

Dossier De Conception (DDC)

du projet

Robot Mini-Sumo

Responsabilité documentaire

Action	NOM Prénom	Fonction	Date	Signature
Rédigé par	BERNARDO Emilie HEBO Dioniso CHAUBAUD Benjamin COP Benjamin KOUTHIEYE Thiaw	Technicien	06/10/2025	
Approuvé par	LEVI Thimothé (IUT GEII Bdx)	Chef de projet	03/12/2025	
Approuvé par	N'KAOUA Gilles (IUT GEII Bdx)	4	03/12/2025	

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	06/10/2025	Première publication
3	07/12/2025	Publication du DDC détaillé

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	9
2.1.3 Choix du pont en H	10
2.1.4 Choix du régulateur de tension	11
2.1.6 Choix de la LED	12
2.1.7 Choix du récepteur IR	13
2.2 Coût et délais	14
2.2.1 Coût	14
2.2.2 Délais	14
2.3 Documentation	15
2.3.1 Drive	15
2.3.2 Format documents	16
2.3.3 Nom des documents	17
2.4 Architecture Informatique	18
2.4.1 Fonctions d'acquisition	19
2.4.2 Fonctions d'action	19
2.4.3 Fonctions d'énergie	21
2.4.4 Fonctions de traitement	22
2.3 Conclusion de la conception préliminaire du produit	23
3. Conception détaillée du produit	23
3.1 Conception détaillée du robot mini-sumo	23
3.2 Conception détaillée de la partie acquisition	25
3.2.1 Capteurs adversaires	25
3.2.1.1 Schéma électrique des capteurs adversaires	25
3.2.1.2 Code informatique des capteurs adversaires	25
3.2.2 Capteurs de sol	26
3.2.2.1 Schéma électrique des capteurs de sol	26
3.2.2.2 Code informatique des capteurs de sol	27
3.2.3 Capteurs de télécommande	27
3.2.3.1 Schéma électrique des capteurs de télécommande	27
3.2.3.2 Code informatique du capteur infrarouge	28
3.3 Conception détaillée de la partie action	28
3.3.1 Pont en H	28

3.3.1.1 Schéma électrique du pont en H	28
3.3.1.2 Code informatique du pont en H	29
3.4 Conception détaillée de la partie énergie	33
3.4.1 Pont diviseur de tension	33
3.4.1.1 Schéma électrique du pont diviseur de tension	33
3.4.1.2 Code informatique du pont diviseur de tension	34
3.4.2 régulateur de tension	34
3.4.2 Batterie	35
3.5 Conception détaillée de la partie traitement	36
3.5.1 Le microcontrôleur ATMEGA328P-PU	36
3.4.1.1 Schéma électrique du microcontrôleur	36
3.3.1.2 Code informatique du microcontrôleur	37
3.6 Coût-Délai	49
4. Dérisquage des solutions techniques retenues	51
4.1 Simulation des capteurs à ultrasons	51
4.2 Simulation de pont en H	55
4.3 Simulation du départ	57
4.4 Simulation des capteurs de sol	60
4.5 Conclusion de la simulation / prototypage rapide du produit	63
5. Conclusion de la conception du produit	63
6. Matrice de conformité du produit	63

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 : CPR_ARCHI_ELEC

Exigences client vérifiées par préconception : EXIG_CARTE

Afin de répondre au cahier des charges, une analyse globale des exigences a conduit à l'architecture fonctionnelle présentée ci-dessous sur la figure 1.

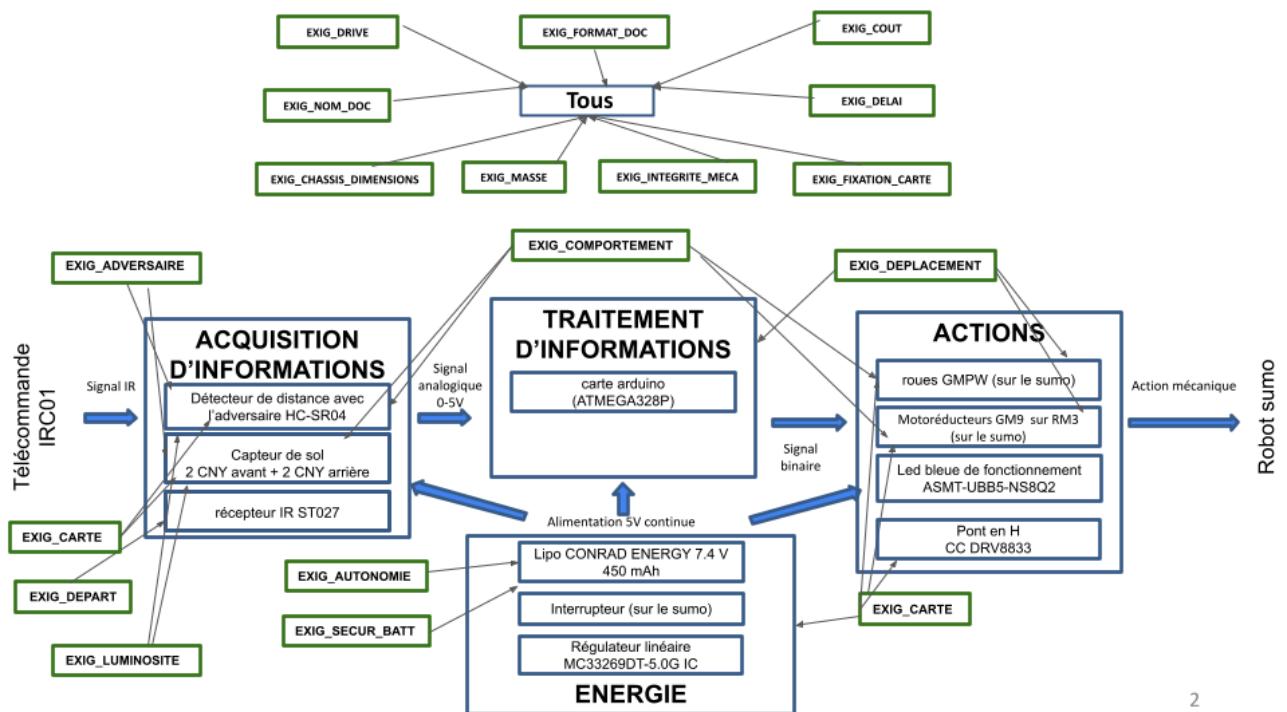


Figure 1 : Architecture électronique du robot sumo

Premièrement, nous avons le bloc acquisition. Dans celui ci, nous devons retrouver, conformément aux exigences EXIG_ADVERSAIRE, EXIG_CARTE, EXIG_COMPORTEMENT, EXIG_LUMINOSITE et EXIG_DEPART, un capteur de détection de la distance avec l'adversaire, un capteur de sol pour détecter les limites du dohyo et un récepteur infrarouge pour pouvoir détecter le signal infrarouge émis par la télécommande lors du départ du combat. Le capteur de distance doit être capable de détecter l'adversaire à une distance minimum de 40 cm selon le CDC. Concernant le capteur de sol, il doit être capable de différencier le ligne blanche qui se situe tout le tour du dohyo et le centre de celui ci qui est noir. De plus, que ce soit pour les 2 capteurs, ils doivent être choisis de façon à ce qu'aucune lumière présente lors du combat vienne perturber le fonctionnement du robot conformément au CDC. Lors de la mise en place du combat, les robots seront placés dos à dos. Grâce au signal émis par la télécommande infrarouge de référence IRC01, les robots pourront démarrer le combat. Ainsi, le récepteur infrarouge est essentiel au fonctionnement de notre robot conformément au CDC.

Deuxièmement, nous avons le bloc traitement. Dans ce projet, nous allons concevoir un shiel réalisant toutes les exigences du CDC. Celui-ci sera placé sur une carte Arduino Uno. Le microprocesseur de la carte Arduino Uno est l'ATMEGA328P. Ainsi, le cœur de notre bloc traitement et donc de notre projet sera l'ATMEGA328P.

Troisièmement, nous avons le bloc action. Dans celui-ci, nous devons retrouver conformément à l'exigences EXIG_CARTE 2 roues GMPW des motoréducteurs GM9 sur RM3. Ces 2 composants sont imposés et déjà placés sur le robot. De plus, selon l'EXIG_CARTE, il nous est demandé d'utiliser et de choisir un pont en H pour pouvoir contrôler le déplacement du robot dans toutes les directions conformément à l'EXIG_DEPLACEMENT. En effet, le robot doit pouvoir se déplacer dans toutes les directions c'est-à-dire en avant, à gauche et à droite grâce aux roues et aux motoréducteurs. Mais ce déplacement sera régi par l'EXIG_COMPORTEMENT. En effet, grâce aux données des capteurs du bloc acquisition qui seront fournies par l'ATMEGA328P du bloc traitement, le robot devra changer de directions en fonction de ces indications. Il doit donc être autonome. Nous avons aussi décidé de rajouter une LED bleue sur notre carte afin de pouvoir faciliter le débogage. Celle-ci nous indiquera si la carte est sous tension.

Finalement, nous avons le bloc énergie. Celui-ci va nous permettre de pourvoir d'alimenter l'entièreté de la carte sous différentes tensions. Conformément à l'EXIG_CARTE, nous devons utiliser un accumulateur LiPo 2s dont la référence exacte sera à choisir, un interrupteur qui est situé sur le robot qui nous permettra de mettre le robot sous tension et un régulateur linéaire si son utilisation nous est nécessaire. Selon l'EXIG_AUTONOMIE, le robot doit être capable de combattre pendant au moins 15 minutes. De plus, selon l'EXIG_SECUR_BATT, si la tension aux bornes de la batterie descend en dessous de 6.7 V, la partie puissance du robot doit être coupée. La tension fournie par un accumulateur LiPo2s est de 7.4 V. Ainsi, nous avons choisi d'utiliser un régulateur linéaire pour pouvoir stabiliser la tension à 5V pour les blocs traitement et acquisition car les composants de ces blocs nécessitent une tension réduite à 5V.

2.1.1 Choix des capteurs adversaire

Référence de pré-conception : CPR_CAPTEUR_ADV

Exigences client vérifiées par préconception : EXIG_ADVERSAIRE

Voici la fiche d'aide à la décision concernant le choix des capteurs d'adversaires, que nous pouvons observer figure 2.

IUT Bordeaux Département GEII	Référence : RMS_DDC_EQ12 Révision : 1 – 06/10/2025	7/61
----------------------------------	---	------

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)
		Distance min et max de détection	Facilité d'utilisation du signal de sortie	Compatibilité de la tension avec les sources disponibles	Faible consommation en courant	Faible Prix	Disponibilité		
Capteur de mesure Sharp GP2Y0A41SK0F	1	4 cm to 30 cm	Sortie en tension analogique	4,5 - 5,5 V	12 mA	9,90 € TTC	indisponible	0	indisponible dans les stocks de la SAE
		2	5	5	5	2	0		
Capteur de mesure Sharp GP2Y0A021YK	1	10cm to 80 cm	nelle à la distance (facile)	4,5 - 5,5 V	30 mA	11,50 € TTC	disponible	900	trop chère et commence à détecter à une trop longue distance
		4	5	5	3	3	1		
Capteur de mesure Sharp GP2Y0A021YK	1	20cm to 150 cm	nelle à la distance (facile)	4,5 - 5,5 V	33 mA	12,80 € TTC	disponible	600	trop chère et commence à détecter à une trop longue distance
		3	5	5	2	4	1		
Capteur HC-SR04	1	2cm to 4m	nelle à la distance (facile)	5 V	15 mA	3,90 € TTC	disponible	2000	choix définitif moins cher, plus grande plage de détection
		5	5	4	4	5	1		

Figure 2 : FAD pour le choix du capteur adversaire

Pour choisir le capteur destiné à détecter les adversaires, nous avons d'abord analysé les datasheets de chaque composant disponible. Cette étape nous a permis de compléter la FAD en comparant les caractéristiques techniques et pratiques de chaque option.

Ensuite, nous avons attribué des notes aux capteurs en fonction de différents critères de sélection tels que : distance min et max de détection, la facilité d'utilisation du signal de sortie, la compatibilité de la tension avec les sources disponibles, faible consommation du courant, la disponibilité sur le stock de la SAE et bien sûr le prix.

Après cette comparaison, nous avons constaté que le capteur à ultrasons HC-SR04 offrait le meilleur rapport qualité-prix parmi tous ceux de la liste. En effet, malgré son faible coût, il permet une détection fiable dans une plage allant de 2 cm à 4 mètres, tout en ayant une consommation relativement faible d'environ 15 mA en moyenne.

Cela en fait donc un composant à la fois économique, efficace et adapté à nos besoins. Ainsi, notre choix définitif est le Capteur HC-SR04.

Pour la partie logique de la détection, se référer à la figure 10 correspondant au grafset.

2.1.2 Choix des capteurs de sol

Référence de pré-conception : CPR_CAPTEUR_SOL

Exigences client vérifiées par préconception : EXIG_LUMINOSITE, EXIG_CARTE

Voici la fiche d'aide à la décision concernant le choix des capteurs de sol, que nous pouvons observer figure 3.

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)
		Distance min et max de détection	Compatibilité de la tension avec les autres dispositifs	Respect de la stratégie de combat	Fiable Prix	Disponibilité		
0 CNY avant + 0 CNY arrière	1	0 to 5mm	5V	5V	0.00€	disponible	625	ne permet pas de détecter les limites du doyo
		5	5	5	1			
1 CNY avant + 0 CNY arrière	1	0 to 5mm	5V	5V	0.85€	disponible	625	fonctionne mais ne permet pas la meilleure détection
		5	5	5	1			
2 CNY avant + 0 CNY arrière	1	0 to 5mm	5V	5V	1.70€	disponible	625	fonctionne mais ne permet pas la meilleure détection
		5	5	5	1			
0 CNY avant + 1 CNY arrière	1	0 to 5mm	5V	5V	0.85€	disponible	625	fonctionne mais ne permet pas la meilleure détection
		5	5	5	1			
1 CNY avant + 1 CNY arrière	1	0 to 5mm	5V	5V	1.70€	disponible	625	fonctionne mais ne permet pas la meilleure détection
		5	5	5	1			
2 CNY avant + 1 CNY arrière	1	0 to 5mm	5V	5V	2.55€	disponible	625	fonctionne mais ne permet pas la meilleure détection
		5	5	5	1			
2 CNY avant + 2 CNY arrière	1	0 to 5mm	5V	5V	3.40€	disponible	625	choix définitif permet la meilleure détection car il possède 4 capteurs
		5	5	5	1			

Figure 3 : FAD pour le choix du capteurs de sol

Pour les capteurs de sol (capteurs de contraste), nous avons pris la seule référence qui nous est proposée et nous avons décidé d'utiliser 4. Le positionnement stratégique de ces 4 capteurs nous offrira une couverture plus large et réduira les zones mortes qui pourraient apparaître avec un nombre plus limité de capteurs. Avec cette solution, le robot pourra réagir plus rapidement lorsqu'il s'approche des limites du doyo et ajuster sa trajectoire de manière plus efficace. Ainsi, notre choix définitif est 2 CNY à l'avant et 2 CNY à l'arrière.

2.1.3 Choix du pont en H

Référence de pré-conception : CPR_PONT_H

Exigences client vérifiées par préconception : EXIG_CARTE, EXIG_COMPORTEMENT

Voici la fiche d'aide à la décision concernant le choix du pont en H, que nous pouvons observer figure 4.

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)
		Compatibilité de la tension avec les sources disponibles	Courant max de sortie du pont	Manière de piloter le pont	Fiable/Pas	Disponibilité		
Pololu Commande de 2 moteurs CC DRV8835 2x1,2A	1	entrée logique (2-7V) => 5V	1,5 A	1 entrée p 2 PWM p	3,826	nible/ non		
Pololu Commande de 2 moteurs CC DRV8833 2x1,2A	1	alimentation (2)	1,2 A	1 entrée p Les 4 entrées	4,596	Disponible	225	moins bonne compatibilité avec la tensions avec les sources + non soudé
			5	4	5	4	400	choix définitif

Figure 4 : FAD pour le choix du pont en H

Pour le choix de notre pont en H, on a dû étudier les datasheet des 2 produits afin de compléter la FAD.

Le Pololu DRV8835 est un pont en H relativement complexe : il nécessite deux sources de tension (7,5 V et 5 V) et impose l'usage de 2 entrées PWM et 2 entrées numériques non-PWM. Cela mobilise des broches dont nous aurions besoin pour d'autres composants.

À l'inverse, le Pololu DRV8833 répond mieux à nos critères. Ses entrées (A1IN, A2IN, etc.) peuvent être utilisées indifféremment en PWM ou en logique, ce qui nous permet d'employer directement 4 broches PWM dédiées aux moteurs et de libérer davantage de broches numériques pour le reste du système. De plus, il ne demande qu'une seule alimentation 7,5 V provenant de l'accumulateur, simplifiant le câblage.

