Dossier De Conception (DDC)

du projet

Robot Mini-Sumo

Responsabilité documentaire

Action	NOM Prénom	Fonction	Date	Signature
Rédigé par	BERNYER Quentin BUNEL Kyrian LARRODE Lucas BATTAGLIN Colin SENES Tom CAPET Guillaume	Technicien	21/09/2022	
Approuvé par	(IUT GEII Bdx)	Chef de projet	21/09/2022	
Approuvé par	Tournoi National de Robotique Sumo (TNRS) (IUT GEII Bdx)	Client	-	

Suivi des révisions documentaires

Indice	Date	Nature de la révision
1	01/09/2022	Publication préliminaire du DDC document à compléter par le Technicien.
2	23/09/2022	Première publication

Documents de références

Sigle	Référence	Titre	Rév.	Origine
[CDC]	RMS_CDC	Cahier des charges	1	IUT GEii Bdx

Table des matières

1. Nature du document 5

2. Conception préliminaire du produit 5

2.1 Architecture Électronique 5

2.1.1 Choix des capteurs adversaire 7

2.1.2 Choix des capteurs de sol 8

2.1.3 Choix du pont en H 9

2.1.4 Choix des moteurs 10

2.1.5 Choix du régulateur de tension 11

2.1.6 Choix de la batterie 12

2.2 Architecture Informatique 14

2.2.1 Fonctions d'acquisition 16

2.2.2 Fonctions d'action 17

2.2.3 Fonctions d'énergie 18

2.2.4 Fonctions de traitement 19

2.3 Conclusion de la conception préliminaire du produit 19

3. Conception détaillée du produit 20

3.1 Conception détaillée du Robot Mini-Sumo 20

3.2 Conception détaillée de la partie acquisition 21

3.2.1 Capteurs adversaires 21

3.2.1.1 Schéma électrique des capteurs adversaires 21

3.2.1.2 Code informatique des capteurs adversaires 21

3.2.2 Capteurs de ligne 22

3.2.2.1 Schéma électrique des capteurs de ligne 22

3.2.2.2 Code informatique des capteurs de sol 22

3.2.3 Capteurs de télécommande 23

3.2.3.1 Schéma électrique des capteurs de télécommande 23

3.2.3.2 Code informatique des capteurs de télécommande 23

3.3 Conception détaillée de la partie action 26

3.3.1 Pont en H 26

3.3.1.1 Schéma électrique du pont en H 26

3.3.1.2 Code informatique du pont en H 26

3.4 Conception détaillée de la partie énergie 28

3.4.1 Pont diviseur de tension 28

3.4.1.1 Schéma électrique du pont diviseur de tension 28

3.4.1.2 Code informatique du pont diviseur de tension 29

3.4.2 régulateur de tension 30

3.4.2.1 Schéma électrique du régulateur de tension 30
3.4.3 Condensateur de découplage 30
3.4.3.1 Schéma électrique condensateur de découplage : 30
3.5 Conception détaillée de la partie traitement 31
3.5.1 Le microcontroleur ATMEGA328P-PU 31
3.5.1.1 Schéma électrique du microcontroleur 31
3.5.1.2 Code informatique du microcontroleur 32
4. Dérisquage des solutions techniques retenues 36
4.1 Dérisquage aquisition 36
Dérisquage action 37
4.3 Conclusion de la simulation / prototypage rapide du produit 39
5. Conclusion de la conception du produit 40
6. Matrice de conformité du produit 41

1. Nature du document

Ce document est un dossier de conception et a pour but de détailler la conception du produit développé. Il apporte ainsi des preuves de la conformité du produit par rapport à l'ensemble des exigences client. Le paragraphe 3 du [CDC] décrit de façon plus détaillée la nature et le positionnement de ce document dans l'arborescence documentaire du projet.

2. Conception préliminaire du produit

Ce chapitre décrit l'architecture fonctionnelle du produit. Il apporte les premiers éléments de preuve de la faisabilité du produit vis-à-vis des exigences client.

2.1 Architecture Électronique

Référence de pré-conception : CPR01

Exigences client vérifiées par préconception :

EXIG_AUTONOMIE || *EXIG_ADVERSAIRE* || *EXIG_DEPLACEMENT* || *EXIG_FUITE* || *EXIG_DEPART* || *EXIG_CARTE* || *EXIG_COMBAT.*

Afin de répondre au cahier des charges, une analyse globale des exigences a conduit à l'architecture fonctionnelle présentée ci-dessous.

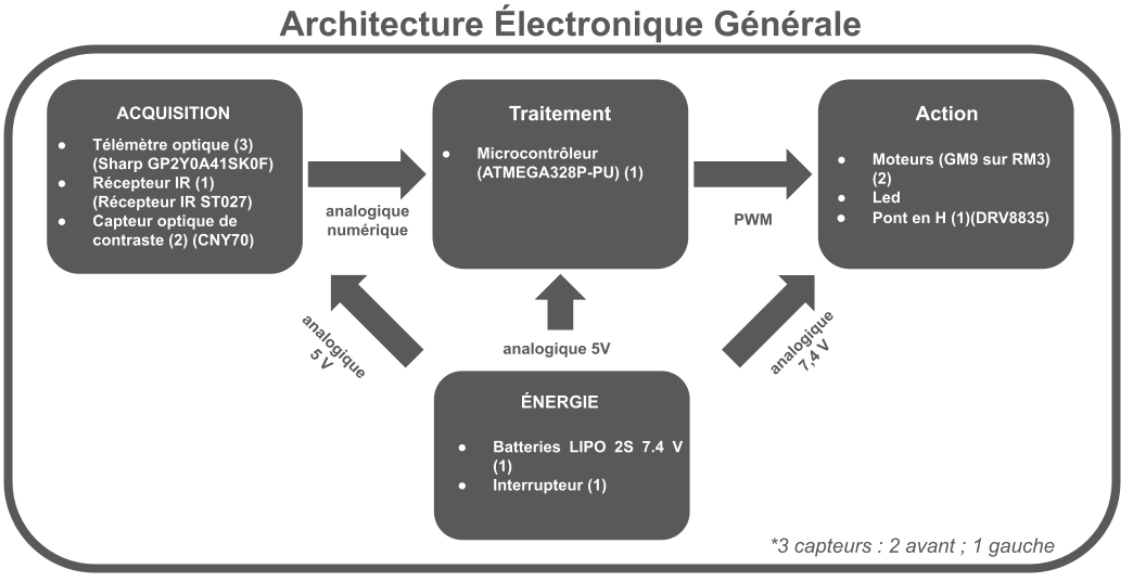


Figure 1 : Architecture électronique RMS

Bloc Acquisition :

➔ Pour répondre à l'exigence EXIG_ADVERSAIRE et EXIG_FUITE le robot calcule l'emplacement de l'adversaire grâce à 3 capteurs optiques Sharp GP2Y0A41SK0F.

Nous allons en mettre deux devants et un sur le côté gauche.

➔ Pour répondre à l'exigence EXIG_DEPART le robot doit recevoir un signal infrarouge provenant d'une télécommande grâce à un récepteur infrarouge, ce qui permet de le mettre en marche.

➔ Afin de respecter les règles du combat de sumo, le robot doit capter s'il sort du dohyo grâce à deux capteur optique de contraste CNY70.

Nous allons mettre un capteur de ligne devant et derrière.

Bloc Traitement :

➔ Pour répondre à l'exigence EXIG_COMBAT le robot est capable d'effectuer des fonctions de traitement grâce à un microcontrôleur ATMEGA328P-PU.

Bloc Action :

➔ Pour répondre à l'exigence EXIG_DEPLACEMENT le robot est libre de déplacement grâce à deux moteurs GM9 sur RM3.

➔ Pour répondre à l'exigence EXIG_DEPLACEMENT un pont en H de référence DRV8835 alimente les moteurs.

Bloc Énergie :

➔ Pour répondre à l'exigence EXIG_AUTONOMIE une batterie LIPO 2S 7,4V alimente le système.

➔ Pour répondre à l'exigence EXIG_INTERUPTEUR un interrupteur contrôle l'alimentation du système.

➔ Pour répondre à l'EXIG_SECUR_BATT, un pont diviseur de tension sera fait en sortie de la batterie et dont la sortie sera reliée à une entrée analogique du microcontrôleur.

2.1.1 Choix des capteurs adversaire

Référence de pré-conception : CPR02

Exigences client vérifiées par préconception : *EXIG_ADVERSAIRE, EXIG_FUITE, EXIG_LUMINOSITE, EXIG_COTE.*

Solution technique proposée	Conformité à toutes les exigences (oui : 1 / non : 0)	Distance min et max de détection	Angle de vision	Facilité d'utilisation du signal de sortie	Compatibilité de la tension avec les sources disponibles	Faible consommation en courant	Boîtier composant facile à utiliser/souder	Faible Prix	Disponibilité	Total (note)	Commentaires (si nécessaire)
Capteur de mesure Sharp GP2Y0A41SK0F	1	4 - 30 cm	ligne droite	oui	4.5 - 5.5V	12 mA /22 max	oui	9,90€	oui	7	Ce capteur a été choisi
		Note :	1	1	1	1	1	1	1		
Capteur de mesure Sharp GP2Y0A021YK	0	10 - 80 cm	ligne droite	oui	4.5 - 5.5V	30mA	oui	11,50€	oui	6	
		Note :	1	1	1	0	1	0.5	1		
Capteur de mesure Sharp GP2Y0A02YK	0	20 - 150 cm	ligne droite	oui	4.5 - 5.5V	33mA	oui	12,80€	oui	6	
		Note :	1	1	1	0	1	0	1		
VISHAY SILICONIX CNY70 CAPTEUR OPTIQ E/S TRANSISTOR		0.5 - 10 mm				50ma					
		Note :									

Figure 2.1.1.1 : FAD pour le choix des capteurs optiques

Nous avons choisi le capteur optique SHARP GP2Y0A41SK0P. Ce capteur a une plage de mesure de 4 à 30 cm. Le Dohyo a un diamètre de 77 cm. La distance de réaction est donc suffisante (30cm). Son coût est le plus faible de tous les capteurs optiques proposés. De plus, en cas de face à face, ce capteur permet de détecter le robot contrairement aux autres qui détectent à une distance minimale de 10 cm. Ce qui correspond à la taille d'un robot et peut poser problème dans le cas d'une rencontre frontale.

Figure 2.1.1.2 : capteur SHARP GP2Y0A41SK0P

2.1.2 Choix des capteurs de sol

Référence de pré-conception : CPR03

Exigences client vérifiées par préconception : *Sans objet (Pas d'exigences dans le CDC, explication dans le document règles)*

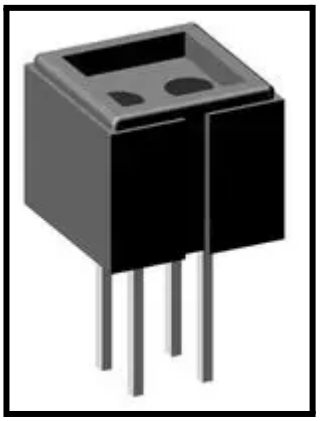


Figure 2.1.2.1 : Capteur CNY70

Un seul capteur de contraste nous est proposé. Il s'agit du capteur CNY70. Ce dernier sera pris par défaut.

Il nous permettra de capter le passage du robot sur la ligne blanche du Dohyo.

Selon les règles du combat de sumo, une sortie du dohyo entraîne une disqualification. Nous avons donc choisi de mettre un capteur devant et derrière pour s'assurer que le robot se rende compte de sa sortie du Dohyo lorsqu'il avance ou qu'il recule.

2.1.3 Choix du pont en H

Référence de pré-conception : CPR04

Exigences client vérifiées par préconception : *EXIG_DEPLACEMENT*

Solution technique proposée	Conformité à toutes les exigences (oui : 1 / non : 0)	Critères de sélection (0 ou 1)								**Total (produit des valeurs précédentes)**	Commentaires (si nécessaire)
		Compatibilité de la tension avec les sources disponibles	Courant max de sortie du pont	Manière de piloter le pont	Rendement énergétique du pont	Boîtier composant facile à utiliser/souder	Faible Prix	Disponibilité	Protections		
Pololu Commande de 2 moteurs CC DRV8835 2x1,2A	1	2 - 11 V	2*1,2 A	PWM		14-pin DIP	12,05€	✓	Under-voltage over-current over-temperature Reverse-voltage	7	✓
		1	1	1		1	1	1	1		
Pololu Commande de 2 moteurs CC DRV8833 2x1,2A	1	2.7 to 10.8 V	2*1,2 A	PWM		16-pin	14,50€	✓	Overcurrent protection Deglitch time Thermal shutdown Undervoltage Lockout	4	X
		0	1	1		0	0	1	1		

Figure 2.1.3.1 : FAD pour le choix du pont en H.

