Waiter robot based on a line-following robot platform with computer vision

Bruno Campello Tôrres de Azevedo Teles
Departamento de Eletrônica e Sistemas
Universidade Federal de Pernambuco
Recife, Brasil
bruno.teles@ufpe.br

Andrea Maria Nogueira Cavalcanti Ribeiro
Departamento de Eletrônica e Sistemas
Universidade Federal de Pernambuco
Recife, Brasil
andrea.marianogueira@ufpe.br

Aline Victória Cavalcanti Pereira
Departamento de Eletrônica e Sistemas
Universidade Federal de Pernambuco
Recife, Brasil
aline.cavalcantipereira@ufpe.br

João Marcelo Teixeira
Departamento de Eletrônica e Sistemas
Universidade Federal de Pernambuco
Recife, Brasil
joao.teixe@ufpe.br

***Abstract*—Following global automation trends, cargo transport has also been modernized, creating autonomous solutions capable of delivering various types of volumes to points of interest. The restaurant sector has shown increasing interest in applying such technology, specifically with waiter robots. These robots can deliver orders to tables quickly and efficiently. The demand for waiter robots has surged due to the Covid-19 pandemic, which heightened interest in any technology that reduces human contact. In this work, a waiter robot proof of concept is presented using a line-following robot platform capable of finding the shortest possible path between two points using graph theory. Path information is obtained through computer vision, and data transmission between the computer and microcontroller is performed via Bluetooth. The developed solution met the expected objectives both in terms of navigation and in the computer vision component.**

***Index Terms*—Waiter robot, Pololu 3pi, Line Follower, Computer Vision**

I. Introduction

The first robot waiters were already used in 1983 at the *Two Panda Deli* restaurant in Pasadena, California, USA [1]. Nicknamed *Tanbo R-1* and *Tanbo R-2*, the robots served food to customers while telling jokes and playing music. However, *Tanbo R-1* and *Tanbo R-2* did not perform their tasks consistently, dropping food or walking in circles when police radios were nearby, or slurring words as if drunk when their batteries were low. Additionally, the costs were high for the amount of *US$ 20,000.00* each and they weighed approximately 36 kg. It was an initial implementation of such technology which still proves to be challenged.

Nowadays the exponential growth of technologies allows more complete, cheaper, lighter, and more robust implementations. Also the sudden lack of staff and the need to reduce human contact due to the Covid-19 pandemic restaurants around the globe began employing robots to serve their customers [2]–[4].

According to Grace Dickinson [5], other decisive factors include: robots do not get tired, being able to perform their function with the same efficiency from the first to the last moment of the day. Also, waiter salaries are constantly increasing over time meanwhile the cost of a robotic solution is becoming more affordable. Furthermore, the robots still act as attractions for restaurants, attracting curious customers to the business.

This paper propose a robot that would move through a defined white space with pre-defined black pathways. Each table had its own indication signal made using a piece of paper. One side painted red and other green to signalise whether the robot is being called. The computer vision observes all tables and sets a path when the indication signal was green. The recognition was through image processing and then sent to the robot indicating that the table was calling it. Communication between the robot and the processing software was enabled using *Bluetooth*. The information was transmitted so the robot would go to the table and then return to a starting point defined as the kitchen [6].

II. Proposed Solution

The paper proposed a prototype of a robot waiter system capable of obtaining a graph representing the paths to the tables through computer vision. The data should be transfer to the robot. Then, the shortest path to the destination table must be defined. The processing information is done using software in *Python* programming language. The code used can be found in the *Github* repository created for the project [1].

To develop this project a line-following robot platform was used. The Pololu 3pi robot is a complete commercial solution for applications involving line-following robots.Its details will be discussed below [7].

The robot does not have a way to communicate with other equipment except through its ISP connector. Therefore, a *Bluetooth* module was attached to establish the connection between the Pololu 3pi and the software running on the computer.

[1]https://github.com/brunoctt/Projeto-Micro-PDI

A. Graph acquisition through computer vision

The graph is obtained using the *OpenCV* library [8]. This library is adaptable to various programming languages and contains several image processing techniques. Due to the chosen programming language was *Python*, the library for this language was used.

The process can be divided into two parts:

- Finding the lines that make up the paths from the input image;
- Processing the obtained lines and get the graph points.

The line detection procedure starts by obtain an image of the surface with the paths. The Figure 1.A illustrated an example using an input photo.

A binary threshold is then applied to the grayscale image. This way there are only two possible values depending on the original value and the threshold value. Therefore any unnecessary information is removed and the process is simplifying. Figure 1.B shows this simplification. Although the new image is easier to analyze, there are still unwanted elements.

