College of Engineering

ELEC 491 – Electrical Engineering Design Project

Final Report

A STUDY OF DIFFERENT SENSORS IN COMMERCIAL INDOOR MAPPING APPLICATIONS

Participant information:

Kaan Toyaksi 64815
Deniz Karadayı 59925
Mehmet Çelik 64851

Project Advisor(s)
Hakan Ürey
Ata Mozaffari

14/06/2021

Abstract

The goal of this project is to do a detailed study of different distance measurement sensors that are used or can be used in indoor mapping applications for commercial use like the autonomous cleaning robots. In this pursuit, we tested different distance measurement sensors that are available in the Turkish market to see whether they can satisfy the required tasks and properties. Lidar, Laser, Ultrasonic and Infrared sensors were chosen for this project. Aside from these we also compared other sensors on a theoretical level such as Radar sensor. These sensors will be compared under different categories, such that cost, feasibility, accuracy, and power efficiency. After these comparisons we will be conducting tests with our 2D indoor mapping robot using these different sensors. We tested each sensor's performance along with its adaptability with the other sensors. So aside from this sensor comparison we built an autonomous indoor mapping robot with 4 different sensors. In this paper you can find these test results, the observations, the measurements, the ideas behind the codes and the robot. We hope that these can provide some inside for sensor selection or building an indoor mapping robot in general.

Table of Contents

1. INTRODUCTION

1.1 Concept

Indoor mapping is a highly used beneficial technology of today's world. It has been partitioning in critical aspects of the life even people are not really aware of. Location navigation, route calculations, supporting decision making process of the automation, and allowing personalization are examples of the indoor mapping technologies take roles. From robot vacuum cleaners to statistical data collection for market strategies, there are lots of areas using the technology. In terms of constructing the mapping of a dynamic place and using the obtained data in application with a use, it is an important popular technology of ongoing life.

Since it is not a new brand area needed for an urgent improvement, the intention is not to bring a brand-new mapping technique. However, a comprehensive study of the sensors used in the mapping technologies is intended to be presented. The aim of this study is to construct a guide for the designer with all kinds of budgets and purposes. The components used in the application is critical in terms of specifying the left money that can be spent on other aspects of the project. In addition, the performance and the specifications of the components must be serving the purpose of the application to make it perfect. In this sense, there is no best component but there is best-fitted component. Based on this idea, the objective is to construct several setups for an automated indoor mapper device consisting of different distance measurement and range finding sensors. In this way, it is aimed to compare the systems in terms of cost, compatibility, accuracy, and power efficiency.

The main setup of the project is a device autonomously moving inside an indoor area and creating a 2D plan of the area. Additionally, it is connected to a computer with Bluetooth module so that it can be controlled over mobile application. The technologies that have been used in the project are robotics, automation, SLAM, data visualization, and wireless communication.

1.2 Objective

LiDAR is a very successful and popular choice for mapping applications, but it drives the final product's price up dramatically. Even if it is very preferrable for high-budget professional applications, it is not supposed to be the only promising alternative for distance ranging and area scanning. Different distance measuring methods and sensors can be considered based on application specific requirements and working conditions.

This documentation could be highly beneficial for the researchers, developers, and designers

in this manner. It takes a lot of time and effort to study before beginning to a project since it is not always possible to assume the behavior of the equipment and the resulted effect on the project. In addition, wrong choices might cause huge wastes of money besides the lost time. On the other hand, purchase of expensive equipment to guarantee the success of the performance also may end up as running out of money to finish the project. Therefore, there is a critical trade off in the beginning of each project that slows down the process of the development. The main goal is to help developers to shorten their decision process with the help of the detailed study related to sensors used in indoor mapping technologies. Providing detailed comparison of these components will constitute a handbook to construct the projects path of future.

As previously mentioned, it should be clarified that the objective of the project is not to present genius indoor mapping technologies but providing data of the performances of different sensors that are available to be used instead of high-quality and expensive LiDAR technologies. It is aimed to provide guidance to low-budget mapping projects with analysis of the results of the project. A focused study of this type of sensors and their cheaper alternatives can provide a unique perspective on the market and allow designers to have alternative approaches.

1.3 Background

In this project, the main framework is shaped by three major way of range finding, which are by light, sound, and radio signal. These corresponds to LiDAR, sonar, and radar technologies in popular and advanced applications. There lots of uses of these sensors in communication, weather precision, transportation, surveying, safety and so on. The mentioned topics are serious and vital for the sustainability of critical transactions and the progress of the usual way of life. Besides, there series of attempts to progress more and more technologies based on these sensors. It is wanted to provide a first-hand experienced and reported data to enlighten future applications.

LiDAR is a highly precise and advanced sensor that is used in lots of developed technologies from planes to map constructions. By using light signals, it can measure the distances and can construct 3d imaging by combining all the data acquired. It provides accuracy, precision, and flexible platform to develop many kinds of application. Unfortunately, it is a high technology at the same time. Therefore, it is not always wise to prefer its usage.

Ultrasonic sensors (sonars) are also an option to be used in range sensing, which are based on ultrasonic sound waves to measures and a cheaper alternative compared to LiDAR. In

addition, it is better alternative in unpreferable conditions of environment such as weather conditions, and under the water.

Radar, on the other hand, uses electromagnetic waves and radio waves to range the environment. It has pretty long measurement capability, and bases on angular positioning of the environment. In addition, it can provide additional data to distinguish the targeted object located at the distance measured.

The common benefit that all these sensors provide is the distance data used in mapping technologies. Mapping is based on the odometry of the sensors and the distance data obtained by it as well. When the information that where the sensor positioning in the space and what the orientation of it is available, the distance data provided by it allow system to build an image of surroundings. This technology called as Simultaneous Localization and Mapping. It is important to build up the map of an unknown environment by using real time data. There are some algorithms used to obtain these maps such as particle fever, Kalman filter, covariance intersection and Graph SLAM . Other than mapping technologies, these are also used in AR/VR applications and autonomous system technologies.

To sum up, the problem based and sensing, localizing, estimating, and finally modeling. In this sense, it is a combination of different data, so the accuracy and precision of these data and closeness of the estimation is critical to come up with an accurate outcome in the end. To do these, a preliminary research is done to use best-fitting sensors, algorithms, equipment and, environment so that the results can represent the most fair and real-like behaviors and performance of the sensors used.

In addition, it is another goal to provide an autonomous movement that enable device to obtain enough information to collect from environment without wasting the power and time so that it can be clear that the system is working with most possible efficiency. Within the process of building the testing environment, choosing the equipment, and deciding the design of the setup, the previous works ad methods tried building indoor mapping solutions are investigated. These different works contain different sensors, mapping algorithms, constructions, and environment so that it can be inferred from their reported results. Some of the works and studies giving inspiration can be found in the reference part of this paper.

1.4 Methodology

The focused sensors are cheap LiDAR, ultrasonic, laser and infrared radar. In order to pick them, an early research was conducted to decide them. Next, specific model of those sensor

types are chosen which are best to be compared. The type of distance measurement and the accuracy of the input data collected by them are the base to construct a simultaneous localization and mapping device.

In this paper, a detailed analysis is presented by explaining the methods that are used, experienced complications and the performance reports of the setups including the drawbacks, advantages, feasibility, accuracy, and costs. For you to understand the conditions and parameters under which the analyses are proceeded, the details of the setup such as the materials, physical equipment, and hardware is shared. Besides, the method used in the implementation of the indoor mapping algorithm and automation of the robot routing, the software used in the testing and simulation procedures, and the environment where the setup is tested are also presented to clarify the circumstances may take part in the performance of the sensors.

In the analysis part, the path is to test the sensors independent of the setup in first-hand to know the individual performance and behavior of the sensors. This gives the initial idea about the difference between these sensors. After that, it is aimed to implement an indoor mapping algorithm that can incorporate with the sensor of interest at each setup. Therefore, a working mechanism with the use of these sensors is developed. The final project is a device able to construct a real-time map for the indoor test environment. The final setup is the one which is decided as the best performing indoor mapping device with the available resources and sensors. Variety of setups are constructed to test the performance, which are presented in the analysis section, so that the comparison of the distance sensors for such applications can be interpreted. Therefore, in the analysis part, the test environment, setup of the device and the resulting data and map are provided. The unsuccessful setups are not failures but a potential indication for which sensor is not preferrable for which case scenarios. Therefore, it is still promising feedback for the analysis.

2. SYSTEM DESIGN

In order to decide the system design, a detailed market analysis of distance sensors that are used for ranging and available related low-cost projects was conducted. The research provides the basis for comparison types and methods to be investigated later. Once the sensor types to be compared are decided, the model and branch of sensors for each type have been selected. This information is provided in the beginning of this section to give an idea of the foundations on which this project is based.

2.1 Pre-Research

A. Sensor Type Decision

When determining the sensor types that are best suited for this project. A range of parameters are used such as pricing, accuracy, range, sensitivity, compatibility and more. Another thing to consider when selecting a distance measurement sensor is availability, unfortunately Turkey has a small market size for sensors. The sensors that were selected for this project covered all the bases required.

Laser

Laser sensor is the first sensor type that was considered for this project. Laser sensors measure distance with the same principle as the other distance measurement sensors. A transmitter will emit a wave from the sensor surface and a receiver will detect this emitted wave. The time that was passed during this process will be measured and the required distance will be calculated from these measurements. In laser sensor's case this wave has 940nm wavelength and the receiver is only activated with this wavelength.

Laser sensor is selected for this project because of the reasons below:

- Accuracy: Laser sensors are highly accurate because they are not majorly affected by the environmental effects such as light intensity, color, etc. So alongside with great performance, this ability to stay accurate in different environments is crucial.
- Sensitivity: Laser sensors sensitivity is considerably high and can respond quickly to change in short amounts of time. It is noted that laser sensors can provide highly sensitive data in indoor applications, whereas in outdoor applications or in case of any intrusion from sunlight, the sensors sensitivity decreases noticeably.
- Range: Laser sensors are designed for non-contact measurements and have high rates of accuracy over considerable distances. Different laser sensor ranges may vary according to their application or need.
- Price: Laser sensor prices may change due to sensors application area, range, accuracy and sensitivity. Sensors that are highly accurate at long ranges with high sensitivities are not feasible for commercial use considering their price. For indoor mapping applications however, laser sensors with feasible prices are available that provide decent enough range.
- Compatibility: Laser sensors are usually designed for being used with a microcontroller like Arduino or Raspberry pi. Their baud rate is in range with most of the sensors that might be utilized simultaneously. Which makes them highly compatible for this project.

Ultrasonic

Ultrasonic sensors work by transmitting a sound wave to an object, and with a receiver the sensor picks up the reflected soundwaves. The time relation between the transmitted and reflected signals, enables for the sensor to convert that information into a distance measurement. For these reasons and reasons that will be further explained in the section below, it is decided that utilization of an ultrasonic sensor would be beneficial on this study.

