Rapport de BE de C++
Réalisation d'une application sur le simulateur Arduino

Enseignant : M. MONTEIL et M. DEAU

Sophie ENGUEHARD - Zacharie HELLOUIN DE CENIVAL

Introduction :

Lors du BE de C++, nous avons choisi un projet en lien avec la médecine étant donné que Zacharie aimerait plus tard travailler dans ce secteur. Nous nous sommes en particulier intéressés aux crises d'épilepsie et aux convulsions qu'elles peuvent engendrer. L'application que nous avons décidé de mettre en place est donc un détecteur de lumière susceptibles de provoquer des crises d'épilepsies ainsi qu'un détecteur de convulsions.

Tout d'abord, il faut savoir que les fréquences de flashs qui sont susceptibles de provoquer des crises d'épilepsie sont autours de 15 à 20 clignotements par seconde. Ce sont ce genre de fréquences que nous avons donc cherché à détecter.

Concernant les convulsions, l'idée est plus générale : il s'agit de détecter les fortes perturbations (secousses) autour d'une position d'équilibre, tout en prenant en compte le fait que la position d'équilibre peut varier.

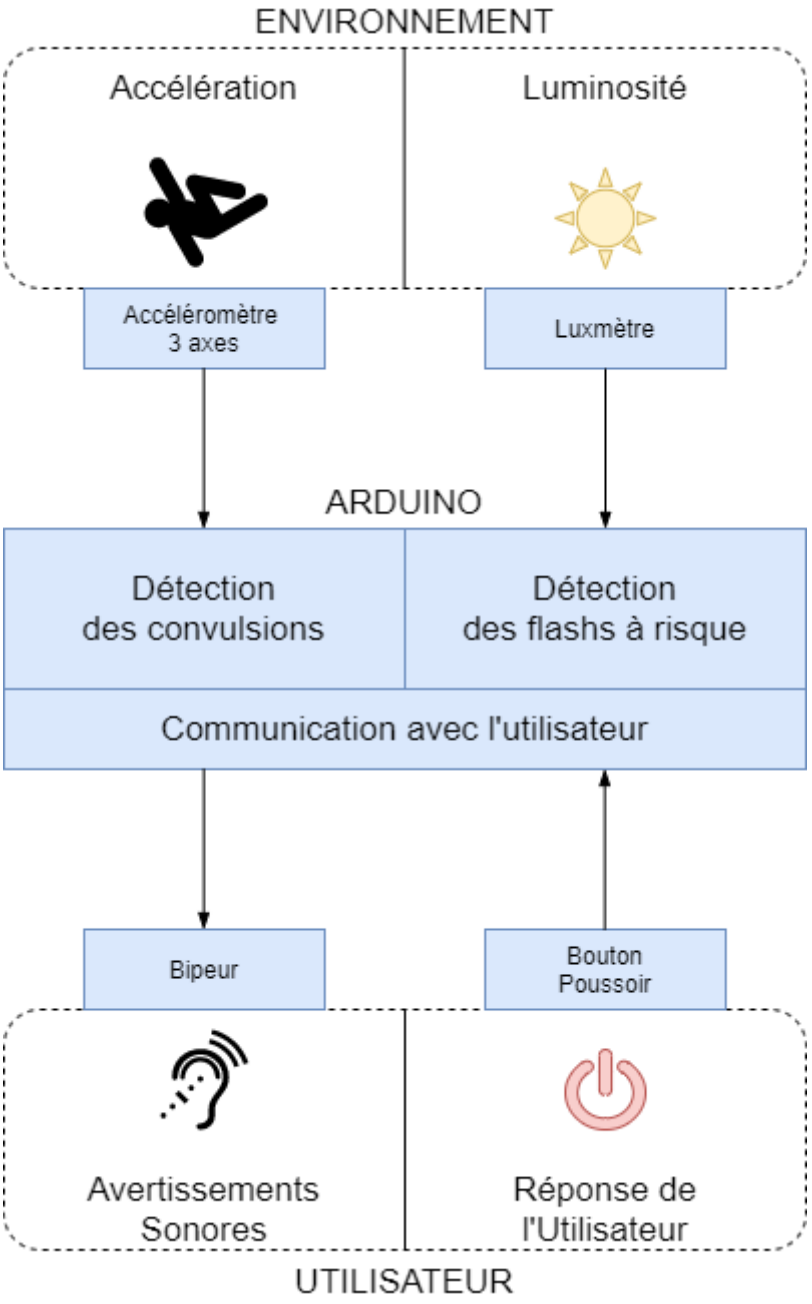
Etant donné que le travail a été réalisé dans des conditions de simulations, la sensibilité de détection et les valeurs utilisées dans le test ont été choisies arbitrairement et le code doit être adapté aux conditions de fonctionnement réelles.

