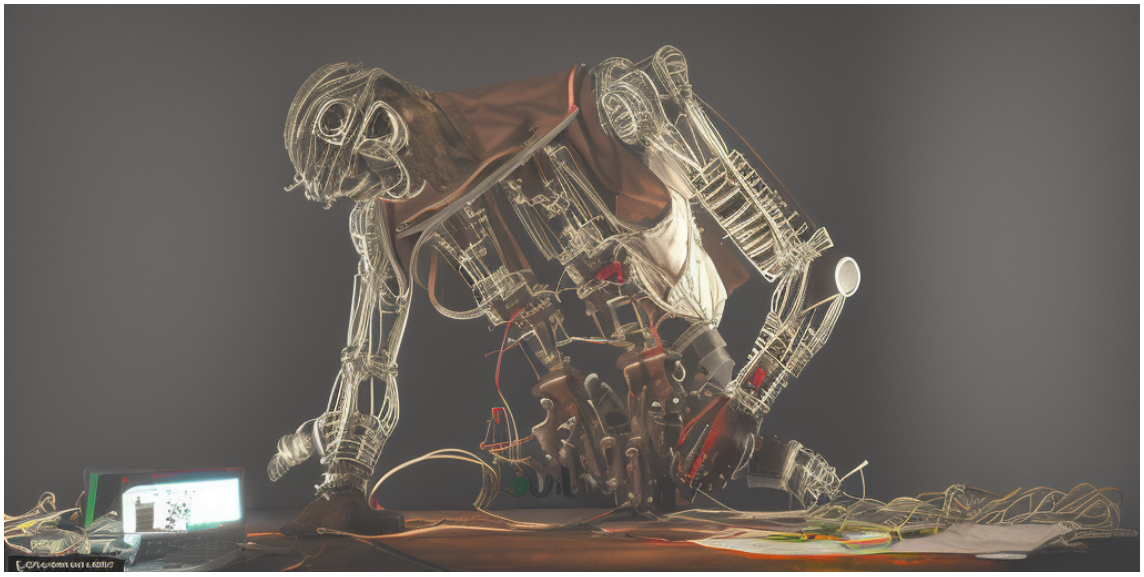
Ingénierie des systèmes cyber-physiques

R5.10 - Mécatronique

Support de cours

Thomas Fromentèze



INSTITUT UNIVERSITAIRE DE TECHNOLOGIE
iut du Limousin

1 Introduction et objectifs

Des compétences solides en mécanique sont incontournables pour la suite de votre carrière professionnelle ou pour vos poursuites d'étude, mais vous serez de plus en plus fréquemment confrontés à des problématiques en lien avec l'automatisation, l'électronique et la programmation. L'ensemble de ces compétences forment la mécatronique (Fig. 1), un domaine d'ingénierie dont les spécialistes sont particulièrement recherchés.

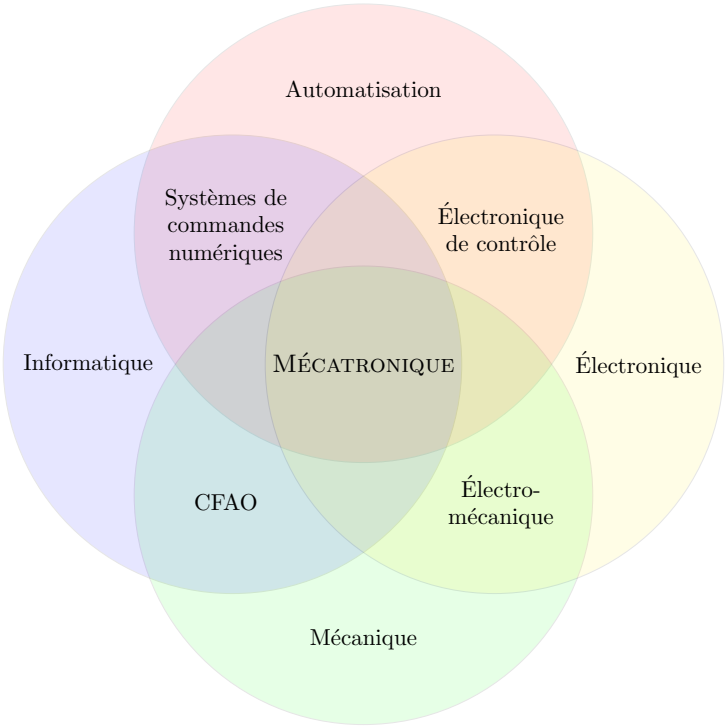


FIGURE 1 – Compétences nécessaires à l'élaboration de systèmes mécatroniques.

Ces compétences permettent le développement de systèmes complexes qui trouvent des applications dans de très nombreux domaines (Fig. 2). Leur maîtrise permet d'assurer la conception de machines programmables à la fois performantes, flexibles, fiables et interactives.

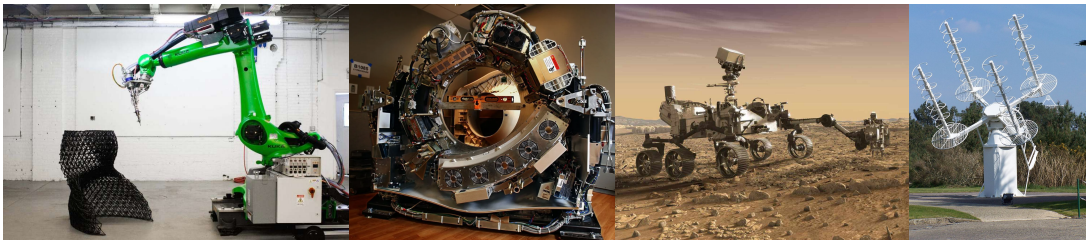


FIGURE 2 – Applications de la mécatronique au domaine industriel, au diagnostic médical, à l'exploration spatiale, aux télécommunications et à la défense - Sources : Raypcb, Imgur, NASA, Wikipedia.

Ce cours est organisé autour d'un projet commun ayant pour objectif **d'assembler et de programmer un véhicule autonome**.

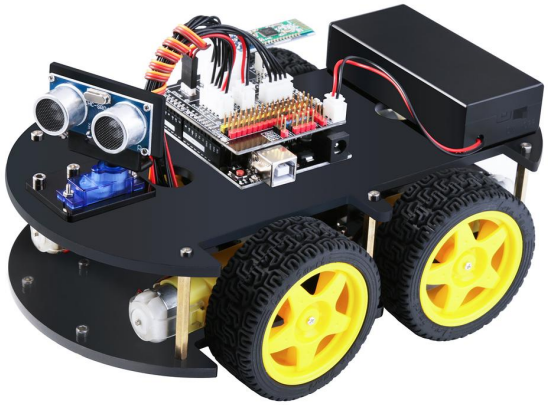


FIGURE 3 – Véhicule programmable développé en TP pour un challenge inter-parcours - source de l'illustration : elegoofrance.fr

Pour l'évaluation des travaux pratiques, vous participerez en binômes ou trinômes à une compétition commune aux trois parcours où votre robot sera évalué pour l'exécution d'un objectif chronométré.

Pour réaliser ce projet, vous devrez maîtriser trois thématiques incontournables :
— L'instrumentation du véhicule et la numérisation des signaux.
— Le contrôle de la chaîne de puissance pour assurer le guidage.
— La programmation Arduino pour automatiser la navigation grâce au retour des capteurs.

L'ensemble de ces éléments forme une chaîne d'information décrite dans la figure ci-dessous :

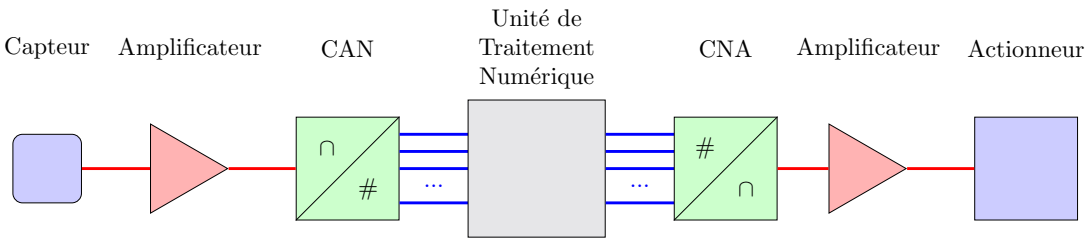


FIGURE 4 – Architecture d'un système mécatronique formant une chaîne d'information

Tous les éléments sont organisés autour d'une unité de traitement numérique qui pour nous sera un microcontrôleur Arduino. Ce dernier est équipé de convertisseurs analogique-numérique et numérique-analogique (CAN et CNA) qui permettent de traduire les informations des capteurs et actionneurs en signaux logiques interprétés par le microcontrôleur. Ces systèmes reposent aussi sur l'exploitation de modules permettant l'amplification et le filtrage des signaux. Ces deux problématiques seront considérées hors-programme dans cette ressource par manque de temps mais peuvent s'avérer incontournables dans certaines situations.

Pour illustrer le fonctionnement de cette chaîne, un exemple de navigation de drone est décrit :

FIGURE 5 – Drone à propulsion omni-directionnelle conçu par Aerix Systems, fondé par Hugo Mayounove et Clément Picaud, DUT GMP de Limoges en 2017.

Collecte de données par des capteurs : Un drone peut être équipé de plusieurs capteurs, dont une caméra, un gyroscope, un accéléromètre, un GPS et un altimètre. Ces capteurs fournissent des informations en temps réel sur l'environnement du drone et ses mouvements.

Conversion analogique-numérique : Les signaux analogiques provenant des capteurs (par exemple, les signaux de la centrale inertielle) sont convertis en signaux numériques pour être traités par l'ordinateur de bord.

Traitement des données et prise de décisions : L'ordinateur de bord traite les données des capteurs pour déterminer la position actuelle du drone, sa vitesse, son orientation, etc. Il peut également analyser les images de la caméra pour identifier des objets ou des personnes. Sur la base des données traitées, l'ordinateur de bord prend des décisions pour piloter le drone. Par exemple, il peut décider de changer la direction du drone pour éviter un obstacle, de modifier l'altitude pour suivre une cible, ou d'atterrir si la batterie est faible.

Conversion numérique-analogique : Les commandes générées par l'ordinateur de bord sont converties en signaux analogiques pour être transmises aux actionneurs.

Gestion des actionneurs : Les signaux analogiques sont envoyés aux actuateurs du drone (moteurs, servomoteurs, etc.) pour exécuter les commandes. Les moteurs sont asservis pour faire décoller le drone, modifier sa direction ou sa vitesse, ou pour le faire atterrir.

La première partie de ce cours permet de découvrir différents types de capteurs et de comprendre la façon dont les signaux peuvent être convertis en informations interprétables par notre microcontrôleur.

2 Instrumentation et numérisation des signaux

2.1 Capteurs pour l'instrumentation

Les capteurs permettent la conversion de grandeurs physiques (température, pression, champ magnétique, ...) en signaux électriques (Fig. 6).

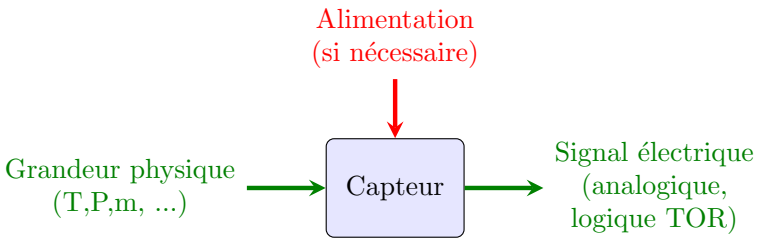
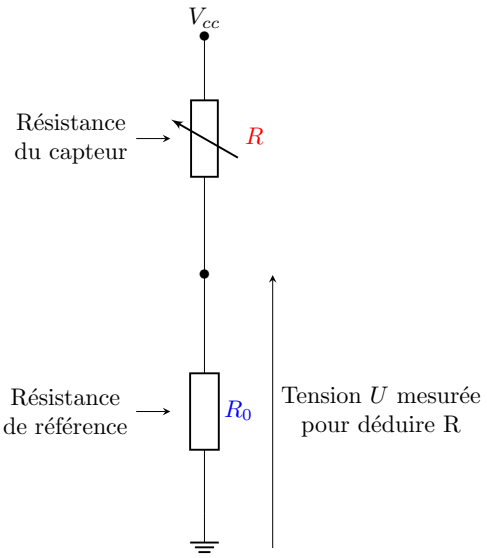


FIGURE 6 – Schéma de principe d'un capteur permettant de convertir une grandeur physique en signal électrique.

Dans les cas les plus simples et les plus courants, les capteurs font varier une valeur de résistance électrique en fonction d'une influence extérieure. La mesure d'un capteur résistif nécessite alors le simple montage d'un pont diviseur de tension à l'aide d'une résistance de référence, dont la valeur est stable et précisément connue. Pour rappel, la formule du pont-diviseur est la suivante :

$$U = \frac{R_0}{R + R_0} V_{cc} \qquad (1)$$



La relation entre la résistance du capteur R et la grandeur physique du capteur G, définie par une fonction $G = f(R)$, est généralement fournie avec la documentation des capteurs. La valeur de la grandeur physique interrogée (température, pression, angle ...) est alors déterminée de la façon suivante :

1. Mesure d'une tension U par un pont diviseur
2. Calcul de la résistance du capteur R en fonction de U, V_{cc} et R_0
3. Calcul de la valeur de la grandeur physique interrogée $G = f(R)$

Ces étapes sont généralement programmées dans le microcontrôleur, qui pourra ainsi réaliser la mesure de tension U et fournir directement la valeur de G, exploitée dans la suite de son code pour des prises de décisions.

