Department of Electronic & Telecommunication Engineering,
University of Moratuwa, Sri Lanka.

3D Mapping Device for Object Tracking Design Documentation - Group B

Submitted By:
210728C Wijewickrama W.K.D.D.D.
210031H Amarasinghe A.A.W.L.R.

Submitted in partial fulfillment of the requirements for the module

EN 2160 : Electronic Design Realization

20th June 2024

Contents

1 Introduction

The 3D Scanner device represents a cutting-edge solution for precise and efficient three-dimensional data acquisition in various applications. This design document comprehensively outlines the component selection process and daily progress of the project. It encapsulates the combined work of our multi-disciplinary team, leveraging expertise in optical engineering, electronics, software development, and human-centered design principles.

The device features a Time-of-Flight (ToF) sensor designed for precise measurement of distances and angles. It utilizes UART communication via ATMEGA 2560 microcontroller pins connected to a USB-to-Serial converter, enabling seamless interfacing with computers equipped with USB ports for data transmission and visualization. This setup ensures efficient handling of distance measurement data and robust communication capabilities, making the device suitable for applications requiring accurate spatial data acquisition and analysis

2 Component Selection and Justifications

2.1 Microcontroller Selection

ATMEGA 2560 MCU

The ATMEGA 2560 microcontroller unit (MCU) was selected as the central processing unit for the 3D surround scanner. This microcontroller was chosen for its adequate memory, processing speed, and overall reliability.

Specifications:

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
- Non-volatile Program and Data Memories
- JTAG (IEEE std. 1149.1 compliant) Interface
- 51/86 Programmable I/O Lines
- Temperature Range: -40°C to 85°C Industrial

Key Features:

- **Memory:** The ATMEGA 2560 offers 256 KB of Flash memory, 8 KB of SRAM, and 4 KB of EEPROM, providing sufficient space for handling the initial stages of data processing.
- **Processing Speed:** It operates at a clock speed of 16 MHz, which meets the demands of real-time data collection and processing required for the 3D mapper.
- **Reliability:** Known for its stable performance, the ATMEGA 2560 is widely used in various applications, making it a dependable choice for our project.

Reasons for Selection:

- **Versatility:** The ATMEGA 2560 supports a wide range of peripherals and interfaces, making it adaptable to various components used in the 3D mapper project.
- **Community Support:** The ATMEGA 2560 has extensive documentation and a large community of users, providing valuable resources and support for troubleshooting and development.

- **Cost-Effective:** Compared to other microcontrollers with similar capabilities, the ATMEGA 2560 offers a good balance between performance and cost, making it an economical choice for the project.
- **Ease of Programming:** The microcontroller is compatible with the Arduino platform, which simplifies programming and prototyping, allowing for faster development cycles.
- **Proven Track Record:** The ATMEGA 2560 has been used in numerous successful projects, indicating its reliability and robustness in real-world applications.

Despite these advantages, the memory capacity of the ATMEGA 2560 was insufficient to store the entire dataset generated during the scanning process. Therefore, the data readings had to be transmitted incrementally to a connected computer for further processing and storage.

2.2 Time-of-Flight (ToF) Sensors

VL53L0 ToF Sensors

Two Time-of-Flight sensors were selected for the 3D mapper to measure the distance of objects with high precision and speed.

Specifications

- **VDD:** Regulated 2.8 V output. Almost 150 mA available to power external components.
- **VIN:** The main 2.6 V to 5.5 V power supply connection. The SCL and SDA level shifters pull the I2C wire high to this level.
- **GND:** The ground (0 V) connection for power supply. I2C control source must also share a common ground with this board.
- **SDA:** Level-shifted I2C data line; HIGH is VIN, LOW is 0 V.
- **SCL:** Level-shifted I2C clock line; HIGH is VIN, LOW is 0 V.
- **XSHUT:** This pin is an active-low shutdown input; the board pulls it up to VDD to enable the sensor by default.

Reasons for Selection:

- **High Precision:** ToF sensors are capable of measuring distances accurately, which is crucial for creating a detailed 3D map.
- **Speed:** These sensors can take measurements rapidly, enabling real-time data collection and processing.
- **Range:** ToF sensors can cover a sufficient range suitable for the intended mapping application.

2.3 I2C Multiplexer

I2C Multiplexer

An I2C multiplexer was used to connect the two ToF sensors to the ATMEGA 2560.

Specifications:

- 1-to-8 Bidirectional Translating Switches
- I2C Bus and SMBus Compatible
- Allows Voltage-Level Translation Between 1.8-V, 2.5-V, 3.3-V, and 5-V Buses
- Operating Power-Supply Voltage Range of 1.65-V to 5.5-V
- Low RON Switches

Reasons for Selection:

- **Multiple I2C Devices:** The multiplexer allows multiple I2C devices to be connected to a single I2C bus, expanding the number of sensors that can be used simultaneously.
- **Simplifies Wiring:** Reduces the complexity of wiring by allowing easy addition of sensors without requiring additional I2C ports on the microcontroller.
- **Efficient Communication:** Ensures efficient communication between the microcontroller and the sensors, maintaining the integrity and speed of data transfer.

2.4 USB Serial Communication

CH340C Chip

The CH340C chip was chosen for USB serial communication between the ATMEGA 2560 and the connected computer.

