

  while(1){
    dlu = 0;
    dru = 0;
    dld = 0;
    drd = 0;

    for (int i=0;i<5;i++){ // Implementation of the moving average filter
      dlu += sensor_1() - 5;
      dru += sensor_2();
      dld += sensor_3();
      drd += sensor_4() - 4;
      delay_ms(10);
    }

    dlu = dlu/5;
    dru = dru/5;
    dld = dld/5;
    drd = drd/5;

    relativeDistance = (dlu + dru + dld + drd)/4;

    thetaP = 0.5*atan((dlu-dld)/dv) + 0.5*atan((dru-drd)/dv); //
       Calculating the Pitch
    thetaY = 0.5*atan((dru-dlu)/dhu) + 0.5*atan((drd-dld)/dhd); //
       Calculating the Yaw
```

```
51
52      thetaP = degrees(thetaP);
53      thetaY = degrees(thetaY);
54
55      trans_string("Relative Distance : ");
56      trans_num(relativeDistance);
57      trans_string(" Pitch Angle : ");
58      trans_num(thetaP);
59      trans_string(" Yaw Angle : ");
60      trans_num(thetaY);
61      trans_string("\n");
62
63      // Write angles as floats to Modbus registers
64      modbus_write_float_registers(MODBUS_SLAVE_ID, 0x0000, thetaP); //
           Write thetaP as float to registers 0 and 1
65      delay_ms(100);
66      modbus_write_float_registers(MODBUS_SLAVE_ID, 0x0002, thetaY); //
           Write thetaY as float to registers 2 and 3
67
68      delay_ms(1000);
69    }
70
71    return 0;
72  }
```

Listing 1: Main Code

```
1  #define VL53L1X_ADDR 0x52 // Defining the ToF Address
2
3  void vl53l1x_init() {
4      // Initializing the ToF
5      i2c_start();
6      i2c_write(VL53L1X_ADDR); // Device address
7      i2c_write(0x00); // Default register address
8      i2c_write(0x01);
9      i2c_stop();
10
11 }
12
13 void vl53l1x_measure_distance(){
14   i2c_start();
15   i2c_write(VL53L1X_ADDR); // Device address
16   i2c_write(0x00);
17   i2c_write(0x02); // Setting the measuring mode
18   i2c_stop();
19 }
20
21 uint16_t vl53l1x_read_distance() {
22     uint16_t distance;
23
24     // Read distance high byte
25     i2c_start();
26     i2c_write(VL53L1X_ADDR); // Device address
27     i2c_write(0x14 + 10); // Register address for distance high byte
28     i2c_start(); // Repeated start
```

```
    i2c_write(VL53L1X_ADDR | 0x01); // Read mode
    uint8_t high_byte = i2c_read_ack();

    // Read distance low byte
    uint8_t low_byte = i2c_read_nack();
    i2c_stop();

    distance = (high_byte << 8) | low_byte;
    // distance = low_byte;

    return distance;
}
```

Listing 2: ToF Library

```
#ifndef ToF
#define ToF

#include <avr/io.h>
#include "TWI.h"

void vl53l1x_init();
void vl53l1x_measure_distance();
uint16_t vl53l1x_read_distance();

#endif
```

Listing 3: ToF Header File

```
void i2c_init() {
    // Set the clock speed
    TWSR = 0x00;
    TWBR = ((F_CPU/400000)-16)/(2*pow(4,(TWSR&((1<<TWPS0)|(1<<TWPS1)))));
}

void i2c_start() {
    // Send start condition
    TWCR = (1 << TWSTA) | (1 << TWEN) | (1 << TWINT);
    while (!(TWCR & (1 << TWINT)));
}

void i2c_stop() {
    // Send stop condition
    TWCR = (1 << TWSTO) | (1 << TWEN) | (1 << TWINT);
}

void i2c_write(uint8_t data) {
    // Write data to the I2C bus
    TWDR = data;
    TWCR = (1 << TWEN) | (1 << TWINT);
    while (!(TWCR & (1 << TWINT)));
}

uint8_t i2c_read_ack() {
    // Read data from the I2C bus with acknowledgment
    TWCR = (1 << TWEN) | (1 << TWINT) | (1 << TWEA);
```

```
    while (!(TWCR & (1 << TWINT)));
    return TWDR;
}

uint8_t i2c_read_nack() {
    // Read data from the I2C bus without acknowledgment
    TWCR = (1 << TWEN) | (1 << TWINT);
    while (!(TWCR & (1 << TWINT)));
    return TWDR;
}
```

Listing 4: TWI Library

```
#ifndef TWI
#define TWI

#include <avr/io.h>

void i2c_init();
void i2c_start();
void i2c_stop();
void i2c_write(uint8_t data);
uint8_t i2c_read_ack();
uint8_t i2c_read_nack();

#endif
```

Listing 5: TWI Header File

```
//receive one byte data
unsigned char USART_Receive()
{
  while(!(UCSR0A & (1<<RXC0)));  //receive buffer empty
  return UDR0; //receive reg
}


//transmit one byte
void UART_Txchar(char ch)
{
  while(!(UCSR0A & (1<<UDRE0))); //WAIT UNTIL EMPTY TRANSMIT BUFFER
  UDR0 = ch; // after transmit buffer empty load data to the uart data
     register

}


//transmit string
void trans_string(char str[]){
  int lenght = strlen(str);
  char trans;
  for(int i=0;i<lenght;i++){
    trans = str[i];
    UART_Txchar(trans);
  }
}
```

