Arroseur automatique de plantes

Informations

- Sean RAMS
- ramsean2001@gmail.com
- Master informatique : SESI

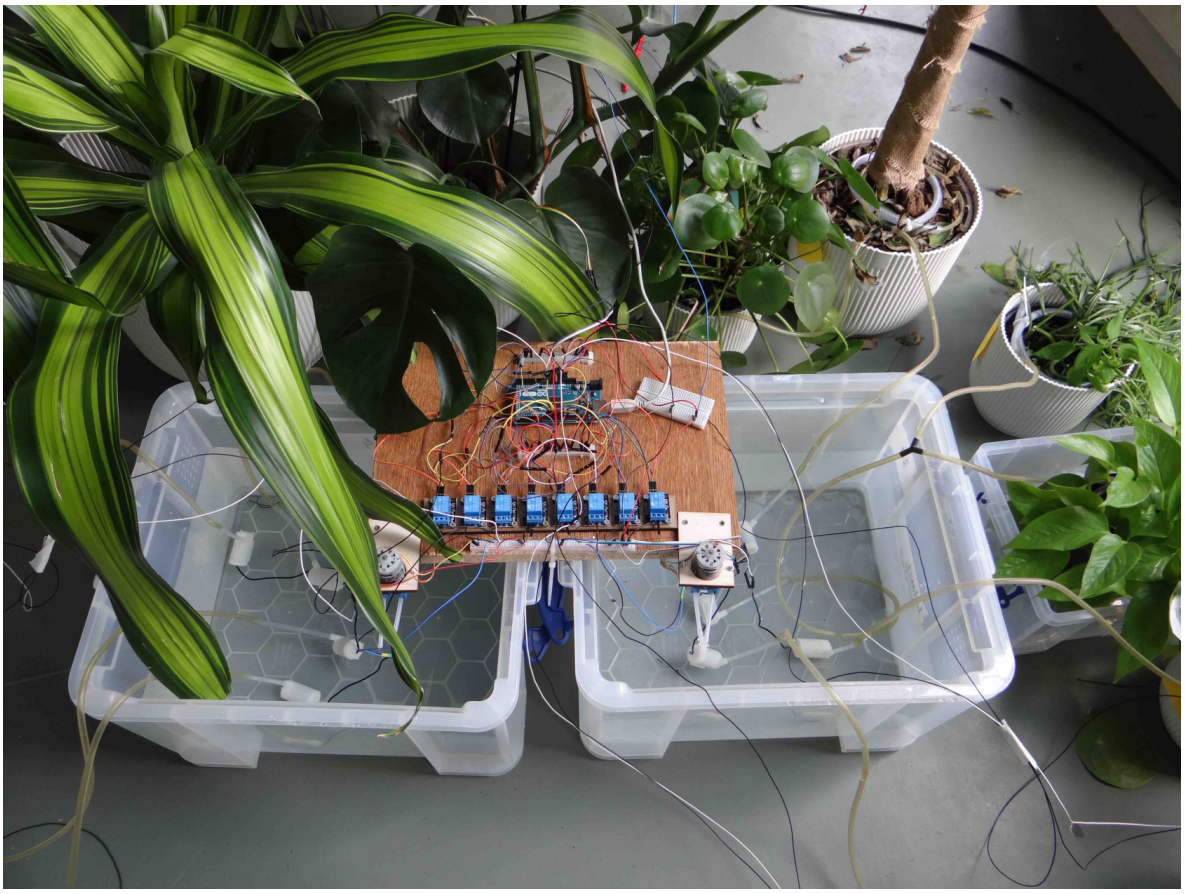
- S02/06/2023 - 20/06/2023

Contexte

Afin de préserver les plantes du fablab durant, je me suis vu attribué la création d'arroseur automatique pour les diverses plantes du fablab.

Objectifs

Réaliser des arroseurs automatique pour diverses plantes.



Matériel

- Arduino mega.
- Pompes immergé (nombre dépendant du nombre de plante).
- Capteurs d'humidité sol (nombre dépendant du nombre de plante).
- Relais (nombre dépendant du nombre de plante).
- Bac étanche (ce qui permette de garde les projet font l'affaire).
- Planche de bois.
- Alimentation 6V, 3A (l'Ampérage n'est pas certains).

Machines utilisées

Aucune

Construction

(Fichiers, photos, code, explications, paramètres d'usinage, photos, captures d'écran...)

