

Robot Physique Strasbourg

Arduino - Kicad

Formation bras robot

<https://github.com/Anna-Clem/RobotArm>

Annaïg CLEMENT
3A généraliste ESE/MNE
(couteau suisse de l'asso)
7 novembre 2020

Table des matières

1	Objectifs	2
2	Prise en main sur Arduino	3
2.1	Les servo moteur	3
	Principe de base	3
	Une routine pour le bras	3
2.2	L'encodeur	3
	Prise en main	3
	Code avec interruptions	6
2.3	L'écran LCD	7
	Principe	7
	Fonctions pour le bras	11
3	Création du circuit imprimé	12
3.1	Création du schéma électronique	12
	1ère étape : Trouvez les bons composants et les poser sur le schéma	12
	2ème étape : Relier les composants entre eux	12
	3ème étape : Test des règles électriques	14
	4ème étape : Assigner les empreintes des composants	14
3.2	Routage du PCB	15
	1ère étape : Placer les composants de manière optimale	15
	2ème étape : Tout relier (sauf le GND pour le moment)	15
	3ème étape : Tracé du contour de la carte	15
	4ème étape : Plan de masse (GND)	15
	5ème étape : Ajouter les mounting holes (optionnel mais c'est toujours mieux)	15
	6ème étape : Valider la carte	16
	7ème étape : Visualisation de la carte (optionnel)	16

Table des figures

1	Servomoteur sg90	3
2	Encodeur rotatif HW-040	4
3	Branchement nécessaire pour l'encodeur	4
4	Principe de l'encodeur rotatif	5
5	Code exemple de l'encodeur rotatif	6
6	Code interruption encodeur	7
7	Ecran LCD	8
8	Branchement du LCD	8
9	Résultat LCD	9

1 Objectifs

Cette formation a pour but¹ de créer un bras robotisé que l'on contrôlera avec un encodeur et un écran LCD. Des routines seront à implémenter et l'encodeur permettra de choisir une de ces différentes routines directement visible sur l'écran LCD. Nous aurons également à créer une carte mère pour ce bras afin que le système soit le plus compact possible.

Les différents logiciels que nous utiliserons seront :

- Arduino pour l'implémentation du code qui contrôlera le robot.
- Kicad pour la création d'un circuit imprimé propre au bras.

Nous aurons également besoin de ces différents composants pendant les tests :

- Un microcontrôleur Arduino nano
- 4 servo moteurs SG90 pour le contrôle du bras.
- Un écran LCD 2004A avec son module de communication i2C
- Un encodeur pour naviguer sur l'écran LCD
- Des fils

Les différents composants pour la création de la carte :

- Connecteurs pour brancher les 4 servomoteurs
- Connecteur pour brancher l'écran i2C
- Connecteur pour brancher l'encodeur rotatif
- Régulateur 5V : "R-785.0-1.0"
- Bornier à vis pour brancher la batterie.
- Diode de protection pour la batterie : "SSA34-E3/5AT" (pour ne pas cramer le circuit si on branche la batterie à l'envers)
- Deux leds
- Résistances 1kOhm pour chaque led
- Capacités de découplage pour les composants (pour éviter les pics de courants)
(Capacité 100uF : PCE3750CT-ND // Capacité 10uF : 399-5091-1-ND)

1. Bien entendu le véritable objectif derrière cette formation est l'utilisation des différents logiciels et la compréhension des composants afin de pouvoir les utiliser et les implémenter vous-même en vue de la Coupe de France de Robotique ou de vos projets personnels.

2 Prise en main sur Arduino

2.1 Les servo moteur

Principe de base

On utilisera des Servomoteurs sg90 pour ce projet.



FIGURE 1 – Servomoteur sg90

Le lien ci dessous explique un peu en détail le fonctionnement et montre les fonctions utiles :

Source à lire pour le servomoteur : <https://arduino.developpez.com/tutoriels/arduino-a-l-ecole/?page=projet-12-utiliser-un-servomoteur>

Vous y trouverez également un premier programme basique pour utiliser un servomoteur.

Une routine pour le bras

Attention aux contraintes d'angles ici une fois les servomoteurs montés sur le bras.

En premier lieu, vous devrez trouver les angles min et max pour chaque servomoteur avant de les coder tous ensemble.

Pas de code solution pour cette partie, vous trouverez une ébauche de routine dans le programme final (sur le github).

Il est bon de remarquer qu'il faudrait utiliser une batterie pour alimenter correctement en courant les servomoteurs.

2.2 L'encodeur

Prise en main

Voici l'encodeur utilisé pour ce projet.

