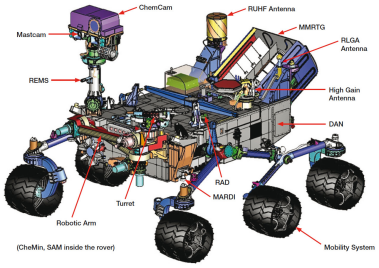
Programmer les principales fonctions et structure de base du "ROBOT" Arduino

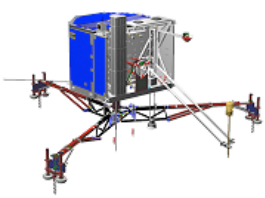
La charte morale de la robotique à été écrite dans une nouvelle américaine "Runaround" en 1942, elle a été modifiée en 1985 par Issac Asimov avec l'ajout la loi ZERO :

Loi Zéro : Un robot ne peut pas porter atteinte à l'humanité, ni, par son inaction, permettre que l'humanité soit exposée au danger.
Première Loi : Un robot ne peut porter atteinte à un être humain, ni, restant passif, permettre qu'un être humain soit exposé au danger, sauf en contradiction avec la Loi Zéro.
Deuxième Loi : Un robot doit obéir aux ordres que lui donne un être humain, sauf si de tels ordres entrent en conflit avec la Première Loi ou la Loi Zéro.
Troisième Loi : Un robot doit protéger son existence tant que cette protection n'entre pas en conflit avec la Première ou la Deuxième Loi ou la Loi Zéro.

Pendant de longues années de programmation vous ne serrez pas contrarié par la prise en compte de cette charte qui fait toujours partie de la science fiction et du cinéma .
La plupart des engins trop rapidement appelés "robots" sont en fait des belles machines automatisées qui adaptent leurs fonctions principales à un environnement variable définit par des limites physiques prévus au cahier des charges et par leur programmation. Quelques uns possèdent de modestes capacités d'apprentissage avec une autonomie énergétique souvent très limitée. Mêmes les sondes sur Mars et sur la comète Tchourioumov-Guérassimenko en sont de très beaux exemples . Dès que les situations initiales changent trop des domaines prévus, ces merveilleuses machines sont reprogrammées partiellement depuis la terre par d'authentiques humains pour que ces "robots" s'adaptent à ces nouveaux contextes.
Le robot Martien "Curiosity" de la Nasa a nécessité 5 Millions de lignes de code pour ses débuts sur MARS en 2012. La séquence de 7 minutes de l'entrée dans l'atmosphère martienne puis la descente jusqu'à l'atterrissage, en entière autonomie et d'une grande complexité technique, a nécessité la moitié de ces lignes (voir https://www.youtube.com/watch?v=Ki_Af_o9Q9s)
Malgré ces grands efforts de préparation, la probabilité d'échec était estimé à 1/3 ! Une très belle réussite technologique de robotique à voir et revoir sur internet.
(2,5 millions de lignes de codes représentent 104 000 pages en police 12 points soit 105 ramettes de recto verso de 500 pages)
Sur cette activité , on va plus modestement débuter avec 10 ou 12 lignes ...

Sommaire :



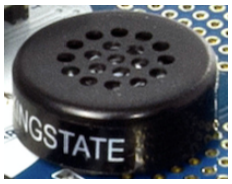


1 Mettre en oeuvres les commandes de base du mini Haut parleur du robot

Ce premier petit programme va seulement émettre des "bips" à fréquence fixe (chaque seconde dans un premier temps)
Les commentaires permettront de prendre en compte les premières contraintes d'écriture d'un programme Arduino.
Chaque ligne noire devra être saisie sans recopier les lignes bleues qui sont des commentaires utiles à la compréhension de l'ensemble de la structure mais aussi des infos sur les détails d'écriture.
Commencer par lancer le programme "ARDUINO"

programme à recopier

```
/* "/*" permet d'ouvrir une série de commentaires
sur plusieurs lignes "
Beep (titre)
Auteur (nom prénom classe)
Tester les différents bips possibles dans la
bibliothèque "Robot.beginSpeaker()"

 Les différents "beeps" du système sont :
 - BEEP_SIMPLE
 - BEEP_DOUBLE
 - BEEP_LONG
 */ ("*/" fin des commentaires de début de
programme)

// Déclaration de bibliothèques Arduino utiles dans
le programme
// "//" les doubles "slash" permettent de rédiger un
commentaire sur une ligne
#include <ArduinoRobot.h>
#include <Wire.h
#include <SPI.h>

void setup() { // début d'initialisation de programme
// "void  setup" se trouvera à chaque fois en tête de
programme
// initialisation du Robot
Robot.begin(); // presque chaque ligne est
terminée par un point virgule ";"
// initialisation des fonctions du haut parleur
  Robot.beginSpeaker();
}

void loop() {   // début de boucle loop
// "void  loop"  Cette BOUCLE se trouvera à
chaque fois en tête de programme. Comme  sont
nom l'indique  "quand c'est fini on recommence"
  Robot.beep(BEEP_SIMPLE);
// simple beep
  delay(1000);
// temporisation (attente) de 1000 millisecondes
soit 1 seconde
  Robot.beep(BEEP_SIMPLE);
  delay(1000);
  Robot.beep(BEEP_SIMPLE);
  delay(2000);                    VERIFIER !
}  // FIN de boucle loop
```



