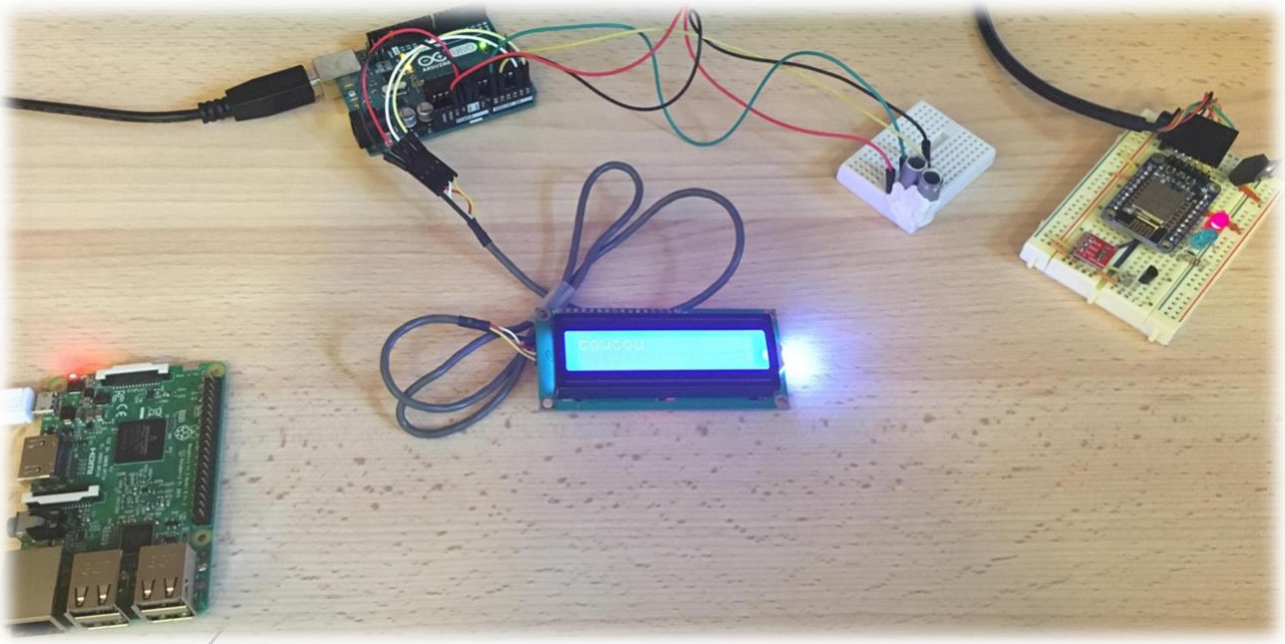
PROJET TECH

17/12/2018

Un outil de diagnostic optique

Rapport final



Benjamin COLLERY, Louis MATHA, Guillaume PISTORIUS
ÉCOLE NATIONALE SUPÉRIEURE DES MINES DE ST-ETIENNE

MINES Saint-Étienne

SOMMAIRE

Table des matières

Introduction

1) Contexte

Dans le cadre de la collection de données en masse, de nombreux capteurs ont été déployés dans le bâtiment de l'Espace Fauriel. Ils permettent de relever des mesures de températures, humidité et luminosité.

Le dispositif mis en place utilise des cartes ESP8266, qui accueillent les capteurs, le protocole de communication MQTT, ainsi qu'un broker. (cf Annexe 3))

Le principal problème dans l'état actuel est qu'il est impossible de détecter le disfonctionnement d'une carte. L'objectif de notre projet est donc de développer une solution permettant de détecter différentes pannes.

2) Objectif

Le cadre du projet était fixé dès le départ. Les LEDs de l'ESP8266 permettent dans un premier temps d'envoyer les erreurs sous forme de clignotements. Le capteur optique que nous devons fabriquer doit être capable d'analyser ces clignotements et d'afficher l'erreur correspondante sur un écran LCD. Le reste du projet n'étant pas définit, notre objectif était de trouver des solutions techniques permettant de respecter les attentes.

