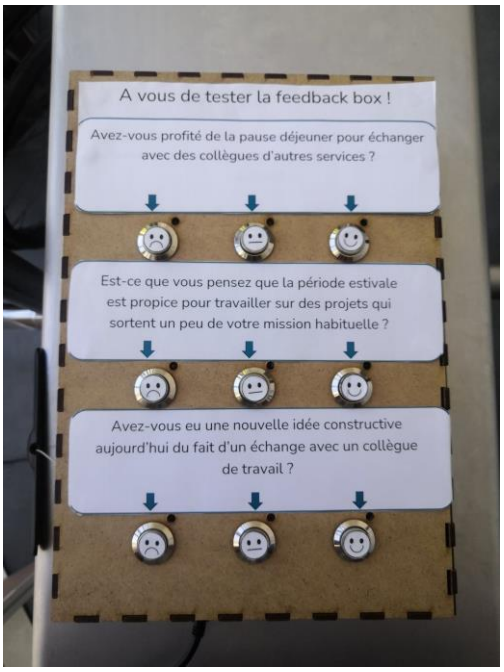
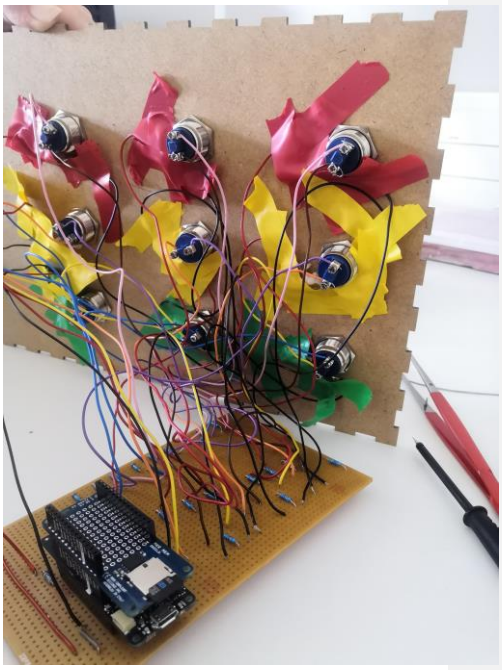




Documentation Technique

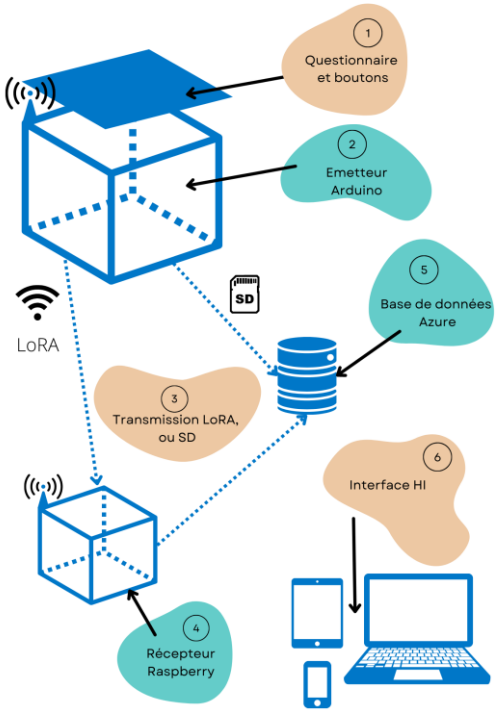
Preuve de Concept : Feedback Box

Documentation Technique d'un projet soutenu par le partenariat entre Handicap International (HI) et l'INSA de Lyon. L'objectif de ce projet est de produire une preuve de concept d'un boîtier low-tech permettant de récupérer l'avis des bénéficiaires de Humanity & Inclusion (HI). Ce boîtier fonctionne à l'instar d'un formulaire semi-électronique, où les réponses aux questions sont récupérées et envoyées électroniquement, mais l'affichage ressemble à du papier.

Table des matières

Introduction

Ce schéma montre les différentes fonctionnalités depuis un boîtier et jusqu'au récepteur, globalement notre plan va détailler ces parties.



Petit contexte de comment cette boîte serait utilisée, en espérant que cela puisse vous aider à imaginer le fonctionnement. Ne pas hésiter à relire ce paragraphe une fois toute la documentation consultée. 😉

Une personne du terrain a plusieurs boîtiers à sa disposition (on va dire 3) ainsi qu'un raspberry, un écran, un clavier et une souris. Elle a pour consigne de la part de l'équipe MEAL (ou autre) d'évaluer la satisfaction des bénéficiaires dans un centre de réadaptation. Elle doit alors disposer les boîtiers à plusieurs endroits différents afin de poser des questions différentes : par exemple, 1 à l'accueil, 1 dans la salle d'attente et 1 à la sortie du soin. Les boitiers doivent être disposés à des endroits assez isolés afin de respecter une certaine confidentialité (forcément relative). Cette personne doit, avant d'installer les boîtiers, les paramétrer et ajouter l'interface. Pour cela, elle utilise l'application de paramétrage et choisit ses paramètres, elle choisit aussi les questionnaires que l'équipe MEAL a composés : 1 différent pour chacun des trois cas.