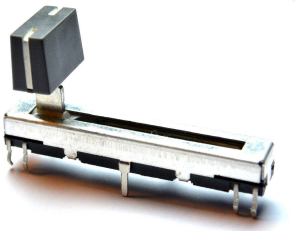
De nombreux autres capteurs reposent sur un fonctionnement plus complexe. Généralement, les circuits sont conçus de façon à réaliser une simple mesure de tension qui peut être convertie en la grandeur physique surveillée. Certains capteurs intègrent parfois des circuits transférant les informations mesurées à l'aide de protocoles de communication plus avancés.

Quelques capteurs sont présentés. Ces exemples sont en partie adaptés de mon précédent cours d'électricité de BUT1 - Motorisation et Capteurs.

2.1.1 Potentiomètres

Un potentiomètre est un composant permettant de faire varier une résistance en fonction de la position d'un curseur. La majorité des potentiomètres sont rotatifs, permettant de lier une position angulaire à une valeur de résistance entre deux de ses bornes. Les joysticks des manettes de jeux vidéo et des interfaces de pilotage sont ainsi généralement constitués de deux potentiomètres (un par axe). Il existe aussi des potentiomètres à glissière (aussi appelés *slider*) où la position du curseur en translation permet de faire varier la résistance entre deux bornes.

On choisit un potentiomètre en fonction de sa résistance totale et de sa loi de variation qui peut être linéaire ou logarithmique suivant la position du curseur. Il est aussi possible de l'associer à des résistances fixes pour modifier la progression de la résistance équivalente.



Un potentiomètre présente trois pattes d'accès et peut être étudié à partir du schéma équivalent suivant :

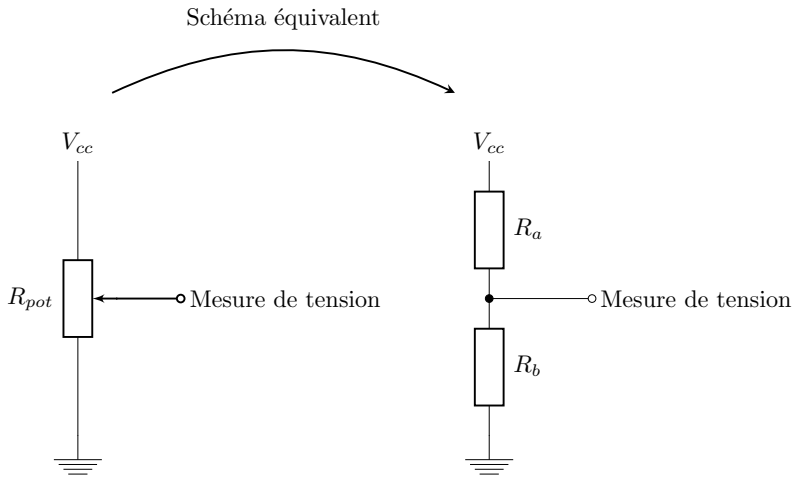


FIGURE 7 – Représentation d'un potentiomètre sous forme de pont diviseur de tension.

En appliquant une tension V_{cc} sur une de ses bornes, on retrouve le schéma d'un pont diviseur de tension sans même avoir besoin d'une résistance supplémentaire. La position du curseur permet alors de faire varier la tension mesurée entre V_{cc} et 0V, qui est le potentiel de la masse.

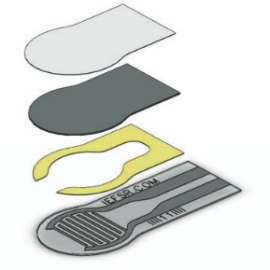
La résistance totale R_{pot} est nécessairement définie telle que $R_{pot} = R_a + R_b$. Pour nos applications, nous utiliserons le plus souvent des potentiomètres dont la résistance totale est de l'ordre du $k\Omega$ mais il existe toute une variété de composants du commerce compris entre quelques Ω et quelques dizaines de $M\Omega$.

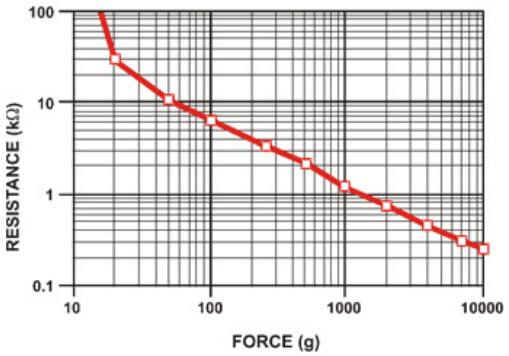
Exemples d'applications :

- **Contrôle de vitesse de moteur** : Ajustement de la vitesse d'un moteur DC en modifiant la tension d'entrée.
- **Réglage de position** : Utilisé comme capteur de position dans des systèmes asservis pour obtenir un retour d'information sur la position d'un élément mobile.
- **Interface utilisateur** : Ajustement du volume, de la luminosité ou d'autres paramètres dans des dispositifs électroniques interactifs.

2.1.2 Capteurs de force

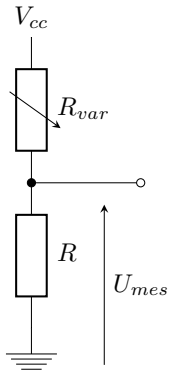
Les capteurs de force (aussi appelé FSR pour Force Sensing Resistor) sont des résistances dont la valeur diminue lorsqu'une force est appliquée sur leur surface. Elles sont le plus souvent réalisées à partir de deux structures inter-digitées associées à un film polymère conducteur dont la résistance varie avec la pression. Ce dernier permet alors de court-circuiter une portion variable de la surface du capteur, faisant diminuer sa résistance en fonction de la force appliquée.





Source : https ://learn.adafruit.com/

En l'associant à une résistance fixe et connue, on peut alors former un pont diviseur. Ce dernier permet d'adapter la plage d'utilisation du capteur à la dynamique de tension de notre appareil de mesure. Une mesure de tension permet alors de déduire la valeur de la résistance du capteur à chaque instant. On associe grâce à la courbe caractéristique du capteur fournie par le constructeur la valeur de la grandeur physique impliquée en fonction de la valeur de la résistance mesurée.



Exemples d'applications :

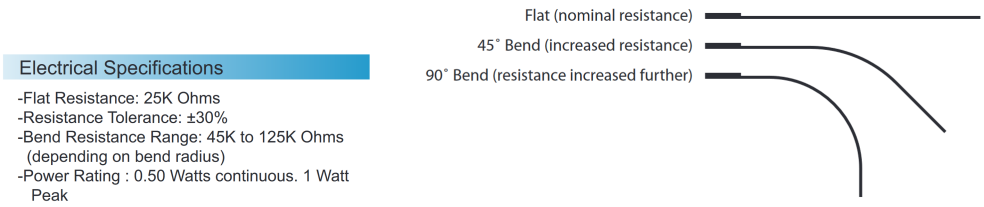
- **Systèmes de détection de niveau de remplissage** : Utilisé dans des applications où le poids d'un récipient varie en fonction du niveau de remplissage, comme une cafetière ou une machine de distribution.
- **Contrôle de la préhension en robotique** : Mesure de la force de préhension d'une pince robotique pour éviter d'endommager des objets fragiles.
- **Interfaces tactiles** : Création d'interfaces utilisateur sensibles à la pression.

2.1.3 Capteurs de flexion

Les capteurs de flexion sont constitués d'un ensemble de cellules montées en série, réalisée à partir du même type de polymère conducteur que celui introduit pour les capteurs de force. Quand une flexion est appliquée, la résistance totale mesurée tend cette fois-ci à augmenter. Ce capteur doit une fois encore être associé à une résistance fixe et connue pour former un pont diviseur de tension. La flexion permet alors de moduler une tension de sortie en fonction de la tension d'alimentation constante appliquée au pont.

On retrouve notamment ces capteurs dans de nombreuses applications de détection de mouvements liées à la robotique.

Des caractéristiques typiques issues d'une documentation sont détaillées :

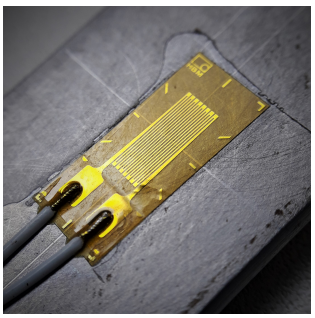


Source : SpectraSymbol - FlexSensor (incluant illustration précédente)

Exemples d'applications :

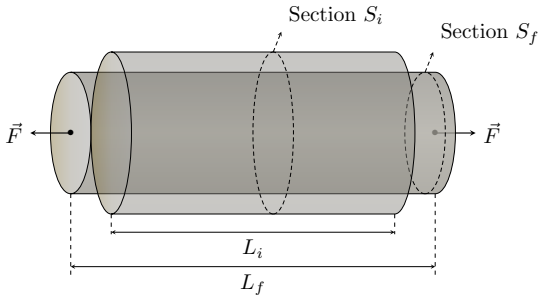
- **Gants interactifs** : Utilisés pour détecter les mouvements des doigts dans des applications de réalité virtuelle ou de contrôle robotique.
- **Surveillance médicale** : Suivi de la mobilité articulaire chez les patients pour la rééducation ou la détection des anomalies.
- **Jouets interactifs** : Intégrés dans des jouets pour détecter les mouvements et activer certaines fonctions ou réponses.

2.1.4 Jauges d'extensométrie



Une jauge d'extensométrie (aussi appelée jauge de déformation) permettent de mesurer une déformation selon une dimension. Il s'agit d'un long fil conducteur replié en serpentin. Une fois collée à la surface d'un matériau subissant une déformation sous l'action d'une contrainte, la longueur de cette jauge peut-être amenée à être légèrement modifiée, impactant ainsi la résistance à ses bornes. Suivant la formule $R_L = \rho L/S$ vue en BUT1, l'allongement (par tension) d'une jauge tend à augmenter sa résistance et une contraction (par striction) tend à la diminuer.

On s'intéresse donc à la variation de résistance ΔR induite par une compression ou une traction induite sur une jauge.



$$\Delta R = R_f - R_i = \frac{\rho L_f}{S_f} - \frac{\rho L_i}{S_i}$$

Pour simplifier ce modèle, on considère que la résistivité linéaire ρ ne varie pas avec la déformation.

Une déformation est une grandeur sans unité, définie telle que $\epsilon = \frac{\Delta L}{L_i}$, avec :

- L : la longueur initiale de la jauge.
- $\Delta L = L_f - L_i$: la variation de longueur, avec $\Delta L > 0$ pour une traction et $\Delta L < 0$ pour une compression.

Pour de faibles déformations, la variation relative de résistance et la déformation suivent une relation linéaire :

$$\frac{\Delta R}{R_i} = k\epsilon = k\frac{\Delta L}{L_i}$$

La constante k est sans unité et correspond à la sensibilité de la jauge. Cette dernière dépend du matériau conducteur utilisé pour sa fabrication.

Une mesure de la variation relative de résistance permet alors de remonter à la déformation ϵ en un point et selon l'axe de la jauge.

En lien avec les éléments introduits en BUT1, les jauges d'extensométrie sont généralement

associées à des pont de Wheatstone, un pont en H formé à partir de deux ponts diviseurs pour gagner en sensibilité sur les variations des jauges et permettre une compensation des effets de température.

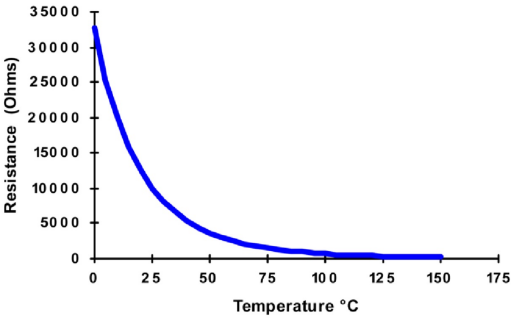
Exemples d'applications :

— **Surveillance de déformations dans les structures** : Utilisées pour détecter et mesurer les contraintes dans les bâtiments, les ponts ou d'autres structures civiles.
— **Capteurs de force et de couple** : Intégration dans des dispositifs pour mesurer directement la force ou le couple appliqué, comme dans les capteurs de charge ou les dynamomètres.
— **Systèmes de feedback en robotique** : Employées pour fournir un retour d'information sur la déformation ou la force appliquée par un élément robotique, améliorant ainsi la précision et la sûreté de l'opération.

2.1.5 Capteurs de température

Les thermistances sont des composants dont la résistance varie en fonction de la température. Dans les thermistances à coefficient de température négatif (NTC - Negative Temperature Coefficient), la résistance diminue avec l'augmentation de la température. Dans les thermistances à coefficient de température positif (PTC - Positive Temperature Coefficient), la résistance augmente avec la température.

Les NTC sont les plus courants, étant généralement plus sensibles, plus fiables, moins chères et capables de couvrir de larges gammes de températures. On retrouve malgré tout des PTC dans certaines applications d'asservissement de température et de protection contre les surintensités.



Source : bapihvac.com - 10K-2 Thermistor

Il existe des alternatives courantes aux thermistances. Les thermocouples fonctionnent par jonction de deux métaux, capables d'induire une tension à ses bornes dépendant de la température (effet Seebeck).

D'autres solutions populaires basées sur des circuits integrés comme le TMP36 ou le LM35 sont aussi particulièrement simples à exploiter, puisque leur caractéristique dépend linéairement de la température. Une tension peut être mesurée sur leur sortie et varie typiquement de l'ordre de 10 mV/°C.

