

RAPPORT DU PROJET

VITIS

- 2024 / 2025 -

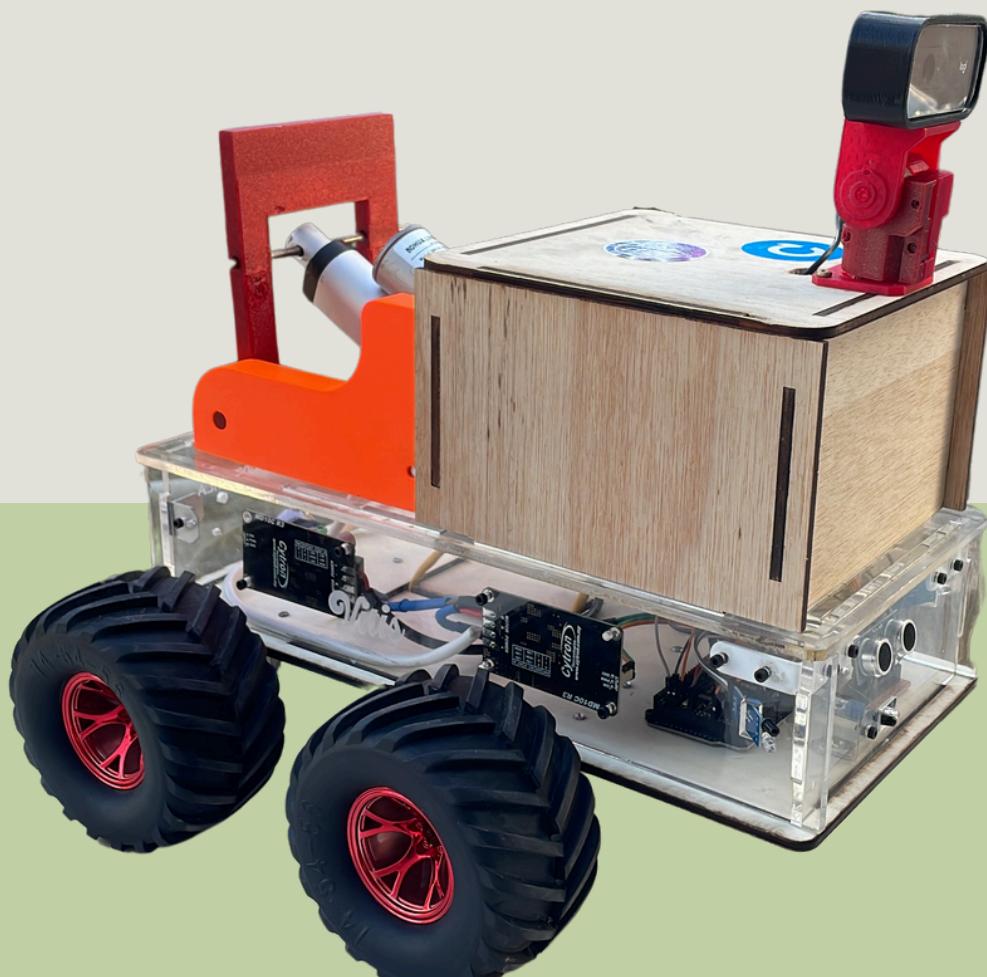


Figure 1 - Image de Vitis

Créé par

*Demoron Tanguy
Chavant Arthur*

tanguy.demoron@etu.unice.fr

arthur.chavant@etu.unice.fr

<https://github.com/arthchav/vitis>

INTRODUCTION

Présentation

Vitis est un robot autonome viticole qui a pour mission le désherbage sous le rang de la vigne. Équipé de 4 roues motrices et 100% électrique, il sera le meilleur ami des petits comme des gros exploitants pour garder une vigne toujours propre. Vitis peut s'occuper de tous types de parcelles même les plus difficiles.



Figure 2 - Vitis de profil

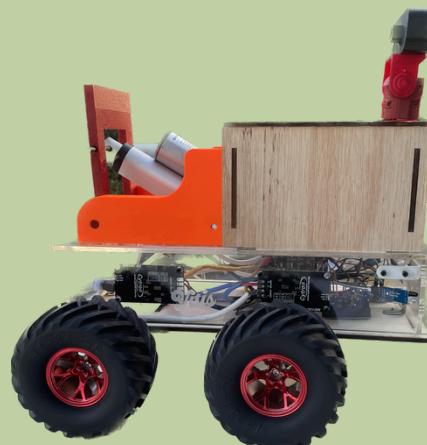


Figure 3 - Vitis de coté



La caméra

La caméra sur le robot viticole permet de détecter les maladies des feuilles, comme le mildiou, grâce à l'analyse d'images. Elle offre un diagnostic rapide et précis, aidant à prévenir la propagation des maladies tout en réduisant l'utilisation de traitements chimiques.



La carte Nvidia

La carte Nvidia permet de traiter rapidement les images capturées par la caméra grâce à ses capacités de calcul GPU, essentielles pour exécuter des algorithmes d'intelligence artificielle. Elle optimise la reconnaissance des maladies en temps réel, rendant le robot plus efficace et autonome.



Le porte-outils

Le porte-outils équipé d'un vérin permet au robot de manipuler divers équipements nécessaires aux travaux viticoles, comme le désherbage ou la pulvérisation. Il offre une polyvalence accrue, permettant d'adapter les interventions en fonction des besoins de la vigne.