Lorsque les boîtiers sont paramétrés, il y a une fonctionnalité qui permet d'imprimer les questionnaires associés. La personne s'occupe de "coller" les questions sur les boîtiers. Après cela, les boîtes sont prêtes, elle peut les installer : il leur faut une alimentation ou alors une batterie chargée pour les alimenter. Maintenant il faut s'assurer que le récepteur fonctionne et le mettre en place. Il faut allumer la Raspberry avec l'écran, clavier et souris qui est déjà prêt à l'utilisation et la déverrouiller à l'aide d'un MDP connu des équipes. Enfin, il faut s'assurer qu'il y a bien une connexion Internet, et enfin lancer un script Bash qui permettrait de lancer toutes les requêtes possibles, ou alors se connecter à IoTHub (cela dépend de la méthode utilisée finalement). Cette partie-là est à réfléchir pour rendre cette action simple et efficace pour les équipes.

Hardware / Emetteur

Voir fiche Emetteur Arduino (voir Dossier)

Boîtier

La structure de la boîte est une boîte à créneau constituée de bois contreplaqué de 3mm d'épaisseur. A cette boîte nous avons ajouté 9 perçages entier pour les boutons, et 9 perçages pour y glisser des leds. De plus, sur la partie inférieure nous avons découpé un rectangle afin d'avoir accès au connecteur mini-usb B de l'arduino, et de pouvoir brancher l'arduino au secteur une fois celle-ci fixée dans la boîte.

Sur le côté inférieur, nous avons également percé un petit trou afin de laisser sortir l'antenne LoRA de la boîte. (Est-ce vraiment utile de sortir l'antenne Lora ? Par rapport à l'étanchéité ?)

Le modèle de cette boîte avec les perçages est disponible (sur github). Pour l'instant il y a simplement les plans dft (prêts à être découpés par une découpe laser), cependant nous allons essayer de trouver les plans solidedge pour que cela soit plus facile à modifier.

Réflexion sur la boîte :

- Comment on fixe les leds de manière plus propre et efficace ?
- Quel matériau est le plus pertinent pour la boîte ? Plastique, impression 3D, bois, carton ? Matériau recyclé ou non ?
- Combien de questions et de réponses par questions ? A priori, 4 réponses par questions seraient davantage cohérentes.

La conception du boîtier pourrait être réfléchie par des étudiants en Mécanique, qui ont plus de connaissances en conception d'objets mais aussi en matériaux.

"Petit" matériel

Le matériel suivant décrit le matériel nécessaire pour un prototype composé de 3 questions, avec 3 choix par questions (🙂, 😐, ☹).

- 9 Boutons

Les boutons utilisés pour le prototype étaient les suivants : boutons amazon

Ils étaient pratiques du fait qu'ils pouvaient se fixer via un système de visserie. Afin de réduire le coût il pourrait être intéressant de trouver une manière de concevoir ou d'acheter des boutons moins chers. Idée : concevoir des boutons avec des bouchons de bouteille plastique en guise de poussoir, et du cuivre pour l'interrupteur.

- 9 Leds

Pour les leds, nous avions utilisé 3 leds rouges, 3 leds oranges et 3 leds vertes. Ces couleurs correspondaient aux smileys de réponses (vert pour le plus satisfait, orange pour le neutre et rouge pour le moins satisfait). Est-ce nécessaire d'utiliser 3 couleurs différentes ? Est-ce que ces couleurs sont pertinentes dans toutes les cultures/pays ?

- 9x 220 Ω Résistance (pour les leds)
- 3x 1K Ω Résistance (pour les boutons)
- 6x 100K Ω Résistance (pour les boutons)
- 3x 1M Ω Résistance (pour les boutons)

Pour les boutons, nous avons utilisé des ponts diviseurs de tension afin de brancher 3 boutons sur une même pin analogique. Ces résistances permettent de faire varier la tension en fonction du bouton appuyé. A termes, il serait sûrement plus efficace d'utiliser un registre à décalage afin d'étendre le nombre de pins pour les boutons, et pour les leds également. (exemple de branchement avec des leds pour un registre à décalage 74hc595)

Problème rencontré : le pont diviseur de tension génère des problèmes lors de changement de source de tension (voir partie source d'alimentation)

Arduino MKR WAN 1310

Le contexte de la feedback box est soumis à des contraintes de réseau très importantes. Il fallait donc imaginer une manière de transmettre les données récoltées par notre boîte afin d'accéder à un récepteur connecté au réseau Internet.

La technologie que nous avons décidé de mettre en œuvre est la technologie LoRA puisqu'elle permet une communication longue distance, très peu protocolaire pour l'instant donc assez facile d'accès. La technologie LoRA correspond à la couche physique, LoRAWAN en revanche représente le protocole qui est encapsulé sur la couche physique LoRA. Pour plus d'informations sur la différence entre LoRAWAN et LoRA, il y a ce site.

Afin d'utiliser cette connectivité LoRA, nous avons cherché une arduino qui le permettait. Des professeurs de l'INSA (Walid Bechkit et Oana Iova) nous ont dirigées vers l'Arduino MKR WAN 1310, qui permet d'utiliser la technologie LoRA tout en conservant une faible consommation énergétique. Cette arduino se connecte à une antenne LoRA, comme celle-ci (Dipole Pentaband Waterproof Antenna).

La MKR WAN 1310 représente un coût non-négligeable (autour de 40€). Dans l'optique de réduire les coûts d'un boîtier afin d'être plus compétitif et pertinent, il se pourrait que d'autres matériels soient plus adaptés.

En effet, l'application que nous en faisons ne nécessite pas un processeur ultra-puissant, l'arduino pourrait facilement être remplacée par une arduino moins cher, moins performante. Par exemple, l'arduino Nano pourrait être une première piste d'efficience prix/performance. La contrainte du LoRA pourrait être contournée grâce à des modules LoRA (i.e. ce module).

Des exemples d'utilisations de ces deux hardwares combinés peuvent être trouvés ici.

