Partie EC2 :

Présentation générale du système supportant le projet :

Acteur majeur sur le marché, Le groupe ONET (fr.groupeonet.com) répond à des problématiques de décontamination et de démantèlement quelque soit leur complexité.

Son service « Ligne de Produit Investigations » souhaite posséder un ROV low cost et nous a proposé de développer un prototype.

Un ROV (Remotely Operated Vehicle) est un « véhicule téléguidé », le plus souvent un petit robot contrôlé à distance (généralement filoguidé). Un ROV permet une acquisition rapide et sécurisée d'informations globales ou précises, physicochimiques et visuelles (sous forme numérique notamment), assez rapidement, à distance de l'opérateur et parfois "en masse".

Certains ont une fonction de plateforme pouvant être équipée à la demande de préleveurs ou de divers capteurs.

Les objectifs sont d'obtenir et gérer des données d'entrée afin de définir un environnement TQC (Tel Que Construit).

Les enjeux du projet sont :

- Capacité à avoir un équipement Low cost avec des briques modulaires
- Capacité à être piloté à distance en liaison filaire
- Capacité à collecter des données d'entrée
- Capacité à traiter les données d'entrée (photogrammétrie ou vidéogrammétrie ou scan)

Expressions des besoins :

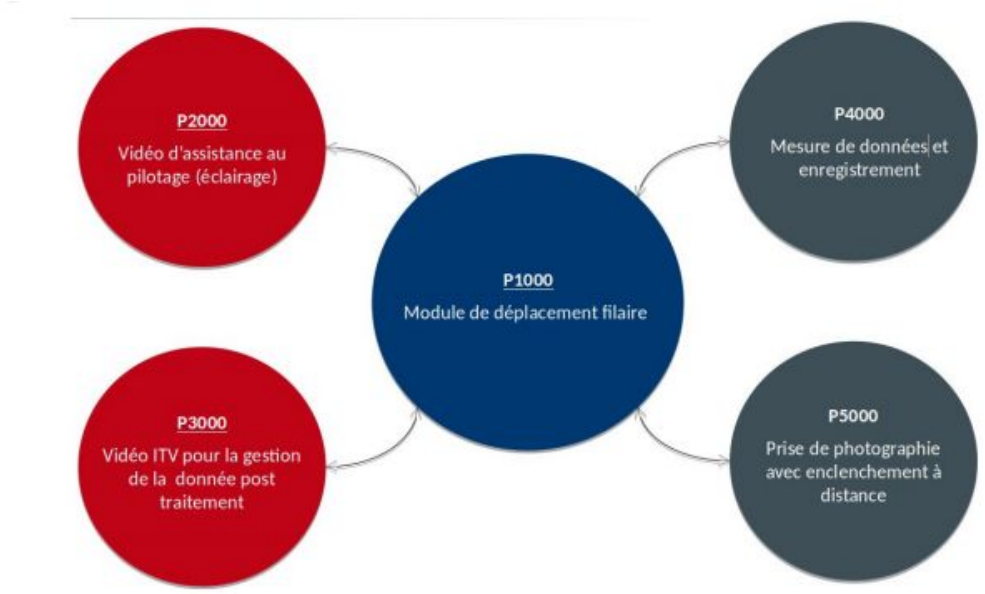


Schéma de principe :

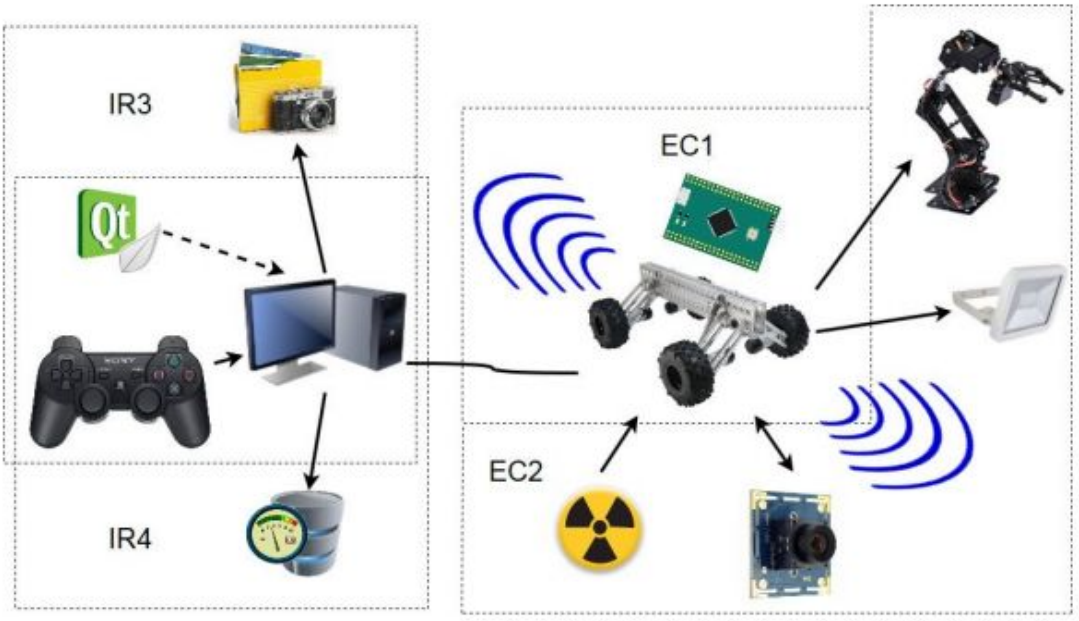
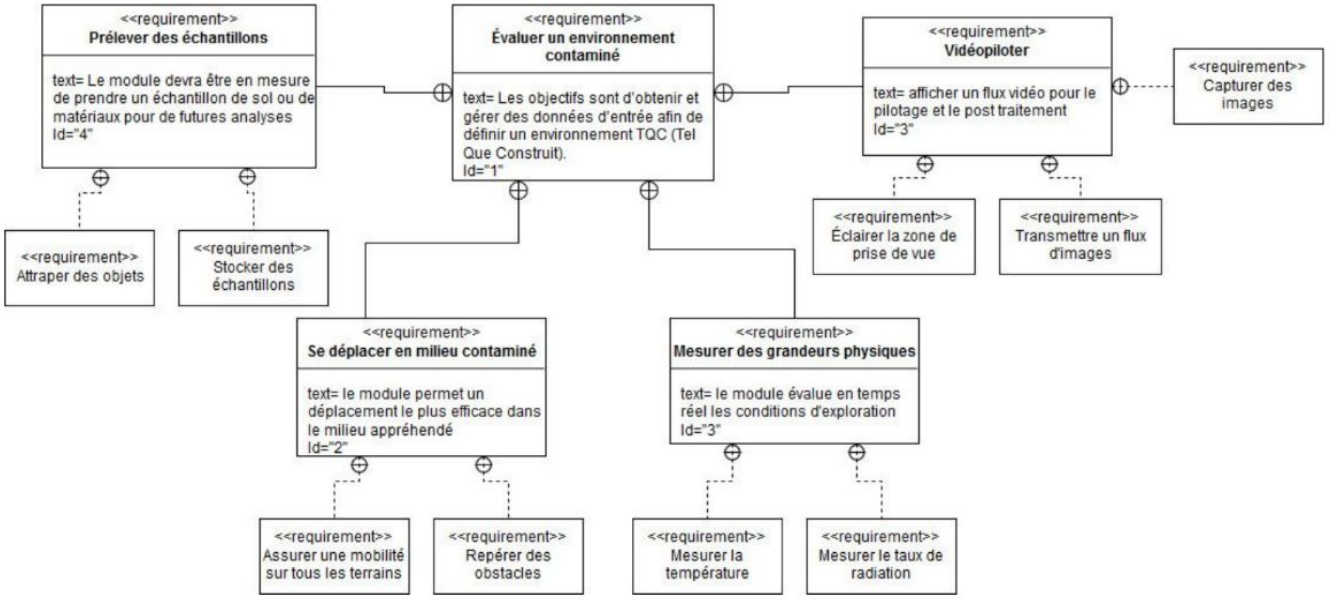


Diagramme des exigences :



Ressources matérielles :

Désignation	*Caractéristiques techniques*	*Acquisition*	*Existant*
ROBOT	Châssis mobile modèle		X
CAPTEURS	Ensemble de capteurs à définir et à monter sur le robot suivant la version à développer		X
BRAS ARTICULE	Bras de robotique avec pince de préhension		X
CAMERA	Module Caméra USB 5M pixels, 2592x1944 15fps MJPEG, 1280x720 30fps MJPEG		X
MANETTE	manette(s) avec boutons permettant le pilotage de la caméra et du bras		X
SE	Système embarqué à définir (Atmega, Arduino ou équivalent)		X
PC	Ordinateur PC		X

Diagramme de définition des blocs :

Cliquez ici.

Diagramme de blocs internes :

Cliquez ici.

