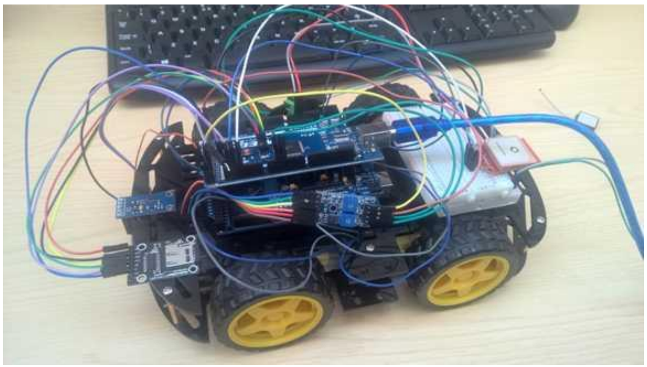
2nd Year Internship Report
–
University of Plymouth



FEASABILITY STUDY: Control of a mobile robot

Sophie TUTON Tutor: Jian Wan

2018 Promotion

June 8 2017 - August 30 2017

Abstract

English

This feasibility study consists in the creation of a mobile robot with two Arduino Mega boards which one is wired with different sensors and another with motors. They communicate together.
To reach a target, robot localises itself in indoor and outdoor situation with some encoders, a gyroscope and a compass or a GPS, thanks to an observer like a dead reckoning or a Kalman Filter. The heading is known thanks to a data fusion of the gyroscope and the compass. The robot is guided by an artificial potential algorithm used with ultrasonic sensors. Then, the robot moves with two proportional-integrative-derivate controllers.
Whereas the line following is done with infrared sensors and uses specific order.

Some tests have been performed in indoor situation to check the functioning of the system. The localisation had needed outdoor test of different algorithms. Tests have been performed with video processed by Tracker and with a recording data via an SD card.

In each mode, the study shows that the creation of a robot which can reach a target in simple situations and avoid obstacles is feasible with low cost sensors. However, safety needs more expensive sensors, notably to reduce sample time and execution time to improve the reactivity of the robot. The line following works correctly but in simple situations with small turn. In complex situations, it may leave the line a long time before find again it. So, it could be interesting to create a more complex command.

Français

Cette étude de faisabilité consiste à crée un robot mobile à partir de deux cartes Arduino Mega dont une d'entre elle est reliée à différents capteurs et l'autre aux moteurs. La première carte communique les informations nécessaires à la seconde carte pour atteindre la cible en évitant les obstacles ou suivre la ligne.
Pour atteindre une cible, les capteurs utilisés permettent de localiser le robot en intérieur et en extérieur à l'aide des odomètres, d'un gyroscope et d'un compas, ou d'un GPS, via un observateur de marche aveugle ou de filtre de Kalman. Le cap du robot est obtenu via une fusion des données du gyroscope et du compas. Le rover est guidé par un algorithme de potentiel artificiel lui permettant également d'éviter les obstacles vus par des capteurs à ultrasons. Puis un contrôleur doté de deux commandes proportionnelles - intégrales – dérivées (PID) permet l'exécution du guidage.
Tandis que le suivi de ligne n'as été abordé qu'avec des capteurs infrarouges placés sous le robot. Des commandes spécifiques sont alors déduites des données capteurs.

Des tests ont été effectués en intérieur pour vérifier le fonctionnement de chaque système. Seule la localisation a nécessité des tests en extérieur et une comparaison entre différents algorithmes en fonction de la situation. Les tests sont menés à l'aide de vidéos traitées ou non par le logiciel Tracker et de l'enregistrement des données via une carte SD.

Dans les deux situations, l'étude montre qu'il est faisable de faire un robot qui est capable d'atteindre une cible dans des conditions simples tout en évitant des obstacles assez espacés à l'aide de capteurs à faible coût. Cependant, pour que ce soit exécuté avec sécurité, il est important de prendre des capteurs à plus haut cout. En effet, cela permettrait d'augmenter l'échantillonnage et d'exécuter les algorithmes plus régulièrement pour augmenter la réactivité de robot.
Le suivi de ligne s'effectue correctement mais dans des circuits complexes avec de fort tournant le

robot quittera longtemps la ligne avant de la retrouver tant bien que mal, c’est pourquoi il serait intéressant de faire varier les commandes de manière plus intelligente en fonction du temps d’absence de ligne.

Table of contents

Introduction

The autonomous car is a current challenge of the society. This challenge is made to improve the road safety, contribute to more fluid traffic and to enhance the user comfort. The main point is to enhance the road safety thanks to an automation of the system.
The aim of this feasibility study, completed at the Plymouth University, was to make a rover which can either follow a line only thanks to infrared sensors or reach a target with an obstacle avoidance. For this, a frame, some low-cost sensors and Arduino components were provided. Another student worked on this project – Nicolas VEYLON -, but only the work I done is explained in this report.

This report is composed of four parts, the first one is a presentation of the hardware used with each used components and sensors. Then, the model chosen for a simulation. After, the first mode of the rover is described through the controller, the observer and the guidance used. And finally, the second mode –the follower line- is described.

I. Hardware part

I.A. The global system

The rover needs not only infrared sensors to follow a line but also:

1. A GPS Module to localise itself,
2. Encoders to compute its velocity,
3. A compass and a gyroscope to measure its heading and the derivate of the heading,
4. Ultrasonic sensors to avoid obstacles.

Data from sensors must be recorded. For this a MicroSD broad is used. An Arduino Mega board wired with the sensors collects data and sends some of them to another Arduino Mega board. This last processes data and feeds the motor controller. This system is described in the following figure with sensors in orange, the SD card in grey, both Arduino Mega are in blue, the battery, the motor controller and the motors are in green.

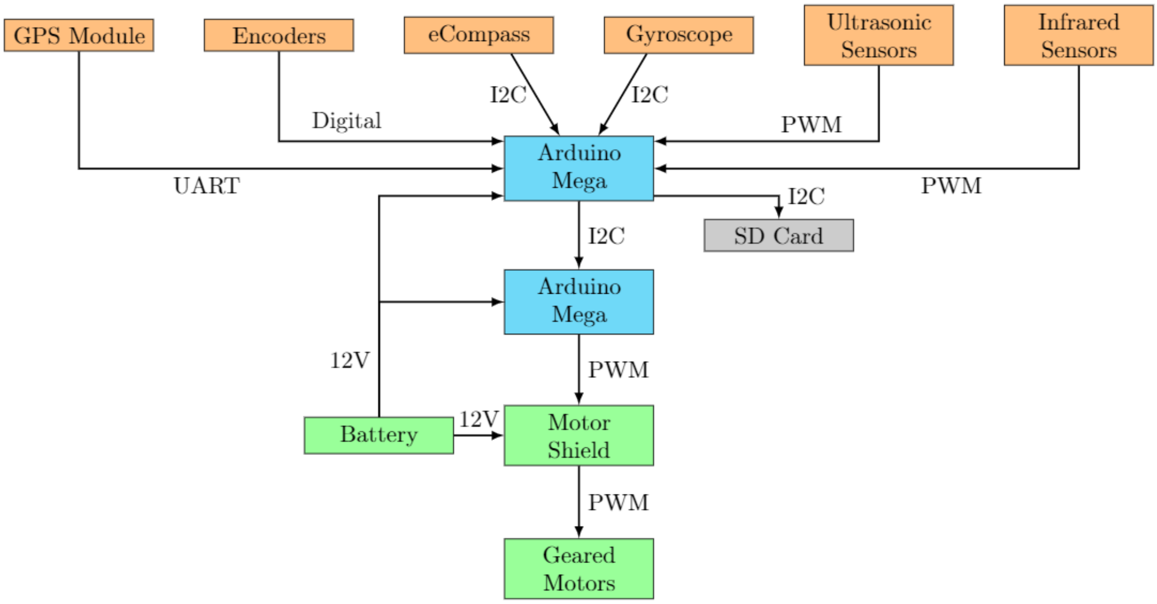


Figure 1: Components and sensors used in the robot

I. B. Sensors and components used

Each sensor had been studied with a task repartition. So, only some sensors had been deeply studied. In this part, there are sensors allocated to me. Data are recorded thanks to an SD card[1] , and some data can be plotted thanks to an I2C LCD[2].

[1] See Annex – Components details, Micro SD board.

[2] See Annex – Components details, I2C LCD.

I.B.1. Microcontroller [1]

The microcontroller used is the ATMega256 on the Arduino Mega 2560. This microcontroller board allows the fusion of a lot of sensors thanks to 54 digital input (but also output) pins operating at 5Volts, and 16 analogue input pins which provide a 10 bits resolution. Moreover, 15 pins can deliver 8-bits PWM signals, 6 can be used to interrupt functions[3]. Serial communications through UARTs (Universal asynchronous receiver/transmitter) and I2C (Inter-Integrated Circuit) communications are allowed by 8 pins and 2pins respectively[4].
The input voltage recommended for this microcontroller board is between 7V and 12V, and the input voltage limit is between 6V and 20V. Two modes of power can be used: power by USB connection, and external power supply (wall-wart adapter or battery).

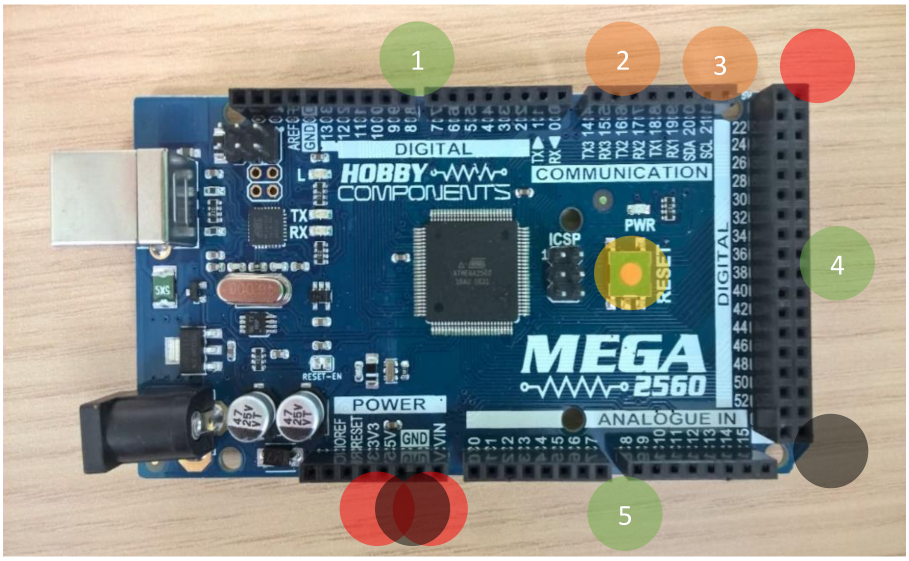


Photo 1: An Arduino Mega board

Key:

1 **PWM**

2 **Serial communications**

3 **I2C**

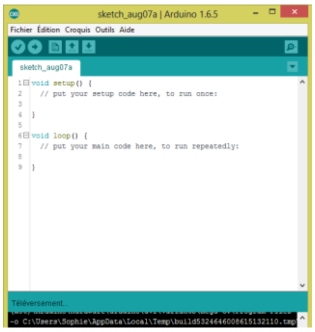
4 **Digital inputs/out-**

5 **Analogue inputs**

● **GND pins**

● **Power**

● **Analogue inputs**


sketch_aug07a | Arduino 1.6.5
Fichier Édition Croquis Outils Aide
sketch_aug07a
Téléversement...

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

```
-o C:\Users\Sophie\AppData\Local\Temp\build5324646008615132110.tmp
```

Photo 2: The IDE

There are 256 KB of flash memory for storing the code. The C++ source code is compiled and then uploaded thanks to an IDE (Integrated Development Environment) provided by Arduino.

[3] See Annex – Components details, Arduino board, Interrupt pins
[4] See Annex – Components details, Arduino board, Communication

I.B.2. Motor Shield

I.B.2.a Description [2]

The shield used is composed of two servomotor interfaces, and can control four DC motors or two stepper motors. The voltage recommended for a DC motor is between 4.5V and 36V. Some pull down resistors deactivate motors during the power up. The power is distributed for the motor – high power – and for the logical part – low power – if the jumper is putted. It uses two motor drivers[5] (L293D) and one shift register for its functions (74HC595).

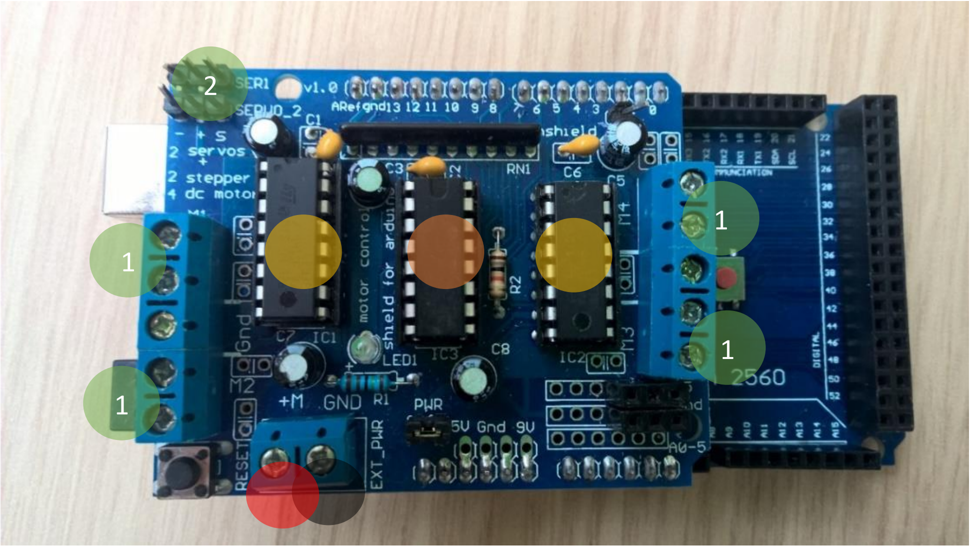


Photo 3: The motor shield

1	**Motor port**	●	**Motor driver chips**	●	**GND pins**
2	**Servomotor**	●	**Shift register**	●	**Power pins**

I.B.2.b. Arduino Function

The library used is *AFMotor* by Adafruit [4]. Motors are “AF_ DCMOTOR” objects and speeds are defined by a number between 0 and 255.

The objects relatives to motor are used in the different functions of motion: *turnLeft*, *turnRight*, *goForward*, *goBackward*, *defineVelocity*, *defineVelocityRight* and *defineVelocityLeft* are the created functions[6].

In this functions *v* is the command for the linear velocity, *delta* is the half of the rotation command, *motor.run( )* takes in argument FORWARD or BACKWARD and makes motors run, *way* is the direction of wheels’ rotation (Forward or Backward). It depends on the command sign: if it is negative the wheels go backward, else they go forward.

[5] See Annex – Components details, Motor driver functioning.

[6] See them in the page 53-57 of the Project Code.

I.B.3. Encoders

I.B.3.a. Description

The FC-02 rotary encoder is used to measure the speed of the robot. It is an optical sensor using an index wheel linked to the motor shaft. This index wheel is composed of many holes. An IR light is emitted by a LED, and if the light goes through a hole, it reaches the receiver (HIGH output). However, if the light is intercepted by the wheel the receiver detects nothing (LOW output).

I.B.3.b. Velocity Computing and Accuracy

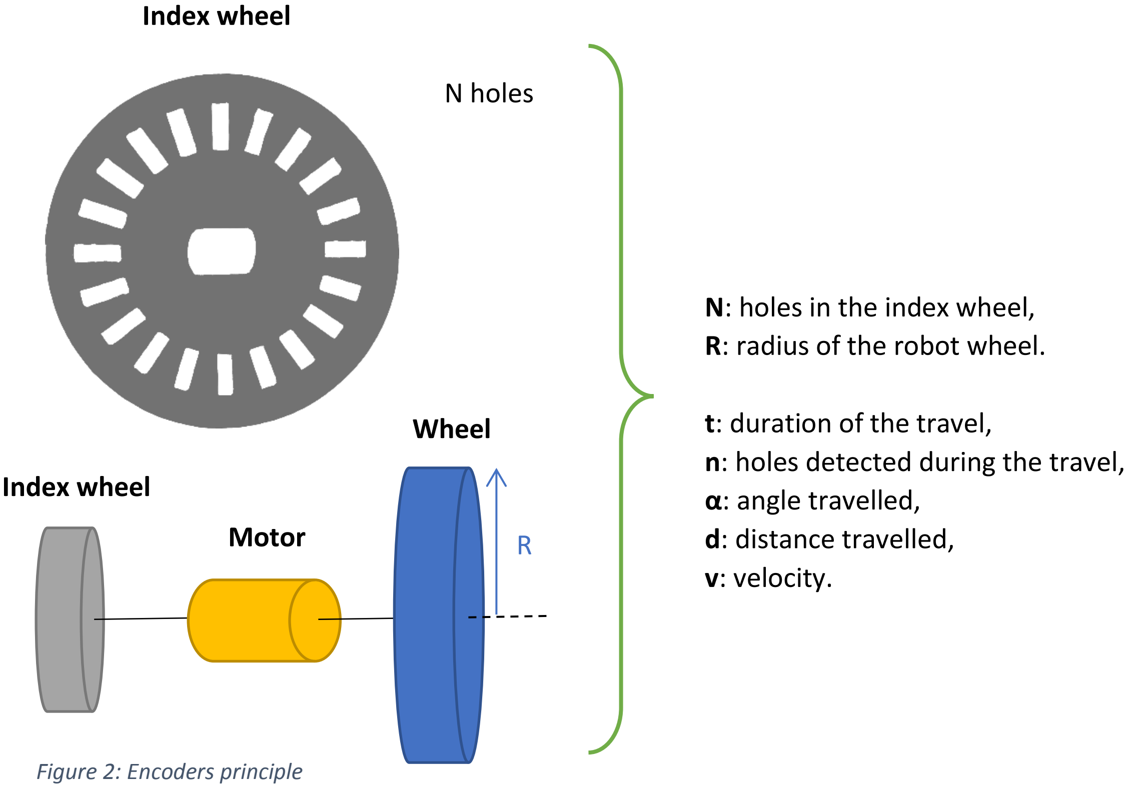

Index wheel
N holes
Wheel
Index wheel
Motor
R
N: holes in the index wheel,
R: radius of the robot wheel.
t: duration of the travel,
n: holes detected during the travel,
α: angle travelled,
d: distance travelled,
v: velocity.

Figure 2: Encoders principle

In t seconds, the angle travelled is $\alpha(t) = \frac{n(t)\cdot 2\pi}{N}$, so the distance travelled is $d(t) = \alpha(t) \cdot \mathrm{R} = \frac{n(t)\cdot 2\pi \cdot R}{N}$. Moreover $v = \frac{d(t)}{t}$. Finally, $\boldsymbol{v} = \frac{\boldsymbol{n(t)\cdot 2\pi\cdot R}}{\boldsymbol{N\cdot t}}$. **(I.1)**

To measure the instantaneous velocity, we must compute with a small t. To have more accuracy, each edge is considered and the number n(t) – now the number of edge detected – is divided by 2 in the equation.

As there are 20 holes, there is an edge each $\frac{2\pi}{40} \approx 0.16rad$.

So, $\Delta \mathrm{d} \approx \frac{2\pi}{40} \cdot \mathrm{R} \approx \frac{2\pi}{40} \cdot 3.4 \approx 0.534 \ \ \mathrm{cm}$.

But $v = \frac{d}{t}$,

So, for a time step of 150ms v is known $\pm\, 3.3cm/s$, whereas for 500ms it is known $\pm\, 1cm/s$ and for 1s it is known $\pm\, 0.5cm/s$. A huge step increases the accuracy of the mean velocities but hide the acceleration.

Each velocity side is computed thanks to an encoder, and the velocity of the car is got by the mean of these velocities. Indeed, when the car is turning, the situation is the following:

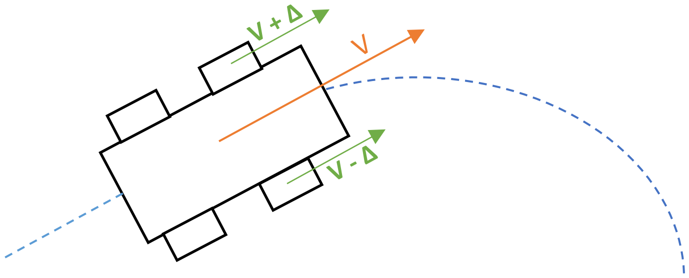


Figure 3: The velocity of the robot

After some tests, it seems that the sensor has switch bounces. That is why edges are not considered during one millisecond after an edge.

I.B.3.c. Arduino code to compute rover velocity

The car is moving and, in the same time, the velocity is computed using interruption. Functions created and used for computing velocity are the following:

- *countL* and *countR* are called when a change is detected in the OUTPUT of encoders. They count the number of holes detected by the left and right encoders[7].
- *computeVelocity* which computes the velocity of the right and left side of the vehicle[8].

I.B.4. Ultrasonic modules [7]

I.B.4.a. Description

Photo 4: Ultrasonic sensor

An ultrasonic ranging module allows to measure distance to object, from 2cm to 4m and with a measuring angle of 30°. It must be alimented with a voltage of 5V DC, and its working current is about 15mA. It is composed of one ultrasonic transmitter and one ultrasonic receiver.

When the module receives a 5Volts pulse for at least 10µs, 8 periods of a 40kHz wave will be transmitted. When the receiver detects this ultrasound, the Echo pin is set to 5V during a time proportional to the distance.

[7] See page 19, lines 28-64 of the Arduino Project Code.

[8] See page 20, lines 66-85 of the Arduino Project Code. These functions are called in the main program in the page 9 at the lines 213-219 for the initialisation, and in the page 10 at the lines 278 -283 for the processing.

I.B.4.b. The position computing[9]

The situation is the following:

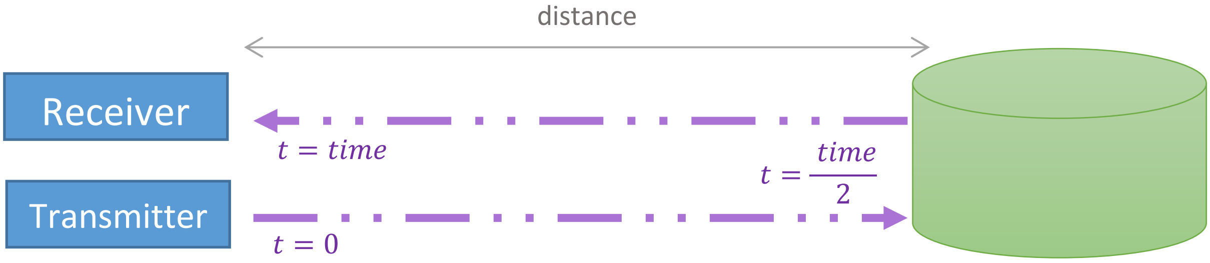


Figure 4: The ultrasonic sensor principle

The wave takes *time* (µs) to travel from the transmitter to the receiver, but it takes $time/2$ to travel from the transmitter to the reflecting object. Therefore, we have the following equation:

$$distance = \frac{time \cdot velocity}{2}$$

With:

- *time* (µs), the time between the transmission and the reception,
- *velocity* (cm.s^{-1}), the velocity of ultrasound in air, i.e. 340m.s^{-1},
- *distance* (cm), the distance from the robot to the object.

