UNIVERSITÉ DE SHERBROOKE

Guide de l'étudiant

APP6 – Session S3

Programmation concurrente

Faculté de génie

Été 2022

Note : En vue d'alléger le texte, le masculin est utilisé pour désigner les femmes et les hommes.

Document GRO-S3-APP6-Guide_étudiant-E22-01-JB

Rédigé par François Ferland

Modifié par Justin Brûlotte et Jean-Philippe Gouin

Table des matières

1 Activités pédagogiques et compétences

GRO300 : Systèmes d'exploitation et architecture des ordinateurs

1. (80%) Programmer des applications concurrentes en utilisant les services d'un système d'exploitation comme les processus, fils d'exécution et mécanisme de synchronisation.
2. (20%) Expliquer le fonctionnement d'un processeur et différents modèles de mémoire d'un ordinateur et analyser la performance d'un programme sur un processeur donné.

2 Synthèse de l'évaluation

La note attribuée aux activités pédagogiques de l'APP est une note individuelle, sauf pour le rapport d'APP qui est une note par équipe de deux. L'évaluation porte sur les compétences figurant dans la description des activités pédagogiques de l'APP à la section 1. Ces compétences, ainsi que la pondération (sur un total de 15000 points pour une session de 15 crédits) de chacune d'entre elles dans l'évaluation de l'unité, sont :

Activités et éléments de compétence	Rapport d'APP	Évaluation sommative	Évaluation finale
Compétence 1 – Systèmes d'exploitation	72	192	216
Compétence 2 – Architecture des ordinateurs	18	48	54
Total	90	240	270

L'évaluation sommative ainsi que l'évaluation finale porteront sur tous les objectifs d'apprentissage de l'unité.

Le pointage obtenu est ensuite utilisé pour attribuer des cotes selon cette grille :

E	D	D+	C-	C	C+	B-	B	B+	A-	A	A+
<50%	50%	53,5%	57%	60,5%	64,0%	67,5%	71,0%	74,5%	78,0%	81,5%	85,0%
0,0	1,0	1,3	1,7	2,0	2,3	2.7	3,0	3,3	3,7	4,0	4,3

3 Qualités de l'ingénieur

Les qualités de l'ingénieur visées par cette unité d'APP sont les suivantes. D'autres qualités peuvent être présentes sans être visées ou évaluées dans cette unité d'APP.

	Q01	Q02	Q03	Q04	Q05	Q06	Q07	Q08	Q09	Q10	Q11	Q12
Touchée	x	x	x	x	x							
Évaluée	x				x							

Les qualités de l'ingénieur sont les suivantes. Pour une description détaillée des qualités et leur provenance, consultez le lien suivant : http://www.usherbrooke.ca/genie/etudiants-actuels/au-baccalaureat/bcapg/.

Qualité	Libellé
Q01	Connaissances en génie
Q02	Analyse de problèmes
Q03	Investigation
Q04	Conception
Q05	Utilisation d'outils d'ingénierie
Q06	Travail individuel et en équipe
Q07	Communication
Q08	Professionnalisme
Q09	Impact du génie sur la société et l'environnement
Q10	Déontologie et équité
Q11	Économie et gestion de projets
Q12	Apprentissage continu

4 Énoncé de la problématique

Un système de diagnostic à synchroniser

Vous travaillez dans une entreprise manufacturière où plusieurs robots sont utilisés pour la fabrication de pièces et l'acheminement de composantes. Jusqu'à tout récemment, votre entreprise employait des robots commerciaux qui étaient mis en place par un consultant extérieur. Cette approche fonctionne très bien pour le moment, mais la direction de l'entreprise souhaite se distinguer davantage de la compétition. Le directeur de la recherche et du développement suggère de concevoir des systèmes sur mesure et à l'interne plutôt que d'adapter des solutions déjà existantes. La suggestion est acceptée, et une équipe a été mise en place pour explorer des solutions alternatives et concevoir des prototypes. Vous avez été invité à joindre cette équipe.

Un des premiers projets retenus est de développer un système d'acheminement de composantes par des robots mobiles se déplaçant sur des rails attachés au plafond de l'usine. Vous avez été mandaté pour mettre en œuvre le logiciel contrôlant le robot mobile sur rail que vos collègues ont conçu. Par chance, ils ont également déterminé la loi de contrôle, ce qui devrait simplifier votre tâche (voir la section 4.2). Or, votre chef de projet a ajouté quelques requis au projet.

Premièrement, le contrôle du prototype, qui ne comporte qu'un seul moteur, doit entièrement être effectué sur un microcontrôleur *Arduino Mega* déjà embarqué sur le robot. Votre chef de projet souhaite également voir un rapport sur le temps d'exécution de la routine de contrôle sur le processeur qui sera utilisé dans le produit final. Celui-ci n'est pas un *Atmel* AVR comme pour l'*Arduino Mega* et possède un jeu d'instructions complètement différent. Ce contrôle sera basé sur une boucle de rétroaction de type PID (proportionnel, intégral, dérivé) et recevra des consignes en vitesse. Ces consignes sont transmises au microcontrôleur par un port série intégré au port USB de *l'Arduino Mega*. Le protocole de communication a déjà été mis en place par un de vos collègues. Une ébauche du programme de contrôle est donc déjà disponible, mais la boucle de rétroaction est incomplète. Il faudra donc la compléter.

Deuxièmement, on vous demande également de présenter l'état du contrôle du robot en temps réel sur une interface graphique s'exécutant sur ordinateur *RaspberryPi* pour des fins de diagnostic. Une interface a déjà été mise en place pour visualiser certaines variables du système, comme la consigne (vitesse désirée), la position et la vitesse en cours du moteur. Or, ces variables d'état ne sont pas toutes communiquées à l'interface. Vous devrez donc compléter ce retour d'information en utilisant le protocole de communication qui a déjà été mis en place.

Troisièmement, le trajet effectué par le robot doit être transmis à l'équipe mécanique pour validation de la loi de contrôle. Ce trajet doit être enregistré dans un fichier facile à charger depuis *MATLAB* pour visualiser la position et la vitesse en fonction du temps. Vos collègues vous suggèrent le format CSV (*comma-separated values*). Cet enregistrement doit également être effectué sur l'ordinateur *RaspberryPi*. Le code s'occupant de cet enregistrement devra être intégré à l'interface graphique déjà disponible.

Vous vous mettez donc à la tâche en commençant par visiter votre chef de projet, qui vous livre le code déjà développé autant pour l'Arduino que pour le *RaspberryPi*. Or, il vous fait part de nouveaux requis et de l'architecture logicielle envisagée pour la suite du projet. En résumé, le prototype final comportera une dizaine de moteurs par robot sur rail, et on souhaite qu'une seule interface graphique de diagnostic puisse présenter l'état de tous ces moteurs. Le nombre de microcontrôleurs Arduino s'occupant de ces moteurs sera également variable, mais ils devront tous se rapporter à un seul *RaspberryPi* pour recevoir des consignes et communiquer leur état. Ainsi, l'interface de diagnostic sur le *RaspberryPi* devra être en mesure de synchroniser des événements arrivant de plusieurs sources dont les horloges ne sont pas nécessairement synchronisées entre elles.

Un de vos collègues, qui a développé l'interface graphique initiale, a commencé à modifier le code pour supporter l'arrivée de données simultanées. Puisque le prototype mécanique n'est pas encore disponible, et que celui dont vous disposez n'a qu'un seul moteur, il a intégré au code de l'interface un simulateur générant des événements de communication comme s'ils provenaient de multiples Arduino en plus de celui réel qui est connecté au montage.

Avec un seul moteur simulé, tout se déroule bien, et le système de diagnostic fonctionne comme prévu. Or, votre collègue rencontre des problèmes dès qu'il augmente le nombre de moteurs simulés. Parfois, certaines données sont carrément ignorées ou dédoublées, et il arrive que des données provenant d'un moteur écrasent celles d'un autre. Dans certains cas, l'application s'arrête carrément suite à une erreur d'accès en mémoire. Bref, les données ne sont pas assez fiables pour valider les lois de contrôle. Votre chef de projet y reconnait un problème classique de programmation concurrente, mais votre collègue avoue ne pas être familier avec ceci. Par chance, vous vous rappelez de vos lectures sur le sujet, et vous proposez de mettre en place des mécanismes de synchronisation pour éviter la corruption des données.