Source : Analog Devices - TMP36

Quelques avantages et inconvénients pour chaque solution sont rassemblés dans le tableau suivant :

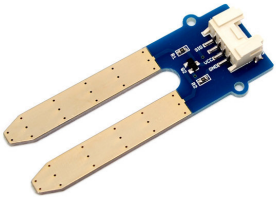
Type	Avantages	Inconvénients
Thermistance	- Haute précision - Coût bas - Réponse rapide - Facile à interfacer	- Non linéaire - Plage limitée - Circuits suppl. nécessaires
Thermocouple	- Plage importante - Robuste - Réponse rapide	- Circuit de compensation - Moins précis à T ambiante - Polarité
Circuit intégré (type TMP36)	- Sortie linéaire - Facile avec microcontrôleur - Calibré - Sans composants suppl.	- Plage limitée - Alimentation nécessaire - Sensible aux interférences

Exemples d'applications :

— **Surveillance thermique des moteurs électriques** : Pour prévenir la surchauffe, optimiser les performances et prolonger la durée de vie des moteurs utilisés dans la robotique ou d'autres systèmes mécatroniques.
— **Régulation de température dans les systèmes d'impression 3D** : Utilisés pour contrôler avec précision la température des têtes d'impression et des plateaux chauffants, assurant ainsi une impression de qualité.
— **Feedback thermique en robotique** : Permettant aux robots de s'ajuster lors d'interactions avec des objets chauds ou froids ou lors de la réalisation de tâches nécessitant une régulation thermique, comme la soudure.

2.1.6 Capteurs d'humidité

Ces capteurs permettent de mesurer des variations relatives de l'humidité d'un environnement, traduits en une variation continue de tension. Les capteurs les plus courants reposent sur un principe identique, mesurant les caractéristiques électriques d'un matériau isolant hygroscopique (qui à tendance à retenir l'humidité). Il est possible de mesurer la résistance ou la capacité au bornes de l'isolant pour en déduire une mesure relative de l'humidité par rapport à une référence connue.



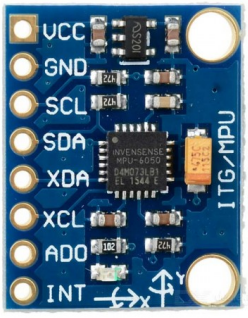
Source : Gotronic

Exemples d'applications :

— **Systèmes d'irrigation intelligents pour l'agriculture** : En intégrant des capteurs d'humidité dans les robots agricoles ou les systèmes d'irrigation, il est possible d'ajuster précisément l'apport en eau en fonction des besoins des plantes.
— **Systèmes de déshumidification domestiques** : Utilisant des capteurs d'humidité pour détecter le niveau d'humidité ambiant et activer automatiquement des déshumidificateurs pour maintenir un environnement confortable dans les habitations.

2.1.7 Centrales inertielles

Une centrale inertielle (ou IMU, pour Inertial Measurement Unit en anglais) est un circuit électronique notamment exploité en navigation et composé d'accéléromètres et de gyroscopes. Le circuit MPU6050 fait parti des références les plus populaires en mécatronique et fonctionne grâce à la technologie MEMS (Micro Electro Mechanical Systems), des éléments mécaniques réalisés sur silicium à l'échelle de quelques micromètres seulement.



Source : Gotronic MPU-6050

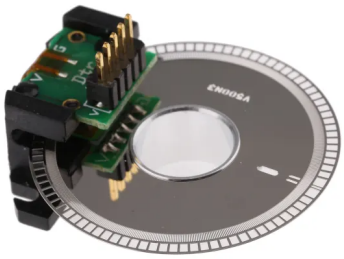
Ce circuit permet de réaliser un suivi de position relatif, en calculant notamment les vitesses de déplacement puis les coordonnées spatiales par une série d'intégrations. Il faut donc disposer d'un point initial pour le calcul de la position, et d'éventuels points de recalages fournis par d'autres outils. En pratique, le cumul rapide d'erreurs d'intégration liées aux incertitudes de mesure impose souvent de coupler ces données avec des informations d'autres capteurs, notamment optiques ou par GPS.

Exemples d'applications :

— **Stabilisation des caméras** : Les IMU fournissent des informations en temps réel sur les mouvements et les inclinaisons, permettant de stabiliser activement les caméras dans les drones, les smartphones ou les équipements professionnels de prise de vue.
— **Contrôle de la navigation des drones** : Les drones utilisent des IMU pour aider à déterminer leur orientation et leur mouvement, en complément d'autres capteurs comme le GPS, pour effectuer des vols stables et suivre des trajectoires prédéterminées.
— **Suivi des mouvements en réalité virtuelle et augmentée** : Les dispositifs de réalité virtuelle et augmentée utilisent des IMU pour suivre les mouvements de la tête ou des mains de l'utilisateur, permettant une interaction fluide et immersive avec les environnements virtuels.

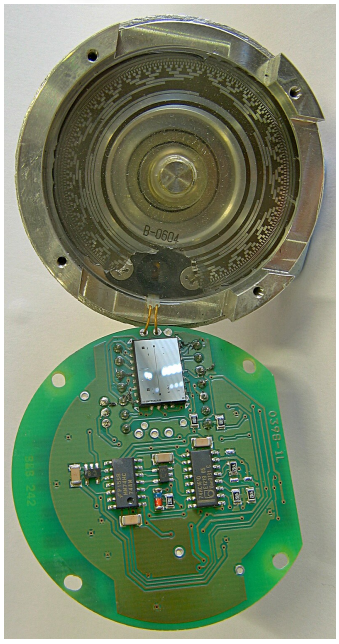
2.1.8 Encodeurs

Les encodeurs (ou codeurs dans certaines ressources) permettent d'assurer le suivi d'un déplacement, le plus souvent sur des axes en rotation au moyen d'encodeurs rotatifs. Ils fournissent ainsi un signal électrique composé d'impulsions interprétables par un microcontrôleur.



Ce dernier peut déduire un changement de position relative au moyen d'encodeurs incrémentaux. Parmi les solutions les plus courantes, un faisceau infrarouge peut être propagé au travers d'un disque à fentes ou d'une roue dentée. Des solutions fonctionnant selon un principe analogue existent aussi en faisant varier un champ magnétique. Des modules plus simples fonctionnent aussi par simple contact.

Il existe aussi des encodeurs numériques absolus plus complexes capables de générer un code binaire d'impulsions électrique sur un ensemble de sorties parallèles qui peuvent à nouveau être interprétées par un microcontrôleur pour en déduire une position angulaire.



Enfin, un potentiomètre peut permettre de constituer un encodeur très simple, assurant une conversion directe entre la rotation du curseur et la résistance mesurée à ses bornes.

Source : Broadcom - Megatron

Exemples d'applications :

- **Contrôle de la position dans la robotique** : Utilisés pour déterminer l'angle précis de rotation et assurer une position exacte des bras ou des articulations robotiques.
- **Systèmes de navigation des véhicules** : Permettant de suivre la direction et la distance parcourue, ce qui est essentiel pour les systèmes de guidage automatisé.
- **Interfaces utilisateur dans les équipements électroniques** : Utilisés comme commandes de volume ou sélecteurs rotatifs dans des dispositifs tels que des amplificateurs audio, des radios et d'autres appareils électroniques.

2.1.9 Capteurs à ultrasons

Les capteurs comme le HC-SR04 utilisent des ondes ultrasonores pour déterminer la distance d'un objet en envoyant une onde et en écoutant son écho. Le temps que met cette onde à revenir au capteur est proportionnel à la distance de l'objet qui peut être déterminée en fonction de la vitesse du son. Il a une portée typique de 2 cm à 400 cm et offre une précision assez élevée pour de nombreuses applications à bas coût.

Source : Radiospare HC-SR04

Exemples d'applications :

- **Détection d'obstacles** : Les robots mobiles utilisent le HC-SR04 pour détecter et éviter les obstacles sur leur chemin, ce qui leur permet de naviguer de manière autonome sans collision.
- **Mesure de niveau** : Le HC-SR04 peut être utilisé pour déterminer le niveau de liquide dans un réservoir en mesurant la distance entre le capteur et la surface du liquide, fournissant ainsi des données en temps réel pour des applications comme l'irrigation ou la gestion de l'eau.

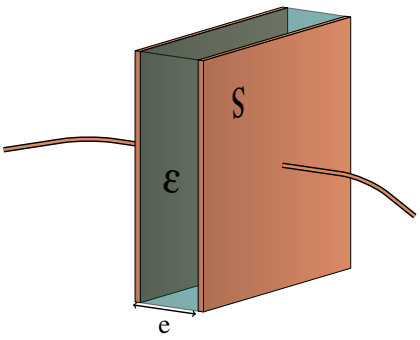
2.1.10 Capteurs capacitifs

Un capteur capacitif détecte des variations de capacité électrique, le plus souvent causées par la proximité d'un objet sans besoin de contact direct. Les capteurs capacitifs sont particulièrement sensibles aux matériaux diélectriques comme le plastique, le verre ou les liquides, mais peuvent aussi détecter des métaux. Grâce à leur capacité de détection sans contact, ils sont idéaux pour des applications où l'objet à détecter pourrait être endommagé par le contact, ou dans des environnements hostiles où les contaminants pourraient affecter d'autres types de capteurs.

En lien avec le programme du BUT2, une capacité est formée par deux plaques métalliques séparées d'un matériau diélectrique (un isolant). La formule d'une capacité, exprimée en Farad, est la suivante :

$$C = \epsilon \frac{S}{e}$$

Avec :
- ϵ la permittivité diélectrique de l'isolant [F.m^{-1}]
- S la surface des conducteurs en regard [m^2]
- e l'épaisseur du diélectrique [m]



Pour rappel, une capacité n'a de sens que pour des signaux transitoires et traduit un effet de stockage temporaire de charges électriques. Pour utiliser ce phénomène pour créer un capteur de proximité, on constitue une capacité à l'aide de deux plaques disposées cette fois côte à côte.

L'application d'une différence de potentiels induit un stockage de charges électriques et l'installation dans l'espace d'un champ électrique $\vec{E}$. La quantité de charges stockées dans chaque plaque dépend directement de la valeur de la capacité.

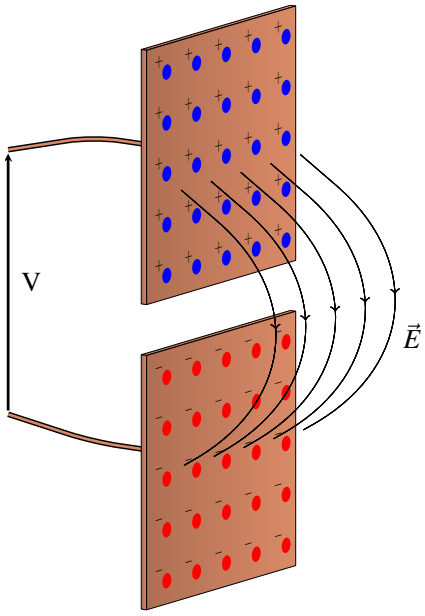


FIGURE 8 – Capteur capacitif formée de deux plaques.

Lorsqu'un objet est approché du capteur, le champ électrique tend à créer une migration de charges électriques à sa surface, induisant un effet capacitif parasite venant perturber la valeur initiale.

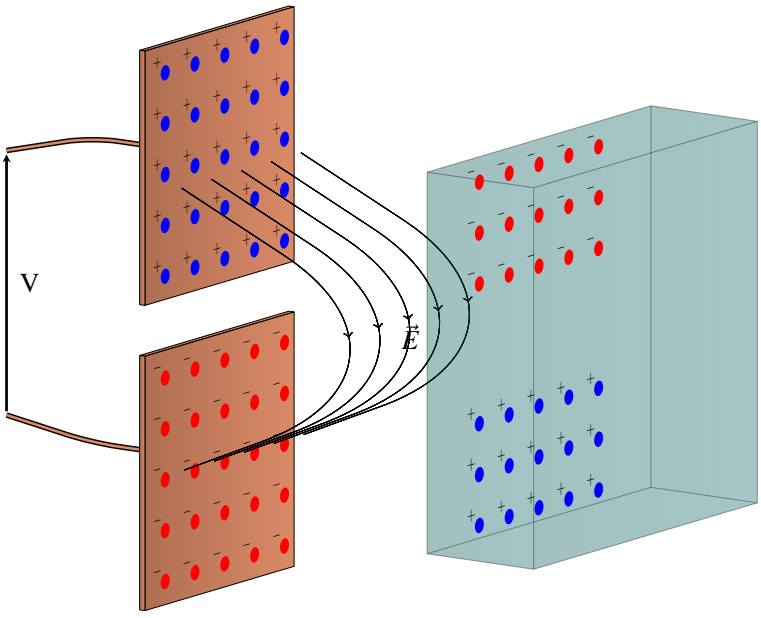


FIGURE 9 – Migration des charges électriques à la surface d'un objet à détecter.

La détection d'une variation de capacité sur un tel montage permet finalement de déterminer si un objet est présent sans contact.

Ces capteurs ne fonctionnant pas en régime continu (la capacité devient un simple circuit ouvert). Ils sont généralement associés à un montage permettant de générer une tension alternative et assurant la détection d'un seuil correspondant à la présence de l'objet. Ce principe est détaillé rapidement dans la description du capteur inductif.

Source : Heschen.com

Exemples d'applications :

— **Détection de niveau de liquides** : Les capteurs capacitifs sont utilisés pour détecter le niveau de liquides dans des réservoirs, même lorsque ces liquides sont contenus dans des récipients non métalliques.
— **Commandes tactiles** : Dans les interfaces utilisateur modernes, comme les écrans tactiles des smartphones et des tablettes, les capteurs capacitifs détectent la présence et la position des doigts de l'utilisateur.
— **Détecteurs de proximité** : Contrairement aux capteurs inductifs, les capteurs capacitifs peuvent détecter des objets non métalliques, comme le plastique ou le verre, les rendant utiles dans des applications telles que la détection de la présence d'une bouteille sur une chaîne de production.

