

OFC: Organisateur des feux de circulation

Présenté par: BOUNOUA Ilyas

N°SCEI: 15546

Session 2023

Plan de présentation

I. Introduction

1. Contextualisation
2. Problématique
3. Cahier des charges fonctionnel

II. Objectifs

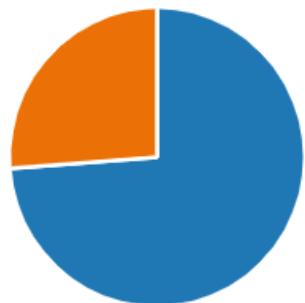
1. Choisir la motorisation convenable
2. Identifier les paramètres du moteur
3. Asservir en position le moteur
4. Établir un ordre de priorité
5. Commander les feux de circulation

III. Réalisation d'un prototype du système

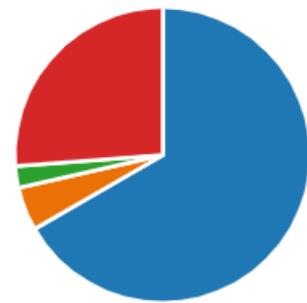
IV. Conclusion

I. Introduction

- Résultats du questionnaire fait à l'aide de «Microsoft Forms» dont la majorité des participants sont des adultes.



74%
Étaient un jour en retard à cause des feux de circulation.



67%
Utilisent la voiture comme moyen de transport.

Figure 01: Résultats du questionnaire.

I.1. Contextualisation

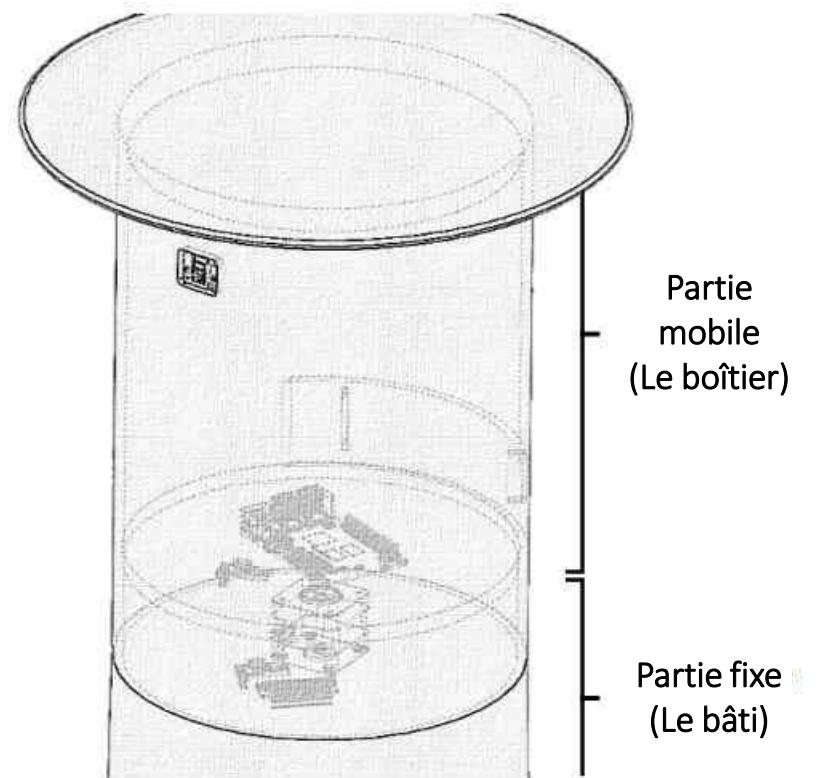
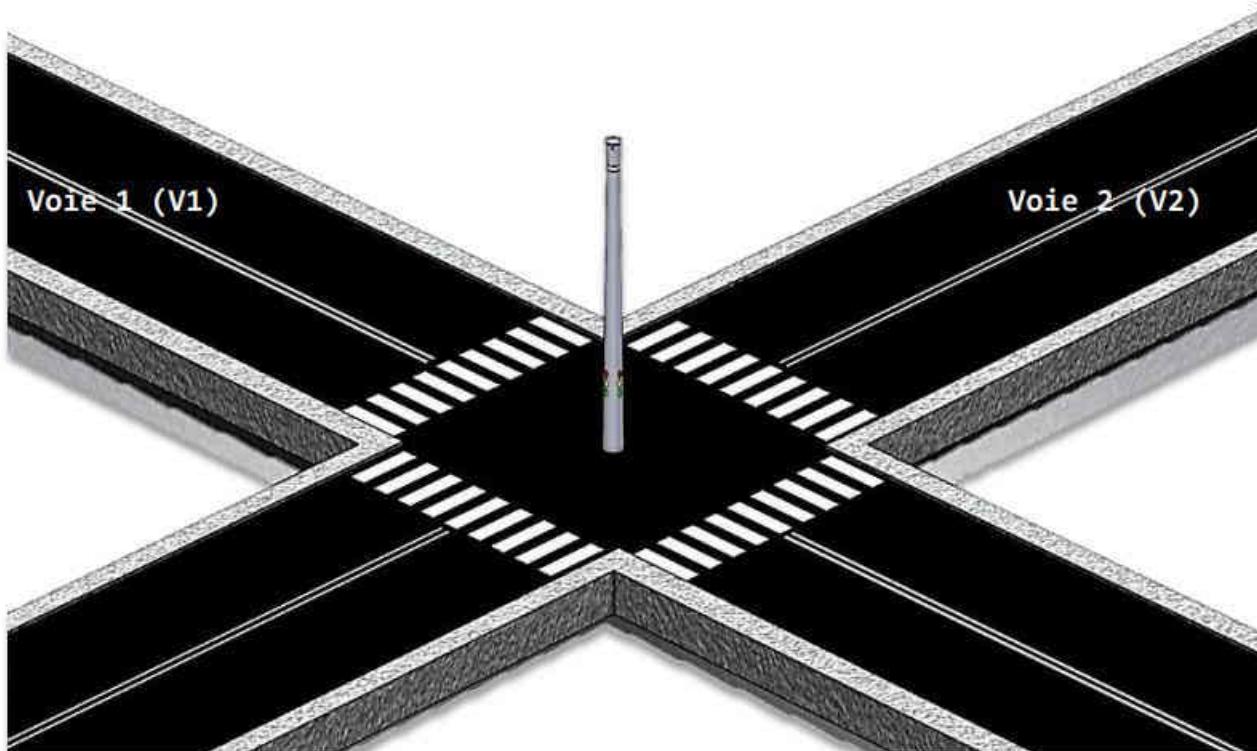


Figure 02: Principe du fonctionnement d'OFC.

I.1. Contextualisation



Figure 03: Les différents composants d'OFC.

I.2. Problématique

➤ **Comment peut-on:**

- ❖ **Orienter le boîtier?**
- ❖ **Acquérir puis traiter l'image de voitures dans chaque voie?**
- ❖ **Déterminer la voie prioritaire?**
- ❖ **Commander les feux de circulation?**

I.3. Cahier des charges fonctionnel

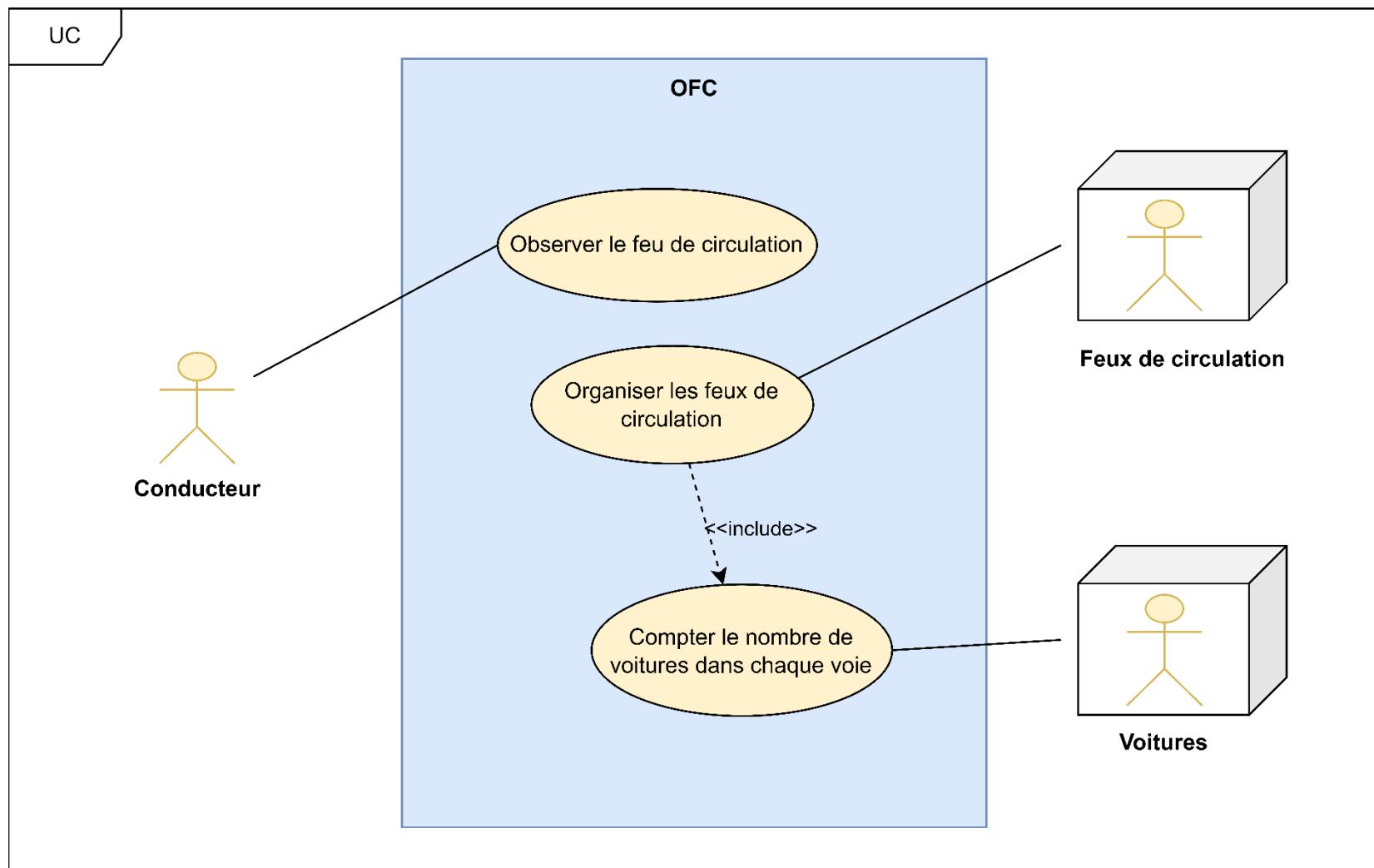


Figure 04: Diagramme de cas d'utilisation.

I.3. Cahier des charges fonctionnel

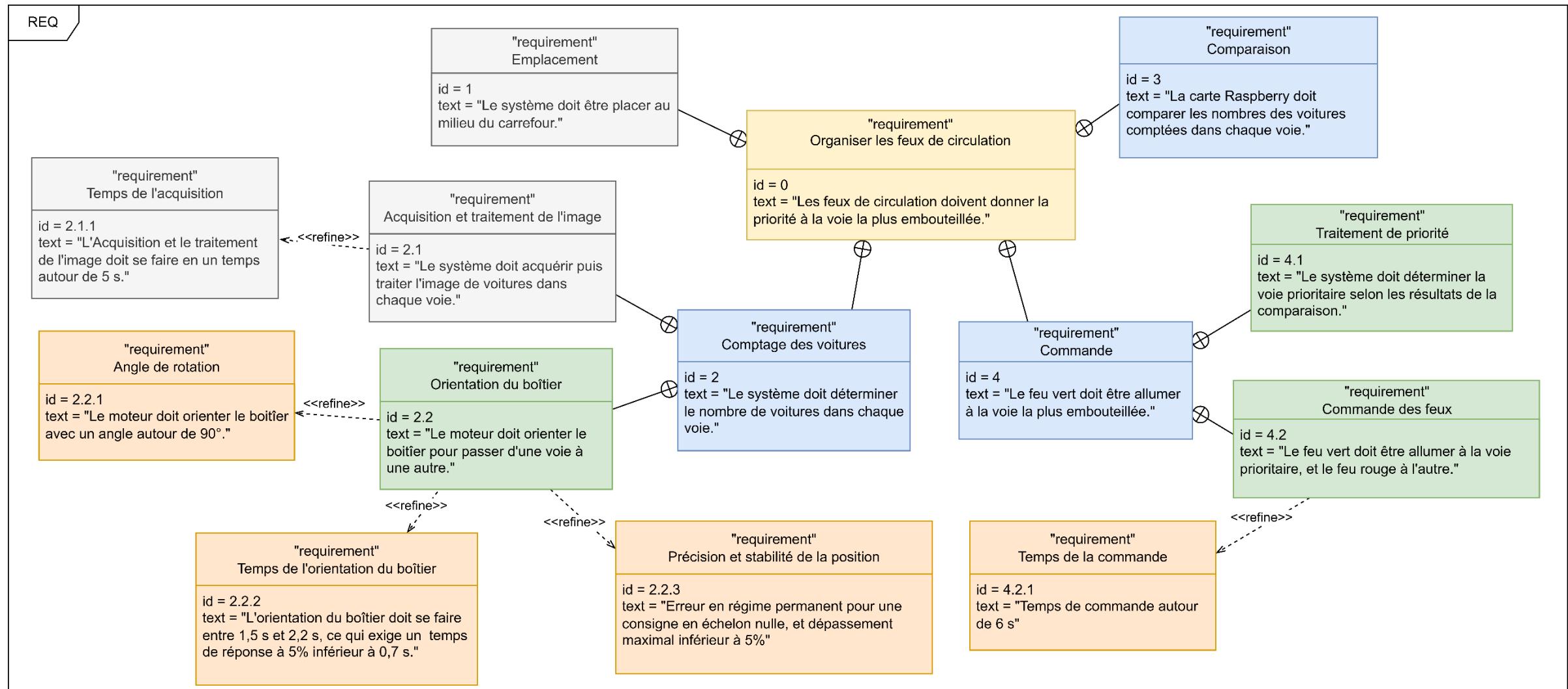


Figure 05: Diagramme d'exigences.