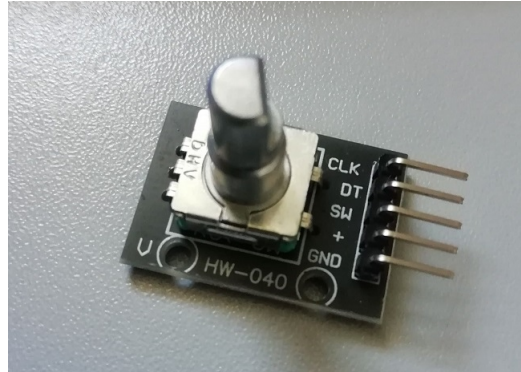


FIGURE 2 – Encodeur rotatif HW-040

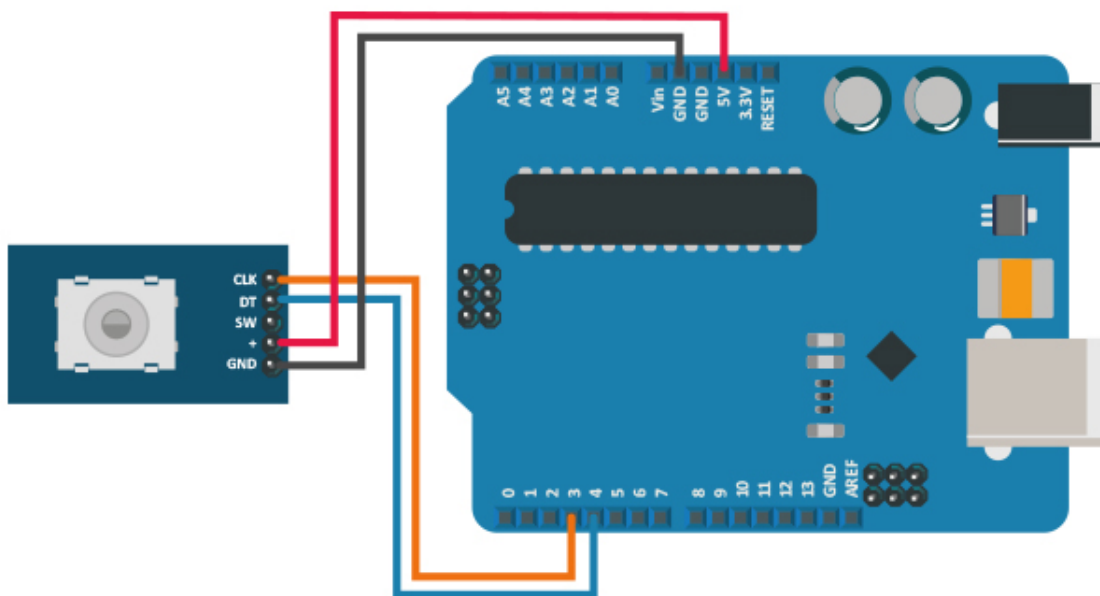


FIGURE 3 – Branchement nécessaire pour l'encodeur

- On notera que la borne "+" de l'encodeur peut être connecté au 3V3 ou au 5V de l'arduino.
- En ce qui concerne la pin "SW" elle correspond au Switch de l'encodeur, en soit le bouton. Sa valeur par défaut est 1, si on appuie, la valeur passe à 0. On branchera cette pin ici à la pin D5 de l'arduino.

Fonctionnement de l'encodeur :

source : <https://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/>

Les deux pins A (CLK) et B (DT) fournissent chacune un signal carré en quadrature. Ci-dessous en bleu, OUTPUT A et en vert OUTPUT B.

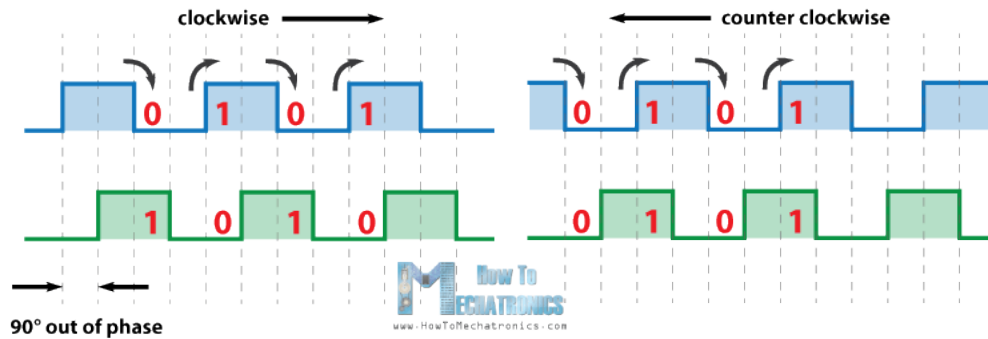


FIGURE 4 – Principe de l'encodeur rotatif

Si A et B ont les mêmes valeurs de sortie alors l'encodeur tourne dans le sens inverse des aiguilles d'une montre. Dans le cas où A et B ont des sorties différentes alors on tourne dans le sens des aiguilles d'une montre.

