Navigation Algorithms with LIDAR for Mobile Robots

Cristian MOLDER, Daniel TOMA, and Andrei ȚIGĂU

***Abstract*—For a robot that performs autonomously, the communication between the person and the robot is the most important factor. Significant awareness has been observed regarding the usage of such technology. This research has trivial involvement in the development of such robots. A robot that functions fully autonomously should not only complete the jobs that are desired of them but also somehow establish a connection between themselves and the person operating them. We propose a human detection method that uses only a single laser range scanner to detect the waist of the target person. The speed and acceleration of a robot are adaptive to the human-walking speed and the distance between the human and robot.**

***Index Terms*—human following, human robot interaction, laser range scanner, depth image mapping.**

I. Introduction

Robotics - the collaboration of sensing, ad hoc networks and microcontrollers, in the real world is on the edge of vast growth. The growth is vastly supported from expanding availability of sensors, computing devices, actuators and microcontrollers. The demand for the new technology is driven by daily needs and national needs for defense and security, elder care, automation of household tasks, customized manufacturing, and interactive entertainment. Engineers working in the robotics industry are mostly trained in one of Computer Engineering, Computer Science, Electrical Engineering, Mechanical Engineering, and Software Engineering. No single discipline provides the breadth demanded by robotics in the future.

To perform this task accurately, robot needs a mechanism that enables it to visualize the person and act accordingly. The robot must be intelligent enough to follow a person in vivid environment and in indoor and outdoor places. The image processing carried out to get the information about the surroundings visually is a very important thing.

The following points should be carefully noted while doing the processing [1, 2].

- The luminosity conditions should be very stable and should not fluctuate.

- The ranges should be set properly for the desired environment on which to perform the tracking. The target should not be very far from the visual sensor as the distance matters a lot.

- We should avoid the use of such colors around the robot that matches with that of the target. Otherwise the robot would get confused.

Typically human following robots are equipped with several different diverse combination of sensors i.e. light detection and ranging sensor, radio frequency identification module (RFID), laser ranger finder (LFR), infrared (IR) sensing modules, thermal imaging sensors, camera, wireless transmitter/receiver etc. for recognition and locating the target. All the sensors and modules work in unison to detect and follow the target.

The capability of a robot to track and follow a moving object can be used for several purposes.

- To help humans.
- To create ease for people.
- Can be used for defense purpose.

In this paper, we presented a method of a human following robot based on tag identification and detection by using a LIDAR (Light Detection and Ranging) [3-5].

II. Objectives

The scope of the research was to design a “Follow me” robot based on the LIDAR.

In the research, we focused on the following aspects:

- Making the mechanical part;
- Digital control of motors;
- The LIDAR Data Acquisition and Processing algorithm;
- “Follow me” navigation algorithm;
- The graphical interface to view LIDAR data

III. Mechanical part and hardware modules

The robot chassis has been chosen so that it can provide high stability at fast turnarounds, but also to be strong enough to support all the necessary components.

The robot chassis has the following features:

TABLE I. GRIPPER SPECIFICATIONS

Length	290 mm
Width	330 mm
Height	120 mm
Ground clearance	40 mm
Wheel Width	70 mm

The robot has been built so it can operate autonomously without power from fixed points and control from the control panel. As a power source, we used a Team Corally X-Treme Pro 90C Li-Po, 7.4 V and 7000 mAh, connected to the H-bridge to supply the Raspberry Pi plate and the four brushless DC motors (Fig. 1).

We used two extra platforms to raise the RPLIDAR A2 sensor to an optimal level, so neither the wheels nor any other components block the scanning field (Fig. 2).

C. MOLDER is with the Center of Excellence in Robotics and Autonomous Systems, Laboratory of Robotics, Military Technical Academy, 050141, Bucharest, Romania (e-mail: cristian.molder@mta.ro).

A. ȚIGĂU is with the Military Equipment and Technologies Research Agency, 077025, Clinceni, Ilfov, Romania (e-mail: tigau.andreii@gmail.com).