2.1.11 Capteurs inductifs

Un capteur inductif est conçu principalement pour détecter des objets métalliques, en particulier ceux qui sont ferreux. Une bobine est alimentée par une tension alternative pour générer un champ magnétique, formant un électroaimant. Lorsqu'un objet métallique est déplacé dans ce champ magnétique, des boucles de courant sont induites à sa surface, formant un champ magnétique secondaire venant s'opposer à celui généré par la bobine.

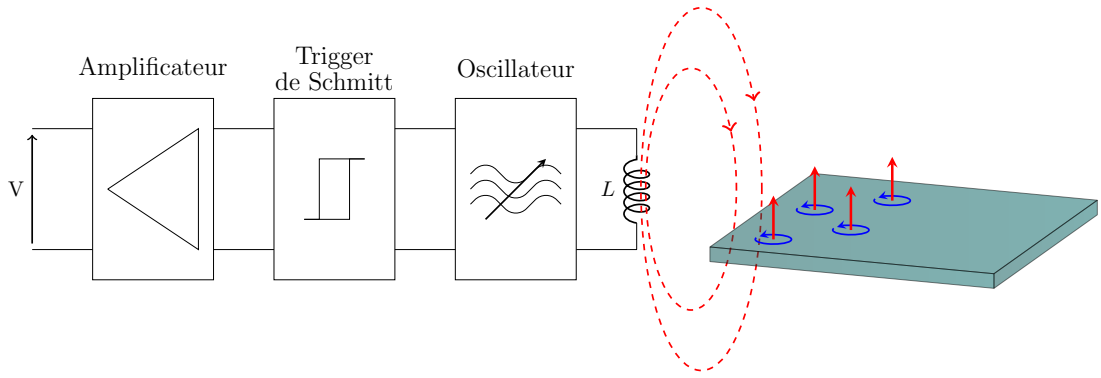


FIGURE 10 – Induction d'un champ magnétique parasite causé par l'approche d'une pièce métallique, alors détectée sans contact.

Comme pour les capteurs capacitifs, ces effets ne sont pas observables avec des signaux de tension continue. On utilise alors la combinaison d'un oscillateur, permettant lé génération d'un signal alternatif, avec un trigger de Schmitt assurant le déclenchement d'un seuil de détection traduit en signal de tension Tout-Ou-Rien.

L'ensemble de ces éléments est livré dans un packaging à l'apparence très proche de celle du capteur capacitif présenté précédemment.

Exemples d'applications :

— **Positionnement en automatisation** : Les capteurs inductifs sont utilisés pour détecter la présence ou l'absence de pièces métalliques sur des chaînes de montage automatisées, garantissant que les pièces sont correctement placées avant les étapes ultérieures du processus.
— **Compteurs de pièces métalliques** : Dans les systèmes de tri et de comptage, les capteurs inductifs détectent et comptent les pièces métalliques à mesure qu'elles passent devant le capteur.
— **Sécurité des machines** : Les capteurs inductifs peuvent être utilisés pour détecter la position fermée ou ouverte des portes de protection sur les machines-outils, assurant la sécurité de l'opérateur.

2.2 Conversion analogique-numérique

L'ensemble des signaux issus des précédents capteurs peuvent être acheminés à un microcontrôleur. Ces informations permettront d'assurer des prises de décisions impactant notamment les différents actionneurs du système considéré.

Il semble donc incontournable de disposer d'un circuit permettant de récupérer des signaux de tension issus des capteurs et les convertir en informations binaires interprétables par un microcontrôleur.

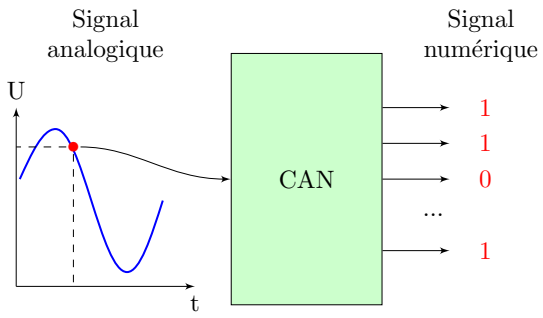


FIGURE 11 – Conversion d'un échantillon de tension issu d'un capteur en signal numérique.

Pour aller plus loin, il est nécessaire de différencier deux types de signaux variant de façon continue ou obéissant à une logique Tout-Ou-Rien. Il sera ensuite utile d'étudier rapidement les opérations de quantification et de numérisation des informations mesurées.

2.2.1 Signaux analogiques continus ou Tout-Ou-Rien

En lien avec les cours d'automatique de BUT1, les signaux électriques peuvent être de nature Tout-Ou-Rien, lorsqu'ils sont par exemples issus de boutons poussoirs. De nombreux capteurs transmettent quant à eux une information continue sur une plage de tension.

En chaque instant, la tension doit être convertie en signaux numériques binaires. Un signal variant de façon continue peut prendre toute une gamme de valeurs de tension. Il apparaît donc nécessaire d'utiliser beaucoup de digits binaires pour représenter un niveau de tension :

$$U(t) = 4.21\text{V} \rightarrow 11\ 0101\ 1110$$

Cette conversion sera détaillée dans la section suivante.

La conversion d'un signal Tout-Ou-Rien répond à une logique beaucoup plus simple. La tension d'entrée ne pouvant prendre que deux valeurs, la conversion binaire est donc directe.

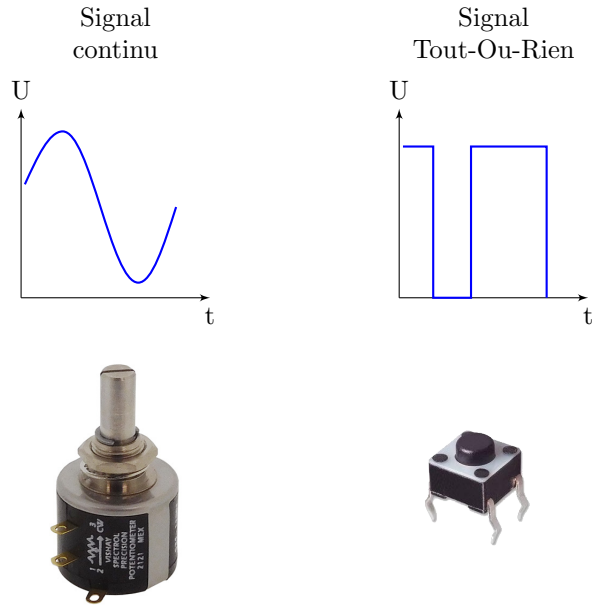


FIGURE 12 – Signaux de tension variant de façon continue ou en logique Tout-Ou-Rien.

Par exemple :

$$\text{si } U(t) = 5\text{V} \rightarrow 1$$
$$\text{si } U(t) = 0\text{V} \rightarrow 0$$

Un signal Tout-Ou-Rien issu par exemple d'un bouton est ainsi très proche d'un signal numérique binaire et nécessite une opération de conversion plus simple.

Ces différences ont un impact direct sur les types de convertisseurs habituellement embarqués au sein de microcontrôleurs, adaptés au choix aux signaux analogiques ou "numériques" (jusqu'à présent appelés Tout-Ou-Rien).

2.2.2 Conversion de tensions en signaux binaires

Un convertisseur analogique-numérique (CAN) permet de traduire un niveau de tension mesuré à un instant donné en un code binaire. Leur opération détaillée est trop complexe pour être étudiée en détail dans votre formation, mais le principe général de fonctionnement est à connaître pour les besoins de nos projets.

Les caractéristiques d'un convertisseur dépendent du nombre de niveaux de tension qui sont convertis et de sa fréquence d'échantillonnage F_s. Cette dernière est liée à la durée T_s entre deux conversions successives, suivant la relation :

$$F_s = \frac{1}{T_s}$$

Le théorème de Shannon impose d'utiliser une fréquence au moins deux fois supérieure à la fréquence maximale du signal à mesurer.

La résolution en amplitude d'un CAN est quant à elle définie par le nombre de niveaux disponibles pour la conversion.

Dans l'exemple simplifié illustré pour faciliter les explications, chaque niveau de tension est codé à l'aide de seulement 3 bits. Le nombre de niveaux répartis sur une dynamique de 5V

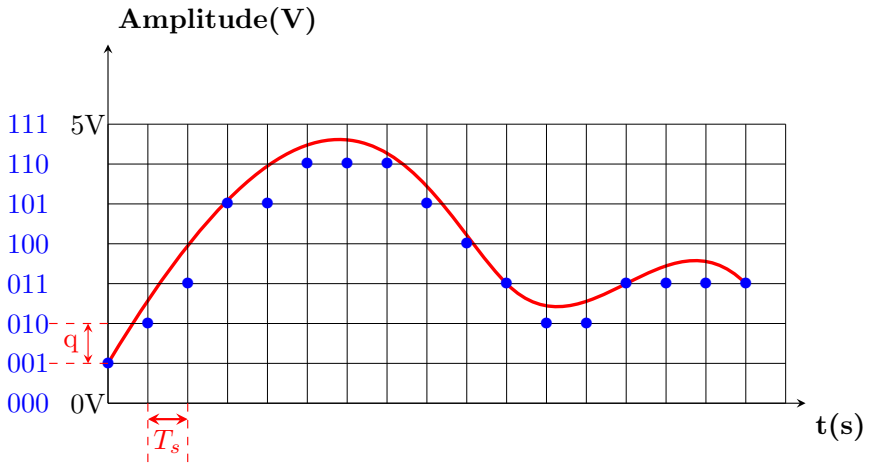


FIGURE 13 – Conversion analogique-numérique, où chaque échantillon bleu sera associé à un code binaire transmis au microcontrôleur.

est alors de $2^3 = 8$. La résolution en amplitude, aussi appelée quantum, est définie pour cet exemple telle que :

$$q = \frac{5}{2^3} = 0.625\text{V}$$

Cette valeur correspond à la différence de tension nécessaire au passage d'un niveau au suivant. Dans cet exemple, les codes binaires transmis au microcontrôleur sont donc :

$$001, 010, 011, 101, 101, 110...$$

Pour diminuer l'erreur liée à la faible résolution en tension de l'exemple considéré, on utilise souvent des CAN codés sur au moins 10 bits, impliquant $2^{10} = 1024$ niveaux disponibles. De façon générale, le quantum prend alors la forme :

$$q = \frac{V_{\text{max}} - V_{\text{min}}}{2^N}$$

avec :

- $V_{\text{max}} - V_{\text{min}}$ correspondant à la plage de tension de fonctionnement. On parle aussi de **pleine échelle**.
- N est le nombre de bits utilisés pour la conversion.

En fonction des architectures considérées, il existe aussi une variante à cette formule :

$$q = \frac{V_{\text{max}} - V_{\text{min}}}{2^N - 1}$$

En comparaison du cas précédent, l'information est codée sur un niveau en moins. La formule adaptée à votre convertisseur sera généralement fournie dans sa documentation.

Note spécifique au projet :

Nos applications seront réalisées à l'aide d'un microcontrôleur Arduino UNO, doté d'un ensemble de 6 CAN 10 bits fonctionnant entre 0V et 5V. Leurs entrées sont nommées de `A0` à `A5`. L'information renvoyée par l'Arduino est un code décimal plutôt qu'un code binaire. Les niveaux de tension seront donc codés entre 0 et 1023.

L'Arduino UNO est aussi équipé d'un maximum de 14 entrées numériques, nommées de `0` à `13`, qui permettent la numérisation des signaux Tout-Ou-Rien, par exemple issus de boutons.

Ces entrées sont codées sur 1 bit, soit deux niveaux de tension.

Cette section a permis de présenter un certain nombre de capteurs rencontrés en automatisation et en mécatronique, ainsi que le principe de conversion analogique-numérique. La liste des capteurs présentés est loin d'être exhaustive et vous serez probablement confrontés à beaucoup d'autres technologies dans votre carrière professionnelle. La prochaine partie se concentre sur le contrôle des actionneurs.

3 Gestion des actionneurs

Le programme de BUT2 vous a permis de découvrir la notion de séparation commande-puissance, généralement rencontrée en automatique pour limiter l'interaction des utilisateurs avec de grandes puissances électriques.

La mécatronique fonctionne selon un principe similaire, plus souvent motivé par les limitations des microcontrôleurs à fournir des puissances électriques importantes. Il sera donc nécessaire pour le fonctionnement de beaucoup d'actionneurs (notamment des moteurs électriques), d'utiliser une chaîne de puissance séparée, contrôlée par nos microcontrôleurs.

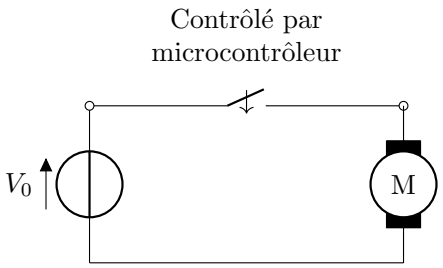


Figure 14 – Alimentation d'un moteur électrique contrôlé par un microcontrôleur.

L'objectif est donc de disposer d'une source capable de délivrer un courant important à tension constante, par l'intermédiaire d'un élément répondant à un programme informatique. La commande d'un tel interrupteur piloté peut être réalisée par des relais. Ces derniers sont notamment fabriqués au moyen d'une bobine dont l'alimentation génère un champ magnétique permettant d'ouvrir ou de fermer des contacts électriques.