Le code ci-dessous permet à l'utilisateur de bien se rendre compte de l'enjeu des deux pins servant à la rotation. On a besoin de mettre les trois sorties en INPUT_PULLUP afin d'éviter que les pins soient flottantes. Ce mode permet de rajouter une résistance de pull-up interne.

Exercice : Prendre en main le bouton de l'encodeur rotatif. On en aura besoin pour sélectionner des items sur l'écran LCD. Le comportement est similaire à celui d'un bouton poussoir classique :-). Libre à vous de vous amuser avec les valeurs en sortie de l'encodeur.

```

1 #include <Arduino.h>
2 #include <Wire.h>
3
4 int pinA = 3; // Connected to CLK
5 int pinB = 4; // Connected to DT, donne la direction de rotation
6 int button = 5; // Connected to SW
7 bool state = 1;
8
9 int counter = 0;
10 int position_init;
11 int position_fin;
12
13 void setup() {
14   pinMode (pinA, INPUT_PULLUP);
15   pinMode (pinB, INPUT_PULLUP);
16   pinMode (button, INPUT_PULLUP);
17   position_init = digitalRead(pinA);
18   Serial.begin (9600);
19 }
20
21 void loop() {
22   position_fin = digitalRead(pinA);
23
24   if (position_fin != position_init){ // on verifie que l'encodeur tourne
25
26     if (digitalRead(pinB) != position_fin) { //On tourne dans le sens
27       horaire
28       counter ++;

```

```

28     Serial.print("Encoder_Position:_");
29     Serial.println(counter);
30 }
31 else { //On tourne dans le sens inverse trigo
32     counter --;
33     Serial.print("Encoder_Position:_");
34     Serial.println(counter);
35 }
36 }
37 ////////////// Exercice : //////////
38 /* Ajouter une fonction qui fait en sorte de remettre la variable counter
39    a 0 des qu'on appuie sur le bouton.*/
40 position_init = position_fin;
41 }

```

FIGURE 5 – Code exemple de l'encodeur rotatif

Remarque : Plus on rajoute de commandes dans notre code, plus le délai d'exécution sera long, ce qui peut entraîner des erreurs lors de la lecture de OUTPUT A et OUTPUT B en fonction de la valeur précédente en mémoire. L'encodeur ne sera plus fiable. On doit faire appel à ce qu'on appelle des "interruptions".

Code avec interruptions

Principe de l'interruption : *Source pour informations (à lire)* <https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/>

L'interruption permet de résoudre les erreurs de timing dans le code. Dans le cas de notre encodeur, nous en avons besoin pour pouvoir lire les changements d'état à n'importe quel moment de notre code afin qu'il soit fiable.

On fera attention aux pins permettant les interruptions sur l'arduino nano (pin D2 et D3)

Exercice : Reconstruire le code précédent avec des interruptions pour la rotation de l'encodeur et le bouton poussoir.

```

1  #include <Arduino.h>
2
3  /*!\ Seules les pin 2 et 3 de l'arduino compatible avec les interruptions
4     */
5
6  int pinA = 2;  // Connected to CLK
7  int pinB = 4;  // Connected to DT
8  int button = 3; // Connected to SW
9
10 int counter = 0;
11 int position_fin;
12 //Ajouter autres variables ?
13 void ISR_rot() {
14     //A completer
15 }
16

```

```
17 void ISR_button() {
18     //A completer
19 }
20
21 void encodeur_rotation() {
22     //A completer
23 }
24 }
25
26 void click() {
27     //A completer
28 }
29
30 void setup() {
31     pinMode(pinA, INPUT_PULLUP);
32     pinMode(pinB, INPUT_PULLUP);
33     pinMode(button, INPUT_PULLUP);
34     //A completer (2 interruptions a definir ISR_rot + ISR_button)
35     Serial.begin(9600);
36 }
37
38 void loop() {
39     if (rotation) {
40         rotation = 0;
41         encodeur_rotation();
42     }
43
44     if (pushed_button) {
45         pushed_button = 0;
46         click();
47     }
48 }
```

FIGURE 6 – Code interruption encodeur

2.3 L'écran LCD

Principe

L'écran LCD utilisé ici sera le 2004A avec son module i2C. C'est un écran 4 lignes.

