Cooperative real life maze search with Arduino Robots

Valentijn van de Kamp, Niels Nijhof
The Hague University
Zoetermeer, Nederland
{V.vandeKamp, N.R.Nijhof}@student.hhs.nl

***Abstract* – In this paper we discuss a cooperative real life maze search using two Arduino robots. This study is part of a larger project.**
During this project, two Arduino robots were built that are able to communicate with each other using the Nordic nRF24L01+ radio module. The robots use analog line tracking sensors to navigate through the maze. Using these sensors, the robots are able to identify patterns and decide where they are located within the maze. During the mapping of the maze, the robots communicate with each other about their findings.

***Index Terms*— Autonomous agents, Arduino, real life cooperative maze search**

I. Introduction

This paper describes the creation of cooperative maze solving autonomous agents. There are already many ways to solve a maze using line tracking agents. This has been done with Arduino and other microprocessors.[9][10]

In this paper, we present a foundation for a cooperative real life maze search using two Arduino agents. This basis is formed by making fundamental decisions and building the Arduino agents that will be able to; follow lines, make choices and communicate with other agents. The questions answered in this study to make fundamental decisions are; what is the best wireless communication technique within a maze, what is the best way to construct the maze, how does the agent navigate through the maze, how do the two agents communicate, how do the agents recognize where they are within the maze relative to the starting position?

Key problems that are solved during this project involve; communication protocols, decision making algorithms, shortest path algorithms, tile recognition and proper line tracking using the limited available resources.

Chapter II describes the background of this project. In Chapter III we explain the methods used to overcome the mentioned challenges. Chapter IV describes how the maze is built. Chapter V explains how the tiles within the maze are recognized by the Arduino. Chapter VI describes the wireless communication used by the agents. Chapter VII explains the use of the right preferential algorithm and how the model is maintained within the Arduino. Chapter VIII concludes our study. Chapter IX discusses future work.

II. Background

This project is offered and facilitated by the Experience Technology lab at The Hague University. The ET lab combines innovation with practical challenges and research. This study about cooperatively solving a maze with autonomous Arduino agents is part of a larger project. The end goal of the project is to; let two agents explore a maze, send each other what they encounter and build an internal representation based on that information.

III. Method

This project contains three key aspects; The wireless communication between two Arduinos, the construction of the maze and the agents themselves.

The wireless communication technique used by the agents is the result of desk research about the different kinds of techniques compatible with Arduino microprocessors. The results of this research are discussed in chapter VI.

We had to decide how to build a changeable, low cost and durable maze. The materials used to create the maze have been selected after consulting an expert from Gamma, a company with large hardware stores in The Netherlands. Chapter IV describes the materials used to construct the maze.

After the maze was made, the development of the agents began. The agent has to recognize tiles, follow a line, and communicate with the other agent. By experimenting with different ways to reach these goals, we have selected methods that fit our needs and demands. The results are described in chapter V and VI.

IV. Maze

The maze consists of different types of tiles. These tiles are made out of Medium Density Fibreboard (MDF) with a white coating and a thickness of three millimeters. Each tile is roughly 30x30 centimeters. The size of the tiles has been selected after roughly measuring the turn circle of the agents. The lines are made out of black isolation tape with a width of 15 millimeters. The width of the tape is roughly the same width as tape used before this project in the ET lab, which was working well with the same line tracker sensors.

A. *Tile types and orientation*

The maze consists out of five types of tiles; corner, t-junction, intersection, dead end, and the finish. Each tile can only be

one of these types. For example, a tile can't both have a corner and a t-junction.

Each tile can have a different orientation within the maze with exception of the intersection. Figure 1 displays the different types and the possible orientations of the tiles.

