



TADC

Transporteur autonome des charges

Mohamed Manessouri
Code CNC : SL003T

Plan de la présentation :

1. *Introduction et problématique*

2. *Cahier des charges*

3. *Objectifs :*

Naviguer le TADC dans un graphe orienté

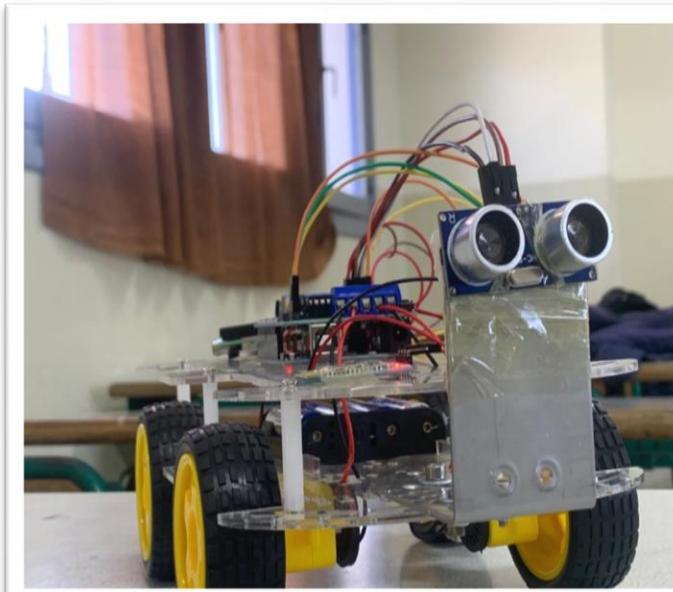
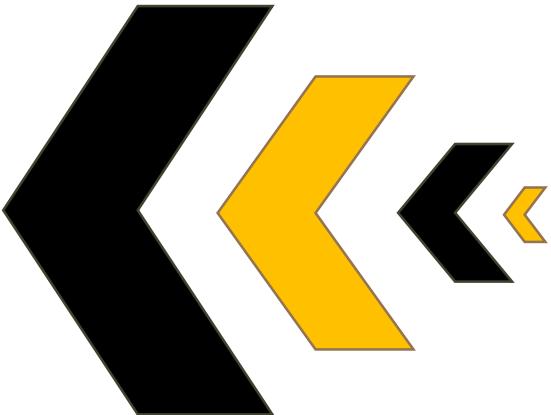
Réaction de TADC au cas d'existence d'un obstacle

Asservissement de la position du TADC

Réalisation d'un prototype final

4. *Conclusion*

Introduction :



Problématique :

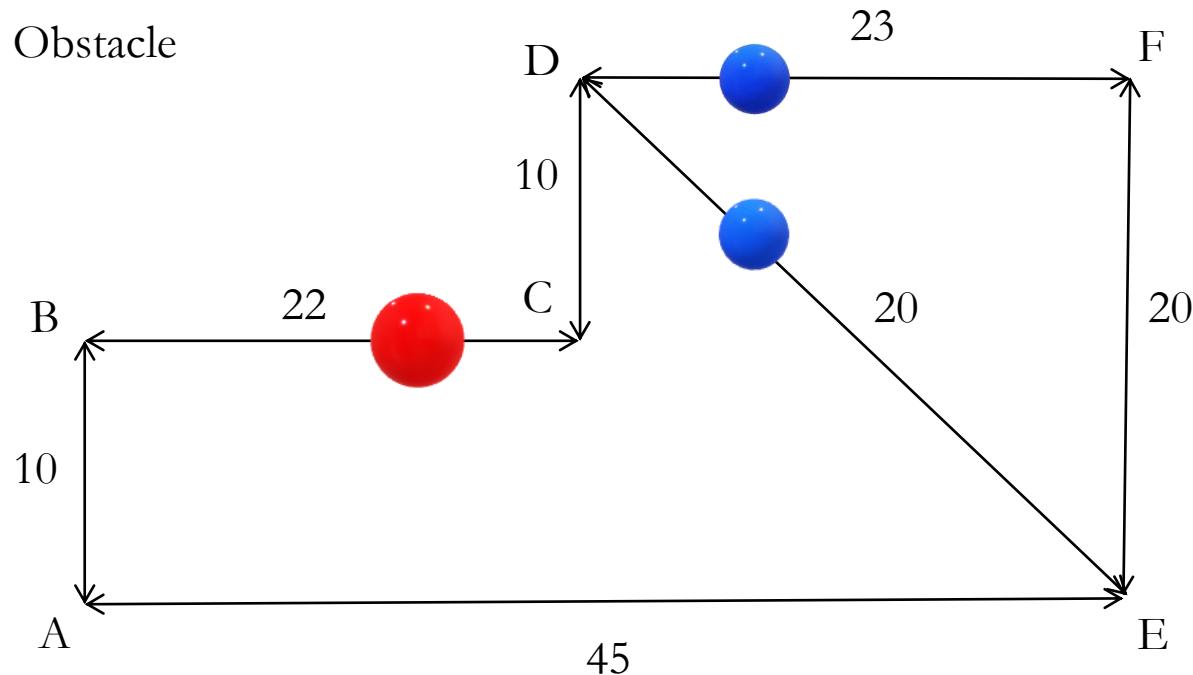
Comment peut-on rendre un **transporteur des charges**
plus autonome ?



TADC



Obstacle



Cahier des charges

- Le TADC doit répondre aux exigences de cahier des charges proposés :



Fonctionnement global du système :

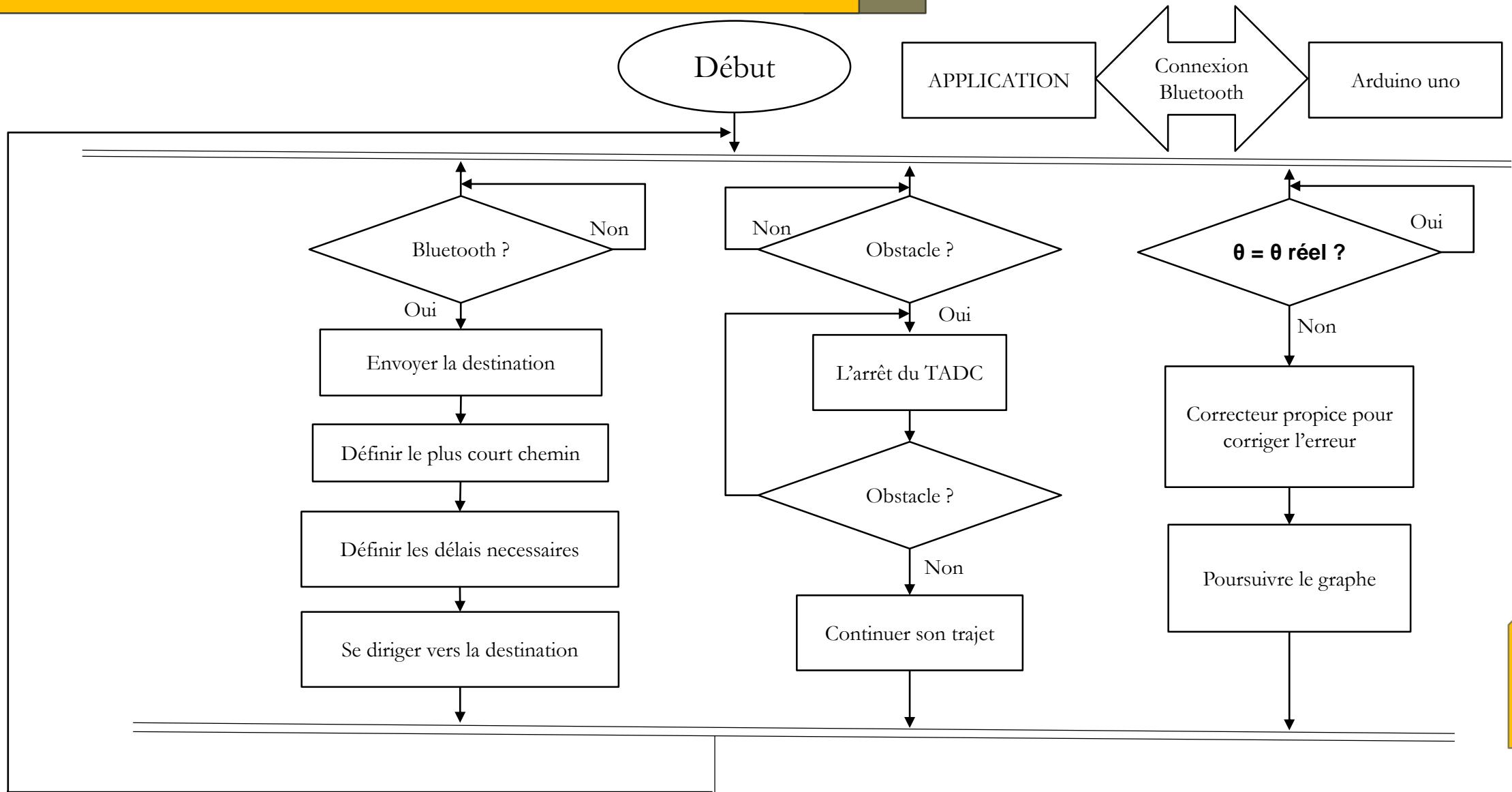


Diagramme d'exigences (REQ) :

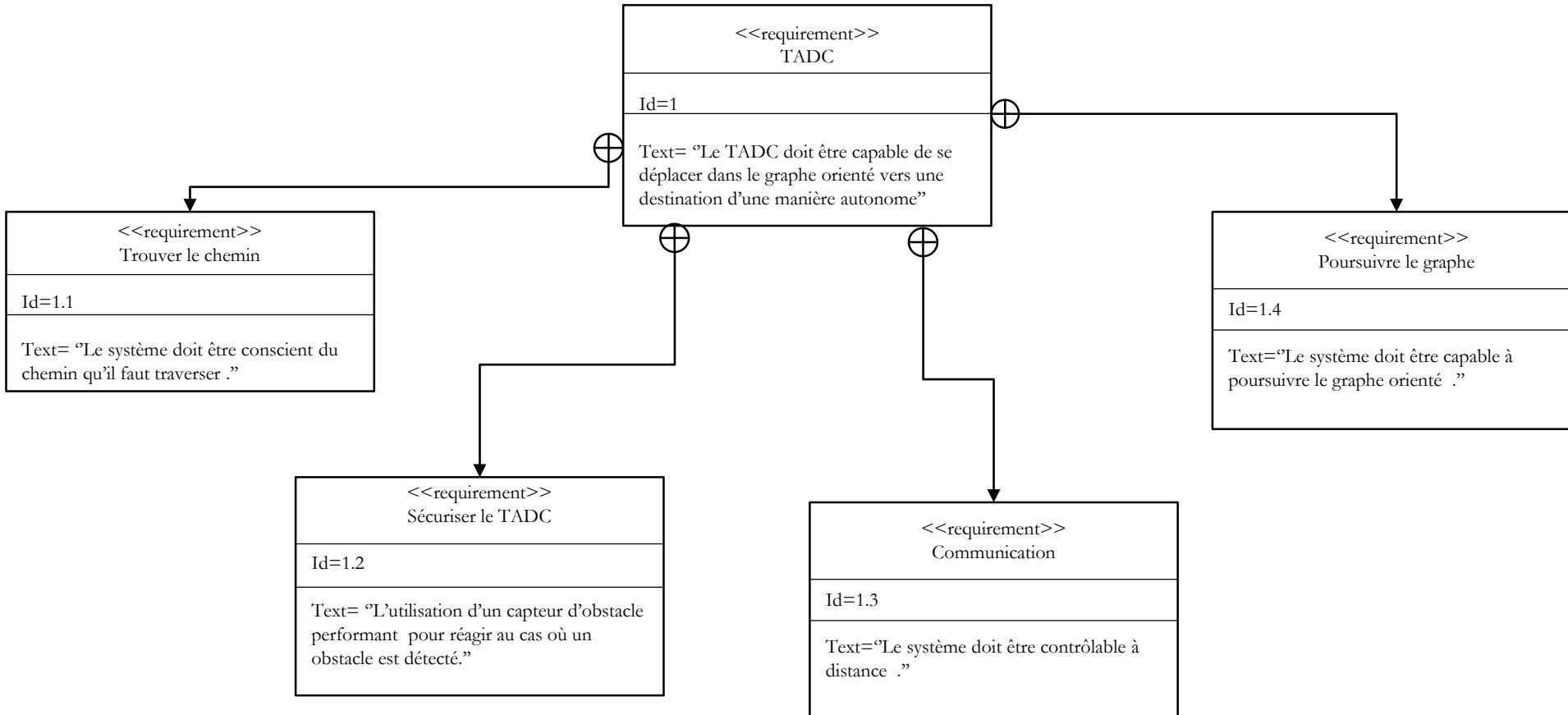


Diagramme d'exigences (suite) :

