Laboratoire de Systèmes Robotiques

Thymio Road Network

Loïc Dubois

Professor:
Fransesco Mondada

Assistants:
Christophe Barraud
Stefan Witwicki

June 1, 2015

LABORATOIRE DE SYSTEMES ROBOTIQUES
CH-1015 LAUSANNE

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

SEMESTER PROJECT — SPRING 2014–2015			
Title:	Thymio Traffic Network - Local Infrastructure		
Student:	Loïc Dubois	**Section:**	Microtechnique
Professor:	Francesco Mondada		
Assistant 1:	Christophe Barraud	**Assistant 2:**	Stefan Witwicki

Project Statement

Context. Thymio is a small mobile robot developed here at EPFL for education and for fun. It has two wheels, 9 infrared sensors, an accelerometer, a microphone, speakers, and 39 brightly-colored LEDs. Imagine a community of Thymios coexisting in an urban environment. Each Thymio has a purpose ; and with that purpose, things to do and places to go. Here, our motivation is to build up a a transportation infrastructure for the Thymio society in the form of a road network.

The road network will be realized in two parts : *local* infrastructure for a Thymio's movement with the network, and *global* infrastructure, for the functioning of the network as a whole. This project is focused on the former part, though will be performed in close collaboration with other half of the project.

Core Objectives.
– Design of method the robots use to stay on the road (barcode + lane markings + controller) Barcode scanning at different speeds
– Obstacle/collision avoidance between the robots
– Design of intersections (in collaboration with the student building the *global infrastructure*). The choice of an appropriate intersection method should be made (crossroad, roundabout, traffic lights, traffic barrier, etc.). The expected advantages and disadvantages of all options should be weighed, and one option should be carefully selected.
– Control at intersections. Given a choice of path, the robot should be able to execute that choice (e.g., turning left, taking an exit, changing lanes, etc.).

Additional Objectives. The following constitute additional goals whose treatment can each significantly increase the contribution of the project *assuming* the core objectives have already been fulfilled to a reasonable degree of satisfaction.

– Implement specialized intersection management hardware (e.g., an electrical or mechanical traffic light) and sensing/control methods for Thymio to interact with that hardware.
– Add communication between robots to manage better obstacle avoidance Design special control strategies for special vehicles (e.g., street cleaner, tow truck) (could be done in collaboration with the other student).
– Create high priority vehicles (police car, ambulance, street cleaner...) and make them have the priority on regular robots (could be done in collaboration with the other student).
– Your own creative ideas (to be discussed with the project assistants).

Observations

A project schedule will be established and submitted to the assistants in charge before **the 2nd of March, 2015**.

An intermediate presentation (consisting of approximately 10 minutes of presentation and 10 minutes of questions and answers) of your work will be held **during the first weeks of the month of April**. The goal here is to give a quick overview of the work already accomplished, to propose a precise plan for the remainder of the project, and to discuss the various options thereof.

A report, including as a preamble the project statement (this document), and followed by a one-page abstract (according to the official canvas), will be submitted **on Monday, 1st of June 2015, before 12:00,** to the project assistant in 3 copies. The focus will be on the experiments and the obtained results. The target audience is an engineer (e.g., EPFL bachelor or master) without extended knowledge of the domain. A preliminary version of the report will be submitted to the assistant(s) on **the 22nd of May 2015**. All documents in digital version including the report (source and pdf), the document used for the oral presentation, an abstract in pdf format, as well as the source code of any developed programs, must be submitted to the assistant according to the instructions you will receive.

A 30-minute defense of the project (approximately 20 minutes of presentation and demonstration, and 10 minutes of questions and answers) will be held **between the 1st and 12th of June 2015**.

Professor in charge:	Assistant in charge:
Signature: [illegible]	Signature: [illegible]
Francesco Mondada	Christophe Barraud

Lausanne, **the 17th of February 2015**

EPFL – IMT - LSRO Semester Project 01.06.2015

Thymio Road Network

Loïc Dubois, MT

Assistant 1: *Christophe Barraud*

Assistant 2: *Stefan Witwicki*

Professor: *Francesco Mondada*

This project consisted in the realisation of a road network for the Thymio, an educative robot with many sensors and actuators.

Source : www.thymio.org

Figure 1. The Thymio robot

A road network consits in a finite number of locations connected by several roads. I had to first design the road. then design barcodes to collect informations from the network and a robust reading method associated with it. I also implemented two different collision avoidance method. One that slows down the speed of the robot until a given threshold and then stops the robot and the other one that directly stops the robot once a threshold is reached. Finally we deigned four different type of intersections branching, roundabout, location and crossroad. Each intersection requires a specific control sequence to fulfil the choice made by the robot.

The main issue for the line following was the loss of the road. Due to the design chosen, a gradient line, the control implemented works well on one side of the road only.I tried different solutions but none of them gave satisfying results.

After developping the method for the reading, it appeared that it was relatively robust for modest changes of speed. Moreover, I also discovered, thanks to the dynamic sequence of action adaptation, that there is a wide rang of speeds in which we can read the barcode. I also implemented a security to avoid triggering the reading when the sensor responsible for this action comes on the black side of the road.

The collision avoidance methods gave also satisfying results. At first I only implemented it for the forward motion but I extended it to the backward motion as well as when there is no line to follow and we must respect a timing. I even created a more complex method in case of the presence of an other robot on the code just before a Thymio starts reading. The coming robot will perform a backward maneuverer and then re-start to move again but at the lowest speed defined to minimize the dynamics effect on the reading.

Finally some intersections caused more problems than others. The branching and the roundabout were simple to design and implement but their control required to cross some road. Therefore the robot had to stop the line following for a while. Due to straightness problem, I enforced a light left turn with the commands sent to the robot to increase its chances to find the road back. In the crossroad, the Thymio had to cross four times a road with small distances to reach the reuired orientation. It appeared that the robot was often missing the last code and could not leave the corresponding state. I forced therefore the last continue action directly after the third one.

The calibration of the robots was mandatory but I discovered I had to balance sharp left turns without leaving the road and right turns without triggering the reading when setting the grey reference for the line following. It was not always possible to avoid both problems.

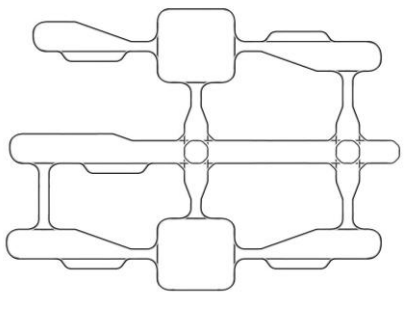


Figure 2. Final implemented map

Contents

1 Introduction

The goal of this semester project was to implement a road network for the Thymio robot. A road network is composed of locations connected with roads. Intersections allows to select among different directions and different locations. The robot then should be able to navigate without losing the road through this network while managing interactions with other users of the network. The following work is more focused on the low-level tasks related to this network.

The first part consisted of designing the basic elements of the road network such as the road design, simple intersections and locations. I chose to use a gradient line as the road, a roundabout as a first intersection and train-like branching tracks as locations. The different piece of informations of the network, such as the speed limit, the kind of intersection, the location of the exit and so on, were encoded in binary barcodes. A reading method independent of the robot's speed was developed once the size of elements composing the barcode were designed. With these first elements, we were able to create a first simple map for the robots to navigate on.

Once the network was designed I could implement the first basic behaviors. The most important of them was the line following; I implemented a feed-forward proportional controller to follow the line with the help of a ground-oriented infrared sensor to measure the position on the line. A simple collision avoidance method was also rapidly implemented as well as the control strategy for the roundabout and the location. Both of them require similar type of action: either exit the intersection (or enter the location) or continue on the road.

Then we developed more evolved behaviors and more complex intersections to challenge the robot. I improved the collision avoidance to create a smoother adaptation of the speed. And I developed also methods to avoid collisions while reading barcodes or when there is no line to follow and the timing is crucial. I also had the possibility to develop a simple backwards maneuverer for the robot to avoid collision in the case of the barcode reading. We also implemented new intersections such as a branching half way between the roundabout and the location already implemented as well as a crossroad which is managed by traffic lights Thymio.

With the new developments we created a second map more complex in terms of composing elements but also in terms of topology. The map tests created the need of a calibration setup that was especially developed for this application. The final tests allowed us to extract some statistics about the robustness of the code we developed and the design of the different elements.

2 Setup

2.1 Thymio Robot

The Thymio is an educational robot developed at EPFL. Numerous sensors and actuators, as well as a graphical and textual programming interface using the Aseba language allow anyone form child to elderly to develop many applications in various domains. Amongst these sensors we can find buttons, many proximity sensors (horizontal all around the body and ground), a 3-axis accelerometer, a temperature sensor, a microphone (only intensity detection). And as actuators we have, 2 motors (used for the motion in differential drive), many LEDs and a speaker [4]. There is also a possibility to use Lego ® to extend its possibilities of applications.

The differential drive is a way of locomotion in which two actuated wheels are fixed on the same axis and a passive castor wheel (2 degree of freedom wheel) or spherical wheel is used for stability purpose [3]. In our case, a slippy static plastic half-sphere is used instead of a third wheel. The motion can be divided in two parts: a linear and a rotational motion. The linear speed is proportional to the sum of angular speed of the two actuated wheels and the rotation speed is proportional to their difference.

In Aseba, the speed of each wheel is commanded by sending a value between -500 and 500. To facilitate the use of its value, I made some measurement to be able to convert these commands in a convenient and comprehensible speed unit. We can observe on the figure 1, the quasi linear relation between the the two variables. However we denote also a plateau corresponding to commands above 450. The tendency line on the graph does not take this plateau into account and gives us the following relation:

$$speed\ [cm/s] = 0.032 \cdot command \tag{1}$$

We have a speed of approximately 15 cm/s on the plateau.

To measure the speed, I used the calibration setup that I will present later in the calibration sectionn(section 10). The Thymio followed a line on a fixed distance and the time was measured by knowing the timing of the proximity sensor event and counting the number occurring during the trajectory. I estimated the inaccuracy to be ±1 count (i.e.: 0.1 second).

Note that these measurements were made with only one Thymio and cannot be applied to all of them; small differences will appear. Moreover, we can have also a dependency on the battery level which will impact the motor speed. However I assumed the global behavior (linear phase and then plateau) to be recurrent in all robots

2.2 Aseba Studio

To program the robot, we used the text programming interface of Aseba Studio. A visual programming interface exists as well but more for programming learning purposes. The Aseba language is an event-based language, which means that portion of the code are executed only when certain events occur. Some of them happen regularly and some of them more occasionally. All the sensors are emitting events at constant frequency but we can also create events to trigger a certain behavior under certain conditions. Some particular aspects are such that we can only use integers values for the variables and the constants and also that their number is limited [4]. The Aseba language syntax is very close from the C or the matlab one. Therefore it is easy to learn it rapidly if one of the former languages is known.

More than just a coding interface, the main window allows us to read all the different values measured by the sensors or sent to the actuators while the Thymio is connected. We can also keep an eye on all the constants defined and the events created thanks to sub-windows next to the coding window. Finally we can rapidly write code with the drag-and-drop function and the key words

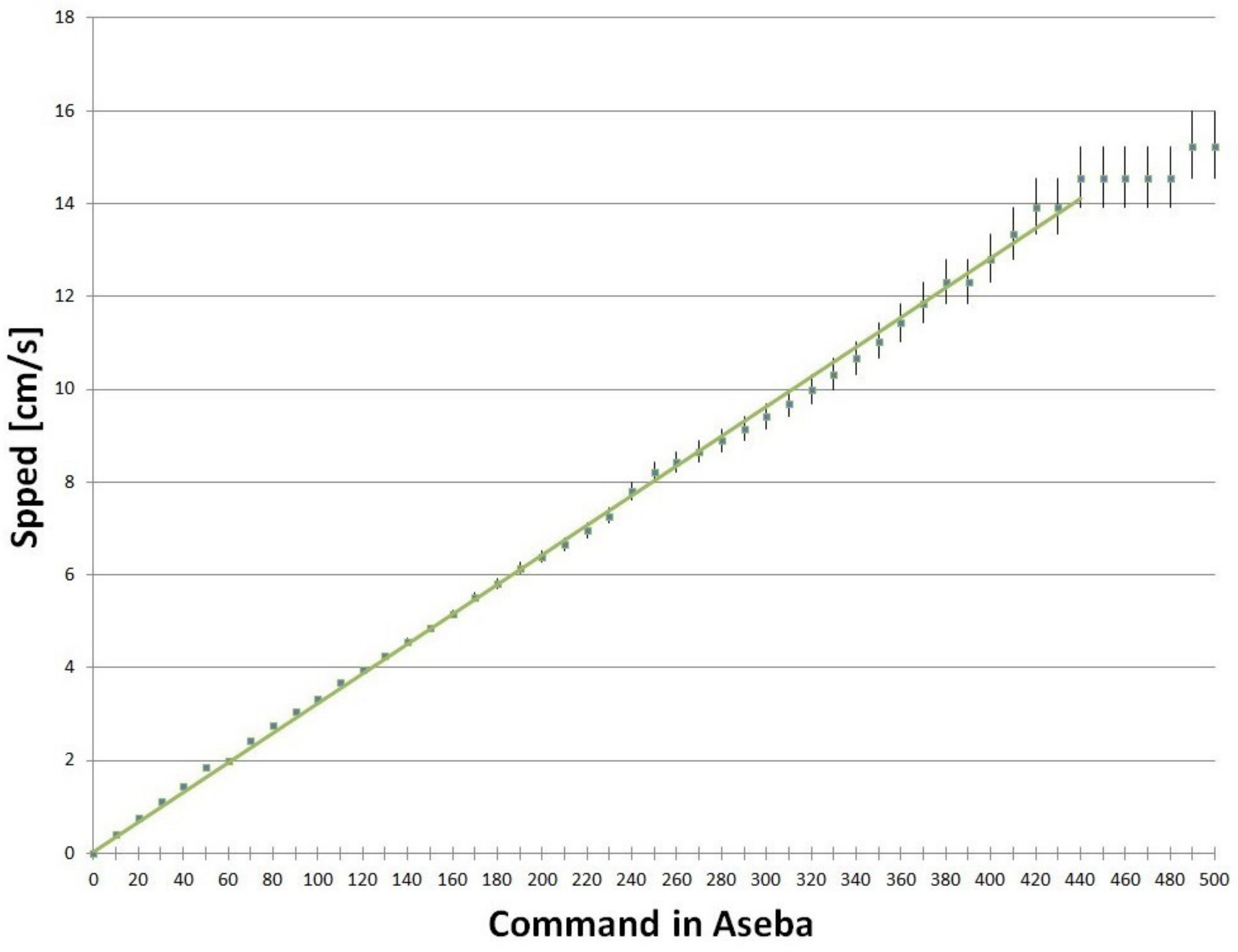


Figure 1: Speed characteristics

located in the sub-windows corresponding to all the different elements. The constant error detection is also a very useful tool to detect mistakes and correct them as early as possible.