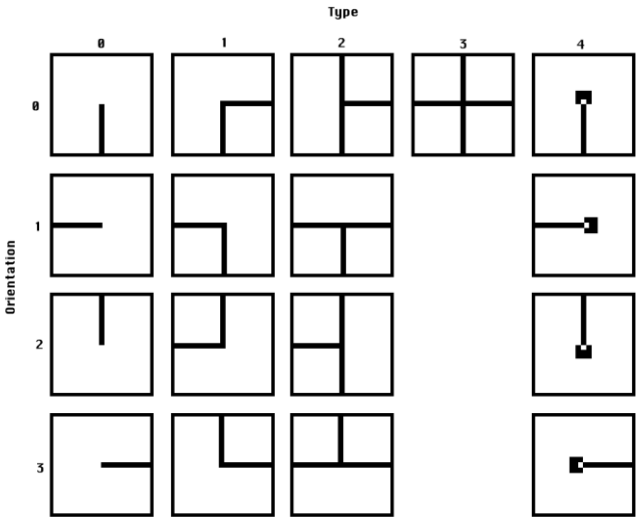

Figure 1: Tile types and orientations

V. Tile Recognition

Analog line tracking sensors were used to enable the agent to recognize and follow a line. Using three sensors on the front of the agent, it is also able to recognize the different types of tiles used within the maze.

A. Arduino Robots

The robots are built using Arduino Uno microprocessors[1]. Each robot is propelled by two DC geared motors[2], which in turn are controlled by the 2A Motor Shield[3]. To enable communication between the Arduinos, both are fitted with a transceiving radio module described in chapter VI. Three analog line tracking sensors[4] are placed under the robot and close to the surface, to track the lines of the maze. Figure 2 below displays a complete robot.

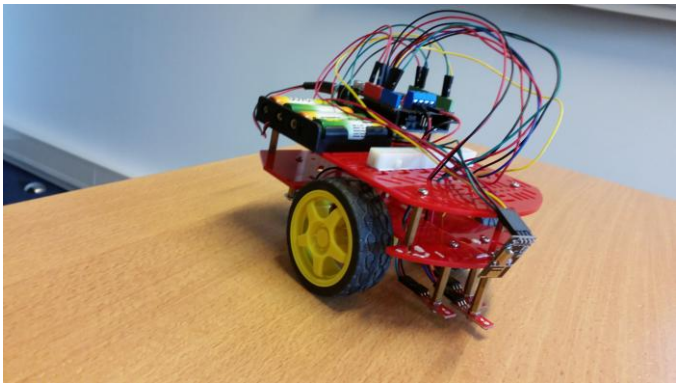

Figure 2: Complete Arduino robot

B. Analog line tracker calibration

Analog line trackers detect the difference between white and black, but they also have a broad gray scale. The line-tracking sensor always returns a value between 0 and 1024, 0 being the whitest and 1024 being the blackest. Because not all line-tracking sensors are precisely the same due to the distance from the tiles and manufacturing differences we cannot set a specific value as white and a specific value for black. We solved this problem by keeping track of the whitest white the line tracker has detected. Then we convert that whitest value and the blackest value (1024) to a percentile. Where 0% is the whitest that sensor has detected and 100% the blackest it can detect. Everything higher or equal to 60% is considered black and everything lower than 60% is considered white. These percentages have been selected after experimenting with different values. When a higher percentage was used, the robot would not be able to follow the line or correctly detect intersections. When the percentage is lowered, the robots aren't able to follow the line. This occurs because it reads gray values from the white surface and thinks it is black. Figure 3 gives a visual representation of the white and black percentage rates.

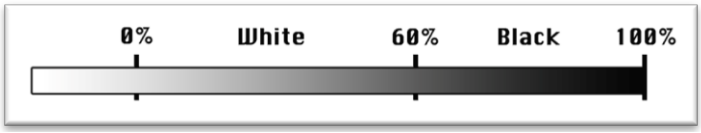

Figure 3: Percentages white and black values

C. Line tracking

The robots need to be able to follow a line. Each robot has three line trackers that can be used to let the robot follow black lines. Figure 4 shows the positioning of the line tracker sensors on a black line. As long as the center line tracker (2) is black it means that the robot is on top of a line. To prevent the robot from losing the line we also use the first and third line tracking sensors. These sensors are positioned just off the sides of the black line. Because both line tracking sensors return a percentage we can detect which side is closer to the line. The line-tracker with a higher black value is closer to the line. Using these values from the sensors, the robot can adjust its course and stay on track.