Après avoir déterminé les critères pour répondre à l'exigence EXIG_DEPLACEMENT nous avons comparé les caractéristiques des composants à notre disposition en consultant leurs datasheet. Nous avons pu conclure que le pont en H Ref : DRV8835 est celui qui répond au mieux à l'exigence grâce à la fiche d'aide à la décision ci-dessus en figure 2.1.3.1.

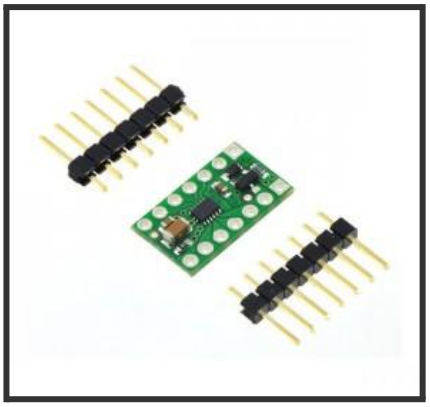


Figure 2.1.3.2 : Pont en H DRV8835

2.1.4 Choix des moteurs

Référence de pré-conception : CPR04
Exigences client vérifiées par préconception : *EXIG_DEPLACEMENT*

Figure 2.1.4.1 : Motoréducteur GM9 RM3

Ce moteur est imposé par la liste des composants disponibles. Il s'agit d'un moteur à courant continu doté d'un réducteur avec un ratio de 224 pour 1. Il nous permet de valider l'exigence EXIG_DEPLACEMENT grâce à sa capacité de tourner à 360° dans les deux sens.

2.1.5 Choix du régulateur de tension

Référence de pré-conception : CPR05

Exigences client vérifiées par préconception : *EXIG_CARTE*

Solution technique proposée	Conformité à toutes les exigences (oui : 1 / non : 0)	Tension max d'entrée	Tension de sortie	Faible chute de tension sortie-entrée	Courant de sortie max	Boîtier composant facile à utiliser/souder	Faible Prix	Disponibilité	
LP2985AIM5-3.3/NOPB 3.3V CMS		16V	3.3V	350mV	150mA	5 pins petit	81 cent	1	
LP2985AIM5-5.0/NOPB 5V CMS		16V	5V	350mV	150mA	5 pins petit	81 cent	1	
MC33269DT-3.3G IC		20V	3.3V	1.35V	800mA	1	77cent	1	
MC33269DT-5.0G IC		20V	5V	1.35V	800mA	1	77cent	1	

Figure 2.1.5.1 : FAD pour le choix du régulateur de tension

Le régulateur de tension est nécessaire afin d'assurer le bon fonctionnement du microcontrôleur. Ici le microcontrôleur qui nous est proposé est l'ATMEGA 328P-PU qui fonctionne sous 1.8V à 5.5V. Il est donc nécessaire d'utiliser un régulateur de tension afin d'assurer la sécurité du microcontrôleur car les batteries proposées fournissent une tension de 7,4V. Notre choix se porte sur le MC33269DT-5.0G qui permet d'établir une tension de 5V et un courant de sortie max de 800 mA.

De plus on choisit celui-ci car on pourra également avoir une tension de 5V en sortie du microcontrôleur ce qui nous permettra de pouvoir alimenter les composants contrôlés par le microcontrôleur alors qu'avec le régulateur qui propose 3.3V en sortie cela risque de ne pas suffire. De plus, les capteurs proposés fonctionnent sous 4.5V-5.5V donc ce régulateur de tension semble être le seul bon choix dans cette liste.

2.1.6 Choix de la batterie

Référence de pré-conception : CPR06

Exigences client vérifiées par préconception : *EXIG_AUTONOMIE*

Solution technique proposée	Conformité à toutes les exigences (oui : 1 / non : 0)	Critères de sélection (exemples) (valeur de 1 à 5)			Total (produit des valeurs précédentes)	**Commentaires (si nécessaire)**
		Encombrement	Faible Prix	Disponibilité		
Lipo CONRAD ENERGY 7.4 V 1200 mAh	1	132 x 22 x 13 mm	20,99€	oui		
		score /5 : 1	3	5	15	
Lipo CONRAD ENERGY 7.4 V 800 mAh	1	56 x 31 x 17.5	13,49€	non		
		Encombrement :3	1	1	3	
Lipo CONRAD ENERGY 7.4 V 450 mAh	1	56 x 31 x 12	9,49€	oui		
		5	5	4	100	

Figure 2.1.6.1 : FAD pour le choix de la batterie

Disponibilité des batteries :

Lipo CONRAD ENERGY 7.4 V 450 mAh : disponible le 21/09/22

Lipo CONRAD ENERGY 7.4 V 800 mAh : non disponible le 21/09/22

Lipo CONRAD ENERGY 7.4 V 1200 mAh : disponible le 21/09/22

L'exigence d'autonomie du cahier des charges nous impose que le robot doit être capable de se déplacer en continu sans aucun contact avec un obstacle pendant 50 minutes.

Il faut utiliser des batteries LIPO 2S qui sont présentes au département.

Seulement il reste à choisir la capacité de la batterie en mAh.

Pour cela nous devons calculer la consommation totale approximative du robot mini sumo.

Sachant que ce sont les deux motoréducteurs GM9 sur RM3 imposés par le cahier des charges qui consomment le plus de courant.

Chacun de ces moteurs consomment 100 mA sans charge environ et 800mA au démarrage mais seulement pendant une seconde à peu près.

On considère que ces deux moteurs fonctionnent sans charge car le robot doit être capable de se déplacer en continu sans aucun contact avec un obstacle ce qui fait qu'il n'y a pas d'autres forces résistives que les forces de frottements entre le dohyo et les roues du robot ainsi qu'entre le robot et l'air.

Il n'y a pas de forces répulsives qui s'appliquent comme si par exemple notre robot luttait contre le robot adverse ou s'il heurtait un objet quelconque ce qui aurait pour effet de créer une force opposée au déplacement.

Le couple résistif qui s'applique sur le moteur est donc très faible ce qui permet de prendre les valeurs de courant pour le moteur sans charge.

Consommation globale du robot mini sumo :

Les deux moteurs consomment 200 mA environ, le microcontrôleur 20 mA en moyenne, les 3 télémètres optiques consomment eux 66 mA, le pont en H consomme seulement 0,45 mA et les deux capteurs optiques de contraste consomment chacun 50 mA.

Il faut que le robot fonctionne pendant 50 minutes soit 0,833 heure.

Consommation globale = 200 + 20 + 66 + 0,45 + 100(50 mA * 2) = 387 mA à peu près.

Car cette consommation globale est approximative.

Batterie à choisir :

Autonomie batterie = 387 mA * 0,833 h = 322 mAh approximativement.

Où on doit ajouter également la consommation des moteurs au démarrage soit : 1730 mA (à peu près 800 * 2) * 1/3600 (1 seconde) = 0,5 mAh.

Donc Autonomie batterie = 322 mAh + 0,5 mAh = 322,5 mAh.

Nous avons donc choisi la batterie Lipo 7.4 V 450 mAh car elle peut fournir jusqu'à 450 mA pendant 1 heure et 322,5 mAh < 450 mAh.

Mais si on se rend compte dans la suite du projet que notre robot consomme plus d'énergie que prévu nous prendrons alors la batterie Lipo 7.4 V 800 mAh.

2.2 Architecture Informatique

Référence de pré-conception : CPR07

Exigences client vérifiées par préconception :

EXIG_ADVERSAIRE || EXIG_FUITE || EXIG_LUMINOSITE || EXIG_DEPART || EXIG_COTE || EXIG_DEPLACEMENT || EXIG_AUTONOMIE || EXIG_SECUR_BAT || EXIG_COMBAT.

Pour la réalisation de l'architecture informatique nous avons choisi de la représenter sous la forme d'un grafcet. Pour cela nous avons anticipé les mouvements du robot en fonction des différentes situations qui sont décrites dans les figures 2.2.0.1 et 2.2.0.2.

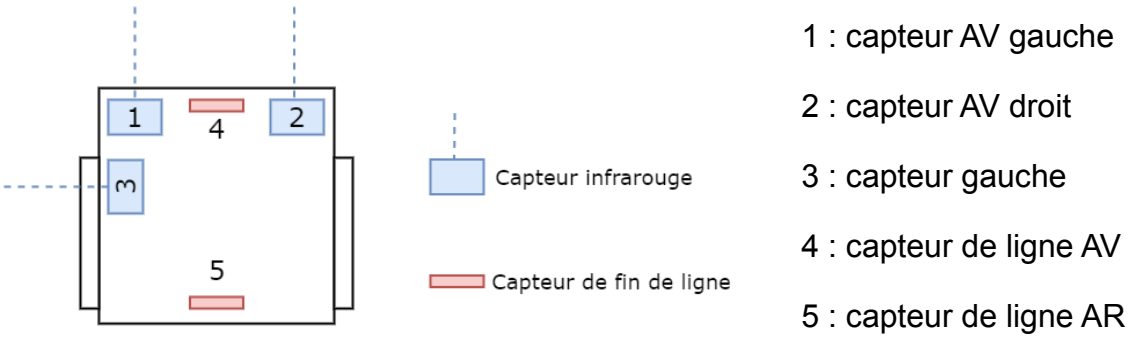


Figure 2.2.0.1 : Schéma de numérotation des capteurs du robot.

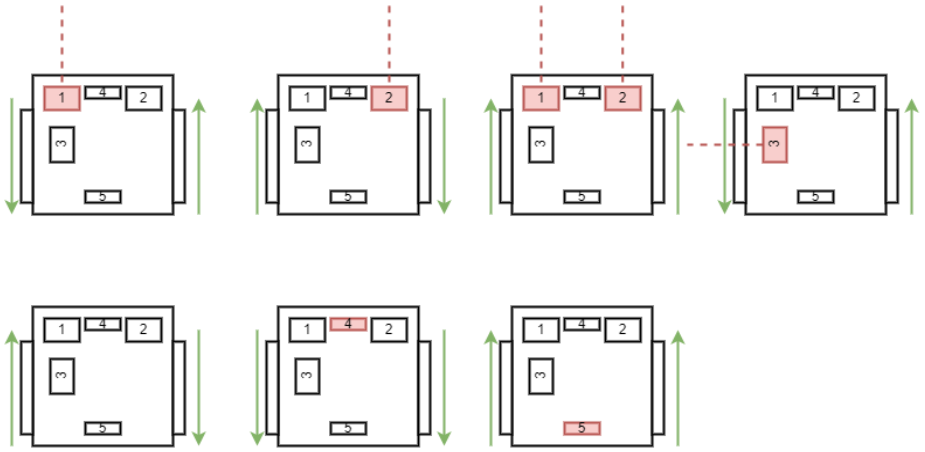


Figure 2.2.0.2 : Schéma du comportement du robot pendant le combat.

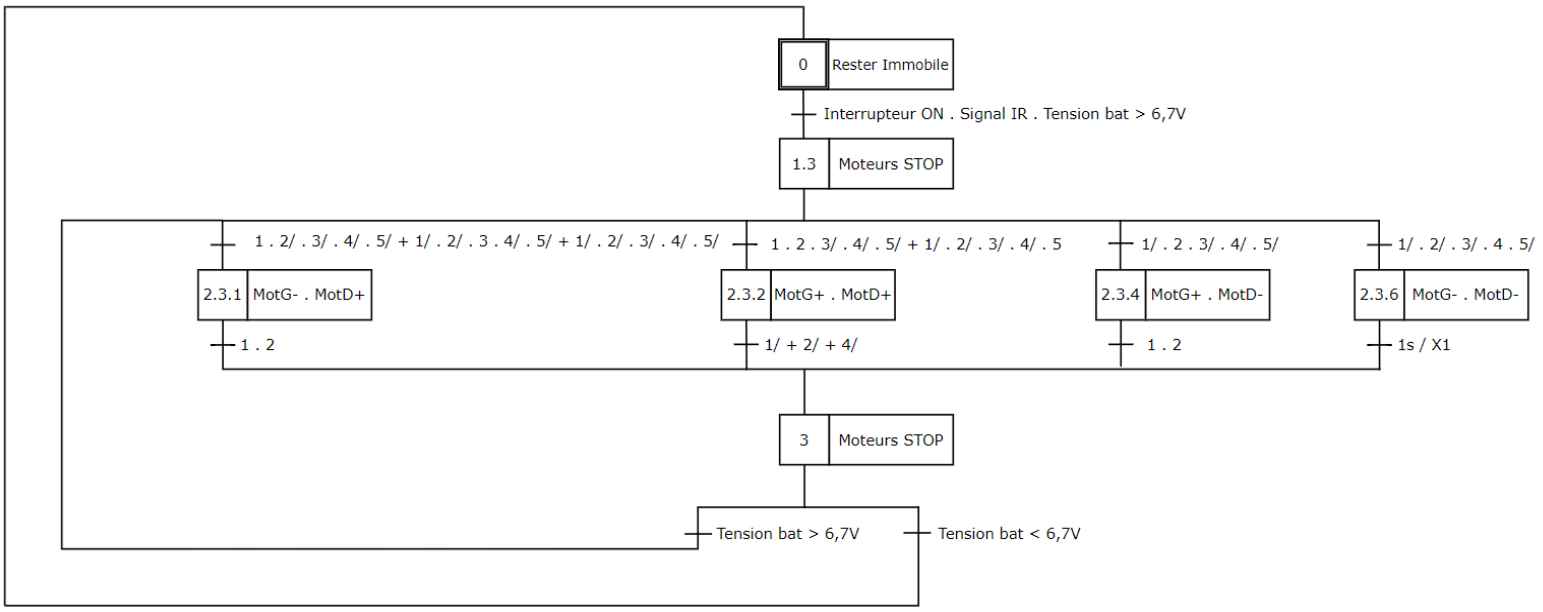


Figure 2.2.0.3 : Grafcet Architecture Informatique.

