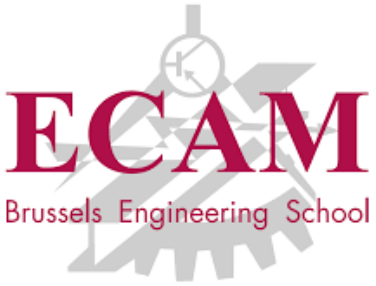

Projet Electronique Embarqué

Canon de Gauss

Auteurs :
Harold Snyers 16243
Bernard Tourneur 16148
Christos Bournousouzis 16219

Professeurs :
Mr. Sébastien Combéfis
Mr. Cédric Marchand

Un canon électromagnétique contrôlé à distance

depuis un tableau de bord

Année académique 2018-2019

Contents

Canon de Gauss

1.1 Introduction

L'objectif du projet est d'implémenter une expérience de laboratoire de niveau secondaire, contrôlable à distance depuis un ordinateur. L'utilisateur doit pouvoir déclencher l'expérience, et récupérer les éventuelles données à collecter, le tout depuis un panneau de bord (ou "dashboard") sur son ordinateur.

Nous avons choisi comme sujet d'expérience l'électromagnétisme et le rendement énergétique. L'expérience consiste à mesurer l'énergie cinétique d'un projectile lancé par un canon de Gauss (ou canon électromagnétique), à la comparer avec l'énergie électrique consommée par le système, pour en déterminer le rendement énergétique.

1.2 Développement électronique

1.2.1 Fonctionnement du canon

Un canon de Gauss consiste en un bobinage cylindrique creux, le projectile placé en son centre, auquel on applique un pic de courant important. Ce dernier va générer une variation de flux magnétique, créant une force magnétomotrice maximale au centre de la bobine et lançant ainsi le projectile. Plusieurs facteurs affectent la valeur de la force magnétomotrice générée; la bobine en elle même, son impédance, le nombre de tours de fils, etc; mais également le courant électrique qui la traverse, et la quantité d'énergie disponible.
Ainsi, pour pouvoir libérer un maximum de courant le plus vite possible, nous avons choisi d'accumuler les charges dans des condensateurs (à capacité relativement grande).
Pour charger les condensateurs (placés parallèlement à la bobine), le circuit doit être ouvert derrière ceux-ci, et ils doivent être mis sous tension. Une fois les condensateurs pleins (lorsque leur tension est plus ou moins égale à la tension de charge), on peut fermer le circuit (à l'aide d'un relais électromécanique); la bobine se comportera alors comme un court-circuit (ou une résistance très faible) et les condensateurs se déchargeront brusquement, provoquant un pic de courant important dans la bobine.

Il est important que le relais ne soit fermé qu'un bref instant, pour que l'impulsion soit momentanée; si elle dure trop longtemps, la force magnétomotrice provoquera une décélération du projectile.

1.2.2 Schémas

Tous les schémas du système se trouvent en annexe; les références vers ceux-ci pointent donc vers la fin de ce document.
Voici le détail des composants du schéma général (Figure A.1):

- Condensateur x 6
- Résistance x 6
- Diode x 1
- Relais x 1
- Bobine x 1

La source de tension est contrôlable à distance; elle n'a pas été intégrée dans notre prototype mais est nécessaire pour un contrôle complet du système à distance.
Analysons le schéma général par bloc fonctionnel.

a Charge des condensateurs

(Figure A.2)

Dans ce bloc, la résistance sert à limiter le courant de charge des condensateurs. Les condensateurs vont donc se charger jusqu'à la tension d'alimentation du système[1], moins la tension présente sur la diode (environ 0.7 V). Cette dernière sert à empêcher la décharge des condensateurs lorsque la source est éteinte; sans elle, les condensateurs pourraient se décharger au travers de la résistance.

b Décharge des condensateurs

(Figure A.3)

Pour fermer le circuit, un relais électromécanique est placé entre les condensateurs et la bobine, faisant office d'interrupteur. L'entrée du relais, (qui commande l'électro-aimant

[1] Dans notre prototype, 50V

activant la commutation) est connectée à une sortie de l'Arduino. Lorsque l'Arduino envoie 5V, le relais est passant et le projectile lancé.

Comme mentionné précédemment, il faut que le relais soit fermé pendant un temps relativement court. Nous avons déterminé par essai-erreur que, sous 50V, un temps "passant" de 150 ms suffit.
Pour éviter que les condensateurs continuent de se charger lors du tir, nous avons choisi d'éteindre la source de courant avant la mise à feu. Nous avons en effet constaté une légère augmentation de performances dans ces circonstances. L'hypothèse retenue est que le courant entrant dans les condensateurs limite le courant sortant, diminuant ainsi la valeur totale de courant dans la bobine.
Pour avoir un grand gain de courant, nous avons décidés de mettre nos condensateurs en parallèle.
Si nous faisons le calcul :
Avec Rtot $\simeq$ 30 ohm, pour un condensateur on a : I = $\frac{50}{30}$ = 1.67A
Itot = I*6 $\simeq$ 10A, on aurait donc +/- 10A qui passerait dans notre bobine pour une tension de 50V.

c Mesure de tension

(Figure A.4)

Pour mesurer la tension sur les condensateurs nous utilisons un diviseur de tension de rapport 10 (valeur expérimentale = 9.1314) car l'ADC de l'Arduino ne convertit pas une tension supérieur à 5V (si nous prenons la référence interne de l'Arduino bien évidemment).