Votre chef est d'accord avec la proposition, et vous transfère donc le développement de l'interface. La mise en place de ce mécanisme de synchronisation ne devra pas affecter l'enregistrement des données du moteur réel. Vous devrez donc faire en sorte que les données soient filtrées pour que les fichiers CSV ne contiennent que les données provenant du moteur réel. Il en va de même pour les consignes en vitesse. Pour le moment, votre chef n'exige pas que l'interface commande les moteurs simulés, seulement celui réel. De plus, comme il est prévu que les données en format CSV seront acheminés à un serveur externe, votre chef demande à ce que l'écriture du fichier soit effectuée dans un fil d'exécution séparé de façon à mieux simuler le comportement du système.

À la fin de votre travail, vous devrez fournir tout le code que vous avez développé et un rapport décrivant le fonctionnement des mécanismes de synchronisation que vous avez conçus, autant pour les données à l'arrivée que pour les données à la sortie dans le fichier CSV. Vous devrez démontrer le bon fonctionnement du système en présentant des données non-corrompues du fonctionnement du moteur. Vous trouverez dans ce guide plus de détails sur les livrables attendus.

4.1 Architecture des ordinateurs et systèmes d'exploitation en robotique

Vous avez sûrement remarqué que le titre du cours associé à cet APP fait référence à deux domaines importants du génie informatique : l'architecture des ordinateurs et les systèmes d'exploitation. Dans le programme de génie informatique, ces deux domaines auraient leurs propres cours séparés, l'objectif étant de former des gens à la conception de produits dans ces domaines. En robotique, nous nous concentrerons plutôt sur l'exploitation de ces concepts. On ne vous demandera pas de créer un processeur de toutes pièces ou d'écrire un nouveau système d'exploitation, mais il sera important de bien comprendre le fonctionnement des deux lorsque viendra le moment de choisir les composantes informatiques à intégrer dans un système robotique.

L'architecture des ordinateurs décrit comment combiner des portes logiques et autres composantes numériques en un système capable d'exécuter un programme. Ici, on ne parle pas d'un fichier de code en C ou en Python, mais d'instructions machines encoder en mémoire. À la base, un processeur exécute une séquence d'instructions primitives qui opèrent sur des registres, un ensemble de mots dans une mémoire interne au processeur. Pour interagir avec l'extérieur, certaines instructions peuvent faire des transactions entre ces registres et l'espace mémoire associé au processeur. Un processeur accède à la mémoire par adresse : chaque adresse fait référence à un emplacement précis dans l'espace, et qui correspond au mot de base du processeur, par exemple un entier 32-bit. Dans la plupart des systèmes, cet espace mémoire est partagé entre la mémoire vive (RAM) et les différents périphériques qui reçoivent des plages d'adresses précises. Par exemple, sur votre Arduino, le port numérique PORTB se retrouve à l'adresse 0x0025. Ainsi, si le processeur écrit une nouvelle valeur à cette adresse, le contenu sera reproduit sur les broches du port plutôt qu'atterrir en mémoire vive. Il faut donc prendre en considération l'espace mémoire complet lorsque l'on écrit un programme interagissant directement avec le matériel.

Évidemment, si vous avez déjà programmé un Arduino, vous avez sûrement utilisé des fonctions comme `digitalWrite(…)` pour modifier l'état d'un port numérique. C'est ce qu'on appelle une abstraction, et c'est exactement le rôle du système d'exploitation de votre ordinateur. Sur Arduino, il est extrêmement primitif, et est entièrement accessible par un ensemble de bibliothèques intégrable à votre code. Par contre, sur votre ordinateur personnel ou même le *RaspberryPi*, vous disposez d'un système d'exploitation complet.

Par système d'exploitation, vous avez peut-être en tête des produits comme *Windows*, *Linux* ou *macOS*. Vous visualisez probablement différentes interfaces graphiques ou le fait qu'une application écrite pour un système d'exploitation n'est pas nécessairement disponible pour un autre. Dans le cadre de cet APP, nous nous intéresserons à la raison première d'un système d'exploitation, qui est associée à son noyau (*kernel*) : la gestion des ressources d'un ordinateur. En effet, si un Arduino est plutôt simple à gérer et n'exécute qu'un seul programme, ce n'est pas vraiment le cas de votre ordinateur personnel. Il peut exécuter des centaines d'applications en parallèle, gérer multiples supports de stockage, et communiquer avec l'extérieur de façon transparente. Prenons un logiciel comme Chrome, qui existe sur à peu près toutes les plateformes populaires, autant mobiles que fixes. Malgré une énorme différence entre une tablette Android et un poste de travail professionnel roulant CentOS, l'application comporte une bonne partie de code qui est commun à toutes les plateformes. C'est que les systèmes d'exploitation offrent à peu près toutes les abstractions utiles aux développeurs. En effet, Chrome n'a pas besoin de connaître où

exactement votre carte *WiFi* est allouée dans l'espace mémoire, il peut utiliser des bibliothèques mettant en œuvre le protocole TCP/IP et communiquer avec l'extérieur en utilisant des abstractions communes.

Pour cet APP, nous nous concentrerons sur deux ressources très liées : le temps du processeur et l'accès à une mémoire partagée entre plusieurs fils d'exécution. Nous disions plus tôt que votre ordinateur personnel pouvait exécuter des centaines d'applications à la fois. Pourtant, toutes ces applications accèdent à une seule et même mémoire vive, sans pour autant qu'une application écrase la mémoire d'une autre[1]. Comment cela est-il possible? Pensez-vous que les développeurs d'une application comme Chrome doivent savoir à l'avance quelle section de la mémoire vive leur sera allouée, et quelles sections seront réservées à d'autres applications? C'est un problème relativement simple sur Arduino, où on contrôle l'ensemble du code embarqué à la compilation, mais ce n'est évidemment pas le cas de votre ordinateur personnel. Or, si le système d'exploitation offre une certaine protection entre les applications, il faut généralement gérer soi-même les interactions à l'intérieur de ses propres applications, surtout si elles souhaitent exécuter des sous-tâches simultanément.

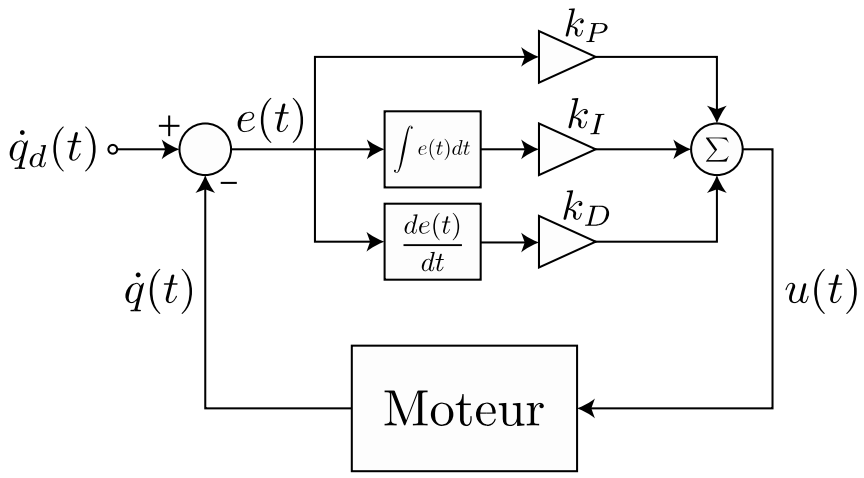
En robotique, on gère naturellement plusieurs tâches simultanément. Par exemple, l'acquisition de données peut se faire en parallèle pour chaque capteur, souvent à des horloges différentes, tout en préparant la prise de décision pour l'application des actions. Si nous faisions tout de façon séquentielle, c.-à-d. un élément à la suite de l'autre, on n'exploiterait très mal les ressources d'un système moderne, en plus d'avoir un système peu réactif. Vous verrez donc comment synchroniser plusieurs tâches et le partage d'information entre celles-ci. Dans ce cours, nous verrons ces concepts à haut niveau et avec un système d'exploitation complet (en l'occurrence Linux sur *RaspberryPi*), mais nous réutiliserons à plus bas niveau ces mêmes concepts plus tard dans le programme.

