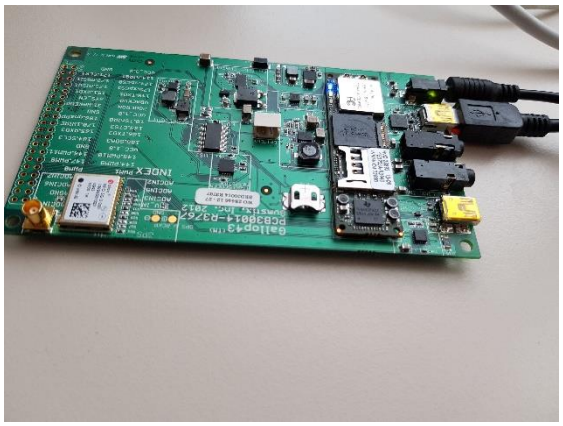
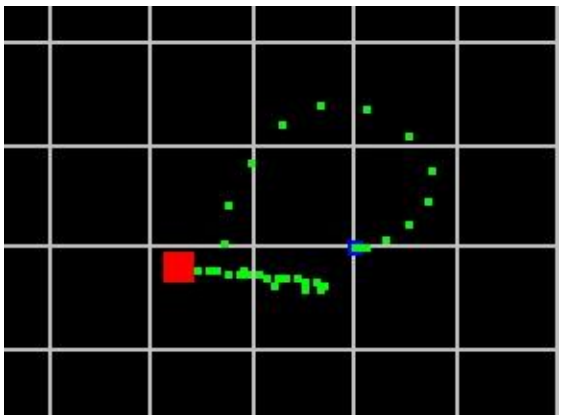




Exploitation des mesures d'un accéléromètre

PROJET AVEC GUMSTIX

Johan Gras | Informatique Embarquée | 2018

Table des matières

Introduction

VUE D'ENSEMBLE

Ce rapport décrit l'implémentation d'une visualisation des déplacements d'un système embarqué en se basant sur l'accéléromètre de ce dernier.

Le projet a été réalisé sur un microcontrôleur Gumstix Overo Air et une carte d'extension Gallop43 sous Linux Angström, cependant à quelques spécificités techniques près les méthodes employées sont facilement généralisables à d'autres architectures.

Ce document peut ainsi être utilisé comme ressource technique pour la réalisation d'un projet similaire, malgré que les méthodes employées soient assez classique il réunit de nombreuses parties distinctes en seulement quelques pages. La seule partie non conventionnelle (et optionnelle) est l'implémentation d'un algorithme d'adaptation automatique de la période des tâches.

OBJET DU PROJET

Le projet consiste à afficher sur un PC distant la position et les déplacements (horizontaux) d'une carte embarquée communiquant par WIFI.

Cet objectif implique plusieurs besoins fonctionnels :

- Récupération des valeurs de l'accéléromètre dans les registres : aller chercher les données avec les fonctions d'entrée sortie du bus i2c en un minimum d'opérations.
- Obtention de la position relative après traitement des données : calibration, filtres, double intégration,…
- Gestion du temps réel mou : traitement optimal des tâches et des échéances.
- Communication entre le système et le PC : liaison wifi ad-hoc et communication UDP.
- Affichage graphique : représentation de l'évolution de la position.