Finally, $\boldsymbol{distance = time \cdot \frac{0.034}{2}}$ **.** The Arduino code computes the distance of obstacles in three directions. That is why three ultrasonic sensors are used in front of the rover as following:

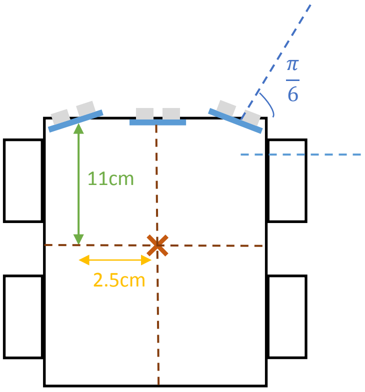


Figure 5: The ultrasonic sensor position on the robot

[9] Functions made are in the pages 22 – 25 of the Arduino Project Code. These functions are called in the main program in the page 10 at the lines 255-252 for the initialisation. And in the page 11 at the lines 299-300 for the reading and processing.

So, to have the position $\begin{bmatrix}x_{obs}\\y_{obs}\end{bmatrix}$ of each obstacle in the local map (map centred in the initial position of the rover, its axes are toward the East and the North) the processing is the following:

$$\begin{bmatrix}x_{obs}\\y_{obs}\end{bmatrix} = \underbrace{\begin{bmatrix}x\\y\end{bmatrix}}_{\text{Rover position in the local map}} + \begin{bmatrix}\cos(\varphi) & -\sin(\varphi)\\ \sin(\varphi) & \cos(\varphi)\end{bmatrix} \underbrace{\begin{bmatrix}x_{sonar}\\y_{sonar}\end{bmatrix}}_{\text{Sensor position from the rover}} \cdot + \underbrace{d}_{\text{Obstacle distance from sensor used}} \cdot \begin{bmatrix}\cos(\varphi+\gamma)\\ \sin(\varphi+\gamma)\end{bmatrix}$$

With:

- $\begin{bmatrix}x\\y\end{bmatrix}$ the rover position in the local map,
- φ the absolute heading of the rover,
- $\begin{bmatrix}x_{sonar}\\y_{sonar}\end{bmatrix}$ the sonar position relative to middle of the rover,
- γ the heading of the sonar relative to the rover
- d the distance of the obstacle.

For example, for an obstacle at the left of the rover $\gamma = \frac{\pi}{6}$ and $\begin{bmatrix}x_{sonar}\\y_{sonar}\end{bmatrix} = \begin{bmatrix}11\\2.5\end{bmatrix}$.

I.B.5. Infrared modules

Infrared modules are used to detect the line that a robot must follow. There are four infrared (IR) modules brought together. Each one is composed of one IR LED, one IR receiver, one display LED which glows when an obstacle is detected, and one potentiometer to adjust manually the detection distance which is between 1mm and 60cm.

There are six pins, four outputs which are low when something is detected, one ground, and one power supply pin.

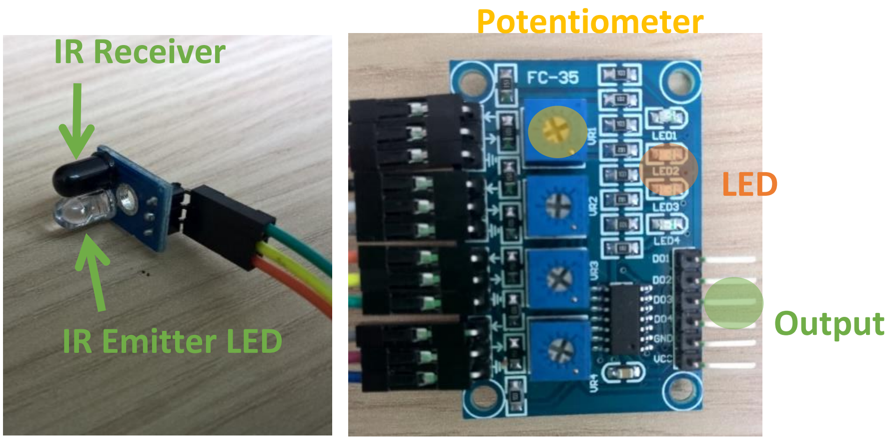


Photo 5: IR sensor

II. Model and Simulation

II.A. Description

To have a great control robot, a dynamic model of the robot must be found and the simulation must be performed. A complete model was found in this research: [9]. It considers the friction and the shear stress.

To validate this dynamic model [10], some motion tests (to go forward with half power, to break…) had been done. During each test, encoders and gyroscope data had been collected. Each test had been recorded to compute the x,y coordinates of the vehicle thanks to a software called Tracker. Each test was compared to the simulation data.

II.A.1 Context

The vehicle is releasing a circular motion about an instantaneous centre of rotation. There are contacts between the road and the wheels, these patches are featured by a grey rectangle in the figure 6. The motion is assumed in a plan (no altitude difference). The following reasoning is for a turn to the left.

II.A.2. Variables

Here are the variables used:

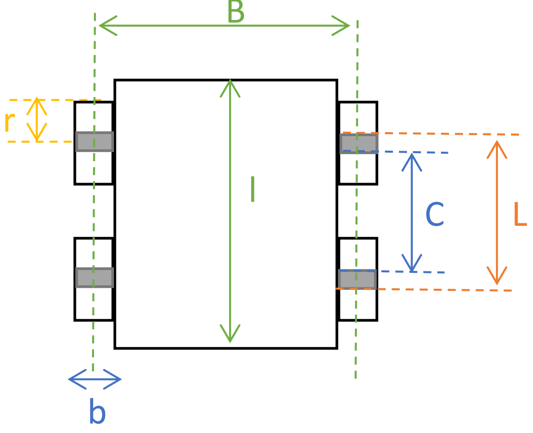


Figure 6: Dimensional constant name

Letter	Meaning	Value or Expression	Source
B	Vehicle width: from a wheel middle to a wheel middle	13.1 cm	Rough measured
L	Vehicle length	25.0 cm	measured
B	Wheel width	2.5 cm	measured
R	Wheel radius	3.4 cm	measured
L	Distance between the beginning of a patch and the end of the patch of the same vehicle side.	12.5 cm	Rough measured
C	Distance between the end of a patch and the beginning of the patch of the same vehicle side.	11.5 cm	Rough measured
m	Vehicle mass	500g	Estimated

I	Moment of Inertia	$I = \frac{m}{12}(l^2 + (B - b)^2)$	Computed in this part
M	Mass matrix, links the coordinates of the vehicle with its kinetic energy [11].	$M = \begin{pmatrix} \frac{mr^2}{4} + \frac{r^2 I}{\alpha B^2} & \frac{mr^2}{4} - \frac{r^2 I}{\alpha B^2} \\ \frac{mr^2}{4} - \frac{r^2 I}{\alpha B^2} & \frac{mr^2}{4} + \frac{r^2 I}{\alpha B^2} \end{pmatrix}$	Computed

Table 1: Dimensional constants

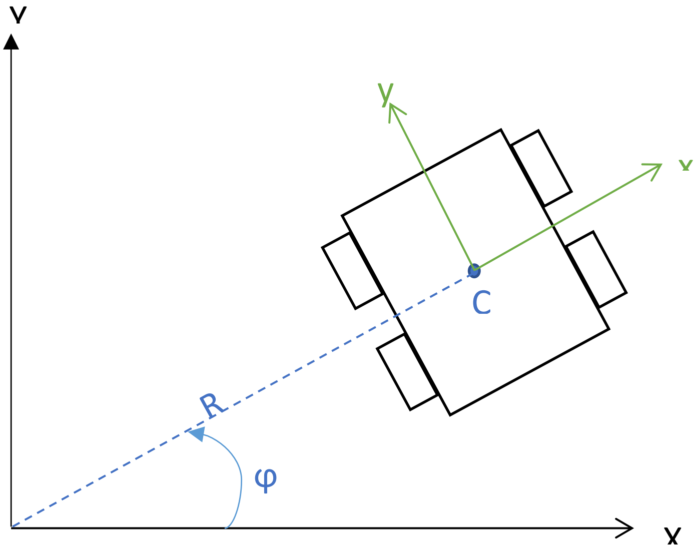


Figure 7: Another constant name

CG = Centre of gravity

Letter	Meaning	Value	Source
α	Terrain-dependant parameter	α = 1.9,	Computed with data recording
R	Radius of curvature		Computed in this part
φ	Angle of the vehicle		
v_y	Velocity in the longitudinal direction	It should be noted that v_x is assumed null.	
ω_l, ω_r	Velocity of the left and right wheels		
$\dot{q}$	[ω_l, ω_r] velocities of wheels		
C(q,q̇)	Resistance term, featured the ground and wheels interactions.	$C = \begin{pmatrix} r \cdot (F_l + f_l) \\ r \cdot (F_r + f_r) \end{pmatrix}$	Computed
G(q)	Gravitational term	Null because of the 2D motion	

Letter	Meaning	Value	Source
F_r,F_l	The sliding longitudinal sliding friction of the right wheels and of the left wheels.		Computed

f_r,f_l	The rolling resistance of the right wheels and of the left wheels	$f_l = f_r = \frac{mg}{2} \cdot \mu_r$	Computed
μ_r	Dimensionless coefficient of the rolling friction.	0.015	[12]
μ	Friction coefficient between tire and regular floor	0.5	[13]
P	Normal pressure	$p = \frac{force}{surface} = \frac{mg}{2b(L-C)}$	Computed
τ_{ss}	The shear stress	$\tau_{ss} = p\mu \cdot (1 - e^{-\frac{j}{K}})$	
j_{rr},j_{fr},j_{rl},j_{fl}	The shear displacement for each wheel	$j_{X_1X_2} = \int_0^t v_{X_1X_2\_X}\, dt$	
(x_{rr}, y_{rr}) (x_{fr},y_{fr}) (x_{rl}, y_{rl}) (x_{fl},y_{fl})	The coordinates of a point in patch of the rear right wheel, front right wheel, rear left wheel, and front left wheel.		
γ_{rr},γ_{fr},γ_{rl},γ_{fl}	The angle of the sliding velocity vector for each wheel	$\gamma_{X_1X_2} = \text{arctan2}(\frac{v_{X_1X_2\_Y}}{v_{X_1X_2\_X}})$	
v_{rr},v_{fr}, v_{rl},v_{fl}	Sliding velocity vectors	$v_{X1X2} = \begin{pmatrix} -y_{X1X2} \cdot \dot{\varphi} \\ \left(R \pm \frac{B}{2} + x_{X1X2}\right) \cdot \dot{\varphi} - r\omega_r \end{pmatrix}$	
K	The shear deformation modulus	5.1×10^{-4}	

Table 2: Other constants

Some properties of motors are used:

Letter	Meaning	Value	Source
τ	[τ_l, τ_r], torque of the left and right motors.	$\tau = \begin{pmatrix} \tau_{sl} \\ \tau_{sr} \end{pmatrix} - \begin{pmatrix} \tau_{sl}/\omega_{nl} & 0 \\ 0 & \tau_{sr}/\omega_{nr} \end{pmatrix} \cdot \dot{q}$	
τ_s	[τ_{sl} ,τ_{sr}] The stall torque of the motors left and right.	It is the torque needed to stop a motor for a voltage V. It is proportional to the voltage. Here $\tau_{sl} = \tau_{sr}$ because same motors are used for right side and left side.	Coefficient of proportionality was estimated and then adapted to follow the real vehicle reactions.
ω_n	[ω_{nl} , ω_{nr}]The no-load speed of the motors left and right.	It is the unloaded motor velocity for a voltage V. It is proportional to the voltage. $\omega_{nl} = \omega_{nr}$ because same motors are used for right side and left side.	Coefficient of proportionality was estimated and then adapted to follow the real vehicle reactions.

Table 3: Motor constants

II.A.3. Global view

The motor shield takes two values between 0 and 255 as inputs. These values are turned into voltage for the motors left and right. Then, these voltages are converted into a torque by motors. This torque begets the rotations of wheels. Some phenomena of friction and shear are considered here.

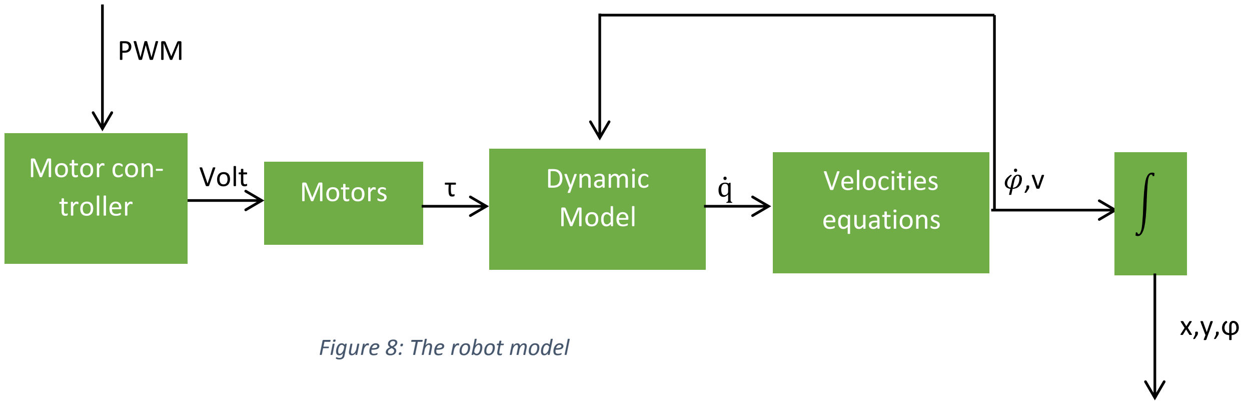


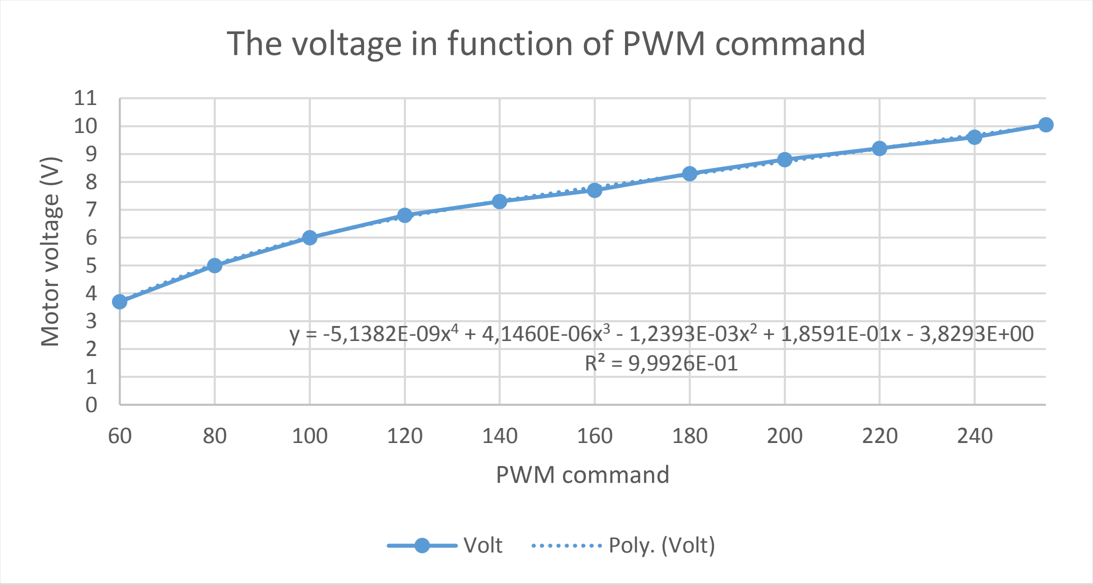
Figure 8: The robot model

II.A.4. Equations [9]

II.A.4.a. Motor Controller

To be able to convert each PWM command into a voltage, a function had been experimentally estimated. Indeed, the input motor voltage had been measured with a voltmeter for each command between 0 and 240 with a step of 20 and for the 255 commands. Then, the commands lower than 60 are not considered because the motors need more than 3 Volt to be efficient.

Here is the curve obtained for commands higher than 60. Dotted curve is the polynomial approximation used in the simulation.



Graph 1: Voltage motor in function of the PWM command

The polynomial degree is not a problem for the simulation because only two command values are used per simulation step. Here the degree used is 4 to have an R-squared value at least equal to 0.999. So, the simulation[10] uses the following function to convert PWM command I into a voltage v (in Volt):

$$\boldsymbol{v} = -5 \cdot 10^{-9} \cdot \boldsymbol{I^4} + 4 \cdot 10^{-6} \ \cdot \boldsymbol{I} - 1.2 \cdot 10^{-3} \ \cdot \boldsymbol{I^2} + 1.859 \cdot 10^{-1} \ \cdot \boldsymbol{I} - 3.8293$$

II.A.4.c. Motors

For the motors of the left side, the torque $\boldsymbol{\tau_l}$ and the rotary speed $\boldsymbol{\omega_l}$ are linked by the following equation [9]:

$$\frac{\boldsymbol{\tau_l}}{\boldsymbol{\tau_{sl}}} + \frac{\boldsymbol{\omega_l}}{\boldsymbol{\omega_{nl}}} = 1$$

$$\Rightarrow \boldsymbol{\tau_l} = \boldsymbol{\tau_{sl}} \cdot (1 - \frac{\boldsymbol{\omega_l}}{\boldsymbol{\omega_{nl}}})$$

The motors of the right side follow the same law, that is why the following equation is verified:

$$\Rightarrow \boldsymbol{\tau} = \begin{pmatrix} \tau_l \\ \tau_r \end{pmatrix} = \begin{pmatrix} \boldsymbol{\tau_{sl}} \\ \boldsymbol{\tau_{sr}} \end{pmatrix} - \begin{pmatrix} \boldsymbol{\tau_{sl}}/\boldsymbol{\omega_{nl}} & 0 \\ 0 & \boldsymbol{\tau_{sr}}/\boldsymbol{\omega_{nr}} \end{pmatrix} \cdot \dot{\mathbf{q}}$$

With $\dot{\mathbf{q}} = \begin{pmatrix} \omega_l \\ \omega_r \end{pmatrix}$ the rotary speed of the wheels.

II.A.4.c. Dynamic Model

The dynamic model is given by the following equation in the local frame x-y [9] where M is the mass matrix, C is the resistance term and G is the gravitational term.

$$M\dot{q} \ + C(q, \dot{q}) + G(q) = \tau$$

i. The gravitational term G(q)

The motion of the vehicle is assumed in a 2D surface, that is why the gravitational term is null.

$$\forall q, \qquad G(q) = 0$$

ii. The mass matrix M

The mass matrix is given by [9]:

$$M = \begin{pmatrix} \frac{mr^2}{4} + \frac{r^2 I}{\alpha B^2} & \frac{mr^2}{4} - \frac{r^2 I}{\alpha B^2} \\ \frac{mr^2}{4} - \frac{r^2 I}{\alpha B^2} & \frac{mr^2}{4} + \frac{r^2 I}{\alpha B^2} \end{pmatrix}$$

[10] See in Annexe - Model, the orange box

With I the moment of inertia computing for a rectangle as the adjacent figure, the inertia matrix is computed as following –with σ the mass per unit area–

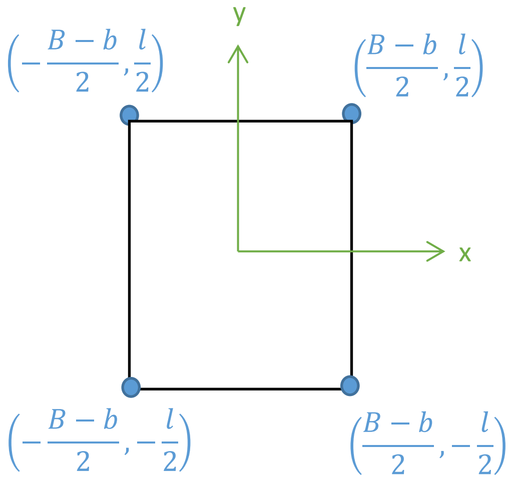

y
x
$\left(-\frac{B-b}{2}, \frac{l}{2}\right)$
$\left(\frac{B-b}{2}, \frac{l}{2}\right)$
$\left(-\frac{B-b}{2}, -\frac{l}{2}\right)$
$\left(\frac{B-b}{2}, -\frac{l}{2}\right)$

Figure 9: The rectangle corresponding to the rover

$$\mathrm{I} = \int (y^2 + x^2) dm$$

$$= \sigma \cdot \int_{-(B-b)/2}^{(B-b)/2} \int_{-l/2}^{l/2} (y^2 + x^2) dy\, dx$$

$$= \sigma \cdot \int_{-(B-b)/2}^{(B-b)/2} \left(\frac{l^3}{12} + x^2 \cdot l\right) dx$$

$$= \frac{\sigma \cdot l \cdot (B-b)}{12} (l^2 + (B-b)^2)$$

But $m = \sigma \cdot l \cdot (B-b)$, that is why:

$$I = \frac{m}{12}(l^2 + (B-b)^2)$$

iii. The resistance term $C(q, \dot{q})$

C is not only due to the rolling resistance f of the wheels but also to the sliding friction F. So, C is equal to the torque resistance of the wheels of the right or left side called τl_res and τr_res . So, C is given by [9]:

$$C = \begin{pmatrix} \tau_{\mathrm{l\_res}} \\ \tau_{\mathrm{r\_res}} \end{pmatrix} = \begin{pmatrix} r \cdot (F_l + f_l) \\ r \cdot (F_r + f_r) \end{pmatrix}$$

- The rolling resistance

The rolling resistance: “is its resistance to movement caused by friction between it and the surface it is rolling on.” (Collins dictionary) Rolling resistance factors are for example elastic deformations or surface irregularities [12] [14]. It is given by

$$f = N \cdot \mu_r$$

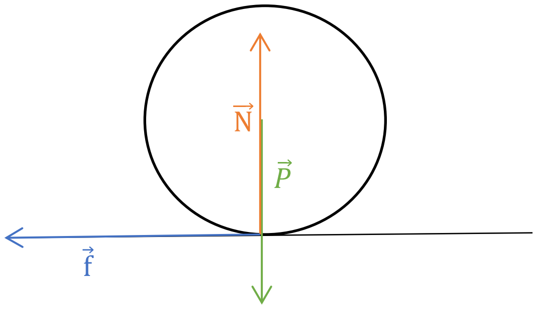


Figure 10: The forces on the wheel

Where N is the normal force (perpendicular to the road) and μ_r is the dimensionless coefficient of the rolling friction.

Per the Newton’s second law the normal force is given by:

$$N = P = \frac{mg}{2}$$

There is a ½ factor because the system is relative to one side of the vehicle (composed of two wheels). In this way, the rolling resistances are given by:

$$f_l = f_r = \frac{mg}{2} \cdot \mu_r$$

- The sliding friction

The sliding friction: The sliding friction is the integration of the shear stress on the patch (i.e. every contact point between the wheel and the road). The shear stress is given by [9]:

$$\tau_{ss} = p\mu \cdot (1 - e^{-\frac{j}{K}})$$

Where p is the normal pressure, µ is the coefficient of friction, j is the shear displacement and K is the shear deformation modulus.
In this case, the sliding velocities for the wheel of the front right side (fr) is given by [9]:

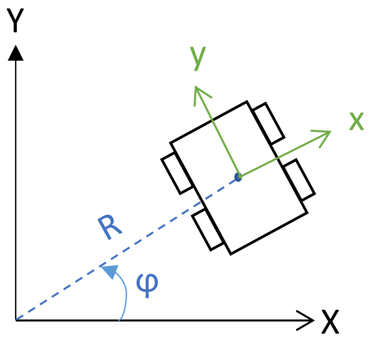


Figure 11: Some constants name

$v_{fr} = \begin{pmatrix} -y_{fr} \cdot \dot{\varphi} \\ \left(R + \frac{B}{2} + x_{fr}\right) \cdot \dot{\varphi} - r\omega_r \end{pmatrix}$ in the x,y spatial system.

