académie Lyon

CITE SCOLAIRE de TARARE

RhôneAlpes Région

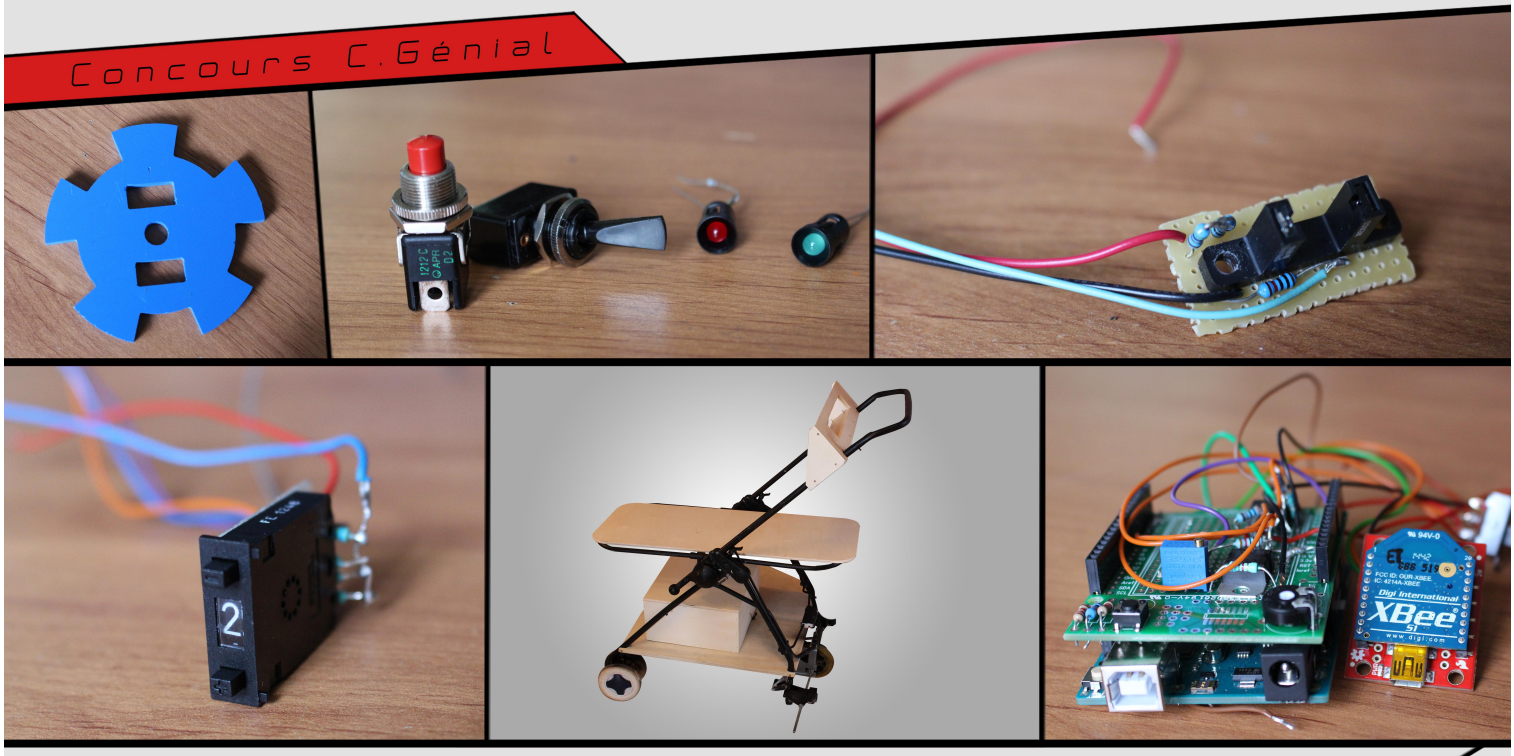
Lyon 1

C.GENIAL Fondation pour la culture scientifique et technique

Sciences à l'École www.sciencesalecole.org

Poussette Universelle

Concours C.Génial



Lycée René Cassin TARARE

2015-2016

CLAVIER Mathieu
DÉMOLLIÈRE Côme
MOLIÈRE Gatien
POLLART Valentin

Table des matières

Présentation du groupe

- Établissement : Lycée René Cassin, Tarare (69)

Nous sommes un groupe de quatre élèves de terminale scientifique option science de l'ingénieur :

- CLAVIER Mathieu
- DÉMOLLIÈRE Côme
- MOLIÈRE Gatien
- POLLART Valentin

Professeur coordinateur : Mustapha Errami
Professeur de physique encadrant : Mustapha Errami
Professeurs collaborateurs : Christian Vallon, François Pinault

Partenaire : M. Roger DUFFAIT, Université Claude Bernard de Lyon

Introduction

Notre ville, en l'occurrence Tarare est légèrement montagneuse, des montées à différentes pentes constituent une réelle difficulté pour les jeunes mamans et papas pour remonter la poussette de leur bébé.

Comment améliorer une poussette afin d'accroître le confort de l'enfant et de l'utilisateur ?

Un objet utile !

Pour répondre à cette problématique, nous avons créé une poussette semi-automatique.

La poussette permettra à l'utilisateur de choisir sa vitesse de fonctionnement grâce à un sélectionneur. Celle-ci s'accompagne d'une fonction permettant la modulation de la puissance suivant le poids et la pente, afin de garder une allure constante. De plus une fonction est rajoutée sur la poussette, afin de contrôler l'inclinaison du berceau en fonction de la pente. L'enfant ne se trouvera donc jamais en danger à cause d'une inclinaison trop importante, il gardera toujours une inclinaison à l'horizontale. La poussette est sécurisée par une connexion via ondes radio qui permet de contrôler en permanence la distance entre la poussette et le parent. Ainsi celle-ci s'arrête lorsque la distance est trop grande et que l'enfant pourrait se retrouver isolé...

Saisie et lecture de la vitesse

Dans le cadre de la poussette automatique, la poussette doit avancer à l'allure choisie par l'utilisateur. Ce dernier aura le choix entre 8 vitesses : de 0 à 7 km/h

Pour cela, il aura à sa disposition un sélectionneur avec lequel il pourra choisir l'une des huit vitesses proposées.

Le programme est découpé en trois parties :

=> La première partie est la consigne saisie par l'utilisateur.

=> La seconde partie est la mesure de la vitesse actuelle de la poussette.

Nous avons alors découpé une petite roue dentée en plastique qui sera fixée à la roue motrice de la poussette. Cette roue passera devant un capteur photoélectrique. Tous les 10 changements d'état du capteur (5* 2) la roue aura fait un tour complet, ce qui nous permettra de déterminer la distance parcourue.

⇨ Grâce à cette formule : $\frac{2\pi * 6.7 * 10^2}{(T2-T1) * 3600} * \frac{255}{7}$

Cette formule permet de détecter la vitesse actuelle de la poussette et de la convertir sous 8 bits.

