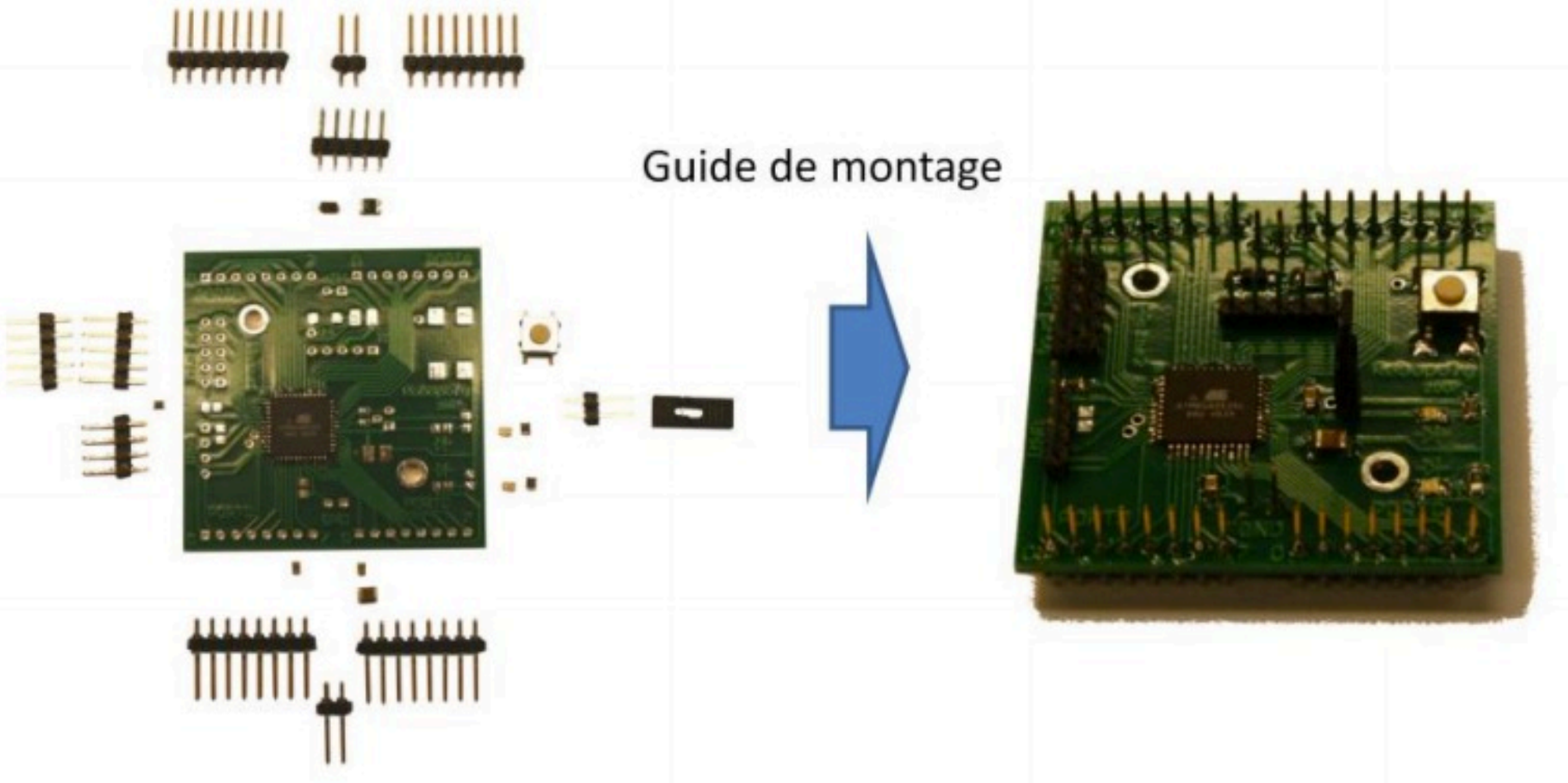
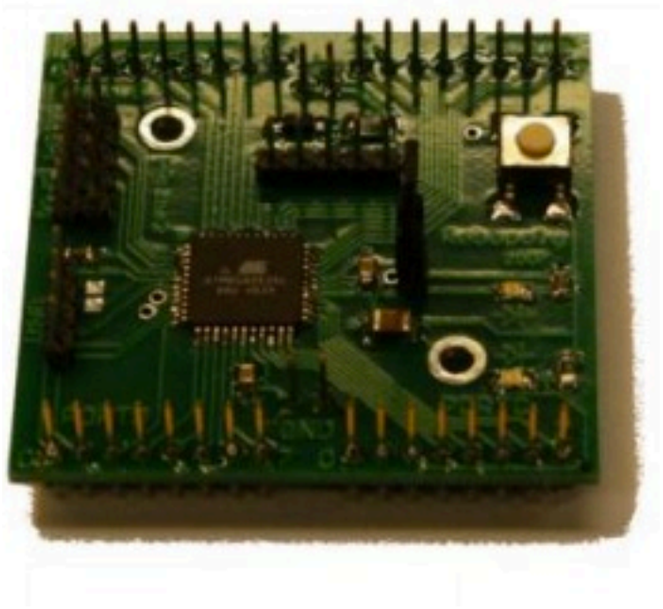