Nous avons utilisé des résistances très grandes pour éviter d'avoir trop de perte de courant. Nous prenons la mesure entre R2 et R3 et nous la connectons à une pin analogique de l'Arduino.

d Mesure de courant

(Figure A.5)

Nous utilisons une résistance de puissance de 1 ohm comme résistance de shunt, pour mesurer le courant passant dans la bobine.
U = R * I $\rightarrow$ U = 1 * I $\rightarrow$ U = I
La valeur de courant est donc directement donnée par la tension lue. Pour éviter d'endommager l'ADC de l'Arduino en cas de pic de courant important, nous utilisons un diviseur de tension de rapport 3 (valeur expérimentale = 2.241).

e Capteur de vitesse

(Figure A.6)

Pour déterminer la vitesse du projectile (qui sera utilisée pour trouver son énergie cinétique via E=($\frac{m*v^2}{2}$), on procède en deux étapes : dans un premier temps, l'utilisateur entre la distance entre le canon et la cible; ensuite, après le tir, le système détermine le temps de parcours du projectile.

Pour calculer le temps de parcours, on place sous l'extrémité du canon un capteur infrarouge (connecté à l'Arduino), détectant le passage du projectile. L'instant de passage du projectile est enregistré dans l'Arduino. Il s'agit d'un capteur digital, qui envoie un 1 logique lorsqu'il détecte un passage. Sur la cible est placé un bouton poussoir, connecté également à l'Arduino. Lorsque le projectile atteint cette cible, il enfonce le bouton poussoir, qui envoie alors un 1 logique à l'Arduino.
Une simple différence tdépart - tarrivée permet de trouver le temps de parcours. La vitesse sera calculée directement dans le dashboard.

1.2.3 Améliorations possibles

La diode pourrait être remplacée par un relais commandé via l'Arduino. Dans ce cas, nous n'aurions plus besoin d'une source commandée mais d'une simple source qui restera constamment branchée, l'Arduino contrôlant quand ouvrir le circuit et donc quand arrêter la charge des condensateurs.
Pour optimiser la puissance du canon, il est recommandé de prendre une bobine avec un noyau ferromagnétique.

Au niveau de la lecture de la tension sur les condensateurs et de la lecture du courant, plusieurs choix s'offrent à nous; soit des résistances beaucoup plus précises pour le diviseur, soit utiliser un ADC externe permettant de lire des tensions plus importantes.

Pour le capteur de vitesse, il faudrait remplacer le bouton poussoir. En effet, l'énergie du projectile est insuffisante pour l'enfoncer. Ce problème pourrait être résolu soit en utilisant un bouton plus sensible, soit en appliquant plus de tensions à la bobine, et donc en fournissant plus d'énergie au projectile.
Le capteur de mouvements pose également problème; il ne détecte pas systématiquement le passage du projectile. Idéalement, il faudrait mieux le positionner, en le fixant au canon, et en s'assurant que le projectile passe systématiquement au dessus du capteur.

1.3 Développement télématique

1.3.1 Choix des protocoles

Nous avons choisi le protocole UART pour la communication BBB-Arduino car :

- La librairie Serial Arduino implémente déjà ce protocole, est simple à utiliser et fiable.
- La BBB et l'Arduino sont les seuls présents sur le bus; il n'y a aucun esclave donc pas de relation maitre-esclave ni d'adressage nécessaire.
- La communication se fait dans les deux sens systématiquement, quand la BBB envoie un msg elle doit recevoir une réponse, il est donc intéressant d'utiliser un protocole full duplex.

Aucun protocole n'est utilisé pour les capteurs, qui envoient de simple signaux logiques si nécessaire.

1.3.2 Configuration Arduino

Le code est disponible dans l'appendice section A.2.1.

Comme précisé, nous implémentons la communication UART à l'aide de la librairie Serial; nous commençons par créer le bus mySerial et choisissons les pins utilisées.
S'en suit l'initialisation des variables et des pins. Dans le void setup, le bus Serial est démarré et les pins les pins uart ou autres composants sont configurées.

Dans le void loop, il y a deux parties. Une partie permet de lire ce que l'Arduino reçoit sur le Serial. Ici, elle attend de recevoir la valeur de la tension à appliquer afin d'activer le premier transistor de puissance qui laissera donc passer la tension requise et donc de charger les capacités. Quand le tension est atteinte, le transistor est coupé. L'autre transistor est alors activé afin d'envoyer le courant sur la bobine.

Le bloc suivant permet de vérifier que la valeur de tension a bien été envoyée.
Enfin, le temps de parcours est calculé en testant d'abord si le projectile a survolé le capteur de mouvement, ensuite s'il a enfoncé le bouton; à chaque évènement, il enregistre le temps de passage, et fini par calculer une différence de temps qu'il peut envoyer à la BBB.