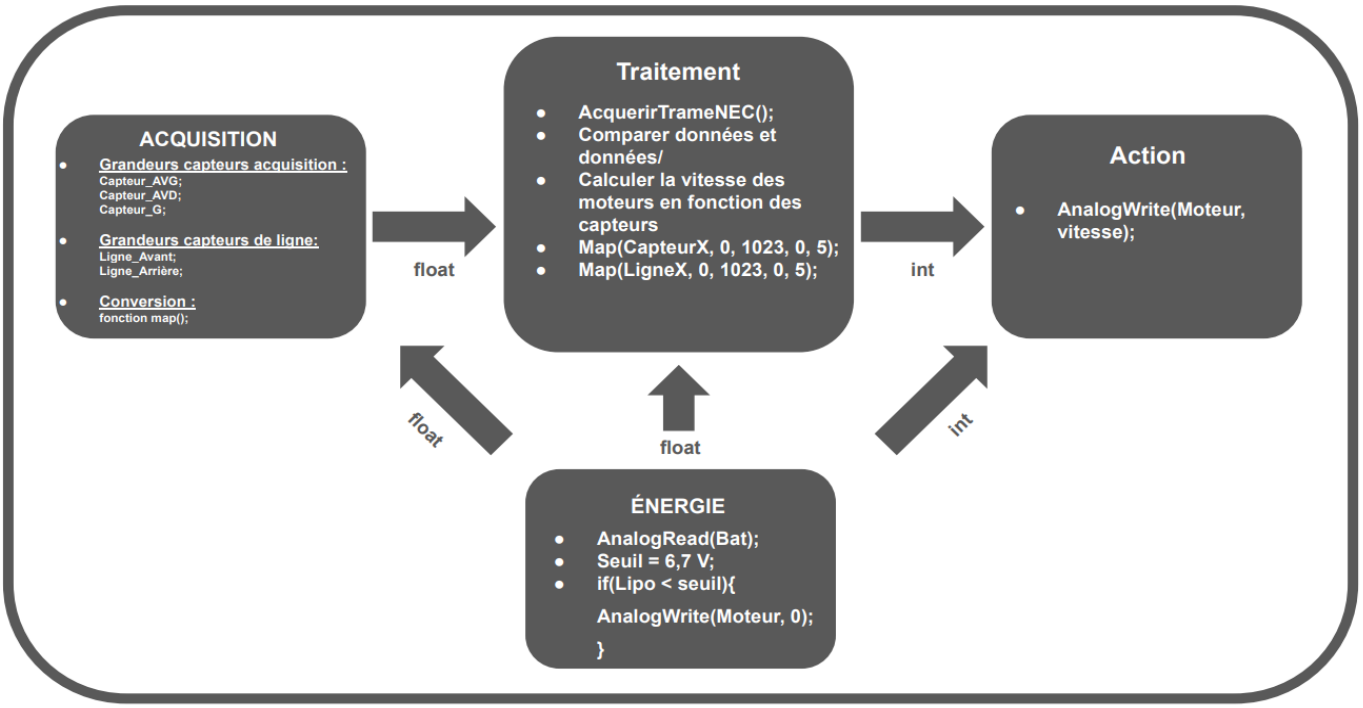


Figure 2.2.0.4 : Synoptique architecture informatique.

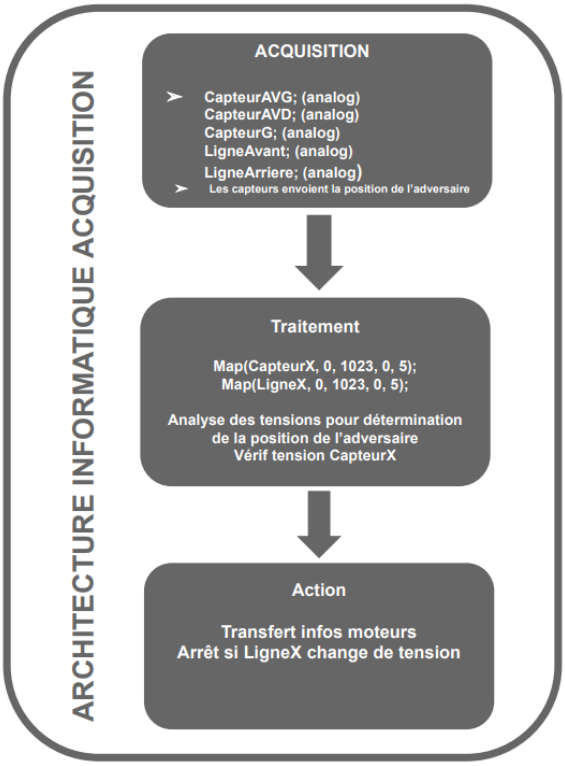
2.2.1 Fonctions d'acquisition

Référence de pré-conception : CPR08

Exigences client vérifiées par préconception :

EXIG_ADVERSAIRE || *EXIG_FUITE* || *EXIG_LUMINOSITE* || *EXIG_DEPART* || *EXIG_COTE.*

Figure 2.2.1.1 : synoptique informatique acquisition



➢ Le microcontrôleur récupère les informations provenant des capteurs optiques et du capteur de ligne.

➢ Ces informations sont traitées pour pouvoir déterminer la position de l'adversaire et les mouvements qui vont devoir être effectués. Les capteurs seront reliés aux broches analogiques de l'Arduino. Nous utiliserons la fonction map() qui permet d'effectuer des conversions pour transformer les grandeurs envoyées par les capteurs. Les capteurs possèdent tous une variable différentes permettant de les différencier.

(Ex : CapteurAVG : Capteur avant gauche)

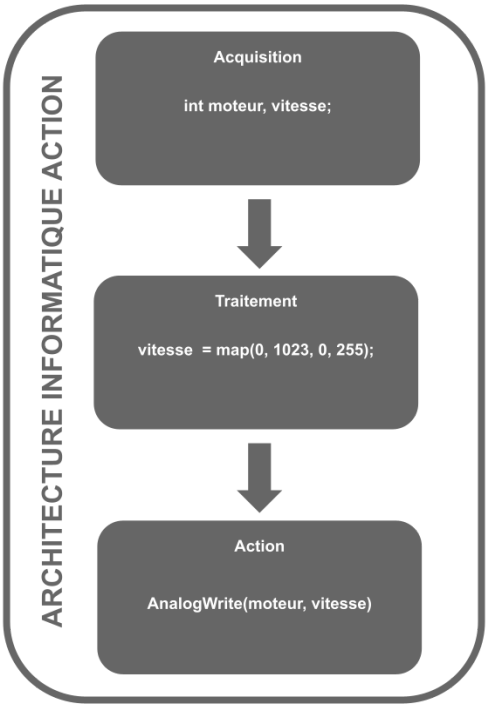
➢ Les informations sont converties pour être transmises aux moteurs.

2.2.2 Fonctions d'action

Référence de pré-conception : CPR09

Exigences client vérifiées par préconception : *EXIG_DEPLACEMENT.*

Figure 2.2.2.1 : synoptique informatique action.



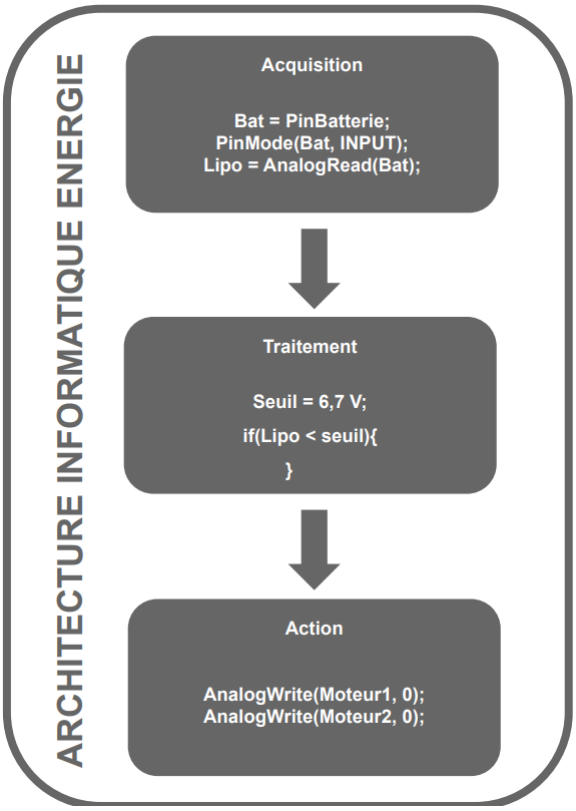
- En fonction de ce que le microcontrôleur va récupérer comme information à partir des capteurs, la fonction AnalogWrite(moteur, vitesse) permettra de contrôler les moteurs en leur envoyant un signal PWM.
- La variable moteur permet de sélectionner le moteur gauche ou droit.
- La variable vitesse permet de faire varier le rapport cyclique du signal PWM.

2.2.3 Fonctions d'énergie

Référence de pré-conception : CPR10

Exigences client vérifiées par préconception : *EXIG_AUTONOMIE || EXIG_SECUR_BAT.*

Figure 2.2.3.1 : synoptique informatique energie



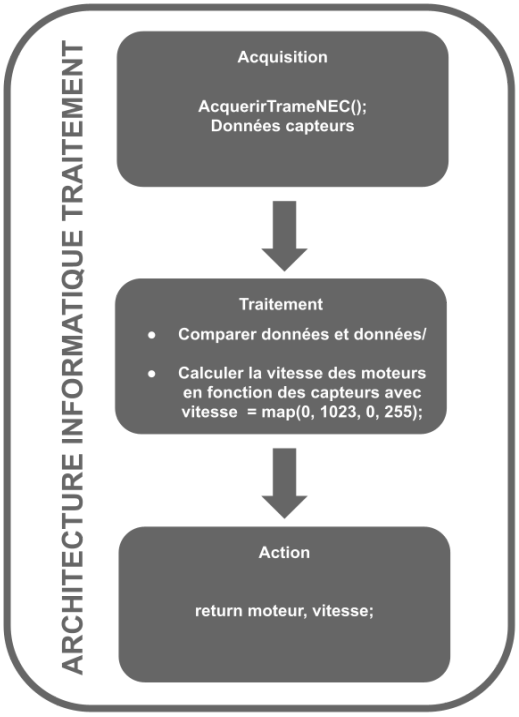
- Pour répondre à l'exigence EXIG_SECUR_BAT le robot doit pouvoir couper l'alimentation des moteurs quand la tension de la batterie passe en dessous des 6,7 V.
- Pour cela nous allons utiliser les fonctions :"PinMode(Bat, INPUT);" qui configure la broche Bat en tant qu'entrée.
- "Lipo = AnalogRead(Bat);" pour que la variable Lipo prenne la valeur de la tension de la batterie.
- "AnalogWrite(Moteur, 0);" afin d'agir sur les moteurs et de les arrêter, en générant un signal PWM avec un rapport cyclique de 0% sur la broche du moteur.

2.2.4 Fonctions de traitement

Référence de pré-conception : CPR11

Exigences client vérifiées par préconception : *EXIG_COMBAT.*

Figure 2.2.4.1 : synoptique informatique traitement.



- Pour répondre à l'exigence EXIG_COMBAT, le robot doit être capable de commencer le combat après le réception de la commande infrarouge de la télécommande et d'exploiter les différents capteurs afin d'agir en fonction des mouvements de l'adversaire.

- Pour cela, nous utiliserons la fonction : "AcquérirTramNEC();" afin de traiter l'information infrarouge provenant de de la télécommande.