Une manière de réduire encore plus le coût de matériaux et d'optimiser le boîtier à nos besoins est de réfléchir à tous les composants individuels nécessaires (processeurs, module LoRA ...) et de les souder. Des étudiant.e.s en génie électrique pourrait avoir davantage de recul sur ce matériel.

Module carte SD

En plus de la connectivité LoRA, nous avons décidé d'insérer un module carte SD. En effet, dans certaines zones du monde il paraît compliqué de déployer un récepteur LoRA suffisamment proche de l'émetteur, et qui a une connexion internet. Ainsi, l'idée serait de récupérer les données sur une carte SD. Il est aussi possible d'utiliser la carte SD comme backup du lien LoRA et de push dans la BDD dans le cas où ceux-ci ne sont pas fiables. Ainsi, les données pourraient être téléchargées en asynchrone sur la base de données, de manière régulière.

Lors de ce téléchargement, il faudrait bien faire attention à ne pas rentrer des informations redondantes dans la base de données (ne pas faire le doublon entre données automatiquement envoyées sur la BDD et données stockées dans la carte SD).

Source d'alimentation

L'émetteur est censé être autonome donc ne pas être branché à un ordinateur en permanence. Nous avions acheté une batterie, qui finalement n'avait pas le bon embout pour être branchée sur l'arduino. Il faudrait soit avoir un adaptateur pour cette batterie, soit prendre une batterie avec l'embout adapté (JST PH d'après cette doc).

Sinon, pour les tests, nous avions utilisé des adaptateurs de téléphone avec des câbles micro-usb. Cependant, nous nous sommes rendu compte qu'avec les adaptateurs de téléphone la tension en entrée de l'arduino variait et pouvait avoir un

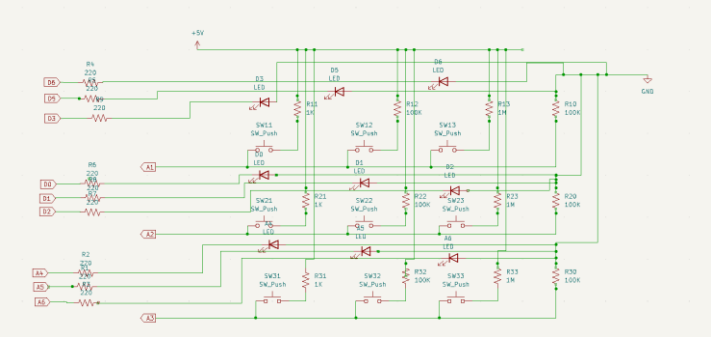
impact sur la perception des boutons. En effet, avec les ponts diviseurs de tensions nous avons trouvé les seuils pour différencier les 3 boutons d'une même question de manière expérimentale, sans regarder vraiment le circuit électrique. Avec un écart de tension en entrée les seuils n'étaient plus les mêmes.

Ainsi, il faudrait trouver une manière de créer ces seuils en fonction de la tension qui arrive en entrée. Cela permettrait d'être plus précis et plus robuste face aux différentes sources d'alimentation.

Schéma électrique

Du fait du nombre limité d'entrées/sorties disponibles sur l'arduino nous avons décidé d'utiliser une broche par question. En d'autres termes, une broche analogique reçoit les informations de trois boutons. En revanche, il y a une broche par led.

Il pourrait être également intéressant d'augmenter la capacité d'entrées sorties grâce à un registre à décalage, exemple : 74HC595.



Robustesse

Dans un premier temps, nous avons fait les tests sur une breadboard, où nous avons remarqué beaucoup d'interférences entre les fils (lorsque l'on bougeait les fils une led s'allumait, ou bien l'arduino comprenait qu'un bouton avait été enclenché...).

Ainsi, nous avons décidé d'investir dans une plaque à trous afin de souder les boutons, fils, leds et que cela soit plus résistant. En allant au FimiTech (fablab de l'INSA), on nous a conseillé une plaque à bandes (les trous d'une même bande sont déjà reliés entre eux) qui étaient plus pertinentes pour l'arduino.

Afin de souder l'arduino sur la plaque à bandes il nous fallait des connecteurs broches femelles/mâles.

Cependant, notre prototype final souffre toujours d'interférences avec des boutons qui s'appuient (envoi de l'information de boutons appuyés, led allumé...) lorsqu'ils ne le sont pas. Nous ne savons pas d'où cela peut venir.

Une manière de rendre les connexions encore plus robustes serait de faire un PCB (*Printed Circuit Board* – Circuit Imprimé) à partir du schéma électrique.

Questions

Affichage des questions

Notre prototype affiche ses questions à l'aide d'une simple feuille imprimée puis fixée grâce à des pastilles adhésives.

Cette solution n'est pas robuste, ni étanche, il faudrait imaginer une autre méthode de fixation.

- Fixer des pochettes plastiques afin d'insérer les questions à l'intérieur
- Plaque de plexiglass (idée écartée parce que très peu écologique)
- Ecran LED (idée écartée car coûteuse en énergie, matériaux, très peu low-tech)

Contenu des questions

Le contenu des questions en termes de satisfaction est à établir par une personne du MEAL. Nous avons également été en contact avec Didier Demey (OP Réadaptation) qui essaie de faire un questionnaire court (4 questions avec 4 réponses) pour les centres de réadaptation fonctionnelle. Ce modèle pourrait être intéressant à étudier et approfondir. Autre contact intéressant pour le contenu des questions : Laura Mosberg (équipe MEAL de la Direction 3I).