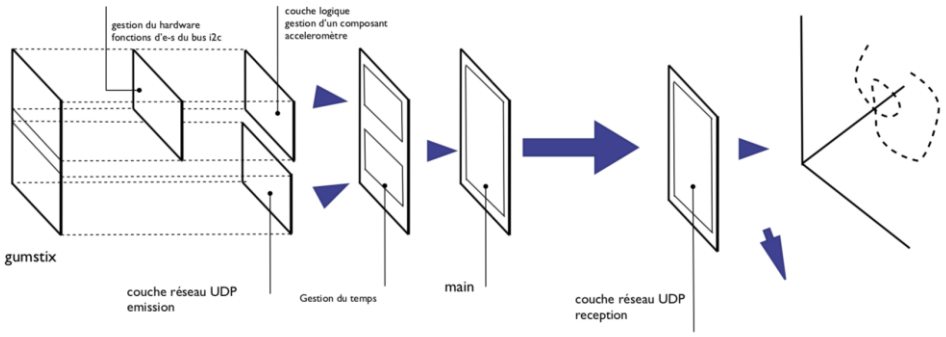


Schéma d'ensemble

Aspect technique et réalisation

COMMUNICATION

Une communication entre un PC et la carte embarquée est nécessaire pour 2 raisons. Dans un premier temps pour pouvoir accéder à la console du système ou encore transférer les programmes à exécuter (après les avoir au préalable cross-compilé sur le PC pour éviter d'encombrer la cible de dépendances facultatives). Et puis, dans un second temps, pour être capable de transférer en temps réel les données contenant la position avec le protocole UDP.

- Pour subvenir à notre premier besoin, une liaison série UART fait parfaitement l'affaire. On peut ainsi accéder à la console du système cible avec un programme comme Minicom et transférer des données si rz est installée.

- Cependant, la liaison UART ne supporte pas UDP nativement, c'est pourquoi on privilégiera l'utilisation d'une liaison Wifi Ad-Hoc qui elle le supporte. On peut accéder à la console en SSH et transférer les fichiers avec la commande scp. Seul bémol l'utilisation de SSH peut être bien plus gourmand en ressources si la quantité d'affichage est importante.

En dehors de l'aspect matériel, l'implémentation de la communication des données est assez simple. Sur le serveur situé sur le PC, on attend que des paquets arrivent pour les traiter, tandis que sur le client on envoie des paquets contenant les données de la position dès que celle-ci a été calculée.

UDP vs TCP

On préférera le protocole UDP qui malgré une communication moins fiable que son confrère, est beaucoup moins gourmand.

Calcul de la position sur le système vs sur le PC

On préférera réaliser les opérations de calcul de position sur le système embarqué malgré un alourdissement de la charge de travail, car si on envoyait directement au PC l'accélération brute, la position finale pourrait être bien plus erronée, car une erreur de transmission de la position est moins gênant qu'une erreur de transmission de l'accélération.

RECUPERATION DE L'ACCELERATION

L'accéléromètre va enregistrer ses mesures courantes dans des registres spécifiques. Il va alors falloir les récupérer en utilisant des fonctions du bus de communication i2c. Un programme existant permet déjà de faire cela (i2cget/i2cset).

Cependant, le code de i2cget est assez complexe, long et contient de nombreuses dépendances car il est conçu pour pouvoir réaliser un large ensemble de tâches. Seule l'accélération brute nous intéresse. Il est alors possible de réduire considérablement la taille du code en enlevant tout le code superflu et les « sécurités inutiles ».

```
void get_acc (int *x, int *y)
{
    file = open(FILENAME, O_RDWR);
    ioctl(file, I2C_FUNCS, &funcs);
    ioctl(file, I2C_SLAVE_FORCE, ADDRESSBUS);

    ioctl(file,I2C_SMBUS,&argsX);
    *x = (int8_t)data.byte;

    ioctl(file,I2C_SMBUS,&argsY);
    *y = (int8_t)data.byte;

    close(file);
}
```

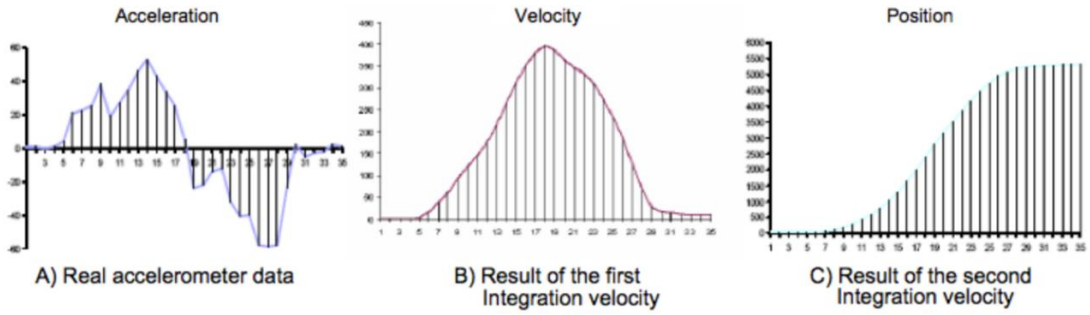
Les valeurs de sortie de l'accéléromètre sont codées en complément à 2. Un cast en int8_t, permet d'obtenir la valeur signée correspondante.