Affectation des broches du microcontrôleur ESP32 :

	ATTRIBUTION BROCHES EN PWM (MLI)

Broches	Configuration	Gestion logicielle	Liaison matérielle
0 (RX)			----------
1 (TX)			----------
GPIO 4	Sortie	Scrutation	PWM moteur avant Droit
GPIO 0	Sortie	Scrutation	PWM moteur avant Gauche
GPIO 2	Sortie	Scrutation	PWM moteur arrière Droit
GPIO 15	Sortie	Scrutation	PWM moteur arrière Gauche

BROCHES	ATTRIBUTION BROCHES NUMÉRIQUES

GPIO 5	Entrée (pull-up)	Scrutation	Capteur de température
GPIO 16	Entrée	Interruption	Réception Capteur Geiger

I2C	Sortie	Scrutation	Sens de Rotation roue avant droite
I2C	Sortie	Scrutation	Sens de rotation roue avant Gauche
I2C	Sortie	Scrutation	Sens de rotation roue arrière Droite
I2C	Sortie	Scrutation	Sens de rotation roue arrière Gauche
GPIO 21	Sortie	Scrutation	Broche LED avant
GPIO 27	Sortie	Scrutation	Pince (Servo 0)
GPIO 26	Sortie	Scrutation	Pince (Servo 1)
GPIO 25	Sortie	Scrutation	Pince (Servo 2)
GPIO 33	Sortie	Scrutation	Pince (Servo 3)
GPIO 32	Sortie	Scrutation	Pince (Servo 4)
GPIO 19	Sortie	Scrutation	Pince (Servo 5)
GPIO 23	Sortie	Scrutation	Caméra (Servo X)
GPIO 18	Sortie	Scrutation	Caméra (Servo Y)
GPIO 13	Sortie	Scrutation	TRIG (HC-SR 04)
GPIO 12	Entrée	Scrutation	ECHO (HC-SR 04)
GPIO 14	Sortie	Scrutation	Servo (Télémètre)

Mesurer les radiations :

Afin de mesurer un taux de radiation, nous pouvons nous orienter vers la solution suivante : utiliser un compteur geiger.
Le compteur Geiger, ou compteur Geiger-Müller (ou compteur G-M), sert à mesurer un grand nombre de rayonnement ionisant (particules alpha, bêta ou gamma et rayons X, mais pas les neutrons).
Généralement, les tubes G-M ont la forme d'un cylindre métallique servant de cathode et sont dotés d'une anode centrale. Certains tubes ont une fenêtre en mica pour la détection des particules alpha et du rayonnement bêta trop faibles pour traverser la paroi métallique du tube.
Un gaz présent dans le tube se ionise lorsqu'une particule alpha ou bêta rentre dans le tube. des paires d'ions chargés positivement et des électrons libres sont créés. Le champ électrique dans le tube accélère les ions positifs vers la cathode et les électrons négatifs vers l'anode. Si les électrons libres gagnent assez d'élan, ils ioniseront d'autres molécules de gaz sur leur trajectoire, créant des électrons libres supplémentaires qui à leur tour ioniseront encore plus de molécules de gaz, et ainsi de suite. Le résultat de cette avalanche est une **impulsion électrique** (que l'on peut détecter).

Des cartes à compteur geiger destinées aux mesures à l'aide de microcontrôleur existent sur le marché.

Référence de la carte de mesure choisie : RadiationD-1v1 cajoe

- Alimentation de la carte : 5V
- Courant d'alimentation : 12mA – 30mA
- Tube Geiger de référence J305
- Spécifications techniques du tube :

- Fabricant: North Optic
- Détection de radiation : Beta, Gamma [β, γ]
- Tension recommandée: 350V
- Sensibilité γ (60Co): 65cps/(μR/s)
- Sensibilité γ (équivalent Sievert): 108cpm / (μSv/h)
- Max cpm: 30000
- cps/mR/h: 18
- cpm/μSv/h: 123.147092360319
- On en déduit le facteur de conversion: 0.00812037037037

Schéma structurel de la carte : Cliquez ici.

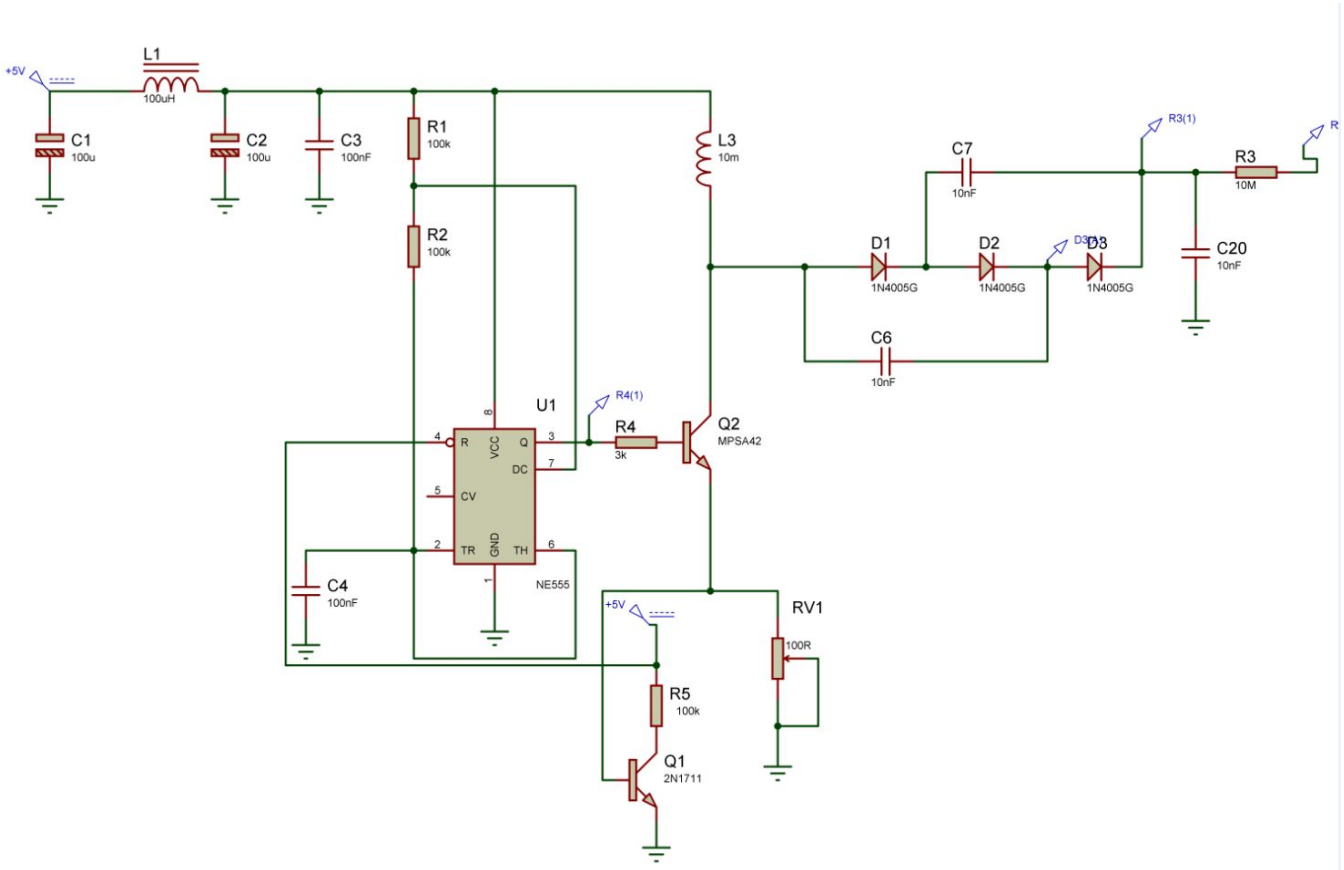
Nous pouvons distinguer deux parties :
Une partie d'adaptation haute tension, afin d'alimenter le tube.
Une partie permettant de générer une impulsion basse détectable sur la broche "Vin"

Schéma de câblage du module :

La broche 16 peut être programmée en tant que interruption (comme toutes celles de l'ESP32). C'est pourquoi nous l'avons choisie.

Simulation de la structure "élévateur de tension" :

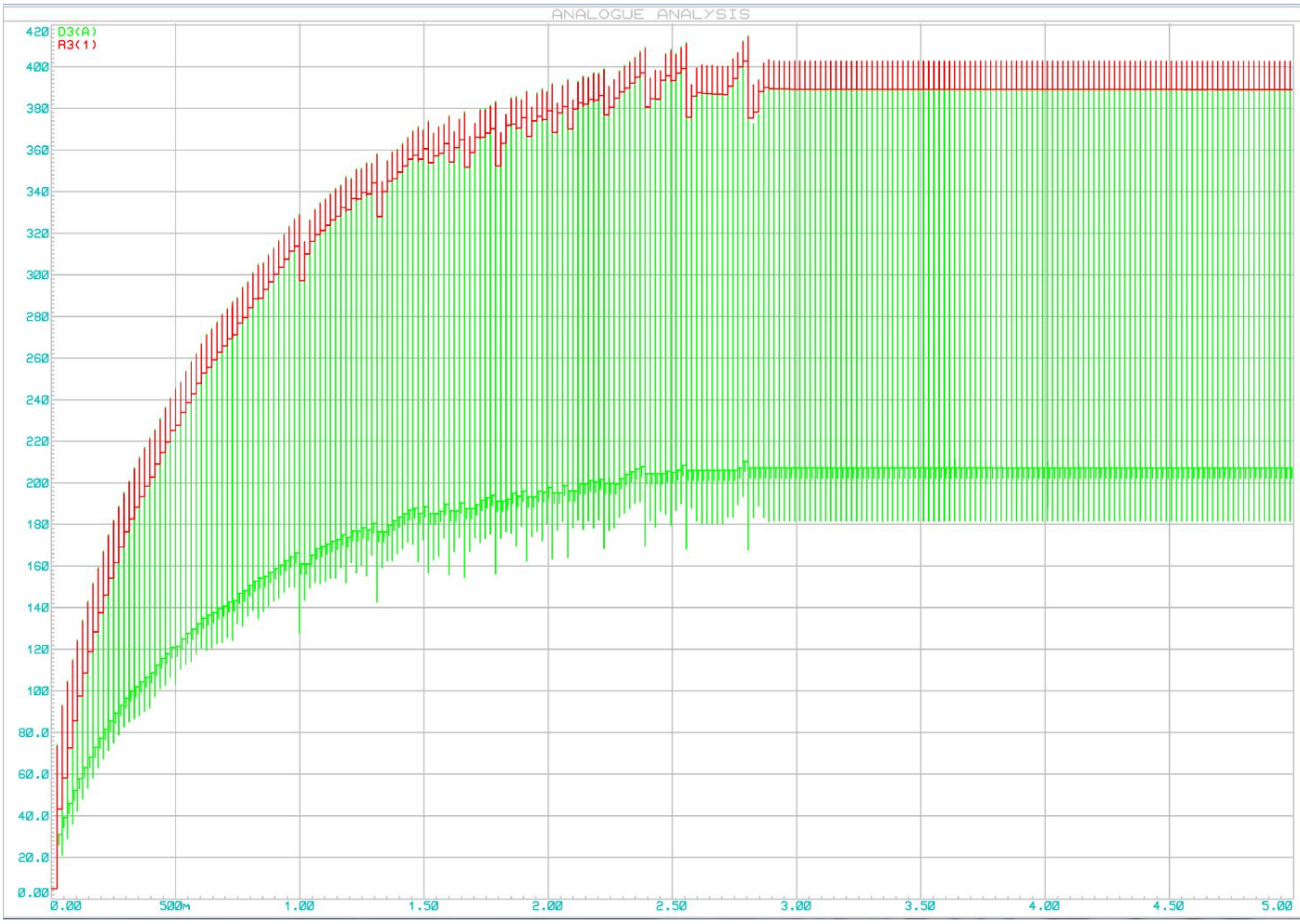
Nous pouvons trouver intéressant de simuler la structure d'élévateur de tension afin de fournir la tension nécessaire au compteur Geiger (les valeurs de R1, R2, et C4 ont été augmenté pour augmenter la rapidité de simulation)



Q1 : Transistor “2N1711”; transistor de jonction bipolaire NPN moyennes tensions (environ 50V)
Q2 : Transistor “MPSA42”,transistor de jonction bipolaire NPN hautes tensions (environ 200/300V)

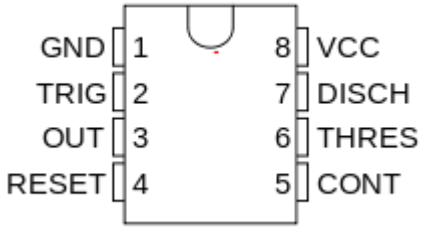
En plaçant la sonde au niveau de la résistance R33 (utilisée comme charge à la place du compteur Geiger), nous remarquons que la tensions est correctement élevée; elle varie de

5V à 400V, qui est la tension recommandée par le constructeur du tube.