3 Low-Level States

The program structure can be represented as a finite state machine (c.f.: figure 2). The Thymio evolves between these different states given certain conditions.

We can describe each state by the actions it implies and conditions to change to another state.

- **ROAD**: In this state, the robot performs line following, collision avoidance and triggers barcode scanning. This is the main state of the Thymio. If the code read corresponds to a different speed, the robot stays in this state. Otherwise it goes to the *ACTION* state. Finally, if a Thymio is present on the barcode when the reading is triggered, the robot goes to the *BACK* state.

- **BACK**: The state used to make the robot going backward. The robot can perform line following and collision avoidance in this mode. After a given time, it comes back to the *ROAD* state.

- **READ**: Here, the robot manage the barcode reading as explained later in the corresponding section. It maintains the line following as well. Once the reading is finished, the Thymio always comes back to its previous state.

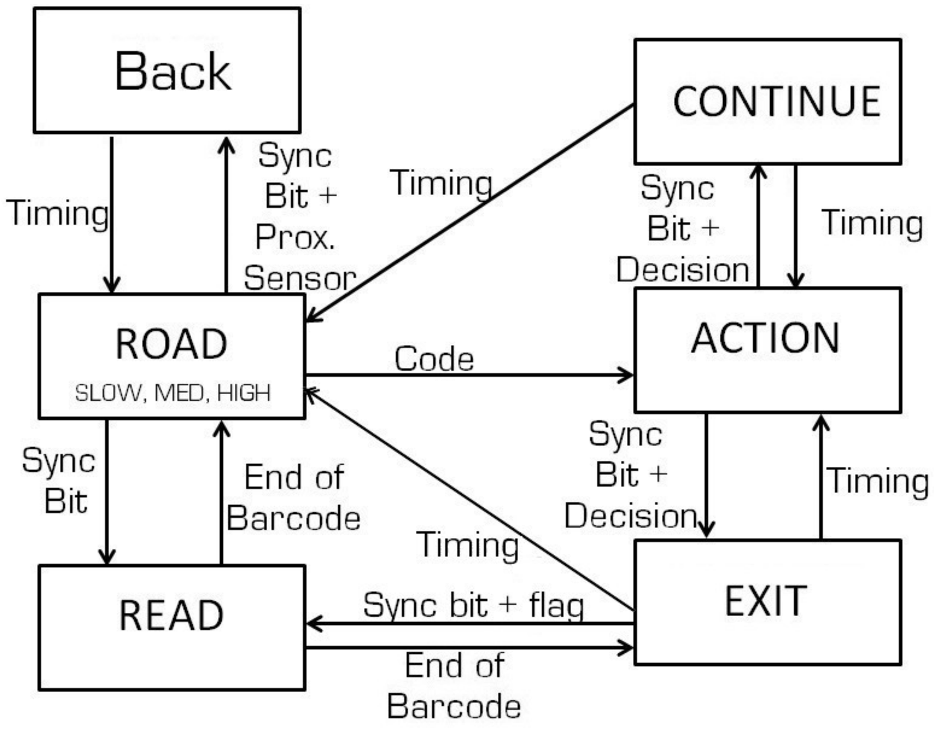


Figure 2: State machine and transitions

- **ACTION**: This states corresponds to movement within the different intersections (Roundabout, Location, Crossroad and Branching type). The robots perform a different kind of reading (except in the Location intersection where they perform also standard reading), after which we have to decide between *EXIT* or *CONTINUE*. The line following and collision avoidance is maintained
- **CONTINUE**: In this case the robot stops following the line for a given time to cross the exit. It either comes back to the *ACTION* state or to the *ROAD* state, depending on the intersection.
- **EXIT**: The robot decides to leave the intersection or the location. To do so it still performs line following and collision avoidance. And, after a given time, comes back to the *ROAD* state. If it is in a location, the Thymio can encounter a barcode and go to the *READ* state.
- **STOP**: This state is not represented in the figure 2 because it can be accessed from any state by pushing the central button or when the robot stops in Location. In this state, no actions are performed and the robot can only be set *EXIT* state (to start by reading a location barcode) using the forward button or, when it is stopped in location, after a given time.

4 Line following

The network is composed by numerous straight lines, turns and intersections of different kind. To navigate in the network, the Thymio should be able to follow these lines and search for some if it is lost. We can compare different possible solutions in the table 1 (solutions illustrated in figure 3).

I finally chose the gradient from left to right as design for my line. The line had to be wide enough to ensure a maximal number of gray level measurable and therefore a smoother control but, on the other hand, it had to be thin enough to allow the robot to be able to read the barcode with one

Table 1: Possible line designs

Method	Pros	Cons
Gradient Left to Right	One level per position on the line Smooth transition between two levels	Depending on printer If road lost, gradient on one side only
Gradient Center to Borders	Smooth transition between two levels If road lost, gradient on the two sides	Depending on the printer Do not know on which side we are
Discrete Black Levels	One level per zone	Less depending on the printer Harsh transition between two zones If road lost, gradient on one side only
White Line Between two Black lines	Simple No loss of the road	Use two sensors to stay on the road No control between the lines

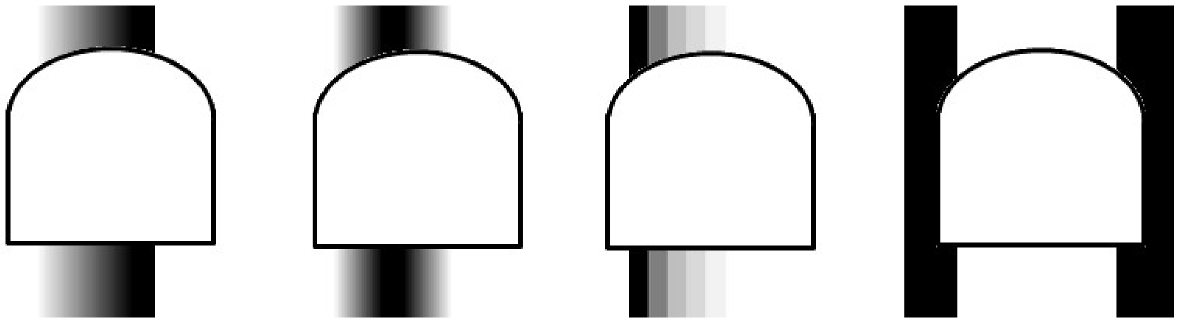


Figure 3: Different possible line designs associated with table 1

sensor while benefiting from the full range of gray levels with the other sensor. Knowing the distance between the two ground sensors to be slightly more than 3 centimeters and that the reference value to be approximately in the middle of the gradient, I chose a line width of 3 centimeters.

Another choice was to select a way of changing direction in the network. I chose to implement turns instead of sharp bifurcations. The former has the advantage to handle any change of direction in a soft way although it requires more space to perform this change. Space not being a constraint, that is why I chose this method. Moreover, with a sharp change of direction, it would become very difficult for the robot to stay on the track and to keep following the line.

For the control on this line, I took inspiration from the light painting project (c.f.: appendix C) and I implemented a proportional feed-forward controller [2]. At each iteration, the robot measures the difference between the reference and the measured gray level (a value between 0 and 1023, c.f.: figure 4). Then this difference is scaled given a gain and the commands for each wheel are sent. The commands are composed of a reference speed and an increment for a wheel and a decrement for the other one corresponding to the scaled difference of gray levels. Which wheel is accelerated and which one is decelerated depends on the sign of the intensity difference. This type of control provides a satisfactory and smooth motion for the Thymio.

Implementing a more advanced controller (like a PD or PID one) would not be useful. Indeed, we are limited by the quality of measurements of the sensor. The values measured for one fixed spot are often oscillating between in a range of ± 20 or ± 50 measurement units. Moreover working with integer values did not facilitate the work. To obtain a more precise tuning, I would need two integer coefficients (one multiplicative and one dividing) and therefore increasing the complexity and the time required for tuning. Or another solution would consist in processing the values read over several iterations (averaging, ...). The latter solution would be useful in case of straight lines to obtain a smoother motion but it would delay the effect of the controller in the beginning of turns and increase the chances to leave the road. Finally, the rounding after the division could imply a loss of information in certain cases.

The gains were determined with the help of a qualitative evaluation. A different state implies a different gain as it has different requirements. And as we are working only with integer variables, in the following paragraphs the term gain refers to a dividing factor.

For the *ROAD* state, I wanted a smooth motion on straight lines while being able to manage a sharp turn at low speed and light turns at higher speed. A gain too large (i.e.: smaller increment) is not sufficient to manage the U-turn with a continuous motion (the robot is always leaving the road). On the other hand, a gain too small (i.e: large increment) results in oscillations while moving on the straight line. The optimal trade-off was found for a gain $k = 3$.
For the *READ* state, I wanted to go as straight as possible so the reading is the least perturbed possible and this for the 3 different speeds. It appeared that for the gain chosen above, for the *ROAD* state, the robot was still oscillating a bit when driving on the road. Therefore I increased the gain until the oscillations became approximately the size of the barcode. Which corresponds to a gain $k = 6$.

Finally for the *ACTION* state, I needed the ability to manage sharp turns while coming fast to a straight orientation after a turn to be able to stop following the line and finding it back afterwards. Therefore I chose a smaller gain than the *ROAD* case, $k = 2$. As the robot goes slowly and there is no complete barcode to read, the oscillations are less a problem.

As I said, one of the disadvantages is the dependence on the printer. Sadly, due to the size of the final map, a resolution superior to 72 dpi (dot per inch) was not feasible. However, thanks to the quality of the work done at the repro, this resolution did not create any problem for the Thymio concerning the line following.

One more disadvantage appears when the Thymio loses the road. The control implemented (depending on the measurement of the sensor) is working on only one side of the road if the road is lost. Indeed if the robot end up on the left side of the road (light side of the gradient) it will behave correctly (i.e.: turn right as it measures a light gray level) and will be able to find the road again. However, if it ends up on the other side of the road, it will read a white corresponding value on the sensor and will turn right as well. In the end the robot will still find the road but it will drive on it in the wrong direction, which is a problem I had not planned to solve.

I tried different solutions to solve this problem. First I tried using a flag indicating on which side of track (light or dark side) the robot is. When the road is lost, the robot will turn depending on which side it was before losing it. First, I defined a zone in which the developed controller is active and a zone in which the road is considered lost. When the robot is in the former, it determines also on which side of the road it belongs. I had one more problem to solve: as the sensor is computing the mean value in a zone measured under it, strange values can be measured when the sensor is part on the black side of the road and part out of it (c.f.: figure 4 between the 1100^{th} and 1200^{th} pixels). To avoid misinterpretation, I added a second condition to determine the side of the road: the robot has to read two times in a row a measurement corresponding to the light side before defining it belongs to this side. In the end, it seemed to work at low speed to find the road back. But due to this zone between the dark side and the white zone next to the road, it often ended in oscillations where the robot is stuck on the border of the road.

I tried also to put a condition on the reading ground sensor (i.e.: prox.groud[1], the one that reads barcodes). When the robot loose the road on the left side, this sensor is on the road and therefore measures a gray value. On the other side, the sensor is out of the road and measures a white value. The disadvantages of this method are that it depends on the proximity sensor and that it will require calibration as all sensors read different values for the same gray level. Moreover, if the robot is completely out of the road on the left side, it will also read a white-corresponding value on

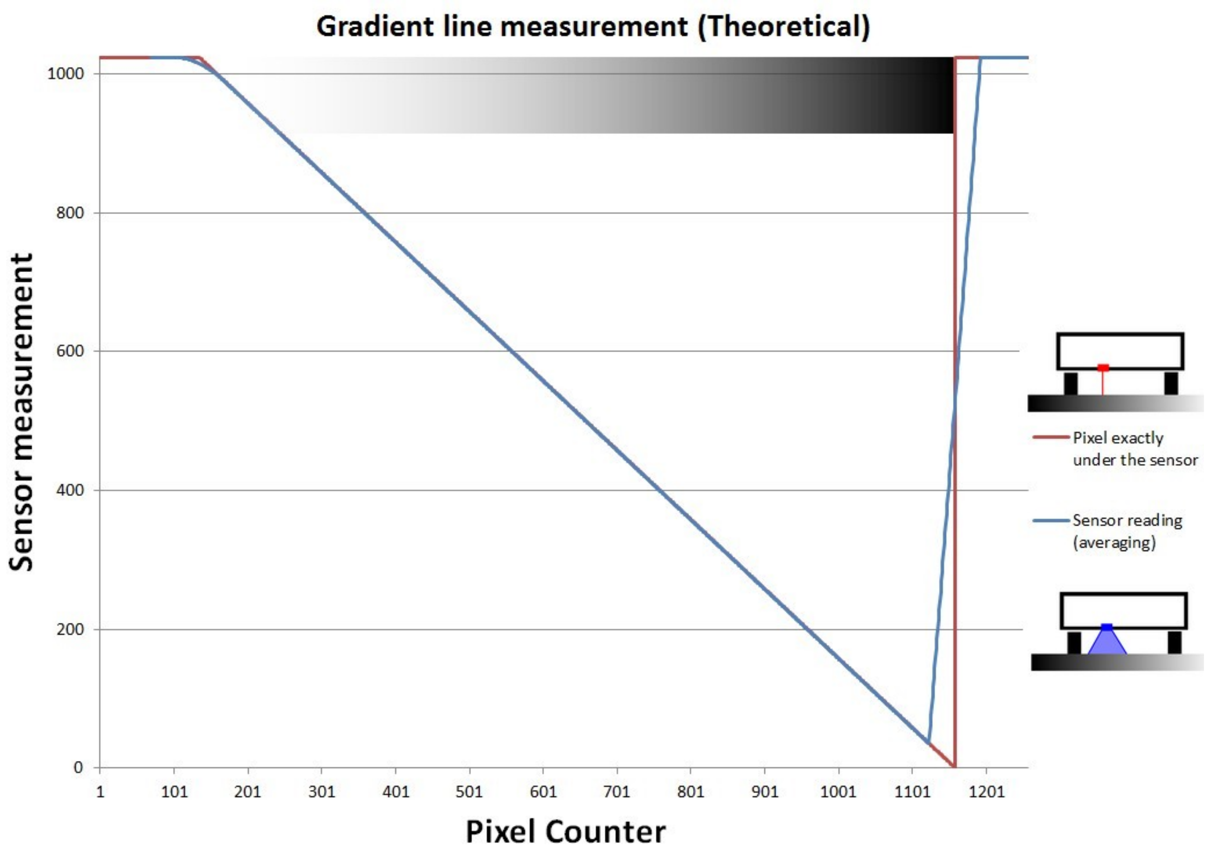


Figure 4: Theoretical example of the measurements of the ground sensor for the line shown at the top

its sensor and will act as it has left the road on the right side.

