Exercices de cours

TD 1 : programmation pour Raspberry Pi Pico

API du Pico SDK

```
// Initialise un GPIO afin de pouvoir l'utiliser comme entrée ou sortie.
// pin : numéro du GPIO (ex. 15 pour GPIO15).
gpio_init(pin);
// Configure la direction d'un GPIO.
// GPIO_IN  : le programme lit l'état électrique de la broche.
gpio_set_dir(pin, GPIO_IN);
// GPIO_OUT : le programme impose l'état électrique de la broche.
gpio_set_dir(pin, GPIO_OUT);
// Impose un niveau logique sur un GPIO configuré en sortie.
// value : 0/false = niveau bas ; 1/true = niveau haut.
// Ex. : allumer/éteindre une LED.
gpio_put(pin, value);
// Lit le niveau logique présent sur un GPIO configuré en entrée.
// Retourne false (niveau bas) ou true (niveau haut).
// Ex. : lire l'état d'un bouton ou d'un capteur numérique.
bool value = gpio_get(pin);

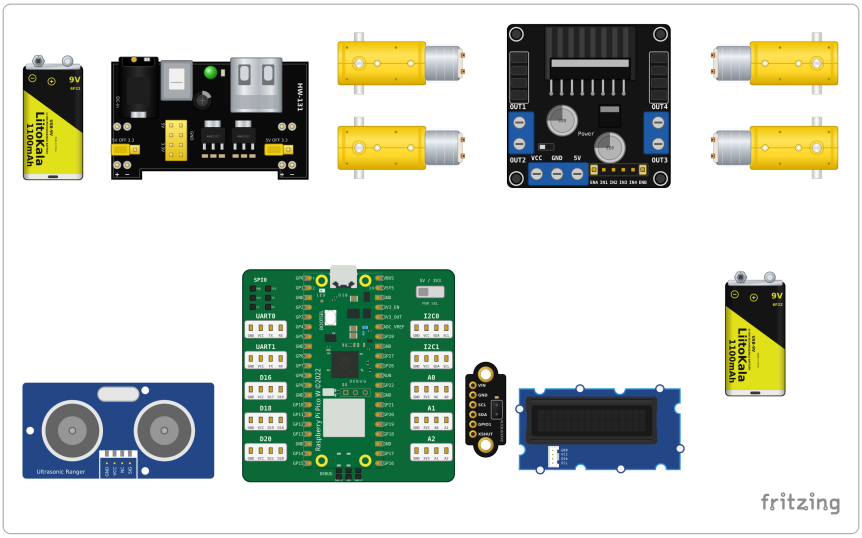
// Active une résistance interne reliant faiblement le GPIO au 3.3 V.
// Permet notamment de fixer l'état d'une entrée lorsqu'aucun composant
// externe ne lui impose de niveau.
gpio_pull_up(pin);
// Même principe, mais la résistance interne relie le GPIO à GND.
gpio_pull_down(pin);

// Suspend l'exécution du programme pendant une durée donnée.
// milliseconds : durée en millisecondes.
// Ex. : sleep_ms(1000) attend environ une seconde.
sleep_ms(milliseconds);

// Initialise les entrées/sorties standard utilisées notamment par printf().
// À appeler une fois au démarrage si le programme utilise printf().
stdio_init_all();
// Affiche du texte et des valeurs sur la sortie standard.
// %d est remplacé ici par la valeur entière passée en argument.
// Ex. : observer la valeur renvoyée par un capteur.
printf("value = %d\n", value);

// Fixe le niveau PWM produit sur un GPIO.
// value est compris entre 0 et la valeur passée à pwm_set_wrap().
// 0 = toujours bas ; max_value = presque/toujours haut selon la configuration.
// Les valeurs intermédiaires font varier le rapport cyclique.
// Ex. : contrôler la luminosité d'une LED ou la vitesse d'un moteur.
pwm_set_gpio_level(pin, value);
```

Exercice 1 : Branchements



Exercice 2 : Hello GPIO

Une LED est reliée au GPIO 15 via une résistance.

1. Configurer GPIO 15 comme sortie numérique.
2. Allumer la LED pendant 1 seconde.
3. Éteindre la LED pendant 1 seconde.
4. Répéter indéfiniment.
5. Modifier le programme pour obtenir une fréquence de clignotement de 5 Hz avec un rapport cyclique de 50 %.
6. Modifier ensuite le clignotement pour obtenir 100 ms allumé et 900 ms éteint.
7. Écrire une fonction, qui allume puis éteint une LED, chaque phase durant `duration_ms` millisecondes, telle que :

```
void blink(uint pin, uint duration_ms);
```

Exercice 3 : Lire un bouton

Un bouton relie GPIO 14 à GND lorsqu'il est appuyé.

1. Configurer GPIO 14 comme entrée.
2. Activer sa résistance interne de pull-up.
3. Lire son état toutes les 100 ms.
4. Afficher l'état brut sur le terminal série sous la forme :

```
button = 1
button = 1
button = 0
...
```

5. Ajouter une variable booléenne `pressed` qui vaut `true` lorsque le bouton est physiquement appuyé.

Questions :

- Pourquoi utilise-t-on ici une résistance de pull-up ?
- Quelle valeur logique est lue lorsque le bouton est relâché ?
- Quelle valeur logique est lue lorsqu'il est appuyé ?
- Pourquoi `pressed` peut-il être l'inverse de la valeur retournée par `gpio_get()` ?