Les points suivants sont les grandes parties du problème :

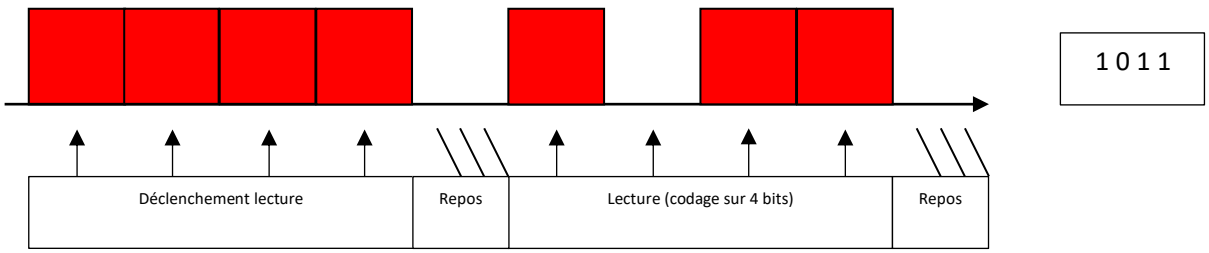
- compréhension du problème
- compréhension du code / MQTT
- choix du protocole de clignotement des LED
- choix des capteurs de détection
- méthode d'analyse des clignotements
- design d'un boitier

I. Solutions envisagées

1) Séquence de transmission

L'énoncé du sujet incitait à utiliser une seule LED pour transmettre l'erreur. En ce sens nous avons pris cette direction. Le problème était de trouver un moyen d'annoncer le début d'une séquence d'erreur. Ainsi, nous avons choisi de bannir la séquence [1,1,1,1] pour la transmission d'une erreur, celle -ci étant réservée à l'annonce du début d'une séquence. Cette séquence doit aussi être entourée d'état bas, pour s'assurer qu'on ne confond jamais une séquence de déclenchement avec une séquence erreur.

Ainsi nous sommes arrivés au format de clignotement de la LED suivant :



(Exemple de lecture pour la séquence 1 0 1 1 : les flèches correspondent aux moments on les valeurs captées par la photorésistance sont relevées)

2) Forme des séquences

Nous avons choisi de coder les séquences représentant les erreurs sur 4 clignotements de LED. La Led étant soit allumée soit éteinte. Cela porte à $2^4 = 16$ le nombre d'erreurs différentes pouvant être transmises, ce qui était suffisant pour ce prototype.

3) Choix de la LED

Les cartes ESP8266 comportant des LEDs intégrées, nous avons choisi d'en utiliser une pour le protocole de transmission décrit ci-dessus. Cela permet en effet de ne pas avoir à remodifier les board supportant les ESP8266 en y ajoutant une nouvelle LED.

4) Choix des capteurs de détection

Par soucis de simplicité nous nous sommes directement orientés vers les photorésistances pour détecter les clignotements de la LED. Nous n'avions en effet besoin que d'un seul renseignement qui était : LED allumée ou éteinte ? Un module RGB n'étant donc pas utile et trop complexe ici.

5) Point d'accès Wifi

Sur une recommandation du professeur nous avons pris une Raspberry Pi pour créer notre point d'accès wifi. Cela était certainement la méthode la plus abordable pour nous.

Une autre solution était de connecter les différents éléments sur un routeur quelconque. Cela ne correspondait pas à notre mode d'utilisation, puisque la Raspberry peut être utilisée dans différents endroit (plus simple que de déplacer un vrai routeur).

Ce choix permettait également d'installer le broker MQTT directement sur la Raspberry Pi.

6) Code Arduino

A ce stade du projet, nous n'avions qu'expérimenter les différents aspects de l'Arduino, et n'avions pas encore d'idées préétablies au niveau du code à écrire