[1] Ça n'a pas toujours été le cas. Les versions de *Windows* avant 95 (excluant NT) et *Mac OS* avant *OS X* n'avaient aucune protection entre les applications. D'autres systèmes d'exploitation dont les dérivés *d'UNIX* l'offraient, mais la mémoire virtuelle et protégée était considérée trop complexe pour les ordinateurs grand public de l'époque.

4.2 Loi de contrôle

La loi de contrôle pour le chariot est un PID recevant une consigne en vitesse. La vitesse doit être ajustée à chaque fois que la consigne change. Il n'est pas nécessaire d'appliquer un profil d'accélération en particulier.

Le PID qu'on vous demande d'implémenter devra suivre cette structure classique :



Où :

- $\dot{q}_d(t)$ est la vitesse désirée à l'instant t ;
- $\dot{q}(t)$ est la vitesse à l'instant t ;
- $e(t)$ est l'erreur entre la consigne et la vitesse à l'instant t ;
- k_P est le facteur proportionnel de l'erreur ;
- k_I est le facteur de l'intégrale de l'erreur ;
- k_D est le facteur de la dérivée de l'erreur ;
- $u(t)$ est la commande transmise au moteur à l'instant t .

Le terme P permet donc de réagir à l'erreur instantanée par rapport à une consigne, le terme I à l'erreur accumulée, et le terme D aux variations de l'erreur. La commande à appliquer, une fois implémentée dans votre Arduino, peut être traduite dans le domaine du temps discret comme ceci :

$$e(t) = \dot{q}_d(t) - \dot{q}(t), I(t) = \sum_0^t e(t) = I(t\ -\ 1) +\ e(t)$$

$$u(t) = k_P\big(e(t)\big) + k_I I(t) + k_D\big(e(t)\ -\ e(t\ -\ 1)\big)$$

De plus, pour éviter de saturer les commandes, il est préférable d'appliquer des limites au terme

$$I(t) = \sum_0^t e(t)$$

4.3 Machine virtuelle pour l'analyse du PID

Tel qu'indiqué dans l'énoncé de la problématique, vous devez analyser le temps de calcul de la loi de contrôle décrite à la section 4.2. Dans un premier temps pour l'APP, vous devez implémenter la loi de contrôle en C++ sur l'Arduino. Or, vous verrez dans vos lectures que nous pouvons analyser le temps d'exécution d'un programme par sa composition en instructions machine.

La programmation en assembleur ou langage machine n'est pas au menu du programme de génie robotique. Par contre, elle permet de comprendre certains concepts des architectures des ordinateurs, et c'est pourquoi nous allons survoler le sujet avec une machine virtuelle et un langage simplifié qui servira de base pour l'analyse de votre PID. Le code de démarrage de l'APP contient un compilateur et un émulateur de cette machine virtuelle pour vous assister dans votre analyse.

Voici donc les caractéristiques de cette machine virtuelle :

- Un seul type de donnée : nombre à virgule flottante (*float*) 32-bit IEEE-754;
- Mémoire accessible par identifiants uniques, un nombre à la fois
- 4 registres (R1-R4)

Le processeur virtuel n'a pas de fonction de branchement, et ne possède que 7 instructions :

Instruction	Opérande 1	Opérande 2	Description	Cycles
`ADD`	`Ra`	`Rb`	`Ra := Ra + Rb`	3
`SUB`	`Ra`	`Rb`	`Ra := Ra - Rb`	3
`MUL`	`Ra`	`Rb`	`Ra := Ra * Rb`	3
`DIV`	`Ra`	`Rb`	`Ra := Ra / Rb`	3
`LDC`	`Ra`	`Cons`	`Ra := Cons`	2
`LDA`	`Ra`	`$A`	`Ra := mem($A)`	5
`STO`	`$A`	`Ra`	`mem($A) := Ra`	5

Où `Ra` et `Rb` sont les registres de R1 à R4, `mem($A)` représente un emplacement mémoire à l'identifiant `A` (alphanumérique, sans espaces), et `Cons` un nombre à virgule flottante (x.y). Par exemple, le programme suivant multiplie par 0.7 la valeur à l'emplacement `INPUT` et enregistre le résultat à l'emplacement `OUTPUT` :

```
LDA R1,          $INPUT     # R1 = mem($INPUT)
LDC R2,          0.7        # R2 = 0.7
MUL R1,          R2         # R1 = mem($INPUT) * 0.7
STO $OUTPUT, R1             # mem($OUTPUT) = R1 = mem($INPUT) * 0.7
```

En regardant la dernière colonne du tableau des instructions, on peut calculer que ce programme prendrait (5 + 2 + 3 + 5) cycles.

5 Connaissances nouvelles

Connaissances déclaratives : Quoi

- Multiprogrammation : gestion de processus et fils d'exécution (*threads*)
- Communication et synchronisation : sections critiques, verrous, sémaphores et moniteurs
- Systèmes de fichiers et entrées/sorties
- Architecture : unité centrale, chemins de données, impacts sur la performance
- Mémoire : types (registres, cache, mémoire vive …) et hiérarchie, gestion de mémoire paginée et virtuelle, représentations binaires (entiers et flottants IEEE 754)

Connaissances procédurales : *Comment*

- Programmer des applications parallèles et concurrentes
- Utiliser les services d'un système d'exploitation
- Expliquer le fonctionnement de différents modèles de mémoire d'un ordinateur
- Analyser les performances d'un programme par rapport à l'architecture utilisée

Connaissances conditionnelles : *Quand*

- Choisir un mécanisme de synchronisation approprié pour une application concurrente

6 Guide de lecture

6.1 Références essentielles à consulter

Toutes les références essentielles pour cet APP proviennent d'extraits de ces livres :

- « Computer Organization & Design – The Hardware/Software Interface, seconde édition », David A. Patterson et John L. Hennessy, Morgan Kaufmann, 54 pages.
- « C++ Concurrency in Action – Practical Multithreading, première édition », A. Williams, Manning Publications, 30 pages.

Autres références :

- **www.cppreference.com**, pour vos questions sur le C++. Disponible en anglais et en français.
- « Introduction au C++ - GRO-300 », François Ferland, 6 pages. Ce document est disponible sur le site web de l'APP.

6.2 Séquence d'étude suggérée

6.2.1 Pour la première activité procédurale et le laboratoire

- « Introduction au C++ » : l'ensemble du document, qui vous aidera à comprendre les exemples donnés par la suite.
- « C++ Concurrency in Action » :
  - Chap. 1, sauf 1.3 (pp. 1-9, 9 pages)
  - Chap. 2, sections 2.1 à 2.1.2 (pp. 15-19, 5 pages)
  - Chap. 3, sections 3.1, 3.2.1, 3.2.2 et 3.2.4 (pp. 33-40, 47-49, 11 pages)
  - Chap. 4, sections 4.1 et 4.1.1 (pp. 67-71, 5 pages)
- « Computer Organization & Design » :
  - Chap. 4, sections 4.1 à 4.3 (surtout un rappel de notions vues en S2) et 4.8 (pp. 210-223, 275-279, 19 pages)

6.2.2 Pour la seconde activité procédurale

- « Computer Organization & Design » :
  - Chap. 2, sections 2.1 à 2.3 (pp. 54-66, 13 pages)
  - Chap. 6, section 6.1 (pp. 436-449, 13 pages)
  - Chap. 7, sections 7.1 et 7.4 (pp. 540-544, 579-582, 9 pages)

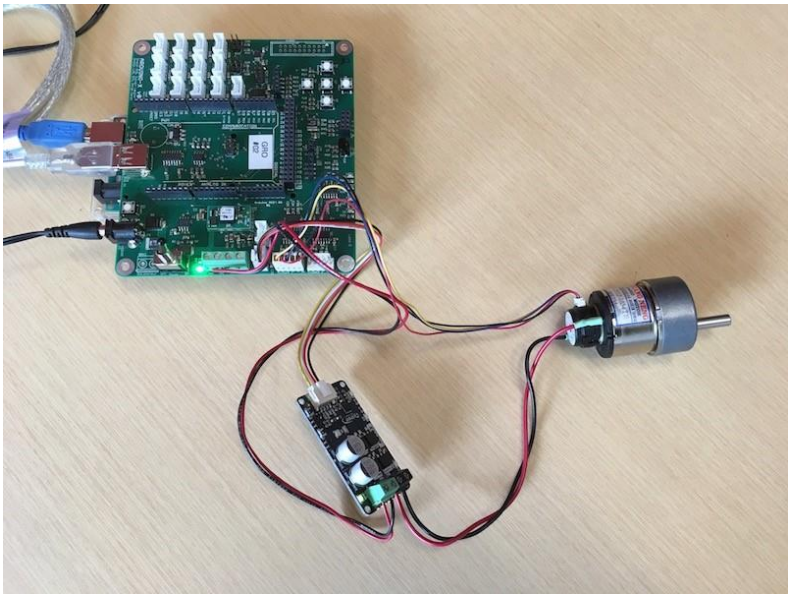
7 Logiciels et matériel

Le matériel utilisé sera le même que pour le projet de session :

- RaspberryPi, où la majorité du travail sera effectué ;
- Arduino Mega, pour le contrôle bas-niveau du moteur ;
- Matériel du projet de S3, pour le moteur et les capteurs.