Avant tout de chose il faut comprendre que la difficulté du projet n'est pas sur le fait d'arroser une plante mais la mise en échelle sur un groupe d'une quinzaine de plante. Je vous revois vers ce projet ci https://wiki.fablab.sorbonne-universite.fr/BookStack/books/projets-due-2022-

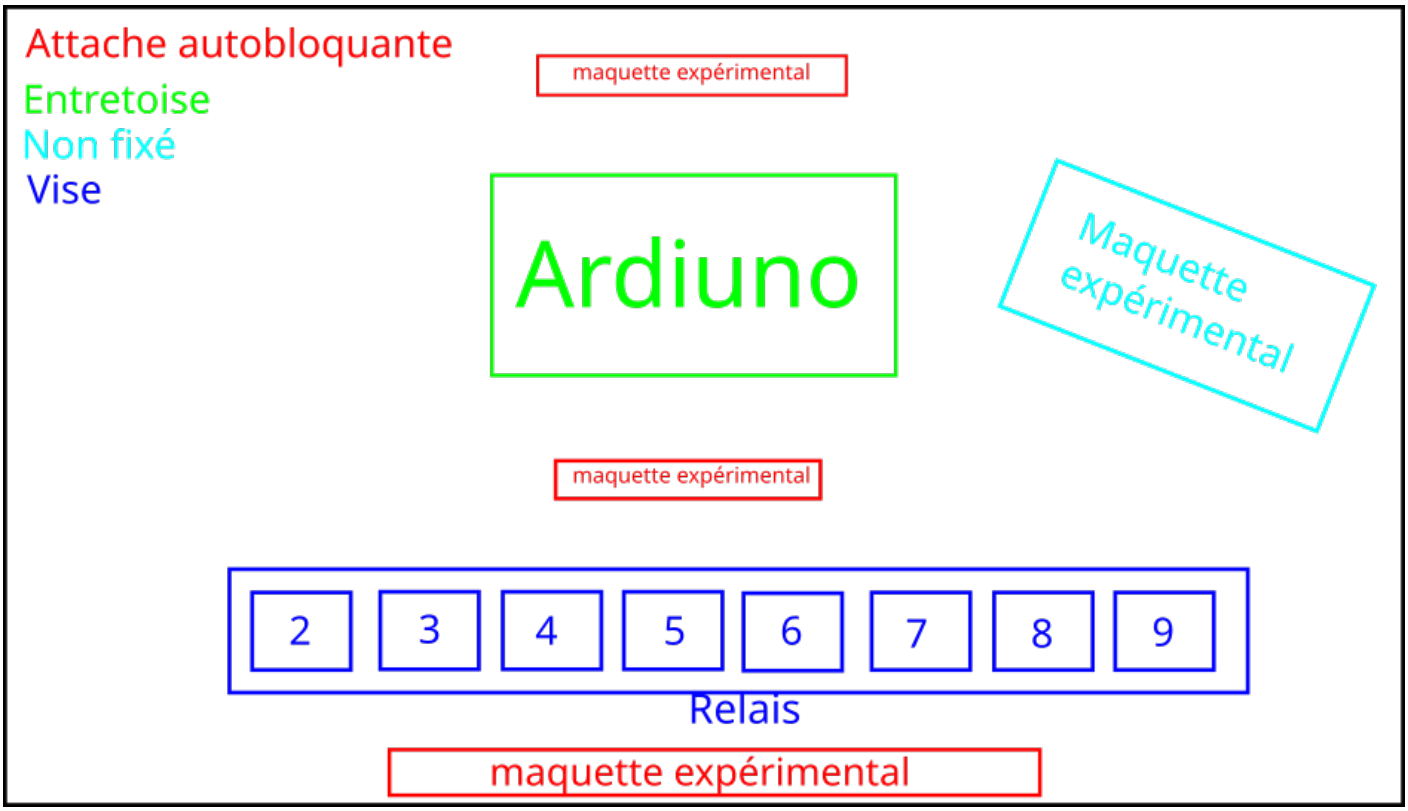
2023/page/groupe-a3-apar qui expliquera le principe d'arrosage d'une plante .

Étape 0 (optionnel)

Dans le cas d'une manque de capteur et/ou de pompe, il faut regrouper les plante en différents groupe. Le critère de sélection va être le volume de la terre à arroser mais aussi la consommation des plantes. La consommation peuvent être mesurer en utilisant les capteur d'humidité et en notant dans un tableau l'évolution de l'humidité du sol sur un période.

Étape 2

Si vous reprenez la planche utiliser pour le fablab, il faut suivre ce schéma : (le numéro des relais serviront dans le code)



Étape 3

Toujours en ce référant aux schéma au-dessus, Il faut établir les différentes connexions. Les maquettes expérimentaux servent uniquement à alimenter les différents composants (Arduino, capteurs, relais et pompes). Tout les composants sont alimentés par la même alimentation (un des générateurs du coin électronique).