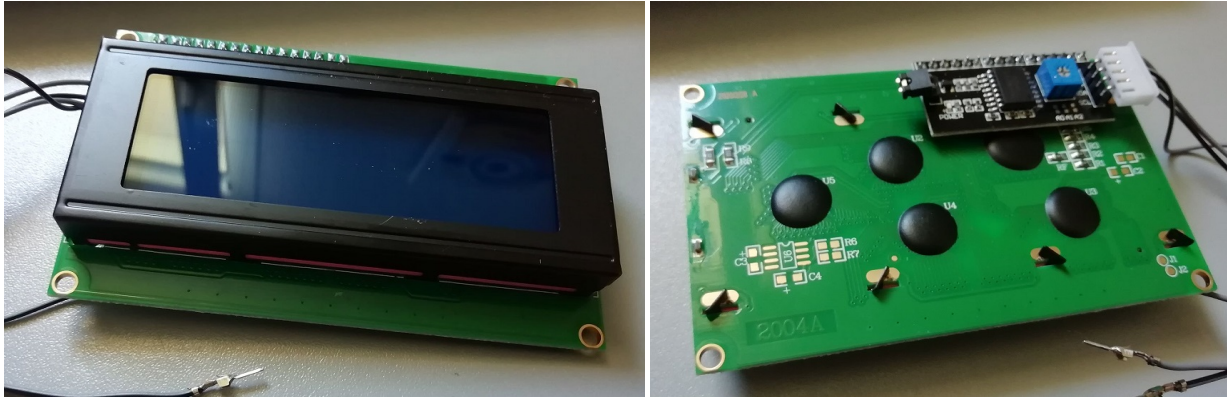


FIGURE 7 – Ecran LCD

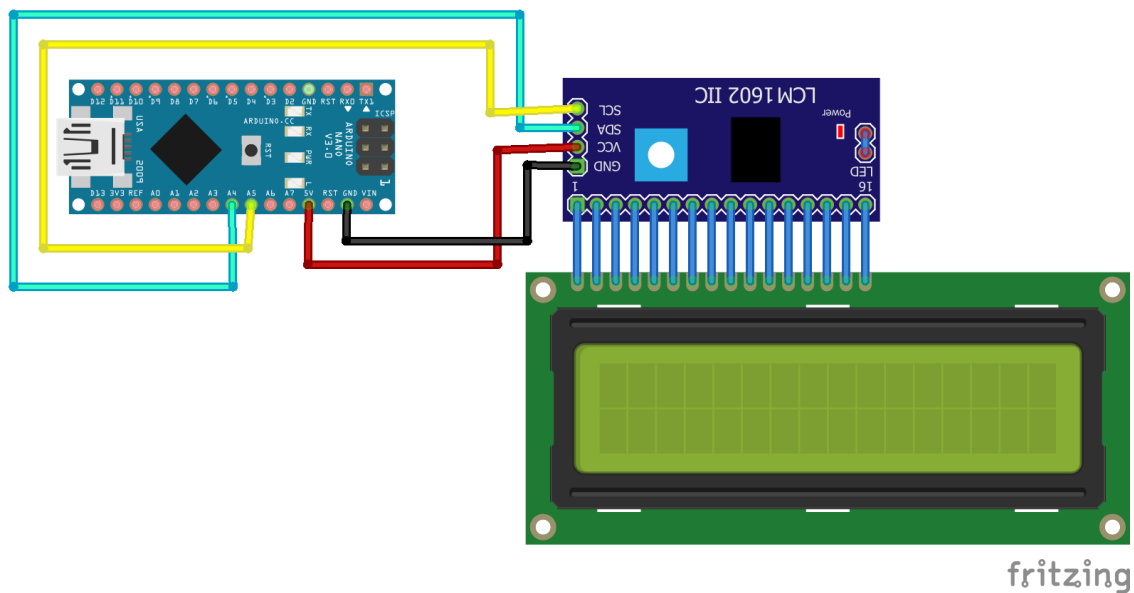


FIGURE 8 – Branchement du LCD

Information sur la bibliothèque LiquidCrystal_I2C :

https://github.com/johnrickman/LiquidCrystal_I2C

Pour créer des logo personnalisés à ajouter sur le LCD :

(à noter que la mémoire du LCD ne permet la création que de 8 logos simultanément *CreateChar(0-7, byte[])*. Si plus sont créés alors les premiers seront écrasés)

<https://maxpromer.github.io/LCD-Character-Creator/>

On va profiter de cette partie pour créer l'écran de bienvenue de notre interface afin de piloter le bras.

Voici le résultat attendu à la fin de cette partie.