It is the same equation for v_{rr} . But in the sliding velocities equations for the wheels of the left side, the coefficient $R + \frac{B}{2} + x_{fr}$ becomes $R - \frac{B}{2} + x_{fr}$.

j is the timing integration of the sliding velocities. Here is the equation to compute each term of j like in [9]:

$$j_{fr_X} = \int_0^t v_{fr_X}\, dt = \int_{y_{fr}}^{\frac{L}{2}} (v_{fr_x} \cdot \cos(\varphi) - v_{fr_y} \cdot \sin(\varphi)) \frac{1}{r\omega_r} dy_r$$

Because $\frac{dy}{dt} = r\omega_r$ and so $t = \int_{y_{fr}}^{L/2} \frac{1}{r\omega_r} dy_{fr}$

But $\int_{y_{fr}}^{\frac{L}{2}} \frac{1}{r\omega_r} dy_{fr} = \frac{L/2 - y_{fr}}{r\omega_r}$ and $\varphi = \dot{\varphi} \cdot t$

So, $\varphi = \dot{\varphi} \cdot \frac{L/2 - y_{fr}}{r\omega_r}$

$$j_{fr_X} = \int_{y_{fr}}^{\frac{L}{2}} \left[-y_{fr} \cdot \dot{\varphi} \cdot \cos\left(\dot{\varphi} \cdot \frac{L/2 - y_{fr}}{r\omega_r}\right) - \left[\left(R + \frac{B}{2} + x_{fr}\right) \cdot \dot{\varphi} - r\omega_r\right] \cdot \sin\left(\dot{\varphi} \cdot \frac{L/2 - y_{fr}}{r\omega_r}\right)\right] \frac{dy_{fr}}{r\omega_r}$$

That is why, $j_{fr\_X} = \left(R + \frac{B}{2} + x_{fr}\right)\left[\cos\left(\dot{\varphi} \cdot \frac{L/2 - y_{fr}}{r\omega_r}\right) - 1\right] - y_{fr} \cdot \sin\left(\dot{\varphi} \cdot \frac{L/2 - y_{fr}}{r\omega_r}\right)$

In the same way $j_{fr\_Y} = \left(R + \frac{B}{2} + x_{fr}\right) \cdot \mathbf{sin}\left(\dot{\varphi} \cdot \frac{L/2 - y_{fr}}{r\omega_r}\right) - \frac{\boldsymbol{L}}{\boldsymbol{2}} + y_{fr} \cdot \mathbf{cos}\left(\dot{\varphi} \cdot \frac{L/2 - y_{fr}}{r\omega_r}\right)$

The shear displacement $j_{r..}$ of one rear wheel follows the same equation than $j_{f..}$ but the length L must be replaced by the value of –C (cf schema of the vehicle dimensions). The shear displacement $j_{..l}$ of one left wheel follows the same equation than $j_{..r}$ but the term $R + \frac{B}{2} + x_{..r}$ must be replaced by $R - \frac{B}{2} + x_{..l}$(cf schema of the vehicle dimensions).

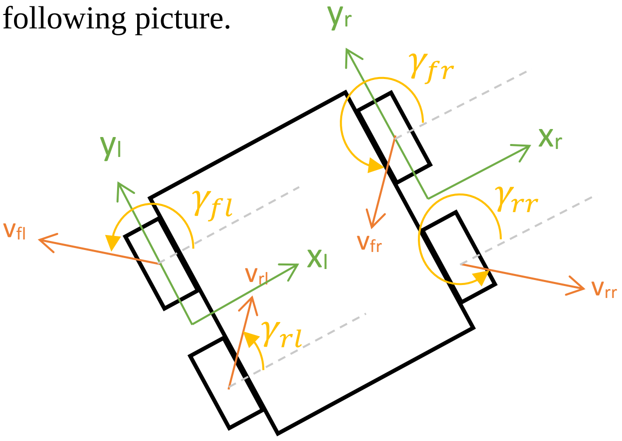
With the shear displacement the sliding friction can be computed thanks to the integration of the sheer stress on each patch. The sliding friction of the right side is given by:

$$F_r = \int_{\frac{C}{2}}^{\frac{L}{2}} \int_{-\frac{b}{2}}^{\frac{b}{2}} p\mu(1 - e^{-j_{fr}/K}) \cdot \sin(\pi + \gamma_{fr})\, dx_r dy_r$$

$$+ \int_{\frac{C}{2}}^{\frac{L}{2}} \int_{-\frac{b}{2}}^{\frac{b}{2}} p\mu(1 - e^{-j_{rr}/K}) \cdot \sin(\pi + \gamma_{rr})\, dx_r dy_r$$

It is the same equation for the sliding friction of the left side but the length L must be replaced by –C.

With $j_{fr} = \sqrt{{j_{fr\_X}}^2 + {j_{fr\_Y}}^2}$,and γ_{fr} the angle of the resultant sliding velocity v_{fr} like in the following picture.



$$\gamma_{..r} = \pi + \arctan(\frac{v_{..r\_Y}}{v_{..r\_X}})$$

and $\gamma_{..l} = \arctan(\frac{v_{..l\_Y}}{v_{..l\_X}})$

Figure 12: The angles of the resulting sliding velocity

In this way, $M\dot{q} + C(q,\dot{q}) + G(q) = \tau$ is an equation between $\ddot{q}$ and $\dot{q}$.

We have $\ddot{q} = M^{-1}\left[\begin{pmatrix}\tau_{sl}\\ \tau_{sr}\end{pmatrix} - \begin{pmatrix}\boldsymbol{\tau_{sl}/\omega_{sl}} & 0\\ 0 & \boldsymbol{\tau_{sr}/\omega_{sr}}\end{pmatrix} \cdot \dot{\mathbf{q}} - C(q,\dot{q})\right]$.

In the simulation[11] $\dot{\mathbf{q}}$ is gotten by a runge-kunta 4 equation.

II.A.2.d velocities equations

Thanks to the dynamic model $\dot{\mathbf{q}} = \begin{pmatrix}\omega_l\\ \omega_r\end{pmatrix}$ is known. The linear velocity and the rotary velocity are given by [9]:

[11] See in Annexe - Model: in the purple box for the Dynamic Model, in the blue box for the shear displacement j, in the green box for the angle γ, in the yellow box for the sliding friction.

$$\begin{pmatrix} v_y \\ \dot{\varphi} \end{pmatrix} = \frac{r}{\alpha B}\begin{pmatrix} \frac{\alpha B}{2} & \frac{\alpha B}{2} \\ -1 & 1 \end{pmatrix}\cdot\begin{pmatrix} \omega_l \\ \omega_r \end{pmatrix}$$

So, the linear velocity is the mean of the rotary velocities multiplied by the wheel radius, whereas the rotary velocity of the vehicle depends on a terrain-dependent parameter α. This term is got by adaptation with the actual results.
Then, others variables like x,y, φ is computed thanks to Euler equations[12].

II.A.2.c Generalisation

The previous equations are for a turn to the left. The adaptation of the equations for a turn to the right is explained.

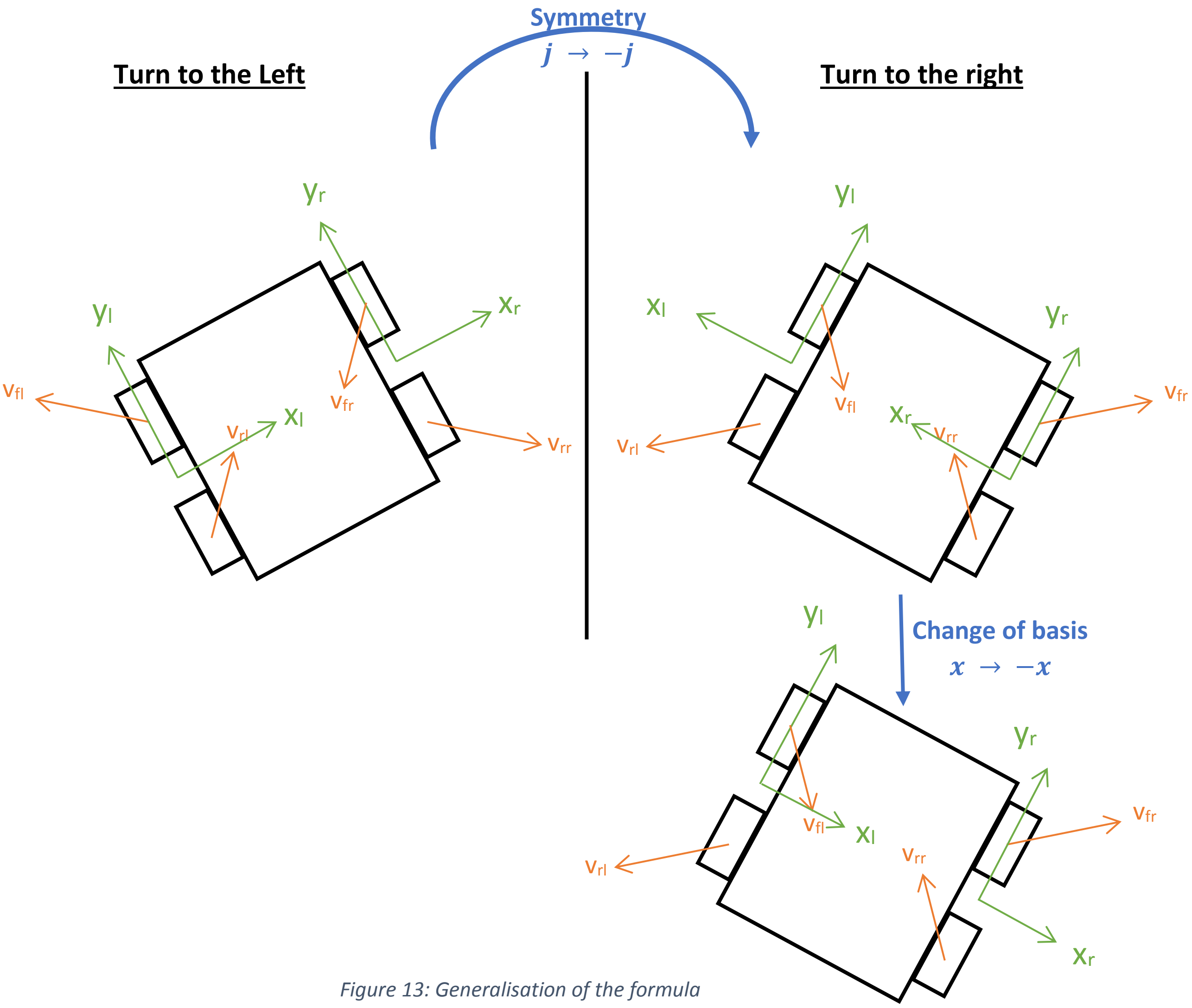


Figure 13: Generalisation of the formula

[12] See in Annex – Model, in the grey box.

The both situations are symmetric. So, j for the first case becomes –j. Then, we don't use the symmetric base so basis must be change with x -> -x. To conclude, j(x) becomes –j(-x). To generalize equations for a forward motion, we use an infinite curve radius R.

See the simulation Python code in Annex - Model.

II.B. Record data & tests

II.B.1. The functioning

The dynamic model uses a lot of computing time, that is why the simulation was modified to use a 2D-cubic regression of the sliding friction F depending on the heading φ and on the velocity. It is a legitimate approximation because the determination coefficient is about 0.993. The test is made by this way:

Sensors/component used	*Data*
Arduino clock	Time since the powering of the Arduino
encoders	Velocity of the left rear wheel
encoders	Velocity of the right rear wheel
Compass	Heading
Gyroscope	Heading
MicroSD board	Recorded file

Table 4: Sensors and Components used to collect and record data

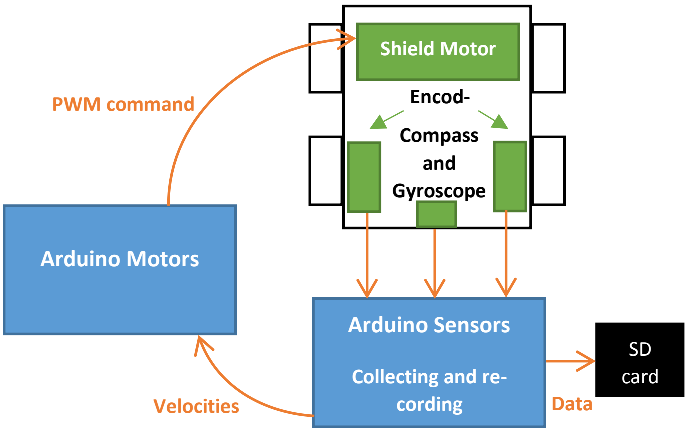


Figure 14: Principle used to test

The both Arduino boards are powered by a 12V battery. So, they are initialised in the same time.
The Arduino called "Arduino sensors" is wired with encoders, compass, MicroSD board and with the "Arduino Motors" via I2C (SDA and SCL).
The number of holes detected by encoders are computed thanks to Interruption (counter is incremented each time there is a change in the signal). Every 500ms the linear velocities of each vehicle side are computed and send to the "Arduino Motor" via I2C communication thanks to the *Wire* library. This long time of 500ms is to have more accuracy values of mean velocity but also to appreciate a little the acceleration. For more details about it see the Encoders section.
Other sensors data are collecting and recording in the SD card every 50ms.
"Arduino Motor" executes a proportional-integral-derivative controller using the inverse kinetic model. Then, it sends the PWM command to the shield motor. See the Project Code to watch the Arduino programs[13].

[13] The recoding part is between the line 359 and the line 415 at the pages 12 - 13.

II.B.2. Data processing and results

The video processing

Each test is recorded and the video is processed by Tracker [15]. For each video a calibration stick, and an offset origin is put as reference for axis. Then a point of the vehicle is used like a point mass for an auto-tracking. This auto-tracking allows us to have the x,y coordinates in function of the time. Thanks to these data an approximation of the instantaneous speed is computed.

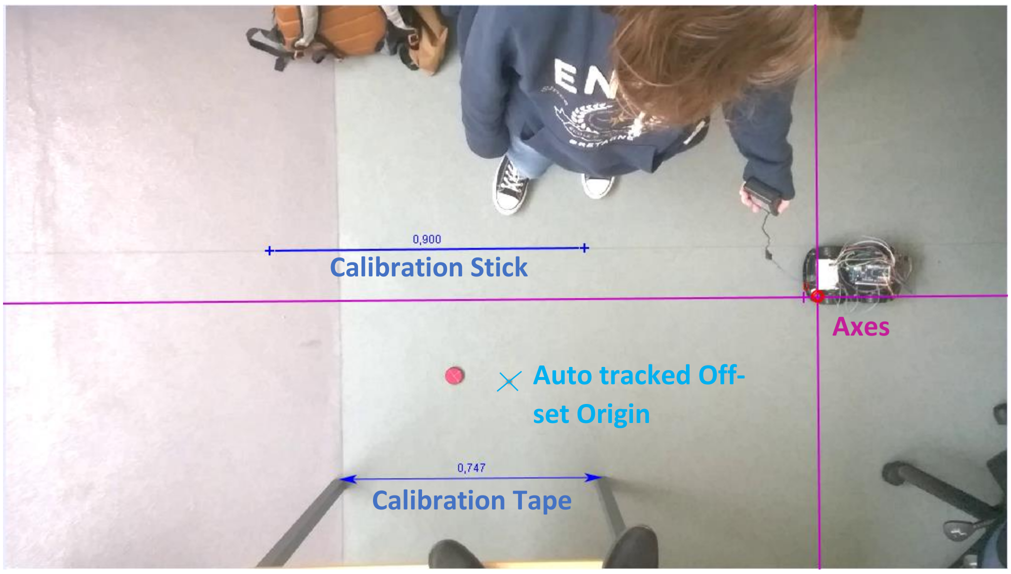


Photo 8: The used Tracker tools

The data processing

Each data is collected in an Excel page and velocity are computed:

- The mean of the speed during the last 500ms thanks to encoders,
- The "instantaneous speed" thanks to Trackers data.

Both is plotted in a figure and then data simulated are created and some parameters are adapted in the simulation like the terrain-dependent coefficient called alpha. For each test, the simulated values (in grey), measured values with sensors (in bleu) and measured values with trackers (in yellow) are plotted.

<u>For the velocity:</u>

The bleu values are computed with the encoders data. Indeed, the value of the mean velocity at the discretised time *i* is computed with Excel and is given by $v_i = \frac{v_{l_i} + v_{r_i}}{2}$

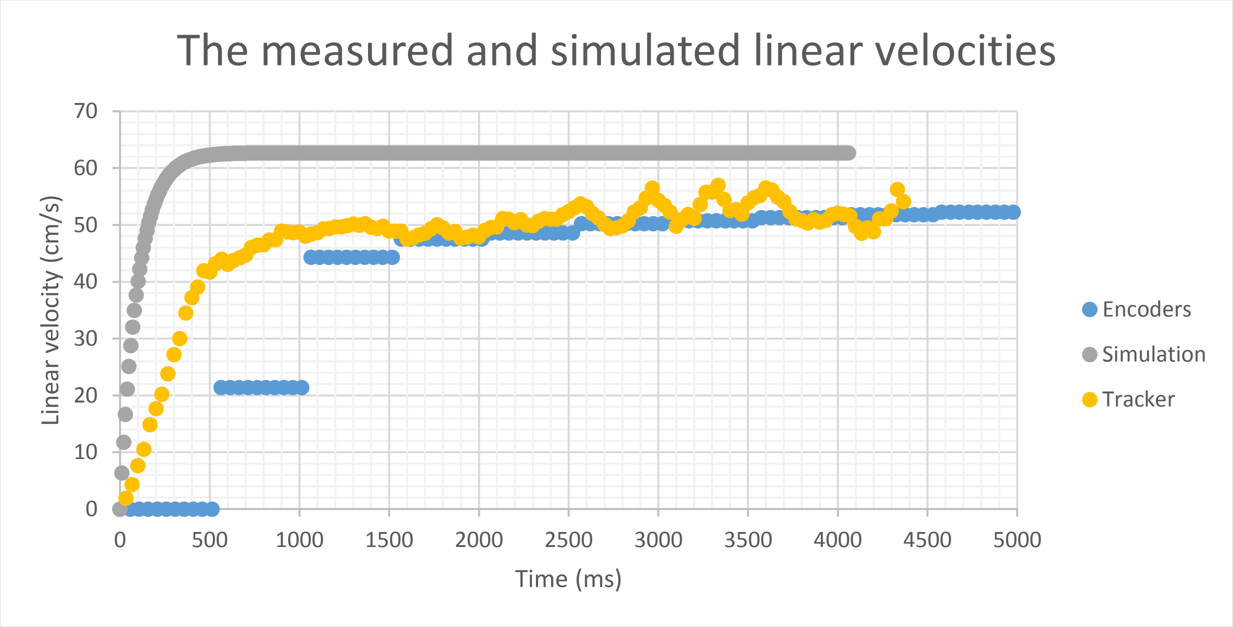
The yellow curve is got with Tracker values. They are given by:

$$v_i = \frac{\sqrt{(x_{i+2} - x_{i-2})^2 + (y_{i+2} - y_{i-2})^2}}{t_{i+2} - t_{i-2}}$$

<u>For the heading:</u>

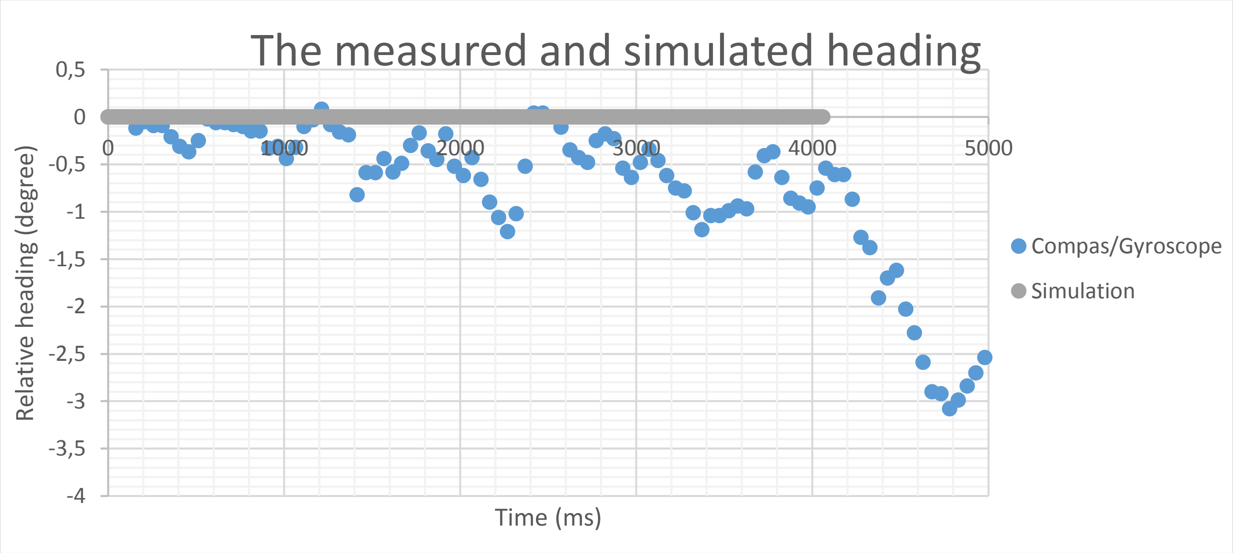
The bleu values got with the relative heading (the fusion between the compass and the gyroscope) and given by $\varphi_i = \varphi_i - \varphi_0$

The constant used in the dynamic model is got with constructor documentation and data recording. Here there is one result for a command of 80 in each motor:



Graph 2: The velocity of the rover, measured by the encoder (blue), measured by Tracker (yellow) and simulated (grey) for a test which a command of 80 in each motor.

We can observe that the simulated rover has a time of responses shorter and a response higher than the actual rover. Indeed, the measured speed of convergence is about 52cm/s (tracker and encoder) whereas the simulation's one is 10cm/s more. The rover is over 95% of its convergence speed after 1,17s whereas it is about 0.27s for the simulated rover, so 4 time smaller. Now, let us study the heading for this experiment:



Graph 3: The heading of the rover, measured by the encoder (blue), and simulated (grey) for a test which a command of 80 in each motor.

The rover slightly drifts. It is explained by a difference of voltage delivered by the motor shield. The simulated rover does a straight line without drift because this issue is not considered. So, after 4s the difference between the rover and the simulated one is about 0.95 degree. See in annex for another test (for a command of 80 for left motors and 150 for the right one's).

Whatever the value of each constant, the simulation and the rover data do not fit correctly. So, the simulation does not simulate the rover used for tests but another one. This is maybe because it is a non-perfect rover which has some difficulties to be modelled with low-cost sensors and substantial overlooked phenomena. Even that, every algorithm was tested at first in the simulation.

II.B.3. Some issues encountered

II.B.3.a. Compass/Gyroscope

Firstly, heading was only given by the compass but the electro-magnetic field in the working room is abnormal, so a gyroscope was added to have better measurements. Both components are used to have a better heading for an indoor situation and an outdoor situation[14]:

1. The predicted heading is computed with gyroscope data and the last heading.
2. If the difference between the predicted value and the value of the compass is too high, the new heading is the predicted.
3. Then, it is the compass value.

Moreover, a threshold on the gyroscope value was added to avoid a huge drift. In this way, when the electromagnetic is disturbed, the heading computation[15] uses the gyroscope value. But when the compass seems give a correct value, it uses this one. These functions are called in the main program in the pages 9 at the lines 185-188 and pages 10 at the lines 277-230 and 233-239 for the initialisation, and in the pages 12 at the lines 376-382 for the reading and processing.

Secondly, gyroscope mpu6050 works if the voltage used is around 3,3V but the Arduino I2C communication is a 5V. So, we needed to add a level shifter to down to 3,3V [16]. Moreover, it blocks sometimes and the Arduino board lock up.

II.B.3.b. I2C bus

Moreover, motors beget a lot of noise in the I2C bus. This noise has a significant negative impact in the buffer. When there are too many noise, the length of the buffer is considered infinite by the TWI Arduino library and it beget an infinite loop when the TWI buffer is copied to a variable. So, a security was added into this library to break the infinite loop and avoid that the Arduino board locks up because of I2C communication.