Questionnaire

On préfère parler de questionnaire, car c'est le set de questions qui composent le questionnaire qui est important. D'après nos observations du fonctionnement de HI, les personnes qui émettent les questionnaires ne sont pas forcément celles qui vont les mettre en place sur le boîtier. Il y a donc des questionnaires "types" qui sont créés par des entités spécialisées (comme les MEAL par exemple) et ces questionnaires sont disponibles comme paramétrage sur la boîte.

Fonctionnalités à améliorer

Afin d'améliorer l'inclusivité du boîtier, on pourrait développer certaines fonctionnalités :

- Enceinte/Haut-parleur dans le but de rendre les questions accessibles aux personnes mal-voyantes
- Microphone dans le but de récolter un avis davantage construit des personnes exprimant leur satisfaction. Ce retour audio serait stocké dans la carte SD. Il soulève le problème de comment analyser ces données ? De l'énergie déployée pour le retour récupéré ? De la confidentialité de parler au milieu du centre de réadaptation par exemple ?

- Imprimer les questions également en braille. Est-ce qu'il y a des imprimantes en braille dans beaucoup de pays ? Y'a-t-il une utilisation courante du braille dans les pays d'intervention ?

Environnement Microsoft

HI travaille dans un environnement Microsoft, cela doit être pris en compte dans le développement de ce projet car sinon il y a des risques de ne jamais pouvoir déployer cet outil.

Base de données

Nous traitons la base de données dans ce paragraphe car nous avons utilisé une BDD Azure qui est un cloud Microsoft. Cette utilisation a énormément impacté notre manière de travailler avec la base de données donc nous avons décidé de l'expliquer ici.

Dans la manière de récupérer nos données, nous avons fait le choix de décorréler nos informations : c'est à dire que chaque question est considérée comme individuelle et n'est pas liée aux autres questions. Nous ne faisons pas de profil type de l'utilisateur.

Choix de la base de données

Nous avons besoin de plusieurs informations :

- Une information est enregistrée dès qu'un bouton du boîtier est pressé, elle est identifiée par un ID unique, cette info contient quel bouton a été appuyé (content, mitigé, mécontent) et à quelle question ce bouton correspondait (1, 2 ou 3), et l'ID correspondant au paramétrage du boîtier qui récupère puis envoie l'information. Dans le diagramme de la base de données nous avons également prévu des champs pour l'heure et la date de chaque mesure.
- Chaque information est associée à un paramétrage de boîtier car chaque boîtier a lui aussi ses paramètres. Un paramétrage est lui aussi identifié par un ID, on peut alors lui associer : l'ID du questionnaire qui lui est attribué, sa localisation de mise en place (pays, projet), son secteur qui permet d'identifier dans quelle structure le boîtier est mis en place (réad, distribution, ...) et enfin une indication sur l'étape du service dans laquelle est située la boîte (accueil, attente, après service).
- Enfin on identifie le questionnaire avec un ID ainsi que ses questions.

+ 1)

Déploiement de la base de données

Afin de stocker les données, il est nécessaire d'avoir un hardware de stockage. Un accès simple pouvait être le serveur local du siège de HI.

Cependant, il est très rempli et l'idée est désormais de déployer ailleurs les nouvelles données. Nous avons donc utilisé une base de données Azure SQL Database. Azure est un service Microsoft qui permet entre autres d'héberger les données de ses clients sous forme de cloud limité en espace. HI utilise essentiellement les services Microsoft donc pour une meilleure interopérabilité avec l'environnement HI, Azure est pertinent.

Nous avons utilisé une BDD configurée par HI et qui nous permet d'avoir un espace de test dans l'espace Azure de HI.

Nous avons aussi pour nos essais, ouvert des versions d'essai Azure qui fonctionnent 1 mois gratuitement. Il faut savoir que Microsoft a énormément d'extensions ou de logiciels afin d'avoir des interfaces graphiques exploitables par un maximum de personnes.

Fiche : se connecter à la base de données (voir dans ce dossier)

Pour se connecter à notre base de données il faut des identifiants et MdP qu'il faut soit demander à la DSI de HI Lyon, soit demander au manager 3I ou à l'équipe Innovation de HI Lyon.

Software / Emetteur

Code Arduino

Afin de faire fonctionner l'émetteur nous avons codé un programme arduino qui permet de gérer la gestion des boutons, des leds et de l'envoi LoRA. Ce code est disponible sur GitHub.

Le code initialise d'abord toutes les pins leds, digitales utiles, ainsi que les modules SD et LoRA. Le module LoRA est initialisé à la fréquence 868,1 MHz afin de correspondre au récepteur Raspberry. En effet, cela ne fonctionnait pas de modifier la fréquence sur le module LoRA Raspberry qui fonctionnait à 868,1 MHz. On pouvait le modifier sur l'arduino. Cependant, d'après la documentation de la MKR WAN 1310 ses fréquences de fonctionnement sont 433,868,915 MHz. Il est possible qu'on ait perdu de la puissance et de la distance parce que l'arduino n'était pas sur sa fréquence de fonctionnement parfaitement.

Ensuite, en boucle il lit la valeur de la tension aux bornes des broches analogiques des questions.

Pour chaque question, selon cette valeur de tension, il sait quel bouton précisément a été appuyé. Toujours pareil il sait selon des seuils qui sont expérimentaux et qui mériteraient d'être plus précis selon la tension d'entrée.

Une fois qu'un bouton a été appuyé il faut éviter les rebonds du bouton. En effet, les boutons ne libèrent pas un signal franc, on observe des petites oscillations qui vont

faire croire que le bouton a été appuyé plusieurs fois au lieu d'une seule. Ainsi, lorsqu'un bouton a été détecté on attend qu'on détecte le même bouton 3 fois avant de décider que ce bouton a été appuyé.

