ÉTUDE MÉCANIQUE D'UNE VOITURE TÉLÉCOMMANDÉE

LEFEBVRE Tom - N°34917

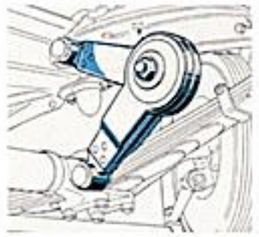
Membre du groupe : FOUQUERAY Nicolas - N°57762

Cycles & Boucles

SOMMAIRE

1. PRÉSENTATION DU SUJET

1.1. NATURE DES AMORTISSEMENTS

Amortisseurs à frictions (par frottements mécanique)	Amortisseurs hydrauliques (comme sur la voiture étudiée)
➔ peu précis et non linéaire (pas proportionnel à la vitesse) ➔ plus employé dans l'automobile 	a) Hydraulique simple (huile seule) ➔ amortissement proportionnel à la vitesse théoriquement (à vérifier par les expériences) ➔ réglage possible à l'origine : viscosité du liquide b) Hydraulique à émulsion (huile+air) ➔ réponse plus progressive théoriquement ➔ réponse est différente à chaud c) Hydraulique à membrane (huile+air séparés) ➔ amortissement plus constant 

1.2. DIFFÉRENTS TYPES DE PNEUS

Pneus “sillon”

Pneus “slick”

1.3. CIRCUIT ÉTUDIÉ

Légende

- Béton
- Terre
- Sable

2. PROBLÉMATIQUE

Comment configurer la voiture télécommandée pour gagner une course ?

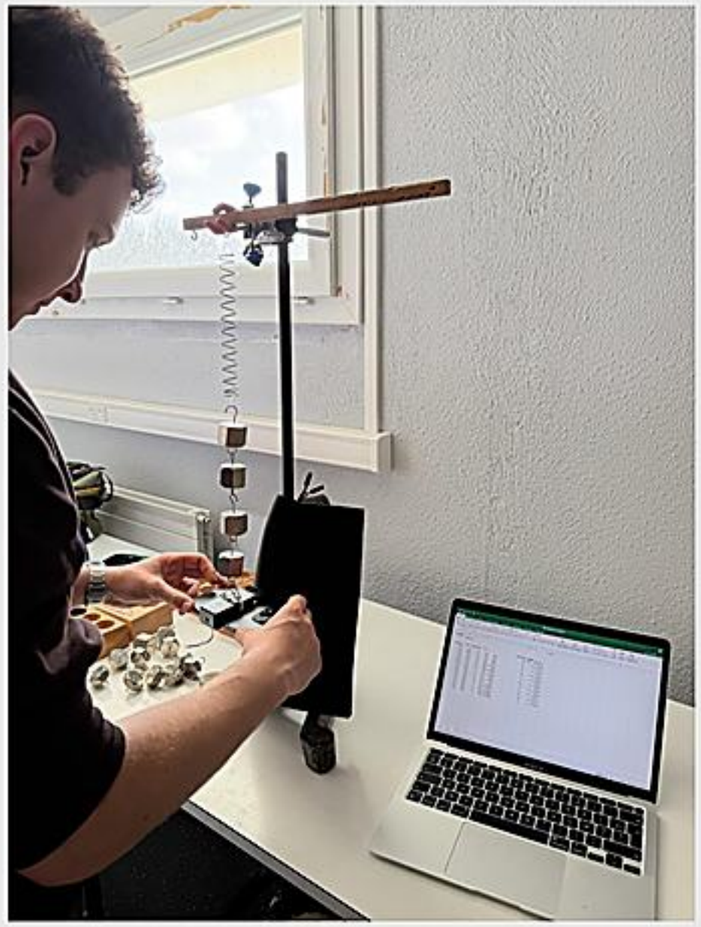
3. OBJECTIF DU TIPE

<u>Optimisation d'une voiture télécommandée :</u>

- Compréhension de l'amortissement
- Type de pneus optimal
- Compréhension de la cinétique
- Modéliser le buggy

4.1. EXPÉRIENCE 1

Détermination de la constante de raideur du ressort présent dans l'amortisseur de la voiture, de manière dynamique et statique (Loi de Hooke)



4.1.1 DETERMINATION DE LA CONSTANTE DE RAIDEUR

En Statique:

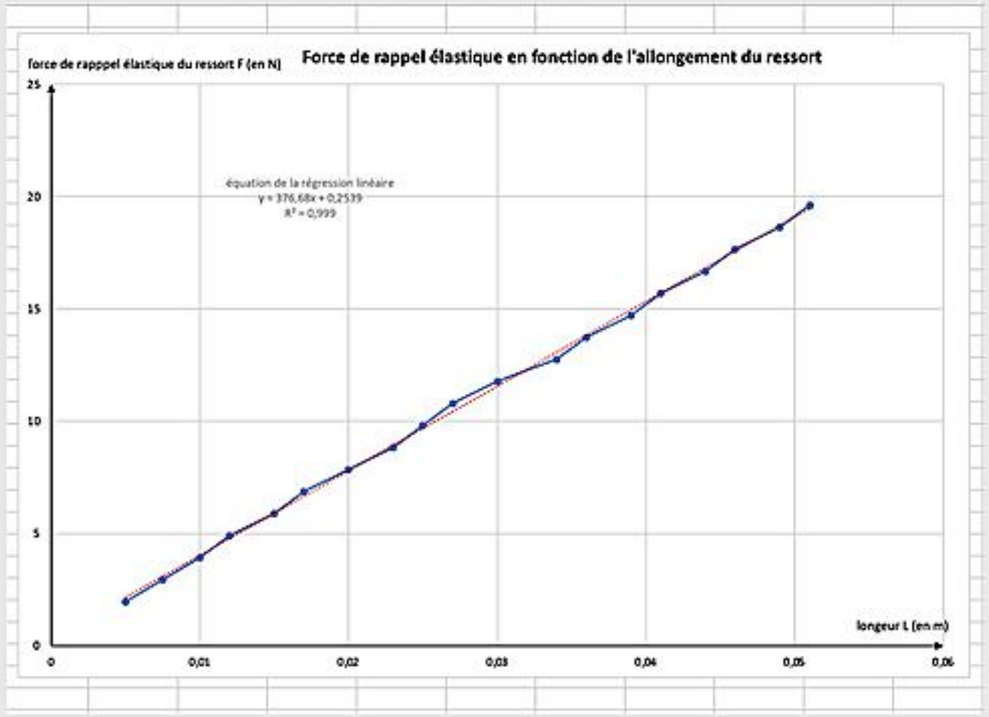
- Loi de Hooke :

$$F = -k\Delta l$$

- Constante obtenue : **k=388±2N/m**

Loi de Hooke			K	Incertitude
F = -kL		Moyen	388,828344	2,07802008
		Pente	376,678	2,88

Longeur (m)	Masse (kg)	F=mg	Delta L	K
0,106	0	0	0	
0,111	0,2	1,962	0,005	392,4
0,1135	0,3	2,943	0,0075	392,4
0,116	0,4	3,924	0,01	392,4
0,118	0,5	4,905	0,012	408,75
0,121	0,6	5,886	0,015	392,4
0,123	0,7	6,867	0,017	403,941176
0,126	0,8	7,848	0,02	392,4
0,129	0,9	8,829	0,023	383,869565
0,131	1	9,81	0,025	392,4
0,133	1,1	10,791	0,027	399,666667
0,136	1,2	11,772	0,03	392,4
0,14	1,3	12,753	0,034	375,088235
0,142	1,4	13,734	0,036	381,5
0,145	1,5	14,715	0,039	377,307692
0,147	1,6	15,696	0,041	382,829268
0,15	1,7	16,677	0,044	379,022727
0,152	1,8	17,658	0,046	383,869565
0,155	1,9	18,639	0,049	380,387755
0,157	2	19,62	0,051	384,705882



4.1.2 DETERMINATION DE LA CONSTANTE DE RAIDEUR

En Dynamique:

