TER Slot Car Autonome

Journée du 25/03/25

Il n'y a pas eu de rapport de la séance précédente, car celle-ci était principalement consacrée aux tests, au débogage et à la discussion sur le master IoT avec vous.

Aujourd'hui, il y a eu beaucoup de progrès, notamment en lissant une bonne partie des problèmes liés à l'algorithme de Q-Learning.

Tout d'abord, nous avions un problème avec les courbes : elles entraînaient une chute de tension et, à faible vitesse, cela mettait la voiture à l'arrêt.
La solution a été d'ajuster la récompense : à faible vitesse, toute action autre qu'accélérer entraîne désormais une pénalité.

Le deuxième problème était que l'algorithme ne pénalisait pas correctement les sorties de circuit. Lorsque la voiture sortait, la tension se stabilisait autour de 14.7V, mais l'état puni était celui correspondant à 14V. Or, une fois la voiture hors du circuit, le système ne peut plus contrôler son état, et aucune action ne peut corriger la situation.
La solution a été d'enregistrer l'état précédent : lorsqu'une sortie est détectée, c'est cet état qui est pénalisé, plutôt que celui à 14V.

Nous avons également expérimenté avec certains paramètres du système.

- Le "step" (ou pas) d'incrémentation/décrémentation de la vitesse

Ce paramètre est crucial pour le modèle. S'il est trop petit, l'instabilité de la mesure de tension fait que l'accélération n'apporte pas de grande récompense. S'il est trop grand, la voiture accélère trop vite et perd en contrôle. Actuellement, un pas de 3 à 5 % semble bien fonctionner.

- Le nombre d'états

Actuellement, nous avons 14 états, soit un état par volt. Nous nous demandons si un découpage plus fin (plus d'états) permettrait un

meilleur contrôle de la voiture. Toutefois, l'instabilité des mesures pourrait rendre cette approche moins pertinente. De plus, plus d'états signifient plus d'informations à traiter et un temps d'entraînement plus long.

En l'état, la voiture parvient déjà très bien à trouver la vitesse moyenne optimale pour ne pas sortir de la piste.

Nouvelle expérimentation

Nous avons décidé de construire une piste beaucoup plus grande, avec une longue ligne droite (en fusionnant les deux pistes). L'objectif était de voir si le système parvenait à accélérer sur les lignes droites. Il semble que oui : la voiture est légèrement plus rapide sur ces portions, mais on pense qu'elle pourrait encore aller plus vite.

Prochaines étapes

Nous aimerions :

- Expérimenter davantage avec le nombre d'états.

- Mettre en place un système de sauvegarde de la Q-table, pour éviter de devoir réentraîner le modèle à chaque session.

- Implémenter un système d'arrêt : pour faire en sorte que, lorsqu'une sortie est détectée, la pénalité ne s'applique qu'une seule fois, et la voiture attend une confirmation de l'utilisateur avant de redémarrer.

- Prendre en compte le temps afin d'évaluer si la voiture peut approximer sa position. Cependant, on se pose des questions sur l'utilité de cette approche, car les temps de tour varient beaucoup en fonction de la vitesse et des sorties de piste. De plus, cela rallongerait encore le temps d'entraînement.

Résultats

À la fin de la séance, la voiture se comporte nettement mieux que la semaine dernière, et presque mieux que nous !

Temps sur 20 tours :

Nous (Javier) : 1 min 15 s

Q-Learning : 1 min 30 s

Journée du 11/03/25

Afin de rattraper tout ce qui a été fait lors des séances où aucun rapport n'a été rédigé, cette partie servira de récapitulatif.

Au départ, nous avons essayé de tout faire fonctionner sur un seul Raspberry Pi. Cependant, limités par nos connaissances en électronique, nous avons "perdu" une séance ainsi qu'un certain temps de travail à la maison, car nous n'avons pas réussi à trouver un circuit permettant de mesurer et de contrôler le courant électrique simultanément.

Malgré cela, nous avons accompli plusieurs tâches importantes :

- **Recueil de fiches techniques sur les composants utilisés**
  Cela nous a permis de mieux comprendre leur fonctionnement et d'identifier certaines failles dans notre circuit. Ces documents pourraient également être utiles à des élèves souhaitant approfondir l'aspect électronique du projet.