En pratique, nos deux applications fonctionnent séparément. Si on détecte une lumière ou bien des convulsions alors on actionnera un bipeur qui sonnera jusqu'à appui sur un bouton poussoir.

Développement:

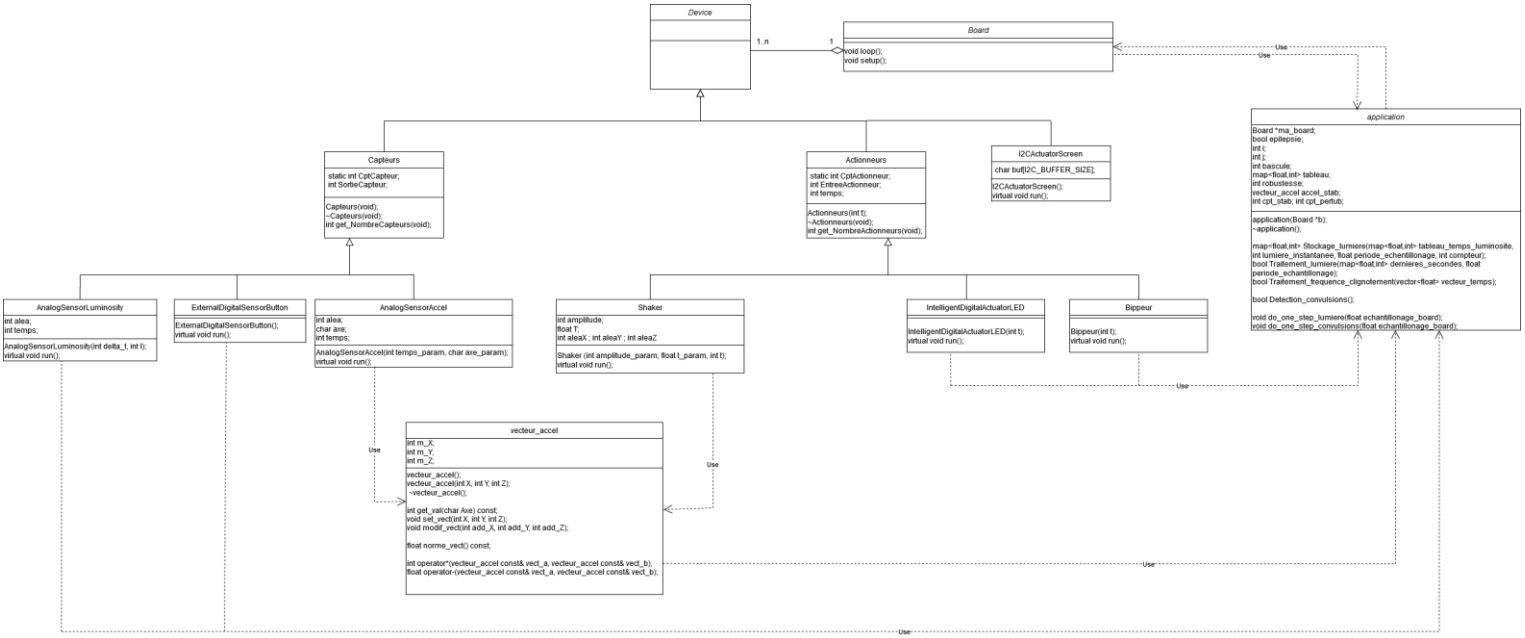
I. Diagrammes de notre application

Le schéma de fonctionnement matériel et logiciel est le suivant :



On communique principalement avec la board via deux capteurs : d'une part l'accéléromètre qui détecte des convulsions et d'autre part avec le luxmètre. On récupère en permanence les informations soit du luxmètre ou bien de l'accéléromètre et on traite les données afin de détecter respectivement des flashs ou bien des convulsions. Si un danger est détecté, on active un avertisseur sonore qui ne s'arrêtera pas tant que le bouton poussoir ne sera pas enfoncé.