Nous retenons donc le module Pololu DRV8833 – Commande de 2 moteurs CC (2×1,2 A) comme pont en H définitif.

2.1.4 Choix du régulateur de tension

Référence de pré-conception : CPR_REGULATEUR

Exigences client vérifiées par préconception : EXIG_CARTE, EXIG_AUTONOMIE, EXIG_SECUR_BATT

Voici la fiche d'aide à la décision concernant le choix du régulateur de tension, que nous pouvons observer figure 5.

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)
		Tension max d'entrée	Tension de sortie	Faible chute de tension sortie entrée	Courant de sortie max	Bon/ mauvais composant facile à utiliser/soudar	Faible Prix		
LP2985AIM5-3.3/NOPB 3.3V CMS	0	16V	3.3V	IV pour 1V en é	150 mA	(CMS) >= très	0.60€	Disponible	0
		4	0	5	5	4	3	1	
LP2985AIM5-5.0/NOPB 5V CMS	1	16V	5V	IV pour 1V en é	150 mA	(CMS) >= très	0.99€	Disponible	4000
		4	5	5	5	4	2	1	
MC33269DT-3.3G IC	0	20 V	3.3V	<1V	800mA	ré => facile à m	0.24 €	Disponible	0
		5	0	4	4	5	5	1	
MC33269DT-5.0G IC	1	20 V	5V	<1V	800mA	cuit intégré (ta	0.27 €	Disponible	8000
		5	5	4	4	5	4	1	

Figure 5 : FAD pour le choix du régulateur

Après avoir consulté les datasheets des composants constituant notre carte, nous avons réalisé que tous les composants ne nécessitent pas la même tension d'alimentation. Ainsi, nous avons décidé d'utiliser un régulateur linéaire pour stabiliser et abaisser la tension afin de pouvoir répondre aux nécessités de chaque composants.

Nous avons ensuite étudié les datasheets des références qui nous ont été proposées et rempli la FAD. Cette analyse nous a permis de comparer les composants de manière efficace et de conclure que la référence MC33269DT-5.0G IC est la plus adaptée à notre projet, en raison de son faible coût et de sa facilité d'intégration.

2.1.5 Choix de la batterie

Référence de pré-conception : CPR_BATTERIE

Exigences client vérifiées par préconception : EXIG_AUTONOMIE, EXIG_SECUR_BATT

Voici la fiche d'aide à la décision concernant le choix de la batterie, que nous pouvons observer figure 6.

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	Fiable Prix	Disponibilité		
Lipo CONRAD ENERGY 7.4 V 1200 mAh	0	75 g, 72x36x12 mm	20,99€	1	0	les dimensions sont supérieures à celles indiqués dans le cahier des charges pour le bloc batterie
Lipo CONRAD ENERGY 7.4 V 800 mAh	0	52 g, 56x52x17 mm	13,49€	1	0	les dimensions sont supérieures à celles indiqués dans le cahier des charges pour le bloc batterie
Lipo CONRAD ENERGY 7.4 V 450 mAh	1	35 g, 56x31x11,5 mm	9,49€	1	25	choix définitif

Figure 6 : FAD pour le choix de la batterie

Après avoir consulté les datasheet des 3 accumulateurs LiPo 2s, nous avons pu remplir la FAD. Selon le CDC, une dimension de la batterie était imposée. En effet, les cotes internes du bloc batterie sont : 57mm x 31mm x 17mm. Ainsi nous avons réalisé notre choix principalement sur ce critère à savoir que seul un des 3 accumulateurs répond à ce critère : Lipo CONRAD ENERGY 7.4 V 450 mAh. De plus il s'avère être le moins cher donc notre choix définitif est le Lipo CONRAD ENERGY 7.4 V 450 mAh.

Pour la consommation totale, nous ne mesurons que la consommation des moteurs, car la consommation des composants électroniques est négligeable par rapport à celle des moteurs.

Nous avons mesuré une consommation de courant à vide de 0,28 A. Nous avons besoin que les moteurs puissent tourner pendant 1,08 h, soit :

$$1,08 \times 0,28 = 300 \text{ mAh.}$$

Ainsi, l'accumulateur LiPo Conrad Energy 7,4 V 450 mAh est suffisant et possède même une marge d'erreur, garantissant que les consommations négligées n'impacteront pas l'autonomie.

2.1.6 Choix de la LED

Référence de pré-conception : CPR_LED

Voici la fiche d'aide à la décision concernant le choix de la LED, que nous pouvons observer figure 7.

IUT Bordeaux Département GEII	Référence : RMS_DDC_EQ12 Révision : 1 – 06/10/2025	12/61
----------------------------------	---	-------

Solution technique proposée	Conformité à toutes les exigences (oui : 1 / non : 0)	(valeur de 1 à 5)							Total (produit des valeurs précédentes)	Commentaires (si nécessaire)
		couleur de la led	intensité lumineuse	tension directe	courant nominal	consommation/rendement	prix unitaire	disponibilité		
ASMB-MTB0-0A3A2	1	RGB	350 mcd	3.1 V	25 mA	90 W	0.76€	disponible	2560	consommation et prix supérieur
ASMT-UBB5-NS8Q2	1	bleu	392 mcd	3.2 V	20 mA	114 mW	0.26€	disponible	5625	choix définitif
ASMT-UGB5-NV702	1	vert	1250 mcd	3.2 V	20 mA	111 mW	indisponible	indisponible	0	Indisponible sur Farnell, la led n'est pas bleue
ASMT-URB4-YU802	1	rouge	980 mcd	2.15V	30 mA	63 mW	indisponible	indisponible	0	Indisponible sur Farnell, la led n'est pas bleue

Figure 7 : FAD pour le choix de la LED

Aucune exigence demande le choix d'une LED sur le robot sumo. Cependant, nous avons décidé d'en ajouter une pour nous faciliter le débogage de la carte. Nous avons décidé que la couleur de la led sera bleue. Ainsi, nous avons consulté les datasheet de 4 leds différentes et avons donc constitué la fiche d'aide à la décision que nous pouvons observer sur la figure 7. Ainsi, nous avons choisi la led de référence ASMT-UBB5-NS8Q2 car celle-ci est bleue, consomme moins et émet une intensité lumineuse supérieure à celle des autres leds.

Ainsi, notre choix définitif est la led ASMT-UBB5-NS8Q2.

2.1.7 Choix du récepteur IR

Référence de pré-conception : CPR_RECEPTEUR

Exigences client vérifiées par préconception : EXIG_DEPART

Pour répondre à cette exigence, nous avons choisi le récepteur IR de référence ST027 car c'est le seul en stock et qu'après essais, il fonctionne très bien.

2.2 Coût et délais

2.2.1 Coût

Référence de pré-conception : CPR_COUT

Exigences client vérifiées par préconception : EXIG_COUT

Le coût total de l'ensemble des composants (mécaniques et électroniques) nécessaires pour la fabrication d'un seul prototype du robot mini-sumo est inférieur à 120 € HT.

Afin de répondre à l'exigence de coût, nous avons réalisé une première estimation du prix. Étant encore en phase de conception préliminaire, nous ne connaissons pas les références exactes ni l'ensemble des composants nécessaires à la réalisation de ce projet. Nous avons donc estimé le prix en utilisant les composants choisis grâce au FAD, afin de nous assurer que, dans le pire des cas, nous respectons l'exigence comme nous pouvons le voir sur la figure 8.

Référence	Quantité	Coût unitaire (€ TTC)	cout total (€ TTC)	cout total préliminaire (€ TTC)
Capteur HC-SR04	2	3,9	7,8	29,71
CNY	4	0,85	3,4	
Pololu Commande de 2 moteurs CC DRV8833 2x1,2A	1	4,59	4,59	
MC33269DT-5.0G IC	1	0,27	0,27	
Lipo CONRAD ENERGY 7,4 V 450 mAh	1	9,49	9,49	
ASMT-UBB5-NS8Q2	1	0,26	0,26	
récepteur IR ST027	1	3,9	3,9	

Figure 8 : Tableau des coûts préliminaire

Ainsi, pour la conception préliminaire, le coût total de notre projet est égal à 29.71 euros TTC. Il nous reste donc une marge de 90.29 euros TTC pour la conception détaillée sans compter la structure du robot en elle même qui est de 30 euros.

2.2.2 Délais

Référence de pré-conception : CPR_DELAIS

Exigences client vérifiées par préconception : EXIG_DELAI

Le projet de développement du robot sumo respecte l'exigence des 84 heures allouées, grâce à une gestion efficace du temps et du suivi du planning de développement de la figure 9.

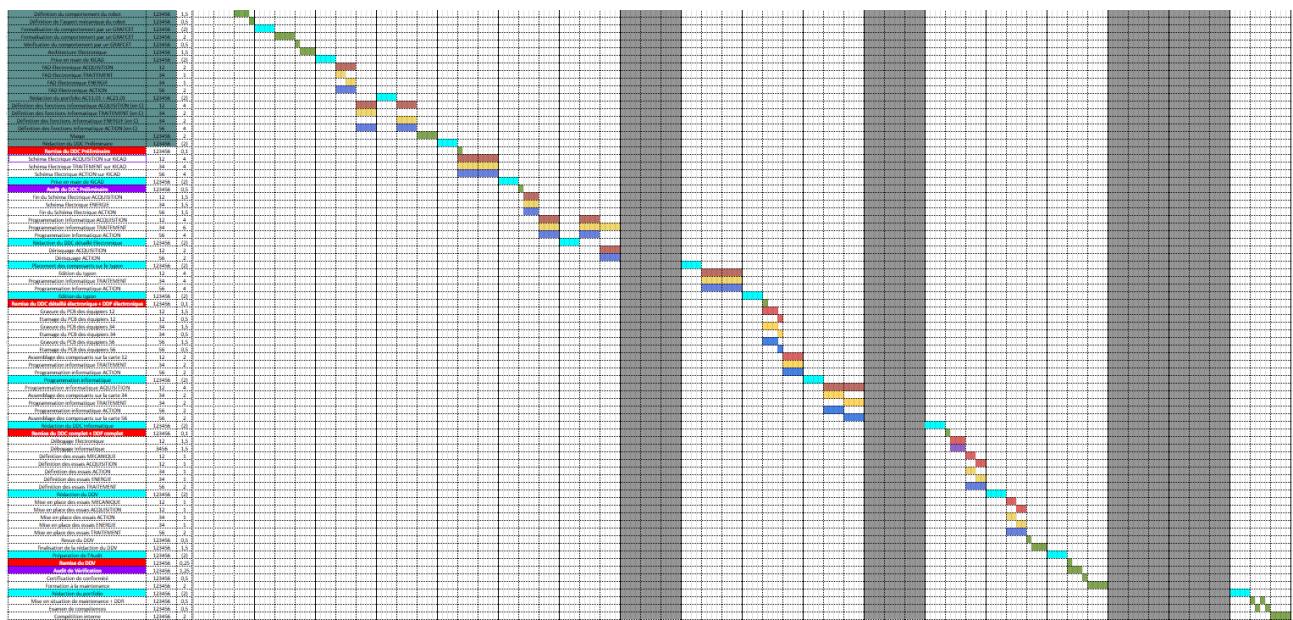


Figure 9 : Planning de développement

2.3 Documentation

2.3.1 Drive

Référence de pré-conception : CPR_DRIVE

Exigences client vérifiées par préconception : EXIG_DRIVE

Afin de pouvoir travailler en équipe, il nous est imposé d'utiliser un dossier GOOGLE DRIVE partagé et d'en fournir le lien à votre hiérarchie.

Nous avons donc réalisé un dossier drive travail et client dans lequel nous allons déposer nos différents documents comme nous pouvons le voir sur la figure 10.

IUT Bordeaux Département GEII	Référence : RMS_DDC_EQ12 Révision : 1 – 06/10/2025	15/61
----------------------------------	---	-------

Mon Drive > SAE sumo equipe2

The screenshot shows a Google Drive interface. At the top, there are four filter buttons: 'Type' (dropdown), 'Contacts' (dropdown), 'Date de modification' (dropdown), and a search bar starting with 'S'. Below these, a sorting header 'Nom' with an upward arrow is followed by a horizontal line. Underneath, three folder items are listed: 'dossier client' (with a folder icon), 'dossier travail' (with a folder icon), and 'Semestre 3' (with a folder icon and two people icons). Each item has a horizontal line below it.

Figure 10 : Dossiers drive

2.3.2 Format documents

Référence de pré-conception : CPR_FORMAT_DOC

Exigences client vérifiées par préconception : EXIG_FORMAT_DOC

Les enseignants évaluent uniquement les DDC, DDF et DDV au format PDF. Les FAD peuvent rester au format Google Sheet. Ainsi, nos FAD et notre dossier de conception préliminaire ont été mis dans le dossier client dans le bon format comme nous pouvons le voir sur la figure 11.

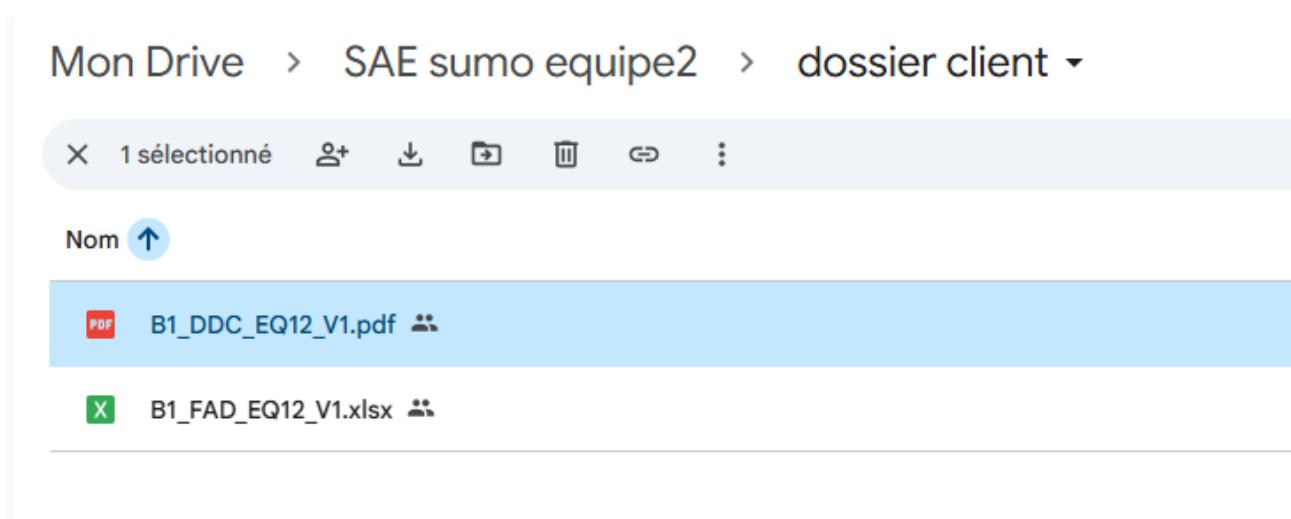


Figure 11 : Dossier client préliminaire

2.3.3 Nom des documents

Référence de pré-conception : CPR_NOM_DOC

Exigences client vérifiées par préconception : EXIG_NOM_DOC

Le nom des fichiers doit respecter un formalisme imposé afin de s'y retrouver rapidement dans les documents reçus. Pour notre équipe, le nom du DDC doit être le suivant : B1_DDC_EQ12_V1.odt. Comme nous pouvons le voir sur la figure 12, c'est bien le nom qui a été donné à notre DDC. Nous avons fait de même pour la FAD.



Figure 12 : Nom DDC

2.4 Architecture Informatique

Référence de pré-conception : CPR_ARCHI_INFO

Afin de répondre au cahier des charges, une analyse globale des exigences a conduit à l'architecture fonctionnelle présentée ci-dessous sur la figure 13. Pour la partie logique, se référer à la figure 12 correspondant au grafcet.