D. TOMA is with the Armaments Department, 061418, Bucharest, Romania.

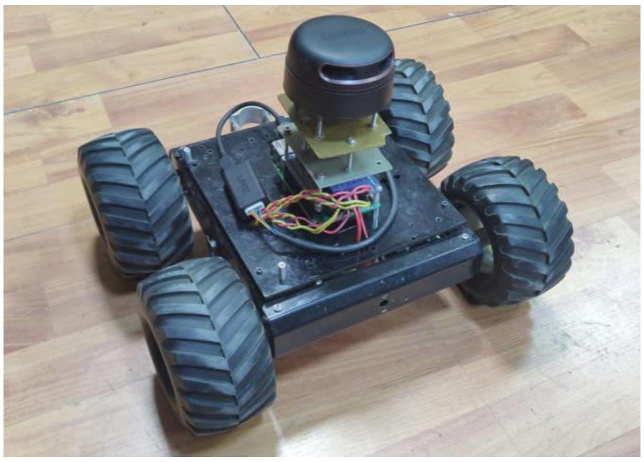


Figure 1. The autonomously robot

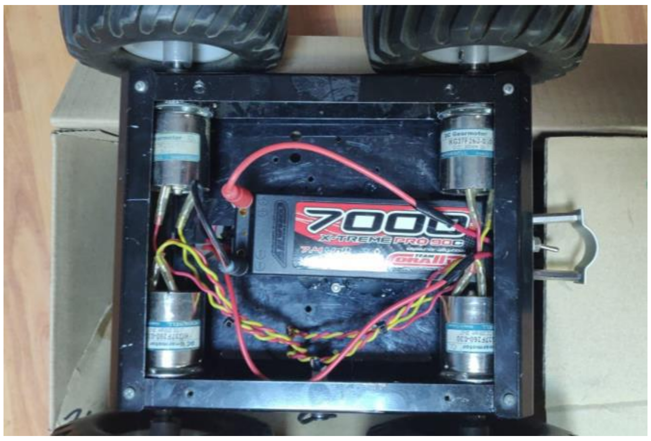


Figure 2. Power supply block

To control the movement of the robot, we used the MC 33886 Raspberry Pi Motor Driver bridge, but the commands are generated by the Raspberry Pi 3 Model B via the GPIO pins. In order to use the 40 available pins, the RPi.GPIO must be imported. In order to send commands to the H bridge, the selected pins must be set as output pins. Motor speed control is achieved through PWM signals. Before starting the DC motors, the desired motion direction must be set. Setting the motion direction is done by setting the pins connected to the motors, one at the logic value 1 and the other at the logic value 0 [8].

IV. Command and Control Algorithm

A. The LIDAR Data Acquisition and Processing algorithm

The acquisition of the data taken from the RPLIDAR A2 sensor and its processing is done through Raspberry Pi 3 Model B. The two components are connected via the SLAMTEC serial-USB adapter. These operations can also be done with a laptop, but I chose Raspberry Pi due to its small size, which makes it easy to fit the chassis used to create the robot [11,12].

In order to communicate between Raspberry Pi and the RPLIDAR sensor scanner, it is importing the RPLidar class from the rplidar library [13].

An advantage offered by the RPLidar class, besides the data collection, is the get_health () method that allows viewing of the sensor's operating state. When the control system detects a potential risk that could cause a hardware failure, it returns the "Warning" value. This value only warns for further verification of the user but does not stop the sensor. It will still work normally. When the sensor enters the "Protection Stop" mode, it will return the "Error" value. In case of a warning or error, an error code with a value other than zero will also be returned.

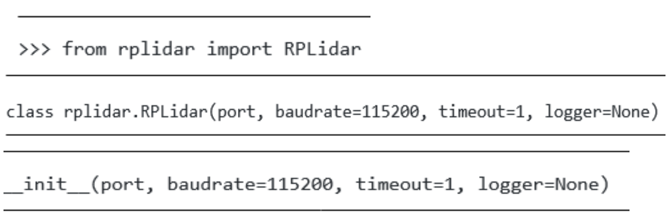

```
>>> from rplidar import RPLidar
```
```
class rplidar.RPLidar(port, baudrate=115200, timeout=1, logger=None)
```
```
__init__(port, baudrate=115200, timeout=1, logger=None)
```