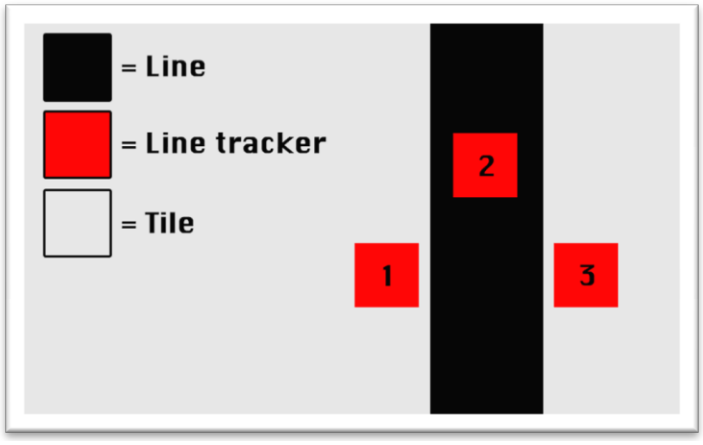

Figure 4: Positioning of the sensors

D. Tile type recognition

The robot has to be able to recognize the different types of tiles it might encounter inside the maze. We have developed a system to accomplish this, using the three analog line tracker sensors.

As described in chapter IV, the maze is built using different types of tiles, for example a corner. When the robot encounters the different tile types, the sensors gives different black and white values. In the case of the corner displayed in Figure 5, the first two sensors are detecting white, and the third sensor is detecting black. This combination won't occur on, for example, an intersection where all sensors would detect black.

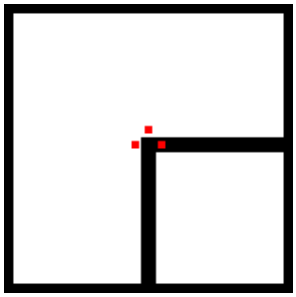

Figure 5: Corner detection

This convention has been used on every tile, since all tiles and orientations give a different combination of white and black readings on the sensors. There is however, one special case: the finish tile. The finish tile is considered special because it had to be designed in a way that it would not conflict with other detection patterns. The regular detection patterns are already in use by the other tile types. For example, we can't say the robot is finished when each sensor detects black, because that would mean the robot is standing on an intersection. The same goes for all white, it would stand on a dead end.

We developed a new pattern to solve this issue. When the sensors detect all white, the robot will move forward for 150 milliseconds. If the sensors still detect white, the robot is positioned on a dead end tile. If the robot detects all black, it is positioned on the finish tile. Figure 6 illustrates the two steps to determine if the robot has finished.

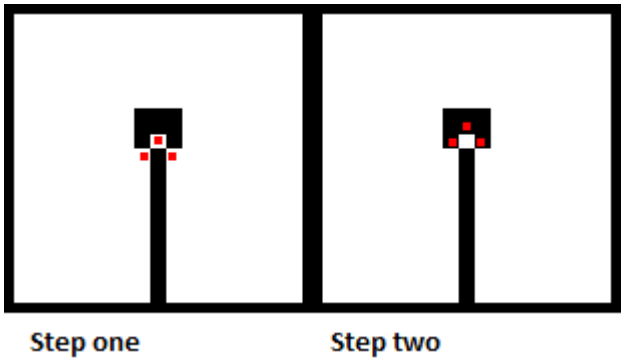

Figure 6: Steps to determine if the robots has finished

VI. Wireless communication

For the communication between the robots, a wireless communication technique has been chosen. There are many options for the Arduino boards like BlueTooth, WiFi, Radio, xBee and more. We have decided to use the Nordic nRF24L01+ radio module. This module is easy to use and fits our requirements.

A. Wireless communication requirements

We have determined four requirements for the wireless communication technique that will be used by the Arduino robots.

1) Data range

The data range has to cover a distance of at least 10 meters. The size of our maze will not exceed the size of a room, yet the possibility must be given for further expansion in the future.

2) Data transfer rate

During transmission, a payload of 12 bytes is sent between the Arduinos. The package contains six integers. Each integer has a size of two bytes. We believe a transfer rate of 250kb/s, the lowest possible transfer rate for most modules[5], will be sufficient.

3) Power usage

The robots are powered by four AA type batteries, from different manufacturers. From our experience, the batteries drain rather fast. To ensure that the robots will be able to finish the maze search without changing batteries, the power usage of the module must be as low as possible.

4) Cost

Since budgets are limited, the costs of the wireless communication modules must be as low as possible.