After the binary image is acquired, the OpenCV function findContours is applied. This function returns the contour of all continuous elements. This is demonstrated in red in Figure 1.C.

However, this image generated has a format that can create duplicate parallel lines. Also, the returned data make analysis difficult. Thus, it is not very interesting for this project. To solve this problem only the largest contour is considered. In such manner the contour is expanded so that the parallel lines become a thicker single line. The new processed image is visible Figure 1.D. This contour image will serve as a mask to be used later.

The Probabilistic Hough Transform [9] is also applied to the binary image. This will find the line segments represented as (x_i, y_i, x_f, y_f). The x_i and y_i are coordinates of the starting point, and the x_f and y_f coordinates of the endpoint of the segment. The image obtained after the Probabilistic Hough Transform is seen in Figure 1.E.

To acquire the graph the data format should be a list of segments. However, the Probabilistic Hough Transform creates difference between the lines. Therefore, a logical *AND* filter is applied. This filter will combine the image obtained after the largest contours consideration (Figure 1.D) and the image acquired by the Probabilistic Hough Transform (Figure 1.E).

Finally, the Probabilistic Hough Transform is reapplied on the image. The result image is simpler and shows fewer than 100 lines. Each segments represent some part of the paths. Nevertheless, there are still more lines than desired. Thus, an algorithm[2] to identify similar lines and merge them is used. This strategy is based on the angle and distance between the lines. After this function is defined a line for each segment of the path is obtained. The final processed image is illustrated in Figure 1.F. Only nine lines obtained for the example in Figure 1.A after this process.

[2]How to merge lines after HoughLinesP?

The following step is to find the intersections of the obtained lines. The algorithm in Figure 2 is applied. In addition to returning the intersection points, this algorithm converts all the lines into *Line* objects from the *Sympy* library [10]. This will be useful for the next steps .

After the acquisition of the intersection points, it is possible to discover all the destination points, or tables. An endpoint will never be an intersection. Therefore, it is only necessary to check if a line contains two intersections or not.

If a line only contains one intersection point, the other point will be a destination. At the same time the points are found, their direction are stored. The graph works using cardinal points as a reference. Thus, above, below, to the left, or to the right are represented by North, South, West, East, respectively.

All remaining points are connected as well. That establish a coordinate between points connected by a line. The number of points connected to a node is then observed to separate those that have been called auxiliary nodes from auxiliary points.

An auxiliary node is connected to more than two other nodes. Therefore, it is relevant to the graph acquisition, as it will be a decision factor. An auxiliary point is nothing more than a point that connects two lines without other bifurcations, not being important for routing decisions.

Figure 3 shows all the points found from the image. The final result of the graph acquired. Nodes 0 to 7 are destination nodes (tables), nodes 8 to 11 are auxiliary nodes (connected to more than two other points), and nodes 12 to 15 are auxiliary points (connected to two other points).

The numbers assigned to the nodes follow only one rule: the destination nodes are numbered from 0 to $n-1$, where n is the number of destination nodes. The auxiliary nodes are numbered n to $n+m-1$ with m representing the number of auxiliary nodes. The auxiliary points are numbered $n+m$ to x for x equal to the total number of points found.

Finally, to transfer the data to the robot waiter, the graph is transformed into an adjacency matrix. This contains the relationship between the nodes in cardinal points in the format readable by the robot.

B. Data transfer to the robot

The data transfer to the robot is the last step. A serial connection is established via *Bluetooth* using the *pySerial* library [11].

After establishing the connection, the data is transfer to the robot. Due to the robot waiter programming, the adjacency matrix information is transferred one by one. The data are stored in internal adjacency matrix. This way, it is possible to operate independently of the software.

The transmission time of the coordinates varies according to the number of connections to be registered. For the test case of Figure 1.A, twenty-two nodes information were registered in 34 seconds. Figure 4 shows part of the registration flow. For example, “0 9 W” indicates that the path from node 0 to node 9 is W, *West*. Automatically it is registered that the opposite path, from 9 to 0, is the opposite of W, E (East).