Les capteurs

Les capteurs du robot, incluant deux infrarouges et un ultrason, assurent la détection des obstacles pour une navigation sécurisée dans les rangs de vignes. Ils permettent au robot de se déplacer de manière autonome tout en évitant les collisions, garantissant une opération fluide et précise.

LES OBJECTIFS

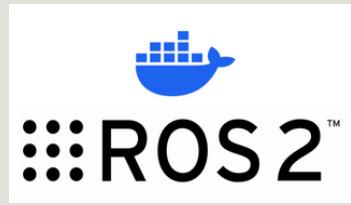


Figure 4 - ros2

Navigation autonome : Développer un algorithme basé sur le Q-learning, couplé à un filtre de Kalman, pour permettre au robot de se déplacer de manière précise et sécurisée dans les rangs de vignes tout en s'adaptant à son environnement.



Figure 5 - GPS



Figure 6 - Feuille de vigne

Création d'un environnement ROS 2 : Structurer et organiser le développement logiciel du robot pour centraliser les fonctionnalités et assurer une communication efficace entre les différents composants.

Reconnaissance de maladies : Capturer des images des feuilles de vigne à l'aide de la caméra, les analyser via un modèle de classification pour identifier et différencier les feuilles saines des feuilles malades.

Notre robot viticole vise à répondre aux besoins des vignerons grâce à trois objectifs clés. D'abord, l'environnement ROS 2 permet d'organiser et de gérer efficacement les composants du robot. Ensuite, la navigation autonome avec un algorithme de Q-learning et un filtre de Kalman permet au robot de se déplacer précisément dans les rangs, sans intervention humaine. Enfin, la reconnaissance des maladies des feuilles grâce à la caméra permet de détecter rapidement les infections, limitant leur propagation et réduisant l'usage de produits chimiques. Ces priorités ont conduit à mettre de côté, pour l'instant, le développement du porte-outils, afin de concentrer nos efforts sur ces fonctionnalités essentielles. Ces avancées rendent la viticulture plus simple, précise et durable.



Figure 7 - Vitis dans la vigne

ROS2

La création d'un environnement ROS 2 pour notre robot viticole constitue une étape fondamentale dans la structuration et l'organisation des fonctionnalités du robot. ROS 2 (Robot Operating System) est une plateforme open source conçue pour le développement de logiciels robotiques modulaires. Cette approche permet d'intégrer les différents composants du robot de manière harmonieuse et de faciliter leur interaction.



Figure 8 - Ros2

Étapes de la mise en place de l'environnement ROS 2

a) Configuration de base

- Installation : ROS 2 Humble a été installé sur Ubuntu 22.04 avec les dépendances nécessaires.
- Workspace : Création d'un espace de travail pour organiser les fichiers et packages du projet.
- Paramétrage initial : Configuration des nœuds, des « topics » pour la communication et des paramètres globaux.

b) Implémentation des nœuds

Les fonctionnalités principales ont été développées sous forme de nœuds :

- Navigation : Gestion du mouvement grâce aux capteurs et à l'algorithme de Q-learning, avec un filtre de Kalman pour affiner les mesures de l'IMU.
- Vision : Traitement des images pour détecter les maladies des feuilles.
- Contrôle : Coordination des capteurs et des actionneurs pour exécuter les commandes (direction, vitesse).

c) Communication entre les nœuds

La communication est réalisée via des « topics » et des « services » :

- Les capteurs publient leurs données sur des topics dédiés.
- Le nœud de navigation utilise ces données pour ajuster la trajectoire en temps réel.
- Les images de la caméra sont transmises au nœud de vision pour traitement.

d) Tests et validation

Chaque nœud a été testé individuellement avant intégration globale. Ces tests ont vérifié la stabilité, la précision des données échangées et la synchronisation des fonctionnalités.

```
❶ topic_list.py > ...
1 TOPIC_LIST = {
2     "/cmd_vel": {
3         "type": "geometry_msgs/msg/Twist",
4         "description": "Commandes de vitesse pour le différentiel (x, y, z)."
5     },
6     "/odometry": {
7         "type": "nav_msgs/msg/Odometry",
8         "description": "Informations d'odométrie du robot (position, orientation)."
9     },
10    "/navigation/goal": {
11        "type": "geometry_msgs/msg/PoseStamped",
12        "description": "Position et orientation cibles pour la navigation autonome."
13    },
14    "/navigation/status": {
15        "type": "std_msgs/msg/String",
16        "description": "État du système de navigation (par ex. 'En cours', 'Terminé')."
17    },
18    "/sensors/ultrasonic": {
19        "type": "sensor_msgs/msg/Range",
20        "description": "Distance détectée par le capteur ultrason."
21    },
22    "/sensors/infrared_left": {
23        "type": "sensor_msgs/msg/Range",
24        "description": "Distance détectée par le capteur infrarouge gauche."
25    },
26    "/sensors/infrared_right": {
27        "type": "sensor_msgs/msg/Range",
28        "description": "Distance détectée par le capteur infrarouge droit."
29    },
30    "/imu/data": {
31        "type": "sensor_msgs/msg/Imu",
32        "description": "Données IMU : accélération, orientation, vitesse angulaire."
33    }
}
```