Journal de bord

02/06 Premier pas dans le projet:

```
int sensorPin = A0;
int sensorValue = 0;
int PinR = 7;
int PinV = 8;
int PinB = 4;



void setup() {
   Serial.begin(9600);
   pinMode(PinR,OUTPUT);
   pinMode(PinV,OUTPUT);
   pinMode(PinB,OUTPUT);
   }
void loop() {
   // read the value from the sensor:
   sensorValue = analogRead(sensorPin);
   Serial.print("Moisture = " );
   Serial.println(sensorValue);
   if (sensorValue < 300){
     digitalWrite(PinV,HIGH);
     digitalWrite(PinR,LOW);
   }
   else {
     digitalWrite(PinR,HIGH);
     digitalWrite(PinV,LOW);
   }
   delay(1000);
}
```

15/06 Test d'utilisation de la pompe.

```
int sensorPin = A0;
int sensorValue = 0;
int PinR = 7;
int PinV = 8;
int PinB = 4;
int pompe = 2;
```

```
int serial = 101;

void setup() {
    Serial.begin(9600);
    pinMode(PinR,OUTPUT);
    pinMode(PinV,OUTPUT);
    pinMode(PinB,OUTPUT);
    pinMode(pompe,OUTPUT);
    }
void loop() {
    // read the value from the sensor:
   /* sensorValue = analogRead(sensorPin);
    Serial.print("Moisture = " );
    Serial.println(sensorValue);
    if (sensorValue < 300){
      digitalWrite(PinV,HIGH);
      digitalWrite(PinR,LOW);
    }
    else {
      digitalWrite(PinR,HIGH);
      digitalWrite(PinV,LOW);
    }*/

    if (serial >'5' ){
      digitalWrite(pompe,HIGH);
    }else{
      digitalWrite(pompe,LOW);
    }
    if(Serial.available() > 0)  {
      serial= Serial.read();
    }
    Serial.println(serial);
    delay(1000);
}
```

L'idée est de combiner la pompe et le capteur pour alimenter la plante lorsque la terre est sèche.

Code fonctionnel pour arroser une plante en fonction de l'humidité de sa terre.

```
int sensorPin = A0;
int sensorValue = 0;
int PinR = 7;
int PinV = 8;
int PinB = 4;
int pompe = 2;
void setup() {
   Serial.begin(9600);
   pinMode(PinR,OUTPUT);
   pinMode(PinV,OUTPUT);
   pinMode(PinB,OUTPUT);
   pinMode(pompe,OUTPUT);

    digitalWrite(pompe,LOW);
   }
void loop() {
   // read the value from the sensor:
   sensorValue = analogRead(sensorPin);
   Serial.print("Moisture = " );
   Serial.println(sensorValue);
   if (sensorValue > 300){

     digitalWrite(PinV,HIGH);
     digitalWrite(PinR,LOW);
   }
   else {

     digitalWrite(PinR,HIGH);
     digitalWrite(PinV,LOW);
     digitalWrite(pompe,HIGH);
     delay(3000);

   }

   digitalWrite(pompe,LOW);
   delay(3000);
}
```

Début de la programmation en tache distincte pour essayer d'utiliser plusieurs capteur/pompe sur une Arduino.

```
/*
int sensorPin = A0;
int sensorValue = 0;
int PinR = 7;
int PinV = 8;
int PinB = 4;
int pompe = 2;
*/

#define MAX_WAIT_FOR_TIMER 4

unsigned int waitFor(int timer, unsigned long period){
  static unsigned long waitForTimer[MAX_WAIT_FOR_TIMER];  // il y a autant de timers que de tâches périodiques
  unsigned long newTime = micros() / period;              // numéro de la période modulo 2^32