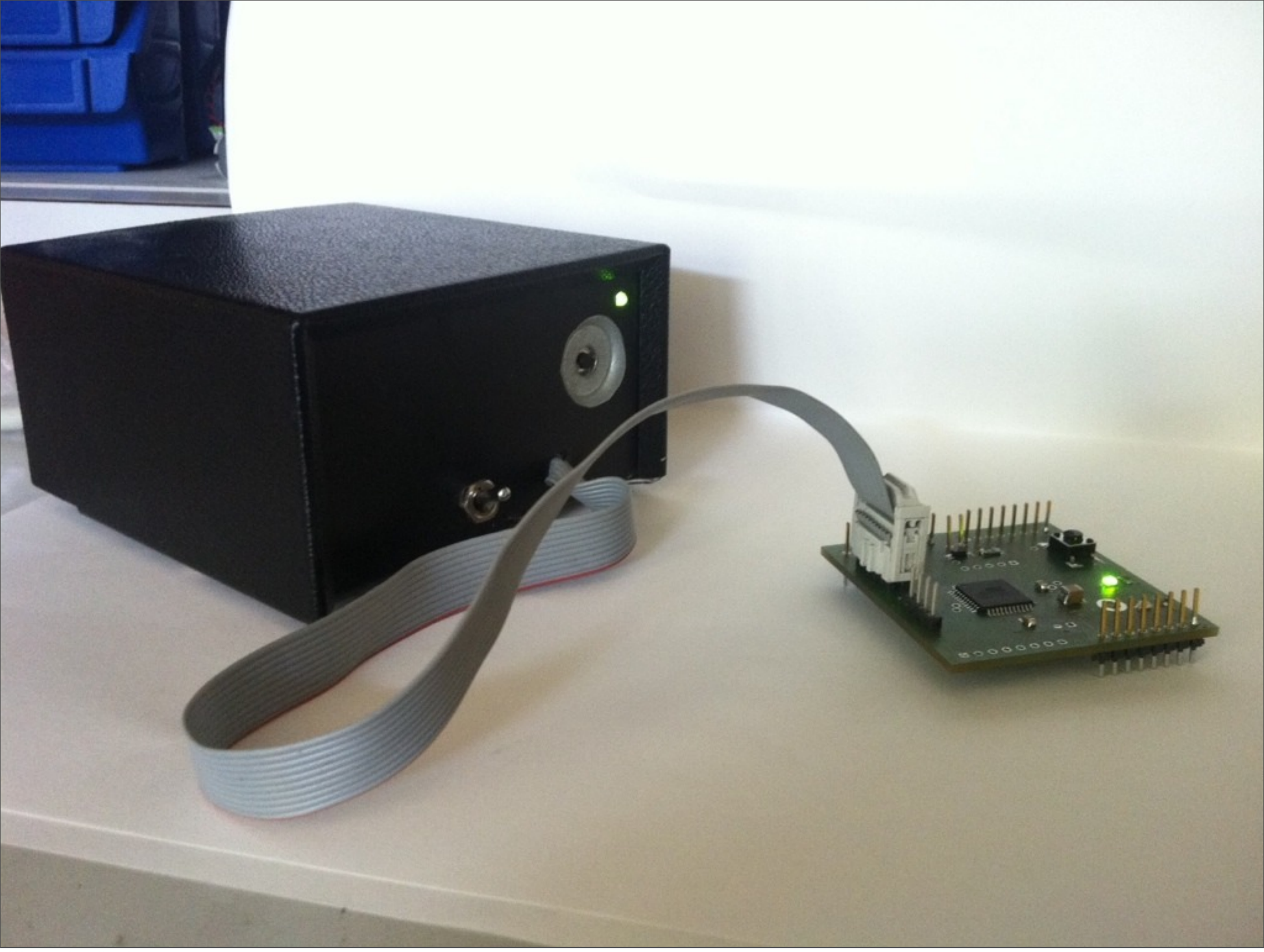