A final idea, I did not tried, will be like scanning. If the robot loses the road, it will first turn in a direction. If nothing is found after a given timing (less than the time required to do a half-turn and drive in the wrong direction of the road). The robot will inverse the speed of each wheel and start turning in the other direction. I can even ask the robot to stop completely if it did not find the road after turning in the other direction. Which means it has left the network. If I want to push the reflection further, this Thymio could send a message, using the communication protocol and if a Thymio receives it, it will come help him to find the road back.

One difficulty that appeared after the choice of line, is the lack of designing program able to create curves of the wanted size with a radial gradient. Moreover, due to our basic knowledge, designing all the different curves was very time-consuming. To avoid wasting all our time on this task, we tried to use as much as possible the curves already created. But, using transformation tools to create a new curve from one already existing, we added some imperfections. If one looks closely, he will see that our curves are not perfectly smooth but, in the end, the robot seemed to manage them without problems. Thus we decided not to spend more time on the design.

5 Barcode scanning

Mainly for space purpose, I decided to put the barcodes on the right side of the road. In this configuration, the robot uses prox.ground[0] for the line following and prox.ground[1] for the scanning. This choice had a small inconvenience: the right side of the track being the dark side of the gradient, in case of a right turn of the road, the IR sensor responsible for the scanning could detect the gra-

Table 2: Possible barcode location

Location	**Pros**	**Cons**
Right side (outside)	Space between the tracks One sensor per task	Possible to interpret black side of the road as a bit
Left side (inside)	No misinterpretation One sensor per task	Less space between the tracks
In the road	No more space required	Line following function can misinterpret the barcode Triggering the reading

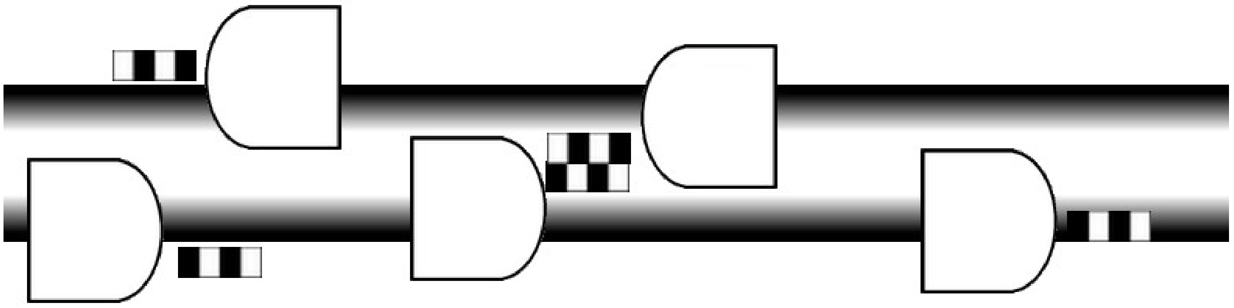


Figure 5: Different possible barcode positions corresponding to table 2

dient as a black bit. I could have resolved this by inverting the gradient direction but the map was already printed, therefore I put a security in the program: when the IR ground sensor responsible for scanning the barcode detect a black zone, the line following sensor is checked. If its value is above a threshold (i.e.: on the white side of the line instead of the middle) the reading is not triggered. This method seems to work quite well and the Thymio never missed any barcode (100% successful triggering over all the tests done) .

I needed a way to trigger the reading state of the robot, this was done by setting the first bit of every barcode to be black. This method requires the security explained above to avoid triggering at the wrong time in the wrong place. Moreover it makes us loose 1 bit to code information given a fixed barcode length but I did not find another appropriate way to trigger the reading. But, in the same time, I did not need many informations to be encoded. I had to encode the three different speeds possible and the four different intersections.

Concerning the design, as the reading rate of the sensor is fixed (10 Hz) and I wanted to set different speeds on the network, I needed to select an appropriate length for each bit. Indeed, the length is linked to the smallest common multiple of the distance that can be covered between two reading. In the table 3, below, the chosen speeds are given in term of Thymio command (a value between -500 and 500), speed and distance covered in 0.1 s.

Table 3: Distance covered in 0.1 s

	SLOW	**MEDIUM**	**HIGH**
Command	141	282	423
Speed	4.5 cm/s	9 cm/s	13.5 cm/s
Distance in 0.1 s	0.45 cm	0.9 cm	1.35 cm

The critical length is the largest one (i.e.: at high speed). Moreover to ensure a robust reading method, even at the highest speed, I doubled this length, leading to a length of 2.7 cm per bit (i.e.: allowing two readings per bit). This is also the smallest common multiple between the different traveled lengths. I chose to use 3 bits to code the information which leads to a barcode of 4 bits measuring 10.8 *cm*. This length seems reasonable as it is small enough to be placed on every straight line, even after a turn, but large enough to encode all the informations required. An example of barcode can be seen below.

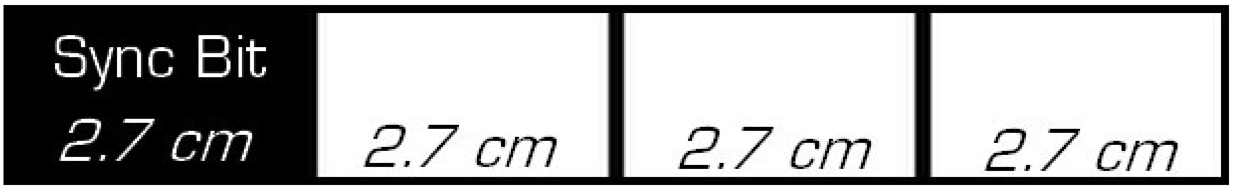


Figure 6: Barcode design with trigger/synchronization bit

The reading is performed by alternating a series of scanning and waiting actions whose number is a function of the speed of the robot. It is required for the robot to scan one and only one time each bit of information. Therefore I can decompose the bit between the two actions (c.f.: table 4). In a simple manner, I could ask the robot to wait the given the number of times then read again once it has detected the synchronization bit. But this method suffer from an inconvenience: the reading starts close from the border of the synchronization bit therefore each following reading action is close from the border between two consecutive bits and can easily generate mistakes: either read the wrong bit or misinterpret value as we saw for line following. To improve this, the robot waits 1.5 times the number of waiting actions after the trigger bit. Therefore, it is now approximately in the middle of the next bit to perform the reading. For the following bits, it keeps waiting as before. One must not forget that we are working only with integer numbers and because of this constraint, there is no difference between the two methods at high speed. The two different methods can be seen on figure 7.

Table 4: Actions required for the reading of one bit

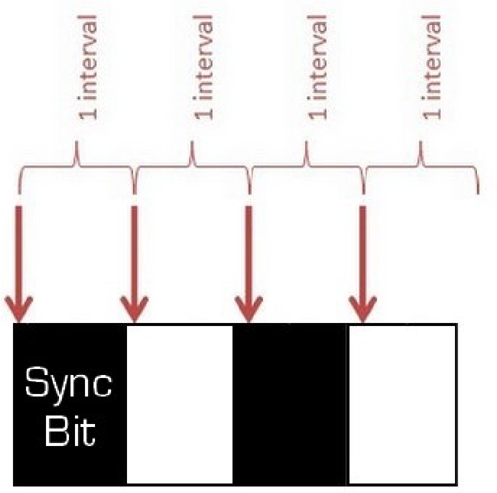
	Number of actions per bit	**Scanning**	**Waiting**
SLOW	6	1	5
MEDIUM	3	1	2
HIGH	2	1	1

To assess the robustness of my reading method, I tried to read barcodes with differences in the speed. In the code (c.f.: appendix A), the interval (number of reading and waiting actions to perform for one bit) is automatically evaluated given the speed of the robot. This explain why the robot can read barcodes even if it is not around one of the three nominal speeds. However this computation depends on the highest speed selected:

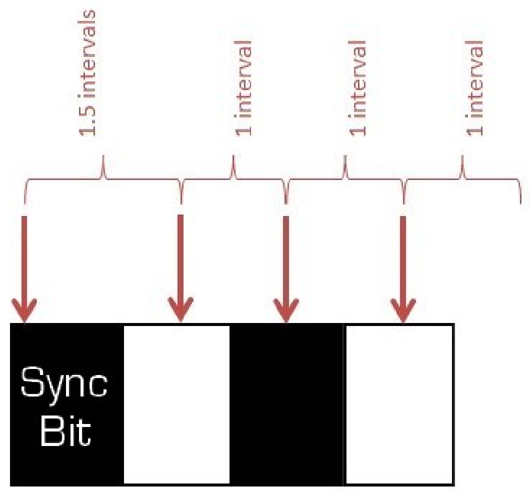
$$interval = 2\frac{Highest \quad speed \quad command}{speed \quad command} \tag{2}$$

Given the large range of working speeds, I decided to choose a value a little larger than the middle value for two main reasons: First, the robot has a range a slower speeds to manage the effect of the discharge of the battery and second, due to the calibration required (c.f.: section 10) I do not want to choose a speed so high that could result in a speed in the plateau shown in the graph of the figure 1 for another Thymio. The chosen values are shown in the table 5 below.

In addition, I computed also the theoretical value for the acceptable speeds. The minimal speed acceptable consists in reading the 4 bits and ending the reading just after the barcode. I assumed the reading to start at the very beginning of the barcode. In the equation below, 2.7 cm is the width of the bit, 0.1 second is the period between two proximity sensors events and 4.5 is the number of intervals required for the complete reading of the barcode.



(a) Simple reading method



(b) More robust reading method

Figure 7: Illustration of two reading methods

$$command_{min} = \frac{4 \cdot 2.7cm}{4.5 \cdot interval \cdot 0.1s \cdot 0.032} \tag{3}$$

The maximal speed can be computed in a similar way, here we want the robot to perform the last bit reading action at the end of the last bit. That is why the number of intervals is now 3.5.

$$command_{max} = \frac{4 \cdot 2.7cm}{3.5 \cdot interval \cdot 0.1s \cdot 0.032} \tag{4}$$

Although the shape is similar, we can observe some differences between the theory and the practice. First, the spikes seem to start for speeds a bit smaller than the theory predicts. It comes probably from the length of the barcode, which is not exactly 10.8 cm once printed. Moreover, the peaks extend more than this difference on the right. It can be simply explained by the fact the reading was ending before the end of the barcodes for the slower speeds when doing the measurements. In this case, it is not a problem but in the network it could be a bit more complicated. Indeed if the reading ends before the end of the barcode and if the last bit is a black one, it could trigger a new reading action and some unfortunate consequences could happen.

Table 5: Speed chosen for the reference robot

	SLOW	**MEDIUM**	**HIGH**
Command	150	260	410
Speed	4.8 cm/s	8.32 cm/s	13.12 cm/s

In the beginning, I also wanted to use different gray levels to reduce the barcode size (i.e.: making the reading method less susceptible to make mistakes) and multiplying the number of states possible. However, a few problem arise. First the memory allocated for the Aseba code is limited and increasing the number states increase the number of possible behavior to implement. I decided therefore to develop less state with a more global behavior that can be modulated with only a few varying parameters for the different situations. Moreover, each sensor reads different value for the same gray level. In this case, it would be extremely difficult to design a robust reading method with three or four reference level of informations. Thus, I finally implemented binary barcodes.

One last aspect to keep in mind, I should take the dynamics into account when placing two barcodes next to each other and if the first one corresponds to a change of speed. I would have to let

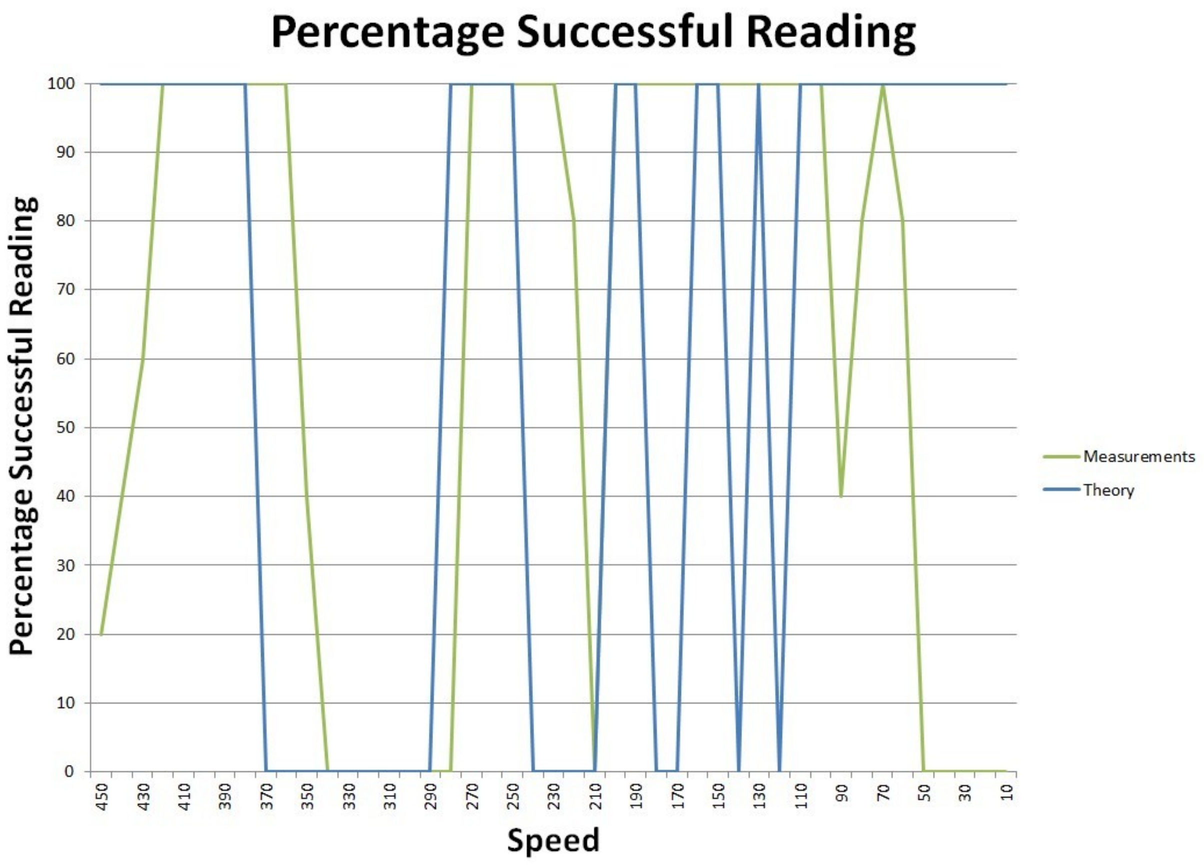


Figure 8: Reading statistics with the second reading method over 5 runs (c.f.: figure 7b)

a sufficient space between the barcodes to let the robot reach is commanded speed before starting the reading of the second barcode. In my project there was never two barcodes close enough to make this problem arise but a similar problem appeared when I asked the robot to go backward. This will be more detailed in the next section about collision avoidance.