Enfin, le sélectionneur de vitesse est un compteur binaire il fonctionne ainsi :

Vitesse demandée	0	1	2	3	4	5	6	7
Broches fermée	-	1	2	1 – 2	4	1 – 4	2 – 4	1 – 2 – 4

Afin de pouvoir utiliser ces données, nous avons converti les valeurs données par les broches en valeur décimale. Pour cela, nous avons additionné les numéros des broches, on les a multiplié par 255 et divisé par 7 afin de les convertir sous 8 bits. Grâce à cela on a pu définir une consigne pour chaque vitesse sélectionnée.

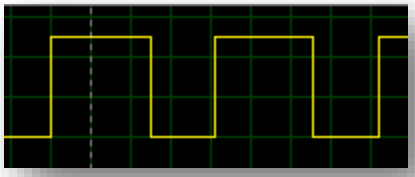
```
if (digitalRead(A2)>50){
  e1 = 1;
}
  else {
    e1 = 0;
  }
if (digitalRead(A3)>50){
  e2 = 2;
}
  else {
   e2 = 0;
  }
if (digitalRead(A4)>50){
  e4 = 4;
}
  else {
   e4 = 0;
  }

if (digitalRead(A5)>50){
  e8 = 8;
}
  else {
   e8 = 0;
  }

byte selectionneur = e1 + e2 + e4 + e8;
```

Régulation (asservissement)

Notre poussette motorisée permettra à l'utilisateur de choisir la vitesse à laquelle il souhaite que la poussette se déplace. Cette fonctionnalité nous semble très importante puisqu'elle permet à la poussette de s'adapter à tout type d'utilisateurs. Pour se faire nous avons choisi d'avoir recours à un asservissement car c'est un procédé qui fonctionne très bien avec l'arduino car il possède des sorties PWM (Pulse Width modulation).

Une vitesse stabilisée

(Sortie PWM: c'est une sortie permettant d'envoyer une valeur sous 8 bits afin d'établir, grâce à des signaux continus, une moyenne sur une certaine durée. Par exemple si la valeur de la variable envoyée à la sortie est 127 elle permettra d'alimenter le composant relié qu'à demi-régime).



L'asservissement est un algorithme qui permet de stabiliser un système (ici le moteur) par rapport à une consigne en le corrigeant en permanence. Autrement dit, cela permet au moteur de donner le couple correspondant à la vitesse souhaitée.

Le principe est alors de corriger en permanence le couple du moteur, en l'alimentant plus ou moins, afin de se stabiliser aux alentours de la vitesse souhaitée.

Pour se faire, il faut alors programmer notre arduino. Comme vu précédemment, la vitesse actuelle de la poussette est déterminée par un capteur, et la vitesse souhaitée est directement saisie par l'utilisateur sur un sélectionneur. Il s'agira alors de comparer ces deux valeurs et d'alimenter le moteur en fonction de leurs différences.

Une consigne à respecter

On se servira des variables déterminées précédemment, soit la variable «consigne» qui est la vitesse souhaitée par l'utilisateur et la variable «vitesse» qui est la vitesse actuelle de la poussette déterminée par le capteur.

Il s'agira alors de comparer ces deux variables sous 8 bits afin d'établir la correction à apporter au moteur. On établit donc une nouvelle variable «result», qui sera la variable sous 8 bits à envoyer à la sortie PWM, qui est le résultat de la différence entre la consigne et la vitesse. Attention cependant, des facteurs extérieurs peuvent perturber cette différence, on pose alors un réel k permettant de stabiliser la valeur de «result».

On obtient alors l'équation : **result = (consigne-vitesse)*k**

```
// Côme

  int  result = (consigne-vitesse)*k; //détermination du résultat

  if (result>255){ //vitesse actuelle trop faible, allimenter le moteur
   result=255;
  }
  if (result<=0){ //vitesse actuelle trop élevée, stopper l'allimentation
   result=0;
  }

  if (arretmoteur == 0) {
 analogWrite (moteurP, result);
  }
```

La vitesse de la poussette va alors osciller en permanence autour de la vitesse souhaitée par l'utilisateur. Cette fonctionnalité permet à la poussette de se déplacer toujours à une vitesse très proche de celle demandée peu importe la qualité de la route ou l'inclinaison de la pente sur laquelle se situe la poussette. Cependant attention, sur une inclinaison de pente trop importante le moteur de

la poussette ne sera plus suffisamment « puissant » pour respecter la vitesse souhaitée. Il aurait fallu alors un moteur plus puissant.

Contrôle de la capacité d'entraînement de notre transmission mécanique

Nous nous sommes alors intéressé à savoir quel était le couple de notre moteur afin de déterminer la limite de notre asservissement en fonction de l'amplitude d'une côte. Notre moteur est issu d'une trottinette électrique 24V permettant de se déplacer à une vitesse de 12km/h. De plus, nous savons qu'il possède une puissance nominale de 100W (Pn). On cherche à déterminer la force de traction du moteur.

Vitesse de fonctionnement : V = 12km/h = 3.3m/s

Rayon de la roue : R_{roue} = 0.067m

$\acute{\omega}_{roue} = \frac{V}{Rroue} = \frac{3.3}{0.067} = 49.3$ rad/s

Sachant que la réduction de la poulie courroie est de 16/88 on a alors :

$\acute{\omega}_{moteur} = 49.3 * \frac{88}{16} = 270.9$ rad/s

Or $C_{moteur} = \frac{Pn}{\Omega moteur} = \frac{100}{270.9r} = 0.37$ N.m

Grâce a une expérience réalisée en salle de laboratoire on détermine que la poussette avec le bébé possède une masse de 15 kg et que la force de traction nécéssite 6N, sur le bitume cette force nécessite 8N. Cependant afin de prendre en compte des facteurs extérieurs divers on établie que la force de traction nécessite 10N.

On a alors $C_{roue} = 0.37 * \frac{88}{16} = 2.03$ N.m

Pour prendre en compte un rendement de la transmission poulie courroie et du guidage en rotation de la roue, on prendra η = 0.8

On a alors $C_{roue} = 2.03 * 0.8 = 1.624$ N.m

Donc $F_{traction} = \frac{1.624}{0.067} - 10 = 14.24$ N > 0 donc elle est supérieure à la force de poussé nécessaire.

On détermine alors la déclivité, soit pour quel angle de côte α le moteur n'est plus capable de répondre au besoin de l'utilisateur.

On prend P = m.g ; avec g = 10 m/s²

$\sin \alpha = \frac{Ftraction}{P} = \frac{14.24}{15*10} = 0.14$ donc α = 5.41 °

Grâce à ces calculs nous pouvons en déduire qu'à partir d'un angle de côte de 5.41° et avec une puissance nominale de 100 W, notre moteur passera en surcharge et ne sera plus capable de répondre correctement au besoin de l'utilisateur. Il manque alors un réducteur pour diminuer la vitesse qui est un peu trop élevée et ainsi augmenter le couple pour permettre un fonctionnement du moteur dans des côtes d'amplitude plus importante. Néanmoins il servira toujours d'assistance au déplacement lors d'une côte raide.

Une liaison radio

Après avoir motorisé la poussette, il nous semblait invraisemblable de ne pas proposer une sécurité pour la poussette. Nous avons donc souhaité détecter en permanence la distance entre la poussette et l'utilisateur (celui qui dirige la poussette).