Puisque le matériel est partagé par équipe de projet, la résolution de la problématique de cet APP le sera également. Or, une bonne partie du travail et des exercices en laboratoire ne nécessite pas d'avoir accès à tout le montage, mais seulement votre *RaspberryPi* ou une machine virtuelle (sur le *sharepoint*). Donc, tous les membres de l'équipe pourront participer simultanément à la résolution du problème.

Pour mettre en œuvre valider la solution, vous devrez employer un moteur/encodeur *Polulu* 12V ainsi que la carte de contrôle *Cytron*. Nous vous recommandons de les brancher dans les ports « Moteur 1 » et « Encodeur 1 » de la carte Arduino X. Le montage (sans le *RasperryPi*) devrait ressembler à ceci :



Tous les logiciels nécessaires à la résolution de l'APP sont facilement installés en suivant le guide sur le site web. Nous utiliserons donc :

- *Arduino IDE*, pour installer le code fourni et programmer le PID sur *l'Arduino Mega* ;
- *QTCreator*, pour le développement de l'interface du système d'enregistrement sur le *RaspberryPi*;
- *Geany* ou *VS Code*, pour certains des exercices du laboratoire.

Si vous avez fait des modifications importantes sur la configuration de votre *RaspberryPi*, nous vous recommandons fortement d'en faire une copie de sécurité et installer la version de démarrage pour vous assurer de ne pas rencontrer d'embûches pendant cet APP. Une copie de sécurité peut être simplement de copier le contenu de la carte SD sur votre machine.

Une machine virtuelle avec tout les outils que vous avez besoin pour la problématique est disponible. Elle va venir très pratique si vous voulez travail sans avoir le Pi :

- Télécharger la VM sur le siteweb de l'APP.
- Attendre que le téléchargement sois terminer
- Importer le fichier .ovf dans votre logiciel de VM
  - Sur Window ou Linux, il faut installer VMware Player;
  - Sur Mac, il faut installer VMware Fusion;
  - Sur les ordinateurs de l'université, il faut utiliser VMware Workstation;
  - Il devrait aussi être possible d'utiliser Virtual Box (non testé).

Mot de passe : ubuntu

8 Sommaire des activités liées à l'unité

Semaine 1 :

- 1re rencontre de tutorat
- Étude personnelle et exercices
- Formation à la pratique procédurale 1

Semaine 2 :

- Formation à la pratique en laboratoire
- Formation à la pratique procédurale 2
- Validation pratique de la solution en laboratoire **en équipe**
- Remise du rapport d'APP **par équipe**.
- 2e rencontre de tutorat - Évaluation formative

Semaine 3 :

- Consultation facultative -Évaluation sommative

9 Productions à remettre

- Validation pratique de la solution en laboratoire (mercredi 20 juillet)
- Rapport d'APP (vendredi 22 juillet, avant 8h30). Les consignes de rédaction du rapport de l'unité d'APP sont données à la section 9.3.

Tout retard sur la remise de livrable entraîne une pénalité de 20 % par jour.

9.1 Formation des équipes

La taille des équipes pour l'APP est fixée à 2. Aucune exception ne sera accordée.

La résolution de cet APP se fait en grande partie sur le Raspberry Pi, mais une partie du laboratoire et la validation s'effectuera sur le matériel du projet. **Vous devrez donc planifier à l'avance avec vos coéquipiers de projet** sur lequel de vos montages vous prévoyez valider votre solution.

9.2 Livrables

Votre solution à la problématique sera évaluée lors d'une séance de validation, mais surtout par un rapport qui décrira vos choix technologiques et le code que vous aurez développé. On attend donc une paire de livrables (rapport et code) par équipe de 2 étudiants.

Rapport

Le rapport, de 5 pages **au maximum**, devra contenir :

- Des descriptions des deux mécanismes de synchronisation que vous avez implémentés pour
  - La réception des nouveaux événements moteurs (autant simulés que réels);
  - L'accès au fichier CSV.

Le choix de chaque mécanisme devra également être justifié, en décrivant notamment le problème qu'il règle à chaque fois et leurs avantages. Pour décrire l'implémentation, vous pouvez inclure des extraits pertinents du code que vous avez modifié.

- Une petite description de l'implémentation du filtrage des données réelles pour l'inscription dans le fichier.
- L'analyse du nombre de cycles que votre PID utilise. Pour cela, vous devez vous baser sur la machine virtuelle décrite à la section 4.3, et montrer le pseudo-code assembleur que vous aurez écrit. L'analyse du PID n'a pas à inclure les tests de passage à zéro de la commande ou de débordement de signes. Il ne s'agit donc que du code calculant $u(t)$, en supposant que toutes les variables et paramètres nécessaires sont disponibles en mémoire. Utiliser un format de tableau pour présenter chaque commande.
- Une démonstration du bon fonctionnement du système d'enregistrement de l'état du moteur. Pour cela, nous souhaitons voir un graphique présentant la vitesse désirée, la vitesse en cours, et la position du moteur sur un durée d'au moins 5 secondes. Le graphique devrait montrer que votre contrôle atteint bien la vitesse désirée. Vous pouvez utiliser le logiciel de votre choix pour générer le graphique, mais nous vous suggérons un environnement comme Matlab ou SciPy (python).

Fichiers de code

Vous devrez remettre les deux fichiers de code qui implémentent votre solution :

- **main.cpp (projet Arduino)**, le code modifié pour implémenter le PID et la communication de l'état complet du système;
- **robotdiag.cpp (projet Qt)**, le système de diagnostic corrigé incluant les mécanismes de synchronisation, le filtrage des données, et l'écriture du fichier CSV dans un fil séparé.

Procédure de dépôt

Le dépôt ce faire avec l'outil de dépôt du département.

11 Évaluations

11.1 Rapport d'APP

Le contenu du rapport sera évalué selon cette pondération :

Activité pédagogique	GR	O300
Compétence	C1	C2
Contenu du rapport		
Mécanismes de synchronisation	-	-
Réception d'événements	18	-
Accès à l'enregistrement	18	-
Enregistrement des événements CSV	-	-
Filtrage du moteur réel	4	-
Justesse de l'enregistrement et graphique d'état	8	-
Analyse de la boucle du PID	-	-
Justesse du pseudo-code	-	4
Justesse de l'analyse du nombre de cycles	-	8
Validation (voir section 10.2)	24	6
Total	**72**	**18**

11.2 Validation

Votre solution sera également validée en personne lors de la séance de laboratoire de la deuxième semaine. Elle sera évaluée de façon critériée. Vous n'avez rien à ajouter au rapport en lien avec la validation, mais le pointage résultant de l'évaluation y sera ajouté lors de la correction. Voici la grille d'évaluation pour la validation:

AP		*GRO300*		
Compétence		*C1*		*C2*
Qualité		*Q01*	*Q05*	*Q05*
	Critère	*Connaître et identifier les problèmes liés à l'utilisation d'une mémoire partagée en programmation concurrente*	*Connaître et maîtriser les mécanismes de synchronisation en programmation concurrente*	*Connaître et maîtriser la transformation d'un programme en instructions machine pour en analyser la performance*
Niveaux	Pondération	12,00	12,00	6,00
Excellent (5)	100,00%	Connait parfaitement le concept de mémoire partagée et identifie plusieurs problèmes liés à son utilisation	Connaît plusieurs mécanismes de synchronisation et maîtrise l'implémentation de chacun	Sait comment transformer un programme en une série d'instructions machines pour un processeur donné, en analyser le temps d'exécution, et peut suggérer des optimisations
Très bien (cible) (4)	82,00%	Connait adéquatement la mémoire partagée et identifie tous les problèmes liés à son utilisation	Connaît au moins deux mécanismes de synchronisation et maitrise l'implémentation d'un de ceux-ci	Sait comment transformer un programme en une série d'instructions machines et est capable d'en analyser le temps d'exécution
Bien (3)	70,00%	Connait adéquatement la mémoire partagée et identifie la plupart des problèmes liés à son utilisation	Connaît au moins un mécanisme de synchronisation et maîtrise bien son implémentation	Sait comment transformer partiellement un programme en une série d'instructions machines et est capable d'évaluer le temps d'exécution

Passable (seuil) (2)	58,00%	Comprend superficiellement le concept de mémoire partagée et identifie partiellement les problèmes liés à son utilisation	Connaît au moins un mécanisme de synchronisation et maîtrise partiellement son implémentation	Sait peu ou partiellement comment transformer un programme en une série d'instructions machines et l'analyse de son temps d'exécution
Non satisfaisant (1)	25,00%	Connait superficiellement la mémoire partagée, mais est incapable d'en identifier les problèmes ou difficultés liées à son utilisation.	Connaît partiellement un mécanisme de synchronisation et son implémentation	Ne sait pas comment transformer un programme en une série d'instructions machines, mais peut analyser partiellement le temps d'exécution d'un programme donné
Non initié (0)	0,00%	Ne connait pas le concept de mémoire partagée.	Ne connait pas de mécanisme de synchronisation	Ne sait ni comment transformer un programme, ni comment en analyser la performance

11.3 Évaluation sommative

L'évaluation sommative porte sur tous les objectifs d'apprentissage de l'unité. C'est un examen théorique qui se fera **sans documentation**.

11.4 Évaluation finale

L'évaluation finale se fera par activité pédagogique et portera sur tous les objectifs d'apprentissage de cette activité pédagogique. Comme pour l'évaluation sommative, c'est un examen théorique qui se fera **sans documentation**.

12 Formation à la pratique procédurale #1

Buts de l'activité

Cette première activité procédurale mettra en pratique les concepts de programmation concurrente et des services offerts par le système d'exploitation à cet effet. Les exemples de code sont donnés en C++, mais les concepts abordés s'appliquent à n'importe quel langage basé sur une mémoire partagée entre plusieurs fils d'exécution.

12.1 Processus et fils (*threads*) (45 min, Williams ch. 1)

Q1.1) Qu'est-ce qu'un processus?

Q1.2) Qu'est-ce qu'un fil d'exécution (*thread*)?

Q1.3) Quelle est la relation entre processus et fils?

Q1.4) Est-il possible d'avoir plus de fils d'exécution que de cœurs de processeur? Si oui, comment?

Q1.5) Comment appelle-t-on le phénomène qui se produit lorsqu'un processeur passe d'un fil d'exécution à un autre? Selon vous, quel effet cela a-t-il sur la performance?

Q1.6) Quelles sont les différences principales entre les fils et les processus lorsqu'on souhaite faire de l'exécution concurrente? Nommez-en au moins 2.

Q1.7) Nommez un avantage et un inconvénient de la communication inter-processus par rapport à la communication inter-fils.

Q1.8) Pourquoi utilise-t-on la programmation concurrente? Donnez deux raisons ou motivations.

Q1.9) Qu'est-ce qu'on entend par séparation des responsabilités (*separation of concerns*) comme motivation pour la programmation concurrente? Donnez un exemple d'application où les responsabilités d'un programme seraient divisées en au moins deux tâches.

Q1.10) En quoi la programmation concurrente peut-elle améliorer la performance?

Q1.11) Pourquoi il n'est pas toujours avantageux de séparer une tâche en plusieurs fils d'exécution?

Q1.12) Quelles sont les deux grandes catégories de parallélisation du travail?

Q1.13) Donnez un exemple de traitement parallèle qui exploite la séparation des données.

12.2 Gestion de fils d'exécution (30 min, Williams ch. 2)

Q2.1) Y a-t-il un nombre minimal de fils d'exécution dans un processus? Si oui, lequel?

Q2.2) Soit ce programme en C++ :

```
#include <cstdio>
#include <thread>

int result_;

void work()
{
  result_ = 0;
  int prod = 1;
  for (int i = 2; i < 100; ++i) {
    prod *= i;
  }

  result_ = prod;
}
void start_work()
{
  std::thread t(work);
}

int main(int argc, char** argv)
{
  start_work();

  printf(«Produit des nombres de 1 à 99 : %d\n», result_);
  return 0;
}
```

Identifiez l'erreur potentielle dans ce programme.

Comment peut-on régler le problème?

Après votre correction, pouvez-vous identifier si le programme effectue du travail en parallèle? Si oui, à quel endroit?

12.3 Mécanismes de synchronisation (45 min, Williams ch. 3)

Q3.1) Selon vous, qu'est-ce qu'une opération atomique?

Q3.2) Qu'est-ce qu'une concurrence critique, aussi appelée section critique, situation de compétition, ou concurrence (« race condition »)?

Q3.3) Soit cet exemple de code C++ (chaque colonne s'exécute dans un fil différent) :

```
int compteur_ = 0;  // Variable globale accessible aux deux fils.
```

```
void fil_A()
{
  while (true) {
    int temp = compteur_;   // A1
    temp++;                 // A2
    compteur_ = temp;       // A3

    // Pause 1000 ms :
     usleep(1000 * 1000);
  }
}
```

```
void fil_B()
{
  while (true) {
    compteur_++;                  // B1
    printf("Nb. d'accès: %d\n", // B2
           compteur_);
     usleep(1000 * 1000);
  }
}
```

Pouvez-vous identifier un endroit où il peut y avoir concurrence critique?

Quel effet cela peut avoir sur la sortie du programme (commande d'affichage en B2)?

Si les deux fils n'étaient pas exécutés simultanément, et que les lignes B1 et B2 étaient toujours exécutées après la ligne A3, quelle serait la sortie du programme (commande d'affichage en B2)?

Proposez un mécanisme pour synchroniser cette concurrence critique.

Supposons maintenant que les lignes A1 à A3 sont remplacées par celle-ci :

```
compteur_++;  // A1
```

Est-ce que cette ligne de code représente une condition critique?

Q3.4) Le code suivant présente deux routines, chacune s'exécutant dans un fil différent. Pouvez-vous identifier le problème potentiel dans ce code? Si oui, comment se nomme-t-il, et comment pouvons-nous le régler?

```
// Les types et variables suivants sont accessibles aux deux fonctions et
// fils :

using mutex = std::mutex;               // Un alias pour le type de mutex.
using lock  = std::unique_lock<mutex>; // Un alias pour le type de verrou.
using queue = std::queue<int>;          // Un alias pour une liste de int.

mutex mutex_int_;                       // Mutex interne.
mutex mutex_ext_;                       // Mutex externe.
queue liste_;                           // Une liste de int
```

```
void producteur()
{
  int i = 0;
   while (true) {
     lock l_int(mutex_int_);    // P1
     lock l_ext(mutex_ext_);    // P2

      liste_.push_back(i++);     // P3
  }
}
```

```
void consommateur()
{
  while (true) {
    lock l_ext(mutex_ext_);      // C1
    lock l_int(mutex_int_);      // C2

     if (!liste_.empty()) {      // C3
       int r = liste_.front();   // C4
       liste_.pop();             // C5

       printf("A reçu %d", r);   // C6
    }
  }
}
```

```
// Quelques explications :
// P1 : Verrou interne.
// P2 : Verrou externe.
// P3 : Ajout du int i à la fin de la liste, puis incrémentation
// C1 : Verrou externe.
// C2 : Verrou interne.
// C3 : liste_.empty() retourne true si la liste est vide.
// C4 : liste_.front() retourne le premier item de la liste (s'il
existe). // C5 : liste_.pop() retire le premier item de la liste.
// C6 : Affiche à l'écran « A reçu … » et l'entier retiré de la liste.
```

12.4 Représentation des points flottants IEEE 754 (30 min, Patterson ch. 4)

Q4.1) Rappel de S2 : En représentation binaire, à quoi correspond le nombre hexadécimal 0x02F0EC91?