Figure 9 - Liste des topics

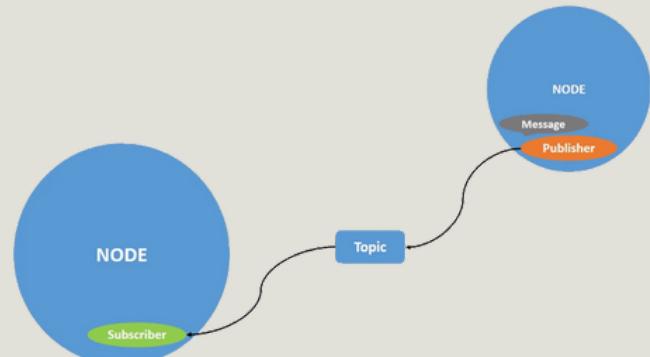


Figure 10 - Communication entre Nodes

Schéma ROS 2 : organisation et fonctionnement sur notre robot viticole

1. Organisation des nœuds et topics

a) Nœuds principaux

- Nœud de navigation :
- Responsable de l'autonomie du robot. Il reçoit les données des capteurs (ultrasons, infrarouges et IMU) pour générer une trajectoire optimisée. Il publie les commandes de vitesse et de direction sur les topics dédiés.
- Nœud de vision :
- Gère les images capturées par la caméra montée à l'avant. Il effectue l'analyse des feuilles de vignes pour détecter les maladies et publie les résultats de classification sur un topic spécifique.
- Nœud de contrôle :
- Centralise les commandes des capteurs et des actionneurs. Ce nœud envoie les instructions au moteur différentiel et au vérin pour le mouvement et les outils.

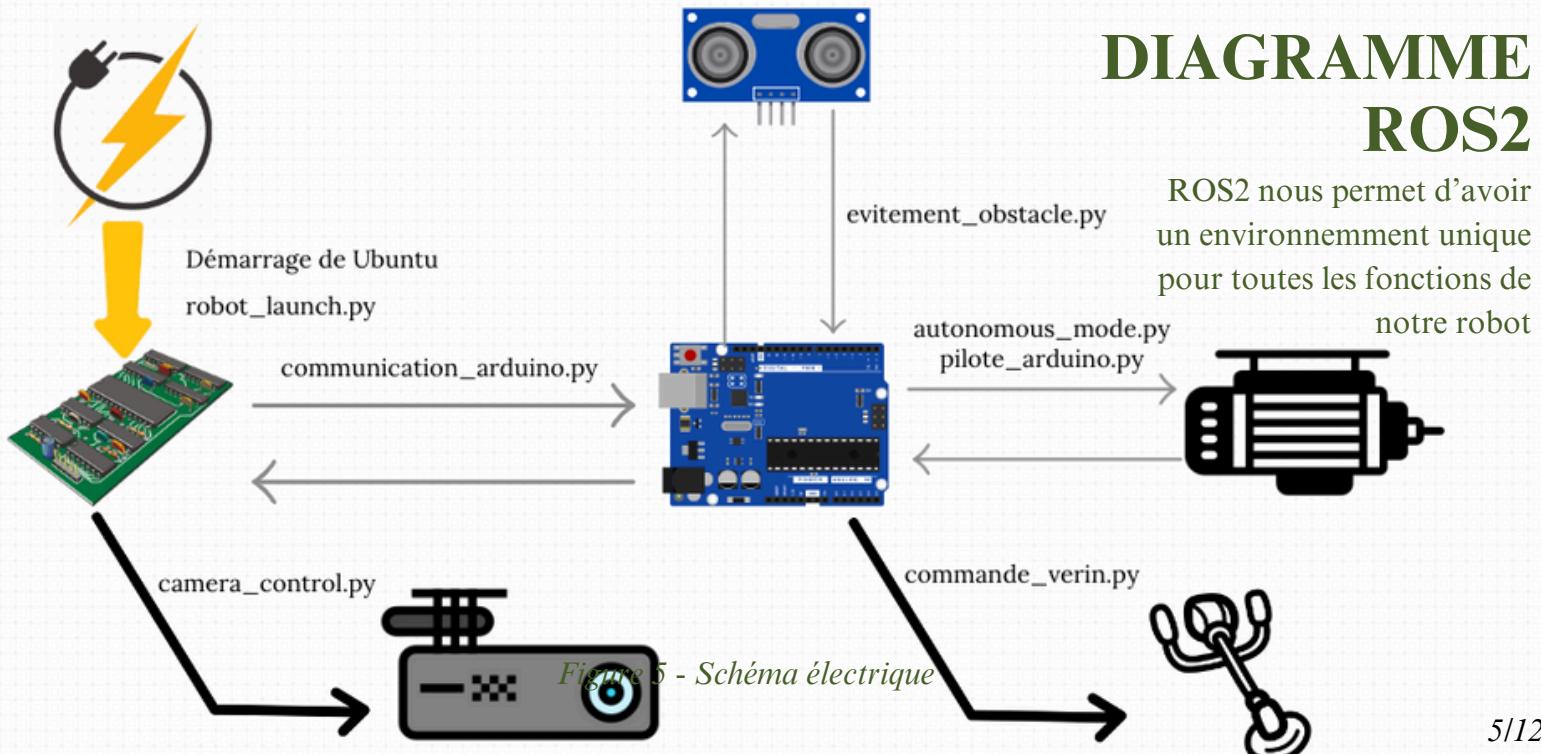
b) Topics utilisés

- /cmd_vel : Topic utilisé pour transmettre les commandes de vitesse et de direction du nœud de navigation au nœud de contrôle.
- /sensor_data : Topic où sont publiées les données brutes des capteurs (ultrasons, infrarouges, IMU).
- /image_raw : Topic sur lequel la caméra publie les images en temps réel.
- /disease_detection : Topic où le nœud de vision publie les résultats de la classification des maladies (exemple : sain/malade).