Specifications:

- Full-speed USB device interface, compatible with USB V2.0
- Fully compatible with serial port applications
- Supports common MODEM contact signals
- Provides RS232, RS485, RS422 interfaces
- Supports 5V and 3.3V power supply voltages

Reasons for Selection:

- **Reliable Communication:** Provides a reliable interface for serial communication over USB, essential for transmitting data to the computer for further processing.
- **Compatibility:** Compatible with a wide range of operating systems, ensuring ease of integration and use.
- **Cost-Effective:** The CH340C is a cost-effective solution for USB to serial conversion, making it an economical choice for the project.

2.5 Stepper Motor

Nema17

The NEMA 17 stepper motor is renowned for its versatility and reliability in various applications requiring precise motion control

Specifications

- **Holding Torque:** 0.35 Nm at continuous current, 0.5 Nm at peak current.
- **Continuous Output Current:** 1.8 A.
- **Step Angle:** 1.8°.
- **Rotor Inertia:** 57 g·cm^2.
- **Weight:** 0.37 kg.

Reasons for Selection

1. **Compact Size:** Suitable for applications with limited space, such as 3D printers and small CNC machines.
2. **Moderate Torque:** Provides adequate torque (0.35 Nm continuous, 0.5 Nm peak) for precise motion control and handling moderate loads.
3. **Precise Control:** 1.8° step angle and high subdivision accuracy (1–32 steps) enable fine resolution and accurate positioning.
4. **Versatility:** Compatible with various driver modules and control systems, offering flexibility in design and integration.
5. **Cost-Effectiveness:** Balances performance and cost effectively, making it a practical choice for both industrial and DIY projects.
6. **Reliability:** Known for robust construction and durability, ensuring reliable performance in demanding environments.
7. **Availability:** Widely available from multiple manufacturers, providing easy access to parts and support.

2.6 Stepper Motor Driver

TB6600

The TB6600 stepper motor driver is a versatile choice for controlling stepper motors in various applications. It offers a range of features suitable for precise and powerful motor control, making it ideal for both industrial and DIY projects.

Specifications:

- **Model:** TB6600
- **Compatible Motors:** Nema 17/23/34 (42/57/86 Stepper Motor)
- **Control Signal:** 3.3VDC–24VDC
- **Subdivision Accuracy:** 1–32
- **Output Current:** 4A
- **Voltage:** 9VDC–40VDC

Reasons for Selection:

- **Motor Compatibility:** Compatible with Nema 17, 23, 34 motors.
- **Output Current:** Up to 4A for sufficient torque.
- **Voltage Range:** Supports 9VDC–40VDC for flexible power options.
- **Subdivision Accuracy:** Precision from 1–32 steps.
- **Control Signal:** Compatible with 3.3VDC–24VDC signals.
- **Reliability:** Known for robust performance in industrial applications.

2.7 Power Step-Down

R-78CK-0.5 12V-5V

A 12V-5V power step-down module was used to supply power to the ATMEGA 2560 and other components.

Specifications

- Efficiency up to 96%
- Pin-out compatible with LM78xx linears
- Compact package
- Wide input range (5V - 40V)
- Short circuit protection, thermal shutdown
- Low ripple and noise

Reasons for Selection:

- **Voltage Regulation:** Ensures stable voltage supply to the microcontroller and sensors, which is crucial for reliable operation.
- **Efficiency:** High efficiency in converting 12V input to 5V output, minimizing power loss and heat generation.
- **Compact Size:** The module is compact, making it easy to integrate into the overall design without adding significant bulk.

2.8 Clock Oscillator

16 MHz Crystal Oscillator

A 16 MHz crystal oscillator was used to provide the clock signal for the ATMEGA 2560.

Reasons for Selection:

- **Precision Timing:** Provides a stable and precise clock signal necessary for the accurate operation of the microcontroller.
- **Frequency Stability:** Ensures consistent performance of the microcontroller by maintaining a stable clock frequency.
- **Compatibility:** Specifically chosen to match the operational requirements of the ATMEGA 2560.

2.9 Decoupling Capacitors

100µF Decoupling Capacitors

100µF decoupling capacitors were used to stabilize the power supply to the microcontroller and other components.

Reasons for Selection:

- **Noise Reduction:** Help to filter out noise and prevent voltage spikes, ensuring a stable power supply.
- **Improved Performance:** Enhances the overall performance and reliability of the microcontroller by maintaining a clean power signal.
- **Protection:** Protects sensitive components from transient voltage fluctuations.

2.10 Pull-up Resistors

2.2k Resistors

2.2k resistors were used in the I2C lines to ensure proper signal levels.

Reasons for Selection:

- **Pull-up Function:** These resistors act as pull-up resistors for the I2C bus, ensuring that the lines are correctly biased and that signals are accurately interpreted.
- **Signal Integrity:** Maintains the integrity of the I2C signals, preventing erroneous data transmission.
- **Standard Value:** 2.2k ohms is a standard value for pull-up resistors in I2C communication, ensuring compatibility and reliable operation.

3 Setting Up Environment

3.1 Components

ToF Sensor

A Time-of-Flight sensor to measure distance and angles.

UART Communication Interface

UART pins on the ATMEGA 2560 connected to a USB-to-Serial converter (e.g., FTDI or CP2102) to interface with the computer.

Computer

A computer with USB ports to receive data from the microcontroller and perform visualization.

3.2 Software Requirements

Python

Install Python 3.10.1 or later on your computer.