Après de nombreuses réflexions en ayant essayé le bluetooth ou encore la Wi-Fi, nous sommes partis sur les ondes radios pour détecter cette distance.

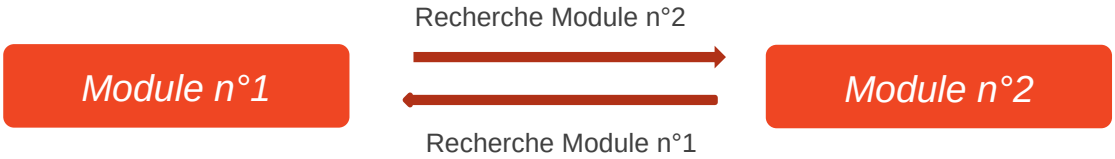
Dans le commerce nous avons pris connaissance des produits XBee. Ce sont des modules de communication sans fil. Nous utilisons le modèle 802.15.4.

Des ondes radios pour communiquer

Les principales caractéristiques du XBee :

– fréquence porteuse : 2.4Ghz
– portées variées XBee 1 et 2 (10 - 100m)
– faible débit : 250kbps
– faible consommation : 3.3V @ 50mA

• Il a fallu tout d'abord configurer les deux petits modules afin que l'un détecte l'autre. Pour cela nous utilisons un logiciel nommé X-CTU, nous permettant de régler l'adresse de chaque XBee ainsi que l'adresse de l'XBee qu'il doit rechercher.



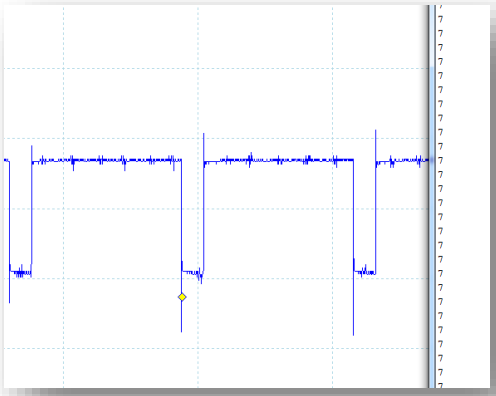
La partie intéressante de ces modules est qu'ils ont une sortie spéciale dédiée à la RSSI (Received Signal Strength Indication). Comme le nom l'indique cette sortie permet de voir l'intensité du signal entre les deux modules. Il advient que plus les modules sont proches, plus le signal est fort. Le signal est mesuré grâce

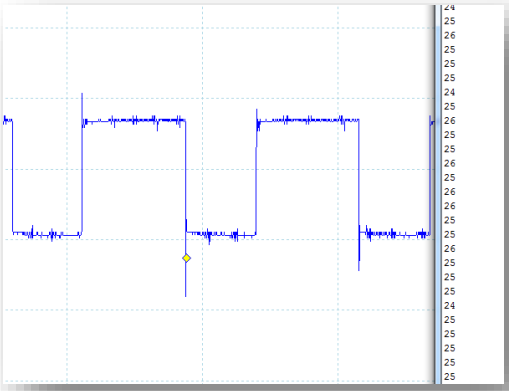
à une carte arduino. Il s'agit de regarder l'amplitude du signal avec la commande **pulseIn :**

```
rssi = pulseIn(erssi, LOW, 100);  // aquisition de la rssi brute
```

Ainsi selon l'amplitude du signal nous avons des valeurs plus ou moins grandes, ici nous regardons le temps ou le signal est à 0V :

Des hauts et des bas



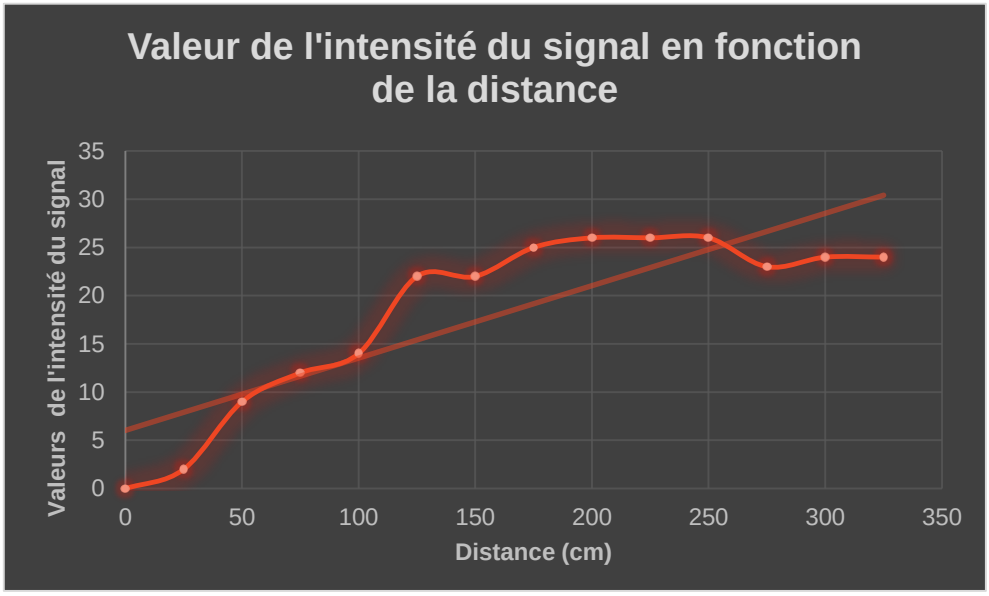


• **Cependant** les valeurs fluctuent extrêmement vite et sont très sensibles, nous les avons donc moyennées grâce à une boucle **pour** (programme complet en annexe **doc 1**) :

```
// ---------------------------------rssi-----------------------------------------------

  int rssi ;
  int valeurmoy = 0;

    for (int i = 1; i <= 500; i++){
        rssi = pulseIn(erssi, LOW, 100);      // aquisition de la rssi brute
        valeurmoy = valeurmoy + rssi;         // Somme de 500 valeurs de la RSSI
        }

     rssi = valeurmoy / 500;                  // calcule de la rssi moyenne
     Serial.println (rssi);
```

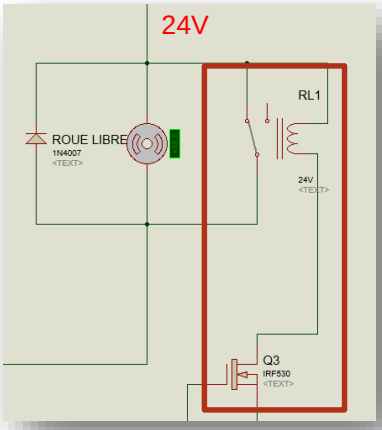
Nous connaissons maintenant l'intensité du signal, il faut maintenant passer aux mesures du réel :

Ces mesures ont été réalisées sans aucun obstacle devant les deux modules radios. Ces données seront erronées une fois que les deux modules seront dans leur boitier respectif. Cela donne tout de même une idée de l'évolution de la RSSI, d'autres mesures en condition réelle seront réalisées.



