Développement d'acquisition de données d'un anémomètre sur carte Arduino

Muriel Lagauzere, Gabriel Pierre André Moreau, Cyril Azzolina

Développement d'acquisition de données d'un anémomètre sur carte Arduino

Muriel Lagauzère, Gabriel Moreau, Cyril Azzolina

Univ. Grenoble Alpes, CNRS, Grenoble INP, LEGI, 38000 Grenoble, France

1. Introduction

Le laboratoire LEGI (Laboratoire des Écoulements Géophysiques et Industriels) à Grenoble accueille environ 140 personnes dont la moitié sont des permanents. Il est spécialisé depuis sa création dans la mécanique des fluides, avec une forte approche expérimentale en partie due à ses nombreux grands équipements (Soufflerie, Plate-forme Coriolis, Canal à Houle, Tunnel Hydrodynamique...). De très nombreuses acquisitions de données sont réalisées sur les expériences du laboratoire avec du matériel souvent surdimensionné en terme de nombre de voies et de fréquence d'échantillonnage. Depuis quelques années, nous explorons des méthodes de pilotages et d'acquisitions avec des matériels moins onéreux, en utilisant si possible des logiciels libres. Dans cette quête du graal, rencontrer l'Arduino est inévitable. Nous avons donc cherché, au travers d'un exemple simple et utile pour le laboratoire, à développer un kit utilisant de nombreuses ressources de l'Arduino qui nous semblent pouvoir être réutilisées dans de futures études expérimentales (Cf Figure 2).

Un nouveau développement d'acquisition de données a donc été réalisé sur carte Arduino pour réaliser l'acquisition du signal d'un anémomètre à coupelles (Cf Figure 3) dans la soufflerie du LEGI (Cf Figure 1). Cette soufflerie est l'un des grands équipements du laboratoire. Elle permet d'atteindre des vitesses jusqu'à 50m/s dans la veine d'essai L'objectif est de surveiller en temps réel la vitesse moyenne de l'air. Un simple Arduino avec un afficheur LED suffit à cette tâche. Nous avons ensuite étendu cette solution afin de pouvoir positionner l'afficheur de la vitesse à l'entrée de la soufflerie via une communication radio entre deux modules Arduino. Enfin, dès que l'on parle acquisition ou monitoring de systèmes, pouvoir récupérer les données et les enregistrer automatiquement sur les serveurs de stockage est de nos jours une nécessité. Nous avons ainsi ajouté une connexion réseau sur l'Arduino distant et connecté celui-ci avec un bus message dédié à l'embarqué.

Figure 1. La soufflerie à bas niveau de turbulence

Figure 2. Le module d'acquisition de vitesse

2. Développement d'une acquisition de données sur carte Arduino et enregistrement sur carte SD

Nous avons récupéré et enregistré le signal de l'anémomètre. La précision de ce capteur correspond à 1% de l'étendue de mesure jusqu'à 50 m/s et à 2% de l'étendue de mesure dans la gamme allant de 50 à 77 m/s. Son temps de réponse est de l'ordre de la milliseconde. Pour obtenir un système portable, l'anémomètre est alimenté par une pile 9 V par le port Vin de la carte Arduino Uno. La sortie du capteur, un signal impulsion (10 Hz par nœud), est connectée à une entrée numérique de l'Arduino qui effectue le comptage de ces impulsions chaque seconde.

Dans un premier temps, l'enregistrement des données a été réalisé sur la mémoire EEPROM de l'Arduino. Cette mémoire est limitée à 1 ko (1024 octets). Nous avons rapidement saturé la mémoire EEPROM de la carte. Nous avons rajouté un shield SD (Cf. Figure 4) avec une carte SD (4 Go) pour augmenter les capacités de stockage et avoir un système d'acquisition de données autonome.