I.3. Cahier des charges fonctionnel

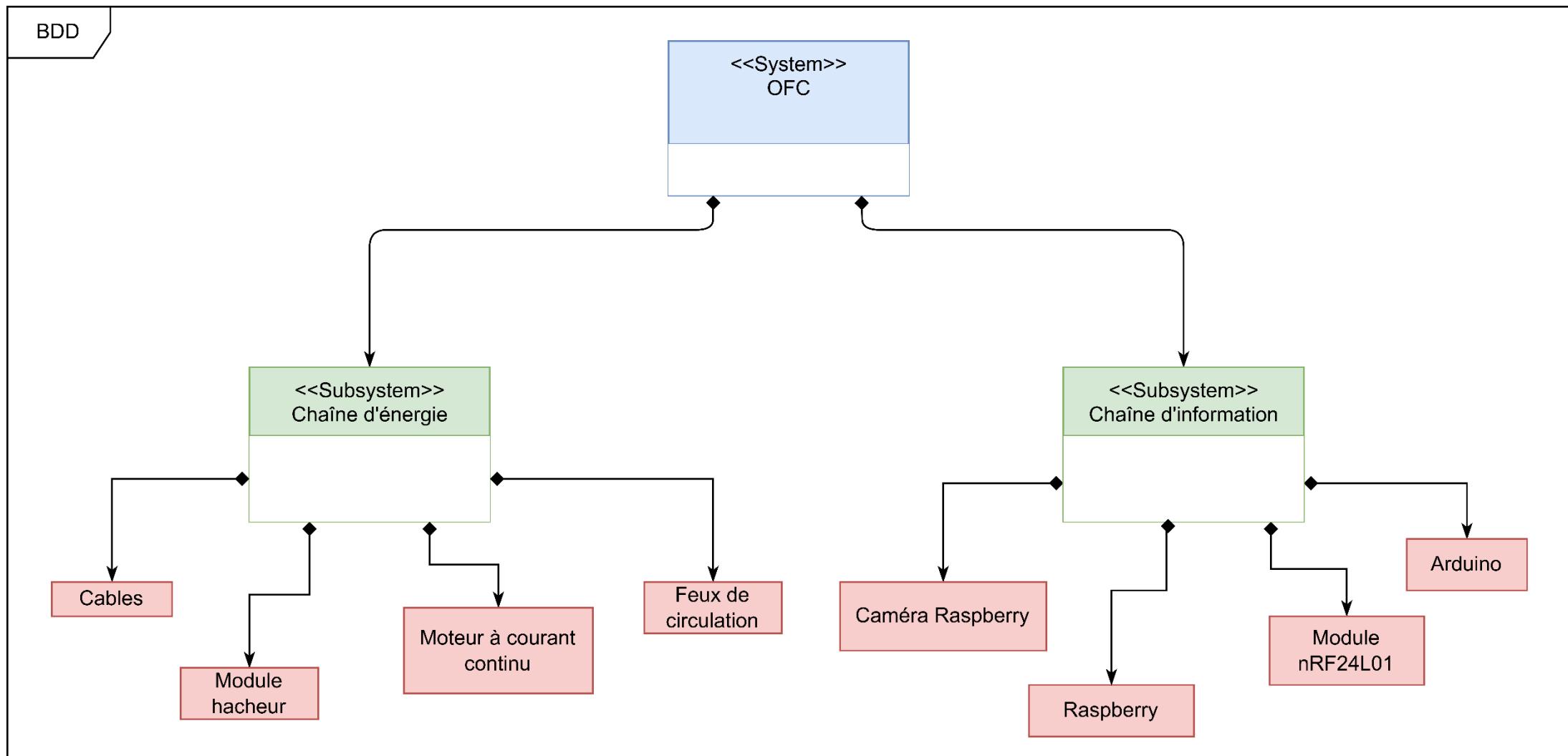
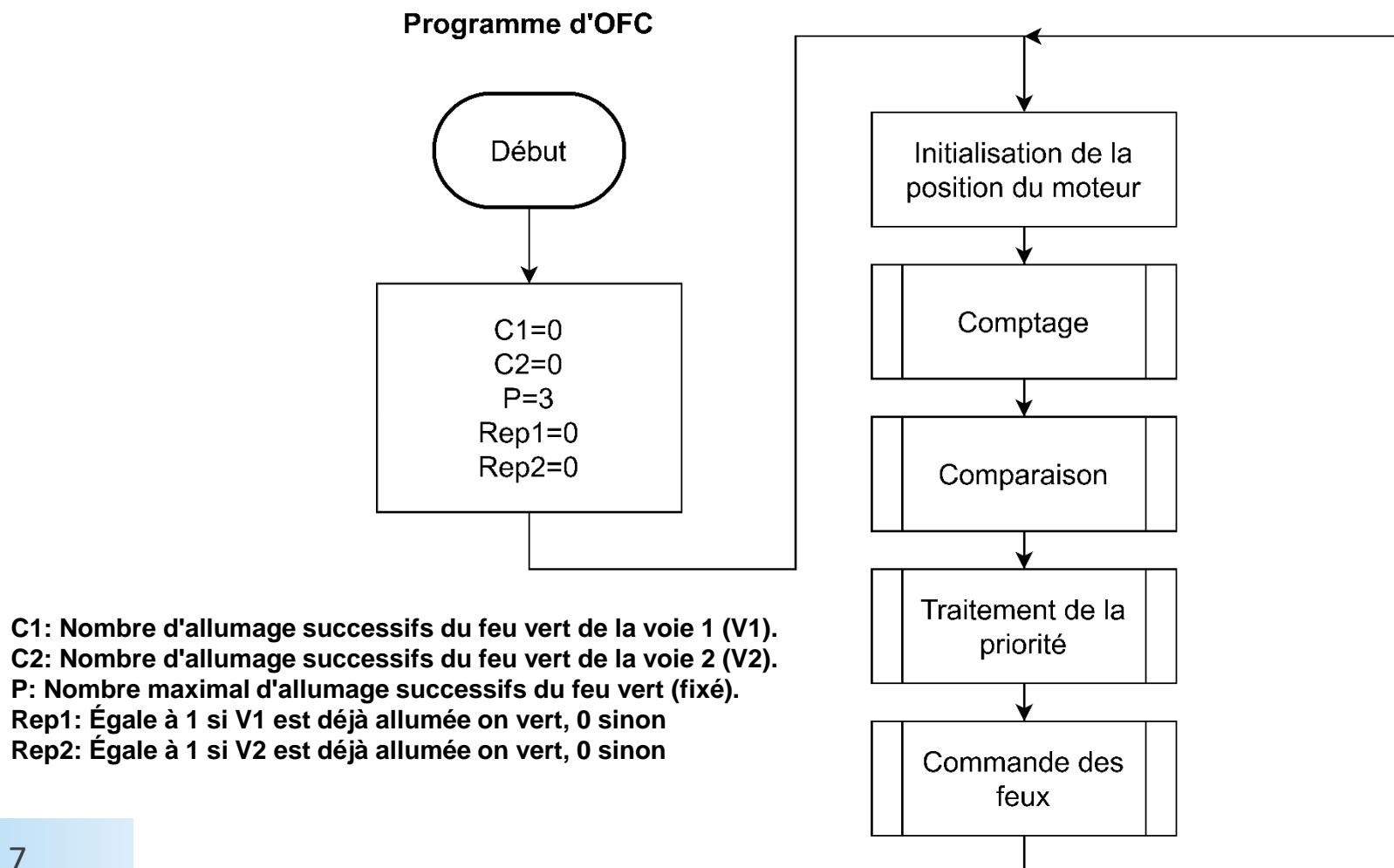


Figure 06: Diagramme de définition de blocs.

I.3. Cahier des charges fonctionnel



Voir l'annexe 7

Figure 07: Organigramme de fonctionnement d'OFC.

III. Objectifs

II.1. Choisir la motorisation convenable

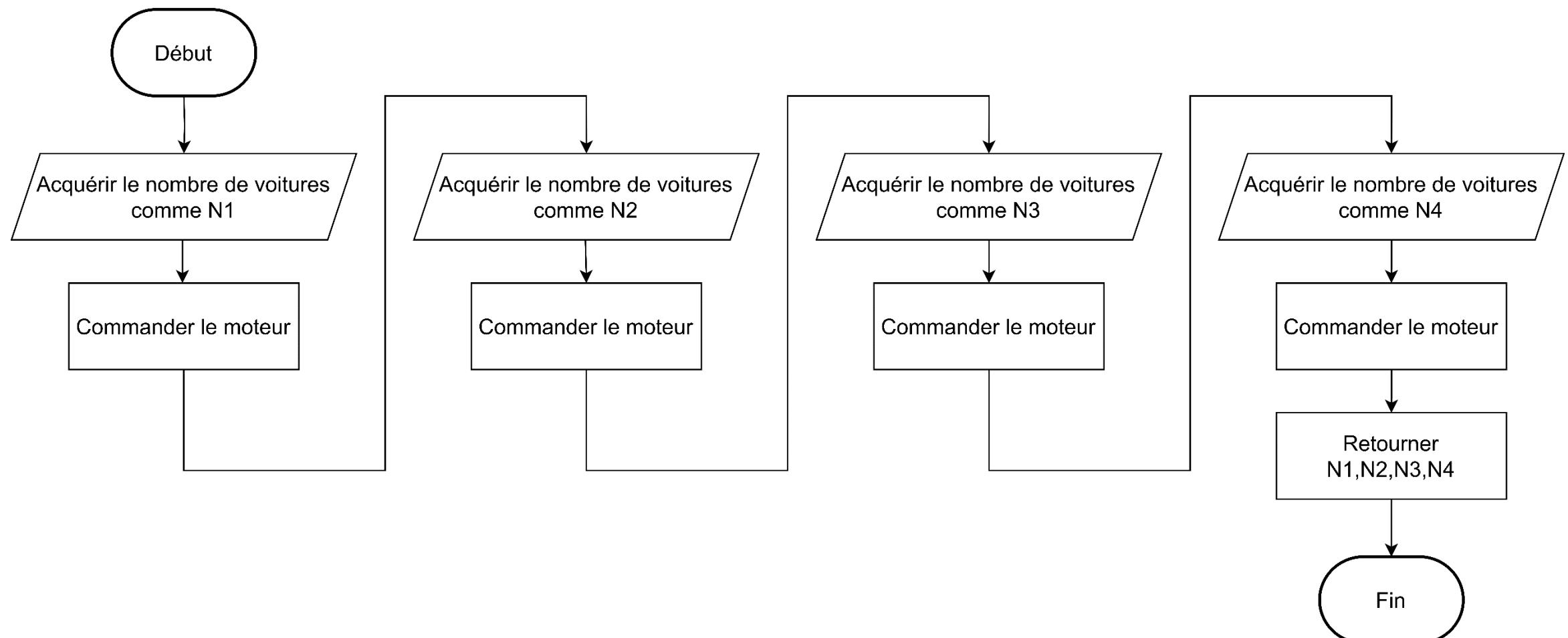
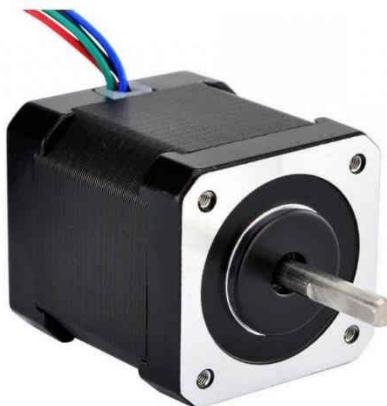


Figure 08: Sous-programme du comptage de voitures.

II.1. Choisir la motorisation convenable



Moteur pas à pas:

- **Mise en œuvre facile pour avoir un système répétable.**
- **Faible couple à haute vitesse.**
- **Consommation continue même à l'arrêt.**



Moteur à courant continu:

- **Mise en œuvre complexe (asservissement, réglage du PID...).**
- **Accélération plus importante et bon couple sur toute sa plage de vitesse.**
- **Consommation négligeable à l'arrêt.**

II.1. Choisir la motorisation convenable

➤ La loi de commande:

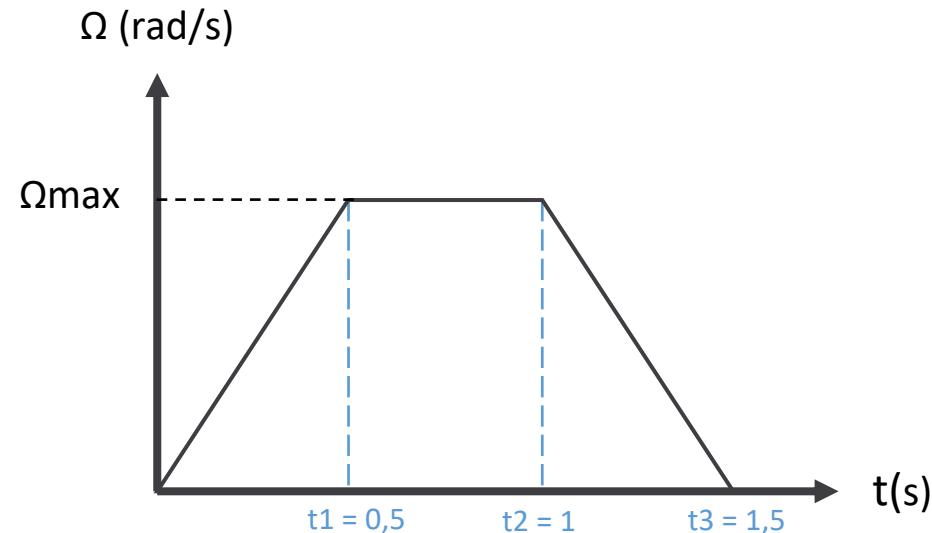


Figure 09: L'allure de la loi en trapèze.

➤ Calcul de Ω_{max} et de l'accélération angulaire:

$$\theta = t_2 \times \Omega_{max}$$

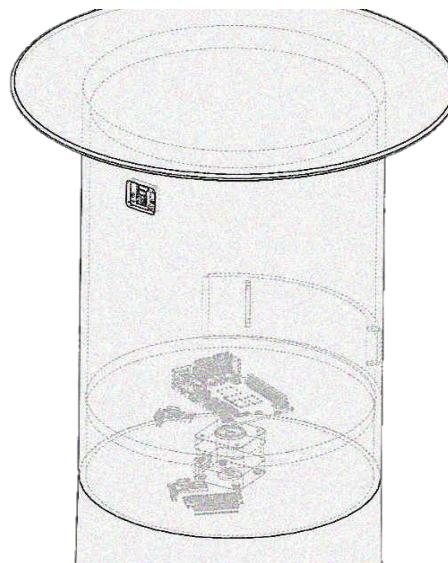
$$\theta = \frac{\pi}{2} \text{ rad et } t_2 = 1 \text{ s}$$

$$\begin{aligned} A.N. \\ \Rightarrow \Omega_{max} = 1,571 \text{ rad/s} \end{aligned}$$

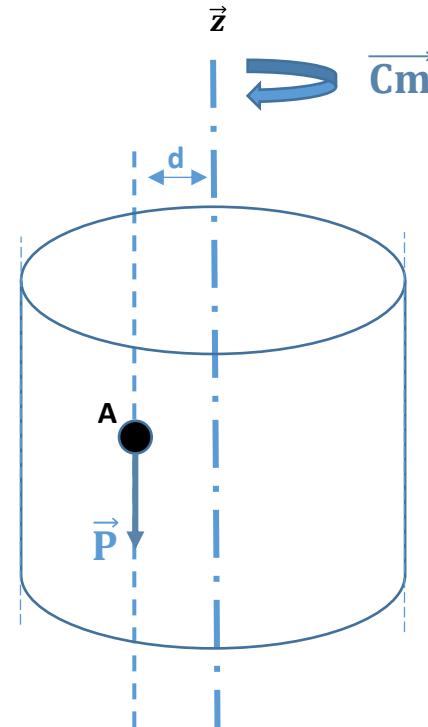
$$\frac{d\Omega(t)}{dt} = \frac{\Omega_{max}}{0,5} = 3,142 \text{ rad/s}^2$$

II.1. Choisir la motorisation convenable

➤ Calcul du couple résistant \mathbf{Cr} :



Assimiler les
composants du boîtier
à un point de masse M



Calcul du couple
résistant en
appliquant TMD

$$\mathbf{Cr} \cdot \vec{z} = 0$$

Figure 10: Assimilation des différents composants du boîtier à un point A.

II.1. Choisir la motorisation convenable

➤ Calcul du couple moteur C_m :

> Il faut d'abord calculer l'inertie équivalente ramenée sur l'axe du moteur:

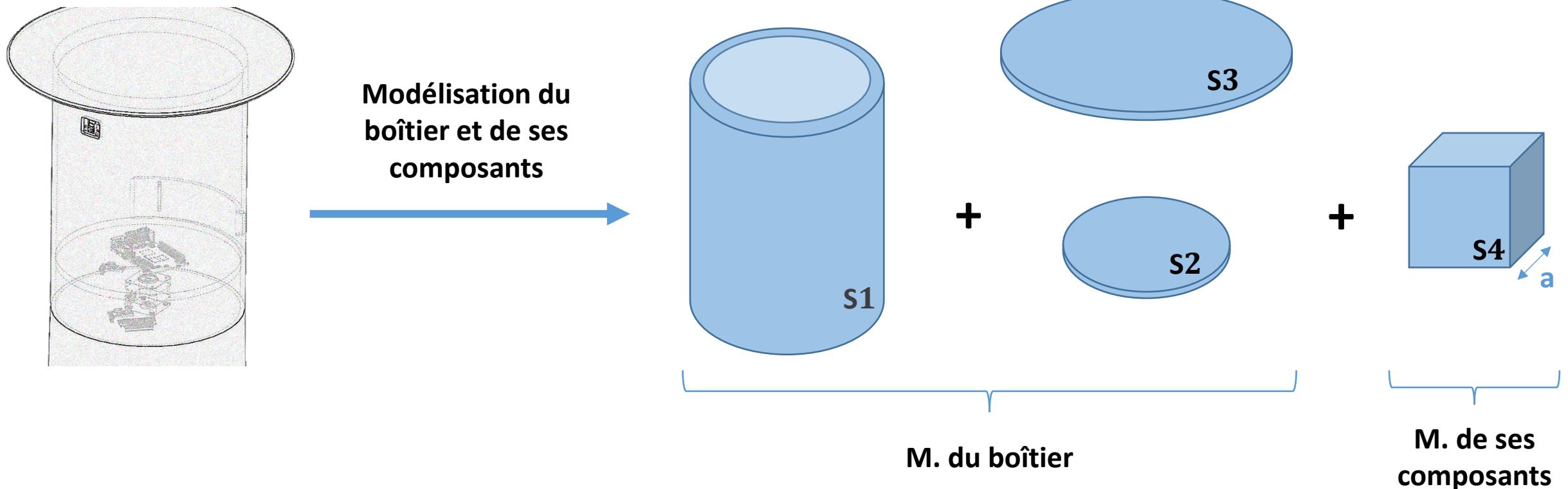


Figure 11: Modélisation du boîtier et de ses composants avec quatre solides usuels.