robopoly



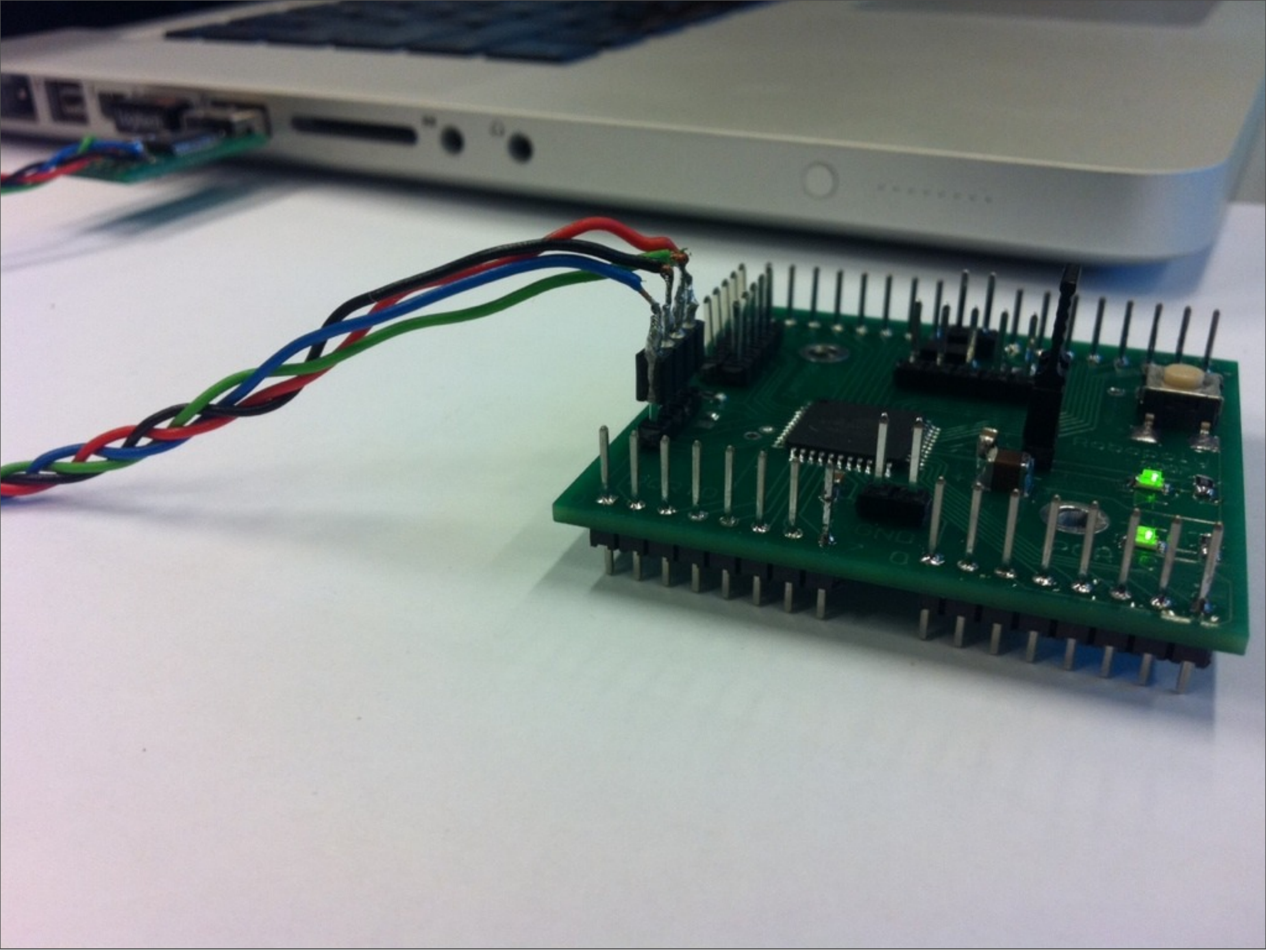
La semaine passée vous avez appris quels étaient les différents composants du kit PRisme, ainsi que la manière de les souder.