```
27
28 //transmit number
29 void trans_num(uint16_t num){
30   String number = String(num);
31   int lenght = number.length();
32   char trans;
33   for(int i=0;i<lenght;i++){
34     trans = number[i];
35     UART_Txchar(trans);
36   }
37   // UART_Txchar('\n');
38 }
39
40 void init_uart(){
41   //Enable transmitter and recirever bits - Use USART0
42
43   UCSR0B |= (1<<RXEN0) | (1<<TXEN0); // ENABLE RX AND RX
44
45   //SET THE DATA SIZE FOR COMMUNIATION
46   UCSR0C &= (~(1<<UMSEL00)) & (~(1<<UMSEL01));   //ENABLE ASYNCHRONOUS
       USART COMMUNICATION
47   UCSR0C &= (~(1<<UPM00)) & (~(1<<UPM01));   // DISABLE PARITY BIT
48   UCSR0C &= (~(1<<USBS0)); //CHOOSE ONE STOP BIT
49
50   //SET DATA LENGHT TO BE 8 BITS
51   UCSR0B &= (~(1<<UCSZ02));
52   UCSR0C |= (1<<UCSZ00) | (1<<UCSZ01);  //SET 8BITS 011 BITS IN UCSZ02,
       UCSZ00,UCSZ01
53
54   //SET THE SPEED FOR TRANSMISSION
55
56   UCSR0A |= (1<<U2X0); //SELSCT HIGH SPEED MODE
57   // UCSR0A &= ~(1<<U2X0); // low speed
58
59   //BAUDRATE
60   UBRR0 = 207; //9600 BAUDRATE FROM DATASHEET DIRECT TABLE FOR U2X0=1 if
       u2x=0 , ubrr0 = 103
61 }
```

Listing 6: UART Library

```
1 #ifndef UART
2 #define UART
3
4 #include <avr/io.h>
5
6 unsigned char USART_Receive();
7 void UART_Txchar(char ch);
8 void trans_string(char str[]);
9 void trans_num(uint16_t num);
10 void init_uart();
11
12 #endif
```

Listing 7: UART Header File

```
void TCA9548A(uint8_t bus) { // Initializing the multiplexer channel
  i2c_start();
  i2c_write(0x70);
  i2c_write(bus);
  i2c_write(0x01);
  i2c_stop();
}

void setID() { // Initializing the ToF sensors

  TCA9548A(0);
  vl53l1x_init();

  TCA9548A(1);
  vl53l1x_init();

  TCA9548A(2);
  vl53l1x_init();

  TCA9548A(3);
  vl53l1x_init();
}

float sensor_1() { // Getting sensor 1 reading
  TCA9548A(0);
  vl53l1x_measure_distance();
  return (float) vl53l1x_read_distance();
}
float sensor_2() { // Getting sensor 2 reading
  TCA9548A(1);
  vl53l1x_measure_distance();
  return (float) vl53l1x_read_distance();
}

float sensor_3() { // Getting sensor 3 reading
  TCA9548A(2);
  vl53l1x_measure_distance();
  return (float) vl53l1x_read_distance();
}

float sensor_4() { // Getting sensor 4 reading
  TCA9548A(3);
  vl53l1x_measure_distance();
  return (float) vl53l1x_read_distance();
}
```

Listing 8: Multiplexer Library

```
#ifndef MULTI
#define MULTI

#include <avr/io.h>
#include "TWI.h"
#include "ToF.h"
```

```
7
8  void TCA9548A(uint8_t bus);
9  void setID();
10 float sensor_1();
11 float sensor_2();
12 float sensor_3();
13 float sensor_4();
14
15 #endif
```

Listing 9: Multiplexer Header File

```
1  void delay_ms(unsigned int ms) {
2    for (unsigned int i = 0; i < ms; i++) {
3        // Configure Timer1
4        TCCR1A = 0;  // Clear Timer1 Control Register A
5        TCCR1B = 0;  // Clear Timer1 Control Register B
6        TCNT1 = 0;   // Clear Timer1 Counter
7
8        // Set Timer1 to CTC mode and set prescaler to 64
9        TCCR1B |= (1 << WGM12) | (1 << CS11) | (1 << CS10);
10
11       // Set compare match register for 1ms delay
12       OCR1A = 249;
13
14       // Wait for compare match flag to be set
15       while (!(TIFR1 & (1 << OCF1A)));
16
17       // Clear compare match flag
18       TIFR1 |= (1 << OCF1A);
19   }
20 }
```

Listing 10: Delay Library

```
1  #ifndef DELAY
2  #define DELAY
3
4  #include <avr/io.h>
5
6  void delay_ms(unsigned int ms);
7
8  #endif
```

Listing 11: Delay Header File

```
1  // Define Modbus Slave ID
2  #define MODBUS_SLAVE_ID 1
3
4  // Define RS485 pins
5  #define MAX485_DE      PB0
6  #define MAX485_RE_NEG  PB1
7
8  // Initialize RS485 pins
9  void init_rs485() {
10   DDRB |= (1 << MAX485_DE) | (1 << MAX485_RE_NEG);
```

```
  PORTB &= ~(1 << MAX485_DE);
  PORTB &= ~(1 << MAX485_RE_NEG);
}

// Enable RS485 transmit mode
void enable_transmit() {
  PORTB |= (1 << MAX485_DE);
  PORTB |= (1 << MAX485_RE_NEG);
}

// Enable RS485 receive mode
void enable_receive() {
  PORTB &= ~(1 << MAX485_DE);
  PORTB &= ~(1 << MAX485_RE_NEG);
}


// Function to calculate CRC for Modbus
uint16_t calculate_crc(uint8_t *buffer, uint8_t length) {
  uint16_t crc = 0xFFFF;
  for (int pos = 0; pos < length; pos++) {
    crc ^= (uint16_t)buffer[pos];
    for (int i = 8; i != 0; i--) {
      if ((crc & 0x0001) != 0) {
        crc >>= 1;
        crc ^= 0xA001;
      } else {
        crc >>= 1;
      }
    }
  }
  return crc;
}

// Function to write two consecutive registers with a float value
void modbus_write_float_registers(uint8_t id, uint16_t address, float
   value) {
  union {
    float f;
    uint16_t parts[2];
  } float_converter;

  float_converter.f = value;