II. Solutions testées

1) Séquence de détection

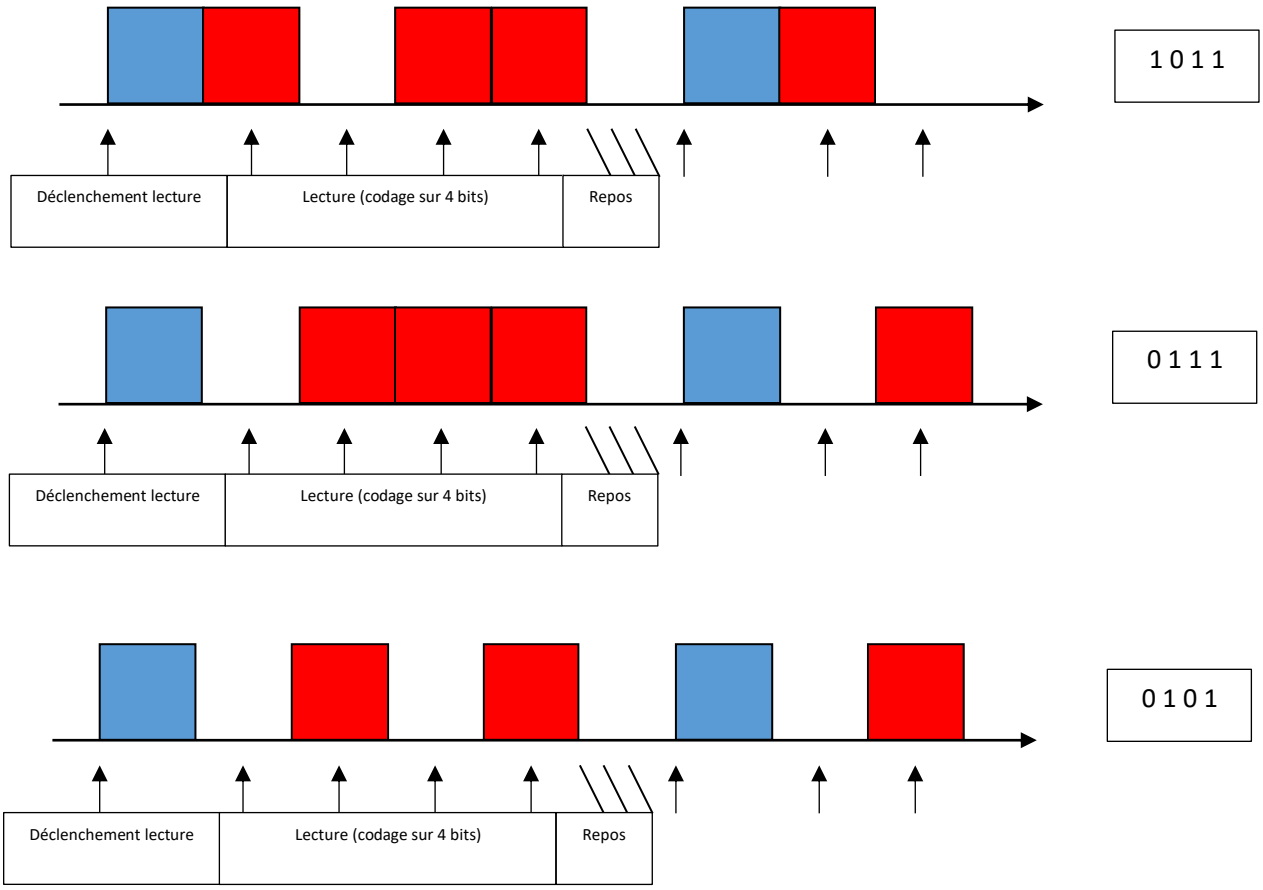
Dans un premier temps nous avons tenté de réaliser la solution envisagée, c'est à dire en utilisant une seule LED et une séquence [0,1,1,1,1,0]. Nous nous sommes rapidement rendu compte en commençant à coder que ce choix n'était pas optimal et pouvant même mener à des confusions.

Nous avons donc choisi d'utiliser deux LEDs. En effet l'ESP8266 possède 2 LEDs intégrées, nous pouvions donc garder notre idée de ne pas ajouter de LED.

La méthode est encore plus simple ici. Nous utilisons la LED bleue de l'ESP8266 comme séquence de détection. C'est à dire simplement un seul clignotement de la LED.

2) Forme des séquences

Pour la forme des séquences nous avons uniquement testé les séquences sous forme de 4 bits prenant les valeurs 0 ou 1 respectivement pour la LED allumée et éteinte.



3) Choix des LEDs

Pour les Leds nous avons dans un premier temps gardé notre idée de départ, c'est à dire utiliser les LEDs intégrées à l'ESP8266.

Nous avons pu réaliser toutes les phases de test de détection des erreurs grâce aux deux LEDs.

Cependant, lorsque nous sommes passés aux tests finaux, c'est à dire la détection et l'affichage des erreurs par l'Arduino, nous nous sommes rendu compte que l'intensité des LEDs était très faible. Cela ne permettait pas de recevoir des valeurs correctes via les photorésistances côté Arduino. Nous aurions pu ici régler les valeurs lues en faisant des ponts diviseurs de tensions, mais nous avons préféré la simplicité d'ajouter deux LED indépendantes. Nous avons donc ajouté sur l'ESP8266 deux LEDs supplémentaires, une bleue et une rouge.

4) Choix des capteurs de détection

Nous avons uniquement utilisé les photorésistances pour l'ensemble de nos tests. L'intérêt d'un capteur RGB était limité même s'il permettait de détecter les clignotements des deux LEDs à la fois. En effet, en séparant les lectures des 2 LEDs, l'implémentation était plus aisée et également plus fiable.

5) Code Arduino

Nous allons ici expliquer les grandes lignes du code, pour plus de détails se référer directement au code *capteur_optique*, qui est commenté.