- Accuracy: Accuracy of the ultrasonic sensors are considered to be quite good and within range of the required applications this project needs to achieve. Accuracy wise ultrasonic sensors are chosen to be applicable.
- Sensitivity: Ultrasonic sensors sensitivity differ from the other sensors that utilizes electromagnetic waves to detect an object. Ultrasonic sensors accuracy is usually affected when there is a soft material present. Soft materials absorb sound waves and make it harder for the sensor to detect its range accurately.
- Range: Range for the ultrasonic sensors is considerably low when compared to other sensors such as laser. However, for indoor applications in tight spaces or by utilizing the sensor considering its range limitations, ultrasonic sensors are found to be applicable for this project regarding their range.
- Price: Ultrasonic sensor prices are considerably low when compared to other sensors on this project. Their use of sound waves and low range measuring abilities can be the reason behind their low-price levels.
- Compatibility: Ultrasonic sensors are usually designed for being used with a microcontroller like Arduino or Raspberry pi. Their baud rate is in range with most of the sensors that might be utilized simultaneously. Which makes them highly compatible for this project.

Infra-red

Infra-red sensors can both transmit and receive infra-red radiation. Their working principle is similar of laser sensor in the way that Infra-red sensors when comes in close proximity to an object, infra-red radiation reflected from the surface of the object is picked up by the receiver. Then the sensor can calculate the distance of the object that the infra-red radiation is emitted from. But unlike the laser sensor, infra-red sensors measure the intensity of the radiation is emitted from the object rather than the time of flight. For the reasons that will be further explained in the section below, it is decided that utilization of an infra-red sensor would be beneficial on this study.

- Accuracy: Accuracy of the IR sensors are considered to be decent and within range of the required applications this project needs to achieve. Accuracy wise IR sensors are chosen to be applicable.
- Sensitivity: IR sensors are very sensitive to light changes and the change in the color of the object detected. They are also very sensitive to sunlight, and this makes them unusable for outdoor applications. However, for this project they will be included for comparison purposes in indoor applications.
- Range: Considering range IR sensors could be considered mid-tier. While they do not have low range, they also do not measure for long distances. For this project, their range is in the required area.
- Price: IR sensor can be considered on the cheaper side of the scale when compared to the other sensors used on this project. For their cheap price and decent performance IR sensors are chosen to be used on this project.
- Compatibility: IR sensors are usually designed for being used with a microcontroller like Arduino or Raspberry pi. Their baud rate is in range with most of the sensors that might be utilized simultaneously. Which makes them highly compatible for this project.

LiDAR

LiDAR sensors utilize continual pulses of laser that's emitted from their transmitter to measure the distances. They use time of flight to determine the distance of the object that reflects the laser sent from its transmitter. They use a similar working principle as the laser sensors however, they have a wider variety of applications due to the abilities their more expensive variants have. LiDAR sensors are considered for this project for their frequent use in indoor mapping applications.

- Accuracy: LiDAR sensors are highly accurate sensors; however, their resolution is not best considered some of the other sensors used in this project. It is known that LiDAR sensors are commonly used in indoor mapping applications.
- Sensitivity: LiDAR sensors are great for both indoor and outdoor use and they do not get affected much by the sudden changes in light levels or changes in the color of the objected reflecting the laser light. Because of these abilities they are widely used in both outdoor and indoor conditions by designers.
- Range: LiDAR sensors are very good at measuring and detecting in long distances. However, they lack in measuring closer distances and data gathered from them at close distances are generally inaccurate.

- Price: Considering their price, LiDAR sensors do not come cheap. With what they provide in quality they claim back it in price. LiDAR sensors was the most expensive sensor that was utilized in this project and unless for specific applications they are not a commercially viable option.
- Compatibility: As for LiDAR's compatibility goes with Bluetooth, since LiDAR works in higher baud rates than Bluetooth there can be a loss of data in the transmission. Like other sensors it also uses a microcontroller board to operate. Having high levels of power consumptions sometimes might affect other sensors as well.

B. Sensor Model Decision

Laser

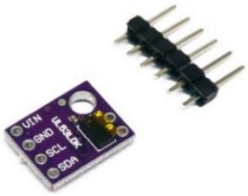

Figure 1: VL53L0X sensor module [1]

As for the laser sensor that utilized in this project, after an adequate amount of research and comparison between different laser sensors, VL53L0X has been selected. It has a range of 120 cm for indoors in default mode, which was the mode that utilized for this project due its higher accuracy. It has an error rate of 3-6% depending on the color of the object in the previously mentioned mode. It also operates with 5V and utilizes an I2C interface which are the optimal options that is used in this project. Also, with its high detection rate of minimum 94% and its small size, it is decided that VL53L0X ToF laser sensor would be the laser sensor of choice for the project. At the time of this project VL53L0X ToF laser sensor was sold for 40 Turkish Liras.

Ultrasonic

Figure 2: Grove-Ultrasonic Ranger V2.0 [2]

As for the ultrasonic sensor that utilized in this project, after an adequate amount of research and comparison between different ultrasonic sensors, Grove-Ultrasonic ranger V2.0 has been

selected. It has a range between 2 – 350 centimeters depending on its datasheet. It has a resolution of 1 centimeter and its measurement angle is 15 degrees. It also operates with 5V and utilizes an I2C interface which are the optimal options that is used in this project. Also, it uses 40kHz frequency range while measuring and it only weighs 13 grams, it is decided that Grove-Ultrasonic ranger V2.0 sensor would be the ultrasonic sensor of choice for the project. At the time of this project Grove-Ultrasonic ranger V2.0 sensor was sold for 50 Turkish Liras.

Infra-Red

Figure 3: Sharp GP2Y0A02YK0F Infrared Distance Sensor [3]

As for the infra-red sensor that utilized in this project, after an adequate amount of research and comparison between different infra-red sensors, Sharp GP2Y0A02YK0F Infrared Distance Sensor has been selected. It has a range between 20 – 150 centimeters according to its datasheet. Its resolution and measurement angle were not specified in the datasheet. It also operates with 5V and utilizes an I2C interface which are the optimal options that is used in this project. Also, according to its data sheet it uses a triangulation method while measuring, this diminishes the effects of sun and other infra-red pollution to the sensor, and it only weighs 4.8 grams. It is decided that Sharp GP2Y0A02YK0F Infrared Distance Sensor would be the infra-red sensor of choice for the project. At the time of this project Sharp GP2Y0A02YK0F Infrared Distance Sensor was sold for 65 Turkish Liras.

LiDAR

Figure 4: TF Mini LiDAR sensor module [4]

In this project the main comparison sensor is the lidar sensor. However, lidar sensors does not come cheap, especially when compared to other sensors that were used in this project. Since the capabilities of high-end lidar sensors are known and would dominate all the other

sensors in this comparison, it is decided that for our project a low-end lidar sensor would be used. Even though it is stated as a low-end lidar sensor, our sensor of choice, TF Mini LiDAR (ToF) Laser Range Sensor, at the time of this project, was sold for 400 Turkish Liras. In comparison that is almost three times more than all the other sensors that are used in this project combined. As for the technical specifications TF Mini LiDAR has a range between 0.3 to 12 meters. It has a resolution of 5mm and for the first 6 meters it has a varying accuracy of 1%. Sensor operates with 5V and utilizes an UART(TTL) communication interface. It does not get affected from the Sun and other light sources easily. It weighs 4.7 grams. All in all, it can be seen that even though a lower end lidar sensor like TF Mini is still has better overall standards than the other sensors.

2.2 Equipment

A. Individual Performance of Distance Sensors

Additional performance tests were done with the sensors before they were attached to the setup, in order to observe their effective range, efficiencies and reliability. The results of these test can be found in the tables below and they will be further discussed for each sensor type.

IR: Sharp GP2Y0A02YK0F Distance Measuring Sensor

Distance Interval (cm)	0-20	20-22	22-25	25-30	31-35	35-55	55-65	65-70	71-85	86-100	101-120	120+
Error (cm)	Out of range	0	-1	-2	-3	-4	4	4	5	10	20	Out of range

For the infra-red sensor that was purchased for the project, initially it can be seen that for the first 20 cm the sensor is out of range. While between 0 – 20 cm, the sensor constantly gives an output of 20 cm regardless of where the distance is at, between 0 and 20 cm. For the ranges between 20 - 55 cm, sensor has a decently accurate output, however from the data it can be observed that as the distance comes closer to 55 cm, error gradually increases on the negative side. Above 55cm, errors sign flips and the sensor start to show the distance is actually larger than it normally is. That is an 8 cm of change and should be kept in mind while checking the results. Between 86 – 120 cm, error from the sensor drastically increases and before going out of range at 120cm the system has a constant error of 20 cm at most. When compared with other sensors it can be seen that infra-red sensor is not a very good option for close distances or for distances over a meter. However, it gives a consistent data output for the ranges in between and that was utilized for the project.

Ultrasonic: Grove Ultrasonic Ranger

Distance Interval (cm)	0-3	3-40	41-60	61-70	70-80	80-90	90-100	100-120	120+
Error (cm)	Out of range	0	-1	-2	-1	-2	-3	-5	Out of range

For the ultrasonic sensor that was purchased for the project, initially it can be seen that for the first 3 cm the sensor is out of range. For the ranges between 3 - 40 cm, sensor has a perfectly accurate output. Above 41cm, the sensor starts give an output with a negative error of 1cm and as the distance increases given error is also gradually increases. At around 120cm it can be seen that the maximum error is 5 cm and for the values above 120cm, the sensor is considered to be out of range due to highly inconsistent outputs. When compared with the other sensors, at first glance ultrasound sensor seems like a very good option due to its ability to measure closer distances and consistent error rate from the tests. However, ultrasound sensor is highly susceptible to change in test conditions and change in the material of the object. Due to this property, it can occasionally give out noisy or fluctuating distance measurements.

LIDAR: TF Mini LiDAR

Distance Interval (cm)	0-30	30-35	35-60	60-80	80-99	100	101-125	125-130	130-140	140-145	150+
Error (cm)	Out of range	-4	-1	-2	-1	0	1	-1	1	3	6

For the lidar sensor that was purchased for the project, initially it can be seen that for the first 30 cm the sensor is out of range. For the ranges between 30 - 35 cm, sensor start with giving out one of its higher error outcomes, this is due to lidar being unreliable in close ranges. Above 35cm however, given error is decreased and this consistent low error output is preserved up to 150 cm. Above 150 cm, error starts to increase in line with the expected error rate obtained from datasheet. When compared with the other sensors, lidar is pretty accurate while it is in range. However, because it loses first 30 cm of the data being out of range, lidar might not be a good option for close range quarters. Also, there is also an occasional data reading compatibility issue with the lidar, meaning occasionally, the sensor gives an error output instead of an actual distance reading.