  // Send high part
  modbus_write_single_register(id, address, float_converter.parts[1]);
  delay_ms(100);

  // Send low part
  modbus_write_single_register(id, address + 1, float_converter.parts[0]);
  delay_ms(100);
}

// Function to write a single register on Modbus
```

```
void modbus_write_single_register(uint8_t id, uint16_t address, uint16_t
    value) {
  uint8_t frame[8];
  uint16_t crc;

  frame[0] = id; // Slave ID from the robot controller
  frame[1] = 0x06; // Function code (Write Single Register)
  frame[2] = (address >> 8) & 0xFF; // Address high byte
  frame[3] = address & 0xFF; // Address low byte
  frame[4] = (value >> 8) & 0xFF; // Value high byte
  frame[5] = value & 0xFF; // Value low byte

  crc = calculate_crc(frame, 6);
  frame[6] = crc & 0xFF; // CRC low byte
  frame[7] = (crc >> 8) & 0xFF; // CRC high byte

  enable_transmit();
  for (int i = 0; i < 8; i++) {
    UART_Txchar(frame[i]); // Transmit each byte of the frame over UART
  }
  enable_receive();
}
```

Listing 12: Modbus Library

```
#ifndef MODBUS
#define MODBUS

#include <stdint.h>
#include <avr/io.h>
#include "uart.h"

// Define Modbus Slave ID
#define MODBUS_SLAVE_ID 1

// Define RS485 pins
#define MAX485_DE      PB0
#define MAX485_RE_NEG  PB1

// Function declarations
void init_rs485();
void enable_transmit();
void enable_receive();
void modbus_write_float_registers(uint8_t id, uint16_t address, float
    value);
void modbus_write_single_register(uint8_t id, uint16_t address, uint16_t
    value);

// Function to calculate CRC for Modbus
uint16_t calculate_crc(uint8_t *buffer, uint8_t length);

#endif
```

Listing 13: Modbus Header File

3 Daily Log Entries

Week 1 (Background Research)

3.1 February 18, 2024

We began our project by conducting initial background research on robotic depalletizing systems, focusing on two primary resources recommended by our professor. The first resource was a YouTube video titled "A Re configurable Gripper for Robotic Autonomous Depalletizing" which provided an overview of innovative gripper technology designed for supermarket logistics. This video highlighted key aspects such as the gripper's ability to handle various box shapes and weights through its re configurable design, and the integration of sensors for adaptive grasping.

3.2 February 21, 2024

We involved a detailed review of the academic paper titled "A Reconfigurable Gripper for Robotic Autonomous Depalletizing in Supermarket Logistics," published in IEEE Robotics and Automation Letters. This paper, authored by G. Andrea Fontanelli et al., detailed the technical specifications and operational principles of the reconfigurable gripper. Key insights included the gripper's mechanical structure, sensor integration, and adaptive capabilities which allow it to handle different sizes and weights of boxes, enhancing efficiency in unstructured environments like supermarkets.

3.3 February 23, 2024

We compared insights from both the video and the paper, identifying additional technologies used in robotic depalletizing systems. These technologies include Time-of-Flight (ToF) sensors, Kinect sensors employing computer vision, and mechanical probes. ToF sensors measure distance by calculating the time it takes for a light signal to reflect off an object and return. Kinect sensors use computer vision to create 3D maps of the environment, aiding in object detection and manipulation. Mechanical probes physically interact with objects to determine their dimensions and orientation. These technologies offer different approaches to enhancing robotic precision and adaptability in various industrial settings.

Week 2 (Comparing Different Approaches)

3.4 February 26, 2024

We focused on comparing different technological approaches for our robotic depalletizing system. On the first day, we explored the use of VL53L0X Time-of-Flight (ToF) sensors to measure pitch and yaw angles. The VL53L0X sensors work by emitting a laser pulse and measuring the time it takes for the pulse to return after reflecting off an object. This time measurement is then used to calculate the distance to the object. To measure pitch and yaw angles, we configured four VL53L0X ToF sensors positioned at known distances apart. By analyzing the distance data from these sensors, we can calculate the angles based on the differences in distance measurements. This method requires precise placement of the sensors to ensure accurate angle calculations, and it proved effective but required careful alignment and calibration.

3.5 February 29, 2024

Then we explored the use of Lidar sensors for measuring pitch and yaw angles. Lidar, which stands for Light Detection and Ranging, uses laser pulses to create detailed 3D maps of the environment.

A Lidar sensor scans the surroundings by emitting laser beams and measuring the time it takes for each beam to return after hitting an object. This process results in a high-resolution point cloud that represents the spatial characteristics of the environment. For our application, we can use Lidar to measure pitch and yaw angles by analyzing the spatial data it provides. The Lidar sensor can detect the orientation and position of objects more comprehensively than individual ToF sensors, making it a robust option for capturing detailed environmental information.

3.6 March 1, 2024

We compared the pros and cons of using VL53L0X ToF sensors versus Lidar sensors for our project, focusing on cost and practical arrangement. VL53L0X ToF sensors are relatively inexpensive and easy to integrate but require precise positioning and calibration, which can be challenging in a dynamic industrial setting. Lidar sensors, on the other hand, provide more comprehensive data and require less meticulous placement but are significantly more expensive. After considering these factors, we concluded that using VL53L0X ToF sensors is the more effective solution for our project. Their lower cost and sufficient accuracy for our application outweigh the benefits of the more detailed but expensive Lidar sensors. This decision aligns with our project's budget constraints and operational requirements, ensuring a practical and cost-effective implementation.

Week 3 (Comparing Different Approaches)

3.7 March 5, 2024

Then we explored the use of Kinect sensors for measuring pitch and yaw angles using computer vision techniques. Kinect sensors, originally designed for gaming, have become popular in various fields for their ability to capture depth and motion. The sensor uses an infrared projector combined with a monochrome CMOS sensor to capture 3D data. For our application, the Kinect sensor captures a detailed depth map of the environment. By processing this depth data with computer vision algorithms, we can calculate the pitch and yaw angles of objects. The Kinect's ability to capture both depth and color information allows for robust and detailed environmental analysis, providing comprehensive spatial data that can be used to determine the orientation and position of objects with high precision.

3.8 March 9, 2024

We compared the pros and cons of using Kinect sensors versus VL53L0X ToF sensors and Lidar sensors for our project, focusing on cost and practical arrangement. Kinect sensors offer the advantage of capturing rich 3D data along with color information, which can be beneficial for more complex analysis and applications beyond simple distance measurements. However, they tend to be bulkier and require more computational power to process the data. Additionally, Kinect sensors are more sensitive to environmental lighting conditions compared to ToF sensors. Despite these limitations, the Kinect's ability to provide comprehensive data makes it a strong candidate for applications requiring detailed environmental mapping. However, due to its higher cost and complexity in integration, we determined that VL53L0X ToF sensors remain the more practical and cost-effective choice for our specific needs, offering sufficient accuracy and ease of integration within our project's budget and operational constraints.

Week 4 (Enclosure Arrangement)

3.9 March 11, 2024

Today, our team collectively drew and analyzed four conceptual enclosure designs. Each member contributed to the brainstorming session, and together we evaluated these designs based on specific criteria. This collaborative effort allowed us to discuss the strengths and challenges of each concept, guiding us towards refining our enclosure design strategy effectively.

Week 5 (Implementation of the code)

3.10 March 17, 2024

We focused on developing an algorithm to measure the distance, yaw, and pitch of an object using distance readings from four sensors placed at the corners of a rectangular shape. Utilizing the following equations, which are derived from the referenced research paper, we were able to achieve our measurements:

$$\theta_p = \frac{1}{2}\tan^{-1}\left(\frac{D_{lu}}{D_v}\cdot\frac{D_{ld}}{D_v}\right) + \frac{1}{2}\tan^{-1}\left(\frac{D_{ru}}{D_v}\cdot\frac{D_{rd}}{D_v}\right) \tag{1}$$

$$\theta_y = \frac{1}{2}\tan^{-1}\left(\frac{D_{ru}}{D_{hu}}\cdot\frac{D_{lu}}{D_{hu}}\right) + \frac{1}{2}\tan^{-1}\left(\frac{D_{rd}}{D_{hd}}\cdot\frac{D_{ld}}{D_{hd}}\right) \tag{2}$$

$$D = \frac{1}{4}(D_{lu} + D_{ru} + D_{ld} + D_{rd}) \tag{3}$$

Where:

- θ_p is the pitch angle,
- θ_y is the yaw angle,
- D is the average distance,
- D_{lu}, D_{ld}, D_{ru}, D_{rd} are the distance readings from the upper-left, lower-left, upper-right, and lower-right sensors respectively,
- D_v is a vertical reference distance,
- D_{hu}, D_{hd} are horizontal reference distances for the upper and lower sensors.

Initial Code - Implemented using Arduino