Figure 3. Importing RPLidar class

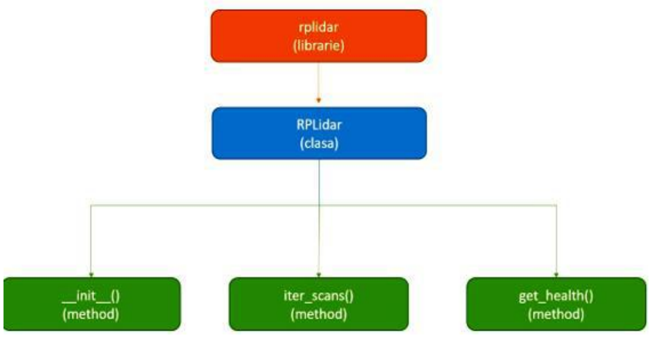


Figure 4. Rplidar library

The main method in the RPLidar class, used to extract data from the sensor to be processed, is the 'iter_scans ()' method. This allows iterations of scans. However, it should be noted that the control and control unit must have a high processing speed, otherwise the data will accumulate in the buffer, and the unit will receive the data late [15].

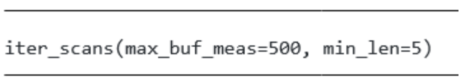

```
iter_scans(max_buf_meas=500, min_len=5)
```

Figure 5. Importing iter_scan method

Parameters:
- max_buf_meas: int (the maximum number of measurements that can be stored in the buffer. Once this number exceeds the limit, the buffer is emptied.)
- min_len: int (the minimum number of measurements per scan)

Yields:
- scan: list

List of measurements. Measurements are "tuple" type:
- quality: int (the intensity of the reflected laser impulse)
- angle: float (measuring angle of measurement, expressed in degrees: [0, 360])
- distance: float (the distance measured to the object from the center of rotation of the sensor. It is expressed in millimeters (mm))

In order to better understand what is the result of the 'yields' command, you need to understand what a generator is. But before the generator, it is necessary to understand what an iterator is [14].

When a list is created, its data can be read one by one. This process is called iteration.

```
>>> listamea = [1, 2, 3]
>>> for i in listamea:
...     print(i)
1
2
3
```

```
>>> listamea = [x*x for x in range(3)]
>>> for i in listamea:
...     print(i)
0
1
4
```

Figure 6. Iterator model

Whenever it uses the ‘for ... in ...’ function, it is an iterator: lists, strings, files. The iterator is very useful because it can be used as many times as needed, but all of these values are stored in memory, which is not very convenient if a large number of values are needed.

The generator is an iterator but can only iterate once. This is because it does not retain all values in memory but generates them on the spot.

```
>>> generatorulmeu = (x*x for x in range(3))
>>> for i in generatorulmeu:
...     print(i)
0
1
4
```

Figure 7. Generator model

It is the same as for the iterator, with the exception of the round brackets (), instead of the straight brackets. However, the 'for i in generatorulmeu' function cannot be used for the second time since the generator can only be used once: the value 0 is calculated, after which it is forgotten and the value 1 is calculated, ending with the calculation of value 4.

Yield is a keyword that is used as the return function, except that it returns a generator.

```
>>> def createGenerator():
...     listamea = range(3)
...     for i in listamea:
...         yield i*i
...
>>> generatorulmeu = createGenerator() # creeaza un generator
>>> print(generatorulmeu) # generatorulmeu este un obiect
<generator object createGenerator at 0xb7555c34>
>>> for i in generatorulmeu:
...     print(i)
0
1
4
```

Figure 8. Yield function

In order to better understand the ‘yield’ command, it should be understood that when the ‘createGenerator ()’ function is called, the code inside the function is not executed. The function returns only a generic object.

The code will be executed each time the 'for' function uses the generator. The first time that the 'for' function calls the generator created by the 'createGenerator ()' function, the code lines inside the function will be executed from the beginning to the 'yield' command, then return the first value of the loop. Then each call will re-execute the code lines in the function line, return the next value until there is no return value. The generator is considered empty when the function is called, but the 'yield' command is no longer executed. This can happen either because the loop has reached the end or because an 'if / else' condition has not been satisfied [16].