Laser: VL53L0X ToF Ranging Sensor

Distance Interval (cm)	0-4	4-7	7-10	10-25	25-80	80-90	90-120	120+
Error (cm)	1	0	-1	0	1	3	4	Out of range

For the laser sensor that was purchased for the project, initially it can be seen that the sensor is never out of range for close distances. For the ranges up to 80 cm, sensor gives out considerably reliable and close to perfect data. Above 90 cm however, data starts to become unreliable and at around 120 cm the sensor becomes out of range. When compared with the other sensors, laser is the best sensor for close ranges and a decent reliable sensor at mid ranges. It does not get affected by its surroundings and its noise output is low. Laser sensor becomes unreliable in the long ranges, and this should be kept in mind while reading the result section of this project,

B. Other Electronic Components

Controller

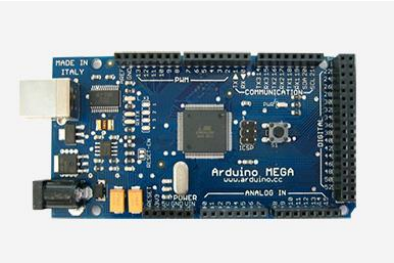

Figure 5: Arduino Mega 2560 microcontroller board that is utilized in this project. [5]

Arduino and Raspberry Pi are the leading companies in single-board microcontrollers (SBM). They are mainly built for DIY project and school projects but at this time they are used in many more applications ranging from R&D prototypes or final products for some companies. They also have a huge catalog of models which a consumer can selected the ideal fit for their project.

Arduino is an open source SBM project that used a coding application to control the board and show the project data; on the other hand, Raspberry Pi is has its own OS and requires a memory to operate. One can say that Arduino is a simpler option compared to Raspberry Pi's complexity. Arduino is a barebones model and requires external modules for operations that Raspberry Pi can easily handle on its own.

In this project the main comparison that was carried out is between the Arduino Mega and Raspberry Pi. The comparison between these two can be found below:

- Price: Arduino is a cheaper alternative to Raspberry Pi, because it is simpler and has only the main components. Also, the OS and the other components that Raspberry Pi needs makes it even more expensive. For a project that focuses on the "best bang for your buck", Arduino is the clear winner.

- Performance: When compared to Arduino, Raspberry Pi has the clear advantage in performance. The newer Pi models have 1 or even 2 GB ram along with its powerful CPU. This makes it the clear winner in this category. This project is not a computational power demanding one so high-end performance is not something that is required. Arduino Mega has enough power to operate the whole project.
- Module Integration - Model Operation Ability: Both SBM models work with the same principle which used I2C or other communication protocols. So, this is not an important part of the comparison. The main difference between the models is that 4 sensors demand different baud rates and power. Arduino really struggles to provide these as its fixed on a baud rate from the start and its power management is ok at best. Raspberry Pi on the other hand can provide these changes without an issue and is the clear winner for this task.
- Coding and Mapping: Raspberry Pi has a built in Python setup and can use other coding environments like C, C++ and more. Arduino requires external help to map the environment. In this project MATLAB is selected for this task. There is no real difference between Python and MATLAB for this project so one can say that it just comes down to preference. For simplicity, this project uses MATLAB hence Arduino is the winner for this comparison front. There are some real and important tradeoffs happening when selecting a SBM for a project, if a high-performance project is the goal, one should select Raspberry Pi instead of Arduino. For this project price is an important topic and Arduino has the upper hand in this specific case.

Motors & Motor Driver

Figure 6: HG7881 DC Motor Driver Card that is utilized in this project. [6]

There was not a big Motor Drive comparison for this project because any 4 Channel DC Motor Driver would have worked. The working principle for these motor drivers is simple. The motor driver gets digital inputs from the SBM and power from an external power source. After that it provides power to the related motors. The digital inputs that are gathered from the controller are mainly used for the switch controller that are located on the motor drive. There

are some analog Motor Drivers as well.

For the related motors, the main decider was the motor power and power consumption. The other aspects for the motor selection were not important for this project. The 4 motors were also identical as well. The motors are small and power efficient for this projects case because the power bank used is small and cannot provide the necessary power needed to operate those big motors. Also, if the motors were too big the distance covered per Arduino loop would be too big to get any meaningful information.

Communication Module

Figure 7: HC-05 Wireless Bluetooth Transceiver Module that is utilized in this project. [7]

After selecting Arduino as the SBM for this project an external wireless communication method became a must have. There are only 2 main options for this task. First is the WIFI Communication Module and the second one which is the Bluetooth Communication Module. MATLAB has a Wireless Arduino Control library that allows users to work with their projects without and cables. So, on that front there is nothing to compare.

WIFI module for this project caused real problems on two different sides. First problem is that the direct Arduino to MATLAB connection was not working like it is supposed to do. The connection showed its existence, but the data transfer was non-existent. To pass this issue some external data transfer methods were selected namely “Thingspeak” which is a web based IoT data transfer and database provider that can work with both Arduino and MATLAB. The problem with this option was the data rate, where the Thingspeak provided the sensor data very slowly. The one of the promises of this project was the real time mapping and with this setup this was practically impossible. These two problems eliminated WIFI Communication Module becoming the main communication module for this project.

On the other hand, Bluetooth Communication Module provided a good connection and fast and two-way data transfer and became the main module for this project. The other differences between these two modules are more focused on the WIFI vs Bluetooth side. When comparing these two the main trade-off comes down to speed, complexity, range and adaptability.

Bluetooth module is also cheaper than the WIFI module and this was the main reason that Bluetooth Communication Module was selected.

Localization and Odometry Sensor

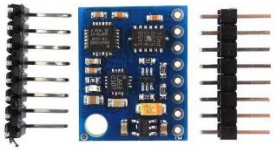

Figure 8: HW-579 9 axis IMU sensor. [8]

IMU sensors are the most popular sensors around the world. They are basically used in every robot-based project imaginable. In this project, the IMU sensors could be used in distance measurements (how much the robot moved), control the turning functions and finding the orientation of the robot. Accelerometer, gyroscope, and magnetometer can carry these tasks in this order. Unfortunately, there were some problems with the Arduino side and these sensors could not function properly.

- Accelerometer: The accelerometer data was fine unlike the other sensors. The noise created some problems but those can be fixed with filters. The main reason that no accelerometer was used is that it was not necessary. The distance covered can be calculated from the data that was obtained from the front sensor.
- Gyroscope: On its own the gyroscope sensor worked well. The data and the angle measurements were on point for the small turns but as the turning angles became larger the gyroscope data gave out wrong numbers. In addition to this gyroscope code depends on the time passed in each loop and with the rest of the code this value increased to a point where any meaningful data from the gyroscope turned into garbage. This is the main reason why no gyroscope was used in this model.
- Magnetometer: This sensor was used for finding the orientation relative to the earth's magnetic field. On its own this was another sensor that was really good, but it is affected by the other electronics around it significantly. So, in a robot setup where everything is electronic magnetometer could not work at all. Hence the reason why the whole IMU trio was disregarded.

Power Sources

It is not easy to supply power for such system since there are numerous electronic parts requiring constant power and actively working during the runs. Besides, the weight of the total

device is effective on battery consumption as the motors will be requiring much power to move the device. In addition, weight should be balanced to be sure that device can move each direction with same behavior. On the other hand, motor driver should be powered by a different source since it is a huge power consumer and significantly affects the system if it would be fed by the same source with other components. Therefore, a chargeable power bank which provides 5V is connected to feed the motor driver, and 9V alkaline batteries which seems to be best performing battery type to feed the controller are used. With other battery types, it is observed that voltage is not constant and fluctuating supplier causes sensor errors in the application.

C. Physical Parts and Set-up

Figure 9: Cruise Omni-wheel Robot Platform is utilized as the chassis.

The rest of the robot setup consists of two sections. The chassis and the omni wheels. The chassis was selected as an octagon because at that time the sensor placements were not final, and this model provided a lot of different combinations. Also, it was lightweight and had enough room for all the project components.

Omni wheels are selected because they offered the best turning and movement possibilities. They were the perfect wheels for this project. There were some problems with sliding and swaying with this setup. The main reason behind this was the inability to do a good soldering by the students. The existing solders were imperfect, and this caused some power differences between the motors and the wheels. Another reason for this problem was caused by the screws. The chassis has some holes for the screws, but they were not equal, and the motors were moving side to side in some places and that caused the swaying as well. These issues were fixed with better soldering and some tight screwing.

2.3 System Diagram and Algorithm

The project mainly consists of two sections, the robot, and the code part. The robot and the code part can communicate between by the Bluetooth Module. This module allows two-way

data transfer without any issue. The data transfer stars with the MATLAB sending the start command and continues with the sensor data transfer and ends with the stop command that comes from the MATLAB code.

The robot is built from 4 sections: sensors, motors, Arduino and the other modules. The sensors are placed on the 4 sides of the chassis. This was a design choice from the students as there were different placement and movement ideas covered. This placement was selected for the final product because it allowed students to try new combinations with ease and simplified the movement algorithm. This also created a 2d map for 360 degrees, so it also has some good benefits as well. The Arduino and the other modules are placed on the top of the chassis. This is done this way to get quick access to the power cables and pins. The motors and the motor control driver are located on the bottom floor of the chassis.

The code is built from 2 sections as well, first section is the Arduino code, and the second section is the MATLAB code. The Arduino code controls the motors movement and gathers the sensor data and passes it to the MATLAB code. The MATLAB code does the mapping, localization, and robot control parts. The image below shows the basic diagram of our project:

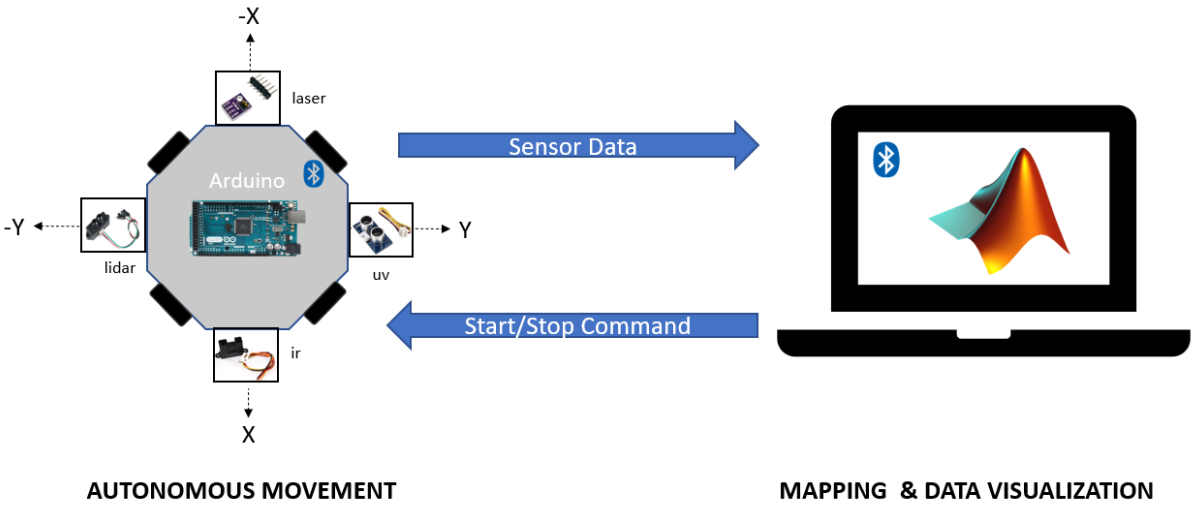

Figure 10: General communication and sensor placement diagram for the project

The working principle of the project is simple. First, the setup is updated both physically and code wise. MATLAB code requires updates if the sensor layouts change so that is the only manual part of this project. Later the robot waits for the start command from the computer. After that, the program cycle starts, and the sensors start collecting the data. The first 15 values are discarded to make sure that there is no problematic data caused by the initial power up.