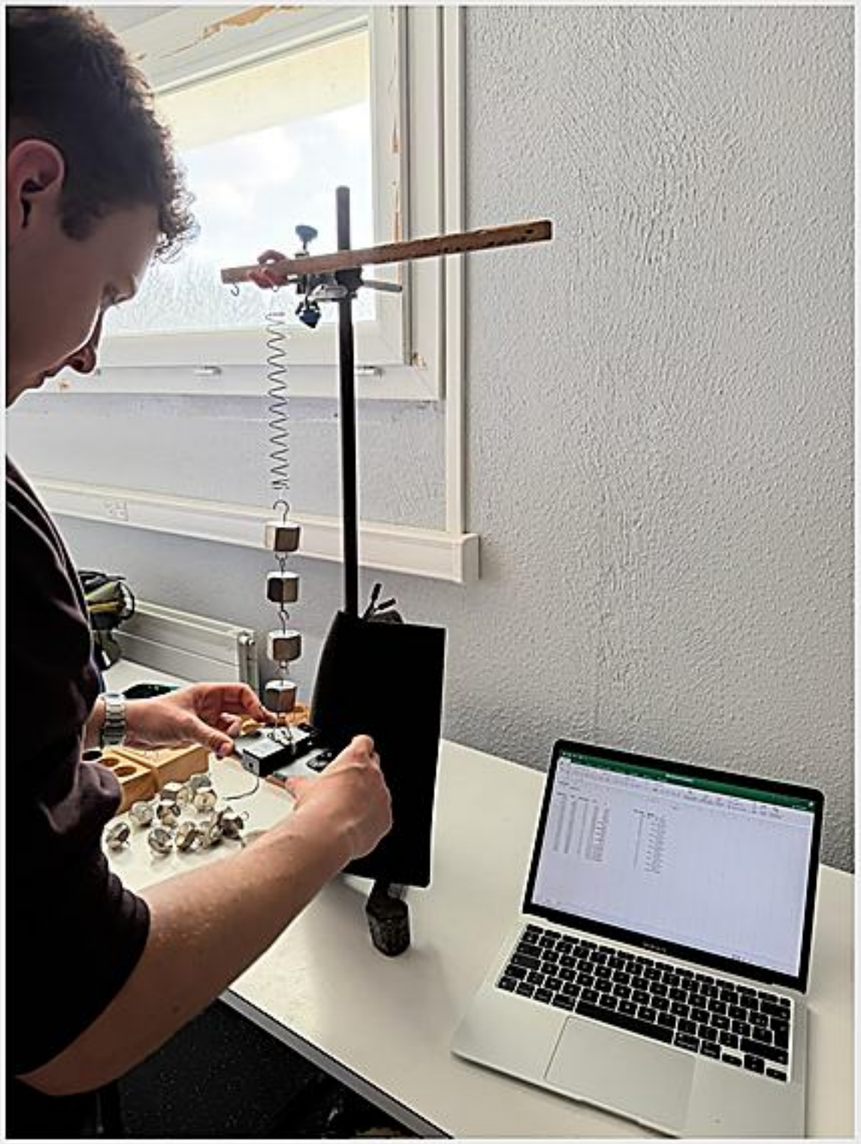
- Mesure à l'aide de *PhyPhox*

- Constante obtenue : **k=366±3 N/m**

$$T = 2\pi\sqrt{\frac{m}{k}}$$

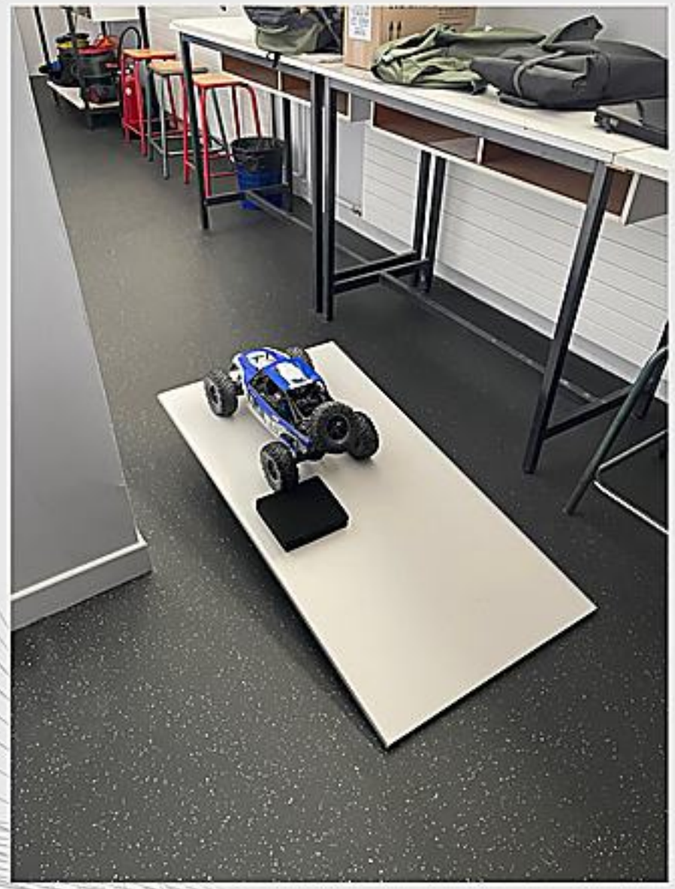
	K - dynamique	Incertitude
	366,149583	2,634754121

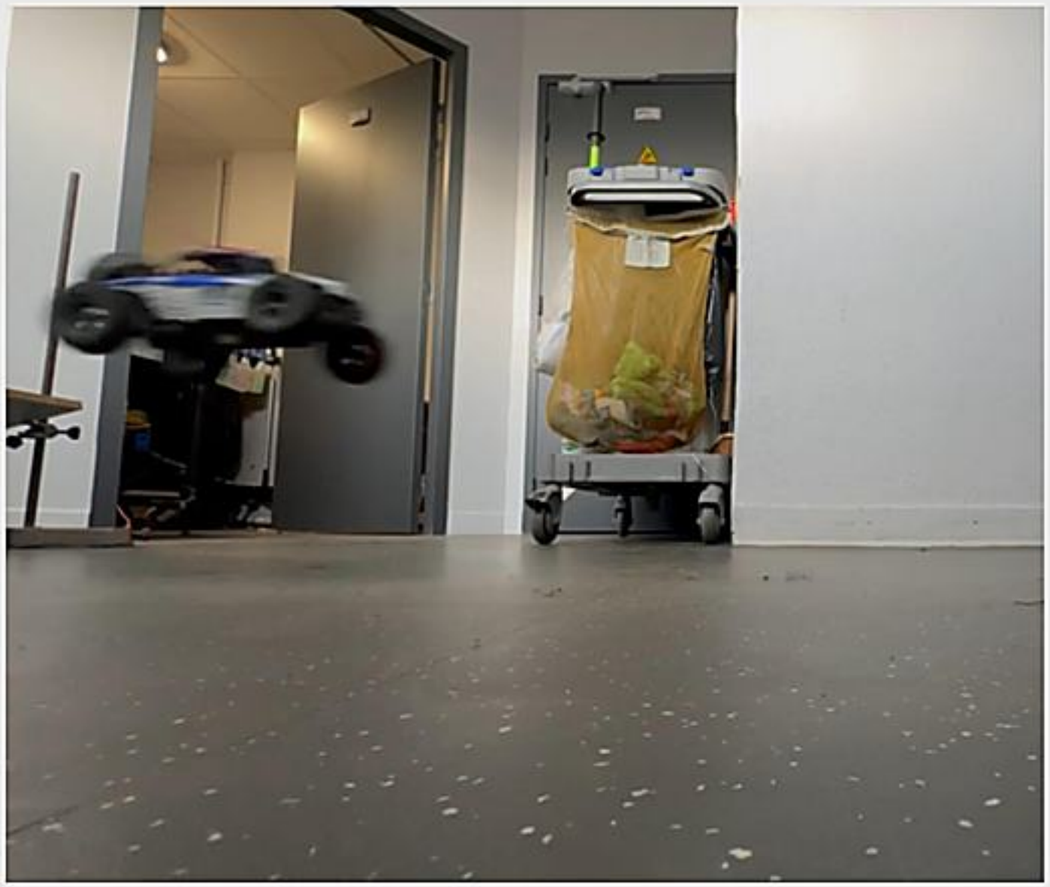
Masse (kg)	Periode	K
0,4	0,26	356,2401589
0,6	0,3	355,3057584
0,8	0,33	366,1451036
1	0,36	368,5870779
1,2	0,39	365,9734965
1,4	0,42	360,3188908
1,6	0,44	369,0905778
1,8	0,46	375,007653
2	0,48	378,67753



4.2. EXPÉRIENCE 2

Saut de la voiture à l'aide d'un plan incliné d'angle réglable permettant l'étude de son amortissement





4.2.1 DETERMINATION DU RÉGIME DE L'AMMORTISSEMENT

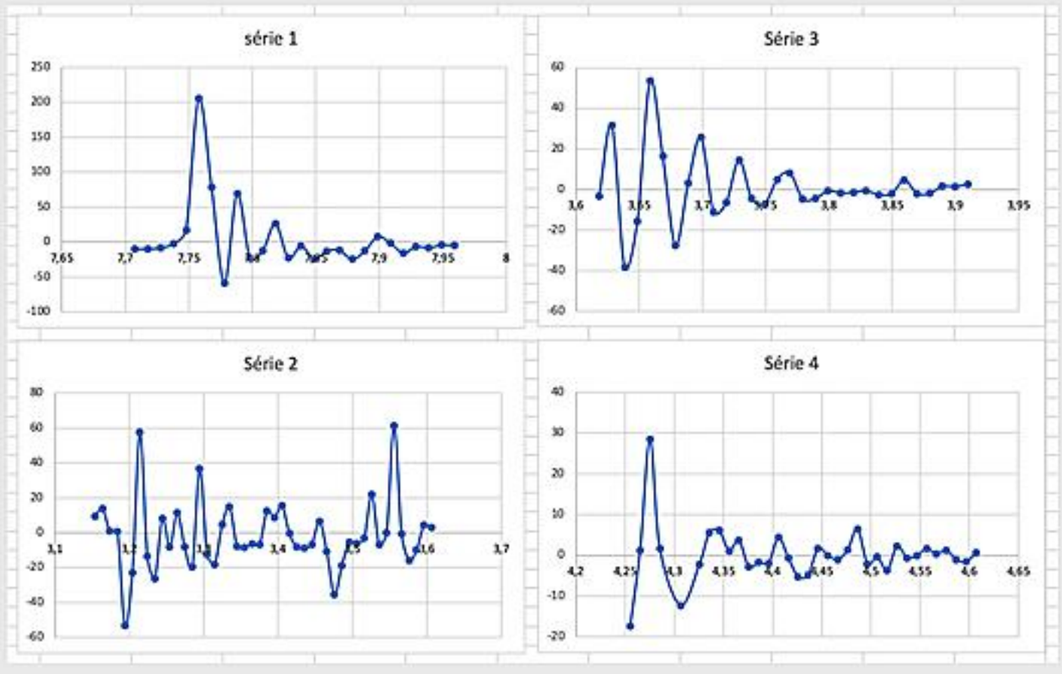
$$\lambda = \sqrt{4km(1 - \frac{\omega_P^2 * m}{k})}$$

$$\omega_0 = \sqrt{\frac{k}{m}}$$

$$Q = \frac{\sqrt{km}}{\lambda}$$

$$\omega_p = \omega_0\sqrt{1 - \frac{1}{4Q^2}}$$

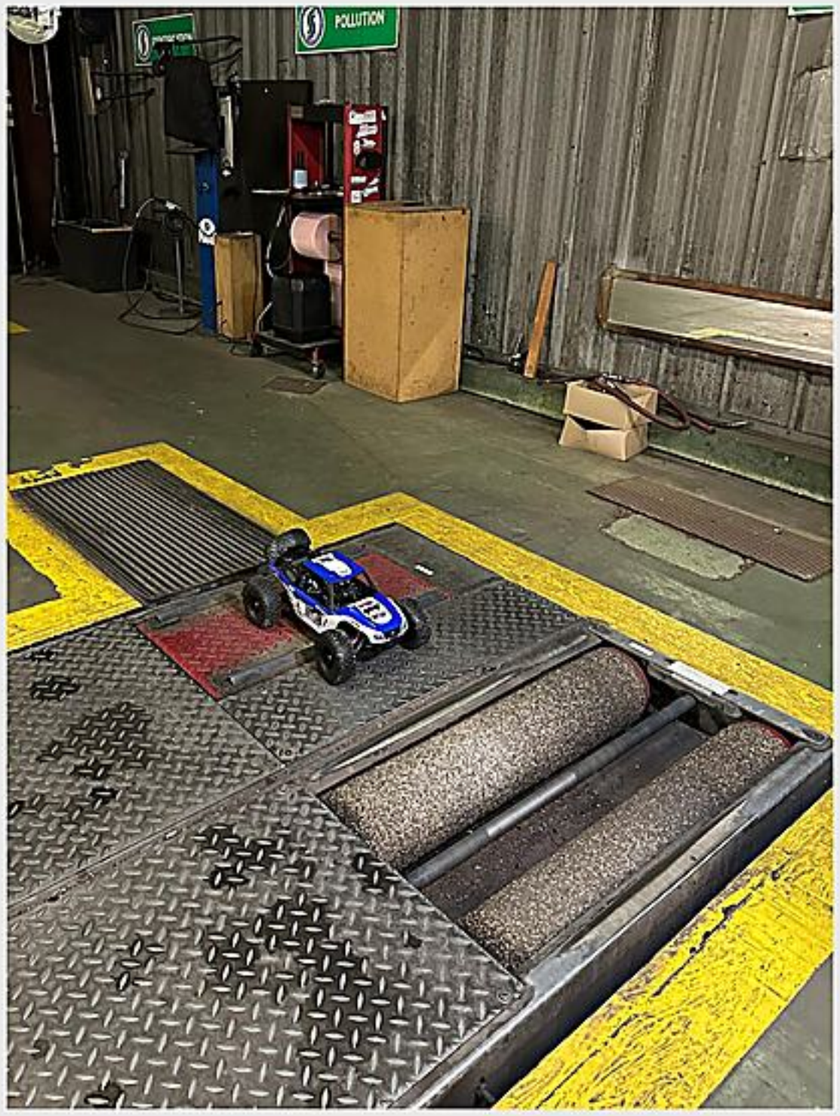
k	388		lambda	Incertitude			
m	2,492		62,1686278	0,00359127			
t(s)	Az			Tp	wp	lambda	
4,58758963	61,6256582		0,020097	0,06029067	0,37881743	62,1612368	Serie 1
4,60768663	30,9642345		0,030145				
4,63783163	20,7639446		0,13063				
4,76846163	7,7616613						
7,75795996	205,707255		0,030146	0,04689433	0,29464579	62,172562	Serie 2
7,78810596	69,38997		0,030146				
7,81825196	26,2627083		0,080391				
7,89864296	7,76852301						
3,21399188	57,4790701		0,08038899	0,06029175	0,37882423	62,1612357	Serie 3
3,29438087	36,6817239		0,11053501				
3,40491588	15,5771018		-0,070341				
3,33457487	14,7692984		0,120584				
3,45515887	6,52025356						
3,65861921	53,353822		0,040195	0,0351705	0,22098277	62,1801493	Serie 4
3,69881421	25,6831744		0,030146				
3,72896021	14,4225047						
3,95003121	-0,8751949						
4,13090721	2,50124297						
4,27520408	28,4501587		0,060293	0,05275575	0,33147415	62,1679554	Serie 5
4,33549708	5,55452216		0,010048				
4,34554508	6,14106572		0,060292				
4,40583708	4,34795214		0,08039				
4,48622708	6,45580848						