- **Création de scripts de configuration**
  Nous avons mis en place des scripts afin de simplifier l'installation de nouveaux Raspberry Pi. Cette automatisation s'est déjà révélée utile et nous a fait gagner du temps.

Nous avons envisagé une solution consistant à utiliser un Raspberry Pi pour mesurer la tension et un autre pour la contrôler. Suite à l'appel du vendredi dernier, nous avons décidé de tester cette approche.

Avant la séance du 11/03/25 on a:

> Nous avons configuré les Raspberry Pi pour communiquer via Ethernet en leur attribuant des IP fixes. Pour ce faire, nous avons modifié la configuration de l'interface Ethernet en ajoutant les lignes suivante au fichier `/etc/systemd/network/eth0.network` :
>
> ```
> [Match]
> Name=eth0
>
> [Network]
> Address=192.168.2.2/24
> ```

Nous avons également **refactorisé une bonne partie du code** pour mieux séparer les tâches. La nouvelle répartition est la suivante :

Raspberry 1 (Capteur - 192.168.2.2)

- `server.py`
- `capteur.py`
- `templates/` (fichiers `.html` pour l'interface web)

Raspberry 2 (Contrôleur - 192.168.2.1)

- `qlearn.py`
- `controleur.py`

Dans cette configuration, le capteur mesure la tension et l'envoie au serveur (qui tourne sur le même Raspberry Pi). Celui-ci transmet ensuite l'information au Raspberry Pi contrôleur et met à jour la page web.

Par ailleurs, nous avons également créé un fichier `tests/` contenant des scripts simples permettant de vérifier que les composants sont bien branchés et fonctionnels, sans passer par le serveur.

Séance du 11/03/25

Nous avons commencé par vérifier que la solution à deux Raspberry Pi fonctionnait correctement et que les données du capteur étaient cohérentes.

Ensuite, nous avons consacré du temps au **débogage** d'une partie du code refactorisé et à la configuration correcte des IPs utilisées dans le programme.

- **La partie capteur et serveur est maintenant complètement fonctionnelle**
  Il ne reste qu'à déterminer l'intervalle optimal entre chaque mesure.

- **Quelques bugs restent sur la partie contrôleur**
  Le brouillon de l'algorithme **Q-learning** devra être adapté. À terme, un seul fichier devra gérer à la fois l'apprentissage par renforcement et le contrôle de la voiture.

Notre objectif avant la prochaine séance est de résoudre ces derniers problèmes afin de pouvoir tester **l'algorithme de Q-learning en conditions réelles**.

Mise à jour du dépôt GitHub
Nous avons également mis à jour notre code sur GitHub pour conserver une trace des modifications et faciliter la collaboration.

https://github.com/Pi3dra/TER_BP_JP

Première estimation du délai de communication
En utilisant `ping`, nous avons mesuré un temps de réponse d'environ

0.6 ms, ce qui semble être un délai suffisamment court pour notre application.

Mesurer le temps de réponse du contrôleur
Pour les prochaines séances on veut vérifier combien de temps s'écoule entre la réception de l'information envoyée par le capteur et l'action correspondante du contrôleur.

Journée du 04/02/25

Aujourd'hui on a:

- **Soudé les pins dès 3 convertisseurs analogiques vers digital**

(Cela a pris un temps un bon moment) conseils pour les futurs étudiants: positionner les pins puis fixer le tout sur un bout de carton. Puis au moment de souder faire contacte avec le pin et l'anneau en fer du PCB afin de bien chauffer toutes les surfaces ou doit adhérer l'étain

- On a mis en place le circuit qui permet de contrôler la vitesse de la voiture, puis on a cherché à mettre en place le système qui nous permettrait de mesurer le voltage de la piste, mais on a pas eu le temps de finir (On a réussi à lire de l'output mais il n'était pas cohérent, il faudra voir si c'est un problème du code ou du circuit)

Cependant on s’est renseigné sur les caractéristiques des différents composants puis comment les utiliser.

Travail qu’on doit faire pour le cahier des charges:

- Recueillir les noms et instructions d’installation des bibliothèques python utilisées dans le code (faire un fichier bash pour automatiser l’installation?).
- Recueillir les fiches techniques de tous les composants utilisées dans le circuit (puis les instructions de comment les brancher)

Plan pour la prochaine séance:

- Finir de mettre en place le système de mesure de la tension
- Intégrer ceci au script client.py afin de le transmettre au serveur
- Commencer à mettre en place le système de Q-learning si on a le temps (Réfléchir à: l'implémenter à la main ou utiliser des bibliothèques externes?)