Déclarations des bibliothèques C et Arduino
#include <ArduinoRobot.h>
#include <Wire.h>
#include <SPI.h>
Déclarations des variables globales;

void setup() {

Initialisation des fonctions complémentaires, premiers calculs
Robot.begin()
Robot.beginSpeaker();

le robot est sous tension ou raccordé à la liaison USB

BEEP

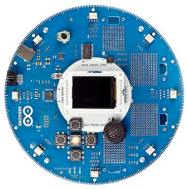
TQ

void loop() {
bip simple
attendre 1"
bip simple
attendre 1"
bip simple
attendre 2"
}

Quand on est un peu perdu avec les accolades cliquer derrière une accolade ouverte ou fermée et l'éditeur de texte vous encadre l'accolade soeur....

```
void loop() {
    Robot.beep(BEEP_SIMPLE);
    delay(100);
    Robot.beep(BEEP_SIMPLE);
    delay(100);
    Robot.beep(BEEP_SIMPLE);
    delay(200);
}
```

Télécharger avec le câble USB , toujours sur la carte supérieure du Robot



Complément sur ce premier travail :
*** Sauvegarder sous un nom "1 beep "**
*** Remplacer deux beep sur les trois par BEEP_DOUBLE et BEEP_LONG prendre garde aux "point virgule"**
*** Modifier les temporisations delay(1000) par des valeurs plus faibles**
***Sauvegarder sous un autre nom "3 beeps différents"**

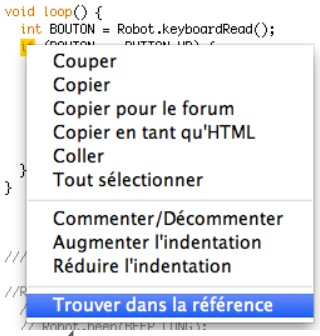
Le programme, qui a peu de choses à faire, débute à la fin du téléchargement grâce à l'alimentation par le port USB. Débrancher l'USB et mettre le robot sur "On ", ça doit repartir

2.1 Ajouter la prise en compte d'action(s) conditionnelle(s) avec un (des) bouton(s) poussoir(s)

A partir du programme précédent commencer par conditionner les trois beeps par l'action sur le bouton poussoir "UP"

// 1 Bouton poussoir 3 Beep

```
#include <ArduinoRobot.h>
#include <Wire.h>
#include <SPI.h>

void setup() {
// initialisation du Robot
Robot.begin();
// initialisation des fonctions du haut parleur
  Robot.beginSpeaker();
}


void loop() {
  int BOUTON = Robot.keyboardRead();
// sélectionner "if" faire un clic droit et sélectionner
"Trouver dans la référence" du menu local pour
consulter la doc qui donne quelques exemples
 if _____ {
_____
 _____
 _____
 _____
 _____
 _____
 _____
 _____
 _____
 }
}
// Une action sur le bouton UP lance le cycle en
gardant ce bouton actionné le programme
reboucle  (activité de la boucle "void loop"
```



Déclarations des bibliothèques C et Arduino
#include <ArduinoRobot.h>
#include <Wire.h>
#include <SPI.h>
Déclarations des variables globales;

Initialisation des fonctions complémentaires, premiers calculs
Robot.begin()
Robot.beginSpeaker();

le robot est sous tension ou raccordé à la liaison USB

void loop() {

Placer le nom du bouton actionné sans une variable locale BOUTON

int BOUTON = Robot.keyboardRead();

Beep si bouton UP actionné

TQ

SI

BOUTON = BUTTON_UP

if (BOUTON == BUTTON_UP) {

bip simple
attendre 0,1"
bip simple
attendre 0,1"
bip simple
attendre 0,2"
}

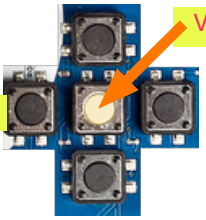
La fonction "Robot.keyboardRead()" revoie un des CINQ noms de bouton ci dessous.

int Variable = Robot.keyboardRead();