2. Fichier de lancement (Launch File)

- Le fichier lance automatiquement les nœuds de navigation, de vision, et de contrôle.
- Des fichiers YAML sont chargés pour définir les paramètres des capteurs (exemple : fréquence d'échantillonnage, seuils de détection).
- La caméra, les capteurs, et les actionneurs sont initialisés via leurs ports respectifs.

Le fichier de lancement est exécuté à l'aide de la commande : ros2 launch vitis_robot start_robot.launch.py



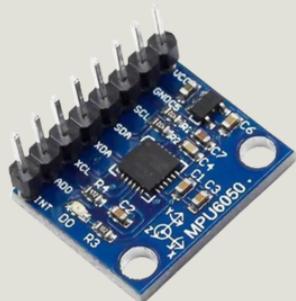
KALMAN



Figure 11 - Image de Vitis en pente

Le filtre de Kalman intégré au robot viticole utilise les données du capteur MPU-6050 pour estimer précisément l'orientation du robot. En combinant les mesures de l'accéléromètre et du gyroscope, il réduit le bruit et améliore la stabilité du robot, ce qui est essentiel pour une navigation autonome fiable en viticulture.

MPU-6050



$$Q_{gyro} = (\text{BruitRMS})^2$$

$$\text{BruitRMS} = 0.05/\text{s}$$

$$Q_{angle} = 0.01$$

$$Q = \begin{bmatrix} Q_{angle} & 0 \\ 0 & Q_{gyro} \end{bmatrix}$$

GYROSCOPE NOISE PERFORMANCE		FS_SEL=0					
Total RMS Noise		DLPFCFG=2 (100Hz)					
Low-frequency RMS noise		Bandwidth 1Hz to 10Hz					
Rate Noise Spectral Density		At 10Hz	0.05	0.033	0.005	%/s-rms	%/s-√Hz

```
Adafruit_MPU6050 mpu;
FilterKalman kalmanPitch(0.001, 0.0025, 0.025);
```

.....> Avec kalmanPitch(Qangle, Qgyro, Rmeasured)

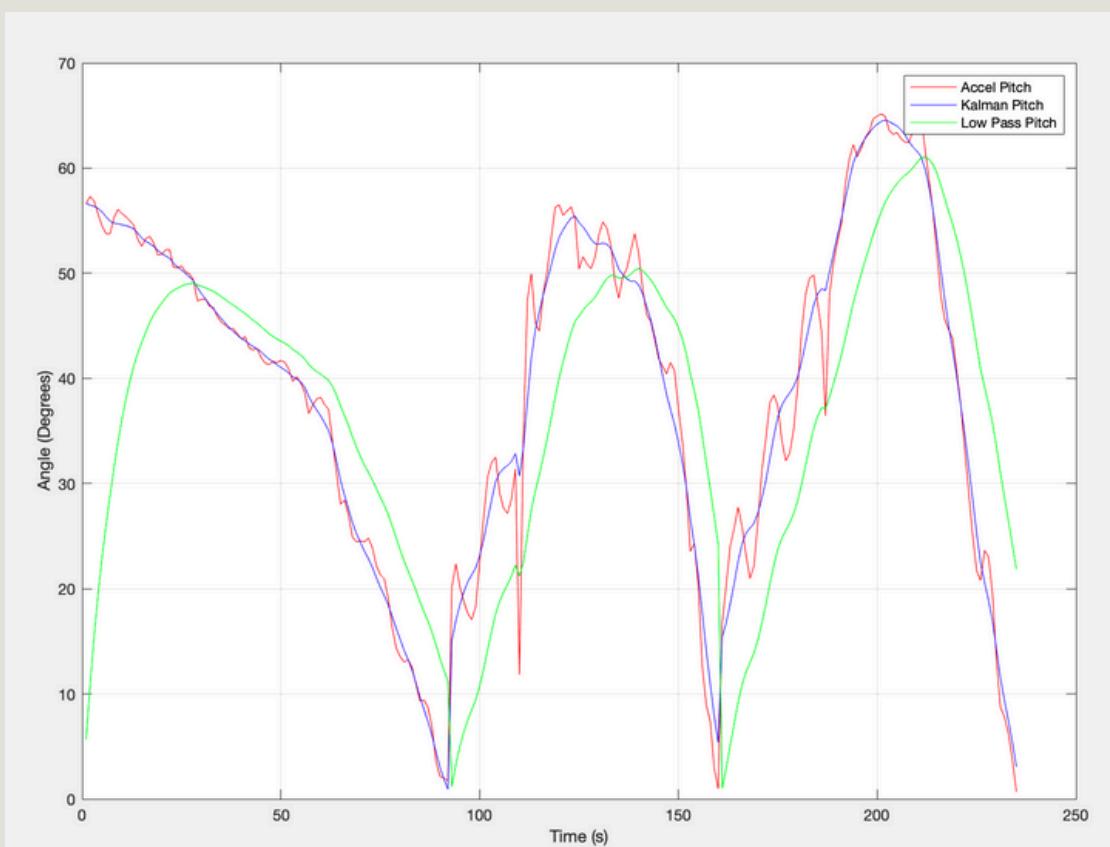


Figure 12 - Courbes de Kalman VS Filtre passe bas

NAVIGATION INTELLIGENTE

Dans ce projet, nous avons cherché à améliorer les déplacements de VITIS en lui fournissant une stratégie de navigation intelligente, au lieu de le laisser se reposer uniquement sur ses capteurs.