B. The ‘Follow Me’ Algorithm

To implement the “Follow me” navigation algorithm, we used the closest point to the robot so that the robot follows the target and maintains a safe distance from it in order to avoid a possible collision.

To be able to receive data from the RPLIDAR A2 sensor, it is necessary to define the port to which the USB adapter is connected:

```
>>> PORT_NAME = '/dev/ttyUSB0'
>>> lidar = RPLidar(PORT_NAME)
```

Figure 9. USB adapter connection

'ttyUSB0' is not the physical USB port number, but is the order in which the devices are detected. 'ttyUSB' is the 'USB serial port adapter', while '0' is the device number. 'ttyUSB0' is the first device detected, and 'ttyUSB1' is second.

As I said in the previous subchapter, I used the 'iter_scans ()' method in the RPLidar class to accomplish this algorithm. This returns a list that contains three values like ‘tuple’. Tuple is a group of orderly values that cannot be changed.

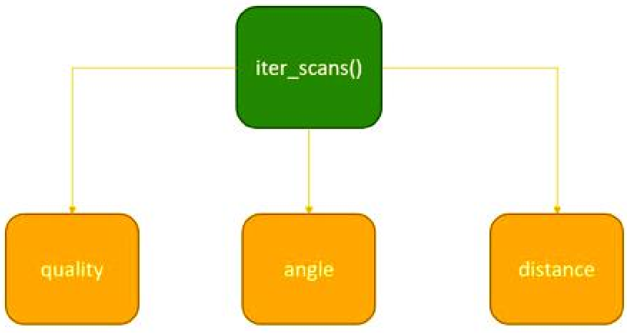
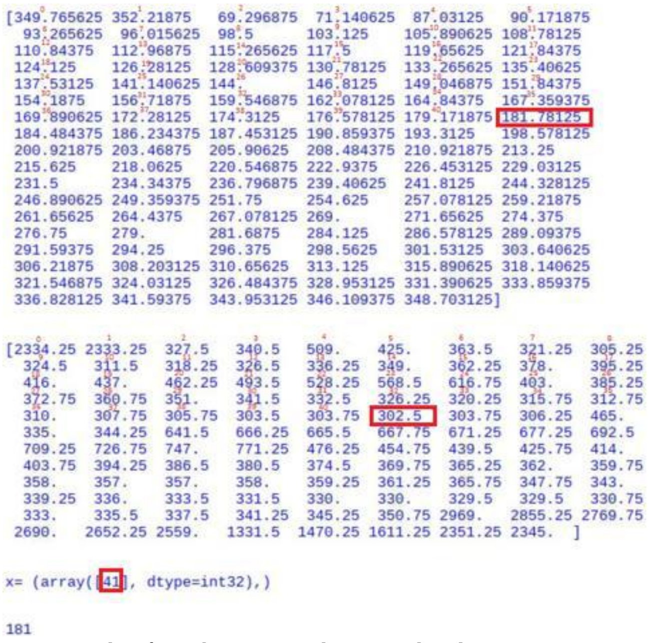