1.3.3 Configuration Beagle Bone Black

Pour lancer l'écoute et l'envoi des données entre la BeagleBone et l'Arduino nous avons 2 programmes, un pour l'écoute (readuart_Arduin-BBB-Dash.py) et l'autre pour l'envoi (writeUart_Dash-BBB-Arduino.py). Ces 2 programmes vont permettre non seulement à la BeagleBone de communiquer avec l'Arduino en UART mais également de communiquer avec le dashboard via MQTT. Nous avons choisi d'écrire ces programmes en python pour ses avantages de clarté et de lisibilité, et parce que la librairie UART utilisée[2] intègre des fonctions C dans le code python, permettant des performances quasi équivalentes à un code C, pour la communication.

La première chose à faire est de choisir sur quel port UART se connecter, puis le port de la connexion MQTT, généralement le port 1883.
Les codes sont disponibles dans l'Appendice section A.2.2.

readuart_Arduin-BBB-Dash.py

La BeagleBone attend de recevoir quelque chose de l'arduino et enregistre le contenu reçu s'il y a des données disponible sur le Serial. Ensuite, il déchiffre les données reçues (chaque information transmise en UART est identifiée par un caractère, permettant à la BBB de différencier les valeurs reçues de l'Arduino) et en fonction, publier les valeurs obtenues sur les topics respectifs. Les données reprises sur le Serial de l'Arduino sont en format string, donc avant de les publier, les données numériques sont converties en int.

writeUart_Dash-BBB-Arduino.py

La BBB attend de recevoir une valeur sur le topic sur lequel elle est abonnée et, lorsqu'une valeur est disponible, envoie cette même valeur sur le Serial de l'Arduino.

1.3.4 Configuration DashBoard

Le dashboard présente l'interface utilisateur (Figure A.7). Il a été configuré à l'aide de l'outil de programmation Node-RED.

Les données entrantes arrivent par 4 topics MQTT : sensor/time, sensor/current, sensor/-voltageIn et sensor/conf. Le dashboard est composé de plusieurs parties. Il y a 4 compartiment, le premier affiche la mesure de la vitesse, le deuxième s'occupe d'envoyer les paramètre de tension et affiche la tension réellement appliquée. Les 2 derniers sont responsables d'afficher la mesure de courant et les la calcul de l'énergie cinétique et électrique.

[2] Adafruit_BBIO.UART

Le flux du dashboard se divise en deux parties.

La première partie (figure A.8), est composée d'un slider pour choisir la tension à appliquer, d'une gauge qui affiche la tension choisie et finalement un switch qui va permettre d'activer l'envoi de la tension sur le topic sensor/voltage. Le topic entrant sensor/conf affiche le status de l'envoi de la tension dans bloc texte.

La deuxième partie (figure A.9), est composée de 3 données entrantes par MQTT. Chacune des valeurs de ces topics est affichée dans leur bloc texte respectif. Ensuite un bloc "inputtext" permettra à l'utilisateur d'entrer la distance entre le canon et la cible.
Une fonction est chargée de récupérer les valeurs des différents topic MQTT et la distance entrée, d'envoyer les valeurs de tension et de courant aux afficheurs respectifs, puis de calculer les valeurs de vitesse, énergie cinétique et électrique, et enfin le rendement. Le code est disponible dans l'annexe à la section A.2.3.

En pratique, la tension choisie n'influence pas le système dans notre prototype; la tension de charge des condensateurs sera toujours égale à la tension d'alimentation, puisque nous ne disposons pas d'alimentation contrôlable électriquement, et n'avons pas implémenté le relais de contrôle mentionné plus tôt. L'envoi de la tension aura donc pour seul conséquence la mise à feu immédiate du canon de Gauss.

1.4 Conclusion

Le projet est fonctionnel, bien qu'il ne soit qu'un prototype et qu'il y ait encore beaucoup de place pour des améliorations. Certaines parties de l'expérience doivent encore être simulées manuellement (il faut appuyer sur le bouton poussoir, cfr section 1.2.3: améliorations possibles).

Le projet a été très instructif à plusieurs niveaux; comme premier projet où nous avons conçu et designé un circuit électronique, nous avons beaucoup appris de nos erreurs et développé des raisonnements de conceptions. De plus, le projet a permis d'approfondir notre compréhension des protocoles MQTT et UART.
Nous sommes donc globalement satisfaits du déroulement du projet.