II.1. Choisir la motorisation convenable

- **Calcul du couple moteur Cm:**
- **À l'aide du Théorème de l'énergie cinétique, on trouve:**

$$Cm - Cfs = J \times \frac{d\Omega(t)}{dt}$$

$$J = [J1 + J2 + J3 + J4 + M \cdot d^2] + Jm$$

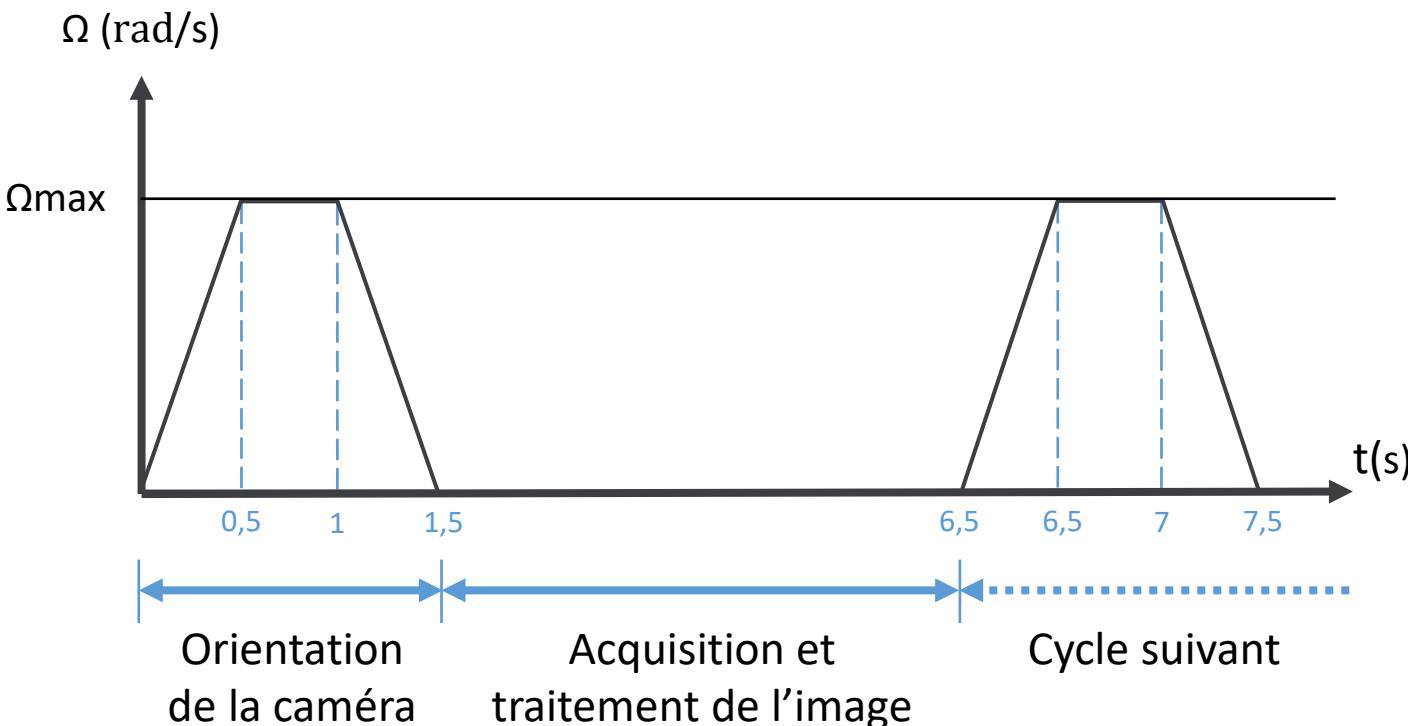
$$J = [m1 \frac{R1^2 - R2^2}{2} + m2 \frac{R2^2}{2} + m3 \frac{R3^2}{2} + M \frac{a^2}{6} + M \cdot d^2] + Jm$$

$$\Rightarrow J = 0,0059 + Jm \text{ (en Kg.m}^2\text{)}$$

$$\Rightarrow Cm = 0,018 + Cfs + 3,142 \cdot Jm \text{ (en N.m)}$$

II.1. Choisir la motorisation convenable

➤ Calcul du couple thermique équivalent:



$$C_{th} = \sqrt{\frac{\sum_i C_i^2 \cdot t_i}{\sum_i t_i}} = \sqrt{\frac{C_m^2 \cdot 0,5 + C_m^2 \cdot 0,5}{6,5}}$$
$$\Rightarrow C_{th} = 0,392 \cdot C_m$$
$$\Rightarrow C_{th} = 0,0073 + 0,392 \cdot C_{fs}$$
$$+ 1,232 \cdot J_m \text{ (en N.m)}$$

Figure 12: La loi de commande de la vitesse.

II.1. Choisir la motorisation convenable

➤ On choisit le moteur convenable tel que:

$$C_{m\max} > C_{th} \text{ et } \Omega_{m\max} > \Omega_{\max}$$

> Le moteur MCC JGB-520, a comme caractéristiques:

$$\Omega_{m\max} = 26,70 \text{ rad/s}$$

$$C_{m\max} = 0,0363 \text{ N.m}$$

$$J_m = 0,012 \text{ Kg.m}^2$$

$$C_{fs} = 0,0171 \text{ N.m}$$

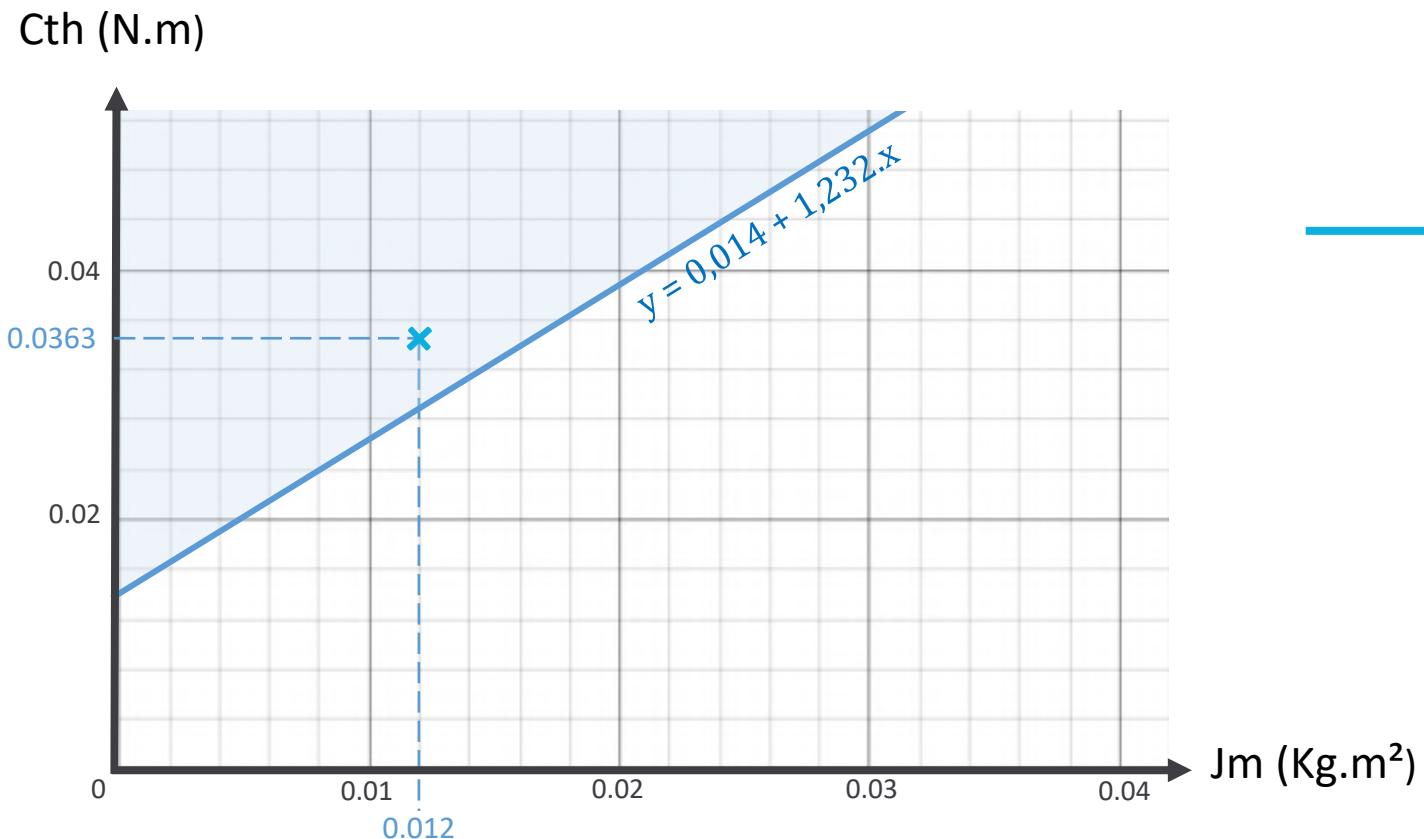
		Model: NFP-JGB37-520-2460											
Voltage	Range	V	12.0-30.0										
	Rated	V	DC 24V										
Reducer	Ratio	i:1	6.25	10	18.8	30	56	90	131	168	270	506	810
	Size "L"	mm	19	19	22	22	24	24	26.5	26.5	26.5	29	29
No Load	Speed	rpm	960	600	319	200	107	66	45	35	22	12	7
	Current	mA	35	35	35	35	35	35	35	30	30	30	30
Rated Load	Speed	rpm	672	480	255	160	85	52	36	28	18	10	5.6
	Torque	Kgf.cm	0.15	0.2	0.37	0.6	1	1.8	2.5	3.3	5	10	16
	Current	mA	150	150	150	120	120	100	80	60	50	40	40
	Output	W	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
Stall	Torque	Kgf.cm	0.6	0.8	1.5	2.4	4	7	10	13	20	40	64
	Current	mA	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
Weight		g	144	144	150	150	158	158	164	164	164	170	170

Figure 13: Datasheet du moteur MCC JGB37-520.

II.1. Choisir la motorisation convenable

➤ On choisit le moteur convenable tel que:

$$C_{m\max} > C_{th} \text{ et } \Omega_{m\max} > \Omega_{\max}$$



$$C_{m\max} > C_{th}$$

$$\begin{aligned}\Omega_{\max} &= 1,571 \text{ rad/s} \\ \Rightarrow \Omega_{m\max} &> \Omega_{\max}\end{aligned}$$

Moteur convenable

Figure 14: Le tracé de C_{th} en fonction de J_m .

II.2. Identifier les paramètres du moteur

➤ Le moteur et le hacheur de notre système:

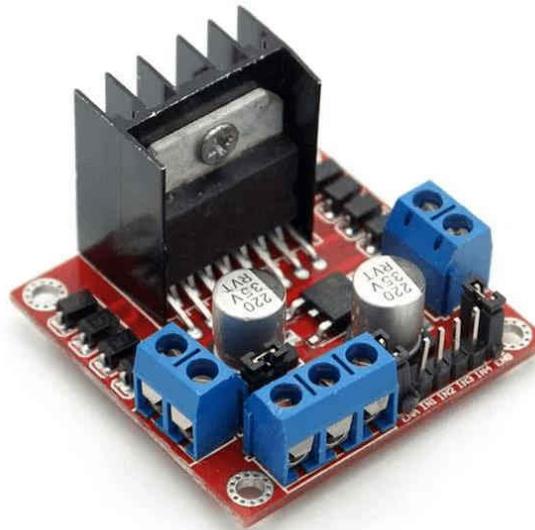


Figure 15: Module hacheur type L298N.



Figure 16: Moteur à courant continu JGB37-520 + codeur incrémental.



II.2. Identifier les paramètres du moteur

- **Expérience : Réponse en vitesse du motoréducteur à un échelon de tension (24v).**

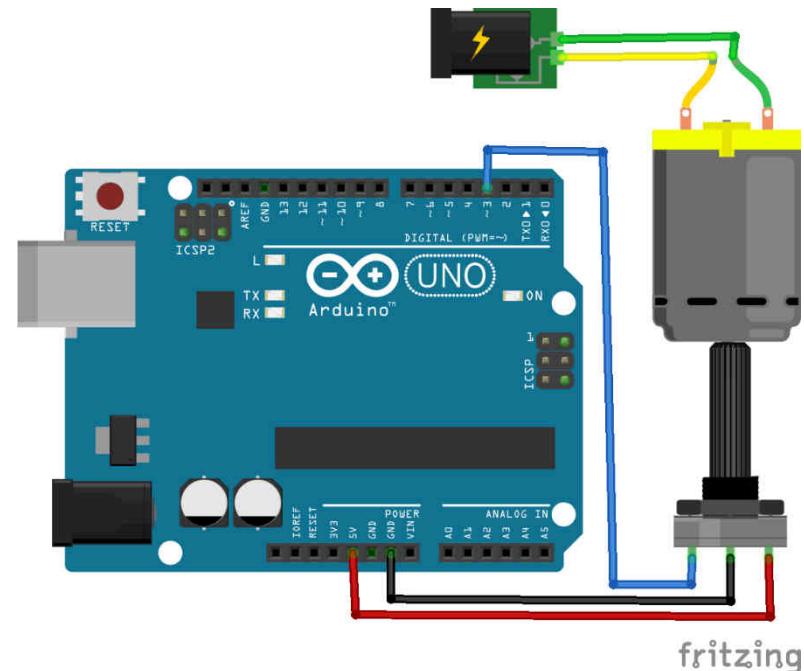
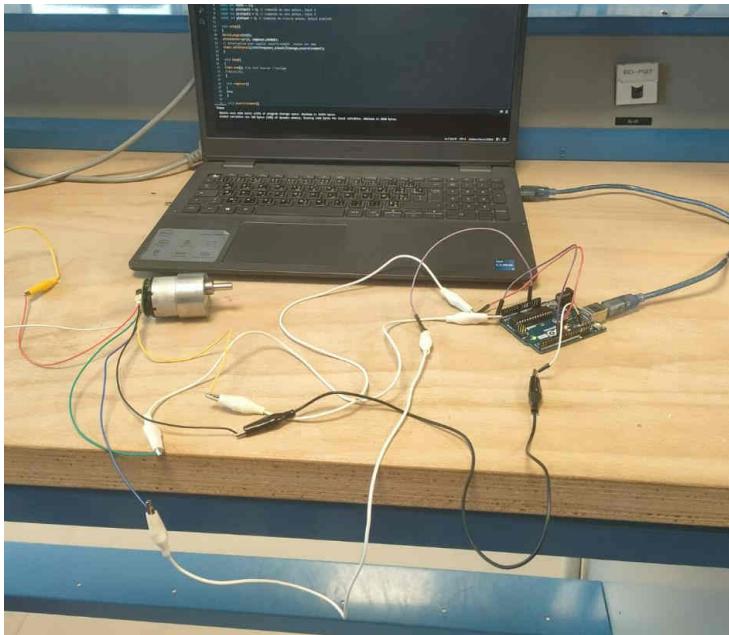
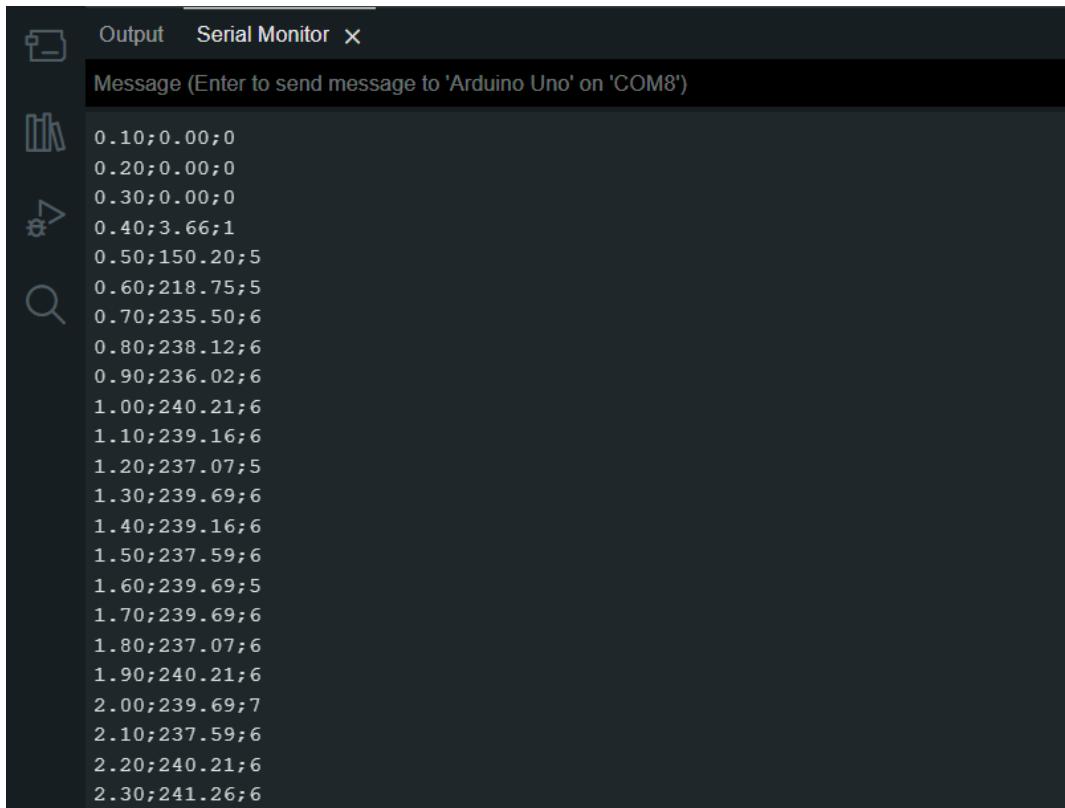


Figure 17: Réalisation de l'expérience.

Voir l'annexe 1

II.2. Identifier les paramètres du moteur



The screenshot shows the Arduino Serial Monitor window. The title bar says "Output" and "Serial Monitor". The main area displays a series of measurements in a table format:

Time	Voltage	Current
0.10	0.00	0
0.20	0.00	0
0.30	0.00	0
0.40	3.66	1
0.50	150.20	5
0.60	218.75	5
0.70	235.50	6
0.80	238.12	6
0.90	236.02	6
1.00	240.21	6
1.10	239.16	6
1.20	237.07	5
1.30	239.69	6
1.40	239.16	6
1.50	237.59	6
1.60	239.69	5
1.70	239.69	6
1.80	237.07	6
1.90	240.21	6
2.00	239.69	7
2.10	237.59	6
2.20	240.21	6
2.30	241.26	6

Figure 18: Extrait du tableau des mesures de l'expérience.