If the front sensor of the robot is measuring greater distances than the selected minimum values, the robot starts moving forward until it reaches the limit. After this if the right is free the robot will turn right if not it will turn to the left. Then the movement algorithm will repeat

itself until the robot reaches the starting point. At this point the robot will get a stop command from the MATLAB code and stops the whole cycle. While all this happens, the robot will also send the sensor data for each cycle via the Bluetooth Module. The image below shows the system diagram for the robot and the Arduino code:

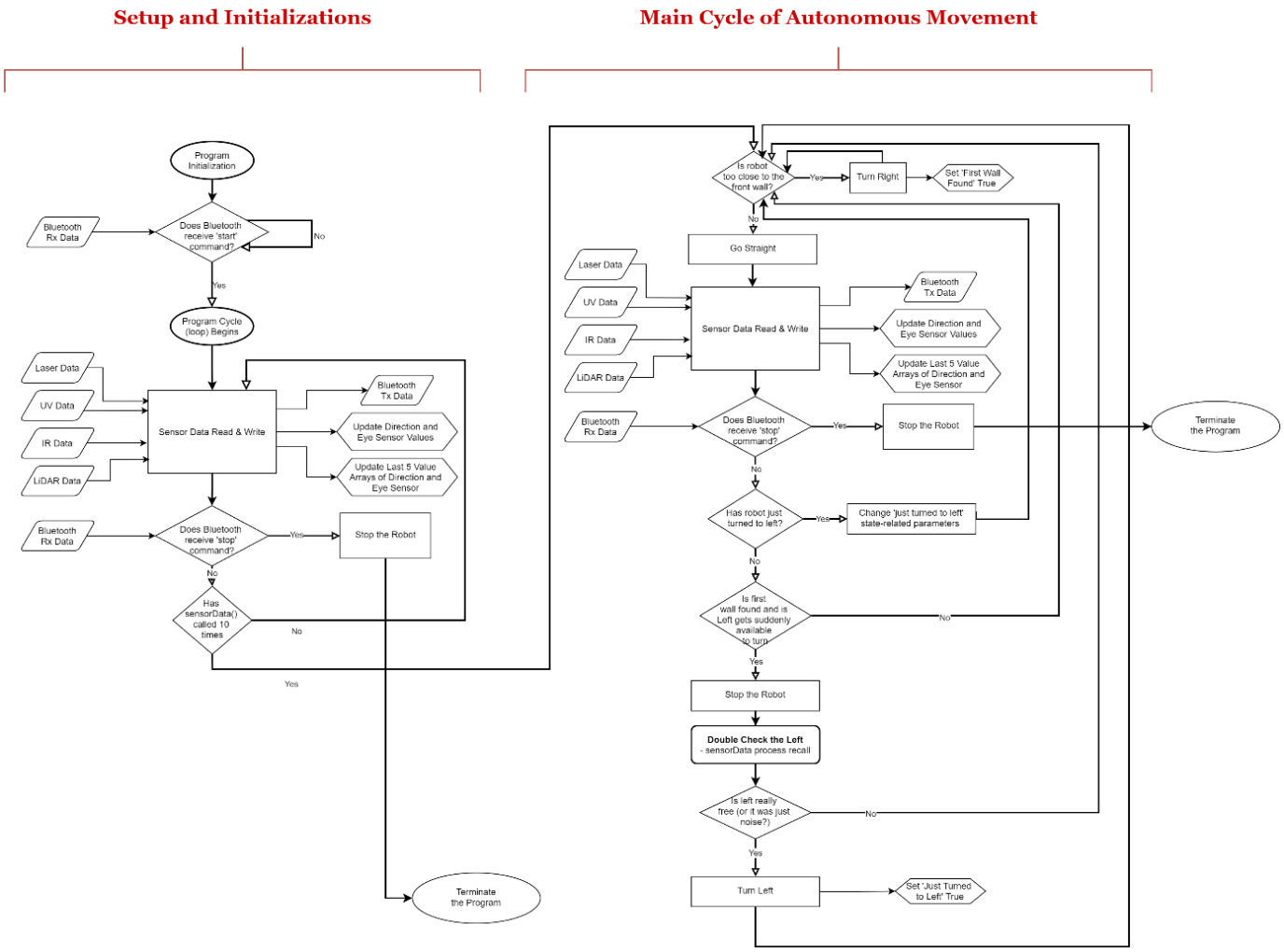

Figure 11: System diagram concerning the initialization and main cycle for the setup.

The MATLAB code is manually edited if the setup is altered. When the code starts running the start command will open the line between the robot and the computer. When the data comes through the port the values are assigned to some variables. These variables then call some related functions according to the movement direction and sensor values. The movement distance measurement for the front sensors data array for both x and y directions. The other sensor values are mapped referenced to these x and y values. If the robot turns the state value will change and this activates another function. This mapping code will run until the robot reaches the starting point for the second time. When this case happens the MATLAB code will send the stop command. The image below shows this process clearly:

MATLAB MAPPING

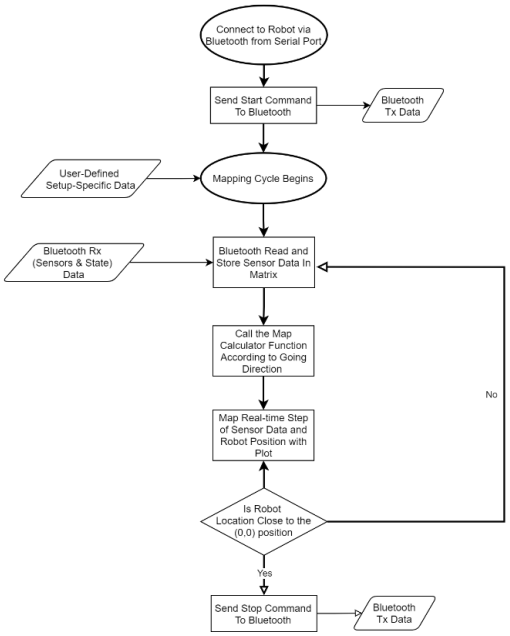

Figure 12: Diagram for MATLAB implementation of the project

3. RESULTS AND ANALYSIS

3.1 Test Environment and Approach

Sensors are allocated in different orders to see the performances of each sensor at each position. As it is previously mentioned, each sensor is placed to one end of the robot (front, right, back, and left) to observe the distance at those directions. In this sense, numerous robot setups are created and tested on the same test environment created by project owners. The following figures shows the constructed indoor area where the tests were run.

Figure 13: The environment where the tests took place.

The methodology is to use same automation movement code which is fed by front (direction sensor) and left (eye sensor) sensors. The locations of the four sensors were changed between the tests to see their performance change. In this report, the most remarkable tests are presented in detail while the rest is shortly explained to avoid data verbiage. The front (direction) sensor is important in terms of localization as well. Since the robot position is calculated based on the front sensor data, the wrongness of that data affects the entire map structure. As it is mentioned in system design section, the robot tends to turn right as long each time it sees a close wall in front. The left (eye) sensor is important to provide left turn impulse when it is needed. Other than that, left (eye) and right sensors are responsible of plotting the side walls while going ahead. The front and back sensors are the checker for the accuracy of different sensors within the different durations of the tests. In the following sections, these accuracy interpretations will be explained in detail.

3.2 Run Results for Different Robot Set-ups

In this section, the results of the runs for different sensor placement combinations will be analyzed. The runs are named according to the order of sensors. Each run will be named starting from front sensor to the others in clockwise direction such in the form “front-right-back-left”. In mapping graphs, data are represented by colors such as laser is dark blue, lidar is green, Ultrasonic is light blue, and IR is red. The robot position and movement are shown with black color. Finally, to reference which part of data is being analyzed in the analysis part, some yellow focus areas are shown and numbered on the graphs.

Run #1: Ultrasonic-IR-Lidar-Laser

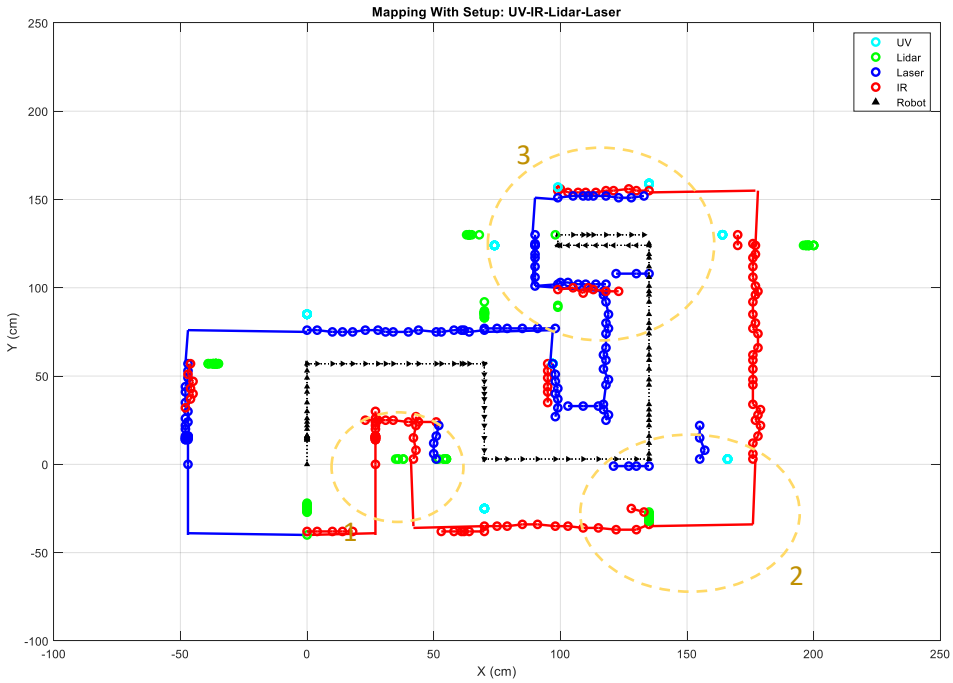

Figure 14: Test outcome with highlighted parts concerning sensor configuration of Ultrasonic-IR-Lidar-Laser

In this run, Ultrasonic is front and laser is left sensor. Considering the overall result, this setup is quite successful in terms of completing the map with small errors in the end. However, there is two major issues in this setup. First one is caused by the IR sensor's inability to measure close distances. The response of IR below 20 cm distance is directly reading 20 cm no matter how close the distance is. Therefore, in the first focus area, the reason why the mapping is kindly collapsing for different surface of the same wall. Since IR is close to the wall on the right of the robot repeatedly it draws to wall farther than it actually is. In this area, the laser data read after fourth turn is the one which correct place of right surface of the wall. The second problem of this setup is caused by laser sensor's inability to measure far distances. When the laser encounters a distance greater than 85 cm, it either reads an absurd data or reads some values around 80-85 cm. The effects of this problem can be seen at second and third focus areas. In these areas, laser draws ghost walls in the middle of the test environments. The actual distance that is read by laser should be the ones which were read by IR when robot went besides those walls. The ghost walls are constructed before third and fourth turn and after fifth turn when there occurs a free space on the left of the robot within the path.