Q4.2) Selon la norme IEEE 754 à précision simple (*single float*), à quel nombre décimal correspond 0x42C00000?

Q4.3) En IEEE 754, quel est le prochain nombre plus grand que 0x42C00000? Quel est l'écart décimal?

Q4.4) Que doit-on faire pour contrer les erreurs d'arrondissement dans des opérations de comparaison?

13 Formation à la pratique en laboratoire #1

Buts de l'activité

Ce laboratoire a pour objectif de vous familiariser avec la programmation concurrente en C++ et l'implémentation d'un PID sur Arduino. Ces deux concepts sont évidemment nécessaires pour la résolution de l'APP, mais vous aidera également dans le développement logiciel de votre projet de session. Une vidéo qui présente la séance Linux et la compilation de code C++ par ligne de commande sera disponible sur le site web et Team. Vos lectures du livre de Williams vous seront extrêmement utiles, et pensez à consulter cppreference.com pendant le laboratoire.

Description du laboratoire

Le laboratoire s'effectue en deux parties, et la deuxième nécessite le matériel fourni pour votre projet. Vous devrez donc partager l'accès à ce matériel entre les membres de votre équipe. **Or, vous n'avez pas besoin des résultats de la première partie pour commencer la deuxième. Vous pouvez donc choisir dans quel ordre vous voulez les compléter**. Par contre, il est très important que chacun des membres de l'équipe complète tous les exercices, particulièrement pour la partie 1. Puisque cette première partie ne nécessite pas le matériel du projet, le manque d'accès ne sera pas une excuse pour ne pas avoir complété les exercices.

Pour les deux parties, vous aurez besoin du code disponible sur la page GitHub de l'APP :

https://github.com/UdeS-GRO/S3APP6r-GRO300-GRO300

Suivez d'abord les instructions sur cette page pour savoir comment télécharger le code et préparer votre espace de travail.

13.1 Partie 1 – Programmation concurrente en C++

La première partie peut être effectuée sur votre propre RaspberryPi avec l'environnement de la session, mais aussi sur votre ordinateur personnel ou les postes du laboratoire, en autant que vous disposez d'un compilateur C++11 sous Linux, ou macOS avec les outils de développement (XCode) installés. Une machine virtuelle avec le même environnement que votre RaspberryPi sera aussi mis à disposition. Or, **nous n'offrirons aucun support pour votre environnement spécifique**. Si vous n'êtes pas à l'aise à configurer votre environnement par vous-même, **on vous suggère fortement d'utiliser votre RaspberryPi et la configuration par défaut de la session ou la machine virtuelle.**

Cette première partie consiste à compléter 3 exercices de programmation en C++. Les exercices sont contenus dans des fichiers exN.cpp. Chaque fichier représente un programme individuel qui tente d'effectuer un travail parallélisé de manière concurrente sur plusieurs fils d'exécution. Or, ils comportent tous des erreurs que vous devrez résoudre. La page GitHub de l'APP pour le laboratoire (fichier lab/README.md) contient des instructions additionnelles sur la façon de compiler et exécuter ces programmes, **la première étape consiste donc à suivre ces instructions avant d'aborder le premier exercice.**

Exercice 1 – Accès concurrent à une seule variable

Ouvrez le fichier `ex1.cpp`. Vous y trouverez un programme qui calcule la somme des nombres entiers de 1 à 10000. Pour accélérer le traitement, le programme divise la tâche en quatre parties égales (1 à 2500, 2501 à 5000, etc.). Chaque partie est confiée à un fil d'exécution séparé. Le programme exécute 100 fois ce processus, et indique à la fin combien d'exécutions ont trouvé la bonne somme (50005000).

Compiler et exécuter le programme sans modifications. Est-ce que vous obtenez 100 exécutions parfaites? Probablement pas … Est-ce que le nombre de sommes justes varient d'une exécution à l'autre? Fort probablement que oui. Vous venez d'assister à un des aléas de la programmation concurrente lorsque l'on tente de paralléliser un traitement sans faire attention aux variables partagées.

Pour régler le problème et obtenir 100 sommes justes à chaque exécution, suivez ces étapes :

Q1.1) Quelle variable est partagée entre les fils d'exécution?

Q1.2) Identifiez la section critique, c.-à-d. où l'accès à la variable partagée se fait de façon concurrente.

Q1.3) Mettez en place un mécanisme de synchronisation pour contrôler l'accès à cette variable. Basez-vous sur les exemples vus dans vos lectures. Débuter par aller lire la documentation et les exemples de la classe mutex avec ce lien : https://en.cppreference.com/w/cpp/thread/mutex.

Q1.4) Performance

Selon la solution que vous avez choisie, vous avez peut-être remarqué une baisse de performance importante dans l'exécution du programme. Si le programme original (mais fautif) s'exécutait en une fraction de seconde, il en nécessite peut-être maintenant plusieurs. Il est possiblement plus lent qu'avec un seul fil d'exécution, malgré le fait que le travail semble divisé en 4. Pouvez-vous expliquer pourquoi et proposer une solution? Modifiez votre code en conséquence.

Exercice 2 – Implémentation d'un moniteur

Lorsque des événements arrivent de façon aléatoire, voir rarement, il est préférable d'être averti de l'arrivée d'un événement plutôt que de vérifier périodiquement si l'événement s'est produit. En effet, l'application de messagerie SMS de votre téléphone mobile ne vérifie pas à toutes les secondes si des nouveaux messages ont été reçus. Elle attend plutôt que le système d'exploitation lui en fasse part. On n'ouvre pas non plus notre boîte de messages à toutes les minutes pour voir ce qui s'y trouve, on attend plutôt d'avoir reçu une notification que des nouveaux messages sont disponibles.

Le fichier `ex2.cpp` présente un problème classique de producteur-consommateur où un fil produit des données à traiter (le producteur) et un autre fil les traite (le consommateur). Une file d'attente (queue) est partagée entre les deux fils pour transférer ces données. Un mécanisme de synchronisation a déjà été mis en place pour gérer l'accès à la file d'attente partagée, donc aucun problème de corruption pour l'instant.

Or, le fil du consommateur s'y prend de manière un peu brusque pour vérifier s'il y a de nouvelles données disponibles. En effet, il opère une boucle infinie qui ne prend aucune pause entre les vérifications. Pour vous en convaincre, ouvrez un deuxième terminal et lancez la commande « `top` ». Vous verrez que le programme `ex2` consomme près de 100 % d'un des cœurs de votre processeur, ce qui est anormalement

élevé pour le travail effectué. L'exercice consiste donc à régler ce problème, et les questions suivantes vous guideront vers la solution.

Q2.1) Qu'est-ce qui cause la consommation abusive du processeur?

Q2.2) Trouvez un moyen d'ajouter une pause, par exemple de 10 ms, entre les vérifications de nouvelles données. Est-ce que cela diminue la consommation?

Q2.3) Même si les vérifications sont moins fréquentes, il y a tout de même des vérifications inutiles. De plus, le système est un peu moins réactif, car il peut y avoir un délai important entre la production de nouvelles données et leur traitement. Proposez et implémentez un nouveau mécanisme de synchronisation (en plus de celui déjà en place) pour éliminer à la fois les vérifications inutiles et le délai.

Q2.4) (En extra) Le seul moyen d'arrêter le programme pour l'instant est de l'interrompre de l'extérieur (avec Ctrl-C ou la commande `kill`) vu que le fil du consommateur ne s'arrête jamais. Supposons que le fil principal veut signaler l'arrêt fil du consommateur en utilisant une variable globale, par exemple « `should_run_ = false` ». Comment feriezvous pour détecter ce changement et arrêter correctement le fil?

Exercice 3 – Comparaisons de nombres en IEEE 754

Le fichier `ex3.cpp` présente un programme tout simple qui calcule et affiche les valeurs de $y = \sin(\pi x)$ pour les valeurs de $x \in [0, 0.1, \dots, 10.0]$. Pour obtenir les valeurs de x, il incrémente une variable par 0.1. De plus, il signale par « !!! » lorsque $y = 0.0$. On s'attend donc à ce que ce signal apparaisse à toutes les valeurs entières de x. Or, ce n'est pas tout à fait le cas.