  int delta = newTime - waitForTimer[timer];              // delta entre la période courante et celle enregistrée
  if ( delta < 0 ) delta = 1 + newTime;                   // en cas de dépassement du nombre de périodes possibles sur
2^32
  if ( delta ) waitForTimer[timer] = newTime;             // enregistrement du nouveau numéro de période
  return delta;
}

enum {EMPTY, FULL};

struct mailbox_s {
  int state;
  int val;
};

struct mailbox_s mb = {.state = EMPTY};

//tache pour la lecteur d'un des capteurs.



struct CaptHum{
  int timer;
  unsigned long period;
  int pin;
}

void setup_hum( struct CaptHum * ctx,struct mailbox_s * mb, int timer, unsigned long period, byte pin){
```

```
  ctx->timer = timer;
  ctx->period = period;
  ctx->pin = pin;
  pinMode(ctx->pin,INPUT);
}

void loop_lum( struct CaptLum * ctx,struct mailbox_s * mb) {
  if (!waitFor(ctx->timer, ctx->period)) return;          // sort s'il y a moins d'une période écoulée
  if (mb->state != EMPTY) return;
  mb->val = analogRead(ctx->pin);
  mb->state=FULL;
}

//Tache d'activation de l'arosage

struct Active{
  int timer;
  unsigned long period;
  int pinpompe;
  int pinledR;
  int pinledV;
}

void setup_active( struct CaptHum * ctx,struct mailbox_s * mb, int timer, unsigned long period, byte pin){
  ctx->timer = timer;
  ctx->period = period;
  ctx->pin = pin;
  pinMode(ctx->pin,INPUT);
}

void loop_active( struct CaptLum * ctx,struct mailbox_s * mb) {
  if (!waitFor(ctx->timer, ctx->period)) return;          // sort s'il y a moins d'une période écoulée
  if (mb->state != EMPTY) return;
  mb->val = analogRead(ctx->pin);
  mb->state=FULL;
}

//tache qui desactive la pompe

struct Desactive{
```

```
  int timer;
  unsigned long period;
  int pinpompe;
  int pinledR;
  int pinledV;
}

void setup_active( struct CaptHum * ctx,struct mailbox_s * mb, int timer, unsigned long period, byte pin){
  ctx->timer = timer;
  ctx->period = period;
  ctx->pin = pin;
  pinMode(ctx->pin,INPUT);
}

void loop_active( struct CaptLum * ctx,struct mailbox_s * mb) {
  if (!waitFor(ctx->timer, ctx->period)) return;          // sort s'il y a moins d'une période écoulée
  if (mb->state != EMPTY) return;
  mb->val = analogRead(ctx->pin);
  mb->state=FULL;
}



void setup() {
    /*
    Serial.begin(9600);
    pinMode(PinR,OUTPUT);
    pinMode(PinV,OUTPUT);
    pinMode(PinB,OUTPUT);
    pinMode(pompe,OUTPUT);
    digitalWrite(pompe,LOW);
    */
    }
void loop() {
    /*
    // read the value from the sensor:
    sensorValue = analogRead(sensorPin);
    Serial.print("Moisture = " );
    Serial.println(sensorValue);
    if (sensorValue > 300){
```

```
    digitalWrite(PinV,HIGH);
    digitalWrite(PinR,LOW);
   }
   else {

    digitalWrite(PinR,HIGH);
    digitalWrite(PinV,LOW);
    digitalWrite(pompe,HIGH);
    delay(3000);

   }

   digitalWrite(pompe,LOW);
   delay(3000);
   */
}
```