Une fois qu'on est sûr qu'il a été appuyé, il faut créer le paquet à stocker et à envoyer. Le paquet a cette forme pour l'instant : "Paramètre_boîte_ID, Numéro_Question, Réponse_Question, Numéro_Séquence". Ensuite ce paquet est envoyé via le module LoRA, et est stocké sur la carte SD. A terme, le paquet contiendra également l'heure et la date d'appui afin de stocker des données plus précises.

Ensuite, on allume la bonne led en fonction du bouton qui a été appuyé.

Les leds sont primordiales pour nous car elles indiquent à l'utilisateur que le bouton a été correctement appuyé. Cela évite les doublons de la même réponse par peur que cela n'ait pas fonctionné.

PowerApps

L'application pour l'émetteur est une application permettant de paramétrer la feedback box pendant le déploiement.

Afin de respecter au mieux l'utilisation Microsoft que fait HI, nous avons décidé de la faire sur PowerApps (fonctionnalité Microsoft). En effet, plusieurs personnes savent comment développer une application PowerApps. De plus, PowerApps permet une intégration facilitée à Teams (utilisé par tout le monde à HI).

PowerApps est destiné à être très instinctive et compréhensible facilement. Cependant, force de facilité, elle est également difficilement modulable en codant, avec des erreurs très imprécises... Nous avons donc eu du mal à développer entièrement cette application, dont voici l'ébauche.

Paramétrage de la boîte

Le paramétrage de la boîte correspond principalement à remplir la table de données qui a une ligne unique par paramètre de la boîte. Les éléments à rentrer sont donc :

- L'ID de la boîte

Chaque boîte aura collé/gravé/marqué son numéro d'identifiant (qui se rapproche d'un numéro de série). Ainsi, nous saurons identifier chaque boîté précisément

- L'ID du questionnaire

Il faut renseigner le questionnaire sélectionné qui va être apposé sur la boîte. Cela permettra de lier les réponses reçues avec les questions demandées.

- Le pays

Lire l'ensemble des pays d'intervention de HI sur la base de données de HI

- Le lieu plus précisément

Mettre une carte afin de pointer un lieu plus précis dans le pays

- L'endroit dans un bâtiment

Si la boîte est mobile, ou à l'accueil du bâtiment, dans une file d'attente, après le service…

- L'ID du projet
- Le secteur d'intervention

Il s'agit de dire que cette boîte est utilisée pour récupérer les avis de quel secteur d'intervention :

  - Aide d'urgence aux populations victimes de crises et de catastrophes naturelles
  - Prévention des handicaps et des maladies invalidantes
  - Appareillage et rééducation des personnes handicapées
  - Insertion scolaire, sociale et économique
  - Action contre les mines, les restes explosifs de guerre et l'utilisation des armes en zones peuplées
  - Promotion des droits des personnes handicapées
  - Autres …

Une fois ce formulaire rempli, il faut envoyer ce formulaire dans la base de données qui crée alors un identifiant unique qui correspond au paramétrage de la boîte à ce moment précis. Il s'agit de l'identifiant de paramètre.

D'autre part, il faut que la boîte soit au courant de "qui" elle est afin d'envoyer cette donnée à l'émetteur. En effet, la deuxième partie est d'associer les données à la bonne boîte, au bon endroit, avec le bon numéro de questionnaire. Pour rappel la donnée est vouée à être envoyée comme "numéro_de_box, numéro_de_question, réponse_de_question, numéro_de_séquence, heure, date".

Nous avons identifié deux manières pour que la boîte envoie le bon numéro :

- Soit l'application PowerApps récupère l'ID du paramètre (l'identifiant unique de paramétrage) puis modifie le code de l'arduino avec le bon numéro. Ensuite, il s'agit de compiler à nouveau et de téléverser dans l'arduino. C'est la première méthode auquel nous avions pensé, cependant le langage arduino est assez particulier et il faut beaucoup de lignes pour compiler le code. Une autre méthode serait d'utiliser Arduino CLI, qui permet de compiler le code via une interface de terminal de commande. Nous avons posé cette question sur le forum arduino, voici les réponses obtenues.
- Une autre méthode serait qu'en "sortie d'usine", l'arduino ait déjà le code correspondant à son identifiant unique (identifiant de la boîte et non du paramètre). Ainsi, en ajoutant la date et l'heure à l'identifiant de paramètre, il suffira de mettre en regard la date et l'heure des données avec celles des paramètres pour savoir quel paramètre cette boîte a, au moment de ces données.

Enfin, une fois que le formulaire est rempli il faut permettre à l'utilisateur.rice d'imprimer la feuille de questionnaire afin de l'apposer sur la feedback box.

Récupération données carte SD

Une autre facette de cette application permettrait de récupérer les données à partir de la carte SD. Nous n'avons pas du tout eu le temps de nous pencher sur cette fonctionnalité. L'idée serait de brancher l'arduino à un ordinateur, puis de l'application pouvoir extraire les données de la carte SD et de les pousser sur la base de données (en faisant attention à la redondance bien sûr).

Lien PowerApps – Base de données

Le lien entre l'application PowerApps et la base de données n'a pas encore été implémenté. Pour cela il suffit de faire un flux via Power Automate qui permet de pousser les données.