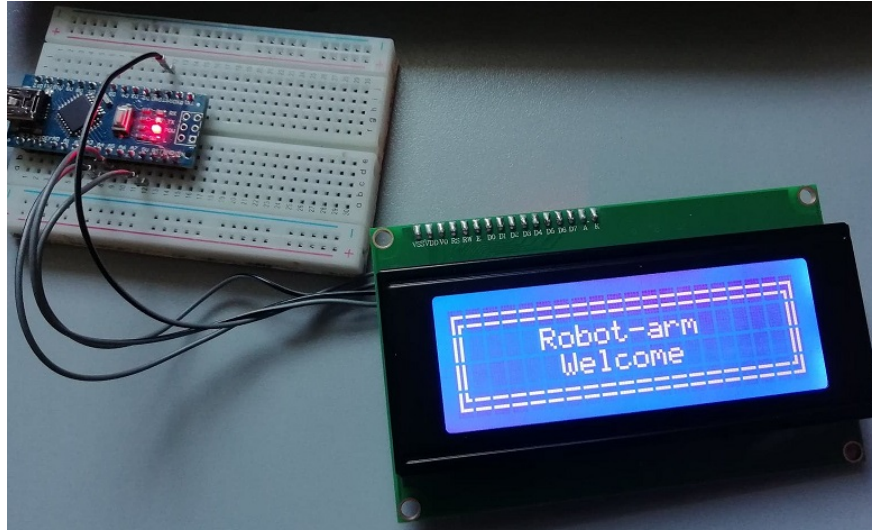


FIGURE 9 – Résultat LCD

```
1
2 #include <Arduino.h>
3 #include <Wire.h>
4 #include <LiquidCrystal_I2C.h>
5
6 LiquidCrystal_I2C lcd(0x27,20,4); // set the LCD address to 0x27 for a 16
   chars and 4 line display
7
8 unsigned int i = 0;
9
10 byte bord_hb[] = {
11     B00000,
12     B00000,
13     B11111,
14     B00000,
15     B00000,
16     B11111,
17     B00000,
18     B00000
19 };
20 byte bord_gd[] = {
21     B01010,
22     B01010,
23     B01010,
24     B01010,
25     B01010,
26     B01010,
27     B01010,
28     B01010
29 };
30 byte coin_h_d[] = {
31     B00000,
32     B00000,
33     B11110,
```

```
34 | B00010,
35 | B00010,
36 | B11010,
37 | B01010,
38 | B01010
39 | };
40 | byte coin_h_g[] = {
41 |     B00000,
42 |     B00000,
43 |     B01111,
44 |     B01000,
45 |     B01000,
46 |     B01011,
47 |     B01010,
48 |     B01010
49 | };
50 | byte coin_b_g[] = {
51 |     B01010,
52 |     B01010,
53 |     B01011,
54 |     B01000,
55 |     B01000,
56 |     B01111,
57 |     B00000,
58 |     B00000
59 | };
60 | byte coin_b_d[] = {
61 |     B01010,
62 |     B01010,
63 |     B11010,
64 |     B00010,
65 |     B00010,
66 |     B11110,
67 |     B00000,
68 |     B00000
69 | };
70 |
71 | void setup()
72 | {
73 |     lcd.init(); // initialize the lcd
74 |     lcd.backlight();
75 |     lcd.clear();
76 |     lcd.begin(16, 4);
77 |
78 |     lcd.createChar(0, coin_h_g);
79 |     lcd.createChar(1, bord_hb);
80 |     lcd.createChar(2, coin_h_d);
81 |     lcd.createChar(3, bord_gd);
82 |     lcd.createChar(4, coin_b_d);
83 |     lcd.createChar(5, coin_b_g);
84 |
85 |
86 | /////// Exercice : ///
```

```
87 |
88 | //Afficher le cadre avec les logos deja crees et le texte.
89 | //La fonction lcd.setCursor(column, row) place le curseur de l'utilisateur
    | la ou il souhaite ecrire.
90 | //Les fonctions lcd.write() et lcd.print() permette d'ecrire sur l'ecran
    | des caracteres et chaines de caracteres. La difference est dans le type
    | de donnees qu'elles exploitent.
91 | //On peut utiliser des boucles "for" comme en C/C++ pour alléger le code.
92 | }
93 |
94 | void loop()
95 | {
96 | }
```

Fonctions pour le bras

Des exercices à faire de A à Z en s'inspirant des programmes précédents pour regrouper les différentes notions vues.

- Encodeur + LCD : Afficher un curseur sur l'écran (pour sélectionner une routine par la suite).
- Ajouter les servos pour une routine. (Si on clique ça doit bouger)

Les deux fonctions ci-dessus sont visibles dans le code final ([github](#))

Ci-dessous, d'autres idées pour améliorer le bras : (ne sera pas dans le code de la formation, pas de led sur la carte prototype disponible mais vous pourrez imprimer votre propre carte une fois la formation kicad passé)



- Menu "Personnaliser" sur l'écran avec comme principe :
 - on clique sur personnaliser : affichage "Servo base angle : XX°"
 - on peut tourner l'encodeur pour choisir la valeur (comme un potentiomètre)
 - on clique pour passer au servomoteur suivant.
- Amusez-vous et codez ce que vous voulez! :)

3 Création du circuit imprimé

Le but ici est de créer la carte mère de ce projet.

Il est vivement recommandé d'avoir effectuée la formation de base "CAO électronique" avant, toutes les étapes ne sont pas détaillées ici. Seuls quelques points essentiels sont repris afin de vous débloquer.