16/06 : Code pouvant utiliser plusieurs capteur et pompe sur un seul arduino (pour seulement deux dans ce code)

```
/*
Ne pas modifier cette partie du code
*/

#define MAX_WAIT_FOR_TIMER 16

unsigned int waitFor(int timer, unsigned long period){
 static unsigned long waitForTimer[MAX_WAIT_FOR_TIMER];  // il y a autant de timers que de tâches périodiques
 unsigned long newTime = millis() / period;              // numéro de la période modulo 2^32
 int delta = newTime - waitForTimer[timer];              // delta entre la période courante et celle enregistrée
 if ( delta < 0 ) delta = 1 + newTime;                   // en cas de dépassement du nombre de périodes possibles sur
2^32
 if ( delta ) waitForTimer[timer] = newTime;             // enregistrement du nouveau numéro de période
 return delta;
}

enum {EMPTY, FULL};
//Structure mail box servant au tache à communiquer.
struct mailbox_capteur {
 int state;
```

```
  int val;
};

struct mailbox_timer {
  int state;
  unsigned long time_stop;
};

//tache pour la lecteur d'un des capteurs.

struct CaptHum{
  int timer;
  unsigned long period;
  int pin;
};

void setup_hum( struct CaptHum * ctx,struct mailbox_capteur * mb_cap, int timer, unsigned long period, int
pin){
  /*
  Fonction permetant d'initialiser une tache qui va périodiquement effectuer des messures d'humidité.

    ctx : Pointeur vers la suctruture qui va être initialiser
    mb_cap: pointeur vers la mailbox permetant d'enregistrer la messure du capteur et la transmetre à la tache
d'activation de la pompe
    timer : Identifier UNIQUE permetant à wait timer d'excuter la messure périodiquement.
    period : Indique le temps d'attente minimal entre deux messure (malereusement en unité inconnue mais
suposser être des millisecondes)
    pin : Pin ANALOGIQUE utiliser pour prendre la messure.
  */
  //Initialisation des timer et period pour le waitfor permetant d'organiser les taches.
  ctx->timer = timer;
  ctx->period = period;

  ctx->pin = pin;
}

void loop_hum( struct CaptHum * ctx,struct mailbox_capteur * mb_cap, struct mailbox_timer *mb_time ) {
    /*
  Fonction permetant effectuer des messures d'humidité périodiquement sur un capteur donné.
```

```
    ctx : Pointeur vers la suctruture qui a été initialiser
    mb_cap : pointeur vers la mailbox permetant d'enregistrer la messure du capteur et la transmetre à la tache d'activation de la pompe
    mb_time : pointeur vers la mailbox permetant d'ârréter la pompe, présente pour eviter de prendre des messure lorsque la pompe est active.
  */
  if (mb_cap->state != EMPTY) return;
  if (mb_time->state != EMPTY) return;
  if (!waitFor(ctx->timer, ctx->period)) return;          // sort s'il y a moins d'une période écoulée


  mb_cap->val = analogRead(ctx->pin);

  Serial.print(String("Moisture ")+String(ctx->timer)+String(" = ") );
  Serial.println(mb_cap->val);

  mb_cap->state=FULL;
}

//Tache d'activation de l'arosage

struct Active{
  int pinpompe;
  int pinledR;
  int pinledV;
  int seuil;
  int time_active;
};

void setup_active( struct Active * ctx, int pinpompe, int pinledR, int pinledV, int seuil, unsigned long time_active){
   /*
  Fonction permetant d'initialiser une tache qui va si la meussure reçu est trop faible activer la pompe.

    ctx : Pointeur vers la suctruture qui va être initialiser
    pinpompe : Pin controlant la pompe.
    pinledR : Pin controlant la  led rouge.
    pinledV : Pin controlant la led vert.
    seuil : Seuil pour lequel la terre est considére comme trop séche.
    time_active : temps d'activation de la pompe en milliseconde (environ)
```

```
  */
  //Initialisation des pin de la pompe, des led rouge et vert pour une tache active
  ctx->pinpompe = pinpompe;
  ctx->pinledR = pinledR;
  ctx->pinledR = pinledR;

  //Setup des différente pin utiliser.
  pinMode(ctx->pinpompe,OUTPUT);
  pinMode(ctx->pinledR,OUTPUT);
  pinMode(ctx->pinledV,OUTPUT);

  //Initialisation des variable pour activer la pompe et sa durée
  ctx->seuil = seuil;
  ctx->time_active = time_active;

  //Désactive la pompe dans son setup
  digitalWrite(ctx->pinledR,LOW);
  digitalWrite(ctx->pinledV,HIGH);
  digitalWrite(ctx->pinpompe,LOW);
}

void loop_active( struct Active * ctx, struct mailbox_capteur * mb_cap, struct mailbox_timer * mb_time) {
    /*
  Fonction permetant d'activer la pompe pour un temps donné.

    ctx : Pointeur vers la suctruture de la tache qui a été initialiser
    mb_cap : pointeur vers la mailbox permetant d'enregistrer la messure du capteur et la transmetre à la tache d'activation de la pompe
   mb_time : pointeur vers la mailbox permetant d'ârréter la pompe, présente pour eviter de prendre des messure lorsque la pompe est active.
  */
  //Test des différentes condiction avant d'activer la pompe
  if (mb_cap->state != FULL) return;
  if (mb_cap->val <= ctx->seuil ) {
    //Alumage de la led rouge et de la pompe (+ étient la led vert) pour signaler que la plante est arroser
    digitalWrite(ctx->pinledR,HIGH);
    digitalWrite(ctx->pinledV,LOW);
    digitalWrite(ctx->pinpompe,HIGH);

    //Initialise la mailbox pour étiendre la pompe
```

```
    mb_time->time_stop =  millis() + ctx->time_active;
    mb_time->state=FULL;
    Serial.println(String("Pompe active ")+ctx->pinpompe);
  }
  mb_cap->state = EMPTY;
}

//tache qui desactive la pompe

struct Desactive{
  int pinpompe;
  int pinledR;
  int pinledV;
};

void setup_desactive( struct Desactive * ctx, int pinpompe, int pinledR, int pinledV){
   /*
  Fonction permetant d'initialiser une tache qui va si la meussure reçu est trop faible activer la pompe.

    ctx : Pointeur vers la suctruture qui va être initialiser
    pinpompe : Pin controlant la pompe. (doit être la même que celle de la tache d'activation de la pompe associé)
    pinledR : Pin controlant la  led rouge.(doit être la même que celle de la tache d'activation de la pompe associé)
    pinledV : Pin controlant la led vert.(doit être la même que celle de la tache d'activation de la pompe associé)
  */
  ctx->pinpompe = pinpompe;
  ctx->pinledR = pinledR;
  ctx->pinledV = pinledV;

  pinMode(ctx->pinpompe,OUTPUT);
  pinMode(ctx->pinledR,OUTPUT);
  pinMode(ctx->pinledV,OUTPUT);



  digitalWrite(ctx->pinledR,LOW);
  digitalWrite(ctx->pinledV,HIGH);
  digitalWrite(ctx->pinpompe,LOW);
}
```