Figure 10. iter_scans() method

In order to process the data returned by the iter_scans () method, it is necessary to browse the data with the 'for' function. Because scan is a generator, we've converted it into an 'array' to make it easier to process. The next step is to determine the closest point to the scanner center of the sensor scanner. At each sensor scan, check the array that contains the object distance data from the robot and determine the lowest value using the ‘min ()’ function:



```
[349.765625 352.21875   69.296875  71.140625  87.03125   90.171875
  93.265625  96.015625  98.5      103.125    105.890625 108.78125
 110.84375  112.96875  115.265625 117.5      119.65625  121.84375
 124.125    126.28125  128.609375 130.78125  133.265625 135.40625
 137.53125  141.140625 144.      146.8125   149.046875 151.84375
 154.1875   156.71875  159.546875 162.078125 164.84375  167.359375
 169.890625 172.28125  174.3125   176.578125 179.171875 181.78125
 184.484375 186.234375 187.453125 190.859375 193.3125   198.578125
 200.921875 203.46875  205.90625  208.484375 210.921875 213.25
 215.625    218.0625   220.546875 222.9375   226.453125 229.03125
 231.5      234.34375  236.796875 239.40625  241.8125   244.328125
 246.890625 249.359375 251.75     254.625    257.078125 259.21875
 261.65625  264.4375   267.078125 269.       271.65625  274.375
 276.75     279.       281.6875   284.125    286.578125 289.09375
 291.59375  294.25     296.375    298.5625   301.53125  303.640625
 306.21875  308.203125 310.65625  313.125    315.890625 318.140625
 321.546875 324.03125  326.484375 328.953125 331.390625 333.859375
 336.828125 341.59375  343.953125 346.109375 348.703125]

[2334.25 2333.25  327.5    340.5    509.     425.     363.5    321.25   305.25
  324.5    311.5    318.25   326.5    336.25   349.     362.25   378.     395.25
  416.     437.     462.25   493.5    528.25   568.5    616.75   403.     385.25
  372.75   360.75   351.     341.5    332.5    326.25   320.25   315.75   312.75
  310.     307.75   305.75   303.5    303.75   302.5    303.75   306.25   465.
  335.     344.25   641.5    666.25   665.5    667.75   671.25   677.25   692.5
  709.25   726.75   747.     771.25   476.25   454.75   439.5    425.75   414.
  403.75   394.25   386.5    380.5    374.5    369.75   365.25   362.     359.75
  358.     357.     357.     358.     359.25   361.25   365.75   347.75   343.
  339.25   336.     333.5    331.5    330.     330.     329.5    329.5    330.75
  333.     335.5    337.5    341.25   345.25   350.75  2969.    2855.25 2769.75
 2690.    2652.25 2559.    1331.5   1470.25 1611.25 2351.25 2345.   ]

x= (array([41], dtype=int32),)

181
```

Figure 11. Identifying the minimum distance and angle

Knowing the distance to which the closest object is, a problem is that of finding the robot's orientation toward this object. Fortunately, the two arrays, 'array_unghi' and 'array_distanta', respectively, have the same dimensions, and each value in 'array_unghi' has a correspondence in 'array_distanta', representing the distance to the point measured in the direction of that angle.

As a result, the lowest value of 'array_distanta' must first be located. With this value, check 'array_unghi' to extract the value at that position, that is, the angle at which the nearest object is from the robot's 0° angle.

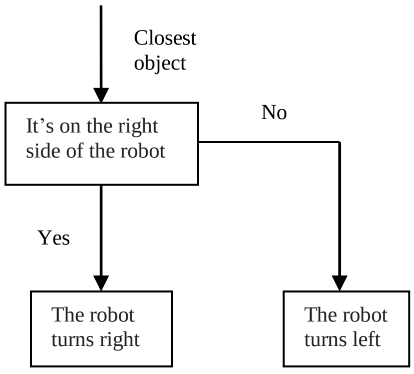


Figure 12. The condition for detecting the closest point

The first tests for target tracking were performed according to Fig. 12. These require the robot to be held in the forward direction (0° direction) facing the closest object. But only with this condition, the robot is in a continuous rotation, to the left or to the right. From the console verification of the value of the required angle is the closest object, we have noticed that only with this condition the detection of the object is within ± 18° from the 0° direction. We have introduced a condition so that the DC motors of the robot stop when the target is within ± 20° of the 0° direction so that the robot does not spin anymore. Subsequently, in the same condition, to begin tracking the target, I replaced the engine shutdown command with the start of both engines in the forward direction.