4.3. EXPERIENCE 3

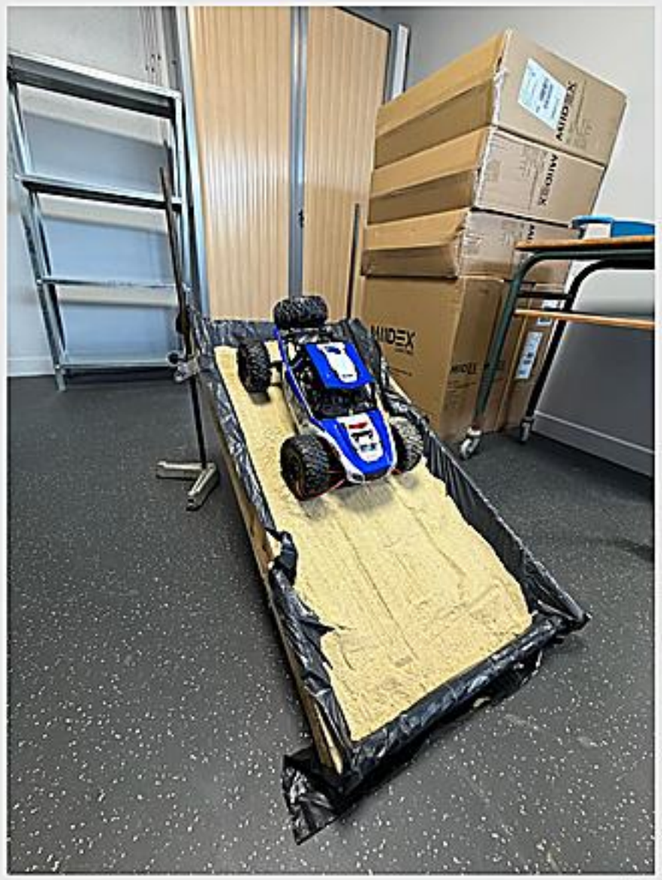
Mesure de l’amortissement sur un banc de suspension pour voiture dans un centre de contrôle technique automobile

→ matériel non adapté au buggy télécommandé



4.4. EXPERIENCE 4

Mesure des coefficients de frottements sur différents revêtements, et avec différents types de pneus



4.4.1. DETERMINATION DU COEFFICIENT DE FROTTEMENT

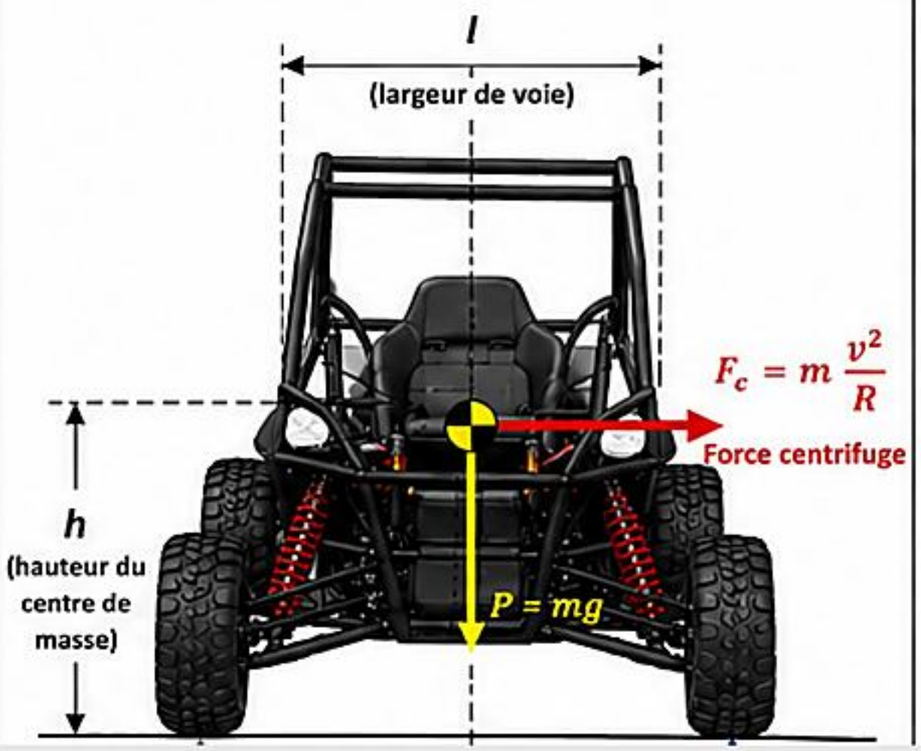
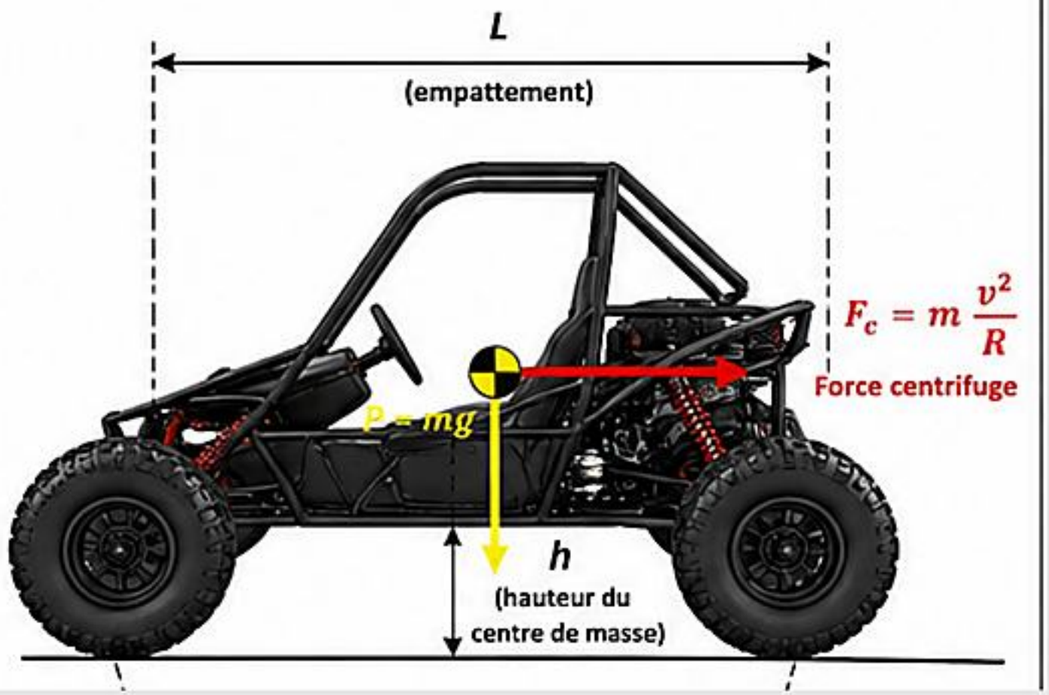
Sable	Hauteur	Angle (rad)	Incertitude			Béton	Hauteur	Angle	Angle (rad)	Incertitude			Terre	Hauteur	Angle (rad)	Angle	Incertitude	
	35	0,357571	0,006117				39	32,29283	0,563616	0,010168				74	0,83307	47,73142	0,016074	
	29	0,294227	Moyenne				40	33,22616	0,579906	Moyenne				54	0,570437	32,68364	Moyenne	
	32	0,325729	0,314857	18,03998			38,5	31,82977	0,555534	0,593641	34,01314			61,5	0,662386	37,95192	0,699667	40,08796
	28	0,283794					40,5	33,69656	0,588116					68,5	0,754604	43,23561		
	26,5	0,268204					39	32,29283	0,563616					73	0,818322	46,88639		
	33	0,336304					42,5	35,60485	0,621422					61	0,656061	37,5895		
	26	0,263022					40,5	33,69656	0,588116					71	0,789498	45,23492		
	27	0,273393					39,5	32,75827	0,57174					61,5	0,662386	37,95192		
	33	0,336304					40,2	33,41401	0,583185					67	0,734209	42,06706		
	31	0,315193					44	37,06637	0,64693					66,5	0,727494	41,68233		
	33	0,336304					38	31,36901	0,547493					65	0,707584	40,5416		
	33	0,336304					42	35,12362	0,613023					57,5	0,612604	35,09963		
	29	0,294227					44,5	37,55979	0,655542					55	0,582364	33,36701		
	32	0,325729					43,5	36,57613	0,638374					63	0,681553	39,05012		
	31,5	0,320457					42	35,12362	0,613023					67,5	0,740965	42,45415		
	35	0,357571					46	39,06023	0,68173					66,5	0,727494	41,68233		
	31,2	0,317297					33	26,87559	0,469068					68	0,747763	42,84364		
	31,8	0,323619					41	34,16956	0,596371					57,5	0,612604	35,09963		
	31,6	0,32151					39	32,29283	0,563616					66	0,720819	41,29987		
	32,5	0,331012					42	35,12362	0,613023					60,5	0,649766	37,22884		
	29	0,294227					42	35,12362	0,613023					64,5	0,701023	40,16568		
	Hauteur	Angle (rad)	Incertitude				Hauteur	Angle	Angle (rad)	Incertitude				Hauteur	Angle (rad)	Angle	Incertitude	
	30	0,304693	0,005364				26,5	21,28539	0,3715	0,010335				44,5	0,461174	26,42334	0,010761	
	23	0,232078	Moyenne				34	27,75899	0,484486	Moyenne				42	0,433445	24,83459	Moyenne	
	26	0,263022	0,272974	15,64027			33	26,87559	0,469068	0,384206	22,01337			44	0,455599	26,10388	0,450826	25,83041
	28	0,283794					27	21,70717	0,378862					43,5	0,450038	25,78529		
	31	0,315193					26	20,86482	0,36416					42	0,433445	24,83459		
	27	0,273393					24	19,19396	0,334998					43	0,444493	25,46756		
	27	0,273393					25,5	20,44543	0,35684					40	0,411517	23,57818		
	29	0,294227					24,5	19,61003	0,34226					43	0,444493	25,46756		
	29	0,294227					33	26,87559	0,469068					44	0,455599	26,10388		
	24	0,242366					24	19,19396	0,334998					45	0,466765	26,74368		
	29	0,294227					26	20,86482	0,36416					40	0,411517	23,57818		
	22	0,221814					27	21,70717	0,378862					45	0,466765	26,74368		
	28	0,283794					28	22,55447	0,39365					38,5	0,395208	22,64374		
	26	0,263022					28	22,55447	0,39365					46	0,477995	27,38711		
	27	0,273393					23	18,36495	0,320529					31	0,315193	18,05923		
	24	0,242366					31	25,12899	0,438584					42,5	0,438962	25,15066		
	28	0,283794					27	21,70717	0,378862					42	0,433445	24,83459		
	29	0,294227					26,5	21,28539	0,3715					48	0,500655	28,6854		
	27	0,273393					24,5	19,61003	0,34226					49	0,51209	29,34058		
	26	0,263022					26,5	21,28539	0,3715					49	0,51209	29,34058		
	26	0,263022					29	23,40702	0,40853					52	0,546851	31,33225		

$$f = tan(\alpha)$$

	Sable	Terre	Beton
Avec Sillion	0,325691	0,841719	0,674842
Sans Sillion	0,279963	0,484074	0,404298