- "AnalogWrite(Moteur, Vitesse);" pour que le robot calcule la vitesse des moteurs en fonction des informations reçues des capteurs.

2.3 Conclusion de la conception préliminaire du produit

L'ensemble des exigences ont été respectées. Nous avons ajouté un capteur de contraste permettant de capter la ligne blanche du Dohyo. Ce capteur n'est soumis à aucune exigence. Nous pouvons passer à la phase de conception du produit.

3. Conception détaillée du produit

Ce chapitre détaille la conception du produit développé. Il constitue une preuve de la conformité du produit. Chaque paragraphe de cette étude fait donc clairement référence aux exigences client issues du [CDC].

3.1 Conception détaillée du Robot Mini-Sumo

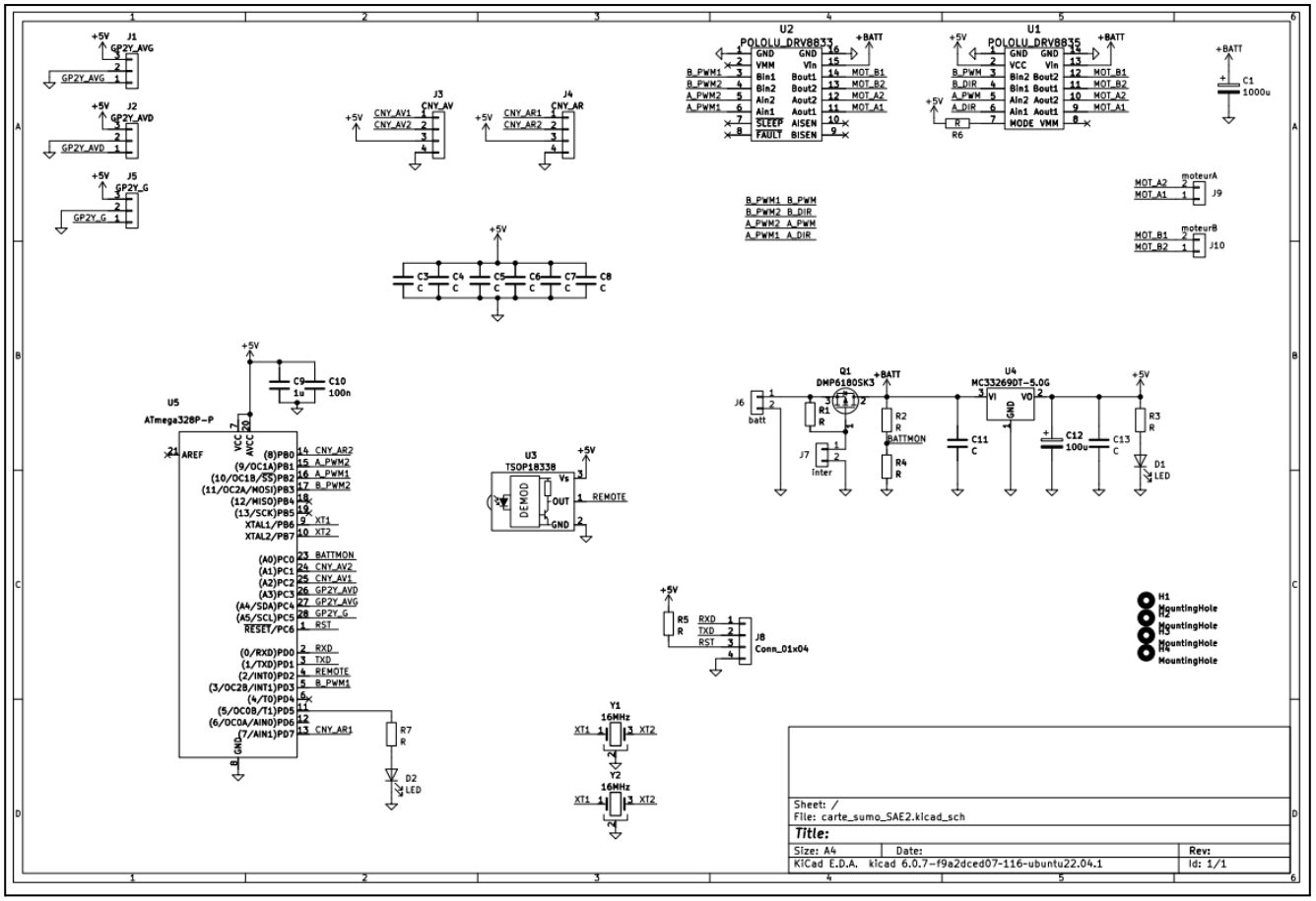
Chaque bloc fonctionnel doit faire l'objet d'un chapitre de conception détaillé, présenté comme suit.

Référence de conception : CDT01

Exigences client vérifiées : *EXIG_AUTONOMIE, EXIG_SECUR_BATT, EXIG_ADVERSAIRE, EXIG_FUITE, EXIG_LUMINOSITE, EXIG_DEPART, EXIG_COTE, EXIG_DEPLACEMENT, EXIG_CARTE, EXIG_COUT*

A la suite de la conception préliminaire, l'activité de conception détaillée a été menée. Le schéma électrique complet de la carte est fourni ci-dessous.

Figure 3.1.1 : schéma électrique de la carte robot mini-sumo



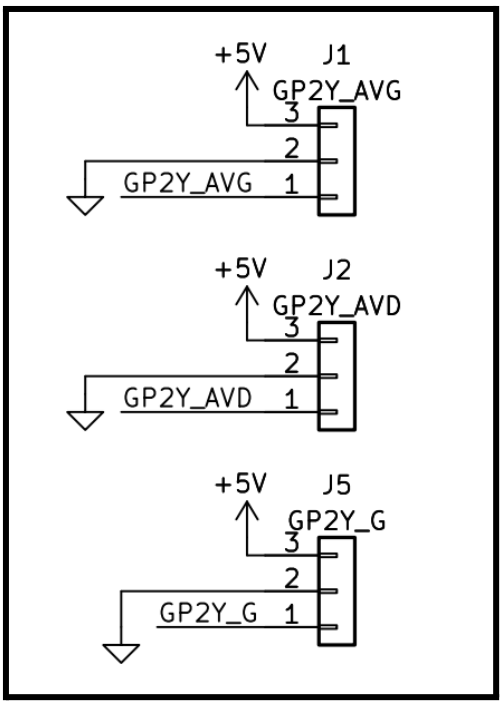
3.2 Conception détaillée de la partie acquisition

3.2.1 Capteurs adversaires

Référence de conception : CDT02

Exigences client vérifiées : *EXIG_ADVERSAIRE, EXIG_FUITE, EXIG_COTE, EXIG_DEPLACEMENT, EXIG_COMBAT, EXIG_LUMINOSITE*

3.2.1.1 Schéma électrique des capteurs adversaires



3.2.1.2 Code informatique des capteurs adversaires

Les capteurs renvoient une tension différente en fonction de la distance de détection. Ainsi on retranscrit ceci dans le code : analogRead permet de prendre l'information des capteurs, ce qui permet de déterminer le mouvement que doit effectuer le robot. Les différentes actions que peut effectuer le robot sont codées dans des switch case, si rien n'est détecté alors le robot tournera sur lui-même vers la droite.

```
#define GP2Y_AVG A4  //  Pin capteur avant gauche
#define GP2Y_AVD A3  //  Pin capteur avant droit
#define GP2Y_G A5  //  Pin capteur gauche
```

```
int val_GP2Y_AVG = 0;  //  Capteur avant gauche
int val_GP2Y_AVD = 0;  //  Capteur avant droit
int val_GP2Y_G = 0;  //  Capteur gauche
```

```
int var;
val_GP2Y_AVG = analogRead(GP2Y_AVG);//  Capteur avant gauche
val_GP2Y_AVD = analogRead(GP2Y_AVD);//  Capteur avant droit
val_GP2Y_G = analogRead(GP2Y_G); //  Capteur gauche
```

```
 val_GP2Y_AVG = map(val_GP2Y_AVG, 0, 1023, 0, 255) ;//  Capteur avant gauche
 val_GP2Y_AVD = map(val_GP2Y_AVD, 0, 1023, 0, 255) ;//  Capteur avant droit
 val_GP2Y_G = map(val_GP2Y_G, 0, 1023, 0, 255) ;//  Capteur gauche

 //******  Ennemi détecté à gauche ******//
 if (val_GP2Y_G > 10 && val_GP2Y_AVG < 10 && val_GP2Y_AVD < 10 ) {
   var = 2;
 }

 //******  Tourner sur lui meme pour trouver l'ennemi  ******//
 if (val_GP2Y_G < 10 && val_GP2Y_AVG < 10 && val_GP2Y_AVD < 10 ) {
   var = 1;
 }

 //******  Avancer vers l'ennemi  ******//
 if (val_GP2Y_AVG > 10 && val_GP2Y_AVD > 10 && val_GP2Y_G < 10 ) {
   var = 3;
 }
```

Ce code permet de détecter si le robot adverse est détecté à gauche, en face ou s'il n'est pas détecté le robot tourne sur lui-même pour trouver l'ennemi.

```
switch (var) {
  case 1: //Tourner gauche
    digitalWrite(MOT_DROIT_xPHASE, LOW); //xPHASE = BIN1 = Direction
    analogWrite(MOT_DROIT_xENABLE, 100); //xENABLE = BIN2 = Vitesse
    analogWrite(MOT_GAUCHE_xPHASE, 100); //xENABLE = BIN1 = Vitesse
    digitalWrite(MOT_GAUCHE_xENABLE, HIGH); //xPHASE = BIN2 = Direction
    break;
  case 2: //Tourner droite
    digitalWrite(MOT_DROIT_xPHASE, HIGH); //xPHASE = BIN1
    analogWrite(MOT_DROIT_xENABLE, 100); //xENABLE = BIN2
    analogWrite(MOT_GAUCHE_xPHASE, 100); //xENABLE = BIN1
    digitalWrite(MOT_GAUCHE_xENABLE, LOW); //xPHASE = BIN2
    break;
  case 3: //Avancer
    digitalWrite(MOT_DROIT_xPHASE, LOW); //xPHASE = Direction
    analogWrite(MOT_DROIT_xENABLE, 250); //xENABLE = Vitesse
    analogWrite(MOT_GAUCHE_xPHASE, 250); //xENABLE = Vitesse
    digitalWrite(MOT_GAUCHE_xENABLE, LOW); //xPHASE = Direction
    break;
```

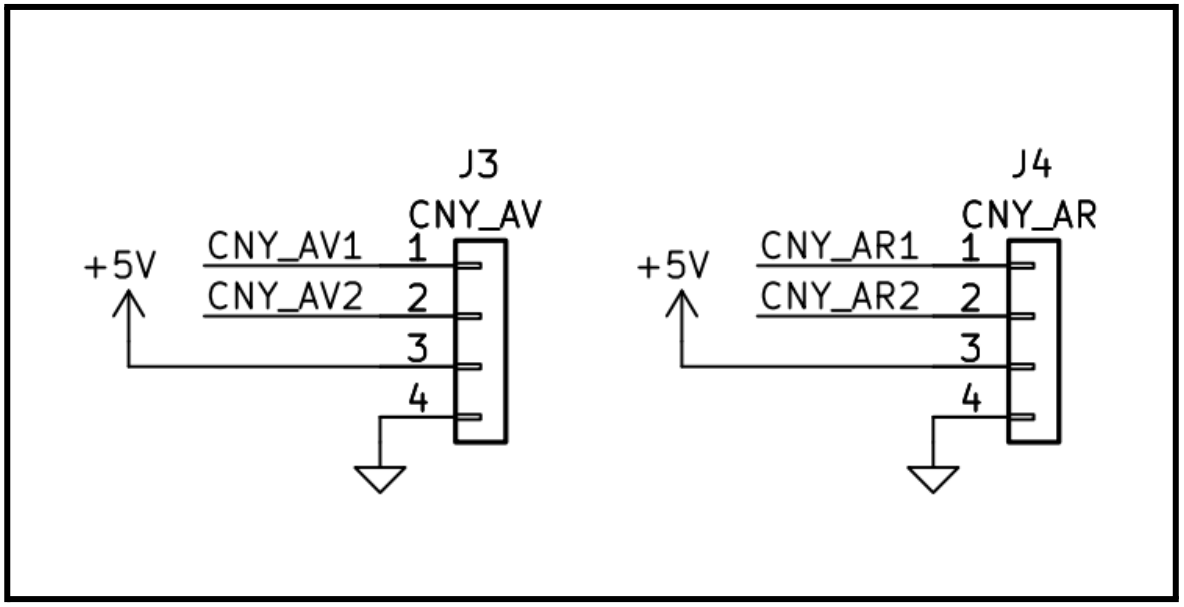
Ensuite on effectue des actions sur notre robot en fonction de l'acquisition de données de ses capteurs.