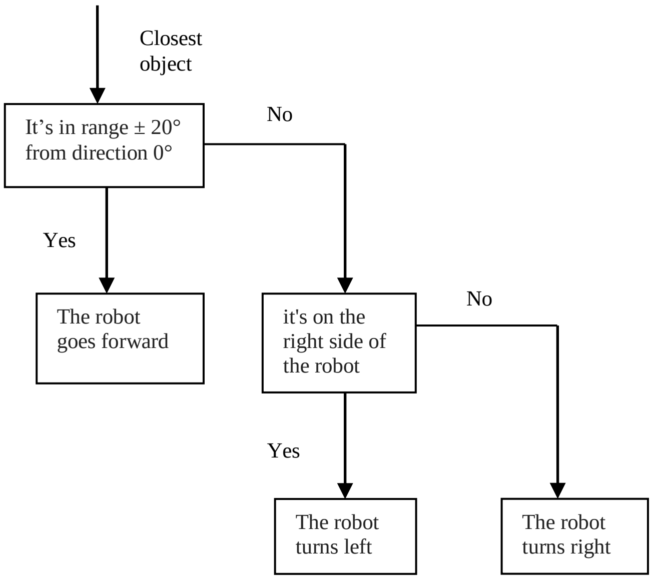


Figure 13. The condition for target tracking

After performing new testing, the robot shows the first signs of tracking the target, but we found a problem. If the target stops, the robot is still advancing because the closest object is still in the range of ± 20° to the 0° direction. To remedy this error, we have introduced a new condition so that if the target is in front of the robot at a distance of less than 450 mm from the center of rotation of the sensor, that is, about 200 mm from the robot, it controls the motor shutdown, thus stopping the track until the target changes its position.

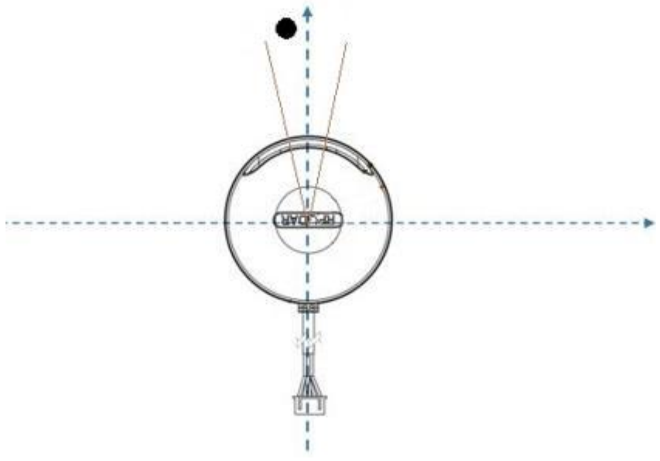


Figure 14. Detecting an object with LIDAR

I noticed that if the robot follows the target and it approaches a wall less than 450 mm, the robot will face the wall facing the wall and stop at a distance of 450 mm. If a new target approaches the robot, at a distance of less than 450 mm, this will become the closest object, and the robot will immediately turn to face it. But when the target exceeds the 450 mm threshold, the robot will no longer track it, because the wall will once again become the closest object. Solving this limitation has been a real challenge, and a great deal has been needed to find a solution.

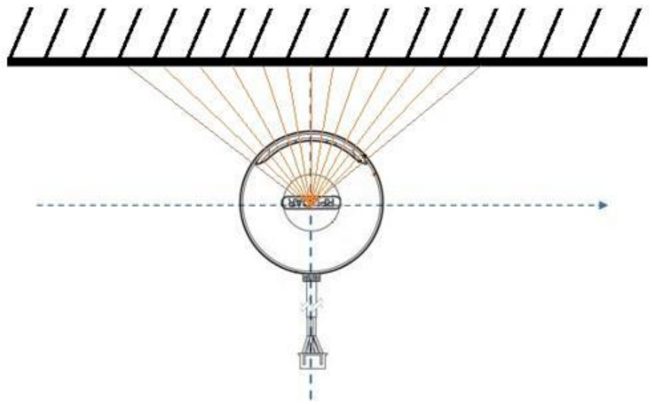


Figure 15. Detecting a wall with LIDAR

After the console view of all the sensor-generated data, along numerous tests, I noticed in the distant array that when the robot faces the wall, the values of the distances corresponding to the angles in the ± 45° range direction of 0° are proportional. While in the case of a human target, for example, in the same angular range, the values of the distances vary greatly.

We started by calculating the distance corresponding to the angle of 45° if there is a wall in front of the robot and the smallest distance is 450 mm and is in the direction of '0' It is known that the cosine value, in the range of 0 – 45°, decreases with the increase of the angle, we can say that all the distances, corresponding to the angles in the same range, have the values between 450–636 mm, if the robot is facing to a wall at a distance of 450 mm. The distance variation will not be greater than 186 mm. But if a human target is in front of the robot, the variation in the range between 0 – 45° will be much larger than 186 mm.