[14] The piece of code corresponding is in the page 17 and 18 of the Project Arduino Code, be-tween the line 116 and the line 153 (fuse function).

[15] The code relatives to the heading is in the pages 11 – 18 of the Project Arduino Code.

III. Autonomous algorithm to reach a target.

III.A. Controller

One issue encountered was the non-identical response of the left side motors and of the right-side motors. These responses begot a dysfunction in the straight-line motion. To solve this, a double PID was created. The PID about velocity depends on the vehicle side.
A PID using the current heading and the desired was created to compute the rotary velocity desired to have a better heading. Then the desired rotary velocity and the desired velocity are used to compute the desired velocities of wheels which are used in the last PID using the current velocities and computing the command for each vehicle side. The controller is featured as following:

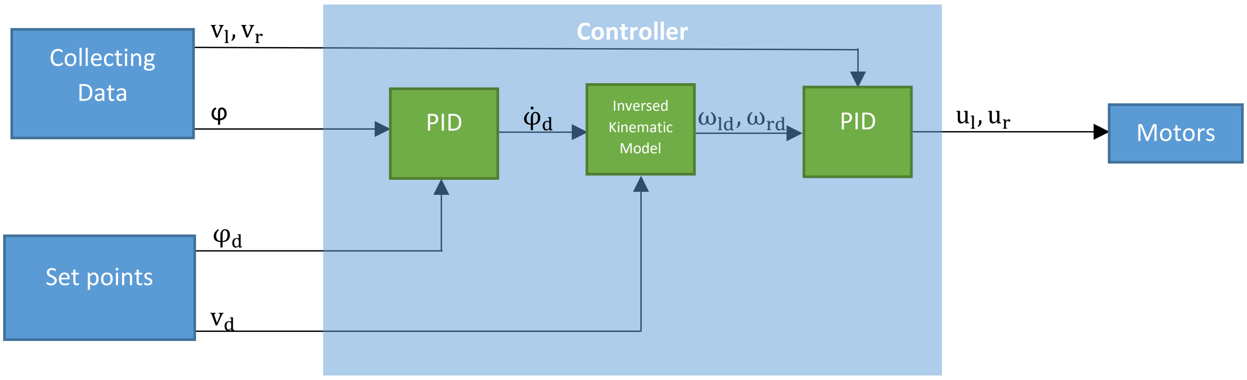


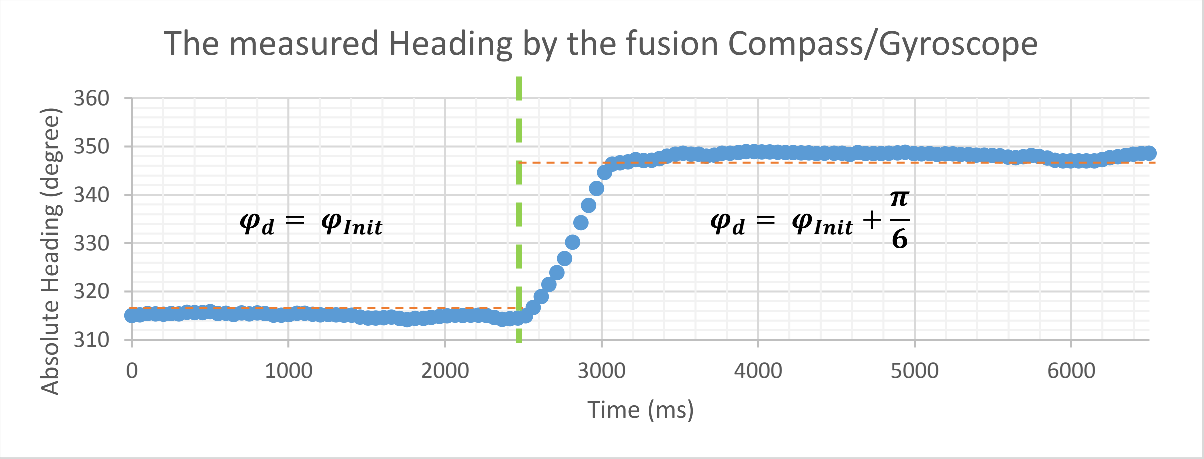
Figure 15: The controller functioning

With v_l, v_r the velocity of vehicle side, ϕ the heading, φ_d and v_d respectively the desired heading and the desired linear velocity of the vehicle, ω_{ld} and ω_{rd} respectively the desired rotary velocity[16] of the left side and right side, u_l and u_r respectively the velocity command for the left wheels and for right wheels[17].

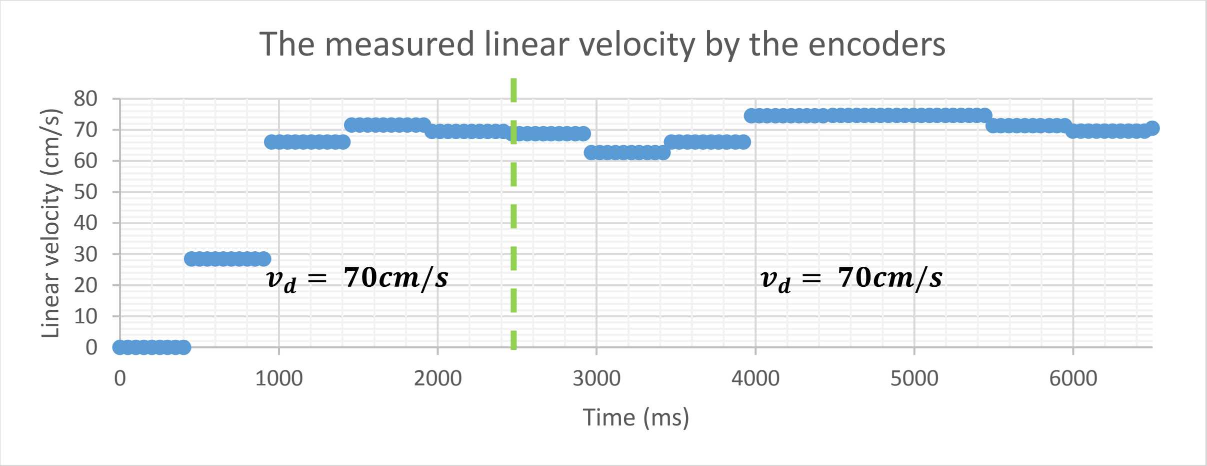
Here is the velocity resulted of a straight line at the initial heading followed by a straight line with a Pi/6 heading added.

[16] Equations are given in Annex - Controller.

[17] See the Arduino code in the Project Code, at the page 48-52. The controller is called by the main code at the page 36 line 327.



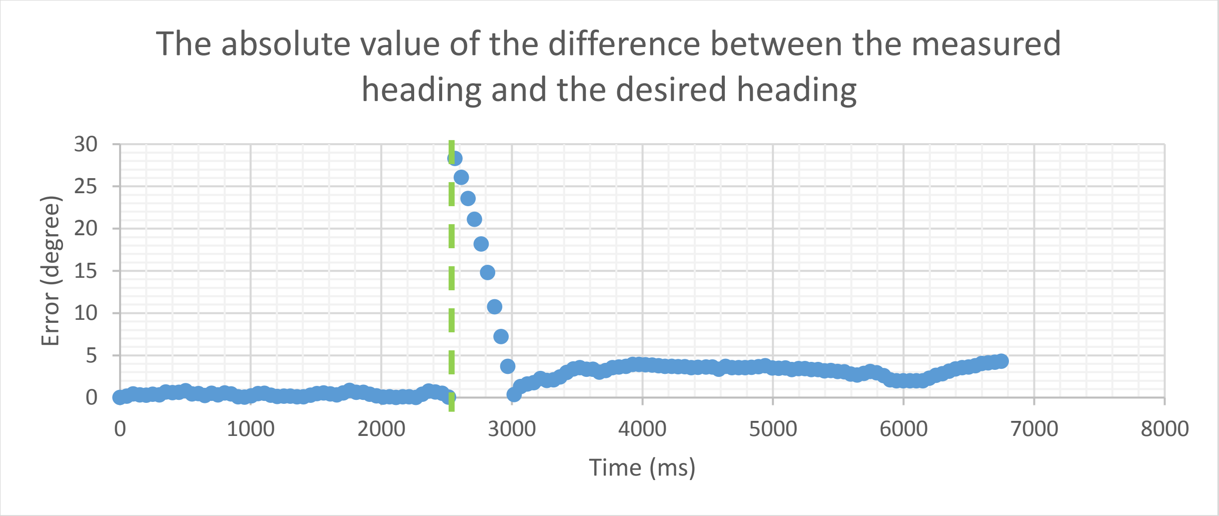
Graph 4: The measured heading for a specific variable command



Graph 5: The measured linear velocity for a specific variable command

The velocity measured by encoder reach 95% of the desired value at 1.46s. We can see that the change of desired heading begets a small velocity disturbance with a small oscillation between 62cm/s and 74cm/s. The convergence is at 70cm/s as desired.

Here are the error curves for the heading:



Graph 4: The absolute difference between the measured heading and the desired one

We can observe that in the first part the heading error is under one degree. Then the error increases because the desired value changes, after half a second the error down to 0.37 degree and then up to 3.7 degree. The second desired heading is followed with an average error of 3 degree which is correct for a controller implemented every half second because of low-cost encoders use. Do not forget that there is also the incertitude due to sensors for the measured heading. However, a slightly increase of the integral term in the first controller could improve the convergence value.

III.B. Observer

III.B.1. General Explanation [17]

An observatory is needed. The system made is the following:

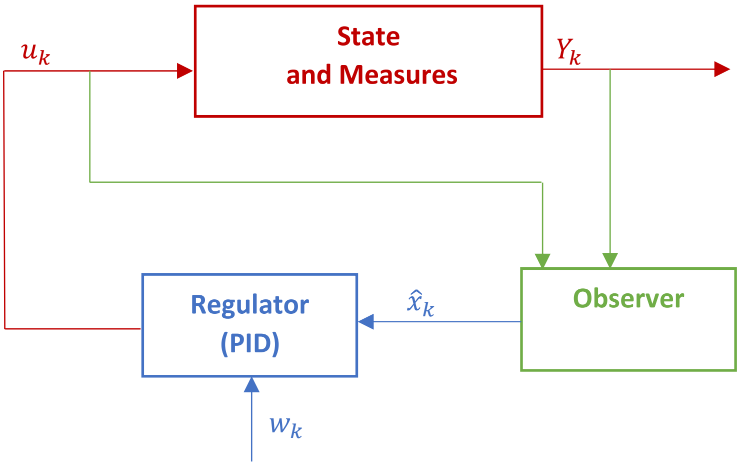


Figure 16: The system composed of the Observer and the Controller

With w_k the velocities and heading desired and $\hat{x}_k$ the predicted state.

III.B.2. Application on the system

III.B.2.a. Application

The first idea was to use an extended Kalman filter as observer (See in annex the explanation of this observer). The fact that the rover used for test is difficult to model and that the dynamic model used is not in accordance with the rover begets a real issue in this observer. Indeed, the observer is based on a model, but if the model is wrong, the observer will be wrong too. So, here is a significant issue on this project: the simulation works, but it is not a simulation of the rover used for test because this rover is too unperfected (low cost sensors, low cost motors, room with electro-magnetic noise…). The bad accuracy of the GPS is also an issue for this part.

To solve both of these issues three choice comes:

- Create a locator which computes the position $\begin{bmatrix} x \\ y \end{bmatrix}$ of the rover with GPS data.
- Use a Kalman Filter on $\begin{bmatrix} x \\ y \end{bmatrix}$ using the GPS data.
- Apply a dead reckoning.

These three solutions were tested, and the two first had been compared.

i. the “locator”

A locator was implanted to computes the position $\begin{bmatrix} x \\ y \end{bmatrix}$ of the rover. The locator is a fuse of the GPS data and a dead reckoning.

- The displacement d_n of the rover from the initial position to the present time position $p_n = \begin{bmatrix} x_n \\ y_n \end{bmatrix}$ given by the GPS, (see in the following figure at the left)
- The mean of all the translated GPS data samples gives a better estimation for the initial position.
- The estimation of the present time position $\hat{p}_n$ is given by the translation of the better estimation of the initial position. (see in the following figure at the right)

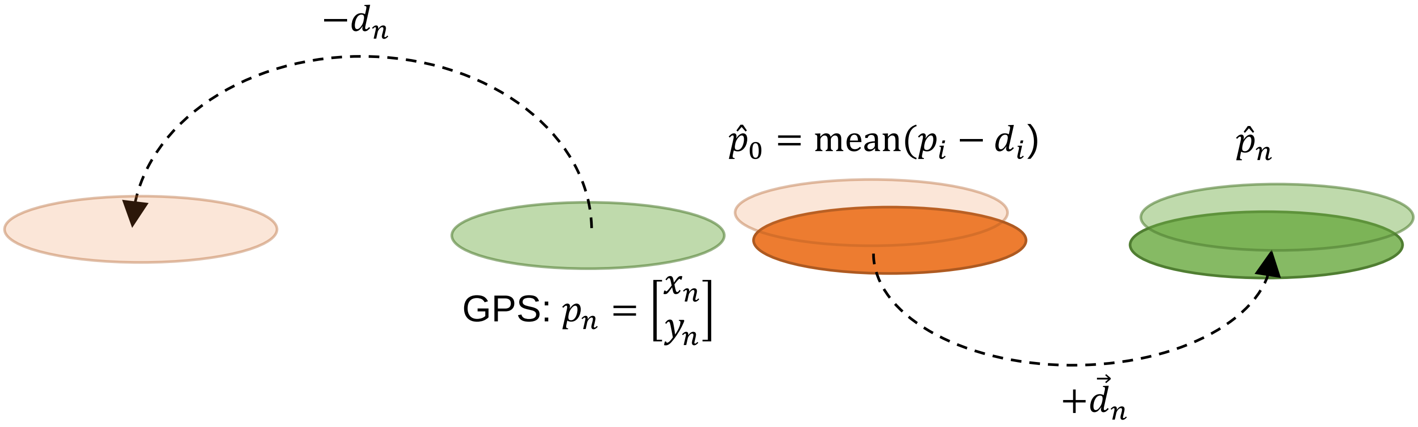

$-d_n$
GPS: $p_n = \begin{bmatrix} x_n \\ y_n \end{bmatrix}$
$\hat{p}_0 = \text{mean}(p_i - d_i)$
$\hat{p}_n$
$+\vec{d}_n$

Figure 17: The "Locator" principle

The function *ComputeXY*[18] is used to update the coordinate of the vehicle in function of the GPS data x,y and the value of v_l, v_r, φ – X – and of the time step dt. In this code x0 and y0 are the first coordinate of the vehicle and is the total displacement of the vehicle.

This solution cannot be used if there are too many GPS disturbances. It is an accurate method if the initial position is well known.

ii. Kalman filter

The third solution is a Kalman filter[19] with:

$$\dot{X} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \frac{v_l + v_r}{2}\cos(\varphi) \\ \frac{v_l + v_r}{2}\sin(\varphi) \end{bmatrix}$$

$$Y = \begin{bmatrix} x \\ y \end{bmatrix}$$

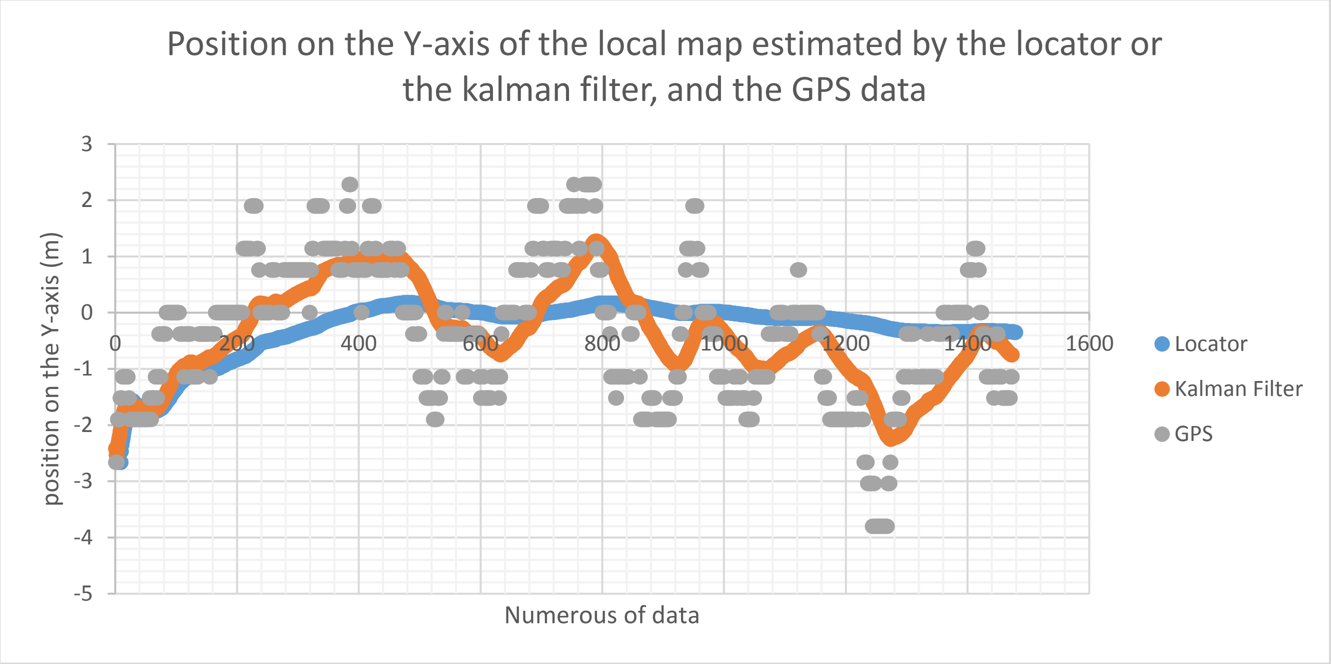
A test has been made to compare this observer with the locator previously explained. The test has been made in a wasteland to avoid reflexion due to building. The first position of the rover is given thanks to google map data with:

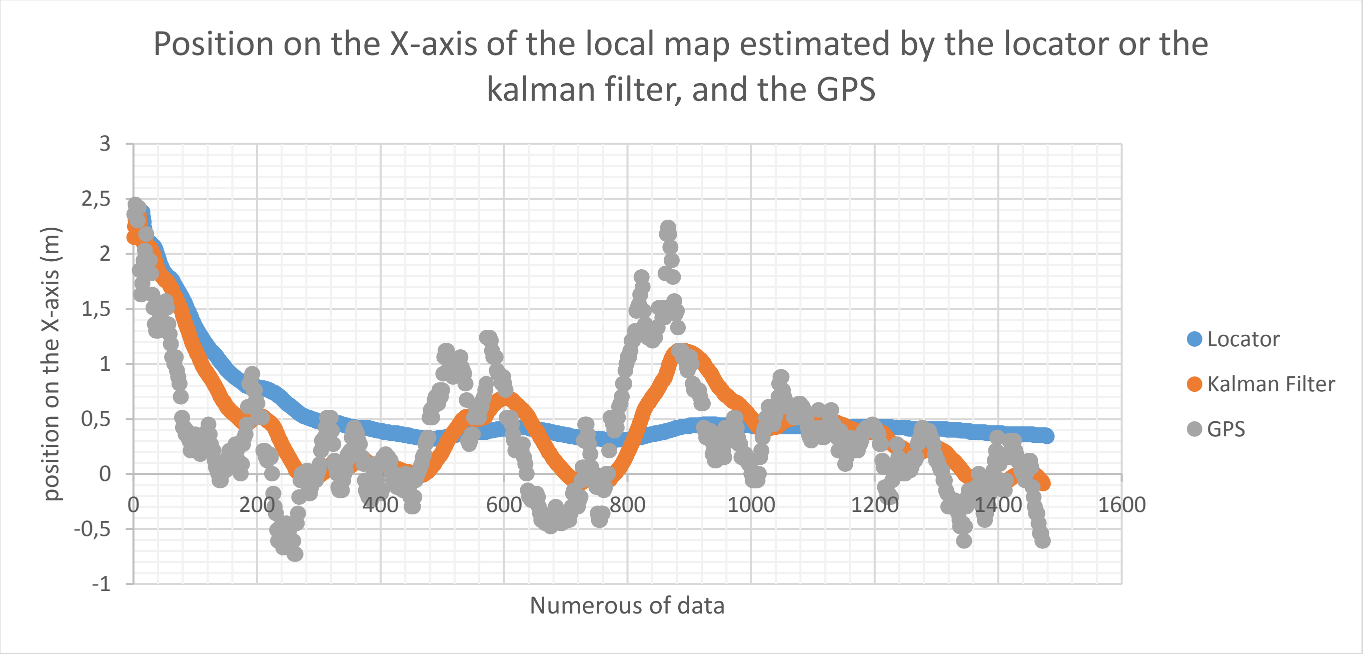
$$\begin{cases} latitude = 50.377799 \cdot \frac{\pi}{180} \\ longitude = -4.126823 \cdot \frac{\pi}{180} \end{cases}$$

The rover is motionless and data computed by the Kalman filter, the locator and measured by the GPS are recorded. So, if the first position is perfectly known the observers should converge at the position (0,0). Here are data got:

[18] See this function at the page 44, lines 237-262 of the Project Code. It can be called in the main code at the page 35, lines 295.

[19] See this observer at the page 42 – 43, lines 85 - 202 of the Project Code. It can be called in the main program at the page 35 line 293





Graph 7-8: The comparison between the Locator result (blue), the Kalman Filter result (orange), and the GPS data (grey)

The GPS noise is not Gaussian despite the choice of a wasteland, so the Kalman filter choice is not very relevant.

We can see that the locator is less disturbed by the GPS noise than the Kalman filter. The locator converges toward a value whereas the Kalman oscillates around the value with an amplitude depending of the GPS noise. For a motionless experiment the locator seemed be more accurate than the Kalman filter. But the difference between both observers are not large: it is a difference less than 1 meter. At the end of the experimentation (i.e. after 12min20s) the locator gives the following value:

$$\begin{cases} x = 0.34\ cm \\ y = -0.35\ cm \end{cases}$$

These values are reached for the first time by the locator after 3min44s for the x value and 2min31s for the y value. Whereas for the Kalman Filter it is 1min58 for the x value and 2min36 for the y value. In annex is the position given by these observers and the GPS data. We can easier see that the locator converges whereas the locator turns around the value.

iii. Dead Reckoning

Here is the fourth solution. It is a dead reckoning given by:

$$\dot{X} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} u_1 \cos(u_2) \\ u_1 \sin(u_2) \end{bmatrix}$$

$$\begin{cases} u_1 = \dfrac{v_l + v_r}{2} \\ u_2 = \varphi \end{cases}$$

The code was made using the *MathsMatrix* library. The function made for the dead reckoning takes in arguments x,y –the last rover localisation- ,φ– the heading-, v_l, v_r- the velocities-, dt – the sample time – and X the state vector which is computed. The function is named *DeadReckoning*[20].

The result of the dead reckoning was written in the LCD screen. In annex are the result for some tests with a desired velocity of 70cm/s. For the test, the local map was rotated of φ_{Init} to be able to check easily the result. The last position is computed by the dead reckoning (printed on the LCD screen) and manually measured.
The difference between the dead reckoning result and the measures are less than 20cm for the x-axis and 4cm for the y-axis. The major part of the error is due to the acceleration of the rover and the high time sample (500ms). The change of heading begets velocity variation which disturb the dead reckoning. Moreover, the more there are velocity variations the less this method is accurate and the more the travel distance is long the less this method is accurate.

III.B.2.a. Conclusion

In an indoor situation, the best choice between the three solution is the dead reckoning because it does not use GPS data and works for small experiment.