Variable = BUTTON_UP

Variable = BUTTON_MIDDLE

Variable = BUTTON_RIGHT



Variable = BUTTON_LEFT

Variable = BUTTON_DOWN

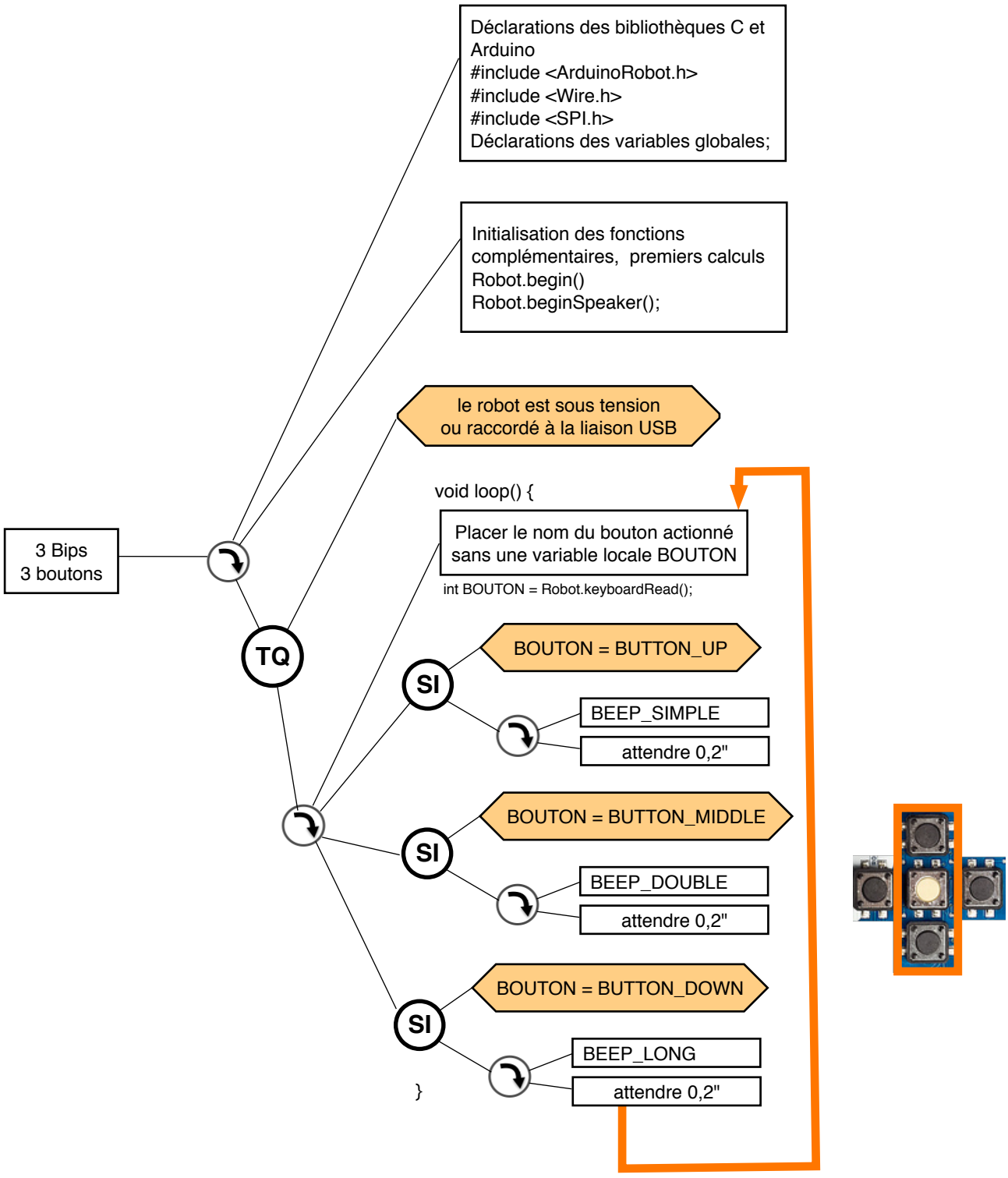
Opérateurs de comparaison sur le conditionnel
IF....THEN....ELSE....
if.....{.....} else {.....}

Comparaison Opérateurs :

```
x == y (x is equal to y)
x != y (x is not equal to y)
x <  y (x is less than y)
x >  y (x is greater than y)
x <= y (x is less than or equal to y)
x >= y (x is greater than or equal to y)
```

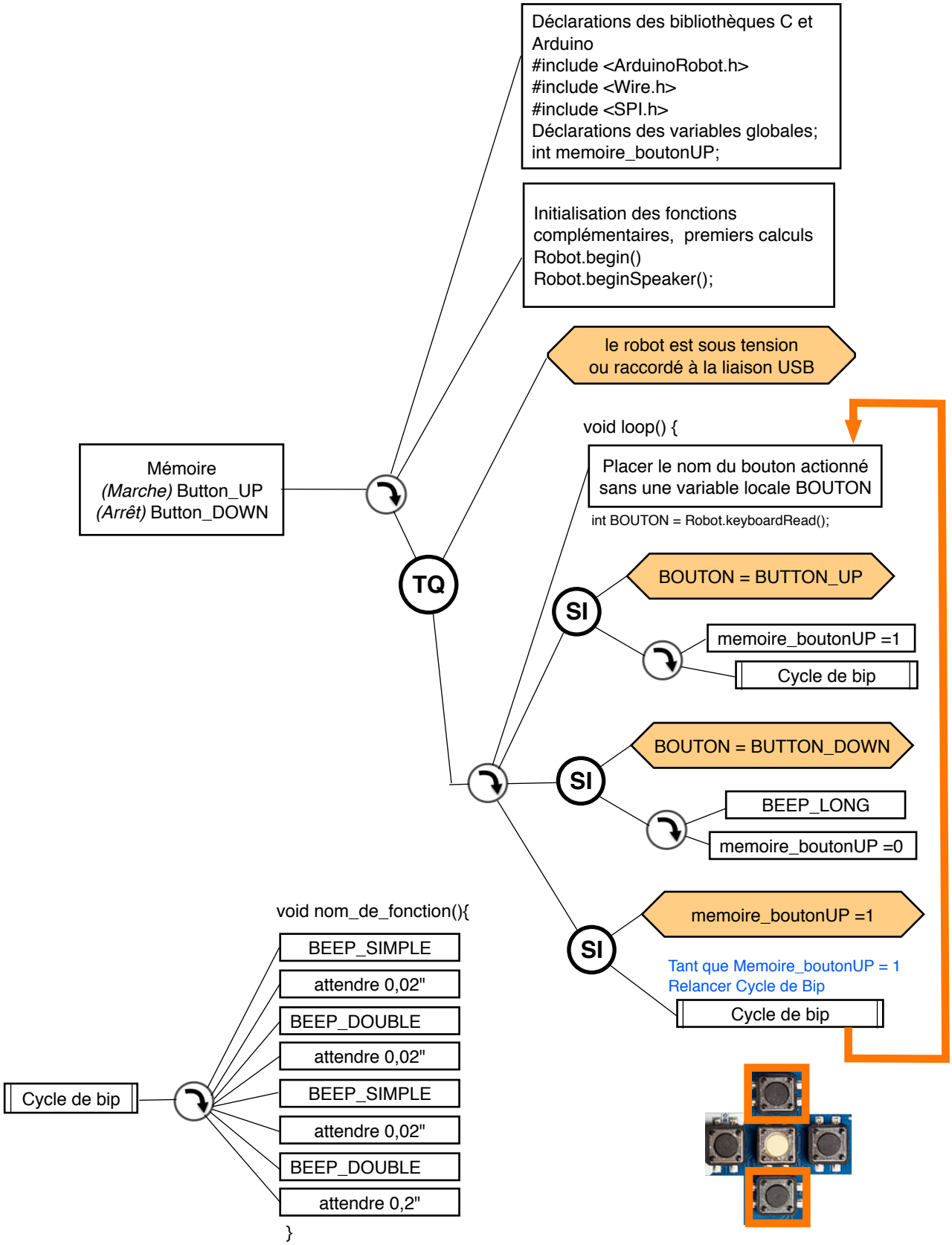
2.2 Ajouter la prise en compte d'action(s) conditionnelle(s) avec un (des) bouton(s) poussoir(s)