Affichage LCD

```
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 20, 4);
```

Afin de réaliser l'affichage des erreurs, nous avons choisi d'utiliser un écran LCD 20 x 4, qui était amplement suffisant pour ce que nous avions à afficher.

L'utilisation du bus I2C permet d'utiliser seulement 4 broches pour transmettre l'information, ce qui est intéressant d'un point de vue câblage mais également codage.

Pour ce qui est du codage justement, nous avons utilisé une librairie créée pour la communication I2C, c'est la librairie LiquidCrystal_I2C

Détection de séquence

Concernant la détection nous avons dans un premier temps procédé de la manière suivante. Une séquence de 4 chiffres est déclarée. Dès que notre séquence de détection est reconnue, on lance la lecture de la séquence. On remplit ainsi la séquence. Une fois terminé, on la compare à toutes les séquences d'erreurs enregistrées. Si elle correspond à l'une d'entre elle on affiche sur l'écran LCD le nom de l'erreur. Sinon on affiche "panne inconnue".

Cependant nous nous sommes rendus compte lors de nos tests qu'il arrivait qu'une séquence soit mal lue. Cela peut être due par exemple au mauvais placement du dispositif de reconnaissance sur les LED.

Nous avons donc fait une seconde version du code permettant d'éviter la reconnaissance d'une mauvaise erreur. Nous avons choisi de déclarer non pas une mais 2 séquences. A chaque itération, les deux séquences sont comparées. Si elles sont égales on admet que la séquence reconnue est la bonne. Sinon on mémorise seulement la dernière lue, et on en lit une nouvelle. Cela jusqu'à ce qu'une erreur soit reconnue.

Cette méthode a l'inconvénient d'allonger le temps de reconnaissance, mais permet d'éviter les erreurs, c'est donc un compromis à choisir.

Lecture des séquences

La première version du code consistait à lire la valeur de la LED bleue selon une fréquence très élevée, ainsi on était assurés de ne pas rater la séquence de détection. Ensuite on lisait les valeurs à intervalle réguliers (2secondes).

Encore une fois nous nous sommes rendu compte de la difficulté qu'avait notre programme à lire les bonnes valeurs.

Nous avons compris que cela venait du fait que nous lisions les valeurs trop près du début des clignotements (voir le schéma A).

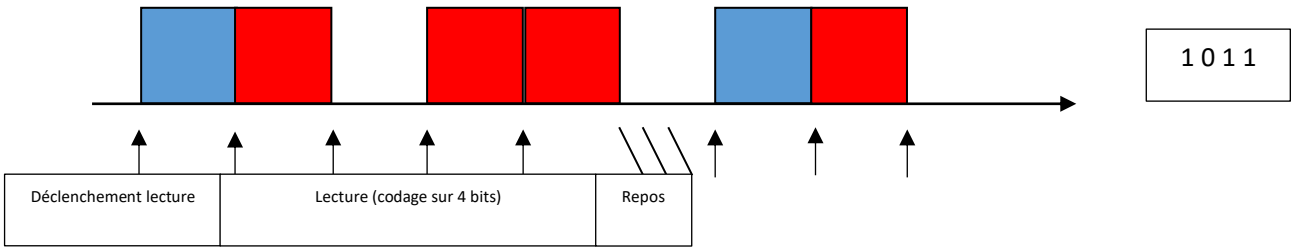


Schéma A : lectures aux bords

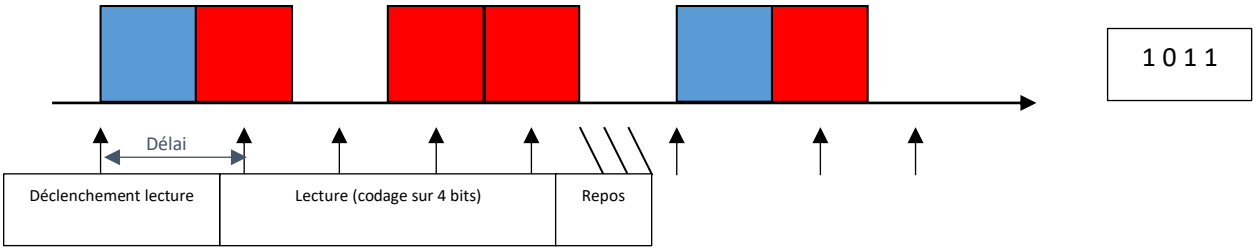


Schéma B : lectures aux milieux

Nous avons donc choisi garder un cadençage rapide de lecture de la LED bleue, mais ensuite nous avons recentré la lecture des valeurs au milieu du clignotement grâce à un simple delay() (voir schéma B ci-dessus). Cela a considérablement réduit le nombre d'erreur de lecture de l'état des LEDs.