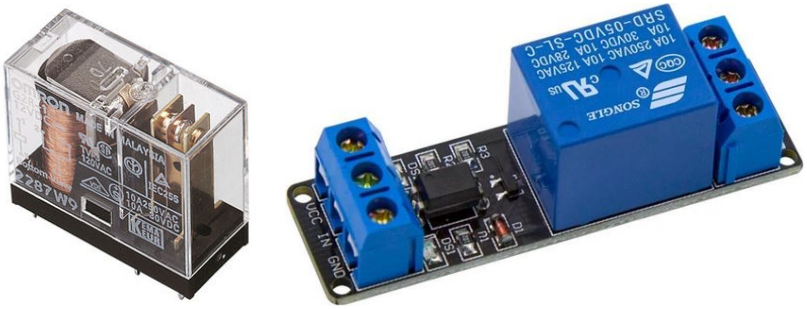


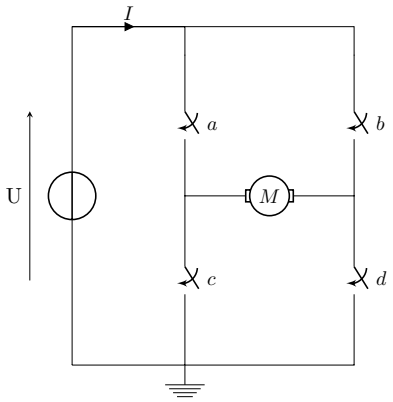
Figure 15 – Relais électriques dans différents boîtiers. Le relai de droite est associé à un octocoupleur pour améliorer l'isolation de la commande et la puissance. Source : Directindustry

Les relais électromécaniques, bien que robustes, souffrent de limitations telles que des temps de commutation assez lents et une usure mécanique. Les ponts en H, en revanche, offrent un contrôle bidirectionnel des moteurs avec une commutation rapide et sans pièces mobiles.

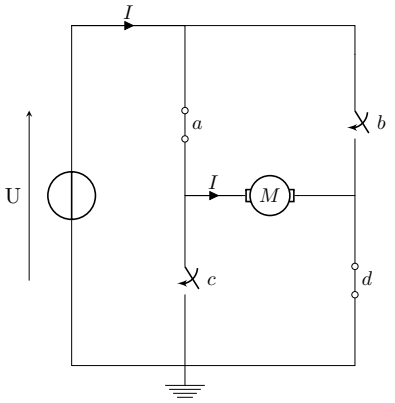
3.1 Rappel sur les ponts en H

Un pont en H est constitué de 4 transistors de puissance, généralement en technologie MOSFET ou bipolaire. Ces derniers sont utilisés en saturation et jouent le rôle d'interrupteurs pilotés par une tension de commande. Ces transistors sont associés à des diodes de protection contre les tensions inverses et les pics de courant générés par les moteurs électriques lors des arrêts et des changements de sens de rotation.

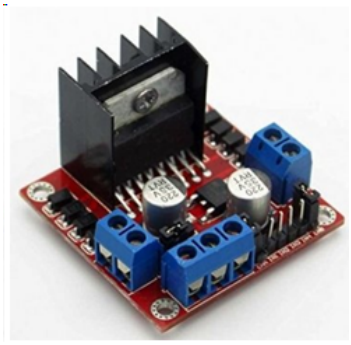
A titre d'exemple, le changement de sens de rotation d'un moteur à balais nécessite d'inverser les potentiels électriques appliqués au collecteur. Plutôt que d'échanger manuellement la position des fils d'alimentation, il est possible d'utiliser ce circuit dédié.



En fermant les contacts a et d, le courant parcourt l'induit du moteur dans un sens.

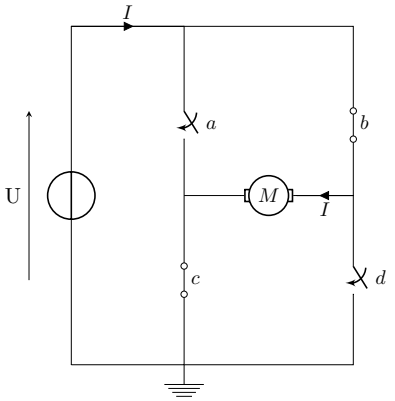


Le L298 de STMicroelectronics, ici monté sur une petite carte de développement est un double ponts en H que l'on pourra être amené à utiliser en mécatronique :



La commutation des transistors est assurée par des entrées de commande qui peuvent être directement connectées aux sorties numériques d'un microcontrôleur.

En fermant les contacts b et c, le courant parcourant l'induit est inversé, permettant de faire tourner le moteur dans l'autre sens.



Il faut bien sûr éviter de fermer simultanément les contacts a et c ou les contacts b et d afin de ne pas court-circuiter la source de tension. Certains ponts en H sont directement câblés pour interdire de telles combinaisons par sécurité.

L'exemple est ici donné pour le cas le plus simple d'un moteur à courant continu mais le principe de fonctionnement est équivalent pour un moteur pas à pas, notamment utilisé pour l'ensemble des machines à commandes numériques comme les imprimantes 3D.

Nous avons donc la possibilité de contrôler l'activation et la désactivation d'une chaîne de puissance à partir d'un signal de commande, ainsi que de changer la polarité appliquée à l'ac-

tionneur. La tension à ses bornes est un peu plus faible que celle de l'alimentation, une partie étant absorbée par les transistors.

Dans la partie suivante, une technique permettant de contrôler le niveau de tension appliquée à un actionneur à partir d'une tension d'alimentation fixe est détaillée.

3.2 Contrôle de tension continue

Nous savons maintenant contrôler l'activation d'un moteur et son changement de sens, mais il n'est pas encore possible de changer sa vitesse de rotation à l'aide d'un pont en H. Les transistors fonctionnant selon une logique Tout-ou-Rien, il n'y a pas d'état intermédiaire dans le niveau de tension communiqué à l'actionneur.

Une astuce basée sur la modulation de largeurs d'impulsions est détaillée dans cette partie. On retrouve généralement le terme PWM sur les applications d'électronique, pour Pulse Width Modulation. Le principe consiste à commuter rapidement l'état des transistors, formant des trains d'impulsions dont on contrôle la largeur.

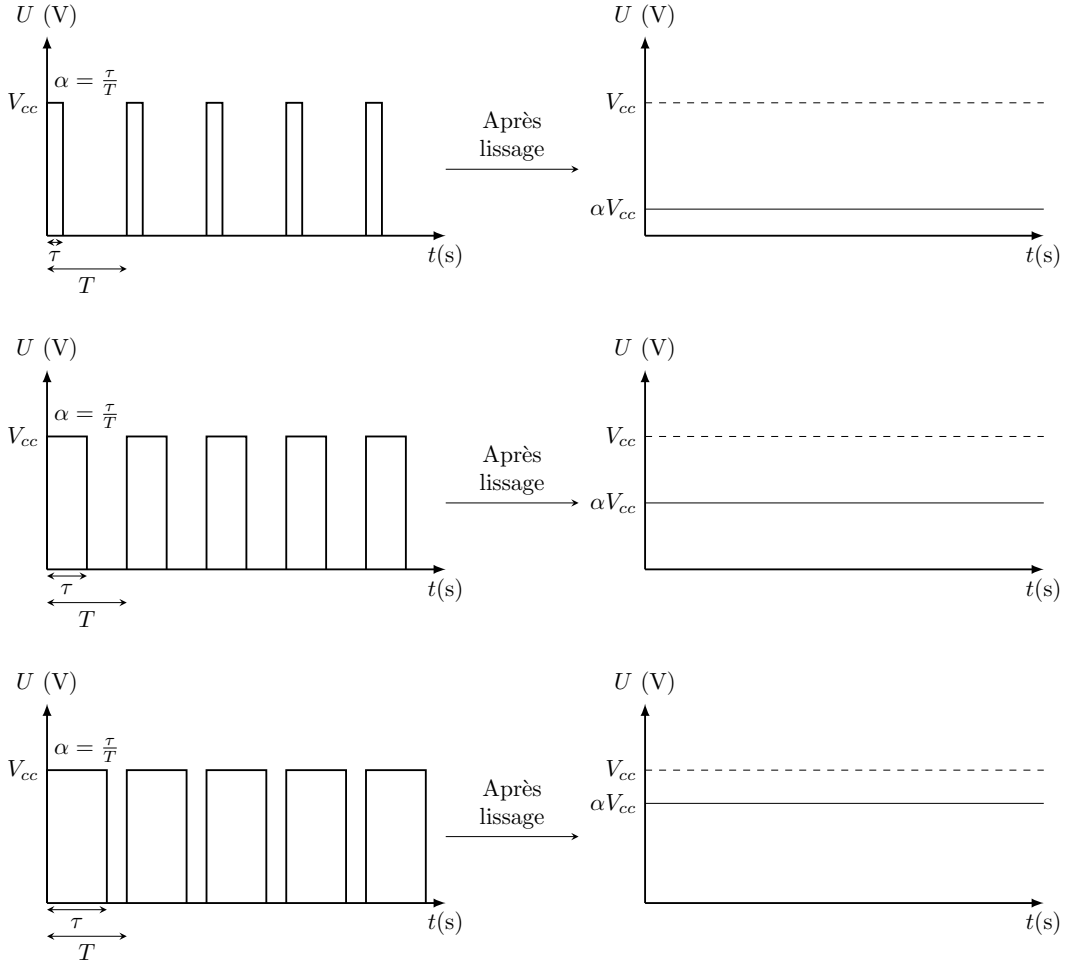


Figure 16 – Modulation de largeurs d'impulsions à l'aide d'un pont en H, permettant après filtrage d'obtenir une tension continue dont la valeur moyenne est contrôlée.

La tension moyenne obtenue après lissage est proportionnelle au rapport cyclique (duty cycle en anglais), défini telle que :

$$\alpha = \frac{\tau}{T}$$

où τ est la durée des impulsions de tension et T la période entre deux impulsions successives. Dans un circuit sans pertes, la tension continue est alors de αV_{cc}, où V_{cc} est la tension d'alimentation du pont en H. En pratique, des pertes de tension induisent une variation sur une plage réduite.

Le lissage des impulsions de tension peut être réalisé au moyen d'un filtre passe-bas, constitué d'une résistance en série et d'une capacité entre la sortie de la résistance et la masse. Les alimentations à découplage fonctionnent selon un principe analogue.

Dans nos cas, il est possible de connecter directement le pont en H sur un moteur électrique. Ce sont cette fois les impulsions de courant dans le bobinage qui seront lissés par effet inductif, permettant de contrôler la vitesse de rotation par alimentation PWM.

Un contrôle de tension continue est finalement possible à condition de disposer de microcontrôleurs équipés de sorties numériques compatibles avec une commande PWM. Pour notre projet, l'Arduino UNO dispose de 6 sorties PWM sur les broches 3, 5, 6, 9, 10 et 11. Ces broches sont marquées par un symbôle "∼".

3.3 Cas particulier des servo-moteurs

Un servo-moteur permet un contrôle précis de la position angulaire, la vitesse, et parfois la force appliquée à son arbre, selon les modèles.

Le servo-moteur se compose généralement d'un petit moteur, d'un système de réduction d'engrenage, d'un potentiomètre pour le retour de position, et d'un circuit de commande. Le moteur est piloté par un signal électrique PWM. Le changement de la largeur de l'impulsion modifie l'angle du rotor du moteur.

L'encodeur connecté à l'arbre du moteur (souvent un simple potentiomètre) fournit un retour de position, qui est comparé à la commande souhaitée. Le circuit de commande interne ajuste le mouvement du moteur en fonction de cet écart.

Source : Gotronic

La plupart des servo-moteurs ont une plage de mouvement limitée, souvent entre 0 et 180 degrés, bien que certains modèles spécifiques puissent tourner en continu.

Le circuit de commande est cette fois intégré au servo-moteur.

Il possible d'alimenter certains petits servo-moteurs directement par Arduino UNO, ce dernier étant capable de fournir un courant maximal de l'ordre de 40 mA par broche sur 5V. Les servo-moteurs plus gros nécessiteront une alimentation séparée pour fournir un couple plus important. Le microcontrôleur ne servira alors qu'à envoyer le signal de commande PWM pour asservir la position angulaire du rotor.

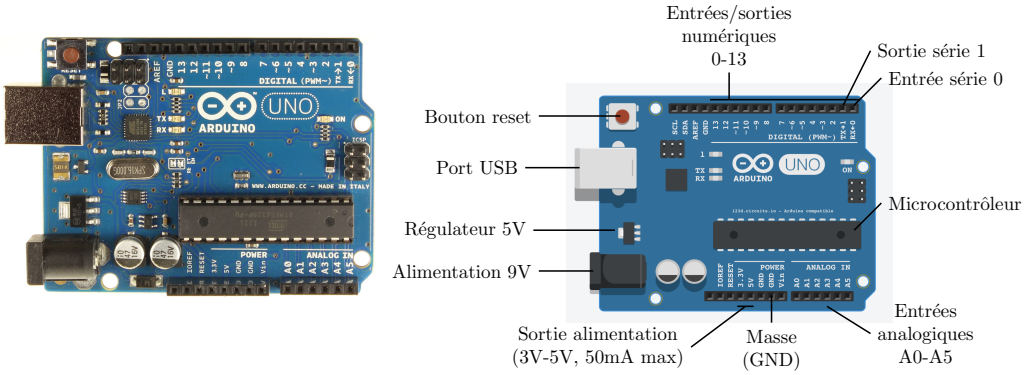
4 Programmation sur microcontrôleur Arduino

4.1 Introduction à l'Arduino

L'Arduino UNO est une carte dont l'architecture est publiée en **licence libre**.