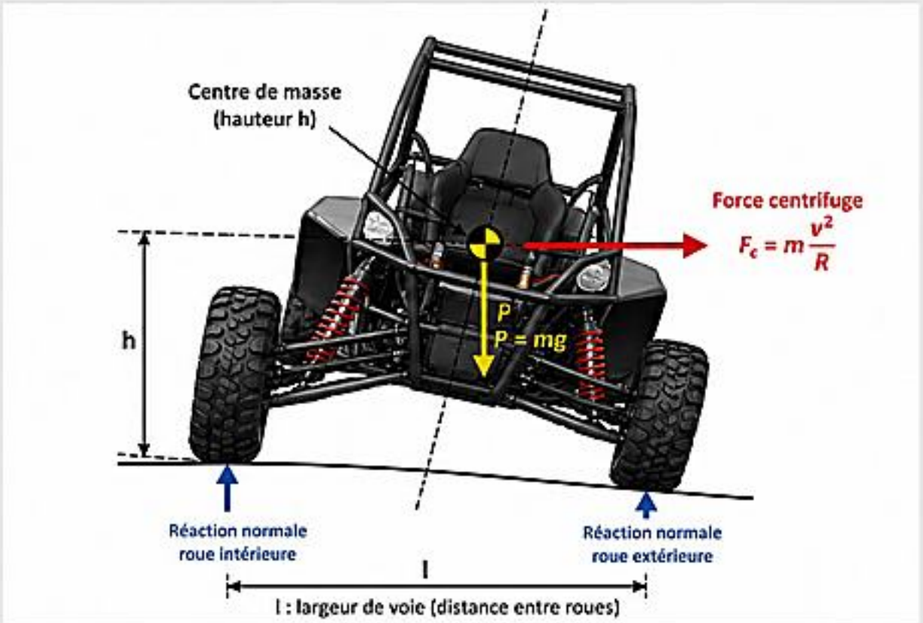
4.5. EXPERIENCE 5

Evaluer les capacités optimal du buggy en virage

VUE DE FACE	VUE DE PROFIL
	

4.5.1. DETERMINATION DE L'ANGLE LIMITE DE BRAQUAGE THÉORIQUE

Démonstration de la formule de la vitesse de retournement



Le Théorème du Moment Cinétique assure que :

$$\vec{L_0} = \sum \vec{M}$$

Ici, comme la voiture est immobile, dans le referentiel de la voiture: $\vec{L_0} = \vec{0}$

On a :

$$\vec{M}(P) + \vec{M}(F_c) = \vec{0}$$

$$m\frac{v^2}{R} * h = mg * \frac{l}{2}$$

$$\Big| \quad v = \sqrt{\frac{glR}{2h}}$$

Or, le rayon de la trajectoire vaut:

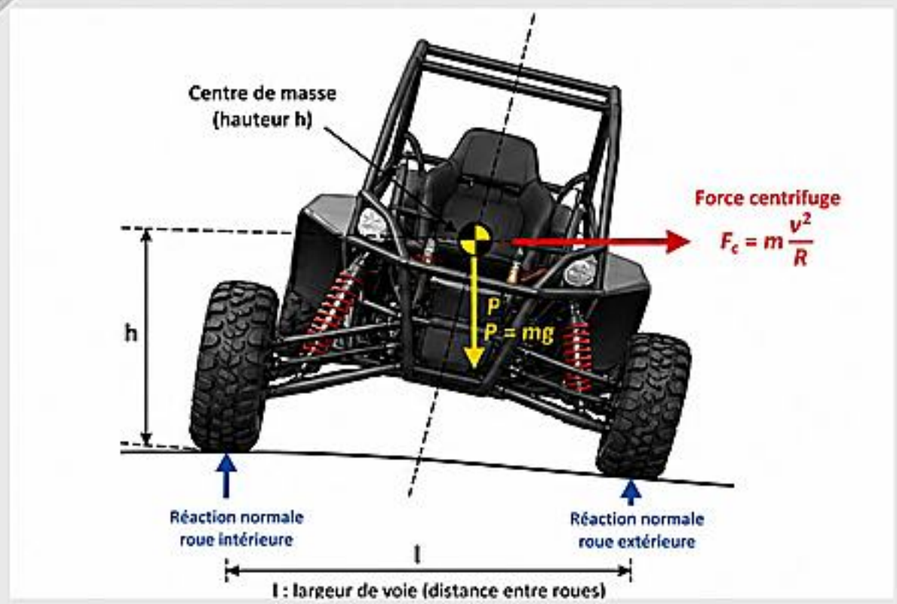
$$R = \frac{L}{tan(\theta)}$$

On obtient ainsi :

$$v_{retournement} = \sqrt{\frac{glL}{2htan(\theta)}}$$

4.5.1. DETERMINATION DE L'ANGLE LIMITE DE BRAQUAGE THÉORIQUE

Démonstration de la formule de la vitesse de retournement



On s'interresse au buggy de masse m, dans le référentiel terrestre (galiléen)

Le Principe Fondamental de la Dynamique assure que :

$$m\vec{a} = \sum \vec{F_{ext}}$$

Ainsi: $m\frac{\vec{v^2}}{R} = \vec{F_f}$

Or les lois de Coulomb donnent :

$$\vec{F_{f,max}} = \mu\vec{N}$$

De plus :

$$N = mg$$

Donc, on a : $m\frac{v^2}{R} = \mu mg$

Et ainsi : $v = \sqrt{\mu g R}$

Or, le rayon de la trajectoire vaut:

$$R = \frac{L}{tan(\theta)}$$

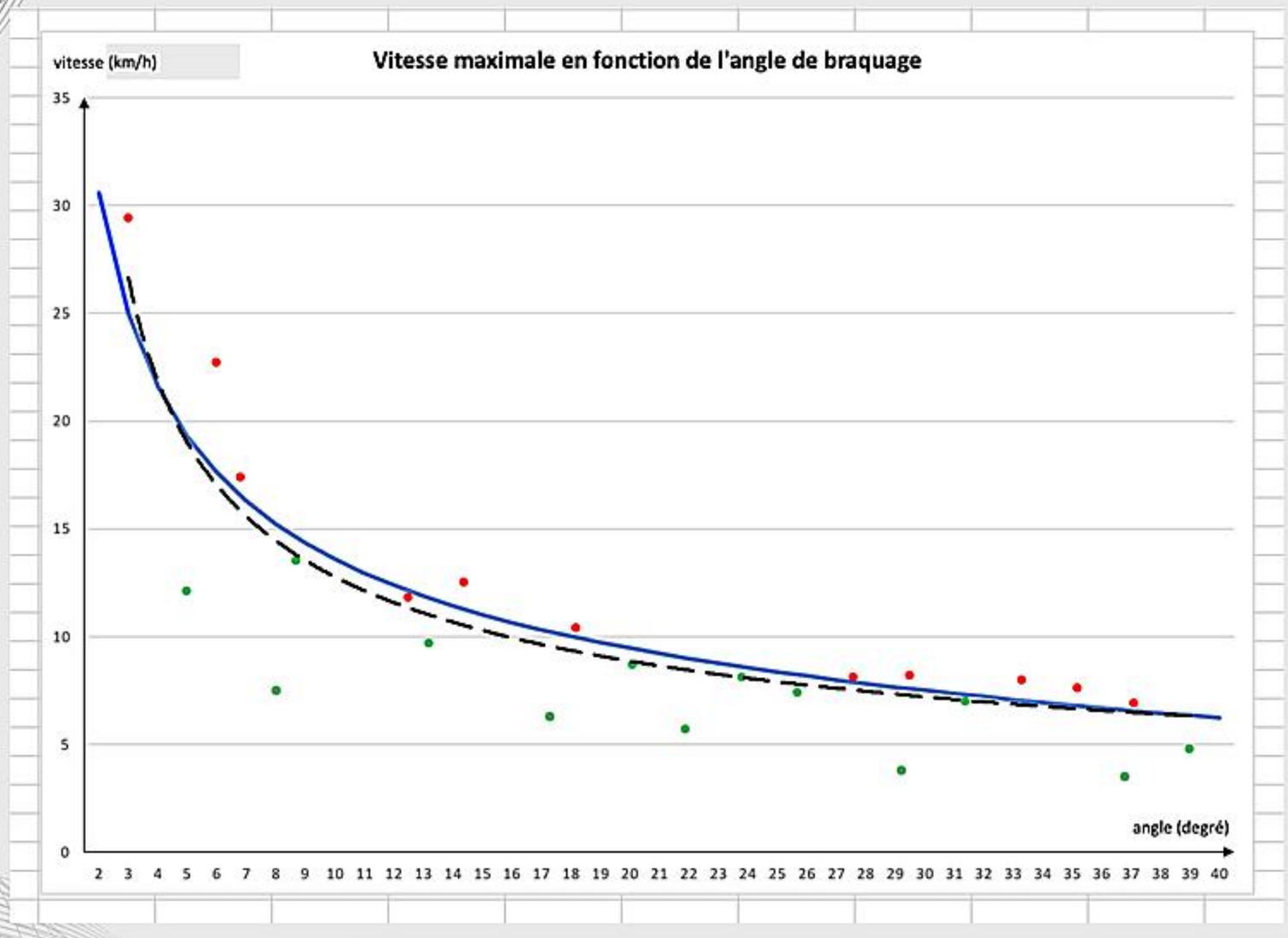
On obtient ainsi :