Le circuit NE555 permet de fournir des signaux carrés de fréquence F.
Calcul de la fréquence (formule trouvée dans la documentation technique) : Foutput = 1.44 / (R1 + 2R2)*C4. R1 et R2 ont une valeur de 33kOmhs et C4 vaut 1nF (dans le schéma réel). La fréquence vaut 14.5kHz environ.
De ce fait, les condensateurs C6, C7, C20 vont pouvoir se charger ainsi que se décharger afin augmenter la tension de sortie du 555. Il est important de noter que la structure se charge par le biais de l'inductance L3.

NE555 :

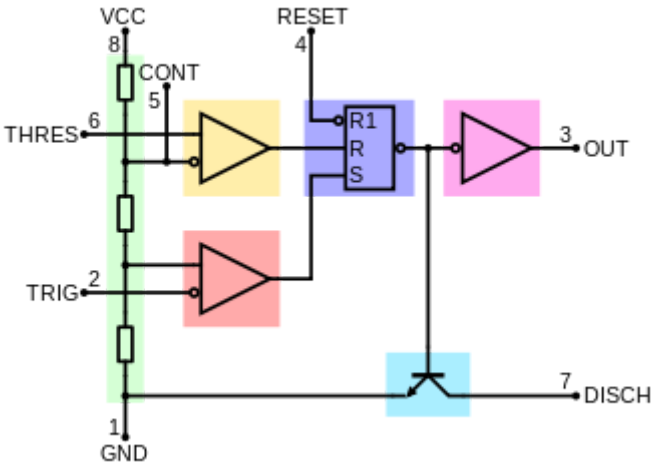


Le NE555 contient 23 transistors, 2 diodes et 16 résistances qui forment 4 éléments :

- deux amplificateurs opérationnels de type comparateur ;
- une porte logique de type inverseur ;
- et une bascule SET-RESET.

Le NE555 peut fonctionner selon trois modes : monostable, astable ou bistable.

Schéma bloc :



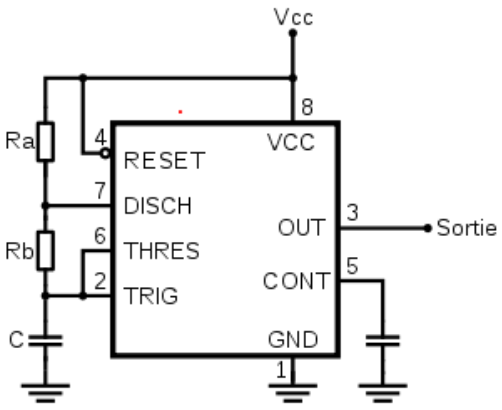
On peut voir à partir du schéma bloc les différents composants du NE555, soit :

- 2 comparateurs (jaune et rouge) ;
- 3 résistances configurées en diviseur de tension. Les deux tensions respectivement de 1/3 et 2/3 de Vcc servent de références aux comparateurs (vert) ;
- 1 bascule SET-RESET contrôlée par les comparateurs (violet) ;
- 1 inverseur (rose) ;
- 1 transistor pour décharger le condensateur de temporisation (bleu ciel).

L'opération du 555 suit la logique de fonctionnement du schéma bloc présenté et peut prendre 4 états différents.

- Le signal RESET est à un niveau bas : La bascule est remise à zéro, le transistor de décharge s'active et la sortie reste impérativement à un niveau bas. Aucune autre opération n'est possible.
- Le signal TRIG est inférieur à 1/3 de VCC : la bascule est activée (SET) et la sortie est à un niveau haut, le transistor de décharge est désactivé.
- Le signal THRES est supérieur à 2/3 de VCC : la bascule est remise à zéro (RESET) et la sortie est à un niveau bas, le transistor de décharge s'active.
- Les signaux THRES et TRIG sont respectivement inférieurs à 2/3 de VCC et supérieurs à 1/3 de VCC : la bascule conserve son état précédent de même que pour la sortie et le transistor de décharge.

Dans notre cas, il est utilisé en mode “astable”, tel que le démontre le schéma de principe ci-dessous indiqué dans la documentation technique :



La fréquence d’oscillation est déterminée à l’aide des résistances ainsi que du condensateur en entrée.

Algorithme :

```
////////////////////////////////////
//                                //
//           ALGORITHME           //
//                                //
//           CompteurGeiger       //
//                                //
//           BRAHMIA Aurélien     //
//                                //
////////////////////////////////////



/************* setup ****************/

1 - INITIALISER LES BROCHES (Vin)
2 - INITIALISER PORT SÉRIE
3 - DECLARER LES VARIABLES (CPM, FacteurConverison, TauxRadiation,
BrocheGeiger..)
4 - INITIALISER LA FONCTION INTERRUPTION (DetecterImpulsion)

/************* loop ****************/
Lancer un décompte;

Si TempsEcoule >= 60000 Alors
      TauxRadiation = CPM * FacteurConversion;
      Envoyer les données via le port série;
```

```
Fsi



/************* DetecterImpulsion ****************/
CPM = CPM + 1;
```

Code : Cliquez ici.

Afin de compter le nombre d'impulsion, nous avons utilisé une interruption.
Une interruption, c'est l'arrêt temporaire de l'exécution normale du programme afin d'exécuter un gestionnaire d'interruption (handler). Une interruption peut être lancée sur un changement d'état d'une entrée (ici, changement d'état vers le bas). Il existe deux types d'interruption, à savoir matérielle et logicielle. Dans notre cas, il s'agit d'un matérielle, car elle est associée aux entrées/sorties.

Relevés de mesures :

Voici l'impulsion générée par le compteur geiger relevée à l'oscilloscope. Le transistor "Q3" en sortie permet de générer un passage de l'état haut à l'état bas.
Lien ici.
De plus, nous pouvons augmenter le nombre d'impulsions générées à l'aide d'un bouton poussoir entre le connecteur J2 et la masse.

Paramétrage de l'oscilloscope :
Trigger sur front descendant (valeur à 3.48V environ)
Base de temps : 500us/div
Résolution tension : 1V/div

Augmenter le nombre d'impulsion :
Il est possible d'augmenter le nombre d'impulsion générée (et donc le nombre de coup par minute relevé par le microcontrôleur) en utilisant un bouton poussoir.
En reliant le réseau résistif , se situant entre le connecteur J4 au connecteur J2, au bouton poussoir, nous pouvons générer une impulsion lors d'un appui sur ce dernier.

Mesurer la température :

Afin de mesurer la température de l'environnement dans lequel va se déplacer notre robot, plusieurs solutions concernant le choix d'un capteur s'offrent à nous ; DHT11, DHT22, LM35…

Voici un tableau comparatif de différents capteurs existants :

	DHT11	DHT22	LM35	BME280
Type de mesure	Température, humidité	Température, humidité	Température	Température, humidité, pression
Moyen de mesure / technologie de mesure	One-wire	One-wire	Analogique	I2C / SPI
Plage de mesure	0 à 50ºC	-40 à 80ºC	-55 à 150ºC	-40 à 85ºC
Précision	+/- 2ºC (de 0 à 50ºC)	+/- 0.5ºC (de -40 à 80ºC)	+/-0.5ºC (à 25ºC)	+/-0.5ºC (à 25ºC)
Disponibilité	En possession	En possession	En possession	A acquérir
Moyen de programmation	Bibliothèque existante	Bibliothèque existante	Lecture d'une tension puis mise à l'échelle	Bibliothèque existante
Prix	2*	3*	2*	1*

*Niveau 1 : Prix très élevé (par rapport aux autres)
*Niveau 2 : Prix moyen
*Niveau 3 : Prix faible

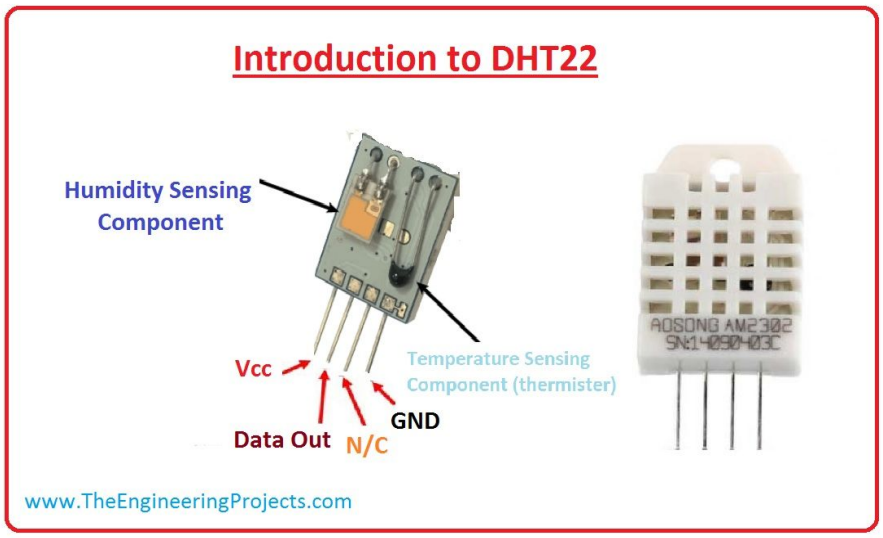
D'après ce tableau, le choix va se faire entre le DHT22 et le LM35, de part leur précision et leur plage de mesure.