Caractéristiques principales :

— Microcontrôleur : ATmega328P (dépend des versions)
— Tension d'opération : 5V
— Entrées/Sorties numériques : 14 (dont 6 peuvent être utilisées comme sorties PWM)
— Entrées analogiques : 6
— Courant continu par broche I/O : 40 mA
— Mémoire Flash : 32 KB (0,5 KB utilisé par le bootloader)
— Fréquence d'horloge : 16 MHz



Avec l'ensemble des éléments présents sur la carte permettant l'acquisition, le traitement et la génération de signaux, il est possible de réaliser des tâches diverses en lien avec la domotique, la robotique, la réalisation de prototypes et l'électronique embarquée.

4.2 Environnement de développement

La programmation Arduino sont réalisés en langages C et C++, qui tendent à avoir de meilleures performances que Python mais imposent plus de contraintes, notamment sur la déclaration des variables et la gestion de la mémoire.

Arduino propose sa propre interface de développement (IDE), qui permet à la fois de tester la compilation des codes et de les téléverser dans la mémoire des microcontrôleurs.

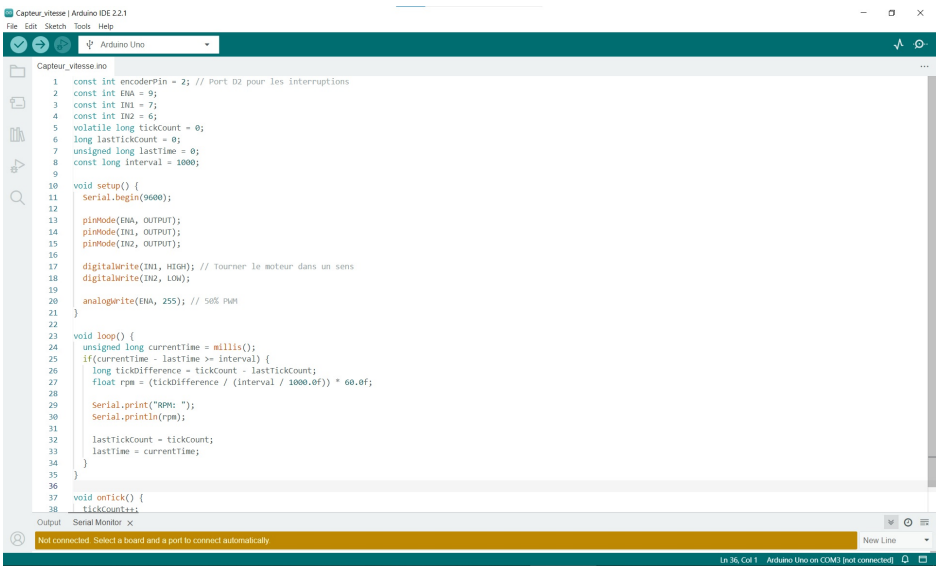
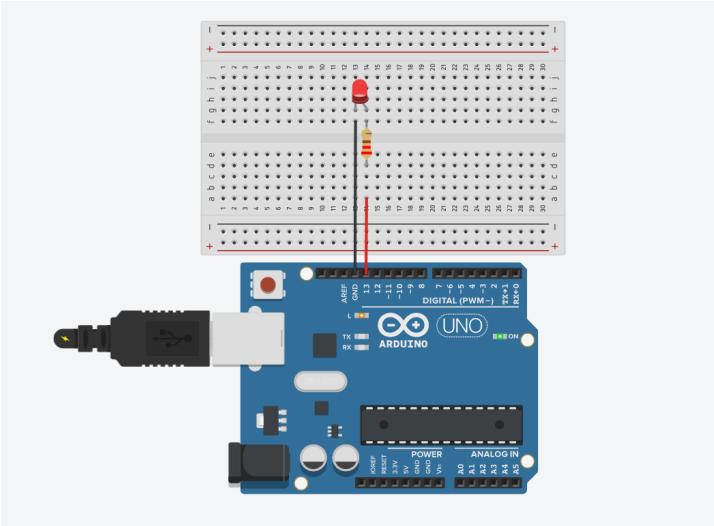


FIGURE 17 – Interface de développement Arduino

La structure des codes est imposées pour le bon fonctionnement de la carte. Un exemple est donné à partir du site de simulation **tinkercad.com** (nécessite une inscription mais gratuit pour l'instant).



Dans cet exemple, une LED est connectée aux ports `13` et `GND` de l'Arduino. L'objectif est de faire clignoter cette dernière. On identifie trois parties dans le programme :

```
1   // Pin 13 has an LED connected on most Arduino boards.
2   // give it a name:
3   int led = 13;
4
5   // the setup routine runs once when you press reset:
6   void setup() {
7     // initialize the digital pin as an output.
8     pinMode(led, OUTPUT);
9   }
10
11  // the loop routine runs over and over again forever:
12  void loop() {
13    digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
14    delay(1000);               // wait for a second
15    digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
16    delay(1000);               // wait for a second
17  }
```

Dans l'exemple proposé, le code commence par une définition de variables qui n'est pas obligatoire. Tous les codes doivent par contre être composés des fonctions `setup()` et `loop()` qui seront décrites dans une prochaine partie.

4.3 Bases de la programmation Arduino

4.3.1 Variables et types de données

En programmation Arduino, une variable est un espace en mémoire où vous pouvez stocker une valeur. Les types de données définissent la nature de cette valeur. Les types de données couramment utilisés en Arduino sont :

- `int` : Pour stocker des nombres entiers. Par exemple, `int a = 5;`
- `float` : Pour stocker des nombres à virgule flottante. Par exemple, `float x = 3.14;`
- `char` : Pour stocker un caractère unique. Par exemple, `char lettre = 'A';`
- `boolean` : Pour stocker des valeurs vrai ou faux (`true` ou `false`).
- `String` : Pour stocker des séquences de caractères. Par exemple, `String nom = "Arduino";`

4.3.2 Opérations arithmétiques de base

Arduino supporte les opérations arithmétiques classiques :
— Addition : `+`
— Soustraction : `-`
— Multiplication : `*`
— Division : `/`
— Modulo (reste de la division) : `%`

4.3.3 Structures de contrôle : conditions (if, else) et boucles (for, while)

Les structures de contrôle permettent d'exécuter certaines portions de code en fonction de conditions ou plusieurs fois de suite.
— `if` : Exécute une portion de code si une condition est vraie.
— `else` : Complémentaire à `if`, exécute une portion de code si la condition `if` est fausse.
— `for` : Exécute une portion de code un certain nombre de fois.
— `while` : Exécute une portion de code tant qu'une condition est vraie.

Exemple 1 : Utilisation de if et else

```
int nombre = 5;

if (nombre > 0) {
  nombre = nombre * 2;
} else if (nombre < 0) {
  nombre = nombre * -1;
} else {
  nombre = 1;
}
```

Dans cet exemple, si le nombre est positif, il est doublé. S'il est négatif, il est converti en valeur positive, et s'il est nul, il est défini à 1.

Exemple 2 : Utilisation de la boucle for

```
int somme = 0;

for (int i = 1; i <= 5; i++) {
  somme = somme + i;
}
```

Cet exemple utilise une boucle for pour calculer la somme des nombres de 1 à 5.

Exemple 3 : Utilisation de la boucle while

```
int i = 0;

while(i < 5) {
  i = i+1;
}
```

Dans cet exemple, la boucle `while` incrémente la valeur de i tant que celle-ci est inférieure à 5. Une fois la condition fausse (c'est-à-dire que i est égal ou supérieur à 5), le programme sort de la boucle.

4.3.4 Fonctions : définition et utilisation

Les fonctions permettent de regrouper des portions de code pour effectuer une tâche spécifique. Chaque programme Arduino a au moins deux fonctions : `setup()` et `loop()` :

— `setup()` : C'est la fonction d'initialisation, appelée une seule fois au démarrage.
— `loop()` : Comme son nom l'indique, cette fonction est exécutée en boucle, juste après `setup()`.

Vous pouvez définir vos propres fonctions pour modulariser votre code et le rendre plus lisible. Une fonction Arduino est définie par un type de retour, un nom, et éventuellement des paramètres :

```
typeDeRetour nomDeLaFonction(typeParametre1 nomParametre1, typeParametre2
    nomParametre2, ...) {
  // corps de la fonction
}
```

Imaginons que nous souhaitons créer une fonction qui prend deux nombres entiers en tant que paramètres et renvoie leur somme. Voici comment nous pourrions le faire :

```
int additionner(int a, int b) {
  return a + b;
}
```

Dans cet exemple, la fonction est nommée `additionner` et prend deux paramètres de type `int` (nombres entiers). Elle renvoie également un résultat de type `int`. Pour utiliser cette fonction dans votre code, vous l'appelleriez de cette manière :

```
int resultat = additionner(5, 3);  // resultat obtient la valeur 8
```

Note : Le type de retour des fonctions `setup()` et `loop()` est `void`. Ce type est utilisé pour les fonctions qui effectuent des tâches mais ne retournent aucun résultat une fois terminées.

La modularité fournie par les fonctions permet de simplifier et d'organiser le code, en particulier pour des programmes plus complexes. Utiliser des fonctions permet également de réduire la redondance en évitant de répéter plusieurs fois les mêmes instructions.

4.4 Utilisation des Entrées/Sorties sur l'Arduino

L'Arduino Uno possède une série de pins, ou broches, qui peuvent être configurés soit comme entrées, soit comme sorties. Ces pins vous permettent d'interagir avec une multitude de composants externes comme des LEDs, des boutons, des capteurs, des moteurs, etc. Sur l'Arduino UNO, les broches numériques sont étiquetées de `0` à `13`.

4.4.1 Pins d'entrée

Quand un pin est configuré comme une entrée, il est utilisé pour lire une tension. Si cette tension est proche de 0V, la broche lit une valeur `LOW`. Si elle est proche de 5V (pour un Arduino Uno standard), elle lit une valeur `HIGH`.

Exemple : Lire la valeur d'un bouton.

Un bouton est branché entre le port `2` et `GND` (la masse). Une résistance de pull-up interne est activée pour ce pin (ce principe sera étudié en TD).

```
void setup() {
  pinMode(2, INPUT_PULLUP); // Configure le pin 2 comme entree avec pull-up
  Serial.begin(9600); // Initialisation de la communication serie
}

void loop() {
  int buttonState = digitalRead(2); // Lecture du pin 2
  if (buttonState == LOW) {
    Serial.println("Button pressed!");
  }
}
```

4.4.2 Pins de sortie

Un pin configuré comme sortie est utilisé pour envoyer une tension. Il peut envoyer 0V (`LOW`) ou 5V (`HIGH`).

Exemple : Clignoter une LED.
Une LED est connectée au pin `13` et à `GND` à travers une résistance de 220Ω.

```
void setup() {
  pinMode(13, OUTPUT); // Configure le pin 13 comme sortie
}

void loop() {
  digitalWrite(13, HIGH); // Allume la LED
  delay(1000); // Attend 1 seconde
  digitalWrite(13, LOW); // Eteint la LED
  delay(1000); // Attend 1 seconde
}
```

4.4.3 Lecture analogique

Les pins `A0` à `A5` sur l'Arduino Uno sont des entrées analogiques et peuvent être utilisées pour lire une tension variant entre 0 et 5V. Ces pins convertissent la tension lue en une valeur entre 0 et 1023.

Exemple : Lire une tension à partir d'un potentiomètre.

```
void setup() {
  Serial.begin(9600); // Initialisation de la communication serie
}

void loop() {
  int sensorValue = analogRead(A0); // Lecture du pin A0
  Serial.println(sensorValue); // Affichage de la valeur
  delay(100); // Attend 100ms
}
```

4.4.4 Sortie analogique (PWM)

Certains pins, comme les pins `3`, `5`, `6`, `9`, `10`, et `11`, supportent la sortie PWM (modulation de largeur d'impulsion) qui peut être utilisée pour simuler une sortie analogique, comme le contrôle de la luminosité d'une LED ou la vitesse d'un moteur. Le rapport cyclique est contrôlé avec un paramètre codé sur 8 bits, et pouvant donc prendre $2^8 = 256$ niveaux.

Exemple : Variation de la luminosité d'une LED à l'aide du PWM.

```
void setup() {
  pinMode(9, OUTPUT); // Configure le pin 9 comme sortie
}

void loop() {
  for (int i = 0; i <= 255; i++) {
    analogWrite(9, i); // Definit la luminosite de la LED
    delay(10); // Attend 10ms
  }
  for (int i = 255; i >= 0; i--) {
    analogWrite(9, i); // Definit la luminosite de la LED
    delay(10); // Attend 10ms
  }
}
```

4.5 Communication Série

4.5.1 Introduction à la communication UART

UART, pour "Universal Asynchronous Receiver-Transmitter", est un type de communication série où les données sont transmises bit par bit, de façon asynchrone. Ce qui signifie qu'il n'y a pas de ligne d'horloge pour synchroniser l'envoi et la réception des signaux. Au lieu de cela, les deux dispositifs qui communiquent doivent convenir à un débit de bauds (bits/s) préétabli. L'Arduino utilise l'UART pour communiquer avec d'autres dispositifs et également pour communiquer avec le PC via le port USB.

4.5.2 Utilisation du port série pour le débogage et la communication avec le PC

L'Arduino Uno possède un seul port série que vous pouvez utiliser pour communiquer avec d'autres dispositifs ou avec votre ordinateur via l'IDE Arduino. Ce port série est essentiel pour le débogage de vos programmes.