Résultats de l'expérience:

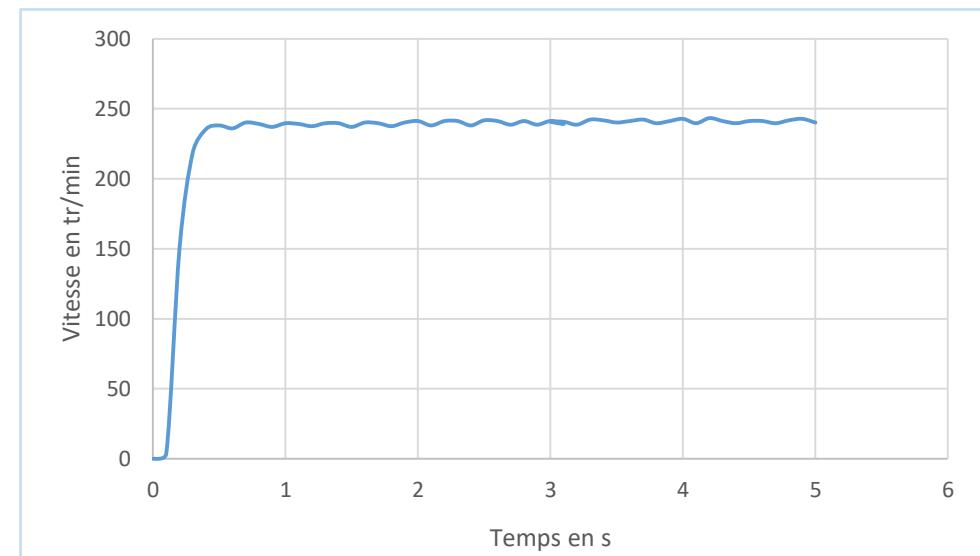


Figure 19: L'allure réelle de la vitesse de rotation en fonction du temps.

II.2. Identifier les paramètres du moteur

- On peut le simuler à un système de premier ordre:

$$K = \frac{N(\infty)}{U(\infty)} \approx \frac{240}{24} = 10 \text{ tr/min/V}$$

$$N(\tau) = 0,63 \cdot N(\infty) \Rightarrow \tau \approx 0,101 \text{ s}$$

$$H(p) = \frac{N(p)}{U(p)} = \frac{K}{1 + \tau \cdot p} = \frac{10}{1 + 0,101 \cdot p}$$

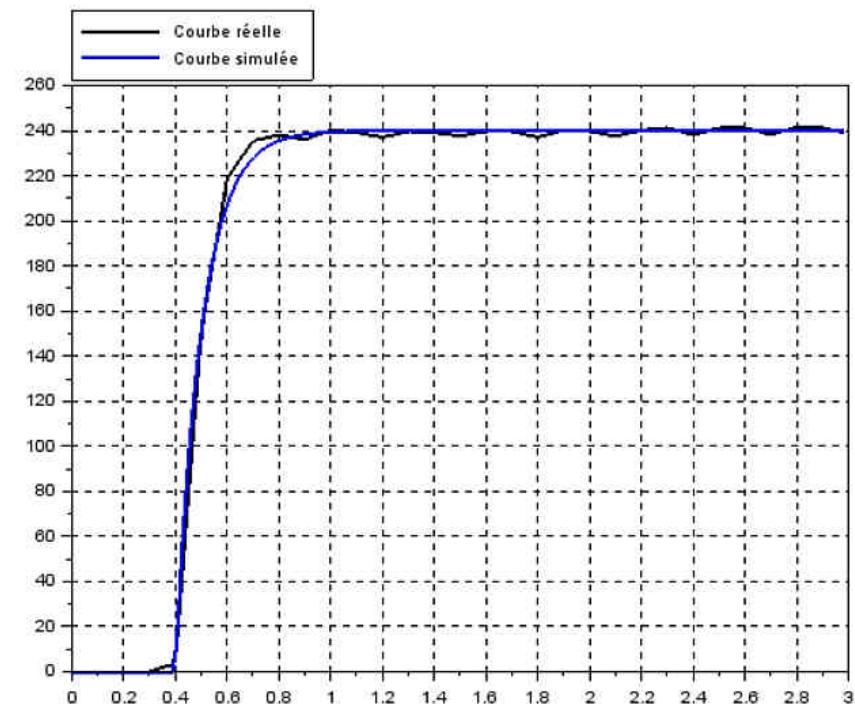
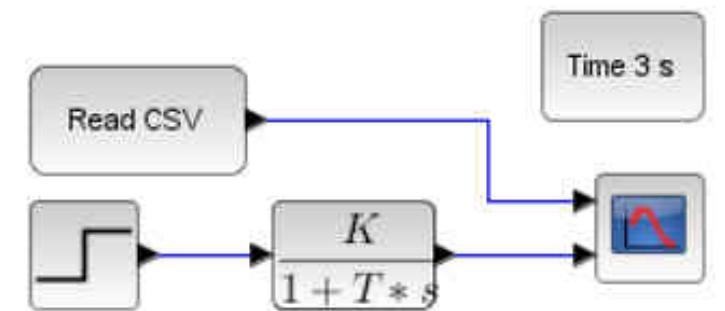
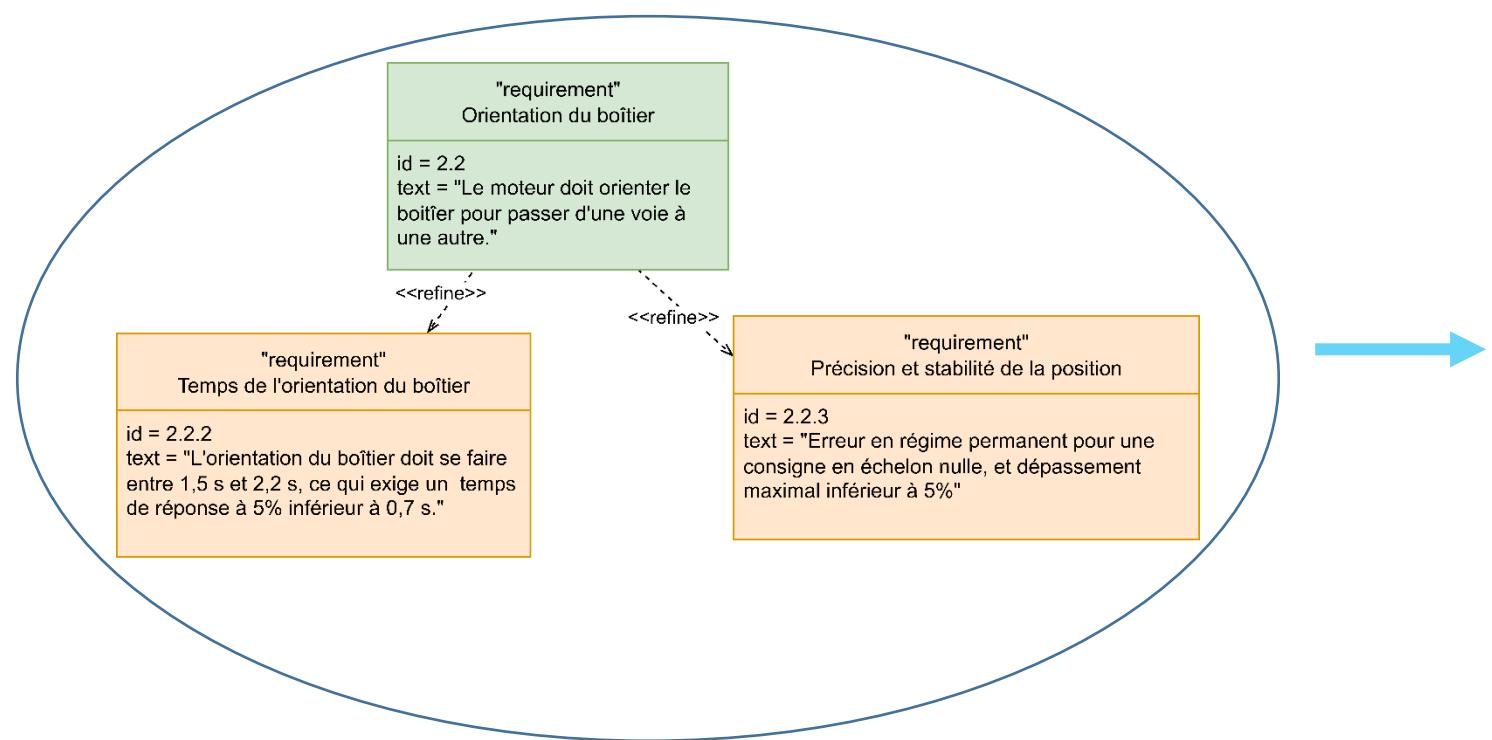


Figure 20: Comparaison entre le système réel et la simulation.

II.3. Asservir en position le moteur



Temps de réponse à 5% < 0,7s
Dépassemement relatif < 5%
Erreur statique nulle

Figure 21: Extrait du cahier des charges associé à l'exigence «Orientation du boîtier».

II.3. Asservir en position le moteur

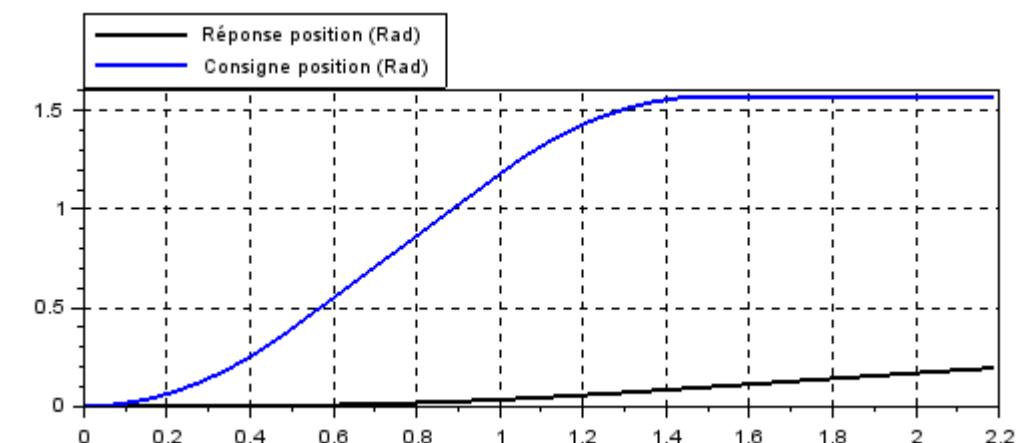
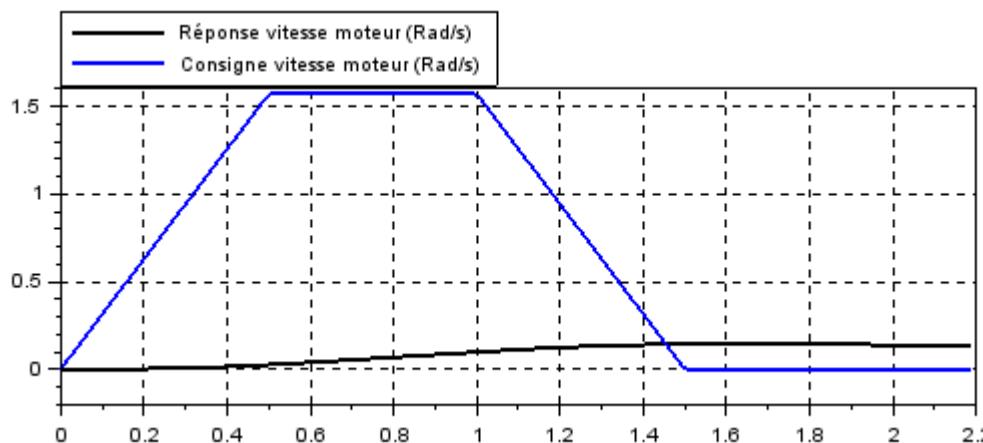
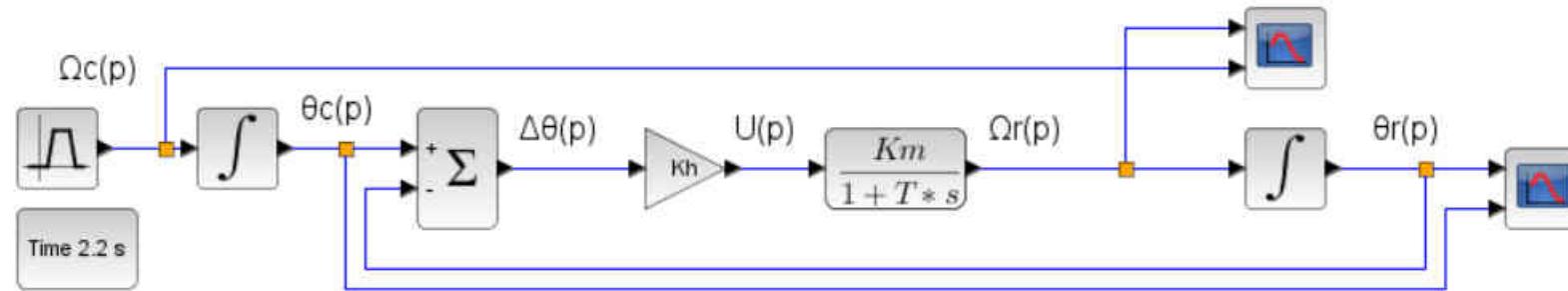
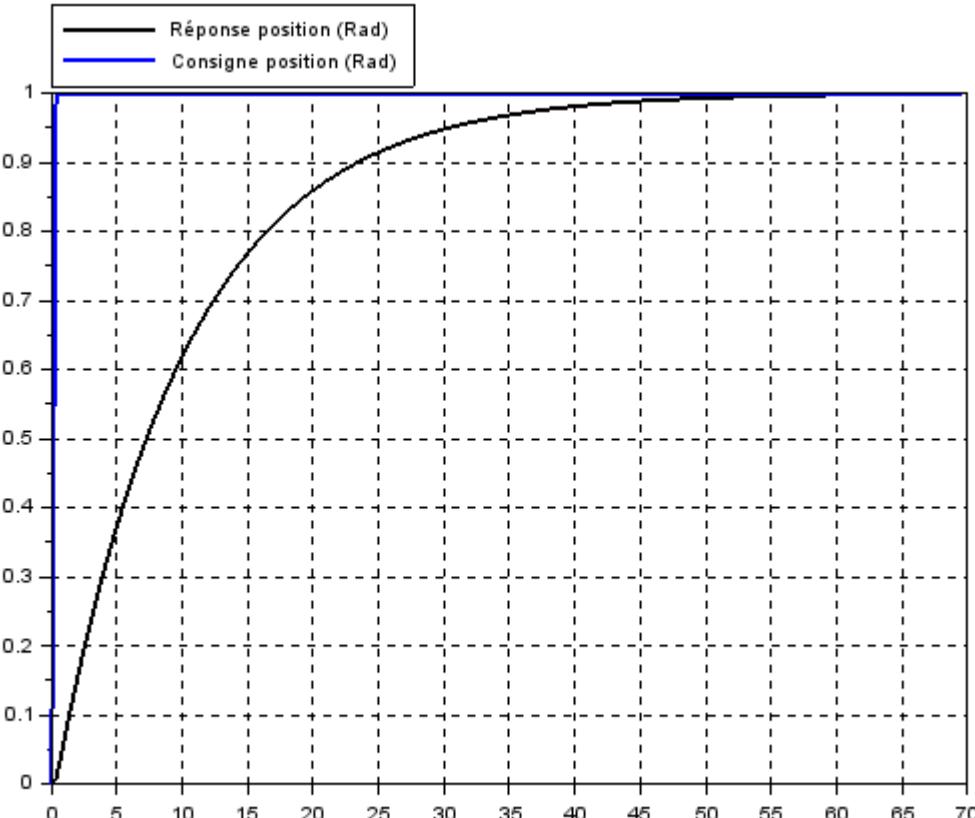


Figure 22: Schémas de la modalisation de l'asservissement du moteur sans correction.

II.3. Asservir en position le moteur

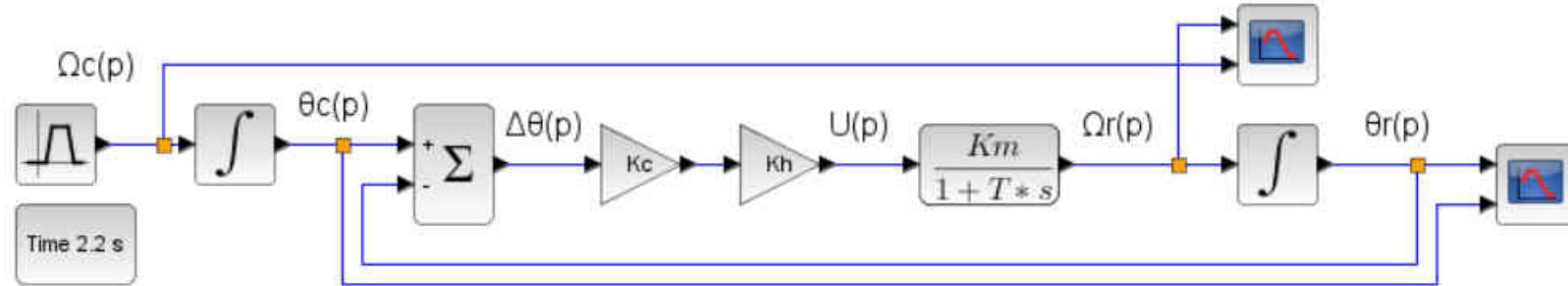


Erreur statique nulle
Dépassemement relatif nul
Temps de réponse à 5% = 30s > 0,7s

Figure 23: Réponse à un échelon sans correction.