A partir du programme précédent, conditionner les trois beeps par l'action de trois boutons poussoirs différents .
Faire l'essai d'actionner une fois un poussoir puis de garder un bouton actionné .
Sans l'action d'un poussoir la boucle "loop" tourne à vide , avec un poussoir maintenu la boucle "TQ" (tant que) doit "tourner sur le même BEEP



3 - Mémoriser des actions et lancer des répétitions de fonctions

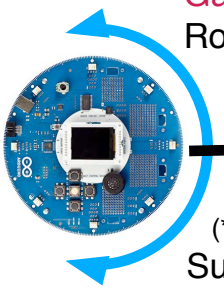
A partir du programme précédent, mémoriser l'action sur BUTTON_UP avec "memoire_boutonUP = 1 " . Repasser à 0 en actionnant BUTTON_DOWN .
Sous la boucle "loop" créer une fonction par "void nom_de_fonction(){" (choisir un nom) pour lister la séquence de beeps....



4 - Assurer des mises marche et arrêts des moteurs .

A partir du programme précédent,
Créer une fonction "void Moteurs(){" à la place de la fonction précédente pour assurer les rotations sur place du robot à 60% de la vitesse maxi, puis essayer les cycles Marche avant / Marche arrière (*1)

(*2)
Sur place à 60% vers la Gauche
Robot.motorsWrite(-150,150);



Arrêt **Robot.motorsWrite(0,0);**

(*3)
Sur place à 60% vers la Droite
Robot.motorsWrite(150,-150)
;

Déclarations des bibliothèques C et Arduino
#include <ArduinoRobot.h>
#include <Wire.h>
#include <SPI.h>
Déclarations des variables globales;
int memoire_boutonUP;

Initialisation des fonctions complémentaires, premiers calculs
Robot.begin()
Robot.beginSpeaker();

le robot est sous tension

Alternances de rotations sur place.

TQ

void loop() {

Placer le nom du bouton actionné dans une variable locale BOUTON

int BOUTON = Robot.keyboardRead();

SI BOUTON = BUTTON_UP

memoire_boutonUP =1

SI BOUTON = BUTTON_DOWN

BEEP_SIMPLE

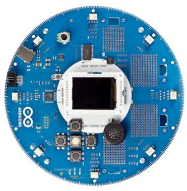
memoire_boutonUP =0

SI memoire_boutonUP =1

Tant que Memoire_boutonUP = 1
Relancer Moteur

Moteurs

(*1)



100% vers la l'Avant
Robot.motorsWrite(255,255);

100% vers la l'Arrière
Robot.motorsWrite(-255,-255);

void moteurs(){

Moteurs

Roue D AV 60% Roue G AR 60% (*2)

attendre 2 secondes

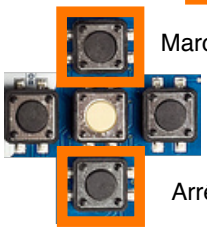
Arrêter 0,5 seconde

Roue D AR 60% Roue G AV 60% (*3)