$$v_{glissement} = \sqrt{\frac{\mu g L}{tan(\theta)}}$$

4.5.2. DETERMINATION DE L'ANGLE LIMITE DE BRAQUAGE EXPERIMENTALEMENT

Pointage numérique

4.5.3. DETERMINATION DE L'ANGLE LIMITE DE BRAQUAGE EXPERIMENTALEMENT



Légende

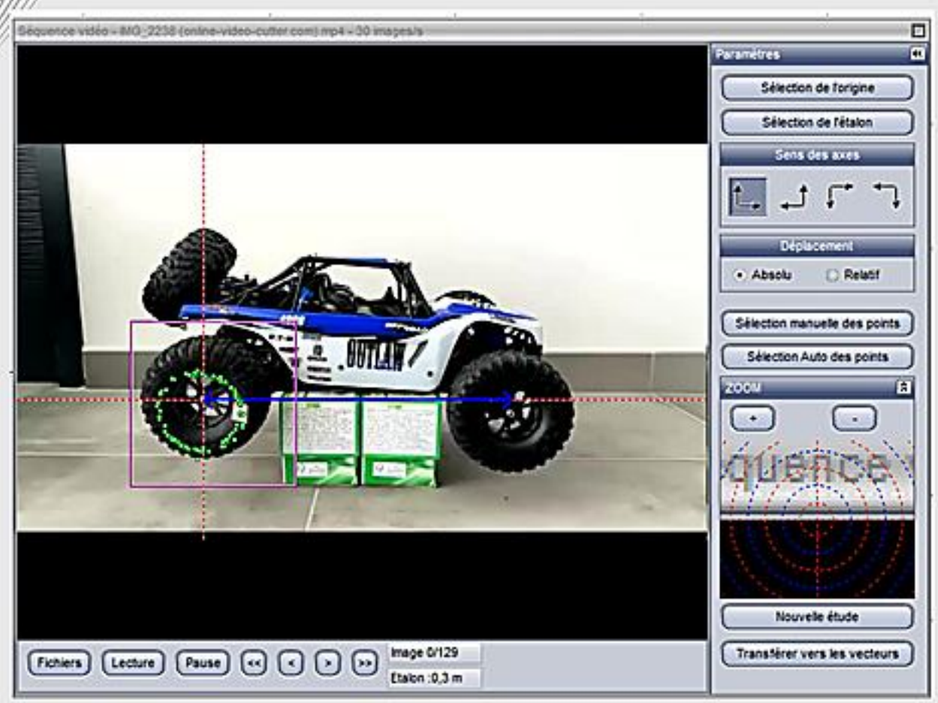
- Courbe théorique
- Courbe experimentale
- Retournement du buggy
- Non retouenement du buggy

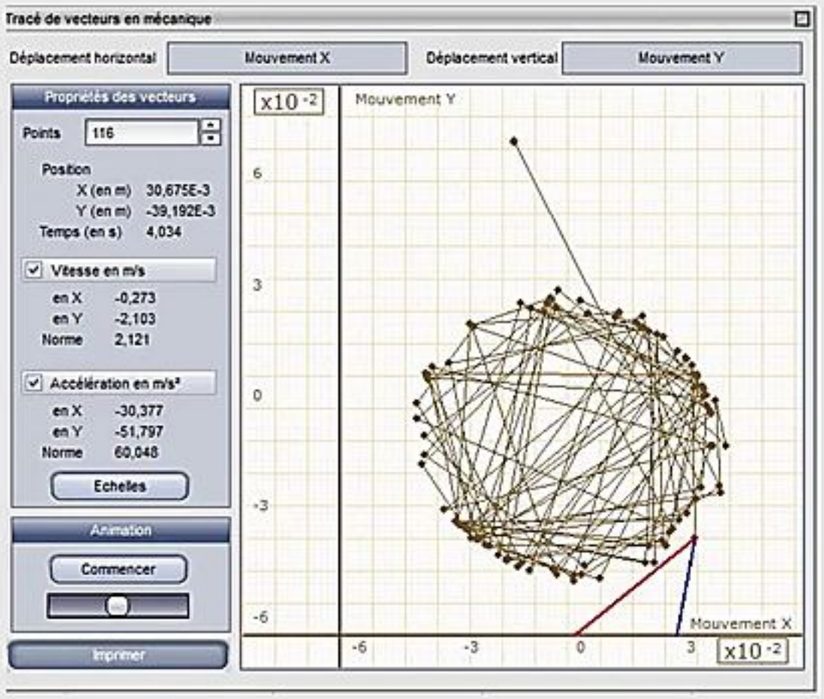
4.6. EXPERIENCE 6

Détermination de l’accélération du moteur du buggy

4.5.2. DETERMINATION DE L'ACCÉLÉRATION DU MOTEUR DU BUGGY

Pointage numérique





$$a = \frac{V_{fin} - V_{debut}}{t_{fin} - t_{debut}} = \frac{2,121 - 0,277}{4,034 - 0,033}$$

$$a \approx 0,46 \text{ m/s}^2$$

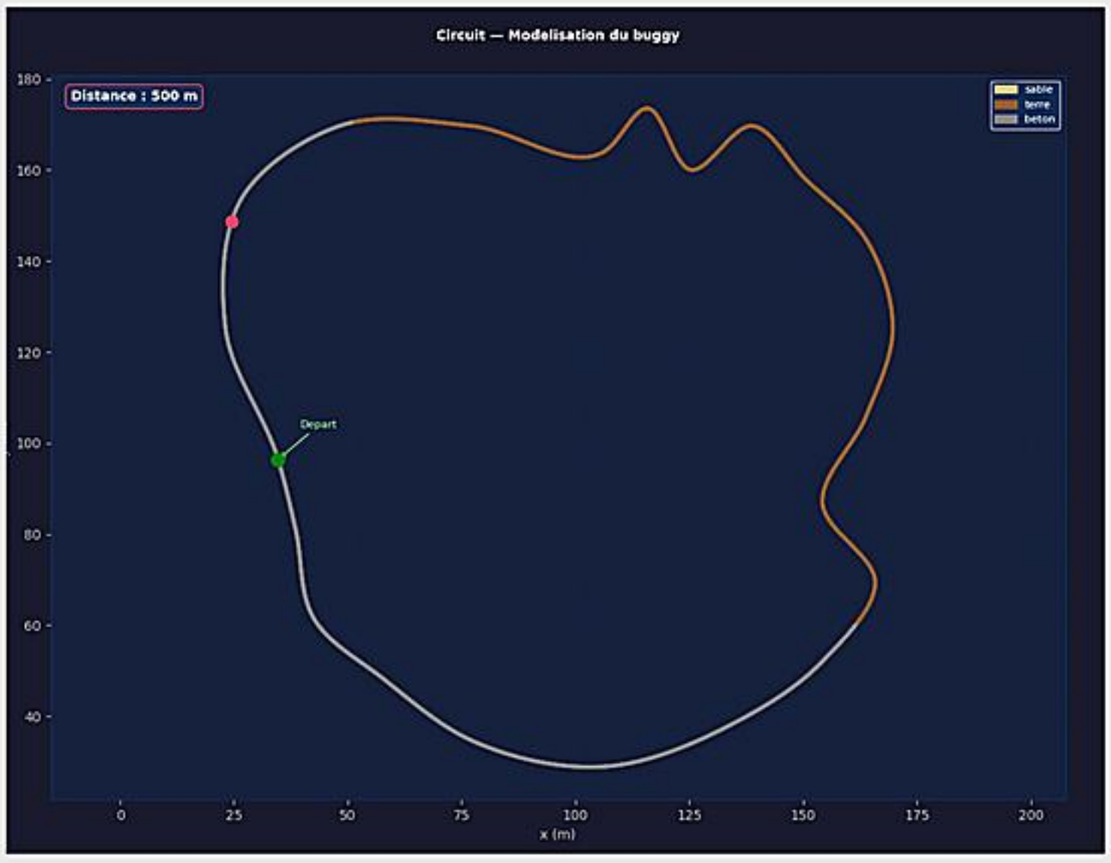
$$a_{réel} = 0,46 \times 16 = \boxed{7,4 \text{ m/s}^2}$$

4.6. MODELISATION

```python
# ============================================================
# 1. PARAMETRES PHYSIQUES
# ============================================================
m     = 2.492        # masse (kg)
g     = 9.81         # gravite (m/s²)
h     = 0.19         # hauteur centre de gravite (m)
l     = 0.30         # demi-voie (m)
V_MAX = 60 / 3.6     # vitesse max moteur (m/s)
A_MOTEUR = 7.4

# ============================================================
# 2. DONNEES EXPERIMENTALES
# ============================================================
# Sensibilite de l'amortissement selon le sol
# alpha proche de 0 : lambda peu important | alpha proche de 1 : lambda tres important
ALPHA_SOL = {"beton": 0.1, "terre": 0.4, "sable": 0.9}

PNEUS = {
    "slick":  {"mu": {"sable": 0.28, "terre": 0.75, "beton": 0.85}},
    "sillon": {"mu": {"sable": 0.42, "terre": 0.84, "beton": 0.67}},
}
RESSORTS = {
    "souple": {"K": 388, "lam": 62.2},
    "rigide": {"K": 927, "lam": 96.1},
}
```