II.3. Asservir en position le moteur

➤ Correcteur proportionnel:



$$\begin{aligned} H_m(p) &= \frac{\Theta_r(p)}{\Theta_c(p)} = \frac{1}{1 + \frac{1}{K_c \cdot K_m \cdot K_h} \cdot p + \frac{\tau}{K_c \cdot K_m \cdot K_h} \cdot p^2} \\ &= \frac{1}{1 + \frac{2 \cdot z}{\omega_0} \cdot p + \frac{p^2}{\omega_0^2}} \end{aligned}$$

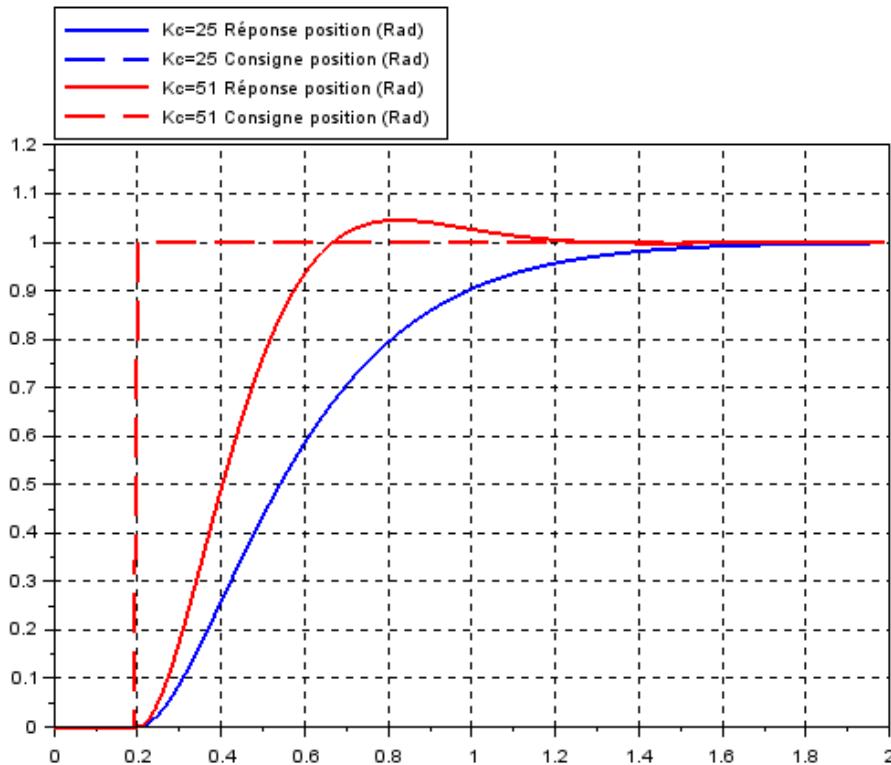
➤ Calcul théorique de K_c :

$$z = 1 \Rightarrow K_c \approx 25$$

$$z = 0,7 \Rightarrow K_c \approx 51$$

II.3. Asservir en position le moteur

➤ Correcteur proportionnel:



➤ $K_c = 25$ (sans dépassement):

Erreur statique nulle

Dépassement relatif nul

Temps de réponse à 5% = 1,160s > 0,7s

➤ $K_c = 51$ (t5% minimum):

Erreur statique nulle

Dépassement relatif = 4,523% < 5%

Temps de réponse à 5% = 0,615s < 0,7s

Figure 24: Réponse à un échelon avec correction.

II.3. Asservir en position le moteur

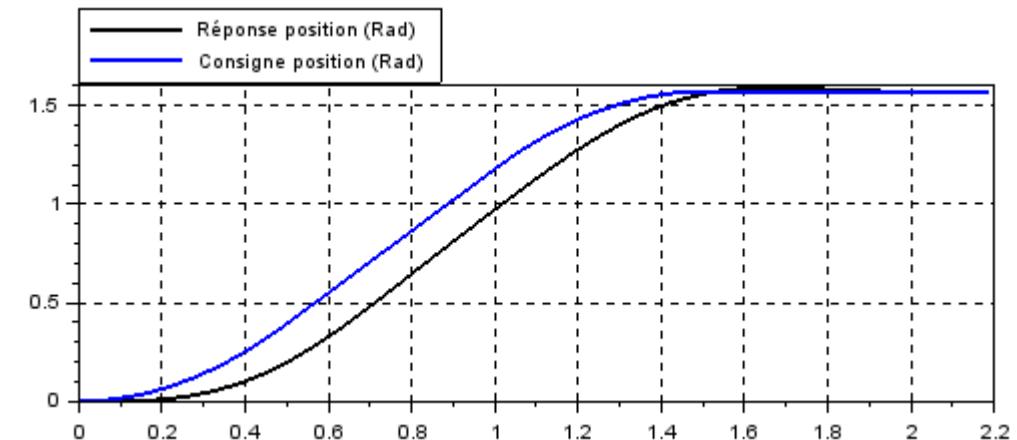
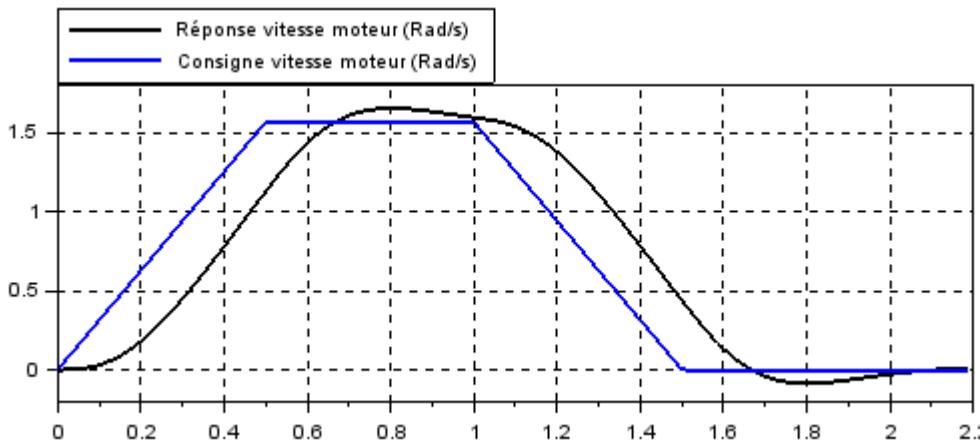
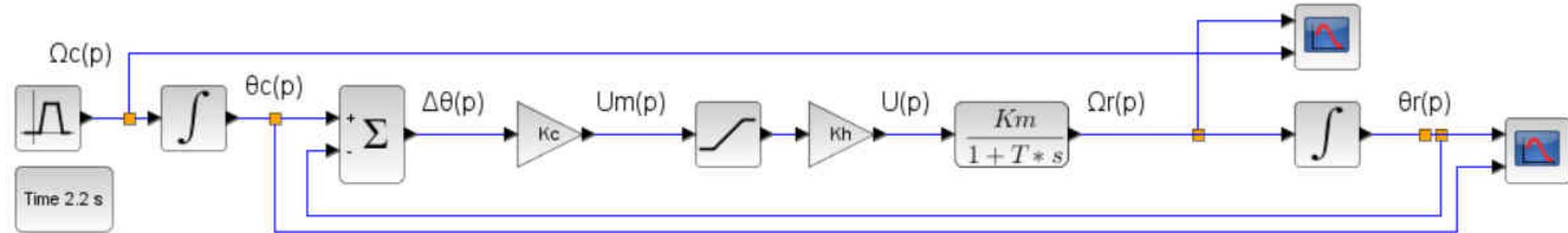


Figure 25: Schéma de la modalisation de l'asservissement du moteur avec correction.

II.3. Asservir en position le moteur

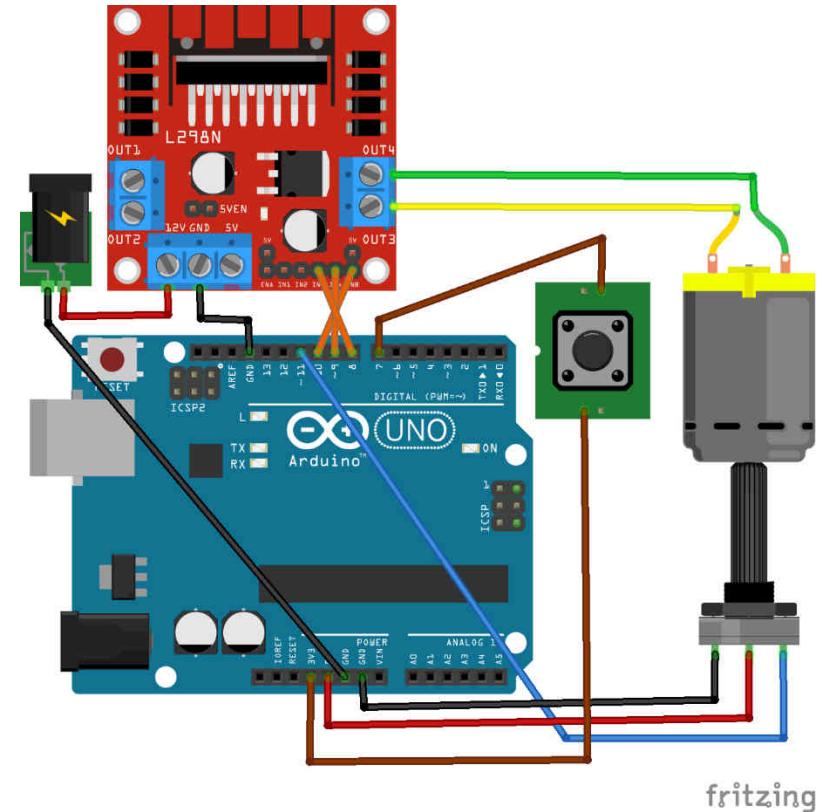
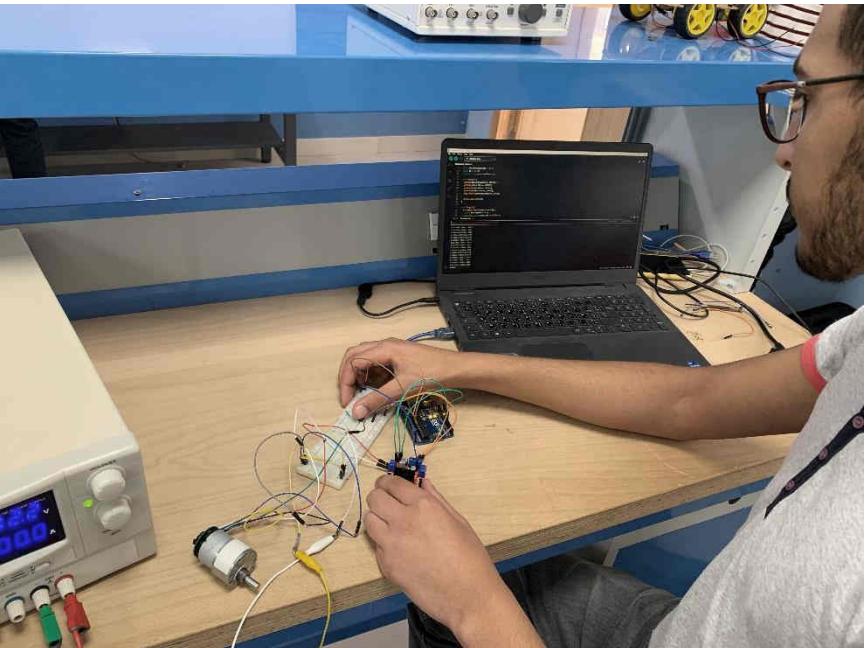
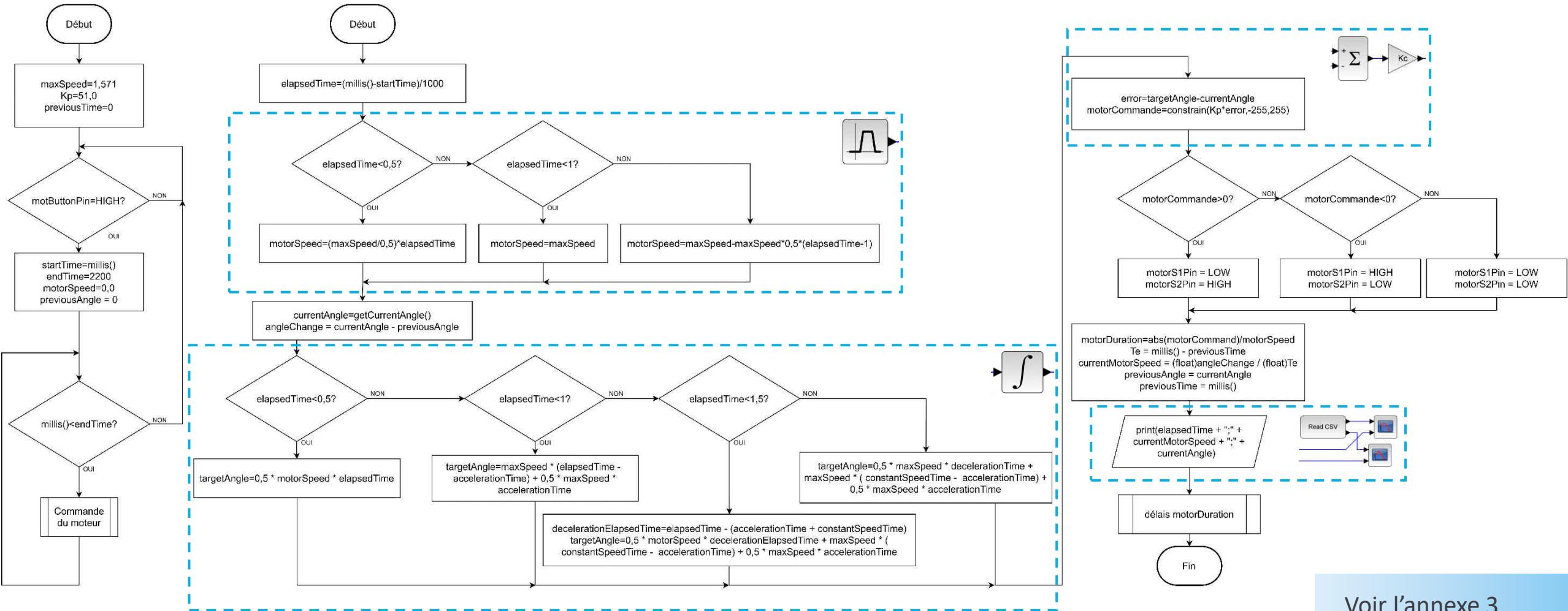


Figure 26: Expérience d'asservissement en position du moteur.

Voir l'annexe 2

II.3. Asservir en position le moteur



Voir l'annexe 3

Figure 27: Organigramme d'asservissement du moteur en position.

II.3. Asservir en position le moteur

The screenshot shows the Arduino Serial Monitor window. The title bar says "Output" and "Serial Monitor X". Below it, a message box says "Message (Enter to send message to 'Arduino Uno' on 'COM9')". The main area displays a list of 30 measurement entries, each consisting of three values separated by semicolons:

- 0;0;0
- 0.002;0.0073;0.0047
- 0.003;0.0073;0.0047
- 0.005;0.0014;0.0036
- 0.025;0.0073;0.0047
- 0.047;0.0073;0.0047
- 0.07;0.0367;0.0047
- 0.093;0.0544;0.0047
- 0.116;0.0838;0.0142
- 0.138;0.1485;0.0047
- 0.162;0.1897;0.0047
- 0.184;0.2661;0.0142
- 0.207;0.3191;0.0284
- 0.23;0.3661;0.0236
- 0.253;0.4367;0.0284
- 0.275;0.5073;0.052
- 0.299;0.5544;0.052
- 0.321;0.6073;0.0667
- 0.344;0.7014;0.0994
- 0.368;0.7897;0.1136
- 0.39;0.8602;0.1372

Figure 28: Extrait du tableau des mesures de l'expérience.

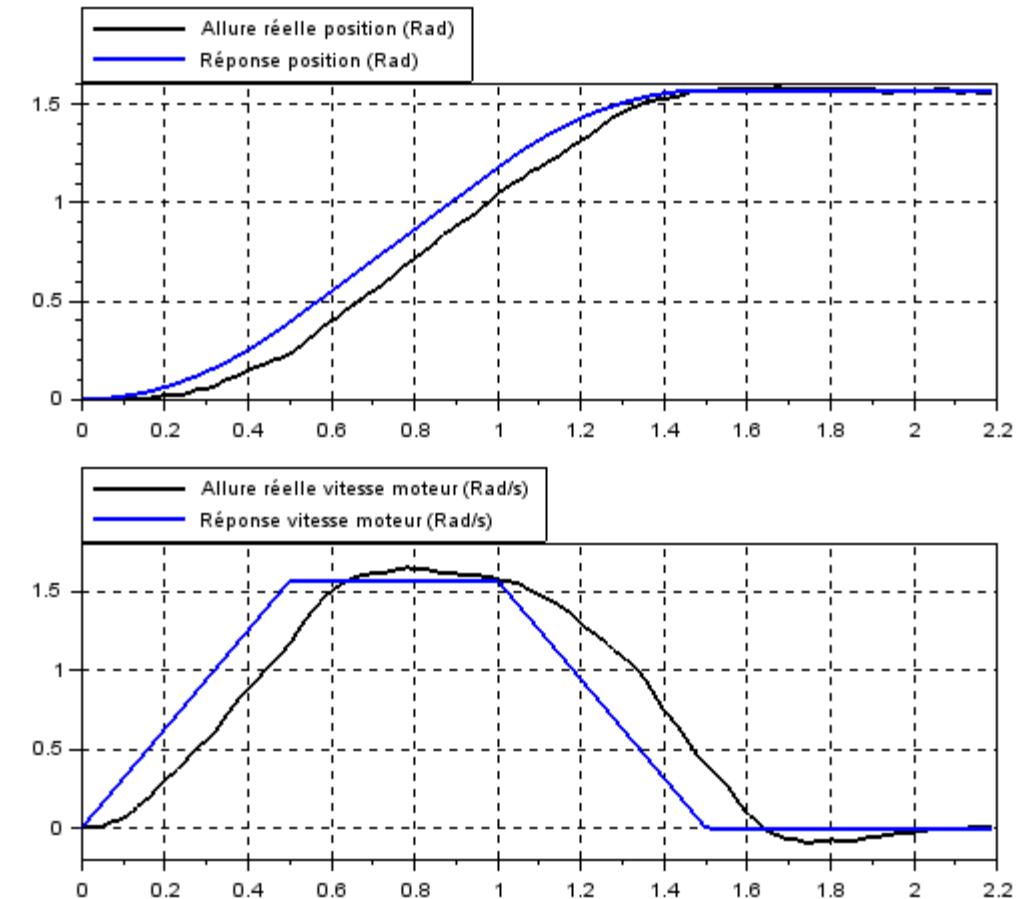


Figure 29: Comparaison entre le système réel et la simulation.