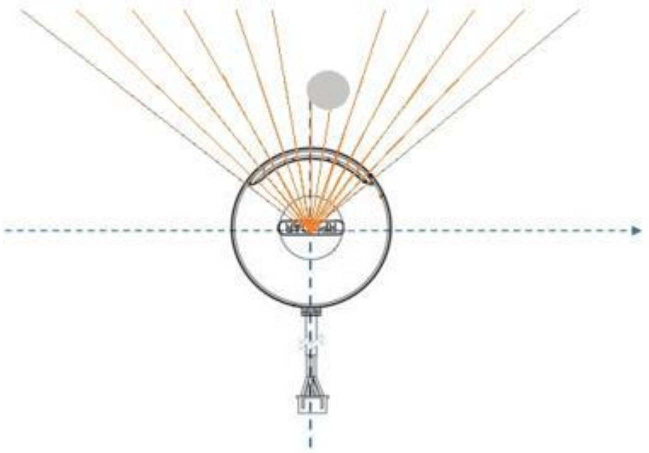


Figure 16. Detecting a human with LIDAR

Knowing the angle at which the closest point is, we build a new array containing only the values of the distances in the ± 45 ° range from the angle of the nearest object. For building the new array, 'array_unghi' must be performed to check each value if it is within the 'umghi_dist_min ± 45°' range. When a value is found within this range, it retains the position it occupies in the array, after which it checks the value that is at the same position in 'array_distanta'. This value is added to the new array – ‘array_cos'.

The first required condition for the robot to track the target when it is near a wall is that the distances in 'array_cos' do not belong to the range '[dist_min, the distance corresponding to the 45 ° angle]'. But this condition is correct only if all values in 'array_cos' do not belong to this range, the target is in front of the robot, and there are no other objects around it. This limitation can be resolved by checking that only a certain percentage of the array_cos value does not belong to that range.

To make the algorithm more efficient, we have allocated a 200 mm window in addition to the new 400 mm threshold (distance from an object on which the robot commander stops the engines) in which the robot will detect the new target and start tracking it. For reasons of safety, we have introduced a condition that when an object approaches a robot at a distance of less than 270 mm, it commands the engine starting in the backward direction until it reaches the 400 mm safety distance again.

A more intuitive way to check the data from the sensor is the graphical mode. Graphical representation I made with the ‘matplotlib’ library, specifically the ‘pyplot’ and ‘animation’ classes.

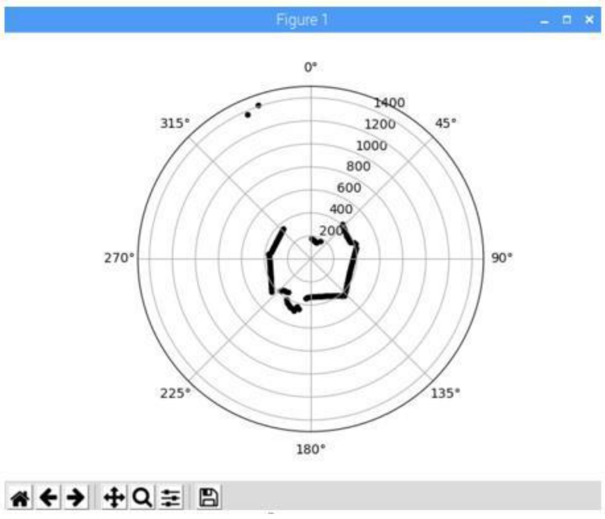


Figure 12. Graphic interface

V. The Limits of Robot Device

A. Data processing speed

The problem is the processing power of the Raspberry Pi 3 Model B, which does not handle the large amount of data transmitted by RPLIDAR A2. The sensor has a buffer that holds the data until it is retrieved and processed by Raspberry Pi, but since it is not picked up fast enough, the number of data retained will exceed the buffer limit, and it will have to be emptied, which will result in loss of data coming from the sensor at this time.

B. Algorithm problems

A disadvantage of the navigation algorithm is that the robot may lose its original target if it is removed and a new target is approaching a smaller distance. As a result, the robot will begin tracking the two targets. Therefore, this system can only be used under certain conditions, and use in rough terrain may result in errors in the navigation algorithm.