B. Wireless communication techniques

We have reviewed several techniques that are commonly used by Arduino microprocessors[5][6]. The table below displays the range, transfer rate, power usage and cost of the researched techniques. Modules from each technique have been used to determine the aspects, such as power usage, of each wireless communication technique[5][7].

Technique	Range in meters	Data transfer rate	power usage in milliampere(mA)	Cost in euro (iPrototype.nl)
WiFi	Depends on router	54+ mb/s	240 @ 3.3v	8 to 17
BlueTooth	10 - 100	2mb/s	30 - <500	32 to 35
Xbee/Zigbee	100	250kb/s	50 @ 3.3v	26,95 excl shield
Synapse	1200 - 2400	250kb/s - 2mb/s	130-193 @3.3v	-
General RF	15 - 1500	4,6 - 350kb/s	8 - 30 @ 3v	4,59(transmitter) + 5,25(receiver) excl antenna
Nordic RF	10 - 100	1-2 mb/s	13 - 24	6,95 incl antenna
Bluetooth Low Energy/ANT	-	-	-	-

Figure 7: Results reviewed communication techniques

As seen in the results above, WiFi and Synapse use a large amount of power. The BlueTooth and Zigbee modules are expensive, and the general RF modules come as a transmitter and a receiver, commonly without an antenna. We were unable to find enough information about BlueTooth low Energy/ANT.

The nRF24L01+ module fulfills each requirement. It has a range of more than 10 meters, a transfer rate of 250kb/s,

1mb/s or 2mb/s, between 13 and 24 mA power usage and is cheap in comparison with the other techniques [5][7].

C. *Communication protocol*

To establish communication between the two Arduino robots, a protocol had to be developed. Each robot transmits the tile it is standing on to the other robot. The other robot will save the information in its own memory, which can be used to make a movement decision within the maze.

The data package contains important information for the other robot. The robot transmits; the type of tile it is standing on, the orientation of the tile, the x coordinate and y coordinate. The coordinates are relative to the starting position. The second robot will know where and how the tile is placed within the maze. During this project, the robots always begin on the same tile within the maze. This means that the x and y coordinates of the tiles that are being sent are still valid for the receiving robot.

During the transmission of data packages, it can occur that a package will get lost. To solve this, Auto Acknowledgement has been used from the RF24 library[8]. Using this function, the transmitting robot will wait for an Auto Acknowledgement from the receiving robot. If the receiving robot does not transmit an Auto Acknowledgement, the transmitting robot will resend the package that had been lost until it receives an Auto Acknowledgement.

We have noticed that communication between the robots has a large impact on their behavior. When the robots are communicating while driving, the communication between the agents interferes with line tracking. To solve this problem, the robots wait for a reply from the other robot when they have detected a tile within the maze. During this waiting period, the robots try to send and receive information to and from the other robot. When the information has been sent, and confirmed with Auto Acknowledgement, the robots continue.

VII. RIGHT PREFERENTIAL ALGORITHM

A. *Tile library*

Arduino sketches are not object-oriented. To make life easier, a C++ library has been developed to store the tile model. Using this library, the robot can create an array of tile objects when navigating through the maze. The tile object contains all the necessary information the robot needs, such as; tile type, tile orientation, x coordinate, y coordinate, and the neighbors of the tile. To give a better insight into the library, a snippet of the header file is shown in Figure 8

```
class Tile
{
  public:
  Tile(int, int, int, int);
  void setNorth(Tile*);
  void setEast(Tile*);
  void setSouth(Tile*);
  void setWest(Tile*);

  Tile getNorth();
  Tile getEast();
  Tile getSouth();
  Tile getWest();

  int getOrientation();
  void setOrientation(int);

  int getType();
  void setType(int);

  int getXCoordinate();
  void setXCoordinate(int);

  int getYCoordinate();
  void setYCoordinate(int);

  bool getOptionNorth();
  bool getOptionSouth();
  bool getOptionEast();
  bool getOptionWest();

  void setOptionNorth(bool);
  void setOptionSouth(bool);
  void setOptionEast(bool);
  void setOptionWest(bool);
```

Figure 8: Header file Tile library

B. *Robot movement behavior*

The starting orientation of the robot is always north, no matter how it is positioned in perspective of its real life position. When the robot turns right for the first time, its orientation is east. When it turns right again, its orientation is south, and so on.