4.6. MODELISATION

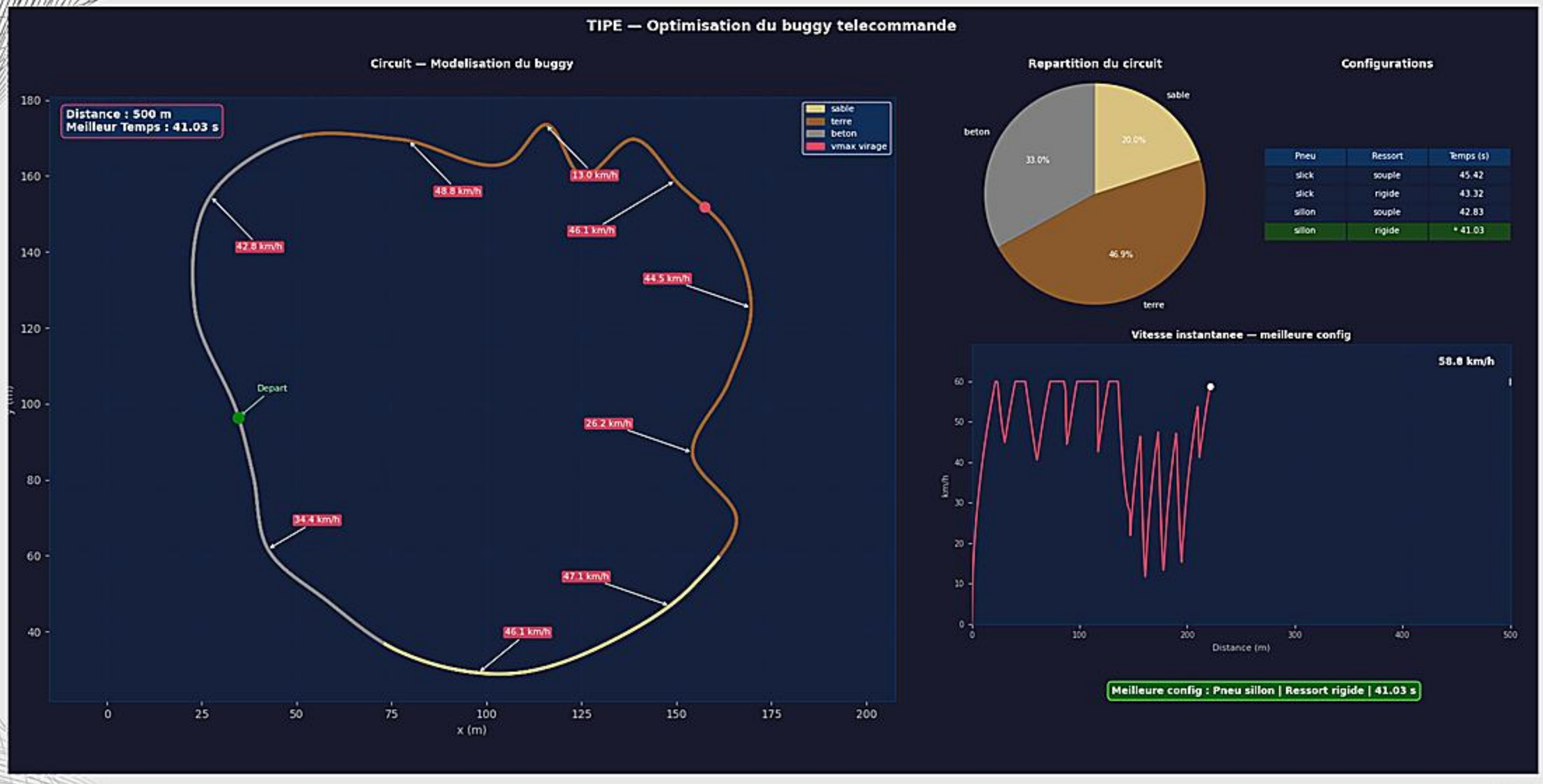
Configurations

Pneu	Ressort	Temps (s)
slick	souple	45.42
slick	rigide	43.32
sillon	souple	42.83
sillon	rigide	* 41.03

5. CONCLUSION

- Constantes de Raideur:
  - **K1= 388±2 N/m**
  - **K2=927±1 N/m**
- Coefficients D'amortissement Fluide:
  - **λ1= 62,20±0,03 N.s/m**
  - **λ2= 96.10±0.02 N.s/m**
- L’amortissement du buggy suit un régime **pseudo-périodique (Q>1/2)**
- L’accélération du moteur du buggy est de **7,4m/s²**

5. CONCLUSION



6. BIBLIOGRAPHIE

[1] MOTORLEGEND : Les amortisseurs :
https://www.motorlegend.com/entretien-reparation
/trains-roulants-voiture/les-amortisseurs/8,11745.html

[2] PHYSIQUE XXI : Les oscillations amorties et forcées :
https://physique.cmaisonneuve.qc.ca
/btardif/NYC/PhysiqueXXI-TomeC-Section-1.7.pdf

[3] CHICHERY ARNAUD : Mesure de la constante de raideur
d'un ressort : http://chichery.
arnaud.free.fr/Documents/TP-Mecanique/files/TPM1.pdf

[4] GWENAEL RAILLET : Travail pratique : Frottement solide :
http://gwenael.raillet.free.fr
/documents/TP/TP-Frottement%20solide.pdf

ANNEXES

Démonstration de la formule du coefficient de frottement

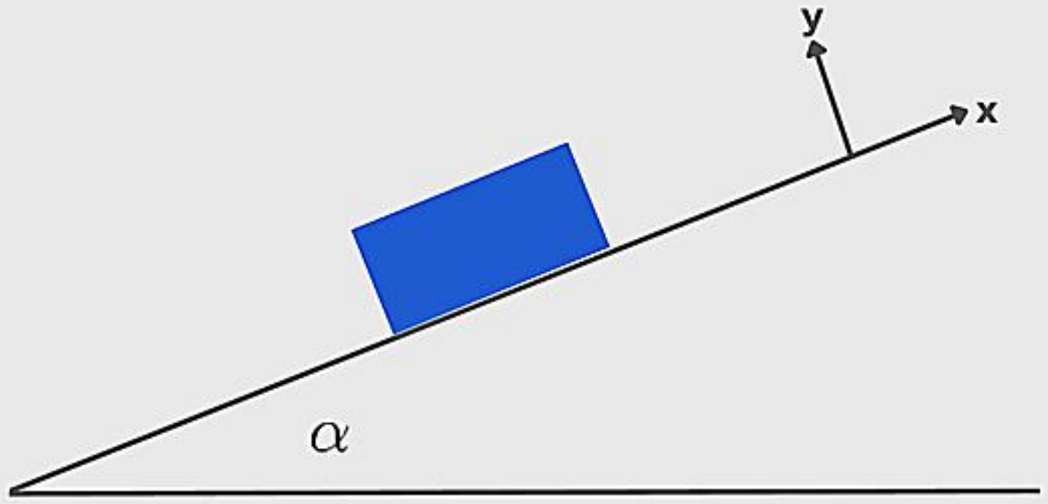
Le Théorème de la Résultante Cinétique s'écrit:

$$m\vec{a}(G) = \vec{T} + \vec{N} + \vec{P}$$

Or, en l'absence de glissement :

$$\vec{a}(G) = \vec{0}$$

Donc on obtient : $\vec{T} + \vec{N} + \vec{P} = \vec{0}$



En projetant sur les deux axes x et y on a :

$$N = mgcos(\alpha) \quad T = mgsin(\alpha)$$

Or, les Lois de Coulomb assurent que : $T = fN$

On obtient finalement que : $f = tan(\alpha)$

ANNEXES

Démonstration de la modélisation du coefficient de frottement

Point de départ

Sur terrain meuble, un ressort mal amorti rebondit

→ la roue perd contact → l'adhérence est réduite.

On modélise cette perte par un facteur de pénalité :

$$\mu_{eff} = \mu \cdot f(\lambda, sol)$$

avec $f \in [0, 1]$ — plus f proche de 1, moins la pénalité est grande.

Construction de f

On veut que f vérifie trois propriétés physiques :

Condition physique	Traduction mathématique
$\lambda \to \infty$ (amorti parfait)	$f \to 1$ (pas de pénalité)
$\lambda \to 0$ (pas d'amortissement)	$f \to 1 - \alpha_{sol}$ (pénalité max)
Béton ($\alpha = 0.1$)	$f \approx 1$ quel que soit λ
Sable ($\alpha = 0.9$)	f très sensible à λ

La forme exponentielle satisfait ces conditions

On pose :

$$f(\lambda) = 1 - \alpha_{sol} \cdot e^{-\lambda / \lambda_{ref}}$$

Normalisation par λ_{ref}

On normalise par $\lambda_{ref} = \lambda_{rigide} = 96.1$ N·s/m

(le ressort le plus amorti sert de référence) :

$$e^{-\lambda/\lambda_{ref}} \ : \ \text{souple} \to e^{-62.2/96.1} = 0.52$$

$$e^{-\lambda/\lambda_{ref}} \ : \ \text{rigide} \to e^{-96.1/96.1} = e^{-1} = 0.37$$

$$\mu_{eff} = \mu \cdot (1 - \alpha_{sol} \cdot e^{-\lambda / \lambda_{ref}})$$

Modèle phénoménologique : la forme exponentielle est justifiée par les propriétés limites. α_{sol} pourrait être calibré expérimentalement en mesurant l'adhérence réelle sur chaque sol.

ANNEXES

Démonstration de la formule de l'accélération réelle du buggy

Limite 1 — Le moteur

Le buggy accélère en ligne droite. Le moteur entraîne les roues par frottement.

Le moteur fournit F_{moteur} aux roues.

Par la 2ème loi de Newton :

$$F_{moteur} = m \cdot a_{moteur}$$

$$a_{moteur} = \frac{F_{moteur}}{m}$$

Constante mesurée expérimentalement, **indépendante du sol.**

Limite 2 — L'adhérence

Pour transmettre la force sans glisser, il faut $F_{traction} \leq F_{frottement,max}$.

La loi de Coulomb donne :

$$F_{frottement,max} = \mu_{eff} \cdot N = \mu_{eff} \cdot mg$$

Si $F_{moteur} > F_{frottement,max}$ → la roue patine.

L'accélération maximale par adhérence :

$$a_{adhérence} = \frac{\mu_{eff} \cdot mg}{m} = \mu_{eff} \cdot g$$

La physique impose le minimum

Le buggy ne peut accélérer qu'aussi vite que la limite la plus contraignante :

$$a_{réelle} = \min(a_{moteur},\ \mu_{eff} \cdot g)$$

ANNEXES

Démonstration de la formule de la vitesse de retournement

Mise en situation

Le buggy prend un virage de rayon R à vitesse v.

On cherche v_r à partir de laquelle

il bascule sur ses roues extérieures.

Bilan des forces

- Poids mg vers le bas en G
- N_i, N_e : réactions normales int./ext.
- Force centrifuge $F_c = \frac{mv^2}{R}$ en G

Condition de renversement

La roue intérieure se soulève :

$$N_i = 0$$

Déformation du ressort

Le transfert de charge latéral comprime

le ressort extérieur d'une quantité δ :

$$\delta = \frac{F_c \cdot h}{K \cdot 2l} = \frac{mv^2 h}{2KlR}$$

On pose $A = \frac{mh}{KlR}$

Bilan des moments / roue ext. ($N_i = 0$)

$\sum M_{ext} = 0$

Force	Bras	Moment
Poids mg	l	$+mg \cdot l$
Force centrifuge $\frac{mv^2}{R}$	h	$-\frac{mv^2}{R} \cdot h$
Ressort $K\delta$	$2l$	$-K\delta \cdot 2l$

Équation du moment nul

$$mg \cdot l = \frac{mv^2}{R} \cdot h + K\delta \cdot 2l$$

En substituant δ, équation du 2nd degré en v^2 :

$$A \cdot v^4 + 2h \cdot v^2 - g \cdot l \cdot R = 0$$

Résolution

Discriminant $\Delta = (2h)^2 + 4AglR$, racine positive :

$$v^2 = \frac{-2h + \sqrt{4h^2 + 4AglR}}{2A}$$

$$\boxed{v_r = \sqrt{\frac{-2h + \sqrt{4h^2 + 4AglR}}{2A}}}$$

ANNEXES

Démonstration de la formule de la vitesse

Point de départ

Mouvement uniformément accéléré (a constante).