II.4. Établir un ordre de priorité

➤ **Après l'extraction de nombre de voitures dans chaque voie, on doit les comparés. (dans ce cas $N1+N3$ est le nombre de voitures dans la voie 1)**

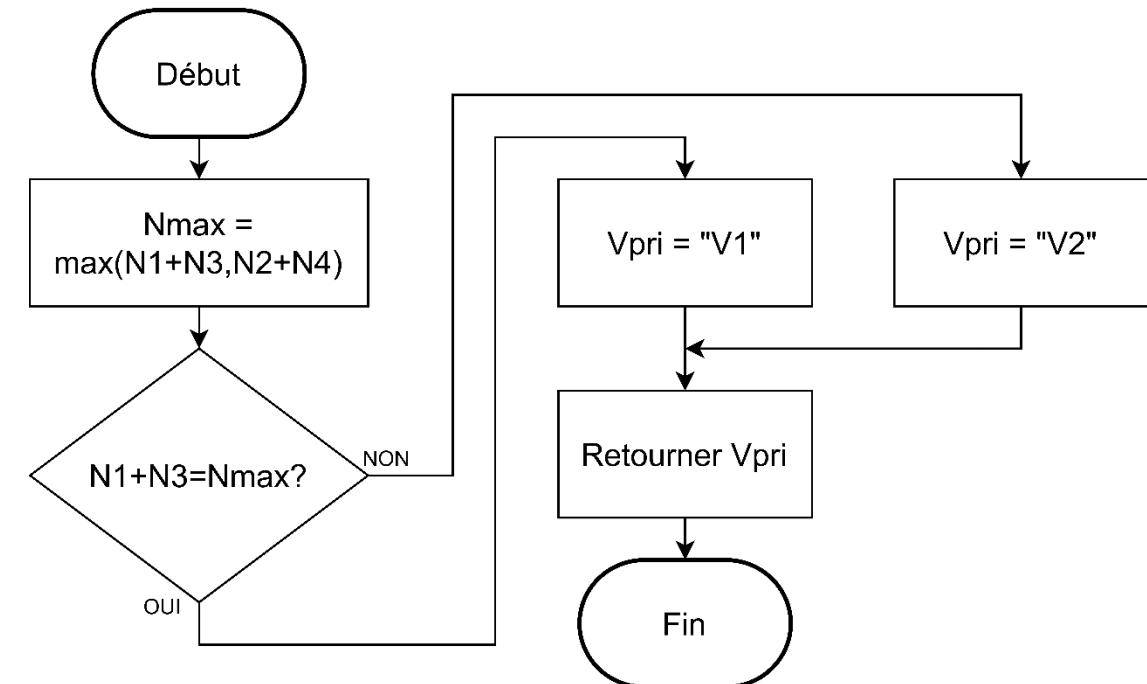
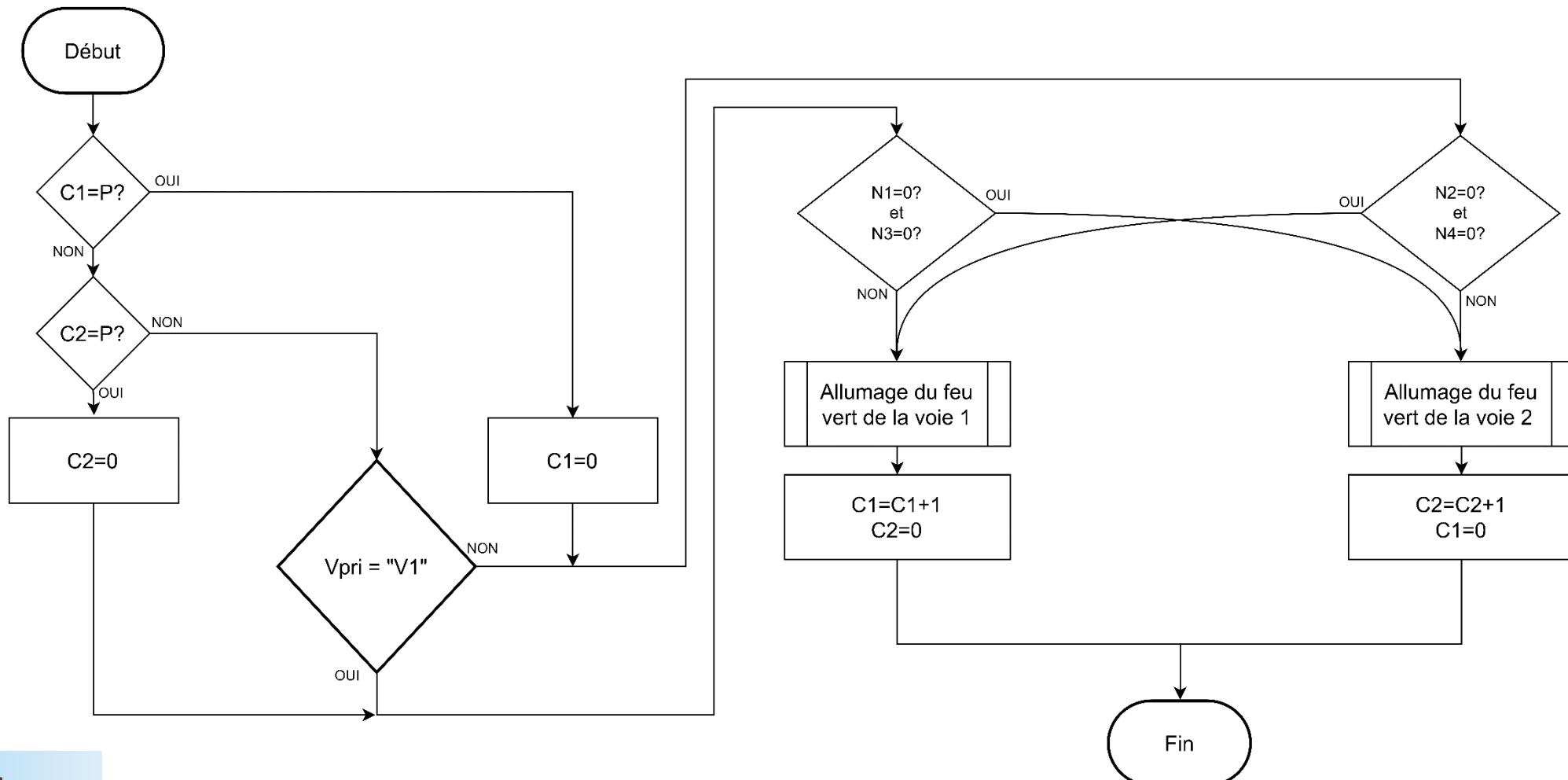


Figure 30: Sous-programme de la comparaison.

Voir l'annexe 4

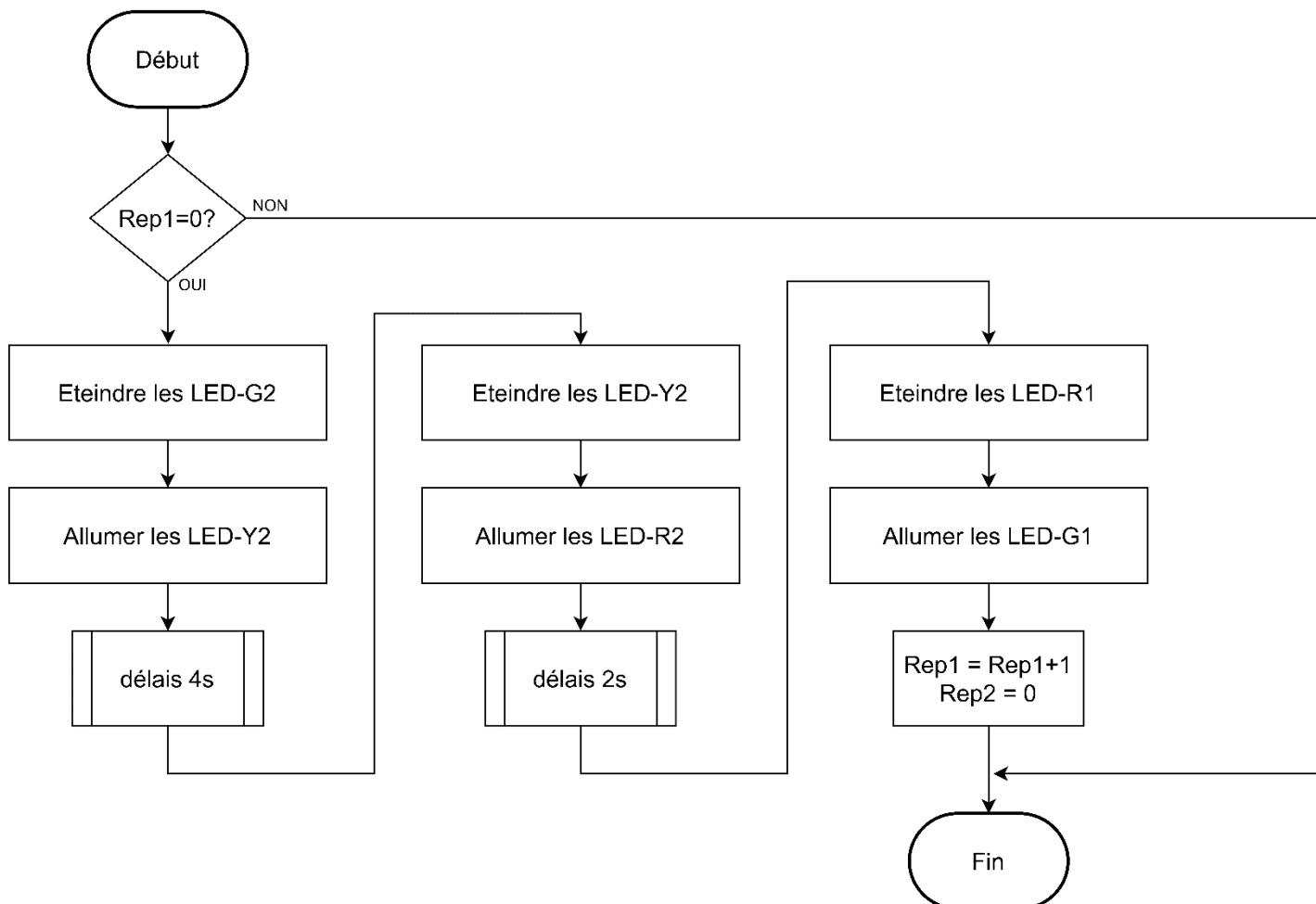
II.4. Établir un ordre de priorité



Voir l'annexe 4

Figure 31: Sous-programme du traitement de la priorité.

II.5. Commander les feux de circulation



Voir l'annexe 5

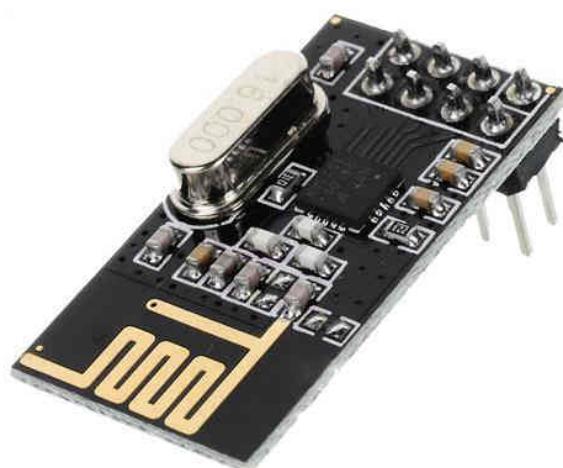
Figure 32: Sous-programme d'allumage du feu vert de V1.

II.5. Commander les feux de circulation



Communication via câbles:

- Le moteur doit tourner dans les deux sens.
- Risque de rupture.



Communication à l'aide du module nRF24L01:

- Communication sans fil: pas de contraintes sur le moteur.
- Plus coûteux.

II.5. Commander les feux de circulation

- **Expérience : Tester la communication entre la carte Arduino Nano et la carte Raspberry Pi 3.**

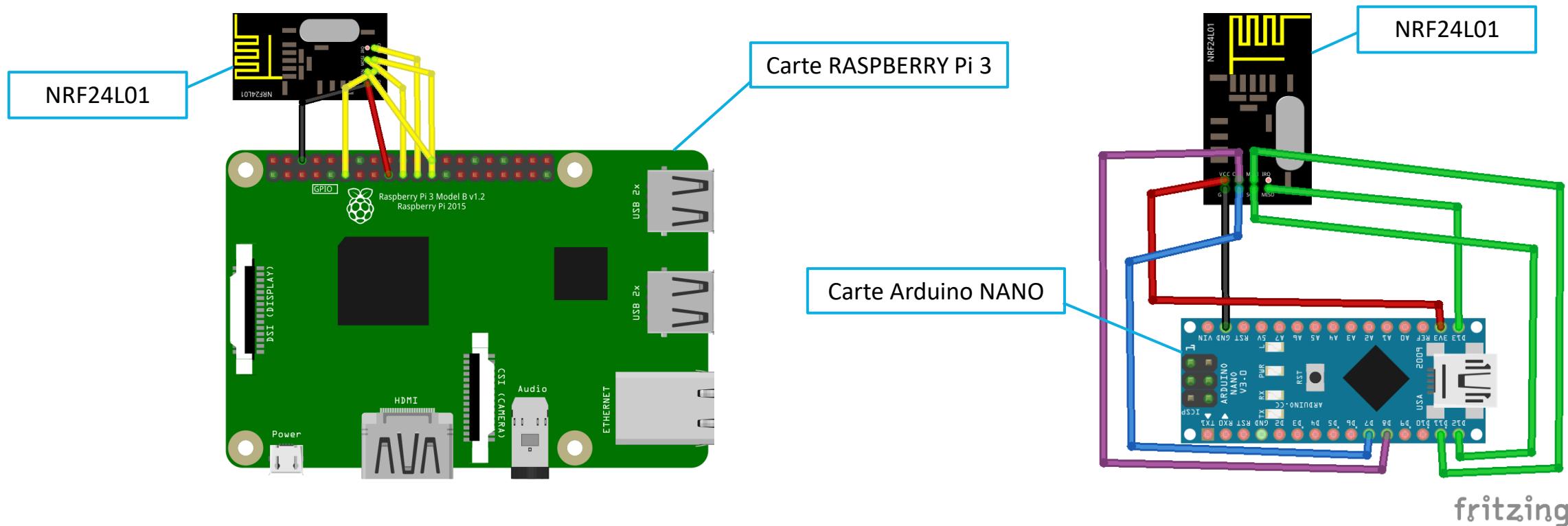
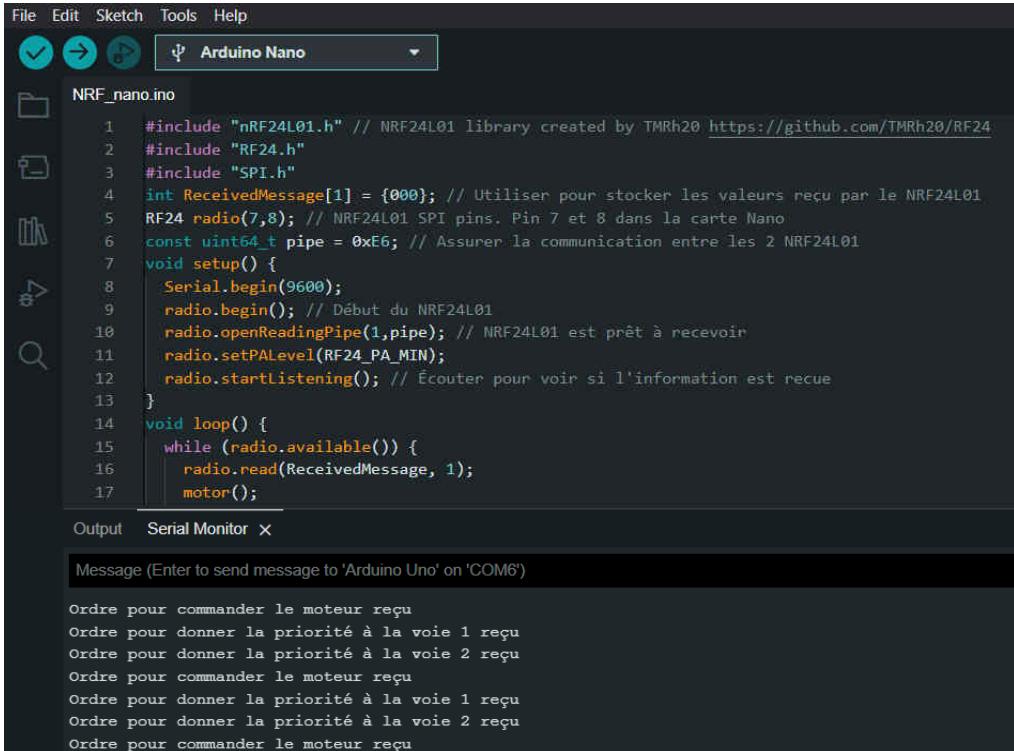


Figure 33: Réalisation expérimentale.