attendre 2 secondes

Arrêter 0,5 seconde

}



Marche

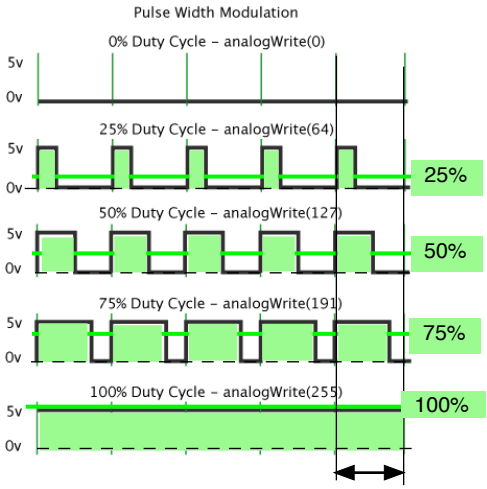
Arrêt

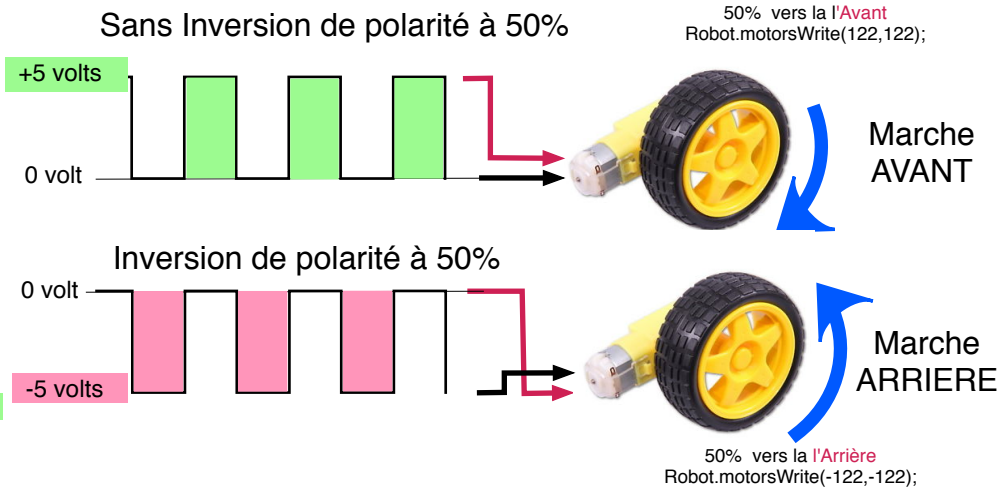
Pour arrêter, il faudra actionner (peut être), plus de 5 secondes, le bouton "DOWN" car la prise en compte se fera à la fin du sous programme "Moteur"

BIEN TÉLÉCHARGER LE PROGRAMME avec l'interrupteur sur "OFF"
Placer le robot sur la zone d'essais avec les rebords de protection

5 - Assurer la prise en compte du potentiomètre et l'affecter au paramètrage des moteurs.

Les commandes PWM Pulse Width Modulation (modulation de la largeur d'impulsion) synonyme de "commande par hacheur" ou par "Variation du rapport cyclique".





500 fois par SECONDE (ou plus)

1/500 Chaque 1/500 de seconde (par exemple), on alimente totalement ou partiellement les moteurs durant CE 1/500 ième de seconde . La tension moyenne dans le temps varie donc de 0 à 100 % de la tension nominale du moteur à courant continu (Cc). Comme il faut tourner dans les deux sens, la fonction(sous prog du robot) gère les deux moteurs avec des valeurs positives ou négatives entre -255 et 255. Robot.motorsWrite(-255,255) donnera une roue dans chaque sens à 100% de la vitesse MAXI.

Sur le programme précédent assurer les modifications permettant de remplacer les valeurs fixes de vitesses (ex Robot.motorsWrite(150,-150)) par des valeurs variables : Robot.motorsWrite(valPOT,-valPOT);
valPOT peut être remplacé par autre chose à condition d'être déclaré de la même façon dans les variables globales.

Pour ces modifications il faut ajouter la déclaration de variable globale "int valPOT;" en haut du programme au dessus de la zone Setup.

Lire la valeur du potentiomètre :
Cette valeur, à la source, est une entrée analogique convertie par le contrôleur en une valeur numérique qui varie de 0 à 1023.
Robot.knobRead() est une fonction de la bibliothèque du robot qui se charge d'aller lire cette valeur variable de 0 à 1023 et vous la "rapatrie" dans votre programme.
Y prendra la valeur du potentiomètre si l'on écrit Y = Robot.knobRead() dans un programme.
Les moteurs "attendent " (par la fonction Robot.motorsWrite) des valeurs entières entre -255 et 255, il faut donc convertir l'intervalle [0,1023] du potentiomètre vers l'intervalle "attendu" par les moteurs.

Entrer une valeur dans "valPOT" en changeant proportionnellement l'intervalle du potentiomètre.
La fonction MAP assure cette conversion et sera utilisée dans d'autre cas.le
map(value, fromLow, fromHigh, toLow, toHigh) EXEMPLES :

```
y = map(x, 1, 50, 50, 1);     Traduction : Une valeur "x" variable de 1 à 50 sera
inversée de 50 à 1 proportionnellement et placée dans "y"
```

```
y = map(x, 1, 50, 50, -100);  Traduction : Une valeur "x" variable de 1 à 50 sera
convertie sur un intervalle  de 50 à -100 proportionnellement et placée dans "y"
```
"toHigh" et "toLow" sont tout à fait théoriques car le high peut être plus petit que Low (ou pas).