```
#include "distance.h"
#include <ModbusRtu.h>


#define MODBUS_SLAVE_ID 1
#define MAX485_DE 10
#define MAX485_RE_NEG 11

Modbus slave(MODBUS_SLAVE_ID, Serial, 0);

uint16_t modbus_regs[10];

// Below values should be configured
float dv = 41; // vertical distance between the ToF sensors
float dhu = 76; // horizontal distance between the upper ToF sensors
float dhd = 76; // horizontal distance between the lower ToF sensors

// Below values are read and calculated
float relativeDistance;
float dlu; // left upper ToF sensor distance
float dru; // right upper ToF sensor distance
float dld; // left down ToF sensor distance
float drd; // right down ToF sensor distance
float thetaP; // pitch angle (angle around the horizontal axis)
float thetaY; // yaw angle (angle around the vertical axis)

void setup() {
  // Initialization
  Serial.begin(9600);
  pinMode(MAX485_DE, OUTPUT);
  pinMode(MAX485_RE_NEG, OUTPUT);
  setID();
  slave.begin(9600);
}

void loop() {
  dlu = 0;
  dru = 0;
  dld = 0;
  drd = 0;

  for (int i = 0; i < 5; i++) { // Implementation of the moving average
     filter
    dlu += sensor_1() - 5;
    dru += sensor_2();
    dld += sensor_3();
    drd += sensor_4() - 4;
    delay(10);
  }

  dlu = dlu / 5;
  dru = dru / 5;
```

```
  dld = dld / 5;
  drd = drd / 5;

  relativeDistance = (dlu + dru + dld + drd) / 4;

  thetaP = 0.5 * atan((dlu - dld) / dv) + 0.5 * atan((dru - drd) / dv);
  thetaY = 0.5 * atan((dru - dlu) / dhu) + 0.5 * atan((drd - dld) / dhd);

  thetaP = degrees(thetaP);
  thetaY = degrees(thetaY);

  Serial.print("Relative Distance: " + String(relativeDistance) + " ");
  Serial.print("Pitch Angle: " + String(thetaP) + " ");
  Serial.println("Yaw Angle: " + String(thetaY) + " ");

  // Store angles in Modbus registers
  modbus_regs[0] = (uint16_t)(thetaP * 100);
  modbus_regs[1] = (uint16_t)(thetaY * 100);

  // Enable RS485 transmit mode
  digitalWrite(MAX485_DE, HIGH);
  digitalWrite(MAX485_RE_NEG, HIGH);

  slave.poll(modbus_regs, 2);

  // Enable RS485 receive mode
  digitalWrite(MAX485_DE, LOW);
  digitalWrite(MAX485_RE_NEG, LOW);

  delay(1000);
}
```

Listing 14: Main code

```
#include <Adafruit_VL53L0X.h>
#include <Wire.h>
#include <Arduino.h>

int sensor1, sensor2, sensor3, sensor4;
float sensor_readings[4] = { 0, 0, 0, 0};