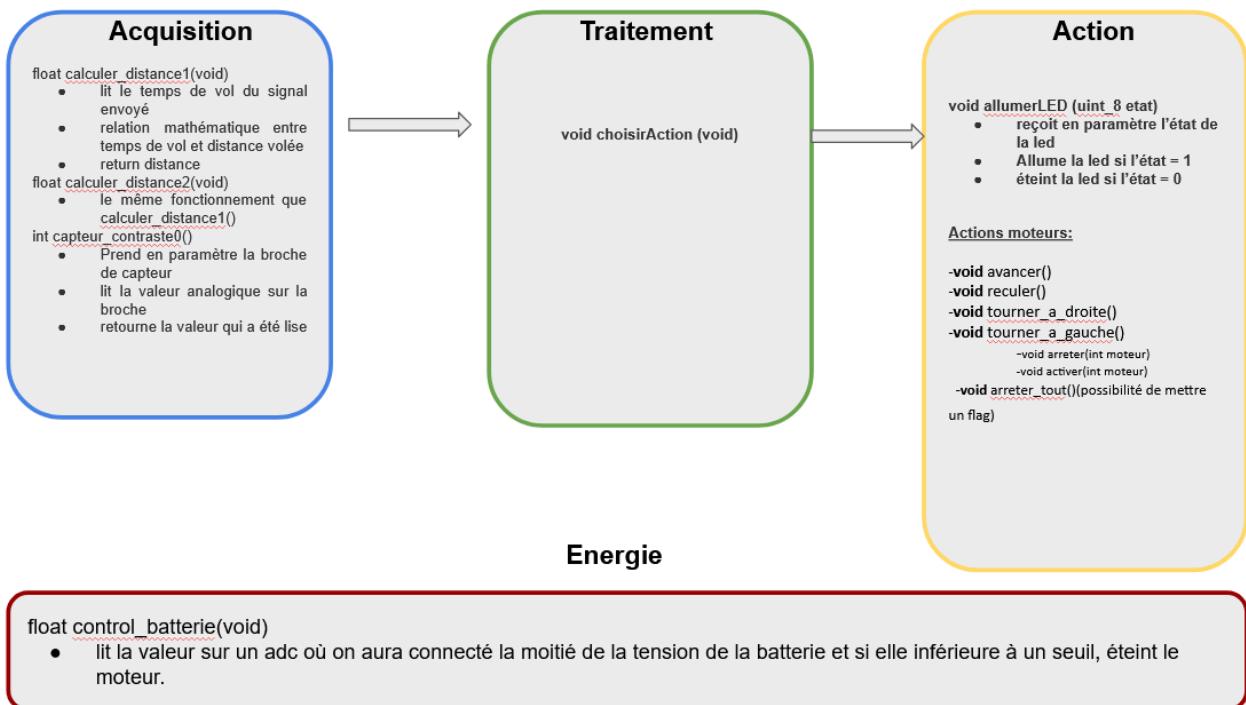


Figure 13 : Schéma de l'architecture informatique du robot sumo

IUT Bordeaux Département GEII	Référence : RMS_DDC_EQ12 Révision : 1 – 06/10/2025	18/61
----------------------------------	---	-------

2.4.1 Fonctions d'acquisition

Référence de pré-conception : CPR_INFO_AQUISITION

Exigences client vérifiées par préconception : EXIG_ADVERSAIRE, EXIG_LUMINOSITE, EXIG_DEPART

Dans cette partie nous nous proposons d' exploiter informatiquement les montages électroniques de manière à pouvoir extraire des informations qui seront utiles pour le comportement de notre robot.

Pour la partie acquisition, nous avons décidé de faire trois fonction :

1. float calculer_distance1(void) : nous allons utiliser cette fonction pour pouvoir mesurer à quelle distance l'adversaire se trouve. Cette fonction ne prend rien en paramètre mais retourne la distance. Elle exploite un capteur ultrasonore de référence HC-SR04.
2. float calculer_distance2(void) : elle fonctionne de manière identique à calculer_distance1() mais elle exploite un autre capteur ultrasonore.
3. int capteur_contraste(uint8_t capteur) : cette fonction reçoit en paramètre la broche à laquelle est connecté le capteur de contraste que nous souhaitons exploiter. Nous avons choisi cette approche car nous aurons quatre capteurs de contraste sur notre robot et pour ne pas avoir quatre fonctions qui font la même chose, nous aurons une qui peut gérer les quatre capteurs.. Cette fonction lit la valeur analogique sur la broche indiquée et retourne la valeur lue. Cette valeur correspond à la tension mesurée, elle-même proportionnelle à la luminosité réfléchie et captée par le capteur.

Les fonctions calculer_distance1(void) et calculer_distance2(void) répondent à l'exigence qui a comme référence EXIG_ADVERSAIRE et la fonction capteur_contraste(uint8_t) répond à EXIG_LUMINOSITE.

Pour répondre à EXIG_DEPART nous n'avons pas fait une fonction, nous avons fait une boucle while infinie qui bloque le code dans la partie void setup() tant qu'aucun signal a été reçu.

2.4.2 Fonctions d'action

Référence de pré-conception : CPR_INFO_ACTION

IUT Bordeaux Département GEII	Référence : RMS_DDC_EQ12 Révision : 1 – 06/10/2025	19/61
----------------------------------	---	-------

Exigences client vérifiées par préconception : EXIG_DEPLACEMENT

La fonction `void activer(int moteur, int sens, int vitesse)` constitue la fonction principale de notre code. Elle prend trois arguments en entrée : *moteur*, qui désigne soit le moteur gauche soit le moteur droit (il est important de ne pas activer les deux en même temps si l'on souhaite effectuer une rotation), *sens*, qui permet de déterminer si le mouvement se fait en avant ou en arrière.

Pour changer le sens de rotation du moteur, on choisit sur quel Pin on applique notre signal PWM et notre signal logique grâce aux Pins X1in et X2in avec X représente A ou B comme nous pouvons le voir sur la figure 14, j'ai choisi d'utiliser les moteurs en mode fast decay pour le contrôle des moteurs de mon robot car il offre une réponse plus rapide et un freinage plus efficace. Ce mode me permet d'obtenir une meilleure réactivité lors des changements de direction ou des arrêts brusques, ce qui est essentiel pour les manœuvres précises que doit réaliser notre robot.

De plus, le fast decay favorise une meilleure maîtrise du mouvement dans les situations dynamiques, ce qui correspond parfaitement aux exigences du projet., et enfin *vitesse*, qui correspond au signal PWM appliqué au moteur. Une valeur de 0 pour la PWM indique l'utilisation de 100 % de la puissance moteur, conformément à notre stratégie.

La fonction `void arreter(int moteur)` permet, quant à elle, d'arrêter un moteur spécifiquement, ce qui est utile pour effectuer un virage, en arrêtant un moteur pendant que l'autre continue de tourner. Cela permet d'effectuer des virages à gauche ou à droite selon le moteur arrêté.

La fonction `void avancer()` repose sur `activer(int, int, int)` en activant à la fois le moteur gauche (Moteur_B) et le moteur droit (Moteur_A) dans le sens avant, avec une PWM réglée à 0 pour une puissance maximale. La fonction `void reculer()` suit la même logique, mais en définissant le sens comme étant arrière pour les deux moteurs, tout en conservant la même stratégie de PWM maximale.

Pour effectuer un virage à droite, la fonction `void tourner_a_droite()` utilise la fonction `activer(int, int, int)` pour faire avancer uniquement le moteur droit (Moteur_A) avec une PWM maximale, tandis que la fonction `arreter(int)` est utilisée pour stopper le moteur gauche. À l'inverse, la fonction `void tourner_a_gauche()` active uniquement le moteur gauche (Moteur_B) en avant avec une PWM maximale, tandis que le moteur droit est arrêté via la fonction `arreter(int)`. Ces combinaisons permettent une gestion fine des déplacements directionnels du robot.

xIN1	xIN2	FUNCTION
PWM	0	Forward PWM, fast decay
1	PWM	Forward PWM, slow decay
0	PWM	Reverse PWM, fast decay
PWM	1	Reverse PWM, slow decay

Figure 14 : Table de contrôle des entrées moteur

2.4.3 Fonctions d'énergie

Référence de pré-conception : CPR_INFO_ENERGIE

Exigences client vérifiées par préconception : EXIG_AUTONOMIE, EXIG_SECUR_BATT

Pour monitorer la tension de la batterie nous utiliserons une entrée ADC du microcontrôleur après avoir abaissé la tension à l'aide d'un pont diviseur de tension.

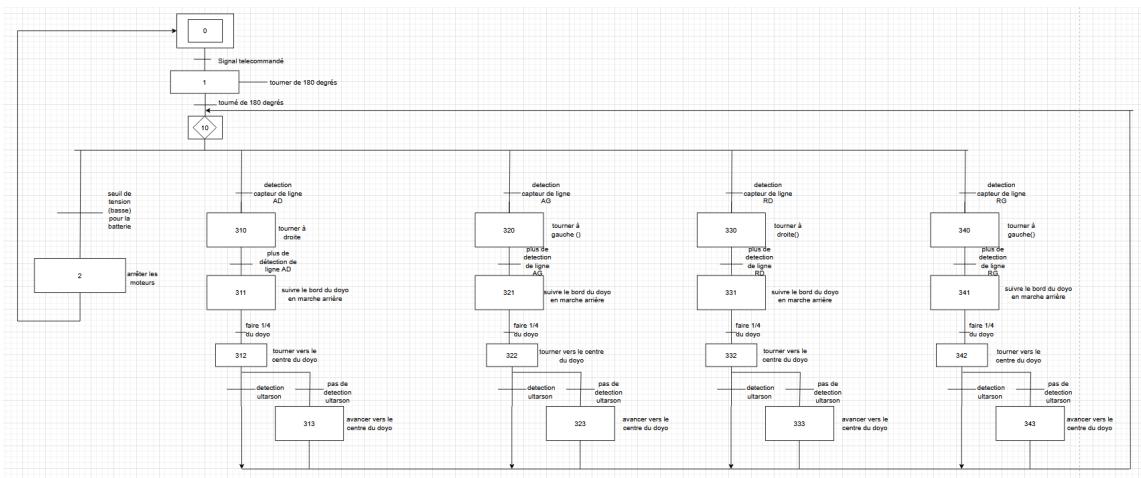
Pour la partie logique, se référer à la figure 15 correspondant au grafct.

2.4.4 Fonctions de traitement

Référence de pré-conception : CPR_INFO_TRAITEMENT

Exigences client vérifiées par préconception : EXIG_COMPORTEMENT

La partie traitement et le comportement du robot sont détaillée dans la figure 10.



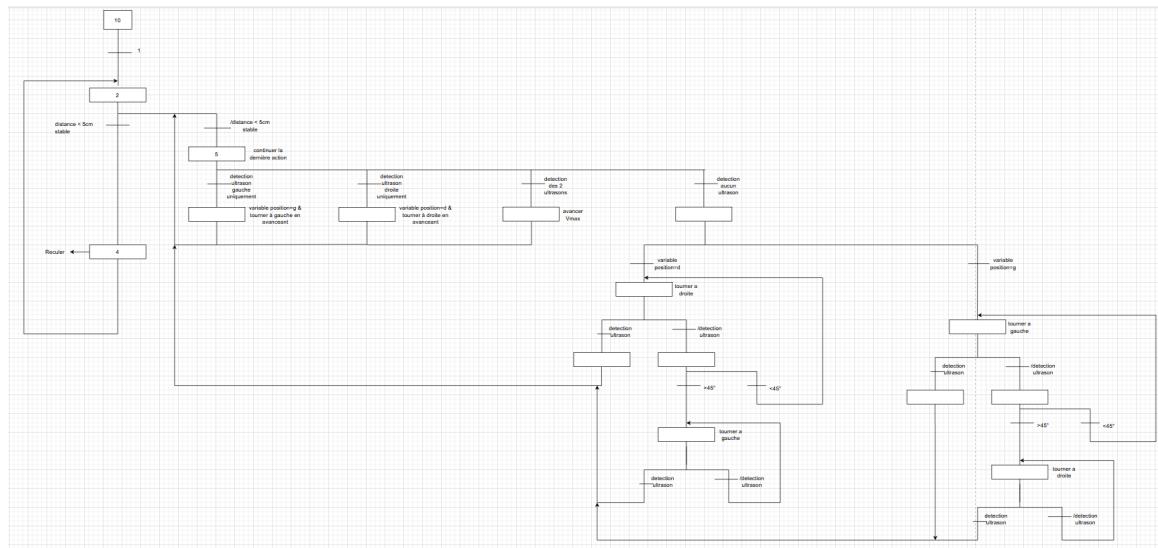


Figure 15 : Grafcet

Ce grafcet est un grafcet de développement permettant de comprendre le comportement détaillé du robot. Il est développé au maximum avec les actions et les transitions les plus simples. Cependant, à terme, le grafcet sera condensé au maximum dans l'optique d'optimiser le temps de réaction du robot et de réduire la durée des séquences. C'est-à-dire qu'il n'y aura qu'une seule étape par action, avec des transitions calculant toutes les possibilités d'activer cette action, comme nous pouvons le voir sur ma figure 16.

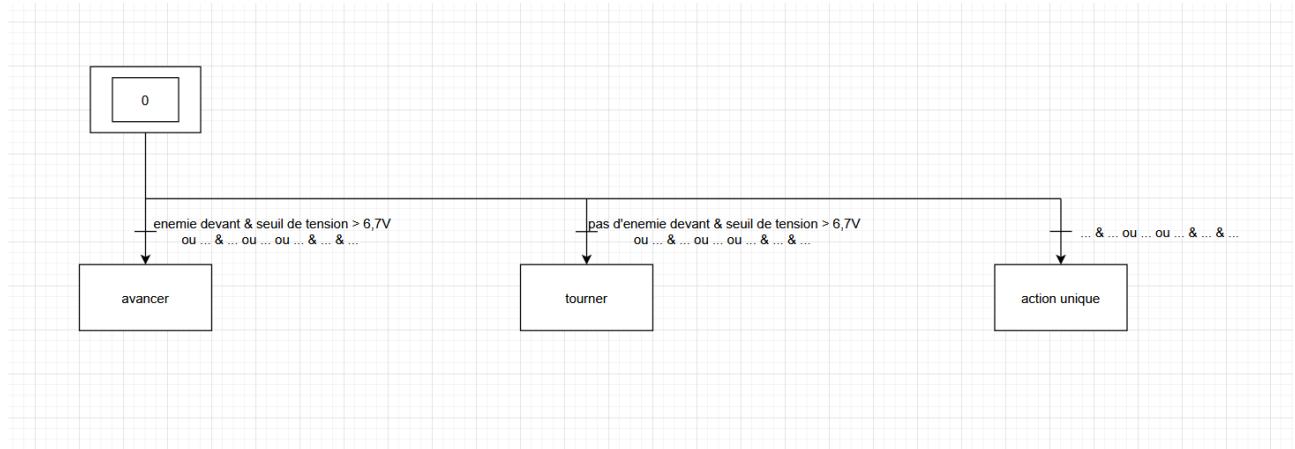


Figure 16 : Grafcet optimisé

2.3 Conclusion de la conception préliminaire du produit

La phase de conception préliminaire nous a permis de développer une architecture électronique et informatique du notre robot sumo. Chaque exigence a été analysée et une solution technique adaptée a été proposée, accompagnée de sa justification. Les choix de dimensionnement des composants seront affinés lors de la phase suivante, celle de la conception détaillée. À ce stade, la faisabilité technique du produit a été validée, assurant ainsi que les solutions retenues répondent aux contraintes techniques et fonctionnelles définies.

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 : CDT_SCHEMA_ELEC

Exigences client vérifiées : EXIG_CARTE, EXIG_FIXATION_CARTE,
EXIG_SECUR_BATT, EXIG_AUTONOMIE, EXIG_ADVERSAIRE, EXIG_DEPART,
EXIG_DEPLACEMENT

À 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 sur la figure 17.