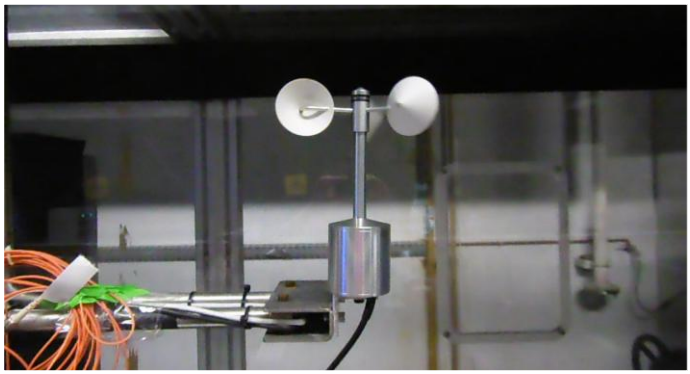

Figure 3. Anémomètre à coupelles

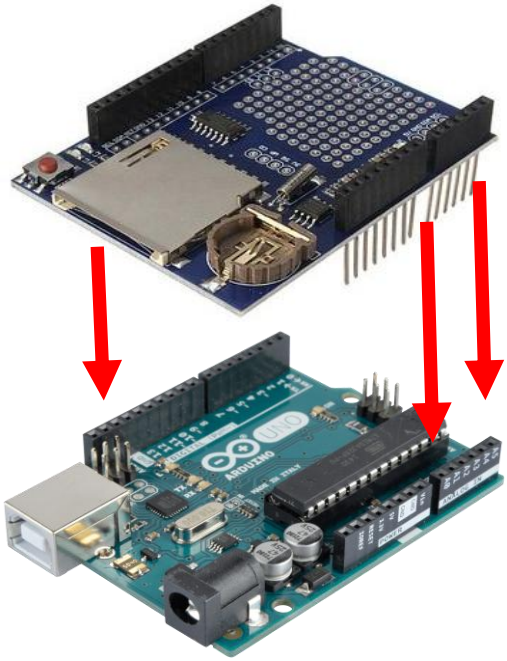

Figure 4. Shield SD sur carte Arduino

Nous avons pu réaliser des acquisitions de plusieurs heures grâce à l'ajout de cette carte SD. Un afficheur à deux lignes a été ajouté afin de lire l'évolution en temps réel de la vitesse de l'anémomètre en m/s. Les données sont enregistrées dans un fichier texte, la première colonne correspondant au temps et la deuxième à la valeur de la vitesse en m/s.

Une calibration de ce système d'acquisition (Cf Figure 5) a été effectuée dans la soufflerie à bas niveau de turbulence du LEGI. Pour cette calibration, nous avons comparé les valeurs de l'anémomètre à coupelles avec celles d'un tube de Pitot placé dans le convergent de la soufflerie. Ce tube de Pitot a lui-même été étalonné dans une soufflerie étalon. En faisant varier la vitesse de la soufflerie dans une gamme allant de 0 à 25 m/s, nous avons relevé les valeurs de tube de Pitot et de l'anémomètre à coupelles (enregistrement sur carte SD et affichage de la vitesse sur un afficheur) pour chaque valeur de vitesse puis établi une courbe de calibration (Cf Figure 7). La sortie analogique de l'anémomètre a pu être récupérée sans ajout de filtre sur la carte Arduino. Dans un deuxième temps, nous avons procédé à un déplacement de l'anémomètre à coupelles sur les quatre mètres de la veine d'essai de la soufflerie et continué cette calibration en comparaison avec le tube de Pitot.