3.2.2 Capteurs de ligne

Référence de conception : CDT03

Exigences client vérifiées : *Sans objet (Voir le document PDF Règlement Robot Mini Sumo)*

3.2.2.1 Schéma électrique des capteurs de ligne



3.2.2.2 Code informatique des capteurs de sol

Si le capteur CNY_AV détecte la ligne blanche du dohyo, le robot doit reculer afin de ne pas sortir de celui-ci.
Si le capteur CNY_AR détecte la ligne blanche du dohyo, le robot doit avancer afin de ne pas sortir de celui-ci.

Les capteurs sont composés d'un émetteur et d'un récepteur.
L'émetteur est une diode qui doit être associée à une résistance :
Courant max diode : 20mA.
Tension 5V.

$$R = \frac{U}{I} = \frac{5}{20*10^{-3}} = 250\Omega$$

Nous prendrons une résistance de 220Ω.
Le récepteur doit être associé à une résistance de 50 kΩ.

```
#define CNY_AV A2 //  Pin capteur de ligne avant
#define CNY_AR 7 //  Pin capteur de ligne arrière
```

```
int val_CNY_AV = 0; //  Capteur de ligne avant
int val_CNY_AR = 0; //  Capteur de ligne arrière
```

```
val_CNY_AV = digitalRead(CNY_AV);//  Capteur de ligne avant
val_CNY_AR = digitalRead(CNY_AR);//  Capteur de ligne arrière:
```

```
//******  Reculer ******//
if (val_CNY_AV == 0) {
  var = 4;
}

//******  Avancer ******//
if (val_CNY_AR == 0) { //Avancer vers l'ennemi
  var = 3;
}
```

```
case 3: //Avancer
  digitalWrite(MOT_DROIT_xPHASE, LOW); //xPHASE = Direction
  analogWrite(MOT_DROIT_xENABLE, 250); //xENABLE = Vitesse
  analogWrite(MOT_GAUCHE_xPHASE, 250); //xENABLE = Vitesse
  digitalWrite(MOT_GAUCHE_xENABLE, LOW); //xPHASE = Direction
  break;
case 4 ://Reculer
  digitalWrite(MOT_DROIT_xPHASE, HIGH); //xPHASE = Direction
  analogWrite(MOT_DROIT_xENABLE, 250); //xENABLE = Vitesse
  analogWrite(MOT_GAUCHE_xPHASE, 250); //xENABLE = Vitesse
  digitalWrite(MOT_GAUCHE_xENABLE, HIGH); //xPHASE = Direction
  break;
```

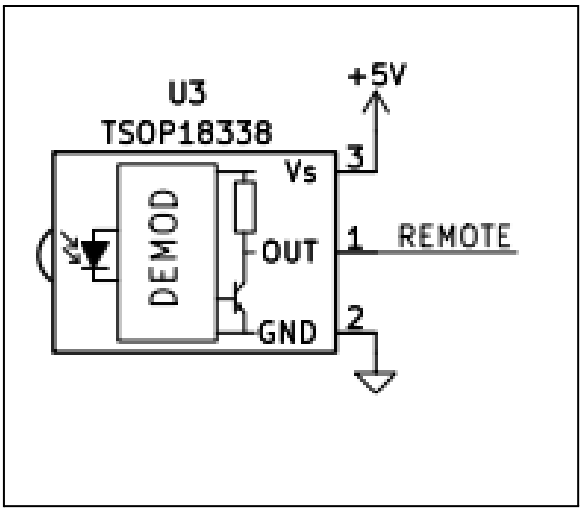
Tout d'abord on déclare les broches du microcontrôleur connectées au capteur de ligne et les variables qui vont permettrent de stocker les valeurs renvoyées par les capteurs.
Ensuite on lit l'état numérique (noir ou blanc) des capteurs de ligne.
Si la valeur d'un capteur vaut 0 cela veut dire qu'il capte du blanc et si elle vaut 1, il capte du noir.
Donc si val_CNY_AV==0 alors on met var à 4 et on exécute ensuite dans le switch case l'action de reculer.
Si val_CNY_AR==0 alors on met var à 3 et on exécute ensuite dans le switch case l'action de d'avancer.

3.2.3 Capteurs de télécommande

Référence de conception : CDT04

Exigences client vérifiées : *EXIG_DEPART, EXIG_COMBAT*

3.2.3.1 Schéma électrique des capteurs de télécommande



3.2.3.2 Code informatique des capteurs de télécommande

```
#define BATTMON A0 //  Pin pont diviseur de tension
```

```
#define REMOTE 2  // Broche du récepteur IR
```

```
IRrecv receiver(REMOTE);  // Création d'un objet receiver de la classe IRrecv
decode_results results;  //  Création d'un objet results de la classe decode_results
```

```
int val_BATTMON = 0; //  Pont diviseur de tension
```

```
int info;           // Variable pour stocker les infos IR reçues
int Start = 0;      // Variable utile pour démmarrer le robot
```

```
void setup() {
  Serial.begin(115200);

  receiver.enableIRIn(); // Active  le processus de réception
  receiver.blink13(true); // Permet le clignotement la LED sur la carte quand un signal IR est reçu
}
```

```
void loop() {
  val_BATTMON = analogRead(BATTMON);//  Pont diviseur de tension

  if (receiver.decode(&results)) {       // Renvoi 1 si on reçoit un signal IR et stock la valeur du signal dans la variable results
    if (results.value != 4294967295) {   // 4294967295 -> Signal reçu si on reste appuyé trop longtemps sur la télécommande
      info = results.value;              // On stocke le code IR dans la var info
    }
    receiver.resume(); // Réinitialise le récepteur IR pour le prochain code
  }

Serial.println(info);
  if (info == -22441 && val_BATTMON > seuil) {
    Start = 1;
    info = 0;
  }

  if (Start == 1) {
    func();
  }

  if (info == 10421 || val_BATTMON < seuil){ //Si bat < seuil stop
    Start = 0;
    digitalWrite(MOT_DROIT_xPHASE, LOW); //xPHASE = Direction
    analogWrite(MOT_DROIT_xENABLE, 0); //xENABLE = Vitesse
    analogWrite(MOT_GAUCHE_xPHASE, 0); //xENABLE = Vitesse
    digitalWrite(MOT_GAUCHE_xENABLE, LOW); //xPHASE = Direction

  }
}
```

Tout d'abord on déclare les constantes de la broche du pont diviseur de tension et du récepteur infrarouge.
Ensuite on crée deux objets respectivement, l'objet receiver de la classe IRrecv et un objet results de la classe decode_results.
Mais également des variables, val_BATTMON qui permet de lire la valeur fournie par l'ADC en sortie du pont diviseur de tension et donc de voir si la tension de la batterie est inférieure au seuil de 6,7 V où dans ce cas on coupe les moteurs.
La variable info permet de stocker les informations IR reçues et la variable start est utilisée pour démarrer le robot.

Les deux lignes :
receiver.enableIRIn();
receiver.blink13(true);

Permettent d'activer le processus de réception de signaux infrarouges et le clignotement de la LED sur la carte quand le récepteur reçoit un signal infrarouge.

Ensuite la ligne receiver.decode(&results) renvoie 1 si on reçoit un signal infrarouge et stocke la

valeur du signal dans la variable results, après nous avons un autre if() dans lequel on teste si le signal infrarouge reçu est différent de 4294967295 qui correspond à la valeur qui est envoyé par la télécommande et reçu par le capteur si on reste appuyé trop longtemps sur un bouton de la télécommande.
Après on a donc une valeur stockée dans la variable info et la ligne receiver.resume(); réinitialise le récepteur infrarouge pour le prochain signal infrarouge que l'on va recevoir.

Ensuite si la tension de la batterie est supérieur au seuil et que info vaut -22441 alors on démarre le robot et remet la variable info à 0.
Et si start vaut 1 on appelle la fonction principale qui contrôle le robot en fonction des données des capteurs.

Enfin si info vaut 10421 ou que val_BATTMON < seuil alors on met la variable start à 0 pour arrêter d'appeler la fonction principale de contrôle du robot et on coupe instantanément les moteurs.

3.3 Conception détaillée de la partie action

3.3.1 Pont en H

Référence de conception : CDT05

Exigences client vérifiées : *EXIG_DEPLACEMENT*

3.3.1.1 Schéma électrique du pont en H

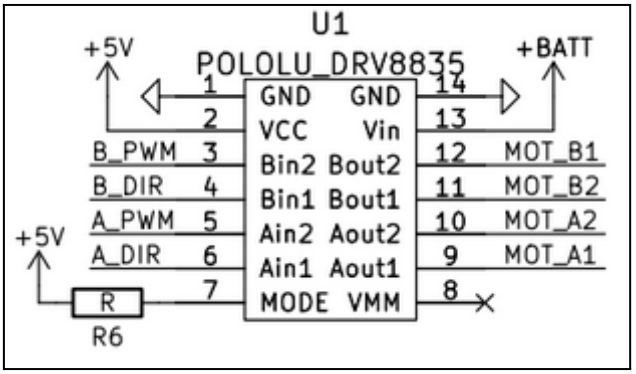


Figure 3.3.1.1 : Schéma électrique pont en H.

Les moteurs sont connectés aux ports “MOT_XX” du pont en H, le port VCC est connecté au 5V du microcontrôleur. Les ports “X_PWM” et “X_DIR” sont connectés aux ports “digital” de l’arduino.
Fonctionnement du pont en H :
Le pont en H est configuré en mode 1” (broche 7 reliée au 5V avec une résistance R6=10kΩ selon la datasheet pour le mode 1).

Table 3. PHASE/ENABLE MODE

MODE	xENABLE	xPHASE	xOUT1	xOUT2	FUNCTION (DC MOTOR)
1	0	X	L	L	Brake
1	1	1	L	H	Reverse
1	1	0	H	L	Forward

Figure 3.3.1.2 : Fonctionnement du pont en H

xENABLE permet de contrôler la vitesse de rotation d’un moteur.

3.3.1.2 Code informatique du pont en H

On définit les broches reliées aux deux moteurs :

```
14 //Moteur droit
15 #define MOT_DROIT_xPHASE 11
16 #define MOT_DROIT_xENABLE 10
17
18 //Moteur gauche
19 #define MOT_GAUCHE_xPHASE 9
20 #define MOT_GAUCHE_xENABLE 8
```

On se réfère au tableau pour connaître l'état des broches pour faire tourner à gauche l'adversaire. Dans ce code, le robot tourne à gauche avec une vitesse de 100 sur 255 (255 = vitesse max).

Code pour faire avancer le robot (si l'adversaire se trouve en face ou si le capteur CNY_AR capte que le robot se rapproche du rebord arrière du dohyo) :

```
case 3: //Avancer
  digitalWrite(MOT_DROIT_xPHASE, LOW); //xPHASE = Direction
  analogWrite(MOT_DROIT_xENABLE, 250); //xENABLE = Vitesse
  analogWrite(MOT_GAUCHE_xPHASE, 250); //xENABLE = Vitesse
  digitalWrite(MOT_GAUCHE_xENABLE, LOW); //xPHASE = Direction
  break;
```

Le capteur avant gauche détecte, le robot tourne à gauche :

```
case 1: //Tourner gauche
  digitalWrite(MOT_DROIT_xPHASE, LOW); //xPHASE = BIN1 = Direction
  analogWrite(MOT_DROIT_xENABLE, 100); //xENABLE = BIN2 = Vitesse
  analogWrite(MOT_GAUCHE_xPHASE, 100); //xENABLE = BIN1 = Vitesse
  digitalWrite(MOT_GAUCHE_xENABLE, HIGH); //xPHASE = BIN2 = Direction
  break;
```

Aucun capteur ne détecte d'ennemi, le robot tourne sur lui même :

```
case 2: //Tourner droite
  digitalWrite(MOT_DROIT_xPHASE, HIGH); //xPHASE = BIN1
  analogWrite(MOT_DROIT_xENABLE, 100); //xENABLE = BIN2
  analogWrite(MOT_GAUCHE_xPHASE, 100); //xENABLE = BIN1
  digitalWrite(MOT_GAUCHE_xENABLE, LOW); //xPHASE = BIN2
  break;
```

Reculer si notre capteur CNY AV se rend compte que l'on sort du dohyo

```
case 4 ://Reculer dans la cas où on sort du dohyo
  digitalWrite(MOT_DROIT_xPHASE, HIGH); //xPHASE = Direction
  analogWrite(MOT_DROIT_xENABLE, 250); //xENABLE = Vitesse
  analogWrite(MOT_GAUCHE_xPHASE, 250); //xENABLE = Vitesse
  digitalWrite(MOT_GAUCHE_xENABLE, HIGH); //xPHASE = Direction
  break;
```