II.5. Commander les feux de circulation



The screenshot shows the Arduino IDE interface with the sketch named "NRF_nano.ino". The code uses the nRF24L01 library to receive messages via radio frequency. It initializes the radio, opens a reading pipe, and starts listening for messages. In the loop, it checks if the radio has available data, reads it into a buffer, and then calls a function named "motor()". The serial monitor window shows several received messages in French, such as "Ordre pour commander le moteur reçu" (Order to command the motor received) and "Ordre pour donner la priorité à la voie 1 reçue" (Order to give priority to road 1 received).

```
#include "nRF24L01.h" // nRF24L01 library created by TMRh20 https://github.com/TMRh20/RF24
#include "RF24.h"
#include "SPI.h"
int ReceivedMessage[1] = {000}; // Utiliser pour stocker les valeurs reçus par le nRF24L01
RF24 radio(7,8); // nRF24L01 SPI pins. Pin 7 et 8 dans la carte Nano
const uint64_t pipe = 0xE6; // Assurer la communication entre les 2 nRF24L01
void setup() {
    Serial.begin(9600);
    radio.begin(); // Début du nRF24L01
    radio.openReadingPipe(1,pipe); // nRF24L01 est prêt à recevoir
    radio.setPALevel(RF24_PA_MIN);
    radio.startListening(); // Écouter pour voir si l'information est reçue
}
void loop() {
    while (radio.available()) {
        radio.read(ReceivedMessage, 1);
        motor();
    }
}
```

Message (Enter to send message to 'Arduino Uno' on 'COM6')

```
Ordre pour commander le moteur reçu
Ordre pour donner la priorité à la voie 1 reçue
Ordre pour donner la priorité à la voie 2 reçue
Ordre pour commander le moteur reçu
Ordre pour donner la priorité à la voie 1 reçue
Ordre pour donner la priorité à la voie 2 reçue
Ordre pour commander le moteur reçu
```

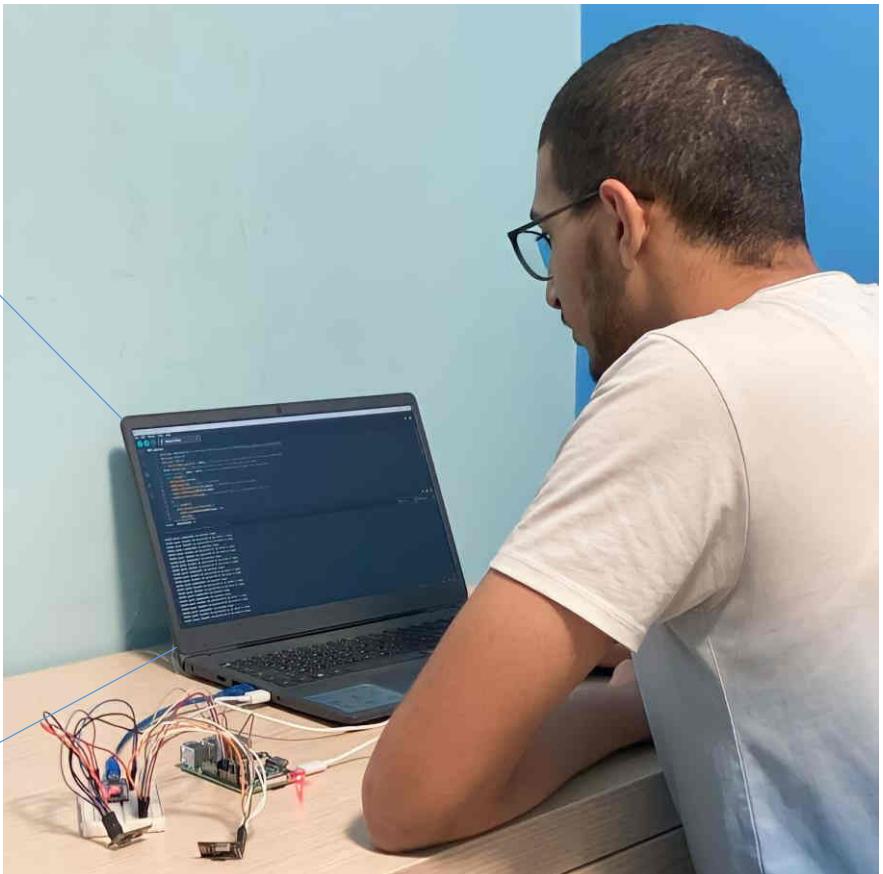


Figure 34: Résultats de la réalisation expérimentale.

Voir l'annexe 6

III. Prototype

III. Réalisation d'un prototype du système

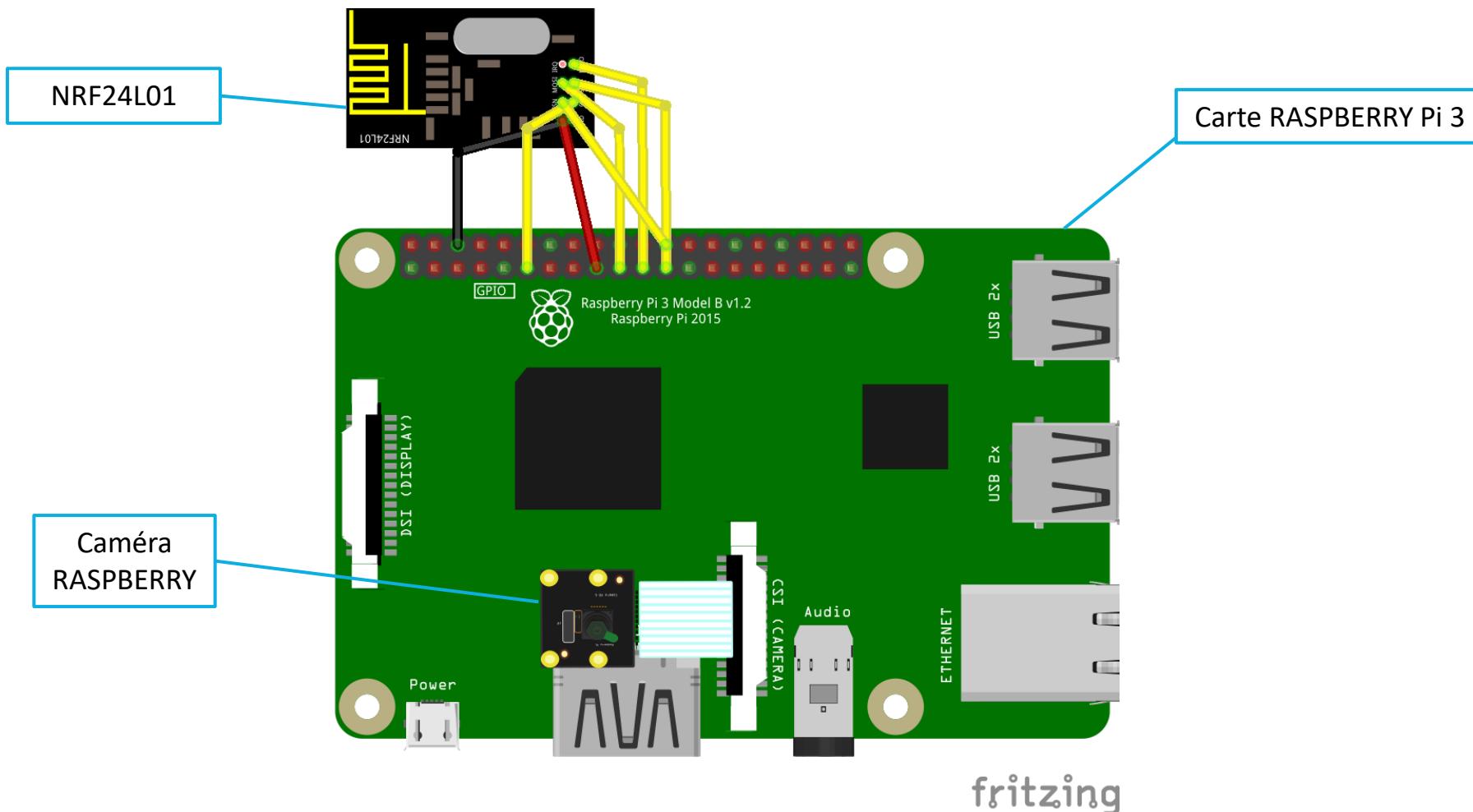


Figure 35: Vue schématique et branchement des composants liés au boîtier.

III. Réalisation d'un prototype du système

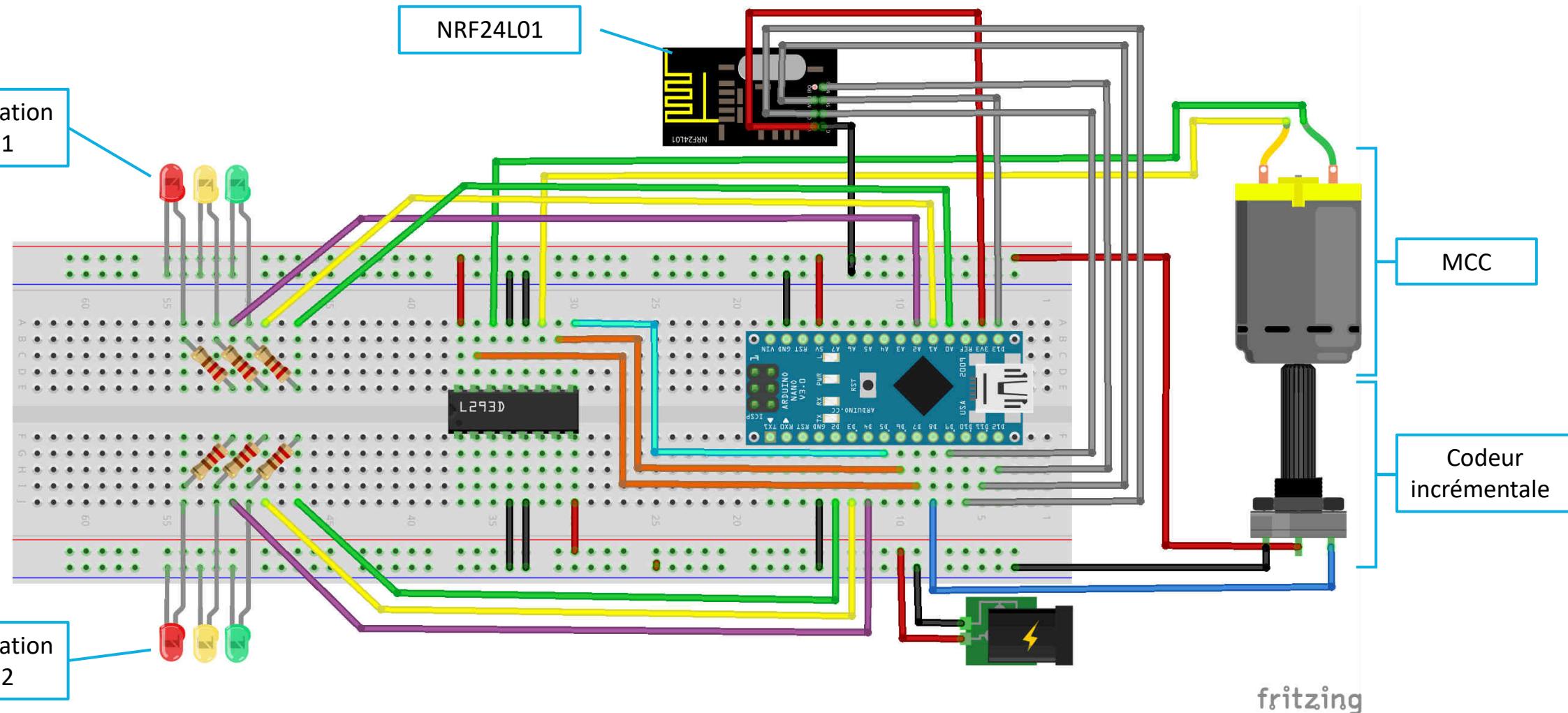
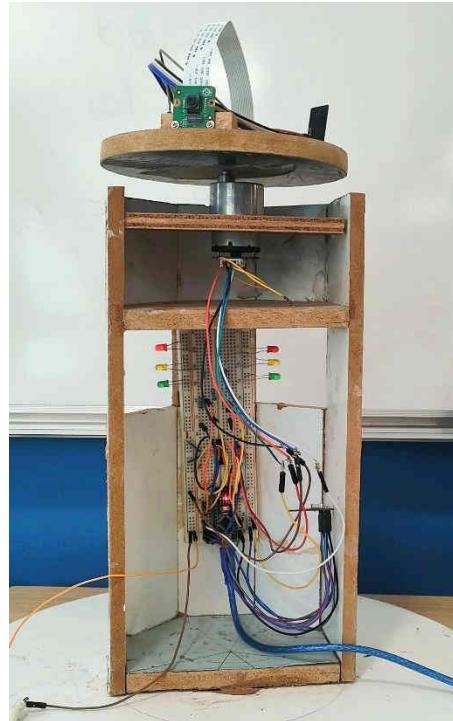


Figure 36: Vue schématique et branchement des composants liés au bâti.

III. Réalisation d'un prototype du système



Voir l'annexe 7

Figure 37: Réalisation du prototype.

IV. Conclusion

Merci pour votre attention.



ANNEXE 1

Programme de la réponse du motoréducteur à un échelon de tension (24v).

```
#include <SimpleTimer.h>

SimpleTimer timer; // Timer pour l'échantillonnage

const int encoderPinA = 3; // Pin d'entrée de l'encodeur
unsigned int n = 0; // Compteur de tick de la codeuse
const int frequence_echantillonnage = 10; // Fréquence d'exécution
de l'asservissement
const int fonte = 12;

void setup() {
  Serial.begin(9600);
  pinMode(encoderPinA, INPUT);
  attachInterrupt(digitalPinToInterrupt(encoderPinA), compteur,
  CHANGE);

// Interruption pour appeler motorControl toutes les 10ms
  timer.setInterval(1000/frequence_echantillonnage,motorControl);
}

void loop() {
  timer.run(); //On fait tourner l'horloge
}

}

void compteur() {
n++;
}

void motorControl() {
float theta = (6.28*n)/fonte;
int TE=1000/frequence_echantillonnage;
float omega = (theta/TE);
float vitesse =60*omega/6.28;
n = 0;

Serial.print(millis()/1000.);
Serial print(";");
Serial print(vitesse);
Serial print(";");
Serial print(theta);
Serial print(";");
Serial.println(n);
}
```

ANNEXE 2 (1/2)

Programme d'asservissement en position du moteur avec correction.

```
// Définition des broches
const byte motorEnablePin = 8;
const byte motorS1Pin = 9;
const byte motorS2Pin = 10;
const byte motButtonPin = 7;
const int encoderPinA = 11;

// Variables de contrôle du moteur
float maxSpeed = 1.571;
float accelerationTime = 0.5;
float constantSpeedTime = 0.5;
float decelerationTime = 0.5;
float Kp = 51.0;
int previousAngle = 0;
unsigned long previousTime = 0;

//Déclaration des fonctions
int getCurrentAngle();

void setup() {
pinMode(motorEnablePin, OUTPUT);
pinMode(motorS1Pin, OUTPUT);
pinMode(motorS2Pin, OUTPUT);
pinMode(motButtonPin, INPUT);
digitalWrite(motorEnablePin, HIGH);
Serial.begin(9600);
}

void loop() {
if (digitalRead(motButtonPin)) {
float motorSpeed = 0.0;
unsigned long startTime = millis();
unsigned long endTime = 2.2 * 1000;
while (millis() < endTime) {
float elapsedTime = (millis() - startTime) / 1000.0;
if (elapsedTime < accelerationTime) {
motorSpeed = (maxSpeed / accelerationTime) * elapsedTime;
} else if (elapsedTime < accelerationTime + constantSpeedTime) {
motorSpeed = maxSpeed;
} else {
float decelerationElapsedTime = elapsedTime - (accelerationTime +
constantSpeedTime);
motorSpeed = maxSpeed - (maxSpeed / decelerationTime) *
decelerationElapsedTime;
}
int currentAngle = getCurrentAngle();
int angleChange = currentAngle - previousAngle;
float targetAngle = 0.0;
if (elapsedTime < accelerationTime) {
targetAngle = 0.5 * motorSpeed * elapsedTime;
} else if (elapsedTime < accelerationTime + constantSpeedTime) {
targetAngle = maxSpeed * (elapsedTime - accelerationTime) + 0.5 *
maxSpeed * accelerationTime;
}
}
```