Run #2: Ultrasonic-Laser-Lidar-IR

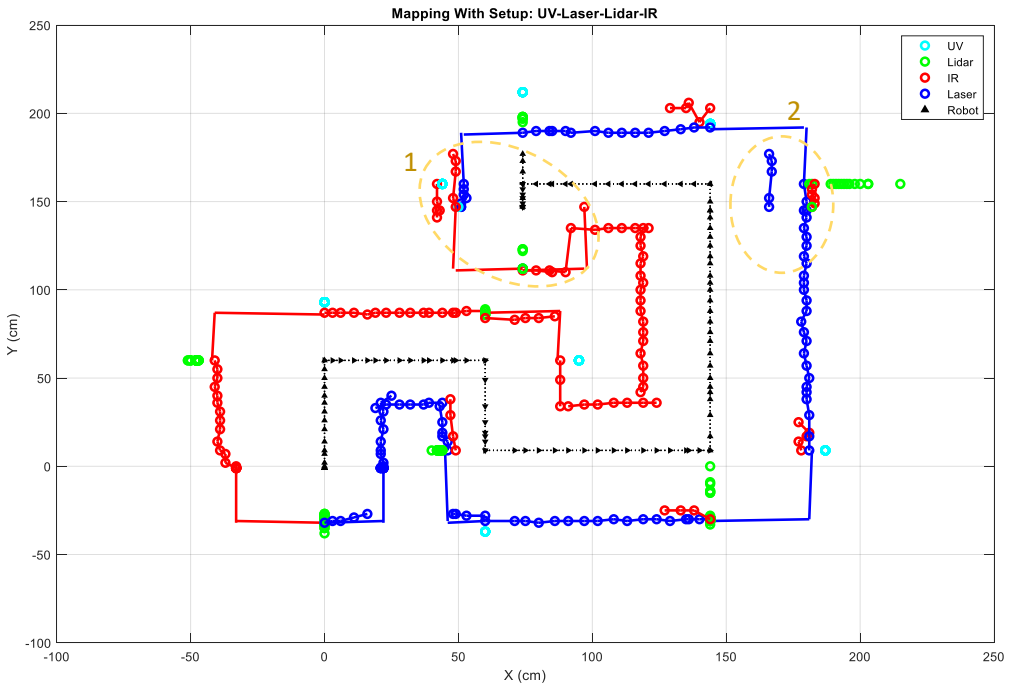

Figure 15: Test outcome concerning sensor configuration of Ultrasonic-Laser-Lidar-IR

In this run, different than the previous run, the laser and IR switch their places. Considering the overall result, this setup is better than even the previous one. The map is quite successfully completed, and the errors are fewer. The problems occur due to IR and laser ranging intervals can

be still observed in focus areas, but since IR sees farther distance more than laser while laser sees closer distances than IR, the final map is better.

Run #3: Laser-IR-Lidar-Ultrasonic

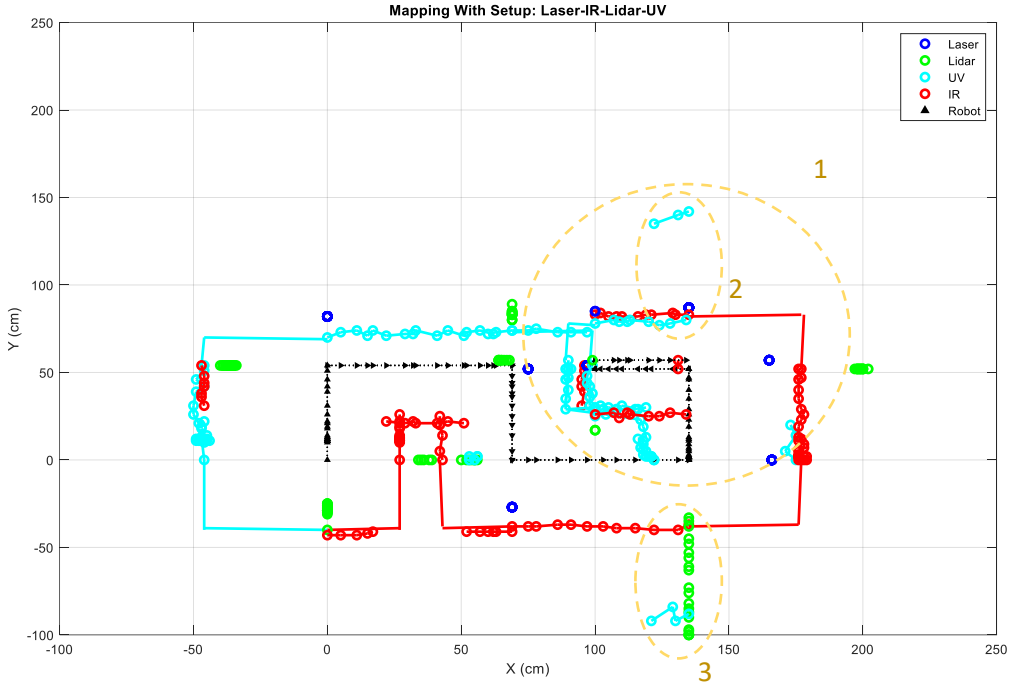

Figure 16: Test outcome concerning sensor configuration of Laser-IR-Lidar-Ultrasonic

In this run, the front sensor is laser and left sensor is Ultrasonic. Considering the overall result, this setup actually fails. The map could not be finished successfully due to major errors. The source of this error is laser being the front sensor. Its inability of reading larger distances was not problematic that much when it is used for left or right sensor. However, it is highly problematic once it is used as front sensor. Since the robot local positioning is based on the value of front sensor, the incorrectness of it causes whole map to shift even though the other sensor data are highly correct. In this run, once the robot has done its fourth turn and look in the y direction, laser faced with an approximate 2 meters distance which it is not capable of, it pins the distance at 80 cm away and updates its value to 80 cm repeatedly until it actually reaches 80 cm distance and then begin to read collect distance data. This leads robots changing side distance sensor data to accumulate based on non-changing point, and the upper part of the actual map collapse to the lower part in the end. This overlap is the reason of the mass in the focus are 1. The details of this problem can be also seen in the focus area 2 and 3. The dark blue value pinned by laser in the second focus area the wrong wall estimation of the robot. The light blue wall is the actual place where the wall should be which is read by Ultrasonic just before fourth turn of the robot. The green accumulative data of lidar in the third focus area is caused

by the laser estimating the distance has not changed while it is in fact changing. The lidar is reading larger values since the robot is getting far away from the behind wall. However, the laser data indicating the robot is still shows the behind wall as going farther back. Besides, the Ultrasonic data I this area is read once the robot did its fifth movement. Ultrasonic measures the actual distances but since the map is squeezed, the data is shown as at the out of the test environment.

Run #4: IR-Ultrasonic-Lidar-Laser

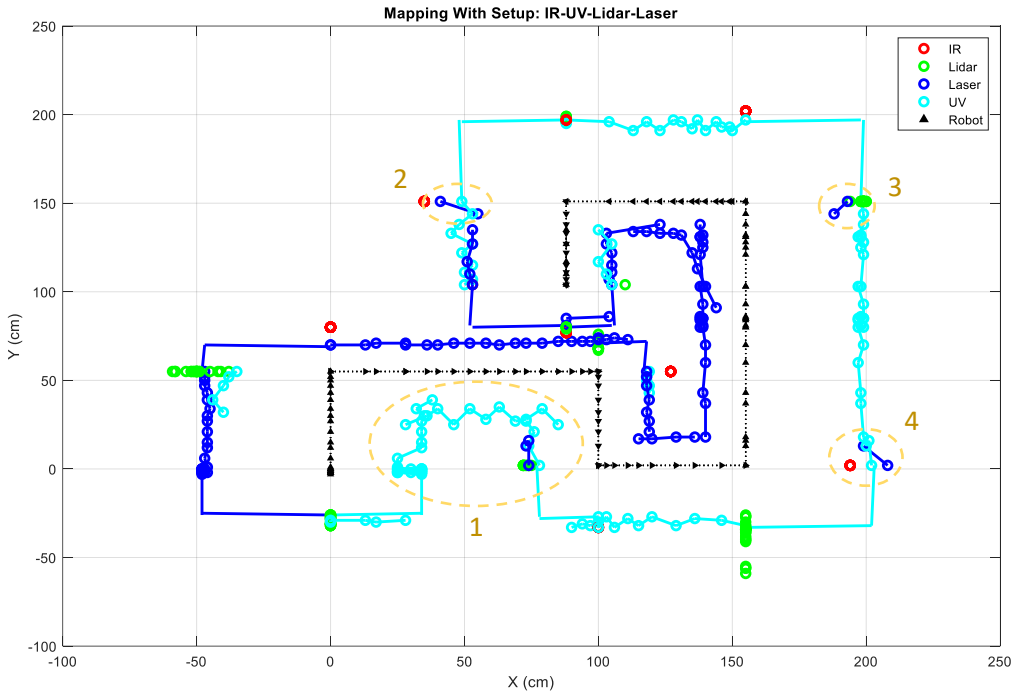

Figure 17: Test outcome concerning sensor configuration of IR-Ultrasonic-Lidar-Laser

In this run, the front sensor is IR and left sensor is laser. Considering the overall result, this setup can be thought as successful as well. The map can be finalized, and the ratios are fine. However, the data of Ultrasonic sensor is very noise at where the lidar reading very small distances. This means that Ultrasonic is affected by lidar sensors somehow due to some sudden voltage changes most probably while laser doing very good job no matter what lidar reading. The parts where Ultrasonic sees are not incorrect but unwantedly fluctuating so this is not very desirable outcome such as in focus are 1. In the focus areas 2, 3 and 4, we observe the laser long distance readings which successful compared to the first run. The reason is the closeness of the robot to the measured wall for this specific run. Since the right turns are controlled by front (direction) sensor IR, and its min distance to turn is different, the robot does close turns

to the wall. On the other hand, IR does very good job on reading the long distance so that this is the best realistic map in terms of height of the closed indoor area.

Other Runs of Different Setups

As other runs, making lidar the front sensor was tried and badly failed. The problem with the lidar is, this one is low-cost and low-performance lidar between the lidar sensors. In addition, its data reading rate does not match the rates of others. Which leads lidar come back due data accumulation in the buffer. In addition, some of the data may lost and the reading turns as empty results. These incompatibilities of the lidar with the rest of the system make TF Mini Lidar undesirable for the front and left sensor since its data reading make robot to hit the wall for most of the cases before finishing the mapping. Therefore, the runs where lidar is kept in the back were the most successful one presented in this paper in detail.

3.3 Final Assessments

With the use of the tests run over the test environment with different setup, sensors' collective performances within the runtime are evaluated based on the outcomes. In this part, the final comments regarding the purpose of this project will be presented. As it is mentioned, the aim was to investigate alternative distance measuring sensors that can be used instead of professional lidar. It is revealed that it is possible to build an indoor mapping application with the use of different optimal sensors by carefully building the design in the manner serving to the purpose.