7) Code ESP8266

clignotement : une seule LED
utiliser les LED de l'ESP8266
capteurs de détection : photorésistance
connexion Raspberry wifi
coder les bits sur 4 états

III. Pistes d'amélioration

Plusieurs points pourraient être améliorés en passant plus de temps sur le projet. En voici quelques un :

1) Lecture des LEDs

En fait tout repose sur un juste milieu entre vitesse de clignotement, synchronisation de lecture, et nombre de séquences successives comparées. La vitesse de clignotement devrait pouvoir être largement augmentée. Cependant il faudrait bien faire attention et faire de nombreux tests pour voir jusqu'à quelle limite on parvient à lire les valeurs des photorésistances correctement. Si notre système est fiable, on peut également gagner du temps en supprimant totalement la partie qui consiste à comparer deux séquences lues successivement. Cela permettrait de diviser le temps de reconnaissance d'une erreur par 2 dans le cas présent.

2) Codage des erreurs

Une autre piste d'amélioration pour la rapidité de lecture est le nombre d'états différents des LEDs. Dans notre cas, les LEDs étaient soit allumées soit éteintes, donc 2 états possibles. On pourrait imaginer une évolution du système où les LEDs auraient 4 ou 5 états d'éclairage différents, et ainsi coder les erreurs sur non pas deux mais 5 états pour chaque bit. C'est à dire avec 2 clignotements : $5^2 = 25$ erreurs programmables. De même les deux LEDs pourraient être remplacées par une seule, avec différents états. Un état étant réservé à la séquence de déclenchement.

3) Compacité

La compacité du dispositif pourrait être augmentée en utilisant une Arduino Micro plutôt qu'une Uno. En effet celle-ci peut être branchée sur une board car elle comporte des broches. Des câbles rigides et plus compacts (comme ceux utilisés sur l'ESP) auraient également pu être utilisés afin d'augmenter la compacité et la maniabilité du dispositif. (Malheureusement un problème inexpliqué empêchait le dispositif de fonctionner avec une telle carte).

Le tout aurait pu être placé dans un boîtier prévu à cet effet, possiblement fabriqué via imprimante 3D.

IV. Prise de recul sur le projet

1) Déroulement

Dès le début l'équipe était vraiment motivée par le sujet. Les technologies utilisées n'étaient pas maîtrisées par les membres de l'équipe. Par exemple aucun de nous n'avait déjà programmé une arduino, une Raspebrry Pi et n'avait jamais utilisé de protocole et broker MQTT. A ceci s'ajoutait le cadre intéressant de l'Internet of Things et des objets connectés.

La première grande étape du projet a consisté à se familiariser avec les différents éléments et technologies. Nous avons rapidement pu comprendre le fonctionnement du protocole MQTT. Ensuite nous avons cherché différents tutoriel Arduino pour comprendre les bases de son fonctionnement et du langage de programmation.

En parallèle de cette phase nous avons commencé à réfléchir à des solutions techniques (voir solutions envisagées). Entre autres nous avons cherché comment produire les séquences avec les LEDs, quel genre de capteur de luminosité choisir.

La phase suivante a été de connecter les éléments entre eux. C'est à dire que pour commencer nous avions besoin de connecter la carte ESP8266 au réseau. Nous avons donc dû créer un point d'accès wifi sur la Raspberry Pi. Là encore même si l'un d'entre nous connaissait déjà linux, c'était tout nouveau pour nous.

Enfin est venu la phase de développement du montage et du code. C'est évidement celle qui nous a pris le plus de temps. Nous avons été confrontés à de nombreuses erreurs notamment dues au fait que nous débutions la programmation Arduino.

2) Organisation

Pour être honnêtes nous n'avons pas réussi à garder un créneau de travail projet tech fixe. Nous devions en effet associer cela aux différentes occupations de chacun dans son temps libre. Nous avons cependant réussi à travailler régulièrement, sauf aux alentours des vacances.

3) Rendu

Les membres de l'équipe sont tous satisfait du travail accompli. En effet, partant de faibles connaissances nous avons réussi à produire un prototype qui répond aux attentes.

Cependant cela est à nuancer avec le fait que nous aurions tous aimé aller plus loin et proposer un prototype plus abouti. Nous aurions notamment pu développer les pistes d'amélioration proposées dans la partie précédente. Cela aurait permis de faire quelque chose de plus propre.