Q3.1) Expliquez pourquoi le test de $y == 0.0$ ne semble pas fonctionner, particulièrement lorsque $x == 1.0$.

Q3.2) Modifiez le code afin que le programme signale bien les valeurs de $y = 0.0$ en supposant qu'on souhaite ne conserver que 4 chiffres après la virgule pour y.

13.2 Partie 2 – Implémentation d'un PID sur Arduino

Pour cette partie du laboratoire, vous aurez besoin de tout le matériel associé à cet APP : Votre *RaspberryPi*, votre Arduino, la carte *ArduinoX* ainsi qu'un moteur et son contrôleur branché tel que montré à la section 4.2.

L'objectif est d'implémenter le contrôleur PID en vitesse tel que décrit dans la section 4.2. Tout le code de démarrage se trouve dans le sous-dossier « prob/ » sous la forme de deux applications basée sur le code fourni dans le cadre du projet : le code à embarquer sur l'Arduino et une application Qt similaire à celle utilisée pour l'exercice d'identification des moteurs.

L'application Qt permet de communiquer avec le code sur l'Arduino par le port série. C'est également l'application qui accueille le code de diagnostic à corriger pour la problématique. L'application permet d'afficher la vitesse en cours du moteur contrôlé, entrer et communiquer une vitesse désirée, et ajuster les paramètres de la loi de contrôle (facteurs P, I, et D). La vitesse en cours et désirée sont affichées dans un graphique dans le bas, et toutes les autres variables d'état communiqués depuis l'Arduino se retrouvent en format texte dans la partie inférieure.

Attention ! Comme cette application contient le code à résoudre pour la problématique, il est tout à fait normal qu'elle ne soit pas fonctionnelle à ce moment-ci de l'APP, à moins que vous n'ayez déjà résolu la partie sur la synchronisation de la problématique. Pour désactiver la partie fautive, vous pouvez tout simplement commenter la méthode qui démarre le simulateur dans le constructeur de la classe RobotDiag, reproduite ici depuis le fichier `robotdiag.cpp` :

```
RobotDiag::RobotDiag()
{
    // Démarre le simulateur:
    robotsim::init(this, 8, 10, 3);   // Spécifie le nombre de moteurs à
                                      // simuler (8) et le délai moyen entre
                                      // les événements (10 ms) plus ou moins
                                      // un nombre aléatoire (3 ms).
```

Ainsi, l'application pourra communiquer sans difficultés avec votre Arduino. Par contre, **n'oubliez pas de réactiver le simulateur pendant la résolution de votre APP !** Sinon, vous risquez de passer à côté du problème.

Ensuite, vous pourrez vous concentrer à programmer votre Arduino pour implémenter le PID. Celui-ci devra entièrement être contenu dans les fonctions `PIDmeasurement()`, `PIDcommand(double)` et `PIDgoalReached()`. Ces fonctions sont entièrement vides, vous devez les adapter au problème de l'APP. Elles doivent essentiellement faire le pont entre l'objet `pid_`, qui s'occupe de gérer la loi de contrôle, et le matériel connecté à l'ArduinoX.

Dans cet exercice, vous devez d'abord récupérer l'état des encodeurs et les communiquer à la loi de contrôle. C'est le rôle de la fonction `PIDmeasurement()`, qui doit retourner la vitesse calculée depuis les encodeurs. Ensuite, la méthode `PIDcommand(double)` doit faire l'inverse : elle reçoit la commande calculée par la loi de contrôle et la communiquer à votre moteur.

Attention ! Selon le moteur, l'encodeur et la transmission utilisée, il est possible que la conversion de vitesse soit erronée avec les valeurs constantes enregistrées dans le fichier de démarrage. Il est donc important que vous confirmiez que les constantes comme le nombre d'impulsions des encodeurs (`PASPARTOUR`) ou le ratio de transmission (`RAPPORTVITESSE`) correspondent au matériel que vous utilisez.

La communication des termes P, I et D et la consigne est déjà couverte par le code de départ. C'est le rôle du tableau `setGoal` dans la fonction `readCmd()` du code Arduino. Par contre, le retour d'information n'est pas complet. Il manque notamment les variables `cur_vel` (la vitesse en cours) et `cur_pos` (la position en cours).

De plus, l'application du côté Qt s'attend à avoir la consigne dans une variable `cmd`. Pour compléter le retour d'information, regardez le contenu de la méthode `sendCmd()`. Si vous vous demandez comment cette information est récupérée, vous pouvez consulter le fichier `mainwindow.cpp` (cette partie n'est pas à modifier).

Pour régler un PID en vitesse à la main, on débute généralement par le terme P. Lorsque celui-ci nous permet d'approcher la consigne de la consigne (mais pas complètement), on peut augmenter le terme I pour s'en approcher davantage. En contrôle en vitesse, le terme I est essentiel. En effet, si nous n'avions qu'un terme P, la commande à appliquer deviendrait nulle lorsque la consigne est atteinte, ce qui freinerait le moteur (à moins d'avoir beaucoup d'inertie dans le système). Le terme I permet donc d'obtenir la commande nécessaire pour maintenir la vitesse. Finalement, le terme D peut être utiliser pour diminuer les oscillations, puisqu'il agit comme un amortisseur.

La qualité du contrôle de votre PID, c.-à-d. le choix de vos facteurs P, I, et D, ne sera pas évaluée dans le cadre de cet APP. On évalue plutôt son implémentation, c.-à-d. comment vous avez traduit la loi de contrôle en code pour l'Arduino. et son analyse du temps d'exécution. Un bon PID sera évidemment essentiel à votre projet de session, mais considérez plutôt ce PID comme un point de départ. S'il atteint la vitesse désirée en un temps raisonnable et sans oscillations, ce sera parfaitement acceptable.

14 Formation à la pratique procédurale #2

Buts de l'activité

La seconde activité procédurale portera essentiellement sur les notions d'architectures des ordinateurs (compétence GRO300-2), notamment la performance et l'organisation de la mémoire. Si cette partie de la théorie est un peu moins utilisée lors de la résolution de la problématique, elle est tout de même importante à votre compréhension de la matière, et donc de l'évaluation de l'APP.

14.1 Mesure de la performance (Patterson ch. 2)

Q1.1) Quelle est la définition du temps de réponse (*response time*)?

Q1.2) Quelle est la définition du débit (*throughput*)?

Q1.3) Pourquoi y-a-t-il une différence entre le temps d'exécution sur l'unité centrale pour un programme (*CPU execution time*) et le temps écoulé réellement pour l'exécution de ce même programme? À quel temps correspond le temps de réponse?

Q1.4) Qu'est-ce qu'on entend par partage de temps de l'unité centrale (*CPU time sharing*)?

Q1.5) Qu'est-ce qui distingue le temps utilisateur (*user time*) par rapport au temps système (*system time*)?

Q1.6) D'un point de vue performance, pouvez-vous imaginer une situation où comparer le temps système par rapport au temps utilisateur peut être utile?

Lorsqu'une vidéo est transmise à YouTube, elle doit être traitée et compressée avant d'être disponible pour tous les utilisateurs. Supposons que YouTube fixe un temps de réponse maximal de 10 minutes par heure de vidéo reçue pour sa mise en ligne, et que la vidéo reçue a été tournée à une moyenne de 30 images par secondes.

Q1.7) Si une heure de vidéo continue est traitée par une seule unité de traitement, quel doit-être le débit en images par minute de cette unité de traitement pour respecter le temps de réponse fixé par YouTube?

Q1.8) En 2017, YouTube recevait en moyenne 300 heures de vidéo par minute. Ceci correspond à un débit de combien d'images par minutes?

Q1.9) De combien d'unités de traitement en parallèle YouTube a-t-elle besoin pour traiter toutes ces images?

Q1.10) Si le nombre d'unités de traitement est fixe, que doit-on faire pour diminuer le temps de réponse global du système?

Q1.11) Si la performance de chaque unité double, mais qu'on conserve le même nombre d'unités de traitement, qu'arrive-t-il au débit total et au temps de réponse moyen?

14.2 Instructions, registres, et cycles d'horloge

Un processeur exécute une séquence d'instructions machines, et non un programme écrit en C (ou autre) directement. Avant l'invention des langages de programmation, les programmeurs inscrivaient directement en mémoire les instructions binaires une à une à l'aide d'un panneau comme celui-ci :

Figure 1 - Ordinateur Altair 8800, commercialisé par MITS en 1975. Source: Wikimedia Commons.