Ainsi nous avons pu obtenir une valeur à partir de laquelle nous pouvons stopper le moteur.

• Pour stopper le moteur couper l'alimentation n'est pas suffisant, nous **court-circuitons** donc le moteur à l'aide d'un relais 24 V.

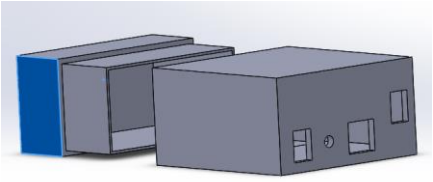


Un boitier pratique

Un boitier pratique

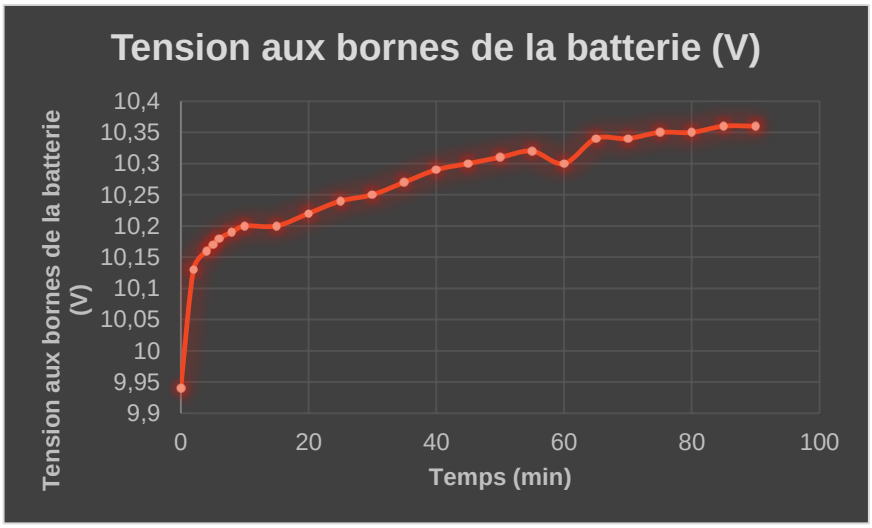
Pour détecter la distance entre la poussette et l'utilisateur il faut qu'un module soit placé sur la poussette (celui placé sur l'arduino principal) et il faut que l'autre module soit placé dans la poche de l'utilisateur (branché à un arduino secondaire).

Ainsi il a été nécessaire de créer un boitier permettant de prendre en charge l'arduino, le module XBee et la batterie qui va avec. Le boitier a été **imprimé en 3D.**



La principale partie de ce boitier est la gestion de la batterie 9 V. Lorsque nous allumons ce boitier, une **led verte** s'éclaire pendant deux secondes si le boitier est suffisamment chargé et prêt pour aller faire une balade. Si le boitier n'est pas suffisamment chargé, une **led rouge** s'éclair.

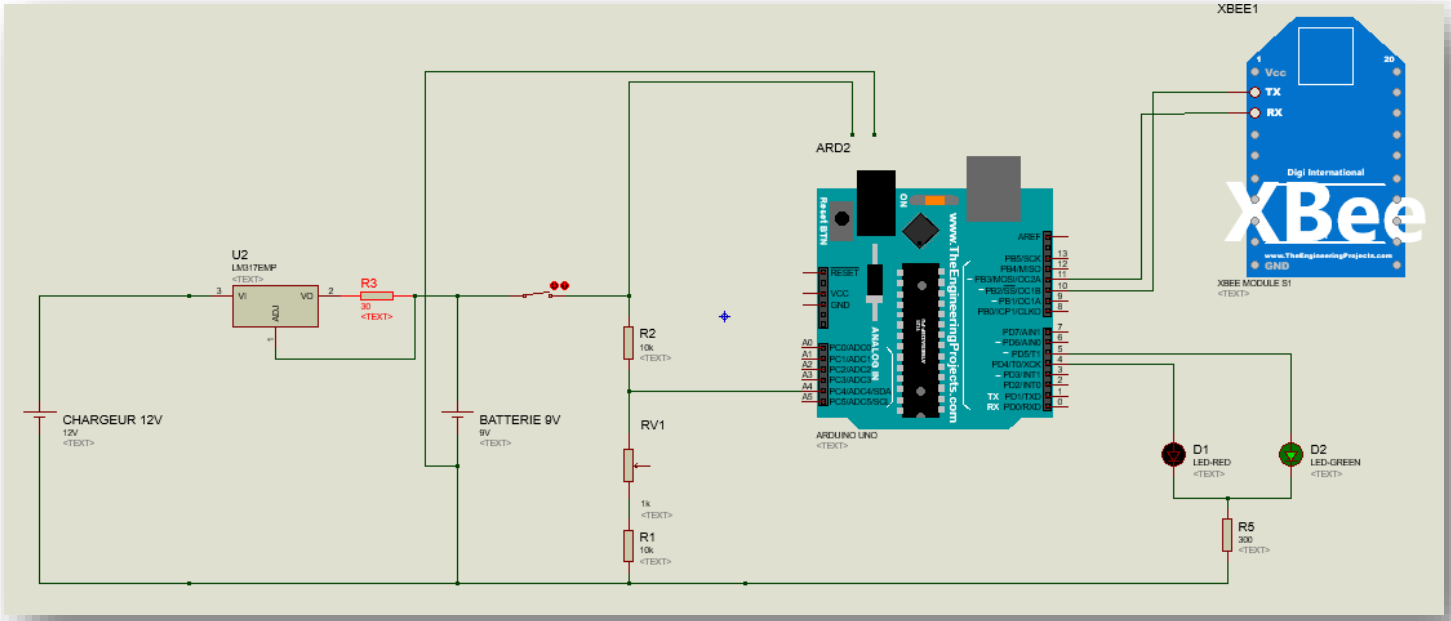
Pour faire charger le boiter il faut le brancher à une prise secteur avec une alimentation **12V fournit avec la poussette**. Pendant le chargement une led rouge est allumée. Cette led devient verte une fois la batterie chargée. Pour cela nous avons dû tester la valeur de la pile pendant le chargement de cette dernière.



La tension aux bornes de la pile est divisée par 2.5 à l'aide d'un pont diviseur de tension. Ainsi nous pouvons mesurer la tension de la batterie grâce à une entrée analogique de l'arduino maintenant que la tension est inférieure à 5 volts.

(Le programme complet est en annexe, doc 2)

• ***Schéma électrique général du boitier :***



Une position relative

Dans cette partie nous allons voir comment nous avons rendu la poussette plus confortable et à la fois plus sécurisée pour l'enfant.