Finally, we can find in the table 6 below, the meaning of the different barcodes. The location position can be seen on figure 22.

Table 6: Barcode interpretation given its location

Code	**1**	**2**	**3**	**4**	**5**	**6**	**7**
On the road	Low speed	Medium speed	High speed	Branching	Crossroad	Location	Roundabout
In a location	Location 1	Location 2	Location 3	Location 4	Location 5	-	-

6 Collision avoidance

At first, a very simple collision avoidance was implemented (c.f.:figure 9). When one of the front sensor is detecting something above a given threshold, the Thymio saves the speed it has and then stops moving. Once the values are again below the limit, it reassigns the saved speeds as the command and start moving again. The main problem of this method is that it created a very jerky movement (like stick-and-slip) when a Thymio is constantly behind another one driving slower.

For the purpose of reading, this option is deactivated when the robot is on a barcode and when it stops following a line. Therefore there is no disturbance during the reading as it strongly depends on timing. However the robot stops as soon as it leaves the *READ* or *CONTINUE* state.

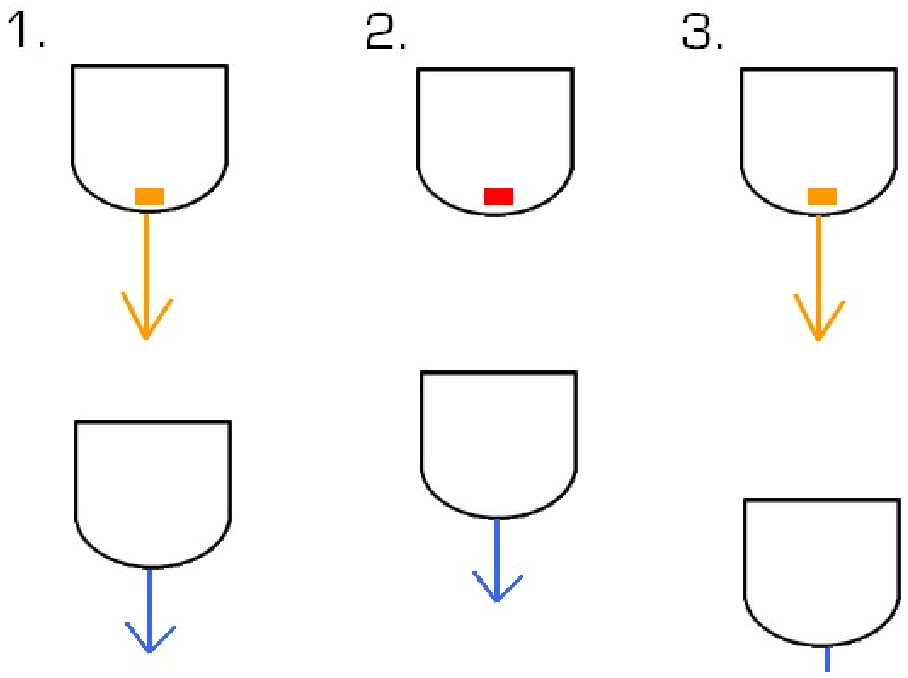


Figure 9: Simple collision avoidance. The arrow represents the speed and the colored rectangle the activation of the proximity sensor

On a second iteration (c.f.: figure 10), I improved the method. Now, above a given threshold value, the Thymio starts to slow down proportionally to its nominal speed (i.e: low, medium or high) and to the maximal value measured by one of the front sensors. If this value is above a second threshold, this time, the robot will stop.

To keep reading the barcode correctly, the collision avoidance is still not working when the robot is in the *READ* state. However, another solution was implemented: when reading the synchronization bit, if there is another Thymio detected, the one behind will go back at the slowest speed defined for 1.2 second and then start moving normally again. This backward motion will let time to the robot in the front to finish its reading and avoid a collision. During this backward movement the robot still performs line following to avoid losing it and a reversed collision avoidance. I re-implemented the first method but, this time, using the sensors on the back of the Thymio. The most important goal of this maneuverer is to give time to another Thymio to finish reading the barcode, therefore the timing is more important than the distance traveled. This is why I am using the first method and in addition a timer. A robot can detect another robot on the barcode until approximately the middle of the length of the barcode. By assuming, it is driving as the slowest speed defined on the road, the robot on the code requires approximately 1.2 seconds to finish reading crossing the code. This corresponds to the timing I assigned on the robot that needs to go backwards. One additional detail: after the backward motion, the distance was not always sufficient for the robot to reach its speed before engaging the maneuverer . Therefore I always assigned the slowest speed possible when starting moving forwards again to avoid problem during the barcode reading.

For the same reasons of timing, the collision avoidance was not working when the robot is not following a line (i.e.: the *CONTINUE* state). But I denoted that this state occurred only at slow speed. Therefore, when the robot is in this state and the front sensors value is above a given threshold, the robot stops moving and it stops the timer responsible for this motion without line following. I used also the first collision avoidance method developed to minimize the effect on the timing. Indeed, the robot stops counting when it stops moving, if it was also slowing down, how to count would become a serious problem. I also assumed the effects of dynamics to be negligible as the robot is always traveling at low speed when it has to stop following the line.

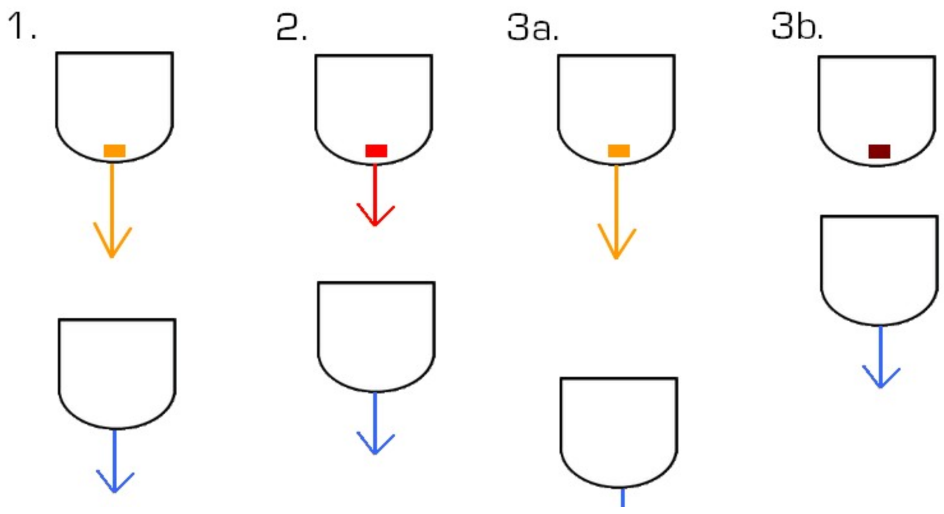


Figure 10: Advanced collision avoidance. The arrow represents the speed and the colored rectangle the activation of the proximity sensor

7 Intersection design and control

The choice of intersections was done with the other student who was also working on the project.

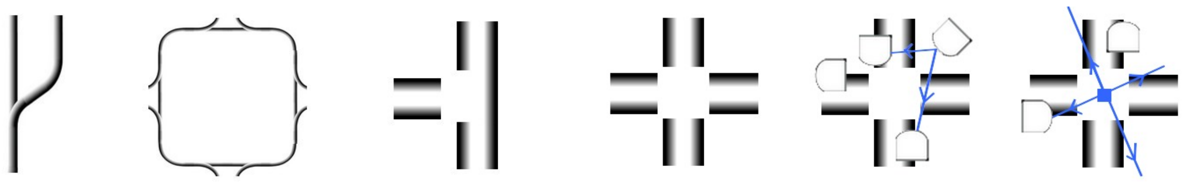


Figure 11: Illustration of different possible intersections. Pros and cons are presented in table 7. The bridge crossroad is not shown

We decided to implement at first a roundabout for the intersection and a branching intersection for the locations. Then, with the time remaining, we implemented also a crossroad and an additional branching intersection.

A particularity about our different intersection is that I am not using complete barcodes for indicating the exit or anything else. In fact I am only using the synchronization bit. Indeed, the decision (exit or continue) is made before the exit and the synchronization bit is just here to apply the decision. The sequence of these different choice depends on the type of intersection. However I am also using full barcodes in the location but this is used for localization in the map. More details about this topics can be found in Benjamin Kern's report [1].

7.1 Roundabout

Thanks to its simplicity, the roundabout was the first intersection implemented. In our initial city design, I used a 2-entry roundabout (c.f.: figure 21). But it is designed in a way that it could be easily extended to additional entries (c.f.: the final map on figure ??). The main problem of this intersection is that the robot has to cross the road and therefore it cannot perform line following.

Table 7: Possible intersections

Intersection	**Pros**	**Cons**
Branching	Simple Smooth and continuous	Limited to 2 different ways Must cross a line
Roundabout	Generalization of branching	Must cross a line sometimes
crossroad **(3 road)**	Less space than the roundabout	Limited to 2 different ways Control differenet for each entry Must move without line following Complex interaction if few vehicles
crossroad (4 road)	3 possible ways Control identical for each entry	Must move without line following Complex interactions if a few vehicle
crossroad (managed)	3 possible ways Control identical for each entry External robot manage interactions	Must move without line following
crossroad (red light)	3 possible ways External robot manage interactions Control identical for each entry	Must move without line following Less intelligent management
crossroad (bridge)	Use branching 3 possible ways	Space Requires building a 2-stage structure Left turn difficult to implement

I had to define a few parameters for our roundabout: where to put the exit bit, how long should the robot stop the line following to cross the road, how long the Thymio should drive in the exit before changing its state?

First of all, due to the sharp turns, the Thymios are using the slowest speed defined in table 5.

Concerning the location of the exit bits, I wanted to minimize the distance without line following. But on the other hand, the robots needed for it to go as straight as possible in this region. Therefore I decided to put the code at he beginning of the exit turn (c.f.: figure 14), so the Thymio is currently on a straight line. However, it appeared that the controller on the Thymio was not strong enough to make it drive straight. To counter this effect, I put a bias on the speed of the left wheel to generate a light left turn. This made the Thymio able to catch the line almost at any time.

Once the robot has crossed an exit, it has to cross the next entry. Although the distance before the entry road and the roundabout merge completely is slightly bigger than the distance to cross the exit, I did not change the timing before starting the line following function to simplify the control. Moreover the two roads are converging, therefore there is no chance to catch the wrong one (c.f.: red part on figure 14).

The timing was determined in a very simple way, I estimated with the the speed of the robot with equation 1 and, given the distance, I computed the required counter limit.

$$\frac{distance\ [cm]}{speed\ [cm/s]} frequency\ [Hz] = \frac{10}{4.8} 10 = 21\ counts$$

The timing for the exit was handled in a similar way. This corresponds to the time before the robot change its state to the one it had before the roundabout.

$$\frac{distance\ [cm]}{speed\ [cm/s]} frequency\ [Hz] = \frac{20}{4.8} 10 \approx 42\ counts$$

I had also to determine where to put a new barcode after leaving the roundabout. To select the appropriate placement, I ran a few tests with different distances from the end of the exit. We can

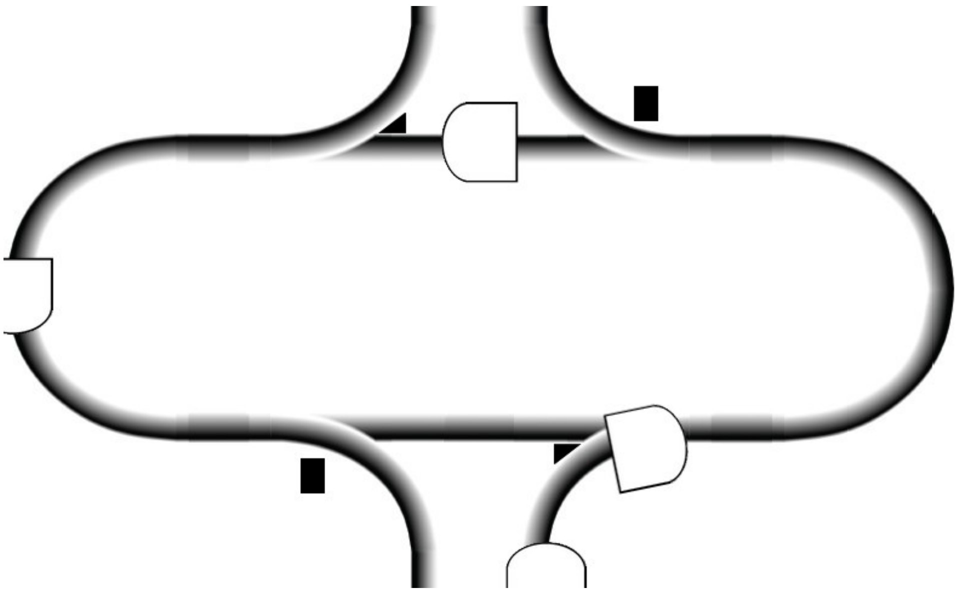


Figure 12: The first implemented roundabout

see the results on the graph on figure 13.