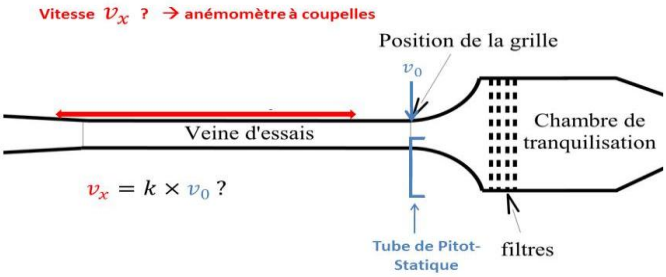

Figure 5. Schéma de calibration

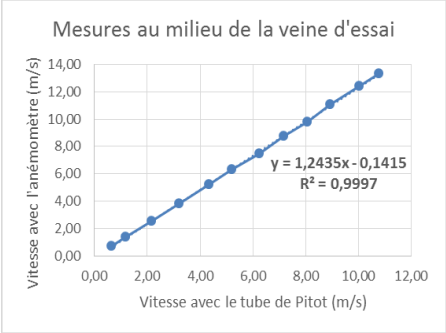

Figure 6. Courbe de calibration

3. Communication radio

Nous avons développé un module de communication radio pour plusieurs raisons : afin de nous affranchir au mieux du bruit de la soufflerie et pouvoir surveiller sa vitesse, afin d'interagir depuis un point fixe dans la pièce avec le module d'acquisition qui est autonome (alimentation par pile, aucun fil extérieur à raccorder) géographiquement.

La radiocommunication est réalisée à l'aide de deux cartes Arduino, d'un module émetteur 433 MHz avec une antenne radio émettrice et d'un module récepteur avec une antenne radio réceptrice (Cf. Figure 7). La bibliothèque VirtualWire est requise pour la communication radio. Des essais ont été menés pour tester les limites en distance et en obstacles de cette communication. En utilisant de simples fils électriques de 173 mm de long comme antennes émettrice et réceptrice, des communications en ligne droite d'une centaine de mètres, de part et d'autre d'une cloison, ont été possibles, ce qui est largement suffisant dans notre configuration actuelle. Cependant, ces limites pourraient être repoussées par l'utilisation d'antennes rigides de plus longues portées.

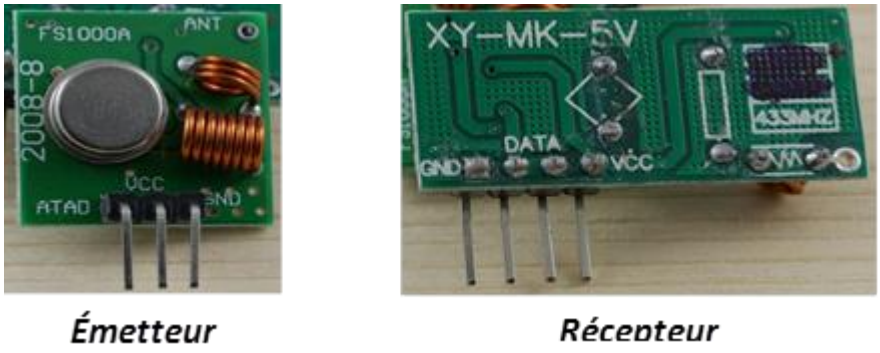

Figure 7. Cartes émetteur et récepteur radio

Dans un futur proche, nous allons essayer de migrer les programmes vers la bibliothèque RadioHead qui remplace et étend VirtualWire désormais en fin de vie. Cependant, cette bibliothèque est plus générique mais aussi plus complexe à l'usage.

4. Shield Ethernet et site web de surveillance du signal de l'anémomètre

La seconde carte Arduino étant dans un lieu fixe, proche d'une prise réseau Ethernet, un shield réseau a été ajouté sur ce montage. Il est ainsi possible de distribuer ou d'obtenir les données

d'acquisition depuis l'intégralité du réseau du LEGI. Cependant, plutôt que de pousser les données vers un serveur de stockage particulier, ou d'aller quérir ces données depuis un poste client sur l'Arduino, nous pensons que l'architecture avec un bus message est particulièrement bien adaptée à plusieurs de nos problématiques d'acquisition basse vitesse et de pilotage. Ce projet était donc l'occasion de la mettre en pratique sur un cas réel.