Journée du 28/01/25

Aujourd'hui, nous avons principalement essayé de mettre en place le circuit qui permettrait de contrôler la voiture avec le Raspberry Pi.

Essai

Tout d'abord, nous avons coupé les câbles d'une manette et les avons branchés à un potentiomètre, ce qui nous permettait de réguler manuellement la résistance (et donc la vitesse de la voiture). Ensuite, nous avons tenté d'utiliser un capteur de tension avec le Raspberry Pi, mais en cherchant sur Internet comment l'utiliser, nous avons rencontré un problème :

L'output de celui-ci est analogique, mais le Raspberry Pi n'a pas d'entrée analogique. Il faudrait donc acheter/trouver un ADC (convertisseur analogique-numérique) afin de pouvoir lire correctement la tension sur le Raspberry Pi.

En cherchant sur Internet, nous avons trouvé deux modèles :

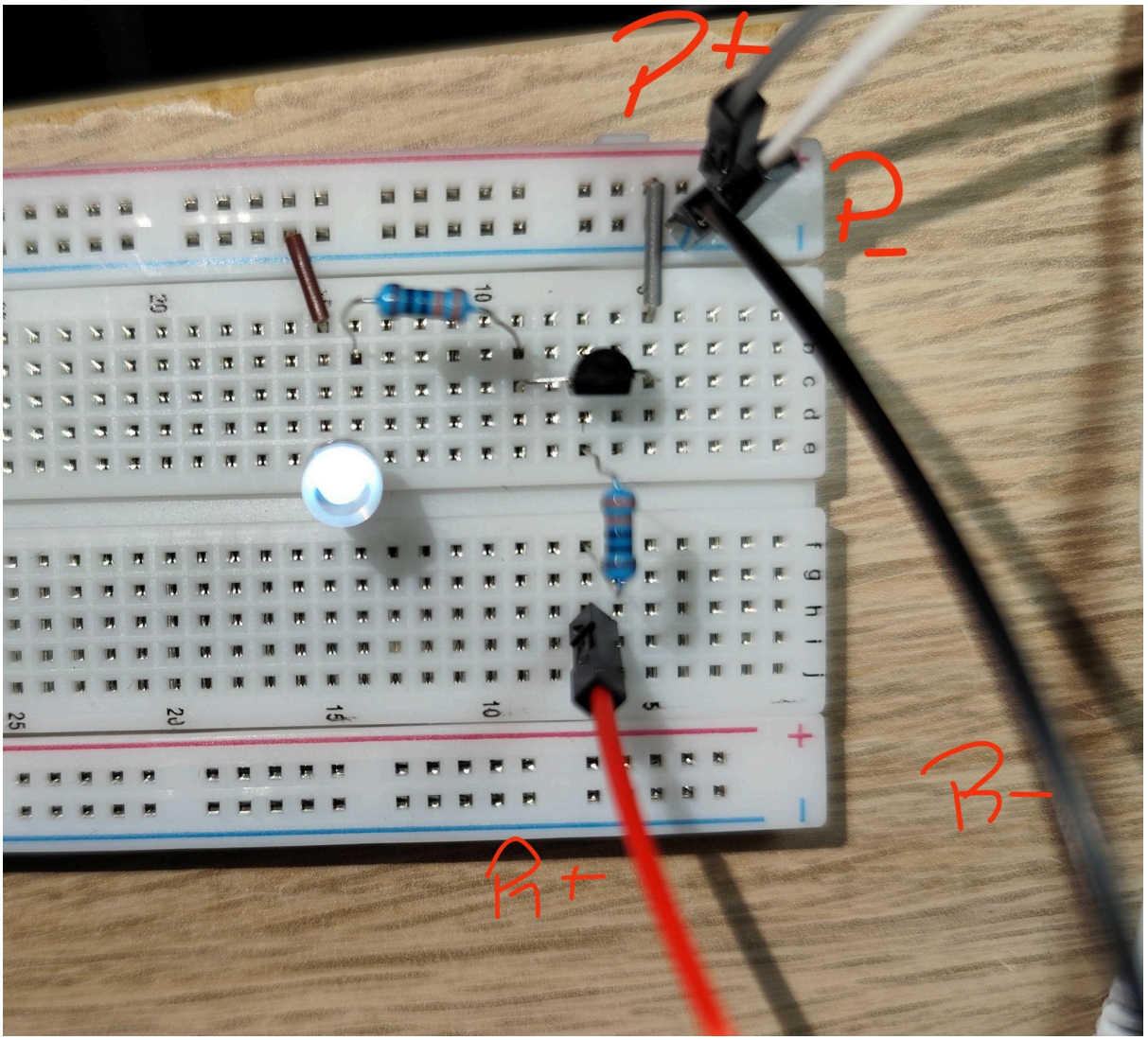
- **MCP3008** : A une sortie sur 10 bits, donc moins de précision que l'ADS1115.
- **ADS1115** : A une sortie de 16 bits, donc plus de précision.