Ces derniers, bien qu'efficaces pour éviter les obstacles, ne suffisent pas pour gérer des scénarios complexes, comme les virages serrés dans les rangées de vignes.



Figure 13 - Illustration de Vitis et ses capteurs



Figure 14 - Drone qui cartographie la vigne

La première étape consiste à cartographier la vigne à l'aide d'un drone ou directement par le viticulteur, ce qui permet de recueillir un plan précis des rangées et des obstacles présents. Ce plan est ensuite converti en une grille 2D, où chaque case représente une portion de terrain. Cette grille sert de base pour simuler les déplacements de VITIS et concevoir une stratégie d'exploration optimale, prenant en compte les contraintes spécifiques du terrain.

Nous avons commencé avec le problème du Cliff Walking, vu en TP de "Reinforcement Learning". L'objectif est de guider un agent sur une grille 2D, du point de départ (S) à l'objectif (G), tout en évitant une zone dangereuse appelée "falaise".

L'agent peut se déplacer avec quatre actions : gauche, droite, haut et bas. Chaque déplacement coûte une petite pénalité ($R = -1$), et tomber dans la falaise inflige une forte pénalité ($R = -100$). Ce problème nous sert de base pour développer des stratégies de navigation plus complexes pour VITIS.

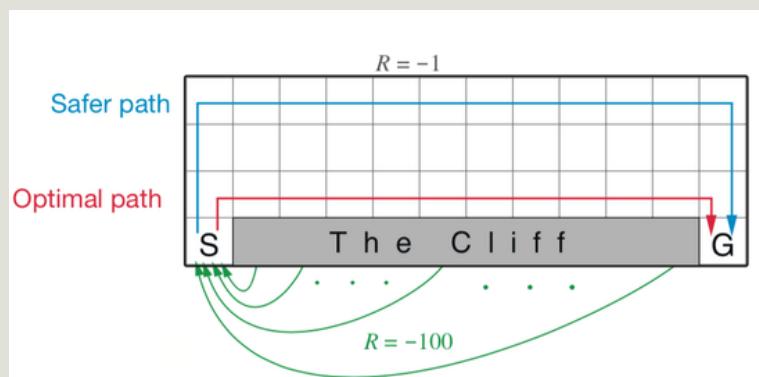


Figure 15- Schéma du problème Cliff Walking

Nous utilisons un algorithme Q-Learning avec une Q-Table pour prendre les décisions de l'agent. Chaque ligne du tableau correspond à un état spécifique, et chaque colonne représente une action possible (par exemple, aller à gauche, droite, haut ou bas). Les valeurs dans le tableau indiquent l'utilité estimée de prendre une action dans un état donné.

$$Q(s, a) = Q(s, a) + \alpha(r + \gamma \max Q(s', a') - Q(s, a))$$

Figure 17 - Formule d'actualisation de $Q(s, a)$

	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
s_0	490	490	490	700	490	490	490	490
s_1	490	490	700	490	490	490	490	490
s_2	490	490	490	700	700	490	490	490
s_3	490	490	700	1000	700	700	490	490
s_4	490	490	490	700	700	700	490	490

Figure 16 - Q-table avec toutes les valeurs $Q(s, a)$

La nouvelle valeur $Q(s, a)$ est calculée en combinant l'ancienne valeur avec la récompense immédiate r et la meilleure valeur future $\max Q(s, a)$. Cela permet à l'agent d'apprendre à choisir des actions optimales sur le long terme.

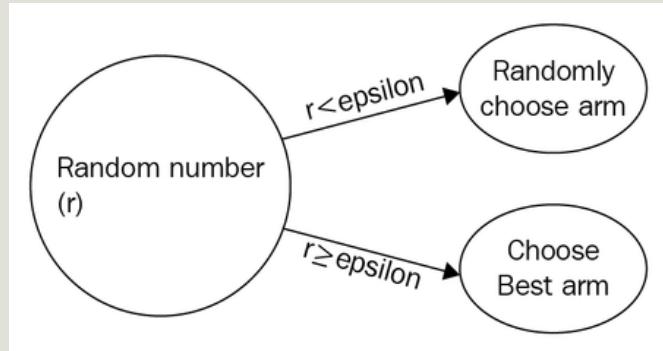


Figure 18 - Illustration de la politique ϵ -greedy

Les paramètres α (taux d'apprentissage) et γ (facteur de discount) de la formule, permettent de contrôler la vitesse d'apprentissage et l'efficacité de l'algorithme de Q-Learning.

- α contrôle la vitesse à laquelle les valeurs $Q(s, a)$ sont mises à jour : une valeur élevée permet un apprentissage rapide, tandis qu'une valeur faible stabilise les décisions.
- γ pondère l'importance des récompenses futures par rapport aux immédiates, influençant la stratégie à long terme.