For a low-cost GPS, the locator will be preferred in outdoor place where the noise is possibly non-Gaussian.

III.C. Guidance

III.C.1. Explanation

The guidance is the algorithm which computes the desired velocity and desired heading to feed the controller. It allows the rover to reach a target and to avoid obstacle.
The guidance made is based on the artificial potential method. The rover is like a charged particle in a potential field, obstacles are same-sign-charged particle and target is a particle with an opposite

[20] See the page 44 at the lines 206 – 233. This function is called by the main code at the page 35 line 294.

sign charge.
So each thing –target and obstacles- create a potential field. And the rover reacts to this field.

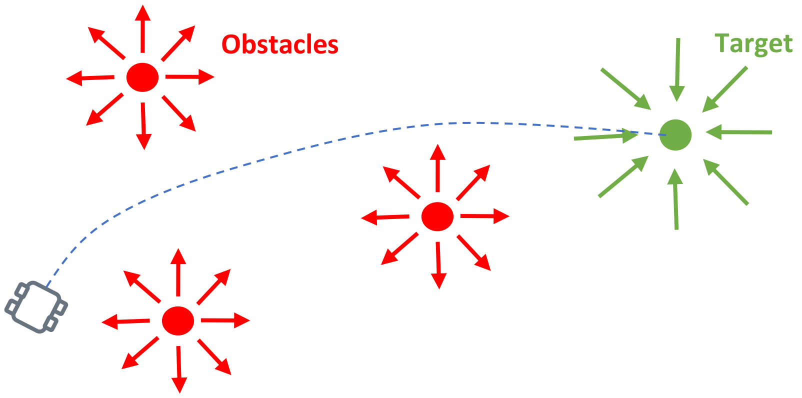


Figure 18: The artificial potential principle

In this part the rover position in the local map is called $p = \begin{bmatrix} x \\ y \end{bmatrix}$, the target position in the local map is called $p_t$and the position of the different obstacles are called q_i.

The resulted potential $V(p)$ is computed as the addition of each potential $V_i(p)$.

$$V(p) = \sum V_i(p)$$

Then, the resulted forced $\vec{f} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$is given by:

$$\vec{f} = -\overrightarrow{grad}(V(p))$$

Finally, the desired velocity and the desired heading are given by:

$$\begin{cases} v_d = \left|\left|\vec{f}\right|\right| \\ \varphi_d = \arctan(\frac{f_x}{f_y}) \end{cases}$$

Here is the force due to the target:

$$\overrightarrow{f_t} = K_t \cdot \frac{p - p_t}{||p - p_t \,||}$$

Obstacles are detected with three ultrasonic sensors put in the front of the rover[21]. The obstacles distances are converted into a coordinate in a local map thanks to the positioning of sensors in the rover and to the localisation of the rover in the local map (see in annex). In this part the position of an obstacle detected by the sensors left is called $p_{obs\_L} = \begin{bmatrix} x_{obs\_L} \\ y_{obs\_L} \end{bmatrix}$.

Here is the force due to obstacles detected at the left and right of the rover:

[21] See I. Hardware part, Ultrasonic module, The position computing.

$$\overrightarrow{f_{obs\_L}} = \overrightarrow{f_{obs\_R}} = K_{obsLR} \cdot \frac{p - p_{obs\_L}}{\left\|p - p_{obs\_L}\right\|^2}$$

Here is the force due to the obstacle detected with the sensor at the middle:

$$\overrightarrow{f_{obs\_M}} = K_{obsM} \cdot \frac{p - p_{obs\_M}}{\left\|p - p_{obs\_M}\right\|^3}$$

K_t, K_{obsLR} and K_{obsM} are three constants experimentally found.

III.C.2. Improvement

III.C.2.a. Local Minimum

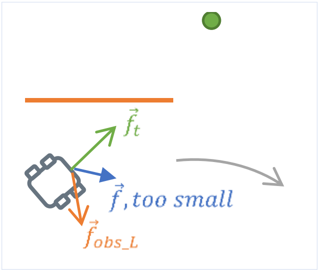


Figure 19: The Local Minimum example

One issue had been encountered with this method: The potential field can be equal to zero. It is a local minimum [18]. In this case the rover does not move and it cannot reach the target. To solve this issue, a second behaviour had been created.

If the potential field is under a threshold v_{min}, the local minimum is assumed and the rover must try to move away from the nearest obstacle (left or right). The decision given by:

$$\begin{cases} v_d = 55\ cm/s \\ \varphi_d = \varphi \pm K_\gamma \cdot \gamma \end{cases}$$

With γ the heading of the sensor used compare to the rover's one φ. Here $\gamma = \frac{\pi}{6}$.
The coefficient K_γ to move away had been found experimentally.

III.C.2.b. Safety

To avoid overshooting the target, the rover is stopped near the target thanks to a threshold d_{min}. When the distance to the target is under this threshold the decisions is the following:

$$\begin{cases} v_d = 0\ cm/s \\ \varphi_d = \varphi \end{cases}$$

This stops the rover. And while the target is unchanged the decision will be this last. To avoid having too higher speed, the speed is limited with a second threshold v_{max}.

III.C.2. Application[22]

The constant experimentally found are the following:

Name	Symbol	Value

[22] The Arduino code is in the Project Arduino Code pages 45 - 48. It is called by the main program at the page 36, line 319.

ATTRACTIVE_TARGET_COEF	K_t	72
REPULSIVE_OBSTACLE_COEF	K_{obsLR}	4
	K_{obsM}	0.8*K_{obsLR}
THRESHOLD_LOCAL_MINIMUM	v_{min}	60
GAMMA_COEF	K_γ	3
OVERSHOOT_SECUTITY_DISTANCE	d_{min}	30 (cm)
THRESHOLD_VELOCITY_SECURITY	v_{max}	100 (cm/s)

Table 5: The constant used in the path planning and found experimentally

Here is a test performed at the end of the internship. The rover begins its test on the yellow cross and its target is four meter further on the other yellow cross. An obstacle -here two cardboard boxes- is placed between the two cross. The rover starts to go forward and after it avoids the obstacle by the right and go to the target. It stops some centimetres before the target because of the dead reckoning precision.

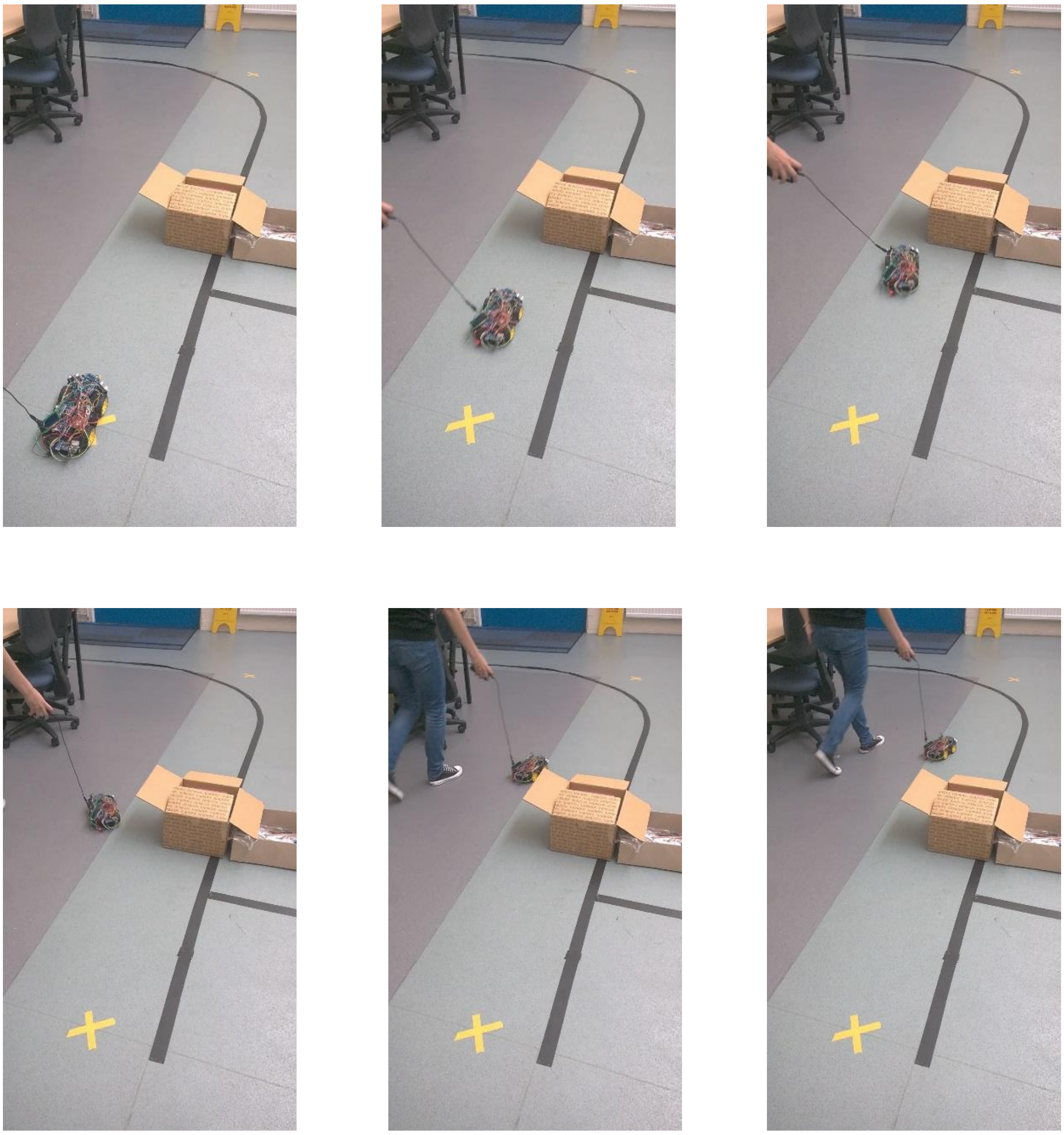
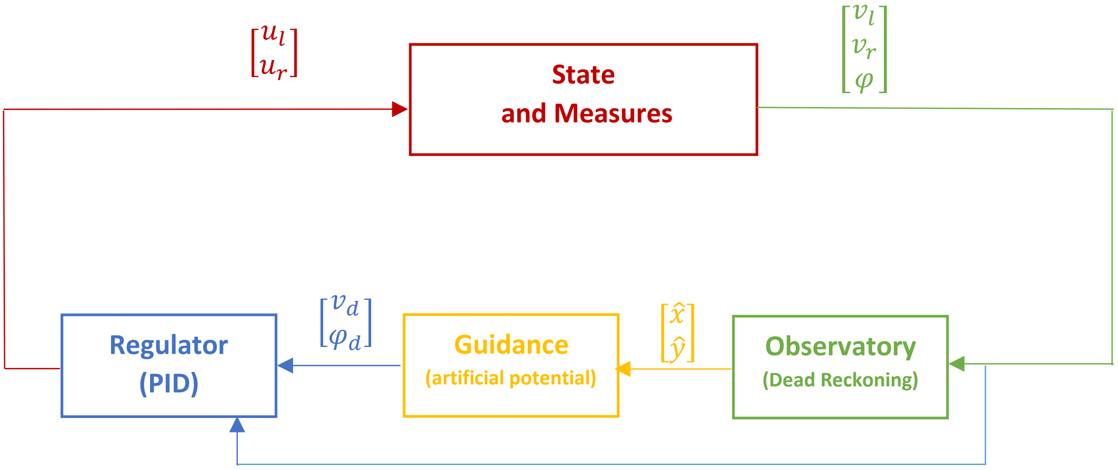


Photo 9 4-14: A test of a target reaching with obstacle avoidance

Easy situation is managed, but if the rover begins too near of an obstacle it often collides the obstacle. Moreover, if the obstacle is a large wall the rover would have some difficulties to reach the target and could collide the wall because the rover would be drawing near the wall and the sample time is large. However, with a better sample time (i.e. better encoders) the obstacle avoidance would be easier and better.

III.D. Sum up

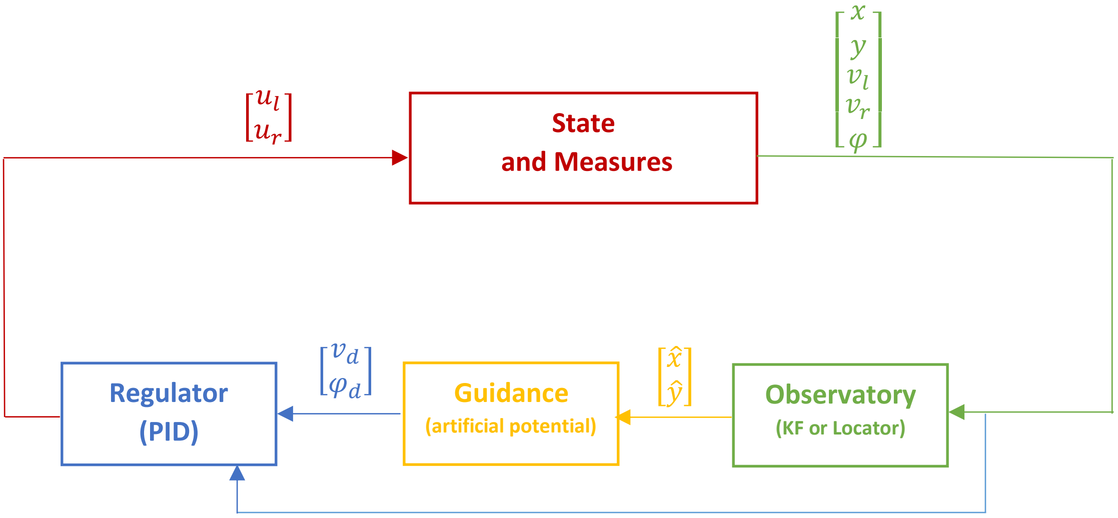
Finally, the system made is the following, for an indoor situation:



$\begin{bmatrix} u_l \\ u_r \end{bmatrix}$
State and Measures
$\begin{bmatrix} v_l \\ v_r \\ \varphi \end{bmatrix}$
Regulator (PID)
$\begin{bmatrix} v_d \\ \varphi_d \end{bmatrix}$
Guidance (artificial potential)
$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix}$
Observatory (Dead Reckoning)

Figure 20: The resulting system created for an indoor situation

For an outdoor situation:



$\begin{bmatrix} u_l \\ u_r \end{bmatrix}$
State and Measures
$\begin{bmatrix} x \\ y \\ v_l \\ v_r \\ \varphi \end{bmatrix}$
Regulator (PID)
$\begin{bmatrix} v_d \\ \varphi_d \end{bmatrix}$
Guidance (artificial potential)
$\begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix}$
Observatory (KF or Locator)

Figure 21: The resulting system created for an outdoor situation

With:

iv. u_l, u_r the pwm command for the motors of the left and right side.

v. x, y the position of the rover measured by the GPS

vi. v_l, v_r the wheels' velocity of the left and right side measured by encoders

vii. φ the heading measured by the fusion of the compass and the Gyroscope
viii. $\hat{x}, \hat{y}$ the estimated position of the rover
ix. v_d, φ_d the desired velocity and the desired heading

To sum up, here is an organigram related to the Arduino program. In this organigram, t_{LCV} is the time of the Last Computation Velocity which is also the time of the last sending data to the Arduino Motor, and t_{LRD} is the time of the Last recording data. Both Interruptions which count the number of holes of encoders are not featured in this organigram, but do not forget that for each encoder output change a function is called to counts in the Arduino Sensors board.

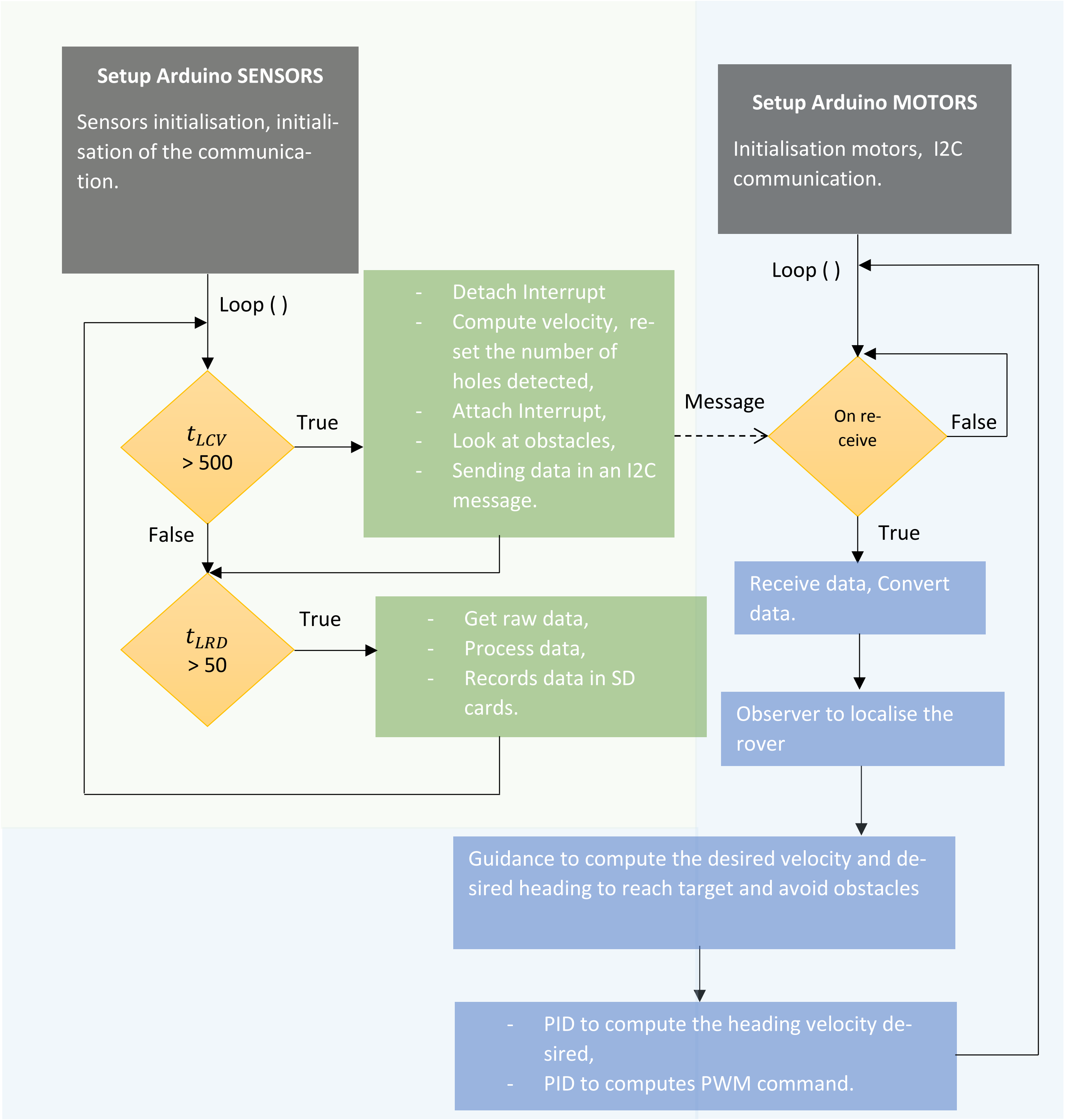

Setup Arduino SENSORS
Sensors initialisation, initialisation of the communication.
Loop ( )
t_{LCV} > 500
True
False
- Detach Interrupt
- Compute velocity, reset the number of holes detected,
- Attach Interrupt,
- Look at obstacles,
- Sending data in an I2C message.
t_{LRD} > 50
True
- Get raw data,
- Process data,
- Records data in SD cards.
Message
Setup Arduino MOTORS
Initialisation motors, I2C communication.
Loop ( )
On receive
False
True
Receive data, Convert data.
Observer to localise the rover
Guidance to compute the desired velocity and desired heading to reach target and avoid obstacles
- PID to compute the heading velocity desired,
- PID to computes PWM command.

Figure 222: The principle of the Arduino program made

This part is done, works with difficulty due to low cost sensors like encoders and GPS which have a very bad accuracy. A better system could be made and work better with more accurate sensors.

IV. Autonomous algorithm to follow a line.

IV.A. Explanation

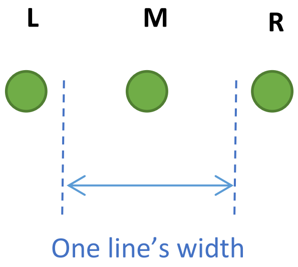


Figure 233: The positioning of the IR sensors under the rover

In this feasibility study, 3 IR sensors are used to make a simple line tracker. The layout of these sensors is described in the adjacent picture. The sensors are aligned underneath the robot front in such a way that:

- The line is only detected by the central sensor if the robot is centred on the line and parallel to the line.
- The line is detected by one or two sensors if the robot follows incorrectly the line.
- A perpendicular line is detected by all sensors.

That is why, both extremities sensors are about one line's width distant. In the initial state, at least one of the sensors must detect the line.

During the line tracking:

1. If L (resp. R) detects the line, the robot must slightly turn at the left (resp. right).
2. If only M detects the line, the robot keeps driving.
3. If L, M and R detects the line, then the rover is a perpendicular line,
    a. If the last state* was (1) ** or (2) it maybe not the line followed by the rover so it keeps driving forward.
    b. Else it rotates to 90 degrees.
4. If L, M and R detect nothing:
    a. If the last state* was (1) the robot turns at extreme left (resp. right) to try to join the line.
    b. If the last state* was (2) it stops.
    c. If the last state* was (3) it goes backward and rotates to 90 degrees.

Situations (1), (2) and (4a) are featured in the following pictures.

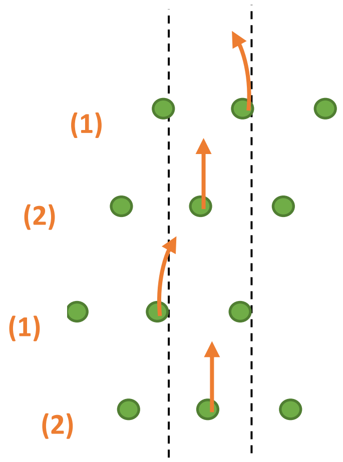


Figure 24: The straight-line example

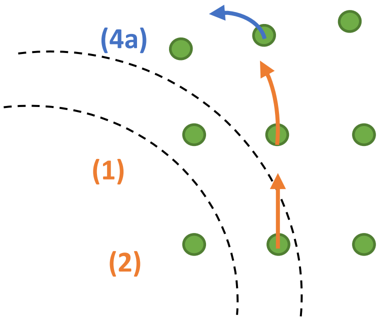


Figure 25: The turn example

With only three sensors, the rover cannot correctly follow every turn. Indeed, very large turn needed a higher rotary velocity. But an improvement had been integrated: the perpendicular turn.

When there is a perpendicular line, all sensors detect the line. At this moment, the rover must go back – against inertia- and rotate about 90 degrees.

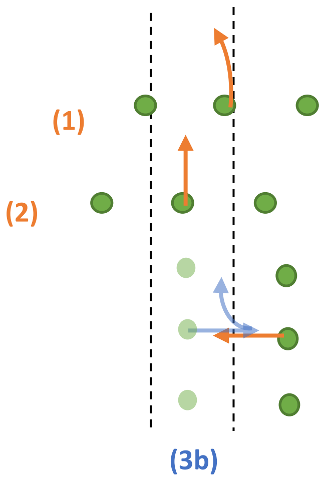


Figure 26: A perpendicular line example

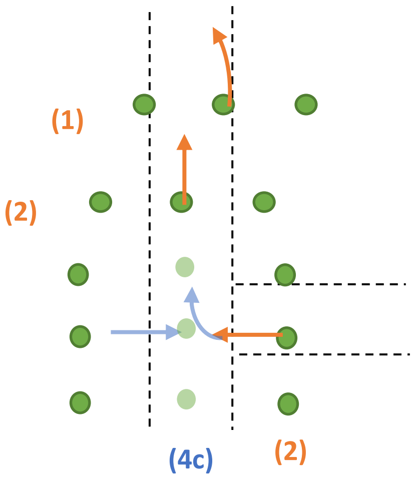


Figure 27: A T-crossing example

However, if the rover is on a perpendicular line (i.e. all sensors detect the line) after being on a line (situation 1 or 2) then it goes forward because it can be a X-crossing. After, if the line disappears it must go back – against inertia- and rotate about 90 degrees.