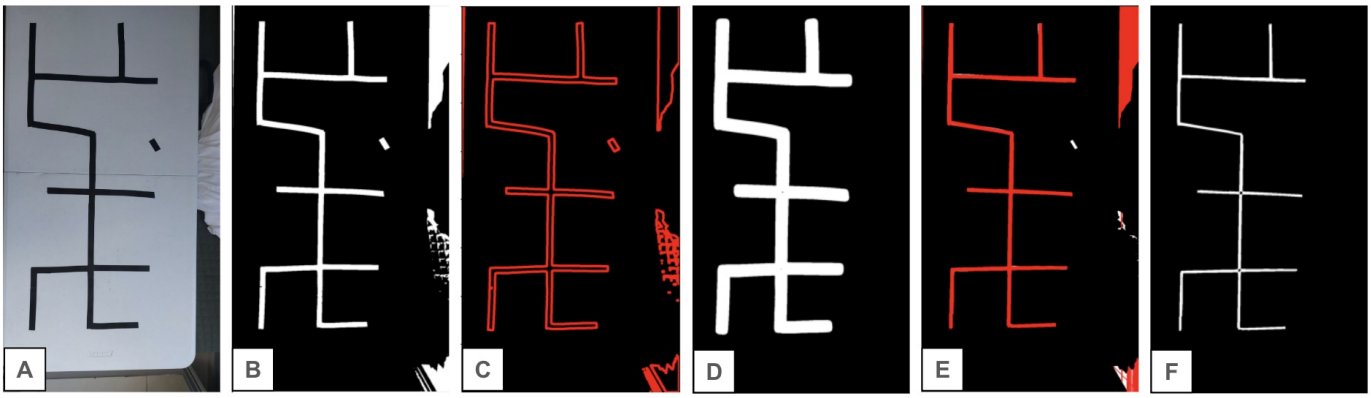

Fig. 1. A) Input photo. B) After binary threshold. C) After function findContours. D) After the removal of duplicate parallel lines. E) Result of the Probabilistic Hough Transform. F) Processed image.

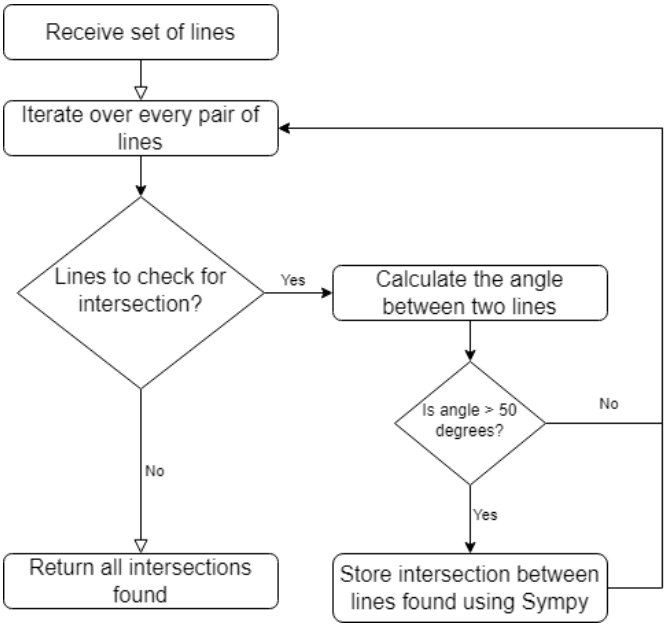

Fig. 2. Algorithm used to find intersection points between the provided lines.

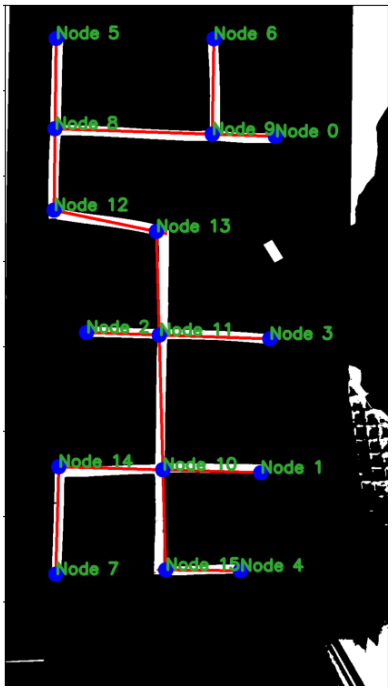

Fig. 3. All points found from the initial image.

Subsequently, it is asked if another node information is desired to be inserted. As long as the software identifies that there are new information to be registered, it will send "y" for *Yes* and register the next it. The Figure 5 illustrated the IDE for this procedure.

After registering all the nodes information, the robot returns the list of all the connections made. Figure 5 shows the IDE terminal for the example. The nodes are indicated in a line. The path to go from the first to the second.

Finally, the robot's position is informed. After this procedure, the code enters a loop. It will request the desired destination point and send it to the robot, wait for it to arrive and then asking again. If the user wants to end the program, just type "q" for *Quit*, and the program will exit the loop and finish. To obtain the shortest path search, it is performed a modified BFS (Breadth-first search) algorithm.

III. Simulation and Results

The following tests were conducted:

A. Graph creation from image Test

The graph creation algorithm was executed multiple times. Different parameters for the minimum angle and minimum distance between lines were test to ensure there was no instability in the obtained result.