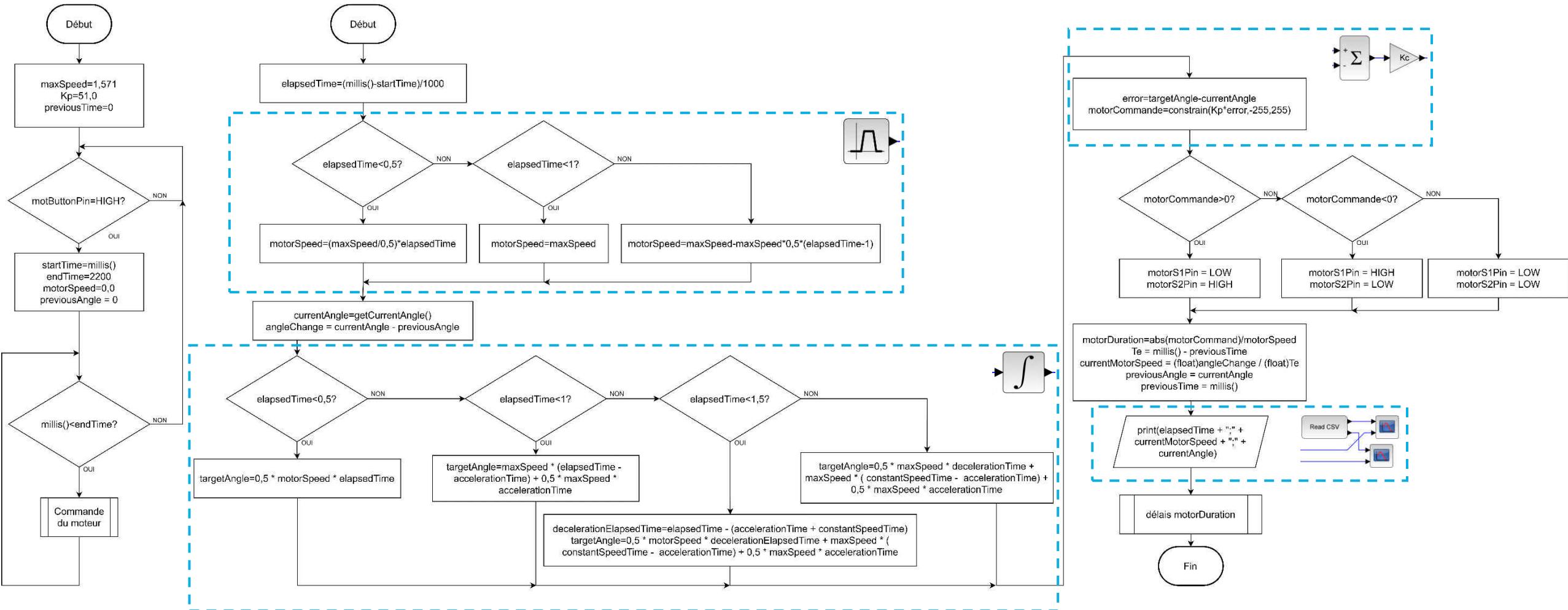
ANNEXE 2 (2/2)

```
} else if (elapsedTime < accelerationTime + constantSpeedTime +  
decelerationTime) {  
float decelerationElapsedTime = elapsedTime - (accelerationTime +  
constantSpeedTime);  
targetAngle = 0.5 * motorSpeed * decelerationElapsedTime + maxS  
constantSpeedTime - accelerationTime) + 0.5 * maxSpeed *  
accelerationTime;  
} else {  
targetAngle = 0.5 * maxSpeed * decelerationTime + maxSpeed * (  
constantSpeedTime - accelerationTime) + 0.5 * maxSpeed *  
accelerationTime;  
}  
int targetAngleInt = int(targetAngle * 180.0 / 3.14159);  
int error = targetAngle - currentAngle;  
int motorCommand = Kp * error;  
motorCommand = constrain(motorCommand, -255, 255);  
if (motorCommand > 0) {  
digitalWrite(motorS1Pin, LOW);  
digitalWrite(motorS2Pin, HIGH);  
} else if (motorCommand < 0) {  
digitalWrite(motorS1Pin, HIGH);  
digitalWrite(motorS2Pin, LOW);  
} else {  
digitalWrite(motorS1Pin, LOW);  
digitalWrite(motorS2Pin, LOW);  
}
```

```
int motorDuration = abs(motorCommand) / motorSpeed;
unsigned long Te = millis() - previousTime;
float currentMotorSpeed = (float)angleChange / (float)Te;
Serial.print(elapsedTime, 4);
(Serial.print(";");
Serial.print(motorSpeed, 4);
Serial.print(";");
Serial.println(currentAngle * 3.14 / 180.0, 4);
delay(motorDuration)
previousAngle = currentAngle;
previousTime = millis();
}
}
}
}
}
}
}
}
}

int getCurrentAngle() {
bool encoderAPinState = digitalRead(encoderPinA);
int encoderState = encoderAPinState;
const int angleOffsets[12] = {0, 30, 60, 90, 120, 150, 180, 210, 240,
270, 300, 330};
int angle = angleOffsets[encoderState];
return angle;
}
```

ANNEXE 3



ANNEXE 4

Programme de la comparaison entre les deux voies.

```
def comparaison(Ni): #Ni =[N1,N2,N3,N4]
    s = (Ni[0]+Ni[2],Ni[1]+Ni[3])
    n = s.index(max(s))
    if n%2 == 0: return 'V1'
    return 'V2'
```

Programme du traitement de la priorité.

```
def trtv1():
    global N
    global M
    if Ni[0]==0 and Ni[2]==0:
        cmd_light2() #Fonction qui allume le feux vert de la voie 2
    (voir annexe 5)
        M+=1
        N=0
    else:
        cmd_light1() #Fonction qui allume le feux vert de la voie 1
    (voir annexe 5)
        N+=1
        M=0
def trtv2():
    global N
    global M
    if Ni[1]==0 and Ni[3]==0:
        cmd_light1()
        N+=1
```

```
        M=0
    else:
        cmd_light2()
        M+=1
        N=0
def cmd_light(Vpri): #Vpri='V1' ou Vpri='V2'
    global N
    global M
    if N==3:
        N=0
        trtv2()
    elif M==3:
        M=0
        trtv2()
    else:
        if Vpri=='V1': trtv1()
        else: trtv2()
```

ANNEXE 5

Programme d'allumage du feu vert des voies 1 et 2.

```
//Définition des broches  
  
const byte R1 = 4;  
const byte Y1 = 3;  
const byte G1 = 2;  
const byte R2 = 16;  
const byte Y2 = 15;  
const byte G2 = 14;  
const byte AllmV1 = 5;  
const byte AllmV2 = 6;  
  
//Variables de contrôle des feux  
int avoidRepeatingLight1 = 0;  
int avoidRepeatingLight2 = 0;  
  
//Déclaration des fonctions  
void lightControl();  
  
void setup()  
{  
    pinMode(R1,OUTPUT);  
  
    pinMode(Y1,OUTPUT);  
    pinMode(G1,OUTPUT);  
    pinMode(R2,OUTPUT);  
    pinMode(Y2,OUTPUT);  
    pinMode(G2,OUTPUT);  
    pinMode(AllmV1, INPUT);  
    pinMode(AllmV2, INPUT);  
}  
  
void loop()  
{  
    lightControl();  
}  
void lightControl()  
{  
    if (digitalRead(AllmV1))  
    {  
        digitalWrite(G2,LOW);  
        digitalWrite(Y2,HIGH);  
        delay(4000);  
        digitalWrite(Y2,LOW);  
        digitalWrite(R2,HIGH);  
        delay(2000);  
        digitalWrite(R1,LOW);  
        digitalWrite(G1,HIGH);  
        avoidRepeatingLight1++;  
        avoidRepeatingLight2 = 0;  
    }  
    if (digitalRead(AllmV2))  
    {  
        digitalWrite(G1,LOW);  
        digitalWrite(Y1,HIGH);  
        delay(4000);  
        digitalWrite(Y1,LOW);  
        digitalWrite(R1,HIGH);  
        delay(2000);  
        digitalWrite(R2,LOW);  
        digitalWrite(G2,HIGH);  
        avoidRepeatingLight2++;  
        avoidRepeatingLight1 = 0;  
    }  
}
```

ANNEXE 6

Programmes du test de la communication entre la carte Raspberry et la carte Arduino.

```
import RPi.GPIO as GPIO
from lib_nrf24 import NRF24
import time
import spidev
GPIO.setmode(GPIO.BCM)
pipe = 0xE6
radio = NRF24(GPIO, spidev.SpiDev())
radio.begin(0,17)
radio.setPALevel(NRF24.PA_MIN)
#Niveau de puissance minimum car
#Raspberry et Arduino sont proches
radio.openWritingPipe(pipe)
SentMessage = [0]
while True:
    #Commander le moteur
    SentMessage[0] = 111
    radio.write(SentMessage)
    time.sleep(1)
    #Donner la priorité à la voie 1
    SentMessage[0] = 1
    radio.write(SentMessage)
    time.sleep(1)
    #Donner la priorité à la voie 2
    SentMessage[0] = 10
    radio.write(SentMessage)
    time.sleep(1)
```

```
#include "RF24.h"

// Variables de communication radio
int ReceivedMessage[1] = {000};
RF24 radio(7,8);
const uint64_t pipe = 0xE6;

// Déclaration des fonctions
void motorControl(int command);
void lightControl(int command);

void setup() {
Serial.begin(9600);
radio.begin();
radio.openReadingPipe(1,pipe);
radio.startListening();
}

void loop() {
while (radio.available()) {
radio.read(ReceivedMessage, 1);
motorControl(ReceivedMessage[0]);
lightControl(ReceivedMessage[0]);
}
}
```

```
void motorControl(int command){
if (command == 111) {
Serial.println("Ordre pour commander le
moteur reçu\n");
}
}

void lightControl(int command){
if (command == 1) {
Serial.println("Ordre pour donner la
priorité à la voie 1 reçu");
}
}

if (command == 10){
Serial.println("Ordre pour donner la
priorité à la voie 2 reçu");
}
}
```

ANNEXE 7 (1/4)

Programme du prototype dans la carte Raspberry.

```
import RPi.GPIO as GPIO
from lib_nrf24 import NRF24
import time
import spidev
GPIO.setmode(GPIO.BCM)
pipe = 0xE6
radio = NRF24(GPIO, spidev.SpiDev())
radio.begin(0,17)
radio.setPALevel(NRF24.PA_MIN)
radio.openWritingPipe(pipe)
SentMessage = [0]
def capter():
    pass
def mot_90():
    SentMessage[0] = 111
    radio.write(SentMessage)
def cmd_light1():
    SentMessage[0] = 1
    radio.write(SentMessage)
def cmd_light2():
    SentMessage[0] = 10
    radio.write(SentMessage)
def trtv2():
    global N
    global M
```

```
if Ni[1]==0 and Ni[3]==0:
    cmd_light1()
    N+=1
    M=0
else:
    cmd_light2()
    M+=1
    N=0
def trtv1():
    global N
    global M
    if Ni[0]==0 and Ni[2]==0:
        cmd_light2()
        M+=1
        N=0
    else:
        cmd_light1()
        N+=1
        M=0
def cmd_light(Vpri):
    global N
    global M
    if N==3:
        N=0
        trtv2()
    elif M==3:
        M=0
        trtv2()
    else:
        if Vpri=='V1': trtv1()
        else: trtv2()
def comparaison(Ni):
    s=(Ni[0]+Ni[2],Ni[1]+Ni[3])
    n = s.index(max(s))
    if n % 2 == 0: return 'V1'
    return 'V2'
def comptage():
    global Ni
    for _ in range(4):
        capter()
        mot_90()
        time.sleep(2)
N=0
M=0
while True:
    Ni=[]
    comptage()
    Vpri = comparaison(Ni)
    cmd_light(Vpri)
    time.sleep(8)
```

ANNEXE 7 (2/4)

Programme du prototype dans la carte Arduino.

```
#include "RF24.h"

// Définition des broches
const byte R1 = 4;
const byte Y1 = 3;
const byte G1 = 2;
const byte R2 = 16;
const byte Y2 = 15;
const byte G2 = 14;
const byte motorEnablePin = 5;
const byte motorS1Pin = 6;
const byte motorS2Pin = 7;
const int encoderPinA = 8;

// Variables de contrôle du moteur
float maxSpeed = 1.571;
float accelerationTime = 0.5;
float constantSpeedTime = 0.5;
float decelerationTime = 0.5;
float Kp = 51.0;
unsigned long previousTime = 0;

// Variables de contrôle des feux
int avoidRepeatingLight1 = 0;
int avoidRepeatingLight2 = 0;
// Variables de communication radio
int ReceivedMessage[1] = {0};
RF24 radio(9, 10);
const uint64_t pipe = 0xE6;

// Déclaration des fonctions
void motorControl(int command);
void lightControl(int command);
int getCurrentAngle();

void setup() {
    radio.begin();
    radio.openReadingPipe(1, pipe);
    radio.startListening();

    pinMode(R1, OUTPUT);
    pinMode(Y1, OUTPUT);
    pinMode(G1, OUTPUT);
    pinMode(R2, OUTPUT);
    pinMode(Y2, OUTPUT);

    pinMode(G2, OUTPUT);

    pinMode(motorEnablePin, OUTPUT);
    pinMode(motorS1Pin, OUTPUT);
    pinMode(motorS2Pin, OUTPUT);
    digitalWrite(motorEnablePin, HIGH);
}

void loop() {
    if (radio.available()) {
        radio.read(ReceivedMessage, 1);
        motorControl(ReceivedMessage[0]);
        lightControl(ReceivedMessage[0]);
    }
}

void motorControl(int command) {
    if (command == 111) {
        float motorSpeed = 0.0;
        unsigned long startTime = millis();
        unsigned long endTime = 2.2 * 1000;
        while (millis() < endTime) {
            float elapsedTime = (millis() - startTime) /
1000.0;
```

ANNEXE 7 (3/4)

```
if (elapsedTime < accelerationTime) {  
    motorSpeed = (maxSpeed / accelerationTime) * elapsedTime;  
} else if (elapsedTime < accelerationTime + constantSpeedTime) {  
    motorSpeed = maxSpeed;  
} else {  
    float decelerationElapsedTime = elapsedTime - (accelerationTime + constantSpeedTime);  
    motorSpeed = maxSpeed - (maxSpeed / decelerationTime) * decelerationElapsedTime;  
}  
  
int currentAngle = getCurrentAngle();  
float targetAngle = 0.0;  
if (elapsedTime < accelerationTime) {  
    targetAngle = 0.5 * motorSpeed * elapsedTime;  
} else if (elapsedTime < accelerationTime + constantSpeedTime) {  
    targetAngle = maxSpeed * (elapsedTime - accelerationTime) + 0.5 * maxSpeed * accelerationTime;  
} else if (elapsedTime < accelerationTime + constantSpeedTime + decelerationTime) {  
    float decelerationElapsedTime = elapsedTime - (accelerationTime + constantSpeedTime);  
    targetAngle = 0.5 * motorSpeed * decelerationElapsedTime + maxSpeed * (constantSpeedTime - accelerationTime) + 0.5 * maxSpeed *  
        accelerationTime;  
} else {  
    targetAngle = 0.5 * maxSpeed * decelerationTime + maxSpeed * (constantSpeedTime - accelerationTime) + 0.5 * maxSpeed * accelerationTime;  
}  
int targetAngleInt = int(targetAngle * 180.0 / 3.14159);  
int error = targetAngle - currentAngle;  
int motorCommand = Kp * error;  
motorCommand = constrain(motorCommand, -255, 255);
```

ANNEXE 7 (4/4)

```
if (motorCommand > 0) {  
    digitalWrite(motorS1Pin, LOW);  
    digitalWrite(motorS2Pin, HIGH);  
} else if (motorCommand < 0) {  
    digitalWrite(motorS1Pin, HIGH);  
    digitalWrite(motorS2Pin, LOW);  
} else {  
    digitalWrite(motorS1Pin, LOW);  
    digitalWrite(motorS2Pin, LOW);  
}  
  
int motorDuration = abs(motorCommand)  
/ motorSpeed;  
  
unsigned long currentTime = millis();  
if (currentTime - previousTime >=  
motorDuration) {  
previousTime = currentTime;  
}  
}  
}  
}
```

```
int getCurrentAngle() {  
    bool encoderAPinState =  
digitalRead(encoderPinA);  
    int encoderState = encoderAPinState;  
    const int angleOffsets[12] = {0, 30, 60, 90,  
120, 150, 180, 210, 240, 270, 300, 330};  
    int angle = angleOffsets[encoderState];  
    return angle;  
}  
  
void lightControl(int command) {  
    if (command == 1 &&  
avoidRepeatingLight1 == 0) {  
        digitalWrite(G2, LOW);  
        digitalWrite(Y2, HIGH);  
        delay(4000);  
        digitalWrite(Y2, LOW);  
        digitalWrite(R2, HIGH);  
        delay(2000);  
        digitalWrite(R1, LOW);  
        digitalWrite(G1, HIGH);  
        avoidRepeatingLight1++;  
        avoidRepeatingLight2 = 0;  
    }  
}
```

```
if (command == 10 &&  
avoidRepeatingLight2 == 0) {  
    digitalWrite(G1, LOW);  
    digitalWrite(Y1, HIGH);  
    delay(4000);  
    digitalWrite(Y1, LOW);  
    digitalWrite(R1, HIGH);  
    delay(2000);  
    digitalWrite(R2, LOW);  
    digitalWrite(G2, HIGH);  
    avoidRepeatingLight2++;  
    avoidRepeatingLight1 = 0;  
}
```