Appendix

A.1 Schéma

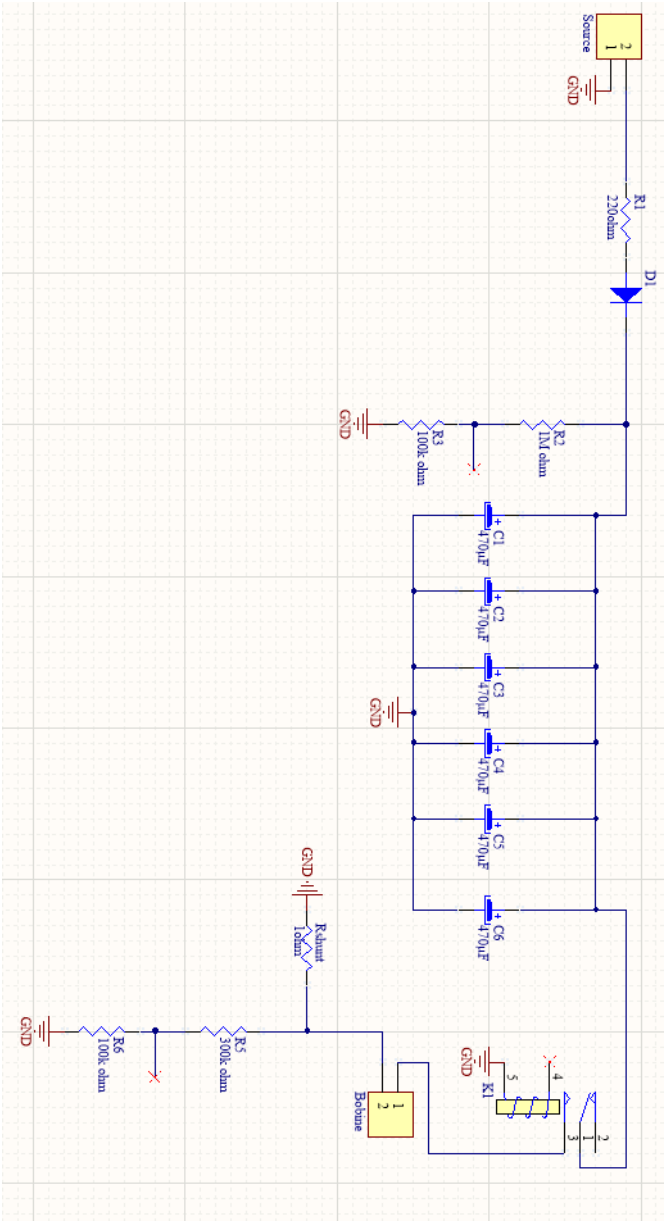

FIGURE A.1: Schéma Global

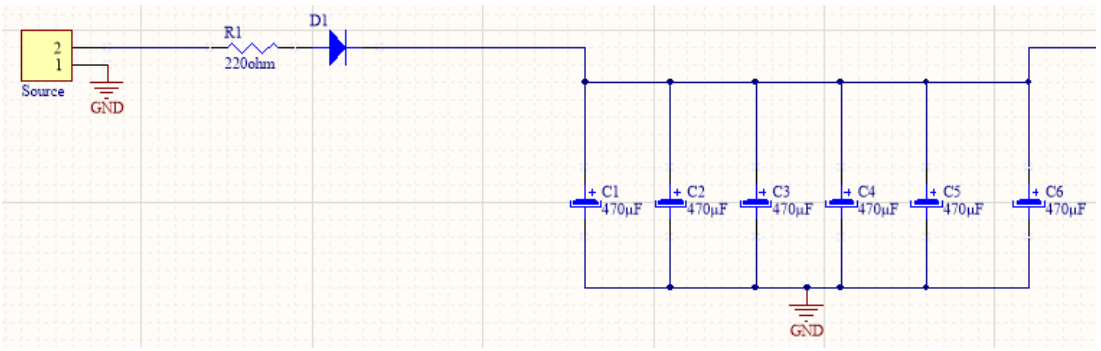

FIGURE A.2: Schéma de Charge

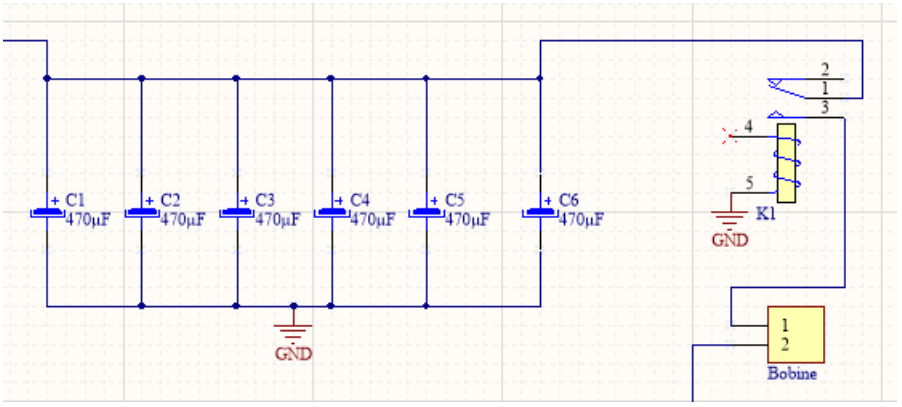

FIGURE A.3: Schéma de Décharge

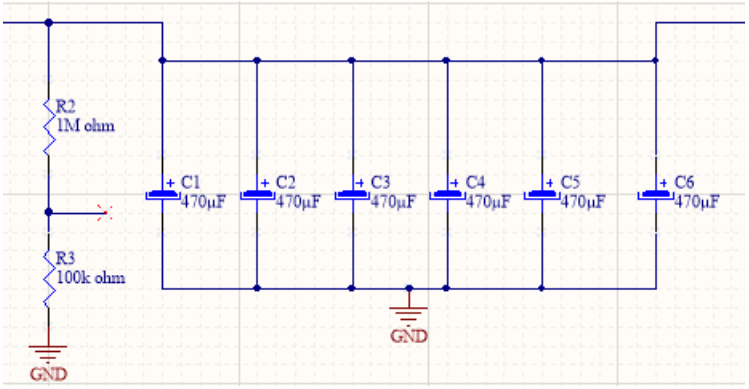

FIGURE A.4: Schéma du Voltmètre

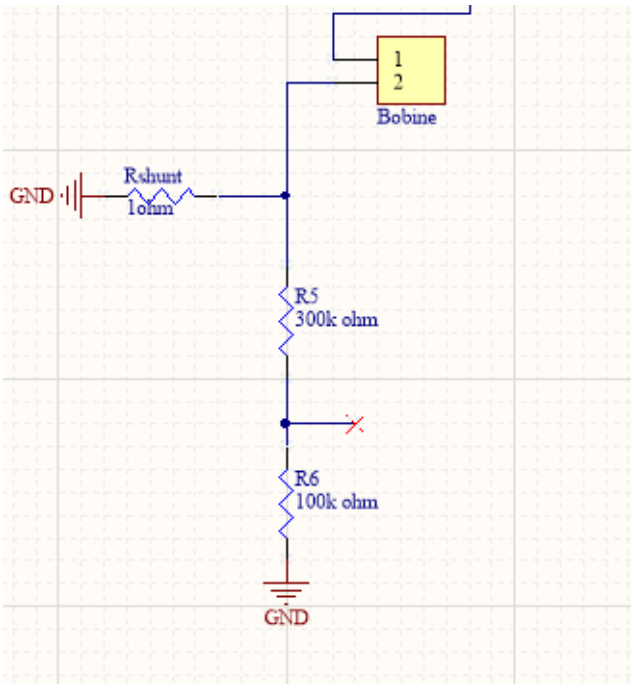

FIGURE A.5: Schéma de l'Ampèremètre

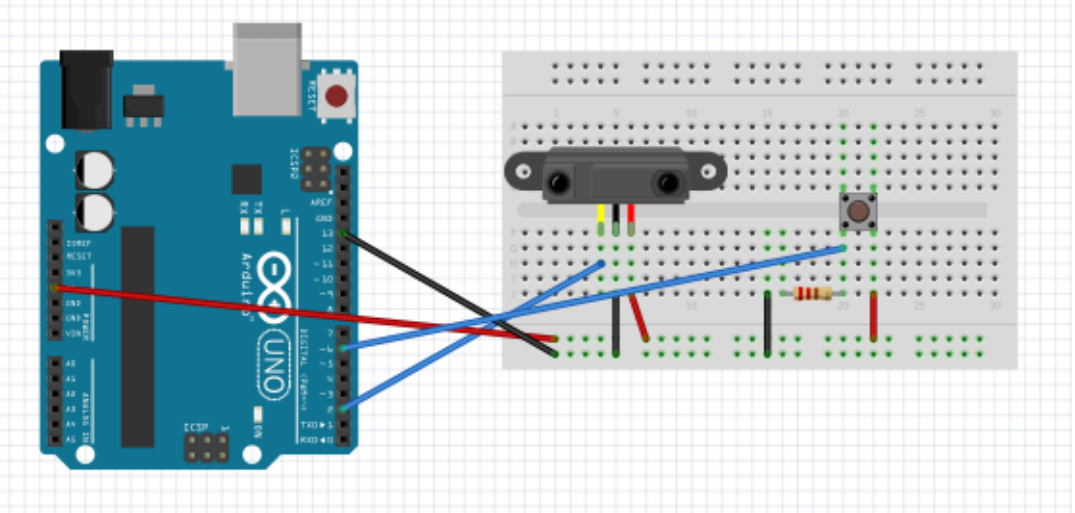

FIGURE A.6: Schéma du Capteur de Passage

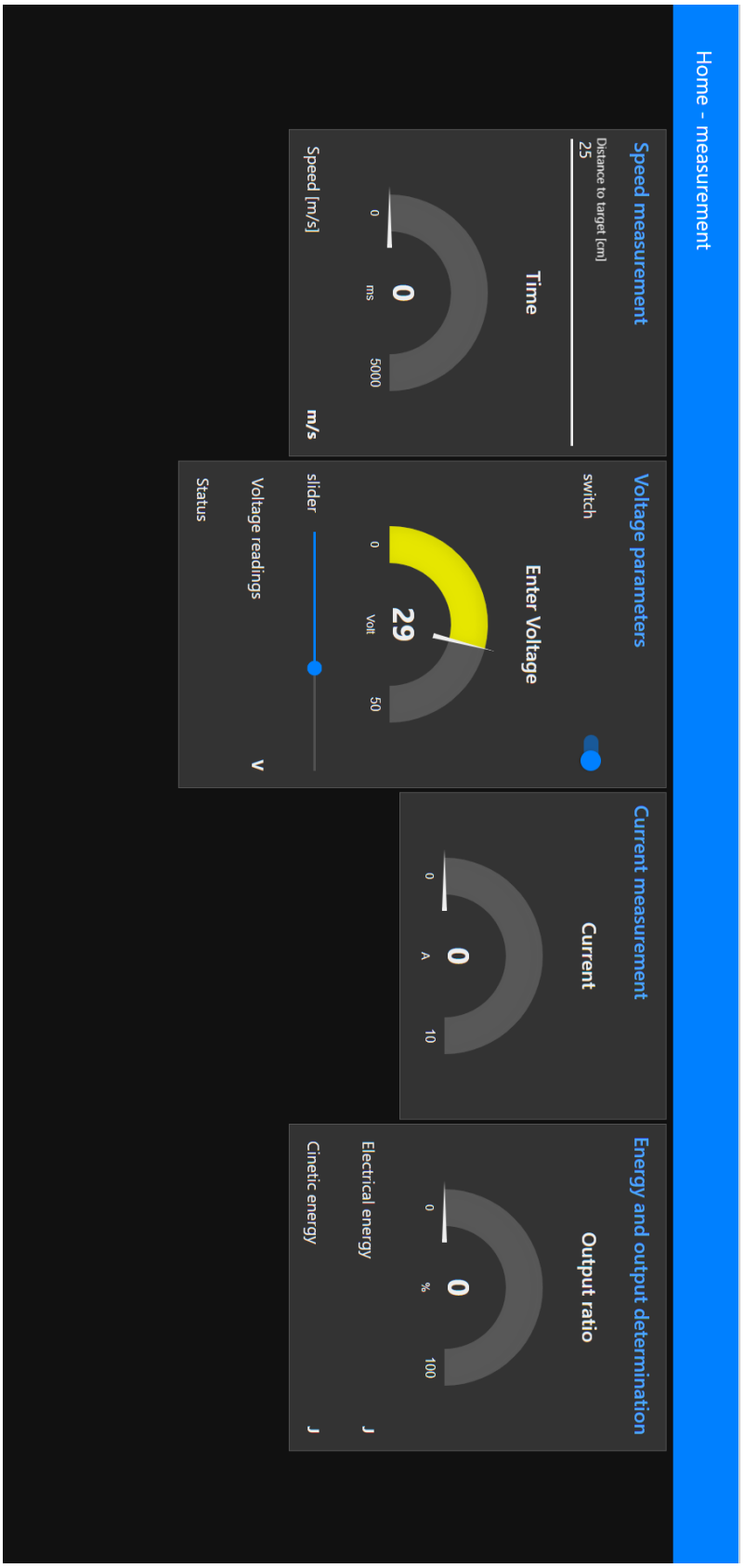

FIGURE A.7: Interface utilisateur

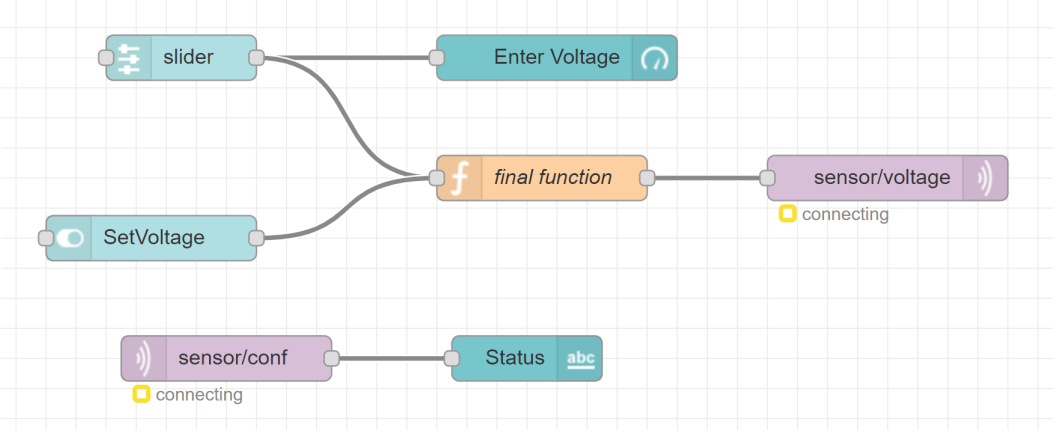

FIGURE A.8: Flux Node-RED pour la partie tension

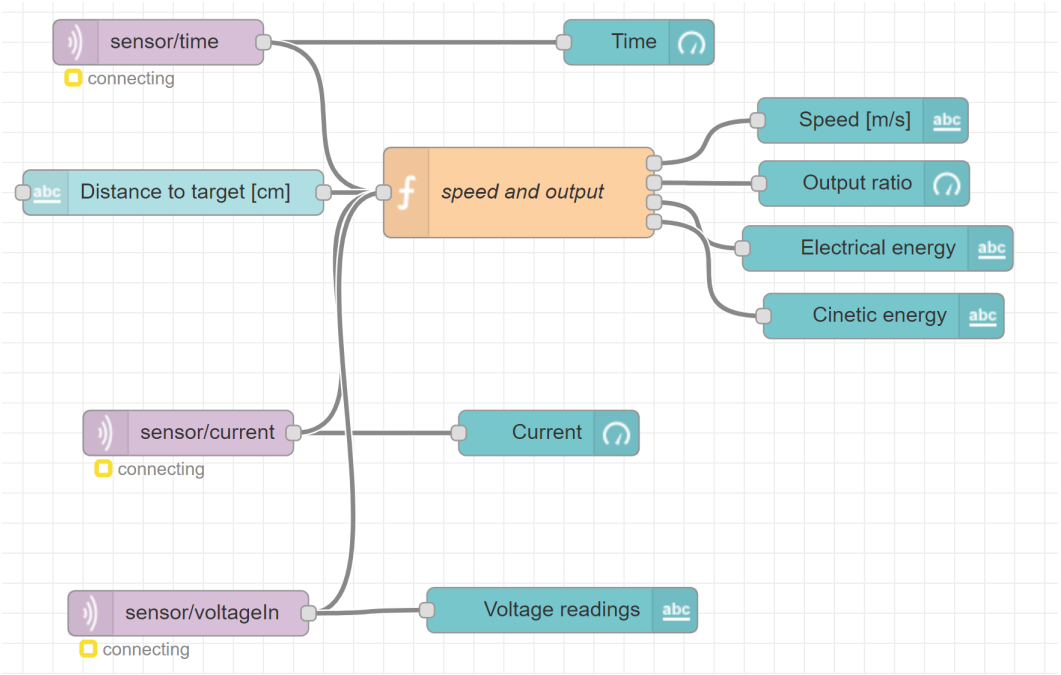

FIGURE A.9: Flux Node-RED pour la partie mesures

A.2 Code

A.2.1 Arduino

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

SoftwareSerial mySerial = SoftwareSerial(10,11);

// set pin numbers
const char IRCaptor = 2;
const int buttonPin = 6;

// initialisation variables
long passageTime;