OBTENTION D'UNE POSITION ACCEPTABLE

Après avoir obtenu l'accélération, il est désormais possible mathématiquement d'obtenir la position relative en effectuant une double intégration de l'accélération.

$$v = \int(\vec{a})dt \text{ and } \vec{s} = \int(\vec{v})dt \therefore \int\left(\int(\vec{a})dt\right)dt$$

L'accélération étant la dérivé seconde de la position, on a pris la formule dans l'autre sens. On peut aussi se représenter la formule de manière plus intuitive :



Une représentation graphique du passage de l'accélération à la position

Cependant, la réalité n'est pas aussi parfaite que la théorie.

On utilise un signal de l'accélération discret et bruité qui ne permet pas de retrouver la position instantanée du système. Pour répondre à ces différentes variables on utilisera un ensemble de techniques visant à réduire l'incertitude :

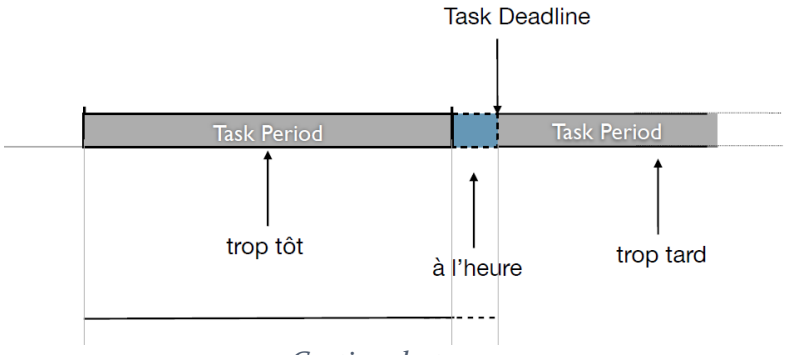
1. Méthode des trapèzes pour l'intégration : utilisée pour réduire l'erreur d'intégration sur des valeurs discrètes.
2. Calibration : permet d'éliminer le biais de la mesure.
3. Filtre basse fréquence : réalisé avec une moyenne mobile (moyenne des échantillons) permettant d'atténuer le bruit mécanique et électrique de l'accéléromètre.
4. Fenêtre de filtrage : ignore les valeurs d'accélération proches de zéro pour contrer le bruit lors des phase de non mouvement.
5. Vérification de fin de mouvement : force l'annulation de la vitesse si l'accélération est nulle pendant une certaine période.

Ma démarche a fortement été inspiré du papier AN3397 - « Implementing Positioning Algorithms Using Accelerometers ».

GESTION DES ECHEANCES

Dans ce projet nous sommes dans une situation de temps réel mou, c'est-à-dire que le respect des échéances des tâches à réaliser est très important. Néanmoins une échéance manquée n'est pas critique pour le bon fonctionnement du système.

Notamment, la formule qu'on utilise pour réaliser la double intégrale de l'accélération nécessite un échantillonnage régulier, c'est pourquoi il est nécessaire de définir une période d'exécution de la tâche (task period), ainsi qu'un laps de temps le plus court possible pour la réaliser (task deadline).



Gestion du temps

Dans un premier temps, l'implémentation d'un code classique pour la gestion des échéances a fait l'affaire : attente active du début de la période, puis exécution de la tâche si on est dans les temps ou ajout d'une échéance manquée dans le cas échéant. Le seul bémol est l'utilisation d'une task period fixée à l'avance (choisi grâce à un test progressif des différentes valeurs), cependant une différence dans la charge de travail du CPU par rapport à la phase de test peut venir fausser la valeur idéale et de ce fait engendrer de nombreuses échéances manquées.

Task period adaptative

Pour tenter de résoudre ce problème, j'ai conçu un algorithme permettant de choisir dynamiquement une task period (couplé à la task deadline) en traquant le pourcentage d'échéances manquées.