Vu que les prix sont presque équivalents selon l'endroit où on les achète, autant opter pour celui avec plus de précision (c'est ce que nous pensons).

Ensuite, nous avons essayé de contrôler le voltage en éteignant et allumant un transistor 2N222 avec un signal PWM d'un des GPIOs du Raspberry Pi afin de "simuler" un voltage continu.

Nous avons "perdu" un bon moment à essayer de mettre en place ce circuit, notamment parce que nous avions deux sources d'alimentation : celle du Raspberry Pi et celle de la piste (et nous ne connaissions vraiment pas grand-chose en circuits électriques). Finalement, nous avons trouvé la solution presque par accident : en branchant le terminal négatif de la piste et celui du Raspberry Pi au même endroit, ce qui complétait le circuit et permettait de contrôler le courant de la piste. (Mais nous avons du mal à comprendre pourquoi, en partageant les pôles négatifs des deux sources, le Raspberry Pi ne subit aucun dégât, alors que la piste fournit 15V).

Recreation du circuit final:



(R: Raspberry , P: Piste)

Cependant, nous n'avons pas eu le temps d'essayer de contrôler la vitesse avec le signal PWM sur la piste. Mais une fois rentrés chez nous, nous avons essayé avec une LED, et cela a fonctionné (en reproduisant le même circuit).

Entre le moment où on essayait de mettre en place le circuit, on s'est arrêté réfléchir et entre temps on a mis en place une solution de communication entre le raspberry et l'ordinateur: un serveur flask en python.

Le choix du serveur http avec python a été fait tout simplement parce qu'on avait déjà de l'expérience avec cela

Cette partie du projet, comme vu en visio, vise à transmettre des informations générées par la piste de courses vers un serveur central qui les stocke et les affiche, en utilisant des capteurs de tension.

1. Fonctionnement du système

Le système repose sur deux composants principaux:

- **Client (Raspberry Pi)** : Génère et envoie des données aléatoires.
- **Serveur (PC central)** : Réceptionne, stocke et affiche les données

1.1. Client : Envoi des données depuis le Raspberry Pi

le script `client.py` tourne en continu sur le Raspberry Pi et exécute les actions suivantes:

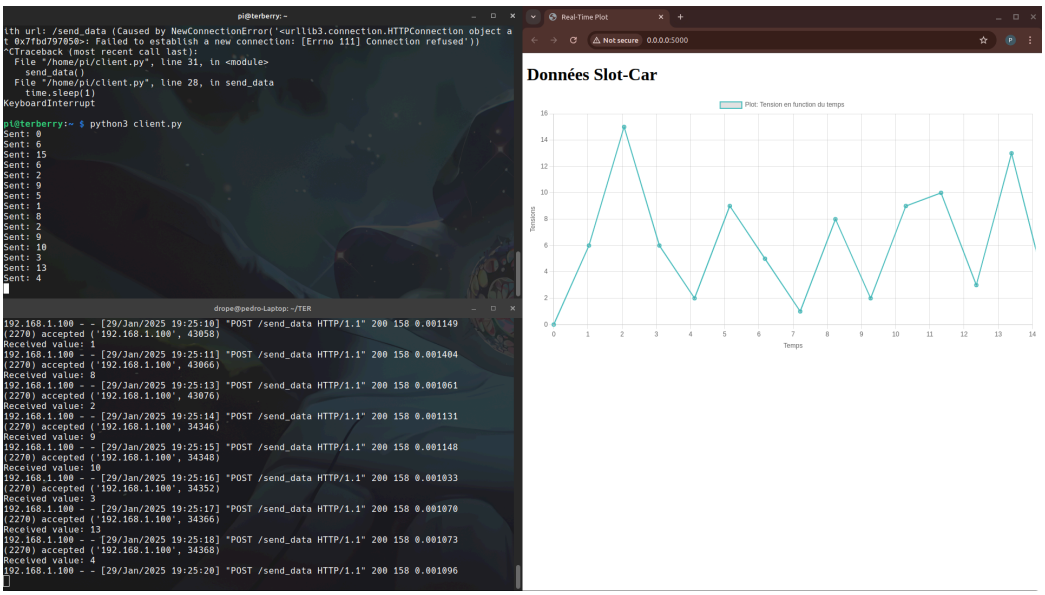
1. **Génération de données** : Un entier aléatoire entre 0 et 15 est généré. (puisqu'on a pas encore les capteurs de tension en place)
2. **Envoi au serveur** : La donnée est envoyée sous forme de requête **HTTP POST** vers le serveur.
3. **Gestion des erreurs** : Si la connexion échoue, un message d'erreur est affiché.
4. **Boucle continue** : L'envoi se fait toutes les secondes. (ceci est arbitraire, on devra surement le diminuer quand on collectera les vrais informations)