VI. Conclusion and Future Challenges

In this project we have studied and implemented an autonomous robot capable of tracking a target using a control and control unit a Raspberry Pi plaque. In order to achieve the tracking algorithm, it was necessary to study the functioning of the RPLIDAR A2 sensor and the Raspberry Pi interface, as well as how to interconnect them.

One of the issues that required extra attention was the graphical display of data generated by the sensor, simultaneously with the execution of the navigation algorithm. Due to the execution of the Python programming language sequences, each sequence could only be executed after the other. To solve this problem, it was necessary to study and then use the threading concept, which allows multiple operations to be performed at the same time. But this advantage comes at the cost of a much larger amount of resources.

In addition, to improve the tracking capability by the robot of a human target, a platform must be constructed so that the sensor is at a height high enough to detect the upper body. This avoids the uneven movement of the robot that occurs when it follows each step of the human target. On a large scale, the robot can be used to map buildings by detecting walls and surrounding objects, while memorizing their position.

References

[1] M. S. Hassan, M. W. Khan, A. F. Khan, "Design and Development of Human Following Robot", *Student Research Paper Conference*, vol. 2, no. 15, July 2015. doi: 10.13140/RG.2.2.10421.73440

[2] Jianzhao Cai, Takafumi Matsumaru, "Human Detecting and Following Mobile Robot Using a Laser Range Sensor," South China University of Technology. doi: 10.20965/jrm.2014.p0718

[3] "Introduction to Autonomous Robots", v1.7, October 6, 2016, Nikolaus Correll, Magellan Scientific

[4] M.S.A. Mahmud, M.S. Zainal Abidin, Z. Mohamed, "Greenhouse Environment Mapping Using a Low-Cost Lidar via Robot Operating System," in *Proc. 7th Int. Graduate Conf. of Engineering, Science and Humanities*, Johor Bahru, Malaysia, 13 -15 August 2018, pp. 28-30

[5] J. Reinhardt, J. Schmidtler, M. Körber, K. Bengler, "Follow Me! How Robots Guide Humans". doi: 10.13140/RG.2.2.29661.61924

[6] T. Krishnamohan, A. Mahendran, A. Selvakumar, V. Paramananthasivam, "Human following Trolley - Auto Walker," Computer System and Network Engineering, Department of Information System Engineering, Faculty of Computing, SLIIT, Malabe, Sri Lanka

[7] Mark Tee Kit Tsun, Bee Theng Lau, Hudyjaya Siswoyo Jo, "An Improved Indoor Robot Human-Following Navigation Model Using Depth Camera, Active IR Marker and Proximity Sensors Fusion," MDPI. doi: 0.3390/robotics7010004

[8] M. A. Hussein, A. S. Ali, F. A. Elmisery, and R. Mostafa, "Motion Control of Robot by using Kinect Sensor," Department of Electronics Technology, Faculty of Industrial Education, Beni-Suef University. doi 10.19026/rjaset.8.1111

[9] H. Febbo, "Autonomous Vehicle Control Documentation"

[10] A. T. Coroiu, O. Hinton, "A Platform for Indoor Localisation, Mapping, and Data Collection using an Autonomous Vehicle"

[11] O. Cameron, "An Introduction to LIDAR: The Key Self-Driving Car Sensor". doi: org/10.7249/j.ctt5hhwgz.

[12] "RPLidar Documentation" - Artyom Pavlov

[13] "RPLIDAR A 1 Low Cost 360 Degree Laser Range Scanner Introduction and Datasheet"

[14] "RPLIDAR A2 Low Cost 360 Degree Laser Range Scanner Introduction and Datasheet"

[15] "RPLIDAR A3 Low Cost 360 Degree Laser Range Scanner Introduction and Datasheet"

[16] https://hackaday.com/2016/01/22/how-to-use-lidar-with-the-raspberry-pi/

[17] https://gist.github.com/platdrag/

[18] https://www.wired.com/story/guide-self-driving-cars/

[19] https://www.superdroidrobots.com/shop/custom.aspx/sensor-support/

[20] https://www.waveshare.com/wiki/RPi_Motor_Driver_Board

[21] https://12vactuators.com/blog/the-three-types-of-dc-motors/

[22] https://rogershobbycenter.com/lipoguide