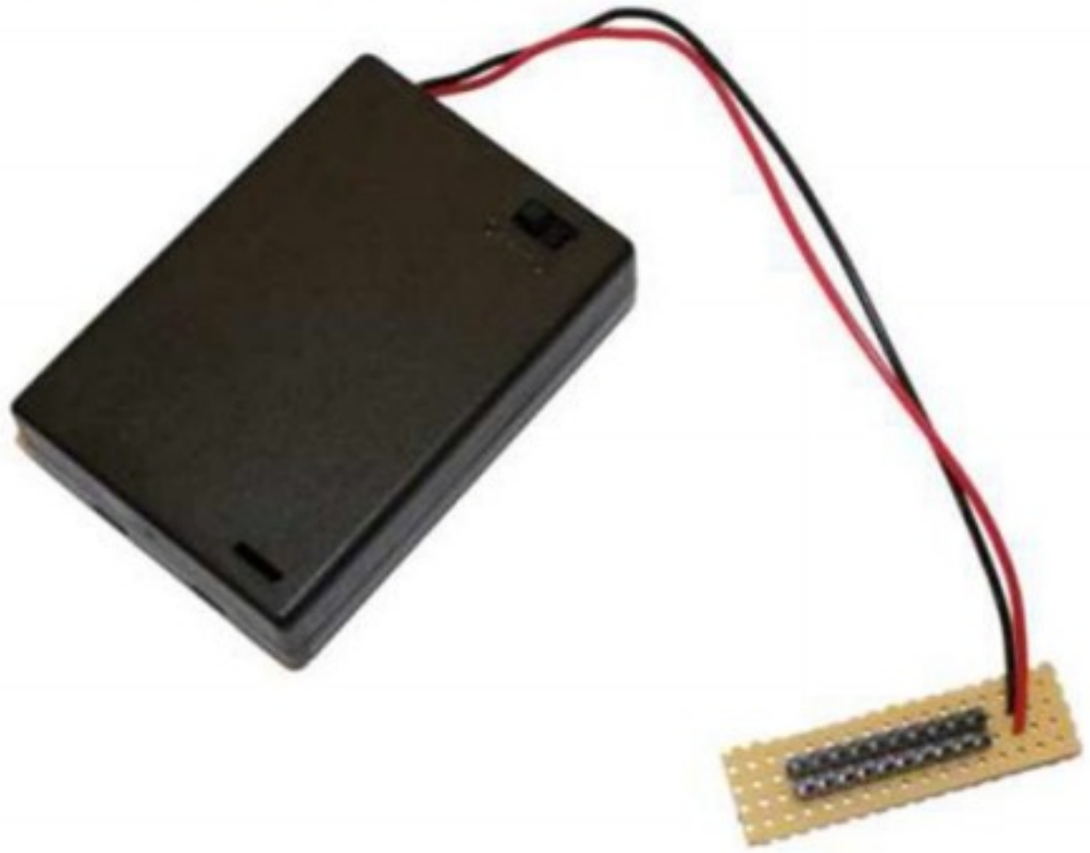
On a maintenant un microcontrôleur monté sur la carte “cerveau” de notre robot, et nous aimerions nous en servir pour commencer à faire des robots. Notre première étape sera la programmation.

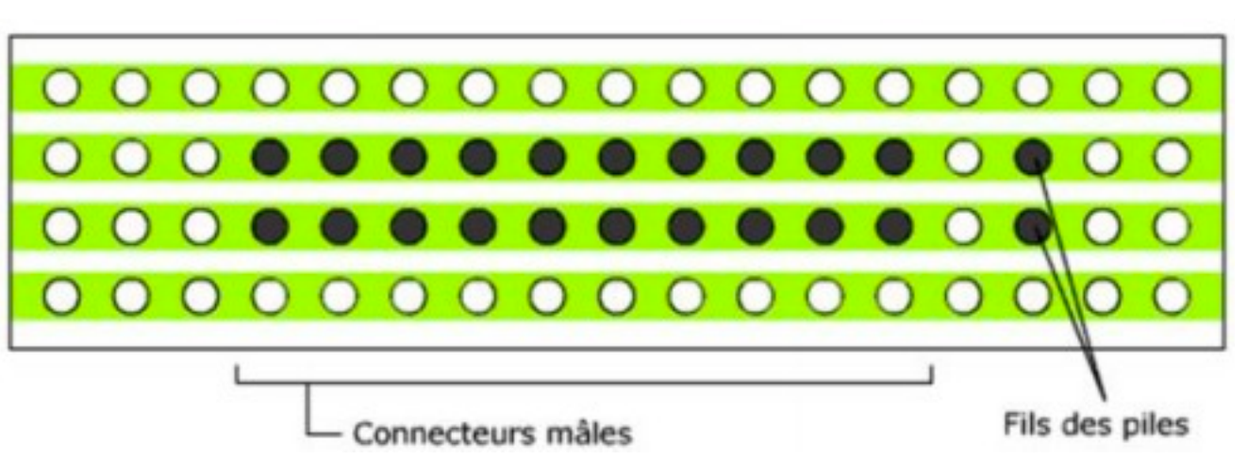


Avant de pouvoir programmer notre microcontrôleur, nous devrons le flasher avec le programmateur de table (en le connectant sur le port SPI) qui se trouve en ELE032. Cela permet de mettre un bootloader sur le microcontrôleur, qui au moment de son reset va vérifier s'il ne reçoit pas du code via le programmateur USB.