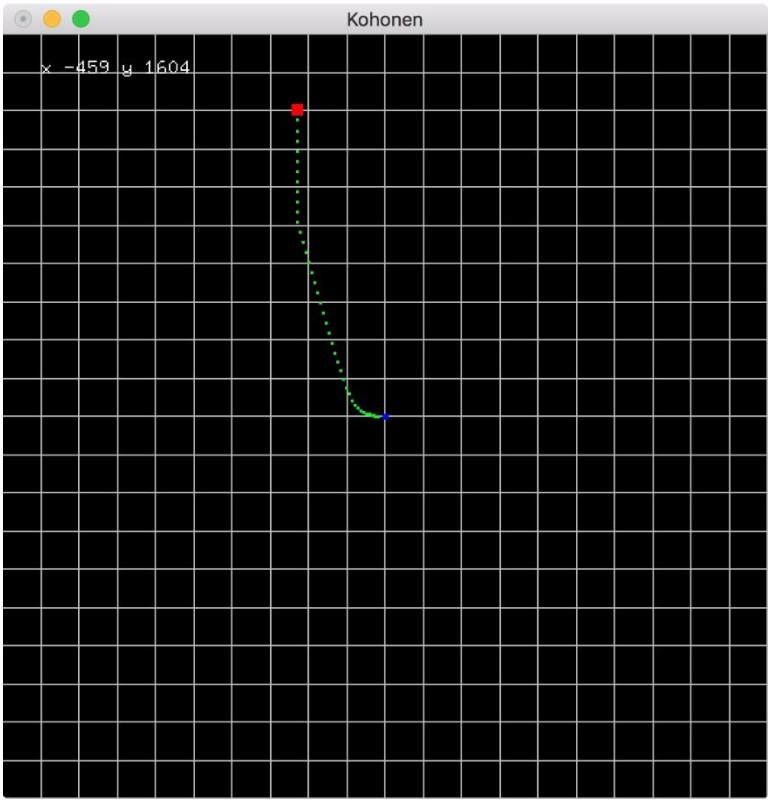
Voici le fonctionnement global de l'algorithme :

1. Utilisation d'une task period de plus en plus petite à chaque mesure tant qu'il n'y a pas d'échéance manquée.
2. Si une échéance est manquée on réalise de nouveau plusieurs mesures avec la même task period pour s'assurer que l'échéance manquée n'était pas exceptionnelle. Si c'est le cas on retourne à l'étape 1.
3. Si les échéances manquées sont confirmés, on va progressivement ré-augmenter la task period (proportionnellement aux nombres d'erreurs) jusqu'à ce que l'on n'en obtienne plus sur nos échantillons.
4. Si on n'a pas d'échéances manquées sur le dernier échantillon on garde alors la task period choisie, tout en surveillant une éventuelle augmentation du nombre d'erreurs.

L'algorithme n'est certainement pas parfait et laisse la place à de nombreuses expérimentations et variantes. Il permet néanmoins une convergence en quelques secondes et malgré cela, il est toujours possible d'utiliser le système en pré-stabilisation. De plus, en post-stabilisation il peut toujours s'adapter à une hausse des échéances manquées.

AFFICHAGE DESKTOP

L'affichage graphique permet de suivre les mouvements du système depuis un PC externe. On peut ainsi comparer facilement la trajectoire calculée par rapport à la trajectoire réelle.



L'application a été codée avec la librairie graphique OpenGL et communique directement en UDP avec la carte cible.

Le point rouge représente la position relative de la Gumstix. La trainée verte, ses positions récentes. Le point bleu, l'accélération instantanée. En haut à gauche on aperçoit la position numérique de cette dernière.

Après une période sans mouvement le centre de référence de l'image se recentre sur la Gumstix.