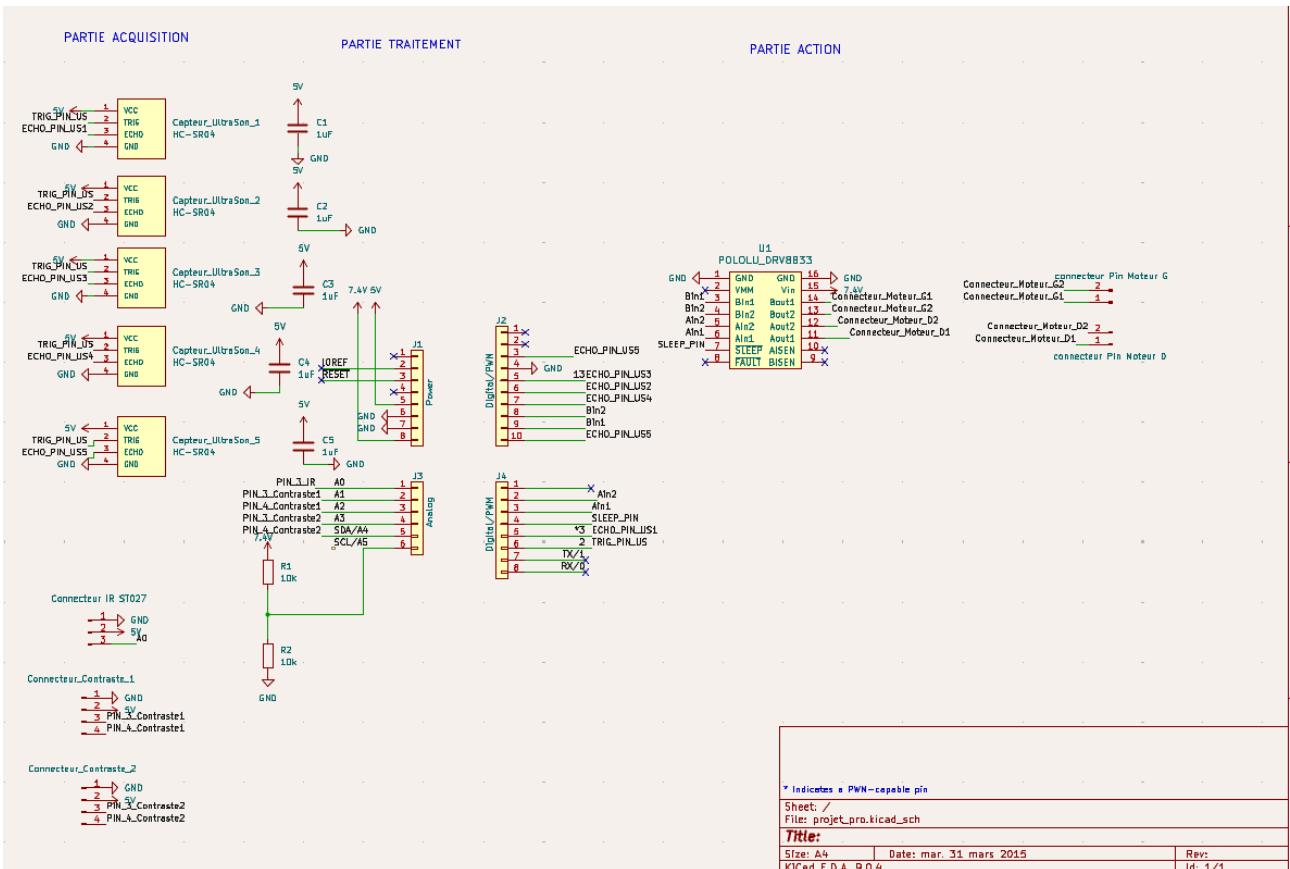


Figure 17 : Schéma électrique de la carte robot mini-sumo

3.2 Conception détaillée de la partie acquisition

3.2.1 Capteurs adversaires

Référence de conception : CDT_CAPTEUR_ADV

Exigences client vérifiées : EXIG_ADVERSAIRE

3.2.1.1 Schéma électrique des capteurs adversaires

Pour détecter les adversaires, nous allons utiliser cinq modules HC-SR04 prêts à l'emploi. Ces capteurs à ultrasons permettent de mesurer la distance entre le robot et les objets autour en envoyant des ondes sonores et en calculant le temps de retour. Ces modules sont faciles à intégrer : il suffit de les câbler correctement aux broches d'alimentation, de masse et de signal du microcontrôleur pour qu'ils fonctionnent, puis d'écrire le code pour lire les distances détectées. Nous pouvons observer le schéma électrique de l'étage pour le module HC-SR04 sur la figure 18.

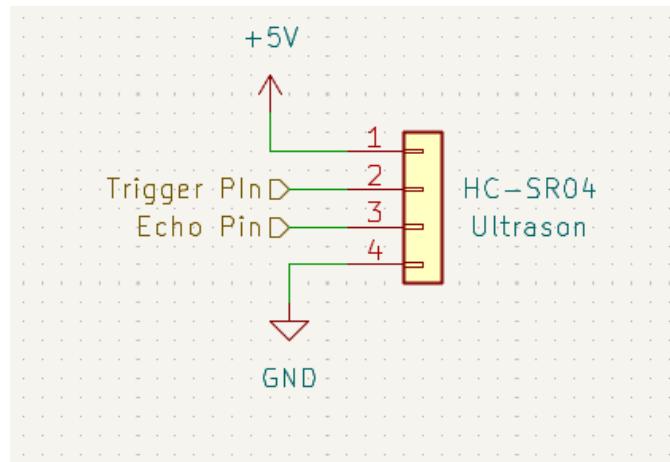


Figure 18 : Schéma électrique de l'étage pour le module HC-SR04

3.2.1.2 Code informatique des capteurs adversaires

```
float calculer_distance(int trigPin, echoPin) {  
    float duration, distance;  
    digitalWrite(trigPin, LOW);  
    delayMicroseconds(2);  
    digitalWrite(trigPin, HIGH);  
    delayMicroseconds(10);  
    digitalWrite(trigPin, LOW);  
    duration = pulseIn(echoPin, HIGH);  
    if (duration == 0) {  
        return -1; // valeur qui indique une erreur  
    }  
}
```

```

    distance = (duration * 0.0343) / 2;
    return distance;
}

```

La fonction calculer_distance() permet de mesurer la distance d'un obstacle avec le capteur ultrason HC-SR04. Elle envoie d'abord une impulsion de 10 microsecondes sur la broche Trig pour déclencher l'émission d'une onde ultrasonore. Ensuite, elle mesure le temps pendant lequel la broche Echo reste haute, correspondant au retour de l'onde après rebond sur un objet. Ce temps, exprimé en microsecondes, est converti en distance en centimètres en utilisant la vitesse du son et en divisant par deux pour tenir compte de l'aller-retour. Si aucun écho n'est détecté, la fonction renvoie -1 pour indiquer une erreur ou un obstacle trop éloigné.

3.2.2 Capteurs de sol

Référence de conception : CDT_CAPTEUR_SOL

Exigences client vérifiées : EXIG_LUMINOSITE, EXIG_CARTE

3.2.2.1 Schéma électrique des capteurs de sol

Pour les capteurs de sol, nous allons utiliser 4 modules CNY70, chaque module utilise deux résistances car il est composé par une led qui joue le rôle de l'émetteur et un phototransistor qui joue le rôle du détecteur.

Pour notre projet, nous voulons distinguer la ligne blanche de la noire, donc avoir un courant de 20mA qui traverse la led sera suffisante, et pour ce courant nous aurons une tension dans la led égale à 1,1V comme nous pouvons le voir ci dessous :

$$5V = 1,1V + R * 20 * 10^{-3}$$

$$R = \frac{5-1,1}{20*10^{-3}} = 195\Omega$$

Cette valeur est une valeur théorique, nous avons deux choix pour la valeur normalisée, 180Ω ou 220Ω . Si nous utilisons la valeur de 220Ω , nous allons réduire le courant ce qui est positif car cela résultera dans une consommation plus basse.

Si nous faisons le calcul dans le sens inverse, nous pouvons trouver quel courant sera fourni en réalité comme nous pouvons le voir ci dessous :

$$I_f = \frac{5-1,1}{220} = 17,7mA$$

Donc en réalité, nous aurons presque 17,7mA dans notre led.

Pour la résistance à mettre en série avec le transistor, nous allons utiliser une valeur qui nous permet de limiter le courant. Après avoir consulté des montages qui marchaient, nous avons décidé de prendre une résistance de $10 k\Omega$.

Nous pouvons observer le schéma électrique final sur la figure 19.

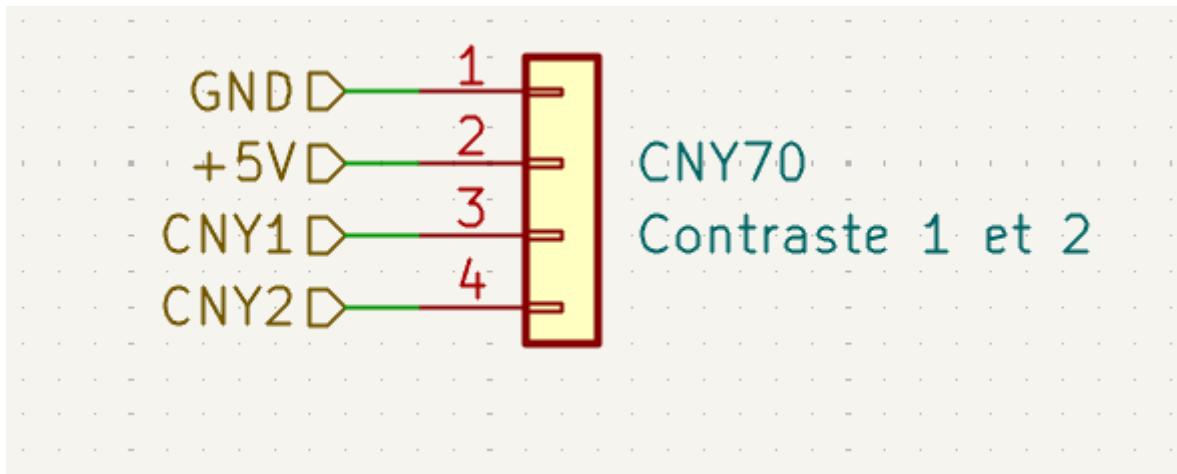


Figure 19 : Schéma électrique de l'étage du CNY70

3.2.2.2 Code informatique des capteurs de sol

```
bool capteur_contraste(int capteur_broche) {
    int contraste = analogRead(capteur_broche);
    return contraste <= 800;
}
```

La fonction `capteur_contraste()` lit la valeur analogique du capteur via `analogRead`. Elle compare cette valeur à un seuil fixé à 800. Si la valeur est inférieure ou égale à 800, la fonction renvoie true (détection du contour blanc) ; sinon, elle renvoie false (aucun contour blanc détecté).

3.2.3 Capteurs de télécommande

Référence de conception : CDT_RECEPTEUR

Exigences client vérifiées : EXIG_DEPART

3.2.3.1 Schéma électrique des capteurs de télécommande

Pour recevoir le signal de la télécommande utilisée par l'arbitre pendant la partie, nous allons utiliser un module récepteur infrarouge (IR Receiver) d'ARDUINO. C'est un module prêt à l'emploi donc nous avons besoin de mettre seulement un capteur sur notre carte comme nous pouvons le voir sur la figure 20.

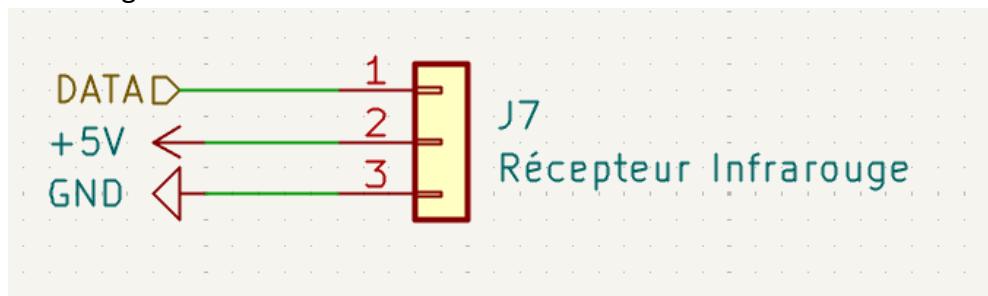


Figure 20 : Schéma électrique de l'étage du capteur IR

3.2.3.2 Code informatique du capteur infrarouge

Plutôt que d'utiliser une bibliothèque logicielle pour décoder la trame binaire IR, nous avons opté pour une approche analogique plus directe. En mesurant la tension en sortie du récepteur, nous avons constaté une élévation systématique du signal lors de l'appui sur n'importe quel bouton. Le décodage précis de la commande étant superflu (nous n'avons besoin que d'un unique signal de départ), nous avons mis en place une détection par seuil de tension. Cette méthode permet de démarrer le robot avec n'importe quelle touche de la télécommande, prévenant ainsi tout échec de lancement si l'arbitre venait à utiliser le mauvais bouton.

```
int PIN_IR = A0;  
  
while (analogRead(PIN_IR) > 900) { // Séquence de démarrage bloquante  
    // On attend que le module IR reçoive le signal  
}
```

3.3 Conception détaillée de la partie action

3.3.1 Pont en H

Référence de conception : CDT_PONT_H

Exigences client vérifiées : EXIG_CARTE, EXIG_COMPORTEMENT

3.3.1.1 Schéma électrique du pont en H

Le pont en H que nous avons sélectionné grâce aux FAD est le Pololu Commande de 2 moteurs CC DRV8833 2x1,2A comme nous pouvons le voir sur la figure 21.. Celui-ci permet au robot de changer de direction de façon autonome en utilisant les données de la partie acquisition et en agissant sur les actionneurs qui sont les motoréducteurs et les roues. Le robot doit être autonome dans ses déplacements à partir de l'appui sur la télécommande.

Nous devons lisser le courant en entrée du pont en H pour avoir une tension stable. Pour ce faire, nous allons placer un condensateur de découplage de $10\mu F$ en entrée du pont en H. Celui-ci n'est pas représenté sur la figure 21 mais les condensateurs nécessaires sont déjà intégrés au pololu.

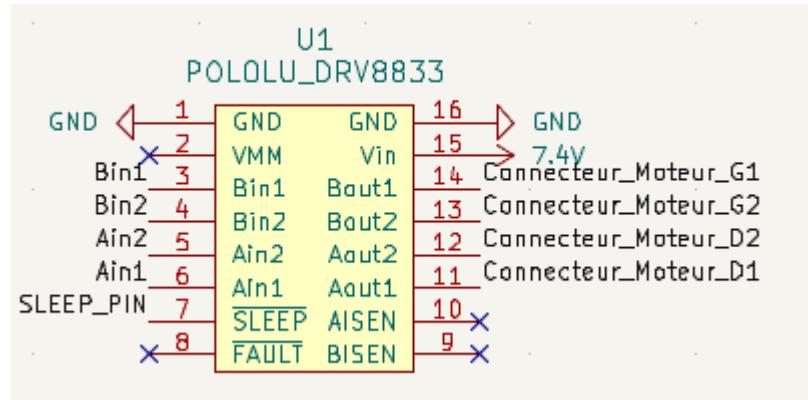


Figure 21 : Schéma électrique de la partie action

3.3.1.2 Code informatique du pont en H

Voici le code commenté que nous avons réalisé pour le pont en H :

```
//moteur droite = moteur A
//moteur gauche = moteur B

int AIN1 = 5; // on définit la broche AIN1 du pont en H sur la carte Arduino
port 5

int AIN2 = 6; // on définit la broche AIN1 du pont en H sur la carte Arduino
port 6

int BIN1 = 9; // on définit la broche AIN1 du pont en H sur la carte Arduino
port 9

int BIN2 = 10; // on définit la broche AIN1 du pont en H sur la carte
Arduino port 10

const int MOTEUR_A=0;// on définit les constantes qui définissent l'état
logique du MoteurA
const int MOTEUR_B=1;// on définit les constantes qui définissent l'état
logique du MoteurB
const int AVANT=0; // État qui va nous faire avancer
const int ARRIERE=1;// Etat qui va nous faire reculer
```

```

void setup()
{
    pinMode(AIN1, OUTPUT); // définition des pins moteursA
    pinMode(AIN2, OUTPUT); // définition des pins moteursA
    pinMode(BIN1, OUTPUT); // définition des pins moteursB
    pinMode(BIN2, OUTPUT); // définition des pins moteursA
}

void loop(){ //le programme de la loop sera modifié en fonction de notre
stratégie
//ce code sert à tester toutes les fonctions

    avancer(); //on avance
    delay(1000); //on attend entre chaque instructions
    reculer(); //on recule
    delay(1000);
    tourner_a_gauche(); //on tourne à gauche
    delay(1000);
    tourner_a_droite(); // on tourne à droite
    delay(1000);
}

void activer(int moteur, int sens, int vitesse) //fonction qui prend en
argument le moteur à choisir le sens et la vitesse des moteurs

// moteur : MOTEUR_A ou MOTEUR_B
// sens : AVANT ou ARRIERE
// vitesse : entre 0 et 255 (inversé)
{
    switch (moteur) // switch case pour définir quel moteur, sa vitesse et son
sens
    {
        case MOTEUR_A :
            if (sens==AVANT) // sens horaire
            {

```

```

        analogWrite(AIN1,vitesse); //On envoie un signal PWM pour changer de
vitesse
        digitalWrite(AIN2,LOW); //les moteurs sont à l'état bas et peuvent
être activés

    }

else
{
    digitalWrite(AIN1,LOW); //les moteurs sont à l'état bas et peuvent être
activés
    analogWrite(AIN2,vitesse); //On envoie un signal PWM pour changer de
vitesse
}

break;

case MOTEUR_B :
if (sens==AVANT)
{
    analogWrite(BIN1,vitesse); //On envoie un signal PWM pour changer de
vitesse
    digitalWrite(BIN2,LOW); //les moteurs sont à l'état bas et peuvent être
activés

}
else
{
    digitalWrite(BIN1,LOW); //les moteurs sont à l'état bas et peuvent être
activés
    analogWrite(BIN2,vitesse); //On envoie un signal PWM pour changer de
vitesse
}

break;

}

}

void arreter_tout() //on met tous les moteurs à l'état bas

```

```

{
    digitalWrite(AIN1, LOW);
    digitalWrite(AIN2, LOW);
    digitalWrite(BIN1, LOW);
    digitalWrite(BIN2, LOW);
}

void arreter(int moteur) // choisit d'arrêter le Moteur A ou B
{
    switch (moteur)
    {
        case MOTEUR_A :
            digitalWrite(AIN1, LOW);
            digitalWrite(AIN2, LOW);
        case MOTEUR_B :
            digitalWrite(BIN1, LOW);
            digitalWrite(BIN2, LOW);
    }
}

void avancer() //fais avancer le robot en ligne droite à vitesse max en
activant les deux moteurs à un PWM max
{
    activer(MOTEUR_A, AVANT, 255);
    activer(MOTEUR_B, AVANT, 255);
}

void reculer()
//fait reculer le robot en ligne droite à vitesse max en activant les deux
moteurs à un PWM max
{
    activer(MOTEUR_A, ARRIERE, 255);
    activer(MOTEUR_B, ARRIERE, 255)
}

void tourner_a_droite() //fait tourner la roue droite vers l'avant et la roue
gauche vers l'arrière

```

```

{
    activer(MOTEUR_A, ARRIERE, 255); // le moteur gauche recule
    activer(MOTEUR_B, AVANT, 255); // le moteur droit avance
}

void tourner_a_gauche() // fait tourner la roue droite vers l'arrière et la
roue gauche vers l'avant
{
    activer(MOTEUR_A, AVANT, 255); // le moteur gauche avance
    activer(MOTEUR_B, ARRIERE, 255); // le moteur droit recule
}

```

3.4 Conception détaillée de la partie énergie

3.4.1 Pont diviseur de tension

Référence de conception : CDT_PDT

Exigences client vérifiées : EXIG_SECUR_BATT

3.4.1.1 Schéma électrique du pont diviseur de tension

Pour surveiller la tension de la batterie, nous avons recours à un pont diviseur de tension composé de deux résistances de $10\text{ k}\Omega$ chacune. La tension résultante de ce pont est ensuite convertie en une valeur analogique par un des ADC de la carte Arduino. Cette valeur sera comparée à un seuil critique de 6,7 V. Nous pouvons observer le schéma électrique du pont diviseur sur la figure 22.