Exercice 4 : Utilisation des moteurs

Le robot possède quatre moteurs, regroupés deux par deux : les deux moteurs de gauche sont commandés ensemble, de même que les deux moteurs de droite. Chaque groupe est piloté par un pont en H à l'aide de deux broches du GPIO.

Pour un groupe de moteurs, la vitesse est comprise entre $[-255, 255]$:

- le signe indique le sens de rotation ;
- la valeur absolue indique la puissance appliquée par PWM ;
- 0 correspond à l'arrêt.

1. Écrire la fonction `void controleMoteurs(int vitesseGauche, int vitesseDroite)` telle que :

   ```
   controleMoteurs(255, 255);     // avancer à pleine puissance
   controleMoteurs(128, 128);     // avancer à environ 50 %
   controleMoteurs(-255, -255);   // reculer à pleine puissance
   controleMoteurs(0, 0);         // arrêt
   ```

   On utilisera les constantes suivantes :

   ```
   #define LEFT_MOTOR_PIN_A   ...
   #define LEFT_MOTOR_PIN_B   ...
   #define RIGHT_MOTOR_PIN_A  ...
   #define RIGHT_MOTOR_PIN_B  ...
   ```

   Pour chaque groupe de moteurs :

   - si la vitesse est positive, appliquer le PWM sur la broche A et mettre B à 0 ;
   - si la vitesse est négative, mettre A à 0 et appliquer le PWM sur B ;
   - si la vitesse est nulle, mettre les deux broches à 0.
2. Qu'est-ce que produit l'appel suivant ?

   ```
   controleMoteurs(200, -200);
   ```

Exercice 5 : Missions

À partir de l'API de commandes décrite dans le cours :

1. Écrire un programme qui fait avancer le robot pendant 1 seconde, puis reculer le robot pendant 1 seconde.
2. Écrire un programme qui fait tourner le robot de 90° vers la gauche.

TD 2 : comportement autonome avec un contrôleur PID

Équation d'un contrôleur PID en temps continu :

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)\,d\tau + K_d \cdot \frac{de(t)}{dt}$$

Et en version discrète, pour une implantation numérique :

$$u[n] = K_p \cdot e[n] + K_i \cdot \sum_{k=0}^{n} e[k] + K_d \cdot (e[n] - e[n-1])$$

Expliquer le rôle de chacune des composantes de la fonction discrète du PID, sachant que :

- $u[n]$: commande envoyée au système au pas de temps n
- $e[n]$: erreur mesurée au pas de temps n, avec $e[n] = distance_{cible} - distance_{mesurée}$

Pour simplifier :

- erreur proportionnelle e_p : distance voulue – distance mesurée
- erreur intégrale e_i : somme des e_p calculées depuis le début
- erreur dérivée e_d : $e_p[n] - e_p[n-1]$

Remarque. La période d'échantillonnage Δt est fixe. Elle est intégrée aux gains : K_i contient le facteur Δt de l'intégrale, K_d contient le facteur $1/\Delta t$ de la dérivée.

Exercice 1 : composante proportionnelle (P)

Comment le terme proportionnel influence-t-il la réaction du robot ?

Un robot doit maintenir une distance de 10 cm avec un mur lorsqu'il le longe. Sa distance actuelle au mur est de 6 cm.

1. Que signifie $e[0]$? Calculez sa valeur.
2. Si le gain proportionnel $K_p = 0.5$, quelle est la correction appliquée ?
3. Que se passe-t-il si $K_p = 2$?

Exercice 2 : composante intégrale (I)

Comment l'intégrale corrige-t-elle les erreurs persistantes ?

Le robot reste trop près du mur pendant plusieurs pas de temps. Aux trois premiers pas de temps : $e[0] = 4, e[1] = 4, e[2] = 4$. On garde $K_p = 0.5$ et on prend un gain intégral $K_i = 0.1$.

1. Calculer la commande totale $u[2]$ après 3 mesures (composantes P et I).
2. Quelle est la part due à l'intégrale ?

3. Que se passe-t-il si K_i est trop grand ?

Exercice 3 : composante dérivée (D)
Comment la dérivée anticipe-t-elle les changements rapides ?

Le robot s'éloigne du mur. On poursuit l'exercice 2 : $e[2] = 4$, puis $e[3] = 1$. On garde $K_p = 0.5$ et $K_i = 0.1$, et on prend un gain dérivé $K_d = 0.2$.

1. Calculer la variation d'erreur $e[3] - e[2]$.
2. Calculer la correction due à la dérivée.
3. Quelle est la commande totale $u[3]$?

Exercice 4 : pertinence des composantes
Comment diagnostiquer un mauvais réglage du contrôleur ?

- Situation A : le robot oscille autour de la distance cible.
- Situation B : le robot corrige trop lentement sa trajectoire.
- Situation C : le robot dépasse systématiquement la distance cible.