Exemple : Initialisation de la communication série.

```
1  void setup() {
2    Serial.begin(9600); // Demarre la communication serie a 9600 bauds
3  }
4
5  void loop() {
6    // Code de votre programme
7  }
```

Exemple : Envoi d'un message sur le port série.

```
1  void setup() {
2    Serial.begin(9600);
3  }
4
5  void loop() {
6    Serial.println("Hello, World!"); // Envoie "Hello, World!" suivi d'un
         retour a la ligne
7    delay(1000); // Attend 1 seconde
8  }
```

Exemple : Lire une donnée envoyée depuis le moniteur série de l'IDE Arduino.

```
1  void setup() {
2    Serial.begin(9600);
3  }
4
5  void loop() {
6    if (Serial.available() > 0) { // Verifie si des donnees sont disponibles
7      char receivedChar = Serial.read(); // Lit le caractere recu
8      Serial.print("Vous avez envoye : ");
9      Serial.println(receivedChar); // Affiche le caractere recu
10   }
11 }
```

Ces fonctions `Serial` fournissent un moyen simple de communiquer entre votre Arduino et votre ordinateur. Utiliser la communication série est particulièrement utile pour le débogage, en vous permettant de voir les sorties de votre programme en temps réel.

4.6 Utilisation des bibliothèques

4.6.1 Qu'est-ce qu'une bibliothèque ?

Une bibliothèque est un ensemble de fonctions et de procédures qui permettent d'exécuter certaines tâches sans avoir à écrire le code à partir de zéro. Ces fonctions sont préécrites et testées, prêtes à être intégrées et utilisées dans différents projets. Dans le contexte de l'Arduino, une bibliothèque fournit souvent des moyens simplifiés d'interagir avec des composants spécifiques, comme des capteurs ou des actionneurs.

4.6.2 Comment installer et intégrer une bibliothèque

Pour installer une bibliothèque sous Arduino :

1. Ouvrez l'IDE Arduino.
2. Naviguez vers Croquis → Inclure une bibliothèque → Gérer les bibliothèques.
3. Utilisez la barre de recherche pour trouver la bibliothèque que vous souhaitez installer.
4. Cliquez sur la bibliothèque, puis sur le bouton "Installer".

Une fois la bibliothèque installée, vous pouvez l'intégrer dans votre code en utilisant la directive `#include`. Certaines bibliothèques sont disponibles par défaut à l'installation.

Exemple : Intégrer la bibliothèque Servo.

```
#include <Servo.h> // Integration de la bibliotheque Servo

Servo monServo;  // Cree un objet Servo pour controler un servo-moteur

void setup() {
  monServo.attach(9); // Defini le servo-moteur au pin 9
}

void loop() {
  monServo.write(90); // Positionne le servo-moteur a 90 degres
  delay(1000);
  monServo.write(0);  // Positionne le servo-moteur a 0 degre
  delay(1000);
}
```

4.7 Interfacer avec des capteurs

4.7.1 Convertir des données brutes en valeurs physiques

Les données lues directement à partir d'un capteur peuvent ne pas être immédiatement significatives. Par exemple, avec Arduino UNO, `analogRead` retourne une valeur entre 0 et 1023. Pour la convertir en une tension réelle :

```
float tension;                          // Variable pour stocker la tension
int valeurAnalogique = analogRead(A0);  // Lecture de la valeur

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

void loop() {
  valeurAnalogique = analogRead(A0);
  tension = (valeurAnalogique / 1023.0) * 5.0;  // Convertit en tension
  Serial.println(tension);
  delay(1000);
}
```

4.7.2 Utilisation des interruptions pour les entrées capteurs

Les interruptions sont des mécanismes qui permettent au microcontrôleur de réagir à un événement externe sans avoir à vérifier constamment l'état d'une entrée. Elles sont particulièrement utiles pour des applications qui nécessitent une réponse rapide, comme la détection d'objets avec une fourche optique.

Principe d'une interruption

Lorsqu'une condition d'interruption est satisfaite (par exemple, un changement d'état d'une broche), le microcontrôleur arrête momentanément son programme principal, exécute une fonction spécifiée (appelée routine d'interruption) avant de reprendre son programme principal.

Configuration d'une interruption avec Arduino

Voici comment configurer une interruption sur l'Arduino UNO :

```
void setup() {
  pinMode(2, INPUT_PULLUP);  // Configure le pin 2 comme entree avec resistance de pull-up interne
  attachInterrupt(digitalPinToInterrupt(2), routineInterruption, CHANGE); // Associe l'interruption a la pin 2 sur un changement d'etat
}

void loop() {
  // Votre code principal ici
}

// La fonction appelee lorsqu'une interruption est detectee
void routineInterruption() {
  // Votre code a executer lors du changement d'etat de la pin 2
}
```

Dans cet exemple, une interruption est configurée pour la broche 2 et est déclenchée chaque fois qu'il y a un changement d'état sur cette broche (passage de `LOW` à `HIGH` ou vice versa). La fonction `routineInterruption` est alors appelée.

Il existe cependant des alternatives à `CHANGE` que vous pouvez utiliser :
- `RISING` : Détecte une transition de `LOW` à `HIGH`.
- `FALLING` : Détecte une transition de `HIGH` à `LOW`.
- `LOW` : Déclenche l'interruption lorsque la broche est à un niveau bas (`LOW`).

Considérations lors de l'utilisation des interruptions

1. Gardez la routine d'interruption aussi courte que possible. Évitez les longues opérations ou l'utilisation de fonctions comme 'delay' dans cette routine.
2. Soyez conscient que toutes les interruptions ne sont pas disponibles sur toutes les broches. Sur l'Arduino UNO, seules les broches 2 et 3 peuvent être utilisées pour les interruptions externes.
3. Si vous modifiez des variables à la fois dans votre boucle principale et dans la routine d'interruption, assurez-vous d'utiliser le mot-clé `volatile` pour ces variables. Supposons que nous ayons un bouton connecté à la broche `2` et une LED connectée à la broche `13`. À chaque fois que le bouton est pressé, nous voulons que la LED change d'état (allumée si elle était éteinte, éteinte si elle était allumée).

```
volatile bool etatLED = LOW; // Variable modifiee dans la routine d'interruption

void setup() {
  pinMode(2, INPUT_PULLUP);  // Bouton
  pinMode(13, OUTPUT);       // LED
  attachInterrupt(digitalPinToInterrupt(2), changerEtatLED, FALLING);
}

void loop() {
  digitalWrite(13, etatLED); // Configure l'etat de la LED selon la variable etatLED
}

// Fonction appelee lorsqu'une interruption est detectee
void changerEtatLED() {
  etatLED = !etatLED; // Inverse l'etat de la LED
}
```

4.7.3 Gérer le bruit et filtrer les données

Le bruit est une variation indésirable ou aléatoire du signal. Pour gérer ce bruit, on peut utiliser un filtre. Voici un exemple de filtre passe-bas (suppression des variations rapides), implémenté directement dans le code.

Exemple : Filtre passe-bas pour lisser des données

```
const float alpha = 0.1;        // Facteur de filtrage
float valeurFiltree = 0;        // Valeur filtree initiale

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

void loop() {
  int valeurAnalogique = analogRead(A0);
  valeurFiltree = alpha * valeurAnalogique + (1.0 - alpha) * valeurFiltree;
  Serial.println(valeurFiltree);
  delay(1000);
}
```

Le paramètre `alpha` détermine le degré de filtrage. Une valeur plus proche de 1 donne moins de filtrage, tandis qu'une valeur plus proche de 0 donne plus de filtrage.

4.8 Commande des actionneurs

4.8.1 Pilotage d'un moteur à courant continu avec un pont en H

Comme expliqué dans une précédente section, le pont en H permet d'inverser la polarité appliquée aux bornes de l'induit d'un moteur, changeant ainsi son sens de rotation.

```
#define IN1 7  // Definir le pin connecte a IN1 sur le pont en H
#define IN2 8  // Definir le pin connecte a IN2 sur le pont en H

void setup() {
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
}

void loop() {
  // Faire tourner le moteur dans un sens
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  delay(2000);

  // Faire tourner le moteur dans l'autre sens
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  delay(2000);
}
```

Dans l'exemple ci-dessus, l'Arduino pilote un pont en H en envoyant des signaux `HIGH` ou `LOW` aux pins `IN1` et `IN2`. En alternant ces signaux, la direction du moteur change.

4.8.2 Utilisation des servo-moteurs avec Arduino

Un servo-moteur est un type d'actionneur qui peut tourner à des angles précis. Avec Arduino, il est facile de contrôler ces moteurs grâce à la bibliothèque Servo.h.

```
#include <Servo.h>  // Inclure la bibliotheque Servo

Servo monServo;  // Creer un objet Servo
int pos = 0;     // Variable pour stocker la position du servo

void setup() {
  monServo.attach(9);  // Attache le servo sur le pin 9
```

```
  }

  void loop() {
    // Faire tourner le servo de 0 a 180 degres
    for (pos = 0; pos <= 180; pos += 1) {
      monServo.write(pos);
      delay(15);
    }

    // Faire tourner le servo de 180 a 0 degres
    for (pos = 180; pos >= 0; pos -= 1) {
      monServo.write(pos);
      delay(15);
    }
  }
```

Dans cet exemple, le servo est commandé pour se déplacer entre 0 et 180 degrés. La méthode `write` de l'objet Servo est utilisée pour définir la position souhaitée du servo.

4.9 Gestion du temps et temporisation

4.9.1 La fonction `delay()` et ses limites

La fonction `delay()` est souvent utilisée dans les programmes Arduino pour créer une pause pendant un certain nombre de millisecondes. Cependant, bien qu'elle soit simple d'utilisation, elle présente des inconvénients majeurs, notamment le blocage de tout autre code pendant sa durée d'exécution.

```
  void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
  }

  void loop() {
    digitalWrite(LED_BUILTIN, HIGH);   // Allume la LED
    delay(1000);                       // Attend 1 seconde
    digitalWrite(LED_BUILTIN, LOW);    // Eteint la LED
    delay(1000);                       // Attend 1 seconde
  }
```

Dans l'exemple ci-dessus, la LED clignote toutes les secondes. Cependant, pendant la temporisation, l'Arduino ne peut pas exécuter d'autres tâches.

4.9.2 Utilisation de `millis()` pour la gestion non bloquante du temps

La fonction `millis()` retourne le nombre de millisecondes depuis le démarrage de l'Arduino. En utilisant cette fonction, vous pouvez créer des temporisations non bloquantes, permettant à l'Arduino d'exécuter d'autres tâches pendant ce temps.

```
  unsigned long tempsPrecedent = 0;
  const long interval = 1000;  // Interval de 1 seconde
  int etatLED = LOW;

  void setup() {
    pinMode(LED_BUILTIN, OUTPUT);
  }

  void loop() {
    unsigned long tempsActuel = millis();

    if (tempsActuel - tempsPrecedent >= interval) {
      tempsPrecedent = tempsActuel;

      // Inverse l'etat de la LED
      if (etatLED == LOW) {
        etatLED = HIGH;
      } else {
        etatLED = LOW;
      }
```

```
21          digitalWrite(LED_BUILTIN, etatLED);
22        }
23
24        // Autres taches ici...
25      }
```

Dans cet exemple, la LED clignote également toutes les secondes. Cependant, grâce à l'utilisation de `millis()`, d'autres fonctions ou tâches peuvent être exécutées simultanément sans être bloquées par la temporisation.

4.10 Astuces et bonnes pratiques

4.10.1 Conseils pour le débogage

Lors de la création de projets Arduino, il est fréquent de rencontrer des bugs ou des comportements inattendus. Voici quelques conseils pour faciliter le débogage :

— **Utilisation du moniteur série :** Le moniteur série est un outil précieux pour afficher des messages de débogage. Il est par contre nécessaire de disposer d'un ordinateur connecté à l'Arduino pour afficher ce message.

```
1       void setup() {
2         Serial.begin(9600);
3         Serial.println("Setup termine");
4       }
5
6       void loop() {
7         // Votre code ici...
8         Serial.println("Valeur lue : " + String(valeurAnalogique));
9       }
```

— **Diviser pour mieux régner :** Testez chaque partie de votre projet séparément. Par exemple, si vous utilisez plusieurs capteurs, testez chacun individuellement avant de les combiner.
— **Visualisation :** Utilisez des LEDs ou d'autres indicateurs visuels pour vérifier le fonctionnement de certaines parties du code.

4.10.2 Gestion de l'alimentation et protection du circuit

— **Protection contre les inversions de polarité :** Utilisez des diodes en série pour éviter les dommages causés par une inversion accidentelle de la polarité de l'alimentation.
— **Condensateurs :** Placez un condensateur près de l'alimentation de votre circuit pour stabiliser la tension et réduire le bruit.
— **Alimentation :** Assurez-vous que votre circuit n'est pas alimenté au-delà de ses spécifications. Pour les composants sensibles, envisagez d'utiliser des régulateurs de tension.
— **Protection contre les surintensités :** Utilisez des fusibles ou des circuits de protection pour éviter d'endommager votre Arduino en cas de court-circuit.

4.10.3 Notions avancées : interruptions, modes de veille

— **Interruptions :** Comme mentionné précédemment, les interruptions permettent une réaction rapide à des événements externes. Elles peuvent être utilisées pour des tâches comme le comptage d'impulsions ou la détection d'événements à haute vitesse.
— **Modes de veille :** Pour économiser l'énergie, l'Arduino peut être mis en mode veille, réduisant sa consommation d'énergie. Ceci est particulièrement utile pour les projets alimentés par batterie. Utilisez la bibliothèque 'LowPower' pour faciliter la mise en veille de l'Arduino.

```
1       #include <LowPower.h>
2
3       void setup() {
4         // Votre initialisation ici...
5       }
```

```
6
7       void loop() {
8         // Votre code ici...
9         LowPower.powerDown(SLEEP_1S, ADC_OFF, BOD_OFF);  // Dors pendant 1
              seconde
10      }
```

Ingénierie des systèmes cyber-physiques

R5.10 - Mécatronique

Travaux Dirigés 1 - Conversion analogique-numérique
Mesures continues ou en logique Tout-ou-Rien

Thomas Fromentèze

iut du Limousin INSTITUT UNIVERSITAIRE DE TECHNOLOGIE

Quelques notions de base

La numérisation des signaux analogiques repose sur l'exploitation de **convertisseurs analogique-numérique** (CAN). La complexité de la conversion dépend de la nature des signaux analogiques à utiliser, pouvant être au choix continus ou suivant une logique Tout-ou-Rien. Une simple manette de jeux vidéo répond aux mêmes problématiques :

Source images : Sony, http://henrysbench.capnfatz.com/henrys-bench/arduino-sensors-and-input/

Un joystick correspond à une association de deux potentiomètres permettant de déduire les directions à suivre selon deux axes. Les boutons peuvent être associés à de simples contacts normalement ouverts qui suivent une logique Tout-ou-Rien. Ce n'est plus vrai depuis les années 2000 avec la PS2 et la Xbox 360 où les boutons sont devenus sensibles à la pression, mais cette fonctionnalité n'est pas souvent exploitée par les développeurs.