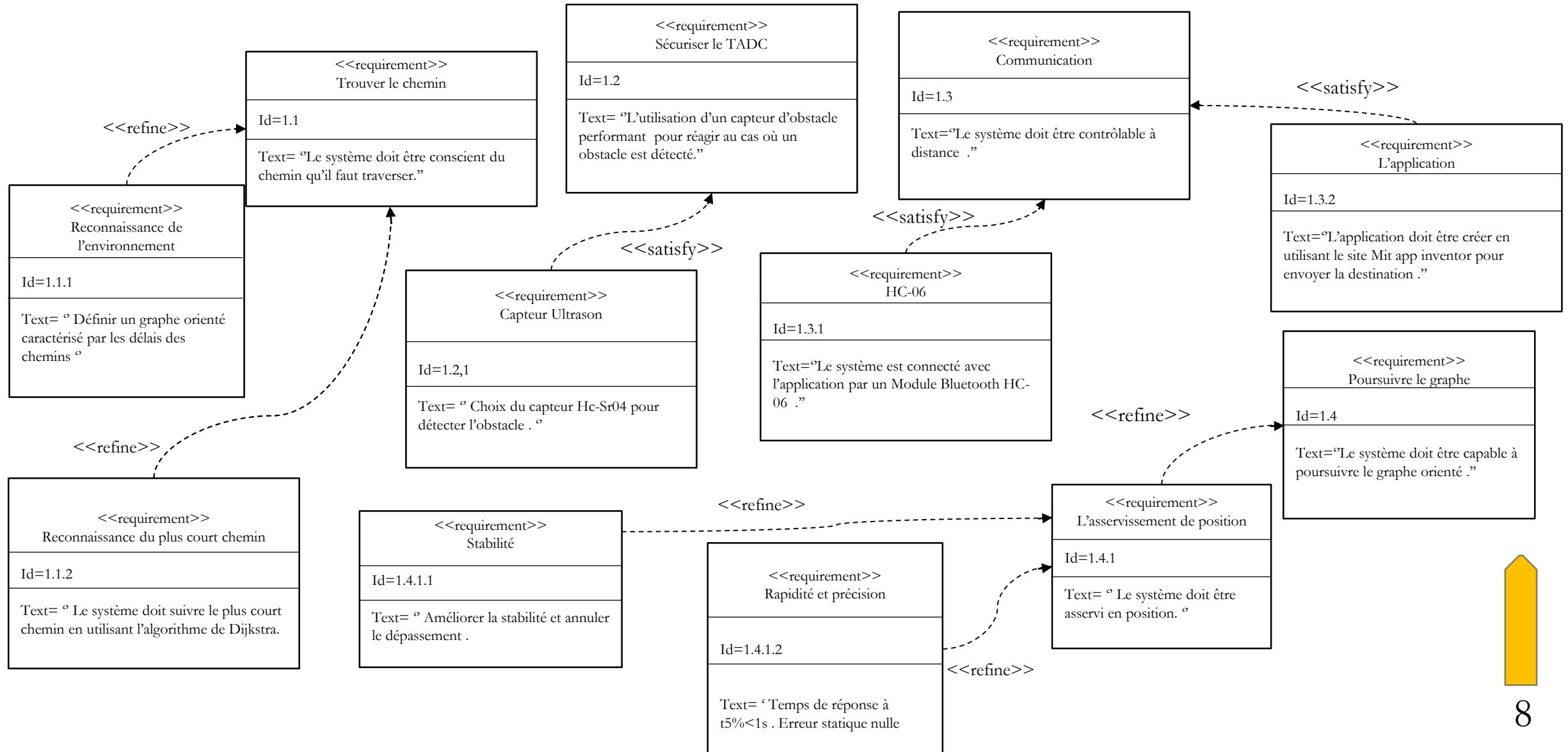
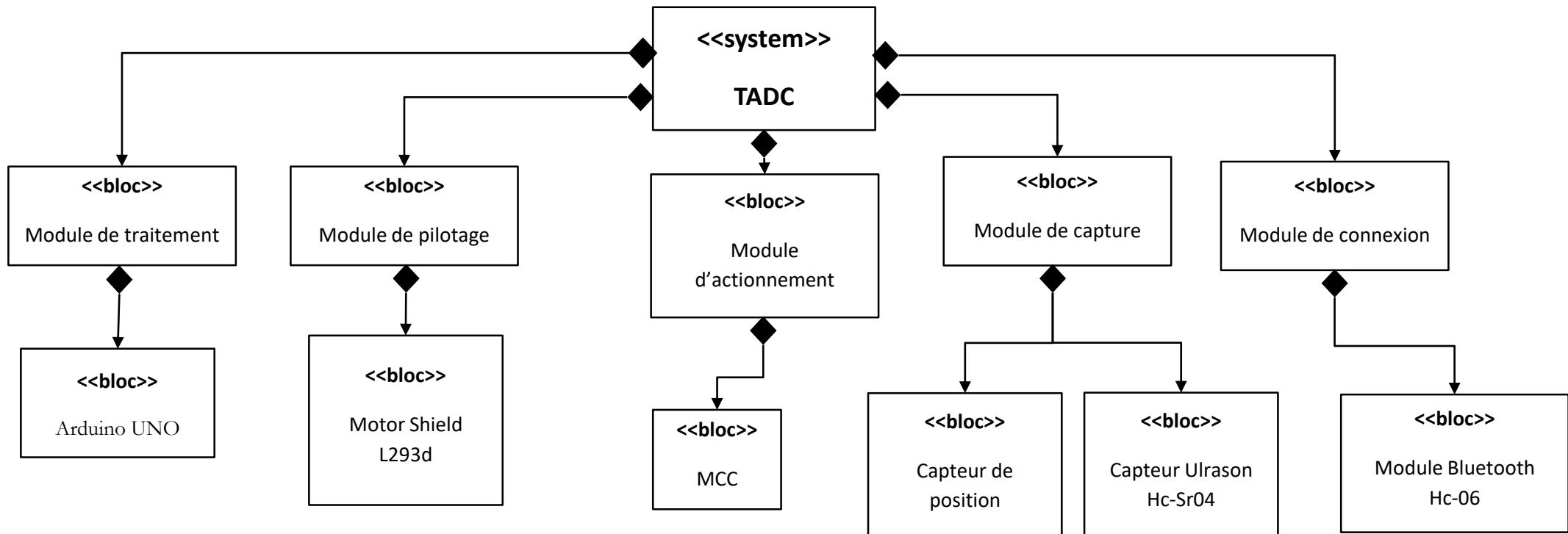
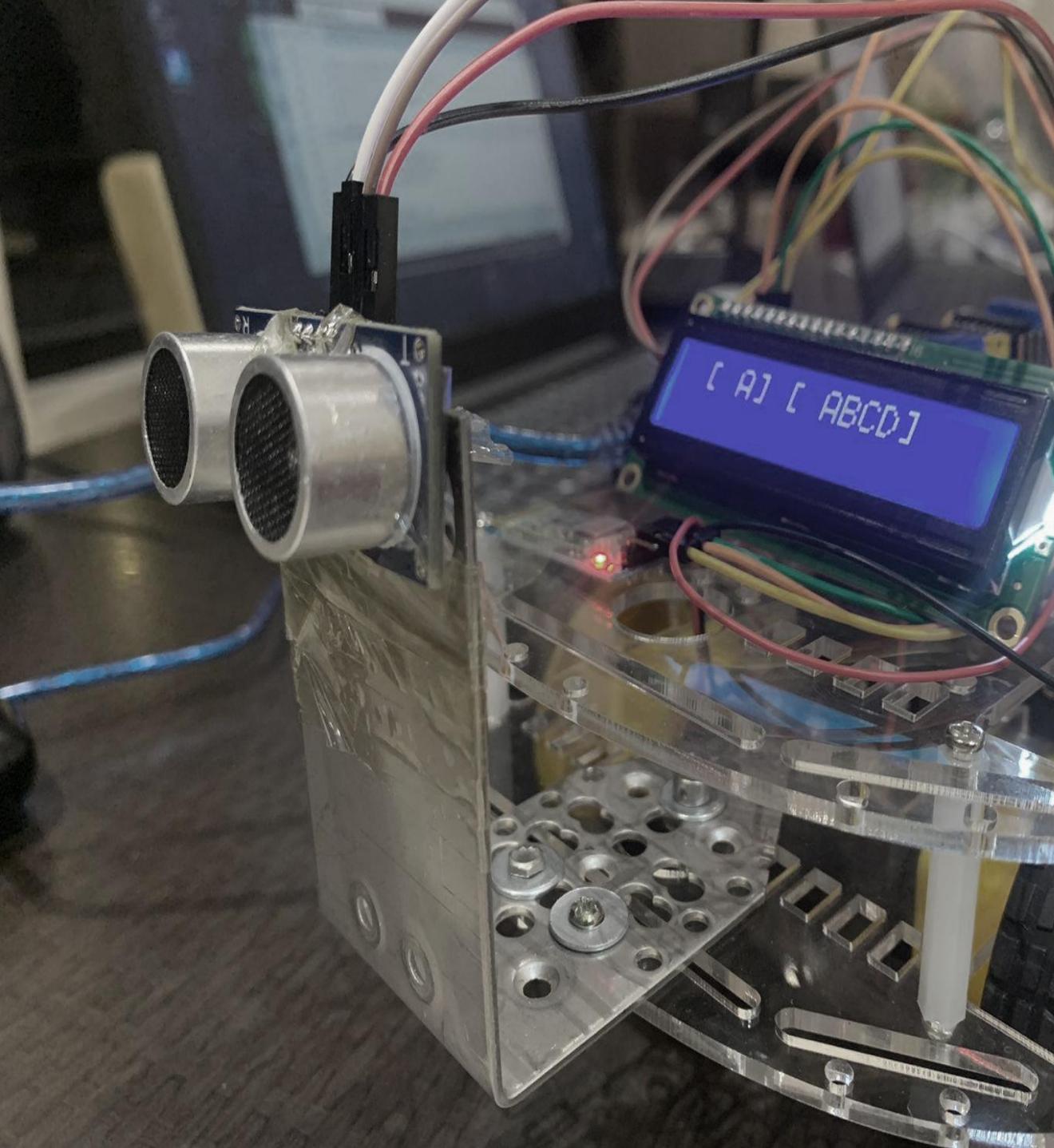


Diagramme de définition de bloc (BDD) :





OBJECTIF 1 :

Comment on peut naviguer le TADC dans un graphe orienté valué ?

- Définir le graphe .
- Trouver le plus court chemin



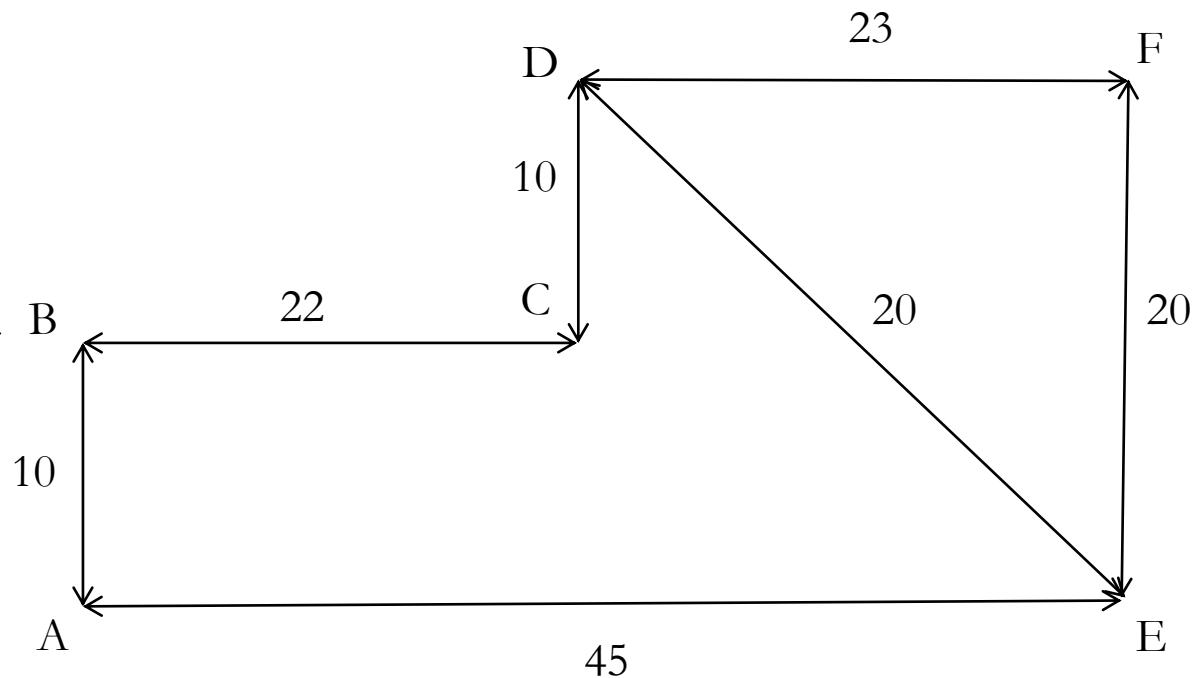
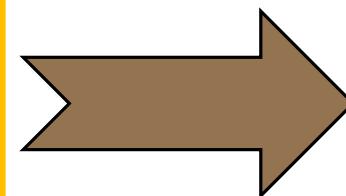


Un graphe est représenté formellement par $G(S,R,V)$:

$$S = \{ A, B, C, D, E, F \}$$

$$R = \{ AB, BC, CD, DF, AE, DE, FE \}$$

$$V = \{ 10, 22, 10, 23, 45, 20, 20 \}$$





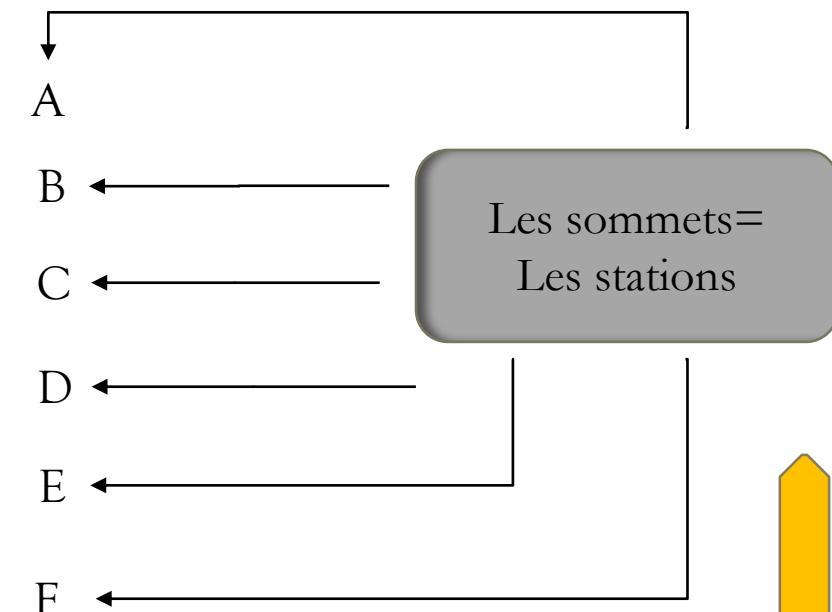
Les délais :

- ❖ On définit le graphe par une matrice adjacente des poids (délais) :

Les poids=
Les délais

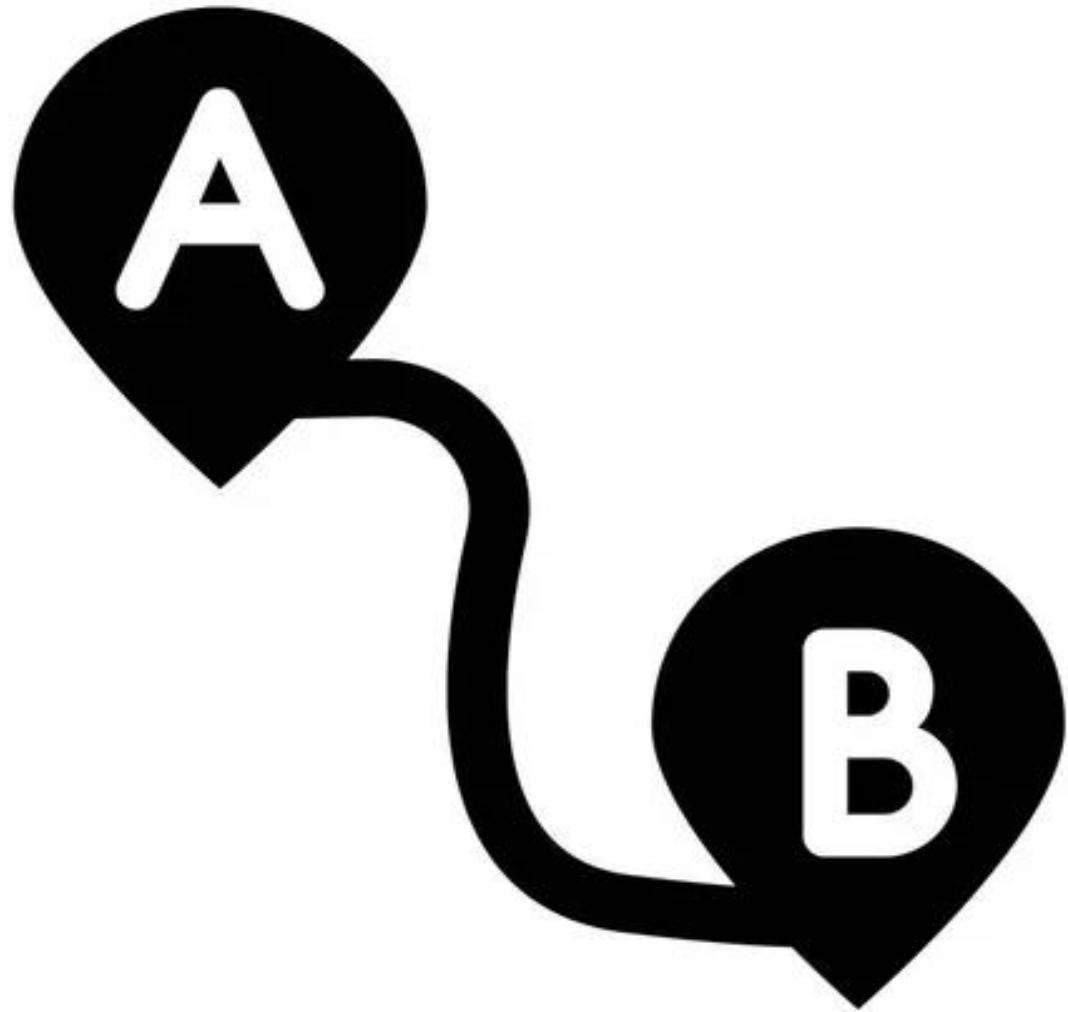
Matrice
adjacente des
poids

$$M = \begin{bmatrix} A & B & C & D & E & F \\ 0 & 10 & - & - & 45 & - \\ 10 & 0 & 22 & - & - & - \\ - & 22 & 0 & 10 & - & - \\ - & - & 10 & 0 & 20 & 23 \\ 45 & - & - & 20 & 0 & 20 \\ - & - & - & 23 & 20 & 0 \end{bmatrix}$$



Les sommets=
Les stations





❖ Etude théorique :

◦ Trouver le plus court chemin entre A et les autres destinations :



Exploitation des délais pour trouver le plus court chemin



L'algorithme de Dijkstra :

1

Efficace lorsque le graphe est dense

2

Adaptable pour résoudre différents problèmes liés à l'optimisation du chemin

3

Flexible et utilisé dans différents types de graphe

4

Large applicabilité

Parcours de l'algorithme de Dijkstra :

Matrice adjacente des poids M :