Par définition :

$$a = \frac{dv}{dt} \quad \text{et} \quad v = \frac{ds}{dt}$$

$$a = \frac{dv}{dt} = \frac{dv}{ds} \cdot \frac{ds}{dt} = v \cdot \frac{dv}{ds}$$

Donc :

$$a \cdot ds = v \cdot dv$$

Intégration

Entre l'état initial (s_0, v_0) et final (s, v) :

$$\int_{s_0}^{s} a\, ds = \int_{v_0}^{v} v\, dv$$

$$a \cdot \Delta s = \frac{v^2 - v_0^2}{2}$$

$$v^2 = v_0^2 + 2a \cdot \Delta s$$

Application dans le code

À chaque pas ds, le buggy accélère avec $a_{réelle}$:

$$v_i = \sqrt{v_{i-1}^2 + 2 \cdot a_{réelle} \cdot ds}$$

Puis on plafonne à v_{max} du virage :

$$v_i = \min\left(\sqrt{v_{i-1}^2 + 2 \cdot a_{réelle} \cdot ds},\ v_{max}\right)$$

Ce qui correspond exactement à la ligne du code : ***(l.160)***

```
v[i] = min(np.sqrt(v[i-1]**2 + 2*acc_arr[i]*DS_ARR[i-1]), vmax_arr[i])
```

Résultat

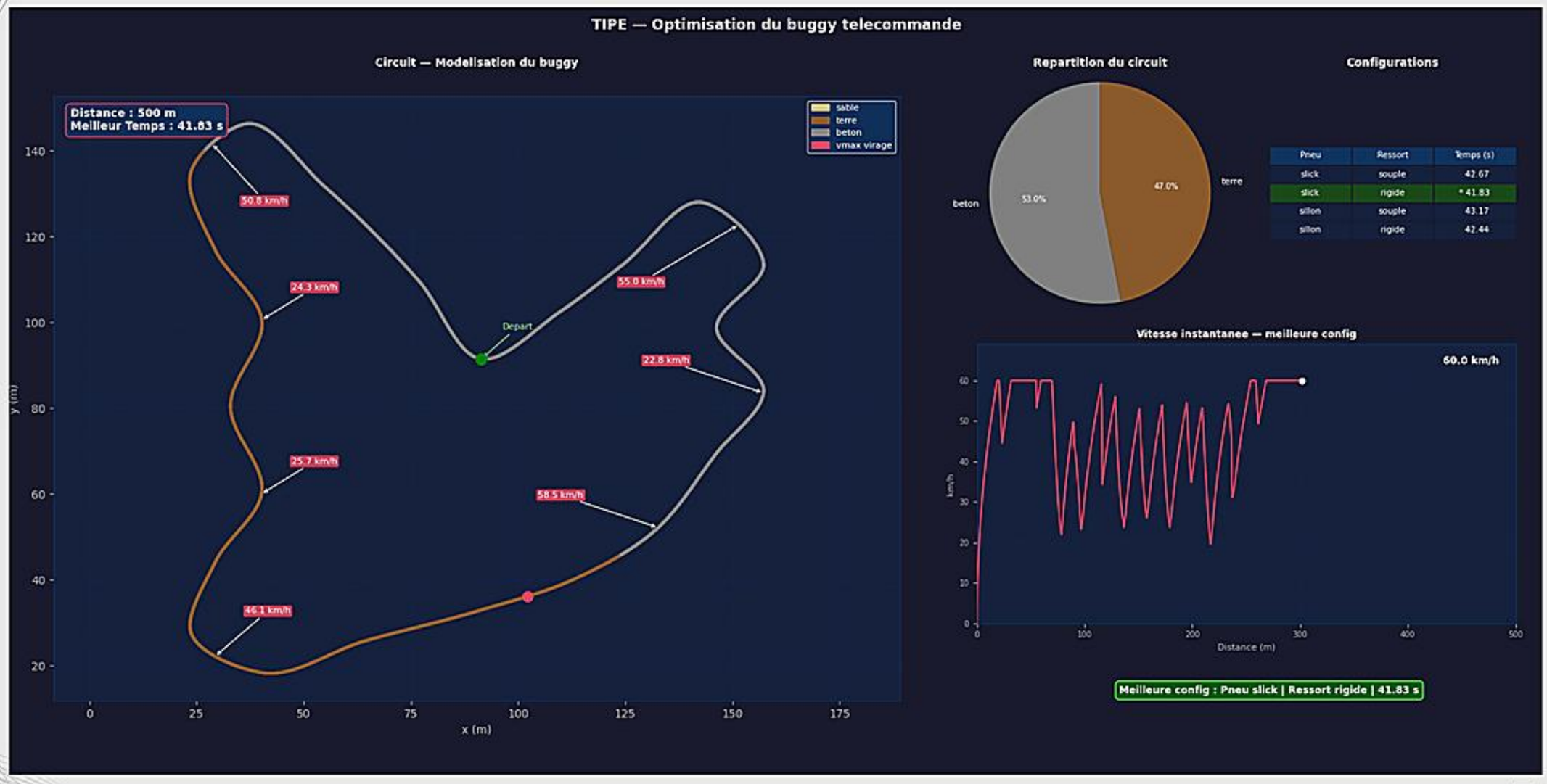
La vitesse simulée tient compte à la fois de l'accélération

réelle et de la limite de virage à chaque instant :

$$v_i = \min\left(\sqrt{v_{i-1}^2 + 2\, a_{réelle} \cdot ds},\ v_{max}\right)$$

ANNEXES

Autre circuit



ANNEXES

Proramme Python

```python
# -*- coding: utf-8 -*-
"""
TIPE — Optimisation du buggy telecommande
Modelisation physique : pneus x ressorts x sols
"""
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.animation import FuncAnimation
from itertools import product as itertools_product

# ============================================================
# 1. PARAMETRES PHYSIQUES
# ============================================================
m     = 2.492        # masse (kg)
g     = 9.81         # gravite (m/s²)
h     = 0.19         # hauteur centre de gravite (m)
l     = 0.30         # demi-voie (m)
V_MAX = 60 / 3.6     # vitesse max moteur (m/s)
A_MOTEUR = 7.4

# ============================================================
# 2. DONNEES EXPERIMENTALES
# ============================================================
ALPHA_SOL = {"beton": 0.1, "terre": 0.4, "sable": 0.9}

PNEUS = {
    "slick":  {"mu": {"sable": 0.28, "terre": 0.75, "beton": 0.85}},
    "sillon": {"mu": {"sable": 0.42, "terre": 0.84, "beton": 0.67}},
}
RESSORTS = {
    "souple": {"K": 388, "lam": 62.2},
    "rigide": {"K": 927, "lam": 96.1},
}
LAM_REF = max(r["lam"] for r in RESSORTS.values())  # normalisation = ressort le plus rigide

COULEURS_SOL = {"sable": "#d9c27f", "terre": "#8b5a2b", "beton": "#808080"}
```

```python
# ============================================================
# 3. MODELES PHYSIQUES
# ============================================================
Tabnine | Edit | Test | Explain | Document
def mu_eff(mu, sol, lam):
    """
    Adherence effective tenant compte de l'amortissement et du sol.

    Formule : mu_eff = mu * (1 - alpha * exp(-lam / lam_ref))

    Physique :
      - lam grand (ressort rigide) → exp ≈ 0 → mu_eff = mu   (pas de penalite)
      - lam faible (ressort souple) → exp ≈ 1 → penalite alpha*mu
      - Sur beton (alpha=0.1) : lambda peu important
      - Sur sable (alpha=0.9) : lambda tres important
    """
    return mu * (1.0 - ALPHA_SOL[sol] * np.exp(-lam / LAM_REF))

Tabnine | Edit | Test | Explain | Document
def a_reelle(pneu, sol, ressort):
    """
    Acceleration reelle = min(a_moteur, mu_eff * g)
    - Limite moteur : a_moteur
    - Limite adherence : mu_eff * g (roues qui patinent)
    """
    lam = RESSORTS[ressort]["lam"]
    mu  = mu_eff(PNEUS[pneu]["mu"][sol], sol, lam)
    return min(A_MOTEUR, mu * g)

Tabnine | Edit | Test | Explain | Document
def vmax_virage(R, sol, pneu, ressort):
    """Vitesse max en virage (adherence + renversement)."""
    if not np.isfinite(R) or R > 300:
        return V_MAX
    lam = RESSORTS[ressort]["lam"]
    K   = RESSORTS[ressort]["K"]
    mu  = mu_eff(PNEUS[pneu]["mu"][sol], sol, lam)
    vg  = np.sqrt(mu * g * R)                          # limite adherence laterale
    A   = m * h / (K * l * R)
    disc = (2*h)**2 + 4 * A * g * l * R
    vr  = np.sqrt((-2*h + np.sqrt(disc)) / (2*A))      # limite renversement
    return min(vg, vr, V_MAX)
```

ANNEXES

Proramme Python

```python
# ============================================================
# 4. CIRCUIT — Catmull-Rom paramétrique ferme
# ============================================================
Tabnine | Edit | Test | Explain | Document
def catmull_rom_ferme(kp, n=50):
    """Spline Catmull-Rom fermee passant par tous les points de controle."""
    def tj(ti, a, b):
        return ti + ((b[0]-a[0])**2 + (b[1]-a[1])**2) ** 0.5

    def segment(p0, p1, p2, p3):
        t0 = 0.
        t1 = tj(t0, p0, p1); t2 = tj(t1, p1, p2); t3 = tj(t2, p2, p3)
        pts = []
        for t in np.linspace(t1, t2, n, endpoint=False):
            A1 = (t1-t)/(t1-t0)*np.array(p0) + (t-t0)/(t1-t0)*np.array(p1)
            A2 = (t2-t)/(t2-t1)*np.array(p1) + (t-t1)/(t2-t1)*np.array(p2)
            A3 = (t3-t)/(t3-t2)*np.array(p2) + (t-t2)/(t3-t2)*np.array(p3)
            B1 = (t2-t)/(t2-t0)*A1 + (t-t0)/(t2-t0)*A2
            B2 = (t3-t)/(t3-t1)*A2 + (t-t1)/(t3-t1)*A3
            pts.append((t2-t)/(t2-t1)*B1 + (t-t1)/(t2-t1)*B2)
        return np.array(pts)

    N   = len(kp)
    pts = np.vstack([segment(kp[(i-1)%N], kp[i], kp[(i+1)%N], kp[(i+2)%N]) for i in range(N)])
    return np.vstack([pts, pts[0]])

# Points de controle normalises [0,1]
_KP_NORM = np.array([[0.18,0.50],[0.12,0.65],[0.14,0.80],[0.25,0.88],
    [0.40,0.88],[0.55,0.85],[0.60,0.90],[0.65,0.83],
    [0.72,0.88],[0.78,0.82],[0.85,0.75],[0.88,0.65],
    [0.85,0.55],[0.80,0.45],[0.86,0.36],[0.80,0.27],
    [0.70,0.20],[0.55,0.15],[0.40,0.18],[0.30,0.25],
    [0.22,0.32],[0.20,0.42],
])

# Mise a l'echelle → longueur totale = 500 m
_tmp = catmull_rom_ferme(_KP_NORM.tolist())
_S   = 500 / np.linalg.norm(np.diff(_tmp, axis=0), axis=1).sum()
CIRCUIT = catmull_rom_ferme((_KP_NORM * _S).tolist())
N_PTS   = len(CIRCUIT)
```

```python
# Zones de sol par fraction de circuit parcouru
_ZONES = [
    (0.00, 0.18, "beton"),
    (0.18, 0.65, "terre"),
    (0.65, 0.85, "beton"),
    (0.85, 1.00, "beton"),
]
_ds   = np.linalg.norm(np.diff(CIRCUIT, axis=0), axis=1)
_frac = np.concatenate([[0], np.cumsum(_ds) / _ds.sum()])
SOLS  = [next(s for u0, u1, s in _ZONES if u0 <= f < u1) if f < 1 else _ZONES[-1][2]
         for f in _frac]

# Rayons de courbure vectorises
Tabnine | Edit | Test | Explain | Document
def rayons_courbure(pts):
    R = np.full(len(pts), np.inf)
    d1 = pts[1:-1] - pts[:-2]
    d2 = pts[2:]   - pts[1:-1]
    cross = np.abs(d1[:,0]*d2[:,1] - d1[:,1]*d2[:,0])
    chord = np.linalg.norm(pts[2:] - pts[:-2], axis=1)
    n1    = np.linalg.norm(d1, axis=1)
    n2    = np.linalg.norm(d2, axis=1)
    mask  = (cross > 1e-12) & (n1 > 1e-12) & (n2 > 1e-12)
    R[1:-1][mask] = n1[mask] * n2[mask] * chord[mask] / (2 * cross[mask])
    return R

RAYONS = rayons_courbure(CIRCUIT)
DS_ARR = np.linalg.norm(np.diff(CIRCUIT, axis=0), axis=1)  # distances inter-points
DS_CUM = np.concatenate([[0], np.cumsum(DS_ARR)])          # distance cumulee