1.2. Serveur : Réception et diffusion des données

Le script `server.py` est exécuté sur un PC central et assure les fonctions suivantes:

1. **Réception des données** : Le serveur Flask reçoit les requêtes HTTP contenant des valeurs envoyées par le Raspberry Pi.
2. **Mise à jour en temps réel** : Grâce à WebSockets (`socketio`), les nouvelles données sont affichés immédiatement

Ce qui donne ceci:



Journée du 21/01/25

Aujourd'hui, on a perdu un peu de temps à cause de deux problèmes :

1. **Problèmes de communication avec Alexis** : Ce n'était pas la faute d'Alexis, mais des soucis liés à la boîte mail. On a donc dû attendre un bon moment qu'il vienne nous ouvrir.
2. **Utilisation d'un oscilloscope** : Comme on n'en avait jamais utilisé, il a fallu prendre le temps de comprendre comment ça fonctionne.

Malgré ces contretemps, on a réussi à avancer. On a répondu à certaines questions, mais on en a aussi soulevé de nouvelles.

On se demandait d'abord comment était caractérisé le signal électrique dans les rails:

Avec les recherches qu'on avait faites avant, on a trouvé que pour les voitures RC, les moteurs électriques sont contrôlés principalement de deux manières :

1. **Contrôle linéaire** : Plus de voltage = plus de vitesse.
2. **Modulation de largeur d'impulsion (PWM)** : Il s'agit d'un signal carré où la durée de chaque impulsion détermine la quantité de puissance envoyée au moteur.
    - Des impulsions courtes signifient une vitesse faible.
    - Des impulsions longues signifient une vitesse plus élevée.

Après avoir mesuré les rails avec l'oscilloscope, voici ce qu'on a observé :

- Le contrôle des voitures se fait de manière **linéaire**, avec une tension maximale de **14,8 volts**.
- Quand on utilise la gâchette, la tension est très bruyante, avec des variations de ±10 V (entre les crêtes maximales et minimales) et en fonction de la position de la gâchette.
- Par contre, lorsque l'on appuie sur le bouton, le voltage monte directement à 14,8 V **sans bruit**.

Ce bruit est un problème. On pense qu'il faudrait calculer une moyenne du voltage sur un certain temps pour avoir une estimation plus stable de la vitesse. Cela dit, cela risquerait de créer des délais dans la mesure des valeurs.

- La tension reste stable partout sur la piste.
- Si les voitures ne sont pas sur les rails, les manettes n'ont aucun effet sur le voltage.

On avait aussi des questions sur le fonctionnement des manettes:

Deux modes de fonctionnement :

- La gâchette permet de réguler la vitesse.
- Le bouton met directement la vitesse au maximum.

À l'intérieur des manettes :
En ouvrant une manette (sans l'endommager), on a constaté qu'elles sont entièrement analogiques. On a trouvé ce qui semble être un potentiomètre linéaire.

- Plus on appuie sur la gâchette, plus la "tête" du potentiomètre descend, réduisant la résistance dans le circuit et augmentant ainsi le voltage. On pense que c'est ce potentiomètre qui génère le bruit dans le signal électrique.
- Le bouton, quant à lui, est simplement un interrupteur qui ferme le circuit sans ajouter de résistance, ce qui explique peut-être l'absence de bruit.

