

02-10

vendredi 17 février 2023 12:57

Choix du projet: Décision de construire une trottinette de A à Z avec les attributs suivant;

- Cruise adaptatif (Principal but du projet niveau programmation et recherches)
- Batterie home made
- Frame de la trottinette home made pliable
- Gadget tech: Display vitesse, Lumière, Klaxon, touch screen pour contrôle etc.

02-15

vendredi 17 février 2023 13:12

Recherches et calculs pour les besoins du moteur. Assez difficiles à calculer nous n'avons pas trouvé de bonne façons d'effectuer les calculs encore pour la puissances requise du moteur. Décisions faite pour les besoin de vitesse: 50km/h en vitesse max pour une masse totale de 130kg (M.Trotinnette + M.Conducteur)

02-17

vendredi 17 février 2023 12:55

Discussion avec le professeur : nous allons finalement utiliser une trottinette déjà existante afin de comprendre le fonctionnement de celle-ci. Ça va aussi nous permettre de commencer le plus important du projet, soit le Cruise adaptatif, au lieu de prendre une grande quantité de temps pour la construction. La V1 se fera donc sur la trottinette déjà montée et si nous avons le temps, nous construirons notre frame tout en effectuant les recherches et la construction du Cruise.

Décision des attributs en ordres d'importance et de création du Cruise:

Cruise:

- Boutons de contrôles comme voitures
- Ralentir ou freiner selon obstacles
- Pause du Cruise lors du freinage
- Mémoire de vitesse set
- Display des infos
- Choix distances?
- Touch screen?

Début des recherches sur le fonctionnement du ACC

Sébastien vient de réaliser la séparation des étapes initiales du projet, soit de créer un Cruise qui fonctionne en gardant une vitesse en mémoire et lorsque cette étape sera terminée rendre le tout adaptatif.

- Décision sur l'ordre des étapes de développement du CCA :
- Cruise fonctionnel (maintient d'une vitesse constante, arrêt lors du freinage)
 - Détection d'objet
 - Analyse des objets (distance & vitesse)
 - Adaptation de la vitesse
 - Ajout de fonctionnalités de sécurité

Suite à une réflexion concernant les effets de l'accélération sur le conducteur, nous avons déceler la **problématique** suivante : si l'accélération est toujours constante, passer d'une basse vitesse à une haute vitesse risque de donner au coup au conducteur qui peut risquer de le faire tomber et un autre coup lorsque la vitesse choisie est atteinte. Pour contrer ce problème, nous allons devoir développer une méthode d'accélération graduelle au début de l'accélération et une décélération graduelle lorsque la vitesse s'approche de la vitesse cible ou d'un obstacle.

Composants pour un Cruise fonctionnel :

- Microcontrôleur	
- Capteur de vitesse	
- Potentiomètre	Un potentiomètre est un composant électronique qui peut être utilisé pour régler la vitesse cible du régulateur de vitesse. Le potentiomètre permet au conducteur de fixer une vitesse cible à laquelle le régulateur de vitesse ajuste la vitesse du véhicule.
- Driver de moteur	Le driver de moteur est un composant électronique utilisé pour contrôler la vitesse du moteur du véhicule. Le driver de moteur est généralement utilisé pour réguler la quantité d'énergie électrique envoyée au moteur.
- Transistors et diodes	Les transistors et les diodes sont des composants électroniques utilisés pour contrôler le flux d'énergie électrique dans le régulateur de vitesse. Les transistors peuvent être utilisés pour contrôler le courant envoyé au moteur, tandis que les diodes peuvent être utilisées pour protéger les autres composants électroniques contre les surtensions.
- Résistances	Permet d'ajuster la tension électrique dans le circuit.



Lecture et annotation des articles recueillis le 02-17 :

Titre	Résumé GPT	Infos tirées de l'article
Adaptive Cruise Control in Electric Vehicles with Field-Oriented Control	La section 2 présente les éléments théoriques nécessaires pour comprendre la régulation de vitesse adaptative et le contrôle orienté champ. La section 3 décrit la modélisation mathématique de la dynamique du véhicule et les équations du contrôleur adaptatif de vitesse.	<ul style="list-style-type: none">o La méthode la plus simple pour construire un CCA serait de les développer séparément : le CCA contrôle la vitesse d'un CC classique. Un module est donc en charge de détecter la distance des obstacles, tandis que l'autre s'occupe de réguler la vitesse.o Puisque cet article traite pas la décélération en fonction d'un frein moteur (ne considère pas l'utilisation d'un frein mécanique), la lecture fût arrêtée à la page 6. Les éléments théoriques ne concordent pas exactement à ce que nous recherchons.
Adaptive Cruise Control Strategy Design with Optimized Active Braking Control Algorithm	L'article présente une méthode de conception d'une stratégie de <u>contrôle de vitesse de croisière adaptative</u> avec un algorithme de <u>contrôle de freinage actif</u> optimisé pour améliorer la sécurité et la performance de la conduite automobile.	<ul style="list-style-type: none">o Cette article donne une idée de comment calculer l'accélération et la décélération de la trotinette selon les paramètres données. Toutefois, il porte sur un CCA mécanique plutôt qu'électrique. Il faut donc convertir les formules données et adapter les conclusion dont il est question afin de l'appliquer à un moteur électrique.o Les explications du contrôle du freinage sont peu pertinentes puisqu'il s'agit de freins hydrauliques et du contrôle de carburant à l'intérieur de pistons.o Une section portant sur le module responsable de la détection des distances et vitesses relatives entre l'obstacle et la trotinette propose des formules complexes (sans explications claires) que je ne comprends pas.o Un terme récurrent entre plusieurs articles consultés revient dans ce texte. Il sera pertinent d'approfondir nos recherches sur ce sujet : "PID algorithm"
Research on Longitudinal Control Algorithm of Adaptive Cruise Control System for Pure Electric Vehicle	Cet article présente un algorithme de contrôle longitudinal adaptatif <u>pour les véhicules électriques purs</u> avec régulateur de vitesse adaptatif, qui améliore l'efficacité énergétique et la sécurité de conduite.	<ul style="list-style-type: none">o Cet article propose un algorithme avancé utilisant des réseaux de neurones pour calibrer automatiquement les paramètres de l'algorithme de commande PID.o Malheureusement, les algorithmes de base ne sont pas montrés dans l'article.o La section 2.1.2 offre une bonne piste pour l'ébauche des tâches à réaliser

Recherche d'un algorithme de CCA adapté à un véhicule électrique :

- Plusieurs types d'algorithmes peuvent être utilisés. Voici un tableau expliquant les décisions prises lors de la sélection d'un type souhaité pour ce projet :

Type d'algorithme	Description	Limitations	Décision
Commande proportionnelle-intégrale-dérivée (PID)	Un algorithme de commande PID utilise la rétroaction de l'état du système pour ajuster un signal de commande en fonction de la consigne souhaitée. L'algorithme PID peut être utilisé pour réguler la vitesse, la position ou tout autre paramètre d'un système, et est couramment utilisé pour réguler des moteurs électriques, des robots, des systèmes de contrôle de température, etc.	<ul style="list-style-type: none">- Le réglage des coefficients PID peut être un processus complexe et délicat, car il nécessite une connaissance approfondie du système à réguler et une expérience pratique.- Performance limitée pour des systèmes complexes : L'algorithme de commande PID est conçu pour des systèmes linéaires et stables, et peut avoir des performances limitées pour des systèmes non linéaires et instables.	Considérant nos ressources et nos connaissances limitées et le fait que nous visons un CCA fonctionnel sans nécessairement l'optimiser, cette méthode, qui est décrite comme offrant de moins bonnes performances de régulation que d'autres, offre tout de même des résultats tout à fait acceptables quant à l'objectif de notre projet. Sa complexité étant d'ailleurs la moins élevée parmi les méthodes proposées : RETENU
Commande par logique floue	Un algorithme de commande par logique floue utilise des règles linguistiques pour contrôler un système. Ces règles sont définies en utilisant des variables linguistiques (par exemple, "faible", "moyen" et "élevé") plutôt que des variables numériques. Les variables linguistiques sont définies à l'aide de fonctions d'appartenance floues, qui affectent un degré d'appartenance à chaque variable linguistique. L'algorithme de commande par logique floue convertit ensuite les entrées du système (par exemple, la vitesse et la position d'un moteur électrique) en variables linguistiques en utilisant les fonctions d'appartenance floues. Les règles de contrôle sont appliquées en combinant les variables linguistiques d'entrée pour obtenir une variable linguistique de sortie. Les règles de contrôle sont généralement exprimées sous la forme "Si condition ALORS action", où la condition et l'action sont des variables linguistiques. La variable linguistique de sortie est ensuite déflouée en une valeur numérique qui est utilisée comme signal de commande pour le système. Cela peut être fait à l'aide d'une méthode de défuzzification, telle que le centre de gravité ou la méthode du maximum.	<ul style="list-style-type: none">- Conception des règles : La conception des règles de logique floue peut être un processus difficile et subjectif, car elle implique souvent une compréhension approfondie du système à contrôler et une expertise en ingénierie. De plus, le nombre de règles nécessaires pour contrôler efficacement le système peut être élevé, ce qui peut rendre la conception et le réglage des règles complexes et longs.- Performance limitée pour des systèmes non linéaires : Bien que la commande par logique floue soit utile pour les systèmes complexes, elle peut ne pas être efficace pour les systèmes non linéaires ou très dynamiques. Cela peut être dû à la difficulté de modéliser précisément le système à contrôler, ainsi qu'à la limitation des fonctions d'appartenance floues qui peuvent ne pas être en mesure de représenter de manière précise le comportement complexe du système. Dans de tels cas, des algorithmes de commande plus avancés, tels que les réseaux de neurones ou les algorithmes génétiques, peuvent être plus appropriés.	Ce concept semble complexe et très abstrait. Apprendre à programmer de tels algorithmes risque d'alourdir la tâche. Donc, REJETÉ