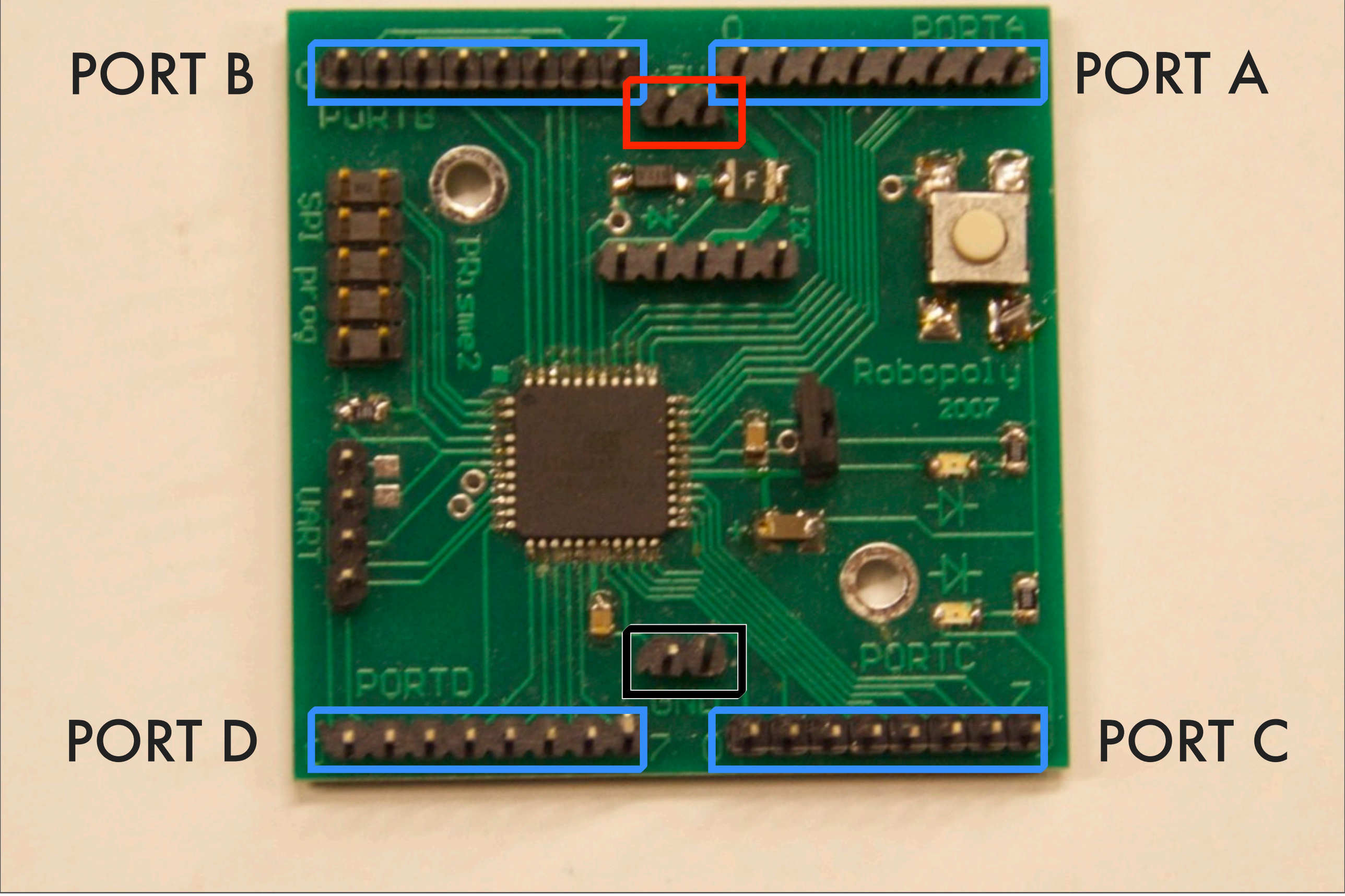


Dorénavant nous pourrons mettre du code sur le microcontrôleur à l’aide du programmateur USB.





Il est nécessaire d’alimenter son microcontrôleur avant de pouvoir s’en servir, nous allons donc assembler un “rack d’alimentation” sur lequel nous pourrons le brancher.



Les pins pour Vcc et la masse sont indiqués ci-dessus.

Notez aussi les 4 ports indiqués en bleu. Il s'agit des ports d'entrée/sortie, chacuns constitués de 8 pins. C'est au travers de ces pins que l'information sort ou entre du microcontrôleur. Cela se fait au travers de 2 niveaux de tensions qui correspondent à 2 niveaux logiques, à savoir : '1' ou la tension d'alimentation Vcc (dans notre cas ce sera 3.6V), et '0' ou la masse (0V).

C'est sur ces ports que nous brancherons nos périphériques tels que les capteurs infrarouges, le pont-H qui gère les moteurs, le servomoteur, etc...

Vous devrez installer AVR Studio 4 afin de pouvoir écrire votre code et le compiler. Vous aurez aussi besoin d'installer le RobopolySetup.exe ainsi que de télécharger les librairies Robopoly pour programmer le kit PRisme.

N'oubliez pas que le Guide de programmation est à votre disposition, et que le Guide de librairie Robopoly est un excellent rappel sur les diverses fonctions de la librairie.