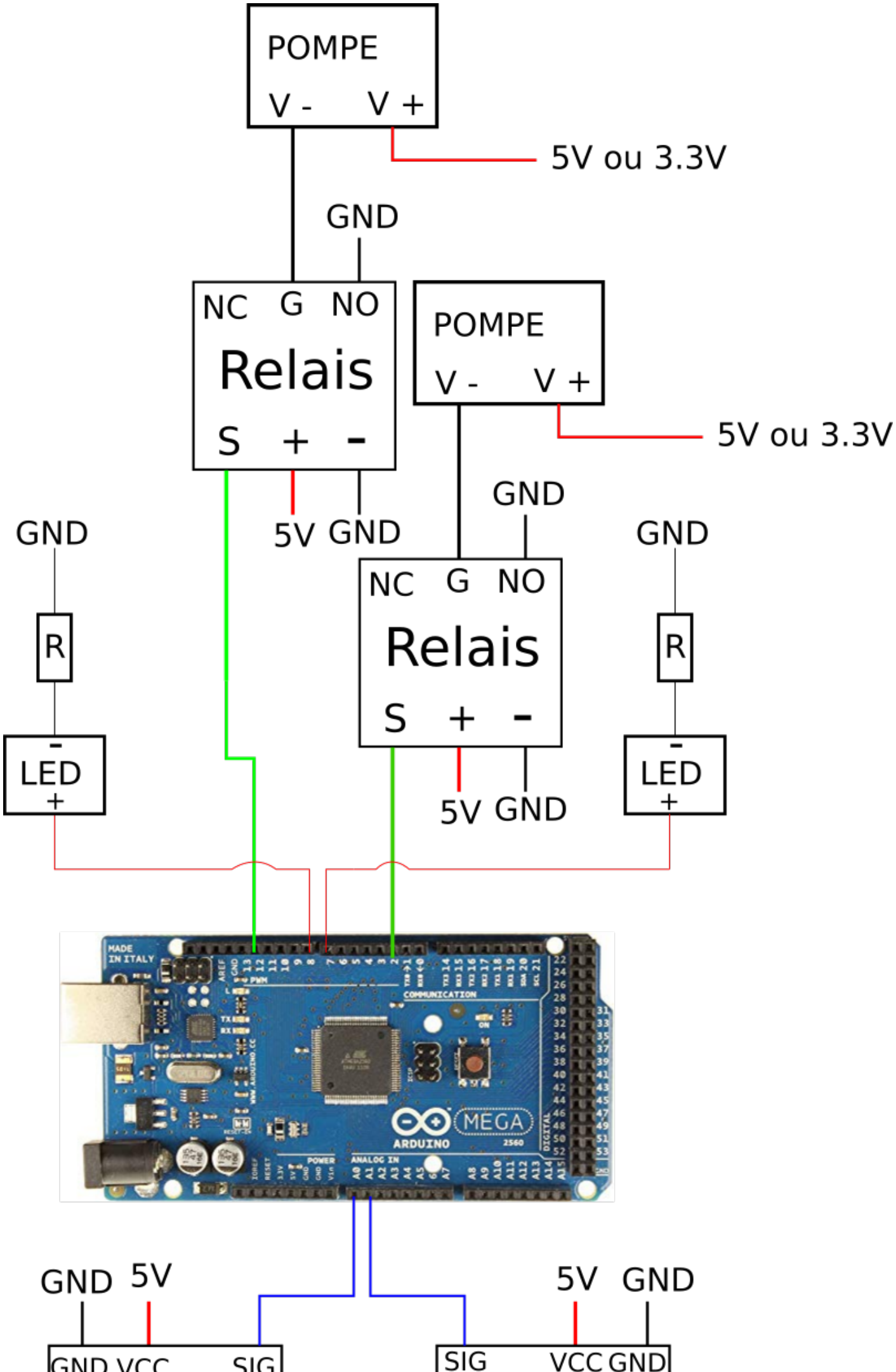
```
void loop_desactive( struct Desactive * ctx, struct mailbox_timer * mb_time) {
  /*
  Fonction permetant de desactiver la pompe apres un temps donné par la mailbox timer.

    ctx : Pointeur vers la suctruture de la tache qui a été initialiser
   mb_time : pointeur vers la mailbox permetant d'ârréter la pompe, présente pour eviter de prendre des messure lorsque la pompe est active.
  */
  Serial.println(String("mb time desa = ")+mb_time->state+" "+mb_time->time_stop);
  Serial.println(String("millis = ")+millis());
  if (mb_time->state != FULL) return;
  if( millis()>= mb_time->time_stop){
    digitalWrite(ctx->pinledR,LOW);
    digitalWrite(ctx->pinledV,HIGH);
    digitalWrite(ctx->pinpompe,LOW);
    Serial.println("Pompe desactive "+ctx->pinpompe);
    mb_time->state = EMPTY;
  }
}

// Declaration des tache et des mailbox (si ajout de capteur et/ou pompe à faire à partir d'ici)

//mail box
struct mailbox_capteur mb_cap0 = {.state = EMPTY};
struct mailbox_timer mb_time0 = {.state = EMPTY};

struct mailbox_capteur mb_cap1 = {.state = EMPTY};
struct mailbox_timer mb_time1 = {.state = EMPTY};

//tache
struct CaptHum Hum0;
struct Active acti0;
struct Desactive desa0;

struct CaptHum Hum1;
struct Active acti1;
struct Desactive desa1;

void setup() {
  Serial.begin(9600);
```

```
  setup_hum(&Hum0, &mb_cap0, 0, 1000, A0);
  setup_active(&acti0, 2, 7, 8, 400, 2000);
  setup_desactive(&desa0, 2, 7, 8);

  setup_hum(&Hum1, &mb_cap0, 1, 1000, A1);
  setup_active(&acti1, 12, 13, 22, 400, 2000);
  setup_desactive(&desa1, 12, 13, 22);
}
void loop() {
  loop_hum(&Hum0, &mb_cap0, &mb_time0);
  loop_active(&acti0, &mb_cap0, &mb_time0);
  loop_desactive(&desa0, &mb_time0);



  loop_hum(&Hum1, &mb_cap1, &mb_time1);
  loop_active(&acti1, &mb_cap1, &mb_time1);
  loop_desactive(&desa1, &mb_time1);
}
```