Dans ce projet, l'objectif est de réduire progressivement α et ϵ pour permettre à l'agent de converger vers un chemin optimisé et définitif, basé sur les connaissances acquises lors de l'exploration initiale.

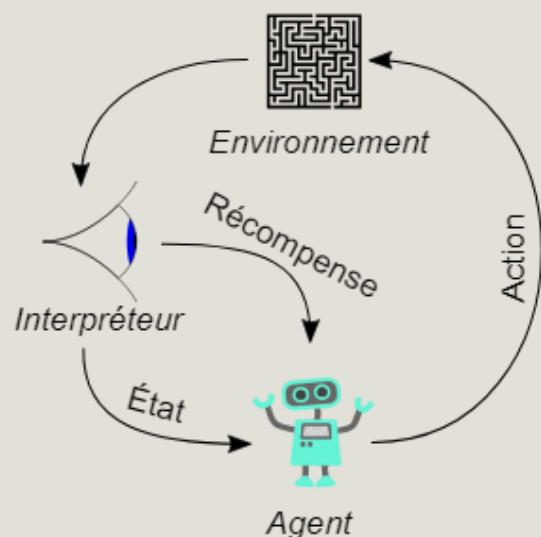


Figure 19 - Principe de l'apprentissage par renforcement

Après avoir recréé l'algorithme du Cliff Walking, nous avons retiré les "falaises" pour transformer la grille en un espace ouvert. Dans cette nouvelle version, l'objectif de l'agent (représenté par la boule bleue) est de trouver le chemin le plus rapide vers la case verte, représentant la sortie. Cette simplification nous a permis de mieux analyser les performances de l'agent dans un environnement dépourvu de pénalités extrêmes, tout en évaluant sa capacité à optimiser son déplacement dans un espace vide.

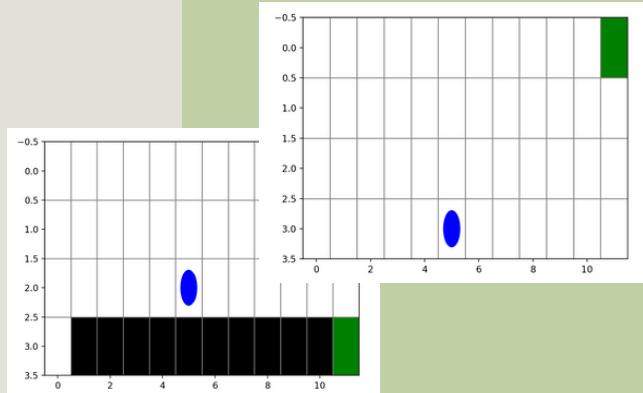
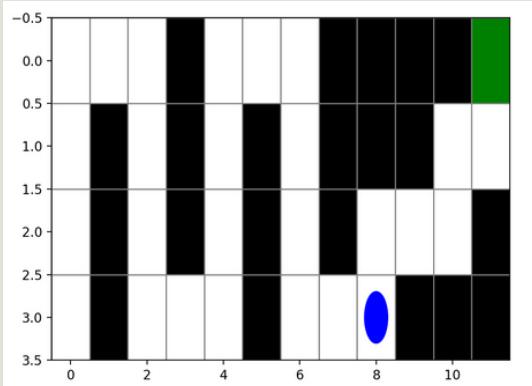


Figure 20 - Adaptation du Cliff Walking à notre projet

Dans cette étape, les obstacles statiques ont été disposés de manière à simuler la structure d'une vigne avec ses rangées et virages caractéristiques. Cette configuration a permis d'optimiser les paramètres clés de l'apprentissage tels que alpha (taux d'apprentissage), gamma (facteur de discount), et epsilon (exploration/exploitation), pour garantir une navigation précise et efficace dans ce type d'environnement.

Figure 21 - Optimisation des paramètres

Nous avons développé un programme qui génère un chemin aléatoire dans la grille, simulant le tracé d'un agriculteur ou d'un drone. Ce chemin sert de base pour tester la navigation de VITIS, en reproduisant les scénarios réels où l'agriculteur ou le drone définirait des itinéraires à suivre dans la vigne.

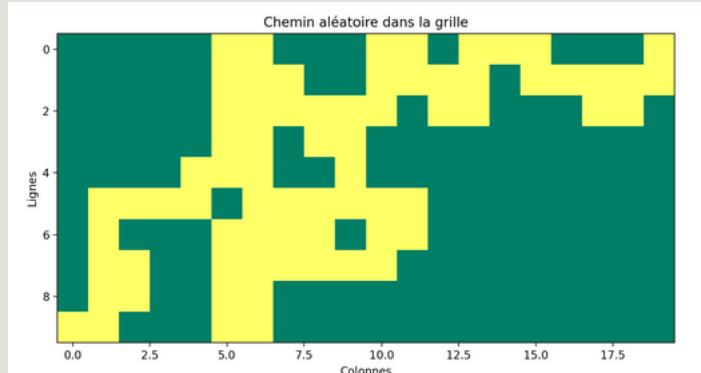
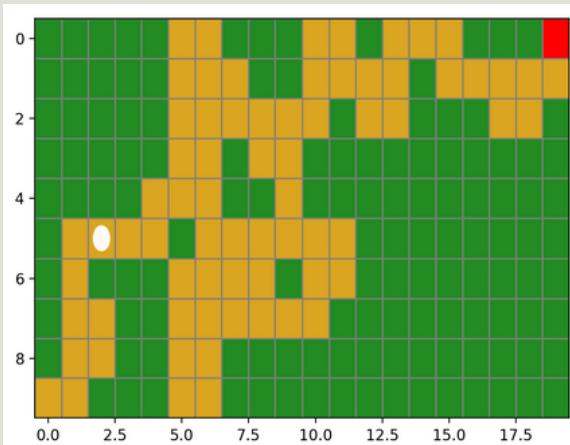