Note : Il est maintenant possible d'utiliser AVR Studio 5 pour programmer votre robot, nous communiquerons plus tard sur la solution que nous proposons.

```c
#include <avr/io.h>
#include "robopoly.h"

void allumerAmpoule(void)
{
   int resultat;

   resultat = digitalRead(A, BYTE);
   digitalWrite (C, BYTE, resultat);
}

int main(void)
{
   while(1)
   {
        allumerAmpoule();
   }

   return 0;
}
```

Programmer le PRisme en C

```c
#include <avr/io.h>
#include "robopoly.h"

void allumerAmpoule(void)
{
   int resultat;

   resultat = digitalRead(A, BYTE);
   digitalWrite (C, BYTE, resultat);
}

int main(void)
{
   while(1)
   {
        allumerAmpoule();
   }

   return 0;
}
```

Librairies

Librairies :
Contiennent des “bouts de code” déjà écrits, et qui pourront être utilisées par nous au travers de fonctions que nous allons exposer plus loin. Cela est très pratique et vous épargne une grosse partie du travail.

```c
#include <avr/io.h>
#include "robopoly.h"

void allumerAmpoule(void)
{
   int resultat;

   resultat = digitalRead(A, BYTE);
   digitalWrite (C, BYTE, resultat);
}

int main(void)
{
   while(1)
   {
        allumerAmpoule();
   }

   return 0;
}
```

main()

Le main constitue le corps principal de notre programme. C’est là dedans que les instructions vont commencer à s’exécuter, et ce de manière séquentielle, jusqu’à l’apparition de l’instruction ‘return 0;’, qui signale la fin du programme.

```
#include <avr/io.h>
#include "robopoly.h"

void allumerAmpoule(void)
{
   int resultat;

   resultat = digitalRead(A, BYTE);
   digitalWrite (C, BYTE, resultat);
}

int main(void)
{
   while(1)
   {
        allumerAmpoule();
   }

   return 0;
}
```

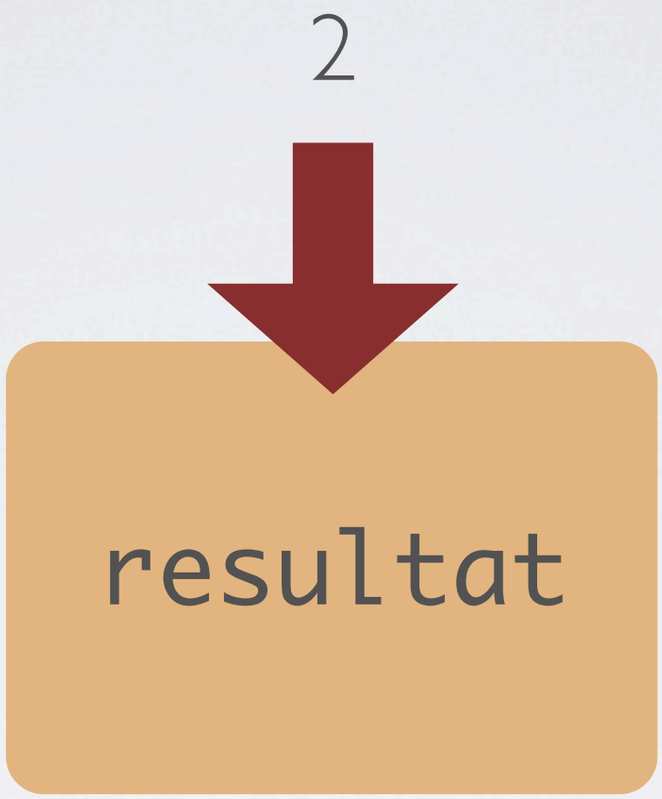
Variables

“int resultat;” correspond à la déclaration d’une variable.

```
int resultat;
```

Déclaration

```
int resultat = 2;
```



Il s’agit simplement d’une “boîte” dans laquelle nous pouvons stocker des valeurs, afin de les réutiliser plus tard dans le programme.

Déclaration

```
int resultat;

float var1 = 1;

int variable_2 = 2;
```

Types

`int`	Nombre entier
`float`	Nombre réel
`char`	Caractère

Utilisation

```
var1 = 3;

resultat = var1 + variable_2;

var1 = variable_2;
```


On peut déclarer une variable avec une variable initiale ou non. Il faut juste prendre garde à ce que le nom de la variable soit en un mot. Dans le cas où vous voullez avoir plusieurs mots, utilisez du camel case ou encore des underscores (“_”).

Il existe plusieurs types de variables :
- les ‘int’ pour integer (ou entier) : ...,-3,-2,-1,0,1,2,3,....
- les ‘float’ pour flottants (ou à virgule) : 42.000
- les ‘char’ pour les caractères, il s’agit d’un nombre entre 0 et 255


;

Le point virgule signale au compilateur qu’il s’agit de la fin d’une instruction. Lorsqu’on a des erreurs de compilation, penser à vérifier si on a pas oublier des ; !!!

```
162 = 0b10100010
```

Décimal Binaire

Il existe plusieurs façon de représenter des nombres, notemment le décimal, l'hexadécimal, l'octal, etc... Mais celui qui nous intéresse plus particulièrement est le binaire.

En effet, un nombre binaire de 8 bits (= 1 octet ou 1 byte) permet de représenter l'état d'un port d'entrée/sortie du microcontrôleur. Chaque bit (0 ou 1) représente le niveau logique de chaque pin du microcontrôleur, numérotés de 0 à 7. Le bit de poids fort se trouvant à gauche correspond au pin 7 et le pin de poids faible se trouvant à droite représente le pin 0.