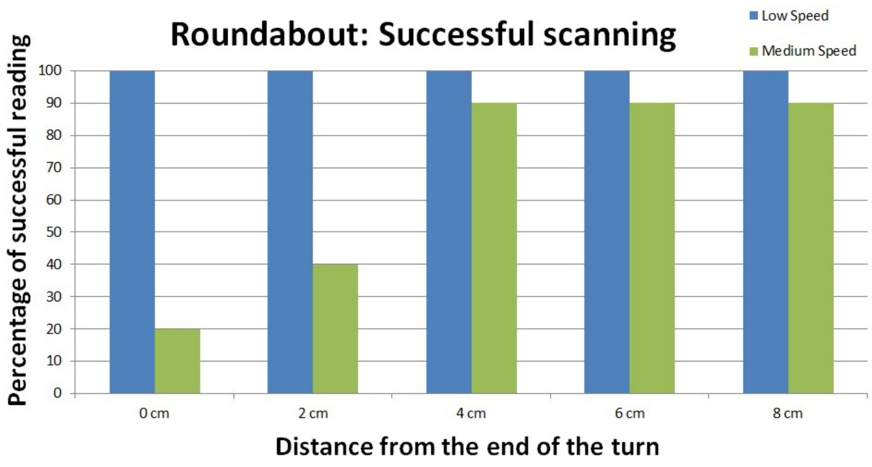


Figure 13: Scanning statistics for 2 different speed over 5 runs

As we can see, below 4 centimeters the results at medium speed are particularly low. On the other hand, increasing the distance above 4 centimeters did not bring better results. That is why I chose to put the code at a distance of 4 centimeters from the end of the turn of the exit.

An example of implementation is shown below in figure 14.

7.2 Location

The design for the location is inspired by railway tracks. It uses the branching intersection to create two parallel tracks. An advantage of this design is that we can put an intersection almost everywhere as long as there is a sufficiently long straight road. Like in the real world, the speed in a location is the smallest speed possible on the the road (c.f.: table 5).

For the timing, the reasoning was exactly the same as for the roundabout. In case of a continuation

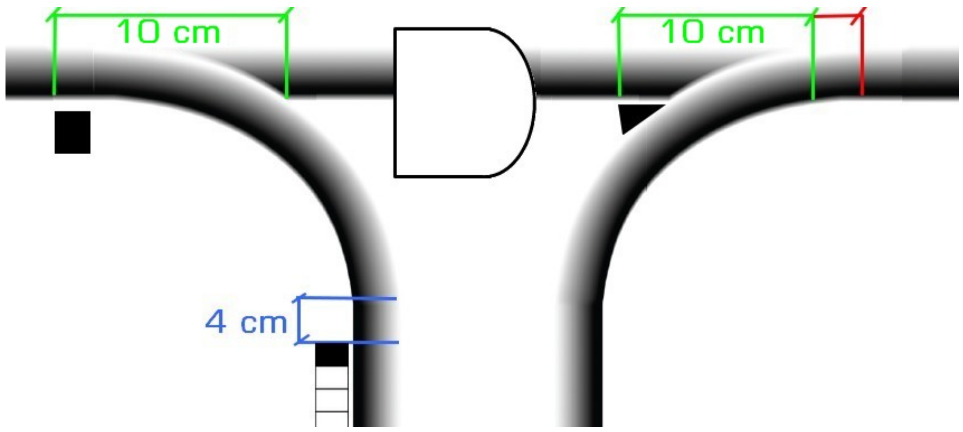


Figure 14: Final implementation of the roundabout: entry and exit

I have:

$$\frac{distance\ [cm]}{speed\ [cm/s]} frequency\ [Hz] = \frac{10}{4.8} 10 \approx 21\ counts$$

When choosing the exit mode, the Thymio will set a flag. This flags concerns the reading of a barcode about the localization in the map. Once the flag is set, the robot can read a standard barcode of 4 bits but it will interpret it differently. To avoid misreading other barcodes, this flag can only be set if the robot is an a location intersection and if it choses to enters the location and after reading the bit indicating the entry. Moreover it is reset as soon as the robot ends its stop in the location. After the barcode, the robot will continue driving for a moment on the track and then stop for a small fixed amount of time (2 seconds, i.e.: 20 proximity sensor events), like if it was delivering something. When the robot starts moving again, it will come back to the *ROAD* state at the lowest road speed, like in a village.

$$\frac{distance\ [cm]}{speed\ [cm/s]} frequency\ [Hz] = \frac{62}{4.8} 10 \approx 130\ counts$$

To find the optimal location where to put a barcode, I also ran some test about scanning in the locations. As we can see on figure 15, the optimal distance where to place the barcode is also 4 centimeters. Due to the symmetry of the entry and the exit, I assumed the result reliable also for the exit of the location.

We can see an example of implementation in the figure 17 with all the different barcodes and bits.

A problem that appeared after testing the map with several Thymios, is the collision when two Thymios arrive in parallel at the end of the location. The two tracks, being close to each other, none of the Thymio is able to detect the other one with the proximity sensors and to start the obstacle avoidance routine (c.f.: figure 16). A solution was found by installing a traffic light Thymio at the exit, whose job is to command to one lane at the time to drive. More details can be found in Benjamin's report [1].

7.3 Branching

The difference between this branching intersection and the location one is the design and the behavior. The goal, here, is to connect two different loop in the new map (c.f.:figure **??**), therefore I do not want for the robot to stop in the middle of track. Moreover, concerning the design, we do not want for the robot to come back in the loop he just left. It is now more similar to the entry-exit design of the roundabout. In fact we took exactly this part from the roundabout with a part of the

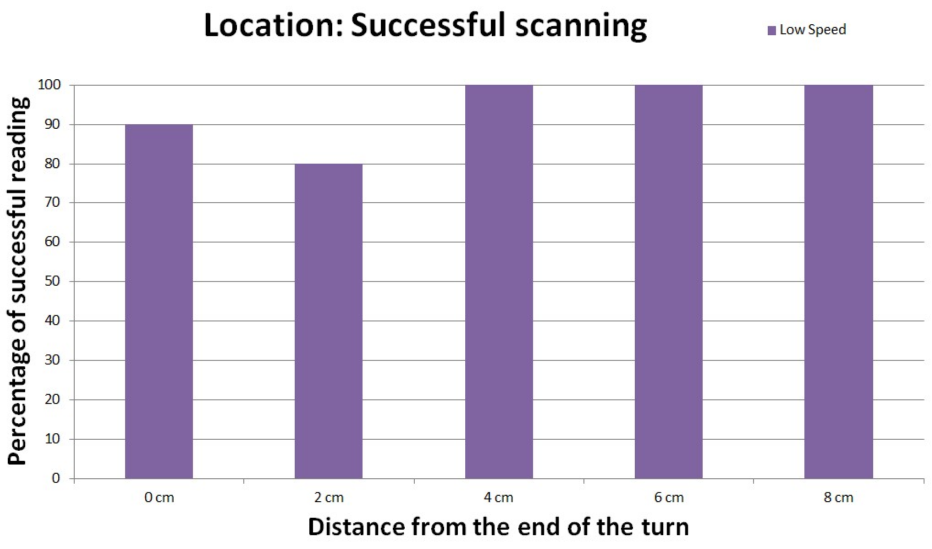


Figure 15: Scanning statistics in a location over 5 runs

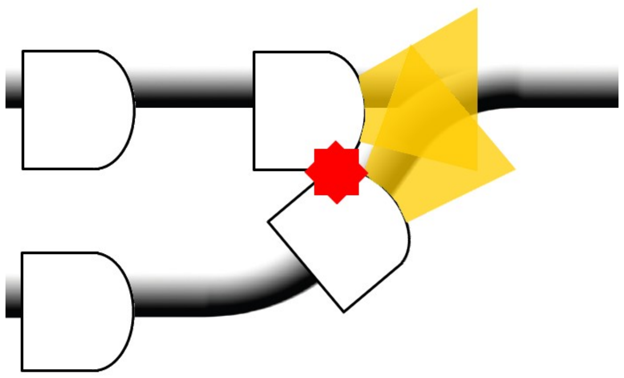


Figure 16: Collision problem when two robots are driving on parallel lanes. The yellow cone corresponds to the area covered by the proximity sensors.

control of the location intersection to create a new intersection with new interesting possibilities. To avoid problems due to the sharp turn, I forced the speed to be also the slowest possible (defined in table 5).

As I said, I used a code similar to the location one and I computed the following values for the counters for the *CONTINUE* state and the *EXIT* state respectively.

$$\frac{distance\ [cm]}{speed\ [cm/s]} frequency\ [Hz] = \frac{10}{4.8} 10 = 21\ counts$$

$$\frac{distance\ [cm]}{speed\ [cm/s]} frequency\ [Hz] = \frac{20}{4.8} 10 \approx 42\ counts$$

Using a similar design as the entry and exit of the roundabout, I did not need to redo all the measurements concerning the location of the bits and the barcodes. When it leaves the intersection, the robot re-assign its speed before it. The final implementation is the same as the one shown in figure 14.

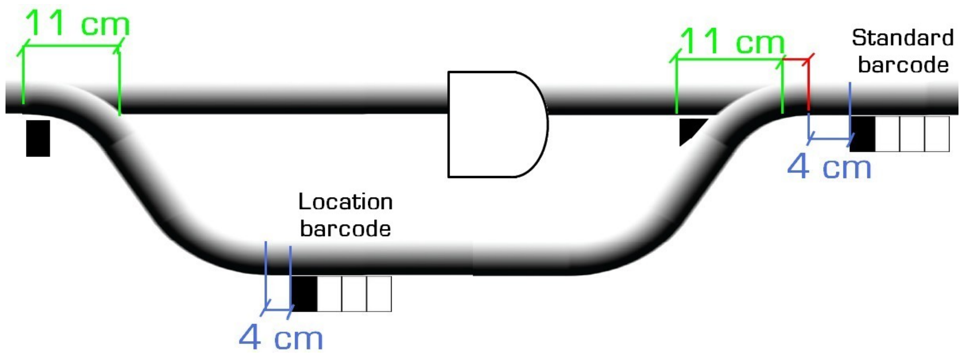


Figure 17: Final implementation of the location

7.4 Crossroad

The last intersection I implemented is a 4-way crossroad. The main advantage is that the control is independent of the entry point. Moreover, it also appeared that this intersection takes significantly less space than the roundabout for the same number of entries.

The design of the intersection is quite far from a real-life cross-road. Indeed, the dependency on the line following function can be very constraining. Therefore I wanted to minimize the distance without line following. In the end, I had to find a trade-off between the distance to travel without line following and the number of bits required (a smaller distance requires more crossings and therefore more bits). The design implemented is shown in figure 18. Due to the complexity of this intersection, I chose once again to force the speed to be the smallest road speed possible. Moreover due to the complexity of the interactions within the intersection, We chose to add two traffic lights Thymio in the center to manage which lanes are open and which are not. More detail can be found in Benjamin's report [1].

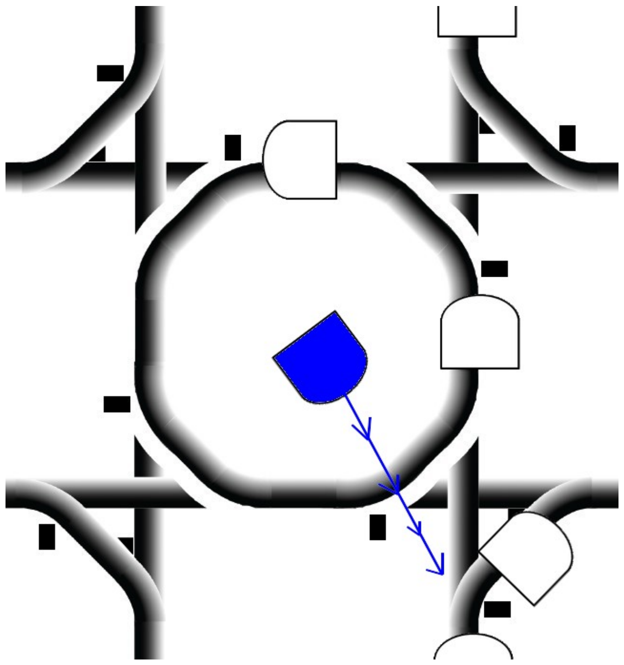


Figure 18: Implementation of the crossroad

I detail the behavior for every possible choice: turn right, go straight or turn left. Each choice implies a particular sequence of specified actions. To turn right, the action is taken on the first bit encountered and the robot will act exactly in the same way as if it was the exit of the roundabout. One little modification is the location of the bit, I will explain how its position was selected in the next paragraph. In the end I just had to adapt the counter limit.

To go straight (c.f.: figure 19), the robot has to cross the right turn (1), then the perpendicular road(2) and reach the circle in the center, the leave this circle (3) directly to get the road on the other side. During this phase it has also to cross the second perpendicular road (3) and finally the robot must cross the last turn (4).

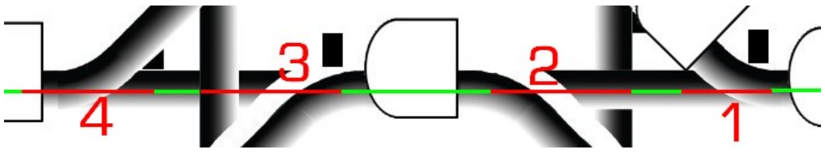


Figure 19: Go straight action scheme. The red line corresponds to an inactive line following (but they are not all of the right dimension)

Note that I did not put any bit to indicate when to cross the first perpendicular road (number 2 in figure 19). To explain this choice assume a robot want to go straight in the crossroad from the left-side entry (c.f.: figure 20). When it will cross the second perpendicular road, if there was a bit (blue square) here for the first robot (green and red line), it could detect it and could stop following the line sooner than expected (purple line). In theory this should not be a problem as the distance without line following is long enough to cross the last turn. But in reality after crossing a road, due to the forced slight left turn during the *CONTINUE* state, the robot often oscillates and if a bit is detected during this oscillation phase, the robot could go in unexpected directions completely leaving the road. That is why the robots are using the black side of the road as a barcode. In summary, the robot has to perform the cross action (i.e: *CONTINUE* state) four times.

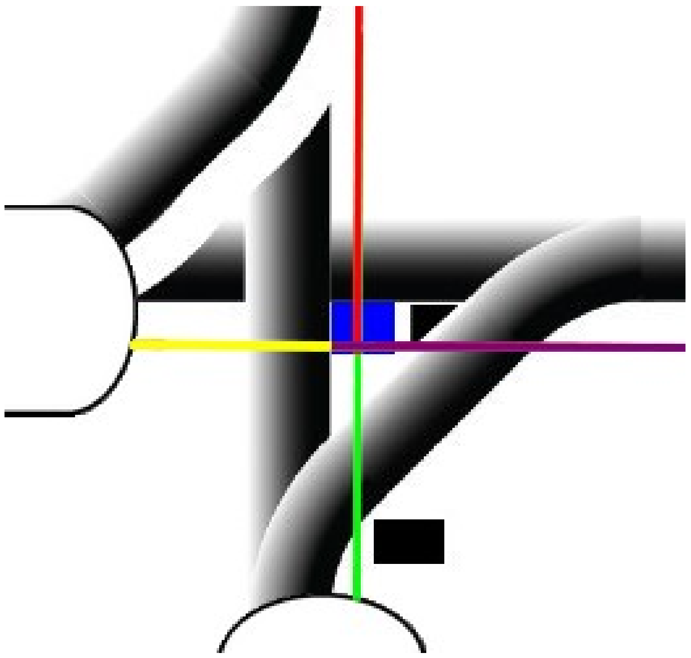


Figure 20: Problem of using an additional bit instead of using the road

To simplify the code, I wanted all the four actions to be similar and the most constraining distance was the one to enter the circle (when there is no bit).

$$\frac{distance\ [cm]}{speed\ [cm/s]} frequency\ [Hz] = \frac{10}{4.8} 10 = 21\ counts$$

Once this distance fixed, I could place the resulting barcodes accordingly.