The decisions are summarising in the following table with 1 for a sensor which detects the line:

	L	M	R	Choice	Old L*	Old M*	Old R*
(1)	**0**	0/1	1	Slightly turn at right			
(2)	**0**	1	0	keep driving			
(1)	**1**	0/1	0	Slightly turn at left			
(3a) **	**1**	1	1	keep driving	0	1	0
(3b) **	**1**	1	1	Go backward and rotate to 90degree	1	0/1	0
					0	0/1	1
					1	1	1
(4a)	**0**	0	0	turn at extreme left	1	0/1	0
(4a)	**0**	0	0	Turn at extreme right	0	0/1	1
(4b)	**0**	0	0	Stop	0	1	0
(4c)	**0**	0	0	Go backward and rotate to 90degree	1	1	1

Table 6: The command chosen in function of the situation

*The last state (Old L, Old M, Old R) is not updated when the state is (0,0,0).
** See the improvement in the next part.

IV.B. Results and enhancement

The adhesive tape used for tests is black and around 5cm in width. So, sensors are separated by 2.5cm.

For this simple method, velocities used are constant. It is significant to choose great rotary velocities for small turns and big turns. Indeed, if there is a too high rotary speed, the rover could lose a straight line in the situation (1) or have huge oscillation. And if it is too small for consistent turn (situation 4a), the rover will lose the line quickly and will be very far to the line. The more the rover stay near the line, the more the system could work correctly.

The first tests are made in the following roads:

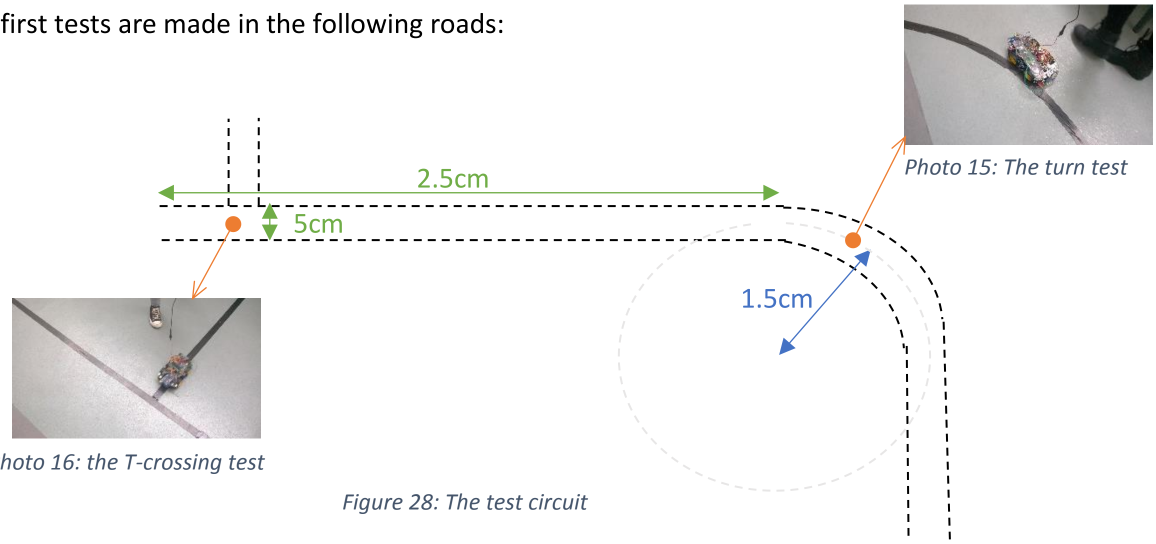


Photo 15: The turn test

Photo 16: the T-crossing test

Figure 28: The test circuit

The code[23] made for the follower line is structured like it (see the next illustration): the first Arduino board collects measurements of IR sensors and creates a message depending on the situation. This message is sent to the second Arduino board in a I2C communication. When the second Arduino

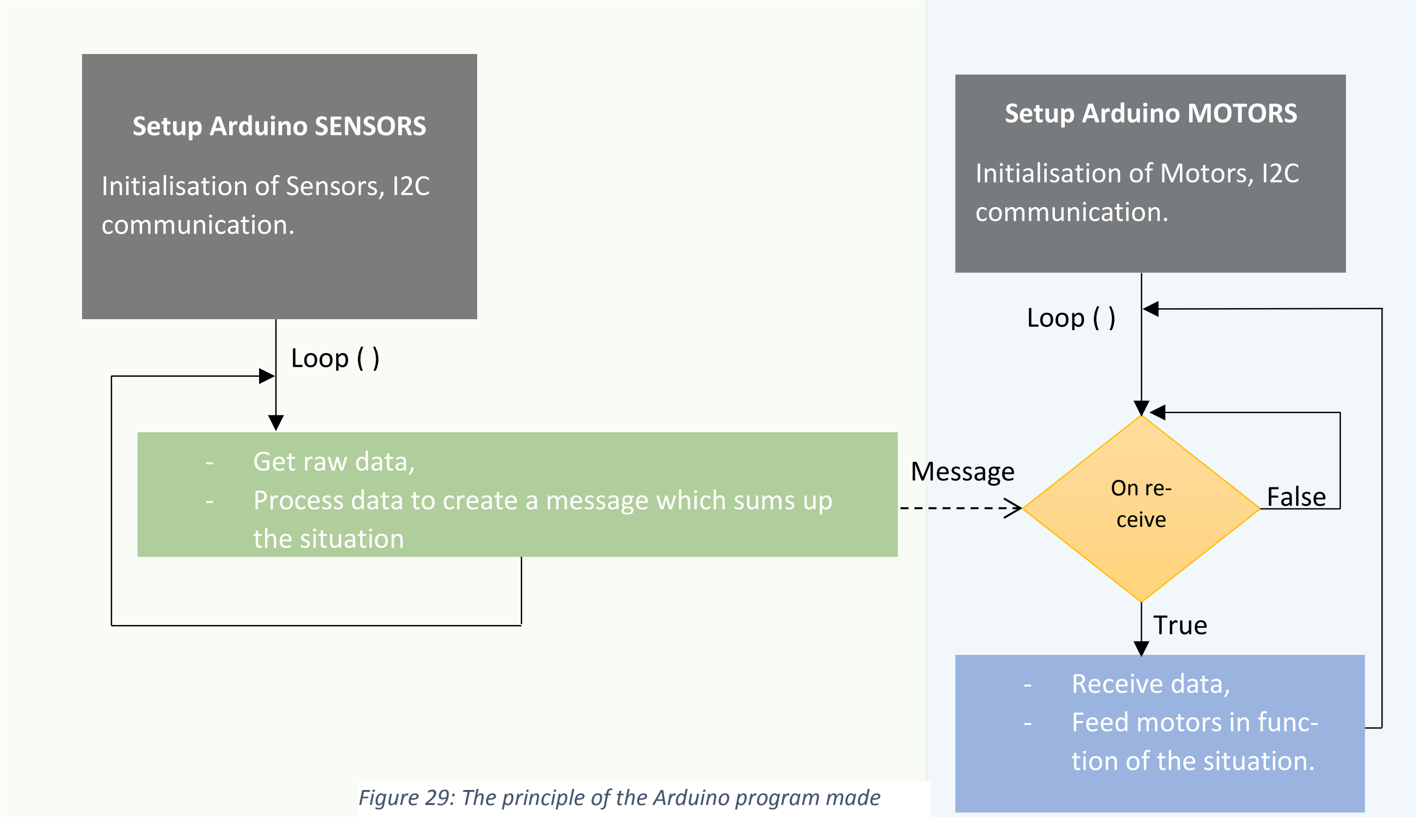

Setup Arduino SENSORS
Initialisation of Sensors, I2C communication.
Loop ( )
- Get raw data,
- Process data to create a message which sums up the situation
Message
Setup Arduino MOTORS
Initialisation of Motors, I2C communication.
Loop ( )
On re-ceive
False
True
- Receive data,
- Feed motors in func-tion of the situation.

Figure 29: The principle of the Arduino program made

[23] The code to follow a line is in the Project Arduino Code. The first part is in the pages 25-27 and second one is in the pages 36-39.

board receives the message, it collects it and feeds the motors in function of the situation translated by the message.

- **Straight line:**

The rover manages correctly a straight line but with some small oscillations. The more the rover begin aligned on the line, the less is the oscillations amplitude. During straight line tests the line was never lost. However, after a too huge turn, when the rover finds against the line it would oscillate with huge amplitude.

- **The consistent turn:**

After a too consistent turn (more than this one), The rover could fail to find against the line because it would arrive on the line with an angle too high and not detect the line. To avoid it, the sample time had been tune to be able to see the line in the worst moment (i.e. when the rover is perpendicular to the line). Here is the computing of the maximum time between two measures:

$$t_s = \frac{l}{v_{max}}$$

with t_s the sample time (s), v_{max} the maximum speed given to the rover (cm/s), l the road width (cm).

Here v_{max} corresponds to a 100 pwm command - 70 cm/s-, and l is equal to 5 cm.

So $t_s \approx 71\ ms$.

That is why the sample time chosen is 50ms.

- **T-crossing:**

Here is one result got for the T-crossing.

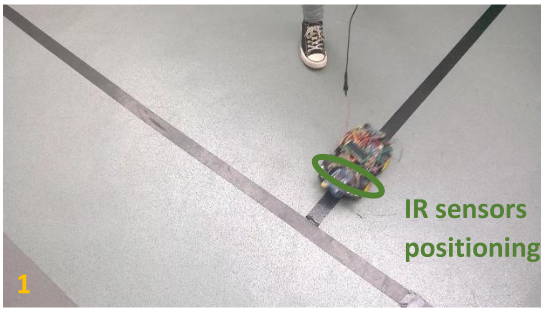
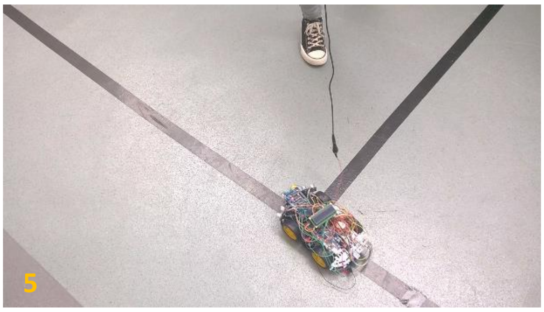


Photo 17: Situation (2), keep driving.

Photo 58: Situation (3), keep driving.

Photo 19: Situation (4c), Step 2-3, turns at right

Photo 206: Situation (4c), Step 4, keep turning at right.



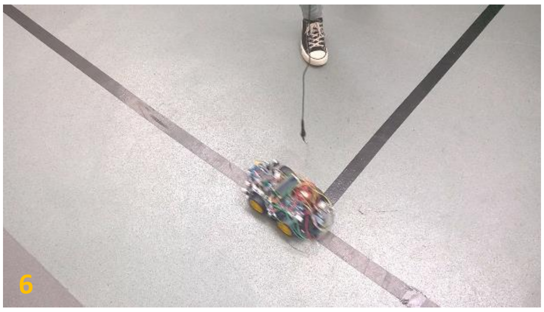


Photo 21: Situation (1), Slightly rotate at right

Photo 22: Situation (2), keep driving

We can see that the rover can manage this kind of situation without lose the line. But some time the rover does not arrive perpendicular to the line in the second picture. So, the situation considered is not the situation (3) but the situation (1). At this moment, it loses the line and the situation (4a) is running and it takes a long time to find again the line. That is why the situation (3) was become:

If L, M and R detects the line, then the rover is a perpendicular line,

- *If the last state* was (2) (i.e. the rover is aligned on the line) it maybe not the line followed by the rover so it keeps driving forward.*
- *Else it rotates to 90 degrees.*

In this way, if the rover was not correctly aligned and that there is a X-crossing, the rover would turn in the crossing.

With these improvements, if there are not very large turns, the rover can follow a line without lose is a long time. At this end of the path or in a deadlock the rover stops. Some ideas of enhancement would be to turn back and continue to follow the line after a deadlock and to manage a T-crossing without a single direction choice (here there is only T-crossing with a right turn). But the works done show that it is feasible to do a line following with a small low-cost rover and only three IR sensors.

Conclusion

This feasibility study is completely done and shows that an autonomous car could work with simple components. But it is obvious that better sensors are, better the algorithm works. So, it could be interesting to improve some sensors, particularly the encoders to have a smaller time sample.

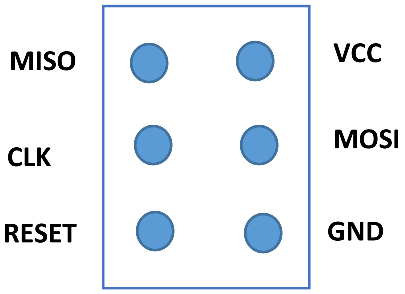
Moreover, the outdoor situation could be more tested to check its limits, and it could be interesting to more fuse the two modes: “reach a target” and “follow a line”.

This internship was stimulating, improving my autonomous and skill to solve issues. It allows me to improve consequently my knowledge in Arduino programming and to have experienced hardware issues.

Annex – Components details

Arduino Board

- ICSP



The 6 ICSP (in-circuit serial programming) pins can be used like a boot-loading or a SPI communication as with a SD board. The ICSP pins are described in the adjacent picture.

For more details about SPI and pin meaning: MicroSD card datasheet.

- Interrupt pins

The Arduino Mega board has some interrupts pins associated to number. There are referenced in this board:

Interrupt pin	2	3	18	19	20	21
Number	0	1	5	4	3	2

These interrupts pin are used to beget interruption in Arduino program. In this project, interrupts are notably used for the measure of velocity.

- Communication

Communication can be:

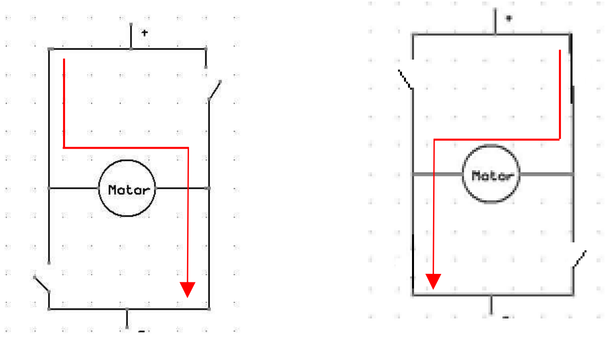
- Through an I2C communication (“wire slave receiver”). For this the first and the most important step is to connect the ground of both Arduino boards. Then, the clock (SCL pin) and data (SDA pin) must be connected.

- With a Serial communication. For this the ground of both Arduino board must be connected. Then Rx and TX pins of one of the Arduino must be connected to the TX and Rx pin of the other one.

To transfer data from sensor from one Arduino to the other one the way chosen was the I2C communication. For this it was needed to increase the length if the buffer because it was limited to 32 bytes by a constant definition in the *Twi* and *Wire* library.

Motor driver functioning

L293D can control two DC motors. Its works with two H-bridges [3] which allow the voltage to go through a motor in either direction. This change of direction begets a change on the rotation direction the motor.



Therefore, this motor shield can control 4 DC motor simultaneously, by adjusting their rotation speeds and directions.

Micro SD board

Here is the MicroSD board use for the project. It is useful notably for data collecting to check the dynamic modelling of the robot. This component uses the serial peripheral interface (SPI) to communicate with the Arduino [5]. This kind of communication is between one master devices and some slaves. Data transmission generated by the master is synchronized by the clock pulses.

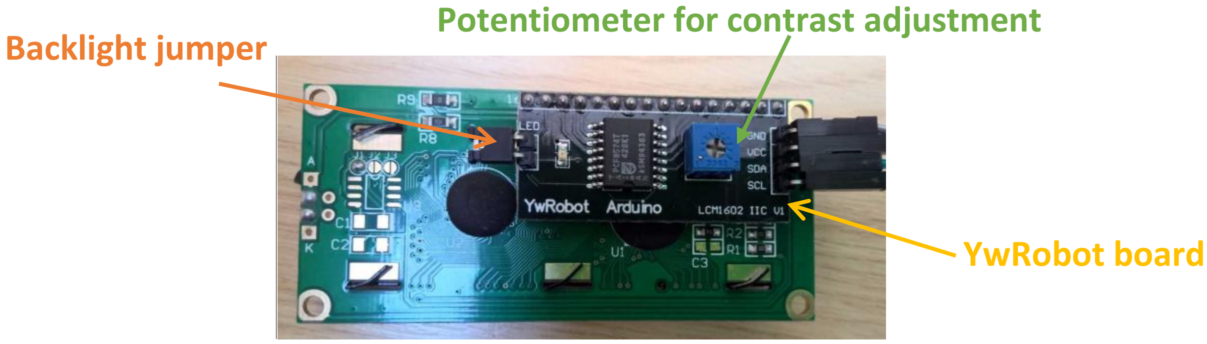
The wires of the MicroSD board are the following for the Arduino Mega [6]:

MICROSD BOARD	MEANING	ARDUINO MEGA
CS	Chip Select	Digital 53
DI	Master Output Slave Input (MOSI) Data output from master	Digital 51/ISCP pin
DO	Master Input Slave Output (MISO) Data output from slave	Digital 50/ISCP pin
CLK	Clock	Digital 52/ISCP pin
GND		GND
5V		5V

This board uses the SD library SPI and SD (Secure Digital). File system recommended for the SD card is the FAT16 (File Allocation Table). This file system uses the 8.3 format for the file Naming, for instance, filenames are composed of at the most 8 characters, following by an extension with a dot and composed of at the most 3 characters.

I2C LCD [8]

The I2C LCD Display used is the first version of it. It is composed of a YwRobot board and a LCD. The I2C (Inter-Integrated Circuit) communication uses two wires: Serial Data Line (SDA) and Serial Clock Line (SCL). It is a multi-master bus (i.e. it uses multiple master bus) and multi-slaves bus (i.e. it uses multiple slave buses). Masters generate the clock and start communication with slaves, whereas slaves receive the clock and reply to masters.



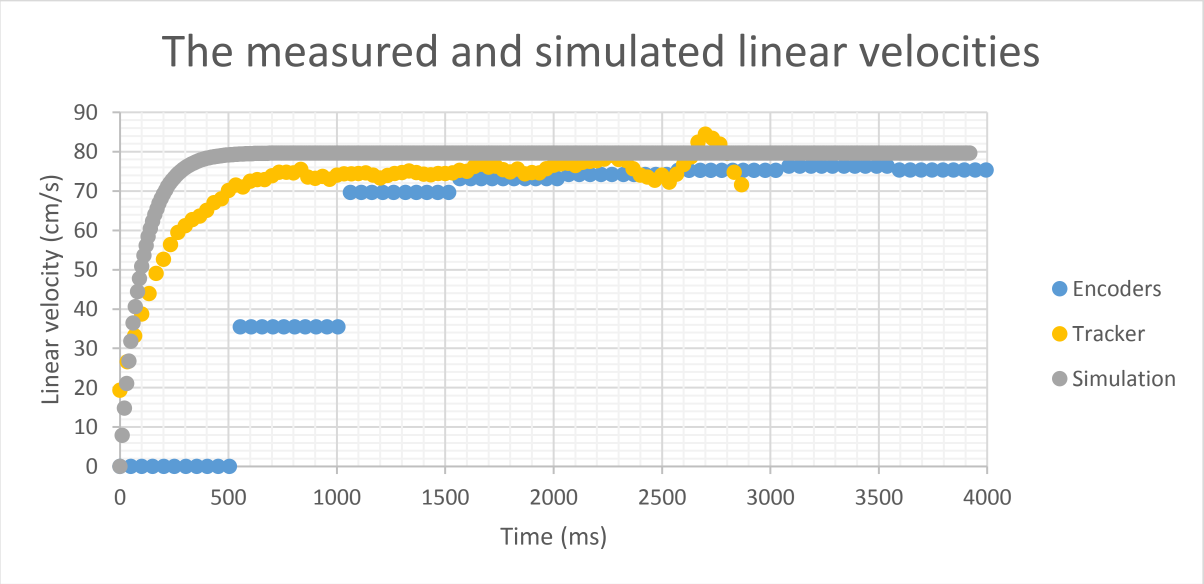
The library *LiquidCrystal_V1.2.1* is used with the LCD I2C.