Petit inconvénient pour le LM35 : le capteur nécessite une alimentation négative pour mesurer des températures en dessous de 0°C. La sortie est proportionnelle à la température avec un rapport environ égale à 1/10, par conséquent -10°C équivaut à -0,1 volt. De fait, avec une simple alimentation 5v, il n'est possible de mesurer que des températures positives.

De plus, le prix dans notre projet n'est pas un critère très important.
C'est pourquoi nous nous orienterons vers le DHT22.

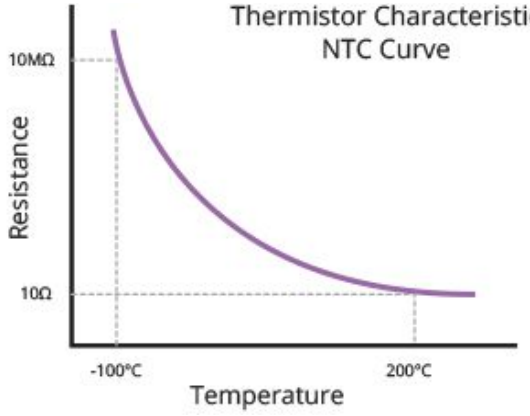
<u>Fonctionnement du capteur :</u>

Afin de mesurer une température, le DHT22 est en réalité composé d'une thermistance (comme montré ci-dessous) :



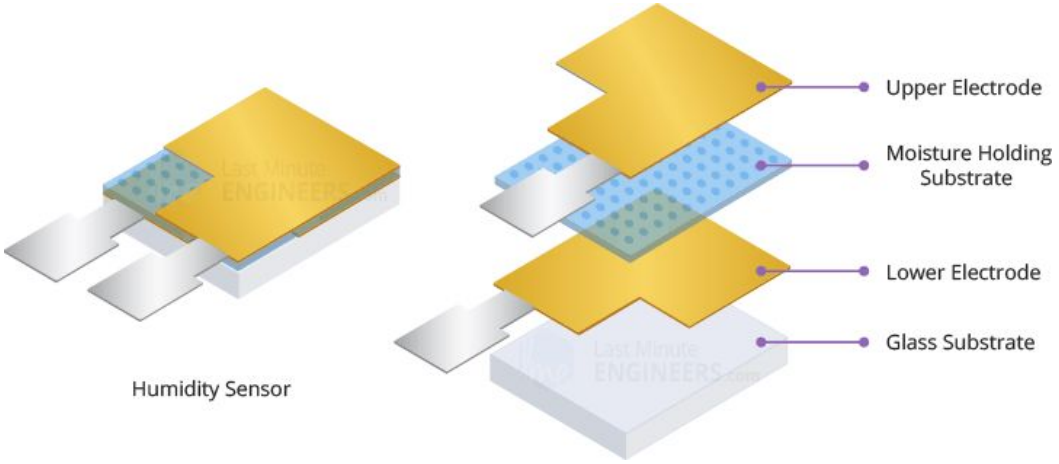
Comme son nom l'indique, une thermistance correspond à une résistance dont sa résistivité va varier en fonction de la température.
Plus précisément, on qualifie se composant comme "CTN" (Coefficient de Température Négatif), ce qui signifie que la valeur de la résistance diminue si la température augmente.


NTC Thermistor



Concernant l'humidité, le composant de mesure de cette dernière est composé de deux électrodes avec un substrat de rétention d'humidité (généralement un polymère plastique ou sel conducteur). Les ions sont libérés par le substrat lorsque la vapeur d'eau est absorbée par cette dernière, ce qui augmente la conductivité entre les électrodes. Le changement de résistance entre les deux électrodes est proportionnel à l'humidité relative. Une humidité relative plus élevée diminue la résistance entre les électrodes, tandis qu'une humidité

relative plus faible augmente la résistance entre les électrodes. Schéma de principe :



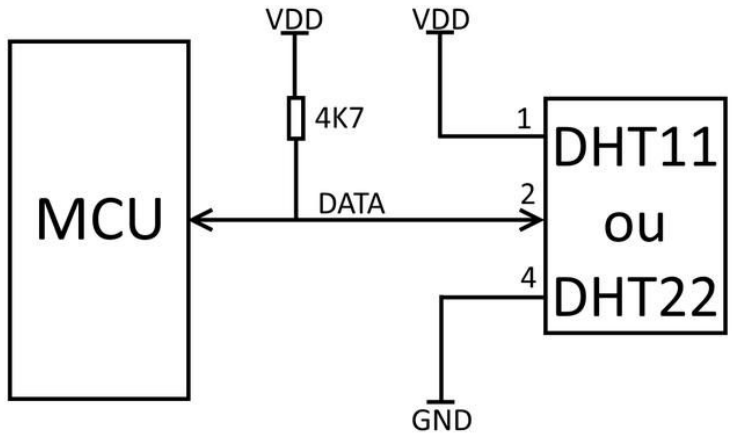
Caractéristiques du capteur :

- Tension de fonctionnement : 3 - 5V
- Courant max. de fonctionnement : 2.5mA
- Fréquence de mesure : 1Hz
- Mode de communication : one-wire

Câblage du composant:

Le dht22 est composé de 4 broches : VCC, DATA, NC, GND.
NC signifie “non-connected”; elle ne sera dans ce cas pas connectée et donc non-utilisée.

Principe de câblage :



Une résistance de pull-up est nécessaire afin d’assurer un niveau récessif (1) du bus de donnée.

La broche de l'esp-wroom 32 choisie est la “GPIO 15”

Explication du protocole de communication :

Comme précisé précédemment, le capteur DHT22 a la particularité de communiquer avec le microcontrôleur via une unique broche d'entrée / sortie.

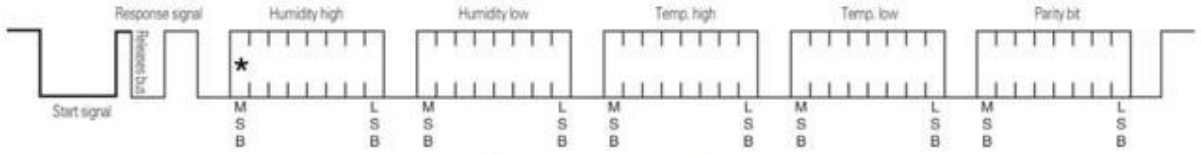
Bien que "One Wire" soit mentionné sur le document constructeur du capteur, il ne s'agit pas d'un véritable bus de communication 1-Wire. Il s'agit simplement d'un protocole de communication propriétaire, utilisant un seul fil et nécessitant des timings très précis.

La communication avec un capteur DHT22 se fait en **3 étapes :**

- Tout d'abord, le microcontrôleur maître (l'ESP32 dans notre cas) réveille le capteur en plaçant la ligne de données à LOW pendant au moins 800µs (au moins 18ms pour le DHT11). Durant ce laps de temps, le capteur va se réveiller et préparer une mesure de température et d'humidité. Une fois le temps écoulé, le maître va libérer la ligne de données et passer en écoute.

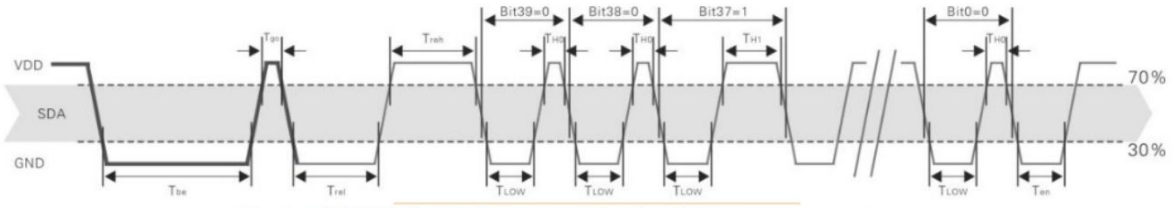
- Une fois la ligne de données libérée, le capteur répond au maître (pour montrer qu'il est bien réveillé) en maintenant la ligne de données à LOW pendant 80µs puis à HIGH pendant 80µs.

- Le capteur va ensuite transmettre une série de 40 bits (5 octets). Les deux premiers octets contiennent la mesure de l'humidité. Les deux octets suivants contiennent la mesure de la température et le cinquième octet contient une somme de contrôle qui permet de vérifier que les données lues sont correctes.



Pic5: AM2302 Single-bus communication protocol

Plus précisément, c'est la durée à un état haut qui va déterminer la valeur d'un bit. Voici un exemple avec toute les durées correspondantes :



Pic 6: AM2302 Single-bus communication timing

Table 6: Single bus signal characteristics

Symbol	Parameter	min	typ	max	Unit
T_{be}	Host the start signal down time	0.8	1	20	mS
T_{go}	Bus master has released time	20	30	200	μs
T_{rel}	Response to low time	75	80	85	μs
T_{reh}	In response to high time	75	80	85	μs
T_{LOW}	Signal "0", "1" low time	48	50	55	μs
T_{H0}	Signal "0" high time	22	26	30	μs
T_{H1}	Signal "1" high time	68	70	75	μs
T_{en}	Sensor to release the bus time	45	50	55	μs

Chaque bit est envoyé sous forme d'une impulsion positive.

N.B. La broche de communication du capteur DHT22 est de type "collecteur ouvert". La sortie du capteur ne génère pas de tension. Elle ne fait que commuter (via un transistor) la tension au niveau de la résistance de tirage sur la ligne de données. Dans ce contexte, HIGH est la tension de la résistance de tirage et LOW la tension à la masse (0 volt).