```

ANNEXES

Proramme Python

```python
# ============================================================
# 5. SIMULATION
# ============================================================
Tabnine | Edit | Test | Explain | Document
def simuler(pneu, ressort):
    vmax_arr = np.array([vmax_virage(RAYONS[i], SOLS[i], pneu, ressort) for i in range(N_PTS)])
    acc_arr  = np.array([a_reelle(pneu, SOLS[i], ressort) for i in range(N_PTS)])

    v  = np.zeros(N_PTS)
    dt = np.zeros(N_PTS - 1)
    for i in range(1, N_PTS):
        v[i]    = min(np.sqrt(v[i-1]**2 + 2 * acc_arr[i] * DS_ARR[i-1]), vmax_arr[i])
        v_moy   = (v[i-1] + v[i]) / 2 or 1e-9
        dt[i-1] = DS_ARR[i-1] / v_moy
    return v, dt.sum()

# ============================================================
# 6. COMPARAISON DES CONFIGURATIONS
# ============================================================
resultats = {}
meilleure_config, meilleur_temps = None, np.inf

for pneu, ressort in itertools_product(PNEUS, RESSORTS):
    v_sim, t = simuler(pneu, ressort)
    resultats[(pneu, ressort)] = {"v_sim": v_sim, "temps": t}
    if t < meilleur_temps:
        meilleur_temps, meilleure_config = t, (pneu, ressort)

longueurs = {}
for i in range(N_PTS - 1):
    s = SOLS[i]
    longueurs[s] = longueurs.get(s, 0) + DS_ARR[i]
longueur_totale = sum(longueurs.values())

```

```python
# ============================================================
# 7. AFFICHAGE GRAPHIQUE
# ============================================================
fig = plt.figure(figsize=(15, 9))
fig.patch.set_facecolor("#1a1a2e")

ax_c = fig.add_axes([0.03, 0.10, 0.55, 0.78])   # circuit
ax_p = fig.add_axes([0.62, 0.54, 0.18, 0.43])   # camembert
ax_t = fig.add_axes([0.82, 0.54, 0.16, 0.43])   # tableau
ax_a = fig.add_axes([0.63, 0.20, 0.35, 0.36])   # vitesse animee

for ax in [ax_c, ax_p, ax_t, ax_a]:
    ax.set_facecolor("#16213e")
    for sp in ax.spines.values():
        sp.set_edgecolor("#0f3460")

TITRE_Y = 0.93
fig.text(0.305, TITRE_Y, "Circuit – Modelisation du buggy",
         color="white", fontsize=10, fontweight="bold", ha="center", va="top")
fig.text(0.710, TITRE_Y, "Repartition du circuit",
         color="white", fontsize=10, fontweight="bold", ha="center", va="top")
fig.text(0.900, TITRE_Y, "Configurations",
         color="white", fontsize=10, fontweight="bold", ha="center", va="top")
fig.suptitle("TIPE – Optimisation du buggy telecommande",
             color="white", fontsize=13, fontweight="bold", y=0.98)

# -- Circuit colore par sol --
segs = {s: ([], []) for s in COULEURS_SOL}
for i in range(N_PTS - 1):
    s = SOLS[i]
    segs[s][0].extend([CIRCUIT[i,0], CIRCUIT[i+1,0], None])
    segs[s][1].extend([CIRCUIT[i,1], CIRCUIT[i+1,1], None])
for s, (sx, sy) in segs.items():
    if sx:
        ax_c.plot(sx, sy, color=COULEURS_SOL[s], linewidth=3, solid_capstyle="round")

```

ANNEXES

Proramme Python

```python
# -- Annotations vmax virages --
pneu_b, res_b = meilleure_config
vmax_pts = precalc = np.array([
    vmax_virage(RAYONS[i], SOLS[i], pneu_b, res_b) * 3.6 for i in range(N_PTS)
])
vmax_pts[vmax_pts >= V_MAX * 3.6 - 0.1] = np.nan

x_min, x_max = CIRCUIT[:,0].min(), CIRCUIT[:,0].max()
y_min, y_max = CIRCUIT[:,1].min(), CIRCUIT[:,1].max()
in_v, zone, poses = False, [], [tuple(CIRCUIT[0])]

for i in range(N_PTS):
    if np.isfinite(vmax_pts[i]):
        zone.append(i); in_v = True
    else:
        if in_v and len(zone) >= 4:
            mid = zone[len(zone)//2]
            px, py = CIRCUIT[mid]
            ox, oy = 7, 7
            if px + ox + 18 > x_max: ox = -28
            if py + oy + 18 > y_max: oy = -14
            if py + oy < y_min + 18:  oy = 10
            if not any(np.hypot(px-ex, py-ey) < 35 for ex, ey in poses):
                ax_c.annotate(f"{vmax_pts[mid]:.1f} km/h", xy=(px, py),
                              xytext=(px+ox, py+oy), fontsize=8, color="white",
                              bbox=dict(boxstyle="round,pad=0.2", facecolor="#e94560",
                                        alpha=0.85, edgecolor="none"),
                              arrowprops=dict(arrowstyle="->", color="white", lw=0.8))
                poses.append((px, py))
        zone, in_v = [], False

ax_c.plot(*CIRCUIT[0], "go", markersize=10, zorder=5)
ax_c.annotate("Depart", xy=CIRCUIT[0], xytext=(CIRCUIT[0,0]+5, CIRCUIT[0,1]+7),
              color="lightgreen", fontsize=8,
              arrowprops=dict(arrowstyle="->", color="lightgreen", lw=0.8))
ax_c.axis("equal")
ax_c.grid(color="#0f3460", linestyle="--", alpha=0.5)
ax_c.tick_params(colors="#aaaaaa")
ax_c.set_xlabel("x (m)", color="#aaaaaa")
ax_c.set_ylabel("y (m)", color="#aaaaaa")
patches = [mpatches.Patch(color=c, label=s) for s, c in COULEURS_SOL.items()]
patches.append(mpatches.Patch(color="#e94560", label="vmax virage"))
ax_c.legend(handles=patches, facecolor="#0f3460", labelcolor="white", fontsize=8)
ax_c.text(0.02, 0.98, f"Distance : {longueur_totale:.0f} m\nMeilleur Temps : {meilleur_temps:.2f} s",
          transform=ax_c.transAxes, color="white", fontsize=10, fontweight="bold",
          va="top", ha="left",
          bbox=dict(facecolor="#0f3460", edgecolor="#e94560", boxstyle="round,pad=0.4", alpha=0.9))

```

```python
# -- Camembert --
lbl = list(longueurs.keys())
_, _, ats = ax_p.pie([longueurs[s] for s in lbl], labels=lbl,
                     colors=[COULEURS_SOL[s] for s in lbl],
                     autopct="%1.1f%%", startangle=90,
                     textprops={"color": "white", "fontsize": 8})
for at in ats: at.set_fontsize(7)
ax_p.text(0, -1.55, f"Total : {longueur_totale:.0f} m",
          ha="center", color="#aaaaaa", fontsize=8)

# -- Tableau --
ax_t.axis("off")
rows = []
for (pneu, res), r in resultats.items():
    star = "* " if (pneu, res) == meilleure_config else "  "
    rows.append([pneu, res, f"{star}{r['temps']:.2f}"])
tbl = ax_t.table(cellText=rows, colLabels=["Pneu", "Ressort", "Temps (s)"],
                 loc="center", cellLoc="center")
tbl.auto_set_font_size(False); tbl.set_fontsize(7.5)
for (row, col), cell in tbl.get_celld().items():
    is_best = row > 0 and rows[row-1][2].startswith("* ")
    cell.set_facecolor("#0f3460" if row == 0 else ("#1a4a1a" if is_best else "#16213e"))
    cell.set_text_props(color="white")
    cell.set_edgecolor("#1a1a2e")
tbl.scale(1, 1.3)

# -- Animation vitesse --
ax_a.set_title("Vitesse instantanee — Meilleure configuration", color="white", fontsize=9, fontweight="bold")
v_kmh = resultats[meilleure_config]["v_sim"] * 3.6
ax_a.set_xlim(0, DS_CUM[-1]); ax_a.set_ylim(0, v_kmh.max() * 1.15)
ax_a.set_xlabel("Distance (m)", color="#aaaaaa", fontsize=8)
ax_a.set_ylabel("km/h",         color="#aaaaaa", fontsize=8)
ax_a.tick_params(colors="#aaaaaa", labelsize=7)
ax_a.grid(color="#0f3460", linestyle="--", alpha=0.4)

line_a, = ax_a.plot([], [], color="#e94560", lw=1.5, animated=True)
dot_a,  = ax_a.plot([], [], "o", color="white", ms=5, animated=True)
txt_a   = ax_a.text(0.97, 0.93, "", transform=ax_a.transAxes, ha="right",
                    color="white", fontsize=9, fontweight="bold", animated=True)
car,    = ax_c.plot([], [], "o", color="#e94560", ms=9, zorder=10, animated=True)

STEP       = max(1, int(N_PTS / (meilleur_temps * 15)))
n_frames   = N_PTS // STEP + 1
interval_ms = meilleur_temps * 1000 / n_frames

```

ANNEXES

Proramme Python

```python
def update(f):
    i = min(f * STEP, N_PTS - 1)
    line_a.set_data(DS_CUM[:i+1], v_kmh[:i+1])
    dot_a.set_data([DS_CUM[i]], [v_kmh[i]])
    txt_a.set_text(f"{v_kmh[i]:.1f} km/h")
    car.set_data([CIRCUIT[i,0]], [CIRCUIT[i,1]])
    return line_a, dot_a, txt_a, car

ani = FuncAnimation(fig, update, frames=n_frames, interval=interval_ms, blit=True)

fig.text(0.820, 0.11,
         f"Meilleure config : Pneu {pneu_b} | Ressort {res_b} | {meilleur_temps:.2f} s",
         color="white", fontsize=9, fontweight="bold", ha="center",
         bbox=dict(facecolor="#0f6016", edgecolor="#60e945", boxstyle="round,pad=0.4"))

plt.show()
```

ANNEXES

Ordre de grandeur

Voiture	Sillion	Lisse
Sable	0,2/0,4	0,1/0,2
Beton	0,7/1	0,9/1,2
Terre	0,4/0,7	0,2/0,4

	K	λ
Voiture	15000/30000	1500/3000