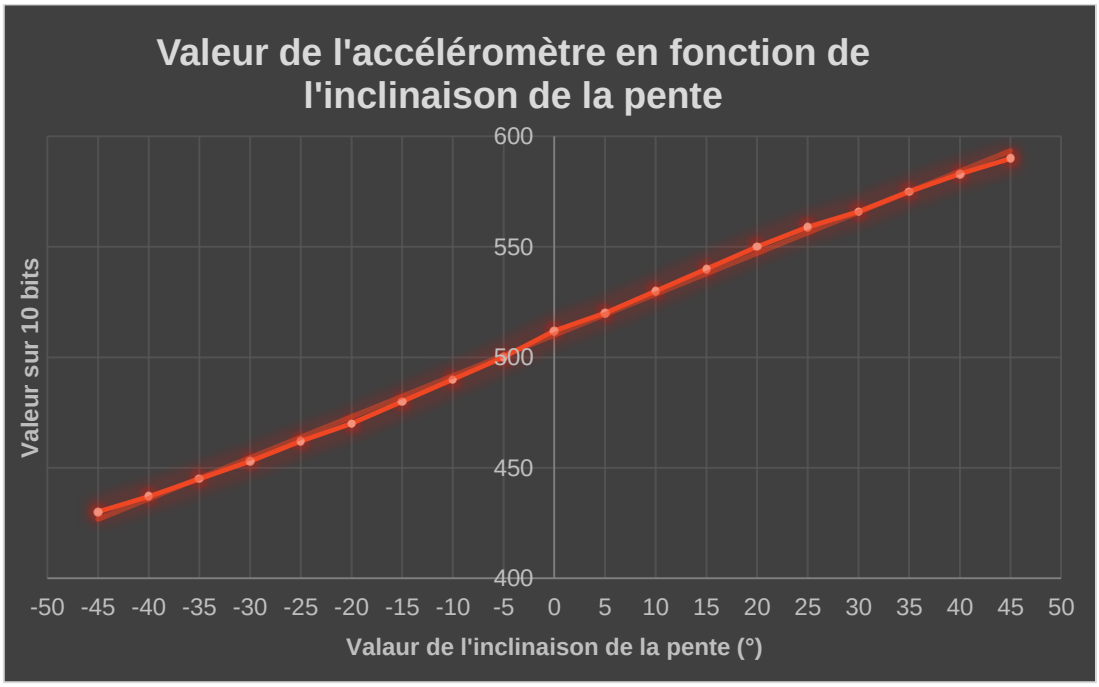
Pour commencer nous avons pensé qu'il serait bien que le berceau soit toujours horizontal pour que le bébé soit toujours dans la même position et ainsi éviter que dans les descentes trop pentues il se retrouve trop en avant et puisse tomber. Pour cela nous utilisons un moteur de deux tours minute que nous verrons dans un deuxième temps.

L'inclinaison du berceau

A] Accéléromètre

Pour gérer cette position relative nous sommes partis sur l'utilisation d'un accéléromètre. Il est positionné sur le berceau pour qu'il puisse suivre son mouvement. L'accéléromètre nous permet de mesurer l'accélération du berceau suivant différents axes. Cette mesure nous permet de connaître la position angulaire du berceau par rapport au sol et ainsi nous permettre de changer son orientation. Nous n'utilisons que l'axe X de l'accéléromètre car le berceau ne possède qu'une rotation autour de l'axe Z. Notre accéléromètre est alimenté en 5V et il nous donne une tension entre 0 et 5V qui est convertie par une entrée analogique sur 10 bits (entre 0 et 1023) pour donner sa position. Grâce à ce procédé, nous en avons déduis que l'accéléromètre est à l'horizontal quand il nous envoie la valeur 512. Nous avons également introduit un condensateur dans le circuit car cela permet de faire une moyenne facilement de la tension reçue et donc de l'orientation du berceau. En effet celui-ci peut bouger très vite si la poussette roule sur un caillou…

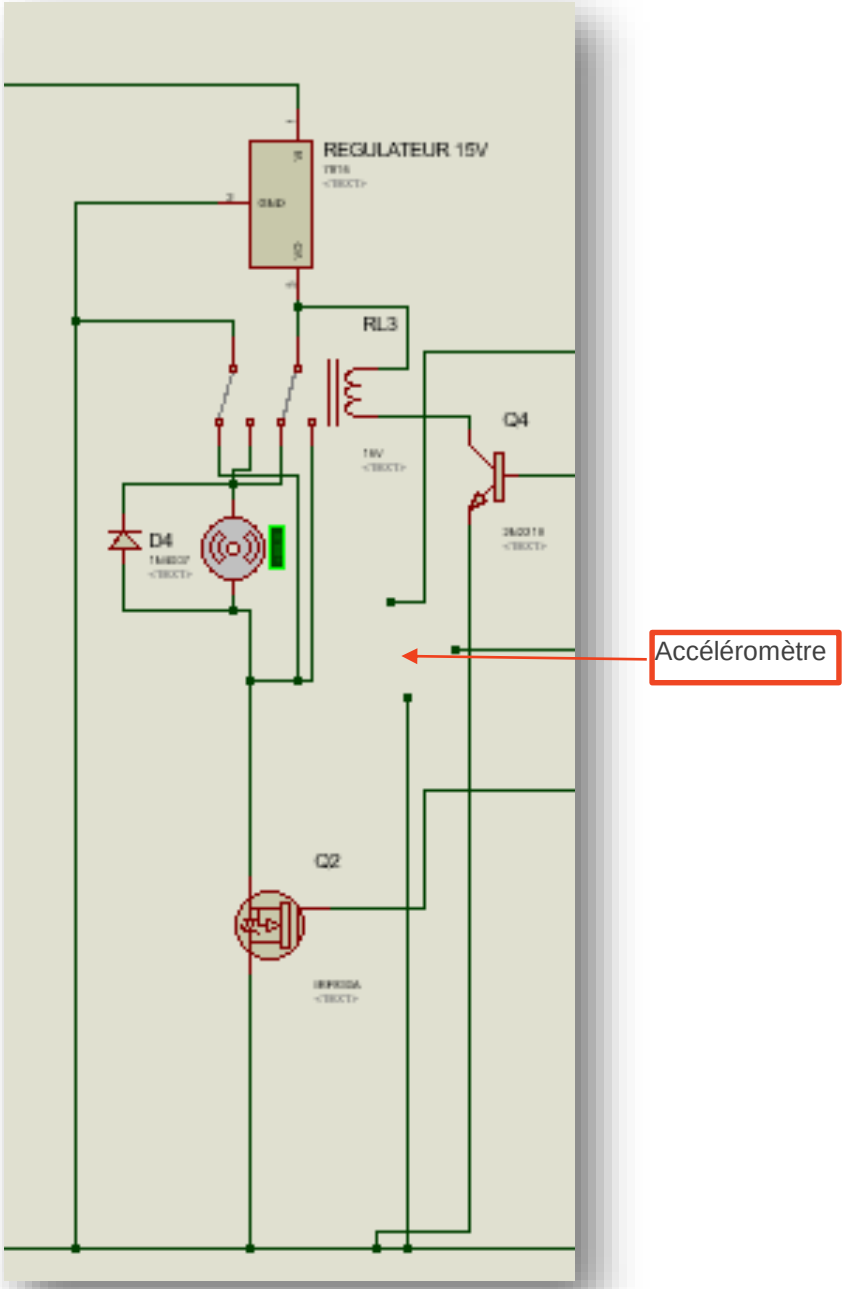
B] Programme et électronique



Pour programmer il nous a fallu dans un premier temps créer un petit programme pour mesurer les valeurs de l'accéléromètre en fonction de l'angle. `Serial.println(accelerometre);` Ceci nous a donné une fonction linéaire d'équation : **y = 1,853x + 512**, venue confirmer notre supposition. En effet l'horizontalité est à la valeur 512 (milieu de 1023 puisque nous sommes sur 10 bits). Par la suite, nous avons programmé notre carte arduino afin que l'accéléromètre soit toujours à l'horizontal. Comme pour la RSSI vu précédemment, nous utilisons une valeur moyennée de la valeur envoyée par l'accéléromètre. Il y a de plus une marge d'erreur acceptée grâce à des paliers qui permettent au moteur gérant l'inclinaison, de ne pas modifier la position du berceau en permanence dès qu'il y a des variations d'angles insignifiants. Cette marge est de l'ordre d'une variation de l'angle supérieur à 5° soit une variation des valeurs de 10 pour l'accéléromètre (le programme complet est en annexe).