On va procéder en deux parties :


-  Création du schéma
-  Routage du PCB

Je rappelle ici les différents composants de notre PCB

- Microcontrôleur Arduino Nano
- Connecteurs pour brancher les 4 servomoteurs
- Connecteur pour brancher l'écran i2C
- Connecteur pour brancher l'encodeur rotatif
- Régulateur 5V : "R-785.0-1.0"
- Bornier à vis pour brancher la batterie.
- Diode de protection pour la batterie : "SSA34-E3/5AT" (pour ne pas cramer le circuit si on branche la batterie à l'envers)
- Deux leds.
- Resistances 1kOhm pour chaque led
- Capacités de découplage pour les composants (pour éviter les pics de courants)
(Capacité 100uF : PCE3750CT-ND pour la batterie
Capacité 10uF : 399-5091-1-ND pour l'encodeur rotatif et le régulateur 5V)








3.1 Création du schéma électronique

1ère étape : Trouvez les bons composants et les poser sur le schéma

Pour ajouter un élément sur le schéma, cliquez sur . (A)

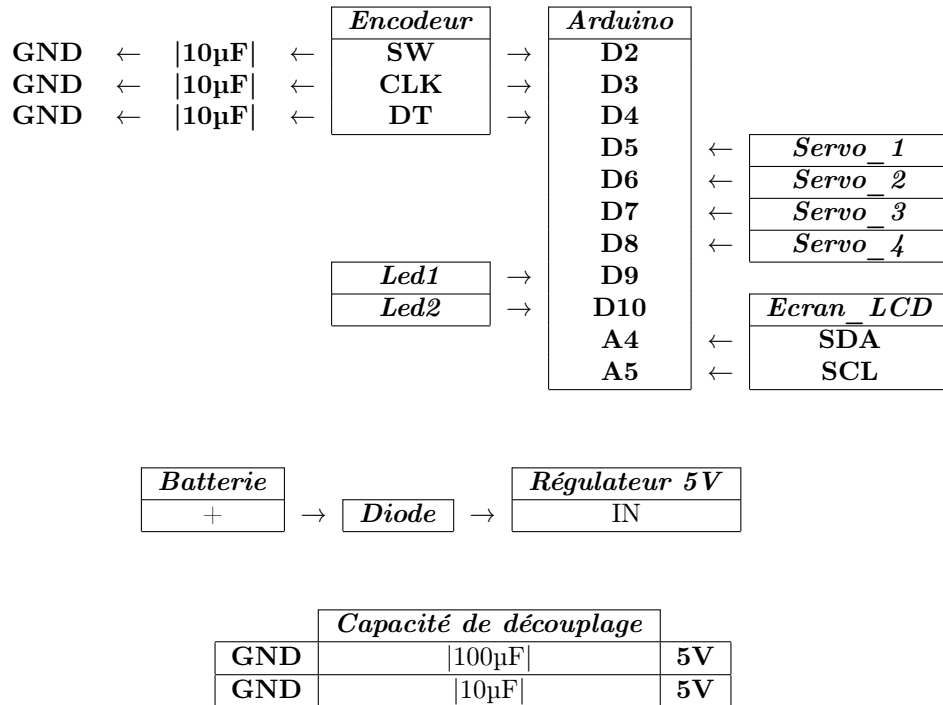
Choisir le bon composant et le placer ensuite sur le schéma en cliquant là où vous souhaitez le positionner.


• **Aide :**

-  Connecteurs : connecteurs "Conn_01x??_Female" ou "Conn_01x??_Male"
?? dépend du nombre de broche voulu.
-  servomoteurs : "Conn_01x03_Female" ou "Conn_01x03_Male" par exemple.
-  Bornier à vis pour brancher la batterie : "Screw_Terminal_01x02"
-  Régulateur 5V "R-785.0-1.0"
-  Resistance : "R" (on leur assignera la valeur 1kOhm)
-  Capacité de découplage : "C" (on leur assignera respectivement la valeur 10uF ou 100uF)
-  Diode schottky : "D"

2ème étape : Relier les composants entre eux


Ci-dessous j'ai récapitulé les différentens connexions entre les composants :




Pour relier deux pins, on cliquera sur :  (W).

Deux options s'offrent à vous ici : Relier bout à bout chaque fil ou utiliser des "labels".

Pour utiliser les labels, il vous suffit de créer une portion de fil relié à votre pin puis en cliquant sur :

 (L) et à nouveau sur votre fil. Vous pouvez ainsi lui donner un nom et réitérer là où vous souhaitez qu'il se connecte. (c'est la méthode que j'ai utilisé dans le schéma donné par la suite.)

Attention, cette méthode bien que pratique et propre pour le schéma n'est pas la plus astucieuse, en effet, on pourrait déjà avoir une vision globale du routage à cette étape si on n'utilisait pas de labels.