Finally the left turn action is the most complex. Until the central circle, the actions to perform are similar to the going straight action. When encountering the third bit, this time the robot has to turn left. To do so, I tricked the robot by asking him to perform like it was exiting the crossroad. Therefore it could achieve line following then I forced him to come back to the *ACTION* state instead of the standard *ROAD* state after an exit. When the robot encounters the next bit on the central circle, it will again perform the same as the going straight action (i.e.: leaving the central circle and crossing the last turn).

As the robots are in the same state for leaving the crossroad and turning left in the central circle, I selected the most constraining distance between the two to compute the counter limit. It was the left turn in the central circle.

$$\frac{distance\ [cm]}{speed\ [cm/s]} frequency\ [Hz] = \frac{14}{4.48} 10 = 29\ counts$$

Despite the limit is a bit low for completely exiting the crossroad when turning right, it causes no problem. Indeed we decided to force the lowest road speed after this decision to help managing the end of the turn. In addition we always put barcodes at the exit with a new speed command if required.

After a few tests, we dicovered that the robot often missed the last bit to exit the intersection. It resulted either in going in arbitrary directions (when reading the bit or something similar) or in taking a track in the wrong direction. To avoid this problem, I decided to force the fourth continue action directly at the end of the third one. Therefore the robot would not oscillate and miss the last bit. On the other hand it will drive further away from the road due to the forced left turn in the *CONTINUE* state. This problem is explained latter.

8 Sound

Among the optional objectives, I could implement priority vehicles. An idea for implementation was to use sound, like a police car or an ambulance. However the tests were not successful. Indeed, the noise generated by the motor ($intensity \approx 150$) was largely covering the sound generated by a Thymio ($intensity \approx 50$ at maximum). Therefore I gave up on this solution.

If I have had more time, I would have explored the same behavior but, this time, using the communication protocol implemented. But I know we are emitting in all directions, therefore it will be hard to communicate the right Thymio at the the right time.

9 Map

During the project, we had the possibility to test our Thymios on two different maps. We first designed a very simple map (c.f.: figure 21). This was very useful to test all the basic behaviors and the intersection management at first with one Thymios. In a second phase, we started to add more Thymios on the map and we could identify additional problems and solve them as well. Among them I can cite the collisions while reading a barcode or when the robot is in the *CONTINUE* state but also the speed after a backward maneuverer and collisions at the exit of a location.

The second map was developed to challenge more the robot on the high-level control (c.f.: Benjamin's report [1]). However it was also a good way to test the controllers developed for all interactions. Unfortunately, we had to cut a bit of the map to fit on the space at disposition. The tested design can be seen on figure 22.

With this new map. new problems appeared. First we discover that despite the calibration of sensors, we had still problems. Indeed, in case of sharp right turns, the reading could trigger although I developed a security to avoid that. I could have avoided this problem by giving a command more

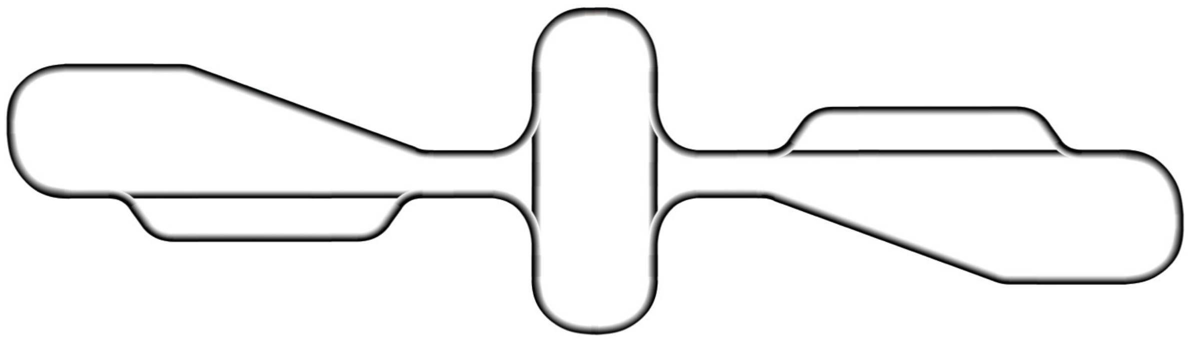


Figure 21: First map implemented

on the black side of the road but due to sharp right turns it was not always possible. I discovered that it was a real trade-off between this two issues was hard to manage depending on the Thymios. Moreover with the variety of Thymios, we had a really wide range of sensors readings for the same area. To be as less dependent as possible on the sensors, I decided to define the white threshold (i.e.: the security on reading) given the reference gray value for each Thymio. Despite all our efforts, it was not possible to make all the Thymios in our possessions drive on the map. In case of extreme values, a trade-off was impossible to find. Inverting the gradient direction would have been a first idea of solution.

As mentioned several times before, I forced a light left turn each time the robot has to stop following the line. Due to the location of some of the branching intersections (i.e.: right after a U-turn), a large bias was required. On the other hand, for the crossroad a much smaller bias was needed to achieve good performance. Instead of trying to find a trade-off, I decided to order the location (i.e.: *actionType* variable) and to define the bias depending on its number. In this case, we have a small bias for the crossroad and the location and a larger one for the roundabout and the branching.

Finally, a last problem that appeared is the decrease of the range of traffic light Thymios for communication. This resulted in robots stopping closer and closer from the center of the crossroad and, in the end, creating a traffic jam. We did not find any solution to avoid this, except stopping the network and charging the traffic light Thymios.

10 Calibration

When it came the time to test several Thymios on the map, a new problem appeared: each Thymio can observe little variation in terms of speed when sending the same command. To solve this problem, I designed and implemented a calibration setup presented on figure 23a. The gradient is present to maintain a trajectory as straight as possible to have the best comparison possible with the reference. It permits also to compensate the differences between the two motors on the same Thymio.

The principle is extremely simple: I established reference measurement with a robot performing well on the map and then I assumed that the product command $\cdot$ counts is a constant. In a more detailed way, if the robot has a number of counts greater than the reference for the same command, it means it is going slower. Therefore I have to send him a bigger command to observe the same speed. After the calibration of all the robots, this method seems to work well.

I had also to calibrate the reference value for the line following. Once again, I used a reference Thymio to build the calibration setup. I printed a small part of a track and I put stoppers in a way that the Thymio ground sensor was on the reference gray level. With the help of Aseba, I was able to read that value read by the ground sensor above the reference gray level and to assign it to be the new reference for the line following. A scheme of the setup can be seen on figure 23b

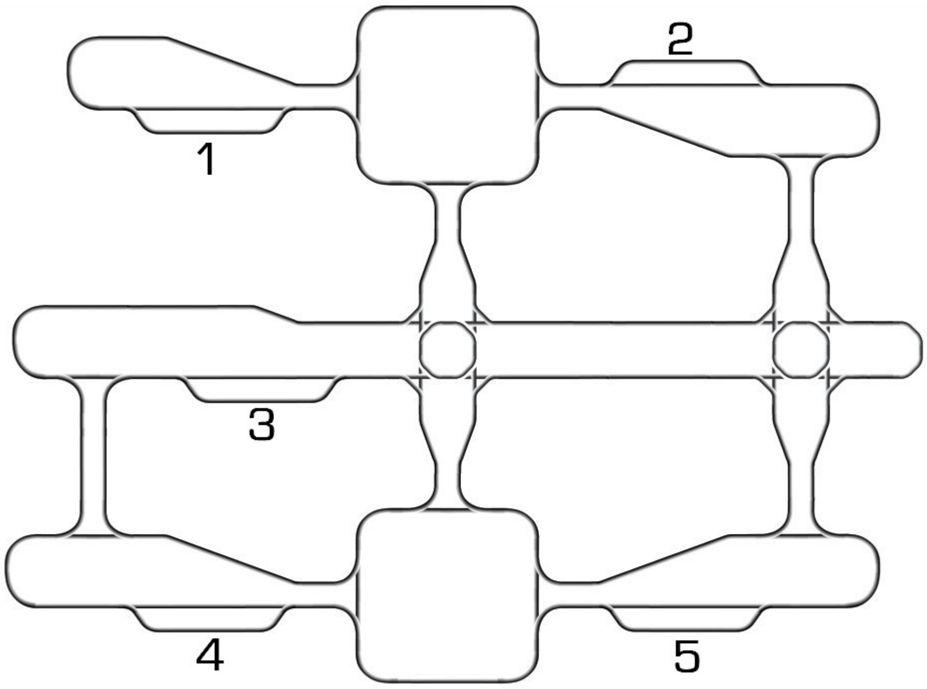
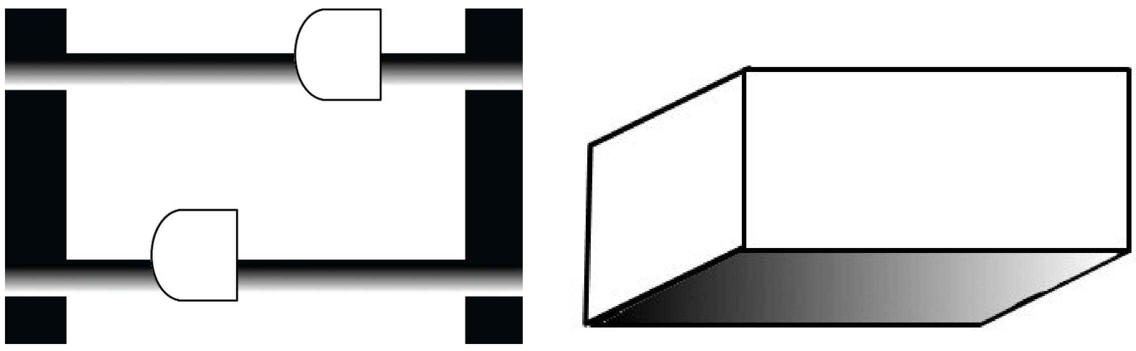


Figure 22: Second and final map implemented



(a) Speed calibration (b) Gray level calibration

Figure 23: Calibration setup

11 Possibilities and Improvements

This is a non-exhaustive list of ideas. Due to the creative aspect of this project, the number of possibilities in almost unlimited. The only limiting factor is the memory size, which can be extended with an SD card.

- Straight change of directions (L-turn):
  The main advantage of this kind of change of direction is the pace in space compared to use of a turn. With this kind of change of direction, I could implement for example real parkings in the different locations to allow to more robots to stay in the same place. It would also allow us to develop a new kind of crossroad with less lines maybe a simpler code in terms of actions to perform.

- Increase globally the speed:
  In the first stage of the project, I chose three different speeds to be used. However once we printed the first map, it appeared difficult for the robot to manage even small change of directions at high speed. Therefore I mostly used the medium and the slow speed on the road. On the second phase, I could use it but only on a small part of the network. The main goal would be to use more the fastest defined road speed. To do so, I would probably have to improve the line following controller and inverting the gradient would be a good idea as most of the turns are going on the left. By inverting the gradient, I would get a better recuperation of the road in case of loss as it was explained before in the report. Moreover we are always using the lowest speed in all the intersections. It could be also interesting to redesign them as well to allow higher speeds and reducing the probability of creating a traffic jam.

- Priority at intersections:
  For the moment, in the non-managed intersections, there is no priority at all. Most of the time we have a zipper-like behavior like it is supposed to be in a traffic jam. However if we want to approach more real-life conditions, we could implement priority management on the robot. When arriving in a given intersection the robot should check if it can enter it instead of directly going. Therefore we should oblige a stop state and a checking motion after reading a barcode.

- Priority vehicles:
  An interesting add-on to our system would be the implementation of different behaviors amongst the drivers. I could for example implement priority vehicles such as ambulance, fire-fighters or cops. In this case I would have to develop a method for the other users of the road to avoid them. For example by leaving the road on one side. After they also would need to find the road back. An other possibility would be also to implement different behaviors amongst the Thymios, some of them driving more aggressively or faster and some of them more careful or driving under the speed limit. In order to achieve this I would need to be able to manage faster robots and therefore have a map and controllers with such capability.

- Two-stage map:
  Our map is completely flat. It would be an interesting challenge to add some morphology to this environment. At first we could implement small hills to evaluate the capability of the Thymio to climb a slope and then we should build a more complex environment. In the end, implementing a two-stage environment could be an interesting idea.

12 Conclusion

During this project, we implemented a complete road network in terms of road, intersections and locations. The roads were implemented using a gray gradient to allow a smooth line following but I had some trouble in the case of loss of the road. I tried different solutions but none where satisfying enough.

To collect informations, barcodes where added on the right side of the road. One of the main problem was to not trigger the reading if the responsible sensor came on the black side of the road. To avoid this problem, I check the value of the road following sensor, indeed if the reading sensor is on the black side of the road, the line following sensor will be on the light part of the road. Therefore I fixed a threshold under which a black measure is interpreted as a bit. Once the barcode designed and the line implemented, I tested the robustness of the reading method and obtained satisfying results for wide range of speeds mainly to the adaptation of the sequence of actions. In the end, I based my choice of final speeds depending upon these results.

When adding more Thymios on the road, collision avoidance became important. I implemented two different methods. One that smoothly decrease the speed until a threshold across which the robot stops and one that simply stops if the measurement is above a given threshold. Both were implemented in different conditions and provided satisfactory results. In case of a barcode reading, I added also a backward maneuverer to let time to the Thymio currently reading to finish it.

To increase the complexity of the network and the interactions between the robots, we designed different intersections. I started first with the control in a roundabout and in a branching location. The main problem was to find a way to cross tracks and find the right one afterwards. To do so, I decided to stop the line following during the minimal amount of time. It appeared that the robot had sometimes problem of straightness of the motion on the road after a turn. To counter this problem, I put a bias on the speed of the left wheel to induce a slight left turn. When I implemented the crossroad, this problem reached a new stage as the robot had to stops line following four time in a row with only small distances to reach straightness again. It resulted that the last bit was often missed but the robot was however sometimes on the right road. Therefore I decided to force the last continue action after the third one to leave the intersection.