Le diagramme de classes que nous avons réalisé est le suivant :



Ce diagramme de classe est affiché en plus grand en annexe (dernière page).

Nous avons décidé d'implémenter deux classes nommées "Capteurs" et "Actionneurs" qui héritent de la classe Device et qui permettent de regrouper nos capteurs d'un côté et nos actionneurs de l'autre afin de rendre leur manipulation plus aisée.

Nous avons ensuite simulé plusieurs capteurs qui étaient utiles dans notre application tel qu'un luxmètre, un bouton poussoir ou bien un accéléromètre.

Du côté des actionneurs, ceux-ci nous ont en particulier permis de modifier l'environnement et ainsi simuler les conditions de convulsion ou bien de lumière provoquant une crise d'épilepsie. Seul le bipeur était un "vrai" actionneur intégré à notre application.

La board communique avec les différents Devices et avec notre application (pour ce faire, on n'écrit qu'une seule ligne de code dans la loop : mon_application->do_one_step_lumiere(10);). Notre application utilise les relevés des capteurs transmis sur chaque pin et met en marche les actionneurs associés.

II. Implémentation de nos deux parties

Les deux parties ont été codées séparément. Nous nous intéresserons donc ici à chacune d'entre elle indépendamment de l'autre.

1) Codage de la lumière

Pour la détection de la lumière, nous utilisons une map (de la STL) permettant d'associer les valeurs de la lumière avec le temps associé. On stocke ainsi les trois dernières secondes dans la fonction Stockage_lumiere. Tant que l'application ne tourne pas pendant trois secondes et donc que le nombre de valeurs dans la map n'est pas stable, on lève une exception et on ne fait pas appel au reste du programme.

Une fois la map remplie, on va traiter ces informations dans une seconde fonction nommée Traitement_lumiere. Pour ce faire, on détecte les fronts présents dans la map et on stocke dans un vecteur (vector de la STL) le temps associé à chaque front. Cela va nous permettre d'analyser dans un troisième temps le nombre de front contenu dans le vecteur et d'analyser les variations de temps entre les différentes prises de valeurs.

La fonction Traitement_frequence_clignotement renvoit un booleen à la fonction Traitement_lumiere, qui retournera la même valeur. Ce booléen indiquera si oui ou non la lumière est susceptible de provoquer une crise d'épilepsie. Si le booléen est à un alors il y a un danger et on active le bipeur jusqu'à appui sur le bouton poussoir.

Les problèmes que nous avons perçu lors du codage de ces quelques fonctions est que les sleeps n'étaient pas forcément toujours exacts : entre deux compilations, nous obtenions des nombres de clignotements différents sans changer les caractéristiques de la LED intelligente. Nous pensons que le problème vient probablement du multithread. Etant donné que ce n'est pas du "vrai" multithread, le système d'exploitation alloue un certain temps à chaque thread. Et s'il y a beaucoup de thread, il est possible que chaque thread ait "peu" de temps disponible. Donc le sleep durera plus longtemps.

Ceci a impliqué plusieurs problèmes : d'une part on ne détecte pas toujours le même nombre de front et alors on ne peut pas être précis lorsque l'on détecte la lumière. De plus, dans ma fonction Traitement_frequence_clignotement on souhaiterai analyser les variations de temps entre les différentes prises de valeurs mais cela est impossible parce que ceux-ci ne reflètent pas la réalité.

2) Détection des convulsions

En ce qui concerne la détection de convulsions, nous avons d'abord mis en place une classe appelée vecteur_accel qui offre un format plus pratique pour manipuler les valeurs d'accélération (vecteurs de taille 3). A cette classe sont associées plusieurs opérations et méthodes, notamment l'opération " - " qui retourne l'angle entre deux vecteurs et la méthode norme_vect qui retourne la norme d'un vecteur donné.