La deuxième année est très difficile car très prenante au niveau associatif et des cours et rendus en majeure. Nous avons certainement fait du projet Tech (malgré notre intérêt) un impératif de second plan. Ceci explique que nous ne sommes pas allés plus loin dans la réalisation du prototype et de ses fonctions.

4) Encadrement

L'ensemble des membres de l'équipe s'accorde à dire que l'encadrement que vous nous avez fourni était très satisfaisant. Vous avez pris beaucoup de temps au départ pour nous expliquer les grandes lignes du projet ainsi que l'architecture entre les différents composants. Par la suite vous avez toujours été très réactif dans les échanges de mails et très conciliant pour les demandes de réunions.

Tout cela a évidemment participé au bon déroulement du projet, nous en sommes très satisfaits.

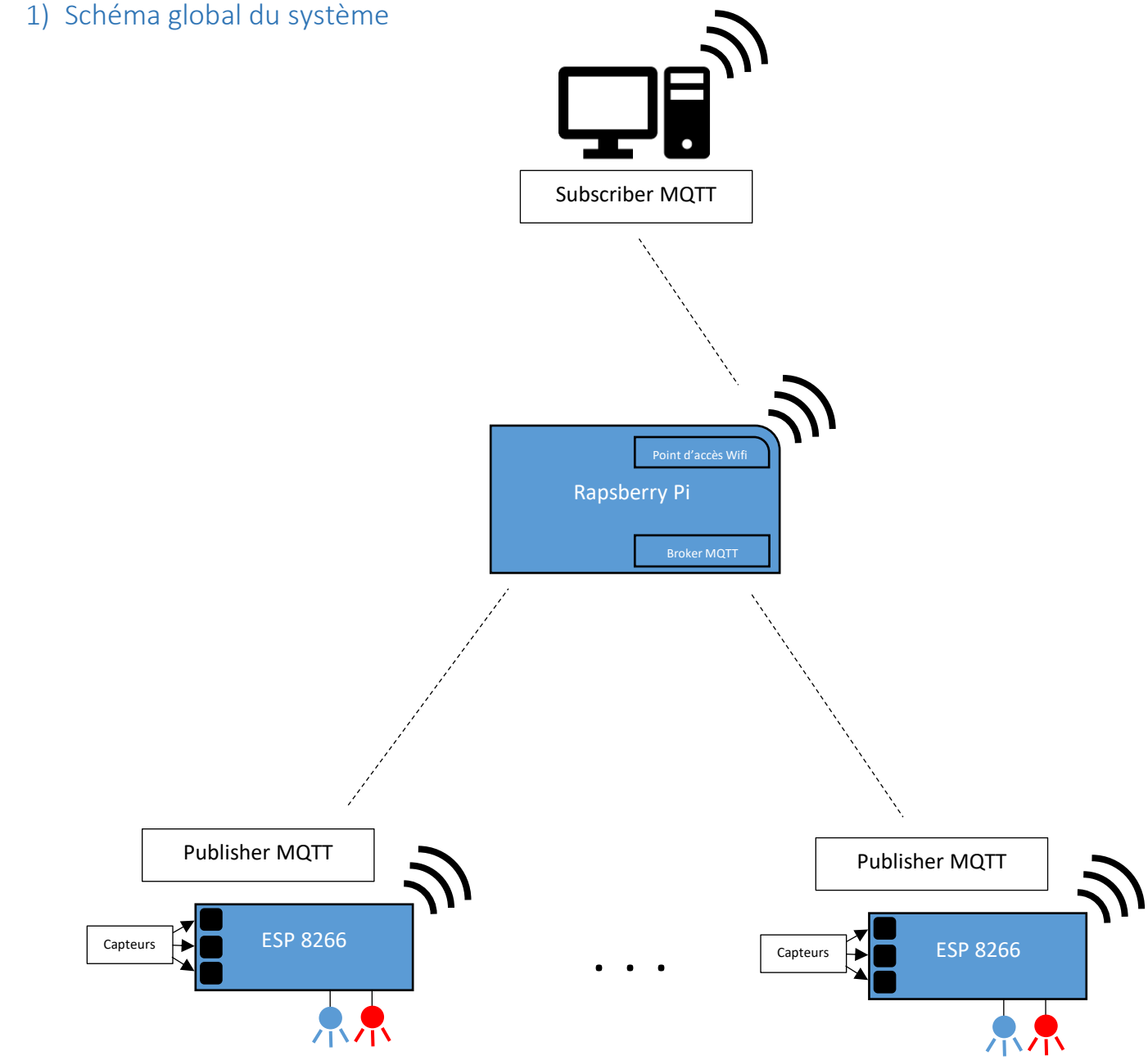
5) Apport du projet

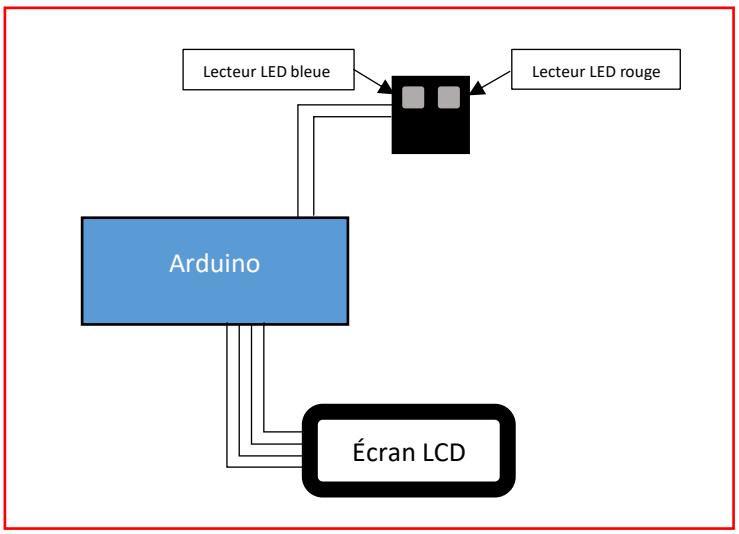
Même si cela n'a pas été simple au départ, nous sommes tous d'accord sur le fait qu'avoir un projet sur lequel nous n'avons pas toutes les compétences au départ est intéressant. En effet, pour la motivation il était important pour nous de pouvoir apprendre des choses pendant la réalisation. Cela a été tout à fait le cas. L'objectif de ce projet était clairement d'enrichir, en plus de nos compétences, notre culture et connaissances informatiques.

Un projet demandant des compétences déjà maîtrisées aurait suscité bien moins de motivation de notre part. Et aurait été peu enrichissant.