I also tried some experiments with the microphone and the speaker to implement priority vehicles. Sadly the sound produced by the speaker is largely overtaken by the sound generated by the motors when the robot is moving. A better idea would be to use the communication protocol.

The final tests, made new problems to appear. I had principally problems with the sensors readings and to find an optimal trade-off between turning left without leaving the road and turning right without triggering a reading. In addition, I had to adapt the bias for the continue action for each intersection. In the end, I was prepared for the decrease of the speed of the driving Thymios because of their batteries but not to the decrease of the communication range, resulting in traffic jams and forcing us to stop the experiment to charge the traffic light Thymios

Due to its creative aspect, this project can be improved or developed in multiple different ways, the only limit is the memory space available on the robot. However this obstacle can be overcome by using a SD card. Such developments can be now rapidly performed as all the basic functions (line following, barcode reading, collision avoidance) are working well.

Bibliography

[1] Benjamin Kern. Thymio road network. Technical report, Ecole Polytechnique Fédérale de Lausanne.

[2] Roland Longchamp. *Commande numérique de Système Dynamique (Vol. 1)*. PPUR, 2010.

[3] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. MIT Press, 2004.

[4] Thymio. http://thymio.org/, April 2015.

Appendices

A Thymio Road Network code

```
<!DOCTYPE aesl-source>
<network>


<!--list of global events-->


<!--list of constants-->
<constant value="500" name="GREY_REF"/>
<constant value="152" name="SPEED_LOW"/>
<constant value="264" name="SPEED_MED"/>
<constant value="416" name="SPEED_HIGH"/>
<constant value="400" name="GREY_THRESH"/>
<constant value="1000" name="OBS_PROX_THRESH"/>
<constant value="3000" name="OBS_PROX_THRESH_MAX"/>
<constant value="0" name="S_STOP"/>
<constant value="1" name="S_ROAD"/>
<constant value="2" name="S_READ"/>
<constant value="3" name="S_ACTION"/>
<constant value="4" name="S_EXIT"/>
<constant value="5" name="S_CONTINUE"/>
<constant value="6" name="S_BACK"/>
<constant value="0" name="A_BRANCHING"/>
<constant value="1" name="A_ROUNDABOUT"/>
<constant value="2" name="A_CROSSROAD"/>
<constant value="3" name="A_LOCATION"/>
<constant value="0" name="DS_NOTHING"/>
<constant value="0" name="DS_EXIT"/>
<constant value="1" name="DS_CONTINUE"/>
<constant value="2" name="DS_TURN_LEFT"/>
<constant value="4" name="DS_STOP"/>
<constant value="16" name="NB_NODE"/>
<constant value="0" name="CS_EAR"/>
<constant value="1" name="CS_TELL"/>


<!--show keywords state-->
<keywords flag="true"/>


<!--node thymio-II-->
<node nodeId="1" name="thymio-II">var state = S_STOP
var oldState = S_STOP
var decisionState =  DS_NOTHING

var speedType = SPEED_LOW
var savedSpeed = SPEED_LOW
var savedSpeedInters = SPEED_LOW
var addSpeedLeft = 0
var addSpeedRight = 0

var controlFactor = 1
var intensityDiff = 0
var intensityObst = 0
var intensityDiffObst = 0
var white_threshold = GREY_REF+200

var counter = 0
var counterContinue = 0
var counterStop = 0
var counterRead = 0
var code = 0
var interval = 0
var readLocation = 0
var random = 0
var actionType = A_LOCATION

var timerON = 0
var incrementCounter=1

########
#Variables for the intern map
var map[NB_NODE]
var maptype[NB_NODE]
var orientedIntersection[8]
var nodeTab[5]
var newNode = 0
var newNodeType = 4
var currentNode =  9
var oldNode = 9
var positionError = 0
var i = 0
var stick
var nodeCount=0
var echange=0
var t=0
var indice
var oldIndice = -1
var counNode = 0
var test2=1
var countExit=1
var nbRond=0
var NbCont=2
#communication variables
var comm_state=CS_EAR
var last_ok =11
var l=0
var countPing = 1

#pos[0]= actual position
#pos[1]= old position
#variable for measurements
var count=1

#Map declaration
#bit[0 and 1] = node type + address in the table orientedIntersection
#bit[following] = 1 if connected with the corresponding node.
#              9876543210
map[0]=0b000000100010100
map[1]=0b001000001000100
map[2]=0b000000000101011
map[3]=0b000000010011100
map[4]=0b000001000001000
map[5]=0b000110000000000
map[6]=0b000110000000000
map[7]=0b000000000001010
map[8]=0b000000000000001
map[9]=0b000000000000001
map[10]=0b000000000000100
map[11]=0b000000000000010
map[12]=0b000000010000000


#Table for the type of the nodes
maptype[0]=0b01
maptype[1]=0b01
maptype[2]=0b10
maptype[3]=0b10
maptype[4]=0b11
maptype[5]=0b11
maptype[6]=0b11
maptype[7]=0b11
maptype[8]=0b00
maptype[9]=0b00
maptype[10]=0b00
maptype[11]=0b00
maptype[12]=0b00


#Node type:
#00=location
#01=roundabout
#10=crossroad
#11=branching
#Information on the order of the nodes in an intersection 001 010 011 100
#The bit bit coding represents the nodes connected with 3 bits(8 possibilities of connections) but it is saved
#on 16 bit therefore maximum 5 nodes per intersections with 001 == smallest connected node and 000 indicates null
orientedIntersection[0]=0b011010001
orientedIntersection[1]=0b011010001
orientedIntersection[2]=0b011010100001
orientedIntersection[3]=0b011010100001
orientedIntersection[4]=0b010001
orientedIntersection[5]=0b010001
orientedIntersection[6]=0b010001
orientedIntersection[7]=0b001010

##########

#Turn off some firmware behaviour
call leds.prox.h(0,0,0,0,0,0,0,0)
call leds.prox.v(0,0)
call leds.temperature(0,0)
call prox.comm.enable(1)
#subroutines called by communication

#Stop the Thymio
sub allStop
  if  state!=S_STOP then
    oldState=state
    state=S_STOP
    motor.left.target=0
    motor.right.target=0
  end

#Make the Thymio moves
sub allStart
  if  state==S_STOP then
  state=oldState
  speedType=savedSpeed
end

#Gives the compatible node in the variable newNode
sub searchCompatibleNode
  nodeTab=[-1,-1,-1,-1,-1]
  nodeCount=0
  stick = map[currentNode]

  #Making of the connected node table, it contains all the nodes connected to the currentNode.
  #It finds the index corresponding to the node were the Thymio comes from
  t=0
  oldIndice=-1
  for i in 2:17 do
    if maptype[currentNode] == 00 then
      if stick%0b10 == 0b1 then
          newNode = i-2
          return
      end
    else
      if stick%0b10 == 0b1 then
```

```
        nodeTab[t]=i-2
        if  i-2==oldNode then
          oldIndice=t
        end
        t++
      end
    end
    stick = stick >> 1
  end

  #Treatment of the case of the branching intersection
  if maptype[currentNode] == 0b11 then
    if  counterContinue==0 then
      newNode=nodeTab[orientedIntersection[currentNode]%0b1000-1]
    else
      newNode=nodeTab[(orientedIntersection[currentNode]>>3)%0b1000-1]

    end
    return
  end

  #Special case 1 treatment (see section 4.3 in Benjamin's report)
  if oldIndice==-1 then

    for i in 0:3 do
      if i<t then


        stick = map[nodeTab[i]]
        stick=stick>>(oldNode)
        if stick%0b10 == 0b1  then
          oldIndice=i
        end
      end
    end
    if oldIndice==-1 then
      #If nothing is found, leave the loop
      return
    end
  end

  stick=orientedIntersection[currentNode]

  test2=1
  t=0
  counNode=0
  #Scan the oriented intersection the old node
  while test2==1 do
    if  stick%0b1000==(oldIndice+1) then
      test2=0
    end
    counNode++
    stick = stick >> 3
    if counNode>4 then
      return
    end
  end

  #Compute the exit taken during last intersection
  counterContinue=counterContinue+2*countExit
  if  counterContinue==0 then
    counterContinue=2
  end

  #Scan the intersection until reaching the right number of exit taken by the robot in it
  countExit=1
  for i in 0:4 do

    if  counNode==5 then
      stick=orientedIntersection[currentNode]
      counNode = 0
    end
    if  stick%0b1000!=0b000 then

      if countExit==(counterContinue/NbCont) then

        newNode=nodeTab[stick%0b1000-1]
        return
      end
      countExit++
    end
    counNode++
    stick = stick >> 3
  end


###Routine of map management
sub mapUpdate
#Called when there is a modification of node(Edit error detection if two roundabout are next to each other)
  callsub searchCompatibleNode

  oldNode = currentNode
  currentNode = newNode
  if (maptype[newNode]%0b100) == newNodeType then
  #LEDs to indicate node type
    #if currentNode==0 then
      #call leds.top(0,32,0)
    #elseif currentNode==1 then
      #call leds.top(32,0,0)
    #else
      #call leds.top(0,0,32)
    #end
  else
  #Special case 2 (see section 4.3 in Benjamin's report)
    if maptype[newNode]%0b100==0b11 then
        counterContinue=2
        countExit=0
      callsub searchCompatibleNode
      oldNode = currentNode
      currentNode = newNode
    else
      positionError = 1
    end
  end
  counterContinue = 0
  countExit = 0

#Start the robot in a location before the barcode
onevent button.forward
  if  button.forward == 1 then
    speedType = SPEED_LOW
    state = S_EXIT
    actionType = A_LOCATION
    readLocation = 1
    decisionState = DS_STOP
    counter = 60
    call prox.comm.enable(1)
  end

#Stop the robot
onevent button.center
  if  button.center == 1 then
    motor.left.target = 0
    motor.right.target = 0
    state = S_STOP
    #comm_state=CS_EAR
    #last_ok =11
    #i=0
    #prox.comm.tx=0
  end

onevent prox
#if the robot is not stopped or across another road, it follows the line
  if state != S_STOP and state != S_CONTINUE then
    intensityDiff = prox.ground.delta[0]-GREY_REF
    #The robot is given a reference mean value. Its speed is controlled to stay as close as possible
    #to this value with a P controller. The gain (dividing factor) depends on the state.
    if  state == S_ROAD or state == S_BACK then
      controlFactor = 3
    elseif  state == S_ACTION or state == S_EXIT then
      controlFactor = 2
    elseif  state == S_READ then
      controlFactor = 6
    end
    addSpeedLeft = intensityDiff/controlFactor
    addSpeedRight = -intensityDiff/controlFactor

    #Obstacle avoidance
    if  (prox.horizontal[1] > OBS_PROX_THRESH or prox.horizontal[2] > OBS_PROX_THRESH or
        prox.horizontal[3] > OBS_PROX_THRESH) and (state == S_ROAD or state == S_ACTION or state == S_EXIT)
         then
      #Collect the maximum intensity detected
      call math.max(intensityObst,prox.horizontal[2], prox.horizontal[3])
      call math.max(intensityObst,intensityObst, prox.horizontal[1])
      intensityDiffObst = intensityObst - OBS_PROX_THRESH
      #Reduce the speed proportionally to the intensity.
      if intensityDiffObst >= 0 then
        motor.left.target = speedType+addSpeedLeft-intensityDiffObst/(75/(1+speedType/SPEED_LOW))
        motor.right.target = speedType+addSpeedRight-intensityDiffObst/(75/(1+speedType/SPEED_LOW))
      end
      #Above the threshold, stop
      if intensityObst > OBS_PROX_THRESH_MAX then
        motor.left.target = 0
        motor.right.target = 0
      end
    #Obstacle avoidance when going backward. If there is something, stop (no adaptation of speed
    elseif  (prox.horizontal[5] > OBS_PROX_THRESH_MAX or prox.horizontal[6] > OBS_PROX_THRESH_MAX) and state ==
           S_BACK then
        motor.left.target = 0
        motor.right.target = 0
    #No obstacle avoidance
    else
      motor.left.target = speedType + addSpeedLeft
      motor.right.target = speedType + addSpeedRight
      #If there is a too big difference, turn on itself (only on white part of gradient)
      if abs intensityDiff > 300  then
        motor.left.target = intensityDiff/4
        motor.right.target = -intensityDiff/4
      end
    end
  end

  #Reading synchro bit + security to be sure not to read the black side of the road
  if state == S_ROAD and  prox.ground.delta[1] < GREY_THRESH and prox.ground.delta[0] < white_threshold then
    #If there is already someone reading the barcode, go backward.
    if  (prox.horizontal[1] > OBS_PROX_THRESH or prox.horizontal[2] > OBS_PROX_THRESH or
    prox.horizontal[3] > OBS_PROX_THRESH) then
        state = S_BACK
        timer.period[0] = 1200
        speedType = -SPEED_LOW
        timerON = 1
      #Go to the reading state.
      else
      oldState = state
      state = S_READ
      counterRead = 0
      code = 0
     end
  end

  #Reading barcode
  if state == S_READ then
    #Interval depending on the speed
    interval = 2*SPEED_HIGH/speedType #count between two readings
    counterRead++
    #First time wait 1,5 interval to ben in the middle of the next bit
    if  counterRead == 1+(3*interval-1)/2 then
      if prox.ground.delta[1] < GREY_THRESH then
        code +=1
      end
    elseif  counterRead == 1+(5*interval-1)/2 then
      if prox.ground.delta[1] < GREY_THRESH then
        code +=2
      end
    elseif  counterRead == 1+(7*interval-1)/2 then
      if prox.ground.delta[1] < GREY_THRESH then
        code +=4
      end
    #Wait 1 additional interval to be sure to be across the barcode
    elseif counterRead == 1+(9*interval-1)/2 then
      state = oldState
      #Change state corresponding to the code read
      if  state == S_ROAD then
        if code == 1 then
          speedType=SPEED_LOW
        elseif  code == 2 then
          speedType=SPEED_MED
        elseif  code == 3 then
          speedType=SPEED_HIGH
        elseif code == 4 then
          savedSpeed = speedType
          speedType=SPEED_LOW
          state = S_ACTION
          actionType = A_BRANCHING
          newNodeType = 0b11
          callsub mapUpdate
        elseif code == 5 then
          savedSpeed = SPEED_LOW
          speedType=SPEED_LOW
          state = S_ACTION
          actionType = A_CROSSROAD
          call math.rand(random)
          decisionState = random%3
          newNodeType = 0b10
          callsub mapUpdate
        elseif code == 6 then
          savedSpeed = SPEED_LOW
          speedType=SPEED_LOW
          state = S_ACTION
          actionType = A_LOCATION
          newNodeType = 0b00
          callsub mapUpdate
        elseif code == 7 then
          savedSpeed = speedType
          speedType=SPEED_LOW
          state = S_ACTION
          actionType = A_ROUNDABOUT
          newNodeType = 0b01
          callsub mapUpdate
        end

      elseif readLocation == 1 then
        #Actual position used to reset the map in real time
        currentNode=code+7
      end
    end
  end

  #Intersection
  if state == S_ACTION then
    #Crossroad
    if  actionType == A_CROSSROAD then
    #decisionState =DS_EXIT
      #After 4 Continue, the robot is out and goes back to road state
      #The 4th continue starts directly after the 3rd one
      if counterContinue == 3 then
        state = S_CONTINUE
         counterContinue++
         counter = 0
      elseif  counterContinue == 4 then
        state=S_ROAD
        speedType = savedSpeed
      end
    else
      #Random choice when reading bit
      call math.rand(random)
      decisionState = random%2
      #Location and Branching
      #If continue was chosen, return to road state after two readings
      if (actionType == A_BRANCHING or actionType == A_LOCATION) and counterContinue == 2 then
          state=S_ROAD
          speedType = savedSpeed
      #Roundabout
      else
        #Exit is possible only on even number of readings
        if decisionState == DS_EXIT  and counterContinue%2 == 0 then
          nbRond++
        else
          decisionState = DS_CONTINUE
        end
      end
    end
  end

  #Reading a bit + security and reading a code in a location
    if (state == S_ACTION or (state == S_EXIT and readLocation == 1)) and prox.ground.delta[1] < GREY_THRESH
      and prox.ground.delta[0] < white_threshold then
    if readLocation == 1 then
      oldState = state
      state = S_READ
      counterRead = 0
      code = 0
    else
      counter = 0
      #Change state
      if  decisionState == DS_EXIT then
        state = S_EXIT
      else
        state = S_CONTINUE
          counterContinue++
         #In case of a crossroad and turning left, if 2 continue done, force exit decision state to follow
         #line
         if  (actionType == A_CROSSROAD and decisionState == DS_TURN_LEFT and counterContinue == 2) then
          decisionState = DS_EXIT
        end
      end
    end
  end

  #Exit state
  if state == S_EXIT then
    counter++
    #In a location, prepare to stop
    if actionType==A_LOCATION then
      if prox.ground.delta[1] > GREY_THRESH then
        readLocation = 1
      end
      decisionState=DS_STOP
    end
    #Counter condition for other intersections
    if  (counter > 41 and (actionType == A_ROUNDABOUT or actionType == A_BRANCHING))
      or (counter > 29 and actionType == A_CROSSROAD) then
      #If decision was to turn left, use exit state to follow line, then goes to action state (and not road)
      countExit++
      if  actionType == A_CROSSROAD and counterContinue != 0 then
        state = S_ACTION
        decisionState = DS_TURN_LEFT
      else
        #Goes back to road state excep for location intersection
        if decisionState!=DS_STOP then
          state = S_ROAD
          speedType = savedSpeed
        end
      end
    end
  end

  #Cross a road or cross a part without gradient
  if state == S_CONTINUE then
    counter++
    #Due to placement after a left turn, the robot is never going straight. Therefore add a bias to make it
    #turn slightly to the left, depends on intersection
    motor.left.target = (85+actionType*3)*speedType/100
    motor.right.target = speedType

    #Obstacle avoidance. If there is something, stop counter and stop moving
    if (prox.horizontal[1] > OBS_PROX_THRESH or prox.horizontal[2] > OBS_PROX_THRESH or
        prox.horizontal[3] > OBS_PROX_THRESH)  then
        savedSpeedInters = speedType
        motor.left.target = 0
        motor.right.target = 0
        counter--
        counterStop = 1
    #No more obstacles
    elseif  counterStop == 1 and (prox.horizontal[1] < OBS_PROX_THRESH and prox.horizontal[2] < OBS_PROX_THRESH
    and prox.horizontal[3] < OBS_PROX_THRESH) then
      counterStop = 0
      speedType = savedSpeedInters
    end

    #Goes back to action state
    if  counter > 21  then
      state = S_ACTION
    end
  end

  #Location stop
  if decisionState == DS_STOP then
    if ((prox.horizontal[1] > OBS_PROX_THRESH_MAX-100 or prox.horizontal[2] > OBS_PROX_THRESH_MAX-100 or
        prox.horizontal[3] > OBS_PROX_THRESH_MAX-100))  then
        counter--
    end
    if counter > 130 then
      #Increment only if traffic light is green
      counter=counter + incrementCounter
      state = S_STOP
      motor.left.target = 0
      motor.right.target = 0
      #After a while goes back to road state
      if counter > 150 then
        decisionState = DS_NOTHING
        speedType = savedSpeed
        motor.left.target = speedType
        motor.right.target = speedType
        readLocation = 0
        state = S_ROAD
      end
    end
  end

  #LEDs to indicate state
  if state == S_ROAD then
    call leds.top(32,(3-((speedType-SPEED_LOW)/SPEED_LOW))*8,0)
  elseif  state == S_READ then
    call leds.top(32,32,32)
  elseif state == S_ACTION then
    call leds.top(0,10*actionType,10*(3-actionType))
  end

#Communication
onevent prox.comm
#  Bidirectionnal communication
# if comm_state==CS_TELL then
#   if  prox.comm.rx==last_ok then
#     #call leds.top(0,0,32)
#     if  last_ok==10 then
#       last_ok=11
#     else
#       last_ok=10
#     end
#     if i==1 then
#       prox.comm.tx=currentNode+60
#     elseif i==2 then
#       prox.comm.tx=oldNode+60
#     else
#       prox.comm.tx=currentNode + 100
#
#       timer.period[1]=2000
#         comm_state=CS_EAR
#         last_ok =11
#
#
#     end
#     count++
#     i++
#   end
#
# end
  if comm_state==CS_EAR then
    if maptype[currentNode]==0b10  then
      if prox.comm.rx==oldNode+20 and counterContinue==0 and countExit == 0 then
        callsub allStop
      else
        if  prox.comm.rx==oldNode+40 then
           callsub allStart
        end
      end
    end
    if counter < 150  then
      if maptype[currentNode]==0b00 and counterContinue == 0 and prox.comm.rx == currentNode+20 then
        callsub allStart
        incrementCounter = 1
      elseif maptype[currentNode]==0b00 and counterContinue == 1 and  prox.comm.rx == currentNode+20 then
        callsub allStop
        incrementCounter = 0
      elseif  maptype[currentNode]%0b100==0b00 and counterContinue==0 and prox.comm.rx == currentNode+40 then
        callsub allStop
        incrementCounter = 0
      elseif  maptype[currentNode]%0b100==0b00 and counterContinue==1 and prox.comm.rx == currentNode+40 then
        callsub allStart
        incrementCounter = 1
      end
    end
#   Code for the communication state machine
# elseif prox.comm.rx==1 then
#     comm_state=CS_TELL
#     prox.comm.tx=3
#     i=0
# Code for the Ping part
#   elseif prox.comm.rx>100 then
#     timer.period[1]=2000
#     if  countPing==1 then
#       prox.comm.tx=prox.comm.rx+20
#       countPing=2
#     else
#       prox.comm.tx=prox.comm.rx
#     end
#
  end

#Timer used to go backward
onevent timer0
  if timerON == 1  then
    speedType = SPEED_LOW
    state = S_ROAD
    timerON = 0
  end

#Timer used for communication
onevent timer1
  if  comm_state==CS_EAR then
    prox.comm.tx=0
    timer.period[1]=0
  end
  countPing = 1
  </node>

</network>
```