long impactTime;
float deltaT = 0;
int buttonState = 0;
int departed = 0;
int voltageValue = 0;
int test = 0;
String confirmation = "";
int verif = 0;
float current;
float inputVoltage;

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

  // uart configuration with BBB
  pinMode(10,INPUT);
  pinMode(11,OUTPUT);
  mySerial.begin(9600);

  // configuration of captors and transistors
  pinMode(IRCaptor, INPUT);
  pinMode(buttonPin, INPUT);

}

void loop()
{
  // BBB to arduino


  if(mySerial.available() > 0)
  {
    String storedData = "";
    // reading data into char array
```

```
47      while(mySerial.available())
48      {
49       char inChar = mySerial.read();
50       storedData += inChar;
51      }
52      voltageValue = storedData.toInt();
53      delay(500);
54      // changing verification status to now that a value has been read
55      verif = 1;
56      // only if voltage value chosen is high enough, activate relays
57      if (voltageValue > 14){
58        Serial.print("helloboys\n");
59        digitalWrite(7, HIGH);
60        delay(150);
61        digitalWrite(7, LOW);
62        confirmation = "launched";
63      }
64    }
65
66    // to detect when the projectile has been launched
67    if (digitalRead(IRCaptor) == LOW and departed == 0)
68    {
69      passageTime = millis();
70      Serial.print("detected\n");
71      departed = 1;
72    }
73
74    // read the state of the pushbutton value:
75    buttonState = digitalRead(buttonPin);
76
77    // check if the pushbutton is pressed.
78    // thus projectile has reached target
79    // if it is, the buttonState is HIGH:
80    if (buttonState == HIGH && departed == 1) {
81      Serial.print("Impact\n");
82
83      impactTime = millis();
84      // Time projectile to target = time(buttonPushed) - time(IRcaptor)
85      deltaT = (impactTime - passageTime );
86      Serial.print(deltaT);
87      Serial.print("\n");
88      departed = 0;
89    }
90
91    // sending of the speed of object and output
92    if (deltaT != 0){
93      int speed1 = deltaT;
94      int voltageIn = (analogRead(A1)*(5.0/1023.0)*9.1314);
95      int current = (analogRead(A0)*(5.0/1023.0)*2.241);
96      String voltInS = String(voltageIn);
97      String currentS = String(current);
98      String speed1S = String(speed1);
```

```
99
100     Serial.print("S" + speed1S);
101     // sends data with 'char' to enable recognition of the data
102     mySerial.println("S" + speed1S + "C" + currentS + "V" + voltInS);
103     Serial.print("\n");
104     Serial.print("Test confirmed\n");
105     deltaT = 0;
106     test = 5;
107   }
108
109   // sends status of the launching
110   if (verif != 0){
111     String conf = "not launched";
112     if (confirmation != ""){
113       conf = confirmation;
114     }
115     mySerial.println("l" + conf);
116     verif = 0;
117   }
118 }
```

A.2.2 BeagleBone

a readuart_Arduin-BBB-Dash.py