Considering the results, the laser sensor, which is VL53L0X sensor module was one of the most fitting and stable sensors within all others. It performs highly well within its range and is not affected by the other components of the system. Its data is very stable and smooth. In terms of size and energy consumption it is also very good choice in terms of feasibility to small projects. However, it is not a good fit for the large indoor areas due to its range. It requires robot to move around longer since it should get closer to each detail of the environment. Nonetheless, since it is not affected by the other components much, it can be used as a part of the system for particular purposes.

IR sensor, which is Sharp GP2Y0A02YK0F Infrared Distance Sensor, is also quite good one especially in terms of measuring the long distances up 2 meters. Also, its accuracy was fairly okay considering the quality of the sensor. It much tends to make noise compared laser but still can be said very stable in general. However, it is observed for the some of the runs, it can be affected by the Lidar voltage fluctuations time to time which is thoughtful. In terms of

feasibility, it is bigger than laser but still small and low energy consumer. This sensor is not very desirable for small and very detailed environments due to its ranging but once it is kept in a distance to the wall, it works fine in general and could be highly preferable for open and less complicated indoor areas.

Ultrasonic sensor, which is Grove-Ultrasonic ranger V2.0, is bearable but not very preferable choice at least at this quality. It is very noisy and can be affected by the other electronic components very easily. Its fluctuation can be very misleading in routing the robot in the area. In addition, when battery begins to run out, it is the first sensor negatively affected and failed. For cases where Ultrasonic is individually used, its performance is actually quite good, and range is the overlap of both IR and laser sensors. Therefore, it may be suggested to use a higher quality Ultrasonic sensor for average size environment.

Lidar, which is TF Mini Lidar, is the sensor which is completely failed in experimental observations. It was a comparatively cheap model of lidar which is the primary reason why it is chosen. However, it is seemed that using low-budget lidar is not a wise alternative to pick instead of high-quality ones. If the purpose is to decrease the cost, preferring low-cost lidar is a wrong choice. Instead, other alternative distance measuring sensor, as it is detailly analyzed in this paper, should be chosen for indoor mapping applications. By the way, TF Mini Lidar seems quite successful for long range area when it is individually run, and the rate is perfectly arranged for its performance range. However, that rate is not feasible to a collective system to collaborate together.

To sum up, Laser and IR can be interactively and collaboratively used in a system and a perfect design can be achieved with the use of right algorithm and physical design. The specifics given in this paper will be enlightening the future designers in building their own applications.

4. CONCLUSION

In conclusion, in this project the main focus is to provide a deep analysis of different distance measurement sensor in an indoor mapping and autonomous driving project while building a SLAM operated indoor mapping robot. The theoretical findings, solo and in-system test results along with the student observations on these sensors and the other aspect of this project is provided in this report. Hopefully, this project shows some insight on how to build an indoor mapping robot and sensor comparisons. These four sensors that were mentioned above have their own advantages and disadvantages which results in some tradeoffs, but one can say that

after reviewing this article that the laser sensor is the best sensor among others. The robot built and the code that was written also explained in detail along with the ideas behind them.

5. REFERENCES

[1] Aideepen, (2016, May), *Arduino CJMCU-530 VL53L0X ToF Mesafe Ölçüm Sensörü Modülü*. https://www.direnc.net/tof-arduino-mesafe-sensoru-modulu-cjmcu-530-vl53l0x

[2] Seeedstudio, (2017, July), *Grove-Ultrasonic ranger V2.0*. https://www.direnc.net/grove-ultrasonik-mesafe-sensoru

[3] Sharp, (2006, December), *Sharp GP2Y0A02YK0F Infrared Distance Sensor*. *https://www.robotistan.com/sharp-gp2y0a02yk0f-infrared-distance-sensor-20-150cm*

[4] DFRobot, *TF Mini LiDAR (ToF) Laser Range Sensor*, https://www.direnc.net/tf-mini-lidar-tof-lazer-menzil-sensoru-dfrobot-en

[5] Arduino, *Arduino Bord Mega 2560*, *https://www.arduino.cc/en/Main/arduinoBoardMega/*

[6] Okn Teknoloji, *HG7881 4 Kanallı Dc Motor Sürücü Kartı, https://www.hepsiburada.com/okn-tekoloji-hg7881-4-kanalli-dc-motor-surucu-karti-p-HBCV000001WD6H*

[7] HAILANGNIAO, *HC 05 RF Wireless Bluetooth Transceiver Modulo*, *https://it.aliexpress.com/item/32872195579.html*