B. Graph transmission to the robot Test

Following the graph creation test, it was analyzes if the serial connection was successfully established and if all relevant information was transmitted.

```
b'Enter node relation (node1 node2 direction12):\r\n'
0 9 W
b'0 9 W\r\n'
b'Add another node relation? (y/n)\r\n'
y
b'Enter node relation (node1 node2 direction12):\r\n'
1 10 W
b'1 10 W\r\n'
b'Add another node relation? (y/n)\r\n'
y
b'Enter node relation (node1 node2 direction12):\r\n'
2 11 E
b'2 11 E\r\n'
b'Add another node relation? (y/n)\r\n'
y
b'Enter node relation (node1 node2 direction12):\r\n'
```

Fig. 4. Part of the activities registered in the *IDE* (Integrated Development Environment) terminal where the responses are automatically sent by the program.

```
b'Add another node relation? (y/n)\r\n'
n
b'Connections list:\r\n'
b'0-9\r\n'
b'W\r\n'
b'1-10\r\n'
b'W\r\n'
b'2-11\r\n'
b'E\r\n'
b'3-11\r\n'
b'W\r\n'
b'4-10\r\n'
b'WN\r\n'
b'5-8\r\n'
b'S\r\n'
b'6-9\r\n'
b'S\r\n'
b'7-10\r\n'
b'NE\r\n'
b'8-5\r\n'
b'N\r\n'
b'8-9\r\n'
b'E\r\n'
b'8-11\r\n'
b'SES\r\n'
b'9-0\r\n'
```

Fig. 5. Part of the activities registered in the *IDE* terminal when the Pololu 3pi returns the established connections.

C. Path printing between nodes Test

The adjacency matrix was registered and the function that returns the shortest path for all nodes in the matrix was used to see if the returned result was as expected.

D. Robot navigation between nodes Test

Similar to the return test, the robot was instructed to go to all possible points. Many line-following robot projects have difficulty in some areas where the robot does not navigate correctly, losing the line or rotating on its axis. After it was verified if it arrived at the points correctly.

E. Fine-tuning of robot movement Test

Observed during the navigation tests, this test aimed to verify if the robot corrected its position even with slight changes in direction.

IV. Conclusion

This paper presents a prototype of a waiter robot that would receive navigation information from software using computer vision. The robot should find the shortest path from its current location to the destination point.

The results obtained shown that the main objective of this work was achieved. The developed system performed all the proposed activities without notable failures in its behaviors. A demonstration video of the robot's operation was recorded and can be found in [12].

As discussed in the section III, it was not possible to exhaustively test the system, so its stability for any case cannot be guaranteed.

Additionally, other activities can be mapped as future work, including:

- Testing the project with more cases/configurations: Different path configurations can result in unexpected behaviors. During the *Bluetooth* tests, there were occasional instances where the connection was not successfully established. A very important test case is in a "real" but controlled environment;
- Optimizing codes in Python and Arduino: There were some redundancies throughout the codes, especially on the *firmware* side. It would be interesting to optimize memory usage, as the current solution has a maximum number of nodes that can exist;
- The robot's movement while calibrating its sensors can sometimes cause it to be slightly displaced from the line;
- Adding an ultrasonic sensor (such as the HC-SR04 model) to avoid collisions.

References

[1] G. W. Records, "First restaurant with robot waiting staff." [Online; accessed 7-Jan-2023].

[2] UOL, "Florida restaurant turns to robots due to a shortage of waiters during the pandemic." [Online; accessed 12-Jul-2024].

[3] O. Digital, "Restaurant in the netherlands uses robot waiter." [Online; accessed 12-Jul-2024].

[4] Yahoo, "Restaurants in malaysia employ robots as waiters during the pandemic." [Online; accessed 12-Jul-2024].

[5] B. of House, "How do robotic waiters work and are they right for your restaurant?." [Online; accessed 7-Jan-2023].

[6] B. Teles, "Apresentação v0 robô garçom." [Online; accessed 12-Jul-2024].

[7] Pololu, "3pi robot." [Online; accessed 13-Jul-2024].

[8] G. Bradski, "The opencv library.," *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, vol. 25, no. 11, pp. 120–123, 2000.

[9] J. Illingworth and J. Kittler, "A survey of the hough transform," *Computer vision, graphics, and image processing*, vol. 44, no. 1, pp. 87–116, 1988.

[10] Sympy, "Sympy's documentation." [Online; accessed 13-Jul-2024].

[11] Pyserial, "Pyserial documentation." [Online; accessed 13-Jul-2024].

[12] B. Teles, "Apresentação robô garçom tcc." [Online; accessed 08-Jan-2023].