Une poussette confortable

Une partie électronique importante a été nécessaire puisque nous avons besoin d'alimenter le moteur dans les deux sens. Nous avons donc réalisé le montage ci-dessous :

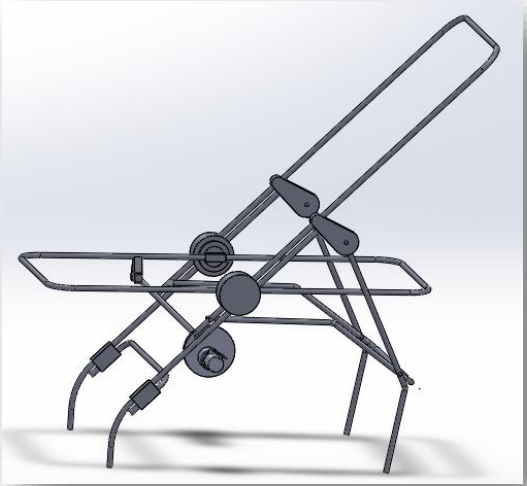


Un vérin de redressement

Nous venons de voir comment nous allons changer l'inclinaison du berceau au niveau de la programmation mais il nous faut aussi le mécanisme pour pouvoir gérer cette variation. Cette gestion se fait avec un moteur qui entraîne un disque qui lui, est relié à une bielle qui est fixée sur le berceau. Ainsi en faisant tourner le moteur, l'angle entre le berceau et le sol change.

A] Solidworks

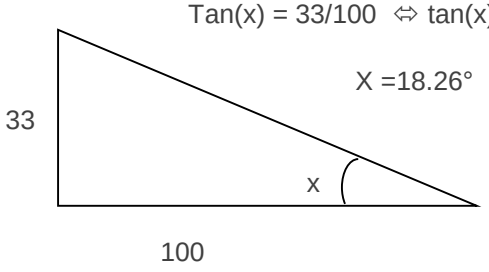
Pour commencer notre projet nous avons construit un modèle solidworks qui nous a permis de visualiser les changements que nous allons faire et les problèmes que cela impliquerait. Donc il nous a été plus facile de construire le prototype par la suite. La modélisation a pris beaucoup de temps à être finalisée et ainsi pouvoir définir les liaisons de l'ensemble cinématique. Cette définition des liaisons permettra également de faire des calculs avec meca3D du couple nécessaire à fournir par le moteur. Nous pourrions après étudier la résistance des matériaux à utiliser pour créer le prototype.

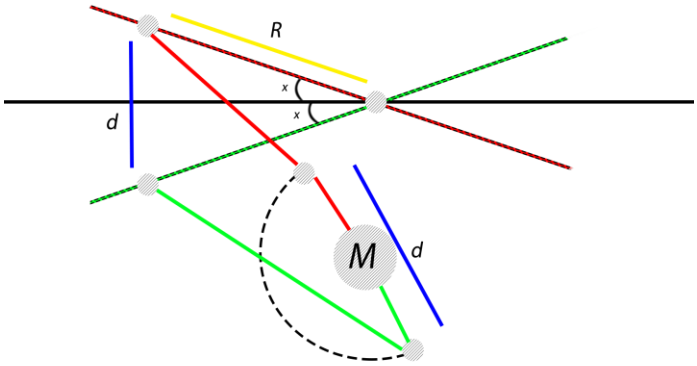


B] Calculs

Pour commencer les calculs nous pensons faire effectuer qu'un demi-tour au moteur pour que la bielle parcoure une certaine distance. Nous sommes donc partis sur la base d'un disque de diamètre 10 cm pour que le point de fixation de la bielle parcoure une distance de 8 cm (d=8). De plus nous avons regardé les pentes les plus pentues de France. Elles sont de 33%. Nous avons donc un angle maximum de ± 18° à modifier.

$\tan(x) = 33/100 \Leftrightarrow \tan(x)^{-1} = 18.26°$





Or 37° correspond à un angle de π/5

donc $d=R*x \Leftrightarrow R= d/x \Leftrightarrow R = \frac{8}{\frac{\pi}{5}}$

R = 12.7 cm

180° = π

37° ≈ π/5

Nous savons donc que si nous voulons une variation de 37° pour le berceau il faut que la bielle soit fixée à 12.7cm du centre de rotation du berceau avec un moteur qui fasse varier la distance de 8 cm.

Conclusion et remerciements

Plus qu'une maquette, un prototype

Pour conclure, ce projet est pour nous une expérience enrichissante. Il nous a appris à répondre à des besoins en s'adaptant à de nombreuses contraintes. Notre poussette est alors capable de maintenir une vitesse constante préalablement sélectionnée par l'utilisateur. De plus, elle est sécurisée par ondes radio capables de donner l'ordre de stopper la poussette en cas d'écart trop important entre la poussette et l'utilisateur. Elle permet également d'incliner le berceau afin qu'il reste toujours à l'horizontal pour le confort et la sécurité du bébé. Pour aller plus loin, nous aurions pu rajouter un réducteur au moteur principal pour de meilleures performances lors de côte. Nous aurions pu également permettre à l'utilisateur de choisir lui-même l'inclinaison du berceau. Une amélioration dans la précision des ondes radio est envisageable afin de choisir sa propre distance de sécurité. Et le boitier pourrait directement être remplacé par le smartphone de l'utilisateur où il pourrait régler ses différents paramètres.