Algorithme :

```
////////////////////////////////////
//                                //
//           ALGORITHME           //
//                                //
//  Capteur température et humidité  //
//                                //
//           BRAHMIA Aurélien     //
//                                //
////////////////////////////////////



/************* setup ****************/
1 - INCLURE LA BIBLIOTHEQUE "DHT.h"
```

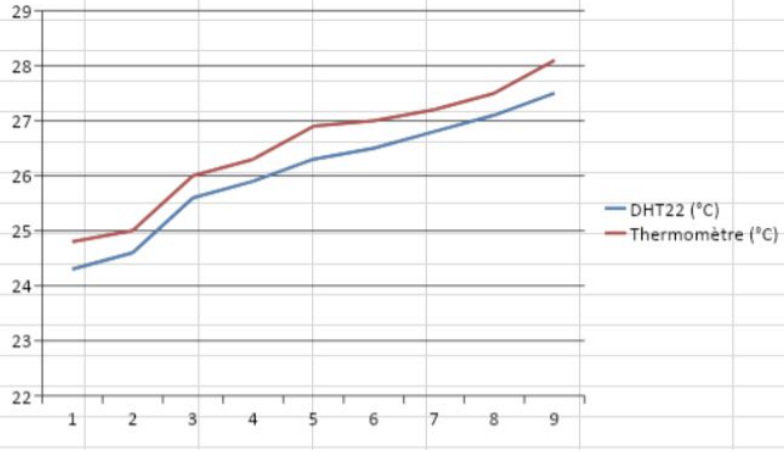
```
2 - INITIALISER LES BROCHES (Vout du DHT22)
3 - INSTANCIER L'OBJET "DHT22"
4 - DÉMARRER LE CAPTEUR
5 - INITIALISER PORT SÉRIE
6 - DECLARER LES VARIABLES (temperature, humidite)

/************* loop ***************/

Lancer une mesure de la température;
Lancer une mesure de l'humidité;
temperature = MesurerTemperature();
humidite = MesurerHumidite();
Envoi des données sur le port série;
```

Code avec les différentes fonctions et variables :

```
#include "DHT22.h"

DHT dht22(DHT22_PIN, DHTTYPE);

void initialiserDHT22() {
    dht22.begin();
}

float mesurerTemperature() {
    float temp = 0.00;
    temp = dht22.readTemperature();
    return temp;
}

float mesurerHumidite() {
    float hum = 0.00;
    hum = dht22.readHumidity();
    return hum;
}
```

Tests réels:

	A	B
1	DHT22 (°C)	Thermomètre (°C)
2	24,3	24,8
3	24,6	25
4	25,6	26
5	25,9	26,3
6	26,3	26,9
7	26,5	27
8	26,8	27,2
9	27,1	27,5
10	27,5	28,1
11		
12		
13		
14		
15		



Voici le comparatif des valeurs mesurées par un thermomètre et par le DHT22.
Nous pouvons relever un écart variant entre 0.5 et 0.6 °C (comme indiqué dans la documentation technique).
Nous avons utilisé un briquet avec d'élever la température. La variation de distance entre la source de chaleur et les instruments de mesures a permis de faire varier la température mesurée.

Mesure trame : Utilisation d'un analyseur logique "Saleae" ainsi que du logiciel d'analyseur de trame "Logic". Cliquez Ici.

Créer et envoyer la trame contenant les données des capteurs :

L'une des principales tâches de ce projet est l'envoi des données concernant l'environnement et la télémétrie au PC, cela de manière filaire en série.

Nous avons alors réalisé le modèle de la trame qui sera envoyée, en accord avec les deux étudiants de l'option Informatique & Réseaux.

Nous avons réalisé deux trames différentes; une contenant les données concernant l'environnement (température, humidité, taux de radiation), et une contenant les données de télémétrie.

La raison principale de la création de ces deux trames au lieu d'une seule est liée à la contrainte de temps entre les envois.
Il paraît plus judicieux d'envoyer les données de télémétries de manière plus fréquente que celles de l'environnement (ces dernières étant moins importantes).

Modèle des trames :

$ENVI;TEMPERATURE;HUMIDITE;RADIATION\r\n

ENVI: Trame d'état des capteurs d'environnement
TEMPERATURE : Température(dixième de °C)
RADIATION : radiation(µS/h)
Période : 5 secondes

$TEL;DISTANCE\r\n

TEL: Trame de télémétrie
: distance (cm)
Période : 250 millisecondes

Algorithme de l'envoi des trames :

```
////////////////////////////////
//                            //
//          ALGORITHME        //
//                            //
//       ENVOI DES TRAMES     //
//                            //
//        TOYOS BRAHMIA       //
//                            //
////////////////////////////////



/************ setup ***************/

1 - INITIALISER LES BROCHES DES COMPOSANTS
2 - INITIALISER PORT SERIE (9600 BAUDS)
3 - INCLURE LES LIBRARIES
4 - INITIALISER L'INTERRUPTION TIMER (transmettreTelemetrie() et
transmettreEnvironnement())

/************ loop ***************/

1 - lireTauxRadiation();
2 - lireHumidite();
3 - lireTemperature();
4 - lireTelemetrie();
5 - creationTrameEnvironnement();
6 - creationTrameTelemetrie();
```

```
/************* transmettreEnvironnement ***********/
envoyerTrameEnvironnement();

/************* transmettreTelemetrie ***********/
envoyerTrameTelemetrie();
```

Une interruption de type “timer” a été utilisée afin de rythmer l’envoi des trames selon les besoins de l’utilisateur. Cette interruption est de type logicielle

Programmation :

Fonction initialiserTimer() (timer pour les données environnementales) :

```
void initialiserTimer(){

// Timer pour l’envoi des données concernant l’environnement

      TimerEnvironnement = timerBegin(0, CADENCEMENT, true);

      timerAttachInterrupt(TimerEnvironnement, &transmettreEnvironnement,
true);

      timerAlarmWrite(TimerEnvironnement, TEMPO_ENVIRONNEMENT, true);

      timerAlarmEnable(TimerEnvironnement);

}
```

Fonction fabriquerTrameEnvironnement() :

```
void fabriquerTrameEnvironnement() {

      trameEnvironnement = ENTETE_TRAME TYPE_CAP SERPARATEUR +
String(temperature) + SERPARATEUR + String(humidite) + SERPARATEUR +
String(Taux_Radiation) + FIN_TRAME ;

}
```

L’envoi de la trame se fait par le port série; il suffit d’utiliser la fonction Serial.print();

Il faut ensuite la placer dans la fonction timer dédiée :

```
void IRAM_ATTR transmettreEnvironnement() {

      envoyerTrameEnvironnement();

}
```

Éclairer la zone de prise de vue :

Afin d’obtenir une vision optimale avec la caméra, il semble nécessaire d’éclairer la prise de vue, dans l’objectif de visualiser correctement l’environnement.

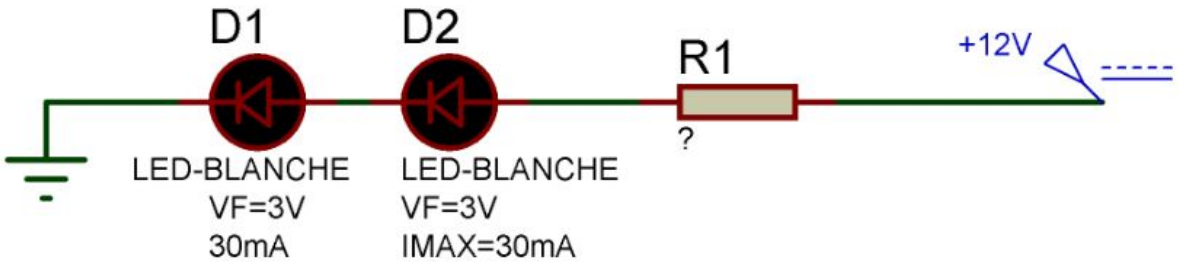
Pour ce faire, nous pouvons utiliser des leds blanches avec d’éclairer des endroits obscurs.

Dimensionnement de la led et caractéristiques :

- Type de Led : Blanche
- Vf = 3V
- If = 30mA
- Intensité lumineuse :

Il faut ensuite choisir une résistance afin de fixer le courant direct de la LED:

Schéma de principe :



En appliquant la loi des mailles nous obtenons :
Vcc = R1*If + 2*Vf

Vcc - 2*Vf = R1*If
R1 = (Vcc - 2*Vf) / If

A.N : Nous prenons If = 30mA et Vf = 3V
Nous trouvons : R1 = 200Ohms (valeur non normalisée)

Nous décidons de choisir une valeur normalisée en se fiant la série E12: nous choisisson R1 = 220 Ohms afin de ne pas fournir un courant au dessus du courant maximal fixé par le constructeur.

En utilisant cette valeur, nous trouvons If = 27mA, ce qui est acceptable.
Puissance dissipée par la résistance : Pres = UR1 * If
UR1 vaut 6V; en appliquant cette valeur, nous obtenons Pres = 16.36 mW

Nous pouvons alors choisir la résistance suivante :
Valeur normalisée : 220 Ohms
Puissance dissipée : ¼ W minimum
Technologie : à couches (carbones ou métalliques)
Type : Traversant
Tolérance : de +/-0.5% à +/-1%

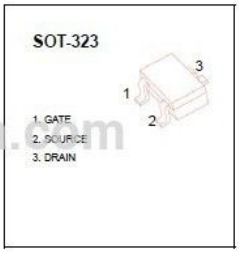
https://www.cdiscount.com/bricolage/electricite/resistance-sourcingmap-r-220-ohm-1-4w-5-resist/f-1661416-sou0608641811766.html#mpos=0|mp

De plus, nous avons décidé de piloter le circuit à l'aide d'un transistor.
Voici le transistor choisi : 2N7002

Technical Specification

SOT-323 Plastic-Encapsulate MOSFETS

2N7002W MOSFET (N-Channel)



FEATURES

- High density cell design for low $R_{DS(ON)}$
- Voltage controlled small signal switch
- Rugged and reliable
- High saturation current capability

Marking: K72

MAXIMUM RATINGS (T_a=25℃ unless otherwise noted)