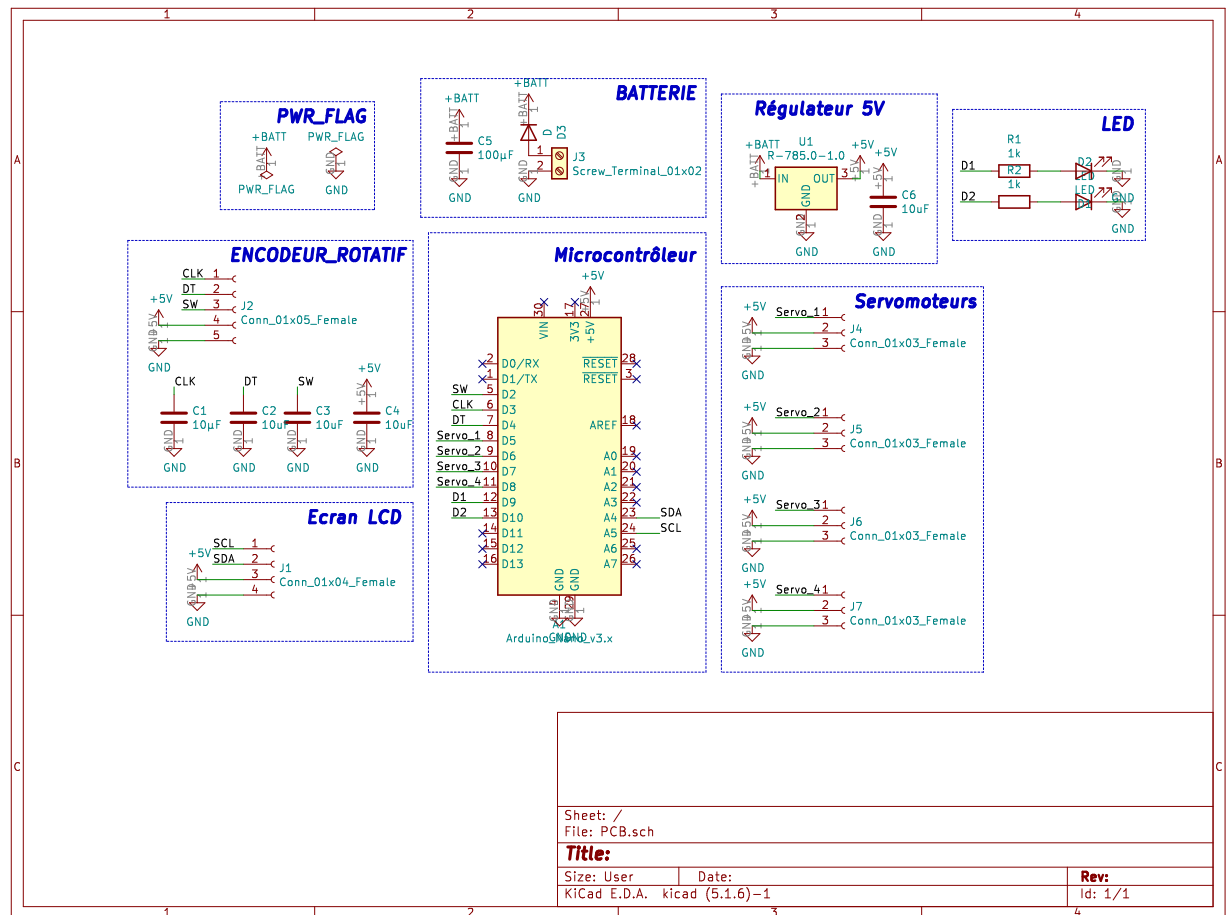
Pour l'alimentation cliquez sur :  (P)

☞ BATTERIE : symbole BATT (il faut un PWR_FLAG également)



☞ VCC : symbole 5V

☞ GND : symbole GND (il faut un PWR_FLAG également)


A la fin de cette étape, vous devez obtenir quelque chose de semblable :



3ème étape : Test des règles électriques

Cliquez sur le bouton :  ou [Inspecter > Testeur des Règles Electriques]. Si vous avez suivis les étapes précédentes vous devriez n'avoir qu'un seul type d'erreur à ce stade : les pins non connectés doivent être indiquées avec le bouton :  [Placer > Symbole de non connexion] (Q)

4ème étape : Assigner les empreintes des composants

Cliquez sur le bouton  "Assigner les empreintes des composants à la schématique". Ici sont données les références des composants. Vous pouvez vous référer à la fiche technique des composants pour trouver l'empreinte correspondante.