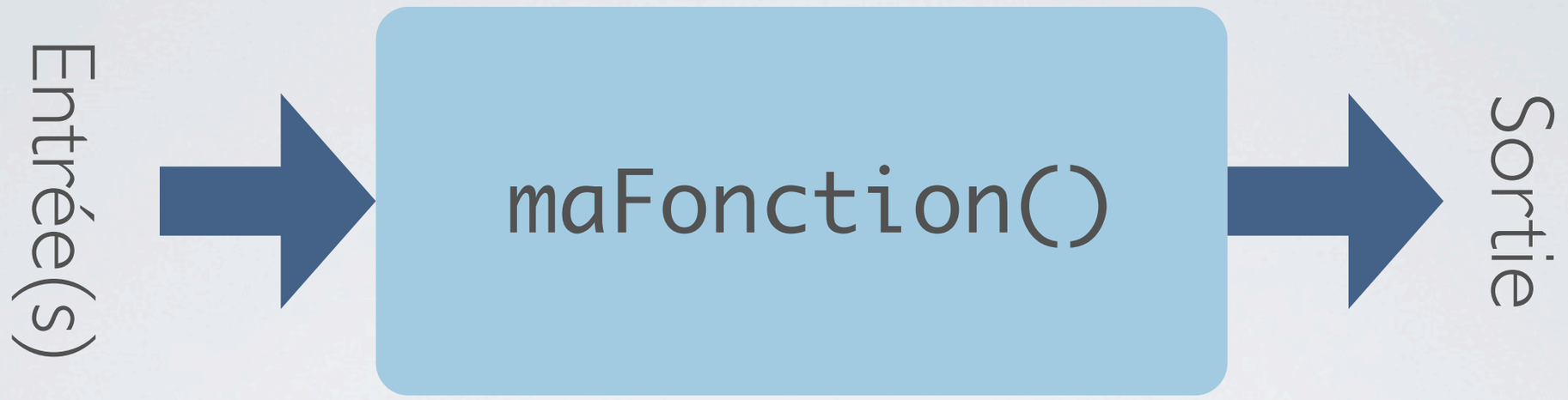
```c
#include <avr/io.h>
#include "robopoly.h"

void allumerAmpoule(void)
{
   int resultat;

   resultat = digitalRead(A, BYTE);
   digitalWrite (C, BYTE, resultat);
}

int main(void)
{
   while(1)
   {
        allumerAmpoule();
   }

   return 0;
}
```

Déclaration

Appel

Fonctions

Les fonctions sont des bouts de code que nous pouvons réutiliser dans notre programme quand nous l'appelons. Il est nécessaire de déclarer une fonction d'abord.




```
int sortie = maFonction(int entree);
```


C’est un peu comme un boîte noire à laquelle nous donnons une (ou des) variable(s) en entrée et qui nous donne une variable en sortie.

```c
int square(int a)
{
    return a*a;
}
```

```c
int b;
b = square(a);
```


Un exemple de fonction mathématique très simple, il s’agit de la fonction carré.

```c
#include <avr/io.h>
#include "robopoly.h"









int main(void)
{
   while(1)
   {
      int resultat;

      resultat = digitalRead(A, BYTE);
      digitalWrite (C, BYTE, resultat);
   }

   return 0;
}
```


L’utilisation de fonction nous permet de simplifier la lecture d’un programme (voir le slide d’après, où la lecture du programme est plus parlante).

Cela permet aussi de simplifier l’écriture de son programme quand nous avons les mêmes sets d’instructions qui reviennent régulièrement.

```c
#include <avr/io.h>
#include "robopoly.h"

void allumerAmpoule(void)
{
   int resultat;

   resultat = digitalRead(A, BYTE);
   digitalWrite (C, BYTE, resultat);
}

int main(void)
{
   while(1)
   {
        allumerAmpoule();
   }

   return 0;
}
```

```
int resultat = digitalRead(port,pin);
```

`port`	A, B, C, D
`pin`	0, 1, 2, 3, 4, 5, 6, 7
	BYTE

Cette fonction nous permet de lire l'état logique du pin d'un port donné (0 ou 1), ou du port entier. Le résultat est stocké dans la variable à gauche de l'égalité.

```
int resultat = digitalRead(A,7);
```

```
resultat = 0 ou 1
```


Dans le cas de la lecture d’un seul pin, le résultat est 0 ou 1.

```
int resultat = digitalRead(A,BYTE);
```

```
resultat = 0b000111111
```


Dans le cas de la lecture d’un port entier (avec BYTE), nous obtenons un nombre en binaire dont chacun des bits représente les pins du port. Dans cet exemple, on a les pins 7 à 5 à ‘0’ et les pins 4 à 0 à ‘1’.

```
digitalWrite(port,pin,value);
```

port	A, B, C, D
pin	0, 1, 2, 3, 4, 5, 6, 7 BYTE
value	0,1 0bxxxxxxxx

On peut aussi écrire une valeur sur un pin d'un port ou sur un port entier, ce qui permet de mettre les tensions correspondant aux niveau logiques respectiques sur ces pins.

```
digitalWrite(C,2,1);
```


Dans ce cas on met le pin 2 du port C à ‘1’, c’est à dire la tension Vcc = 3.6V.

```
digitalWrite(C,BYTE,0b01000011);
```

```
int resultat;

resultat = digitalRead(A, BYTE);
digitalWrite (C, BYTE, resultat);
```


Dans l'exemple ci-dessus on met les pins 6, 1 et 0 du port C à '0', et tout le reste à '1'.