6. APPENDICES

A. Arduino Code: Data Read and Autonomous Movement

```
// LIBS
#include <Wire.h>
#include <VL53L0X.h>       // laser sensor
#include "Ultrasonic.h"    // Ultrasonicsensor
#include <SharpIR.h>       // IR sensor
#include <DFRobot_TFmini.h> // Lidar sensor
#include <I2Cdev.h>
#include <ADXL345.h>  // ADXL345 Accelerometer Library
#include <HMC5883L.h> // HMC5883L Magnetometer Library
#include <ITG3200.h>  // ITG3200 Gyroscope Library
#include <math.h>

// PIN CONFIG
#define IRPin A0
#define IRmodel 20150
#define UVPin 52
SoftwareSerial LidarSerial(13, 12); // RX, TX
SoftwareSerial Bluetooth(10, 11); // RX, TX

// SENSOR OBJECTS
```

```
DFRobot_TFmini  TFmini; /* Model : GP2Y0A02YK0F --> 20150 GP2Y0A21YK0F --> 1080
GP2Y0A710K0F --> 100500 GP2YA41SK0F --> 430 */
SharpIR mySensor = SharpIR(IRPin, IRmodel);   // Analog, 5V
Ultrasonic ultrasonic(UVPin);                 // UV: Digital Pin, 5V
VL53L0X laser;                                // LASER: SDA - SCL, 5V

// SENSOR DISTANCES
long UVdistance_cm;
long IRdistance_cm;
long LaserDistance_cm;
uint16_t LidarDistance,strength;

// SENSOR OFFSETS
long IROffset = 2.0;
long UVOffset = 2.5;
long LaserOffset = -3.0;
long LidarOffset = 0;

//---------------------- MOTORS AND DRIVER PARAMETERS
int MOTOR_A1 = 39; //brown
int MOTOR_A2 = 41; //red
int MOTOR_B1 = 43; //orange
int MOTOR_B2 = 45; //yellow
int MOTOR_C1 = 47; //green
int MOTOR_C2 = 49; //blue
int MOTOR_D1 = 51; //purple
int MOTOR_D2 = 53; //white

  /* dirs for goStraight() */
int UVdir = 1;
int LASERdir = 4;
int IRdir = 2;
int LIDARdir = 3;
  /* dirs for diagonalMove()*/
int UV_Las = 1;
int Las_Lid = 2;
int Lid_IR = 3;
int IR_UV = 4;
  /* dirs for wheel turns*/
int right = 1;
int left = 2;
int breaks = 3;
int released = 4;
//---------------------- MOTORS AND DRIVER PARAMETERS ENDS


String start = "start";
String stopp = "stop";
String command = "";


// Sensor Limits
  int UVmin = 5;
  int UVmin_stop = 22;
  int UVmax = 80;
  int IRmin = 20;
  int IRmin_stop = 22;
  int IRmax = 65;
  int LIDARmin = 30;
  int LIDARmin_stop = 30;
  int LIDARmax = 140;
  int LASERmin = 7;
  int LASERmin_stop = 18;
  int LASERmax = 80;

// Automated Movement Vars
  int dirToGo = UVdir;
  int goingDir = LASERdir;    // ********* USER DEFINED
  int dirSensorVal;
```

```
  int dirSensorMin;
  int eyeSensorVal;
  int eyeSensorMin;
  int keptEyeSensorVal;
  int directionState = 0;
  int loopCounter = 0;
  float toleranceDist = 6;
  bool eyeSensorMinFixed = false;
  bool wallFound = false;
  bool justLeftTurned = false;
  int leftTurnWaiter = 0;
  int prevEyeSensorVal[] = {0,0,0,0,0};
  int prevDirSensorVal[] = {0,0,0,0,0};


// Bluetooth Waited Commands
void waitStartCommand(){
   while(Bluetooth.available()== 0){}
   command = Bluetooth.readString();
   command.trim();
   while(!command.equalsIgnoreCase(start)){
      while(Bluetooth.available()==0){}
      command = Bluetooth.readString();
      command.trim();
   }
}

bool hasStopCmdCome(){
   command = Bluetooth.readString();
   command.trim();
   if(!command.equalsIgnoreCase(stop)){
      return true;
   }
   return false;
}

// ************************ PROGRAM SETUP ********************************
void setup()
{
  // MOTOR INITIALIZATION
  pinMode( MOTOR_A1, OUTPUT );
  pinMode( MOTOR_A2, OUTPUT );
  pinMode( MOTOR_B1, OUTPUT );
  pinMode( MOTOR_B2, OUTPUT );
  pinMode( MOTOR_C1, OUTPUT );
  pinMode( MOTOR_C2, OUTPUT );
  pinMode( MOTOR_D1, OUTPUT );
  pinMode( MOTOR_D2, OUTPUT );
  digitalWrite( MOTOR_A1, LOW );
  digitalWrite( MOTOR_A2, LOW );
  digitalWrite( MOTOR_B1, LOW );
  digitalWrite( MOTOR_B2, LOW );
  digitalWrite( MOTOR_C1, LOW );
  digitalWrite( MOTOR_C2, LOW );
  digitalWrite( MOTOR_D1, LOW );
  digitalWrite( MOTOR_D2, LOW );

  // SERIAL INITIALIZATIONS (Bluetooth, Lidar, Arduino Serial)
  pinMode(10, INPUT);
  pinMode(11, OUTPUT);
  pinMode(13, INPUT);
  pinMode(12, OUTPUT);
  Serial.begin(9600);
  delay(200);
  Bluetooth.begin(9600);
  delay(200);
  waitStartCommand();
  delay(200);
  Wire.begin();
```

```
  TFmini.begin(LidarSerial);

  // LASER SETUP
  laser.setTimeout(500);
  if (!laser.init()){
    Bluetooth.println("Failed to detect and initialize laser!");
  }
  laser.startContinuous();

}


// ------------------ FINDING GOING DIR AND OBSERVATION DIR ----------
void updateDirectionAndEyeSensor(){
    if(goingDir == 1){
        if(UVdistance_cm == 2){
          dirSensorVal = prevDirSensorVal[0];
          UVdistance_cm = dirSensorVal;
        }else{
           dirSensorVal = UVdistance_cm;
        }
        dirSensorMin = UVmin_stop;
        eyeSensorVal = LaserDistance_cm;
        eyeSensorMin = LASERmin_stop;
    }else if(goingDir == 2){
        dirSensorVal = IRdistance_cm;
        dirSensorMin = IRmin_stop;
        eyeSensorVal = LaserDistance_cm;
        eyeSensorMin = LASERmin_stop;
    }else if(goingDir == 3){
        dirSensorVal = LidarDistance;
        dirSensorMin = LIDARmin_stop;
        eyeSensorVal = LaserDistance_cm;
        eyeSensorMin = LASERmin_stop;
    }else if(goingDir == 4){
        dirSensorVal = LaserDistance_cm;
        dirSensorMin = LASERmin_stop;
        eyeSensorVal = IRdistance_cm;
        eyeSensorMin = IRmin_stop;

    }
}
// ------------------ FINDING GOING DIR AND OBSERVATION DIR ENDS ----------



// **** TURN FUNCS
void turn90(int LR){
  turnTo(LR);
  delay(685);

  if(LR == left){
  justLeftTurned = true;
    delay(43);
    directionState--;
  }else if(LR == right){
    directionState++;
  }
  if(directionState == -1){
    directionState = 3;
  }
  directionState = directionState % 4;
  stopWith(released);
  delay(50);
  goStraightIn(dirToGo);
  delay(300);
}

bool LeftIsFree(int n){
```

```
  int  prev = prevEyeSensorVal[n];
  if(justLeftTurned){
    leftTurnWaiter++;
  }
  if(leftTurnWaiter == 4){
    justLeftTurned = false;
    leftTurnWaiter = 0;
  }

  if(!justLeftTurned){
    if(eyeSensorVal > prev + 15){
                return true;
        }
  }
  return false;

}

// *************** PROGRAM CYCLE *******************************
void loop(){
  sensorData();

  if(loopCounter>10){
      while(dirSensorVal>dirSensorMin){
        if(hasStopCmdCome()){                                  // terminates the robot
            stopWith(breaks);
            while(true){ delay (10); }             // inf loop

        }
        goStraightIn(dirToGo);
        sensorData();
        if(wallFound && LeftIsFree(1)){     // first degree check, if wall is
found.
          stopWith(breaks);
          sensorData();
          if(LeftIsFree(2)){               // double check
            goStraightIn(dirToGo);
            delay(200);
             sensorData();
            turn90(left);
          }
        }
      }
      stopWith(breaks);
      delay(500);
      turn90(right);
      wallFound = true;
  }

  loopCounter++;
}
// *************** PROGRAM CYCLE ENDS ********************************


// ------------------ SENSOR READ AND WRITE TO BLUETOOTH --------------------------
void sensorData(){

      // LASER CODE Read&Write
      LaserDistance_cm = laser.readRangeContinuousMillimeters()/10.0 + LaserOffset;
      Bluetooth.print(LaserDistance_cm);
      Bluetooth.print("\t");
      if (laser.timeoutOccurred()) {
                Bluetooth.println('Laser Timeout!');
      }

      // UV CODE Read&Write
      UVdistance_cm = ultrasonic.MeasureInMillimeters()/10.0 + UVOffset; // two
measurements should keep an interval
      if(UVdistance_cm == 2){
```

```
            Bluetooth.print(prevDirSensorVal[0]);//0~400cm
        }else{
            Bluetooth.print(UVdistance_cm);//0~400cm
        }
        Bluetooth.print("\t");


        // IR CODE Read&Write
        IRdistance_cm = mySensor.distance() + IROffset;
        Bluetooth.print(IRdistance_cm);
        Bluetooth.print("\t");

         // LIDAR CODE Read&Write
        if(TFmini.measure()){
          LidarDistance = TFmini.getDistance() + LidarOffset;
          Bluetooth.print(LidarDistance);
        }else{
          Bluetooth.print("999");
        }

     // STATE Read&Write
        Bluetooth.print("\t");
        Bluetooth.println(directionState+1);


        delay(200);

        updateDirectionAndEyeSensor();
        prevKeeper();
}
// ------------------ SENSOR READ AND WRITE TO BLUETOOTH ENDS  --------------------

// ------------------ SENSOR DATA BUFFERS ----------------------------------------
void prevKeeper(){
   int temp[] = {0,0,0,0,0};
   for(int i=0;i<5;i++){
      temp[i] = prevEyeSensorVal[i];
   }
   prevEyeSensorVal[0] = eyeSensorVal;
   for(int i=0; i<4; i++){
      prevEyeSensorVal[i+1] = temp[i];
   }

   for(int i=0;i<5;i++){
      temp[i] = prevDirSensorVal[i];
   }
   prevDirSensorVal[0] = eyeSensorVal;
   for(int i=0; i<4; i++){
      prevDirSensorVal[i+1] = temp[i];
   }
}
// ------------------ SENSOR DATA BUFFERS ENDS -----------------------------------

//---------------------- MOTORS AND DRIVER FUNCTIONS ---------------
void turnWithFrontWheels(int dir, int LR){

     if(dir==UVdir){
       motorA(left);
       motorB(left);
       motorC(right);
       motorD(right);
   }else if(dir==LASERdir){
       motorA(right);
       motorB(left);
       motorC(left);
       motorD(right);
   }else if(dir==LIDARdir){
       motorA(right);
       motorB(right);
```

```
      motorC(left);
      motorD(left);
   }else if(dir==IRdir){
      motorA(left);
      motorB(right);
      motorC(right);
      motorD(left);
   }
}
  /* goes in the direction of selected sensor*/
void goStraightIn(int dir){
   if(dir==UVdir){
      motorA(left);
      motorB(left);
      motorC(right);
      motorD(right);
   }else if(dir==LASERdir){
      motorA(right);
      motorB(left);
      motorC(left);
      motorD(right);
   }else if(dir==LIDARdir){
      motorA(right);
      motorB(right);
      motorC(left);
      motorD(left);
   }else if(dir==IRdir){
      motorA(left);
      motorB(right);
      motorC(right);
      motorD(left);
   }
}
  /* turns to the given direction*/
void turnTo(int dir){
      motorA(dir);
      motorB(dir);
      motorC(dir);
      motorD(dir);
}
  /* stops with break or releasing wheels*/
void stopWith(int choice){
      motorA(choice);
      motorB(choice);
      motorC(choice);
      motorD(choice);
}
  /* turns to the given direction*/
void diagonalMove(int dir){
   if(dir==UV_Las){
      motorA(released);
      motorB(left);
      motorC(released);
      motorD(right);
   }else if(dir==Las_Lid){
      motorA(right);
      motorB(released);
      motorC(left);
      motorD(released);
   }else if(dir==Lid_IR){
      motorA(released);
      motorB(right);
      motorC(released);
      motorD(left);
   }else if(dir==IR_UV){
      motorA(left);
      motorB(released);
      motorC(right);
      motorD(released);
```

```
    }
  }
  //---------------------- MOTORS AND DRIVER FUNCTIONS ENDS ---------------

  //---------------------- MOTOR CONFIGURATIONS FOR DRIVER ---------------
  void motorA(int dir) {
    if (dir == 1) {
      digitalWrite(MOTOR_A1, HIGH);
      digitalWrite(MOTOR_A2, LOW);
    } else if (dir == 2) {
      digitalWrite(MOTOR_A1, LOW);
      digitalWrite(MOTOR_A2, HIGH);
    } else if (dir == 3) {
      digitalWrite(MOTOR_A1, HIGH);
      digitalWrite(MOTOR_A2, HIGH);
    } else {
      digitalWrite(MOTOR_A1, LOW);
      digitalWrite(MOTOR_A2, LOW);
    }
  }
  void motorB(int dir) {
    if (dir == 1) {
      digitalWrite(MOTOR_B1, HIGH);
      digitalWrite(MOTOR_B2, LOW);
    } else if (dir == 2) {
      digitalWrite(MOTOR_B1, LOW);
      digitalWrite(MOTOR_B2, HIGH);
    } else if (dir == 3) {
      digitalWrite(MOTOR_B1, HIGH);
      digitalWrite(MOTOR_B2, HIGH);
    } else {
      digitalWrite(MOTOR_B1, LOW);
      digitalWrite(MOTOR_B2, LOW);
    }
  }
  void motorC(int dir) {
    if (dir == 1) {
      digitalWrite(MOTOR_C1, HIGH);
      digitalWrite(MOTOR_C2, LOW);
    } else if (dir == 2) {
      digitalWrite(MOTOR_C1, LOW);
      digitalWrite(MOTOR_C2, HIGH);
    } else if (dir == 3) {
      digitalWrite(MOTOR_C1, HIGH);
      digitalWrite(MOTOR_C2, HIGH);
    } else {
      digitalWrite(MOTOR_C1, LOW);
      digitalWrite(MOTOR_C2, LOW);
    }
  }
  void motorD(int dir) {
    if (dir == 1) {
      digitalWrite(MOTOR_D1, HIGH);
      digitalWrite(MOTOR_D2, LOW);
    } else if (dir == 2) {
      digitalWrite(MOTOR_D1, LOW);
      digitalWrite(MOTOR_D2, HIGH);
    } else if (dir == 3) {
      digitalWrite(MOTOR_D1, HIGH);
      digitalWrite(MOTOR_D2, HIGH);
    } else {
      digitalWrite(MOTOR_D1, LOW);
      digitalWrite(MOTOR_D2, LOW);
    }
  }
  //---------------------- MOTOR CONFIGURATIONS FOR DRIVER ENDS ---------------
```

B. MATLAB Code: Mapping Algorithm and Visualization

```
clear all
close all
```

INITIALIZATIONS

```
s = serialport("COM8",9600);    % Bluetooth Connection
A = zeros(5,300);
d = zeros(2,300);
i = 1;
s.writeline('start');           % Bluetooth Start Command
x=0;
y=0;
```

CHANGE HERE FOR EACH SET-UP

Arranges Plotting Parameters and Mapping Inputs According to Set-up

```
FrontSide = 2;
RightSide = 3;
BackSide = 4;
LeftSide = 1;
% colors
frontCol = "c";
backCol = "g";
rightCol = "r";
leftCol = "b";
setup = 'UV-IR-Lidar-Laser';    % for title
```