V. Annexe

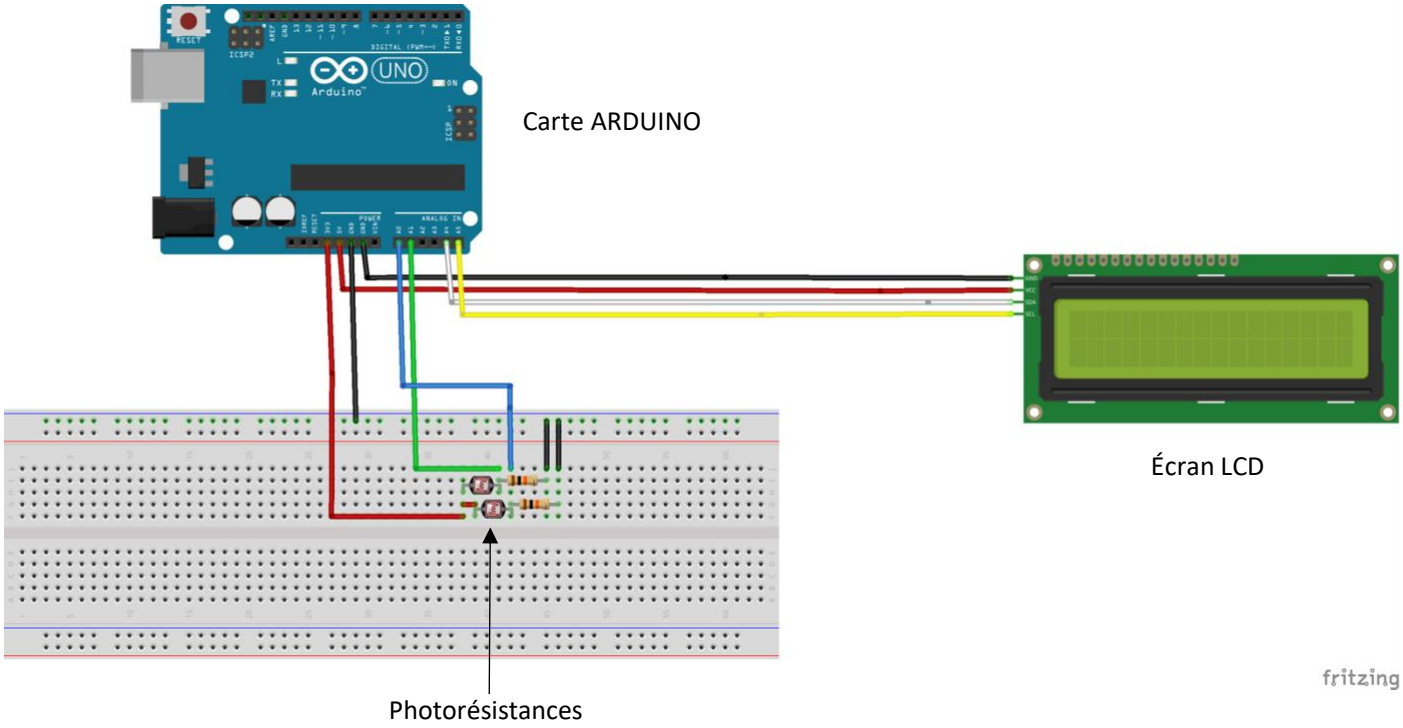
1) Schéma global du système



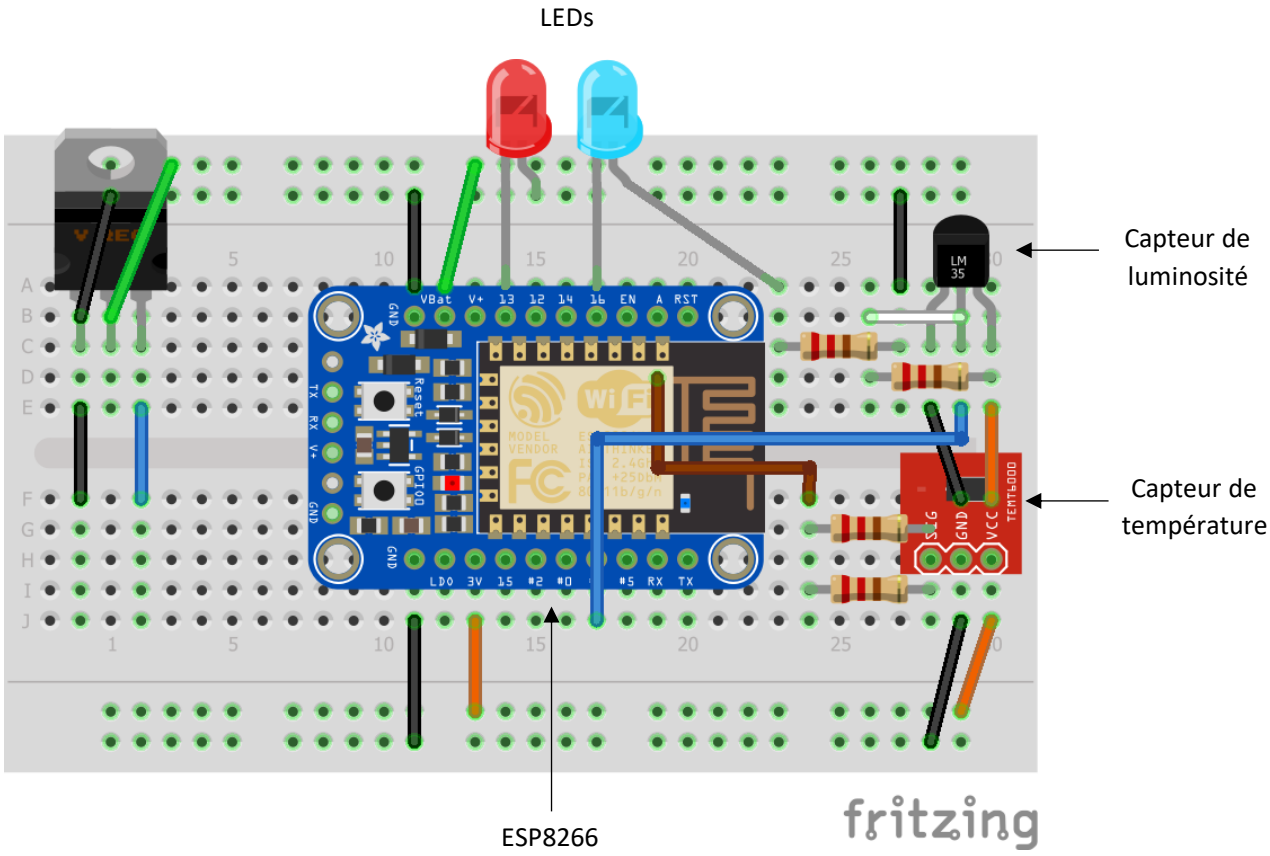


Outil de diagnostic optique

2) Montage électronique du dispositif de lecture



3) Montage électronique des capteurs avec microcontrôleur



4) Codes ARDUINO UNO et ESP8266

Cf les fichiers .ino dans le dossier fournis avec le rapport.