When the robot detects a tile, it will always try to go right. If right is not possible, it will try to go straight. If it can't go straight, it will turn left. Is the tile a dead end, the robot will turn around. We use this right preferential algorithm so that the robot will eventually find the finish tile.

To prevent the robot from going into a loop, we track how many times the robot has been on the same tile with the same orientation. As an example, let's say that the robot encounters an intersection and its orientation is north. The first time it encounters the intersection, the robot will go right. After some time, the robot returns to the same intersection on the same orientation. It checks if it has already been on that tile with the same orientation, if so, it chooses to go straight. The third

time, the robot goes left. When it has had all three possibilities, the robot will start over and turn right. This strategy prevents the robot from going into a loop. This information is also used by the other robot within the maze. When it encounters the same intersection, and the first robot has gone right, it will go straight and send the first robot that it has also been on that same tile. The next time one of the robots encounters the intersection on the same orientation, it will turn left

VIII. Conclusion

Using the presented tile types and orientations, the creation of a challenging maze is certainly possible. The three millimeter MDF tiles are starting to bend slight. We believe that this might cause a problem in the near future.

The concept of using three line trackers to follow black lines and detect different types of tiles based on patterns was successful during this study.

The nRF24L01+ module was the right choice for this project. Using this module, we were able to set up a successful communication protocol.

With the right preferential algorithm, the robots are able to explore the maze and eventually find the finish tile.

IX. Future Work and Discussion

The robots are very inconsistent. The DC geared motors used to propel the robots are of low quality and it seems like each motor has a slight difference in power, which interferes with the successful movement of the robots.

During this project, it was noted that the wheels of the robots often slip. For example, the robot wanted to turn right but the wheel slipped so it went straight and the internal representation of both robots became invalid because the x and y coordinates were not correct anymore.

The robots consume a lot of power. When the battery power is low, the motors start stalling. The robot would need a little push to continue. One set of four AA batteries did not last for a day. We suggest that battery packs are purchased in the future.

The three line tracker sensors are attached to the robot using screws. These screws would often come loose which would give inaccurate readings. We suggest to use a line tracker array in the future, which is easier to attach to the robot.

During the testing of the robots, they would often collide. We suggest to use the x and y coordinates to create collision detection or use sensors for this purpose.

We have created internal representations based on the same starting position. In the future, the robots should be able to start from different positions within the maze.

It might be worth looking into the Polulu 3pi robot[9]. This robot is fast, accurate and can be expanded with shields.

We were unable to create the Dijkstra algorithm that should be implemented during this project. Writing the algorithm for the Arduino caused several issues. We think, but are not sure, the memory of the Arduino has reached its maximum when executing the algorithm. The Arduino would crash and the entire process would reset. We used standard arrays to hold the tile objects for the algorithm. Because it is not possible to remove an element from the array in C++, we had to keep the elements in the array and set them to NULL. We believe this caused the memory to reach its maximum, but we are not entirely sure. In the future, it might be useful to find a library for a container that supports deletion, because we were unable to find a good alternative for arrays. It can also be useful to expand the memory of the Arduino using, for example, an SSD module.

References

[1] iPrototype.nl, Arduino UNO Rev3. From, https://iprototype.nl/products/arduino/boards/uno

[2] iProtoype.nl, DC Geared Motor. From, https://iprototype.nl/products/robotics/servo-motors/dc-geared-motor

[3] iPrototype.nl, Motor Shield 2A. From, https://iprototype.nl/products/arduino/shields/motor-2A

[4] iPrototype.nl, Line Tracking sensor – Analog. From, https://iprototype.nl/products/components/sensors/line-tracking-analog

[5] Sparkfun, Wireless Buying Guide. From, https://www.sparkfun.com/pages/wireless_guide

[6]OpenElectronics, Top 5 wireless ways to communicate with your controller(2015). From, http://www.open-electronics.org/top-5-wireless-ways-to-communicate-with-your-controller/

[7]iPrototype.nl, Communication. From, https://iprototype.nl/products/components/communications

[8]Maniacbug, Arduino driver for nRF24L01. From, https://github.com/maniacbug/RF24

[9] Polulu, Polulu 3pi Robot. From, https://www.pololu.com/product/975

[10] Instructables, Robot Maze Solver. From, http://www.instructables.com/id/Robot-Maze-Solver/