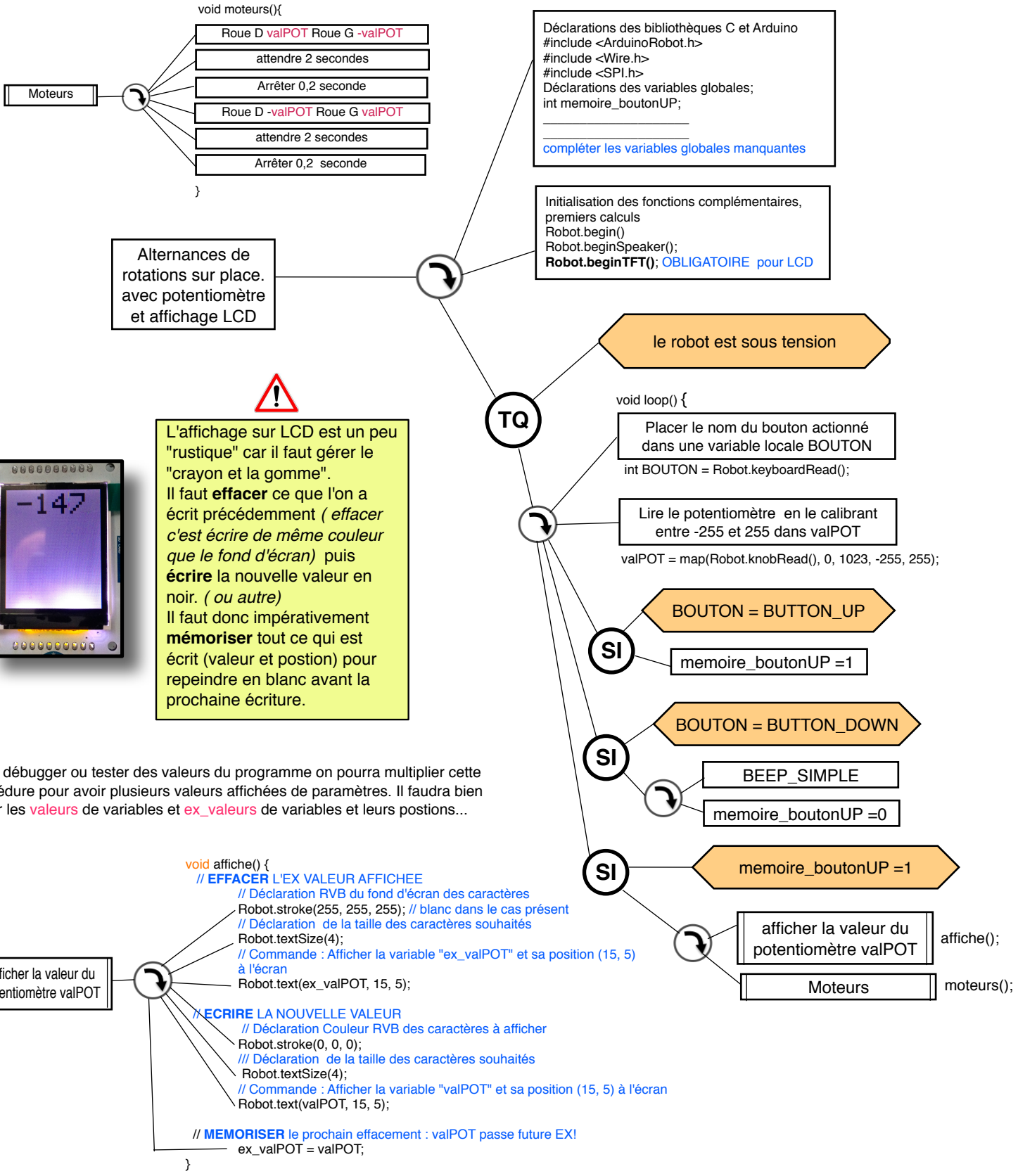
Robot.knobRead()

valPOT = map(Robot.knobRead(), 0, 1023, -255, 255);
Ira donc chercher la valeur du potentiomètre *Robot.knobRead* variable de [0,1023] et la re-conditionnera (par map) dans l'intervalle [-255,255]

Cette ligne sera à placer en début de boucle Loop. La prise en compte du potentiomètre nouvellement actionné sera effective après sortie du sous programme Moteur.

Bien modifier le sous programme comme indiqué en haut du paragraphe. Baisser les tempo de marche à 2 secondes et les arrêts à 0,2 seconde. Le potentiomètre sera donc lu toutes les 4,4 secondes dans cette structure.

6 - Afficher des variables sur l'écran LCD



Pour débugger ou tester des valeurs du programme on pourra multiplier cette procédure pour avoir plusieurs valeurs affichées de paramètres. Il faudra bien gérer les valeurs de variables et ex_valeurs de variables et leurs postions...

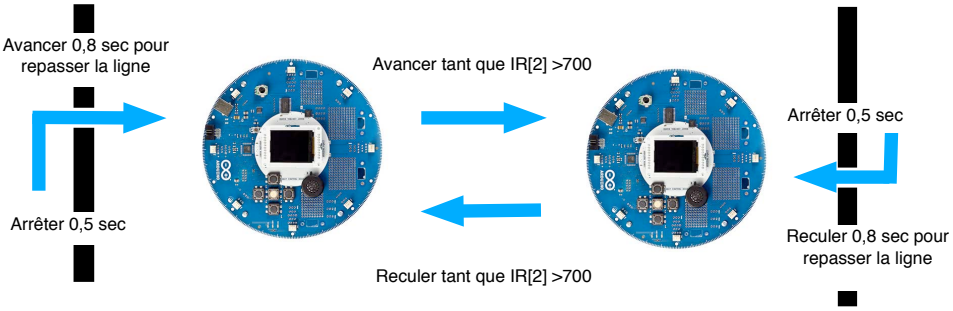
Afficher la valeur du potentiomètre valPOT

```
void affiche() {
  // EFFACER L'EX VALEUR AFFICHEE
        // Déclaration RVB du fond d'écran des caractères
        Robot.stroke(255, 255, 255); // blanc dans le cas présent
        // Déclaration  de la taille des caractères souhaités
        Robot.textSize(4);
        // Commande : Afficher la variable "ex_valPOT" et sa position (15, 5)
        à l'écran
        Robot.text(ex_valPOT, 15, 5);

  // ECRIRE LA NOUVELLE VALEUR
         // Déclaration Couleur RVB des caractères à afficher
        Robot.stroke(0, 0, 0);
        /// Déclaration  de la taille des caractères souhaités
         Robot.textSize(4);
        // Commande : Afficher la variable "valPOT" et sa position (15, 5) à l'écran
        Robot.text(valPOT, 15, 5);

  // MEMORISER le prochain effacement : valPOT passe future EX!
        ex_valPOT = valPOT;
}
```