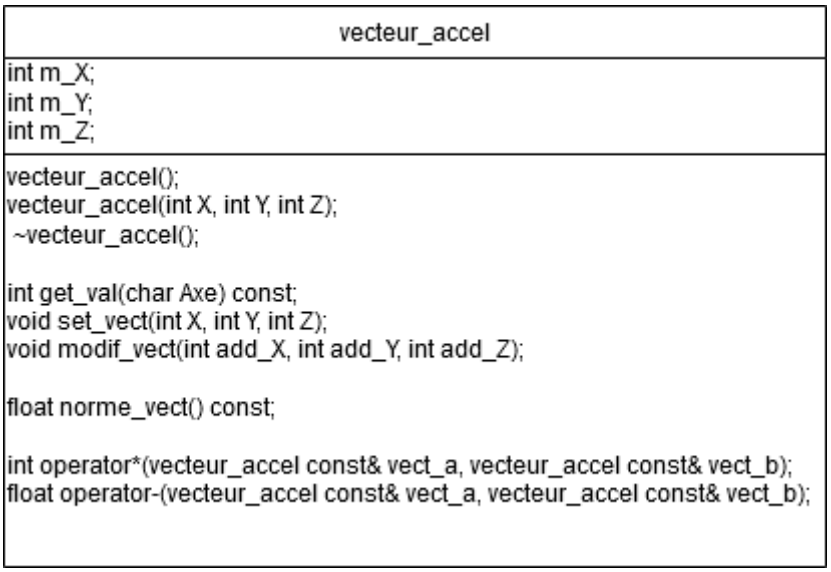


Figure 1 : Classe vecteur_accel

Trois variables sont de ce type.

Dans un premier temps, une variable globale de type vecteur_accel appelée accel_env permet de stocker l'état de l'accélération du système. Elle est modifiée par un actionneur appelé Shaker. Le Shaker n'est pas un actionneur "réel" mais sert à modifier l'environnement de manière aléatoire afin de tester notre application (on peut assimiler ce shaker à un banc de test qui secouerait notre accéléromètre pour simuler des convulsions).

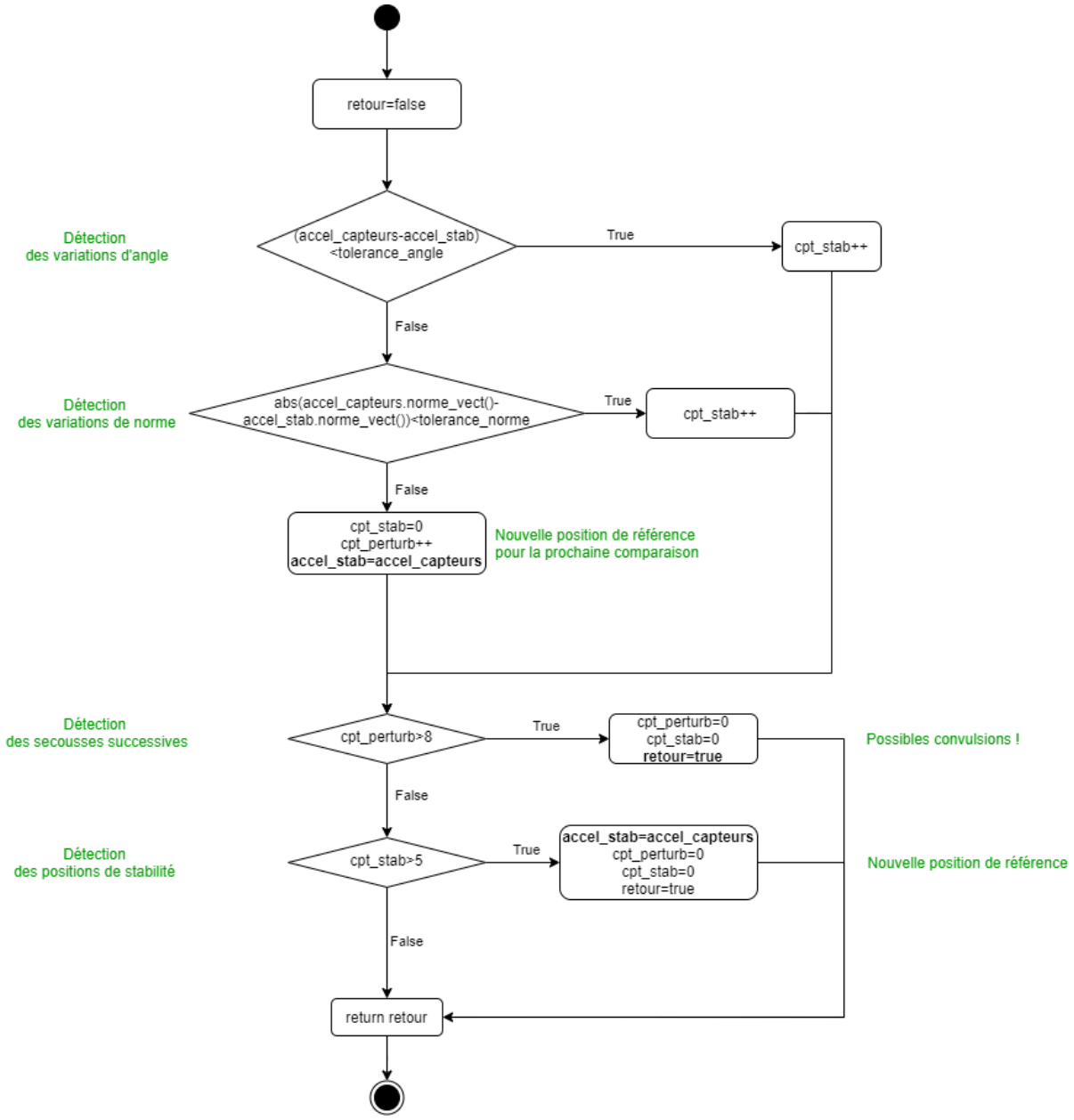
Dans un second temps, un attribut de notre classe "application" appelé accel_stab permet de stocker la dernière position d'équilibre détectée par la méthode application::Detection_convulsions().