Commande prédictive de modèle (MPC)	L'algorithme de commande prédictive de modèle est une technique de commande avancée qui utilise un modèle mathématique du système pour prédire son comportement futur et générer des actions de commande en conséquence. Le MPC fonctionne en résolvant un problème d'optimisation à chaque itération, en utilisant le modèle du système pour prédire l'état futur du système et en choisissant la commande optimale qui minimise une fonction de coût spécifiée.	- Complexité de mise en œuvre : L'implémentation d'un MPC peut être complexe et nécessite une connaissance approfondie du système à contrôler, ainsi que des compétences en ingénierie pour la modélisation et la prédiction du comportement du système. Le processus d'optimisation est également intensif en calcul, ce qui peut nécessiter des ressources informatiques importantes. - Nécessité d'un modèle précis : L'efficacité de la commande prédictive de modèle dépend fortement de la qualité du modèle mathématique utilisé pour prédire le comportement futur du système. Si le modèle est imprécis ou incorrect, cela peut entraîner des prédictions incorrectes et des commandes inappropriées, ce qui peut conduire à une performance insuffisante du système. La construction d'un modèle précis peut être difficile et peut nécessiter des données d'expérience ou des modèles théoriques complexes pour le système à contrôler.	Nous ne disposons pas des connaissances nécessaires à l'élaboration d'un modèle optimisé. Même si nous pouvons peut-être en trouver un déjà adapté à nos besoins, nos ressources informatiques ne nous permettrons pas d'effectuer de tels calculs à chaque itération. REJETÉ
Commande par réseaux de neurones	Un algorithme de commande par réseaux de neurones est une méthode de commande qui utilise un réseau de neurones artificiels pour prédire la sortie du système en fonction de l'entrée de commande et des paramètres de la commande. Le réseau de neurones est entraîné sur un ensemble de données d'apprentissage qui contient des exemples d'entrées de commande et de sorties correspondantes pour le système à contrôler. Lors de la phase d'entraînement, le réseau de neurones ajuste automatiquement les poids et les biais des neurones pour minimiser l'erreur de prédiction entre les sorties du système et les sorties prédites par le réseau. Une fois que le réseau de neurones est entraîné, il peut être utilisé pour prédire la sortie du système en temps réel en fonction de l'entrée de commande. Les entrées de commande sont présentées au réseau de neurones, qui calcule une sortie en utilisant les poids et les biais appris pendant la phase d'entraînement. Cette sortie est alors utilisée comme entrée pour le système à contrôler. L'avantage de l'algorithme de commande par réseaux de neurones est qu'il peut s'adapter à des systèmes non linéaires et complexes, pour lesquels les algorithmes de commande traditionnels peuvent être inefficaces. Cependant, l'inconvénient majeur est que le processus d'entraînement du réseau de neurones peut être long et nécessiter beaucoup de données d'apprentissage pour produire des prédictions précises. De plus, les réseaux de neurones peuvent être sensibles au bruit et à la variance dans les données, ce qui peut réduire la performance de la commande.	- Besoin de données d'entraînement et de validation : Le processus d'entraînement des réseaux de neurones nécessite une grande quantité de données d'entraînement et de validation pour produire des prédictions précises. Il peut être difficile d'obtenir ces données pour des systèmes électriques complexes, en particulier si les conditions de fonctionnement sont variables. - Complexité de mise en œuvre et de paramétrage : La mise en œuvre d'un algorithme de commande par réseaux de neurones peut être complexe et nécessite des compétences en ingénierie pour la modélisation et la prédiction du comportement du système. Le processus de paramétrage des réseaux de neurones peut également être difficile, car il peut y avoir de nombreux hyperparamètres à ajuster pour obtenir les meilleurs résultats.	Dû à la complexité de l'emploi d'un tel algorithme : REJETÉ

Source : ChatGPT

Recherches concernant le PID algorithm :

A PID (Proportional Integral Derivative) controller consists of three components that are adjusted based on the difference between a set point (SP) and a measured process variable (PV).

$$e(t) = SP - PV$$

The output of a PID controller ($u(t)$) is calculated using the sum of the Proportional, Integral, and Derivative terms where K_P , K_I , and K_D are constants that can be adjusted to fine-tune the performance of the controller.

$$u(t) = K_P e(t) + K_I \int_0^t e(t)dt + K_D \frac{de(t)}{dt}$$

Output = **Proportional** + **Integral** + **Derivative**

Proportional (P) control: This component adjusts the output of the process based on the current error between the setpoint and the process variable (PV). The larger the error, the larger the correction applied.

Proportional = $K_P e(t)$

Integral (I) control: This component adjusts the output based on the accumulated error over time. It helps eliminate steady-state error and can improve the stability of the control system.

$$\text{Integral} = K_I \int_0^t e(t)dt$$

Derivative (D) control: This component adjusts the output based on the rate of change of the error. It helps to dampen oscillations and improve the stability of the control system but is often omitted because P1 control is sufficient. The derivative term can amplify measurement noise (random fluctuations) and cause excessive output changes. Filters are important to get a better estimate of the process variable rate of change.

$$\text{Derivative} = K_D \frac{de(t)}{dt} = -K_D \frac{d(PV)}{dt}$$

The derivative of the error is substituted with the derivative of the Process Variable (PV) to avoid derivative kick when there is a setpoint change. The value of the controller output $u(t)$ is transferred as the system input. The K_P , K_I , and K_D parameters can also be written in terms of controller gain K_c , integral reset time τ_I , and derivative time constant τ_D .

$$K_P = K_c, \quad K_I = \frac{K_c}{\tau_I}, \quad K_D = K_c \tau_D$$
$$u(t) = u_{bias} + K_c e(t) + \frac{K_c}{\tau_I} \int_0^t e(t)dt - K_c \tau_D \frac{d(PV)}{dt}$$

Output = Bias + Proportional + Integral + Derivative

The u_{bias} term is a constant that is typically set to the value of $u(t)$ when the controller is first switched from manual to automatic mode. This gives *bumpless transfer* if the error is zero when the controller is turned on. The three tuning values for a PID controller are the controller gain, K_c , the integral time constant τ_I , and the derivative time constant τ_D . The value of K_c is a multiplier on the proportional error and integral term and a higher value makes the controller more aggressive at responding to errors away from the set point. The integral time constant τ_I (also known as integral reset time) must be positive and has units of time. As τ_I gets smaller, the integral term is larger because τ_I is in the denominator. Derivative time constant τ_D also has units of time and must be positive. The set point (SP) is the target value and process variable (PV) is the measured value that may deviate from the desired value. The error from the set point is the difference between the SP and PV and is defined as $e(t) = SP - PV$.

À faire aujourd'hui :

une ébauche du fonctionnement détaillé du CCA en incluant les notions acquises lors des dernières recherches, soit : l'algorithme d'un CCA pour un véhicule électrique, l'utilisation de l'algorithme PID pour traiter des changements, le traitement du signal partant du capteur jusqu'au moteur

02-20

18 février 2023 16:15

Au terme des recherches du 02-18, une idée concrète commence à se former quant au **fonctionnement détaillé du CCA** et des modules à développer :

Le capteur détecte un obstacle.	
Le capteur mesure à la fois la différence de vitesse entre les deux objets et la distance entre les eux.	Nous avons choisi de mesurer la vitesse et la distance parce que ; - en mesurant seulement la distance, pour déterminer si la trottinette s'approche ou s'éloigne, il faut déterminer si la distance diminue ou augmente entre deux moments dans le temps. Or, si la vitesse des deux objets est presque identique, il sera difficile d'avoir une intervalle de distance significative si l'intervalle de temps est trop petite. Il faudrait donc augmenter l'intervalle de temps. Cependant, si l'intervalle de temps est trop grande, cela pose un problème de sécurité si la vitesse des deux objets est très différente, puisqu'une deuxième lecture trop tardive pourrait résulter en une collision. - si nous mesurons seulement la vitesse, il sera plus complexe de rester à une distance sécuritaire d'un objet et ou de suivre celui-ci. Il faudrait mesurer la distance grâce à l'effet doppler plutôt qu'utiliser un capteur prenant directement la mesure requise.
Le signal est alors transmis à trois modules (Adaptation -> Cruise -> Mouvement).	Ce principe permet de généraliser l'utilisation du système en divisant les niveaux de complexité. En effet, chaque module ajoute un niveau au précédent, qui peut fonctionner de façon autonome. Concrètement, cela implique que le module M permet le déplacement et la gestion de la vitesse et peut fonctionner à lui seul (à l'aide de la gâchette et du moteur). En lui ajoutant le module C, on peut maintenir une vitesse stable sans l'utilisation de la gâchette. En ajoutant au module C le module A, on peut ajuster la vitesse réelle sans modifier la vitesse voulue. En utilisant ce principe, il sera plus facile de développer notre CCA et offrira à l'utilisateur différents modes de conduite au produit final.
Le module A calcule le nouveau set point et l'envoie au module C en fonction de la vitesse de l'obstacle (s'il y a lieu).	
Le module C utilise le set point (soit celui de l'utilisateur ou celui du module A) pour déterminer si la trottinette doit accélérer ou ralentir pour atteindre la vitesse désirée de façon constante et communique la variation de vitesse requise au module M.	Le module C doit remplacer l'utilisation de la gâchette et du levier à frein tout en tenant compte des changements de vitesse dus aux pentes et autres perturbations.
Le module M contrôle la puissance du moteur ou le freinage requis selon la donnée envoyée par le module C.	En analysant le fonctionnement du Cruise control de notre modèle (le set point est défini en gardant la gâchette à la même position pendant quelques secondes), nous avons supposé que le set point est une résistance plutôt qu'une vitesse. Cette prémisse est appuyée par le fait que dans une pente, la vitesse diminue lorsque le Cruise control est activé : c'est donc la force du moteur qui est constante et non la vitesse. Cette méthode ne convient toutefois pas à nos besoins, puisqu'il sera nécessaire de maintenir une vitesse constante même dans une pente pour des raisons de sécurité (ralentissement dans une pente descendante / maintien de la distance dans une pente ascendante pour éviter les dépassements dangereux). Notre modèle ne nous aidera donc pas à concevoir le Cruise control, mais uniquement pour comprendre la relation entre la gâchette et la puissance du moteur.

(Avec un point de vue de construction d'un programme procédural) Discussion sur différentes fonctions et algorithmes que notre programme aura besoin pour contrôler la trottinette :

Fonction	Paramètres	Type de retour	Description
_Freinage			
_Accélération			
_Conversion_Résistance_Moteur			
_Distance_Sécuritaire_Respectée		bool	
_Accélération_Nulle		bool	

Algorithme	Problème à résoudre	Séquence d'instructions globale	Description (pourquoi on en a besoin)
@Ratio_Vitesse_Distance	Savoir le signe de l'accélération à appliquer		Mesurer la force requise à appliquer pour un changement de vitesse proportionnelle à la situation (et éviter les coups) autant pour freiner que pour accélérer. Si le ratio est positif, il faut accélérer (si demandé). Si le ratio est négatif, il faut ralentir/freiner.
@Calcul_Vitesse_Obstacle	Connaître la vitesse de l'obstacle.		Cette algorithme permet de calculer la vitesse de l'obstacle en lui envoyant une onde électromagnétique de fréquence connue et en calculant la différence avec la fréquence retournée causée par l'effet Doppler.
	Conversion d'une vitesse voulue en résistance électrique / force de freinage à appliquer à tout moment.		
@Speedomètre	Connaître la vitesse réelle de la trottinette	Pistes de solutions : - Ratio nb de tours / puissance - GPS	Pour conserver une vitesse constante malgré les pentes et autres perturbations, il faut trouver une façon de mesurer la vitesse de la trottinette qui ne dépend pas de la puissance requise par le moteur (pour une même vitesse, la puissance ne sera pas la même dans une pente).