Nous remercions M.PINAULT, M.VALLON, M.ERRAMI et M.DUFFAIT pour leurs aides précieuses dans ce projet

poussette.universelle@gmail.com

ANNEXES :

Programme principale, arduino de la poussette :

```
#include<SoftwareSerial.h>
#include <XBee.h>

// -------------- Noms des entrées / sorties ---------------------

    //Partie ValentinCôme :

// Valentin
        int capt = 13; // Capteur photoélectrique


// Côme
        int k = 1;        // coefficient
        byte moteurP = 9;  // fil blanc

    // Partie Mathieu :

        long erssi = 2;
        int Rx = 10;
        int Tx = 11;
        byte ledV = 3;
        byte ledR = 4;
        byte arretmoteur = 5; //fil orange
        byte securite = 12;
        byte boutton = 7;
        byte arretmoteurValeur = 0; // Permet de savoir si le moteur est
actuellement court-circuité

    // Partie Gatien

       int acc = A0; //variable de la valeur de l'accéléromètre
       byte arret15 = 6;    // fil bleu
       byte sensmoteur = 8; // fil blanc et orange



// ---------- Recevoir les données vers la poussette, configuration xBee
-----------------

SoftwareSerial serial1( Rx, Tx); // RX, TX
XBee xbee = XBee();
Rx16Response rx16 = Rx16Response();


void setup()
{
```

```
//Partie Valentin et Côme

   pinMode(A2,INPUT);    //pattes du sélectionneur de vitesse
   pinMode(A3,INPUT);
   pinMode(A4,INPUT);
   pinMode(A5,INPUT);
   pinMode(capt, INPUT);

   pinMode (moteurP, OUTPUT); // Moteur principal de la poussette


// Partie Mathieu
   pinMode (erssi,INPUT); // Rssi xBee
   pinMode (ledV, OUTPUT);
   pinMode (ledR, OUTPUT);
   pinMode (arretmoteur, OUTPUT); //court-circuit moteur
   pinMode (securite, INPUT);  //interrupteur pour savoir si la sécurisé
est voulu
   pinMode (boutton, INPUT); //  bouton prévenir que l'utilisateur est
de nouveau vers la poussette
   Serial.begin(9600);   // connexion module xBee...
   serial1.begin(9600);
   xbee.setSerial(serial1);


 // Partie Gatien

   pinMode (acc, INPUT); // acceleromètre
   pinMode (sensmoteur, INPUT);  // choisir le sens de rotation du moteur
   pinMode (arret15, INPUT); // arrêter le moteur d'inclinaison du
berceau

}

void loop()
{

  valentincome();
  mathieu();
  gatien();

}

void valentincome() {

// Valentin
```

```
// ------------ Vitesse------------

byte CaptCompteur = 0;
byte CaptEtat=0;
byte CaptDernierEtat = 0;
long T1;
long T2;
long vitesse;



if (CaptCompteur == 0){
  T1 = millis();  // On déclenche le compteur du temps
}

CaptEtat = digitalRead(capt);

if (CaptEtat != CaptDernierEtat) { // détecte les changements d'état du
capteur photoélectrique
    CaptCompteur ++;
}

if (CaptCompteur == 10) {
  T2 = millis();    // arret du temps T2 après un tour complet de la roue
   vitesse = ((2*3.14159265*6.7*100)/((T2-T1)*3600))*(255/7); // Mesure
de la vitesse et convertissement sous 8 bits
     // (ici il manque à arrondir cette valeur pour le PWM d'après,
travail sur une formule mathématiques en cour)
  CaptCompteur = 0;
}


if (vitesse > 255) {
  vitesse = 255;
}

CaptDernierEtat = CaptEtat;

// VITESSE PRENDRE LA VALEUR ARRONDI !!!!!!!!!!!!!!!!


// ---------- Consigne -----------

byte e1 = 1;  // pattes du sélectionneur
byte e2 = 2;
```

```
byte e4 = 4;
byte e8 = 8;

if (digitalRead(A2)>50){  // On convertit le signal binaire en numérique
  e1 = 1;
}
  else {
    e1 = 0;
  }
if (digitalRead(A3)>50){
  e2 = 2;
}
  else {
   e2 = 0;
  }
if (digitalRead(A4)>50){
  e4 = 4;
}
  else {
   e4 = 0;
  }

if (digitalRead(A5)>50){
  e8 = 8;
}
  else {
   e8 = 0;
  }

byte selectionneur = e1 + e2 + e4 + e8;


// ------- Conversion de la consigne sur 8 bits --------

byte consigne ;

      if (selectionneur == 0){  // affectation de la consigne
         consigne = 0;
      }
      if (selectionneur == 1){
         consigne = 36;
      }
      if (selectionneur == 2){
         consigne = 73;
      }
      if (selectionneur == 3){
```