Mesure des performances

PERIODE DES TACHES

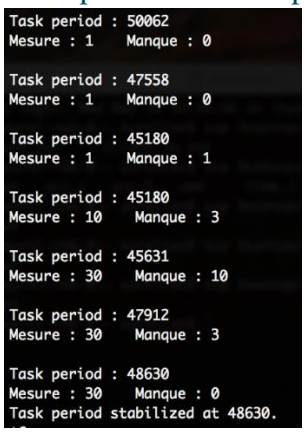
Sans period task adaptative

Après avoir réalisé de nombreux tests en réduisant la période, il est possible de conclure empiriquement.

- Dans un cas d'utilisation normale (sans stress test), quasiment toutes les échéances sont accomplies correctement (quelques exceptions arrivent parfois). Cependant, dès qu'une certaine limite est dépassée le ratio d'échéance passe directement à 50% (1 tâche sur 2 n'est pas effectuée dans les temps). Cela signifie que la tâche précédente s'exécute en plus de temps que la période elle-même. Sur ma Gumstix, j'ai remarqué qu'une période de 50 ms est une bonne valeur.

- Dans un cas d'utilisation extrême cependant (avec stress test), il n'est pas possible d'établir une borne à partir de laquelle il n'existe plus d'échéance manquée. Pour à peu près n'importe quelle période raisonnable on obtiendra toujours des échéances manquées. Je n'ai donc pas pu construire de règle exacte pour gérer ces conditions extrêmes. En revanche, une augmentation de la task deadline peut grandement améliorer les performances de rafraîchissement au prix d'une perte de précision.

Avec period task adaptative



Pour essayer de répondre à une incertitude sur la période des tâches optimales (condition différente, architecture différente,), j'ai implémenté un algorithme permettant de répondre à ce besoin sans supervision humaine.

- Dans un cas d'utilisation normale, l'algorithme va converger vers une task period stable (48 ms) en environ 8 secondes et au bout de 5000 mesures il aura un taux d'échéances manquées de 0% environ.

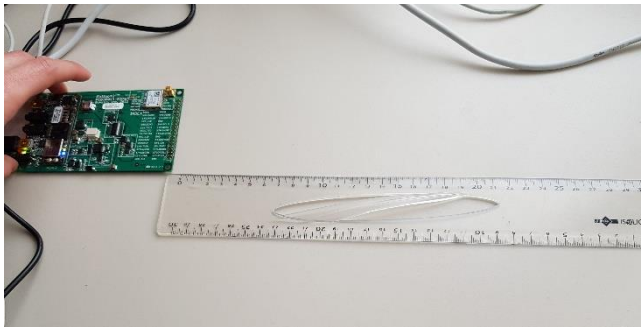
- Dans un cas d'utilisation extrême, il va converger vers une task period « stable » (90 ms) en environ 30 secondes et au bout de 5000 mesures il aura un taux d'échéances manquées de 4,4 %. Cependant, il ne faut pas oublier que les conditions du stress test sont très loin de la réalité et celle-ci ne risque pas de se produire « dans la nature ».

POSITION REELLES VS CALCULEES

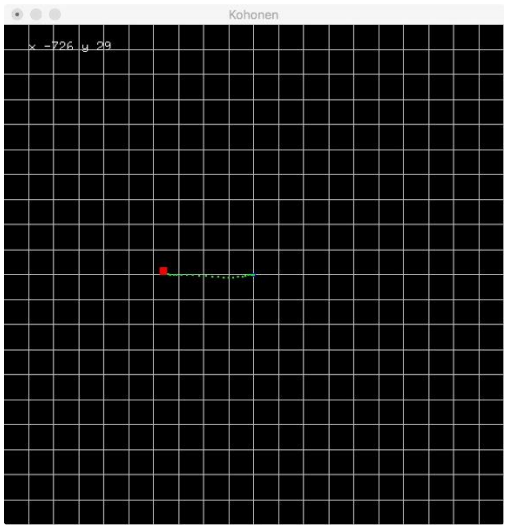
Après avoir réalisé l'ensemble de l'application il est intéressant de se rendre compte quelle type de performance est-t-on capable d'atteindre.

On s'est limité ici à réaliser le tracking sur les axes X et Y, pour ne pas à avoir à gérer le bruit généré par l'accélération terrestre (axe Z). On tentera de réaliser des translations sur un support plat tout en veillant à ne pas effectuer de rotation. Une légère rotation modifiera les valeurs à cause de l'accélération terrestre potentiellement mise en jeu.