Variables (\$) / constantes (&)	Description
&distance_minimale	Constante de distance minimale de sécurité entre le conducteur et l'objet devant lui.
&distance_maximale	Constante de distance entre le conducteur et l'objet devant lui mais ignorant les obstacles trop loin (ex.: un objet à 4km ne sera pas considéré).
\$facteur_distance	Puisque la distance sécuritaire doit augmenter proportionnellement à la vitesse de la trottinette (plus la vitesse est grande, plus la distance sécuritaire doit être grande), ce facteur variable sera multiplié aux distances ci-haut pour tenir compte de cette problématique
\$vitesse_obstacle	Vitesse de l'obstacle
\$vitesse_trottinette	Vitesse réelle de la trottinette indépendante de la puissance du moteur
\$vitesse_set	Vitesse réglée par l'utilisateur pour le Cruise control

À faire aujourd'hui :

Analyse de différentes situations pour mieux évaluer et définir le fonctionnement du système
(ex.: Trotinette rattrape un obstacle : qu'est-ce qu'il se passe? --> doit freiner [...])

Situation	Fonctionnement du système
Cruise control désactivé	La vitesse est contrôlée par l'utilisateur à l'aide d'une gâchette qui fait varier la résistance électrique et qui augmente / diminue ainsi la puissance fournie par le moteur.
Bouton ON appuyé	Le display change de couleur, mais rien ne se passe tant que l'utilisateur n'a pas défini une vitesse.
Bouton SET	Si le CC est à ON : la vitesse actuelle est maintenant définie comme set point. Cette vitesse restera affichée au bas de l'écran. La gâchette devient obsolète, puisque la vitesse est gérée automatiquement. Pour changer la vitesse, il faut maintenant utiliser les boutons + / -
Bouton CANCEL	Si le CC est à ON : mise en pause du CC (la valeur du set point est gardée en mémoire). Le bas de l'écran affiche maintenant « Veuillez appuyer sur RESUME ». Le contrôle de la vitesse se fait à nouveau avec la gâchette. Pour réactiver le CC, on peut aussi SET une nouvelle vitesse.
Freinage lorsque le CC est activé	Si le CC est à ON et en action : l'actionnement manuel des freins met en pause le CC.
Bouton RESUME	Si CC est à ON : la gâchette devient obsolète et la vitesse est automatiquement ajustée pour atteindre la vitesse définie précédemment.
Bouton OFF appuyé	Le display change et la vitesse définie est supprimée (reset du programme de Cruise control). Le contrôle de la vitesse se fait à nouveau avec la gâchette.
Pente ascendante avec CC	La puissance fournie au moteur augmente afin de maintenir la vitesse demandée. Si le moteur est à 100% de sa capacité, un message s'affiche au bas de l'écran pour avertir l'utilisateur que la vitesse demandée ne pourra pas être atteinte. Ce message disparaît au moment où la charge sur le moteur diminue (fin de la pente ou nouvelle vitesse).
Pente descendante avec CC	Si la vitesse réelle dépasse le set point, les freins seront automatiquement actionnés (doucement) afin de réduire la vitesse jusqu'à atteindre la vitesse voulue. Cette fonctionnalité permet une sécurité accrue.
Trotinette rattrape un obstacle	La vitesse de l'obstacle devient le nouveau set point temporaire de la trotinette. Le système calcule donc la vitesse de l'obstacle et ajuste celle de la trotinette afin de maintenir une distance sécuritaire prédéfinie.

Ébauche

```

public float vitesse_actuelle;
public float vitesse_cible;
public float erreur_vitesse;
public float puissance_moteur;
public float puissance_freinage;

const float KP = 0.1;
const float KI = 0.01;
const float KD = 0.05;
const float KP_frein = 0.5;
const float KI_frein = 0.05;
const float KD_frein = 0.25;

float Calcul_Vitesse_Trotinette()
{
    return valeur_speedometre;
}

void Set_Vitesse_Trotinette()
{
    vitesse_cible = Calcul_Vitesse_Trotinette();
    return;
}

void Regulation_Vitesse()
{
    if(CC_ON == false)
        return;

    vitesse_actuelle = Calcul_Vitesse_Trotinette();

    erreur_vitesse = vitesse_cible - vitesse_actuelle;

    // Calcul PID
    puissance_moteur = KP * erreur_vitesse + KI * somme_erreurs + KD * variation_erreurs;
    puissance_freinage = KP_frein * erreur_vitesse + KI_frein * somme_erreurs_frein + KD_frein * variation_erreurs_frein;

    somme_erreurs += erreur_vitesse;
    variation_erreurs = erreurs_vitesse - erreur_precedente;
    erreur_precedente = erreur_vitesse;

    somme_erreurs_frein += erreur_vitesse;
    variation_erreurs_frein = erreurs_vitesse - erreur_precedente_frein;
    erreur_precedente_frein = erreur_vitesse;

    // Ajustement des incohérences
    if (puissance_moteur < 0)
        puissance_moteur = 0;
    else if (puissance_moteur > 100)
        puissance_moteur = Puissance_Max();

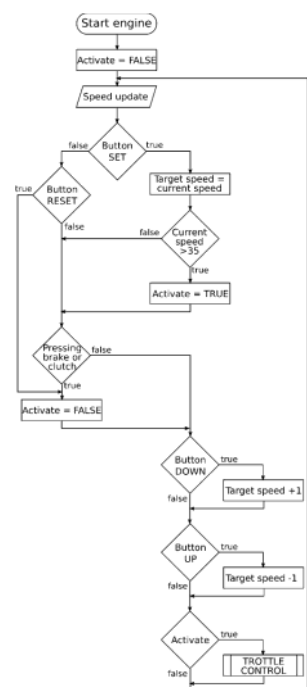
    if (puissance_freinage < 0)
        puissance_freinage = 0;
    else if (puissance_freinage > 100)
        puissance_freinage = 100;

    // Contrôle de la trotinette en fonction de la puissance moteur et du freinage
    if (erreur_vitesse > 0) // Accélération : appliquer la puissance moteur et désactiver les freins
    {
        appliquer_puissance_moteur(puissance_moteur);
        desactiver_freins();
    }

    else if (erreur_vitesse < 0) // Freinage : appliquer la puissance de freinage et désactiver le moteur
    {
        appliquer_puissance_freinage(puissance_freinage);
        desactiver_moteur();
    }

    else // Vitesse constante : désactiver le moteur et les freins
    {
        desactiver_moteur();
        desactiver_freins();
    }
}

```



```
        delay(10);  
        Regulation_Vitesse();  
    }  
float Puissance_Max()  
{  
    Set_Message_Display("Puissance maximale atteinte");  
    return 100;  
}
```

Recherche sur l'ajustement des constantes KP, KI et KD par Yohan :

[pid_widget.ipynb - Colaboratory \(google.com\)](https://colab.research.google.com/widget/ipython-notebook)

03-01

1 mars 2023 08:53

À faire aujourd'hui :

Calendrier et description du projet

Chaque tâche sera divisée en ces étapes :

- Recherche
- Planification
- Développement d'algorithme
- Implémentation et programmation
- Construction
- Débogage

★ Identification des tâches et des livrables :

Phase	Tâche	Livrable	Description	Matériel requis
0	Identification des matériaux	Liste de matériaux		
2	Détecteur de distance	Détecteur de distance fonctionnel	Retourne la distance entre la trottinette et un obstacle	- Capteur Lidar - Contrôleur Raspberry Pi - Écran LCD - Filage - Batterie (temporaire)
1	Speedomètre		Retourne la vitesse précise de la trottinette	- Capteur à effet Hall - Condensateur (filtre les interférences électromagnétiques) - Contrôleur Arduino - Écran LCD - Filage - Batterie (temporaire)
2	Détecteur de vitesse		Retourne la vitesse de l'obstacle	(Matériel utilisé pour le détecteur de distance et le speedomètre)
1	Hacker trottinette			- Outils
1	Contrôle fait par le CC de la puissance du moteur			(Signal émis par le Arduino)
1	Conversion d'une vitesse voulue en puissance moteur			(Calculs effectués par le Raspberry Pi)
4	Contrôle du freinage automatique			- Freins informatisés (nature à déterminer)
4	Contrôle de la puissance de freinage			(Signal émis par le Arduino)
4	Conversion d'une vitesse voulue en puissance de freinage			(Calculs effectués par le Raspberry Pi)
1	Contrôle de la vitesse voulue (boutons du CC)			- Boutons - (Écran LCD)
1	Affichage			- (Écran LCD)
5	Calcul de l'autonomie de la batterie			
1	Apprendre Python et l'utilisation d'un Raspberry Pi			
3	Jumeler les modules conçus à la phase 1 et 2			

Recherche sur la nature du capteur de distance :

la meilleure option serait un Lidar pour sa précision, simplicité et grande portée.

https://www.amazon.ca/-/fr/gp/product/B099NFSDCK/ref=ox_sc_act_title_1?smid=AU7WJJN4HG6E&psc=1Coding : [How to connect TFmini Plus with Arduino](#)

Retour sur la décision du 02-20 : seul un capteur de distance sera nécessaire pour déterminer la vitesse de l'obstacle. Le calcul est plus simple que prévu et se fait de façon tellement rapide qu'il n'aura pas d'effet considérable sur le fonctionnement du CCA.

03-03

3 mars 2023 13:03

Aujourd'hui, on continue de définir les tâches du 03-01 et le calendrier.

Nous avons décidé de ne pas tenir compte des obstacles en hauteur (ex.: une branche au niveau de la tête).
Yohan dit qu'il s'agit, à ce point, de sélection naturelle si le conducteur ne se penche pas pour l'éviter.

Lien vers le calendrier :

<https://docs.google.com/spreadsheets/d/16NnSW113FqPnb55YD9EPCDNTB6-6vdKxjk3LqSkJVRU/edit#gid=0>

Définition des phases :

1. Maintenir une vitesse constante (Cruise control) avec contrôles (boutons et affichage)
2. Apprentissage et maîtrise des instruments de mesure et des capteurs
3. Implémentation des capteurs au Cruise control pour augmentation de vitesse (contrôle du moteur)
4. Implémentation des capteurs au Cruise control pour réduction de vitesse (contrôle des freins)
5. Design final et fermeture des circuits

À faire aujourd'hui

03-08

8 mars 2023 08:48

- Validation du calendrier et questions
- Démontage de la trottinette

En mesurant la résistance du circuit contrôlant la gâchette, aucune valeur de sortie. On conclue donc que la gâchette n'est pas un potentiomètre et que le signal n'est pas une variation de résistance.

La seule autre possibilité serait donc que la gâchette envoie une différence de potentiel au système.