Annexe - Model

```python
# -*- coding: utf-8 -*-

"""Simulation based on equation of the following researches:
Analysis and Experimental Verification for Dynamic
Modelling of A Skid-Steered Wheeled Vehicle
Wei Yu, Student Member, IEEE, Oscar Ylaya Chuy, Jr., Member, IEEE,
Emmanuel G. Collins, Jr., Senior Member, IEEE, and Patrick Hollis

Dynamic modelling and experimental validation of a skid-steered
vehicle in the pivotal steering condition.
Jun Ni and Jibin Hu

Dynamic Modelling and Motion Planning for Robotic Skid-Steered Vehicles
Nikhil Gupta


Simulation made by Sophie TUTON and Nicolas VEYLON"""


import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integrate
import time

coefStallTorque = 0.027/3  # stall-torque
coefNoLoadSpeed = (110/3)*2*np.pi/60  # no-Load speed
m = 0.500 #weight (kg)
r = 0.034 #wheel radius (m)
B = 0.131 #robot width  (m)
b = 0.025 #wheel width (m)
l = 0.25  #robot length (m)
L = 0.125 #from the beginning of a rear patch to the end of the front patch
C = 0.115 #from the end of a rear patch to the beginnig of the front patch
p = m*9.8/(4*((L-C)*b/2)) #normal pressure (Pa)
I  = m*(l**2 + (B-b)**2)/12 #Moment of inertia of a rectangle  (kg.m2)
# mu = 1.1 #friction coefficient
mu = 0.5
Kr = 5.4e-4
f = (m/2)*9.8*0.03 #rolling resistance of the motor
# alpha = 3
alpha = 1.9
m1 = m*r**2/4
m2 = r**2*I/(alpha*B**2)
M = np.array([[m1+m2, m1-m2],
              [m1-m2, m1+m2]])
Minv = np.linalg.inv(M)

kinMat = r/(alpha*B) * np.array([[alpha*B/2, alpha*B/2],[-1,1]])
invKinMat = np.linalg.inv(kinMat)


def j(x,y,q_dot,Kin,sideX,sideY):
    """Compute the shear displacement"""
    sign = (1-2*(sideX == "l"))*np.sign(Kin[1,0])  # different signs between the inner and the outer wheels
    w = q_dot[int(sideX == "r"),0]
    var = -C*(sideY == "r") + L*(sideY == "f")
    if Kin[0,0] == 0:
        R = 0
    elif Kin[1,0] == 0:
        R = 1e16
    else:
        R = abs(Kin[0,0]/Kin[1,0])
    jX = np.sign(Kin[1,0])*((R+sign*B/2+np.sign(Kin[1,0])*x)*(np.cos((var/2-y)*abs(Kin[1,0])/(r*w))-1) - y*np.sin((var/2-y)*abs(Kin[1,0])/(r*w)))
    jY = (R+sign*B/2+np.sign(Kin[1,0])*x)*(np.sin((var/2-y)*abs(Kin[1,0])/(r*w))) - var/2 + y*np.cos((var/2-y)*abs(Kin[1,0])/(r*w))
    j = np.sqrt(jX**2 + jY**2)

    return(j)

def gamma(x,y,q_dot,Kin,sideX,sideY):
    sign = (1-2*(sideX == "l"))*np.sign(Kin[1,0])  # different signs between the inner and the outer wheels
    w = q_dot[int(sideX =="r"),0]

    if Kin[0,0] == 0:
        R = 0
    elif Kin[1,0] == 0:
        R = 1e16
    else:
        R = abs(Kin[0,0]/Kin[1,0])
    gamma = np.arctan2((R+sign*B/2+np.sign(Kin[1,0])*x)*abs(Kin[1,0]) -r*w, -y*Kin[1,0])
    return(gamma)


def F(q_dot,sideX):
    functionF = lambda y,x : p*mu*(1-np.exp(-j(x,y,q_dot,Kin,sideX,"f")/Kr))*np.sin(np.pi+gamma(x,y,q_dot,Kin,sideX,"f"))
    functionR = lambda y,x : p*mu*(1-np.exp(-j(x,y,q_dot,Kin,sideX,"r")/Kr))*np.sin(np.pi+gamma(x,y,q_dot,Kin,sideX,"r"))
    a1 = integrate.dblquad(functionF,-b/2,b/2,lambda x:C/2,lambda x:L/2)[0]
    a2 = integrate.dblquad(functionR,-b/2,b/2,lambda x:-L/2,lambda x: -C/2)[0]
    F = (a1+a2)
    return(F)


def DynMod(VL,VR,dt,q_dot):
    #motors equations
    stallTorqueR = VR*coefStallTorque
    stallTorqueL = VL*coefStallTorque
    noLoadSpeedR = VR*coefNoLoadSpeed
    noLoadSpeedL = VL*coefNoLoadSpeed

    stallTorque = np.array([[stallTorqueL,stallTorqueR]]).T
    if VL == 0 and VR == 0:
        stallMatrix = np.array([[0,0],
                                [0,0]])
    elif VL == 0:
        stallMatrix = np.array([[0,0],
                                [0,stallTorqueR/noLoadSpeedR]])
    elif VR ==0:
        stallMatrix = np.array([[stallTorqueL/noLoadSpeedL,0],
                                [0,0]])
    else :
        stallMatrix = np.array([[stallTorqueL/noLoadSpeedL,0],
                                [0,stallTorqueR/noLoadSpeedR]])

    if abs(q_dot[0,0] - q_dot[1,0]) < 0.001 or True:
        C = np.array([[r*np.sign(Kin[0,0])*f,r*np.sign(Kin[0,0])*f]]).T
    else:
        C = np.array([[r*(F(q_dot,"l")+np.sign(Kin[0,0])*f),r*(F(q_dot,"r")+np.sign(Kin[0,0])*f)]]).T

    #RK4
    k1 = np.dot(Minv, 2*(stallTorque - np.dot(stallMatrix,q_dot)) - C)
    k2 = np.dot(Minv, 2*(stallTorque - np.dot(stallMatrix,q_dot + dt*k1/2)) - C)
    k3 = np.dot(Minv, 2*(stallTorque - np.dot(stallMatrix,q_dot + dt*k2/2)) - C)
    k4 = np.dot(Minv, 2*(stallTorque - np.dot(stallMatrix,q_dot + dt*k3)) - C)
```

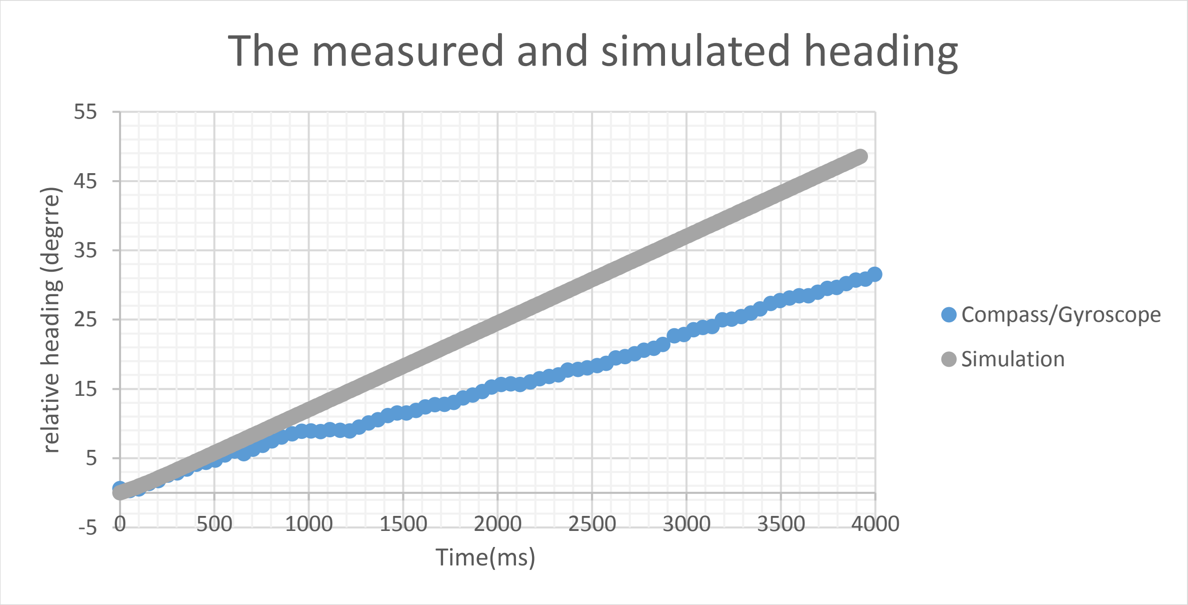
```python


    return(q_dot + dt*(k1 + 2*k2 + 2*k3 +k4)/6)

def KinMod(X,dt,q_dot):
    vL, vR = q_dot[0,0]*r, q_dot[1,0]*r
    Kin = kinMat.dot(q_dot)
    X = X + dt*np.array([[Kin[0,0]*np.cos(X[2,0])],
                         [Kin[0,0]*np.sin(X[2,0])],
                         [Kin[1,0]]])
    return(X, vL, vR, Kin)

def Input2Volt(i):
    """Convert the command (0 -> 255) into a voltage"""
    if i < 55:
        v = 0
    else:
        v = -5.138236173E-09*i**4 + 4.146027104E-06*i**3 - 0.001239302393*i**2 + 0.1859126277*i - 3.829291658
    return(v)


if __name__ == "__main__":
    q_dot = np.array([[0.0000000001,0.00000000001]]).T
    X = np.array([[0,0,0]]).T
    tmin = 0
    tmax = 3
    dt = 0.001
    T = np.arange(tmin,tmax,dt)
    uL = [80]*len(T)
    uR = [80]*len(T)
    i = 0

    records = [] #records x,y,phi,vL,vR

    X,vL,vR,Kin = KinMod(X,dt,q_dot) #X = x,y,phi ; Kin = v, phidot
    V = []
    for t in T:
        VL,VR = Input2Volt(uL[i]), Input2Volt(uR[i])
        #dynamic
        q_dot = DynMod(VL,VR,dt,q_dot)
        #kinematic
        X,vL,vR,Kin = KinMod(X,dt,q_dot) #X = x,y,phi ; Kin = v, phidot
        V.append(Kin[0,0])
        records.append([t, vL, vR, X[2,0], X[0,0], X[1,0]])
        i+=1

     #recording in file
     file = open("simu_forward_speed_70.txt",'w')
    file.write("t(ms)\tvL(cm/s)\tvR(cm/s)\tphi(°)\tX(m)\tY(m)\n")
    for i in range(0, len(records), 33):
            rec = records[i]
           file.write("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\n".format(rec[0]*1000, rec[1]*100, rec[2]*100, rec[3]*180/np.pi, rec[4], rec[5]).replace('.',','))
    file.close()

    records = np.array(records).T

    v80 = records[1,-1]

    v_exp = 0.58215*(1-np.exp(-T/0.350))

     #plot data
    plt.figure(1)
    plt.subplot(221)
    plt.plot(T,records[1,:],label = "vL")
    plt.plot(T,records[2,:],label = "vR")
    plt.plot(T,V,label = "v")

    plt.grid()
    plt.legend()
    plt.xlabel("Time ($s$)")
    plt.ylabel("Velocity ($ms^{-1}$)")



    plt.subplot(222)
    plt.plot(T,records[3,:],label = "φ")
    plt.grid()
    plt.legend()
    plt.xlabel("Time ($s$)")
    plt.ylabel("φ ($rad$)")


    plt.subplot(223)
    plt.plot(T,records[4,:], label = "X")
    plt.grid()
    plt.legend()
    plt.xlabel("Time ($s$)")
    plt.ylabel("X ($m$)")

    plt.subplot(224)
    plt.plot(T,records[5,:], label = "Y")
    plt.xlabel("Time (s)")
    plt.ylabel("Velocity ($ms^{-1}$)")
    plt.grid()
    plt.legend()
    plt.xlabel("Time ($s$)")
    plt.ylabel("Y ($m$)")


    plt.show()


```

Here there is one result for a command of 80 for left motors and 150 for the right one's:



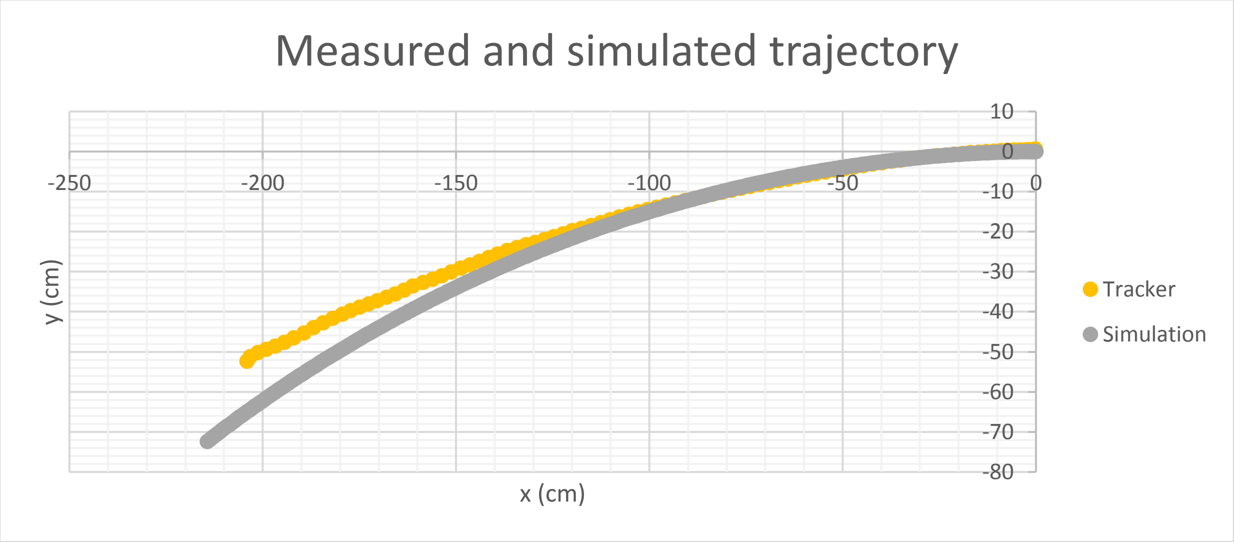
The velocity of the rover, measured by the encoder (blue), measured by Tracker (yellow) and simulated (grey) for a test which a command of 80 in left motors and 150 in rights motors.

In this test, we can observe that the convergence speed is nearer with a difference about 4cm/s. Moreover, the time for the first value at 95% of this value is about 600ms for the rover of test however the simulated one is about 300ms.



The heading of the rover, measured by the encoder (blue), and simulated (grey) for a test which a command of 80 in left motors and 150 in right motors.

The heading is increasing in both case but the simulated evolution is quicker that the current rover. Indeed, the heading values are identical for the first half second, but after the difference is about 5t with t the time in seconds. Indeed, the error of heading is about 5 degrees at 1second, and 10 degrees at 2second, about 20degree at 4s…



We can see that the trajectory of the simulated rover -the grey curve- and of the current rover -the yellow curve-are similar at the begin until one meter of travel in the X-axis. Then, the current rover turns slighter than the simulated one.

Annex - Controller

- Equation of the controller

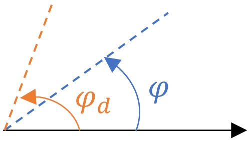
Equations are the followings:

$$
\begin{cases}
\dot{\varphi}_\mathrm{d} = K_p \cdot e_\varphi + K_i \cdot \int e_\varphi dt + K_d \cdot \frac{de_\varphi}{dt} \\
\omega_{l_d} = \frac{1}{r}(v_d + \dot{\varphi}_\mathrm{d} \cdot \alpha \cdot \frac{B}{2}) \\
\omega_{r_d} = \frac{1}{r}(v_d - \dot{\varphi}_\mathrm{d} \cdot \alpha \cdot \frac{B}{2}) \\
\mathrm{u_l} = K'_p \cdot e_{\omega_l} + K'_i \cdot \int e_{\omega_l} dt + K'_d \cdot \frac{de_{\omega_l}}{dt} \\
\mathrm{u}_r = K'_p \cdot e_{\omega_r} + K'_i \cdot \int e_{\omega_r} dt + K'_d \cdot \frac{de_{\omega_r}}{dt}
\end{cases}
$$

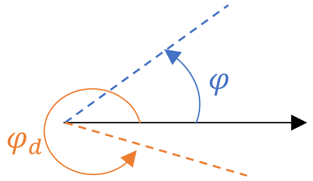
With $(K_p, K_i, K_d) = (2, 6, -0.1)$
$(K'_p, K'_i, K'_d) = (3, 0.3, 1.2)$

And e_φ the error in heading,
e_{ω_l} and e_{v_r} the error in velocity.

Errors are given by:



case 1



case 2

$$
e_{\omega_l} = \omega_{l_d} - \frac{v_l}{r} \qquad e_{\omega_r} = \omega_{r_d} - \frac{v_r}{r}
$$

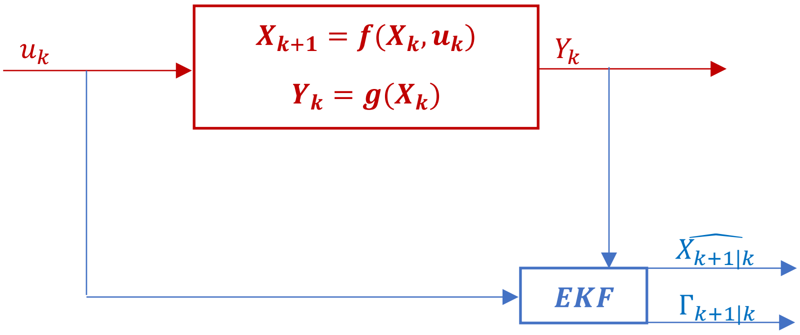
and

$$
e_\varphi = \begin{cases} \varphi_d - \varphi & if\ |\varphi_d - \varphi| < \pi \text{ (Case 1)} \\ \frac{\varphi - \varphi_d}{|\varphi_d - \varphi|} \cdot (360 - |\varphi_d - \varphi|) & else \text{ (Case 2)} \end{cases}
$$

Annex - Observer

- Equation of the EKF

To simplify the context of the explanation is the one featured in the following illustration. As we can see, the Extended Kalman Filter use the command u_k and the measures Y_k to compute a prediction of the next state $\widehat{X_{k+1|k}}$ and the next covariance matrix $\Gamma_{k+1|k}$.



It is composed of two step: the correction and the prediction. The extended Kalman Filter is having the advantage to be nonlinear whereas the Kalman Filter is linear. However, if the first value of the state is not correctly known, the filter may diverge.

The system is described by these following equations:

$$X_{k+1} = f(X_k, u_k)$$

$$Y_k = g(X_k)$$

With X_{k+1} the state vector at the discretized time k+1, u_k the command and Y_k the measures at the time k. The state vector is not perfectly known, that is why there is a non-zero covariance matrix Γ_k corresponding to the incertitude on X_k.

Let us linearize X_{k+1} and Y_k around $\widehat{X_k}$:

$$X_{k+1} = f(\widehat{X_k}, u_k) + \frac{\partial f}{\partial X_k}(\widehat{X_k}, u_k) \cdot (X_k - \widehat{X_k}) + \alpha_k$$

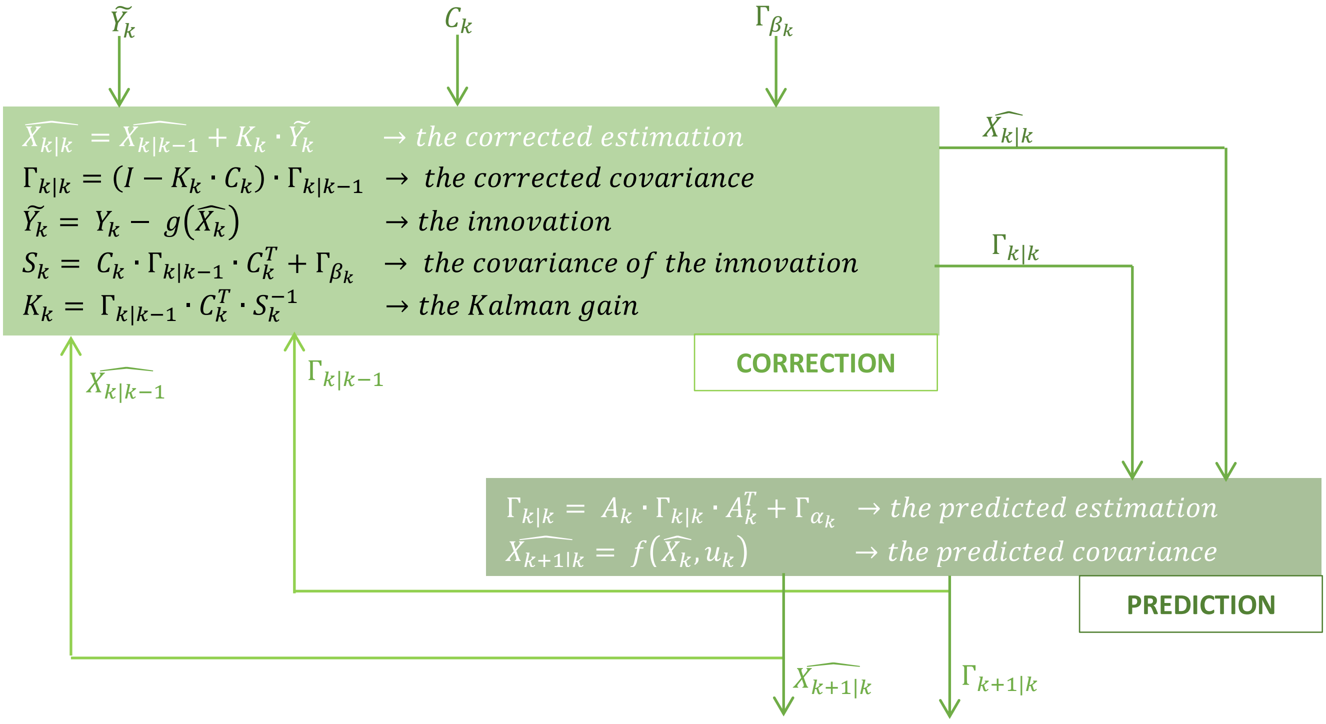
$$Y_k = g(\widehat{X_k}) + \frac{dg}{dX_k}(\widehat{X_k}) \cdot (X_k - \widehat{X_k}) + \beta_k$$

With α_k and β_k errors of linearization which correspond to Gaussian noise white in time. Thanks to these equations we can compute the expectation $\widehat{X_{k+1|k}}$. Before, let us use these following notation: $A_k = \frac{\partial f}{\partial X}(\widehat{X_k}, u_k)$ and $C_k = \frac{dg}{dX}(\widehat{X_k})$.

$$X_{k+1} = A_k \cdot X_k \underbrace{+ f(\widehat{X_k}, u_k) - A_k \cdot \widehat{X_k}}_{\textit{Like the command in a non-extended Kalman Filter}} + \alpha_k$$

$$\underbrace{Y_k - g(\widehat{X_k}) + C_k \cdot \widehat{X_{k|k}}}_{\textit{Like the innovation in a non-extended Kalman Filter}} = C_k \cdot X_k + \beta_k$$

That is why, EKF equation are the followings:



$\widehat{X_{k|k}} = \widehat{X_{k|k-1}} + K_k \cdot \widetilde{Y_k}$ → *the corrected estimation*

$\Gamma_{k|k} = (I - K_k \cdot C_k) \cdot \Gamma_{k|k-1}$ → *the corrected covariance*

$\widetilde{Y_k} = Y_k - g(\widehat{X_k})$ → *the innovation*

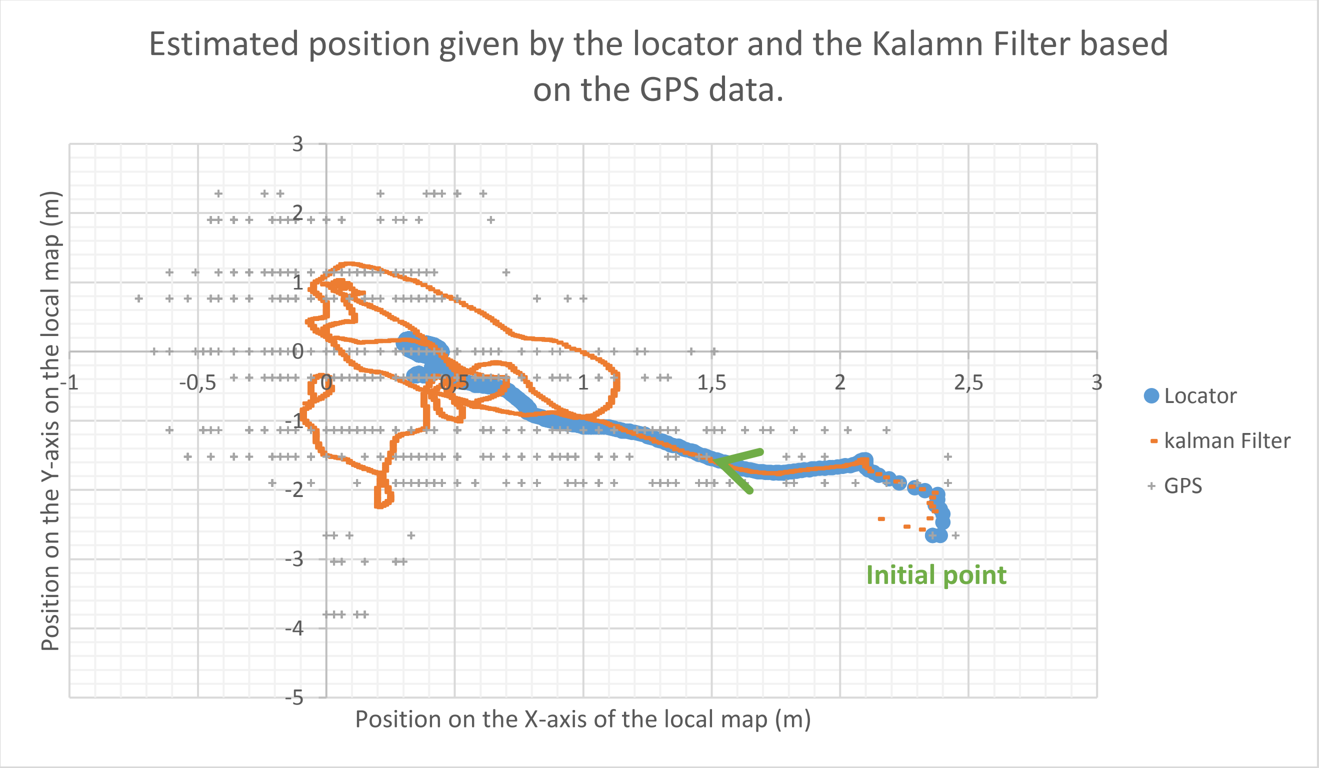
$S_k = C_k \cdot \Gamma_{k|k-1} \cdot C_k^T + \Gamma_{\beta_k}$ → *the covariance of the innovation*

$K_k = \Gamma_{k|k-1} \cdot C_k^T \cdot S_k^{-1}$ → *the Kalman gain*

$\Gamma_{k|k} = A_k \cdot \Gamma_{k|k} \cdot A_k^T + \Gamma_{\alpha_k}$ → *the predicted estimation*

$\widehat{X_{k+1|k}} = f(\widehat{X_k}, u_k)$ → *the predicted covariance*

- **Estimated positions given by the locator and the Kalman Filter based on GPS data**


Estimated position given by the locator and the Kalamn Filter based on the GPS data.