Il est possible que ce flux soit payant (PowerApps – Base de données Azure), donc il faudrait peut-être utiliser l'intermédiaire des Listes SharePoint. Ainsi, il faudrait deux flux : l'un pour pousser les données dans la liste SharePoint, et l'autre pour récupérer les données de SharePoint et les pousser dans la base de données Azure. Voir avec JB Gayet. Attention avec les listes SharePoint, elles ont une limite de lignes à mettre dans la liste, dans le cas où l'on passe par une liste SharePoint il faut donc prévoir un moyen de libérer les lignes une fois qu'elles ont été mises en ligne sur la BDD.

Traitement des données

L'objectif de cette feedback box est de faire des statistiques et d'utiliser ces données. Nous imaginions pouvoir créer des dashboards PowerBI qui pourraient soit être inclus dans notre application PowerApps mais il y aurait moins de personnes qui iraient voir, soit sur P-Square, ou sur HInside. Ce serait une partie importante à creuser pour pouvoir utiliser les données créées. Pour cela, il faudrait beaucoup être en lien avec l'équipe data (3I).

Hardware / Récepteur

Nous développons un système de communication MISO (Multiple Input Single Output) : notre récepteur doit pouvoir recevoir les informations issues de plusieurs boîtiers différents. Dans notre cas, nous avons seulement essayé avec un boîtier émetteur mais l'idée est de le décliner pour plusieurs émetteurs (plusieurs boîtiers). Cette architecture est illustrée dans le schéma suivant :

Un récepteur, qu'est-ce que c'est ?

Dans notre cas, notre récepteur correspond au point central dans le cas où l'on utiliserait une technologie de communication sans fil. C'est l'équipement qui reçoit toutes les informations des boîtiers "émetteurs". Les boîtiers qui récoltent les données émises par l'utilisateur (ici via notre interface avec les boutons) envoient ensuite ces données à notre récepteur à l'aide d'une technologie de communication sans fil, par exemple la technologie LoRA.

Arduino VS Raspberry

Dans un premier temps, nous voulions tester seulement le lien LoRA entre émetteur et récepteur. Pour cela, nous avons d'abord travaillé avec un récepteur Arduino MKR 1310 ; par la suite nous avons réfléchi à utiliser un Raspberry Pi.

Arduino

Un Arduino MKR1310 a besoin de plusieurs choses pour fonctionner comme récepteur :

- Une alimentation électrique en permanence (batterie ou secteur),
- "Téléverser" notre code (on envoie un paramétrage à l'arduino qui lui indique quoi faire des informations physiques qu'elle reçoit),
- Une carte SD et un module carte SD si l'on veut stocker des données localement,
- Un lien avec un lieu de stockage des données (SharePoint, Excel,...) dans le cas où nous n'avons pas de carte SD,
- Un lien avec Internet afin de pouvoir transmettre les données reçues en direct dans la base de données si on veut les rajouter en temps réel,
- Une antenne LoRA à fixer sur l'arduino.

Le second point nécessite un ordinateur connecté à l'arduino de manière ponctuelle, le quatrième point nécessite une arduino avec une connexion Internet qui permettrait de pousser ensuite dans la base de données. Cependant, la contrainte que nous avons retenue est que l'émetteur n'a pas forcément de connexion Internet là où il sera placé.

Commenté [CB1]: ou le quatrième point nécessite une arduino avec une connectivité internet ?

Commenté [EV2R1]: Ah oui mais je savais pas si ça existait et ensuite si l'arduino peut push sur la BDD avec un lien server

Commenté [CB3R1]: oui pour les deux : https://arduinogetstarted.com/tutorials/arduino-mysql

Lorsque nous l'avons testé, nous avions une Arduino MKR wan 1310 et une antenne LoRA à fixer dessus. Nos tests ont été faits dans les conditions suivantes : arduino branchée à notre PC via USB (l'alimentation électrique, le paramétrage (téléversement) et la récupération des données se font par ce lien).

Dans l'environnement Microsoft, il y a une manière très simple de visualiser les données en direct sous forme d'Excel, c'est le module DataStreamer. C'est un outil Microsoft qui s'ajoute sur l'outil Excel. Nous avons choisi cette extension pour visualiser les données.

Fiche : créer son récepteur arduino (voir dossier)

Raspberry Pi

Raspberry Pi est un équipement qui ressemble un peu à Arduino mais cette fois il a le rôle d'un véritable ordinateur.

Les besoins d'un Raspberry pour être un récepteur LoRA sont les suivants :

- Une alimentation électrique en permanence (batterie ou secteur),
- Une connexion internet Wi-Fi ou éthernet (le Raspberry a un port éthernet et peut aussi se connecter en Wi-Fi),
- Une carte SD avec une mémoire assez conséquente (au moins 16 Go),
- Un module LoRA à fixer sur la Raspberry,
- Une antenne LoRA

Dans notre cas, nous avons : un Raspberry Pi 3 modèle B v1.2, un module Hat GPS/LoRA V1.4 (lien pour acheter + documentation) avec une antenne LoRA (https://www.robotshop.com/eu/fr/hat-raspberry-pi-lora-gps.html), une mini carte SD de 32 Go.

Nous avons fait le choix de prendre ce module Hat GPS/LoRA cependant il n'est plus disponible à la vente et la librairie (WiringPi) n'est plus maintenue par le distributeur (Dragino) mais par des utilisateurs. Ce module n'est donc par forcément le plus optimal pour d'autres prototypes futurs.

Software / Récepteur

La Raspberry a deux rôles principaux :

- Recevoir les données envoyées en LoRA
- Pousser les données sur la base de données

Programmation

Recevoir les données en LoRA

Afin de recevoir les données en LoRA, nous avons expérimenté plusieurs librairies, en C et en Python.