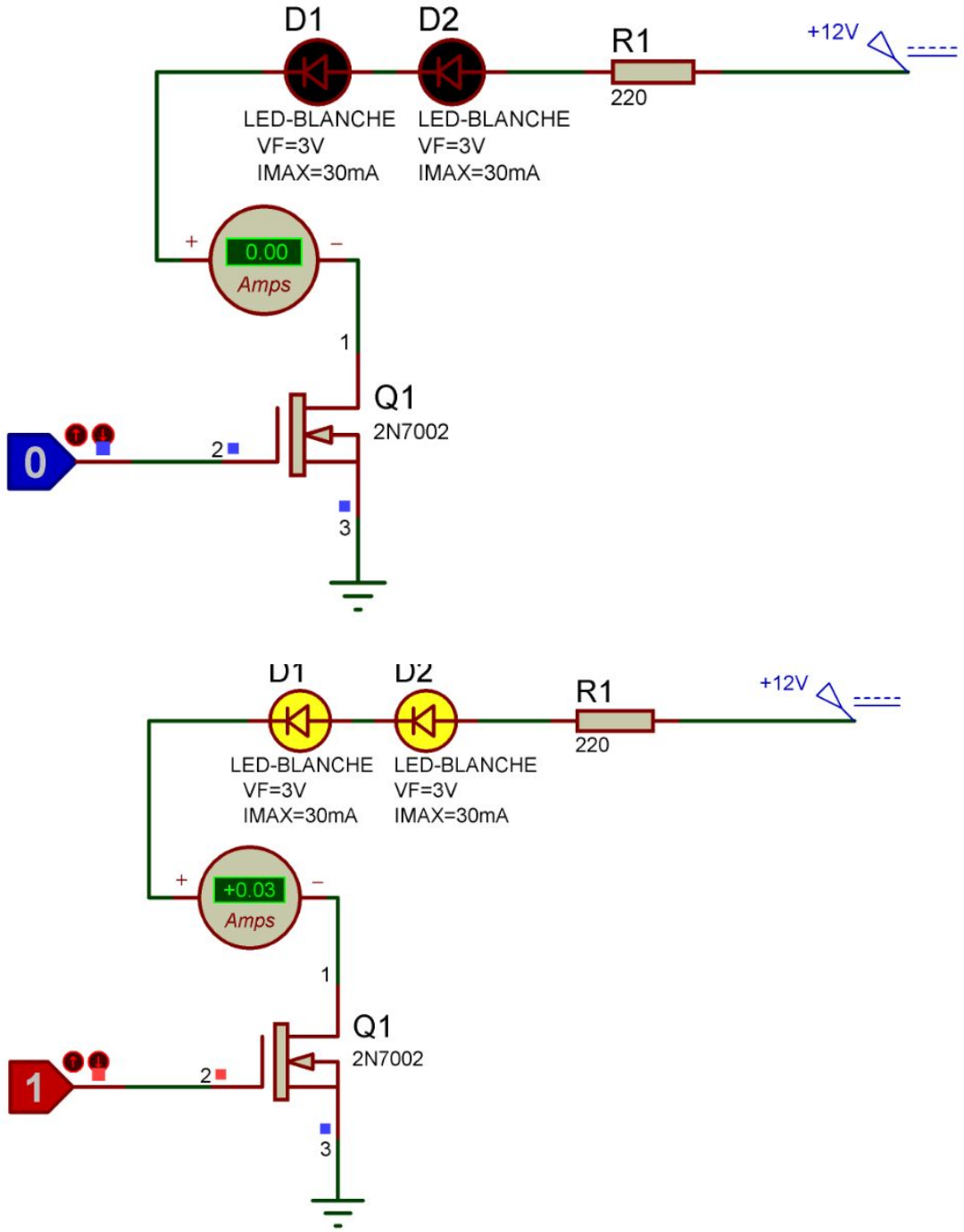
Parameter	Symbol	Value	Unit
Drain-Source Voltage	V_{DS}	60	V
Gate-Source Voltage	V_{GS}	20	V
Continuous Drain Current	I_D	0.115	A
Power Dissipation	P_D	0.200	W
Thermal Resistance from Junction to Ambient	$R_{\theta JA}$	625	℃/W
Junction Temperature	T_J	150	℃
Storage Temperature	T_{stg}	-50 ~+150	

C'est un transistor MOSFET canal N à enrichissement.

ON CHARACTERISTICS (Note 8)						
Gate Threshold Voltage	$V_{GS(th)}$	1.0	—	2.5	V	$V_{DS} = V_{GS}$, $I_D = 250\mu A$

Une tension Vgs supérieure ou égale à 2.5V suffit rendre le transistor passant.

Simulation du montage :



Lorsqu'une tension est appliquée entre la grille et la source du transistor, la LED s'éclaire. De plus, le courant la traversant est égale à 30mA (sensiblement inférieur dans la réalité en raison de la valeur normalisée de la résistance).

Type de caméra à disposition :

$LEDS;{ON|OFF}\r\n

LEDS: Trame de commande de l'éclairage
{ON|OFF} : État des LEDs (**ON** = Allumer les LEDs, **OFF** = Eteindre les LEDs)

Algorithme :

```
////////////////////////////////
//                            //
//          ALGORITHME        //
//                            //
//        PILOTAGE LEDs       //
//                            //
//          BRAHMIA           //
//                            //
////////////////////////////////



/************ setup ***************/

1 - INITIALISER LES BROCHES DES LEDS
2 - INITIALISER PORT SÉRIE (9600 BAUDS)
3 - DÉCLARATION DES VARIABLES


/************ loop ***************/

Si recevoirDonnees() == true ALORS
lireTrameRecue();
decouperTrameLEDs();
Fsi

piloterLEDs();
```

Piloter le positionnement de la caméra :

L'objectif de cette partie et d'offrir des images en temps réel au PC de supervision. De plus, l'utilisateur doit être en mesure de piloter le champs de vision à l'aide d'une manette.

Type de caméra à disposition :

La caméra que nous utiliserons est une caméra "USB", 1M pixels,1280x720 30fps MJPEG. La liaison série USB permet directement de transmettre les images capturées par cette dernière directement vers le PC.

Le modèle utilisé est le suivant : ELP-USB100W05MT-RL36

Caractéristiques:
*Résolution 720 P 1280x720 P
* Module de caméra d'interface USB pour stéréo Monochrome
* Format de compression YUY et MJPEG en option
* Capteur CMOS 720 P pour une image de haute qualité et une faible consommation d'énergie. Référence : OV9712
* Interface USB 2.0 haute vitesse pour interface caméra PC haute résolution
* Technologie de pixels élevée pour une image nette et une reproduction précise des couleurs
* Faible performance lumineuse-idéal pour toutes les conditions d'éclairage
* Fréquence d'images élevée-fournit 30 fps en résolution 1280X720
* Objectif de haute qualité, image couleur vraie et non déformée
* Faible consommation d'énergie-idéal pour les équipements portables
* UVC pour une utilisation sous Linux, Windows XP, WIN CE, MAC, SP2 ou plus
* Caméra avec UVC
* Tension : 5V DC
* Consommation max : 150mA

Pilotage de la caméra :

Afin de piloter son positionnement, nous utiliserons des servomoteurs.
Voici un module de positionnement envisageable :

Le support est composé de deux servomoteurs:
- Un servomoteur pour le déplacement de la caméra sur l'axe horizontal "X". On emploiera le terme "Pan" (faire un panoramique)
- Un servomoteur pour le déplacement de la caméra sur l'axe vertical "Y". On emploiera le terme "Tilt" (qui se traduit par incliner en anglais)

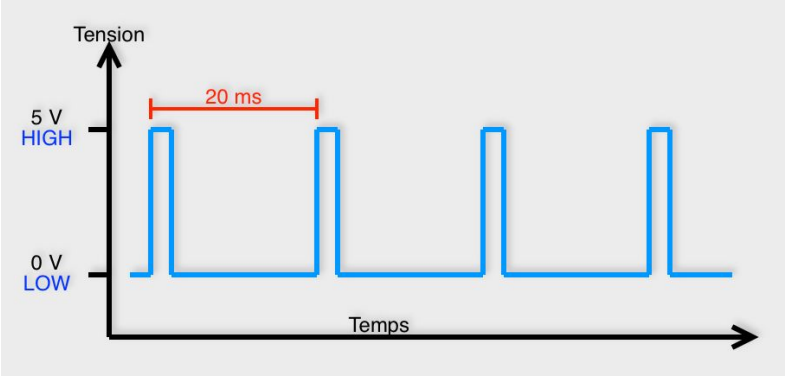
Les servomoteurs utilisés sont de type "SG90"

<u>Fonctionnement d'un servomoteur :</u>
Un servomoteur est un moteur capable de maintenir sa position par rapport à une tension de consigne donnée. une opposition à un effort statique et dont la position est vérifiée en continu et corrigée en fonction de la mesure. C'est donc un système asservi.

Pilotage et gestion de position

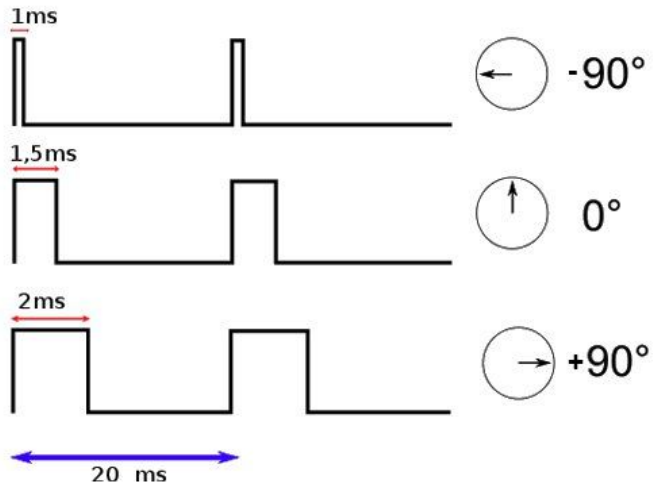
Un servo-moteur est un actionneur asservi. Pour qu'il maintienne sa position, il faut lui appliquer en permanence un signal de consigne. Cependant, l'énergie nécessaire à son fonctionnement est fournie par une source d'alimentation différente de celle du signal de commande.

Période de rafraîchissement : 20ms



Positionnement du servomoteur :

Il faut générer un impulsion de 1 ms à 2 ms pour réaliser son excursion maximale.
L'impulsion de 1,5 ms sera généralement appelé la position "neutre".



Un servomoteur s'alimente de 4,5V à 6V (généralement 5V) par les fils rouge (+) et noir (-). Le troisième fils restant est utilisé pour le signal de consigne. Dans notre cas, il est orange.