We can observe the GPS noise in grey cross. After few time, it seemed centred on the rover positioning (0,0). But it begins at the position (2.45, -2.66). That is why the Locator and the Kalman Filter begins the estimate a position around (2.45, -2.66). and which go forward (0,0). But they not converge at (0,0): the Kalman filter turns around (0.5,0) whereas the locator converges at (0.5,0).

- The dead reckoning result

Test 1: Straight Line at the initial heading (2s)

The last point computed with the dead reckoning is $\begin{cases} x = 1.158\ m \\ y = 0.011\ m \end{cases}$, whereas the last point got is $\begin{cases} x = 1.176\ m \\ y = 0.01\ m \end{cases}$. So, there is a difference of less than 2cm in the x-axis, and of less than 1cm on the y-axis.

Test 2: Straight Line at the initial heading (6s)

The last point computed with the dead reckoning is $\begin{cases} x = 3.82\ m \\ y = 0\ m \end{cases}$, whereas the last point got is $\begin{cases} x = 3.91\ m \\ y = 0.04\ m \end{cases}$. So, there is a difference of less than 10cm in the x-axis, and of 4cm on the y-axis.

Test 3: Straight Line at the initial heading (12s)

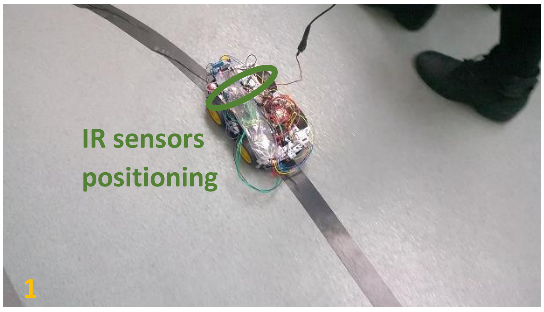
The last point computed with the dead reckoning is $\begin{cases} x = 8.20\ m \\ y = -0.022\ m \end{cases}$, whereas the last point got is $\begin{cases} x = 8.05\ m \\ y = -0.08\ m \end{cases}$. So, there is a difference of 15cm in the x-axis, and about 6cm on the y-axis.

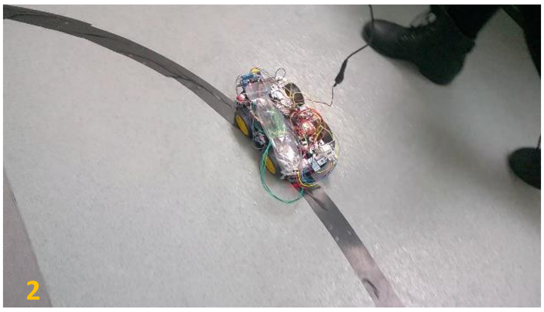
Test 4: Step 1(4s): Straight Line at the initial heading
Step 2(2s): Straight Line at PI/6 of the initial heading

The last point computed with the dead reckoning is $\begin{cases} x = 3.57\ m \\ y = 1.13\ m \end{cases}$, whereas the last point got is $\begin{cases} x = 3.74\ m \\ y = 1.09\ m \end{cases}$. So, there is a difference of 17cm in the x-axis, and of 4cm on the y-axis.

Annex - Line following

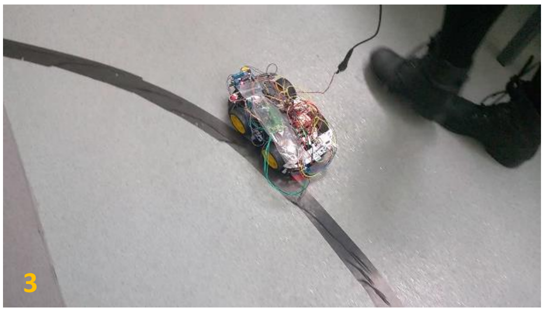
Here is one result got for the turn.

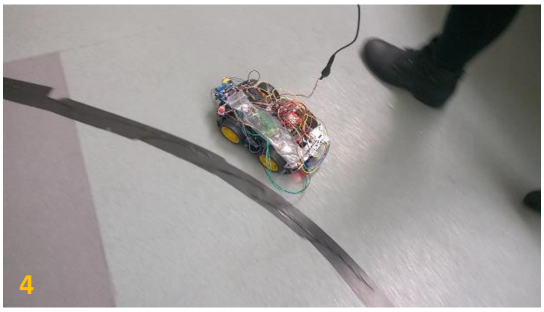




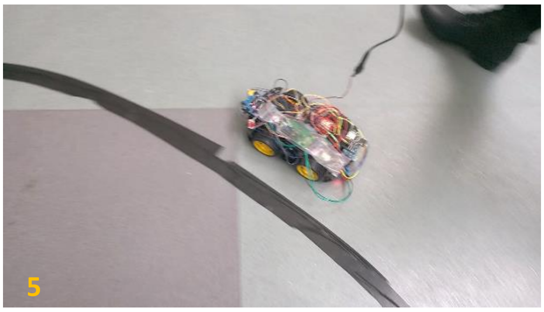
Situation (2), keep driving forward

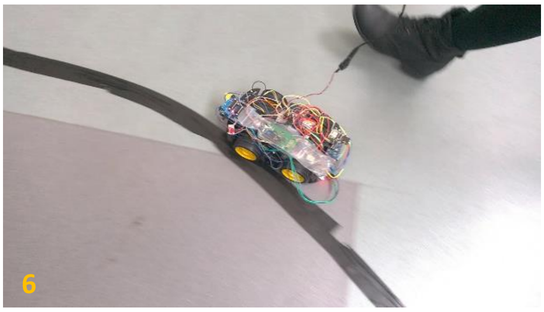
Situation (1), Slightly rotate at left



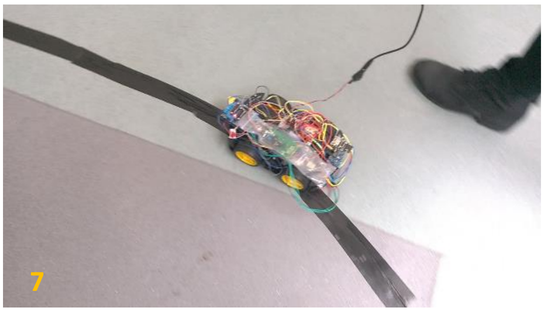


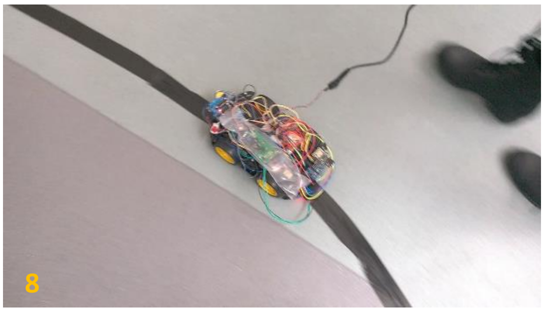
Situation (4a), turn at left





Situation (4a), turn at left





Situation (1), Slightly turn at left

Situation (2), keep driving forward

Annex - Arduino Code

The Arduino code is in the *Project Arduino Code* pdf. The code made manage some different mode like “The Line following” and “The target reaching”. But in “The target reaching” mode it is needed to choose between the indoor mode and the outdoor mode. And then in the outdoor mode it can choose between “GPS target coordinates” and “relative target coordinates.” To choose the mode follow these instructions:

TO FOLLOW A LINE:

1. In both "Constant.h" file, uncomment the definition of LINE_FOLLOWER.
(about l.26 for MainSensorsBoard and l.23 for MainMotorsBoard)

2. Then upload programs

TO REACH A TARGET:

1. In both "Constant.h" file, comment the definition of LINE_FOLLOWER.
(about l.26 for MainSensorsBoard and l.23 for MainMotorsBoard)

2. INDOOR/OUTDOOR:

FOR INDOOR SITUATION:

2a. Comment each line which refers to the GPS in the MainSensorsBoard.cpp and MainMotorsBoard.cpp where there is a comment.
For this, Ctrl + F --> GPS and read comments.

2b. In MainMotorsBoard.cpp choose the DeadReckoning observer
(about l.293)

FOR OUTDOOR SITUATION:

2a. Uncomment each line which refers to the GPS in the MainSensors-Board.cpp and MainMotorsBoard.cpp where there is a comment.
For this, Ctrl + F --> GPS and read comments.

2b. In MainMotorsBoard.cpp choose the KalmanFilter or the ComputeXY (locator) observer (about l.293)

2c. In MainMotorsBoard.cpp adapt the initial state to the observer choose (about l.154) and the covariance matrix if the observer is the Kalman Filter.

3. TARGET COORDINATE:

FOR GPS COORDINATE (ONLY IF GPS IS USED):

3a. Uncomment each line which refers to the the target position in the Main-SensorsBoard.cpp and MainMotorsBoard.cpp where there is a comment.
For this, Ctrl + F --> GPS and read comments.
WARNING: Comment two lines in the MainMotorsBoard.cpp (about l.258)

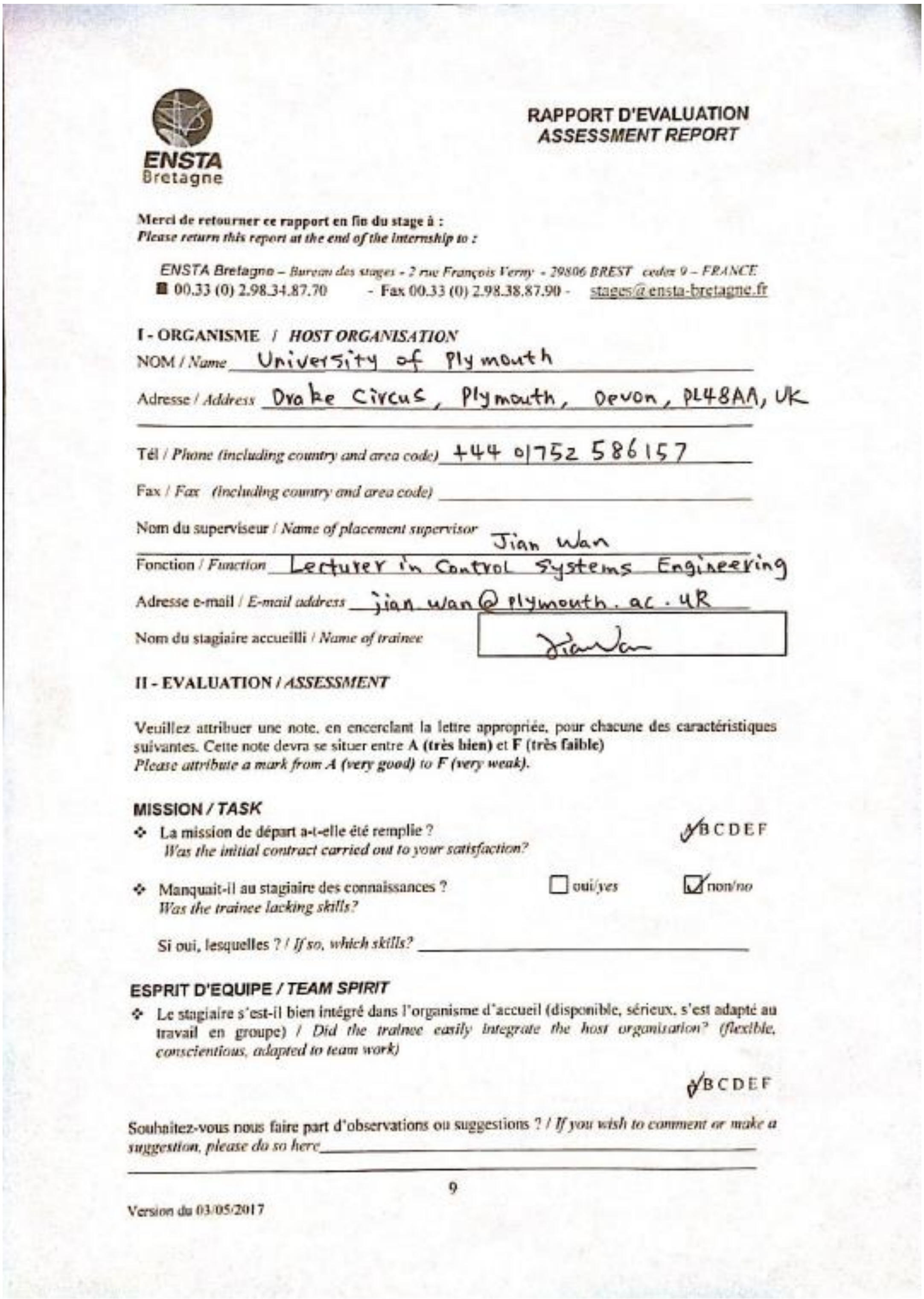
FOR RELATIVE COORDINATE:

3a. Comment each line which refers to the the target position in the Main-SensorsBoard.cpp and MainMotorsBoard.cpp where there is a comment. For this, Ctrl + F --> GPS and read comments.
WARNING: Uncomment two lines in the MainMotorsBoard.cpp (about l.258)

3b. Write the relative coordinate in the lines uncommented (about l.258)

4. Then upload programs

Annex – ASSESSMENT REPORT



ENSTA Bretagne

RAPPORT D'EVALUATION
ASSESSMENT REPORT

Merci de retourner ce rapport en fin du stage à :
Please return this report at the end of the internship to :

ENSTA Bretagne – *Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – FRANCE*
☎ 00.33 (0) 2.98.34.87.70 - Fax 00.33 (0) 2.98.38.87.90 - stages@ensta-bretagne.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* University of Plymouth

Adresse / *Address* Drake Circus, Plymouth, Devon, PL48AA, UK

Tél / *Phone (including country and area code)* +44 01752 586157

Fax / *Fax (including country and area code)* ____

Nom du superviseur / *Name of placement supervisor* Jian Wan

Fonction / *Function* Lecturer in Control Systems Engineering

Adresse e-mail / *E-mail address* jian.wan@plymouth.ac.uk

Nom du stagiaire accueilli / *Name of trainee* JianWan

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une note, en encerclant la lettre appropriée, pour chacune des caractéristiques suivantes. Cette note devra se situer entre A **(très bien)** et **F** (très faible)
Please attribute a mark from A (very good) to F (very weak).

MISSION / *TASK*

- La mission de départ a-t-elle été remplie ?
  Was the initial contract carried out to your satisfaction? [✓ A] B C D E F

- Manquait-il au stagiaire des connaissances ? ☐ oui/*yes* ☑ non/*no*
  Was the trainee lacking skills?

  Si oui, lesquelles ? / *If so, which skills?* ____

ESPRIT D'EQUIPE / *TEAM SPIRIT*

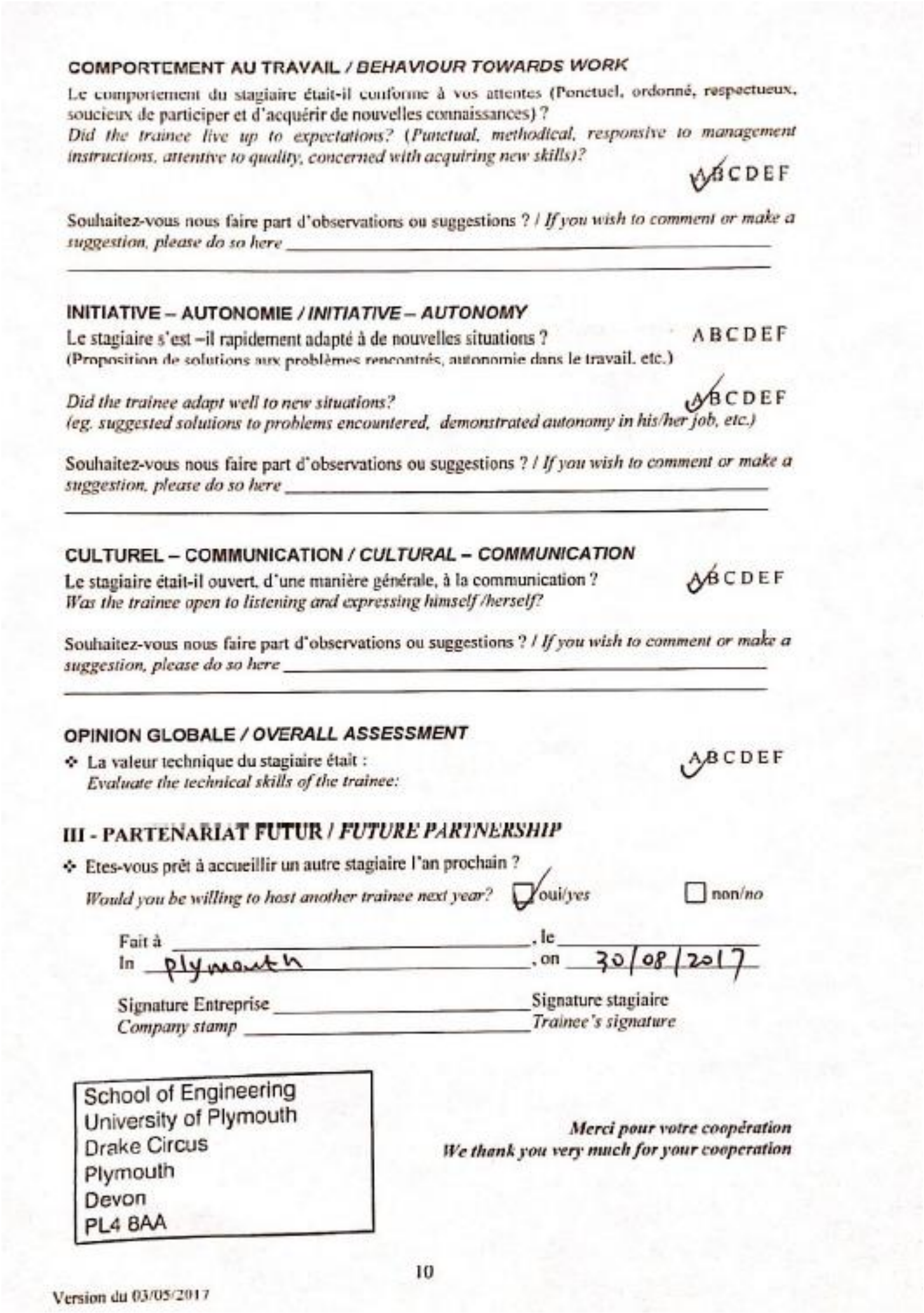
- Le stagiaire s'est-il bien intégré dans l'organisme d'accueil (disponible, sérieux, s'est adapté au travail en groupe) / *Did the trainee easily integrate the host organisation? (flexible, conscientious, adapted to team work)*

  [✓ A] B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* ____

9

Version du 03/05/2017



COMPORTEMENT AU TRAVAIL / *BEHAVIOUR TOWARDS WORK*

Le comportement du stagiaire était-il conforme à vos attentes (Ponctuel, ordonné, respectueux, soucieux de participer et d'acquérir de nouvelles connaissances) ?
Did the trainee live up to expectations? (Punctual, methodical, responsive to management instructions, attentive to quality, concerned with acquiring new skills)?

[✓ A] B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* ______

INITIATIVE – AUTONOMIE / *INITIATIVE – AUTONOMY*

Le stagiaire s'est –il rapidement adapté à de nouvelles situations ? A B C D E F
(Proposition de solutions aux problèmes rencontrés, autonomie dans le travail, etc.)

Did the trainee adapt well to new situations? [✓ A] B C D E F
(eg. suggested solutions to problems encountered, demonstrated autonomy in his/her job, etc.)

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* ______

CULTUREL – COMMUNICATION / *CULTURAL – COMMUNICATION*

Le stagiaire était-il ouvert, d'une manière générale, à la communication ? [✓ A] B C D E F
Was the trainee open to listening and expressing himself /herself?

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* ______

OPINION GLOBALE / *OVERALL ASSESSMENT*

- La valeur technique du stagiaire était : [✓ A] B C D E F
  Evaluate the technical skills of the trainee:

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

- Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?
  Would you be willing to host another trainee next year? ☑ oui/yes □ non/no

Fait à ______, le ______
In Plymouth, on 30/08/2017

Signature Entreprise ______ Signature stagiaire
Company stamp ______ *Trainee's signature*

School of Engineering
University of Plymouth
Drake Circus
Plymouth
Devon
PL4 8AA

Merci pour votre coopération
We thank you very much for your cooperation

10

Version du 03/05/2017

Bibliography

[1] Robotshop, “Arduino Mega 2560 Datasheet,” http://www.robotshop.com/media/files/PDF/ArduinoMega2560Datasheet.pdf.

[2] Tiptopboards, “Controleur de moteurs L293D pour Arduino,” http://tiptopboards.com/89-controleur-de-moteurs-l293d-pour-arduino.html.

[3] Adafruit-Industries, “Adafruit Motor Shield,” https://cdn-learn.adafruit.com/downloads/pdf/adafruit-motor-shield.pdf, 2015.

[4] Cytron-Technologies-Sdn.Bhd, “Product User’s Manual – HC-SR04 Ultrasonic Sensor,” https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-x2qR4vP8saG73rE/edit, 2013.

[5] Wei Yu, Student Member, IEEE, Oscar Ylaya Chuy, Jr., Member, IEEE,, “Analysis and Experimental Verification for Dynamic,” 2010.

[6] Kevin J Worrall and Euan W McGookin, “A MATHEMATICAL MODEL OF A LEGO, DIFFERENTIAL DRIVE ROBOT,” International Control Conference, 2006.

[7] “Mass matrix,” https://en.wikipedia.org/wiki/Mass_matrix, 2017.

[8] Ron Kurtus, "Rolling Friction / Rolling Resistance," http://www.school-for-champions.com/science/friction_rolling.htm#.WaaXarpFyUl, 2016.

[9] IUT Lyon1, “Dimensionnement Des Structures”.

[10] The Engineering ToolBox, “Rolling friction and rolling resistance,” http://www.engineeringtoolbox.com/rolling-friction-resistance-d_1303.html.

[11] Tracker, “Video Analysis and Modeling Tool,” http://physlets.org/tracker/index.html.

[12] Sparkfun, “Using the Logic Level Converter,” https://learn.sparkfun.com/tutorials/using-the-logic-level-converter.

[13] L. Jaulin, “Mobile Robotics,” 2015.

[14] Fethi MATOUI, Boumedyen BOUSSAID, Mohamed Naceur ABDELKRIM, “Local minimum solution for the potential field method in multiple robot motion planning task,” http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7505223&tag=1, Monastir,Tunisia, 2015.

[15] Ron Robotics, rakeshmondal.info, “L293D Motor Driver IC,” http://www.rakeshmondal.info/L293D-Motor-Driver, 2013.

[16] Arduino-Reference, “SPI,” https://www.arduino.cc/en/Reference/SPI.

[17] Arduino-Tutorial, “Card Info,” https://www.arduino.cc/en/Tutorial/CardInfo.

[18] Henry's-Bench, “YWRobot LCM1602 IIC V1 LCD Arduino Tutorial,” http://henrysbench.capnfatz.com/henrys-bench/arduino-displays/ywrobot-lcm1602-iic-v1-lcd-arduino-tutorial/.