Symbole: Attribution Empreintes

1	A1 -	Arduino_Nano_v3.x : Module:Arduino_Nano
2	C1 -	10µF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
3	C2 -	10uF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
4	C3 -	10uF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
5	C4 -	10uF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
6	C5 -	100µF : Capacitor_SMD:CP_Elec_6.3x7.7
7	C6 -	10uF : Capacitor_SMD:C_1206_3216Metric_Pad1.42x1.75mm_HandSolder
8	D1 -	LED : LED_THT:LED_D5.0mm
9	D2 -	LED : LED_THT:LED_D5.0mm
10	D3 -	D : Diode_SMD:D_SMA_Handsoldering
11	J1 -	Conn_01x04_Female : Connector_PinHeader_2.54mm:PinHeader_1x04_P2.54mm_Vertical
12	J2 -	Conn_01x05_Female : Connector_PinHeader_2.54mm:PinHeader_1x05_P2.54mm_Vertical
13	J3 -	Screw_Terminal_01x02 : TerminalBlock:TerminalBlock_bornier-2_P5.08mm
14	J4 -	Conn_01x03_Female : Connector_PinHeader_2.54mm:PinHeader_1x03_P2.54mm_Vertical
15	J5 -	Conn_01x03_Female : Connector_PinHeader_2.54mm:PinHeader_1x03_P2.54mm_Vertical
16	J6 -	Conn_01x03_Female : Connector_PinHeader_2.54mm:PinHeader_1x03_P2.54mm_Vertical
17	J7 -	Conn_01x03_Female : Connector_PinHeader_2.54mm:PinHeader_1x03_P2.54mm_Vertical
18	R1 -	1k : Resistor_SMD:R_1206_3216Metric_Pad1.42x1.75mm_HandSolder
19	R2 -	1k : Resistor_SMD:R_1206_3216Metric_Pad1.42x1.75mm_HandSolder
20	U1 -	R-785.0-1.0 : Converter_DCDC:Converter_DCDC_RECOM_R-78E-0.5_THT

3.2 Routage du PCB

1ère étape : Placer les composants de manière optimale

Le premier PCB n'est jamais parfait et vous aurez le temps de vous entrainer pour parfaire la disposition. Gardez à l'idée qu'il faut éviter de placer deux pins qui doivent être reliés l'un de l'autre... On travaillera sur un PCB double face : Top et Bottom. Les composants traversants et le condensateur 100uF SMD devront se trouver sur la partie TOP de la carte, ceux en SMD restants sur la partie bottom. On facilite le routage ainsi dans l'optique où la carte est fabriquée avec la micrograveuse du fablab.

2ème étape : Tout relier (sauf le GND pour le moment)

(petite astuce pour un routage plus joli : une direction de fil par couche, ex : top vertical, bottom horizontal;)

On privilégiera une trace plus épaisse au niveau de la batterie et de l'alimentation des composants. (1.2mm) Ne pas oublier qu'on peut utiliser des vias pour connecter les deux couches du PCB ensemble. (1.6mm diamètre, trous 0.8mm) Pour changer rapidement de couche : "v".

3ème étape : Tracé du contour de la carte

Sur la couche "Edge.cut" avec l'outil : ligne.


4ème étape : Plan de masse (GND)

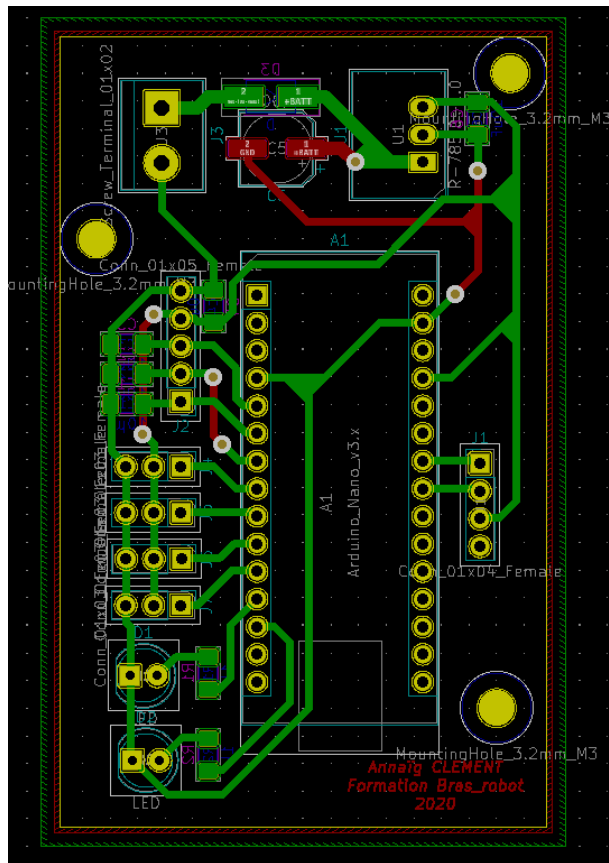
Sélectionnez l'outil "zone" et cliquez sur le premier point de votre plan de masse. Une fenêtre s'affiche, sélectionnez le top layer et le GND puis validez. Répétez cette action pour le plan de masse sur le bottom layer.

5ème étape : Ajouter les mounting holes (optionnel mais c'est toujours mieux)

Appuyez sur "O" ou allez dans [Placer > Empreinte] puis cherchez "MountingHole_3.2mm_M3". Placez-en 3 sur votre carte.

6ème étape : Valider la carte

Exécutez le DRC, c'est la petite coccinelle : . Si aucune erreur ne s'affiche, vous venez de terminer le PCB! :)

**7ème étape : Visualisation de la carte (optionnel)**

Affichage > 3D visualisateur (enjoy;))