C'était évidemment très laborieux comme travail, et on inventa rapidement des langages informatiques et des interfaces plus pratiques, comme les terminaux à écran et claviers. Dans le cas de l'Altair 8800, un des premiers micro-ordinateurs « grand public », il fallait tout de même programmer à la main le pilote de base qui permettait de communiquer avec un port série primitif, puisque l'ordinateur n'avait pas de mémoire morte (ROM) qui survivait d'un démarrage à l'autre. Ensuite, on pouvait programmer la mémoire vive (RAM) à l'aide d'un autre ordinateur communiquant sur ce port série. C'est le concept du *bootstrapping*, d'où vient justement le terme « *boot* » pour le démarrage d'un ordinateur.

Le rôle d'un compilateur est de transformer votre programme écrit dans un langage haut niveau en une séquence d'instructions propre à l'architecture visée. Vu qu'il s'agit d'un flux binaire, son analyse est difficile dès qu'un programme prend de l'ampleur. C'est pourquoi on emploie une représentation intermédiaire, le langage assembleur, qui est une représentation textuelle des instructions machines numériques. En assembleur, chaque ligne représente une instruction. Prenons par exemple un programme en C très simple :

```
int a = 4;
int b = 0;
b++;
a = a + b;
```

On pourrait le traduire en pseudo-assembleur ainsi (commentaires en #, Rn les registres, et $A représentant l'adresse mémoire associée à la variable A telle que déterminée par le compilateur) :

```
01 STO $A, 4        # Enregistre à l'adresse $A la valeur 4
02 STO $B, 0        # Enregistre à l'adresse $B la valeur 0
03 LDA R1, $B       # Charge dans R1 le contenu à l'adresse $B
04 ADD R1, R1, 1    # Enregistre dans R1 la somme de R1 et 1
05 STO $B, R1       # Enregistre à l'adresse $B le contenu de R1
06 LDA R2, $A       # Charge dans R2 le contenu à l'adresse $A
07 ADD R2, R2, R1   # Enregistre dans R2 la somme de R2 et R1
08 STO $A, R2       # Enregistre à l'adresse A le contenu de R2
```

Vous vous demandez peut-être pourquoi les lignes 03 à 05 ne peuvent pas être remplacées par quelque chose comme « `ADD $B, $B, 1` ». Il y a une explication très simple : Les unités arithmétiques d'un processeur ne peuvent pas opérer directement sur des emplacements mémoire. Ils ne peuvent opérer que sur des registres ou des valeurs directes. De plus, les processeurs n'ont aucune notion des boucles, seulement de branchements. Ainsi, une boucle en C qui fait la somme des nombres de 1 à 100 pourrait avoir cet équivalent en assembleur :

```
int somme = 0;
int i = 100;
while (i != 0) {
   somme += i;
   i--;
}
```

```
01 STO  R1,  0      # R1 pour somme
02 STO  R2,  100    # R2 pour i
03 ADD  R1,  R1, R2 # somme = somme + i
04 ADD  R2,  R2, -1 # i = i - 1
04 BNEZ $03, R2.    # Saut à la ligne 03 si i != 0
                    # (Branch if Not Equal Zero)
```

Quand on évalue le nombre d'instructions d'un programme, c'est de ce type d'instructions dont on parle, et non du nombre de lignes de code en C.

Exercice : Considérez ce programme en C :

```
float q_des = 1.0;  // Position désirée
float q_cur = 0.0;  // Position en cours
float cmd   = 0.0;  // Commande (en cours et à appliquer)
float k_p   = 0.5;  // Coefficient proportionnel

float err = (q_des – q_cur);
cmd = k_p * err;
```

Vous disposez de ces instructions, de 4 registres (R1 à R4), et d'une mémoire accessible par nom de variable :

- `LDC Rx, VAL     # Charge la constante VAL dans le registre Rx`
- `STO $A, Rx      # Stocke Rx en mémoire à la variable $A`
- `ADD Ra, Rb      # Ra = Ra + Rb`
- `SUB Ra, Rb      # Ra = Ra – Rb`
- `MUL Ra, Rb      # Ra = Ra * Rb`

Q2.1) Traduisez le programme en C en pseudo-assembleur à l'aide des instructions données. Associez d'abord chaque variable à un registre séparé en enregistre.

Q2.2) Si une opération arithmétique (ADD, SUB, MUL) prend 3 cycles, une d'affectation (LDC) 1 cycle, et un accès mémoire (STO) 5 cycles, en combien de temps s'exécutera votre programme si votre processeur est cadencé à 1 GHz? On rappelle que 1 GHz correspond à 1 milliard de coup d'horloge par seconde.

14.3 La mémoire (Patterson 7.1 et 7.4)

Q3.1) Qu'est-ce que le principe de localité (*locality*) lorsqu'on fait référence à la mémoire d'un ordinateur?

Q3.2) Quels sont les deux types de localité pour la mémoire?

Q3.3) Quel principe est exploité par la hiérarchisation de la mémoire?

Q3.4) Quelles sont les trois grandes catégories de mémoire généralement utilisées?

Q3.5) Excluant les registres du processeur, répondre par vrai ou faux pour la mémoire d'un seul ordinateur autonome (sans connexion réseau) :

a) La cache est l'élément de la mémoire le plus loin du processeur.
b) La cache est la mémoire la plus rapide.
c) Il est possible d'effectuer un transfert direct d'un support de stockage (ex. disque dur) jusqu'à la cache.
d) En termes de $/octet, le support de stockage est l'élément le plus cher de la mémoire d'un ordinateur.

Q3.6) La hiérarchisation de la mémoire permet de faire un compromis lors de la conception d'un ordinateur. Lequel?

Q3.7) Comment appelle-t-on le phénomène qui se produit lorsqu'on tente d'accéder à un élément en mémoire qui n'est pas déjà en cache?

Q3.8) Comment est-ce que la cache exploite le principe de localité spatiale pour optimiser les accès en mémoire?

Q3.9) Qu'est-ce que la mémoire virtuelle? Qu'est-ce qu'elle permet?

Q3.10) Qu'est-ce qui se produit lorsqu'on tente d'accéder à une page de mémoire virtuelle qui n'est pas présentement en mémoire physique?

Q3.11) À quoi sert la traduction d'adresse? À quel problème permet-elle de répondre?

14.4 Les pipelines (Patterson 6.1)

Q4.1) Qu'est-ce qu'on entend par le paradoxe du pipeline?

Q4.2) Est-ce qu'un pipeline augmente le débit ou la vitesse d'exécution des instructions d'un système?

Q4.3) Qu'est-ce qu'un aléa de pipeline (*pipeline hazard*)?

Q4.5) Pouvez-vous suggérer un mécanisme qui réduit les aléas de contrôle lors de branchement?

Soit ce programme en C, qui affiche les nombres de 1 à 100 et indiquent s'ils sont divisibles par 5 :

```
01   for (int i = 1; i <= 100; ++i) {
02     printf("Nombre : %d", i);
03     if ((i % 5) == 0) {
04         printf(" divisible par 5.");
05      }
06      printf("\n");
07   }
08   printf("Fin.\n");
```

On rappelle qu'un branchement est un saut dans l'exécution séquentielle des instructions d'un programme si une certaine condition est établie.

Q4.6) En ignorant le contenu de « `printf(…)` », identifiez quelle(s) ligne(s) provoquent des branches.

Q4.7) Supposons que votre architecture prédit que les branchements ne sont jamais pris.

Combien d'aléas de contrôle auront lieu? Détaillez votre raisonnement.

15 Validation pratique de la solution à la problématique

Dans cette activité de laboratoire, vous devrez montrer votre solution au tuteur de l'APP. Chaque équipe aura environ 10 minutes pour présenter sa solution et répondre à quelques questions qui viseront à évaluer la compréhension des deux membres de l'équipe. Vous n'avez rien à remettre à la fin de la validation. L'évaluation de la validation est décrite plus en détails à la section 10.2.

L'horaire de validation sera déterminé en fonction du matériel de votre équipe de projet que vous aurez signalé en début d'APP par le formulaire en ligne disponible sur la page web de la session.

L'horaire de validation sera affiché au moins 24 heures à l'avance.