→ Wiring Pi & Rpi LoRA Tranceiver

Dans un premier temps, la première librairie fonctionnelle était celle des exemples du manuel d'utilisation du module : LoRa_GPS_HAT_UserManual_v1.0.pdf. Une autre source d'information sur le module Dragino LoRa Hat : celle-ci.

La réception des données LoRA passe forcément par la librairie Wiring Pi qui permet simplement la gestion des pins avec ce module. Ensuite nous avons utilisé un code GitHub programmé en C (RPI LoRa Tranceiver) qui permettait de recevoir les données en LoRA, ainsi que la gestion des registres (mémoire/stockage).

Voir fiche recevoir des données avec une Raspberry en C (voir Dossier)

Nous n'avons pas réussi à recevoir des données LoRA en Python, mais nous avons identifié deux librairies qui permettraient d'effectuer cela.

→ PyLoRa

https://pypi.org/project/pyLoRa/

→ Raspi-lora

https://pypi.org/project/raspi-lora/

Pousser les données dans la base de données Azure

La Raspberry doit être connectée à Internet pour se connecter à la BDD Azure qui est un cloud (par câble éthernet ou par wifi).

Cette connexion fait appel à l'interopérabilité entre un environnement 100% propriétaire et peu flexible et un environnement libre et très collaboratif : Microsoft Office et Linux Raspberry (OS : Raspbian).

Pour pousser les données dans la base de données Azure, nous avions également plusieurs possibilités : soit en codant en Python ou en C, soit via des API développées par Microsoft.

Librairies nécessaires au code :

- UnixODBC

Afin d'accéder à la base de données Microsoft, il faut utiliser une API. Une API (Aplication Programming Interface) est "une interface logicielle permettant de connecter un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités". Plus couramment, ce serait une mise en lien entre deux entités qui ne se comprendraient pas d'habitude (langages de programmation différents, fonctionnalités différentes etc...). Dans notre cas il s'agit de mettre en relation le logiciel de gestion de la base de données Azure et l'"application" développée sur la Raspberry. Ces API sont spécialisées selon les logiciels et fonctionnalités. Ainsi, l'API adaptée pour notre cas est ODBC (Open DataBase Connectivity). ODBC contient des "bibliothèque[s] contenant des routines d'accès aux données", il permet d'accéder au système de management de la base de données.

UnixODBC est simplement la librairie codée en C qui contient ces routines, adaptée au système d'exploitation Unix (Linux, Debian, Raspbian, Alpine...).

Il est donc nécessaire d'installer cette librairie si l'on veut communiquer entre un système Unix et la base de données Azure.

- FreeTDS

La deuxième librairie utile afin de se connecter à la base de données Azure est FreeTDS. FreeTDS est l'implémentation du protocole TDS (Tabular Data Stream). Le protocole TDS est un protocole de communication de la couche applicative pour que la base de données Azure comprenne ce qu'on lui demande.

Il peut remplacer des drivers ODBC. Je crois qu'on en a besoin pour avoir la libtdsodbc.so qui est primordiale pour la configuration des drivers.

- Code en C

Nous n'avons pas réussi à utiliser ces librairies en codant en C, parce qu'il ne trouvait pas les fonctions définies dans ODBC. Voir : https://www.easysoft.com/developer/languages/c/odbc_tutorial.html

- Code en Python

Pour coder en Python, il y a une librairie plus simple d'utilisation : pyodbc. Afin d'utiliser pyodbc, il faut installer unixODBC et tds. Il serait possible d'installer d'autres drivers ODBC qu'unixODBC, cependant des drivers comme mssql (le plus fréquent) n'existent pas pour l'architecture de la Raspberry, et sous l'OS Raspbian.

Devant les blocages de recevoir les données avec LoRA en Python, ou de pousser les données dans la base de données en C, nous avons décidé de faire un système de client/serveur sur la Raspberry. Ainsi, nous avons modifié le code de rpi-lora-tranceiver afin que cela devienne un client, qui envoie les données reçues en LoRA. Le serveur est le code en Python qui va donc recevoir les données extraites du code en C pour les pousser dans la base de données avec pyodbc.

Voir fiche Récepteur Raspberry (voir Dossier)

Sources utiles pour pyodbc :

https://stackoverflow.com/questions/44527452/cant-open-lib-odbc-driver-13-for-sql-server-sym-linking-issue

https://help.interfaceware.com/kb/904

http://mdupont.com/Blog/Raspberry-Pi/azure-python3.html

D'un point de vue efficience énergétique l'idéal serait que le code pour recevoir les données et pousser dans la base de données soit exclusivement en C. Cependant, du fait du faible coût de l'utilisation de l'environnement Microsoft par les ONG humanitaires comme HI, cela pourrait être plus facile et pertinent d'utiliser les APIs Microsoft.

APIs Microsoft

De notre expérience, nous avons suivi parfaitement les tutos que l'on va vous présenter, cependant nos données n'étaient pas correctement ajoutées sur la BDD. De plus nous avons réussi à simuler des données d'un équipement mais pas à récupérer nos données réelles émises par le Raspberry. Nous n'avons pas eu le temps d'approfondir cette erreur, ni de questionner des personnes qui maîtrisent bien ces outils. Cette piste n'a donc pas été favorisée pour nos tests mais elle reste selon nous intéressante et assez prometteuse dans les cas où comme HI, l'accès aux outils Microsoft est assez facile.