1 Pont diviseur de tension

Pour pouvoir aller plus loin, il faut rappeler le principe du pont diviseur de tension qui servira à la mesure des deux types de capteurs présentés précédemment.

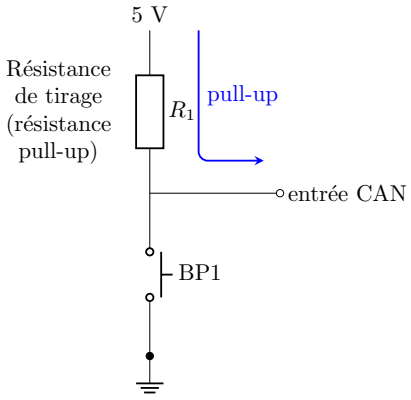
1. Rappeler les résistances équivalentes d'un voltmètre, d'un interrupteur ouvert et d'un interrupteur fermé.
2. Rappeler le circuit équivalent du pont diviseur de tension et démontrer la formule associée. On considérera une tension d'alimentation V_{cc} délivrée dans deux résistances R_1 et R_2 montées en série.

2 Mesure de l'état de boutons poussoirs par un CAN

Pour que la console puisse interpréter l'action réalisée par un joueur, les boutons peuvent être montés des deux façons présentées dans la figure suivante. Ces montages permettent de garantir que l'**entrée du Convertisseur Analogique-Numérique ne soit jamais flottante**, pour tous les états possibles de BP1.

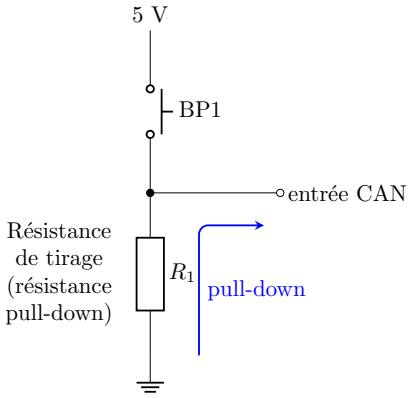
2.1 Montage pull-up

1. Calculer la tension mesurée par le CAN dans les deux états possibles du bouton BP1, avec une résistance d'entrée du CAN $R_{\text{in}} = 100\text{M}\Omega$ et une résistance de tirage $R_1 = 10\text{k}\Omega$.
2. Quelle est la valeur du courant entrant dans le convertisseur analogique-numérique ?



2.2 Montage pull-down

1. Calculer les tensions mesurées pour les deux états possibles de BP1 pour ce nouveau montage.
2. Quelle est la valeur du courant entrant dans le convertisseur analogique-numérique ?



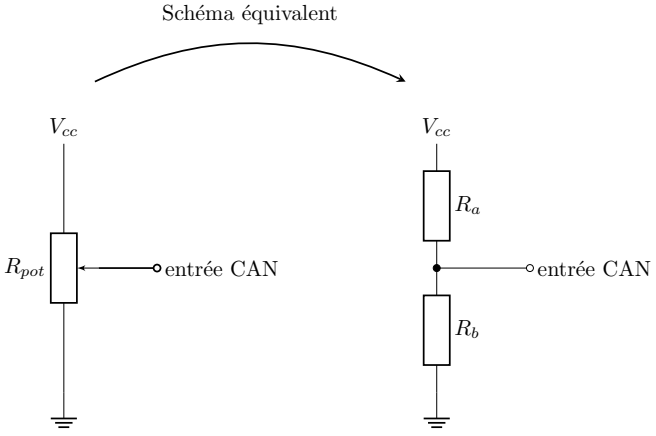
2.3 Bilan

- Les arduinos disposent d'un mode d'entrée `INPUT_PULLUP` permettant de brancher un bouton poussoir directement sur un port numérique (D0-D13). En déduire le schéma de montage de ce bouton. Quelle simplification est permise grâce à ce mode ?
- Les automates des plateformes électro-pneumatiques vues en BUT2 permettent la mesure de signaux Tout-Ou-Rien. Déduisez le montage exploité par le CAN du système dans ce cas.

3 Mesure de capteurs analogiques

Il est possible de comprendre le fonctionnement d'un joystick grâce à son schéma équivalent, donné en introduction de ce TD. La détection de position du joystick est réalisée en utilisant deux potentiomètres internes (un pour chaque axe).

1. À partir de l'analyse du schéma suivant, donner la relation entre la résistance totale du potentiomètre R_{pot} et les résistances R_a et R_b du schéma équivalent.



1. Lorsque le joueur ne touche pas au joystick, quelle est la valeur de tension mesurée par l'entrée du CAN connectée au potentiomètre Y ?
2. Représenter alors dans un graphique la tension mesurée en fonction du temps à cette même entrée lorsqu'un joueur effectue un mouvement du joystick de bas en haut.

Ingénierie des systèmes cyber-physiques

R5.10 - Mécatronique

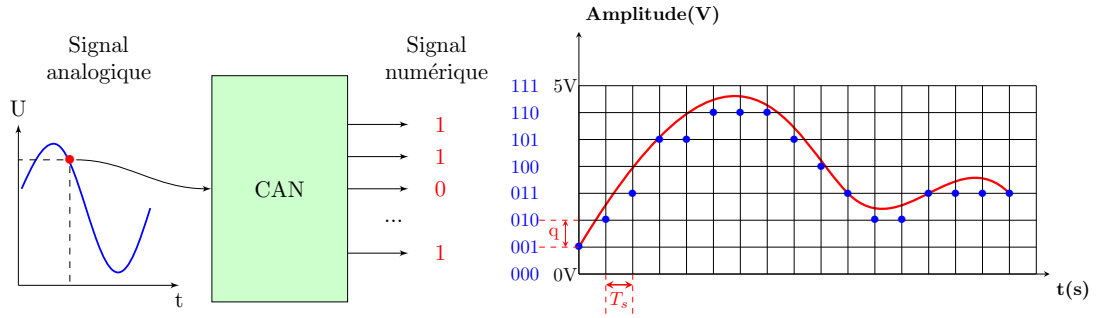
Travaux Dirigés 2 - Conversion analogique-numérique
Quantification et préparation au TP1

Thomas Fromentèze

iut du Limousin
INSTITUT UNIVERSITAIRE DE TECHNOLOGIE

1 Quantification

Suivant les éléments introduits dans le support de cours, l'objectif de cette première partie est de comprendre comment des signaux analogiques de tension peuvent être convertis en informations numériques interprétables par un microcontrôleur comme l'Arduino UNO.



1.1 Résolution en amplitude

L'arduino UNO fonctionne sur une plage de tension (la pleine échelle) comprise entre 0V et 5V. Sur les ports A0-A5, la numérisation est réalisée sur 10 bits, soit 10 valeurs de 0 ou 1 successives codant un unique échantillon mesuré à un instant donné.

1. Rappeler les deux formules du quantum données dans le cours.
2. Avec un schéma, on expliquera cette différence dans un cas simplifié en codant les informations sur des niveaux ou des intervalles.
3. Calculer la valeurs de ces quanta pour les deux variantes.
4. Considérant le code suivant pour une mesure analogique, en déduire la variante retenue pour l'Arduino UNO.

On rappelle que l'Arduino convertit directement le niveau de tension associé à un code binaire en code décimal. La variable `valeurAnalogique` est donc un nombre entier compris entre 0 et 1023.

```
1   float tension;                          // Variable pour stocker la tension
2   int valeurAnalogique = analogRead(A0);  // Lecture de la valeur
3
4   void setup() {
5     Serial.begin(9600);
6   }
7
8   void loop() {
9     valeurAnalogique = analogRead(A0);
10    tension = (valeurAnalogique / 1023.0) * 5.0;  // Convertit en tension
11    Serial.println(tension);
12    delay(1000);
13  }
```

5. On souhaite numériser un signal analogique avec une résolution en tension inférieure à 2mV. Comment atteindre cette valeur sur une pleine échelle de 5V ?

1.2 Résolution temporelle

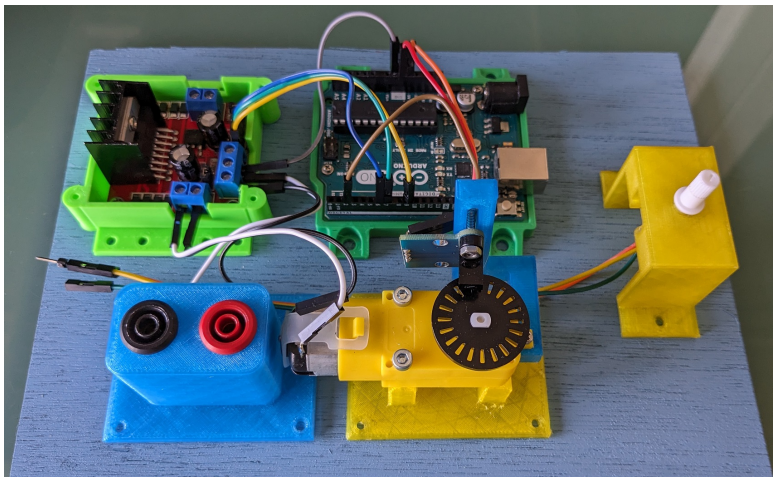
La résolution temporelle d'un CAN est habituellement associée à sa fréquence d'échantillonnage.

1. Rappeler la relation entre fréquence et période d'échantillonnage.
2. Pour information, l'Arduino UNO dispose d'une fréquence d'échantillonnage de 9.6 kHz. Quelle est la plus haute fréquence qu'il est possible de mesurer sans repliement ?

2 Préparation au TP1

La première séance de travaux pratiques porte sur le contrôle de vitesse d'un moteur.

Une version temporaire du montage est présentée ci-dessous et le code associé est donné à la page suivante :



Ce montage se compose :

- d'un moteur à courant continu.
- d'une fourche optique associée à un disque à fente monté sur le rotor.
- d'un pont en H L289.
- d'un Arduino UNO.
- d'un potentiomètre 10kΩ.
- d'un bornier permettant de relier une alimentation de tension continue.

1. Identifier les éléments sur la photo et distinguer les entrées et sorties du point de vue du microcontrôleur.
2. Considérant l'ensemble des éléments présents et le code fourni, comment fonctionne ce montage ?

   Le potentiomètre est alimenté par une tension de 3.3V et relié à la masse. Sa patte centrale est connectée au port `A0` de l'Arduino.

3. Quelle est la plage de tension qui peut être mesurée sur le port `A0` ?
4. En déduire la plage de codes décimaux qui seront retournés par le microcontrôleur.
5. Identifier enfin la partie du code qui exploite ces valeurs pour le contrôle moteur.

Précisions :

`volatile` indique au compilateur que la variable peut être modifiée à tout moment, en dehors du flux d'exécution normal du programme (ici dans `loop()` et pendant une interruption). Sans le mot-clé `volatile`, le compilateur pourrait supposer, lors de l'optimisation, que la variable ne change pas dans la boucle principale et pourrait donc ne pas lire sa valeur réelle

à chaque fois. Cela peut entraîner des erreurs difficiles à diagnostiquer.

`unsigned` indique que la variable peut seulement contenir des valeurs positives (y compris zéro). Pour un type numérique signé (c'est-à-dire sans le mot-clé `unsigned`), la plage de valeurs serait divisée entre des valeurs négatives et positives. Dans le cas d'`unsigned`, toute la plage est utilisée pour des valeurs positives, ce qui double essentiellement la valeur maximale qu'une variable peut avoir par rapport à sa version signée.

`long` est un format de données qui, sur la plupart des architectures Arduino, est codé en mémoire sur 4 octets (32 bits). Cela signifie qu'une variable `unsigned long` peut contenir des valeurs allant de 0 à 4 294 967 295 (2^32 - 1).

Code chargé dans l'Arduino UNO :

```
const int ENA = 9;
const int IN1 = 7;
const int IN2 = 6;
const int encoderPin = 2;
const int potPin = A0;

volatile unsigned long pulseCount = 0;
unsigned long previousMillis = 0;
const long interval = 1000;

void setup() {
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);

  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);

  pinMode(encoderPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(encoderPin), countPulse, RISING);

  pinMode(potPin, INPUT);  // Configuration de l'entree pour le
      potentiometre

  Serial.begin(9600);
}

void loop() {
  int potValue = analogRead(potPin);
  int motorSpeed = map(potValue, 0, 675, 0, 255);  // Adapte la valeur du
      potentiometre a la plage de controle PWM

  analogWrite(ENA, motorSpeed);  // Reglage de la vitesse du moteur selon la
       position du potentiometre

  unsigned long currentMillis = millis();

  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;

    unsigned long speed = pulseCount;

    Serial.print("Vitesse : ");
    Serial.print(speed);
    Serial.println(" unite manquante");

    pulseCount = 0;
  }
}

void countPulse() {
  pulseCount++;
}
```