🧠 1. Le problème que nous avons résolu

On veut construire un **contrôleur flou** pour un **véhicule autonome**.

Ce contrôleur décide automatiquement la **vitesse de sécurité** du véhicule à partir de deux informations provenant de la vision artificielle :

Entrée	Signification
Distance	Distance entre le véhicule et l'objet détecté
Visibilité	Qualité de vision (météo, brouillard, pluie…)

La sortie est :

Sortie	Signification
Vitesse	Vitesse recommandée pour conduire en sécurité

Ce type de problème est parfait pour la **logique floue**, car la décision n'est pas binaire (stop / go), mais progressive et proche du raisonnement humain.

🧩 2. Modélisation floue

Nous avons transformé ces grandeurs réelles en **ensembles flous**.

🧱 Distance (0 à 100 m)

- **Proche**
- **Moyenne**
- **Loin**

Chaque ensemble est représenté par des fonctions d'appartenance **triangulaires** et **trapézoïdales**.

🌫️ Visibilité (0 à 100 %)

- **Mauvaise**
- **Moyenne**
- **Bonne**

🚗 Vitesse (0 à 120 km/h)

- **FreinageUrgent**
- **Lent**
- **Normal**
- **Rapide**

Ces ensembles permettent de représenter des notions humaines : *« assez proche »*, *« visibilité mauvaise »*, *« vitesse lente »*, etc.

🧾 3. Règles de décision (Cerveau du système)

Ensuite, nous avons écrit des règles floues de type :

> **SI** Distance est Proche **ET** Visibilité est Mauvaise
> **ALORS** Vitesse est FreinageUrgent

Ces règles reproduisent le comportement logique d'un conducteur humain.

Nous avons organisé ces règles dans une **matrice d'inférence** cohérente, qui garantit la sécurité.

🏗️ 4. Implémentation sous Matlab

Nous avons organisé le projet comme un vrai projet d'ingénierie :

Fichier	Rôle

`create_fis.m`	Construit le contrôleur flou
`simulate.m`	Teste des situations
`visualize.m`	Affiche les surfaces
`main.m`	Programme principal

Matlab crée un **FIS (Fuzzy Inference System)** qui contient :

- les entrées
- les ensembles flous
- les règles
- la méthode d'inférence

🌄 5. Visualisation

Avec `surfview`, on obtient une surface 3D :

Distance + Visibilité → Vitesse

Cette surface montre que :

- plus l'objet est proche → plus la vitesse baisse
- plus la visibilité est mauvaise → plus la vitesse baisse

Le comportement est **continu, stable et sécurisé**.

🧪 6. Simulation

Exemple réel :

```
Distance = 15 m
Visibilité = 40 %
```

Résultat :

```
Vitesse ≈ 35 km/h
```

Ce résultat est logique :

- objet assez proche
- visibilité faible
  → on ralentit fortement.

🧑‍💻 Explication du projet Matlab

📂1️⃣ Fichier : `main.m`

```
clear; clc;
```

🔹 Efface l'ancienne mémoire et nettoie l'écran.

```
fis = create_fis();
```

🔹 Appelle la fonction qui **construit tout le système flou**
et stocke le résultat dans la variable `fis`.

```
visualize(fis);
```

🔹 Affiche la **surface de décision 3D** du contrôleur.

```
v = simulate(fis,15,40);
```

🔹 Teste une situation réelle :
Distance = 15 m, Visibilité = 40 %
et calcule la vitesse correspondante.

🧩2️⃣ Fichier : `create_fis.m`

```
fis = mamfis('Name','VitesseSecurite');
```

- Crée un système flou de type **Mamdani**.

➕ Création de l'entrée Distance

```
fis = addInput(fis,[0 100],'Name','Distance');
```

- Définit l'univers de discours : Distance entre 0 et 100 m.

```
fis = addMF(fis,'Distance','trapmf',[0 0 10 25],'Name','Proche');
```

- Définit l'ensemble flou **Proche** avec une fonction trapézoïdale.

```
fis = addMF(fis,'Distance','trimf',[15 40 65],'Name','Moyenne');
```

- Définit **Moyenne** avec un triangle.

```
fis = addMF(fis,'Distance','trapmf',[50 70 100 100],'Name','Loin');
```

- Définit **Loin**.

➕ Création de l'entrée Visibilité

Même principe : univers [0–100], ensembles
Mauvaise, **Moyenne**, **Bonne**.

➕ Création de la sortie Vitesse

Univers : [0–120 km/h]
Ensembles :

- FreinageUrgent
- Lent

- Normal
- Rapide

Chaque ensemble correspond à une **zone de vitesse**.

🗒️ Ajout des règles

```
rules = [
1 1 1 1 1;
...
3 3 4 1 1
];
```

Chaque ligne est une règle :

Colonne	Signification
1	Ensemble de Distance
2	Ensemble de Visibilité
3	Ensemble de Vitesse
4	Poids de la règle
5	AND (1) / OR (2)

Exemple :

```
1 1 1 1 1
```

→ SI Distance = Proche
ET Visibilité = Mauvaise
ALORS Vitesse = FreinageUrgent

```
fis = addRule(fis,rules);
```

- 🔹 Enregistre les règles dans le système.

🧪3️⃣ Fichier : `simulate.m`

```
vitesse = evalfis(fis,[distance visibilite]);
```

🔹 Fait passer les entrées dans le contrôleur flou
et calcule la sortie.

```
fprintf(...)
```

🔹 Affiche le résultat proprement.

🌄4️⃣ Fichier : `visualize.m`

```
surfview(fis);
```

🔹 Affiche la surface de décision 3D.

🧠 Ce que tu dois retenir

Le programme :

1. crée un modèle flou
2. définit ses règles
3. raisonne comme un humain
4. produit une décision de vitesse sûre