Globalement les APIs utilisés sont IoTHub – Stream Analytics Job – SQL Database. Ces trois APIs sont disponibles via Azure Portal Explorer qui est le portail Azure en ligne qui regroupe un grand nombre des APIs concernant les BDDs.

L'idée est de connecter la Raspberry à IoTHub afin de récupérer ses données, et les autres outils permettent de faire le lien avec la BDD.

Pour relier la Raspberry à IoTHub : https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-raspberry-pi-kit-c-get-started

Pour mettre les données de l'IoT dans la BDD Azure : https://www.mssqltips.com/sqlservertip/6335/how-to-capture-iot-data-in-azure/

Des modifications sont possibles en particulier sur les requêtes SQL de l'exemple qui ne sont pas forcément adaptées à notre exemple.

Pour aller plus loin…

Les idées de fonctionnalités à améliorer sur le récepteur :

- Faire fonctionner les APIs Microsoft ou migrer tout le code en Python ou en C
- Fiabiliser les échanges de données en demandant des acknowledgments des données envoyées en LoRA
- Est-ce qu'il faut rajouter de la sécurité sur la transmission ?
- Utiliser la Raspberry sans écran clavier et souris. C'est-à-dire lancer le client et le serveur automatiquement
- Sur le serveur en Python faire des erreurs non-bloquantes, pour ne pas avoir à redémarrer le programme selon les erreurs mais plutôt qu'il se relance automatiquement, ou qu'il continue de boucler…

Phases de tests : explication et retours

Voir fiche Récepteur Raspberry (voir Dossier)

Durant une semaine nous avons mis en place un test dans le hall à HI Lyon. Ainsi, nous avions l'émetteur fixé, avec les bonnes questions. Pour l'alimentation de l'émetteur nous avons utilisé un adaptateur de chargeur de téléphone et un câble micro-usb.

Problème rencontré : En changeant d'adaptateur les seuils de tension des boutons ont changé et n'étaient plus forcément pertinents : un seuil allumant plusieurs boutons / un seuil n'allumant aucun bouton.

Ensuite, nous avons tiré une rallonge pour pouvoir placer le boîtier sur le portique en sortie.

Problème rencontré : Nous avons eu des problèmes d'interférences ou des moments où certains boutons étaient ressentis comme appuyés en continu alors qu'ils ne l'étaient jamais. Est-ce que c'était un faux contact ? Un problème de tension également ?

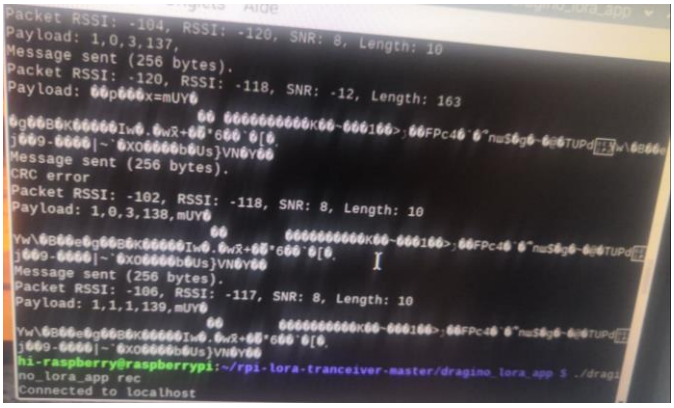
De plus, un problème que nous avons rencontré est au démarrage / à la mise sous tension de l'arduino, si la carte SD n'est pas dans le module SD, cela ne fonctionne pas. Il faudrait donc mettre cette partie du code en non-bloquante si jamais, ou alors mettre un message lumineux d'erreur avec les leds pour prévenir qu'il n'y a pas de carte SD.

Dans une seconde partie, il a fallu s'occuper d'où mettre le récepteur (la Raspberry).

Problème rencontré : Malgré les longues distances attendues grâce à la technologie LoRA, nous n'arrivions pas à recevoir les données de notre bureau (1er étage à l'opposé de l'entrée). Est-ce que la synchronisation en fréquence pouvait être meilleure ? Est-ce encore une fois une histoire de tension ? Est-ce le fait que la fréquence de l'arduino 868,1 MHz n'était pas sa fréquence de fonctionnement parfaite ?

Nous avons alors installé la Raspberry dans la salle derrière l'entrée. D'ici nous captions bien les premiers jours, puis certains jours nous ne captions plus vraiment les informations envoyées par l'émetteur.

Problème rencontré : En plus de ne pas toujours capter les données envoyées, il arrivait de recevoir des données corrompues.



A côté de la feedback box, nous avons placé un QR code dirigeant vers un questionnaire Microsoft Forms permettant de recueillir l'avis des "testeurs" de la feedback box. Le questionnaire :

Microsoft Forms

Voir pdf avec les réponses du formulaire dans ce dossier.

Sources

FreeTDS, UnixODBC, pyodbc, ... :

https://help.interfaceware.com/kb/904

http://www.unixodbc.org/doc/FreeTDS.html

https://pastebin.com/qFQMUAxn

https://stackoverflow.com/questions/44527452/cant-open-lib-odbc-driver-13-for-sql-server-sym-linking-issue

https://stackoverflow.com/questions/48813238/connecting-raspberry-pi-3-to-mssql-server-using-pyodbc

https://stackoverflow.com/questions/64366258/trying-to-connect-to-an-azure-sql-database-with-python-pyodbc-and-a-raspberry

http://mdupont.com/Blog/Raspberry-Pi/azure-python3.html

https://github.com/denisenkom/pytds

https://circuitdigest.com/microcontroller-projects/raspberry-pi-with-lora-peer-to-peer-communication-with-arduino