Ce qui distingue un serveur de message d'un serveur de stockage est son fonctionnement. Un serveur de stockage de masse va enregistrer sur un support de longue durée, une baie de disque par exemple, les données sous forme de fichier. Un serveur de message ou bus permet d'interconnecter des programmes entre eux. Ceux-ci s'échangent des messages sur un bus virtuel : les données d'acquisition, les ordres de pilotages…, au travers du bus. Chaque programme, indépendant les uns des autres, se connecte sur le bus et sur lui seul. Il n'y a pas de dialogue direct entre les différents programmes. Le bus que nous avons choisis utilise le protocole MQTT (Message Queuing Telemetry Transport). Ce protocole est particulièrement bien adapté aux équipements connectés de faible puissance. L'implémentation MQTT utilisé est le serveur `Mosquitto`. Cette architecture autour d'un bus est très souple et permet de démarrer, de relancer les programmes dans un ordre quelconque. Il est aussi possible de s'y connecter via des consoles en mode texte afin d'envoyer ou de recevoir des messages manuellement à des fins de débogages.

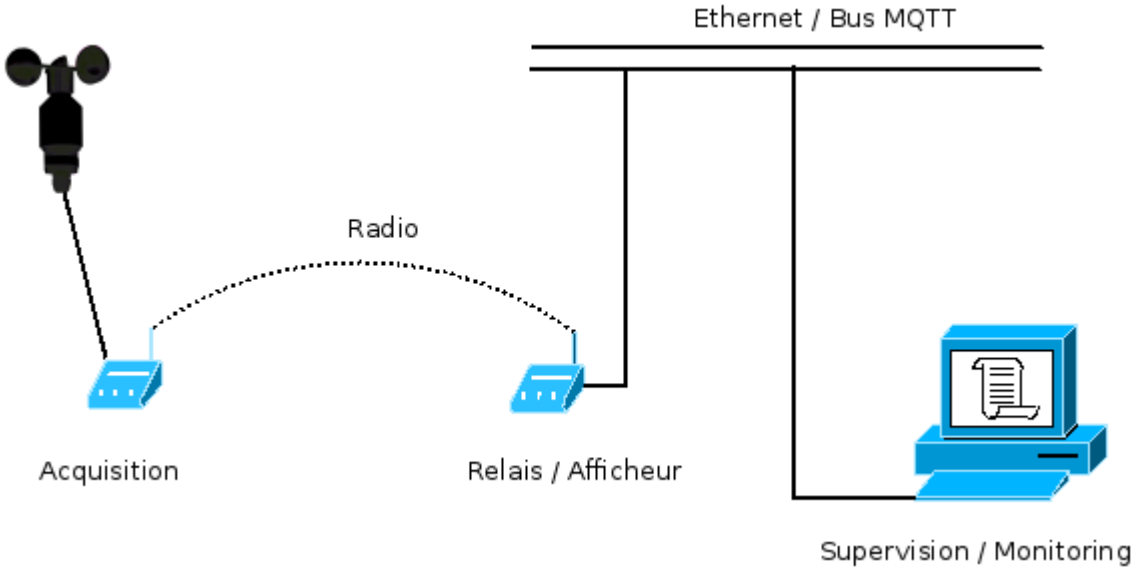

Figure 8. Schéma de l'installation finale

Le serveur `Mosquitto` est configuré pour écouter par défaut sur le port 1883 normalisé et il a été déployé dans une machine virtuelle Debian GNU/Linux Au niveau du second Arduino, nous avons utilisé la bibliothèque PubSubClient qui est assez légère en termes de ressource pour programmer un client permettant de publier et de souscrire à des messages (*topics* dans le vocabulaire MQTT). Malgré l'absence de sécurisation par chiffrement TLS des transferts avec cette bibliothèque, les *topics* MQTT sont tout de même sécurisés par un couple login/password dédié pour chaque client. Ainsi, l'Arduino ne peut écrire que sous les topics « `anemometer/bucket/speed` » et « `anemometer/bucket/status` » alors que le superviseur peut lire toute l'arborescence « `anemometer/bucket/#` ».

Après la réception des données sur la deuxième carte Arduino, celles-ci sont envoyées sur le serveur de message distant en utilisant donc le protocolebus MQTT. Une fois publié sur le bus MQTT, tout est possible. Ce projet était de nouveau l'occasion de tester des nouvelles technologies et de ne pas faire un outil de supervision *classique* dépendant de la plate-forme et à déployer sur les postes utilisateurs. L'idée a été de faire un client web fonctionnant dans tout navigateur et permettant son usage sur tout poste, même sur un moniteur de type télévision passive.