Enfin, au sein de la méthode Detection_convulsions, la variable locale accel_capteurs du type vecteur_accel stocke la valeur retournée par l'accéléromètre. L'accéléromètre est en fait un ensemble de trois capteurs de la classe AnalogSensorAccel : Accel_X, Accel_Y et Accel_Z. Chacun de ces capteurs écrit sur un pin la valeur de l'accélération de l'environnement selon un axe (X, Y ou Z).

A chaque appel de la méthode Detection_convulsion, le relevé des valeurs de l'accéléromètre s'effectue comme suit :

```
//Récupération du vecteur d'environnement capté
vecteur_accel accel_capteurs(ma_board->analogRead(7),
                             ma_board->analogRead(8),
                             ma_board->analogRead(9));
```

Ce relevé est ensuite comparé à la valeur de référence évoquée plus haut, accel_stab. Le principe détaillé de cette comparaison permettant la détection est expliqué par le schéma suivant :



Cette méthode est appelée par une autre méthode de la classe application; application::do_one_step_convulsions(float echantillonage_board) qui prend en paramètre la fréquence à laquelle on effectuera la détection. Il est à noter qu'il n'est pas utile de choisir une fréquence supérieure à l'échantillonnage des capteurs.

Lorsque des convulsions sont détectées, do_one_step_convulsions permet l'émission d'un son jusqu'à ce que l'utilisateur appuie sur un bouton.

Les valeurs de comparaison pour les compteurs (8 pour cpt_perturb et 5 pour cpt_stab) ainsi que les valeurs de tolerance_angle et tolerance_norme sont à adapter aux conditions réelles après tests. Actuellement, la détection fonctionne bien pour une amplitude de Shaker entre 1 et 3 (à 1, on ne détecte pas de convulsions, à 3 on détecte des convulsions).

Conclusion

Nous avons réussi à traiter la problématique jusqu'à un certain point. En effet, la détection de lumière et de convulsion se font mais pas en même temps à cause de problèmes de synchronisation.

Pour la partie détection de lumière, comme indiqué plus haut, le problème a été d'ordre temporel étant donné que les différents sleeps ne renvoyaient pas toujours les bonnes valeurs. De plus, les valeurs que nous renvoyaient nos capteurs simulés étaient toujours très « lisses » et dessinaient de beaux pics que nous n'aurions jamais eu en réel. Ce cas est donc un cas d'étude.

Pour la partie détection de convulsions, une des difficultés rencontrées a été les possibilités de transmission de données par les pins. En effet, on ne pouvait pas transmettre une valeur d'accélération négative par le biais d'un pin. Un offset a donc été ajouté afin de contourner ce problème.

Enfin, la première perspective d'évolution est d'implémenter le programme sur une carte Arduino réelle et de faire fonctionner les deux applications en même temps. Dans un second temps, on pourrait imaginer qu'au lieu d'actionner un bipeur si des convulsions sont détectées, on pourrait directement appeler les secours ou prévenir un proche en communiquant avec un téléphone par bluetooth par exemple.

ANNEXE : diagramme de classes