19/06:

Pour ajouter des nouveaux capteurs, il faut créé deux nouvelles mail box de chaque type (mailbox_capteur et mailbox_timer), une nouvelle tâche de chaque type (CaptHum, Active et Desactive).Il faut ensuite les setups de la même manière que sur le code déjà présent chaque variable est expliqué dans les commentaire de chaque fonction. Il est important de d'avoir en commun sur active et desactive les pin pour la pompe et led. Les leds servent pour le débug est ne sont pas nécessaire à brancher. Puis il faut ajouter les fonction loop avec les tache et les mail créer.

Schéma des connexions pour faire fonctionner le code au dessus (sans certaine led).

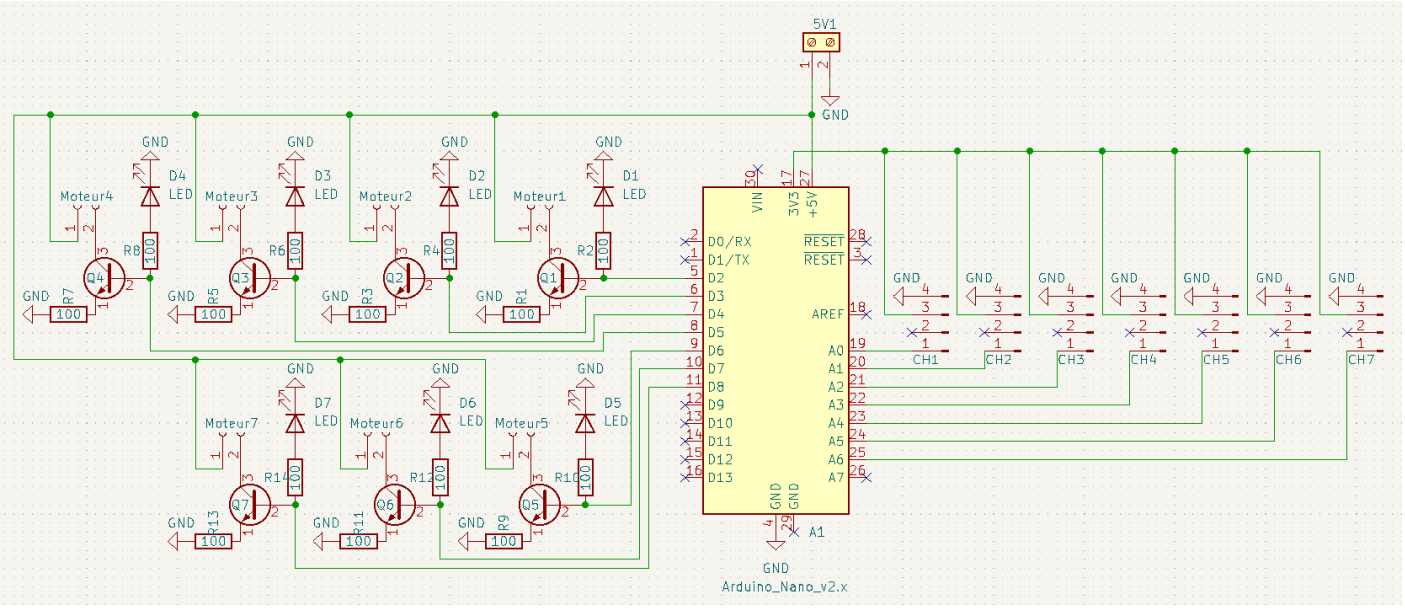

POMPE
V - V +
5V ou 3.3V
GND
NC G NO
Relais
S + -
5V GND
POMPE
V - V +
5V ou 3.3V
GND
GND
R
LED
+ -
NC G NO
Relais
S + -
5V GND
GND
R
LED
+ -
MADE IN ITALY
COMMUNICATION
MEGA
ARDUINO
2560
POWER
ANALOG IN
DIGITAL
GND 5V
GND VCC SIG
5V GND
SIG VCC GND

Il faut noté que le capteur à 4 pin cependant la pin non labéliser (entre VCC et SIG) ne sert a rien et donc nécessite aucune connexion.