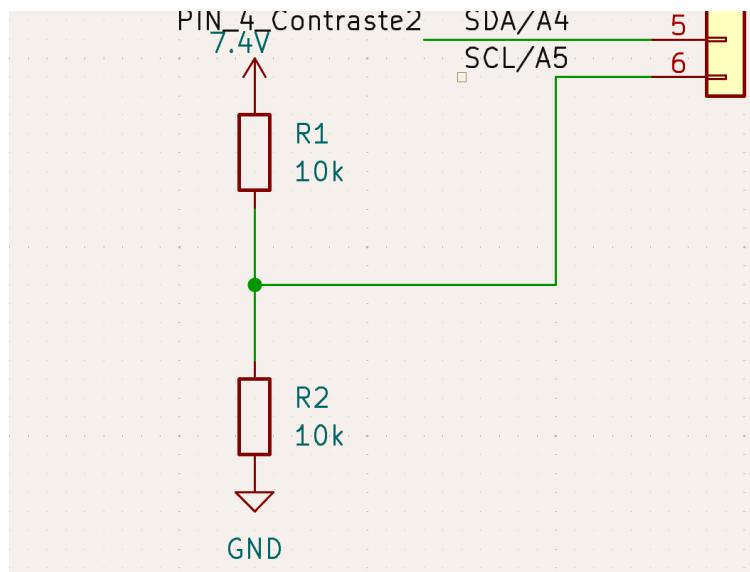


Figure 22 : Schéma du pont diviseur de tension

3.4.1.2 Code informatique du pont diviseur de tension

Étant donné que le pont diviseur fournit une tension égale à la moitié de la tension de la batterie, la comparaison s'effectuera avec la moitié du seuil, soit 3,35 V, ce qui correspond à la valeur numérique de 685 en analogique. Si cette valeur est inférieure à 685, nous allons arrêter les moteurs.

```
if (analogRead(tension_pile) < 685) { // ~3.35V  
    etatActuel = ATTENTE;  
    gererMoteurs();  
    digitalWrite(sleep, LOW);  
    return; // STOP ! On arrête la boucle ici pour ce tour.  
}
```

3.4.2 régulateur de tension

Référence de conception : CDT_REGULATEUR

Exigences client vérifiées : EXIG_CARTE

Pour réguler la tension en sortie de l'accumulateur, nous allons utiliser un régulateur de tension. En effet, on dispose déjà d'un régulateur de tension sur la carte arduino elle-même, sa référence est le AMS1117-5.0 comme on peut le voir sur la figure 23.

La régulation est essentielle car l'accumulateur LiPo 2s fournit une tension nominale de 7.4 V, alors que le microcontrôleur ATMEGA328P et les composants du bloc acquisition (capteurs) nécessitent une tension stabilisée et réduite à 5 V pour un fonctionnement optimal.

En connectant la sortie 7,4 V de la batterie à l'entrée d'alimentation (broche Vin ou prise Jack) de la carte Arduino Uno, le régulateur AMS1117-5.0 intégré sera automatiquement sollicité pour baisser et stabiliser cette tension.

Le 5 V régulé sera ensuite distribué au microcontrôleur et aux autres composants de faible puissance via les broches de sortie 5V de l'Arduino, simplifiant ainsi le circuit d'alimentation de la carte shield.

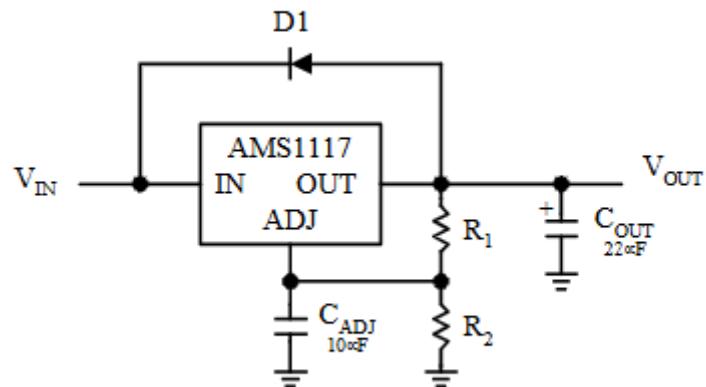


Figure 23 : Régulateur linéaire de l'arduino

3.4.2 Batterie

Référence de conception : CDT_BATTERIE

Exigences client vérifiées : EXIG_AUTONOMIE

L'énergie électrique stockée dans le mini-sumo doit permettre au robot de combattre face à un adversaire pendant au moins 15 minutes. Cela est équivalent à un déplacement en continu du d'une durée de 65 min sans aucun obstacles. Dans la conception préliminaire et grâce au FAD, nous avons choisi d'utiliser l'accumulateur LiPo Conrad Energy 7,4 V 450 mAh.

Pendant les essais, nous avons appris que les moteurs consomment plus que les autres composants et nous nous sommes basés sur cela pour calculer l'autonomie, ils consomment 330 mA en continu . Nous avons une batterie avec une capacité de 450 mAh mais, pour question de sécurité, nous n'allons utiliser que 80% de sa capacité, soit 360 mAh.

$$Autonomie = \frac{Capacité}{Consommation} = \frac{360 \text{ mAh}}{330 \text{ mA}} = 1,09 \text{ h}$$

Nous prévoyons une autonomie de 1,09 h qui équivaut à 65 min et 24 secondes.

3.5 Conception détaillée de la partie traitement

3.5.1 Le microcontrôleur ATMEGA328P-PU

Référence de conception : CDT_MICRO_CONTROLEUR

Exigences client vérifiées : EXIG_COMPORTEMENT, EXIG_DEPLACEMENT, EXIG_DEPART, EXIG_ADVERSAIRE, EXIG_SECUR_BATT

3.4.1.1 Schéma électrique du microcontrôleur

Le bloc traitement est au cœur du robot mini-sumo. Il est responsable de l'acquisition des

IUT Bordeaux Département GEII	Référence : RMS_DDC_EQ12 Révision : 1 – 06/10/2025	35/61
----------------------------------	---	-------

signaux provenant des capteurs adversaires (détection de distance), des capteurs de sol (détection des limites du dohyo), du récepteur IR (détection du signal de départ), et du pont diviseur de tension (surveillance de la batterie). Il traite ces informations pour commander les actions du robot : déplacement via le pont en H et gestion des comportements (attaque, évitement des bords, sécurité batterie). Conformément à la conception préliminaire, nous utilisons le microcontrôleur ATMEGA328P-PU intégré à la carte Arduino Uno, sur laquelle est monté notre shield personnalisé. Ce choix assure une compatibilité avec les bibliothèques Arduino tout en respectant les exigences de performance et de fiabilité. Nous pouvons observer les schéma électrique de la partie traitement sur la figure 24.

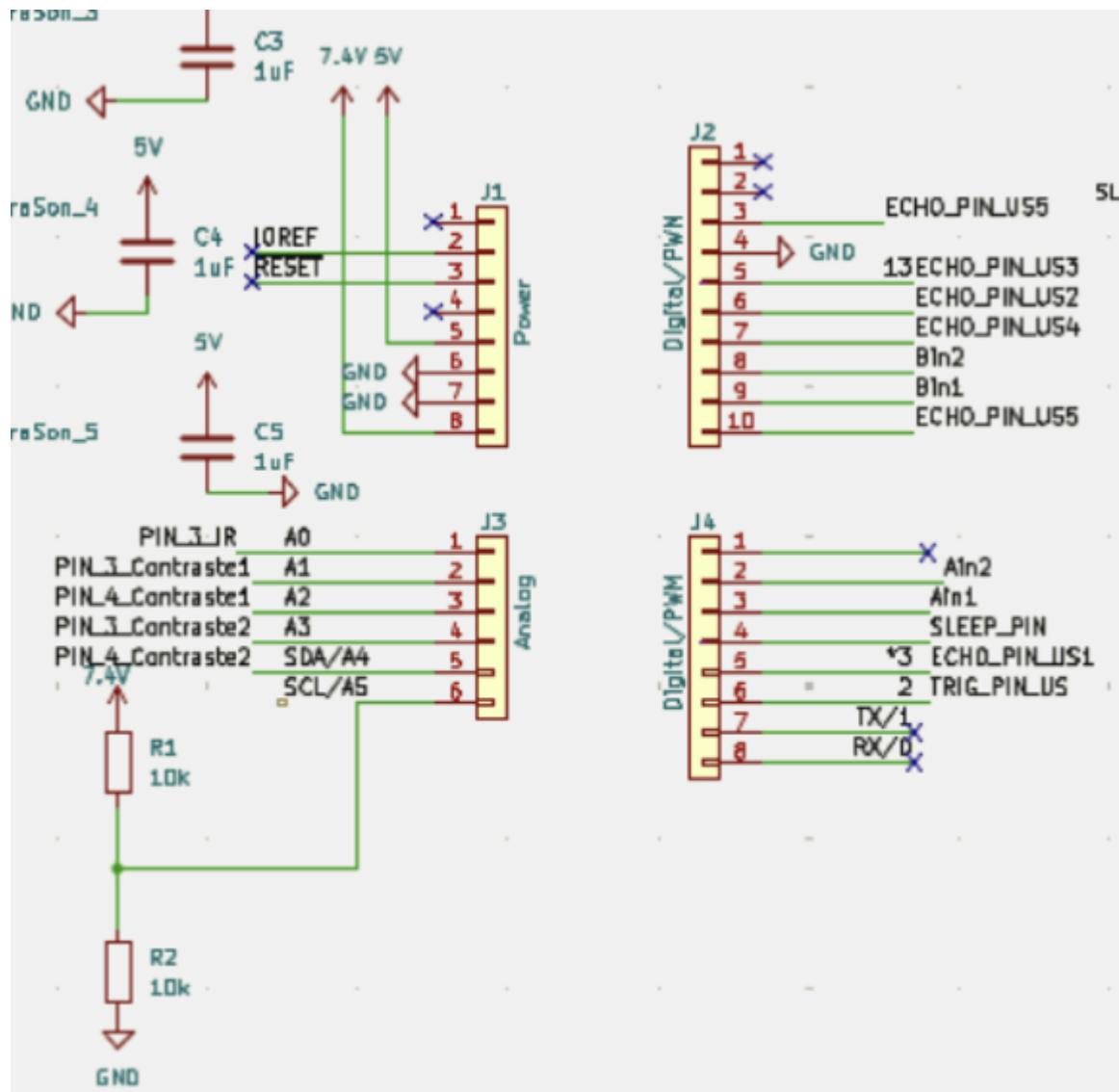


Figure 24 : Schéma électrique de la partie traitement

3.3.1.2 Code informatique du microcontrôleur

Pour plus de clarté et une meilleure compréhension, les explications ont été intégrées directement dans le code. De cette façon, les commentaires se trouvent juste à côté des éléments qu'ils éclairent.

L'optimisation des performances a été notre priorité absolue. Nous avons visé un temps d'exécution minimal tout en assurant un point crucial, une architecture non bloquante.

Contraintes techniques :

- Boucles : Usage exclusif de l'instruction loop. Aucune boucle for ou while
- Gestion mémoire : Stockage en mémoire uniquement pour les données réutilisées ; les données à usage unique sont traitées à la volée.
- Aucune utilisation de delay() et ajout de timeout pour les fonctions bloquantes comme pulseIn

```
// --- DÉFINITIONS ET VARIABLES --- //////////////

// Variables des broches
int PIN_IR = A0; //PIN infra rouge
int contraste_FR = A1; //PIN capteur de contraste avant droit
int contraste_FL = A2; //PIN capteur de contraste avant gauche
int contraste_RR = A3; //PIN capteur de contraste arrière droit
int contraste_RL = A4; //PIN capteur de contraste arrière gauche
int tension_pile = A5; //PIN relié au pont diviseur de tension
int trigPin = 2; //PIN envoyant un signal aux capteurs ultrasons pour les déclencher
int echo_FL = 3; //PIN réponse du capteur ultrason avant gauche
int sleep = 4; //PIN permettant de désactiver les moteurs
int Ain1 = 5; //PIN permettant de contrôler le pololu DRV8833 pour contrôler le moteur droit
int Ain2 = 6; //PIN permettant de contrôler le pololu DRV8833 pour contrôler le moteur droit
int echo_FR = 8; //PIN réponse du capteur ultrason avant droit
int Bin1 = 9; //PIN permettant de contrôler le pololu DRV8833 pour contrôler le moteur gauche
int Bin2 = 10; //PIN permettant de contrôler le pololu DRV8833 pour contrôler le moteur gauche
int echo_R = 11; //PIN réponse du capteur ultrason droit
int echo_B = 12; //PIN réponse du capteur ultrason arrière
```

```

int echo_L = 13; //PIN réponse du capteur ultrason gauche

// constantes permettant de contrôler les moteurs
const int MOTEUR_A = 0;
const int MOTEUR_B = 1;
const int AVANT = 0;
const int ARRIERE = 1;

// mémoire de combat, nous avons fait le choix d'allouer toute la mémoire
// dont le robot pourrait avoir besoin en global pour ne pas qu'il ai à le
// faire pendant le combat lui donnant un temps de réaction plus rapide
bool intelligence = 1; //intelligence permet au robot de connaître la
// dernière position connue de l'adversaire
float dist_Left; //ces 5 variables stockent le résultat du calcul de
// distance entre le robot et l'ennemi
float dist_FrontL;
float dist_FrontR;
float dist_Back;
float dist_Right;
//bool ligneAV_G; //ces 4 variables stockent l'état des capteurs de ligne
// cependant la lecture est effectuée une fois unique pendant loop, la stocker
// ne nous fait alors pas gagner du temps mais au contraire ajoute des
// instructions supplémentaires ralentissant l'exécution du code
//bool ligneAV_D;
//bool ligneAR_G;
//bool ligneAR_D;
unsigned long duration; //cette variable est utilisé dans la fonction
// calculant la distance de l'ennemi

// Gestion des États ---
enum EtatRobot {
    ATTENTE,           // Attente signal IR
    RECHERCHE,         // Tourne pour chercher
    POURSUITE,         // Avance vers l'ennemi
    ALIGNEMENT_G,      // Tourne vers la gauche (ennemi détecté à gauche)
    ALIGNEMENT_D,      // Tourne vers la droite (ennemi détecté à droite)
    SURVIE_RECUL_D,    // Ligne blanche derrière à droite!
}

```

```

SURVIE_RECUL_G,    // Ligne blanche derrière à gauche !
SURVIE_AVANCE_D,  // Ligne blanche devant à droite!
SURVIE_AVANCE_G,  // Ligne blanche devant à gauche!
CALIBRAGE_G,      //ennemi devant à gauche
CALIBRAGE_D,      //ennemi devant à droite
};

EstatRobot etatActuel = ATTENTE;
unsigned long tempsDebutAction = 0; // Pour remplacer les delay()

// Paramètres de temps (à ajuster selon les tests en conditions réelles)
const int DUREE_ESQUIVE = 400;      // ms
const int DUREE_DEGAGEMENT = 300;   // ms pour tourner après un recul
const int DISTANCE_ATTAQUE = 60;    // cm (seuil de détection) // potentiellement redondant avec le time out de pulseIn
const int VITESSE_VIRAGE = 100;     // Plus ce chiffre est bas, plus le virage est serré. //cette constante sert à tourner tout en avançant et devra sûrement être diviser en deux constantes différentes : une pour suivre l'ennemie, l'autre pour esquiver le bord du dohyo

// --- PROTOTYPES --- /////////////
void avancer();
void reculer();
void tournerD();
void tournerG();
void arreter_tout(); //arrête les moteurs
void activer(int moteur, int sens, int vitesse); //active les moteurs
float calculer_distance(int echoPin); //distance calculé par le capteur ultrason
bool capteur_contraste(int capteur_broche); //résultat des capteurs de contraste
void gererMoteurs(); // centralise les actions en fonction de l'état actuelle du robot
void avancer_tournerD();
void avancer_tournerG();
void reculer_tournerD();
void reculer_tournerG();