// objects for the vl53l0x
Adafruit_VL53L0X lox1 = Adafruit_VL53L0X();
Adafruit_VL53L0X lox2 = Adafruit_VL53L0X();
Adafruit_VL53L0X lox3 = Adafruit_VL53L0X();
Adafruit_VL53L0X lox4 = Adafruit_VL53L0X();

// this holds the measurement
VL53L0X_RangingMeasurementData_t measure1;
VL53L0X_RangingMeasurementData_t measure2;
VL53L0X_RangingMeasurementData_t measure3;
VL53L0X_RangingMeasurementData_t measure4;

/*
    Reset all sensors by setting all of their XSHUT pins low for delay(10)
```

```
       , then set all XSHUT high to bring out of reset
    Keep sensor #1 awake by keeping XSHUT pin high
    Put all other sensors into shutdown by pulling XSHUT pins low
    Initialize sensor #1 with lox.begin(new_i2c_address) Pick any number
       but 0x29 and it must be under 0x7F. Going with 0x30 to 0x3F is
       probably OK.
    Keep sensor #1 awake, and now bring sensor #2 out of reset by setting
       its XSHUT pin high.
    Initialize sensor #2 with lox.begin(new_i2c_address) Pick any number
       but 0x29 and whatever you set the first sensor to
 */

void TCA9548A(uint8_t bus) {
  Wire.beginTransmission(0x70);  // TCA9548A address is 0x70
  Wire.write(1 << bus);          // send byte to select bus
  Wire.endTransmission();
}

void setID() {

  // initing LOX1
  TCA9548A(0);
  if (!lox1.begin(0x29)) {
    // Serial.println(F("Failed to boot first VL53L0X"));
    while (1)
      ;
  }

  //initing LOX2
  TCA9548A(1);
  if (!lox2.begin(0x29)) {
    // Serial.println(F("Failed to boot second VL53L0X"));
    while (1)
      ;
  }

  // //initing LOX3
  TCA9548A(2);
  if (!lox3.begin(0x29)) {
    // Serial.println(F("Failed to boot third VL53L0X"));
    while (1)
      ;
  }


  // //initing LOX4
  TCA9548A(3);
  if (!lox4.begin(0x29)) {
    // Serial.println(F("Failed to boot fourth VL53L0X"));
    while (1)
      ;
  }
}

```

```
float sensor_1() {
  TCA9548A(0);
  lox1.rangingTest(&measure1, false);
  if (measure1.RangeStatus != 4) {
    sensor1 = measure1.RangeMilliMeter;
  } else {
    sensor1 = 4000;
  }
  return sensor1;
}
float sensor_2() {
  TCA9548A(1);
  lox2.rangingTest(&measure2, false);
  if (measure2.RangeStatus != 4) {
    sensor2 = measure2.RangeMilliMeter;
  } else {
    sensor2 = 4000;
  }
  return sensor2;
}

float sensor_3() {
  TCA9548A(2);
  lox3.rangingTest(&measure3, false);
  if (measure3.RangeStatus != 4) {
    sensor3 = measure3.RangeMilliMeter;
  } else {
    sensor3 = 4000;
  }
  return sensor3;
}

float sensor_4() {
  TCA9548A(3);
  lox4.rangingTest(&measure4, false);
  if (measure4.RangeStatus != 4) {
    sensor4 = measure4.RangeMilliMeter;
  } else {
    sensor4 = 4000;
  }
  return sensor4;
}
```

Listing 15: Implementation of the distance measurement library

```
#ifndef DISTANCE
#define DISTANCE
#include "distance.h"
#include <Arduino.h>
#include <Adafruit_VL53L0X.h>
#include <Wire.h>

void setID();
float sensor_1();
float sensor_2();
```

```
11 float sensor_3();
12 float sensor_4();
13
14 #endif
```

Listing 16: Header file for the distance measurement library

3.11 March 20, 2024

We implemented the above developed code on an Arduino Uno board to ensure its functionality. We attached the four Time-of-Flight (ToF) sensors to a dot board and connected their inputs using a multiplexer. We monitored the outputs through the serial monitor, confirming that the required angles were properly calculated.

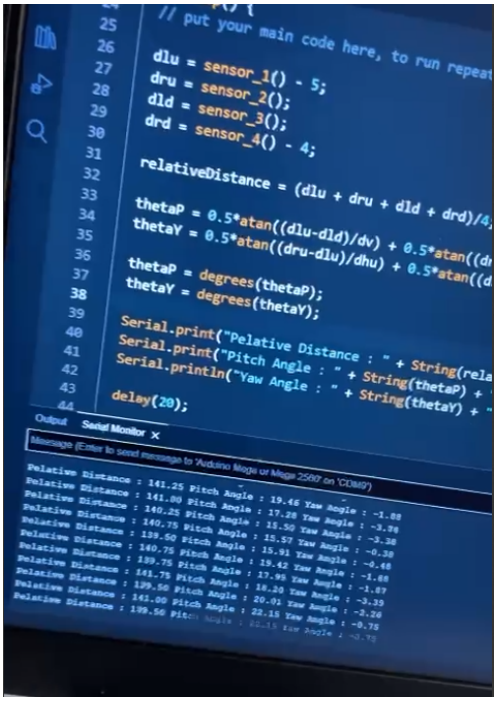


Figure 1: Serial Monitor Output

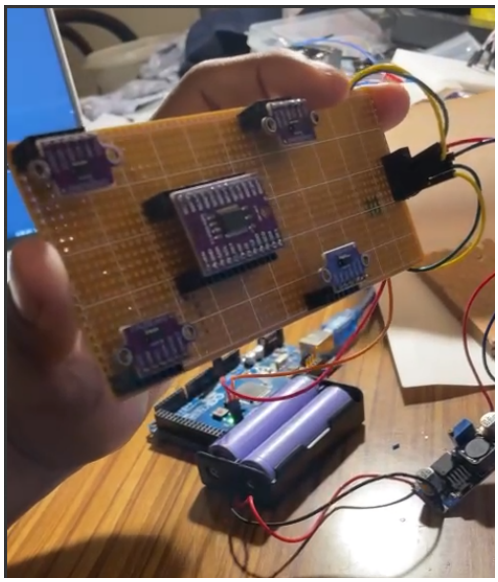


Figure 2: Sensor Arrangement

Week 6 (Library Optimization)

3.12 March 24, 2024

During our testing phase with the VL53L0X distance sensor, we observed that the library provided for its use consumes over 50% of our available memory resources. This raised concerns about memory efficiency. To address this issue, we explored a more direct approach by programming the sensor using register-level interactions instead of relying on the library. By accessing the sensor's hardware registers directly, we bypass the overhead associated with library functions, potentially leading to better memory utilization.

3.13 March 25, 2024

This method involves writing code that communicates directly with the sensor's registers, enabling us to control its functionalities without the overhead of library functions. While this approach demands a deeper understanding of the sensor's specifications and more meticulous coding, it offers the prospect of significantly reducing memory usage while maintaining desired functionality. Implementing this approach grants us finer control over the sensor's operations and allows for tailored optimizations suited to our specific needs.

3.14 March 26, 2024

Aspect	Library Usage	Register Level Programming
Memory Usage	The library consumes over 50% of available memory resources, potentially limiting space for other functionalities.	Directly accessing memory registers typically results in lower memory usage compared to using high-level libraries. This can be advantageous in memory-constrained environments.
Programming Complexity	Utilizing high-level libraries simplifies programming by abstracting away low-level details. Developers can use pre-defined functions and APIs, reducing complexity.	Programming with memory registers requires a deep understanding of the microcontroller's architecture and the sensor's specifications. It involves precise manipulation of memory addresses and bit-level operations, making it more complex.
Control and Flexibility	While libraries offer convenience, they may limit control and customization options. Developers are often restricted to the functionalities and configurations provided by the library.	Interacting with memory registers provides granular control over the sensor's functionalities. Developers have the flexibility to customize code according to specific project requirements.
Performance	Performance may vary depending on the efficiency of the library implementation. While libraries generally provide stable performance, they may introduce overhead due to abstraction layers.	Programming with memory registers can result in optimized performance, as it allows for direct and efficient communication with the sensor hardware.
Resource Efficiency	High-level libraries may consume additional resources, impacting both memory and processing efficiency. This can be a concern in resource-constrained embedded systems.	Leveraging memory registers can lead to efficient resource utilization, particularly in terms of memory and processing power.

Table 1: Comparison between Library Usage and Register Level Programming for VL53L0X Sensor

When prioritizing low complexity and easy debugging, high-level libraries often offer a more favorable solution:

1. **Simplicity**: High-level libraries abstract away low-level details, simplifying the development process. They provide pre-defined functions and APIs, which developers can easily integrate into their projects without needing an in-depth understanding of the underlying hardware architecture or sensor specifications.

2. **Ease of Use**: With high-level libraries, developers can focus on implementing the desired functionalities rather than worrying about low-level programming intricacies. This streamlined approach accelerates development and reduces the likelihood of errors or bugs introduced due to complex manual coding.

3. **Debugging Support**: High-level libraries are often well-documented and widely supported by the community. If issues arise during development or debugging, developers can rely on

documentation, forums, and community resources to troubleshoot problems efficiently. Additionally, many libraries come with built-in error handling mechanisms, making it easier to identify and address issues.

4. **Faster Iteration**: High-level libraries enable rapid prototyping and iteration by providing ready-to-use functionalities. Developers can quickly test and iterate on their code without the need for extensive low-level debugging, resulting in shorter development cycles and faster time-to-market.

Week 7 (Modbus Transmitter Development)

3.15 March 31, 2024

Understanding Modbus and Its Importance

Modbus is a serial communication protocol designed for industrial use, created in 1979 by Modicon. It facilitates communication between electronic devices and is widely adopted due to its simplicity, reliability, and support for both serial (RS232, RS485) and TCP/IP communications. In industrial settings, Modbus is crucial for enabling different devices, such as sensors and controllers, to exchange information seamlessly. Its robust error-checking and standardized format ensure accurate and efficient data transmission, minimizing the risk of communication errors that could disrupt operations.

3.16 April 4, 2024

To implement Modbus in our project, we used the MAX485 TTL to RS485 Converter Module to convert TTL signals from sensors into RS485 signals suitable for long-distance communication. The ATmega2560 microcontroller's TX and RX pins are connected to the DI and RO pins of the MAX485 module. The DE (Data Enable) and RE (Receive Enable) pins on the MAX485 are controlled by digital pins on the ATmega2560 to manage data flow direction. The RS485 output pins (A and B) from the MAX485 module connect to the robot control panel's RS485 input, ensuring accurate sensor data transmission.

3.17 April 5, 2024

To verify the Modbus setup, we used a USB to RS485 Communication Module to connect the RS485 bus to a PC. This setup allows for monitoring and debugging of data transmission. By connecting the USB side to the PC and the RS485 side to the RS485 bus, we use terminal software or Modbus Poll to inspect the data flow. Modbus Poll, a Modbus master simulator, helps send queries and visualize responses from the RS485 bus devices. This ensures the integrity and accuracy of data before final integration with the robot control panel, enabling real-time monitoring and prompt issue resolution.

Week 8 (Component Selection)

3.18 April 7, 2024

The components were selected based on their functionality, compatibility, market availability, feasibility, robustness, ease of soldering, and cost-effectiveness. Alternative options were considered to ensure optimal performance and cost-effectiveness.

Microcontroller

- **Selected Component:** ATMEGA2560-16AU
- **Description:** 8-bit AVR Microcontroller, 4.5-5.5V, 16MHz, 256KB Flash, 4KB EEPROM, 8KB SRAM, 86 GPIO pins, 100-pin TQFP, Industrial Grade (-40°C to 85°C), Pb-Free
- **Purpose:** Main processing unit for data handling and communication.
- **Reason for Selection:** High GPIO count, ample memory, industrial grade suitable for robust applications.
- **Market Availability:** Widely available.
- **Feasibility:** High, commonly used microcontroller.
- **Cost-Effectiveness:** Economical and reliable.
- **Alternative Considered:** PIC18F46K22 not selected due to lower GPIO count and smaller memory capacity.

3.19 April 9, 2024

Distance Measurement Sensor

- **Selected Component:** VL53L0X
- **Description:** Time-of-Flight (ToF) Laser Rangefinder Module
- **Purpose:** Provides accurate distance measurements for applications requiring precise ranging and object detection.
- **Reason for Selection:**The VL53L0X offers high-precision distance measurement capabilities with a range of up to 2 meters. It is compact and easy to integrate into various projects, making it ideal for applications such as robotics, drones, and proximity sensing. The sensor's ability to operate in a wide range of lighting conditions and its immunity to ambient light interference enhance its reliability and performance.
- **Market Availability:** Widely available from major suppliers such as Digi-Key, Mouser, and other electronic component distributors.
- **Feasibility:** High, due to its common use in various applications and extensive documentation and support available from the manufacturer.
- **Cost-Effectiveness:** Cost-effective compared to similar sensors, offering a good balance between performance and price. Its widespread availability also contributes to its cost-effectiveness.
- **Alternative Considered:** VL6180X

Multiplexer

- **Selected Component:** PCA9546ADG4
- **Description:** 4-Channel I2C and SMBus Multiplexer with Reset Functions, 2.3 to 5.5V, -40 to 85°C, 16-pin SOIC (D), Green (RoHS no Sb/Br)
- **Purpose:** Multiplexing I2C signals from multiple TOF sensors to the microcontroller.

- **Reason for Selection:** Efficient channel management, compact SOIC package.
- **Market Availability:** Widely available.
- **Feasibility:** High, commonly used in I2C applications.
- **Cost-Effectiveness:** Economical and reliable.
- **Alternative Considered:** VL6180X
- **Reason for Not Selecting:**The VL6180X has a shorter range and is more suited for proximity sensing and ambient light measurement rather than precise long-distance ranging, making the VL53L0X a better fit for applications requiring accurate distance measurements.

3.20 April 11, 2024

Crystal Oscillator

- **Selected Component:** ABLS2-16.000MHZ-D4Y-T
- **Description:** HC/49US (AT49) Low Profile Surface Mount Microprocessor Crystal
- **Purpose:** Provides clock signal for the microcontroller.
- **Reason for Selection:** Reliable frequency stability, compact package.
- **Market Availability:** Easily available.
- **Feasibility:** High, standard component for microcontroller applications.
- **Cost-Effectiveness:** Economical and reliable.

USB Connector

- **Selected Component:** USBA1HSW6
- **Description:** USB-A (USB TYPE-A) Receptacle Connector, 4 Position
- **Purpose:** USB connectivity for data transfer or power supply.
- **Reason for Selection:** Standard USB-A receptacle with through-hole mounting.
- **Market Availability:** Readily available.
- **Feasibility:** High, standard component.
- **Cost-Effectiveness:** Economical and reliable.

3.21 April 12, 2024

Resistors

- **Selected Component:** RC0805JR-070RL
- **Description:** Chip Resistor Ohm, Jumper, ±5%, 0.125W, 0805 (2012 Metric), RoHS, Tape and Reel
- **Purpose:** Jumper resistors for circuit configuration.

- **Reason for Selection:** Standard 0805 size for easy placement and soldering.
- **Market Availability:** Widely available.
- **Feasibility:** High, due to standard use.
- **Cost-Effectiveness:** Economical and reliable.

Headers

- **Selected Component:** M22-5320305R
- **Description:** Header, 3-Pin, Dual row
- **Purpose:** Connection interface for external components or programming.
- **Reason for Selection:** Standard dual-row header for reliable connections.
- **Market Availability:** Commonly available.
- **Feasibility:** High, standard component with broad availability.
- **Cost-Effectiveness:** Economical and reliable.

Capacitors

- **Selected Component:** C1206C104K2RACTU
- **Description:** CAP CER 0.1UF/0.3UF 200V 10% X7R 1206
- **Purpose:** General-purpose decoupling and filtering for stable power supply to various components.
- **Reason for Selection:** X7R dielectric provides stable capacitance; 1206 footprint suits compact PCBs.
- **Market Availability:** Widely available from major suppliers.
- **Feasibility:** High due to common use.
- **Cost-Effectiveness:** Balanced performance and price.
- **Alternative Considered:** GRM31CR61A106KE15L not suitable due to different dielectric material.

LED

- **Selected Component:** CMP-1488-00009-1
- **Description:** Typical RED GaAs LED
- **Purpose:** Status indication.
- **Reason for Selection:** Reliable, widely used, good visibility, low power consumption.
- **Market Availability:** Readily available.
- **Feasibility:** High in various applications.
- **Cost-Effectiveness:** Economical and reliable.

Week 9 (Schematic and PCB Design)

3.22 April 15, 2024

Today marks the beginning of the schematic design phase for our PCB project. Following careful component selection to meet project requirements, our team has commenced the schematic layout process.

- Translating component specifications into circuit diagrams.
- Ensuring compatibility and functionality through detailed design review.

The next steps involve thorough validation and verification of the schematic before proceeding to PCB layout. Coordination among team members remains crucial to ensure alignment with project timelines and goals.

3.23 April 17, 2024

After thorough validation and verification of the schematic, we have begun the process of placing components on the PCB. Our primary focus is to maintain the minimum size of the PCB while ensuring optimal layout for functionality and manufacturability.

- Started placing components strategically to optimize space utilization.
- Ensured adherence to design constraints and specifications during placement.
- Implemented initial routing considerations to minimize signal interference and achieve efficient signal paths.

Two days into component placement, we transitioned to routing using JL PCB rules to further refine the layout:

- Applied JL PCB rules for routing to ensure signal integrity and minimize noise.
- Continued to prioritize compactness while maintaining clear separation between high-speed and low-speed signals.
- Iteratively reviewed and adjusted routing paths to optimize performance and reliability.

The next phase involves comprehensive review and validation of the routed design before finalizing for production.

3.24 April 20, 2024

Today, we completed the finalization and validation of the PCB design. Following the meticulous routing according to JL PCB rules, we have successfully generated Gerber files and NC drill files required for manufacturing.

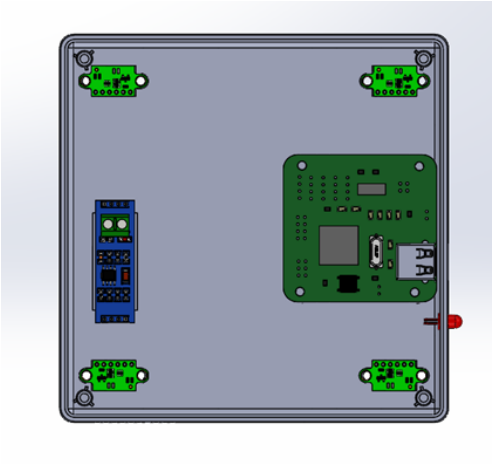
- Conducted a detailed review and validation of the routed PCB design to ensure it meets all specifications and functional requirements.
- Verified signal integrity, power distribution, and thermal management aspects of the design.
- Prepared comprehensive documentation, including assembly drawings and manufacturing notes, to support smooth production.

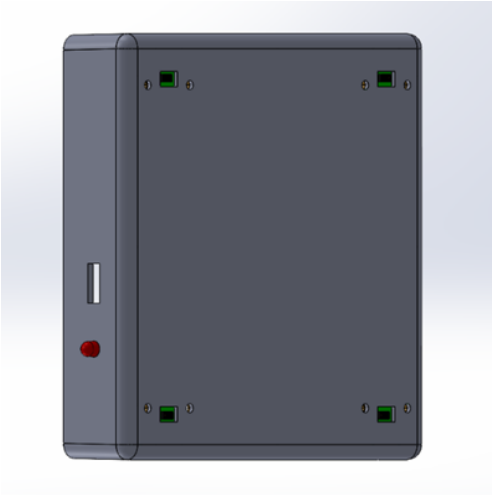
The finalized design is now ready for manufacturing in China. The Gerber files and NC drill files have been carefully checked and packaged for transmission to our manufacturing partner.

Week 10 (Enclosure Design)

3.25 April 22, 2024

Today, we created a SolidWorks design for a simple box-like enclosure to incorporate all the TOF sensors, aimed at demonstrating our concept. Initially, we considered building a vacuum gripper, but due to feasibility issues, we focused on the enclosure. After exploring various conceptual designs, we selected a feasible option and modeled it in SolidWorks. However, the design required the TOF sensors to be spaced 10cm apart, resulting in excessive use of material and space. This issue needs to be addressed in future iterations to optimize the design.





3.26 April 26, 2024

As the project advanced, we discussed the possibility of creating an adjustable enclosure. The aim is to provide users with the flexibility to modify the product components based on the dimensions of different vacuum grippers. This new design would allow the four TOF sensors to be re positioned as needed and would include a box housing the PCB, which could be attached to the gripper. Our objective is to ensure compatibility with a wide range of vacuum grippers without compromising on performance or increasing the overall size. This adjustable enclosure concept is intended to enhance the versatility and user-friendliness of our product, making it adaptable to various applications and user requirements.

Week 11 (Enclosure Design)

3.27 April 28, 2024

Today, we extensively discussed the challenges encountered while developing the adjustable enclosure for our project:

- Variability in Gripper Sizes: The significant size differences among vacuum grippers necessitated a highly adaptable sensor arrangement.
- Attachment Reliability: Finding a universal attachment method that could accommodate the diverse designs of grippers proved challenging.
- Sensor Mobility: Design complexity increased due to the requirement for TOF sensors to move in two directions.

- Mechanical Durability: Ensuring the mechanical strength and stability of the adjustable components posed significant challenges.

These challenges prompted us to reassess our approach. We concluded that focusing on a specific vacuum gripper model would enable us to devise a more robust and efficient design, ensuring optimal performance in our product.

3.28 April 30, 2024

Today's efforts were focused on several critical tasks aimed at advancing our project:

We began by designing a sample vacuum gripper that incorporates designated mounting spots specifically tailored for the TOF sensors. This meticulous approach ensures that the sensors maintain an unobstructed line of sight, thereby preserving the gripper's functionality and enhancing overall performance.

In parallel, we leveraged and enhanced an existing gripper model from this link to meet our precise project requirements. This model provided a solid foundation that we modified to seamlessly integrate with our sensor arrangement and operational needs.

Additionally, we developed a separate enclosure designed specifically for the PCB. This enclosure features a strategically placed USB port that facilitates convenient data retrieval directly from the PCB. Inside this enclosure, the TOF sensors connect directly to the PCB, ensuring streamlined functionality. The entire assembly can be securely attached to the gripper using nuts, which guarantees stability during operation.

These integrated advancements represent a significant stride towards achieving our project goals. They ensure that our sensor technology operates effectively within the context of the vacuum gripper's functionality, paving the way for robust and reliable performance in practical applications.