Arrêt des deux moteurs

```
case 5 ://STOP
  digitalWrite(MOT_DROIT_xPHASE, LOW); //xPHASE = Direction
  analogWrite(MOT_DROIT_xENABLE, 0); //xENABLE = Vitesse
  analogWrite(MOT_GAUCHE_xPHASE, 0); //xENABLE = Vitesse
  digitalWrite(MOT_GAUCHE_xENABLE, LOW); //xPHASE = Direction
  break;
```

3.4 Conception détaillée de la partie énergie

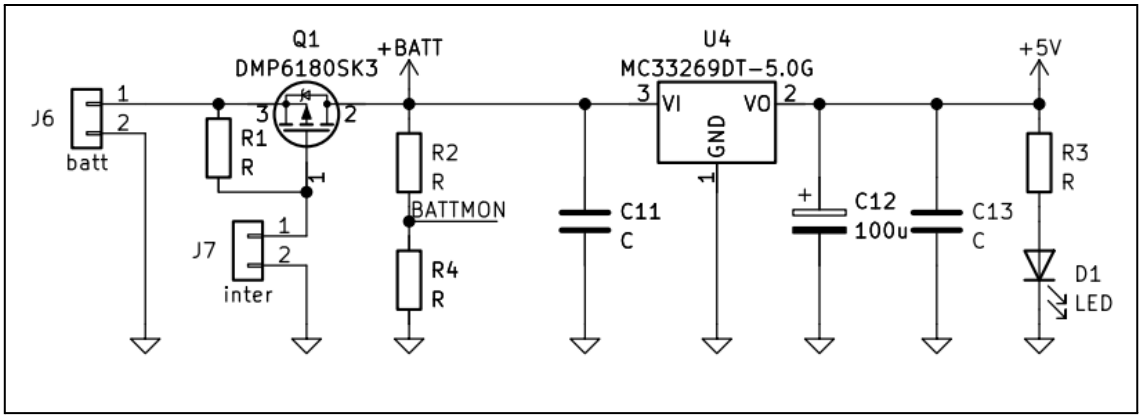


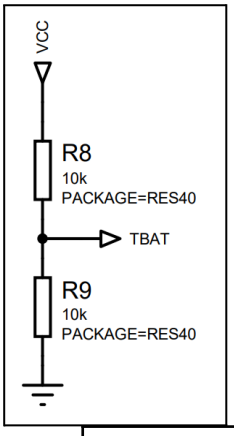
Figure 3.4.1 : Schéma électrique Bloc énergie.

3.4.1 Pont diviseur de tension

Référence de conception : CDT06

Exigences client vérifiées : *EXIG_SECUR_BATT*

3.4.1.1 Schéma électrique du pont diviseur de tension



Le pont diviseur de tension ci-dessus permet de monitorer la tension de la batterie, c'est-à-dire couper la partie puissance (moteurs) quand la tension aux bornes de celle-ci est inférieure à 6,7V.

Nous avons choisis deux résistances de 10kΩ afin de diviser la tension de la batterie par 2 ($\frac{R9}{R8+R9} = \frac{10k}{10k+10k} = 0,5$), car l'ADC de l'ATMEGA328P ne peut supporter que 5v maximum en entrée.

Avec ce pont diviseur, nous avons une tension de sortie au niveau de la broche *TBAT* qui se situe entre 0v et $\frac{V_{cc}}{2}$, soit 4,2v. Au moment de la programmation, l'ADC convertit

cette tension analogique en un entier (que nous appelons *val_BATTMON* dans le code) codé sur 10 bits, 1023 en décimal, cette valeur est atteinte pour *TBAT* = 5v, donc pour notre valeur de tension, nous aurons *val_BATTMONmax* = 860 au maximum.

L'exigence *EXIG_SECUR_BATT* nous impose de couper les moteurs quand la tension de la batterie passe en dessous des 6,7v, ce qui correspond à un niveau de charge d'environ 5%. Nous devons donc calculer la valeur de seuil de *val_BATTMON* afin que la tension ne descende pas en dessous de la limite des 5% imposée par le cahier des charges. On a donc : $val\_BATTMON_{seuil} = \frac{\frac{6,7}{2} . 2^{10}}{5,0} = 685,41.$ Nous devons donc couper les moteurs quand *val_BATTMON* est inférieur à 685.

3.4.1.2 Code informatique du pont diviseur de tension

Nous avons donc créé le code ci-dessous afin de récupérer la valeur de l'ADC en sortie du pont diviseur et arrêter les moteurs quand cette valeur est inférieure à la valeur de seuil.

```
5 #define BATTMON A0 //  Pin pont diviseur de tension
```

Cette première ligne permet définir une variable appelée *BATTMON* et de lui attribuer la broche A0 du microcontrôleur.

```
6 #define seuil 685 //  Seuil de coupure des moteurs
```

Puis, on définit une variable seuil, qui prend la valeur du seuil minimum que nous avons calculé précédemment.

```
26 int val_BATTMON = 0; //  Pont diviseur de tension
```

On déclare un entier nommé *val_BATTMON*, qui correspond à la variable de l'ADC codée sur 10 bits, on initialise la variable à 0 pour éviter que le robot démarre s'il n'arrive pas à lire la valeur de l'ADC.

```
43   val_BATTMON = analogRead(BATTMON);//  Pont diviseur de tension
```

Ensuite, on lit la valeur que renvoie l'ADC avec la fonction *analogRead();* et on l'attribue à la variable *val_BATTMON* afin de la stocker en mémoire.

```
61  if (info == 10421 || val_BATTMON < seuil){ //Si bat < seuil stop
62    Start = 0;
63    digitalWrite(MOT_DROIT_xPHASE, LOW); //xPHASE = Direction
64    analogWrite(MOT_DROIT_xENABLE, 0); //xENABLE = Vitesse
65    analogWrite(MOT_GAUCHE_xPHASE, 0); //xENABLE = Vitesse
66    digitalWrite(MOT_GAUCHE_xENABLE, LOW); //xPHASE = Direction
67
68  }
```

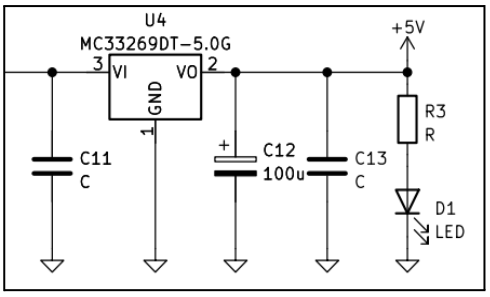
Puis on teste si notre variable *val_BATTMON* est inférieur à la valeur de seuil à l'aide d'un *if()*, et si c'est le cas on entre dans le if, qui consiste à sortir du mode de fonctionnement normal et arrêter les moteurs. L'intérieur de cette fonction sera détaillée plus bas dans la partie action du programme.

3.4.2 régulateur de tension

Référence de conception : CDT07

Exigences client vérifiées : EXIG_AUTONOMIE, EXIG_SECUR_BATT

3.4.2.1 Schéma électrique du régulateur de tension



Le régulateur de tension de 5V se justifie par le fait qu'on utilise un microcontrôleur ATmega328P-P qui fonctionne avec une alimentation en 5V, ainsi que tous les capteurs qu'on utilise et le pont en H.
Une Diode électroluminescente indique l'alimentation en 5V du microcontrôleur. La datasheet de la diode indique qu'elle doit être alimenter en 20mA, la résistance R de la diode est donc de 250Ω.

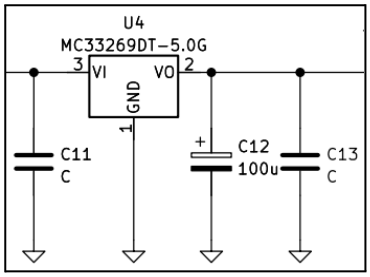
Figure 3.4.2.1 : Régulateur de tension.

3.4.3 Condensateur de découplage

Référence de conception : CDT08

Exigences client vérifiées : EXIG_AUTONOMIE, EXIG_SECUR_BATT

3.4.3.1 Schéma électrique condensateur de découplage :



En se référant au datasheet du régulateur de tension nous avons dimensionné les condensateurs :
C11 = 0.33uF et C13 = 10uF.

Figure 3.4.3.1 : Condensateurs du régulateur de tension.

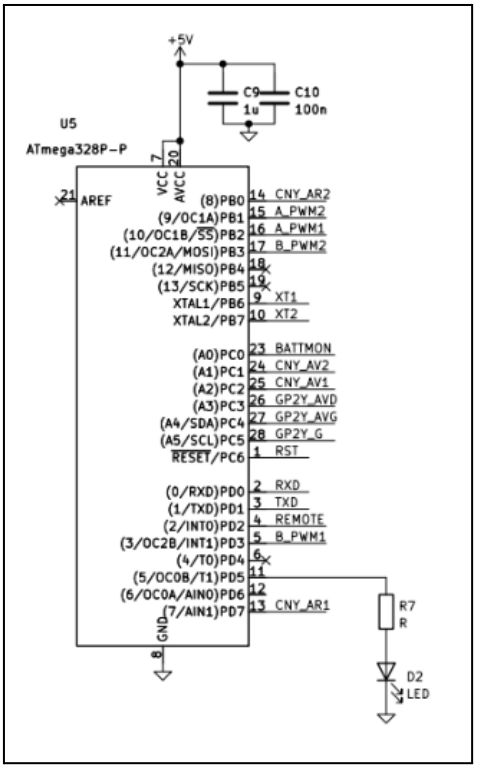
3.5 Conception détaillée de la partie traitement

3.5.1 Le microcontroleur ATMEGA328P-PU

Référence de conception : CDT09

Exigences client vérifiées : EXIG_AUTONOMIE, EXIG_SECUR_BATT

3.5.1.1 Schéma électrique du microcontroleur



Voici sur le schéma ci-dessus le schéma électrique complet du microcontrôleur avec ses branchements associés.

3.5.1.2 Code informatique du microcontroleur

Code_FINAL_SUMO_v5

```
#include "IRremote.h"
#include <arduino.h>

//Pin capteurs
#define BATTMON A0 //  Pin pont diviseur de tension
#define seuil 685 //  Seuil de coupure des moteurs
#define GP2Y_AVG A4  //  Pin capteur avant gauche
#define GP2Y_AVD A3  //  Pin capteur avant droit
#define GP2Y_G A5  //  Pin capteur gauche
#define CNY_AV A2 //  Pin capteur de ligne avant
#define CNY_AR 7 //  Pin capteur de ligne arrière
#define REMOTE 2  // Broche du récepteur IR

//Moteur droit
#define MOT_DROIT_xPHASE 11
#define MOT_DROIT_xENABLE 10

//Moteur gauche
#define MOT_GAUCHE_xPHASE 9
#define MOT_GAUCHE_xENABLE 8

IRrecv receiver(REMOTE); // create a receiver object of the IRrecv class
decode_results results; // create a results object of the decode_results class

//Variables capteurs
int val_BATTMON = 0; //  Pont diviseur de tension
int val_GP2Y_AVG = 0;  //  Capteur avant gauche
int val_GP2Y_AVD = 0;  //  Capteur avant droit
int val_GP2Y_G = 0;  //  Capteur gauche
int val_CNY_AV = 0; //  Capteur de ligne avant
int val_CNY_AR = 0; //  Capteur de ligne arrière
int info;
int Start = 0;

void setup() {
  Serial.begin(115200);

  receiver.enableIRIn(); // enable the receiver
  receiver.blink13(true); // enable blinking of the built-in LED when an IR signal is received
}
```