Figure 22 - Grille générée aléatoirement

Après l'entraînement via le Reinforcement Learning, le programme est capable d'identifier le chemin optimal pour atteindre la sortie.

Les valeurs de la Q-Table se stabilisent, guidant les décisions de l'agent pour minimiser les pénalités et maximiser les récompenses dans l'environnement.

Figure 23 - Chemin trouvé par le programme

DETECTION DE MALADIES PAR CAMERA

Les maladies comme le mildiou ou l'oïdium affectent gravement la qualité et le rendement des vignes. Aujourd'hui, leur détection repose sur des inspections manuelles, souvent longues et peu précises. On veut automatiser cette tâche grâce à une caméra embarquée et des algorithmes d'intelligence artificielle. Ce système capture des images des feuilles, détecte les signes visuels de maladies (taches, décolorations) et les classe en « saines » ou « malades ». Cette approche permet une détection précoce, réduit les traitements chimiques et contribue à une viticulture plus durable et efficace.



Mildiou



Oïdium



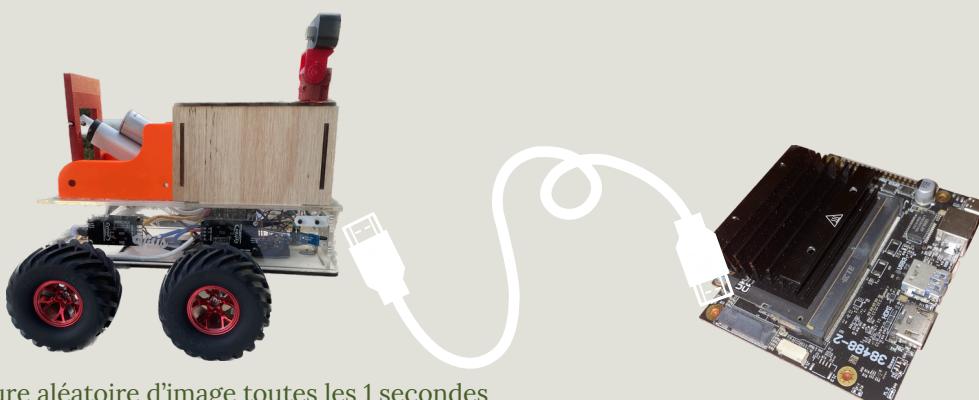
Flavescence dorée



Excoriose

Avant l'analyse des images capturées par la caméra du robot, un pré-traitement est effectué pour garantir des données exploitables et fiables. Ce processus améliore la qualité des images et optimise les performances de l'algorithme d'analyse :

1. Redimensionnement : Les images sont ajustées à une taille standard afin de réduire la charge de calcul et d'assurer une compatibilité avec le modèle d'apprentissage automatique.
2. Réduction du bruit : Des filtres sont appliqués pour supprimer les artefacts et les éléments parasites qui pourraient perturber l'analyse, comme les ombres ou le flou.
3. Ajustement des contrastes et de la luminosité : Les niveaux de contraste et de luminosité sont normalisés pour faire ressortir les détails visuels importants, notamment les taches ou décolorations indicatives de maladies.
4. Conversion de couleur : Dans certains cas, les images sont converties en niveaux de gris ou dans d'autres espaces colorimétriques (comme HSV) pour isoler des caractéristiques pertinentes.



Capture aléatoire d'image toutes les 1 secondes

Pré-traitement des images

Création d'un dossier pour la classification

Fonctionnement de l'algorithme

• Capture d'images

La caméra du robot capture des images aléatoirement toutes les secondes lorsqu'il est en déplacement dans le vignoble. Ces images sont stockées avec des métadonnées importantes, comme l'emplacement GPS et l'heure de capture, pour permettre une analyse contextualisée.

• Pré-traitement des images

Avant d'être analysées par le modèle, les images passent par plusieurs étapes de pré-traitement :

Redimensionnement : Chaque image est ajustée à une taille standard (par exemple, 224x224 pixels) pour s'adapter à l'entrée du modèle.

Filtrage du bruit : Des techniques de réduction de bruit éliminent les interférences telles que les ombres ou les variations de lumière.

Amélioration du contraste : Les feuilles sont mises en évidence pour accentuer les caractéristiques visuelles pertinentes.

Normalisation : Les valeurs de pixels sont transformées pour se situer dans une plage comprise entre 0 et 1, assurant une stabilité dans le traitement.

• Analyse par le réseau de neurones convolutif (CNN)

Couches convolutionnelles : Ces couches identifient les motifs spécifiques liés aux maladies, comme les taches ou les décolorations.