1. Que dire de K_p, K_i et K_d dans chacune de ces trois situations ?
2. Comment peut-on corriger ces dysfonctionnements ?

TD 3 : distribution du calcul et étude des compromis

Le robot embarque un Raspberry Pi Pico WH :

- microcontrôleur RP2040 : deux cœurs Cortex-M0+, sans unité de calcul flottant ;
- 264 KB de SRAM, 2 MB de mémoire flash ;
- puce radio CYW43439 : Wi-Fi 802.11n dans la bande 2,4 GHz.

Le robot crée son propre point d'accès Wi-Fi. Le téléphone s'y connecte, lit la télémétrie par `GET /api/lines` et envoie les commandes par `POST /api/command`.

Exercice 1 : temps de calcul et fréquence de lecture

Le capteur VL53L1X mesure en continu, indépendamment du processeur. Il produit au plus une mesure toutes les 20 ms (**50 Hz**).

- Pico WH cadencé à 125 MHz
- Lecture d'un résultat sur le bus I2C : ~0,5 ms
- Calcul d'un PID en `float` : ~20 µs

1. La boucle de contrôle attend chaque nouvelle mesure, la lit, puis calcule le PID. Quelle est la fréquence maximale de cette boucle ? Quel élément la limite ?
2. Quelle part du temps processeur la lecture et le calcul occupent-ils à cette fréquence ? Citer d'autres tâches du robot qui utilisent le reste.
3. Que se passe-t-il si l'on fait tourner la boucle à 100 Hz ?
4. Proposez une fréquence raisonnable pour un robot roulant à 0.5 m/s.

Exercice 2 : occupation mémoire

On veut stocker un historique de 100 erreurs pour calculer un PID discret.

- Pico WH : 264 KB de SRAM
- Chaque erreur est un `float` (4 octets)

1. Quelle quantité de mémoire est utilisée pour stocker 100 erreurs ? Quelle part de la SRAM cela représente-t-il ?
2. Quelles valeurs le PID discret du TD 2 doit-il réellement conserver d'un pas de temps au suivant ? Combien d'octets faut-il ?
3. On veut garder sur le robot un historique de la télémétrie pour le tracer sur le téléphone. Un échantillon contient `ms` (`uint32_t`), `front_mm`, `side_mm`, `left` et `right` (`int16_t` chacun). À 50 Hz, combien d'octets par seconde faut-il ? Combien de secondes tiennent dans un tampon de 32 KB ?
4. Proposer une solution pour réduire cette empreinte mémoire sans perdre l'information utile.

Exercice 3 : énergie et efficacité

Un robot utilise un PID mal réglé qui provoque des oscillations.

- Les moteurs consomment 100 mA en ligne droite
- En cas d'oscillations : consommation moyenne = 150 mA
- Batterie des moteurs = 1000 mAh

1. Quelle est l'autonomie des moteurs avec et sans oscillations ?
2. Quel est le gain en autonomie si le PID est bien réglé ?
3. Quelle stratégie PID permet de réduire les oscillations ?

Exercice 4 : précision et bruit de mesure

Le robot est immobile face à un mur situé à 200 mm. Le VL53L1X renvoie des distances entières en millimètres.

- Mesures successives : 198, 202, 195, 205, 200
- Distance cible du PID : 200 mm

1. Calculez l'erreur moyenne et l'écart-type des mesures.
2. Calculez les termes $e[n] - e[n-1]$ du PID pour cette série. Quelle est l'influence du bruit sur le terme dérivé ?
3. Proposez une méthode pour filtrer les mesures avant le calcul du PID. Quel est son coût en latence à 50 Hz ?

Exercice 5 : latence dans une architecture distribuée

Pour alléger le Pico, on propose de calculer le PID sur le téléphone, dans `robot.js`. Le robot envoie ses mesures dans la télémétrie ; le téléphone calcule la commande et la renvoie par Wi-Fi.

- Capteur VL53L1X : une mesure toutes les 20 ms
- Télémétrie : une ligne toutes les 100 ms (`{"telemetry":100}`)
- Le téléphone lit les lignes en attente par `GET /api/lines` toutes les 100 ms
- Trajet d'une requête ou d'une réponse HTTP sur le Wi-Fi : ~5 ms quand le canal est libre, ~50 ms quand il est encombré
- Calcul du PID dans le navigateur : ~1 ms

- Vitesse du robot : 0.5 m/s

1. Quelle est la latence entre la mesure et l'application de la commande, dans le meilleur et dans le pire cas, quand le canal est libre ?
2. Même question quand le canal est encombré.
3. Quelle distance le robot parcourt-il pendant ces latences ?
4. Est-ce acceptable pour un robot devant éviter des obstacles à 20 cm ?
5. Une salle de TP compte une douzaine de robots, chacun avec son point d'accès 2,4 GHz. Combien de canaux de 20 MHz ne se chevauchent pas dans cette bande ? Quel est l'effet sur la latence ?
6. Proposez une stratégie pour réduire l'impact de cette latence (*e.g.* prédiction, calcul local, hybridation).