Nous commençons notre code en incluant les bibliothèques et en définissant nos pins par rapport à l'arduino, ce qui facilitera l'écriture et la lecture du code par la suite.
Ensuite dans le Void setup, on place un serial.begin qui nous servira lors du débogage et de la maintenance du produit afin d'avoir accès aux valeurs des capteurs.

```
41
42 void loop() {
43   val_BATTMON = analogRead(BATTMON);//  Pont diviseur de tension
44
45   if (receiver.decode(&results)) { // decode the received signal and store it in results
46     if (results.value != 4294967295) {
47       info = results.value;
48     }
49     receiver.resume(); // reset the receiver for the next code
50   }
51
52   if (info == -22441 && val_BATTMON > seuil) {
53     Start = 1;
54     info = 0;
55   }
56
57   if (Start == 1) {
58     func();
59   }
60
61   if (info == -15811 || val_BATTMON < seuil) { //Si bat < seuil stop
62     Start = 0;
63     digitalWrite(MOT_DROIT_xPHASE, LOW); //xPHASE = Direction
64     analogWrite(MOT_DROIT_xENABLE, 0); //xENABLE = Vitesse
65     analogWrite(MOT_GAUCHE_xPHASE, 0); //xENABLE = Vitesse
66     digitalWrite(MOT_GAUCHE_xENABLE, LOW); //xPHASE = Direction
67
68   }
69 }
70
71 void func() {
72
73   Serial.println(val_GP2Y_G);
74
75   int var;
76   val_GP2Y_AVG = analogRead(GP2Y_AVG);//  Capteur avant gauche
77   val_GP2Y_AVD = analogRead(GP2Y_AVD);//  Capteur avant droit
78   val_GP2Y_G = analogRead(GP2Y_G); //  Capteur gauche
79   val_CNY_AV = digitalRead(CNY_AV);//  Capteur de ligne avant
80   val_CNY_AR = digitalRead(CNY_AR);//  Capteur de ligne arrière:
```

```
81
82  val_GP2Y_AVG = map(val_GP2Y_AVG, 0, 1023, 0, 255) ;//  Capteur avant gauche
83  val_GP2Y_AVD = map(val_GP2Y_AVD, 0, 1023, 0, 255) ;//  Capteur avant droit
84  val_GP2Y_G = map(val_GP2Y_G, 0, 1023, 0, 255) ;//  Capteur gauche
85
86
87
88  //******  Ennemi détecté à gauche ******//
89  if (val_GP2Y_G > 10 && val_GP2Y_AVG < 10 && val_GP2Y_AVD < 10 ) {
90    var = 2;
91  }
92
93  //******  Tourner sur lui meme pour trouver l'ennemi  ******//
94  if (val_GP2Y_G < 10 && val_GP2Y_AVG < 10 && val_GP2Y_AVD < 10 ) {
95    var = 1;
96  }
97
98  //******  Avancer vers l'ennemi  ******//
99  if (val_GP2Y_AVG > 10 && val_GP2Y_AVD > 10 && val_GP2Y_G < 10 ) {
100   var = 3;
101 }
102
103 //******  Reculer ******//
104 if (val_CNY_AV == 0) {
105   var = 4;
106 }
107
108 //******  Avancer ******//
109 if (val_CNY_AR == 0) { //Avancer vers l'ennemi
110   var = 3;
111 }
112
113 switch (var) {
114   case 1: //Tourner gauche
115     digitalWrite(MOT_DROIT_xPHASE, LOW); //xPHASE = BIN1 = Direction
116     analogWrite(MOT_DROIT_xENABLE, 100); //xENABLE = BIN2 = Vitesse
117     analogWrite(MOT_GAUCHE_xPHASE, 100); //xENABLE = BIN1 = Vitesse
118     digitalWrite(MOT_GAUCHE_xENABLE, HIGH); //xPHASE = BIN2 = Direction
119     break;
120   case 2: //Tourner droite
```

```
121     digitalWrite(MOT_DROIT_xPHASE, HIGH); //xPHASE = BIN1
122     analogWrite(MOT_DROIT_xENABLE, 100); //xENABLE = BIN2
123     analogWrite(MOT_GAUCHE_xPHASE, 100); //xENABLE = BIN1
124     digitalWrite(MOT_GAUCHE_xENABLE, LOW); //xPHASE = BIN2
125     break;
126   case 3: //Avancer
127     digitalWrite(MOT_DROIT_xPHASE, LOW); //xPHASE = Direction
128     analogWrite(MOT_DROIT_xENABLE, 250); //xENABLE = Vitesse
129     analogWrite(MOT_GAUCHE_xPHASE, 250); //xENABLE = Vitesse
130     digitalWrite(MOT_GAUCHE_xENABLE, LOW); //xPHASE = Direction
131     break;
132   case 4 ://Reculer
133     digitalWrite(MOT_DROIT_xPHASE, HIGH); //xPHASE = Direction
134     analogWrite(MOT_DROIT_xENABLE, 250); //xENABLE = Vitesse
135     analogWrite(MOT_GAUCHE_xPHASE, 250); //xENABLE = Vitesse
136     digitalWrite(MOT_GAUCHE_xENABLE, HIGH); //xPHASE = Direction
137     break;
138     /*    case 5 ://STOP
139           digitalWrite(MOT_DROIT_xPHASE, LOW); //xPHASE = Direction
140           analogWrite(MOT_DROIT_xENABLE, 0); //xENABLE = Vitesse
141           analogWrite(MOT_GAUCHE_xPHASE, 0); //xENABLE = Vitesse
142           digitalWrite(MOT_GAUCHE_xENABLE, LOW); //xPHASE = Direction
143           break;
144     */
145 }
146 return ;
147 }
```

Ici nous avons les différentes possibilités de notre switch, les différentes pin correspondent à notre pont en H. Si on met le xPHASE en low, cela fera avancer le robot et si on le met en HIGH cela le

fera reculer. Les vitesses sont définies entre 0 et 255 et donc pour les moments où le robot est à la recherche de l'adversaire, nous avons choisi une vitesse moindre afin d'être sûr que les capteurs n'auront aucun mal à repérer l'adversaire.

4. Dérisquage des solutions techniques retenues

Ce chapitre détaille les activités de dérisquage des solutions techniques retenues : simulation et/ou prototypage rapide. Il constitue une preuve partielle de la conformité du produit. Chaque paragraphe de l'étude fait donc clairement référence aux exigences client issues du [CDC].

Il permet également de confirmer les résultats théoriques effectués aux paragraphes 2 et 3 en vérifiant le fonctionnement à travers des simulations et/ou prototypages rapides. Pour chaque simulation et/ou prototypage rapide est renseigné le protocole de mise en œuvre. Les résultats des simulations et/ou prototypages rapides sont confrontés aux résultats de l'étude théorique.

L'ensemble des fichiers est disponible dans le dossier : renseignez ici le chemin du dossier où sont situés les fichiers de simulation et/ou prototypage rapide du projet.

4.1 Dérisquage Acquisition :

Référence de la simulation : SIM01

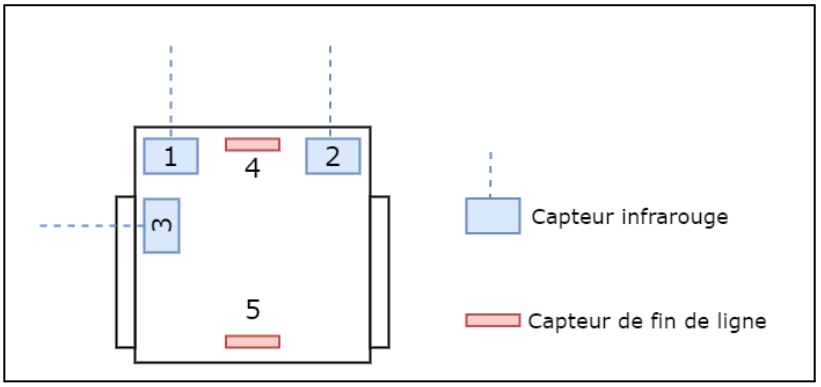
Exigences client vérifiées : EXIG_ADVERSAIRE

But de l'essai : Le but est de vérifier que le robot capte bel et bien un adversaire à n'importe quel endroit du dohyo.

Fichiers : Aucun

Procédure de simulation :

Les capteurs sont fixés sur le robots aux endroits qui leurs ont été attribués.



Résultats attendus :

Les capteurs GP2Y renvoient des valeurs en fonction de la distance d'approche d'un objet (adversaire). Plus l'adversaire est éloignée, plus la valeur se rapproche de 0 (elle varie entre 0 et 10, dû à des perturbations).

Grandeur	Valeur attendue
Valeur capteur AVG	>10
Valeur capteur AVD	>10
Valeur capteur G	>10

Résultats obtenus :

Nous avons placé un seuil de 10 (sur 255) dans le programme car en dessous de ce seuil, les valeurs relevées varient énormément. Les valeurs relevées sont toutefois conformes à celles attendues.

```
//******  Ennemi détecté à gauche ******//
  if (val_GP2Y_G > 10 && val_GP2Y_AVG < 10 && val_GP2Y_AVD < 10 ) {
    var = 2;
  }

//******  Tourner sur lui meme pour trouver l'ennemi  ******//
  if (val_GP2Y_G < 10 && val_GP2Y_AVG < 10 && val_GP2Y_AVD < 10 ) {
    var = 1;
  }

//******  Avancer vers l'ennemi  ******//
  if (val_GP2Y_AVG > 10 && val_GP2Y_AVD > 10 && val_GP2Y_G < 10 ) {
    var = 3;
  }
```

```
GP2Y_AVG = 598
GP2Y_AVD = 600
GP2Y_G = 601
```

```
GP2Y_AVG = 3
GP2Y_AVD = 3
GP2Y_G = 2
```

Grandeur	Valeur mesurée	Conf/Non conf.
Valeur capteur AVG	>10	Conforme
Valeur capteur AVD	>10	Conforme
Valeur capteur G	>10	Conforme

Statut de l'essai : L'essai est concluant, le robot est conforme à l'exigence EXIG_ADVERSAIRE

Problèmes rencontrés : R.A.S

Référence de la simulation : SIM02

Exigences client vérifiées : EXIG_LUMINOSITE

But de l'essai : Cet essai a pour but de vérifier le fonctionnement des capteurs optiques de contraste CNY70 qui vont être utilisés afin de capter si le robot sort du dohyo et d'agir en conséquence.

Fichiers : Aucun

Procédure de simulation : Nous connectons deux capteurs CNY70 avec le V_{CC} au 5v de l'arduino, le GND à la masse, le signal du premier capteur à la broche A2 et le signal du second capteur à la broche 7 de l'arduino qui dispose du même microcontrôleur que le robot final. Puis, nous implantons dans la carte le code en figure 4.1.1 ci-dessous et lisons les valeurs que nous renvoie ce programme via le moniteur série.

Nous pouvons donc lire la valeur des variables renvoyées dans le moniteur série et observer leurs évolutions quand nous changeons la couleur de la surface sous le capteur en le déplacant d'une surface balance à une surface noir.

Déclaration des broches auxquelles sont reliées les CNY70

```
#define CNY_AV A2 //  Pin capteur de ligne avant
#define CNY_AR 7 //  Pin capteur de ligne arrière
```

figure 4.1.1 (a)

Déclaration des deux variables qui vont servir à stocker la valeur de chaque capteur indépendamment

```
int val_CNY_AV = 0; //  Capteur de ligne avant
int val_CNY_AR = 0; //  Capteur de ligne arrière
```

figure 4.1.1 (b)

Lecture de la valeur digitale sur la broche des capteurs et attribution de cette valeur à la variable pour la retenir

```
val_CNY_AV = digitalRead(CNY_AV);
val_CNY_AR = digitalRead(CNY_AR);
```

figure 4.1.1 (c)

Affichage de l'état des variables qui contiennent l'état des capteurs dans le moniteur série

```
Serial.print("CNY_AV = ");
Serial.println(val_CNY_AV);     // On affiche la valeur binaire
Serial.println(" ");           //  On saute une ligne pour l'affichage
Serial.print("CNY_AR = ");
Serial.println(val_CNY_AV);  //    On affiche la valeur binaire
```

figure 4.1.1 (d)

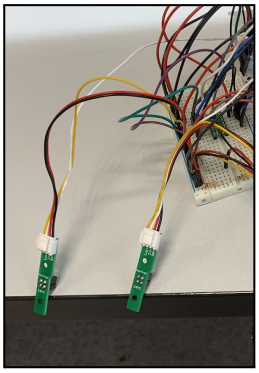
Résultats attendus :

A partir des exigences client issues du Cahier Des Charges et de la datasheet des capteurs, nous obtenons donc le tableau ci-dessous.

Grandeur	Couleur de la surface	Valeur attendue
Binaire	noir	1
Binaire	blanc	0

Résultats obtenus :

Nous obtenons donc les résultats suivants :



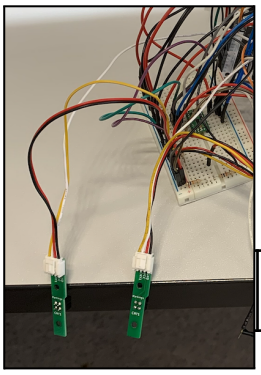
Tout d'abord, nous plaçons les capteurs (en vert) contre une surface blanche et nous lisons la valeur renvoyé par les capteur dans le moniteur série ci-dessous

```
CNY_AV = 0

CNY_AR = 0
```

On peut voir ci-contre que en présence d'une surface clair (ici du blanc) la variable qui prend la valeur de la sortie du capteur veut 0, ce qui correspond a un niveau logique de 0v.

Puis, nous plaçons les capteurs (en vert) contre une surface noire et nous lisons la valeur renvoyé par les capteur dans le moniteur série ci-dessous



```
CNY_AV = 1

CNY_AR = 1
```

On peut voir ci-contre que en présence d'une surface sombre (ici du noir) la variable qui prend la valeur de la sortie

du capteur veut 1, ce qui correspond a un niveau logique de 5v.