Mesure du voltage de la gâchette :

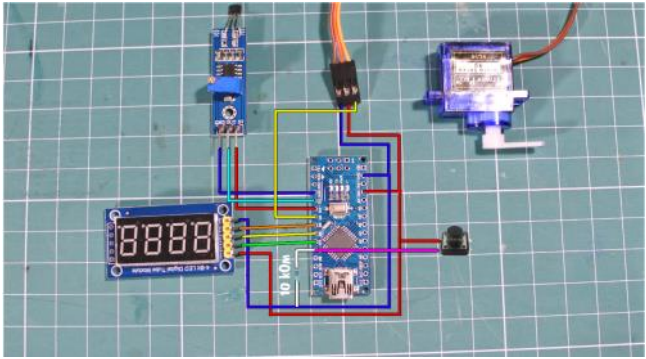
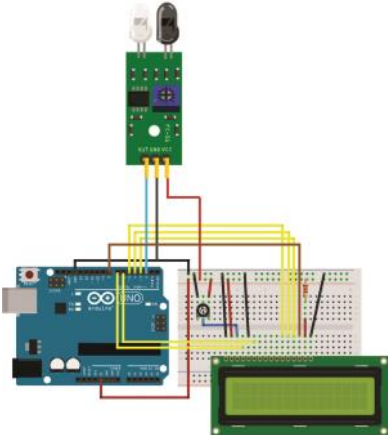

	Rouge	Bleu	Noir
Rouge		3.7V	4.5V
Bleu	3.7V		0.8V
Noir	4.5V	0.8V	

Pour que deux systèmes indépendants fonctionnent ensemble, il faut qu'il partage le même ground. Il faudra tenir compte de cela pour contrôler la trottinette avec notre propre système.

Tests de la distance de freinage à 15.5 mph : environ 5-7m. Un capteur lidar de 12m de portée serait donc suffisant.

Recherches sur la construction d'un speedomètre :

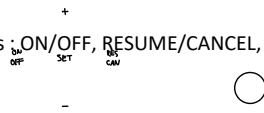
Méthode	Nous allons utiliser un capteur à effet hall qui détecte un aimant sur la roue de la trottinette.
---------	---

Modèle	Construction	Codage
Aiguille (Hall sensor)		https://github.com/AlexGyver/EnglishProjects/tree/master/Arduino_speedometer
Écran (AR sensor)		 bike-speed ometer

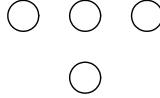
Nous développons l'interface physique aujourd'hui.

Boutons

Nous aurons besoins de 5 boutons : ON/OFF, RESUME/CANCEL, SET, +, -



Voici un schéma du placement de chaque bouton :



Bouton	Système
ON/OFF	Réinitialise la valeur de la variable "vitesse_set" et change la valeur de la variable "is_on" de true à false.
SET	Définit la valeur de la variable "vitesse_set" qui est la vitesse à maintenir.
+	Incrémente la valeur de la variable "vitesse_set" de 1 km/h.
-	Décrémente la valeur de la variable "vitesse_set" de 1 km/h.
RES/CAN	Change la valeur de la variable "is_activated" de true à false.

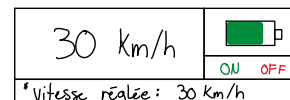
Branchement d'un bouton sur un Arduino : [PUSH BUTTON ARDUINO Tutorial | PUSH BUTTON ARDUINO Nano with Code](#)

Si notre circuit requiert plus de port Arduino qu'il n'y en a de disponible, une méthode utilisant des résistances pour brancher plusieurs boutons sur la même pin existe. Il suffit d'analyser dans le code quelle est la variation de résistance dans le circuit (en ayant une résistance différente associée à chaque bouton).

Écran

Nous aurons besoin d'un écran d'environ 15cm x 10cm qui affichera l'état ON/OFF du cruise control, la vitesse actuelle et l'état du cruise control (ex.: vitesse set / en pause / aucune vitesse).

Bouton	À l'écran
ON/OFF	Change le voyant lumineux selon l'état (ON ou OFF). En passant de OFF à ON, le bas de l'écran affiche "Veuillez définir une vitesse".
SET	Le bas de l'écran affiche maintenant "Vitesse réglée : 00 km/h".
+	Change la valeur affichée au bas de l'écran.
-	Change la valeur affichée au bas de l'écran.
RES/CAN	CANCEL : affiche dans le bas de l'écran "Appuyez sur RESUME". RESUME : affiche la vitesse réglée



* s'allume en fonction de l'état
* n'affiche rien si off/cancel

* appuyez sur RESUME

Pour mieux orienter nos recherches, nous avons décidé d'attendre de choisir l'écran parmi les propositions du professeur avant de faire notre plan de montage et de programmation.

Hacking

Nous avons déterminé que pour contrôler la vitesse de la trottinette, il suffit d'envoyer un signal (probablement analogue) au système intégré de la trottinette imitant celui de la gâchette. Nous pourrions donc brancher un Arduino sur la gâchette pour d'abord connaître les valeurs qu'elle envoie au système avant de les reproduire.

Il faudra aussi trouver un moyen de faire coexister les deux systèmes.

En attendant que le professeur se libère et nous fournisse les outils nécessaires à l'étude de la trottinette, nous avons commencé une ébauche du code pour l'interface physique avec les boutons.

Hacking de la trottinette :

À l'aide d'un multimètre équipée d'une fonction de logique, nous cherchons à déterminer si la sortie est de type analogue ou digitale. Nous n'avons pas réussi à trouver une réponse concluante.

Nous allons donc brancher un contrôleur Arduino sur la trottinette pour analyser le signal produit par la gâchette pour ensuite le reproduire avec notre programme. Pour le branchement, nous avons déterminé que **le GND de la trottinette est le fil noir**. Voir le fichier "hack_sketch".

Nous venons de réaliser que le fil utilisé n'était pas celui de la gâchette pour tous nos tests depuis les deux dernières semaines.

Nous avons mesurer à nouveau la résistance :

Noir rouge : 762 k Ω

Noir vert: 0

Rouge vert: 0

Nous avons mesurer à nouveau la tension :

Noir rouge : 4.27V

Noir vert : variable entre 0.8 à 3.92V

Rouge vert : variable entre 1.15 à 3.46V



Conclusion : nous voulons envoyer un signal analogue qui varie entre **220** et **711** en valeur analogue sur la pin verte.

Nous cherchons maintenant à déterminer comment brancher sécuritairement le Arduino au circuit de la trottinette pour faire coexister les deux systèmes (le microcontrôleurs et la gâchette) tout en reproduisant le signal à l'aide du Arduino. Nous pouvons brancher le contrôleur directement sur la prise 5V du circuit de la trottinette sans problème.

Il faut créer un signal PWM (Pulse Width Modulation) sur une pin PWM (3-5-6-9-10-11, celles avec un tilde).

Nous avons essayé de contrôler la trottinette avec un potentiomètre externe pour simuler le contrôle fait par notre futur programme. Nous n'arrivons pas à retirer le message d'erreur qu'affiche la trottinette et donc n'avons pas réussi à actionner le moteur. Il faudra analyser davantage comment est traité le signal envoyé par la gâchette.

Réflexion : serait-il possible que pour faire disparaître le message d'erreur, le signal ne doit jamais atteindre 0? Puisque la valeur analogue minimale lue pour le signal est de 220 et la maximale de 711, il faudrait toujours qu'un certain voltage soit fourni au circuit?

Oui.

On continue l'ingénierie inversée de la trottinette pour réussir à la contrôler avec notre Arduino.

Hacking

Nous avons réussi à démonter la gâchette : **il s'agit d'un capteur à effet Hall muni de deux aimants**. Cela explique pourquoi le voltage n'est jamais de zéro. Il faut donc reproduire son fonctionnement avec notre Arduino. Des recherches sur le fonctionnement d'un capteur à effet Hall sont de mise.

↗ ↘

En faisant face au capteur à effet Hall, à partir de la gauche, il y a le fil vert, noir, et rouge :

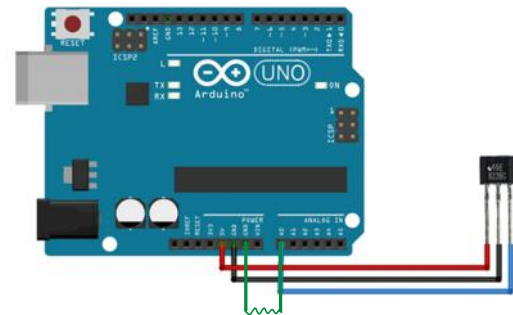
Pour analyser le signal fourni par le capteur, nous allons le brancher sur un Arduino comme suit :

<https://electropeak.com/learn/interfacing-ss49e-linear-hall-effect-sensor-module-with-arduino/>

Notre première lecture est inconcluante. Nous avons une variation de 1 (999 à 1000), ce qui ne fait pas de sens. Nous considérons donc qu'une résistance soit intégrée au circuit de la trottinette. Nous allons donc l'inclure à notre circuit temporaire (celui tiré du site ci-haut). **Nous l'avons mis entre le GND et le signal.**

Résistances testées :

Résistance (Ω)	Val analogue min	Val analogue max
0	999	1000
10k	806	818
100k	1000	1000
1k	354	787
300	165	790



Puisque la résistance de 300Ω est celle offrant une plus grande variation entre son min et son max, nous l'utiliserons pour notre circuit.

Nous avons réussi à contrôler le moteur en utilisant la gâchette qui était branchée dans le Arduino. Nous devons trouver la bonne intervalle de sortie pour éviter d'avoir le message d'erreur, puisqu'à certaines valeurs le message apparaît, alors qu'à d'autre la roue tourne sans problème. Nous avons même réussi à utiliser un potentiomètre au lieu de la gâchette. Nous sommes très près du but.

Boutons

Le code pour les boutons de l'interface physique a été finalisé en parallèle. Il a été testé avec un seul bouton et semble fonctionner comme il le faut. Il ne reste plus qu'à construire le circuit avec les 5 boutons et à inclure l'écran à l'interface physique et sa gestion dans le code.

```
// Définition des pins pour chaque bouton
const int set_buttonPin = 3;
const int on_off_buttonPin = 2;
const int res_can_buttonPin = 7;
const int plus_buttonPin = 5;
const int minus_buttonPin = 4;
// État d'un bouton (1 lorsqu'il est appuyé)
int buttonState = 0;
// État du Cruise control (ON/OFF et ACTIVÉ/EN PAUSE)
bool is_on = false;
bool is_activated = false;
// Valeurs fictives
float vitesse_set = 0;
float vitesse_trottinette = 20;
void setup() {
  Serial.begin(9600);
  // Initialize the button inputs:
  pinMode(set_buttonPin, INPUT);
  pinMode(on_off_buttonPin, INPUT);
  pinMode(res_can_buttonPin, INPUT);
  pinMode(plus_buttonPin, INPUT);
  pinMode(minus_buttonPin, INPUT);
}
```

```

void loop() {
  // Read if on_off was pressed
  buttonState = digitalRead(on_off_buttonPin);
  if (buttonState == HIGH) {
    if(is_on) { // Turn off
      is_on = false;
      is_activated = false;
      vitesse_set = 0;
    }
    else { // Turn on
      is_on = true;
      is_activated = false;
      vitesse_set = 0;
    }
  }

  // Read next buttons only if on
  if(is_on) {
    // Read if set was pressed
    buttonState = digitalRead(set_buttonPin);
    if (buttonState == HIGH) {
      vitesse_set = vitesse_trottinette;
    }
  }
  // Read next buttons only if a speed was set
  if(vitesse_set > 0) {

    // Read if res_can was pressed
    buttonState = digitalRead(res_can_buttonPin);
    if (buttonState == HIGH) {
      if(is_activated) { // Turn off
        is_activated = false;
      }
      else { // Turn on
        is_activated = true;
      }
    }
  }
  // Read next buttons only if activated
  if(is_activated) {
    // Read if + was pressed
    buttonState = digitalRead(plus_buttonPin);
    if (buttonState == HIGH) {
      ++vitesse_set;
    }
  }
  // Read if - was pressed
  buttonState = digitalRead(minus_buttonPin);
  if (buttonState == HIGH) {
    --vitesse_set;
  }
}
}
}
}
}
}
}

```


Afin de trouver l'intervalle de données valides que la trottinette prend en entrée, nous avons branché un Arduino muni d'une boucle incrémentant une valeur à chaque seconde et avons noté les valeurs entraînant le message d'erreur :

Valeur analogue	Erreur
0	Oui
...	Oui
8	Oui
9	Non
10	Non
...	Non
50	Non
...	Non
60 (roue)	Non
...	Non
110	Non
...	Non
168	Non
169	Oui
170	Oui
...	Oui
260	Non
...	Non

```
void setup() {
  Serial.begin(9600);
  pinMode(2, INPUT); // button (pause count when pressed)
  pinMode(9, OUTPUT); // pin PWM
}
int value = 0;
void loop() {
  if(digitalRead(2) != HIGH) {
    analogWrite(9, value);
    Serial.println(value);
    delay(1000);
    value++;
  }
}
```

En testant les bornes, j'ai remarqué que si la trottinette s'éteint et qu'en la rallumant une valeur valide est déjà envoyée au système, le message d'erreur s'affiche. Ce phénomène a été observé notamment lorsque la trottinette s'éteint automatiquement (autour de 150) et même si la valeur initiale est baissée à 100, le message d'erreur ne s'enlève pas. **Il faut absolument repartir de 9.**

Il est toutefois possible de partir à 9, puis de changer la valeur pour 100 directement (sans incrémenter entre les deux) sans afficher de message d'erreur.

En faisant tourner la roue, les valeurs changent. Il faut une valeur de **60 pour lancer le mouvement de la roue**, et la **valeur maximale avant d'afficher une erreur est de 168**.

Lorsqu'on atteint 260, la roue se remet toutefois à tourner. En diminuant la valeur du délai, nous avons déterminé que les valeurs font un cycle régulier. Nous notons aussi que la puissance fournie par le moteur donne des coups. Le délai n'est pas responsable des coups non plus (nous avons essayé de l'enlever complètement et cela n'a aucun impact sur le mouvement de la roue pour une valeur constante). Nous allons donc refaire ce test en ajoutant au circuit un **filtre RC**.

Recherches sur le filtre RC

https://web.stanford.edu/class/archive/engr/engr40m.1178/slides_sp17/lecture14.pdf

<http://sim.okawa-denshi.jp/en/CRLowkeisan.htm> (calculatrice)

Fin du cours :

Il nous reste à trouver les bonnes valeurs à utiliser dans la fonction `map()` de notre sketch "hack". Les bornes ne sont pas respectées (`i = map(i,155,755,9,168);`). Il faut aussi inclure notre filtre RC à notre circuit semi-temporaire.

```
i = map(i,155,755,9,168); // Serial.print(i) affiche 6
```

).

Il faut aussi inclure notre filtre RC à notre circuit semi-temporaire.

Nous sommes maintenant prêt à concevoir le plan du circuit fonctionnel.

03-30

31 mars 2023 12:44

Réalisation du plan du contrôle de la trottinette.

Branchement des composantes requises pour le contrôle de la trottinette.

Ajustement du `map()` : nous avons réussi à retirer complètement le message d'erreur en changeant les bornes (augmenter le minimum et réduire le maximum) de la sortie. Cette méthode n'est toutefois pas optimisée.

En débranchant le Arduino de l'ordinateur (sans voir les valeurs du `map`) nous n'arrivons plus à atteindre la vitesse maximale de la trottinette. Nous supposons que le voltage de 5V fourni par l'ordinateur est trop grand comparé à celui fourni par la trottinette.

Nous avons ajouté les boutons au circuit. Il ne nous reste plus qu'à jumeler le code des sketches "boutons" et "hack" pour que le Cruise control devienne fonctionnel. Il faudra par la suite installer le speedomètre pour que la vitesse utilisée par le programme soit la vraie et non celle basée sur la donnée envoyée par gâchette.

04-05

5 avril 2023 09:04

Trouvé les bornes

Juste du code

Test --> cruise fonctionnel

Fin du cruise

04-08

12 avril 2023 09:53

Programmation d'un sketch pour l'utilisation d'un joystick plutôt que des boutons.

Les 4 directions fonctionnent, mais le bouton non. Il est détecté à des moments aléatoires, mais pas lorsqu'il est appuyé.

04-12

12 avril 2023 09:02

Puisque nous avons accumulé un certain retard, nous allons essayer de travailler sur des éléments différents le plus possible à partir de maintenant afin d'avancer plus rapidement.

Sébastien

Écran maintenant fonctionnel (OLED SSD1315). Ajouté à la trottinette.

Puisque nous ne pouvons pas avoir d'écran plus grand et que l'utilisation de plusieurs petits écran rend les choses plus complexes, nous avons décidé d'utiliser des lumières pour afficher l'état des variables booléennes à la place.

Le problème du joystick était la logique du code. Le tout est maintenant fonctionnel. Le sketch a été ajouté au code de la V2.

DIFFÉRENCE V1 / V2 : la version 1 était celle avec 5 boutons et sans l'emploi d'un écran (le débogage se faisait à l'aide de la console). La version 2 est dotée d'un écran, de voyants lumineux, d'un écran et d'un Arduino Mega plutôt que Uno. Il s'agit donc de la version finale de notre Cruise Control classique.

L'exécution de la V2 est très lente. Il faut donc trouver une solution à ce problème.

En déboguant, j'ai réussi à trouver la source du problème : il s'agit de l'écran OLED. Je vais donc utiliser un écran LCD plutôt qu'OLED. Après ajustements, l'interface physique est maintenant terminée. Il ne reste qu'à faire le montage final.

Un nouveau plan avec les composantes changées devra être fait. Nous allons aussi changer le microcontrôleur pour un Arduino Mega afin de ne pas manquer de pins ou d'espace mémoire.

```
// Définition des pins
const int button_Pin = 2;
const int x_Pin = A1;
const int y_Pin = A2;

// État du bouton et des deux axes
int buttonState = 0;
int xValue = 0;
int yValue = 0;

// Variable décrivant l'état actuel du joystick
String state = "ERROR";

void setup() {
  Serial.begin(9600);

  // Initialize the inputs:
  pinMode(button_Pin, INPUT_PULLUP);
  pinMode(x_Pin, INPUT);
  pinMode(y_Pin, INPUT);
}

void loop() {

  // Lecture des valeurs retournées par le joystick
  xValue = analogRead(x_Pin);
  yValue = analogRead(y_Pin);
  buttonState = digitalRead(button_Pin);

  // Vérification si le bouton a été appuyé avant tout
  if(buttonState != HIGH) {
    state = "PRESS";
  }

  // Si le bouton n'a pas été appuyé, vérification de la direction où le joystick pointe
```

```

else {
  // Up
  if((xValue <= 1000 && xValue >= 500) && (yValue <= 500 && yValue >= 0)) {
    state = "UP";
  }
  // Down
  else if((xValue <= 1000 && xValue >= 500) && (yValue >= 1000)) {
    state = "DOWN";
  }
  // Right
  else if((xValue >= 1000) && (yValue <= 1000 && yValue >= 500)) {
    state = "RIGHT";
  }
  // Left
  else if((xValue <= 500 && xValue >= 0) && (yValue <= 1000 && yValue >= 500)) {
    state = "LEFT";
  }
  else {
    state = "x";
  }
}

if(state != "x")
Serial.println(state);
delay(100);
}

```


04-13

13 avril 2023 12:02

Réalisation du plan pour le circuit utilisant le code V2.

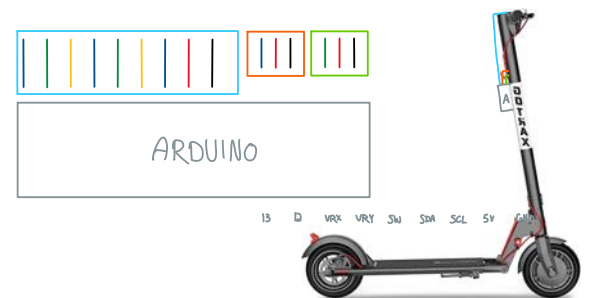
Je veux pouvoir retirer facilement les composantes du projet de ma trottinette afin de pouvoir l'utiliser en dehors des cours. Il faut aussi qu'elles soient faciles à remettre au début de chaque cours pour éviter de perdre trop de temps à réinstaller / retirer les différents modules chaque fois. Il faut donc trouver une solution à ce problème.

Réflexion sur où / comment placer chaque élément sur la trottinette en prenant compte des éléments suivants :


- être facilement démontable (pour pouvoir utiliser ma trottinette en dehors du cours);
- avoir l'interface physique facilement accessible pour l'utilisateur;
- filage;
- encombrement du guidon.

Voici l'idée qui est ressortie :

Faire un gros paquet de fil entre le Arduino et l'interface physique. Voir s'il est possible de créer notre propre connecteur en plastique pour n'avoir "qu'un seul fil" entre l'interface physique et l'Arduino. Puisqu'il s'agit d'un montage semi-temporaire, nous pourrions utiliser des fils encore collés entre l'Arduino et les éléments, qui sont divisés avec d'autres fils dans leurs boîtiers respectifs. Nous pourrions peut-être utiliser les boîtes en plastique vert si nous pouvons en avoir plusieurs. Il y aurait donc des femelles sortant de chaque boîtier (et fixés sur place) et un double mâle les reliant. Les femelles effectueraient les connexions à l'intérieur du boîtier. Nous aurions quatre autres connecteur se rendant au Arduino (qui serait placé près du bouton d'alimentation) : la gâchette, le contrôle de la trottinette, le speedomètre et les capteurs. Nous pourrions retirer l'entrée de charge de la trottinette pour faire passer la gâchette et le contrôle de la trottinette, ce qui permet de fermer la trottinette tout en y ayant accès par l'extérieur de la façon la moins invasive possible.



Circuit de la V1 retiré de la trottinette afin d'installer les boîtiers de la V2. Beaucoup d'attente pour les outils / matériaux, donc recherches sur le Lidar faites en parallèle.

Lidar temporaire : TOF050F (Max Distance: 50cm)	 202102240 33236TOF...
---	---

Puisque le Lidar utilise le port I2C et que notre écran aussi, nous devons utiliser leur adresse. Voici donc un code pour y parvenir :

```
#include <Wire.h>

void setup()
{
  Wire.begin();
  Serial.begin(115200);
  Serial.println("\nI2C Scanner");
}

void loop()
{
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 0; address <= 127; address++ )
  {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();
    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address, HEX);
      Serial.println(" !");
      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknow error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");
  delay(3000);
}
```

À partir de l'adresse <<https://www.electronicclinic.com/tof10120-laser-range-finder-arduino-display-interfacing-code/>>

Adresses I2C :

Écran Grove-LCD RBG Backlight	0x03 0x3E 0x62 0x70
TOF050F Lidar Sensor	0x29

Plusieurs tests effectués avec des bibliothèques d'autres marques de Lidar. Il est impossible de lire les données lues par le capteur.

En essayant de brancher deux appareils sur les ports I2C, il est très difficile d'y avoir accès. Il faut faire plus de recherches si nous voulons utiliser un port I2C pour le Lidar au lieu du port série.

Boitier de l'interface physique fait et fonctionnel. Il est prêt à être installé sur la trottinette, mais ne restera pas fixé dessus pour pouvoir utiliser la trottinette en dehors du cours.

Soudure d'une troisième branche sur les fils de la trottinette. Le montage / démontage se fait donc maintenant très rapidement. Finition du circuit aussi (ajout du filtre RC, de colle chaude pour éviter les courts-circuits, etc.).

Problème décelé : lorsque l'écran LCD est alimenté par la trottinette, il est presque impossible de lire ce qu'il affiche. Ce problème est probablement dû aux fils utilisés : ils n'arrivent pas à fournir assez de courant au circuit. Il faudra donc ajouter une batterie 9V pour alimenter le microcontrôleur et obtenir de meilleurs résultats.

En ajoutant la batterie de 9V, l'écran fonctionne, mais un message d'erreur s'affiche sur la trottinette. La cause est probablement les bornes du code. Ce serait donc le même problème que nous avons lors du débogage : il fallait utiliser des valeurs min et max différentes si le Arduino était branché dans l'ordinateur.

En effet, le problème était lié aux bornes. Il a été réglé.

Ajout du bouton ON/OFF de la trottinette (pas du CC) sur le boîtier.

04-21

21 avril 2023 12:05

Réception du hall sensor et du Lidar. Début du speedomètre et du capteur de distance.

Speedomètre

Les templates trouvées précédemment ne fonctionnent pas. Il faut donc écrire notre propre code. Brancher le hall sensor dans le Arduino est assez simple, mais calculer la vitesse est plus complexe. Je n'ai pas réussi à le faire pour le moment. À travailler.



NJK-5002C Hall Effect Proximity Sensor

The NJK-5002C is a magnetic Hall Effect proximity sensor which is used as an indicator of position or proximity of magnetic materials. When the NJK-5002C is close to magnetic object, its output sends a control signal in addition to having a status indicator LED which visually supports us in the detection. It can also be used as a speed counter.



SKU: [SSR1047](#)

Brief Data:

- Model: NJK-5002C
- Supply voltage: 5-30VDC
- Detection Distance: 10mm (effective detection distance 0-10mm).
- Load current: <150mA
- Output Form: NPN.
- Output state: Normally Open (NO)
- Detection Objects: Magnetic Material.
- Output Indication: LED (red).
- Probe Dimension: Ø12x32mm.
- Cable Length: 110cm.
- Weight: ~40g.

Mechanical Dimension:

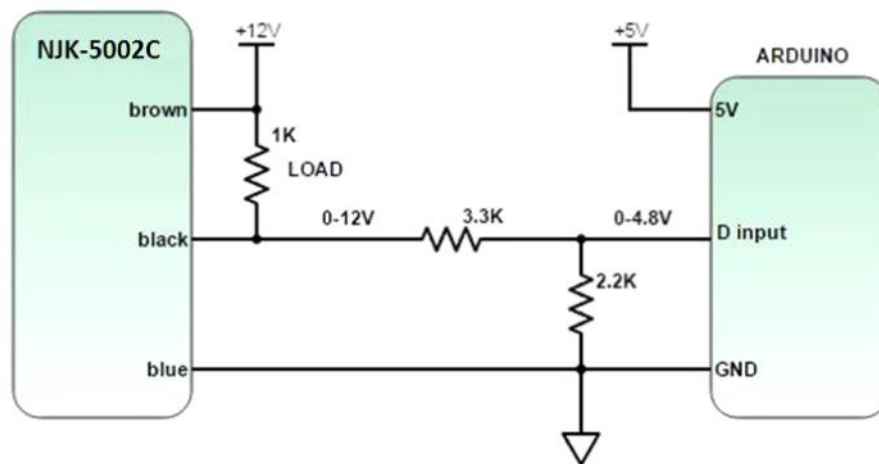
Unit: mm



Terminal Assignment:



How to Connect to Arduino Board ?





Handsontec.com

We have the parts for your ideas

HandsOn Technology provides a multimedia and interactive platform for everyone interested in electronics. From beginner to diehard, from student to lecturer. Information, education, inspiration and entertainment. Analog and digital, practical and theoretical; software and hardware.



HandsOn Technology support Open Source Hardware (OSHW) Development Platform.

Learn : Design : Share

www.handsontec.com



The Face behind our product quality...

In a world of constant change and continuous technological development, a new or replacement product is never far away – and they all need to be tested.

Many vendors simply import and sell without checks and this cannot be the ultimate interests of anyone, particularly the customer. Every part sell on Handsontec is fully tested. So when buying from Handsontec products range, you can be confident you're getting outstanding quality and value.

We keep adding the new parts so that you can get rolling on your next project.



www.handsontec.com

[Breakout Boards & Modules](#)



[Connectors](#)



www.handsontec.com

[Electro-Mechanical Parts](#)



www.handsontec.com

[Engineering Material](#)



www.handsontec.com

[Mechanical Hardware](#)



[Electronics Components](#)



www.handsontec.com

[Power Supply](#)



[Arduino Board & Shield](#)

Tools & Accessory



www.handsontec.com

[Tools & Accessory](#)

Speedomètre

J'ai finalement fait le code moi-même avec l'aide de ChatGPT. Il est maintenant fonctionnel. Il reste simplement à mesurer le diamètre de la roue. Le principe du speedomètre est le suivant : on compte le nombre de révolutions de la roue (selon le nombre de fois où l'aimant est détecté) pour un certain laps de temps. À partir de ces deux variables, nous pouvons déterminer la vitesse de la roue. Si le délai de lecture est trop grand, la vitesse est alors nulle.

```
// Définition de la pin
const int hall_pin = 18;

// Définition de constantes
const float DIAMETRE_ROUE = 0.255;
const int AIMANT_PAR_TOUR = 1;
const int DELAI_MESURE = 100; // Délai entre chaque mesure pour filtrer le bruit (ex.: à basse vitesse, on détecte l'aimant trop longtemps, etc.)
const int DELAI_ZERO = 900; // Délai permettant de déterminer si la vitesse est nulle (aucune mesure depuis trop longtemps)

// Définition de variables
unsigned int compteur_revolutions = 0; // Compteur de révolutions
float vitesse = 0;
unsigned long temps_precedent = 0; // Temps précédent pour calculer la vitesse

// Méthode indépendante de la fonction loop() permettant de compter le nombre de tours pendant l'exécution du code (évite les pertes de données)
void detecter_changement_etat() {
    if (digitalRead(hall_pin) == HIGH) { // Si l'état du capteur est HIGH (aimant détecté)
        compteur_revolutions++; // Incrémenter le compteur de révolutions
    }
}

void setup(){
    Serial.begin(9600);
    pinMode(hall_pin, INPUT_PULLUP);
    attachInterrupt(digitalPinToInterrupt(hall_pin), detecter_changement_etat, CHANGE); // Attacher l'interruption pour détecter les changements d'état du capteur
}

void loop(){
    unsigned long temps_actuel = millis(); // Temps actuel en millisecondes
    unsigned long delai_ecoule = temps_actuel - temps_precedent; // Calcul du temps écoulé depuis la dernière mesure
    float distance_par_tour = DIAMETRE_ROUE * PI; // Calcul de la distance parcourue par tour en mètres
    float distance_totale = compteur_revolutions * distance_par_tour; // Calcul de la distance totale parcourue en mètres
    if (delai_ecoule >= DELAI_MESURE && compteur_revolutions != 0) { // Si le délai entre chaque mesure est atteint
        temps_precedent = temps_actuel; // Mettre à jour le temps précédent

        // Calcul de la vitesse en m/s
        vitesse = (distance_totale / compteur_revolutions) / (delai_ecoule / 1000.0);

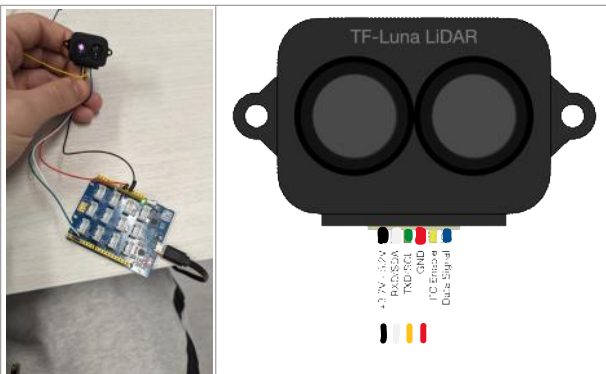
        // Conversion en mph
        vitesse *= 2.23694;
    }

    compteur_revolutions = 0; // Réinitialiser le compteur de révolutions

    // Si aucune mesure depuis trop longtemps, la vitesse est nulle
    if (delai_ecoule >= DELAI_ZERO) {
        vitesse = 0;
    }
    Serial.println(vitesse);
}
```

Lidar

Après plusieurs essais de branchement, le lidar fonctionne en étant plug dans un port UART. Il reste cependant étrange que notre branchement utilise le fil noir pour l'alimentation, mais il ne s'agit pas d'une erreur. Nous détectons maintenant une distance, mais aucun calcul n'est effectué pour le moment.



Code :

```
/*
  This program is the interpretation routine of standard output protocol of TFmini-Plus product on Arduino.
  For details, refer to Product Specifications.
  For Arduino boards with only one serial port like UNO board, the function of software visual serial port is
  to be used.
  */
#include <SoftwareSerial.h> //header file of software serial port
SoftwareSerial Serial1(2,3); //define software serial port name as Serial1 and define pin2 as RX and pin3
/* For Arduinoboard with multiple serial ports like DUEboard, interpret above two pieces of code and
directly use Serial1 serial port*/
int dist; //actual distance measurements of LiDAR
int strength; //signal strength of LiDAR
float temprature;
int check; //save check value
int i;
int uart[9]; //save data measured by LiDAR
```

Source : <https://forum.arduino.cc/t/arduino-mega-serial-ports/689085>



Journal Page 36

Le code de la V3 a été avancé. Il intègre le speedomètre et le lidar à la V2 ainsi qu'une nouvelle variable "speed_output" (vitesse_trottinette était jusqu'alors utilisée pour la valeur analogue envoyée par le système, mais décrit plutôt la vitesse réelle).

Il y a un problème de logique quelque part qui empêche d'obtenir la vitesse.

Comme nous sommes bientôt à cours de fils et que le code du lidar et du capteur à effet hall ne sont pas standard, les couleurs ont toutes été mélangées. Voici donc un récapitulatif des branchements à faire en dehors du boîtier :

Contrôle de la trottinette

	Origine	Boîtier
GND	F noir	Grove x3
5V	F rouge	x
Input	F vert	M vert
Output	M bleu	F orange

Speedomètre

	Origine	Boîtier
GND	M bleu	Grove x3
5V	M orange	F rouge
Signal	M blanc	F bleu

Lidar

	Origine	Boîtier
GND	M rouge	Grove x3
5V	M noir	F rouge
Signal 3	M blanc	F mauve
Signal 2	M jaune	F vert

À noter que les fils identiques sortant du boîtier ont la même fonction. Tous ceux ayant une fonction spécifique sont uniques.

05-03

3 mai 2023 09:06

Au dernier cours, je pensais qu'un problème de logique empêchait d'obtenir la vitesse du speedomètre. Après l'avoir révisé plusieurs fois au cours de la semaine, je n'ai rien trouvé. Je suis donc revenu au sketch de base utilisé lors des tests. Même ce sketch ne fonctionne plus. Le problème n'est pas dû à un faux contact ou une soudure brisée, puisque le code peut lire la valeur du capteur à effet hall.

Problème trouvé : il faut utiliser la pin 2 ou 3 avec le capteur à effet hall. Toute autre pin digitale ne fonctionne pas.

Nouveau problème : En voulant faire des tests sur le lidar, nous avons essayé de remettre son sketch sur le Arduino Uno. Ce sketch ne fonctionne plus non plus.

Problème réglé : nous avons testé les soudures et les connexions avec un multimètre. Tout semble normal. En rebranchant le tout après les tests, tout est fonctionnel.

Autre problème : le lidar et le speedomètre doivent tous les deux être branché sur la pin 2.

Problème réglé : la fonction `attachInterrupt()` ne peut utiliser que certaines pins. Dans le cas d'un Arduino Uno (utilisé pour les tests), seules les pins 2 et 3 peuvent être utilisées. Dans le cas d'un Arduino Mega, nous pouvons prendre la pin 18.

attachInterrupt()

[External Interrupts]

Description

Digital Pins With Interrupts

The first parameter to `attachInterrupt()` is an interrupt number. Normally you should use `digitalPinToInterrupt(pin)` to translate the actual digital pin to the specific interrupt number. For example, if you connect to pin 3, use `digitalPinToInterrupt(3)` as the first parameter to `attachInterrupt()`.

BOARD	DIGITAL PINS USABLE FOR INTERRUPTS
Uno, Nano, Mini, other 328-based	2, 3
Uno WiFi Rev.2	all digital pins
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21
Micro, Leonardo, other 32u4-based	0, 1, 2, 3, 7
Zero	all digital pins, except 4
MKR Family boards	0, 1, 4, 5, 6, 7, 8, 9, A1, A2
Due	all digital pins
101	all digital pins (Only pins 2, 5, 7, 8, 10, 11, 12, 13 work with CHANGE)

Test des données du Lidar

Distance "0"	Distance lue lorsque la roue touche au mur	14 cm
Distance sécuritaire	Distance définie comme étant la valeur minimale à respecter	250 cm
Distance de	Distance définie comme étant la valeur à laquelle la trottinette	500 cm

ralentissement	commence à ralentir.	
Distance d'incertitude	Incertitude à respecter lors des mesures des distances pour assurer la sécurité	10 cm

La précision semble très bonne (environ 3-4 cm) même à 5 m de distance.

Test du CC avec le speedomètre fonctionnel

La logique des if du loop de la V3 ne fonctionne plus depuis l'intégration de la variable speed_output. Il est aussi possible qu'une boucle sans fin se trouve quelque part, puisqu'après avoir set une vitesse, tous les boutons deviennent non fonctionnels. À corriger.

05-04

4 mai 2023 16:45

Création du programme V4 :

- Reconstruction du code complet avec une approche orientée objet afin d'alléger la fonction loop() et mieux diviser chaque module physique. Chaque module aura sa propre classe (ex.: joystick, display, etc.).
- Utilisation de fichiers .cpp et .h.
- Changement des commentaires et méthodes pour les mettre en français.
- Révision de la logique du code.

05-05

5 mai 2023 14:00

Pendant que Sébastien débogue la V4, Yohan avance le rapport synthèse.

Après 1h30 de débogage pour essayer de comprendre pourquoi la vitesse donnée par la classe Speedometre est toujours égale à 0, j'ai finalement réalisé que la méthode Calculer_vitesse, qui calcule la vitesse en fonction du nombre de révolutions de la roue, n'était jamais appelée.

Fonctionnalités à retravailler : boutons + et -, obtenir la vitesse réelle lorsque le CC est en marche est en marche.

05-10

10 mai 2023 01:29

En débuggant la V4 afin de régler les problèmes actuels, j'ai remarqué que l'aimant collé sur la roue est tombé. Je n'aurai donc pas fait de tests pendant le cours d'aujourd'hui. Je vais donc plutôt travailler sur le rapport synthèse.

05-12

17 mai 2023 01:01

Fin de l'analyse synthèse à remettre avant la fin des cours.

05-13

25 mai 2023 10:30

Création de la V5 : tentative de faire fonctionner proprement la version orientée objet comprenant le speedomètre (V4) en conservant l'utilisation de classes, mais en rendant tout public. Même s'il s'agit de code peu optimisé et qui ne respecte pas les conventions de programmation, l'important est qu'il soit fonctionnel.

Simplement rendre tout publique ne suffit pas à résoudre notre problème.

05-15

25 mai 2023 10:30

Toutes les classes dont un seul objet est nécessaire (speedomètre, cruise control, contrôle moteur, etc.) ont été modifiées pour devenir statique.

Création d'une solution VStudio afin de séparer chaque classe dans différents fichiers .cpp

Compilation réussie, mais impossible de le tester avec la trottinette puisque je n'y ai pas accès pour le moment.

05-16

25 mai 2023 10:30

Réflexion sur le comportement du CC adaptatif :

- si un objet se trouve devant la trottinette, on change temporairement la vitesse set pour ralentir
- plus l'objet est proche, plus l'avertisseur sonore sonne fort

Recherches sur l'utilisation d'un buzzer

Sauvegarde d'un lien pour l'avertisseur sonore qui sera bientôt intégré au système :

<https://www.ardumotive.com/how-to-use-a-buzzer-en.html>

Si les changements apportés (classes static et public) sont insuffisants à régler le problème actuel :

Réflexion : le problème de la V4 (la vitesse affichée lorsque le CC est en marche n'est pas la bonne) pourrait être dû à ce qui suit : **l'exécution des méthodes de la classe Cruise_control empêche le bon fonctionnement de l'interruption?** Cela expliquerait pourquoi il n'y pas de problème lorsque la trottinette est contrôlée par la gâchette, mais seulement lorsque le CC est activé.

Piste de solution : s'il s'agit de la méthode utilisée pour contrôler la valeur analogue à envoyer au moteur (incrémenter jusqu'à ce que la vitesse set soit atteinte), il serait possible de faire de plus gros pas que 1 (ex.: += 5) et d'ajouter un délai avant la prochaine incrémentation afin de limiter l'utilisation de la mémoire vive de cette méthode.

À ajouter : vérification que les bornes sont respectées lors du contrôle du moteur (si la vitesse set n'est pas respectée, on incrémente la valeur analogue jusqu'à ce que ce soit le cas, il faut donc mettre une limite si le moteur est déjà à 100% de sa capacité pour éviter les erreurs). Il serait aussi possible d'afficher sur l'écran lorsque cette situation se produit ("Moteur à 100%").

05-19

25 mai 2023 10:30

Encore des problèmes avec le speedomètre : il ne fonctionne plus à nouveau. J'ai assez rapidement découvert que le voyant lumineux du capteur à effet Hall ne s'allume pas, donc le problème vient soit de la connexion des fils avec le boîtier, les fils en tant que tel ou l'aimant du speedomètre.

La connexion a été vérifiée et tout semble bien.

Comme je n'ai pas de multimètre à ma disposition, je ne peux pas tester les fils pour savoir si l'un d'entre eux est endommagé.

Finalement, il s'agit de l'aimant : le capteur ne le détecte plus. Lorsque je prend un autre aimant, tout fonctionne bien. Il se pourrait donc que la colle chaude utilisée pour fixer le capteur à effet Hall n'était pas sèche lors des derniers tests et qu'il se soit déplacé légèrement depuis.

L'aimant a donc été décollé et remis en place.

Test de la V5 : mêmes problèmes que la V4.

Conclusion : ce n'est pas lié à l'encapsulation ou aux objets des classes.

En retirant la méthode Pause(), tout semble fonctionner correctement.

On observe pour la première fois comment la trottinette se comporte avec la méthode employée pour contrôler la vitesse avec notre CC (augmenter / diminuer la valeur analogue pour que la vitesse soit maintenue près de la vitesse set).

Le moteur donne beaucoup de coups et même en réduisant la constante d'erreur permise, ce problème est trop visible pour dire que nous sommes satisfait des résultats. Il faut donc trouver une autre méthode pour maintenir une vitesse stable.

Suite à une réflexion approfondie, ce problème pourrait être lié au fonctionnement même de notre speedomètre. En effet, puisque celui-ci compte le nombre de tours de la roue, il lui faut un certain moment (le temps que la roue fasse quelques tours) afin de calculer la vitesse. Notre programme serait donc trop vite par rapport à cette contrainte physique et augmente / diminue la valeur analogue plus rapidement que notre speedomètre ne peut calculer la vitesse en temps réel.

Pour contrevenir à ce problème, je vais essayer d'ajouter un compteur qui empêche la méthode MAJ_input de la classe Cruise_control de modifier la valeur analogue à chaque itération de la méthode loop().

En ajoutant un compteur, le problème est effectivement réglé. Toutefois, un nouveau problème survient : le moteur roule à pleine puissance lors du lancement du CC, avant de très doucement réduire sa vitesse pour arriver à la vitesse souhaitée, qui sera alors maintenue.

Pour régler le problème du moteur qui roule à pleine puissance lors du lancement du CC, deux solutions me viennent en tête :

1. Gérer la valeur initiale de l'input. Cela semble être la meilleure solution, mais après avoir réviser le code, cela ne semble pas être la cause du problème, puisque la valeur initiale est celle de la gâchette. Si on ne change pas l'environnement, la vitesse ne devrait donc pas augmenter comme elle le fait.
2. Au lieu d'augmenter / diminuer la valeur analogue par bons de 1, on pourrait **implémenter une nouvelle méthode qui retourne une valeur proportionnelle à l'écart entre la vitesse de la trottinette et la vitesse réglée**. En ayant des bons proportionnels, ce problème pourrait, à défaut d'être réglé, diminué, puisque la diminution jusqu'à atteindre la bonne vitesse serait plus rapide (plus l'écart est grand, plus les bons sont grands, donc plus rapide).

Esquisse de la méthode Proportion_ecart(unsigned int) :

La valeur de l'input doit être entre 155 et 755 pour imiter correctement la gâchette.
Cela laisse donc 600 valeurs possibles. Les bons de 1 sont donc beaucoup trop lents.

Invention de valeurs arbitraire pour me donner une idée de comment procéder :

Vitesse réelle (mph)	Vitesse set (mph)	Ecart1	Val. input réelle	Val. input voulue	Ecart2	Ecart2 / Ecart1
15	15	0	750	750	0	-
15	10	5	750	550	200	40
15	5	10	750	250	500	50
10	15	5	550	750	200	40
10	10	0	550	550	0	-
10	5	5	550	250	300	60
5	15	10	250	750	500	50
5	10	5	250	550	300	60
5	5	0	250	250	0	-

Premier test : multiplier l'écart par 9. Insatisfaisant.

Deuxième test : barème arbitraire en fonction de l'écart. Insatisfaisant.

Cette solution semble créer davantage de problèmes. J'ai donc abandonné cette idée. Je vais plutôt tenté de gérer l'input initial lors de sa mise en marche tout en conservant les bons de 1.

En ajoutant à ma fonction loop() un print pour afficher la valeur analogue de l'input au moment où le problème du moteur qui roule à pleine puissance lors du lancement du CC survient, voici ce que j'ai noté :

- on passe d'une valeur cohérente (entre 150 et 600) à plus de 7500 pour aucune raison.

JE NE COMPRENDS PAS

En essayant de régler le problème de différentes façons, la valeur de 7500 change souvent pour d'autres valeurs incohérentes comme -4200 ou -32000. Pourtant, je reviens toujours au même code qu'auparavant.

Problème **FINALEMENT** réglé : si la valeur de l'input n'est pas comprise entre les bornes, on lui donne une valeur valide de 520 (correspond à une vitesse moyenne).

Autre problème réglé (qui avait été ignoré jusqu'à maintenant) : la méthode Pause() est maintenant fonctionnelle. Le problème était qu'elle était appelée dès que le CC était en marche parce que la valeur de détection de la gâchette était trop petite et donc détectait que la gâchette avait été appuyé tout le temps plutôt que lorsqu'elle était vraiment enclenchée :

Avant	<pre>// Si le CC est en fonction (ON et activé), retour de la valeur de l'input par la classe Cruise_control if (Cruise_control::on && Cruise_control::active) { input_val = Cruise_control::MAJ_input(input_val); // Si la gâchette est appuyée pendant que le CC est en fonction, mettre en PAUSE if (analogRead(input_pin) > 60) Cruise_control::Pause(); } // Si le CC est OFF ou en PAUSE, utilisation de la valeur de la gâchette comme input else input_val = analogRead(input_pin);</pre>
Correction	<pre>// Si le CC est en fonction (ON et activé), retour de la valeur de l'input par la classe Cruise_control if (Cruise_control::on && Cruise_control::active) { input_val = Cruise_control::MAJ_input(input_val); // Si la gâchette est appuyée pendant que le CC est en fonction, mettre en PAUSE if (analogRead(input_pin) > bornes[0] + 50) Cruise_control::Pause(); } // Si le CC est OFF ou en PAUSE, utilisation de la valeur de la gâchette comme input else input_val = analogRead(input_pin);</pre>

Le CC combiné avec le speedomètre est donc 100% fonctionnel

Il faut maintenant ajouter les fonctionnalités adaptatives

05-24

24 mai 2023 02:48

Création de la V6, qui intègre le LiDAR.

La conversion de la vitesse en km/h se fait à l'affichage uniquement afin de s'assurer de ne rien changer à la logique du code.

Le sketch du lidar ne fonctionne plus à nouveau.
Les fils étaient mal branchés :(

Modification du sketch lidar pour l'adapter à la V6 et s'assurer de son fonctionnement lorsqu'il est isolé (impossible d'obtenir une distance avec la V6 complète). J'ai toutefois déterminé l'origine du problème : le test

```
if(Serial1->available()) { [...] }
```

retourne toujours false. Il faut donc trouver pourquoi le port série ne reçoit pas de données du lidar.

Création du .ino lidar_sketch2, qui est une nouvelle version où la classe statique Lidar est implémentée.

Le sketch lidar_sketch2 est maintenant identique au code implémenté dans la V6 et tout fonctionne. La logique n'est pas la cause du problème.

Le sketch a initialement été testé avec un Arduino Uno. En ne changeant rien au code, mais en utilisant un Arduino Mega branché uniquement au lidar, nous avons le même problème (même si les mêmes broches sont utilisées).

Conclusion : le problème relève du choix de microcontrôleur. Il faut donc trouver comment adapter le code pour un Arduino Mega.

En faisant des recherches sur comment régler ce problème, je suis tombé sur le même site d'où nous avons tiré le code du premier sketch. Il s'agit d'un forum et le même problème avait été auparavant observé par quelqu'un d'autre. Voici le lien : <https://forum.arduino.cc/t/arduino-mega-serial-ports/689085>.

Il se pourrait que la cause soit la suivante : un Uno n'a qu'un port série, contrairement au Mega. Il faut donc gérer les ports du Mega.

J'ai finalement trouvé la cause : les broches utilisées ne permettaient pas d'utiliser une interruption (même problème qu'avec le speedomètre, voir journal 05-03). J'y avais pensé, mais les informations que je trouvais en ligne m'avait confirmé que les broches 2 et 3 étaient supposées le permettre, alors que finalement non. J'ai aussi essayé les pins 18 et 19, qui étaient sensées être bonnes. C'est en regardant un exemple de la librairie SoftwareSerial, qui permet de gérer plusieurs ports série, que j'ai trouvé les bonnes broches pour une interruption avec un Arduino Mega (10 et 11).

Le lidar a finalement été implémenté à la trottinette.

Nouveau problème : la détection de distance est très lente (environ 3 secondes pour se mettre à jour).

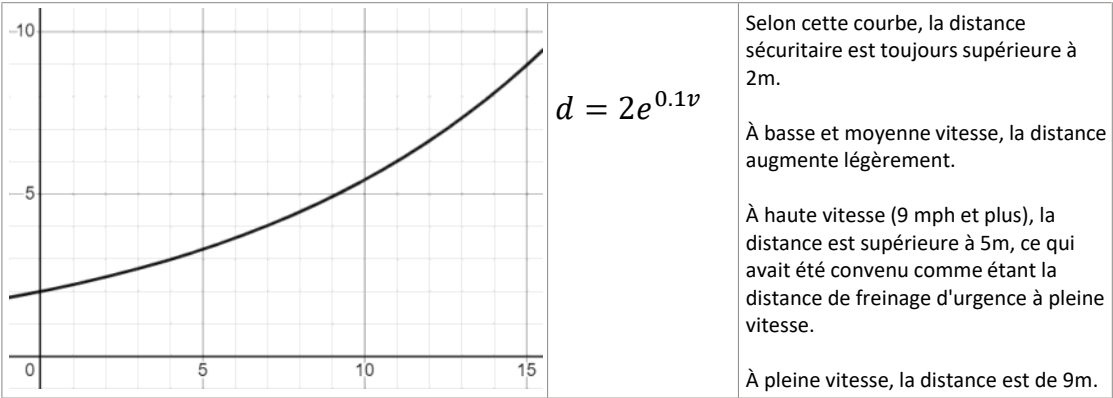
Il s'agit de la logique du code et non de la méthode Calcul_distance puisque ce problème n'était pas présent lors des tests avec le sketch.

Créer une interruption pour le calcul de la distance ne change rien. Ce problème sera donc laissé de côté compte tenu du peu de temps restant. Il va toutefois avoir un grand impact sur les fonctionnalités adaptatives. Je préfère quand même ajouter ces fonctionnalités avant d'optimiser le lidar.

Création de la V7 : implémentation des fonctionnalités adaptatives

Afin d'optimiser la fonctionnalité adaptative tout en assurant une sécurité accrue, j'ai décidé d'utiliser une fonction exponentielle pour déterminer la distance sécuritaire à respecter en fonction de la vitesse de la trottinette. Ainsi, selon la formule $d = a \times e^{b \times v}$, où a et b sont des constantes, d la distance et v la vitesse, j'ai déterminé à l'aide de l'application Desmos la valeur des constantes pour avoir une courbe qui me convient.

Voici le résultat :



Définition d'un tableau décrivant l'état de la trottinette en fonction de l'écart entre la distance à respectée et la distance réelle :

Écart (distance réelle - distance sécuritaire) (m)	Indice de l'état	Nom de l'état	Effet sur le Cruise control
> 0	0	Sécurité	CC non adaptatif
> -0,5	1	Prudence	Changement temporaire de la vitesse set
> -1	2	Vigilance	Changement temporaire de la vitesse set
> -1,5	3	Risque	Ralentissement immédiat
> -2	4	Danger	Ralentissement immédiat
> -2,5	5	Collision	Ralentissement immédiat

Réalisation du schéma final

Je n'ai pas réussi à obtenir les résultats attendus pour les fonctionnalités adaptatives. Comme la colloque des sciences est dans quelques heures seulement, je vais plutôt mettre notre rapport synthèse à jour et préparer notre présentation. S'il me reste encore du temps après cela, je verrai si je peux rendre la V7 fonctionnelle. Sinon, nous utiliserons la V6 pour notre présentation.