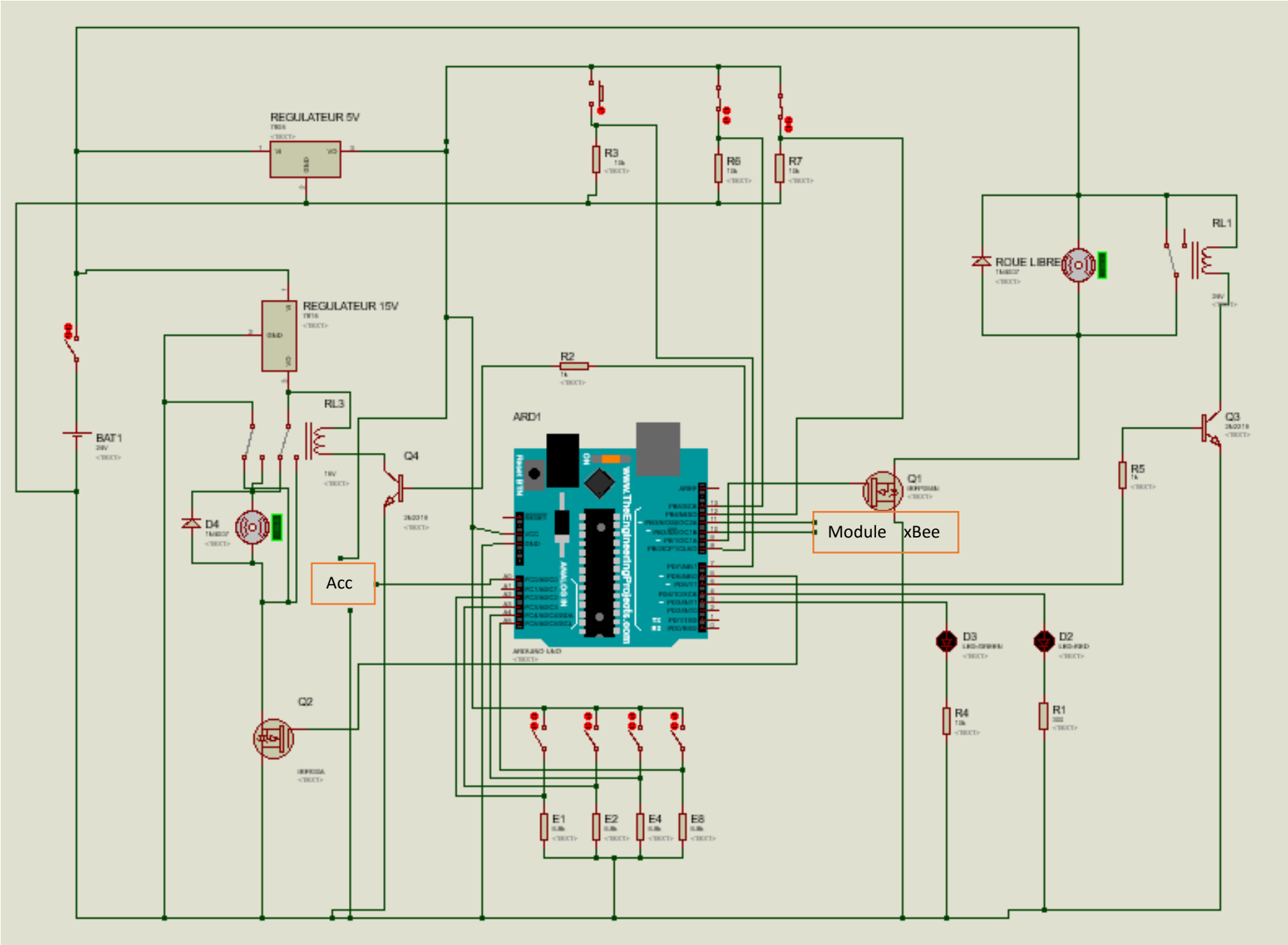
```
          consigne = 109;
       }
       if (selectionneur == 4){
          consigne = 146;
       }
       if (selectionneur == 5){
          consigne = 182;
       }
       if (selectionneur == 6){
          consigne = 219;
       }
       if (selectionneur == 7){
          consigne = 255;
       }

// Côme

  int  result = (consigne - vitesse)*k; //détermination du résultat

  if (result>255){ //vitesse actuelle trop faible, allimenter le moteur
   result=255;
  }
  if (result<=0){ //vitesse actuelle trop élevée, stopper l'allimentation
   result=0;
  }

  if (arretmoteurValeur == 0) {
     analogWrite (moteurP, result);
  }

  }

void mathieu() { //Partie sécurité de la poussette

  // ------------- Sécurité voulu ou pas ------------

if (analogRead(securite) == 1) { //la sécurité est souhaitée


  // ------------ Connexion ------------------

  int connexion = 0;

  for (int t=1; t<= 20;t++) {                        // Connexion perdue
ou pas
  connexion = connexion + digitalRead(2);
```

```
  }

  if ( connexion >= 1  ) {                        // Poussette connectée
      digitalWrite (ledV, HIGH);                  // Signal que la sécurité
est en fonctionnement
      digitalWrite (ledR, LOW);


 // ---------------rssi-----------------

    int rssi ;
    int valeurmoy = 0;

      for (int i = 1; i <= 500; i++){
          rssi = pulseIn(erssi, LOW, 100);        // aquisition de la rssi
brute
          valeurmoy = valeurmoy + rssi;           // Somme de 500 valeurs
de la RSSI
           }

       rssi = valeurmoy / 500;                    // calcule de la rssi
moyenne
      Serial.println (rssi);   // (permet de simuler voir la valeur
numérique sur ordinateur)

  // ----------------test arret ----------------

    int testarret = 0;

      for (int u=1; u <=100 ; u++) {
        testarret = testarret + rssi;
         }

        testarret = testarret/100;

     if (testarret > 18) {                            // Regarder si la
poussette est trop loin de l'utilisateur
       digitalWrite (moteurP, LOW);                   // Arreter le
moteur puis
       digitalWrite (arretmoteur, HIGH);              // Commande de
court-circuiter le moteur
       arretmoteurValeur = 1;
      Serial.println ("Arreter la poussette");  // (simulation ordinateur
   }
   else {
    if (digitalRead(boutton) == 1) {
```

```
      digitalWrite (arretmoteur, LOW);              // Arrêter de
court-circuité le moteur
        arretmoteurValeur = 0;
        }
   }

}

// ----------- Connexion perdue ----------------

else {
  digitalWrite (ledV, LOW);
  digitalWrite (ledR, HIGH);              // Signal que la sécurité ne
marche plus
  digitalWrite (moteurP, LOW);            // Arreter le moteur puis :
  digitalWrite (arretmoteur, HIGH);       // On bloque le moteur,
activation du frein
  arretmoteurValeur = 1;
  Serial.println ("connexion perdue");    // simulation ordinateur
}
} // Fin securité

else {
  digitalWrite (ledV, LOW);
  digitalWrite (ledR, HIGH);              // Signal que la sécurité ne
marche pas
  digitalWrite (arretmoteur, LOW);       // Le moteur ne peut pas être
bloquer à cause de la sécurité
  arretmoteurValeur = 0;
}

}


void gatien() {


  if (analogRead(acc) > 520) {            // berceau trop incliné
   digitalWrite (arret15, HIGH);
   digitalWrite (sensmoteur,LOW);
}

  if (analogRead(acc) < 500) {             // berceau trop incliné
    digitalWrite (arret15,HIGH);
    digitalWrite (sensmoteur,HIGH);
}
```

```
  if ((analogRead(acc) == 511) || (analogRead(acc) == 512) ||
(analogRead(acc) == 513)){  // berceau a l'horizontal
    digitalWrite (arret15, LOW);
   }

  }
```

Schéma électrique générale de la poussette :



Programme du boitier, arduino dans la poche de l'utilisateur :

```
#include<SoftwareSerial.h>
#include <XBee.h>

// --------------- Envoi les données vers la poussette
----------------------------

 XBee xbee = XBee();
uint8_t payload[] = { 'H', 'i' };
XBeeAddress64 addr64 = XBeeAddress64(0, 231);
Tx16Request tx16 = Tx16Request(addr64, payload, sizeof(payload));

// -------------- Noms des entrées / sorties
---------------------------------------


int Rx = 10;
int Tx = 11;
byte ledR = 4;
byte ledV = 5;
int pile = A4;          // fil violet
byte recharge = 7;      // fil orange

SoftwareSerial serial( Rx, Tx); // RX, TX

void setup()
{
   pinMode (pile, INPUT);
   pinMode (recharge, INPUT);
   pinMode (ledV, OUTPUT);
   pinMode (ledR, OUTPUT);
   serial.begin(9600);
   Serial.begin(9600);
   xbee.setSerial(serial);

}

void loop() {

// ----------------- Test batterie suffisante -----------------

if (analogRead(pile) > 654) { // si la pile est charger a plus de 8 V, led Verte
pendant 2 sec, près pour la promenade

  digitalWrite(ledV, HIGH);
  delay (2000);
```

```
  digitalWrite(ledV, LOW);
}

else {                        // si la pile est charger a moins de 8 V, led Rouge
pendant 2 sec, à recharger avant la promenade

  digitalWrite(ledR, HIGH);
  delay (2000);
  digitalWrite(ledR, LOW);
}

// ------------ envoie de données pour la rssi --------------

xbee.send( tx16 );   //  Pas sur que cela soit nécessaire, à TESTER


// ---------- gestion batterie pile en recharge -----------

  Serial.println(analogRead(pile)); // simulation ordinateur

  if (analogRead(pile) < 818) { // Si la pile est supérieur a 10 V, alors en
recharge led Rouge
    digitalWrite(ledV, LOW);
    digitalWrite(ledR, HIGH);
  }
  else {
    digitalWrite(ledR, LOW); // Si la pile est supérieur a 10.35 V, alors en
chargée led Verte
    digitalWrite(ledV, HIGH);
  }

}
```

Schéma électrique du boitier dans la poche de l'utilisateur :