En fonction de ces contraintes on a pensée a 3 systèmes possibles :

Système 1 :
C'est peut-être l'un des plus compliqués à implémenter, mais on pense qu'il présente aussi des avantages intéressants. L'idée serait d'alimenter un Arduino Nano BLE 33 à l'intérieur de la voiture via les rails, et que cet Arduino contrôle le moteur à travers un petit potentiomètre ou un transistor (PWM). Cet Arduino a des dimensions de 5x2 cm, pèse seulement 5 g, et est équipé du Bluetooth, d'un gyroscope, d'un accéléromètre, et peut réguler jusqu'à 21V. Pour l'alimenter, il suffirait de bloquer le bouton d'une manette afin de fournir 14,8V sans bruit.

(On a déjà trouvé les pièces nécessaires pour ce système.)

Points positifs	Points Négatifs
Mesures supplémentaires et de bonne qualité (accéléromètre et gyroscope) Pas besoin de connaître sa position, le gyroscope peut nous donner une approximation (en prenant en compte les courbes) Rajoute un peu de poids à la voiture. On trouve qu'elles sont très légères. Parfois rien qu'à cause de la brosse qui fait contact avec les rails, les roues avant ne sont plus en contact avec la piste, et elles ont donc moins de friction sur la piste et moins de contact avec les rails. Tout est sur la voiture donc on peut la changer de piste très facilement sans avoir à installer des choses supplémentaires sur la piste	Faudrait acheter plus des pièces Il faudrait probablement imprimer des pièces 3D pour protéger l'arduino Si la voiture sort de la piste l'arduino n'est plus alimentée, on se demande donc comment informer l'ordinateur d'une sortie de piste (On a pensée a une petite batterie, ou se dire que si a partir de x temps il n'y a plus de communication considérer la voiture comme sortie).

Système 2 :
Celui-ci est plus simple à implémenter en termes de matériel, mais il pourrait être plus compliqué en ce qui concerne la qualité des données.

L'idée serait d'utiliser le capteur de tension sur les rails et un détecteur de position, puis de couper le câble de la manette et de le brancher sur un potentiomètre digital ou un transistor (en utilisant du PWM), le tout étant contrôlé par une Raspberry Pi.

(On n'a pas encore les pièces précises pour ce système.)

Points positifs	Points négatifs
Le matériel et l'installation est simple On en a déjà presque tout ce qu'il faut	Il faut sacrifier une manette Il faut utiliser la mesure du voltage qui ne nous semble pas fiable. Requiert d'installer des capteurs a des endroits importants sur la piste

Systeme 3:
C'était l'une des premières idées, mais maintenant on pense qu'elle n'est pas très pratique.

L’idée serait d’utiliser le capteur de tension sur les rails et un détecteur de position, puis d'utiliser un moteur servo pour actionner la gâchette de la manette.

(On n’a pas encore les pièces précises pour ce système.)

Points positifs	Points négatifs
Le matériel n’est pas très compliqué à utiliser et installer ce n’est pas nécessaire de sacrifier une manette	Il faut du materiel supplementaire (surement une source d'électricité pour le moteur, puis le moteur et tout un système pour tenir en place la manette et bouger la gâchette) Il faut toujours utiliser la mesure du voltage qui n’est pas fiable. Requiert toujours l’installation des capteurs a des endroits importants. Introduit un délai important: communication pc -> arduino + communication arduino -> moteur + temps que prend le moteur a bouger a la position indiquée

Liens interessants:

(Capteurs IR + Signaux et codes de la piste CARRERA) (dépend de la piste)
https://www.slotforum.com/threads/project-to-build-an-autonomous-slot-car-racing-system.212491/

(Accéléromètre + Gyroscope)
https://www.youtube.com/watch?v=aFaaDHfdo6w

(Construit carrément un slotcar entier)
https://www.youtube.com/watch?v=xRONSEk5cUU

- Gyroscope, accéléromètre et détecteurs IR
- Encoders sur les essieux pour déterminer la vitesse et la distance

(Uniquement accéléromètre)
https://www.youtube.com/watch?v=pXSHy4pLMRk

(Idee interessante, premier passage pour récupérer des données sur la piste)