```
import Adafruit_BBIO.UART as UART

import serial

import paho.mqtt.publish as publish
import paho.mqtt.client as mqtt


# selecting the uart on which you want to communicate
UART.setup("UART1")

disp = serial.Serial(port = "/dev/ttyO1", baudrate = 9600)

disp.close()
disp.open()

while True:
        client = mqtt.Client()

        client.connect("localhost", 1883, 60)
        # initializing lists for string separation
        chars = []
        charsSpeed = []
        charsVoltIn = []
        charsCurrent = []
        charsConf = []
        case1 = 0
        case2 = 0

        if disp.isOpen():

                print "I'm reading\n"
                line = disp.readline()
                print line
                # getting all data retrieved from uart connection in same list
                for i in line:
                        chars.append(i)

                size1 = len(chars)
                i = 0
                while i < size1:
                        # getting speed data
                        if chars[i] == 'S':
                                case1 = 1 #means start of frame
                                i = i + 1
                                while chars[i] != 'C':
```

```
47                                      charsSpeed.append(chars[i])
48                                      i = i + 1
49
50                              # getting voltage data
51                              if chars[i] == 'C':
52                                      i = i + 1
53                                      while chars[i] != 'V':
54                                              charsCurrent.append(chars[i])
55                                              i = i + 1
56
57                              # getting voltage data
58                              if chars[i] == 'V':
59                                      i = i + 1
60                                      while chars[i] != '\r':
61                                              charsVoltIn.append(chars[i])
62                                              i = i + 1
63
64                              # getting confirmation data
65                              if chars[i] == "l":
66                                      case2 = 1 #means end of frame =>status received
67                                      i = i + 1
68                                      while chars[i] != "\r":
69                                              charsConf.append(chars[i])
70                                              i = i + 1
71                              i = size1
72
73                      # publish the different datasets on the right topics through MQTT
74                      if case1 == 1:
75                              valueSpeed = int("".join(map(str, charsSpeed)))
76                              valueVoltIn = int("".join(map(str, charsVoltIn)))
77                              valueCurrent = int("".join(map(str, charsCurrent)))
78                              publish.single("sensor/time", valueSpeed, hostname="localhost")
79                              publish.single("sensor/voltageIn", valueVoltIn, hostname="localhost")
80                              publish.single("sensor/current", valueCurrent, hostname="localhost")
81                      if case2 == 1:
82                              valueConf = ''.join(charsConf)
83                              print 'publish'
84                              publish.single("sensor/conf", valueConf, hostname="localhost")
85
86
87      disp.close()
```

b writeUart_Dash-BBB-Arduino.py

```
import Adafruit_BBIO.UART as UART

import serial

import paho.mqtt.client as mqtt
import paho.mqtt.subscribe as subscribe

client = mqtt.Client()

# selecting the uart on which you want to communicate
UART.setup("UART1")

disp = serial.Serial (port = "/dev/ttyO1", baudrate=9600)

disp.close()
disp.open()

while True:
        if disp.isOpen():

                print "Serial is Open\n"
                client.connect("localhost", 1883, 60)

                # reading the data coming from the topic it is subscribed to
                msg = subscribe.simple("sensor/voltage", hostname="localhost")
                print("%s %s" % (msg.topic, msg.payload))
                msg = msg.payload
                print(msg)
                # sending only the voltage value
                if msg != "true":
                        disp.write(msg)
                disp.write("\n")

print "Sorry!!! You not able to do communicate with device"
disp.close()
```

A.2.3 Node-RED

Fonction dans le flux Node-RED responsable de la récupération des mesures

```
context.data = context.data || {};
var speedp;
var cinetic_energy = 0;
var electrical_energy = 0;
var output = 0;
var voltage = 0;
switch (msg.topic){
    case "sensor/time":
        context.data.time = msg.payload;
        msg = null;
        break;
    case "distance":
        context.data.distance = msg.payload
        msg = null;
        break;
    case "sensor/voltage":
        voltage = msg.payload;
        msg = null;
        break;
    case "sensor/current":
        context.data.currentVal = msg.payload;
        msg = null;
        break;
    default:
        msg = null;
        break;
}

if (context.data.time !== null && context.data.distance !== null){
    var speed = context.data.distance / context.data.time * 10;
    context.data = null;
    speedp = speed.toPrecision(6);
    cinetic_energy = 0.01*(speedp*speedp)/2; //mass = 10g
}
/*we consider power value as a peak
  forming a linear function (first degree) of slope P/delta Time (dt)
  which can be integrated to find energy = E =(P/dt)* (dt)²/2 = (P*dt)/2
  for one side of the peak
  => E = P*dt for both sides of the peak*/
if ((voltage !== null)&&(context.data.currentVal !== null)){
    var power = voltage * context.data.currentVal; //peak of power
    electrical_energy = power*0.05; //dt = 50ms for a 100 ms peak
}

if ((cinetic_energy !== 0)&&(electrical_energy !== 0)){
    output = cinetic_energy/electrical_energy;
}
```

```
48 return[{payload:speedp}, {payload:output}, {payload:electrical_energy},
49        {payload:cinetic_energy}];
```