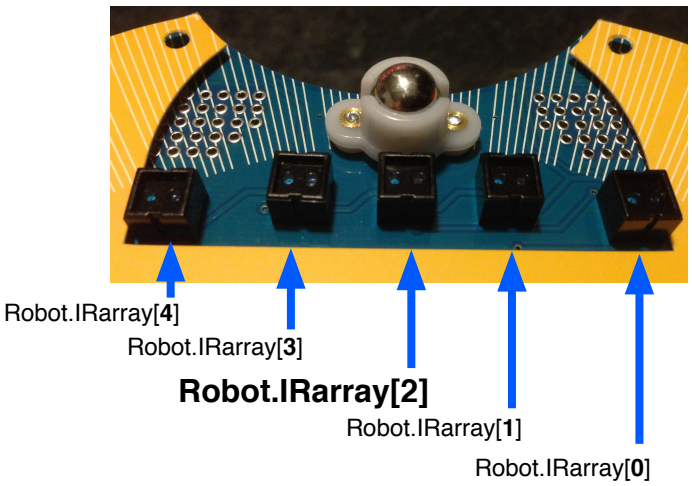
Assurer un double affichage sur l'écran LCD , mémoriser Robot.knobRead() dans une variable "POT" dans la boucle "loop", déclarer les variables globales POT et ex_POT, compléter le sous programme affiche() avec ces nouveaux paramètres en écrivant sous la variable "valPOT" du potentiomètre déjà programmée.

7 - Mettre en oeuvre les capteurs optiques de la carte moteur

Ce programme va prendre en compte l'état du capteur optique central sous le robot. C'est donc le Robot.IRarray[**2**] qui va inverser les sens de marche des moteurs... Ne pas modifier le corps principal du programme précédent, modifier seulement la procédure "Moteurs"

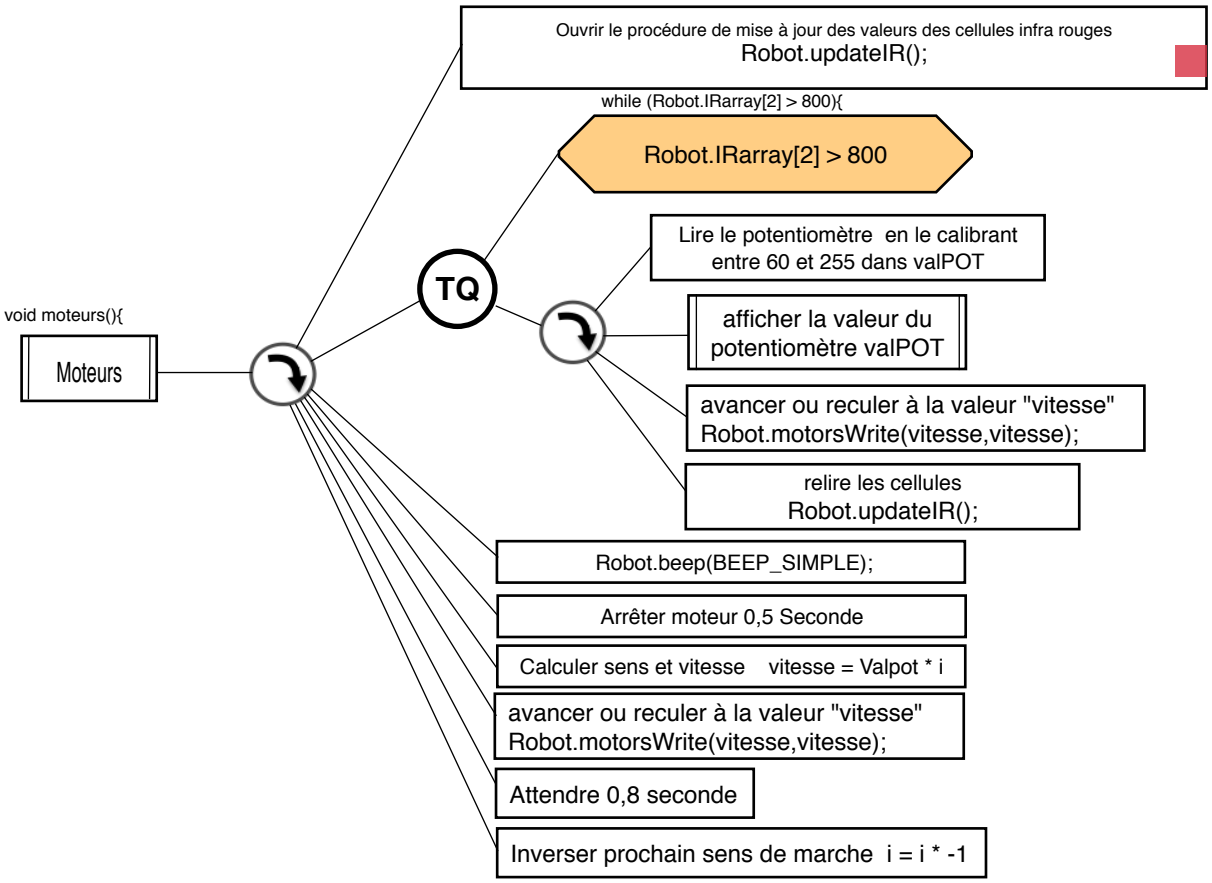


Robot.updateIR(); *cette fonction lit les CINQ valeurs des cinq capteurs infra-rouge sous la carte moteur. (de 0 à 4)*