```

```

// --- SETUP --- ///////////////
void setup() {
    // Configuration des broches
    pinMode(PIN_IR, INPUT);
    pinMode(contraste_FR, INPUT);
    pinMode(contraste_FL, INPUT);
    pinMode(contraste_RR, INPUT);
    pinMode(contraste_RL, INPUT);
    pinMode(tension_pile, INPUT);
    pinMode(echo_FL, INPUT);
    pinMode(echo_FR, INPUT);
    pinMode(echo_R, INPUT);
    pinMode(echo_B, INPUT);
    pinMode(echo_L, INPUT);
    pinMode(trigPin, OUTPUT);
    pinMode(sleep, OUTPUT);
    pinMode(Ain1, OUTPUT);
    pinMode(Ain2, OUTPUT);
    pinMode(Bin1, OUTPUT);
    pinMode(Bin2, OUTPUT);

    digitalWrite(sleep, HIGH); //les moteurs peuvent être activer

    while (analogRead(PIN_IR) > 900) { // Séquence de démarrage bloquante
        (seulement au début)
        // On attend que le module IR reçoive le signal
    }

    etatActuel = RECHERCHE; //le robot commence le combat
}

// --- LOOP (Cerveau Non-Bloquant) --- ///////////////
void loop() {

    // 0. SÉCURITÉ BATTERIE
    if (analogRead(tension_pile) < 685) { // ~3.35V = tension de seuil/2
        etatActuel = ATTENTE; //tous les moteurs sont désactivé
}

```

```

gererMoteurs();

/*digitalWrite(sleep, LOW);*/ //tous les moteurs sont désactivé jusqu'au
reset du robot, cette ligne a été désactivé car si il y a une erreure
d'acquisition le robot restera bloqué jusqu'au prochain reset

return; // STOP ! On arrête la boucle ici pour ce tour.

}

// 1. LECTURE DES CAPTEURS VITIAUX (Lignes)
// On lit les lignes en priorité absolue
//ligneAV_G = capteur_contraste(contraste_FL);
//ligneAV_D = capteur_contraste(contraste_FR);
//ligneAR_G = capteur_contraste(contraste_RL);
//ligneAR_D = capteur_contraste(contraste_RR);

// 2. LOGIQUE DE TRANSITION (Décision)

// --- PRIORITÉ 1 : SURVIE (Lignes blanches) ---
if (capteur_contraste(contraste_FR)) {
    if (etatActuel != SURVIE_AVANCE_D) { //pour ne pas remettre le timer à
zéro à chaque fois
        etatActuel = SURVIE_AVANCE_D;
        tempsDebutAction = millis(); // On lance le chrono
    }
} else if (capteur_contraste(contraste_FL)) {
    if (etatActuel != SURVIE_AVANCE_G) {
        etatActuel = SURVIE_AVANCE_G;
        tempsDebutAction = millis(); // On lance le chrono
    }
} else if (capteur_contraste(contraste_RR)) {
    if (etatActuel != SURVIE_RECUL_D) {
        etatActuel = SURVIE_RECUL_D;
        tempsDebutAction = millis();
    }
} else if (capteur_contraste(contraste_RL)) {
    if (etatActuel != SURVIE_RECUL_G) {
        etatActuel = SURVIE_RECUL_G;
        tempsDebutAction = millis();
    }
}

```

```

}

// --- PRIORITÉ 2 : GESTION DES TIMERS DE SURVIE ---
// Si on est en train de reculer, on vérifie si c'est fini
else if (etatActuel == SURVIE_AVANCE_D || etatActuel == SURVIE_AVANCE_G || etatActuel == SURVIE_RECUL_D || etatActuel == SURVIE_RECUL_G) {
    if (millis() - tempsDebutAction > DUREE_ESQUIVE) {
        etatActuel = RECHERCHE; // Retour au combat
    }
}

// --- PRIORITÉ 3 : COMBAT (Si pas de ligne et pas en manœuvre de survie)
---

else {
    // Lecture des ultrasons (seulement maintenant pour ne pas ralentir la détection de ligne)
    // float dist_FL = calculer_distance(echo_FL); // Optionnel, prend du temps
    // On optimise en lisant les capteurs pertinents

    // Note: calculer_distance utilise pulseIn qui bloque un peu (quelques ms).
    // Pour être ultra réactif, on lit d'abord l'avant.

    dist_FrontL = calculer_distance(echo_FL);
    dist_FrontR = calculer_distance(echo_FR);

    // Détection Ennemi DEVANT
    // certaines conditions ont été commentées car redondantes avec le timeout pulseIn mais à vérifier avec des tests
    if ((dist_FrontL > 0 /*&& dist_FrontL < DISTANCE_ATTAQUE*/) && (dist_FrontR > 0 /*&& dist_FrontR < DISTANCE_ATTAQUE*/)) { //ennemie détecté devant
        etatActuel = POURSUITE; //on fonce droit sur lui
    } else if ((dist_FrontL > 0 /*&& dist_FrontL < DISTANCE_ATTAQUE*/) && (dist_FrontR < 0 /*|| dist_FrontR > DISTANCE_ATTAQUE*/)) { //ennemie détecté devant à gauche
        etatActuel = CALIBRAGE_G; //on fonce en s'alignant
    }
}

```

```

    } else if ((dist_FrontL < 0 /*|| dist_FrontL > DISTANCE_ATTAQUE*/) &&
(dist_FrontR > 0 /*&& dist_FrontR < DISTANCE_ATTAQUE*/)) { //ennemie détecté
devant à droite
    etatActuel = CALIBRAGE_D;
} else {
    // Détection Ennemi GAUCHE (Si rien devant)
    dist_Left = calculer_distance(echo_L);
    if (dist_Left > 0 /*&& dist_Left < DISTANCE_ATTAQUE*/) { //ennemie
détecté à gauche
        etatActuel = ALIGNEMENT_G; //on tourne à gauche
    } else {
        // Détection Ennemi DROITE (Si rien devant)
        dist_Right = calculer_distance(echo_R);
        if (dist_Right > 0 /*&& dist_Right < DISTANCE_ATTAQUE*/) { //ennemie
détecté à droite
            etatActuel = ALIGNEMENT_D; //on tourne à droite
        } else {
            // Détection Ennemi ARRIERE
            dist_Back = calculer_distance(echo_B);
            if (dist_Back > 0 /*&& dist_Back < DISTANCE_ATTAQUE*/) { //ennemie
détecté derrière
                // Choix stratégique : se retourner vite (ex: tourner droite)
                if (etatActuel == ALIGNEMENT_D) { //vérification pour garder
l'élan à vérifier en test si c'est utile
                    etatActuel = ALIGNEMENT_D;
                }
                else {
                    etatActuel = ALIGNEMENT_G;
                }
            } else {
                etatActuel = RECHERCHE; //on ne détecte pas l'ennemie donc on le
cherche
            }
        }
    }
}
// RIEN DU TOUT -> RECHERCHE

```

```

}

// 3. EXÉCUTION (Action Moteurs)
gererMoteurs();
}

// --- FONCTIONS --- //////////////

void gererMoteurs() { //gérer les moteurs en fonction de l'état du robot
    switch (etatActuel) {
        case RECHERCHE:
            if (intelligence == 1/*droite*/) { //vérifie la dernière position connue
                tournerD(); // Tourne sur lui-même pour scanner
            } else if (intelligence == 0/*gauche*/) { //vérifie la dernière position connue
                tournerG(); // Tourne sur lui-même pour scanner
            }
            break;

        case POURSUITE:
            avancer(); // Fonce sur l'adversaire
            break;

        case CALIBRAGE_G:
            avancer_tournerG(); // Fonce sur l'adversaire en ajustant la trajectoire à gauche
            intelligence = 0; //gauche //enregistre la dernière position connue
            break;

        case CALIBRAGE_D:
            avancer_tournerD(); // Fonce sur l'adversaire en ajustant la trajectoire à droite
            intelligence = 1; //droite //enregistre la dernière position connue
            break;

        case ALIGNEMENT_G:
            tournerG();
    }
}

```

```

        break;

    case ALIGNEMENT_D:
        tournerD();
        break;

    case SURVIE_RECUL_D:
        reculer_tournerG(); //reculé avec un angle de virage élevé pour
        esquiver le bord et ne pas forcer contre le robot qui peut être en train de
        nous pousser //c'est pour ça qu'il faudrait une valeur de vitesse de
        rotation différente de celle pour le calibrage pour avoir un angle beaucoup
        plus aigu
        break;

    case SURVIE_RECUL_G:
        reculer_tournerD();
        break;

    case SURVIE_AVANCE_D:
        avancer_tournerG();
        break;

    case SURVIE_AVANCE_G:
        avancer_tournerD();
        break;

    case ATTENTE:
        arreter_tout();
        break;
    }

}

// --- fonctions ---


void avancer() {
    activer(MOTEUR_A, AVANT, 255);
    activer(MOTEUR_B, AVANT, 255);
}

```

```

void reculer() {
    activer(MOTEUR_A, ARRIERE, 255);
    activer(MOTEUR_B, ARRIERE, 255);
}

void tournerD() { //tourner à droite
    activer(MOTEUR_A, ARRIERE, 255);
    activer(MOTEUR_B, AVANT, 255);
}

void tournerG() { //tourner à gauche
    activer(MOTEUR_A, AVANT, 255);
    activer(MOTEUR_B, ARRIERE, 255);
}

void avancer_tournerD() {
    // Pour aller à droite en avançant : Roue GAUCHE vite / Roue DROITE lente
    activer(MOTEUR_A, AVANT, 255);           // Moteur Gauche à fond
    activer(MOTEUR_B, AVANT, VITESSE_VIRAGE); // Moteur Droit ralenti
}

void avancer_tournerG() {
    // Pour aller à gauche en avançant : Roue GAUCHE lente / Roue DROITE vite
    activer(MOTEUR_A, AVANT, VITESSE_VIRAGE); // Moteur Gauche ralenti
    activer(MOTEUR_B, AVANT, 255);           // Moteur Droit à fond
}

// Note : En reculant, la logique est la même. //Pour que l'arrière aille à
// droite,
// il faut que la roue gauche recule plus vite.

void reculer_tournerD() {
    // L'arrière du robot part vers la droite
    activer(MOTEUR_A, ARRIERE, 255);
    activer(MOTEUR_B, ARRIERE, VITESSE_VIRAGE);
}

```

```

void reculer_tournerG() {
    // L'arrière du robot part vers la gauche
    activer(MOTEUR_A, ARRIERE, VITESSE_VIRAGE);
    activer(MOTEUR_B, ARRIERE, 255);
}

void arreter_tout() { //tous les moteurs s'arrêtent
    digitalWrite(Ain1, LOW);
    digitalWrite(Ain2, LOW);
    digitalWrite(Bin1, LOW);
    digitalWrite(Bin2, LOW);
}

void activer(int moteur, int sens, int vitesse) { //fonction bas niveau pour
communiquer avec le DRV8833
    switch (moteur) {
        case MOTEUR_A:
            if (sens == AVANT) {
                digitalWrite(Ain1, HIGH);
                analogWrite(Ain2, vitesse);
            } else {
                digitalWrite(Ain2, HIGH);
                analogWrite(Ain1, vitesse);
            }
            break;
        case MOTEUR_B:
            if (sens == AVANT) {
                digitalWrite(Bin1, HIGH);
                analogWrite(Bin2, vitesse);
            } else {
                digitalWrite(Bin2, HIGH);
                analogWrite(Bin1, vitesse);
            }
            break;
    }
}

float calculer_distance(int echoPin) { //fonction bas niveau pour détecter

```

```

l'ennemi en face des capteurs ultrasons
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    // Ajout du paramètre timeout (8000 µs max ~50/70cm)
    // Cela empêche le robot de figer s'il ne voit rien.
    duration = pulseIn(echoPin, HIGH, 8000); //redondance avec les calculs
dans loop vérifiant si la distance est inférieure à 60cm il faut vérifier si
on peut les enlever (après test)

if (duration == 0) {
    return -1; // Pas d'obstacle détecté dans la zone des 50cm
}

return (duration * 0.0343) / 2;
}

bool capteur_contraste(int capteur_broche) { //fonction bas niveau pour lire
les capteurs de contraste
    return analogRead(capteur_broche) <= 800; // Seuil blanc
}

```

Lors du développement logiciel et des nombreuses itérations du code, le PIN relié à la fonction sleep du DRV8833 s'est avéré obsolète. Ce PIN libéré aurait pu nous permettre d'intégrer un accéléromètre pour obtenir plus de données. Notre idée principale était de détecter un contact avec le robot ennemi pour, en cas d'attaque, reculer et le frapper à nouveau en boucle, ce qui serait bien plus efficace que de simplement le pousser.

Ce PIN aurait aussi pu servir à séparer les broches Trig pour éviter les interférences, principalement sur les deux capteurs avant (les autres étant à l'opposé, ils sont moins sujets aux perturbations). Enfin, nous aurions pu l'utiliser pour contrôler de petits servomoteurs déployant des drapeaux sur les côtés. Le robot adverse croirait détecter notre présence alors que non, et foncerait sur un obstacle qui ne le retiendrait pas, risquant ainsi de sortir du dohyo plus facilement.

Ces solutions pourraient être implémentées dans une V2 du shield.

3.6 Coût-Délai

Référence de conception : CDT_COUT

Exigences client vérifiées : EXIG_COUT

Le coût total de l'ensemble des composants (mécaniques et électroniques) nécessaires pour la fabrication d'un seul prototype du robot mini-sumo est inférieur à 120 € HT.

Le respect de cette exigence nécessite :

- * une budgétisation initiale du robot mini-sumo avec allocation de coût bloc par bloc
- * la réalisation d'une nomenclature détaillée et financièrement chiffrée.

Afin de répondre à l'exigence de coût, nous avons réalisé l'estimation du prix. Étant en phase de conception détaillée, nous connaissons maintenant les composants nécessaires à la réalisation de ce projet. Nous avons donc répertorié cela dans le tableau de la figure 25.

Référence	Quantité	Coût unitaire (€ TTC)	Coût total (€ TTC)	Coût final (€ TTC)
Capteur HC-SR04	5	3.8	19	107,01
YAGEO CC1206KRX7R9BB104 Condensateur céramique CMS, Série CC, 0.1 µF, ± 10%, X7R, 50 V, 1206	5	0,13	0,65	
YAGEO RC1206FR-0710KL Résistance CMS à couche épaisse, Série RC, 10 kohm, 250 mW, ± 1%, 200 V, 1206	2	0,12	0,24	
CNY70	3	0,8	2,4	
Pololu Commande de 2 moteurs CC DRV8833 2x1,2A	1	13	13	
Connecteur HE14 MH100 sécable droit 1 x 36 pts	1	0,6	0,6	
Motoréducteur GM9	2	6,9	13,8	
Roue GMPW noire pour moteurs série GM	4	4,4	17,6	
Lot de 5 câbles Grove 20 cm	1	2,95	2,95	
Connecteur Grove droit	1	0,15	0,15	
Connecteur Grove coudé	1	0,15	0,15	
connecteur vampire 4 contacts	1	1,3	1,3	
connecteur vampire 3 contacts	1	1,8	1,8	
connecteur vampire 2 contacts	1	1,5	1,5	
Interrupteur à bascule Noir Unipolaire à une direction (1NO) Verrouillable	1	3,79	3,79	
10 entrées 5mm	1	0,8	0,8	
10 vis M3 20mm	1	4,29	4,29	
10 vis M3 30mm	1	4,29	4,29	
10 vis auto-taraudeuses 2,9x6,5mm	1	3,07	3,07	
10 vis auto-taraudeuses 2,9x9mm	1	3,07	3,07	
10 vis auto-taraudeuses 2,9x13mm	1	3,07	3,07	
Packs d'accus Lipo 7,4 V 450 mAh	1	9,49	9,49	

Figure 25 : Tableau des coûts

L'estimation du prix montre que nous sommes actuellement à 107.01 euros TTC, ce qui nous laisse ainsi une marge de moins de 12.99 euros.

Référence de conception : CDT_DELAIS

Exigences client vérifiées : EXIG_COUT

Le projet de développement du robot sumo respecte l'exigence des 84 heures allouées, grâce à une gestion efficace du temps et du suivi du planning de développement de la figure 26.

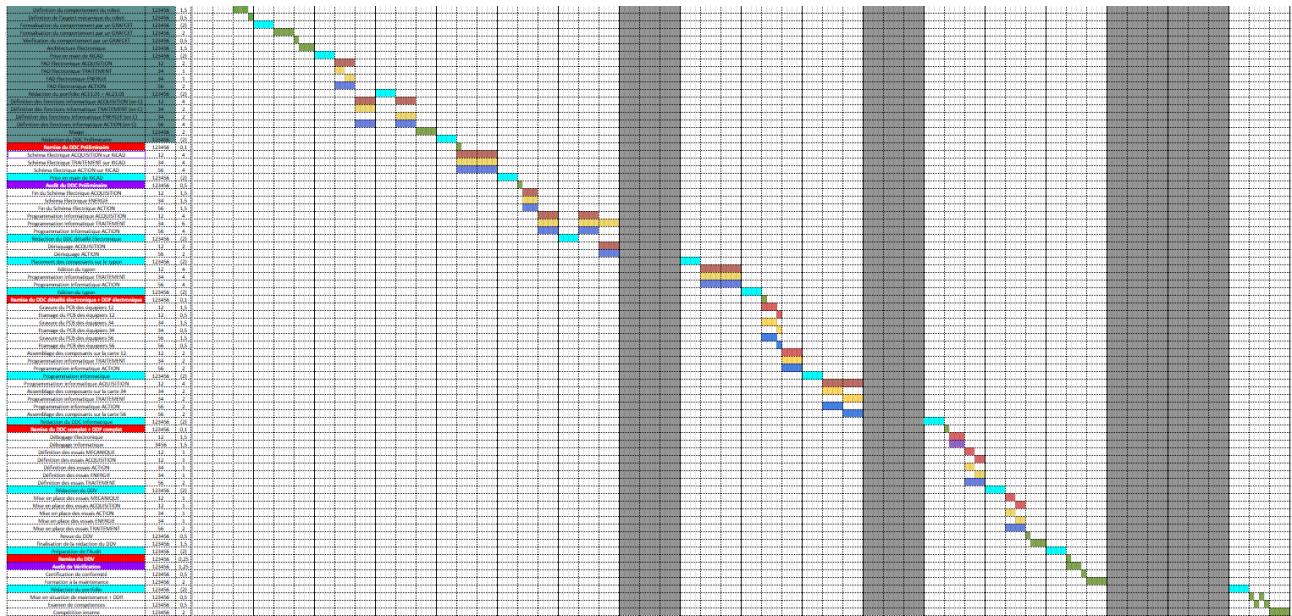


Figure 26 : Planning de développement

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 Simulation des capteurs à ultrasons

Référence de la simulation : SIM_CAPTEUR_ADV

Exigences client vérifiées : EXIG_ADVERSAIRE

But de l'essai : Le capteur doit détecter un objet devant lui à une distance minimum de 40 cm.

Fichiers : Voici le code que nous avons utilisé : [CODE_SIM_CAPTEUR_ADV](#).

IUT Bordeaux Département GEII	Référence : RMS_DDC_EQ12 Révision : 1 – 06/10/2025	50/61
----------------------------------	---	-------

Procédure de simulation :

Nous allons relier un capteur ultrason HC-SR04 avec une carte arduino via une breadboard comme nous pouvons le voir sur la figure 27.

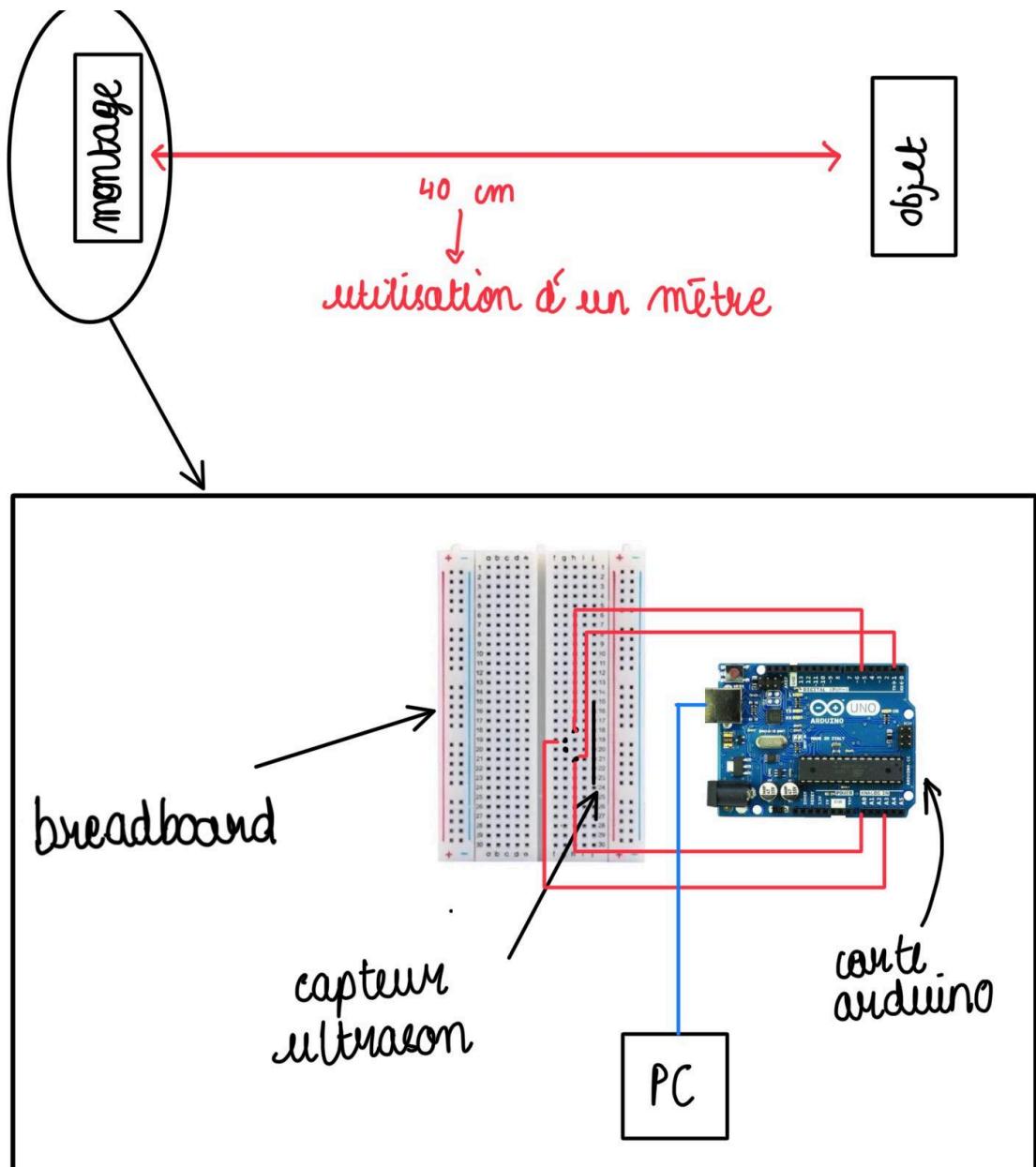


Figure 27 : Schéma de câblage du test des capteurs à ultrason

La carte arduino est reliée à un ordinateur sur lequel se situe le code. Nous allons placer un objet quelconque à une distance de 40 cm et nous allons regarder ce que nous renvoie la carte arduino sur l'ordinateur. Si le capteur détecte un objet, la carte arduino renvoie la distance en question donc ici ça sera 40 cm et s'il ne détecte rien il renvoie -1.

Résultats attendus :

Si le capteur détecte un objet, la carte arduino renvoie la distance en question donc ici ça sera 40 cm et s'il ne détecte rien il renvoie -1.

Grandeur	Valeur attendue
Si il ya détection	Distance: 40
Si il n'y a pas de détection	-1

Résultats obtenus :

Nous avons donc réalisé la simulation et avons obtenu les résultats des figures 28 et 29.

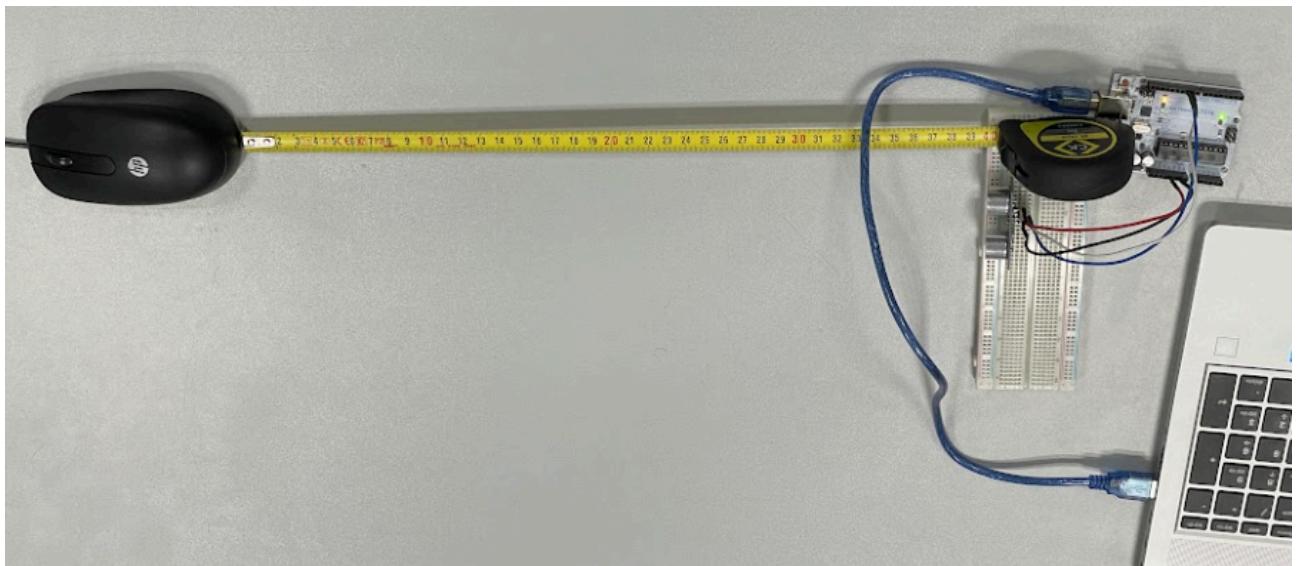


Figure 28 : Simulation du capteur ultrason

The screenshot shows the Arduino IDE interface. The top part displays the code for `ultrason_7_novembre.ino`. The code defines a function `calculer_distance` that measures distance using pulseIn, and a `setup` function that initializes pins 9 and 10. The bottom part shows the `Serial Monitor` window with the title "Output" and "Serial Monitor". It contains a message input field with "Message (Enter to send message to 'Arduino Uno' on 'COM3')". Below the message field, the serial output shows repeated distance measurements: "Distance: 42.24", "Distance: 42.14", "Distance: 42.22", "Distance: 42.24", "Distance: 42.14", "Distance: 42.24", "Distance: 42.24", "Distance: 42.14", "Distance: 42.12", "Distance: 42.22", and "Distance: 42.24".

```
ultrason_7_novembre.ino
1 float calculer_distance(int trigPin, int echoPin) {
2     float distance;
3     unsigned long duration;
4     digitalWrite(trigPin, LOW);
5     delayMicroseconds(2);
6     digitalWrite(trigPin, HIGH);
7     delayMicroseconds(10);
8     digitalWrite(trigPin, LOW);
9
10    duration = pulseIn(echoPin, HIGH);
11    if (duration == 0 || duration > 5250 ) {
12        //5250us correspond à 90cm
13        return -1; //
14    }
15    distance = (duration * 0.0343) / 2;
16    return distance;
17 }
18 void setup() {
19     pinMode(9, OUTPUT);
20     pinMode(10, INPUT);
21     Serial.begin(9600);
22 }
```

Output Serial Monitor X

Message (Enter to send message to 'Arduino Uno' on 'COM3')

Distance: 42.24
Distance: 42.14
Distance: 42.22
Distance: 42.24
Distance: 42.14
Distance: 42.24
Distance: 42.24
Distance: 42.14
Distance: 42.12
Distance: 42.22
Distance: 42.24

Figure 29 : Ecran du PC suite à la simulation

Comme nous pouvons le voir sur les figures X et X, la capteur a bien détecté la résistance de l'objet car la carte arduino renvoi la distance en question qui est ici de 40.24 cm. De plus, cette distance est supérieure à 40 cm donc nous répondons bien à l'exigence EXIG_AVERSAIRE.

Grandeur	Valeur mesurée	Conf/Non conf.
Détection du signal	Distance: 42.24	Conforme

Statut de l'essai : Conforme

Problèmes rencontrés : Aucun problème n'a été rencontré

4.2 Simulation de pont en H

Référence de la simulation : SIM_PONT_H

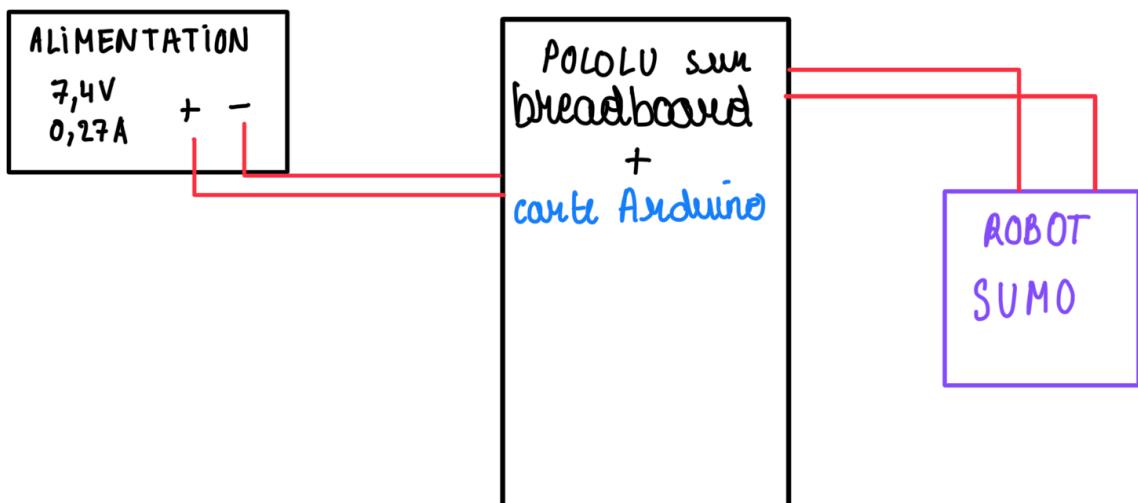
Exigences client vérifiées : EXIG_DEPLACEMENT, EXIG_COMPORTEMENT

But de l'essai : Le robot doit être capable d'avancer, de tourner, de tourner à gauche et à droite.

Fichiers : Pour cette simulation, aucun code informatique n'est nécessaire.

Procédure de simulation :

Nous allons relier le pololu DRV8833 avec le robot sumo via une breadboard. Le pololu sera alimenté en 7.4 V via une alimentation de table ce qui correspond à la tension nominale de notre accumulateur LiPo 2s. Nous pouvons observer tout cela sur la figure 30 .



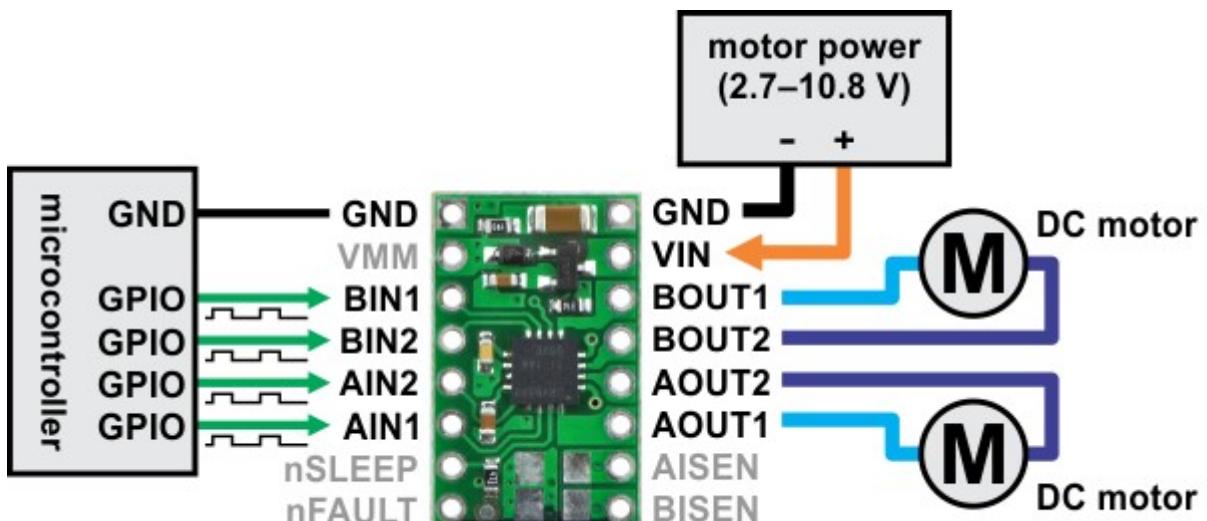


Figure 30 : Schéma de câblage du test du pont en H

Résultats attendus :

Lorsque le Pololu est alimenté, que BIN1 et AIN1 reçoivent un échelon 5V et qu'en même temps AIN2 et BIN2 reçoivent une PWM, quel que soit le rapport cyclique, les roues du robot doivent être actionnées.

Grandeur	Valeur attendue
Actionnement des roues si le pololu est alimenté	les roues tournent

Résultats obtenus :

Nous avons réalisé la simulation et nous pouvons observer le résultat sur la figure 31.

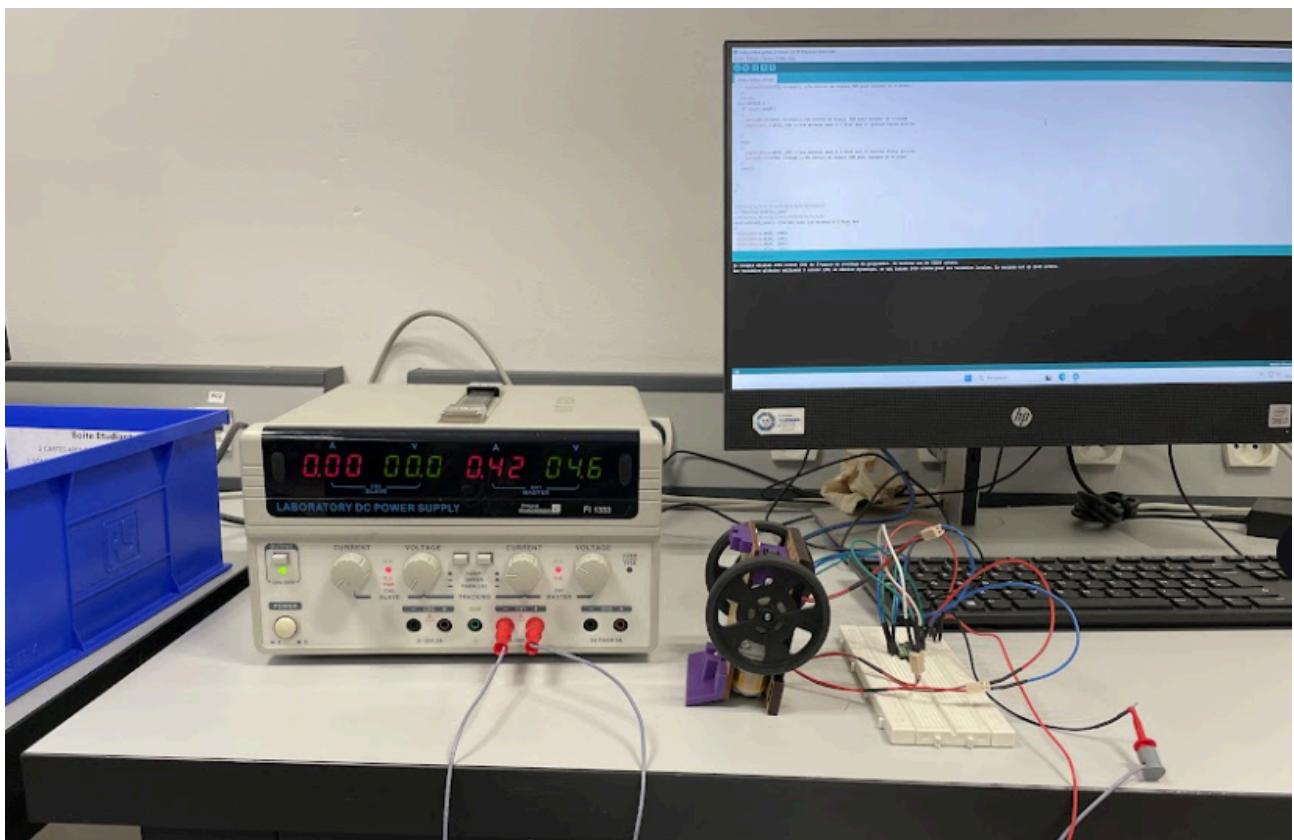


Figure 31 : Simulation du pont en H

Nous pouvons pas voir sur la figure 31 que les roues tournent, mais non avons réalisé une vidéo lors de l'essai afin de voir que les roues sont bien en action. Vous pouvez y accéder en suivant le lien suivant : [SIM_PONT_H](#). Nous obtenons donc les résultats suivants :

Grandeur	Valeur mesurée	Conf/Non conf.
Actionnement des roues si le pololu est alimenté	les roues tournent	Conforme

Statut de l'essai : Conforme

Problèmes rencontrés : Aucun problème rencontré

4.3 Simulation du départ

Référence de la simulation : SIM_DEPART

Exigences client vérifiées : EXIG_DEPART

IUT Bordeaux Département GEII	Référence : RMS_DDC_EQ12 Révision : 1 – 06/10/2025	56/61
----------------------------------	---	-------

But de l'essai : Le robot doit démarrer dès qu'une touche de la télécommande infrarouge IRC01 est appuyée.

Fichiers : Voici le code que nous avons utilisé : [CODE SIM DEPART](#)

Procédure de simulation :

Nous avons relié le pololu DRV8833, le récepteur infrarouge IR ST027, la carte arduino et le robot sumo via une breadboard. La carte arduino est reliée à un ordinateur qui lui compile le code. Le pololu est alimenté en 7.4 V via une alimentation de table. Ce qui correspond à la tension nominale de notre accumulateur LiPo 2s. Nous utiliserons aussi une télécommande infrarouge IRC01. Nous pouvons observer tout cela sur la figure 32.

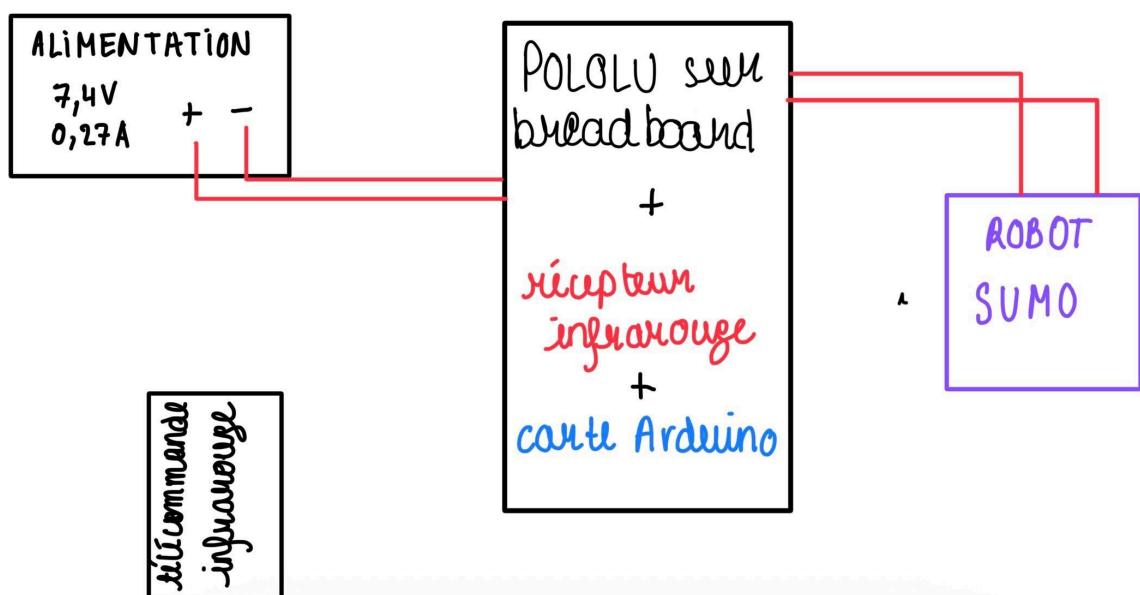


Figure 32 : Schéma de câblage de la simulation de l'exigence départ

Résultats attendus :

Les roues du robot sont immobiles. Lorsque qu'un bouton de la télécommande infrarouge est actionné, les roues du robot sumo doivent tourner.

Grandeur	Valeur attendue
Bouton de la télécommande infrarouge appuyé	les roues du robot tournent
Bouton de la télécommande infrarouge non appuyé	les roues sont immobiles

Résultats obtenus :

IUT Bordeaux Département GEII	Référence : RMS_DDC_EQ12 Révision : 1 – 06/10/2025	57/61
----------------------------------	---	-------

Nous avons réalisé l'essai. Au départ, le pololu est alimenté et le robot est immobile. Nous observons bien que lorsqu'un bouton de la télécommande infrarouge est appuyé, les roues du robot s'actionnent. Pour pouvoir observer ces résultats, nous avons réalisé une vidéo de la simulation, il suffit d'aller sur le lien suivant pour la consulter : [IMG_7039.MOV](#).

Ainsi, les résultats sont les suivants :

Grandeur	Valeur mesurée	Conf/Non conf.
Bouton de la télécommande infrarouge appuyé	les roues du robot tournent	Conforme
Bouton de la télécommande infrarouge non appuyé	les roues sont immobiles	Conforme

Statut de l'essai : Conforme

Problèmes rencontrés : Aucun problème n'a été rencontré

4.4 Simulation des capteurs de sol

Référence de la simulation : SIM_CAPTEUR_SOL

Exigences client vérifiées : EXIG_COMPORTEMENT

But de l'essai : Le capteur de sol doit détecter si le robot est au centre du dohyo (zone noire) ou au bord de doyo (zone blanche) et doit soit reculer si il est sur les bords ou continuer d'avancer si il est au centre du dohyo.

Fichiers : Voici le code que nous avons utilisé : [CODE SIM CAPTEUR SOL](#).

Procédure de simulation :

Nous avons relié un capteur de contraste CNY70, le pololu DRV8833 et la carte arduino via une breadboard. La carte arduino est connectée à un ordinateur qui compile le code. Le pololu est alimenté grâce à une alimentation de table en 7.4 V ce qui correspond à la tension nominale de notre accumulateur LiPo 2s. Il est connecté au robot. Nous pouvons observer tout cela sur la figure 33.

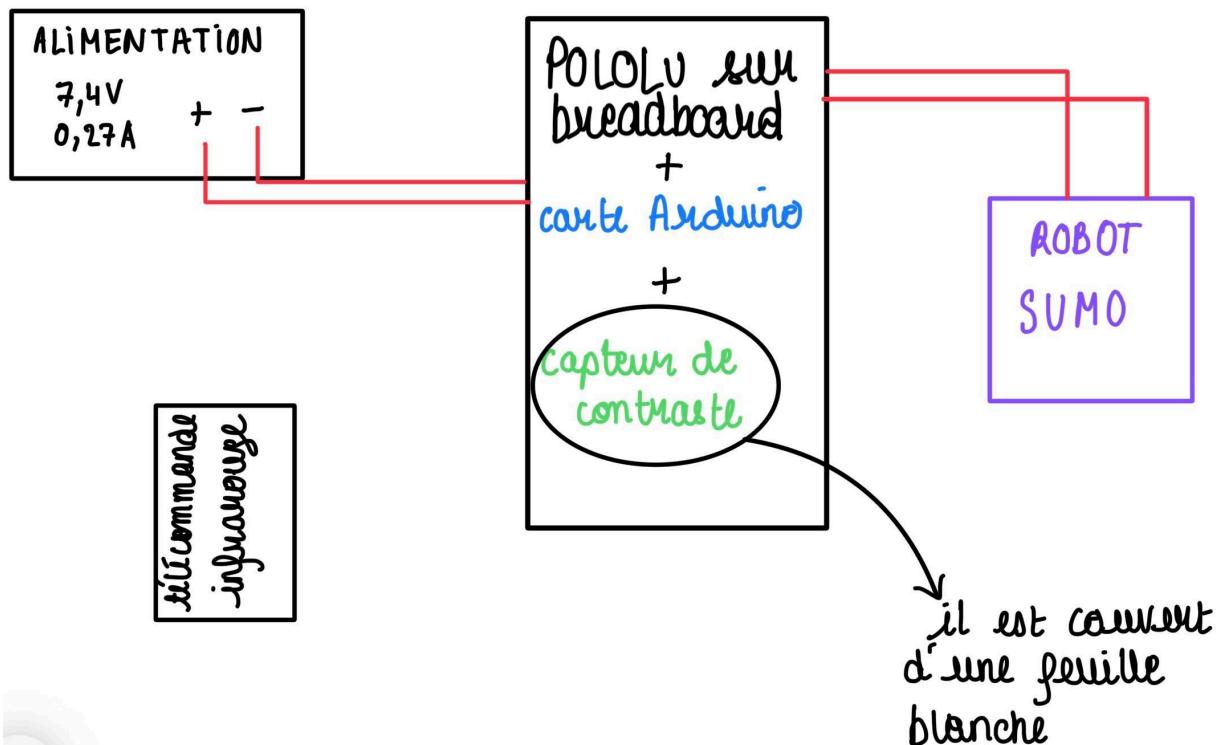


Figure 33 : Schéma de câblage de la simulation des capteurs de contraste

Nous allons donc alimenter le pololu et placer une feuille blanche sur le capteur de contraste.

Résultats attendus :

Lorsque la feuille sera placée, les roues du robot devront changer de sens donc reculer et lorsque la feuille sera enlevée elles rechargeront de sens pour avancer. Nous devons donc obtenir les résultats suivant :

Grandeur	Valeur attendue
Valeur renvoyée si la surface est blanche	le robot recul : changement de sens et donc recul
Valeur renvoyée si la surface est noire	le robot avance

Ainsi si la surface est blanche, le robot doit reculer, si la surface est noire, il doit avancer.

Résultats obtenus :

Nous avons donc réalisé la simulation et nous avons obtenu que le robot change bien de sens si on place la feuille blanche sur le capteur de contraste. On peut observer ce comportement en suivant le lien suivant qui renvoie à la vidéo filmée lors de la simulation : [SIM CAPTEUR SOL](#).

Les résultats sont donc les suivants :

Grandeur	Valeur mesurée	Conf/Non conf.
Valeur renvoyée si la surface est blanche	le robot change de sens : il recule	Conforme
Valeur renvoyée si la surface est noire	le robot continu d'avancer	Conforme

Statut de l'essai : Conforme

Problèmes rencontrés : Aucun problème n'a été rencontré

4.5 Conclusion de la simulation / prototypage rapide du produit

Nous avons donc simulé la conformité des capteurs, du polulu et de la télécommande infrarouge. Tous les essais ont été réalisés avec succès et nous n'avons rencontré aucun problème. Ainsi, les exigences EXIG_ADVERSAIRE, EXIG_DEPLACEMENT, EXIG_COMPORTEMENT, EXIG_DEPART et EXIG_COMPORTEMENT sont donc toutes conformes.

5. Conclusion de la conception du produit

La phase de conception détaillée confirme la pertinence des choix effectués lors de la conception préliminaire. Les exigences du cahier des charges ont globalement été respectées sans nécessiter de modifications majeures.

Seule la partie acquisition a fait l'objet d'un léger ajustement, visant à simplifier le montage et à réduire les coûts en évitant l'ajout d'une résistance supplémentaire. La partie traitement, quant à elle, est restée strictement identique entre les deux phases, preuve de la robustesse des solutions retenues.

Les éléments liés à la partie action n'ont subi aucun changement, tandis que la partie énergie n'a été modifiée qu'au niveau du dimensionnement de la batterie LiPo, afin de garantir le respect des contraintes d'autonomie et de coût.

Dans l'ensemble, la conception détaillée valide pleinement l'ensemble des solutions techniques retenues et confirme la conformité du produit aux exigences fixées.

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.

IUT Bordeaux Département GEII	Référence : RMS_DDC_EQ12 Révision : 1 – 06/10/2025	60/61
----------------------------------	---	-------

Exigence	Méthodes Vérification	Eléments vérifiant l'exigence	Statut
EXIG_CHASSIS_DIMENSIONS	Vérification	non renseigné	Conf.
EXIG_MASSE	Vérification	non renseigné	Conf
EXIG_INTEGRITE_MECA	Vérification	non renseigné	Conf
EXIG_FIXATION_CARTE	Conception Conception/Fab Vérification	FAB04	Conf
EXIG_AUTONOMIE	Conception Conception/Fab Dérisque Vérification	CDT_BATTERIE	Conf
EXIG_SECUR_BATT	Conception Conception/Fab Vérification	CDT_PDT	Conf
EXIG_ADVERSAIRE	Conception Dérisque Conception/Fab Vérification	CDT_CAPTEUR_ADV SIM_CAPTEUR_ADV	Conf
EXIG_LUMINOSITE	Conception Conception/Fab Vérification	CDT_CAPTEUR_SOLµ SIM_CAPTEUR_SOL	Conf
EXIG_DEPART	Conception. Vérification	CDT_RECPTEUR SIM_DEPART	Conf
EXIG_DEPLACEMENT	Conception Dérisque Vérification	CDT_MICROCONTRÔLEUR	Conf
EXIG_COMPORTEMENT	Conception Dérisque Vérification	CDT_PONT_H SIM_PONT_H SIM_CAPTEUR_SOL	Conf