Plotting the Map

```
i=1;
figure
while 1
    % Bluetooth Data Delivery and Assignments
    i;
      data = readline(s);
      trimmed_data = split(data);
      trimmed_data(6) = [];
      B = str2double(trimmed_data);
      B = B'

      % Bluetooth Data Delivery
      A(1,i) = B(1); % Laser
      A(2,i )= B(2); % UV
      A(3,i) = B(3); % IR
      A(4,i) = B(4); % LIDAR
      A(5,i) = B(5); % State


    % MAPPING
    if i==1
        front1 = A(FrontSide,1)+5;
        right1 = A(RightSide,1)+5;
```

```
        back1 = A(BackSide,1)+5;
        left1 = A(LeftSide,1)+5;

        front_1 = front1;
        right_1 = right1;
        back_1 = back1;
        left_1 = left1;

        plot([0, right1],[-back1, -back1+1],rightCol,"LineWidth",2);
        hold on
        plot([0,-left1],[-back1,-back1+1],leftCol,"LineWidth",2);
        hold on
        plot([-left1,-left1],[-back1,0],leftCol,"LineWidth",2);
        hold on
        plot([right1,right1],[-back1,0],rightCol,"LineWidth",2);
    else
        front_1 = A(FrontSide,i-1)+5;
        right_1 = A(RightSide,i-1)+5;
        back_1 = A(BackSide,i-1)+5;
        left_1 = A(LeftSide,i-1)+5;
    end

    fronti = A(FrontSide,i)+5;
    righti = A(RightSide,i)+5;
    backi = A(BackSide,i)+5;
    lefti = A(LeftSide,i)+5;


    statei = A(5,i);
    if i ~= 1
        state_1 = A(5,i-1);
    else
        state_1 = 0;
    end

    % PLOTTING
    x_old = x;
    y_old = y;
    switch statei
        case 1
            [x,y,backi,fronti,righti,lefti,front_1,back_1,right_1,left_1] =
forward_y(x_old,y_old,x,y,front_1,back_1,right_1,left_1,fronti,backi,righti,lefti,statei,state
_1,frontCol,backCol,rightCol,leftCol,i);
            if(i==2)
                title(['Mapping With Setup: ', setup ]);
                xlabel('X (cm)');
                ylabel('Y (cm)');
                grid on
            else
                scatter(x,y,20,"k^",'filled');
            end

        case 2
            [x,y,backi,fronti,righti,lefti,front_1,back_1,right_1,left_1] =
forward_x(x_old,y_old,x,y,front_1,back_1,right_1,left_1,fronti,backi,righti,lefti,statei,state
_1,frontCol,backCol,rightCol,leftCol);
            scatter(x,y,20,"k>",'filled');
        case 3
            [x,y,backi,fronti,righti,lefti,front_1,back_1,right_1,left_1] =
```

```
backward_y(x_old,y_old,x,y,front_1,back_1,right_1,left_1,fronti,backi,righti,lefti,statei,stat
e_1,frontCol,backCol,rightCol,leftCol);
            scatter(x,y,20,"kv",'filled');
        case 4
            [x,y,backi,fronti,righti,lefti,front_1,back_1,right_1,left_1] =
backward_x(x_old,y_old,x,y,front_1,back_1,right_1,left_1,fronti,backi,righti,lefti,statei,stat
e_1,frontCol,backCol,rightCol,leftCol);
            scatter(x,y,20,"k<",'filled');
    end

    if(i>20)
        if(abs(x)<5 && abs(y)<5)
            s.writeline('stop');
            break
        end
    end


    for j = 1:4
        axis([-100 250 -100 250]);
        hold on;
        plot([x_old,x],[y_old,y],"k:","LineWidth",1);
    end

    drawnow;
    i=i+1;
end
```

FUNCTIONS

Mapper for Paths in Y-Direction

```
function [x,y,backi,fronti,righti,lefti,front_1,back_1,right_1,left_1] =
forward_y(x_old,y_old,x,y,front_1,back_1,right_1,left_1,fronti,backi,righti,lefti,statei,state
_1,frontCol,backCol,rightCol,leftCol,i)

    % Map and Localization Calculations
    left_raw = lefti;
    right_raw = righti;
    x = x;
    if state_1 == statei
        y = y+front_1-fronti;
    else
        y = y;
    end
    backi = y-backi;
    fronti = y+fronti;
    righti = x + righti;
    lefti = (x - lefti);


    % Scatters
    if(i==2)
        p1 = scatter(x,fronti,frontCol,"LineWidth",2,'DisplayName',sensName(frontCol));
        hold on;
        p2 = scatter(x,backi,backCol,"LineWidth",2,'DisplayName',sensName(backCol));
        hold on;
        p3 =  scatter(lefti,y,leftCol,"LineWidth",2,'DisplayName',sensName(leftCol));
```

```
        hold on;
        p4 = scatter(righti,y,rightCol,"LineWidth",2,'DisplayName',sensName(rightCol));
        hold on;
        p = scatter(x,y,20,"k^",'filled','DisplayName','Robot');
        legend([p1 p2 p3 p4 p],{sensName(frontCol), sensName(backCol), sensName(leftCol),
sensName(rightCol), 'Robot'},'AutoUpdate','off');
    else
        scatter(x,fronti,frontCol,"LineWidth",2);
        hold on;
        scatter(x,backi,backCol,"LineWidth",2);
        hold on;
        scatter(lefti,y,leftCol,"LineWidth",2);
        hold on;
        scatter(righti,y,rightCol,"LineWidth",2);
        hold on;
    end

    % Plot Cases
    if abs(left_raw-left_1)<30 && (state_1 == statei)
        plot([(x_old-left_1),lefti],[y_old,y],leftCol,"LineWidth",2);
        hold on;
    end

    if abs(right_raw-right_1)<30 && (state_1 == statei)
        plot([(x_old+right_1),righti],[y_old,y],rightCol,"LineWidth",2);
        hold on;
    end

    % Corner Cases
    if state_1 == 4
        plot([lefti,x],[y-1-left_1,y-left_1],leftCol,"LineWidth",2);
        hold on
        plot([x-left_raw-1,x-left_raw],[y-left_1,y],leftCol,"LineWidth",2);
        hold on
    end
    if state_1 == 2
        plot([righti,righti+1],[y-right_1,y],rightCol,"LineWidth",2);
        hold on;
        plot([x,righti],[y-1-right_1,y-right_1],rightCol,"LineWidth",2);
        hold on
    end
end
```

Mapper for Paths in (-Y)-Direction

```
function [x,y,backi,fronti,righti,lefti,front_1,back_1,right_1,left_1] =
backward_y(x_old,y_old,x,y,front_1,back_1,right_1,left_1,fronti,backi,righti,lefti,statei,stat
e_1,frontCol,backCol,rightCol,leftCol)
    % Map and Localization Calculations
    left_raw = lefti;
    right_raw = righti;
    x = x;
    if state_1 == statei
        y = y-(front_1-fronti);
    else
        y = y;
    end
```

```
    backi = y+backi;
    fronti= y-fronti;
    righti = x - righti;
    lefti = (lefti+x);

    % Scatters
    scatter(x,fronti,frontCol,"LineWidth",2);
    hold on;
    scatter(x,backi,backCol,"LineWidth",2);
    hold on;
    scatter(lefti,y,leftCol,"LineWidth",2);
    hold on;
    scatter(righti,y,rightCol,"LineWidth",2);
    hold on;

    % Plot Cases
    if abs(left_raw-left_1)<30 && (state_1 == statei)
        plot([(x_old+left_1),lefti],[y_old,y],leftCol,"LineWidth",2);
        hold on;
    end
    if abs(right_raw-right_1)<30 && (state_1 == statei)
        plot([(x_old-right_1),righti],[y_old,y],rightCol,"LineWidth",2);
        hold on;
    end

    % Corner Cases
    if state_1 == 2
        plot([lefti-1,lefti],[y,y+left_1],leftCol,"LineWidth",2);
        hold on;
        plot([x,x+left_raw],[y+left_1-1,y+left_1],leftCol,"LineWidth",2);
           hold on
    end
     if state_1 == 4
        plot([x-right_raw,x],[y+right_1-1,y+right_1],rightCol,"LineWidth",2);
        hold on;
       plot([x-right_raw-1,x-right_raw],[y+right_1,y],rightCol,"LineWidth",2);
       hold on
    end
end
```

Mapper for Paths in X-Direction

```
function [x,y,backi,fronti,righti,lefti,front_1,back_1,right_1,left_1] =
forward_x(x_old,y_old,x,y,front_1,back_1,right_1,left_1,fronti,backi,righti,lefti,statei,state
_1,frontCol,backCol,rightCol,leftCol)
    % Map and Localization Calculations
    left_raw = lefti;
    right_raw = righti;
    if state_1 == statei
        x = x+front_1-fronti;
    else
        x = x;
    end
    lefti = y+lefti;
    righti = y-righti;
    fronti = x + fronti;
    backi = x-backi;
```

```
    % Scatters
    scatter(fronti,y,frontCol,"LineWidth",2);
    hold on;
    scatter(backi,y,backCol,"LineWidth",2);
    hold on;
    scatter(x,lefti,leftCol,"LineWidth",2);
    hold on;,
    scatter(x,righti,rightCol,"LineWidth",2);
    hold on;

    % Plot Cases
    if abs(left_raw-left_1)<30 && (state_1 == statei)
        plot([x_old,x],[(y_old+left_1),lefti],leftCol,"LineWidth",2);
        hold on;
    end
    if abs(right_raw-right_1)<30 && (state_1 == statei)
        plot([x_old,x],[(y_old-right_1),righti],rightCol,"LineWidth",2);
        hold on;
    end

    % Corner Cases
    if state_1 == 1
        plot([x,x-left_1],[lefti-1,lefti],leftCol,"LineWidth",2);
        hold on
        plot([x_old-left_1-1,x_old-left_1],[y,lefti],leftCol,"LineWidth",2);
        hold on
    end
    if state_1 == 3
        plot([x-right_1, x],[y-right_raw-1,y-right_raw],rightCol,"LineWidth",2);
        hold on
        plot([x-right_1-1,x-right_1],[y,y-right_raw],rightCol,"LineWidth",2);
        hold on
    end
end
```

Mapper for Paths in (-X)-Direction

```
function [x,y,backi,fronti,righti,lefti,front_1,back_1,right_1,left_1] =
backward_x(x_old,y_old,x,y,front_1,back_1,right_1,left_1,fronti,backi,righti,lefti,statei,stat
e_1,frontCol,backCol,rightCol,leftCol)
    % Map and Localization Calculations
    left_raw = lefti;
    right_raw = righti;
    y = y;
    if state_1 == statei
        x = x-(front_1-fronti);
    else
        x=x;
    end
    lefti = y-lefti;
    righti = y+righti;
    fronti = x - fronti;
    backi = x+backi;

   % Scatters
    scatter(fronti,y,frontCol,"LineWidth",2);
```

```
    hold on;
    scatter(backi,y,backCol,"LineWidth",2);
    hold on;
    scatter(x,lefti,leftCol,"LineWidth",2);
    hold on;
    scatter(x,righti,rightCol,"LineWidth",2);
    hold on;

    % Plot Cases
    if abs(left_raw-left_1)<30 && (state_1 == statei)
        plot([x_old,x],[(y_old-left_1),lefti],leftCol,"LineWidth",2);
        hold on;
    end
    if abs(right_raw-right_1)<30 && (state_1 == statei)
        plot([x_old,x],[(y_old+right_1),righti],rightCol,"LineWidth",2);
        hold on;
    end

    % Corner Cases
    if state_1 == 1
        plot([x+right_1,x+right_1+1],[y,righti],rightCol,"LineWidth",2);
        hold on;
        plot([x,x+right_1],[righti-1,righti],rightCol,"LineWidth",2);
        hold on
    end
    if state_1 == 3
        plot([x+left_1,x+left_1+1],[y,lefti],leftCol,"LineWidth",2);
        hold on;
        plot([x+left_1,x],[lefti,lefti-1],leftCol,"LineWidth",2);
        hold on
    end
end
```

Name Assigner for Plot Legend

```
function name = sensName(color)
   switch color
       case "c"
           name = 'UV';
       case "r"
           name = 'IR';
       case "b"
           name = 'Laser';
       case "g"
           name = 'Lidar';
       otherwise
           name = 'ERROR';
   end
end
```

Published with MATLAB® R2020b