Grandeur	Couleur de la surface	Valeur mesurée	Conf/Non conf.
Binaire	noir	1	conforme
Binaire	blanc	0	conforme

Statut de l'essai : Nous pouvons donc conclure que cet essai est conforme à l'exigence EXIG_LUMINOSITE du cahier des charges et aux attentes du aux datasheets du composant

Problèmes rencontrés : Aucun problème n'a été rencontré durant cet essai

Référence de la simulation : SIM03

Exigences client vérifiées : EXIG_DEPART

But de l'essai : Le robot doit rester immobile en attendant le départ. Le robot mini-sumo démarre dès qu'il reçoit l'information d'une télécommande infrarouge.

Fichiers : Aucun

Procédure de simulation : Pour cela, nous plaçons le robot sur un support afin que les roues ne touchent pas le sol et pouvoir observer leurs comportement lorsque nous appuyons sur la télécommande.

Résultats attendus :

A partir des exigences client issues du Cahier Des Charges, nous obtenons donc le tableau ci-dessous.

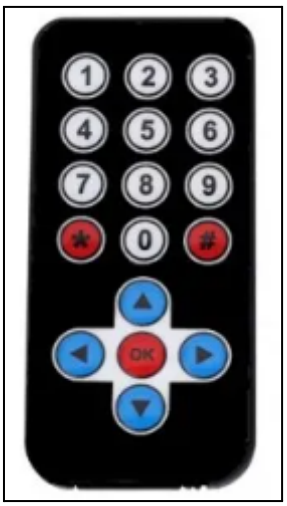
Grandeur	état des boutons de la télécommande	Valeur attendue
Mouvement	allumage du robot	moteurs à l'arrêt
Mouvement	Start = 1 Stop = 0	démarrage des moteurs
Mouvement	Start = 0 Stop = 1	arrêt des moteurs

Résultats obtenus :

Nous obtenons donc les résultats suivants :

Lorsque nous alimentons le robot en 7,4 V qui est la tension nominale de la batterie, les roues du robot ne démarrent pas.

Nous prenons ensuite la télécommande infrarouge et appuyons sur le bouton entrer qui est le boutons que nous avons programmé pour démarrer le robot et constatons que les roues du robot se mettent à tourner et réagissent à la présence du robot adverse comme le montre la figure suivante :



Enfin, nous pressons le bouton stop de la télécommande qui est celui que nous avons programmé pour arrêter les moteurs du robot. Nous remarquons que les moteurs ont cessé de tourner et que le robot ne réagit plus à la présence du robot ennemi.

Nous obtenons donc le tableau récapitulatif suivant :

Grandeur	état des boutons de la télécommande	Valeur attendue
Mouvement	allumage du robot	moteurs à l'arrêt
Mouvement	Start = 1 Stop = 0	démarrage des moteurs
Mouvement	Start = 0 Stop = 1	arrêt des moteurs

Statut de l'essai : Nous pouvons donc conclure que cet essai est conforme au cahier des charges et respecte l'exigence EXIG_DEPART.

Problèmes rencontrés : Aucun problème n'a été rencontré durant cet essai

Référence de la simulation : SIM04

Exigences client vérifiées : EXIG_COTE

But de l'essai : Le robot est capable de déterminer, pendant la phase d'immobilité, si l'adversaire est à gauche ou pas en début de manche.

Fichiers : Aucun

Procédure de simulation : Avant d'appuyer sur la télécommande pour démarrer le robot, nous plaçons notre main sur un des deux côtés du robot et observons son comportement

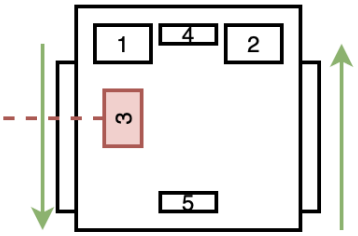
Résultats attendus :

A partir des exigences client issues du Cahier Des Charges et de la datasheet des capteurs, nous obtenons donc le tableau ci-dessous.

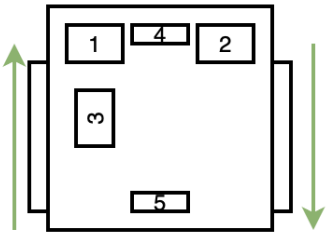
Grandeur	emplacement de l'adversaire	Comportement attendu
Mouvement	côté gauche	rotation à gauche
Mouvement	côté droit	rotation à droite

Résultats obtenus:

Lorsque nous plaçons une main à la gauche du robot afin de simuler la présence de l'adversaire, nous remarquons que les deux roues tournent dans le sens horaire comme le montre le schéma ci-dessous.



Puis nous retirons notre main afin qu'aucun capteur ne détecte la présence du robot adverse, comme il dispose de capteurs devant, et à gauche, l'adversaire est forcément derrière ou à droite. Nous observons le comportement des roues du robot et remarquons qu'elles changent de sens, elles tournent donc dans le sens anti-horaire comme le montre le schéma ci-dessous.



A la suite de ses essais, nous obtenons donc le tableau ci dessous:

Grandeur	emplacement de l'adversaire	Comportement attendu
Mouvement	côté gauche	rotation à gauche
Mouvement	côté droit	rotation à droite

Statut de l'essai : L'essai est concluant, le robot est conforme à l'exigence EXIG_COTE.

Problèmes rencontrés : R.A.S

4.3 Dérisquage Énergie :

Référence de la simulation : SIM05

Exigences client vérifiées : EXIG_AUTONOMIE

But de l'essai : Le but est de vérifier que le courant consommé par l'ensemble du robot ne dépasse pas 540mA, ce courant est le courant maximal que peut fournir notre batterie pour une autonomie de 50 min ($\frac{450mA}{0,833} = 540mA$ avec 50 min = 0,833h).

Fichiers : Aucun

Procédure de simulation :

L'alimentation en condition normale du robot est nécessaire à la mesure de courant.

Pour cela la batterie est simulée par une alimentation continue en 8,4v, qui est la tension de la batterie chargée à 100% le robot avance donc à pleine puissance.

Un ampèremètre est branché en série à l'entrée de l'alimentation comme le montre la figure 4.3.1 ci-dessous.

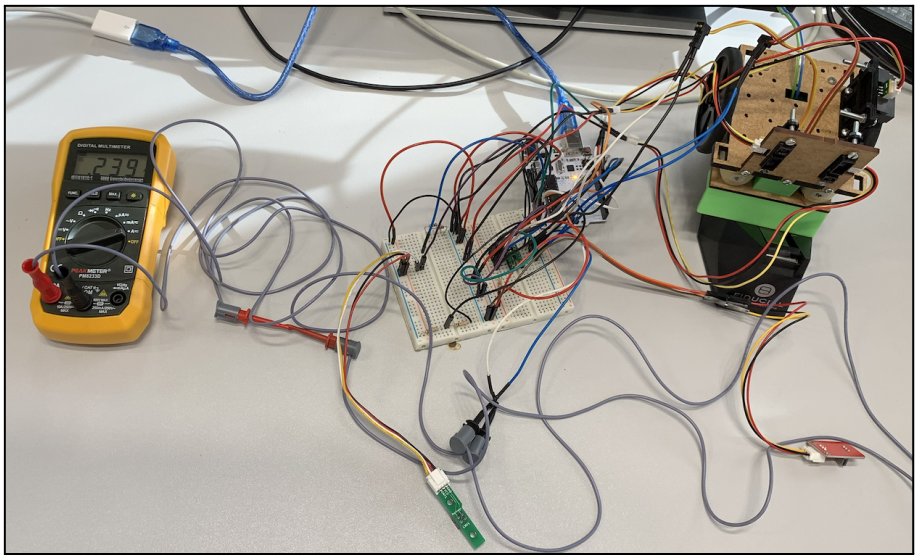


figure 4.3.1 : mesure du courant consommé par le robot

Résultats attendus :

Grandeur	Valeur attendue
Courant mesuré	< 540 mA

Résultats obtenus :

Grandeur	Valeur mesurée	Conf/Non conf.
Courant mesuré	238 mA	Conforme

Statut de l'essai : L'essai est concluant, le robot est conforme à l'exigence EXIG_AUTONOMIE, nous pouvons ajouter qu'avec un tel courant le robot a une autonomie d'environ 1h50 ($\frac{450mAh}{238mA} = 1,89h = 113min = 1h53$).

Problèmes rencontrés : R.A.S

4.4 Dérisquage Action :

Référence de la simulation : SIM07

Exigences client vérifiées : EXIG_COMBAT

But de l'essai : Le robot doit pouvoir engager le combat face à l'adversaire lorsqu' il reçoit le signal infrarouge.

Fichiers : Aucun

Procédure de simulation :

Pour cela, nous plaçons le robot sur un support afin que les roues ne touchent pas le sol et pouvoir observer leur comportement lorsque nous appuyons sur la télécommande. Le robot est alimenté. Nous envoyons un signal au robot via la télécommande infrarouge. Le robot doit effectuer une rotation à gauche puis avancer.

Résultats obtenus :

Grandeur	Etat de la télécommande	Conf/Non conf.
Télécommande	Start = 1	Conforme
Mouvement	Gauche	Conforme
Mouvement	Avant	Conforme

Statut de l'essai : L'essai est concluant, le robot est conforme à l'exigence EXIG_COMBAT.

Problèmes rencontrés : R.A.S

4.5 Dérisquage Méca :

Référence de la simulation : SIM08

Exigences client vérifiées : EXIG_CHASSIS_DIMENSIONS

But de l'essai : Le but de cet essai est de s'assurer que le robot ne dépasse pas 100 mm en largeur et 100 mm en longueur.

Fichiers : Aucun

Procédure de simulation : Afin de vérifier cette exigence, nous utilisons un pied à coulisse afin de mesurer le châssis du robot car il représente les dimensions finales.

Résultats attendus :

A partir de l'exigence client EXIG_CHASSIS_DIMENSIONS issues du cahier des charges, nous pouvons dresser le tableau ci-dessous.

Grandeur	valeur attendue (en mm)
longueur	100
largeur	100

Résultats obtenus :

Nous mesurons ensuite les dimensions réelles du châssis du robot et nous obtenons le tableau ci-dessous :

Grandeur	valeur mesurée (en mm)
longueur	95,1
largeur	94, 6

Statut de l'essai : Nous pouvons donc conclure que cet essai est conforme au cahier des charges et respecte l'exigence EXIG_CHASSIS_DIMENSIONS.

Problèmes rencontrés : Aucun problème n'a été rencontré durant cet essai

4.6 Conclusion de la simulation / prototypage rapide du produit

Pour la partie dérisquage, seule l'exigence EXIG_ADVERSAIRE nécessite d'être confirmée. Celle-ci s'est révélée concluante. Nous avons tout de même vérifier que les capteurs CNY étaient opérationnels afin de pouvoir être en mesure de respecter les règles du combat de sumo.

5. Conclusion de la conception du produit

La conception du produit est terminée, le cahier des charges a été respecté. Nous pouvons passer à la phase de production.

6. Matrice de conformité du produit

Ce chapitre synthétise par l'intermédiaire d'un tableau la conformité du produit développé par rapport aux exigences issues du Cahier des Charges.

Exigence	Méthodes Vérification	Eléments vérifiant l'exigence	Statut
EXIG_CHASSIS_DIMENSIONS	Test	SIM08	Conforme
EXIG_MASSE	Test		
EXIG_INTEGRITE_MECA	Test		
EXIG_AUTONOMIE	Test Conception	CPR01, CPR06, CDT01	Conforme
EXIG_SECUR_BATT	Conception Fabrication	CDT05	Conforme
EXIG_ADVERSAIRE	Test Conception	CPR01, CPR02, CPR07, CPR08, CDT01, CDT02	Conforme
EXIG_FUITE	Test Conception	CPR01, CPR02, CPR07, CPR08, CDT01, CDT02	
EXIG_LUMINOSITE	Test Conception	CPR02, CPR07, CPR08, CDT01, CDT02	
EXIG_DEPART	Test Conception	CPR01, CPR07, CPR08, CDT01, CDT04	Conforme
EXIG_COTE	Test Conception	CPR02, CPR07, CPR08, CDT01, CDT02	Conforme
EXIG_DEPLACEMENT	Test Conception	CPR01, CPR04, CPR07, CPR09, CDT01, CDT02, CDT05	Conforme
EXIG_COMBAT	Test Conception	CPR01, CPR07, CPR11, CDT02, CDT04, CDT08	Conforme
EXIG_CARTE	Test Conception	CPR01, CPR05, CDT01	
EXIG_DELAI	Test Conception	DDF	Conforme
EXIG_COUT	Test Conception	DDF	Conforme