Dans l'exemple du dessous, on lit tous les pins du port A et le résultat est stocké dans la variable résultat. On copie ensuite cette valeur dans le port C. Au final, nous effectuons une copie des entrées du port A vers les sorties du port C.

```c
#include <avr/io.h>
#include "robopoly.h"

void allumerAmpoule(void)
{
  int resultat;

  resultat = digitalRead(A, BYTE);
  digitalWrite (C, BYTE, resultat);
}

int main(void)
{
  while(1)
  {
      allumerAmpoule();
  }

  return 0;
}
```


Vous pouvez maintenant comprendre ce que fait ce code !

```
#include <avr/io.h>
#include "robopoly.h"

void allumerAmpoule(void)
{
  int resultat;

  resultat = digitalRead(A, BYTE);
  digitalWrite (C, BYTE, resultat);
}

int main(void)
{
  while(1)
  {
      allumerAmpoule();
  }

  return 0;
}
```


A l’exception de cette instruction while...

```
while(condition)
{
    INSTRUCTION 1;
    INSTRUCTION 2;
    INSTRUCTION 3;
    INSTRUCTION 4;

}
```

En principe, les instructions s’exécutent séquentiellement, mais si nous voulons qu’elles s’éxecutent plusieurs fois (ce qui forme une boucle), nous pouvons utiliser des instructions qui agissent sur le flux des instructions tel que l’instruction while.

Cette instruction répètera le code qui se trouve entre les accolades ({ et }) tant que la condition entre parenthèste est ‘vraie’, c’est à dire, ne vaut pas ‘0’.

```
while(1)
{

    INSTRUCTION 1;
    INSTRUCTION 2;
    INSTRUCTION 3;
    INSTRUCTION 4;

}
```

Mettre un 1 comme paramètre pour le while fait que la boucle sera infinie, et les instructions entre les accolades tourneront en boucle tant que le robot est allumé. C’est très pratique dans notre cas !

Égalité	`test = (a == b)`
Différence	`test = (a != b)`

Inférieur (ou égal)	`a < b`	`(a <= b)`
Supérieur (ou égal)	`a > b`	`(a >= b)`

Opérateurs logiques

Voici quelques opérateurs logiques qui vous permettront d’écrire des conditions. Lorsque le résultat est VRAI, il vaut ‘1’. S’il est FAUX, il vaut ‘0’.

```
if(condition)
{
  SET D'INSTRUCTIONS A
}
else
{
  SET D'INSTRUCTIONS B
}
```


On peut aussi faire agir notre robot selon une condition avec un if(condition)... else (si la condition est vraie ... sinon ...)

```
if(verre_plein == 1)
{
     bois();
}
```

Test logique

```
while(1)
{
    if(verre_plein == 1)
    {
        bois();
    }
    else
    {
        remplir_verre();
    }
}
```

Test logique

Un exemple très simple de if / else


BENDËRBRAU
I approve, meatbags!
Cold-fusion Stream Beer

```
int boutons = digitalRead(A,BYTE);

if(boutons == 0b11111101)
{
    digitalWrite(C,BYTE,0b11111111);
}
else
{
    digitalWrite(C,BYTE,0b00000000);
}
```


Dans l’exemple ci-dessous, si le pin 1 du port A est à ’0’ et le reste des pins sont à ‘1’, il va mettre tous les pins du port C à ‘1’. Si ce n’est pas le cas, il les met tous à ‘0’.

```
waitms(temps);          // en millisecondes

waitus(temps);          // en microsecondes
```

Boucles d'attente

On peut effectuer des boucles d'attentes qui permettent au robot de ne rien faire et attendre pendant un certain temps. On a une fonction pour faire des attentes de quelques microsecondes, et une autre pour les attentes de quelques millisecondes.

// Commentaires !

```
int main(void)
{
   while(1)
   {
      waitms(500);                          // Attendre 500ms
      digitalWrite(C,BYTE,0b11110000);  // Mettre tous les pins du port a 1
      waitms(500);                          // Attendre 500ms
      digitalWrite(C,BYTE,0b00001111);  // Mettre tous les pins du port a 0
   }

   return 0;
}
```

Clignotement

Un exemple d’utilisation de la fonction waitms();

La semaine prochaine...

Le prochain démon portera sur l'utilisation des moteurs et le PWM, ce qui vous permettra de faire déplacer votre robot !