Pour cette supervision Web, le choix s'est donc porté sur une solution de prototypage rapide en se basant sur un site web statique qui fait appel à des scripts en JavaScript, notamment la bibliothèque

Paho. On évite ainsi la mise en place du serveur web de type Apache. Il est probable qu'il nous faille passer par l'étape d'un serveur web dans une version finale de production afin d'améliorer la sécurité et l'authentification des utilisateurs. Lors de la réception d'un message sur le *topic* « `anemometer/bucket` », un événement JavaScript permet de modifier l'arbre DOM de la page HTML et d'afficher ou non certains paragraphes ou images, ainsi que de changer en temps réel la valeur de la vitesse dans la veine d'essai.

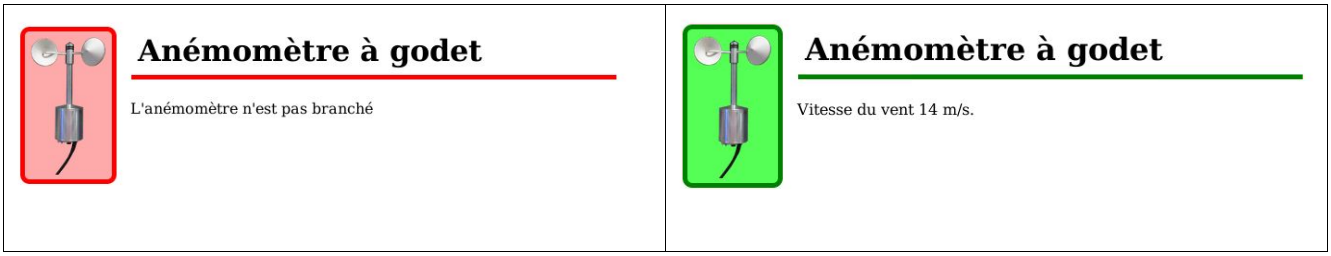

Figure 9. Interface client web en HTML/Javascript

Le débogage de la partie MQTT du projet a pu se faire très facilement en utilisant les deux outils en ligne de commande « `mosquitto_pub` » (publier) et « `mosquitto_sub` » (souscrire). Tant la partie Arduino que la partie supervision Web peuvent être ainsi développés et validés indépendamment. La partie Web n'utilisant de serveur dynamique, l'ouverture du fichier sur son propre poste de travail suffit à faire fonctionner l'application.

CONCLUSION

Pour conclure, le signal de l'anémomètre à coupelles peut être soit enregistré sur la mémoire EEPROM de la carte Arduino, soit sur une carte SD. Il peut également être transmis par communication radio à un autre Arduino ou diffusé sur le réseau Ethernet via un serveur distant qui pourra centraliser les messages provenant de centaines de capteurs et les rediffuser à la demande, un serveur MQTT par exemple. À noter que l'ensemble des développements ont été rapides et le code écrit est suffisamment générique et documenté pour être réutilisé dans de nombreux autres projets du laboratoire. C'est la raison pour laquelle nous avons souhaité que celui-ci soit librement diffusé sur la forge du LEGI sous la licence permissive dite MIT.

Muriel et Gabriel tiennent à remercier Cyril pour son sérieux et son investissement tout le long de son stage d'IUT réalisé au LEGI. Sans son apport fondamental, ce projet serait resté à l'état de concept.

5. Références

1. C.AZZOLINA, « Développement d'acquisition de données sur une carte Arduino », Rapport de stage 2ème année IUT GEII (2018)
2. S.MARTINS, « Calibration d'un tube de Pitot et amélioration d'un programme LabVIEW», Rapport de stage 2ème année IUT Mesures Physiques (2017)
3. MQTT PubSubClient Arduino Client - https://pubsubclient.knolleary.net/
4. MQTT Paho Arduino Client - https://www.eclipse.org/paho/clients/c/embedded/
5. MQTT Paho JavaScript Client - https://www.eclipse.org/paho/clients/js/