Couches de pooling : Elles réduisent la taille des données tout en conservant les caractéristiques importantes, optimisant ainsi la performance.

Couches pleinement connectées : Ces couches finales classifient les images comme étant de feuilles saines ou malades.

• Décision et étiquetage

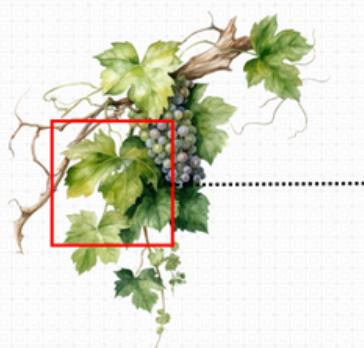
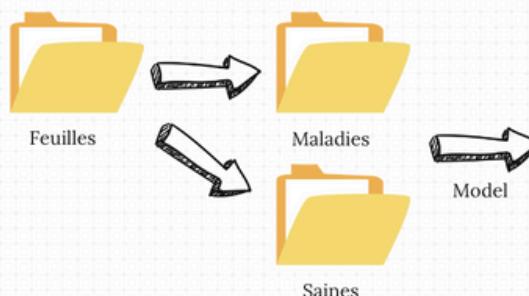
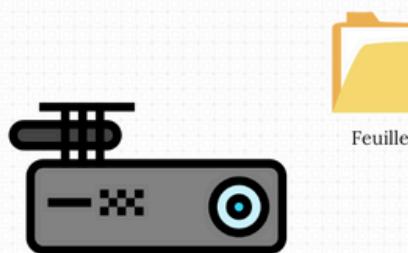
Après traitement par le modèle, une probabilité est associée à chaque classe (par exemple, saine ou malade). Si la probabilité dépasse un certain seuil, l'image est classée en conséquence.

• Enregistrement des résultats

Les résultats sont enregistrés avec :

- L'étiquette (sain ou malade)
- Le taux de confiance du modèle

Diagramme classification



Pré-traitement
Redimensionnement, Normalisation

Layer / opération	Détail / Shape	Nombre
conv1_1 / Conv2D	(None, 368, 160, 32)	64
max_pool1 / MaxPool2d	(None, 18, 80, 32)	2
conv1_2 / Conv2D	(None, 18, 80, 32)	64
max_pool2 / MaxPool2d	(None, 9, 40, 32)	2
conv2_1 / Conv2D	(None, 9, 40, 32)	768,000
max_pool3 / MaxPool2d	(None, 2, 20, 32)	4
conv2_2 / Conv2D	(None, 2, 20, 32)	1,792,000
dense1 / Dense	(None, 64)	1,792,000
dense1_1 / Dropout	(None, 64)	0
dense2 / Dense	(None, 32)	64
dense2_1 / Dropout	(None, 32)	0
dense3 / Dense	(None, 1)	64

Total param : 4,124,400 (38.42 Mo)

Trainable param : 4,032,401 (38.42 Mo)

Modèle CNN :
Apprentissage (Forward Pass)
Calcul des Erreurs (Loss Function)
Mise à Jour des Poids (Backpropagation)
Validation sur Jeu de Données

Modèle Entraîné :
Couches Convolutionnelles | (Extraction des Caractéristiques)
Couches de Pooling | (Réduction des Dimensions)
Couches Fully Connected | (Consolidation en Prédiction)
Sortie (Classe : Sain ou Malade)



CONCLUSION



Au terme de ces trois années, le projet robot VITIS a permis de développer un prototype presque fonctionnel d'assistance à la viticulture autonome. Grâce à l'intégration de capteurs, de la plateforme ROS et d'un système de reconnaissance des maladies des feuilles, le robot pourrait répondre aux besoins des viticulteurs en optimisant les tâches liées à la gestion de la vigne.

Les défis techniques rencontrés ont permis de renforcer nos compétences en ROS, IA, C++, Linux..., en essayant de mettre au point un prototype viable. Ce projet est loin d'être aboutit mais nous a permis de travailler sur un secteur que nous apprécions tout les deux : la robotique agricole. Domaine dans lequel nous continuerons de travailler en stage.

En conclusion, ce projet nous a permis de développé nos compétences dans un secteur qui nous passionne et dans lequel nous voulons nous orienter.

PERSPECTIVES



Les perspectives de développement pour le robot VITIS sont vastes et prometteuses. En améliorant la reconnaissance d'images pour une détection encore plus précise des maladies des vignes et en affinant les algorithmes de navigation, le robot pourrait devenir un outil incontournable pour les viticulteurs. L'intégration de nouvelles fonctionnalités, comme la gestion de la récolte ou l'optimisation des interventions phytosanitaires, renforcerait encore sa valeur ajoutée.

BIBLIOGRAPHIE



- <https://vitibot.fr/>
- <https://www.naio-technologies.com/>
- <https://github.com/dusty-nv/jetson-inference#hello-ai-world>
- <http://users.polytech.unice.fr/~pmasson/Enseignement-arduino.htm>
- <https://www.youtube.com/@NVIDIADeveloper>
- <https://exxact-robotics.com/en/>
- <https://www.ros.org>
- <https://www.nvidia.com/en-us/>

LIENS

GITHUB:

<https://github.com/arthchav/vitis>

YOUTUBE:

<https://www.youtube.com/playlist?list=PLRQ-Vj44Z8fejHV95dyqV7iqJ9uM6nV9p>