B Calibration code

```
<!DOCTYPE aesl-source>
<network>


<!--list of global events-->


<!--list of constants-->
<constant value="350" name="GREY_REF"/>
<constant value="300" name="BLACK_THRESH"/>
<constant value="150" name="SPEED_LOW"/>
<constant value="260" name="SPEED_MED"/>
<constant value="410" name="SPEED_HIGH"/>
<constant value="65" name="COUNT_REF_LOW"/>
<constant value="38" name="COUNT_REF_MED"/>
<constant value="24" name="COUNT_REF_HIGH"/>


<!--show keywords state-->
<keywords flag="true"/>


<!--node thymio-II-->
<node nodeId="1" name="thymio-II">var intensityDiff=0
var speedType = SPEED_LOW
var counter = 0
var stop = 1
var startCount = 0

var speed_l = 0
var speed_m = 0
var speed_h = 0
var grey_ref
var code_ref

#Start moving
onevent button.forward
  if  button.forward == 1 then
    counter = 0
    stop = 0
    startCount = 0
  end

#Stop moving
onevent button.center
  if  button.center == 1 then
    stop = 1
    motor.left.target = 0
    motor.right.target = 0
  end

onevent prox
  #Get the values measured by the sensors
  grey_ref = prox.ground.delta[0]
  code_ref = prox.ground.delta[1]

  if stop == 0 then
    #Line following
    intensityDiff = prox.ground.delta[0]-GREY_REF
    motor.left.target = speedType+intensityDiff/3
    motor.right.target = speedType-intensityDiff/3
    #Count while on white
    if prox.ground.delta[1] > BLACK_THRESH then
      counter = counter+1
      startCount = 1
    end
  end
  #LEDs to indicate state
  if prox.ground.delta[1] > BLACK_THRESH then
    #Sensor on white
    call leds.top(0,32,0)
  else
    #Sensor on black
    call leds.top(32,0,0)
    #If the robot has started calibration, stops on black
    if  startCount == 1 then
      stop = 1
      motor.left.target = 0
      motor.right.target = 0
      #Compute speeds
      speed_l = SPEED_LOW*counter/COUNT_REF_LOW
      speed_m = SPEED_MED*counter/COUNT_REF_LOW
      speed_h = SPEED_HIGH*counter/COUNT_REF_LOW
    end
  end</node>


</network>
```

C Lightpainting code

More informations are available on: https://www.thymio.org/en:thymiolightpainting

```
<!DOCTYPE aesl-source>
<network>


<!--list of global events-->
<event size="2" name="data"/>
<event size="2" name="motors"/>


<!--list of constants-->
<constant value="0" name="S_WAIT_SYNC"/>
<constant value="1" name="S_READING"/>
<constant value="5" name="INTERVAL"/>
<constant value="650" name="BW_LIMIT"/>
<constant value="32" name="ON"/>
<constant value="0" name="OFF"/>


<!--show keywords state-->
<keywords flag="true"/>


<!--node thymio-II-->
<node nodeId="1" name="thymio-II">### VARIABLES DECLARATION ###
var running = 0
var counter = 0
var state
var intensityDiff
var code
#Colors tables
var r[]=[OFF,OFF,OFF,OFF,ON,ON,ON,ON]
var g[]=[OFF,OFF,ON,ON,OFF,OFF,ON,ON]
var b[]=[OFF,ON,OFF,ON,OFF,ON,OFF,ON]

call leds.top(r[code], g[code], b[code])
#0=colorless,1=blue,2=green,3=lightblue,4=red,5=darkpink,6=yellow,7=purple

### TURNING OFF THE LEDs ###
call leds.prox.v(0,0)
call leds.prox.h(0,0,0,0,0,0,0,0)
call leds.top(0,0,0)
call leds.temperature(0,0)

### EVENT BUTTON FORWARD ###
onevent button.forward
# starts line tracking and waits for sync bit
if button.forward == 1 then
  running = 1
  state = S_WAIT_SYNC
end

### EVENT BUTTON CENTER ###
onevent button.center
#stops running and motors, turns LEDs off
if  button.center == 1 then
  running = 0
  motor.left.target = 0
  motor.right.target = 0
  call leds.top(0,0,0)
end

### EVENT PROXIMITY SENSORS (EVERY 100 ms => 10 Hz)###
onevent prox
#if the robot is running, steering and bar code reading are activated
#we use the prox.ground.delta[1] to steer, the ...[0] to check the bar code

  if running == 1 then

  ### STEERING PART ###
  intensityDiff = prox.ground.delta[1]-525
  #speed of robot is adjusted to keep around a gray mean value. If
  #difference is too important, the robot spins until it finds the line
  #again. The steering is controled by the gray gradient of the trail.
  if abs(intensityDiff) < 170 then
    motor.left.target = 115+intensityDiff/8
    motor.right.target = 115-intensityDiff/8
  else
    motor.left.target = intensityDiff/4
    motor.right.target = -intensityDiff/4
#edit to stop colours if line is lost
    state = S_WAIT_SYNC
  end

  ### BAR CODE CHECKING PART ###
  #if Thymio is waiting for sync and crosses a black line, it starts bar
  #code reading and turns LEDs off.
  if state == S_WAIT_SYNC and prox.ground.delta[0] < BW_LIMIT then
    state = S_READING
    call leds.top(0,0,0)
    counter = 0
    emit data [prox.ground.delta[0],1000]
  end
  emit data [prox.ground.delta[0],0]
  #if state is reading, the robot reads the bar code
  if state == S_READING then

    #we wait to count to 2 times the timer = 100 ms (counter is
    #increment at the end of the code) so we are in the middle of the
    #black sync box
    if counter == 2 then
      code=0
    #we wait the first two counter + one INTERVAL (4 * 50ms) to measure
    #the first block of bar code. This can be tuned to match with the
    #speed of the robot
    elseif counter == (2+INTERVAL) then
    emit data [prox.ground.delta[0],1000]
    #if bit is white set code to 1  (c0=001b)
      if  prox.ground.delta[0]>BW_LIMIT
      then code=1
      else code=0
      end
    #test again after another INTERVAL
    elseif counter == (2+2*INTERVAL) then
    emit data [prox.ground.delta[0],1000]
      #if bit is white add 2^1 to code (c1=010b)
      if  prox.ground.delta[0]>BW_LIMIT
      then code +=2
      end
    elseif counter == (2+3*INTERVAL) then
    emit data [prox.ground.delta[0],1000]
      #if bit is white add 2^2 to code (c2=100b)
      if  prox.ground.delta[0]>BW_LIMIT then code +=4
      end

      #LEDs are turned on according to binary code (c2 c1 c0)
      call leds.top(r[code],g[code],b[code])
    #again another INTERVAL and we go back to S_WAIT_SYNC to wait for
    #the next bar code
    elseif counter == (2+4*INTERVAL+3) then
      state = S_WAIT_SYNC
    end
    counter += 1
  else
    call leds.circle(0,0,0,0,0,0,0,0)
  end
end</node>


</network>
```