<u>Trame reçue :</u>
Voici le modèle de la trame envoyée lors du pilotage réalisé à l'aide de la manette:

$CAM;{G|D|0};{H|B|0}\r\n

CAM : Trame de pilotage de la caméra
{G|D|0} : Axe X de déplacement (**G** = Gauche, **D** = Droite, **0** = Pas de déplacement)
{H|B|0} : Axe Y de déplacement (**H** = Haut, **B** = Bas, **0** = Pas de déplacement)

Algorithme :

```
////////////////////////////////
//                            //
//          ALGORITHME        //
//                            //
//       PILOTAGE CAMÉRA      //
//                            //
//          BRAHMIA           //
//                            //
////////////////////////////////



/************* setup ***************/

1 - INITIALISER LES BROCHES DES SERVOMOTEURS
2 - INITIALISER PORT SÉRIE (9600 BAUDS)
3 - INCLURE LES LIBRARIES (SERVO.H)


/************* loop ***************/

Si recevoirDonnees == true ALORS
lireTrameRecue();
decouperTrameCamera();
Fsi

piloterCamera(); //en fonction des données qui ont été extraites
```

Pour piloter le positionnement, nous pouvons choisir d'incrémenter l'angle de position de la caméra d'un pas pré-défini (l'angle peut varier de 1 à 15°).

Code :

Fonction decouperTrameCamera() :

```
void decouperTrameCamera() {

    if (Trame.startsWith(ENTETE_CAMERA)) {
        //si la trame correspond à la trame CAMERA, on repère les deux délimiteurs
        //ainsi que la fin de trame
        delimiter_1Cam = Trame.indexOf(DELIMITER_CAMERA);
        delimiter_2Cam = Trame.indexOf(DELIMITER_CAMERA, delimiter_1Cam + 1);
        endCamera = Trame.indexOf("\r\n");

        //on extrait les données nécessaires au déplacement de la caméra
        deplacement_Camera_Pan = Trame.substring(delimiter_1Cam + 1, delimiter_2Cam);
        deplacement_Camera_Tilt = Trame.substring(delimiter_2Cam + 1, endCamera);


    }
}
```

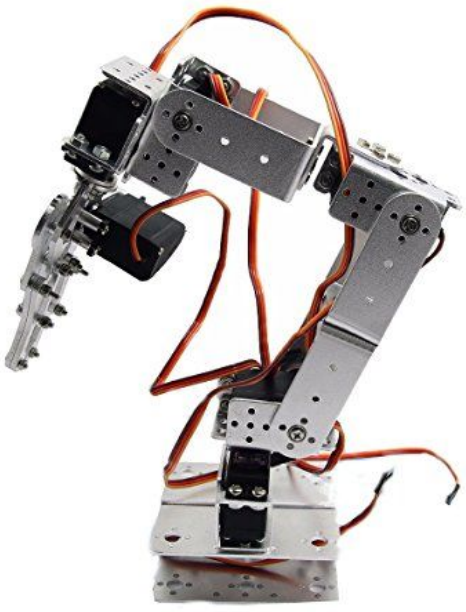
Fonction piloterCamera() :

```
void piloterCamera() {

    if (deplacement_Camera_Pan == "G") {
        positionPan = positionPan - PAS;
    }

    else if (deplacement_Camera_Pan == "D") {
        positionPan = positionPan + PAS;
    }

    else if (deplacement_Camera_Tilt == "H") {
        positionTilt = positionTilt - PAS;
    }

    else if (deplacement_Camera_Tilt == "B") {
        positionTilt = positionTilt + PAS;
    }
```

```
    if (positionPan >= ANGLE_MIN_CAMERA && positionPan <= ANGLE_MAX_CAMERA) {
        Servo_Pan.write(positionPan);
    }

    if (positionTilt >= ANGLE_MIN_CAMERA && positionTilt <= ANGLE_MAX_CAMERA)
{
        Servo_Tilt.write(positionTilt);
    }

}
```

Commander le bras articulé et la pince de préhension :

Le bras articulé que nous utiliserons est le suivant :



Nous avons en notre possession un bras robotique 6 axes (il possède alors 6 degrés de liberté). Ce bras est constitué de 6 servomoteurs, chacun correspondant à une fonction d'un bras. Chaque servomoteur peut être associé à une des fonctions génériques de mouvement d'un bras.

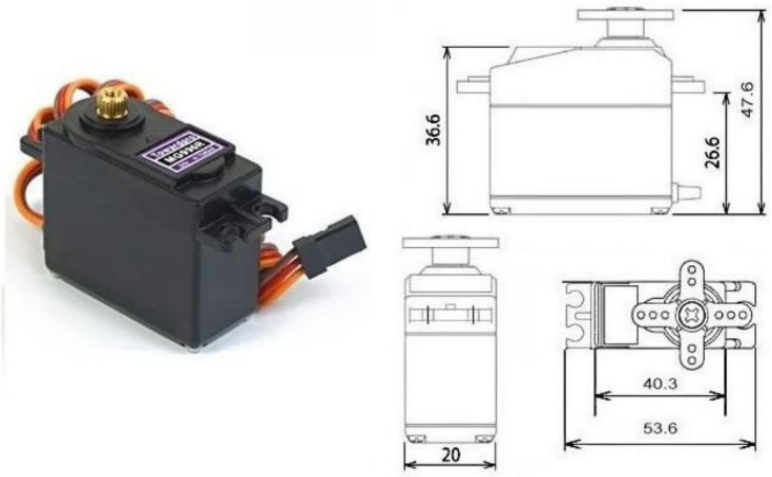
Voici un tableau récapitulant chaque fonction de chaque servomoteur :

	Partie	Fonction

Servo 0	Pince	Saisir (object. etc..)
Servo 1	Main	Pivoter main
Servo 2	Poignet	Déplier (partie 2)
Servo 3	Coude	Déplier (partie 1)
Servo 4	Epaule (lever)	Élévation du bras
Servo 5	Epaule (tourner)	Rotation du bras

Servomoteurs utilisés :

6 servomoteurs de référence “MG996R”

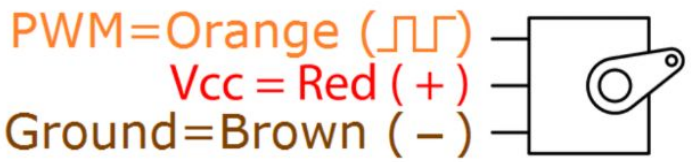


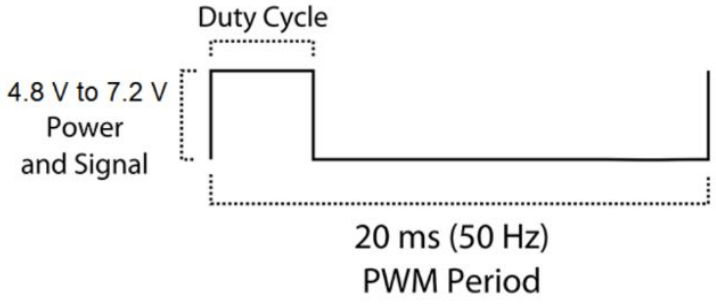
Caractéristiques :
• Poids: 55 g

• Dimension: 40.7 x 19.7 x 42.9 mm approx.

• Operating voltage: 4.8 V à 7.2 V

• Running Current 500 mA – 900mA





Mouvements réalisables :

Trames reçues :
3 trames sont envoyées pour le pilotage du bras.

Voici le modèle de ces dernières :

$BRAS;{GAUCHE|DROITE|MONTE|DESCEND|AVANCE|RECULE}\r\n

> **BRAS** : Trame de commande du bras
> **{GAUCHE|DROITE|MONTE|DESCEND|AVANCE|RECULE|0}** : Type de demande de pilotage du bras

$ORDRE;{INIT|BAC}\r\n

> **ORDRE** : Trame d'ordre au bras
> **{INIT|BAC}** : Type d'ordre (**INIT** = Retour à l'état initial de la pince, **BAC** = Mettre l'objet dans le bac)

$PINCE;{O|0|F}\r\n

> **PINCE** : Trame de commande de la pince
> **{OPINCE|0|FPINCE}** : État de la pince (**OPINCE** = Ouvrir la pince, **0** = Pas de mouvement, **FPINCE**= Fermer la pince,)

```
////////////////////////////////
//                            //
//          ALGORITHME        //
//                            //
//        PILOTAGE BRAS       //
```

```
//                                //
//           BRAHMIA              //
//                                //
////////////////////////////////////



/************* setup ***************/

1 - INITIALISER LES BROCHES DES SERVOMOTEURS
2 - INITIALISER PORT SÉRIE (9600 BAUDS)
3 - INCLURE LES LIBRAIRIES (SERVO.H)
4 - INITIALISER BRAS
5 - DÉCLARATION DES VARIABLES


/************* loop ***************/

Si recevoirDonnees == true ALORS
lireTrameRecue();
decouperTrameDirectionBras();
decouperTrameOrdre();
decouperTramePince();
Fsi

piloterDirectionBras();
appliquerOrdreBras();
piloterPince();
```

Codage :

Le choix des angles à appliquer pour la position initiale, la position pour aller dans le bac à objets, la position de soutien des servomoteurs pour monter/descendre et avancer/reculer à été réalisé à partir de tests sur le bras.