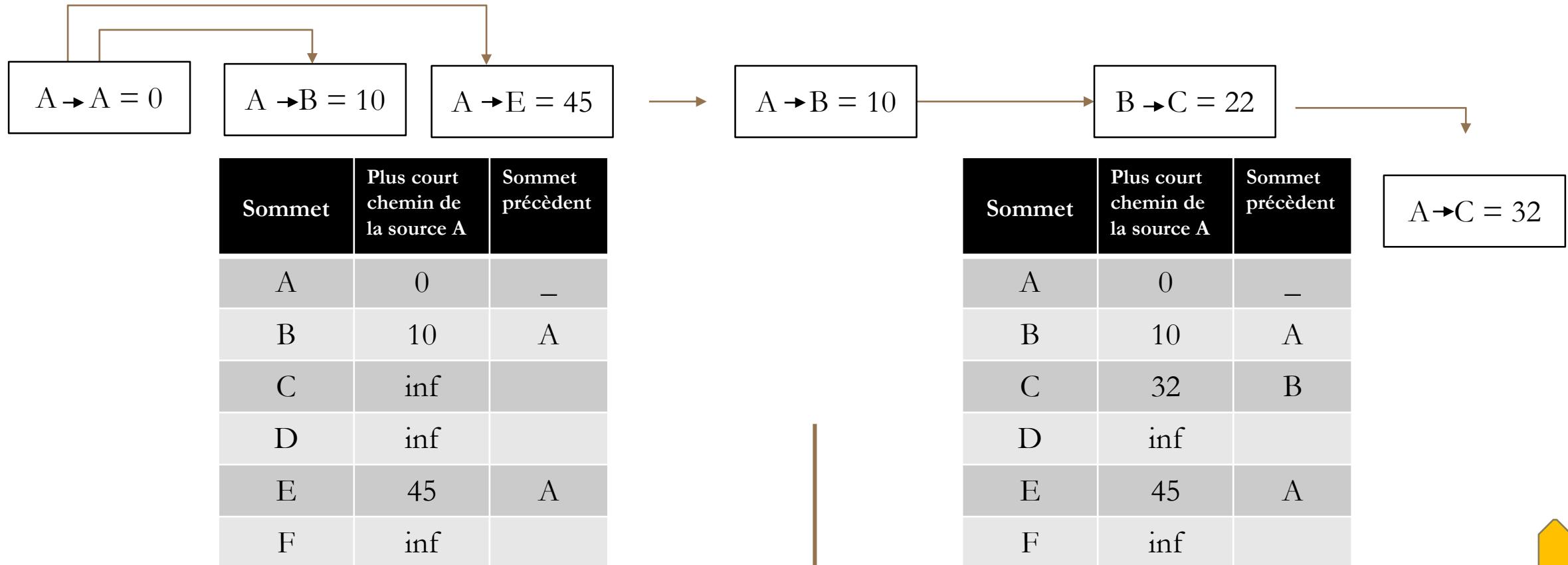
$$M = \begin{bmatrix} A & B & C & D & E & F \\ 0 & 10 & \text{inf} & \text{inf} & 45 & \text{inf} \\ 10 & 0 & 22 & \text{inf} & \text{inf} & \text{inf} \\ \text{inf} & 22 & 0 & 10 & \text{inf} & \text{inf} \\ \text{inf} & \text{inf} & 10 & 0 & 20 & 23 \\ 45 & \text{inf} & \text{inf} & 20 & 0 & 20 \\ \text{inf} & \text{inf} & \text{inf} & 23 & 20 & 0 \end{bmatrix}$$

Sommet	Plus court chemin de la source A	Sommet précédent
A	0	—
B	inf	—
C	inf	—
D	inf	—
E	inf	—
F	inf	—

*Visités: [] .

*Non visités : [A,B,C,D,E,F] .

Parcours de l'algorithme de Dijkstra :



*Visités: [A] .

*Non visités : [B,C,D,E,F] .

*Visités: [A,B] .

*Non visités : [C,D,E,F] .

Parcours de l'algorithme de Dijkstra :

$$A \rightarrow C = 32 \rightarrow C \rightarrow D = 10$$

Sommet	Plus court chemin de la source A	Sommet précédent
A	0	—
B	10	A
C	32	B
D	42	C
E	45	A
F	inf	

$$A \rightarrow D = 42 \rightarrow D \rightarrow E = 20 \rightarrow D \rightarrow F = 23$$

Sommet	Plus court chemin de la source A	Sommet précédent
A	0	—
B	10	A
C	32	B
D	42	C
E	45	A
F	65	D

*Visité: [A,B,C] . *Non visité : [D,E,F] .

*Visité: [A,B,C,D] . *Non visité : [E,F] .

Parcours de l'algorithme de Dijkstra :

$$A \rightarrow E = 45 \rightarrow E \rightarrow F = 20$$



Sommet	Plus court chemin de la source A	Sommet précédent
A	0	—
B	10	A
C	32	B
D	42	C
E	45	A
F	65	E

*Visité: [A,B,C,D,E] .

*Non visité : [F] .

*Visité: [A,B,C,D,E,F] .

*Non visité : [] .

Conclusion :

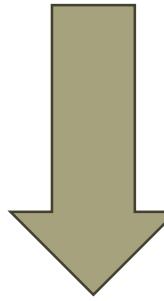
_ L'algorithme de Dijkstra donne deux listes comme résultat :

- Liste des délais minimales par rapport à la source A : [0, 10, 32, 42, 45, 65] .
- Liste des prédecesseurs de chaque station (sommet) : [-1, 0, 1, 2, 0,] = [__, A, B, C, A, E] .

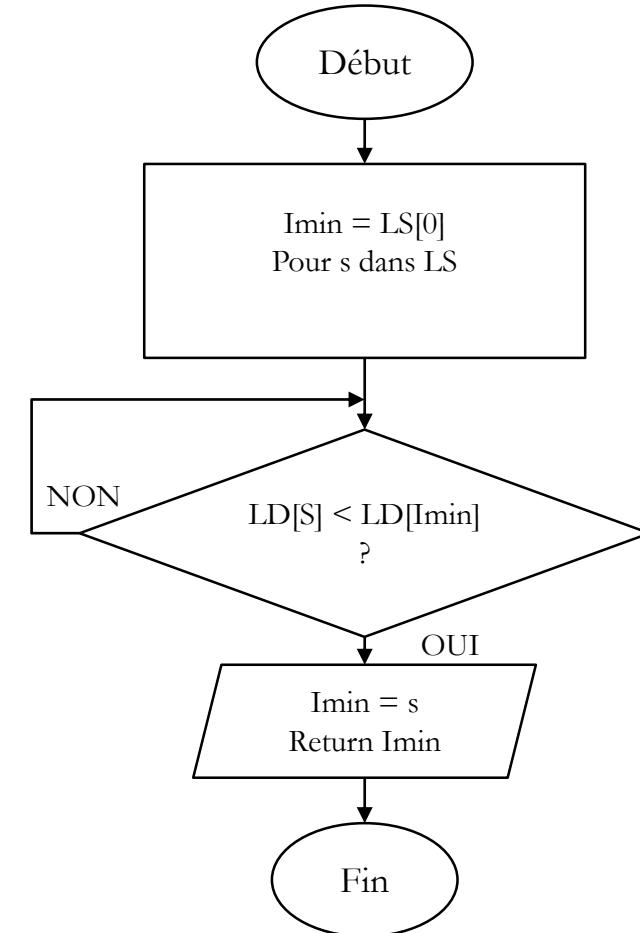
_ Il faut l'exploiter pour trouver le plus court chemin



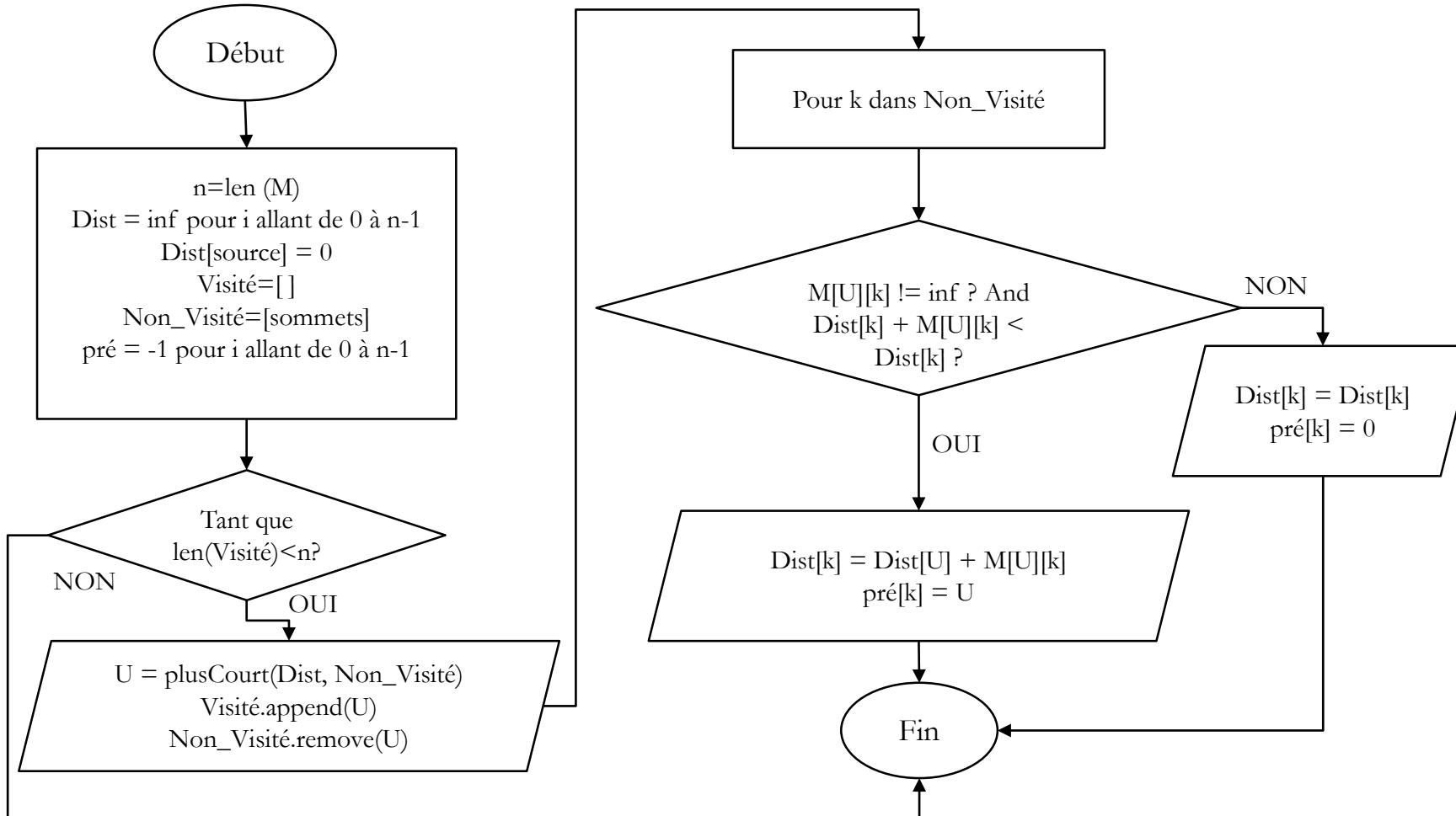
Fonction qui retourne la plus courte distance :



plusCourt (LD,LS) :
Liste des distances : LD : Dist .
liste des sommets : LS .

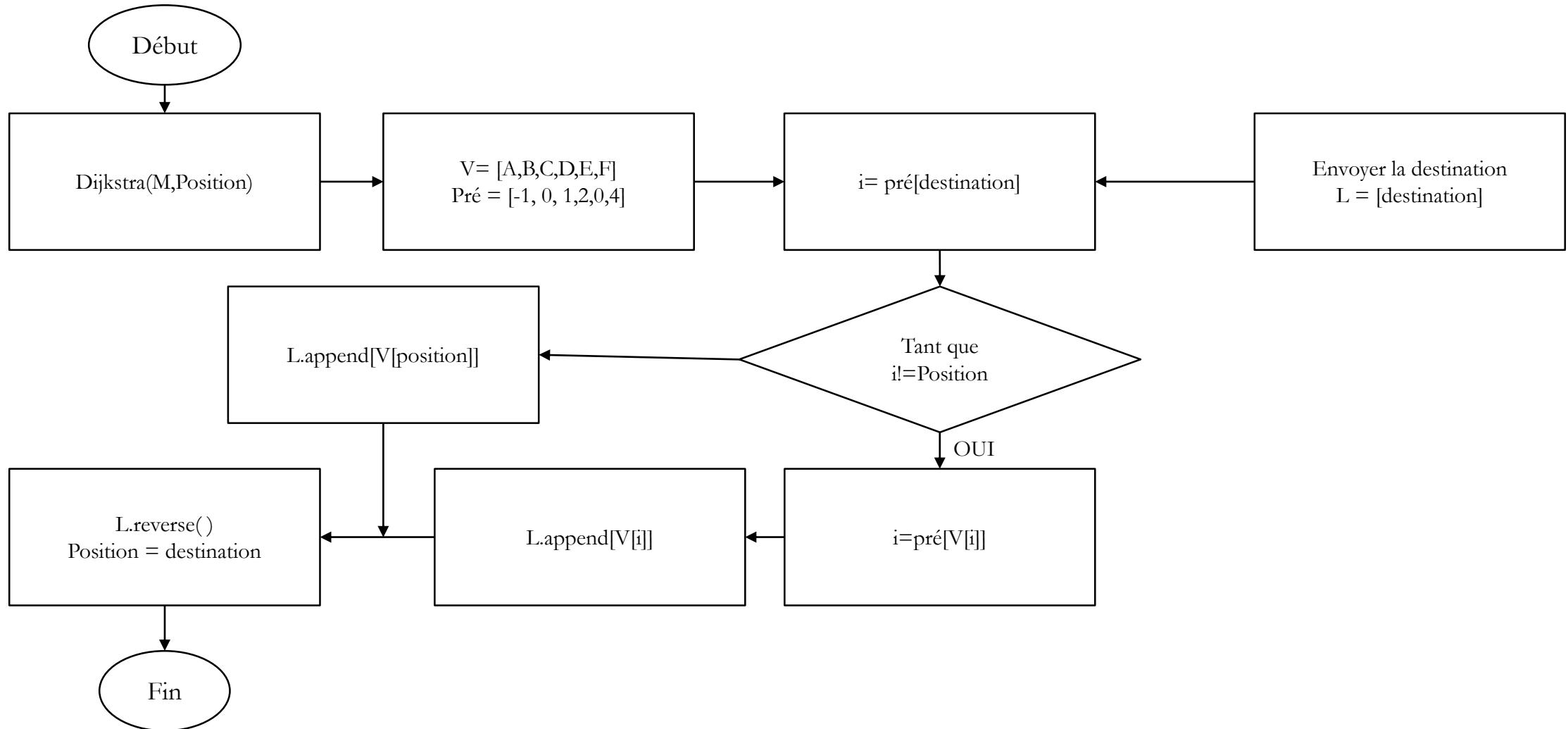


Organigramme explicatif de l'algorithme de Dijkstra :



Organigramme explicatif pour trouver le plus court chemin :

Récursivité pour retourner une liste qui contient le plus court chemin entre la position du TADC et la destination :



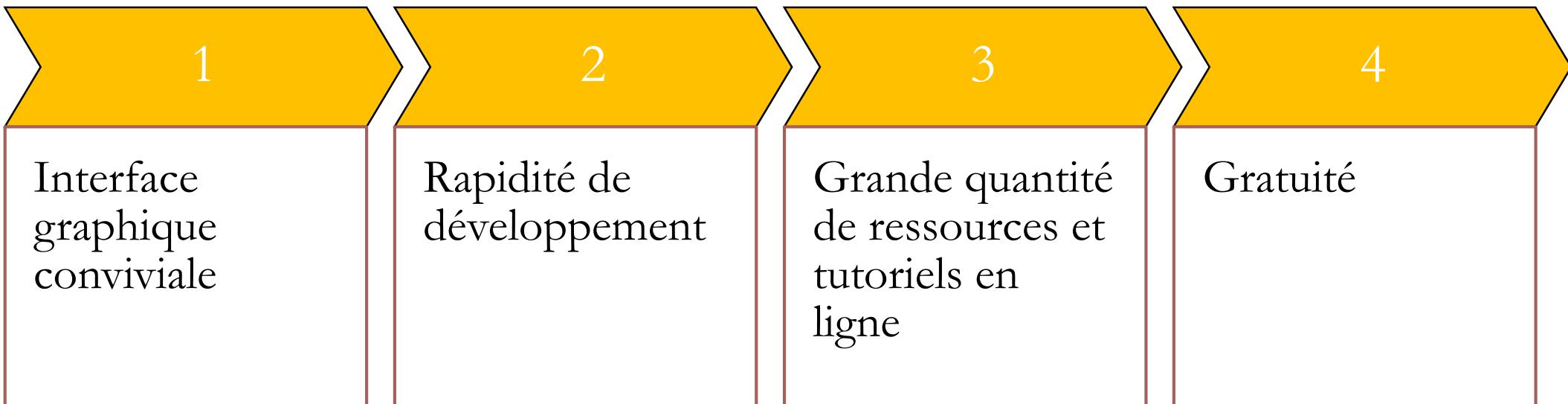


❖ Comment le TADC reçoit la commande de la Destination ?

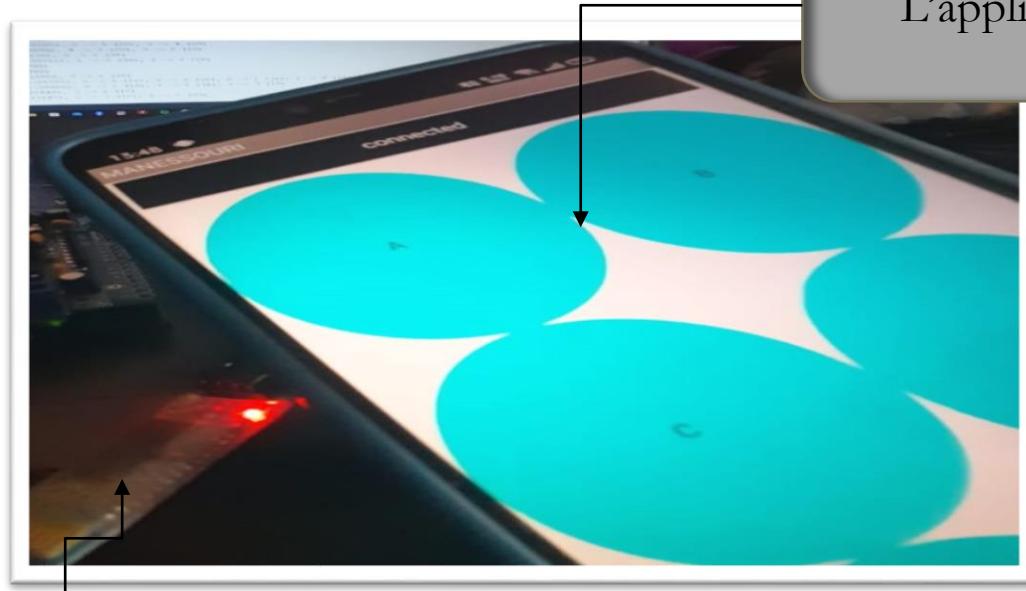
- Construire une application de commande en utilisant MIT app Inventor



MIT app Inventor :



Examiner l'efficacité de l'application:



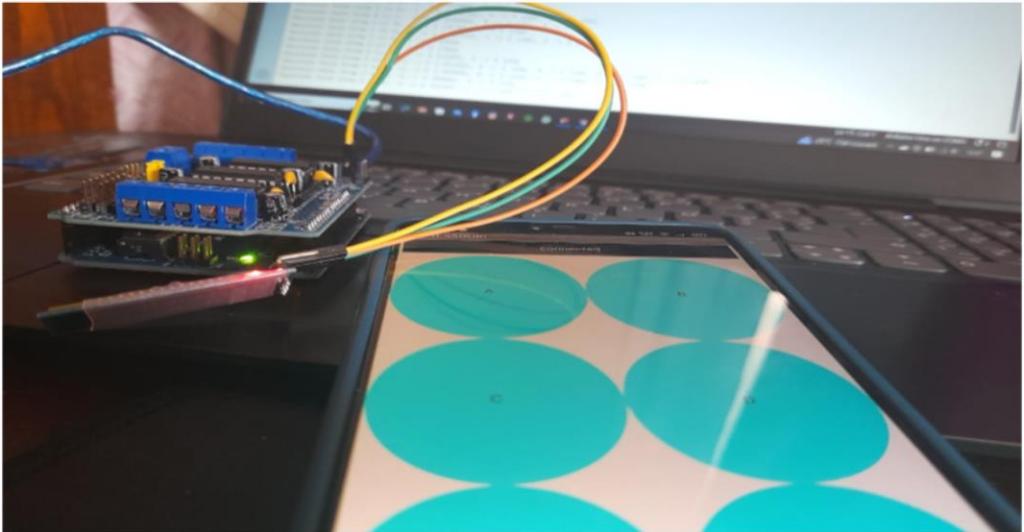
Module Bluetooth
HC_06

Output Serial Monitor ×

Message (Enter to send message to 'Ardu')

```
la destination A est envoyée
la destination B est envoyée
la destination F est envoyée
la destination D est envoyée
la destination C est envoyée
la destination E est envoyée
la destination A est envoyée
la destination D est envoyée
la destination D est envoyée
la destination F est envoyée
la destination C est envoyée
la destination E est envoyée
la destination B est envoyée
la destination A est envoyée
la destination D est envoyée
```

Expérience : Visualisation des résultats :

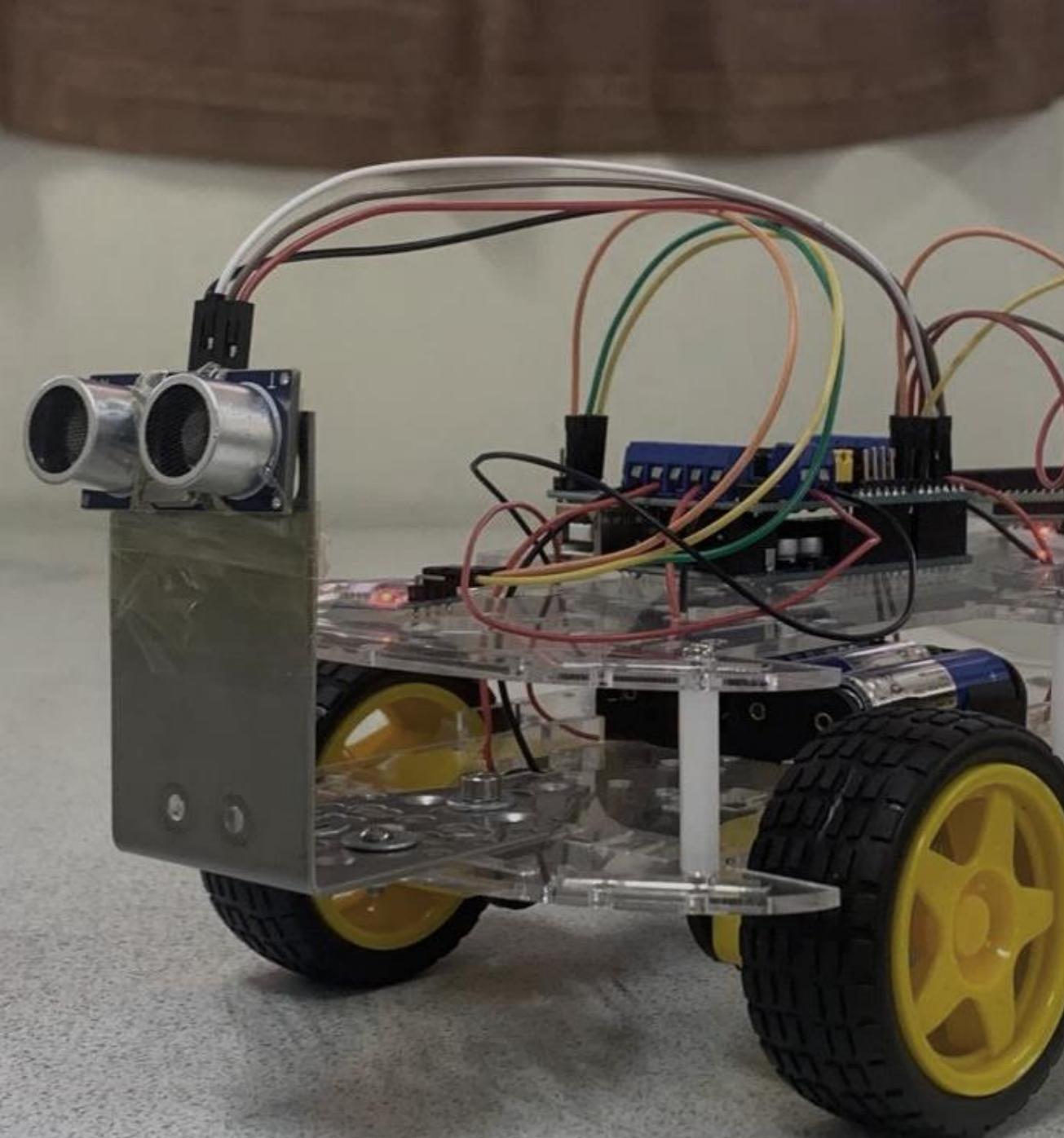


```
Message (Error to send message to Arduino Uno on COM5)

Shortest Path from A to D: [A, B, C, D]
Shortest Path from D to C: [D, C]
Shortest Path from C to F: [C, D, F]
Shortest Path from F to B: [F, D, C, B]
Shortest Path from B to C: [B, C]
Shortest Path from C to A: [C, B, A]
Shortest Path from A to D: [A, B, C, D]
Shortest Path from D to F: [D, F]
Shortest Path from F to E: [F, E]
Shortest Path from E to C: [E, D, C]
```

Conclusion :

- _ Le TADC va être conscient du plus court chemin qu'il doit traverser .
- _ Sa position se met à jour quand il atteint la destination .



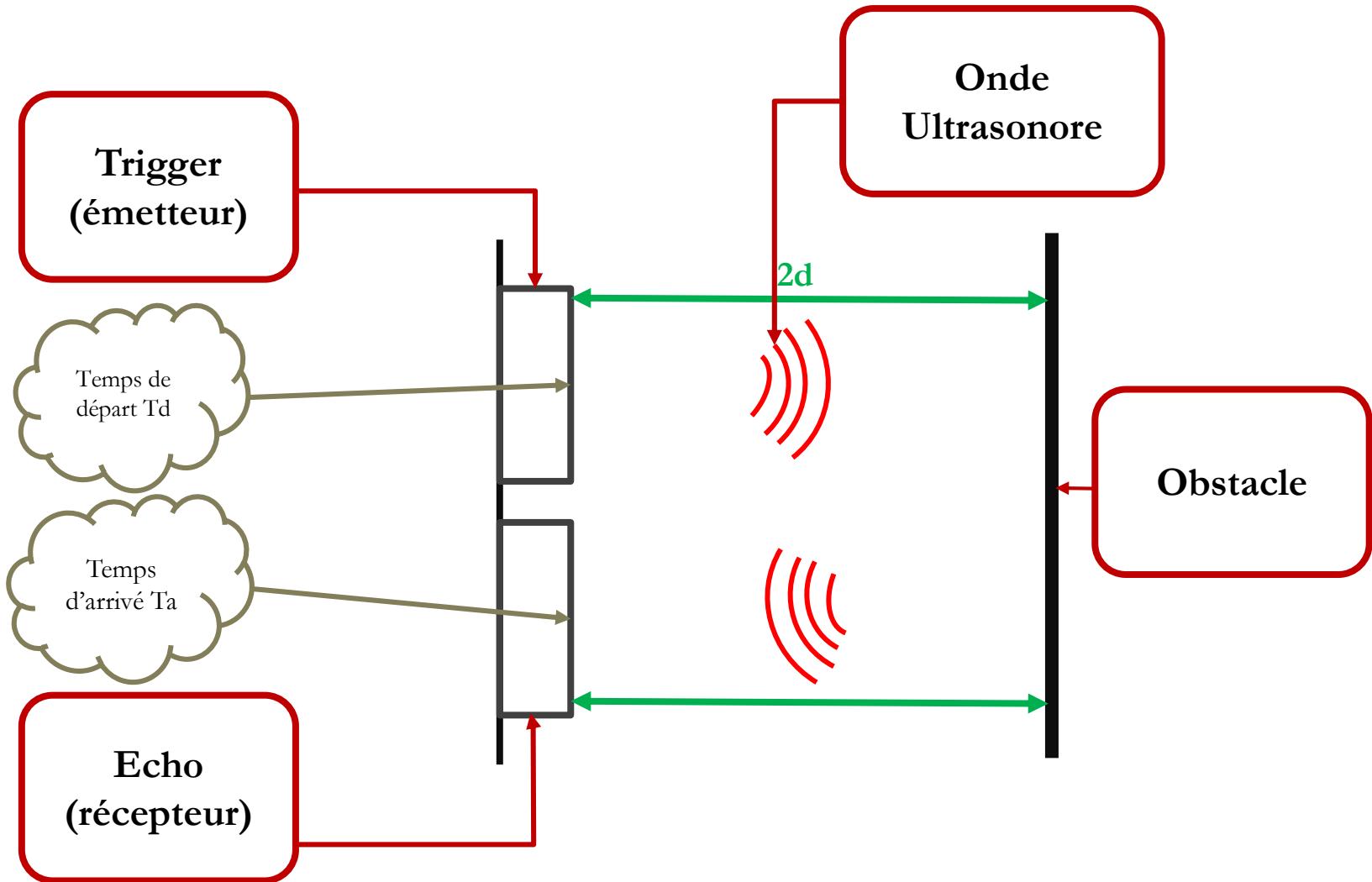
OBJECTIF 2 :

❖ Détection d'obstacle
et réaction de
TADC ?

- Etude du capteur Ultrason hc_sr04.
- Organigramme pour arrêter le TADC



Fonctionnement principal:



La vitesse d'onde Ultrasonore = 340 m/s
La distance parcourue par l'onde = $2d$
La durée $t = T_a - T_d$

$$V = 2d/t$$

$$D = V*t / 2$$

Etude des caractéristiques :

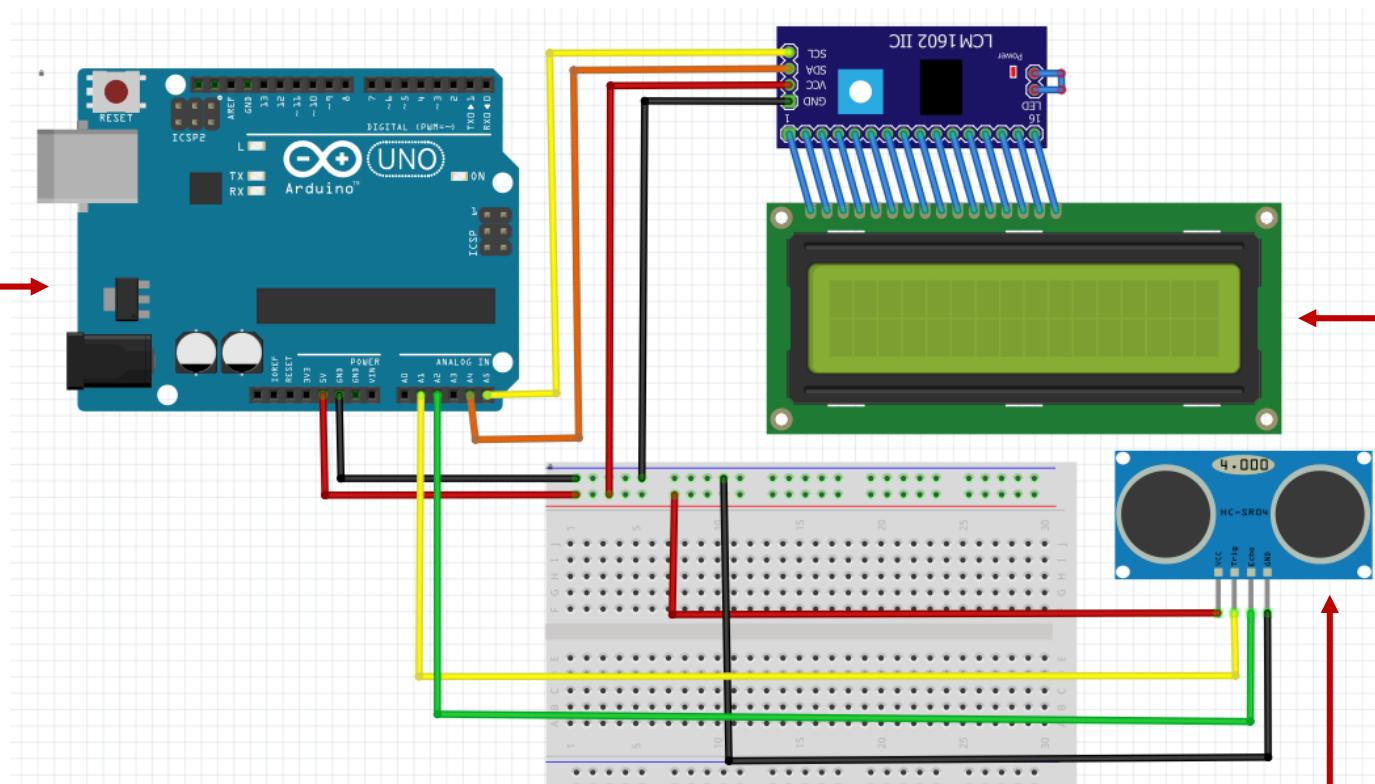
Simulation de l'expérience :

Carte Arduino

Communication
I2c

Afficheur LCD

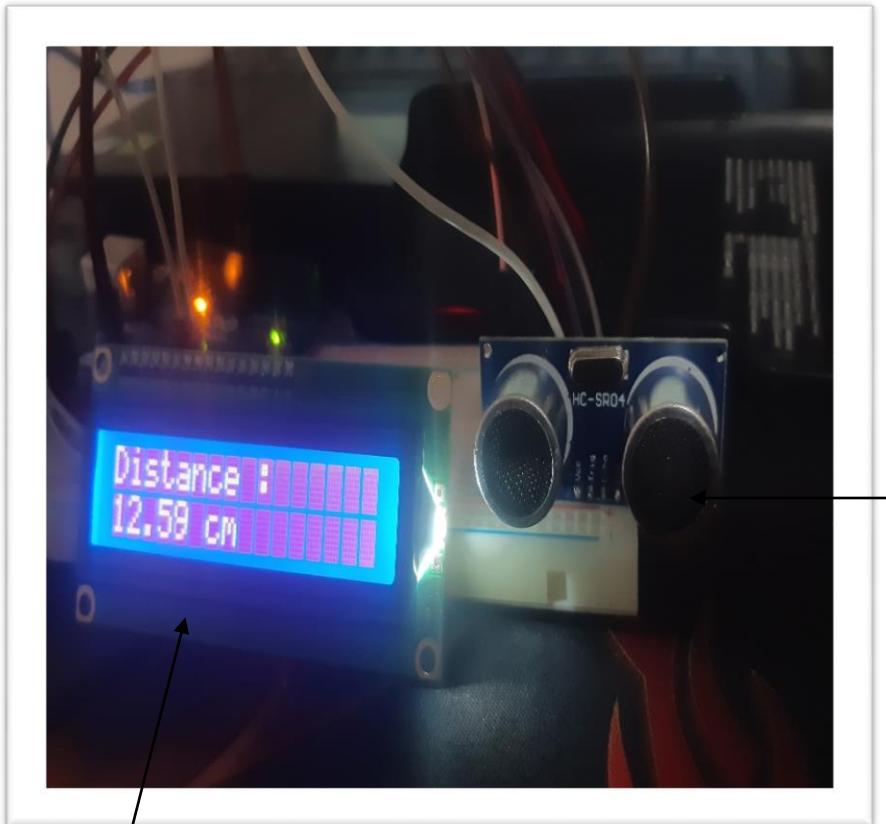
Capteur
Ultrason



fritzing

Etude des caractéristiques :

Expérience pour tracer la caractéristique $U(d)$:



Capteur
Ultrason

obstacle

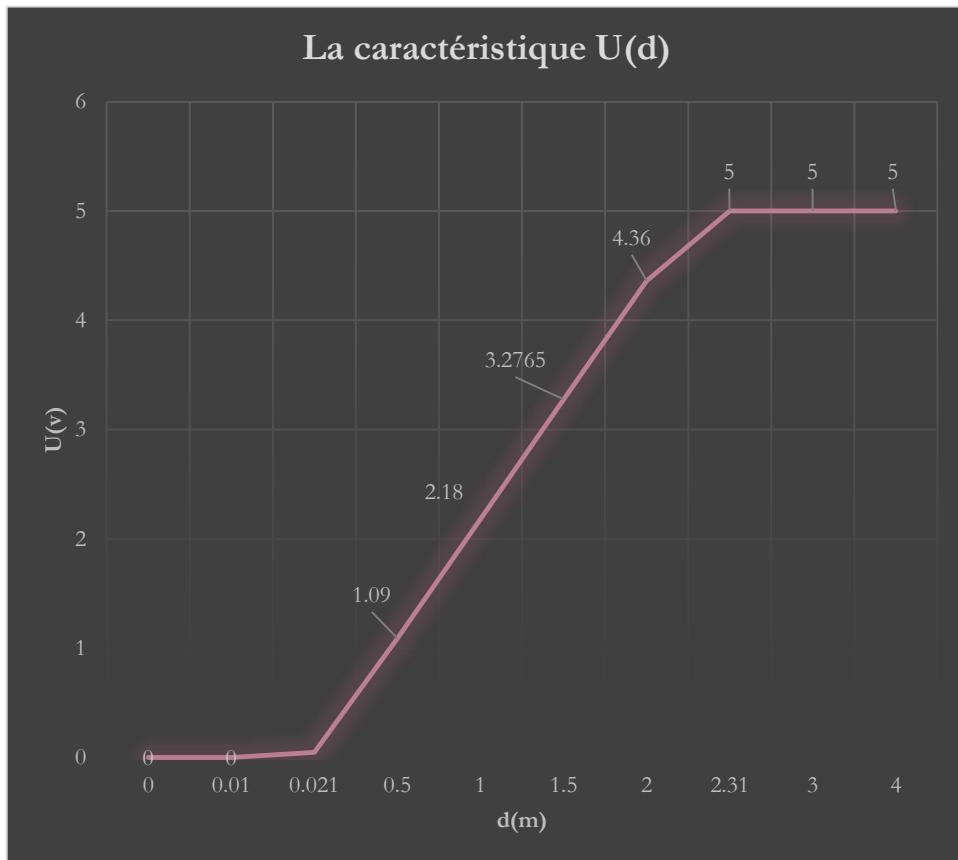
On trace $U(d)$ en connaissant la marge de mesure et en calculant le coefficient directeur de la fonction :



Afficheur LCD

Etude des caractéristiques :

Résultat d'expérience :



Sensibilité = 2,18 v/m

$D_{\min} = 0,021 \text{ m}$
 $D_{\max} = 2,31 \text{ m}$

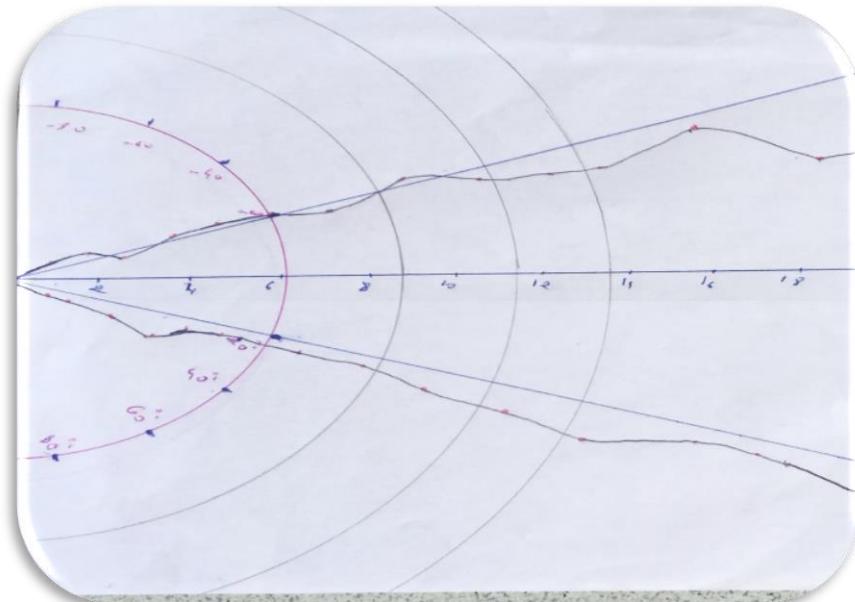
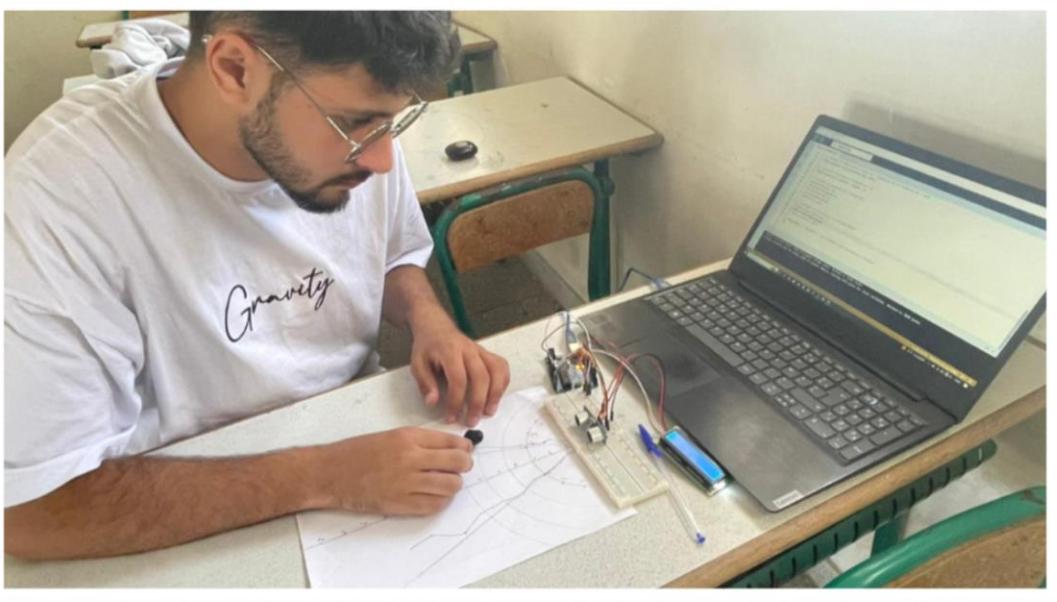
Étendue de mesure = $2,31 - 0,021 = 2,289 \text{ m}$

Temps de réponse = 500ms

Précision = $(\text{Valeur réel} - \text{Valeur mesurée}) / \text{Valeur réel} =$
 $(13-12,59)/13 *100 = 3,15 \%$

Etude des caractéristiques :

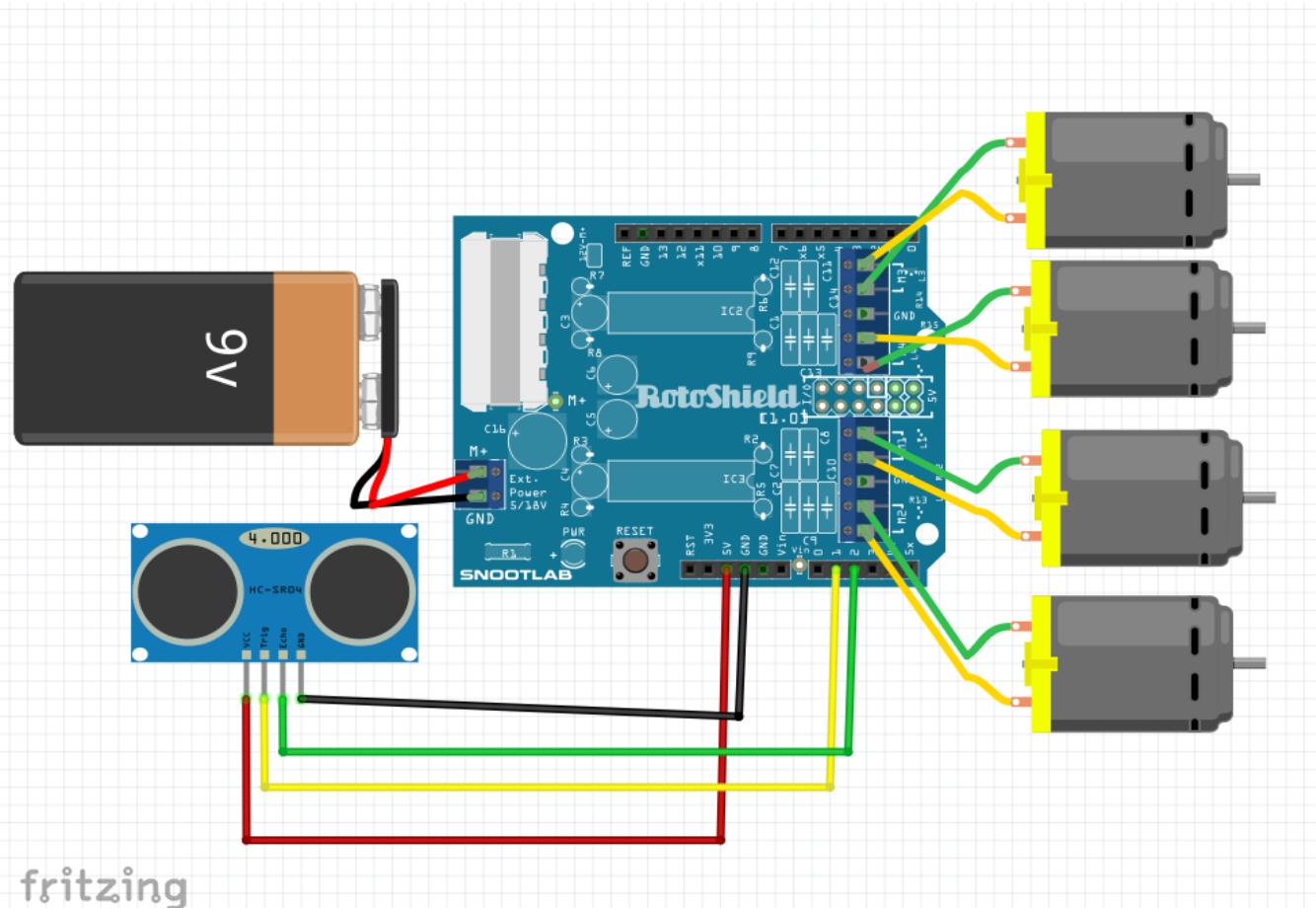
Le cône de détection :



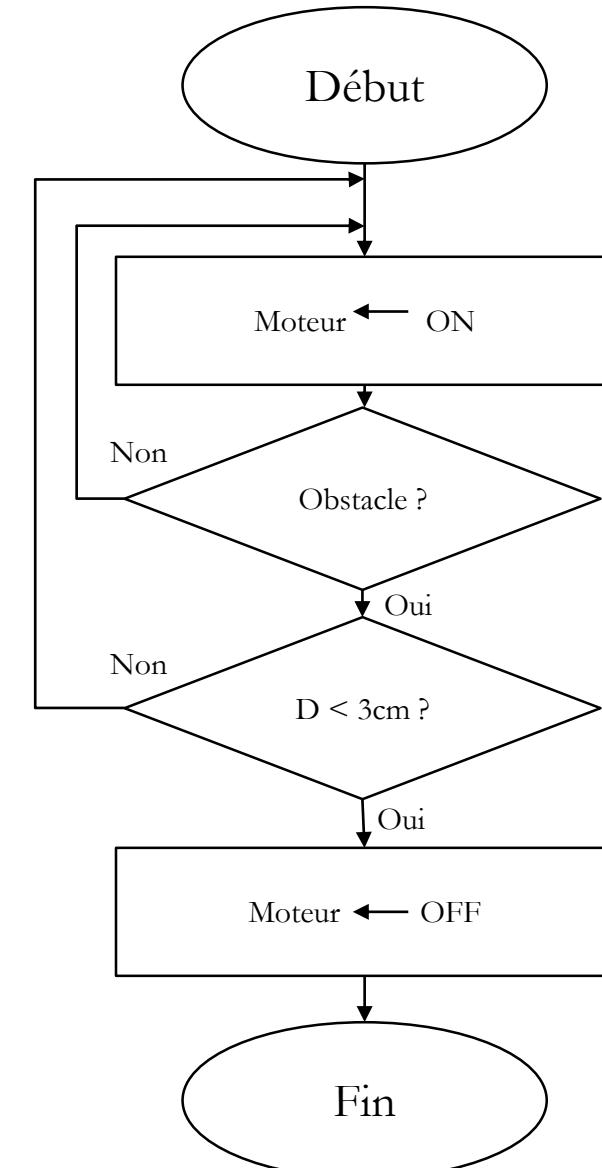
Réaction du TADC lors de la détection de l'obstacle :

Organigramme explicatif :

Simulation :



fritzing

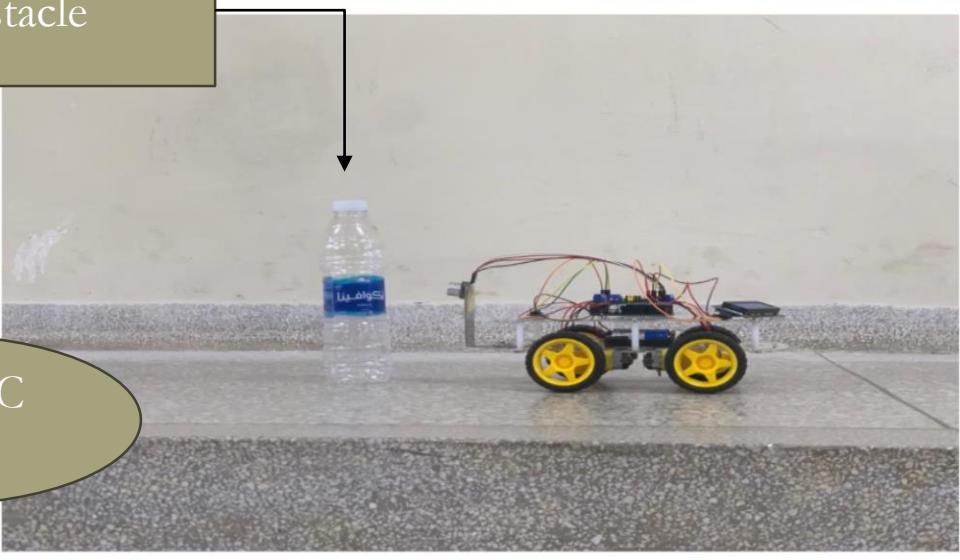


Réaction du TADC lors de la détection de l'obstacle :

Résultat en expérience:

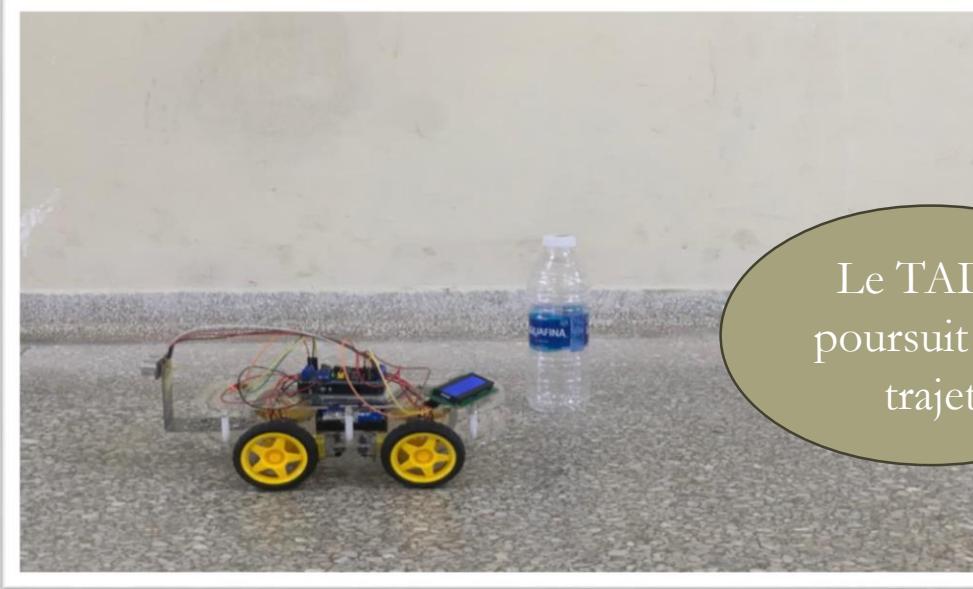
Obstacle

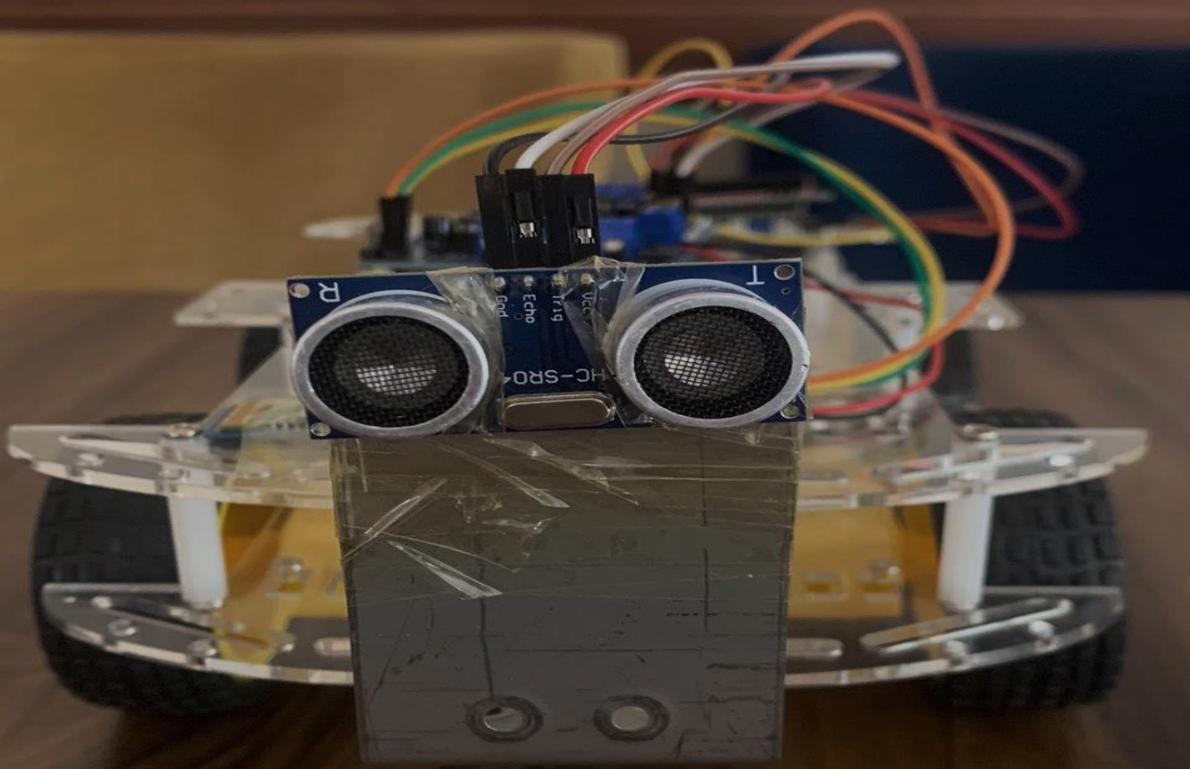
Le TADC s'arrête



Conclusion :

Le capteur Ultrason hc_sr04 répond bien aux exigences du cahier des charges





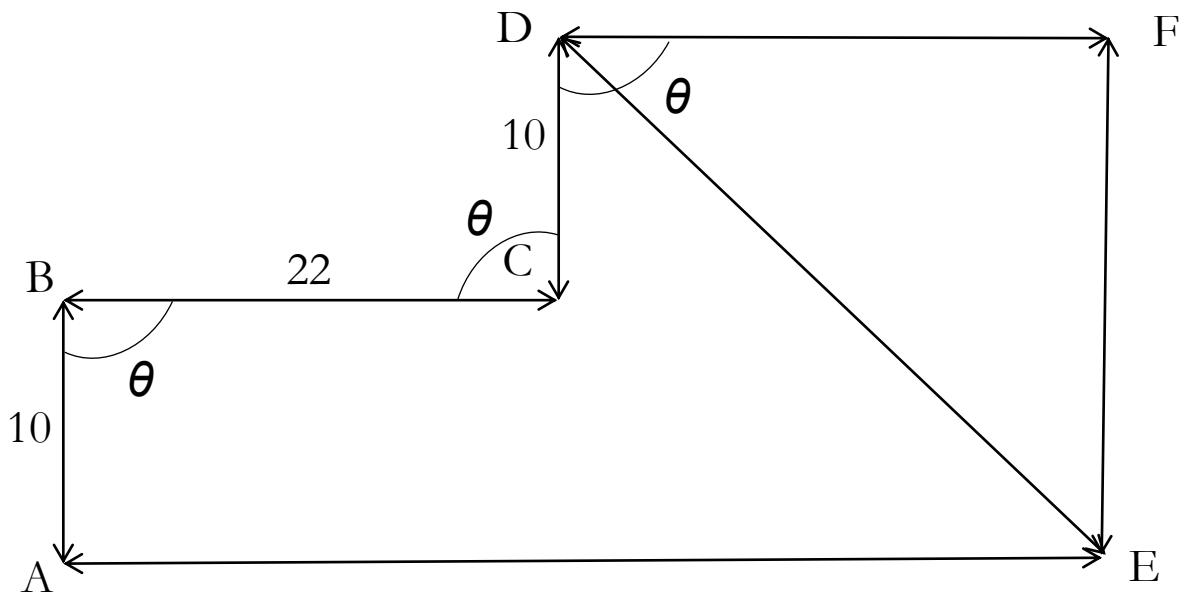
OBJECTIF 3 :

- ❖ Comment on peut réaliser une poursuite de graphe à partir des angles stockés ?
- Asservissement de la position du TADC .



Réaliser une poursuite de graphe :

On programme les positions angulaires qui dépend du délai (Objectif1).
Le but est d'asservir la position des roues à celle du graphe en fonction des délais.



Hypothèse :

_ On se limite sur une trajectoire définie de A vers F . Choisir une autre trajectoire exige une programmation des nouvelles positions angulaires .

Identification du système qui va subir un asservissement

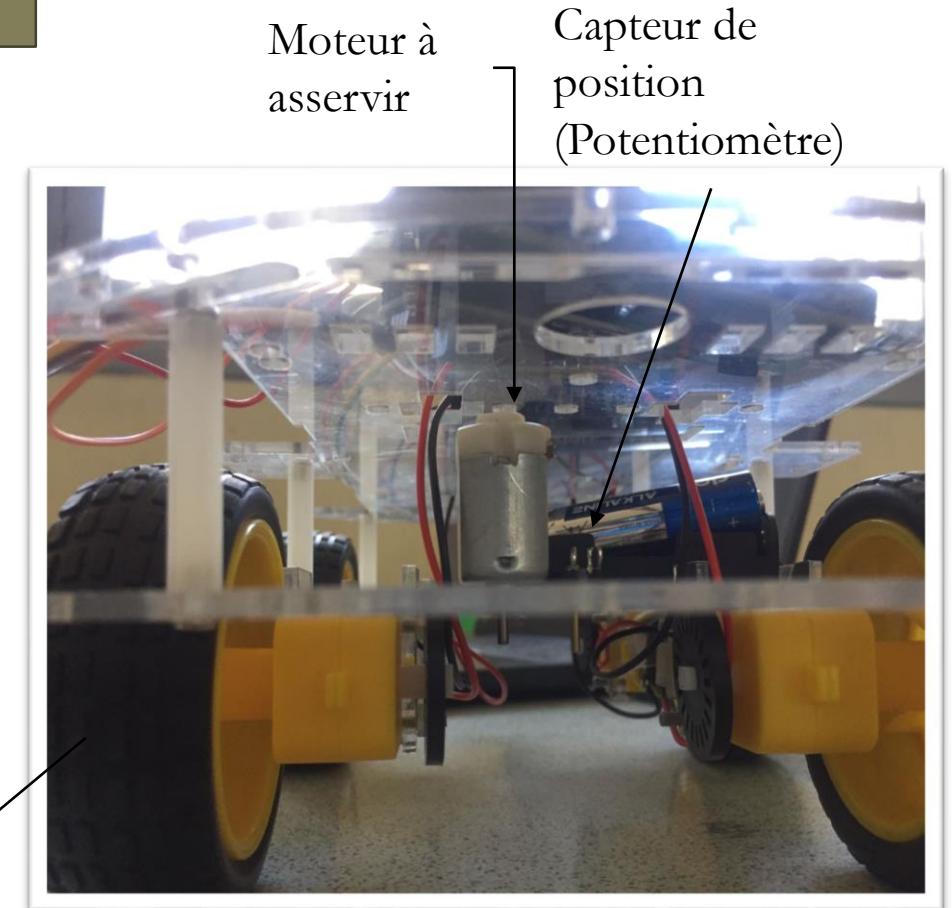
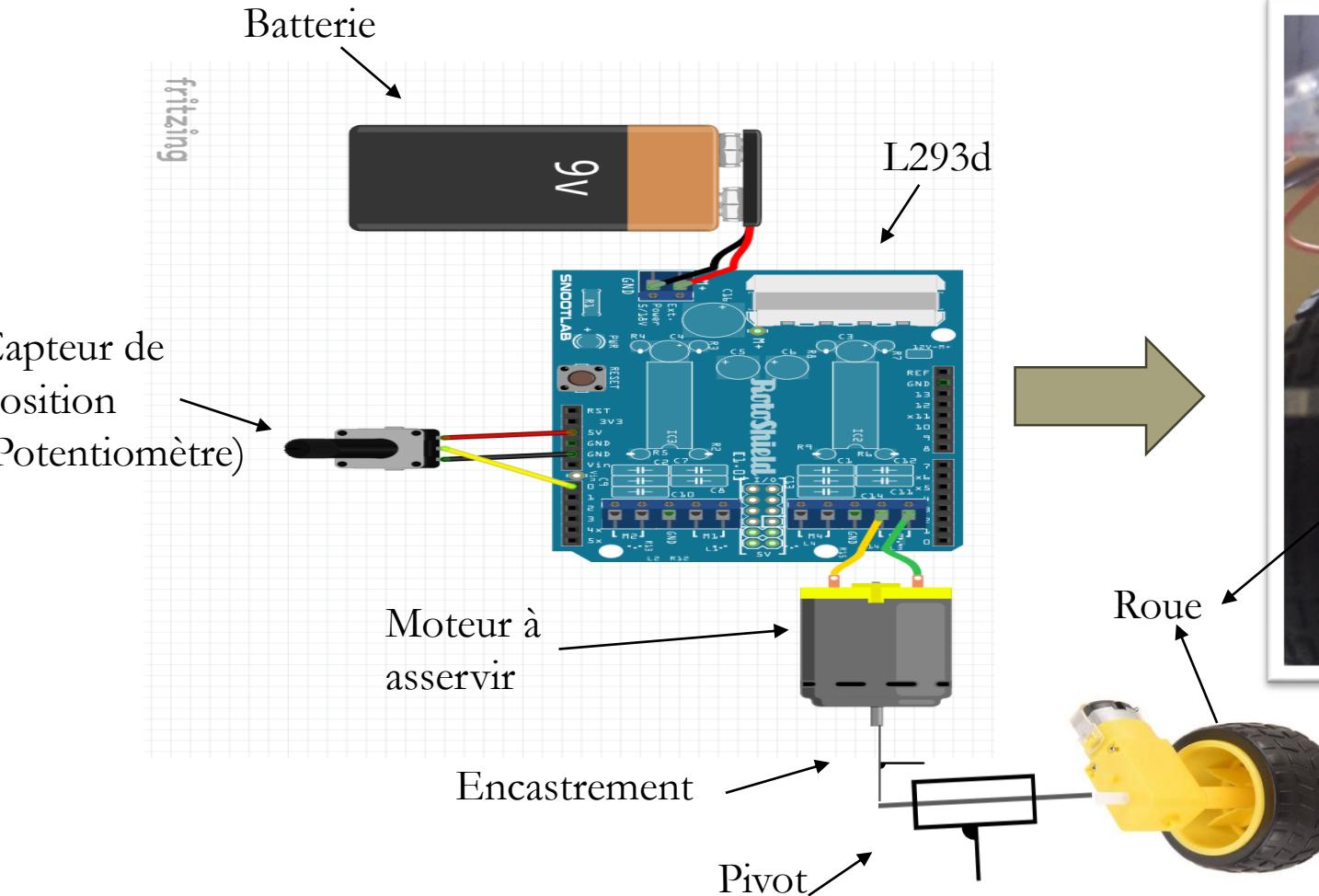
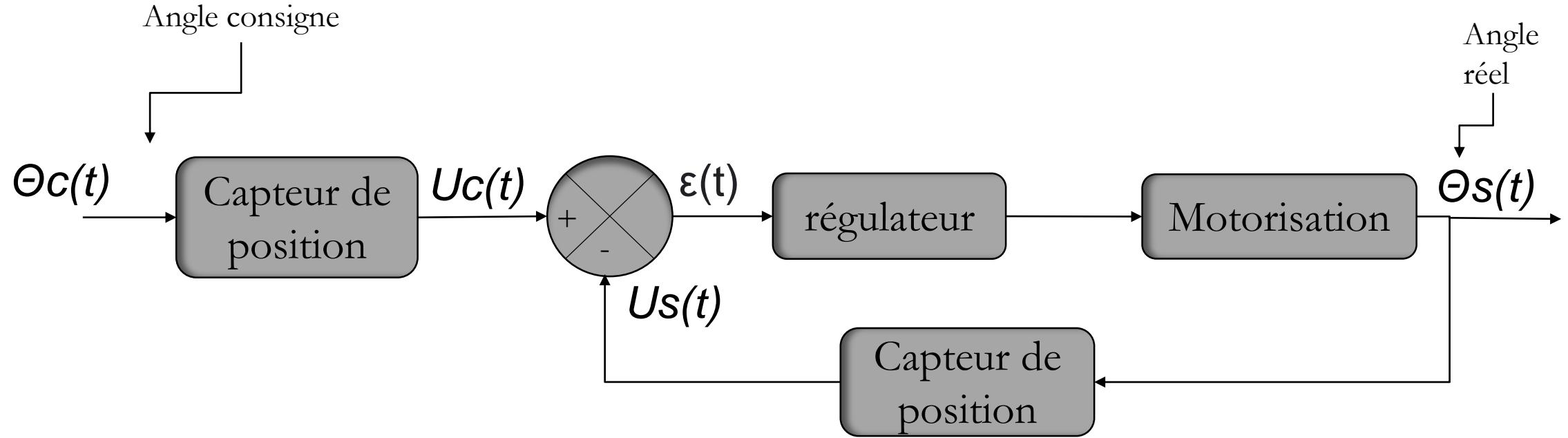
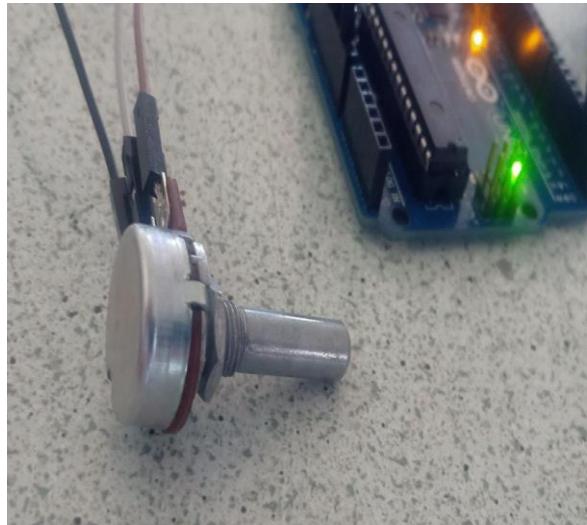
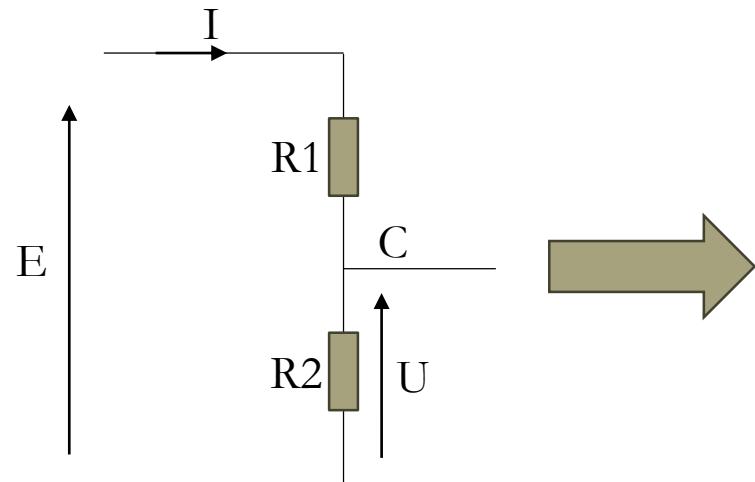


Schéma fonctionnel de l'asservissement de la position

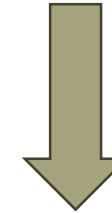


Choix du capteur de position : Potentiomètre linéaire

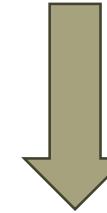
Modélisation du potentiomètre :



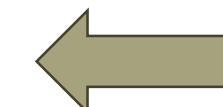
La résistance variable est proportionnelle à la position angulaire du curseur



$$U(R) = E/(R_1 + R_2) * R_2 \\ = \\ U(\theta) = E/(R_1 + R_2) * \theta$$



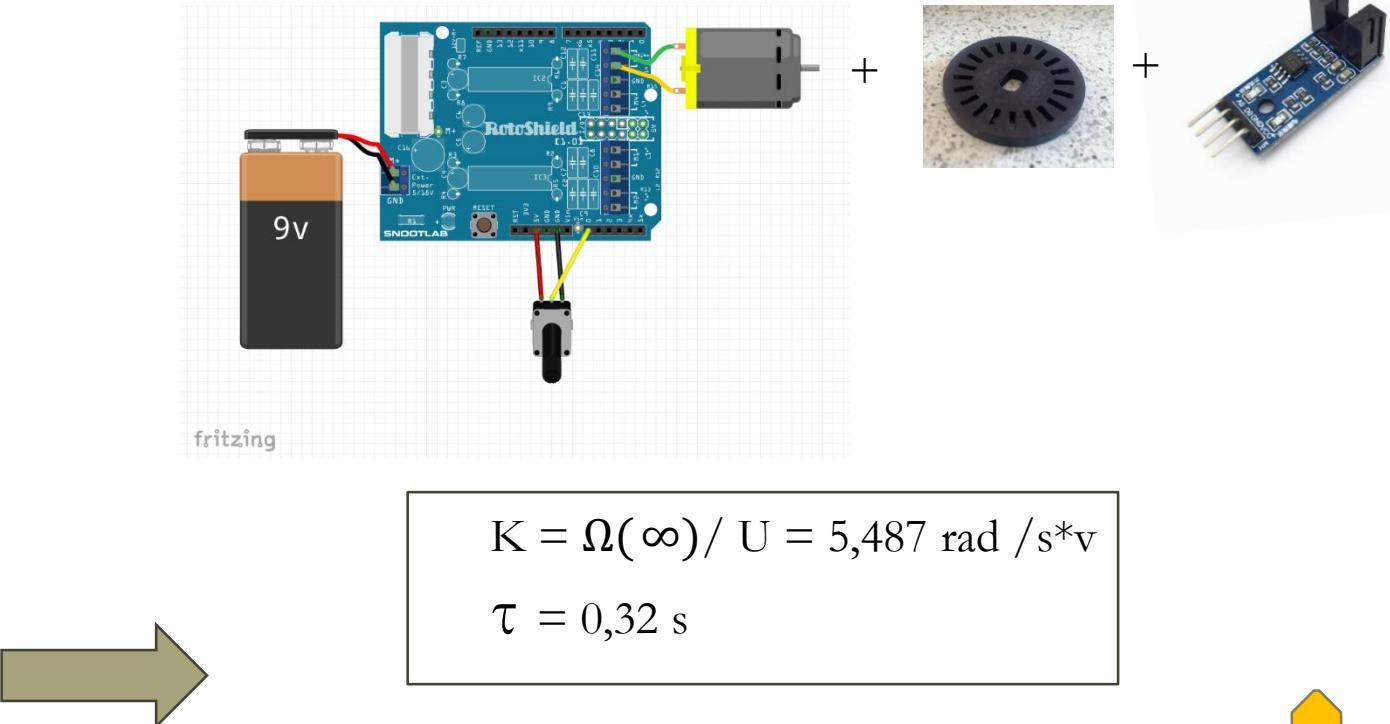
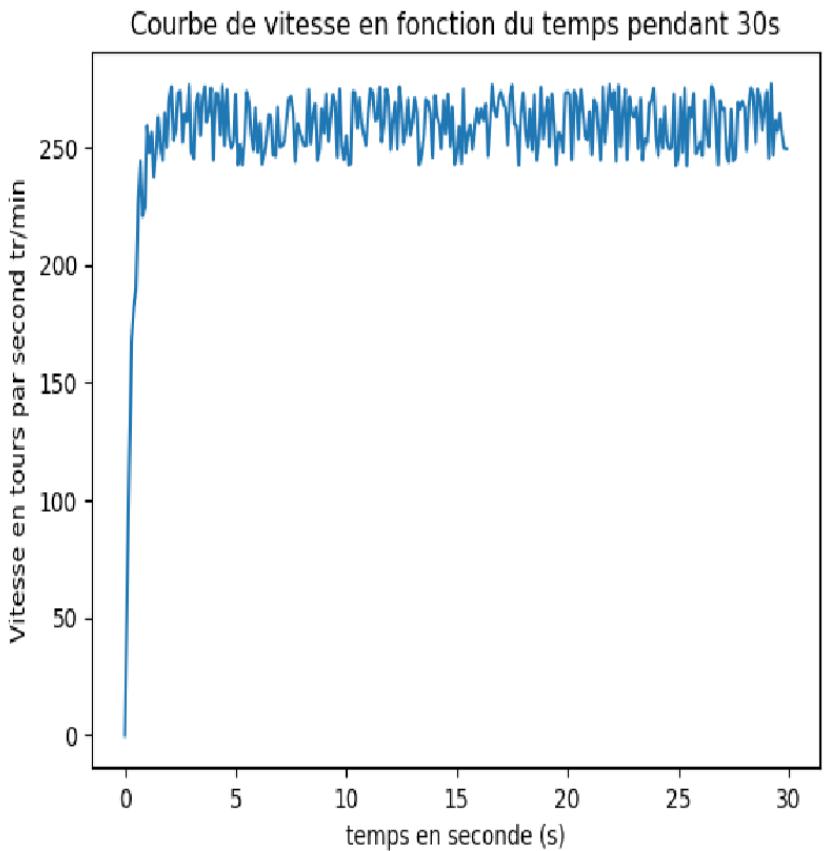
$$I = 5 \text{ V} / 10 \text{ kOhm} \\ = 0,5 \text{ kA}$$



$$I = E/(R_1 + R_2)$$

Identification des paramètres de la fonction du transfert :

On attaque le système par un échelon de 5 v et on mesure la vitesse chaque 0,1s à l'aide d'un codeur incrémental comme montre la simulation :



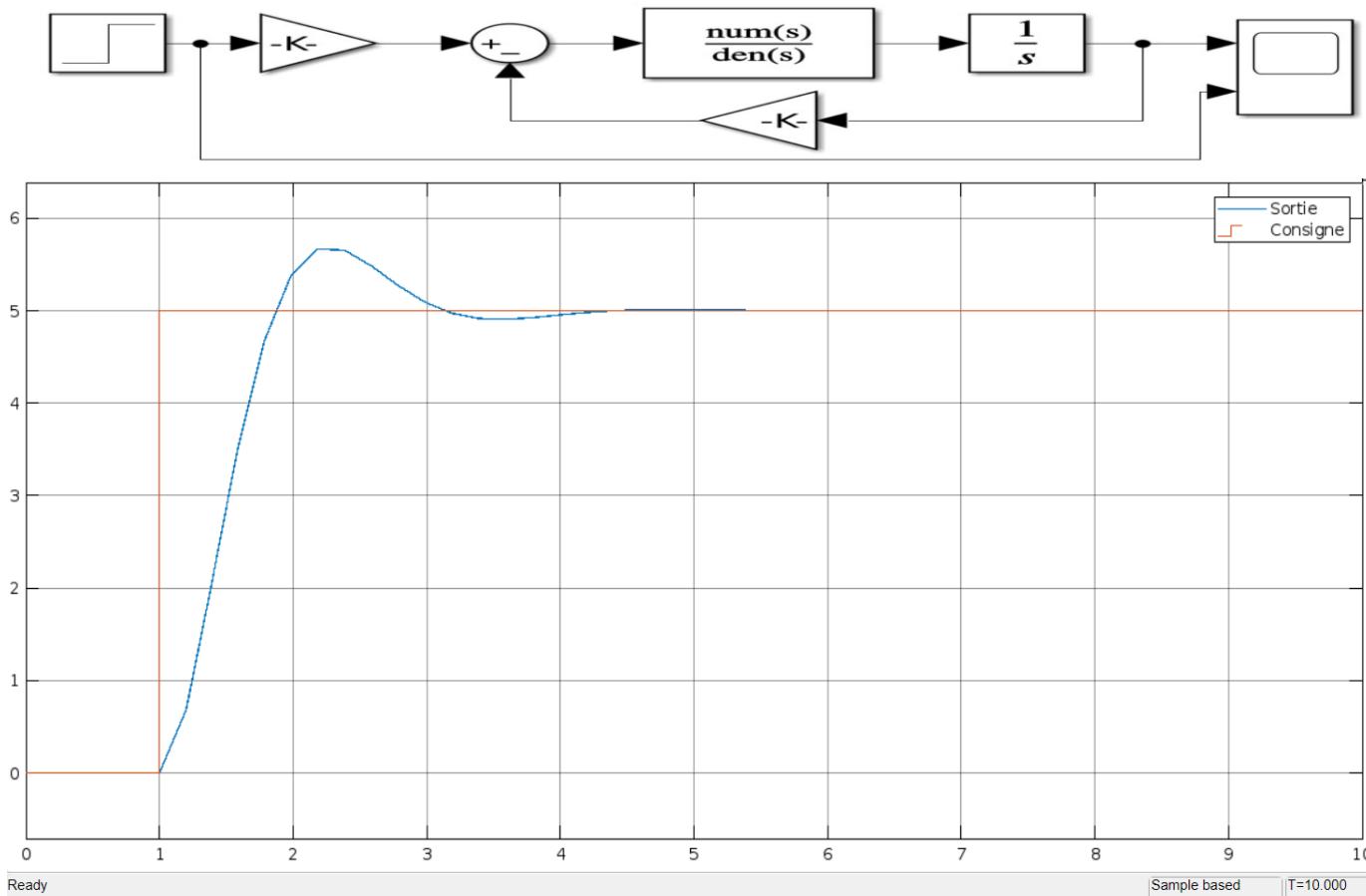
$$K = \Omega(\infty) / U = 5,487 \text{ rad / s} * \text{v}$$

$$\tau = 0,32 \text{ s}$$

$$H(p) = \frac{5,487}{1 + 0,32 * \tau}$$

Système à asservir sans correction :

Réponse indicielle de 5 v :



$$K = 0,5 \text{ kA}$$

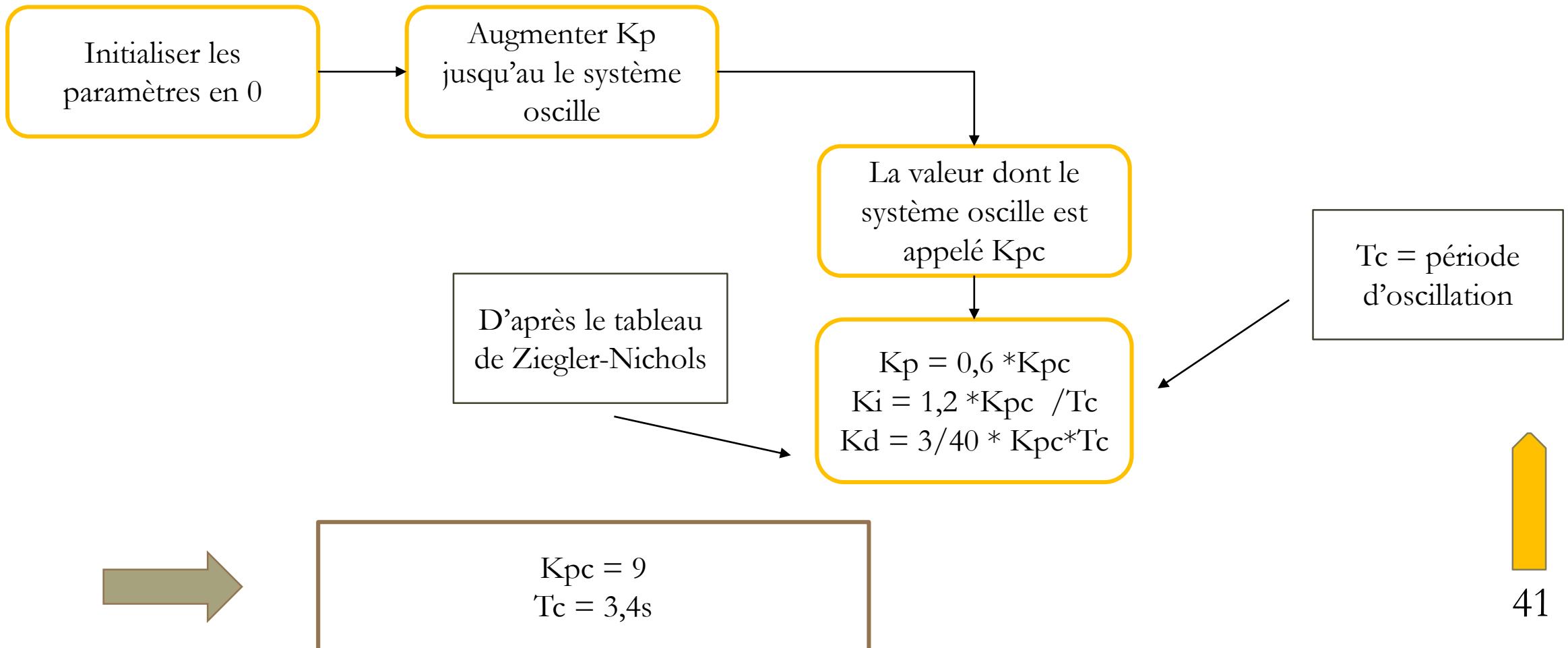
$$H(p) = \frac{5,487}{1 + 0,32 * \tau}$$

Erreur statique nulle
Dépassement important
Temps de réponse > 1s

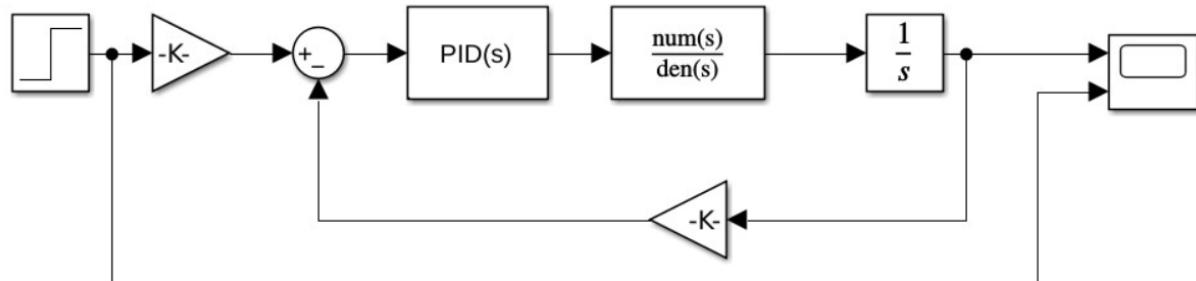
Solution : Choix du correcteur PID :

Système à asservir avec correction :

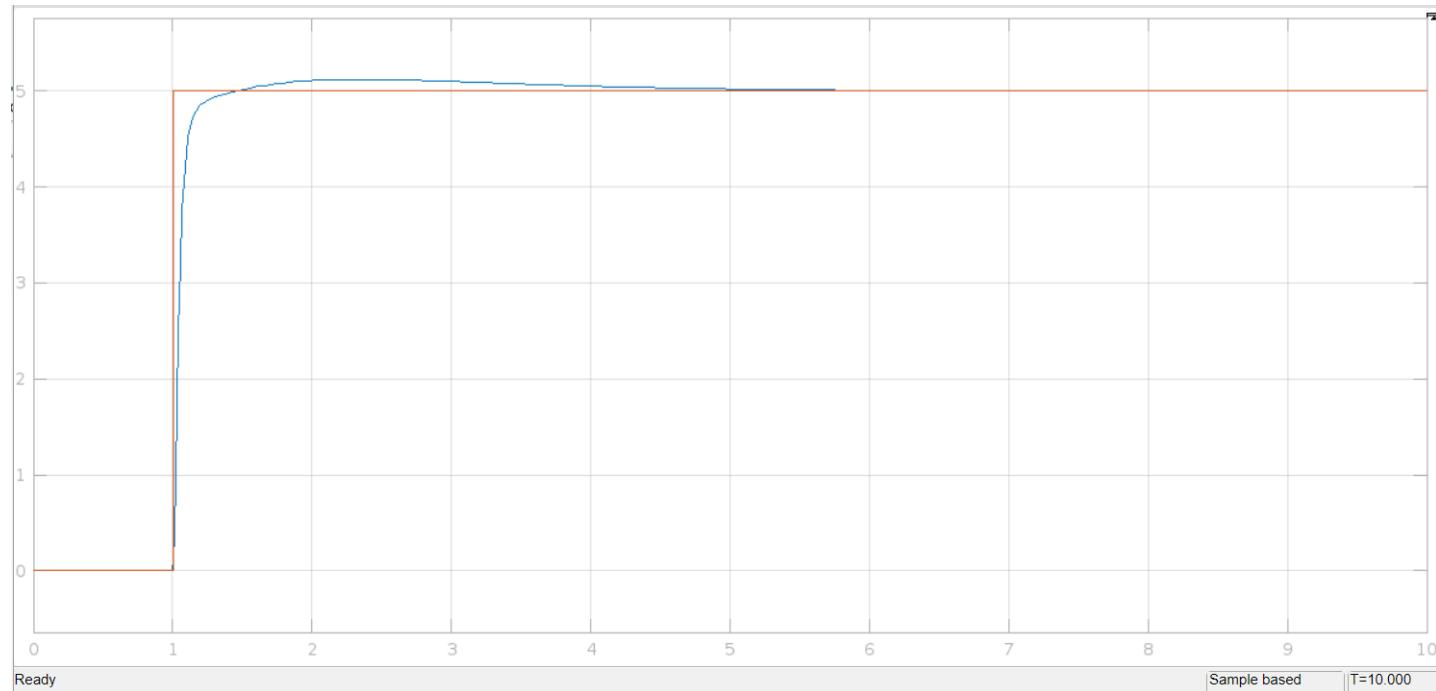
Déterminer les paramètres K_p , K_i , K_d par la méthode de réglage de Ziegler-Nichols :



Système à asservir avec correction :



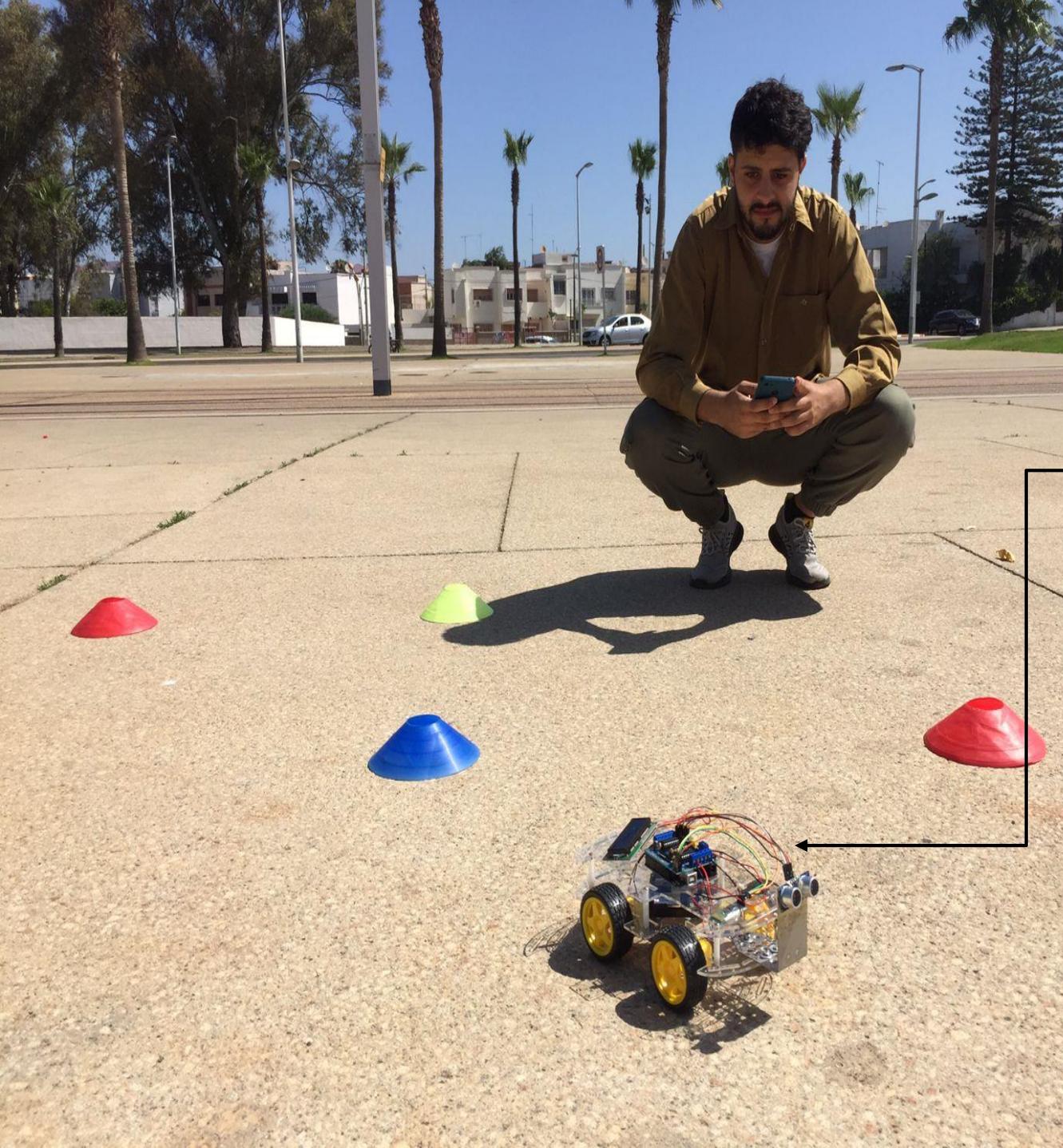
$$\begin{aligned} K_p &= 5,4 \\ K_i &= 3,17 \\ K_d &= 2,295 \end{aligned}$$



Dépassemement pratiquement négligeable
Temps de réponse <1s
Erreur statique nul

Le cahier des charges est
répondu





OBJECTIF 4 :

❖ Réaliser un prototype final



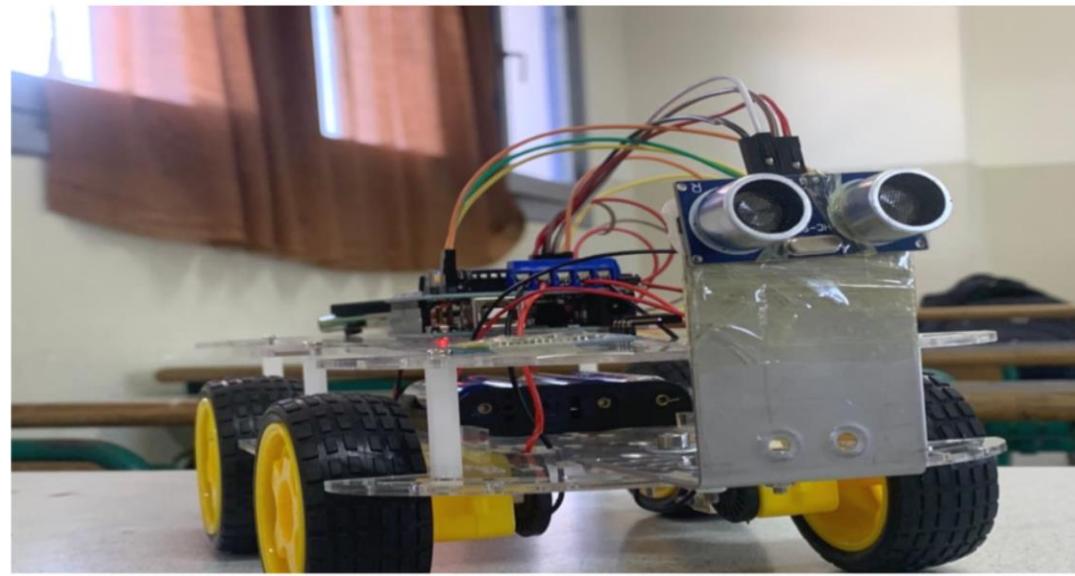
CONCLUSION :

Objectif 1 

Objectif 2 

Objectif 3 

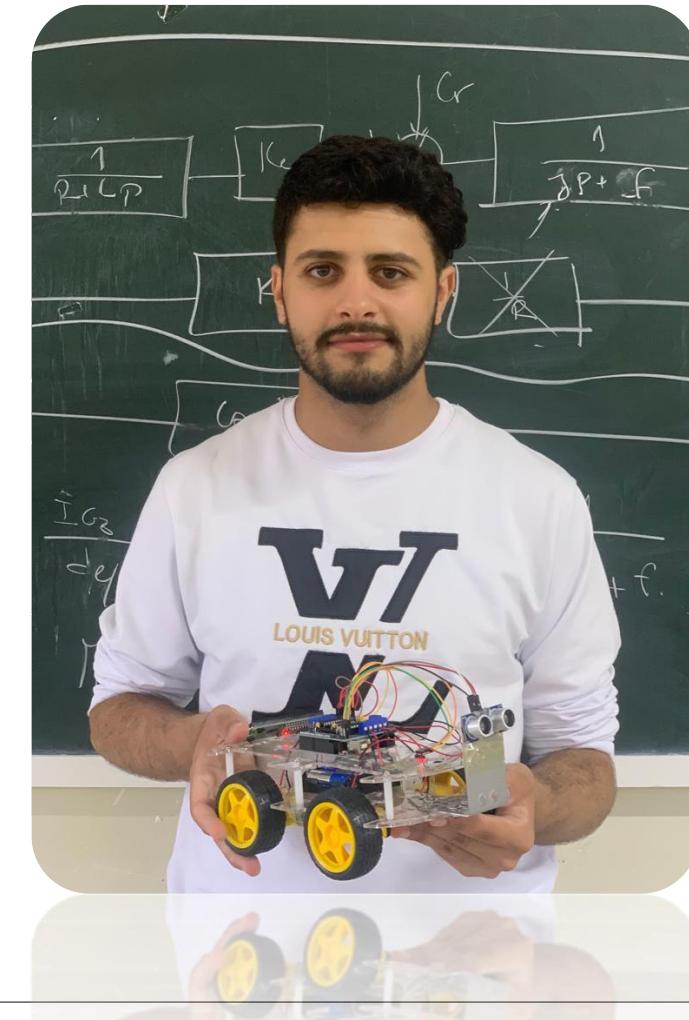
Objectif 4 



Le cahier des charges est
répondu



Merci pour
votre attention



ANNEXE :

Code Arduino :

Fonction Pluscourt(LD,LS) :

```
int pluscourt(int dist[], bool visited[]) {  
    int minDist = INF;  
    int minIndex = 0;  
  
    for (int v = 0; v < V; v++) {  
        if (!visited[v] && dist[v] < minDist) {  
            minDist = dist[v];  
            minIndex = v;  
        }  
    }  
    return minIndex;  
}
```

Dijkstra :

```
void dijkstra(int Adj[][V], int src, int dest, int prev[]) {  
    bool visited[V];  
    int dist[V];  
  
    for (int i = 0; i < V; i++) {  
        dist[i] = INF;  
        prev[i] = -1;  
        visited[i] = false;  
    }  
  
    dist[src] = 0;  
  
    for (int count = 0; count < V - 1; count++) {  
        int u = pluscourt(dist, visited);  
        visited[u] = true;  
  
        for (int v = 0; v < V; v++) {  
            if (!visited[v] && Adj[u][v] != INF && dist[u] + Adj[u][v] <  
                dist[v]) {  
                dist[v] = dist[u] + Adj[u][v];  
                prev[v] = u;  
            }  
        }  
    }  
}
```

ANNEXE :

Code Arduino :

Trouver le plus court chemin :

```
void getShortestPath(int prev[], int dest, int shortestPath[][2], int& pathLength, int Adj[][V]) {  
  
    if (prev[dest] != -1) {  
        getShortestPath(prev, prev[dest], shortestPath, pathLength, Adj);  
    }  
  
    shortestPath[pathLength][0] = prev[dest];  
    shortestPath[pathLength][1] = dest;  
    pathLength++;  
  
}
```



ANNEXE :

Code Arduino :

Réaction du TADC si un obstacle est détecté :

```
#include <AFMotor.h>

AF_DCMotor motor(1);
const int trigPin = 2;
const int echoPin = 3;

void setup() {
    Serial.begin(9600);
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);

    motor.setSpeed(255);
}
```

```
void loop() {
    long duration, distance;

    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    duration = pulseIn(echoPin, HIGH);

    distance = duration / 29 / 2;

    Serial.print("Distance : ");
    Serial.print(distance);
    Serial.println(" cm");

    if (distance < 3) {
        motor.run(RELEASE);
    } else {
        motor.run(FORWARD);
    }
}
```

ANNEXE :

Tableau de Ziegler-Nichols :

Méthode de Ziegler-Nichols ¹					
Type de contrôle	Kp	Ti	Td	Ki	Kd
P	0.5Kpc	-	-	-	-
PI	0.45Kpc	Tc/1.2	-	0.54Kpc/Tc	-
PD	0.8Kpc	-	Tc/8	-	Kpc Tc/10
PID ²	0.6Kpc	Tc/2	Tc/8	1.2Kpc/Tc	3Kpc Tc/40
PIR (<i>Pessen Integral Rule</i>) ²	0.7Kpc	Tc/2.5	3Tc/20	1.75Kpc/Tc	21Kpc Tc/200
léger dépassement ²	0.33Kpc	Tc/2	Tc/3	0.666Kpc/Tc	Kpc Tc/9
aucun dépassement ²	0.2Kpc	Tc/2	Tc/3	(2/5)Kpc/Tc	Kpc Tc/15

ANNEXE :

Code Arduino :

Poursuivre le trajectoire A,B,C,D,F :

```
#include <Servo.h>

const int servoPin = 9;
Servo servoMotor;
int angles[] = {0,90, -90, 90};
unsigned long delays[] = {10000, 22000,
10000,20000};

const int numSteps = sizeof(angles) /
sizeof(angles[0]);

void setup() {
  servoMotor.attach(servoPin);
}

void loop() {
  for (int i = 0; i < numSteps; i++) {
    servoMotor.write(angles[i]);
    delay(delays[i]);
  }
}
```