Compléter le haut du programme et les quelques variables globales.

```
#include <ArduinoRobot.h> //
#include <Wire.h>
#include <SPI.h>
// décalration des variables
int memoire_bouton;
int valPOT;
int ex_valPOT;
int i;
int vitesse;
//int ex_valeur;

void setup() {
  // init
  Robot.begin();
  Robot.beginSpeaker();
  Robot.beginTFT();
  serial.begin(9600)
  i = 1;
  vitesse = 70;
}
```



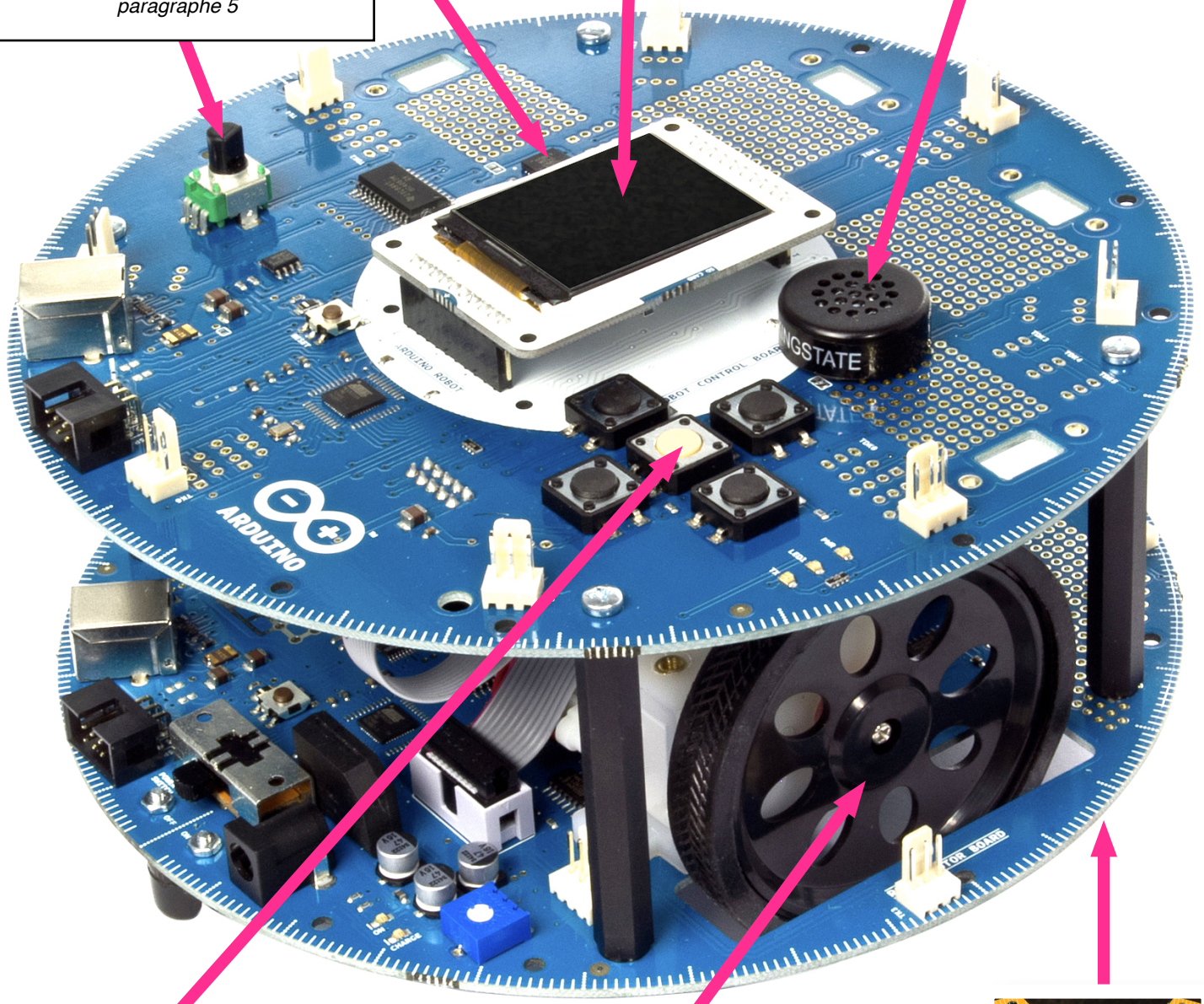
Rappel et sommaires des commandes principales sur le robot

"Lire" le COMPAS
variable = Robot.compassRead();
paragraphe 8

Afficher sur écran LCD
init Robot.beginTFT();
Robot.stroke(0, 0, 0);
Robot.textSize(4);
Robot.text(variable, 15, 5);
paragraphe 6

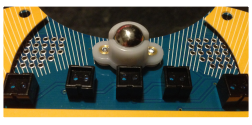
Commander mini HP
init Robot.beginSpeaker();
(Robot.beep(BEEP_SIMPLE);
Robot.playMelody(chaine);
paragraphe 1

"Lire" le potentiomètre
variable = Robot.knobRead(),
paragraphe 5



"Lire" les boutons actionnés
int variable = Robot.keyboardRead();
paragraphe 2.1

Commander les moteurs
Robot.motorsWrite(-255,255);
Robot.motorsStop();
paragraphe 4 et 5



"Lire" les capteurs optiques
init serial.begin(9600)
Robot.updateIR();
Robot.IRarray[variable de 0 à 4]
paragraphe 7