PySerial

A Python library to handle serial communication.
Installation: `pip install pyserial`

Matplotlib

A Python library for creating static, animated, and interactive visualizations.
Installation: `pip install matplotlib`

NumPy

A Python library for numerical operations.
Installation: `pip install numpy`

3D Visualization Libraries

Matplotlib supports 3D plotting via `mpl_toolkits.mplot3d`.
Installation: `pip install matplotlib` (already mentioned)

4 Daily Progress

4.1 Distance Measurement using TOF sensor (4 - 10 March 2024)

4.1.1 Arduino Code

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

Adafruit_VL53L0X lox = Adafruit_VL53L0X();

void setup() {
  Serial.begin(115200);

  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1);
  }

  Serial.println(F("VL53L0X test"));
}

void loop() {
  VL53L0X_RangingMeasurementData_t measure;

  lox.rangingTest(&measure, false);

  if (measure.RangeStatus != 4) {  // phase failures have incorrect data
    Serial.print(F("Distance (mm): "));
    Serial.println(measure.RangeMilliMeter);
  } else {
    Serial.println(F("Out of range"));
  }

  delay(100);
}
```

Listing 1: Arduino Code for VL53L0X Sensor

4.1.2 Code Explanation

Include Libraries

- `#include <Wire.h>`
  - `Wire.h`: Library to communicate with I2C devices.
- `#include <Adafruit_VL53L0X.h>`
  - `Adafruit_VL53L0X.h`: Library specific to the Adafruit VL53L0X sensor.

Create Sensor Object

- `Adafruit_VL53L0X lox = Adafruit_VL53L0X();`
  - Creates an instance of the VL53L0X sensor named `lox`.

Setup Function

```
void setup() {
  Serial.begin(115200);

  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1);
  }

  Serial.println(F("VL53L0X test"));
}
```

- Initializes serial communication at 115200 baud.
- Tries to initialize the sensor using `lox.begin()`.
- If the sensor fails to initialize, it prints an error message and enters an infinite loop.
- If initialization is successful, it prints a success message.

Loop Function

```
void loop() {
  VL53L0X_RangingMeasurementData_t measure;

  lox.rangingTest(&measure, false);

  if (measure.RangeStatus != 4) {
    Serial.print(F("Distance (mm): "));
    Serial.println(measure.RangeMilliMeter);
  } else {
    Serial.println(F("Out of range"));
  }

  delay(100);
}
```

- Declares a variable `measure` of type `VL53L0X_RangingMeasurementData_t` to store the measurement data.
- Calls `lox.rangingTest(&measure, false)` to perform a ranging test and store the result in `measure`.
- Checks the `RangeStatus` to determine if the measurement is valid (status not equal to 4).
  - If valid, prints the measured distance in millimeters.
  - If not valid (out of range), prints an error message.
- Delays for 100 milliseconds before taking another measurement.

4.2 Stepper Motor Check (11 - 17 March 2024)

4.2.1 Arduino Code

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

// Define stepper motor connections
#define STEP_PIN 10
#define DIR_PIN 11

#define STEP_PIN_2 12
#define DIR_PIN_2 13

#define STEPS_PER_REVOLUTION 200
#define ANGLE_INCREMENT 1.8

// Create a stepper object
AccelStepper stepper(AccelStepper::DRIVER, STEP_PIN, DIR_PIN);
AccelStepper stepper2(AccelStepper::DRIVER, STEP_PIN_2, DIR_PIN_2);

void setup() {
  // Set up the speed and acceleration of the stepper motor
  stepper.setMaxSpeed(1000);
  stepper.setAcceleration(500);

  stepper2.setMaxSpeed(1000);
  stepper2.setAcceleration(500);

  // Set the motor to run continuously
  stepper.moveTo(0);
  stepper2.moveTo(0);
}

void loop() {
  // Run the stepper motor
  stepper.run();
  stepper2.run();
}
```

Listing 2: Arduino Code for Two Stepper Motors with AccelStepper

4.2.2 Code Explanation

Include Libraries

- `#include <AccelStepper.h>`
  - `AccelStepper.h`: Library for controlling stepper motors.
- `#include <Wire.h>`
  - `Wire.h`: Library to communicate with I2C devices.
- `#include <Adafruit_VL53L0X.h>`
  - `Adafruit_VL53L0X.h`: Library specific to the Adafruit VL53L0X sensor.

Define Stepper Motor Connections

```
    \item \texttt{\#define STEP_PIN 10}
    \item \texttt{\#define DIR_PIN 11}
```

- Defines the step and direction pins for the first stepper motor.

```
    \item \texttt{\#define STEP_PIN\_2 12}
    \item \texttt{\#define DIR_PIN\_2 13}
```

- Defines the step and direction pins for the second stepper motor.

```
    \item \texttt{\#define STEPS\_PER\_REVOLUTION 200}
    \item \texttt{\#define ANGLE\_INCREMENT 1.8}
```

- Defines the number of steps per revolution and the angle increment.

Create Stepper Objects

```
AccelStepper stepper(AccelStepper::DRIVER, STEP_PIN, DIR_PIN);
```

- Creates an instance of the `AccelStepper` class for the first stepper motor.

```
AccelStepper stepper2(AccelStepper::DRIVER, STEP_PIN\_2, DIR_PIN\_2);
```

- Creates an instance of the `AccelStepper` class for the second stepper motor.

Setup Function

```
void setup() {
  stepper.setMaxSpeed(1000);
  stepper.setAcceleration(500);

  stepper2.setMaxSpeed(1000);
  stepper2.setAcceleration(500);

  stepper.moveTo(0);
  stepper2.moveTo(0);
}
```

- `void setup()`
  - Initializes the setup function.
- `stepper.setMaxSpeed(1000);`
  - Sets the maximum speed for the first stepper motor.
- `stepper.setAcceleration(500);`
  - Sets the acceleration for the first stepper motor.
- `stepper2.setMaxSpeed(1000);`
  - Sets the maximum speed for the second stepper motor.

- `stepper2.setAcceleration(500);`
  - Sets the acceleration for the second stepper motor.
- `stepper.moveTo(0);`
  - Moves the first stepper motor to position 0.
- `stepper2.moveTo(0);`
  - Moves the second stepper motor to position 0.

Loop Function

```
void loop() {
  stepper.run();
  stepper2.run();
}
```

- `void loop()`
  - Initializes the loop function.
- `stepper.run();`
  - Runs the first stepper motor.
- `stepper2.run();`
  - Runs the second stepper motor.

4.3 3D Cloud Generation (18 - 24 March 2024)

4.3.1 Python Code

```python
import ast
import numpy as np
import pyvista as pv

# Open the file in read mode
with open('example.txt', 'r') as file:
    # Read the file content
    content = file.read()

# Use ast.literal_eval to convert the string to a 2D list
points = np.array(ast.literal_eval(content))

# Create a PolyData object
cloud = pv.PolyData(points)

# Plot the point cloud
cloud.plot()

# Create a 3D Delaunay triangulation of the point cloud
volume = cloud.delaunay_3d(alpha=0.6)

# Extract the outer surface of the volume
shell = volume.extract_geometry()

# Plot the resulting mesh
shell.plot()
```

Listing 3: Python Code for 3D Delaunay Triangulation

4.3.2 Code Explanation

4.3.3 Import Libraries

- `import ast`
  - `ast`: Library for safely evaluating strings containing Python expressions.
- `import numpy as np`
  - `numpy`: Library for numerical operations on arrays.
- `import pyvista as pv`
  - `pyvista`: Library for 3D plotting and mesh analysis.

Open and Read File

```python
with open('example.txt', 'r') as file:
    content = file.read()
```

- Opens the file `'example.txt'` in read mode and reads its content.

Convert String to 2D List

```python
points = np.array(ast.literal_eval(content))
```

- Safely evaluates the string content and convert it into a 2D list.
- Converts the 2D list to a NumPy array for further processing.

Create PolyData Object

```
cloud = pv.PolyData(points)
```

- Creates a `PolyData` object from the points array using PyVista.

Plot the Point Cloud

```
cloud.plot()
```

- Plots the point cloud using PyVista.

Create 3D Delaunay Triangulation

```
volume = cloud.delaunay_3d(alpha=0.6)
```

- Creates a 3D Delaunay triangulation of the point cloud with an alpha value of 0.6.

Extract Outer Surface

```
shell = volume.extract_geometry()
```

- Extracts the outer surface of the triangulated volume.

Plot the Resulting Mesh

```
shell.plot()
```

- Plots the resulting mesh of the outer surface using PyVista.

4.4 2D Scan (25 - 31 March 2024)

4.4.1 Arduino Code

```
#include <Wire.h>
#include <Adafruit_VL53L0X.h>
#include <Arduino.h>
#include "A4988.h"

#define STEPS_PER_REVOLUTION 200
#define ANGLE_INCREMENT 1.8

int Step = 10; //GPIO14---D5 of Nodemcu--Step of stepper motor driver
int Dire  = 11; //GPIO2---D4 of Nodemcu--Direction of stepper motor driver
int Sleep = 14; //GPIO12---D6 of Nodemcu-Control Sleep Mode on A4988
int MS1 = 13; //GPIO13---D7 of Nodemcu--MS1 for A4988
int MS2 = 16; //GPIO16---D0 of Nodemcu--MS2 for A4988
int MS3 = 15; //GPIO15---D8 of Nodemcu--MS3 for A4988

//Motor Specs
const int spr = 200; //Steps per revolution
int RPM = 100; //Motor Speed in revolutions per minute
int Microsteps = 1; //Stepsize (1 for full steps, 2 for half steps, 4 for quarter
    steps, etc)

//Providing parameters for motor control
A4988 stepper(spr, Dire, Step, MS1, MS2, MS3);

Adafruit_VL53L0X lox = Adafruit_VL53L0X();
//Stepper stepper(STEPS_PER_REVOLUTION, 2, 3, 4, 5);  // Adjust pin numbers
    accordingly

const int num_measurements = STEPS_PER_REVOLUTION;
float measurements[num_measurements][2];

String arrayToString(int arr[], int size) {
  String result = "{";
  for (int i = 0; i < size; i++) {
    result += String(arr[i]);
    if (i < size - 1) {
      result += ", ";
    }
  }
  result += "}";
  return result;
}

void setup() {
  Serial.begin(9600);
  Serial1.begin(9600); // Use Serial1 for communication on Arduino Mega

  pinMode(Step, OUTPUT); //Step pin as output
  pinMode(Dire,  OUTPUT); //Direcction pin as output
  pinMode(Sleep,  OUTPUT); //Set Sleep OUTPUT Control button as output
  digitalWrite(Step, LOW); // Currently no stepper motor movement
  digitalWrite(Dire, LOW);

  // Set target motor RPM to and microstepping setting
  //stepper.begin(RPM, Microsteps);

  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1);
  }

  pinMode(13, OUTPUT);  // Set pin 13 as an output

//  stepper.setSpeed(500);  // Adjust the speed as needed
```

```

  Serial.println(F("VL53L0X test with Stepper Motor"));
}

void loop() {
  digitalWrite(12, HIGH);
  digitalWrite(Sleep, HIGH); //A logic high allows normal operation of the A4988 by
      removing from sleep
  stepper.rotate(360);

  for (int i = 0; i < num_measurements; i++) {
    // Rotate stepper motor by ANGLE_INCREMENT degrees
  //  stepper.step(ANGLE_INCREMENT);

    // Take distance measurement
    VL53L0X_RangingMeasurementData_t measure;
    lox.rangingTest(&measure, false);

    if (measure.RangeStatus != 4) {  // phase failures have incorrect data
      measurements[i][0] = i * ANGLE_INCREMENT;
      measurements[i][1] = measure.RangeMilliMeter;
    } else {
      measurements[i][0] = i * ANGLE_INCREMENT;
      measurements[i][1] = 10000;
    }

    // No delay or minimal delay between steps
  }

  // Print the 2D array after 360 degrees rotation
  Serial.print("aaa[");
  for (int i = 0; i < num_measurements; i++) {
    Serial.print("[");
    Serial.print(measurements[i][0]);
    Serial.print(",");
    Serial.print(measurements[i][1]);
    Serial.print("],");
  }
  Serial.println("]");
  //Serial.println(arrayToString());

  // Reset the stepper motor to its initial position
//  stepper.step(-360 * STEPS_PER_REVOLUTION / 360);
  digitalWrite(13, LOW);

  // Delay before starting a new measurement
  delay(5000);
}
```

Listing 4: Arduino Code for VL53L0X Sensor with Stepper Motors and A4988 Driver

4.4.2 Code Explanation

Import Libraries

- `#include <Wire.h>`
  - Provides I2C communication.
- `#include <Adafruit_VL53L0X.h>`
  - Provides functions for the VL53L0X sensor.
- `#include <Arduino.h>`
  - Provides core Arduino functions.

- `#include "A4988.h"`
  - Provides functions for controlling the A4988 stepper motor driver.

Define Constants and Variables

- `#define STEPS_PER_REVOLUTION 200`
  - Defines the number of steps per revolution for the stepper motor.
- `#define ANGLE_INCREMENT 1.8`
  - Defines the angle increment for each step.
- `int Step = 10;`
  - Defines the GPIO pin for the step signal.
- `int Dire = 11;`
  - Defines the GPIO pin for the direction signal.
- `int Sleep = 14;`
  - Defines the GPIO pin for the sleep signal.
- `int MS1 = 13;`
  - Defines the GPIO pin for the MS1 signal.
- `int MS2 = 16;`
  - Defines the GPIO pin for the MS2 signal.
- `int MS3 = 15;`
  - Defines the GPIO pin for the MS3 signal.

Motor Specifications and Setup

- `const int spr = 200;`
  - Sets the steps per revolution for the motor.
- `int RPM = 100;`
  - Sets the motor speed in revolutions per minute.
- `int Microsteps = 1;`
  - Sets the microstepping mode.
- `A4988 stepper(spr, Dire, Step, MS1, MS2, MS3);`
  - Initializes the A4988 stepper motor driver.
- `Adafruit_VL53L0X lox = Adafruit_VL53L0X();`
  - Initializes the VL53L0X sensor.

Setup Function

```
void setup() {
  Serial.begin(9600);
  Serial1.begin(9600); // Use Serial1 for communication on Arduino Mega

  pinMode(Step, OUTPUT); //Step pin as output
  pinMode(Dire,  OUTPUT); //Direction pin as output
  pinMode(Sleep,  OUTPUT); //Set Sleep OUTPUT Control button as output
  digitalWrite(Step, LOW); // Currently no stepper motor movement
  digitalWrite(Dire, LOW);

  // Set target motor RPM and microstepping setting
  //stepper.begin(RPM, Microsteps);

  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1);
  }

  pinMode(13, OUTPUT);  // Set pin 13 as an output

//  stepper.setSpeed(500);  // Adjust the speed as needed

  Serial.println(F("VL53L0X test with Stepper Motor"));
}
```

- Initializes serial communication.
- Sets pin modes for the stepper motor driver.
- Initializes the VL53L0X sensor.
- Prints a test message to the serial monitor.

Main Loop

```
void loop() {
  digitalWrite(12, HIGH);
  digitalWrite(Sleep, HIGH); //A logic high allows normal operation of the A4988 by removing from sleep
  stepper.rotate(360);

  for (int i = 0; i < num_measurements; i++) {
    // Rotate stepper motor by ANGLE_INCREMENT degrees
  //  stepper.step(ANGLE_INCREMENT);

    // Take distance measurement
    VL53L0X_RangingMeasurementData_t measure;
    lox.rangingTest(&measure, false);

    if (measure.RangeStatus != 4) {  // phase failures have incorrect data
      measurements[i][0] = i * ANGLE_INCREMENT;
      measurements[i][1] = measure.RangeMilliMeter;
    } else {
      measurements[i][0] = i * ANGLE_INCREMENT;
      measurements[i][1] = 10000;
    }

    // No delay or minimal delay between steps
  }

  // Print the 2D array after 360 degrees rotation
  Serial.print("aaa[");
  for (int i = 0; i < num_measurements; i++) {
```

```
    Serial.print("[");
    Serial.print(measurements[i][0]);
    Serial.print(",");
    Serial.print(measurements[i][1]);
    Serial.print("],");
  }
  Serial.println("]");
  //Serial.println(arrayToString());

  // Reset the stepper motor to its initial position
//  stepper.step(-360 * STEPS_PER_REVOLUTION / 360);
  digitalWrite(13, LOW);

  // Delay before starting a new measurement
  delay(5000);
}
```

- Activates the A4988 driver.
- Rotates the stepper motor 360 degrees.
- Takes distance measurements with the VL53L0X sensor at each step.
- Stores measurements in a 2D array.
- Prints the measurements to the serial monitor in a JSON-like format.
- Resets the stepper motor.
- Delays before starting a new measurement cycle.

4.5 3D scan (1 - 7 April 2024)

4.5.1 Arduino Code

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

#define STEP_PIN_1 8
#define DIR_PIN_1 9
#define ENA_PIN_1 10

#define STEP_PIN_2 4
#define DIR_PIN_2 3
#define ENA_PIN_2 2

#define XY_REV 200
#define XZ_REV 50

#define ANGLE_INCREMENT 1.8

Adafruit_VL53L0X lox = Adafruit_VL53L0X();

float measurements[XZ_REV][3];

void setup() {
  Serial.begin(9600);  // Initialize the serial monitor

  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1);
  }
  pinMode(4,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(2,OUTPUT);

  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);

  digitalWrite(ENA_PIN_1, LOW);
  digitalWrite(ENA_PIN_2, LOW);

  Serial.println(F("VL53L0X test with Stepper Motor"));
}

void loop() {
  // put your main code here, to run repeatedly:
  for (int i=0; i < XY_REV; i++) {
    if (i%2 == 0) {
      digitalWrite(DIR_PIN_2,HIGH);
    } else {
      digitalWrite(DIR_PIN_2,LOW);
    }
    for (int j=0; j < XZ_REV; j++) {
      VL53L0X_RangingMeasurementData_t measure;
      lox.rangingTest(&measure, false);

      if (i%2 == 0) {
        if (measure.RangeStatus != 4) {  // phase failures have incorrect data
        measurements[j][0] = j * ANGLE_INCREMENT - 45;
        measurements[j][1] = i * ANGLE_INCREMENT;
        measurements[j][2] = measure.RangeMilliMeter;
      } else {
        measurements[j][0] = j * ANGLE_INCREMENT - 45;
        measurements[j][1] = i * ANGLE_INCREMENT;
        measurements[j][2] = 10000;
      }
      } else {
```

```
        if (measure.RangeStatus != 4) {  // phase failures have incorrect data
          measurements[XZ_REV - j -1][0] = j * ANGLE_INCREMENT - 45;
          measurements[XZ_REV - j - 1][1] = i * ANGLE_INCREMENT;
          measurements[XZ_REV - j - 1][2] = measure.RangeMilliMeter;
        } else {
          measurements[XZ_REV - j - 1][0] = j * ANGLE_INCREMENT - 45;
          measurements[XZ_REV - j - 1][1] = i * ANGLE_INCREMENT;
          measurements[XZ_REV - j - 1][2] = 10000;
        }
        }
        digitalWrite(STEP_PIN_2, HIGH);
        digitalWrite(STEP_PIN_2, LOW);
      }
      Serial.print("aaa[");
    for (int k = 0; k < XZ_REV; k++) {
      Serial.print("[");
      Serial.print(measurements[k][0]);
      Serial.print(",");
      Serial.print(measurements[k][1]);
      Serial.print("],");
      Serial.print(measurements[k][2]);
      Serial.print("],");
      }
    Serial.println("]");
    digitalWrite(STEP_PIN_1, HIGH);
    digitalWrite(STEP_PIN_1, LOW);
    }

}
```

Listing 5: Arduino Code for VL53L0X Sensor with Stepper Motors

4.5.2 Code Explanation

Import Libraries

- #include <Wire.h>
  - Provides I2C communication.
- #include <Adafruit_VL53L0X.h>
  - Provides functions for the VL53L0X sensor.

Define Constants and Variables

- #define STEP_PIN_1 8
  - Defines the GPIO pin for the step signal of the first stepper motor.
- #define DIR_PIN_1 9
  - Defines the GPIO pin for the direction signal of the first stepper motor.
- #define ENA_PIN_1 10
  - Defines the GPIO pin for the enable signal of the first stepper motor.
- #define STEP_PIN_2 4
  - Defines the GPIO pin for the step signal of the second stepper motor.
- #define DIR_PIN_2 3
  - Defines the GPIO pin for the direction signal of the second stepper motor.

- `#define ENA_PIN_2 2`
  - Defines the GPIO pin for the enable signal of the second stepper motor.
- `#define XY_REV 200`
  - Defines the number of steps per revolution for the XY plane.
- `#define XZ_REV 50`
  - Defines the number of steps per revolution for the XZ plane.
- `#define ANGLE_INCREMENT 1.8`
  - Defines the angle increment for each step.
- `Adafruit_VL53L0X lox = Adafruit_VL53L0X();`
  - Initializes the VL53L0X sensor.
- `float measurements[XZ_REV][3];`
  - Initializes a 2D array to store the measurement data.

Setup Function

```
void setup() {
  Serial.begin(9600);  // Initialize the serial monitor

  if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while (1);
  }
  pinMode(4,OUTPUT);
  pinMode(3,OUTPUT);
  pinMode(2,OUTPUT);

  pinMode(7,OUTPUT);
  pinMode(8,OUTPUT);
  pinMode(9,OUTPUT);

  digitalWrite(ENA_PIN_1, LOW);
  digitalWrite(ENA_PIN_2, LOW);

  Serial.println(F("VL53L0X test with Stepper Motor"));
}
```

- Initializes serial communication.
- Initializes the VL53L0X sensor.
- Sets pin modes for the stepper motor driver.
- Enables the stepper motor drivers.
- Prints a test message to the serial monitor.

Main Loop

```
void loop() {
  // put your main code here, to run repeatedly:
  for (int i=0; i < XY_REV; i++) {
    if (i%2 == 0) {
      digitalWrite(DIR_PIN_2,HIGH);
    } else {
      digitalWrite(DIR_PIN_2,LOW);
    }
    for (int j=0; j < XZ_REV; j++) {
      VL53L0X_RangingMeasurementData_t measure;
      lox.rangingTest(&measure, false);

      if (i%2 == 0) {
        if (measure.RangeStatus != 4) {  // phase failures have incorrect data
        measurements[j][0] = j * ANGLE_INCREMENT - 45;
        measurements[j][1] = i * ANGLE_INCREMENT;
        measurements[j][2] = measure.RangeMilliMeter;
      } else {
        measurements[j][0] = j * ANGLE_INCREMENT - 45;
        measurements[j][1] = i * ANGLE_INCREMENT;
        measurements[j][2] = 10000;
      }
      } else {
        if (measure.RangeStatus != 4) {  // phase failures have incorrect data
        measurements[XZ_REV - j -1][0] = j * ANGLE_INCREMENT - 45;
        measurements[XZ_REV - j - 1][1] = i * ANGLE_INCREMENT;
        measurements[XZ_REV - j - 1][2] = measure.RangeMilliMeter;
      } else {
        measurements[XZ_REV - j - 1][0] = j * ANGLE_INCREMENT - 45;
        measurements[XZ_REV - j - 1][1] = i * ANGLE_INCREMENT;
        measurements[XZ_REV - j - 1][2] = 10000;
      }
      }
      digitalWrite(STEP_PIN_2, HIGH);
      digitalWrite(STEP_PIN_2, LOW);
    }
    Serial.print("aaa[");
  for (int k = 0; k < XZ_REV; k++) {
    Serial.print("[");
    Serial.print(measurements[k][0]);
    Serial.print(",");
    Serial.print(measurements[k][1]);
    Serial.print("],");
    Serial.print(measurements[k][2]);
    Serial.print("],");
    }
  Serial.println("]");
  digitalWrite(STEP_PIN_1, HIGH);
  digitalWrite(STEP_PIN_1, LOW);
  }

}
```

- Runs the main loop repeatedly.
- Controls the direction of the second stepper motor.
- Takes distance measurements with the VL53L0X sensor at each step.
- Stores measurements in a 2D array.
- Prints the measurements to the serial monitor in a JSON-like format.
- Steps the first stepper motor.

4.6 Serial Communication - Receiver (22 - 28 April 2024)

4.6.1 Python Code

```python
import serial
import json
from math import cos, sin, pi
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

from datetime import datetime

# datetime object containing current date and time
now = datetime.now()

print("now =", now)

# dd/mm/YY H:M:S
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")

printed = False

def plot_lines(coordinates):
    x_values, y_values = zip(*coordinates)
    plt.plot(x_values, y_values, color='blue', linestyle='-', linewidth=2,
        label='Line')

    plt.title('Graph with Connected Line (No Dots)')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')

    # Set axis limits to always be 0 to 300
    #plt.xlim(0, 300)
    #plt.ylim(0, 300)

    # Set the aspect ratio to be equal
    plt.gca().set_aspect('equal', adjustable='box')

    plt.grid(True)
    plt.legend()
    plt.show()

def plot_dots(coordinates):
    x_values, y_values = zip(*coordinates)
    plt.scatter(x_values, y_values, color='blue', marker='o')
    plt.title('Graph with Dots')
    plt.xlabel('X-axis')
    plt.ylabel('Y-axis')

    #plt.xlim(0, 300)
    #plt.ylim(0, 300)

    plt.gca().set_aspect('equal', adjustable='box')

    plt.grid(True)
    plt.show()


def plot_3d_surface(coordinates):
    x_coords = [coordinate[0] for coordinate in coordinates]
    y_coords = [coordinate[1] for coordinate in coordinates]
    z_coords = [coordinate[2] for coordinate in coordinates]

    # Convert coordinates to numpy arrays
    x = np.array(x_coords)
    y = np.array(y_coords)
    z = np.array(z_coords)
```

```

    # Create a 3D plot
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Plot the scatter plot
    ax.scatter(x, y, z, c='b', marker='o', s = 1)

    # Connect the points with lines
    for i in range(1, len(x)):
        ax.plot([x[i-1], x[i]], [y[i-1], y[i]], [z[i-1], z[i]], c='b')

    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')

    plt.show()

def plot_3d_wireframe(coordinates):
    x_coords = [coordinate[0] for coordinate in coordinates]
    y_coords = [coordinate[1] for coordinate in coordinates]
    z_coords = [coordinate[2] for coordinate in coordinates]

    # Convert coordinates to numpy arrays
    x = np.array(x_coords)
    y = np.array(y_coords)
    z = np.array(z_coords)

    # Create a 3D plot
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Plot the wireframe
    ax.plot_trisurf(x, y, z, linewidth=0, antialiased=False)

    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')

    plt.show()


def str2list(input_string):
    input_string = input_string.replace('\r', "")
    input_string = input_string.replace('\n', "")

    temp_buffer = []
    temp_text = ''
    selected_dots = []

    for stringData in input_string.split('\n'):
        temp_buffer.append(temp_text + stringData.split('\n')[0])
        temp_text = ''

        if temp_buffer[-1][:3] == "aaa":
            temp_text = ''
            build_temp = temp_buffer[-1][3:].rstrip('\n\r,')

            # Handle the case when the string ends with a trailing comma
            if build_temp[-2]:
                build_temp = build_temp[:-2]+build_temp[-1]

            try:
                nested_list = [list(map(float, innerList)) for innerList in
                    json.loads(build_temp)]
            except json.decoder.JSONDecodeError as e:
                print("Error decoding JSON:", e)
                print("Problematic data:", build_temp)
```

```
                return False

            for nested_item in nested_list:
                x_temp = ((nested_item[2] * cos(nested_item[0] * (pi / 180)) * cos(nested_item[1] * (pi / 180))) + 2000) /10
                y_temp = ((nested_item[2] * cos(nested_item[0] * (pi / 180)) * sin(nested_item[1] * (pi / 180))) + 2000) /10
                z_temp = ((nested_item[2] * sin(nested_item[0] * (pi / 180))) + 2000) /10

                if x_temp < 1000 and y_temp < 1000:
                    selected_dots.append([x_temp, y_temp,z_temp])
                    #print("Selected dots:", selected_dots)
            return selected_dots

    return False


class ReadLine:
    def init(self, s):
        self.buf = bytearray()
        self.s = s

    def readline(self):
        i = self.buf.find(b"\n")
        if i >= 0:
            r = self.buf[:i+1]
            self.buf = self.buf[i+1:]
            return r.decode("utf-8")

        while True:
            i = max(1, min(2048, self.s.in_waiting))
            data = self.s.read(i)
            i = data.find(b"\n")
            if i >= 0:
                r = self.buf + data[:i+1]
                self.buf[0:] = data[i+1:]
                return r.decode("utf-8")
            else:
                self.buf.extend(data)

ser = serial.Serial('COM3', 9600)
rl = ReadLine(ser)

can_update = False
list_for_3d = []

while not printed:
    line = rl.readline()
    #print("Received:", line)
    print_temp = str2list(line)
    if print_temp != False:
        #print(print_temp)
        if (int(print_temp[0][1]) == 0 and int(print_temp[0][0]) == 0):
            can_update = True
        if True:
            for i in print_temp:
                list_for_3d.append(i)
            print(len(list_for_3d)/50)
            if len(list_for_3d) >= 200*50:
                #plot_dots(print_temp)
                #plot_lines(print_temp)
                #print(len(list_for_3d))
                # Specify the file path
                file_path = "example" + dt_string + ".txt"

                # Open the file in write mode
                with open(file_path, 'w') as file:
```

```
                    # Write the text to the file
                    file.write(str(list_for_3d))

                print(f"Text saved to {file_path}")
                #print(list_for_3d)
                plot_3d_surface(list_for_3d)
                printed = True
```

Listing 6: Python Code for receiver

4.6.2 Code Explanation

Import Libraries

- `import serial`
  - Provides serial communication capabilities.
- `import json`
  - Used for parsing JSON data.
- `from math import cos, sin, pi`
  - Provides trigonometric functions and the value of π.
- `import matplotlib.pyplot as plt`
  - Used for plotting graphs.
- `import numpy as np`
  - Provides support for large, multi-dimensional arrays and matrices.
- `from mpl toolkits.mplot3d import Axes3D`
  - Enables 3D plotting.
- `from datetime import datetime`
  - Provides date and time functionalities.

Datetime Initialization

```
now = datetime.now()
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
```

Initializes the current date and time and formats it as a string.

Plotting Functions

Defines functions for plotting different types of graphs:

- `plot_lines(coordinates)`: Plots lines connecting the coordinates.
- `plot_dots(coordinates)`: Plots dots at the coordinates.
- `plot_3d_surface(coordinates)`: Plots a 3D surface.
- `plot_3d_wireframe(coordinates)`: Plots a 3D wireframe.

String to List Conversion

```
def str2list(input_string):
    ...
```

Converts the input string from the serial data into a list of coordinates.

ReadLine Class

```
class ReadLine:
    def init(self, s):
        self.buf = bytearray()
        self.s = s

    def readline(self):
        ...
```

Class for reading lines from the serial port.

Main Code Block

```
ser = serial.Serial('COM3', 9600)
rl = ReadLine(ser)
can_update = False
list_for_3d = []

while not printed:
    line = rl.readline()
    print_temp = str2list(line)
    if print_temp != False:
        if (int(print_temp[0][1]) == 0 and int(print_temp[0][0]) == 0):
            can_update = True
        if True:
            for i in print_temp:
                list_for_3d.append(i)
            if len(list_for_3d) >= 200*50:
                file_path = "example" + dt_string + ".txt"
                with open(file_path, 'w') as file:
                    file.write(str(list_for_3d))
                plot_3d_surface(list_for_3d)
                printed = True
```

Establishes a serial connection and continuously reads data. When sufficient data is collected, it saves the data to a file and plots a 3D surface.