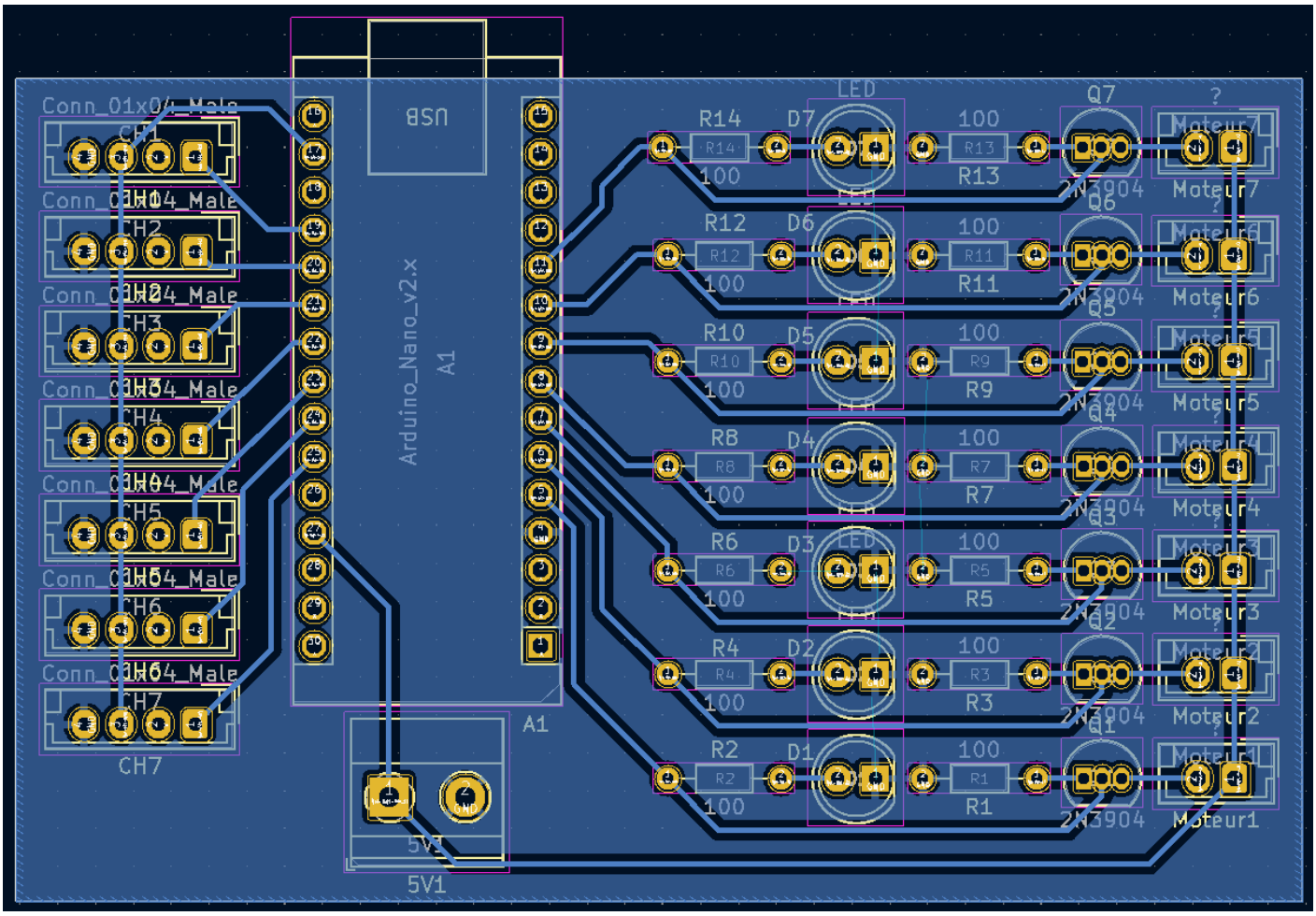
23/06:

Simplification électronique, la commande des moteurs a été grandement simplifié. Au lieu d'utilisé des relais pour commander les moteurs, nous utilisons maintenant des transistors (comme switch électronique). Les moteurs son alimenté indépendamment de l'Arduino nano.

Nouveau schéma électronique :



Et dessin de la PCB :



Nous utilisons 7 capteurs d'humidités et 7 pompes (le nombre max dispo au fablab).

Le système de signalement lors du fonctionnement a été modifié. Au lieu de 2 LED (Verte et Rouge) pour indiquer que rien ne se passe ou il y a besoin d'arrosage, cela est remplacé par une LED sur chaque moteur, si elle est allumé alors il y a arrosage.

Modification du code pour correspondre au besoin.

03/07

Par manque de compréhension du PCB, nous continuons le projet avec la première solution.

Revision #1
Created 19 May 2024 11:22:52 by Ouerfili Chaima
Updated 19 May 2024 11:22:52 by Ouerfili Chaima