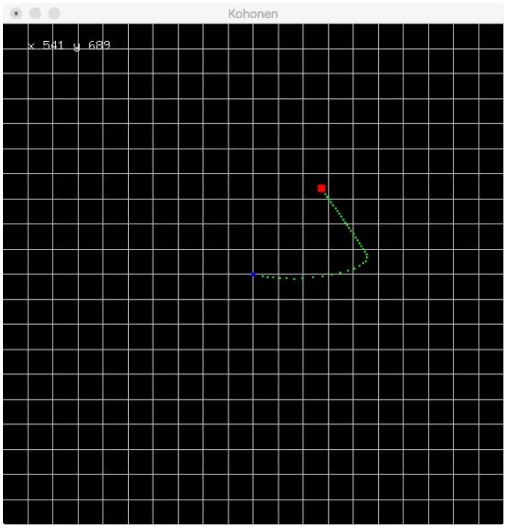
Malgré toutes les techniques mises en place et les contraintes, on arrive à observer grossièrement les mouvements, mais ceux-ci sont loin d'être précis et constants.



J'ai alors tenté d'évaluer la distance parcourue pour 30 cm dans une direction pour chacun des axes. Tout en sachant que cette technique est loin d'être parfaite : il est impossible de produire manuellement à la main un mouvement parfait, constant, sur un seul axe et sans rotation. Cependant on tentera d'en conclure empiriquement.

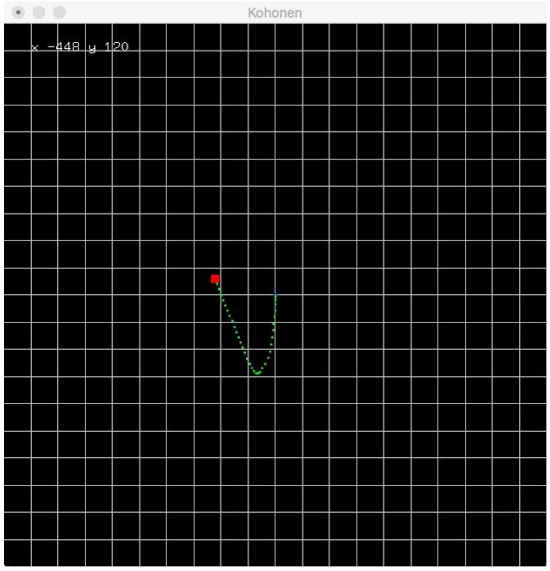


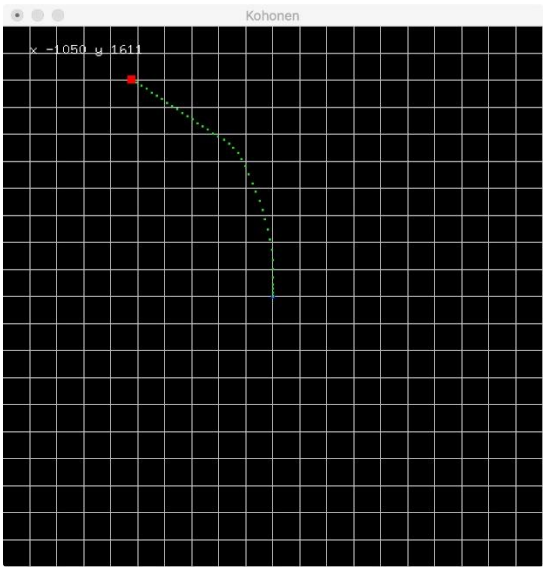
30 cm à gauche



30 cm à droite

Sur l'axe X, lorsque l'on va vers la gauche tout se passe pour le mieux, on obtient un tracé marqué. Vers la droite la direction est bien marquée, mais lors de la décélération celle-ci vient inverser la vitesse actuelle et provoque un retournement de position…





30 cm en bas *30 cm en haut*

Sur l'axe Y, ce sentiment est encore plus prononcé lorsque l'on va vers le bas. Le système a tendance à bien plus privilégier la direction haute.