Fonction pour initialiser le bras :

```
void initialiserBras() {
    bras.tournerEpaule.attach(PIN_TOURNER_EPAULE); //5
    bras.leverEpaule.attach(PIN_LEVER_EPAULE); //4
    bras.coude.attach(PIN_COUDE); //3
    bras.poignet.attach(PIN_POIGNET); //2
    bras.main.attach(PIN_MAIN); //1
```

```
    bras.pince.attach(PIN_PINCE); //0

    initialiserPosition();
}
```

InitialiserPosition() :

```
void initialiserPosition() {
    position.elevation = ANGLE_INITIAL_LEVER_EPAULE;
    position.deplier_1 = ANGLE_INITIAL_COUDE;
    position.deplier_2 = ANGLE_INITIAL_POIGNET;
    position.rotation = ANGLE_INITIAL_TOURNER_EPAULE;
    position.pivoter = ANGLE_INITIAL_MAIN;
    position.saisir = ANGLE_INITIAL_PINCE;

    bras.leverEpaule.write(position.elevation);
    delay(DELAY_INITIALISATION);
    bras.coude.write(position.deplier_1);
    delay(DELAY_INITIALISATION);
    bras.poignet.write(position.deplier_2);
    delay(DELAY_INITIALISATION);
    bras.tournerEpaule.write(position.rotation);

}
```

allerDansBac() :

```
void allerDansBac() {
    position.elevation = LEVER_EPAULE_BAC;
    position.deplier_1 = COUDE_BAC;
    position.deplier_2 = POIGNET_BAC;
    position.rotation = TOURNER_EPAULE_BAC;
    position.saisir = PINCE_BAC;

    bras.tournerEpaule.write(position.rotation);
    delay(DELAY_ALLER_BAC);

    bras.leverEpaule.write(position.elevation);
       delay(DELAY_ALLER_BAC);
```

```
    bras.coude.write(position.deplier_1);
       delay(DELAY_ALLER_BAC);

    bras.poignet.write(position.deplier_2); //ouverture de la pince
    delay(DELAY_ALLER_BAC);

    bras.pince.write(position.saisir);

}
```

Fonctions pour découper les trames :

```
void decouperTrameBras() {

////////////Trame de direction du bras ////////////////////////
    if (Trame.startsWith(ENTETE_DIRECTION)) {
    //test avec d'identifier le type de trame
    //obtenir la position du delimiteur aninsi que de la fin de trame
        delimiteur_Trame = Trame.indexOf(DELIMITER);
        fin_Trame = Trame.indexOf(FIN_TRAME);

        //extraire la donnée située entre le délimiteur et la fin de trame
        Direction_Bras = Trame.substring(delimiteur_Trame + 1, fin_Trame);
    }


///////////// Trame d'ordre du bras ////////////////////////////
    if (Trame.startsWith(ENTETE_ORDRE)) {
    //test avec d'identifier le type de trame
    //obtenir la position du delimiteur aninsi que de la fin de trame
    delimiteur_Trame = Trame.indexOf(DELIMITER);
    fin_Trame = Trame.indexOf(FIN_TRAME);

    //extraire la donnée située entre le délimiteur et la fin de trame
    Ordre_Bras = Trame.substring(delimiteur_Trame + 1, fin_Trame);

    }


///////////// Trame de pilotage de la pince ////////////////////////////
    if (Trame.startsWith(ENTETE_PINCE)) {
```

```
        //test avec d'identifier le type de trame
        //obtenir la position du delimiteur aninsi que de la fin de trame
        delimiteur_Trame = Trame.indexOf(DELIMITER);
        fin_Trame = Trame.indexOf(FIN_TRAME);

        //extraire la donnée située entre le délimiteur et la fin de trame
        Ordre_Pince = Trame.substring(delimiteur_Trame + 1, fin_Trame);

    }

}
```

Fonctions pour appliquer les ordres reçus :

```
void piloterBras() {

    if(Direction_Bras == "GAUCHE") {
        allerGauche();
    }

    else if (Direction_Bras == "DROITE") {
        allerDroite();
    }

    else if (Direction_Bras == "MONTE") {
        Serial.print("M");
        Monter();
    }

    else if (Direction_Bras == "DESCEND") {
        Descendre();
    }

    else if (Direction_Bras == "AVANCE") {
        Avancer();
    }

    else if (Direction_Bras == "RECULE") {
```

```
        Reculer();
    }

    else if (Ordre_Bras == "INIT") {
        initialiserPosition();
    }

    else if (Ordre_Bras == "BAC") {
        allerDansBac();
    }

    else if (Ordre_Pince == "OPINCE") {
        ouvrirPince();
    }

    else if (Ordre_Pince == "FPINCE") {
        fermerPince();
    }
}
```

Les fonctions de mouvement sont uniquement composées d'un incrémentation (PAS prédéfini) du servomoteur en question, accompagné de délai avec d'éviter tout mouvement "brusque".

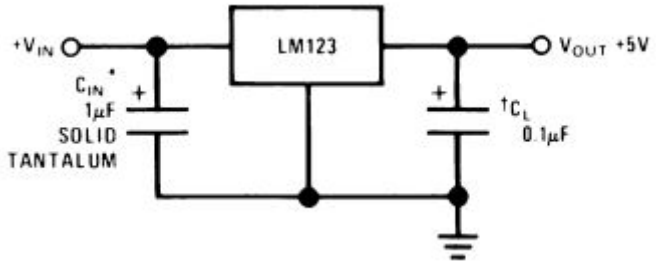
N.B : Après avoir mis en application le code, j'ai remarqué que les timer associés à l'envoi des trames des capteurs causaient des perturbations concernant le déplacement et le fonctionnement du bras et de la caméra. C'est pourquoi les fonctions de pilotage ont par la suite été placées dans un timer de 20 ms.

Réaliser la partie alimentation du robot :

L'alimentation de notre robot sera divisée en deux parties :
- Une partie alimentée en 12V/3A, afin d'alimenter les moteurs des roues du robot, ainsi que les deux LEDs à l'avant;
- Une partie alimentée en 12V/3A, afin d'alimenter le microcontrôleur, les capteurs, les servomoteurs.

Nous avons considéré que les moteurs du robot ne pourront pas fonctionner pas en même temps que les servomoteurs du bras articulé; ces deux parties étant les plus grandes sources de consommation du robot. C'est pourquoi 3A suffiront à alimenter notre robot.

Afin de convertir les 12V/3A en 5V/3A, nous décidons d'utiliser un régulateur de référence LM323.
Schéma issu de la documentation technique :

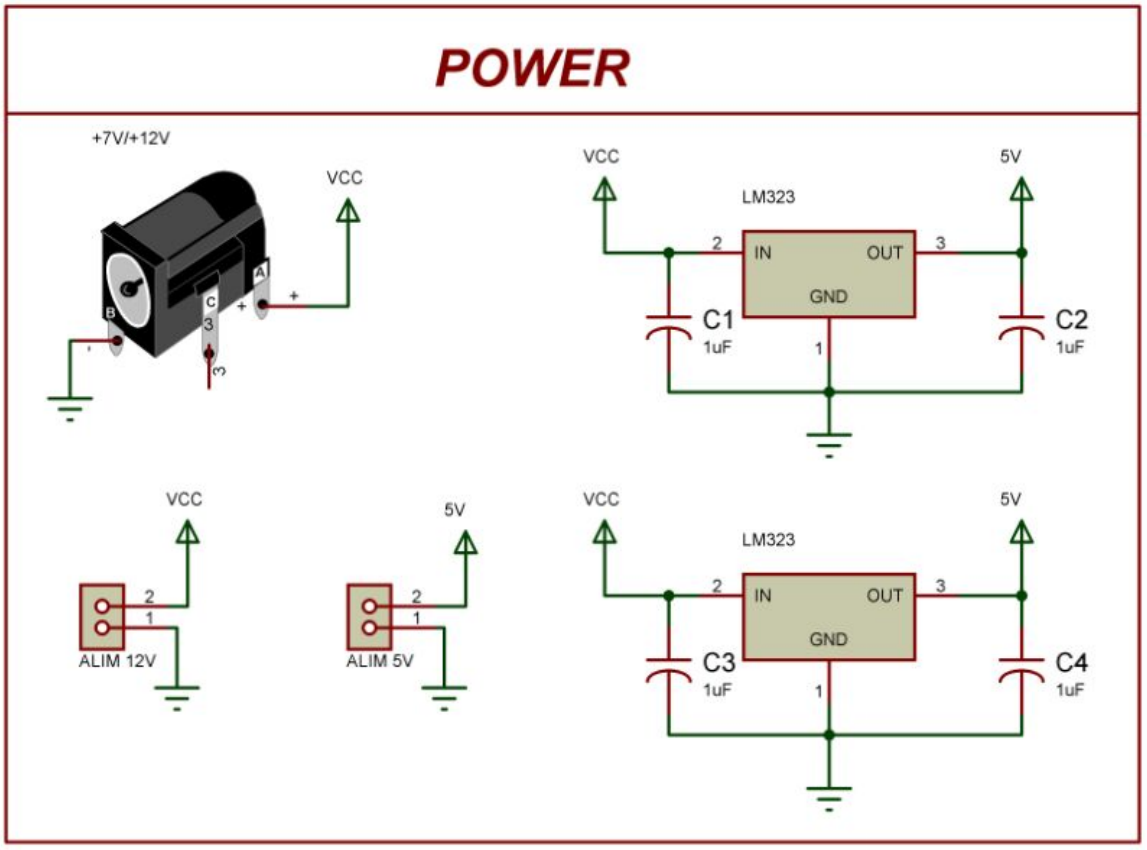


Deux condensateurs de découplage doivent être ajoutés autour du régulateur.

Réaliser le PCB:

Schéma structurel (partie composants) : Cliquer ici.

Schéma structurel (partie alimentation) :



Choix dissipateur (partie alimentation) :

Nous possédons un régulateur de tension de type LM323 (en boîtier TO220) qui fournit un courant de 3A. Sa tension d'entrée est de 12V.

D'après les renseignements techniques du constructeur :

- TJMAX = 125°C
- Rth(J-C) = 2 °C/W.

Déterminons le type de dissipateur nécessaire au montage pour maintenir son boîtier à 90°C (Tc) avec une température ambiante Ta de 25°C (montage à sec : nous prendrons Rth(C-R)=0.25° C/W).

Puissance dissipée par le régulateur : $Pd = (Ve - Vs).Is = (12 - 5).3 = 2.3W$

Sachant que :

$$Pd = \frac{T_j - T_A}{R_{thJ-A}} = \frac{T_j - T_A}{R_{thJ-C} + R_{thC-R} + R_{thR-A}}$$

On peut donc déterminer Rth(R-A): $$R_{thR-A} = \frac{Tc - Ta}{Pd} - R_{thC-R}$$

Application numérique :

$$R_{thR-A} = \frac{90-25}{2.3} - 0,25 = 27.6°C/W$$

Choix du modèle : FK 249 SA 220
https://fr.farnell.com/fischer-elektronik/fk-249-sa-220/heatsink/dp/1892323

Avec ce choix :

$$Tb = (R_{thR-A} + R_{thC-R}).Pd + Ta = (17 + 0,25).2.3 + 25 = 65.25°C < 90°Cmax$$

Température de jonction : $$Tj = R_{thJ-C}Pd + Tb = 4.67 + 65.25 = 69.91°C < 125°Cmax$$

Tests de validation: