



OPTION ROBOTIQUE

~ RAPPORT DE PROJET

RÉNOVATION D'UNE MAQUETTE D'HÉLICOPTÈRE 3DOF



Encadrant :
Mathieu COURCELLE

Etudiants :
Matteo IOVINO
Hugo STALIN

Durée du projet :
25 Octobre 2017 ~ 30 Mars 2018

Table des matières

Table des figures	II
Liste des tableaux	III
1 Introduction	1
2 Paramétrisation de la maquette	2
3 Réhabilitation du circuit	3
4 Résumé des modifications apportées à la maquette et résolution de problèmes	4
5 Intégration des circuits à la maquette	8
6 Modèle dynamique simplifié du robot	10
7 Synthèse de la loi de commande	12
8 Simulations	15
9 Conclusions et travaux futures	18
Annexes	I
A Liste des cartes Arduino	I
B Code C de la loi proportionnelle/dérivée	II
C Datasheet du moteur	X
D Datasheet du codeur	XI
E Spécifications de l'hélice <i>Graupner 1315.20.15</i>	XX

Table des figures

1	Maquette d'hélicoptère Quanser.	1
2	Module de puissance original de la maquette d'hélicoptère Quanser.	1
3	Schéma de la maquette avec détail des axes de rotations.	2
4	Schéma du circuit électrique imprimé de la maquette.	3
5	Circuits imprimés de la maquette : carte de la base (vers les connecteurs, (a)) et de la partie tournante (vers les moteurs et codeurs, (b)).	3
6	Schéma du circuit Arduino (les deux cartes sont connectées en les superposant).	5
7	Schéma final du circuit Arduino (les deux cartes sont connectées en les superposant).	5
8	Circuit de test de codeurs avec la carte Arduino Mega.	6
9	Étude à l'oscilloscope du problème d'interférence.	6
10	Carte originale de la maquette.	8
11	Soutien de la maquette avec le <i>Prototype Shield</i> et Relais.	8
12	Maquette intégrée du <i>Motor Shield</i> et de l'Arduino.	9
13	Ajout du ventilateur.	9
14	Schéma expérimental de la maquette.	10
15	Schéma du modèle simplifié et vues depuis les axes.	11
16	Hypothèses initiales.	12
17	Schéma blocs de l'auto-pilote.	13
18	Hélice de la maquette et capteur de vitesse.	14
19	Construction des courbes $\omega(x)$ par interpolation.	14
20	Schéma Simulink implémentant le modèle simplifié.	15
21	Schéma Simulink implémentant le modèle réel.	15
22	Graphes du modèle simplifié.	16
23	Graphes du modèle réel.	17
24	Cartes et <i>shields</i> utilisés.	I
25	Hélice <i>Graupner</i> utilisée sur la maquette.	XX

Liste des tableaux

1	Paramétrisation de l'hélicoptère à 3 ddl.	2
2	Spécifications techniques du moteur PITTMAN AMETEK TIP.	2
3	Paramétrisation du modèle simplifié.	10
4	Spécifiques de l'hélice <i>Graupner</i> .	XX

1 Introduction

Dans le cadre de l'option disciplinaire Robotique, nous avons été amenés à réaliser un projet d'une durée d'environ 80 heures, regroupé en binômes.

Le sujet que nous avons choisi, à savoir la “*Rénovation d'une maquette d'hélicoptère 3DOF*” proposé par notre encadrant COURCELLE Mathieu, a pour objectif de faire fonctionner la maquette avec une carte arduino à la place des modules de puissance (Figure 2) et de l'ordinateur la contrôlant auparavant. Dans un deuxième temps, nous devions entreprendre la synthèse d'une loi de commande afin de permettre à un hélicoptère de déposer une charge précisément sur un point donné tout en évitant le ballottement de la charge qu'il transporte.



FIGURE 1 – Maquette d'hélicoptère Quanser.

Pour cela nous avons commencé par automatiser la maquette de hélicoptère bi-moteur (Figure 1) à 3 degrés de liberté (dans la suite ddl) qui pivote sur un axe et à définir la loi de commande le stabilisant.



(a) Module de puissance Quanser.



(b) Détail du module.

FIGURE 2 – Module de puissance original de la maquette d'hélicoptère Quanser.

La nécessité de trouver des solutions embarquées pour la génération de puissance, notamment à l'aide des cartes électriques comme Arduino ou Raspberry, répond aussi au besoin de pouvoir y intégrer tout le code visant la génération de la loi de commande et l'acquisition des données depuis les capteurs.

2 Paramétrisation de la maquette

En ayant été achetée pour des fins pédagogiques, la maquette dispose d'un manuel d'utilisation dont nous allons reporter, dans cette section, les paramètres de l'hélicoptère (Tableau 1).

<i>Symbol</i>	<i>Description</i>	<i>Valeur</i>	<i>Unité de Mesure</i>
R_m	Résistance d'armature du moteur	0,83	Ω
K_t	Constante courant-couple du moteur	0,0182	N.m/A
J_m	Moment d'inertie du moteur	$1,91 \cdot 10^{-6}$	$\text{kg} \cdot \text{m}^2$
M_h	Masse de l'hélicoptère	1,426	kg
M_w	Masse du contrepoids	1,87	kg
M_f	Masse de l'hélice avant	$M_h/2$	kg
M_b	Masse de l'hélice arrière	$M_h/2$	kg
L_a	Distance entre l'axe <i>travel</i> et le corps de l'hélicoptère	0,660	m
L_h	Distance entre l'axe <i>pitch</i> et les moteurs	0,178	m
L_w	Distance entre l'axe <i>travel</i> et le contrepoids	0,470	m
g	Constante gravitationnelle	9,81	m/s^2
$K_{EC,LN,T}$	Résolution du codeur <i>travel</i>	8192	counts/rev
$K_{EC,LN,P}$	Résolution du codeur <i>pitch</i>	4096	counts/rev
$K_{EC,LN,E}$	Résolution du codeur <i>elevation</i>	4096	counts/rev
$K_{EC,T}$	Gain de calibration du codeur <i>travel</i>	$7,67 \cdot 10^{-4}$	rad/counts
$K_{EC,P}$	Gain de calibration du codeur <i>pitch</i>	$1,50 \cdot 10^{-3}$	rad/counts
$K_{EC,E}$	Gain de calibration du codeur <i>elevation</i>	$-1,50 \cdot 10^{-3}$	rad/counts

TABLE 1 – Paramétrisation de l'hélicoptère à 3 ddl.

Les axes, *travel*, *pitch*, *élévation*, autours desquelles la maquette peut pivoter, sont mis en évidence en Figure 3.

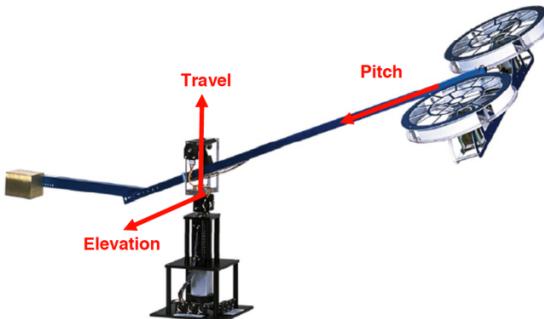


FIGURE 3 – Schéma de la maquette avec détail des axes de rotations.

Enfin nous avons récupéré les informations relatives aux moteurs (Tableau 2) dans le catalogue des moteurs électriques fournis par PITTMAN AMETEK TIP.

<i>Grandeur</i>	<i>Valeur</i>	<i>Unité de Mesure</i>
Couple Maximal	6,1	N.m
Tension d'alimentation	12	V
Vitesse	6151	rpm
Diamètre de la tige	3,97	mm
Longueur	61	mm
Largeur (Diamètre)	40,13	kg
Courant nominal	330	mA

TABLE 2 – Spécifications techniques du moteur PITTMAN AMETEK TIP.

3 Réhabilitation du circuit

Afin de bien comprendre le circuit pour le relier à l'arduino et trouver le moyen d'ajouter l'électro-aimant, nous avons démonté la maquette lors de la première séance. En particulier le travail a été d'identifier chaque câble afin d'identifier lesquels servent à alimenter ou à échanger des informations. Nous avons aussi vérifié qu'il restait des câbles libres dans le connecteur à balais (représenté par le rectangle central) qui seraient disponibles pour l'électro-aimant ou d'autres fonctions. Pour ce faire nous avons analysé et schématisé le circuit imprimé sur la base de la maquette et celui juste sous le bras (après le connecteur à balais, resp. "arm" et "base" en Figure 4).

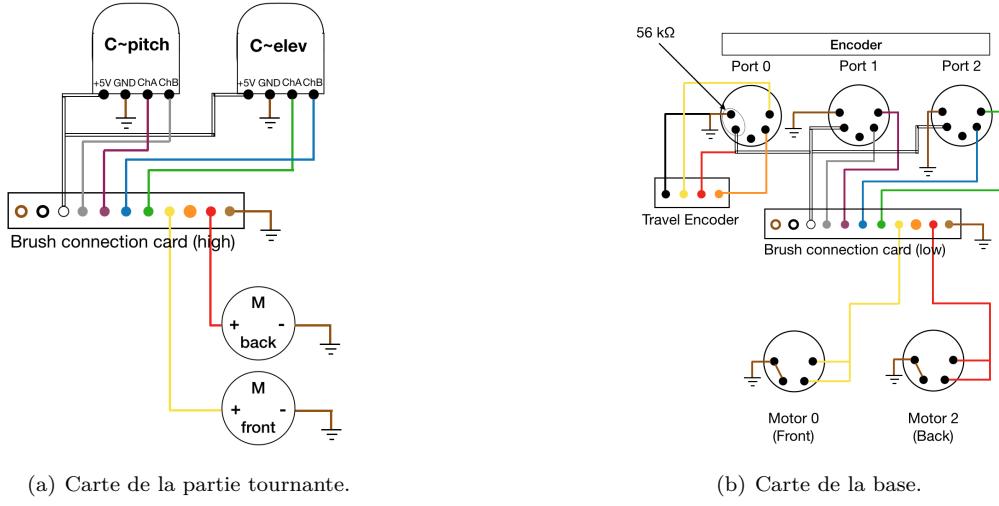


FIGURE 4 – Schéma du circuit électrique imprimé de la maquette.

Comme l'on peut voir en Figure 4, le circuit (a) est celui de la partie tournante de la maquette (entre le connecteur à balais et les différents moteurs et capteurs) tandis que le circuit (b) est celui de la base, entre le connecteur à balais et les blocs d'alimentation et de contrôle. Les deux circuits communiquent grâce à ce même connecteur, selon le code-couleur montré en Figure 4.

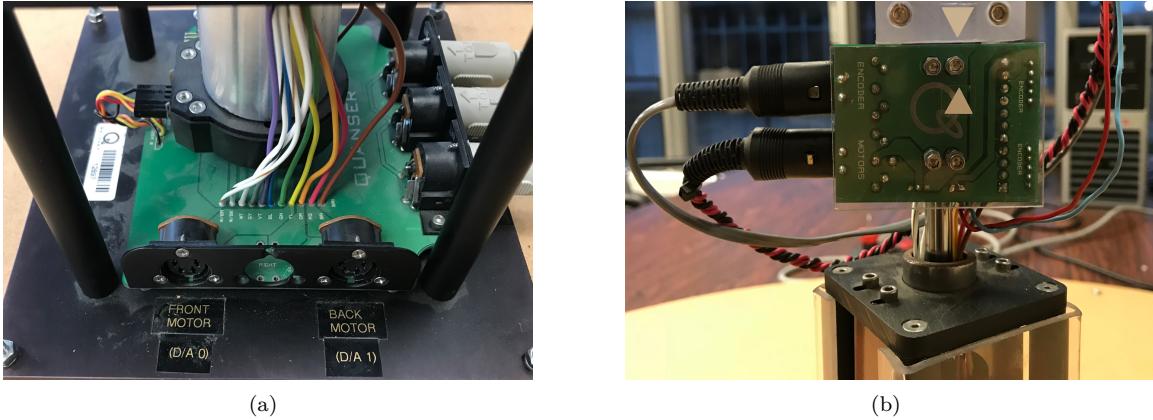


FIGURE 5 – Circuits imprimés de la maquette : carte de la base (vers les connecteurs, (a)) et de la partie tournante (vers les moteurs et codeurs, (b)).

4 Résumé des modifications apportées à la maquette et résolution de problèmes

En partant du circuit initial, nous avons d'abord testé les codeurs incrémentaux, en les reliant à une carte Arduino. Nous avons vérifié qu'ils étaient fonctionnels et fournissaient les bonnes informations. Le code utilisé pour un codeur se base sur une routine d'interruption qui incrémente un compteur en fonction du sens de variation de l'angle à chaque variation de la valeur de la piste 1 du codeur (paramètre CHANGE dans la fonction attachInterrupt) :

```
1 attachInterrupt(digitalPinToInterrupt(codeurPitchPinA),  
    GestionInterruptionCodeurPitchPinA, CHANGE);  
  
void GestionInterruptionCodeurPitchPinA() {  
    // Routine de service d'interruption attachée à la voie A du codeur incrémental  
    // du pitch  
    // On utilise la fonction digitalReadFast2 de la librairie digitalWriteFast  
6     // car les lectures doivent être très rapides pour passer le moins de temps  
     // possible  
    // dans cette fonction (appelée de nombreuses fois, à chaque impulsion de codeur  
    // )  
    tpitch=millis();  
    dtpitch=tpitch-tpitchm1;  
  
11 if (digitalReadFast2(codeurPitchPinA) == digitalReadFast2(codeurPitchPinB)) {  
    ticksPitch++;  
}  
else {  
    ticksPitch--;  
16 }  
  
if (dtpitch>0){  
    pitchi=(ticksPitch%2048)*360/2048.;  
    dpitchi=(ticksPitch-ticksPitchm1)/dtpitch*360./2048.*1000;  
21 ddpitchi=(dpitchi-dpitchim1)/dtpitch*1000;  
}  
  
tpitchm1=tpitch;  
dpitchim1=dpitchi;  
26 ticksPitchm1=ticksPitch;  
}
```

Nous avons ensuite testé les moteurs DC; pour les commander nous utilisons un Arduino Motorshield R3 (Figure 24c en Annexe A) qui permet un contrôle facile de ce type de moteurs. Nous avons découpé le circuit de contrôle et celui de puissance en retirant la pâte 12V du Shield et en l'alimentant avec une alimentation externe (Figure 6).

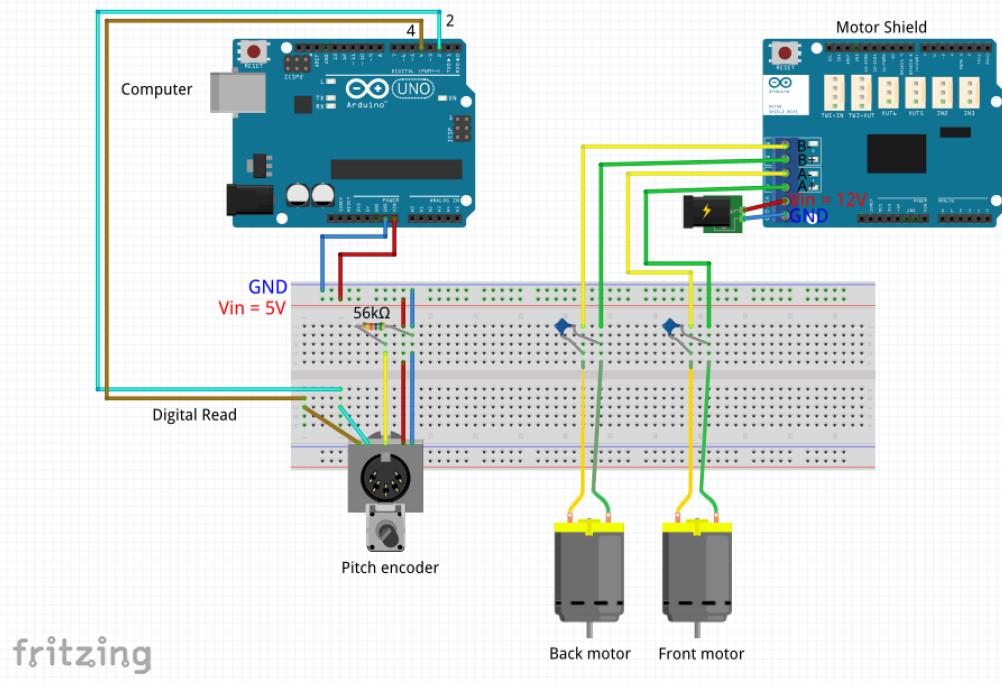


FIGURE 6 – Schéma du circuit Arduino (les deux cartes sont connectées en les superposant).

A ce point du projet, contrôler les deux moteurs, tout comme prendre l'information des trois codeurs nous était aisément (nous avons pour cela choisi une carte Arduino MEGA, Figure 24b en Annexe A, disposant de 6 timers, comme 3 sont nécessaires, un pour chaque codeur). Mais lorsque nous avons réalisé le montage global (Figure 7), nous avons rencontré quelques problèmes.

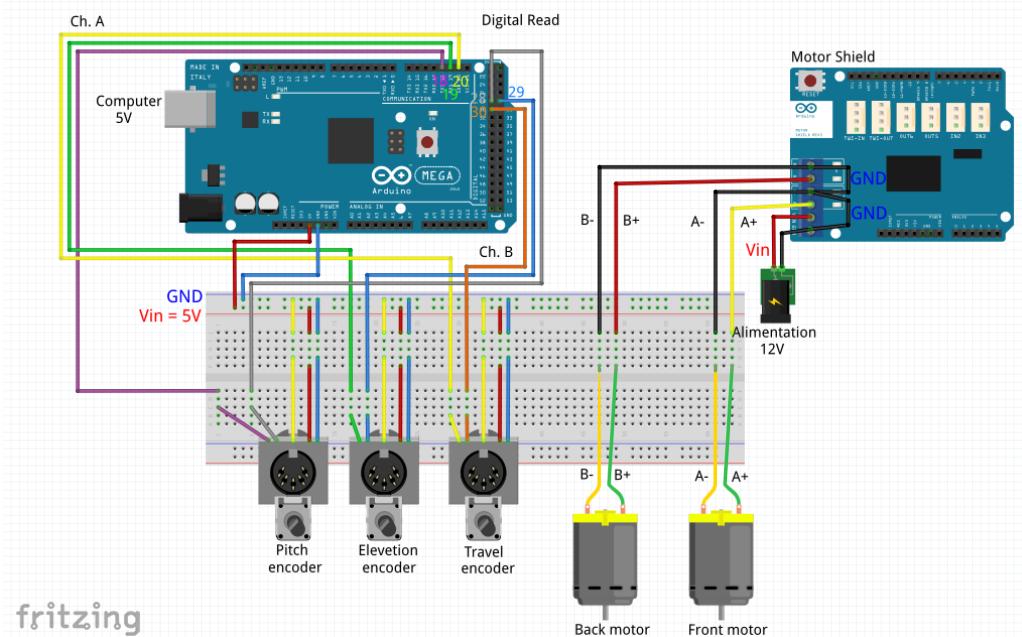


FIGURE 7 – Schéma final du circuit Arduino (les deux cartes sont connectées en les superposant).

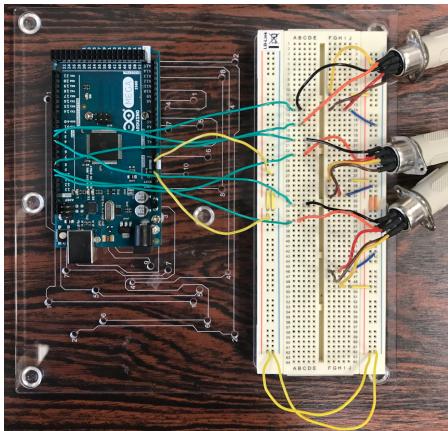


FIGURE 8 – Circuit de test de codeurs avec la carte Arduino Mega.

En effet, les moteurs commandés en PWM créaient des interférences sur la mesure des capteurs. Nous avons alors effectué les expériences suivantes pour essayer de comprendre d'où venait le problème :

Exp. 1 : Montage complet et moteurs à 100% (pas de PWM) : les codeurs renvoient les bonnes valeurs.

Exp. 2 : Montage complet et moteurs à 0% (pas de PWM) : les codeurs renvoient les bonnes valeurs.

Exp. 3 : Montage complet et moteurs à 50% : les codeurs divergent, à l'oscilloscope on mesure des variations de tension de 0,3 V (Figure 9).



FIGURE 9 – Étude à l'oscilloscope du problème d'interférence.

Exp. 4 : Montage complet et moteurs à 50%, la routine des codeurs n'est pas basée sur des interruptions (on ne fait que lire les valeurs logiques des pistes A et B des codeurs et incrémenter la variable des codeurs lorsqu'on détecte une variation) : les codeurs sont presque ok, il y a une lente divergence de l'angle (sur quelques secondes, la référence dévie de quelques degrés, la solution n'est pas satisfaisante car la commande durera plusieurs dizaines de secondes). Nous procédons alors à la première implémentation du stabilisateur PD, fonctionnel à la dérive près (Code en Annexe B).

Nous identifions alors le problème comme dépendant de la PWM, et nous effectuons des recherches sur des projets similaires présentant les mêmes difficultés. Nous obtenons très peu de résultats sur internet, et aucune solution. Nous mettons en question l'efficacité de la carte Arduino : est-ce qu'utiliser la PWM et des routines d'interruption en même temps est possible sans que ces deux fonctionnements interfèrent ? Pour vérifier qu'il ne s'agit pas d'un conflit de timer, nous utilisons un nouveau code sur la carte Arduino MEGA en faisant attention à bien utiliser des timers différents pour la PWM et les interruptions des codeurs. Les résultats ne changent pas, retour à la case départ (la résolution de ce problème électrique a duré près de 20h, pendant ce temps nous avons pu commencer à rédiger le rapport).

Nous supposons également des interférences de type électrique, Mathieu nous fournit des condensateurs et des bobines de ferrite pour filtrer les interférences mais sans résultat probant.

C'est en mesurant la tension aux bornes du moteur pour vérifier l'allure de la PWM que nous trouvons la solution : en reliant le pôle négatif du moteur à la terre, nous n'avons plus d'interférences. (Depuis les pôles négatifs des deux moteurs sont reliés directement à la masse, ce qui est tout à fait logique car sur le circuit imprimé précédent (Figure 5) les moins et la masse étaient communs).

Nous supposons que l'on commandait à l'Arduino d'imposer une tension (certes proche de 0 mais non nulle) entre le moins du moteur et la masse, eux-mêmes reliés par un long fil (donc une résistance très faible mais non nulle) et donc peut-être par conséquent parcourue par un courant fort (on rappelle que $u = Ri$, ceci serait à vérifier mais nous manquons de temps).

Les problèmes d'interférences réglés, nous avons fait fonctionner les codeurs et le moteur en même temps sans problème et avons alors commencé à simuler l'hélicoptère sous MATLAB afin de le commander à terme. Depuis nous avons implémenté à nouveau (en opérant quelques mises à jour) le contrôleur PD de stabilisation.

Nous avons alors entrepris la rénovation du circuit, en remplaçant le vieux circuit imprimé et les câbles d'alimentation/d'échange de données par un montage directement sur un *Prototyping Shield* Arduino. Nous en avons profité pour installer l'électroaimant (passer les câbles dans le bras du robot, souder une nouvelle piste et l'attacher avec une corde jouant le rôle de treuil) et son relais qui transformera le signal logique de l'Arduino (5 V) en tension de 12 V (à la base prévu pour 24 V mais pour qui 12 V suffisent à porter la balise/charge). Nous avons également fixé (avec l'aide de Mathieu pour une manipulation) ce *Shield* sur la base du robot et ajouté une fiche pour l'alimentation, pour avoir une sorte de boîtier plus compact et plus un amas de câbles.

Nous notons, en stabilisant le *pitch* du robot avec une commande PD, que les moteurs n'arrivent pas à donner un angle *elevation* élevé : il faut penser à équilibrer le robot différemment. Nous avons donc choisi de l'équilibrer de sorte à ce que l'électroaimant touche tout juste le sol à l'équilibre, ce qui définit la référence de l'angle *elevation* (ε).

5 Intégration des circuits à la maquette

Une fois que les problèmes électriques concernant l'utilisation simultanée de la PWM et des routines d'interruption pour la lecture des codeurs, ont été résolus, nous avons pu intégrer les cartes à la maquette. D'abord nous avons démonté la maquette pour enlever la carte originale de la base (Figure 10) et la remplacer par une interface plus facile entre le connecteur et la carte Arduino.

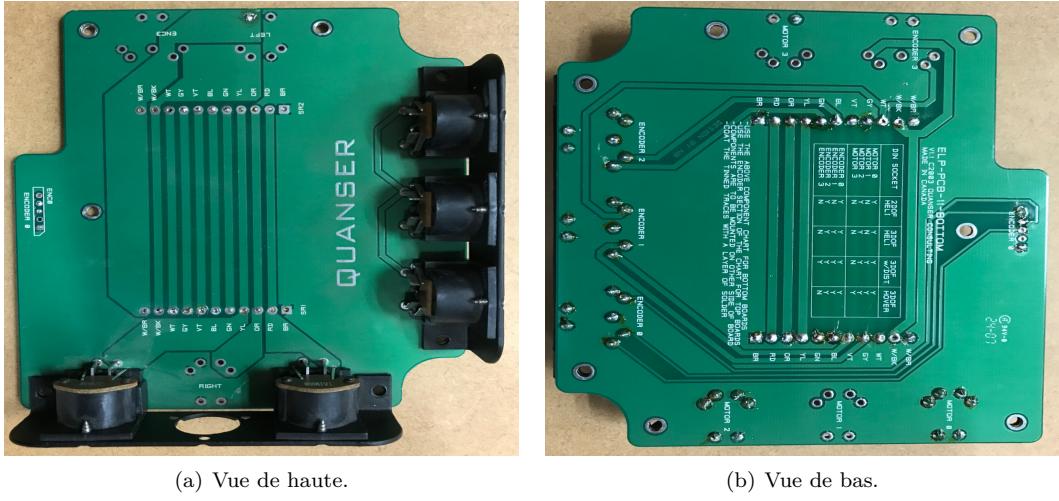


FIGURE 10 – Carte originale de la maquette.

Ensuite nous avons monté le *Prototype Shield* (Figure 24d en Annexe A) comme en Figure 11a. Ce Shield permet la liaison électrique entre les câbles qui viennent des codeurs et des moteurs et le *Motor Shield*. Nous avons aussi connecté le relais de l'électro-aimant (Figure 11b). Cet électro-aimant sera à terme utilisé pour attraper une charge posée au sol et ensuite de la relâcher.

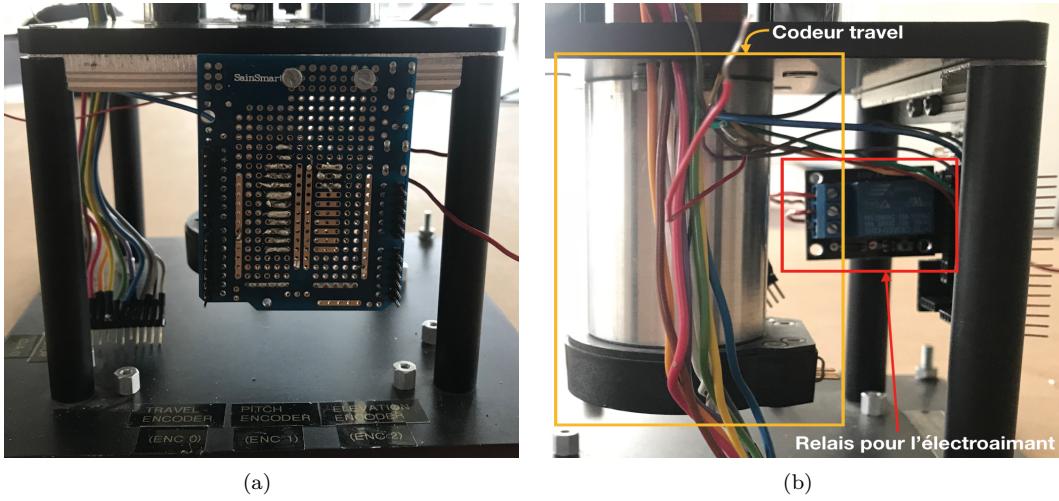
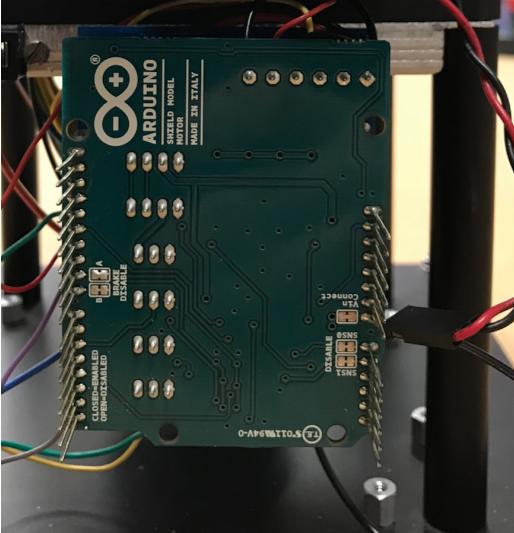
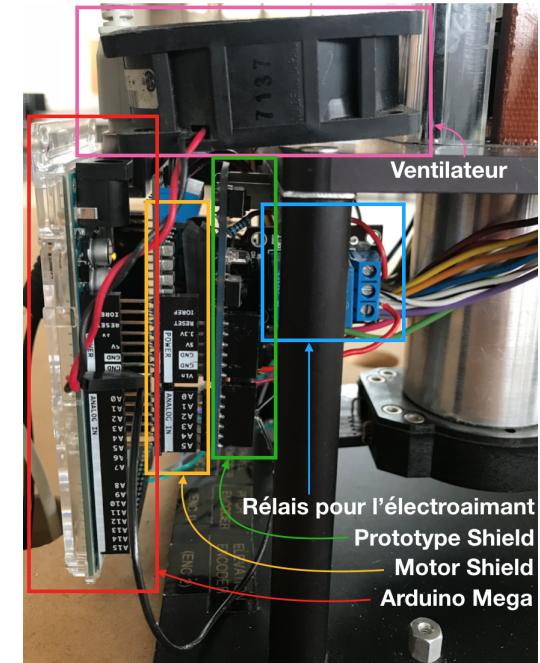


FIGURE 11 – Soutien de la maquette avec le *Prototype Shield* et Relais.

L'opération suivante a été de connecter le *Motor Shield* au *Prototype Shield* (Figure 12a). Le *Motor Shield* gère les deux moteur de la maquette et l'alimentation externe quand il est connecté à la carte Arduino Mega. Comme dernière étape nous avons rajouté la carte Arduino Mega (Figure 12b) : la maquette est prête pour les essais.



(a) *Motor Shield.*



(b) *Ventilateur.*

FIGURE 12 – Maquette intégrée du *Motor Shield* et de l’Arduino.

Après différents essais à niveaux variables de PWM nous nous sommes rendu compte que le *Motor Shield* provoquait de chutes de tension dues à son chauffage : pour résoudre ce problème nous avons rajouté un ventilateur pour dissiper la chaleur et refroidir les cartes (Figure 13).

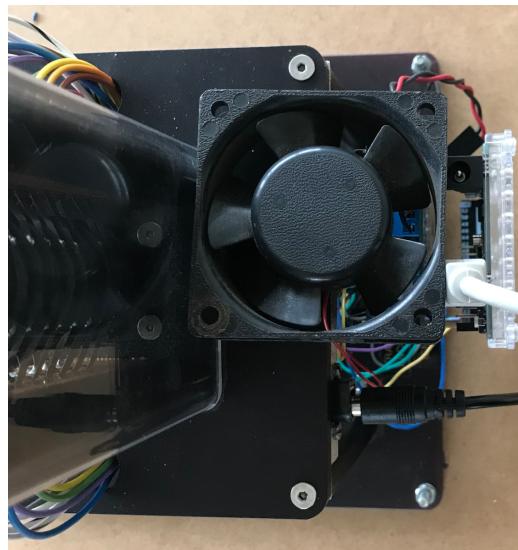
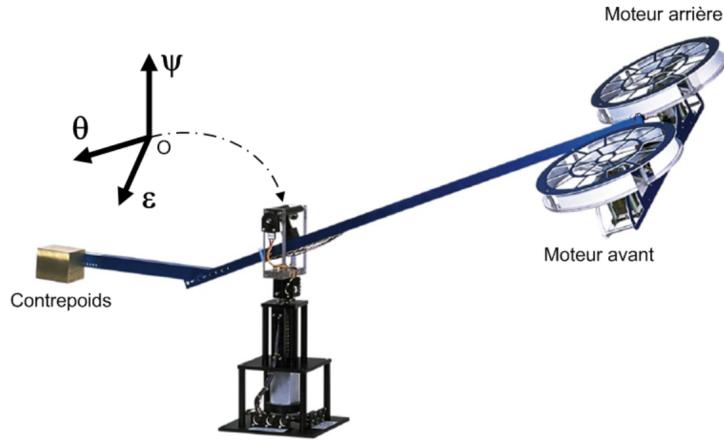


FIGURE 13 – Ajout du ventilateur.

6 Modèle dynamique simplifié du robot



$$\varepsilon = \text{Elevation} \quad \theta = \text{Pitch} \quad \psi = \text{Travel}$$

FIGURE 14 – Schéma expérimental de la maquette.

Le schéma expérimental de la maquette est celui de la Figure 14, où on place le repère du robot sur le pivot de l'axe *elevation*.

Pour la modélisation du robot ont été faites les assomptions suivantes :

- les axes *travel* et *elevation* sont perpendiculaires ;
- toutes les axes s'intersectent en un même point ;
- le repère de l'hélicoptère et du centre du contrepoids sont colinéaires avec l'axe du *pitch* ;
- le frottement des joints, la résistance de l'air et les forces centrifuges sont négligés ;
- la poussée est proportionnelle à la tension des moteurs et la dynamique entre moteur/hélice est négligée.

Les paramètres qu'on va utiliser pour le modèle simplifié du robot sont en Tableau 3 qui, avec les assomptions faites, amènent au schéma de la Figure 15.

<i>Symbol</i>	<i>Description</i>	<i>Valeur</i>	<i>Unité de Mesure</i>
V_f et V_b	Tension DC pour les moteurs <i>front</i> et <i>back</i>	[−12; +12]	V
K_f	Constante force-poussée de l'hélice	0,1188	N/V
M_h	Masse de l'hélicoptère	1,426	kg
M_w	Masse du contrepoids	1,87	kg
L_a	Distance entre l'axe <i>travel</i> et le corps de l'hélicoptère	0,660	m
L_h	Distance entre l'axe <i>pitch</i> et les moteurs	0,178	m
L_w	Distance entre l'axe <i>travel</i> et le contrepoids	0,470	m
g	Constante gravitationnelle	9,81	m/s^2
J_ε	Moment d'inertie par rapport à l'axe <i>Elevation</i>	1,0348	kg.m^2
J_θ	Moment d'inertie par rapport à l'axe <i>Pitch</i>	0,0451	kg.m^2
J_ψ	Moment d'inertie par rapport à l'axe <i>Travel</i>	1,0348	kg.m^2

TABLE 3 – Paramétrisation du modèle simplifié.

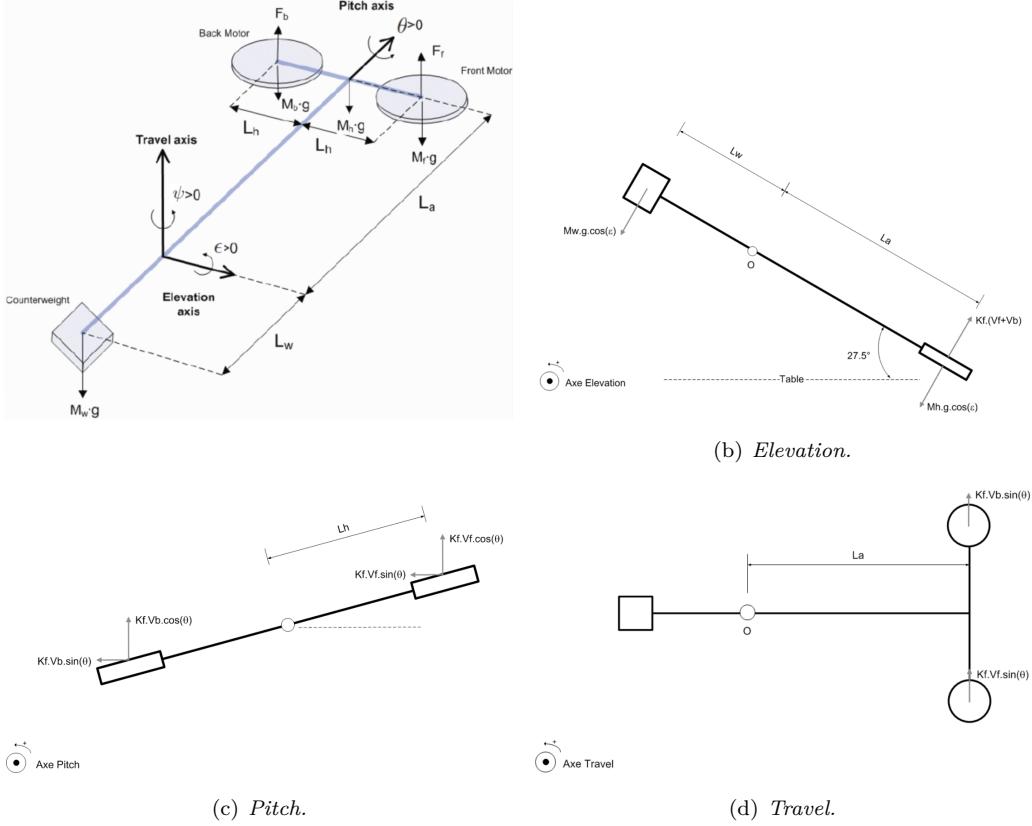


FIGURE 15 – Schéma du modèle simplifié et vues depuis les axes.

Depuis ce modèle on peut maintenant écrire les équations relatives aux trois axes :

— axe *elevation* :

$$J_\varepsilon \ddot{\varepsilon} = g(M_h L_a - M_w L_w) \cos(\varepsilon) + K_f(V_f + V_b) \cos(\theta) L_a \quad (1)$$

— axe *pitch* :

$$J_\theta \ddot{\theta} = K_f(V_f + V_b) L_h \quad (2)$$

— axe *travel* :

$$J_\psi \ddot{\psi} = K_f(V_f + V_b) \sin(\theta) L_a \cos(\varepsilon) \quad (3)$$

Le modèle ainsi devient :

$$\begin{aligned} J_\varepsilon \ddot{\varepsilon} &= g(M_h L_a - M_w L_w) \cos(\varepsilon) + L_a \cos(\theta) u_1 \\ J_\theta \ddot{\theta} &= L_h u_2 \\ J_\psi \ddot{\psi} &= L_a \sin(\theta) \cos(\varepsilon) u_1 \end{aligned} \quad (4)$$

où

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} F_f + F_b \\ F_f - F_b \end{bmatrix} \quad (5)$$

A cause de contraintes mécaniques, le domaine du travail est défini de la façon suivante :

- l'angle *Pitch* θ est tel que $-45 \text{ deg} \leq \theta \leq +45 \text{ deg}$;
- l'angle *Elevation* ε est tel que $-27.5 \text{ deg} \leq \varepsilon \leq +30 \text{ deg}$.

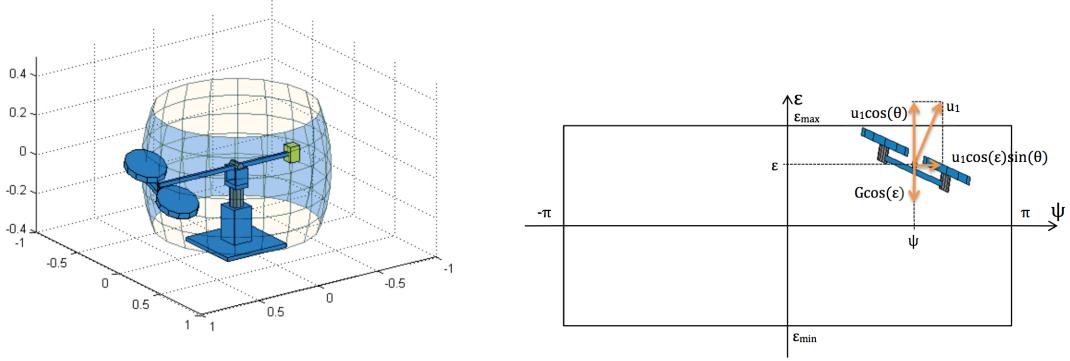
A ce modèle on peut rajouter l'équation de sortie, donnée par la mesure de trois codeurs :

$$y = [\varepsilon \ \theta \ \psi]^T \quad (6)$$

7 Synthèse de la loi de commande

Considérons d'abord le système en tant que un point localisé à la fin du bras, au milieu du segment défini par les deux moteurs :

- l'hélicoptère évolue dans l'espace bleu défini en Figure 16 (a) ;
- les forces le long des axes des dynamiques ε et ψ peuvent être découplées (Figure 16 (b)).



(a) Espace de travail.

(b) Décomposition des forces.

FIGURE 16 – Hypothèses initiales.

Dans ce cas, la position de l'hélicoptère est complètement définie par l'espace du travail sphérique de deux dimensions (ψ, ε) de la Figure 16 (a).

En définissant le vecteur de contrôle virtuel $\nu(u_1, \varepsilon, \theta) = [\nu_1 \ \nu_2]^T$ tel que :

$$\begin{aligned} \nu_1 &= u_1 \cos \varepsilon \sin \theta \\ \nu_2 &= u_1 \cos \theta - G \cos \varepsilon \end{aligned} \tag{7}$$

avec :

$$G = -\frac{g}{L_a}(M_h L_a - M_w L_w)$$

D'après (4) on obtient :

$$\begin{aligned} \ddot{\psi} &= \frac{L_a}{J_\psi} \nu_1 \\ \ddot{\varepsilon} &= \frac{L_a}{J_\varepsilon} \nu_2 \\ \ddot{\theta} &= \frac{L_h}{J_\theta} u_2 \end{aligned} \tag{8}$$

Donc, si l'entrée virtuelle existe, le système est transformé en un système découpé, c'est-à-dire que ψ , ε et θ sont respectivement contrôlés par ν_1 , ν_2 et u_2 .

La technique qui nous a amené au système linéaire (8) est dite *feedback input/output linearisation*, qui réduit le système à une chaîne d'intégrateurs. En choisissant de façon opportune ν_1 et ν_2 (par retour d'état) et en définissant l'erreur $e = x^d - x$ ($x(t)$ est une trajectoire quelconque) le système peut s'écrire sous la forme :

$$\begin{bmatrix} \dot{e} \\ \ddot{e} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k_0 & -k_1 \end{bmatrix} \begin{bmatrix} e \\ \dot{e} \end{bmatrix} \tag{9}$$

Le choix qu'on a fait est donc :

$$\begin{aligned} \nu_1 &= \frac{J_\psi}{L_a}(\ddot{\psi}^d + k_1 \dot{e} + k_0 e) \\ \nu_2 &= \frac{J_\varepsilon}{L_a}(\ddot{\varepsilon}^d + k_1 \dot{e} + k_0 e) \end{aligned} \tag{10}$$

D'après (7) on obtient

$$\begin{aligned} u_1^2 \sin^2 \theta &= \frac{\nu_1^2}{\cos^2 \varepsilon} \\ u_1^2 \cos^2 \theta &= (\nu_2 + G \cos \varepsilon)^2 \end{aligned} \quad (11)$$

d'où

$$u_1^2 = \frac{\nu_1^2}{\cos^2 \varepsilon} + (\nu_2 + G \cos \varepsilon)^2 \quad (12)$$

Ensuite on obtient

$$u_1 = S \cdot \sqrt{\frac{\nu_1^2}{\cos^2 \varepsilon} + (\nu_2 + G \cos \varepsilon)^2} \quad (13)$$

avec S définie en tant que

$$S = \begin{cases} \text{sign}(\nu_2 + G \cos \varepsilon) & \text{if } \nu_2 \neq 0 \\ 0 & \text{if } \nu_2 = 0 \end{cases} \quad (14)$$

De plus, θ doit satisfaire

$$\tan \theta = \frac{\nu_1}{\cos \varepsilon (\nu_2 + G \cos \varepsilon)} \quad (15)$$

Finalement, une moyenne de découpler le système comme en (8), c'est de forcer, grâce à l'entrée de contrôle u_2 , l'angle θ à suivre la trajectoire désirée

$$\theta^d(t) = \tan^{-1} \left(\frac{\nu_1}{\cos \varepsilon (\nu_2 + G \cos \varepsilon)} \right) \quad (16)$$

tandis que l'entrée de contrôle u_1 est définie en (13). Cette stratégie de contrôle permet de définir le schéma d'auto-pilote comme en Figure 17.

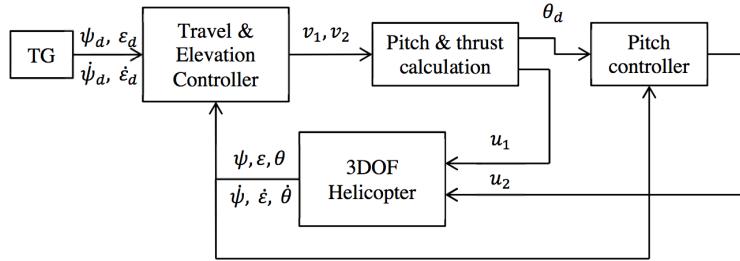


FIGURE 17 – Schéma blocs de l'auto-pilote.

- La première partie de l'auto-pilote permet de calculer les entrées de contrôle ν_1 et ν_2 à travers les erreurs entre ψ et ε et leurs trajectoires désirées $\psi_d(t)$ et $\varepsilon_d(t)$;
- D'après l'étape d'avant, on obtient d'abord l'entrée de contrôle u_1 avec (13), ensuite cette étape fournit la trajectoire désirée du *pitch* $\theta_d(t)$ avec (16). Avec cette trajectoire, le contrôleur *pitch* peut finalement fournir u_2 .
- Pour u_2 nous pouvons utiliser le même principe que pour les entrées virtuelles :

$$u_2 = \frac{J_\theta}{L_h} (\ddot{\theta}^d + k_1 \dot{\theta} + k_0 \theta) \quad (17)$$

Dans le modèle (4) les entrés de commande sont les forces de poussée des deux hélices mais en étant donné que le contrôle est fait à partir d'une carte Arduino, qui envoie la consigne aux moteurs sous forme de tension, il est impératif trouver :

- i) la relation entre poussée de l'hélice et vitesse de rotation des pales;
- ii) la relation entre vitesse de rotation et tension (plus précisément en pourcentage de PWM).

Concernant ce problème, nous avons d'abord trouvé la formule de Abbott qui permet de calculer la poussée d'une hélice en fonction du son pas, son diamètre et de la vitesse de rotation :

$$F = 28,35 \cdot p \cdot d^3 \cdot \omega^2 \cdot 10^{-10} \quad (18)$$

La maquette monte une hélice tripale *Graupner* (Figure 18a et spécifications en Annexe E) de pas $p = 6''$ et diamètre $d = 8''$ (données en pouces). La vitesse angulaire en fonction de la tension a ensuite été calculée de façon empirique avec le capteur de vitesse de Figure 18b (dans le cercle rouge est visible l'autocollant réfléchissant pour la lecture de l'impulsion du capteur) en changeant la PWM des intervalles de 10%.

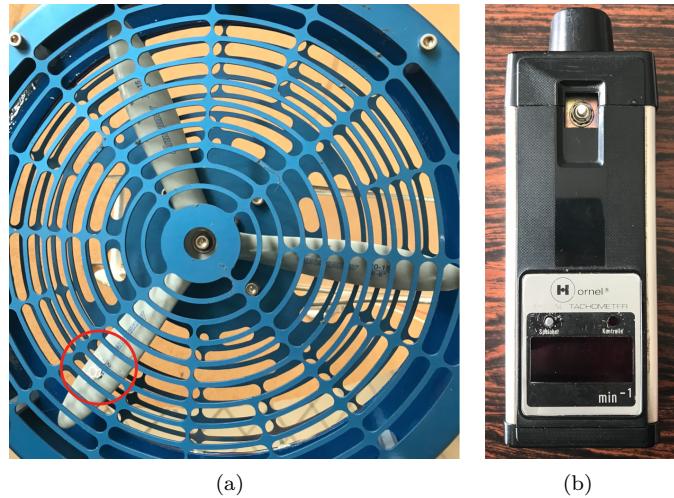


FIGURE 18 – Hélice de la maquette et capteur de vitesse.

Le tableau des données PWM/vitesse a été interpolé (avec $R^2 > 0.99$) à travers Excel, qui a renvoyé la loi suivante :

$$\begin{aligned}\omega_f(x) &= 1257,5 \cdot \ln(x) + 3820,1 \\ \omega_b(x) &= 1163,6 \cdot \ln(x) + 3656,8\end{aligned} \quad (19)$$

où x est le pourcentage de PWM envoyé aux moteurs et les indices f, b sont référencés aux moteurs *front* et *back* (Figure 19).

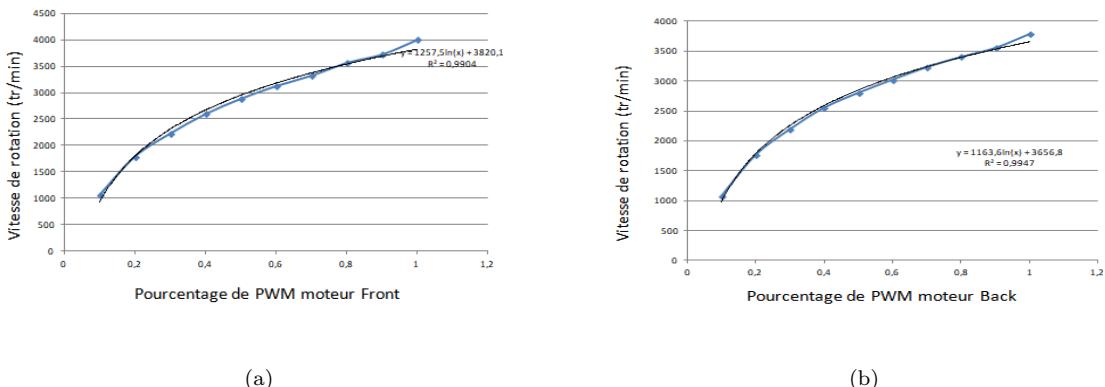


FIGURE 19 – Construction des courbes $\omega(x)$ par interpolation.

8 Simulations

Avant de tester les lois de commande introduites dans la section précédente, il nous a paru important et intéressant de les implémenter sur Simulink pour en étudier les prestation et par conséquent régler les gains. Pour ce faire deux modèle Simulink ont été créés, le premier (Figure 20) implémentant le modèle simplifié (8) et le deuxième implémentant le modèle réel (4) (Figure 21).

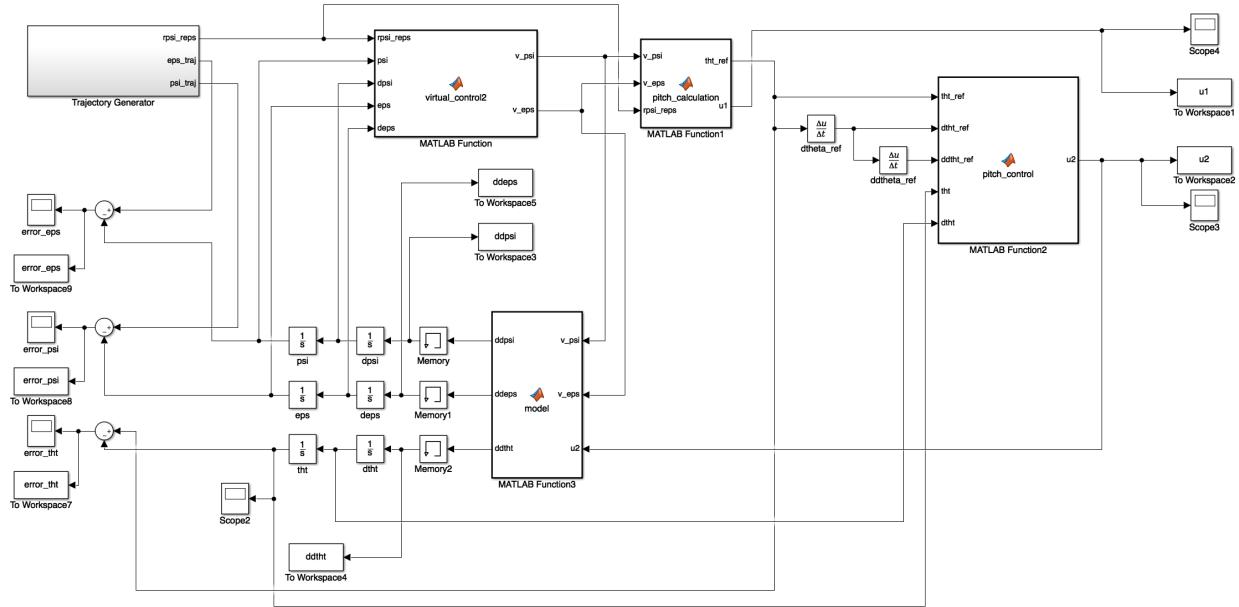


FIGURE 20 – Schéma Simulink implémentant le modèle simplifié.

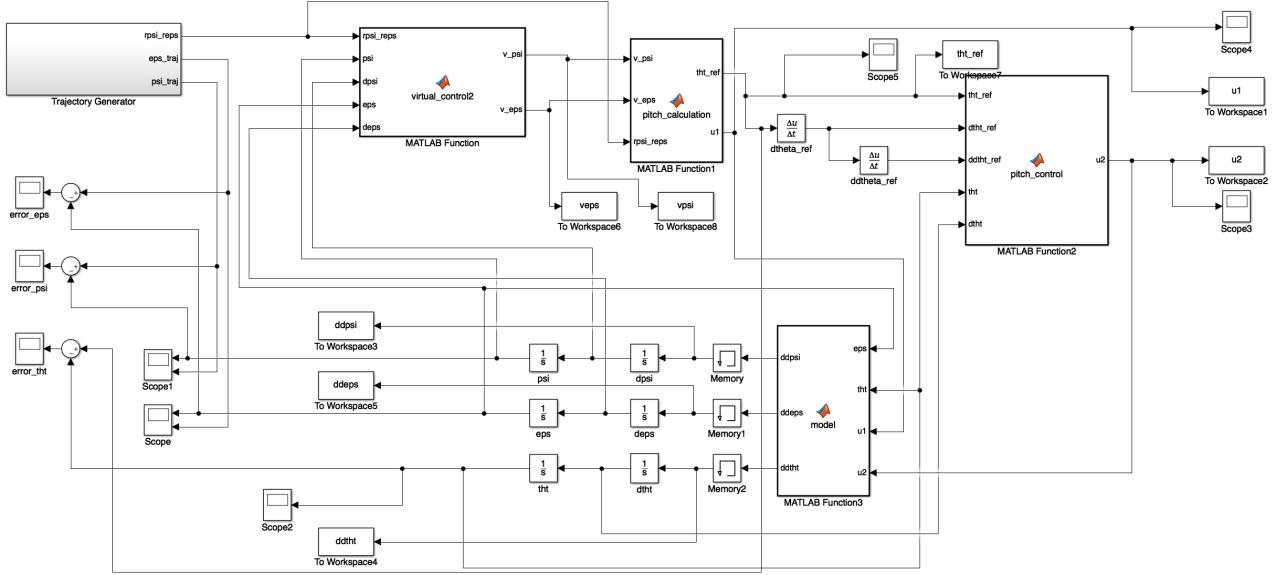


FIGURE 21 – Schéma Simulink implémentant le modèle réel.

Concernant les références, $\varepsilon^*(t)$, $\psi^*(t)$ suivent des sinusoïdes car elle sont des fonctions C^∞ et elles nous permettent de ne pas avoir des singularités dues à l'opération de dérivation. Notamment :

$$\begin{aligned}\varepsilon^*(t) &= \sin(\omega t) + 0.5 \\ \psi^*(t) &= \sin(\omega t)\end{aligned}\tag{20}$$

pour $\omega = 0.1$ rad (l'offset pour $\varepsilon^*(t)$ est nécessaire car l'élévation doit tenir compte du zéro donné par la table).

Comme nous avons anticipé dans la Section 7 l'erreur suit la dynamique d'un système du deuxième ordre, dont le dénominateur de la fonction de transfert peut être écrit :

$$s^2 + 2\xi\omega_n s + \omega_n^2 \quad (21)$$

donc les gains en (10) et (17) sont contraints par la relation :

$$\begin{cases} k_1 = 2\xi\omega_n \\ k_0 = \omega_n^2 \end{cases} \quad (22)$$

Pour le cas du système simplifié ont été faits les choix suivants :

- $k_0 = 100$;
- $k_1 = 20$;

pour tous les gains. Le modèle étant simplifié, la dynamique de l'erreur est très satisfaisante et converge vite à zéro. Les pics dans les premiers instants de la simulation proviennent du fait que les commandes ne sont pas bornées.

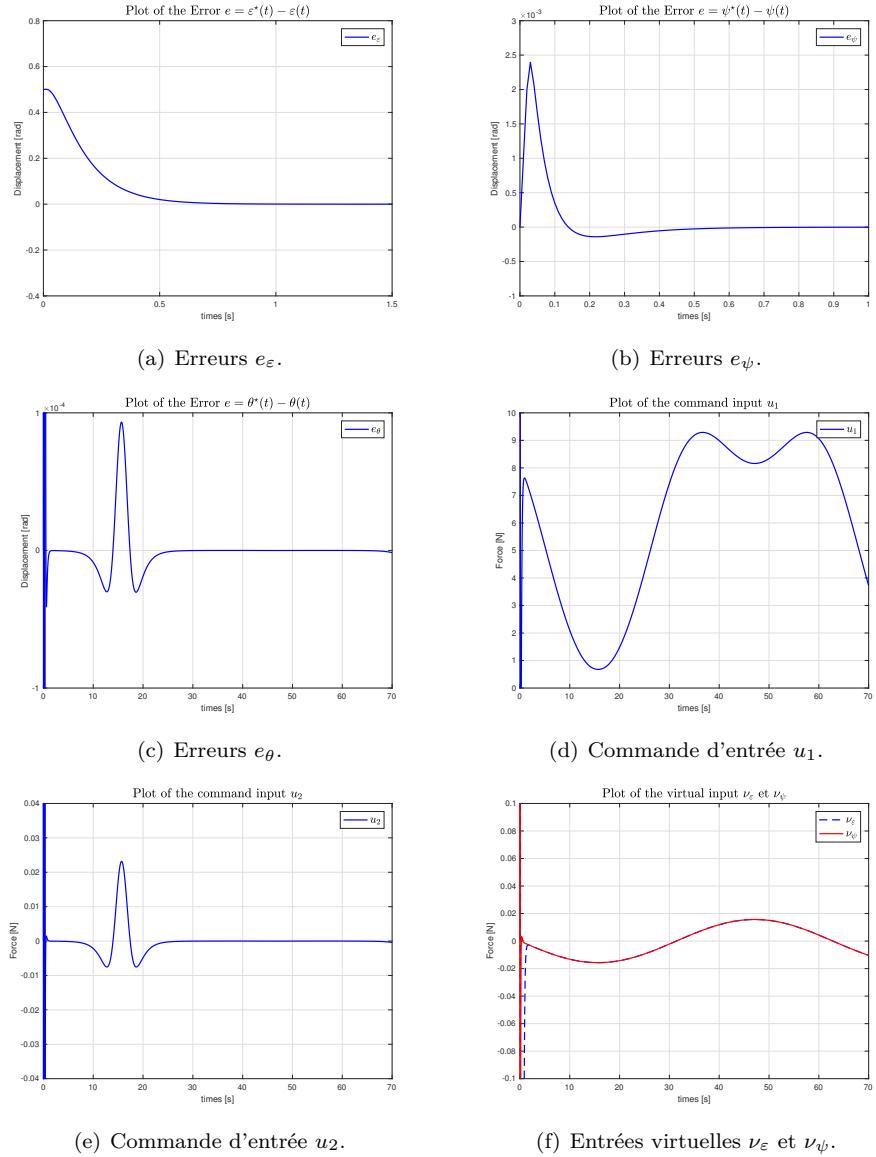


FIGURE 22 – Graphes du modèle simplifié.

Concernant le modèle réel, nous avons apporté les modifications suivantes :

- rajout d'un bloc de saturation sur u_1 sur l'intervalle $[-1, 1]$;
- changement des gains pour les entrées virtuelles : $k_1 = 16$, $k_0 = 64$;
- changement des gains pour l'entrée u_2 : $k_1 = 0,4$, $k_0 = 0,04$.

Dans ce cas l'erreur est présente et élevée car ici nous prenons en compte les vrais moteurs (donc la saturation) et la connaissance du modèle n'est pas bonne : suite aux modifications que nous avons présentées il est nécessaire de ré-identifier les paramètres du modèle.

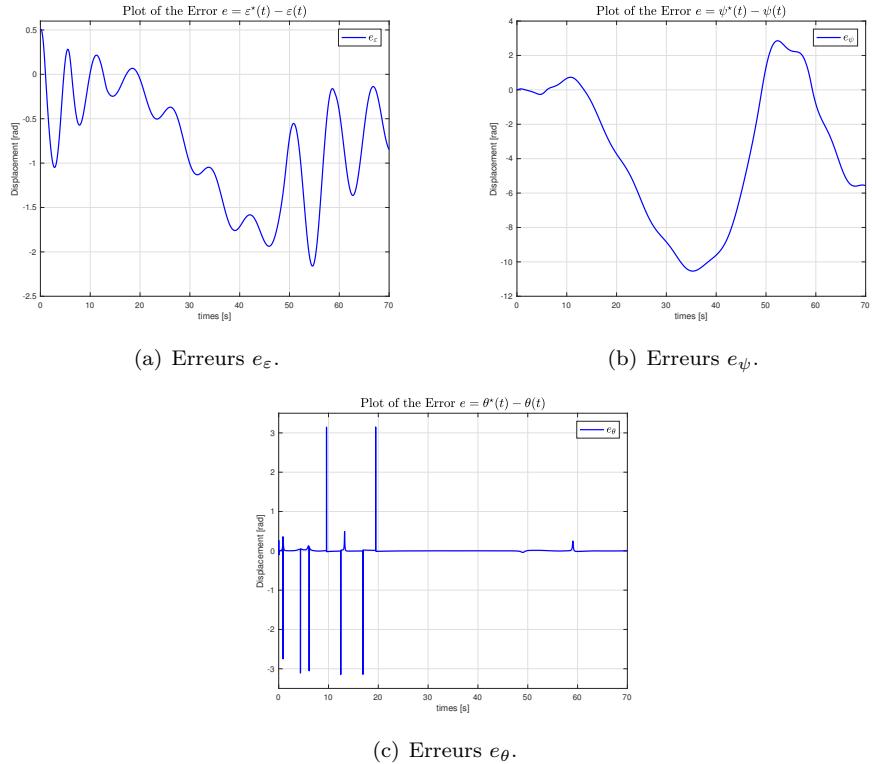


FIGURE 23 – Graphes du modèle réel.

9 Conclusions et travaux futures

Il est possible aujourd’hui tirer une conclusion sur notre travail au vu des objectifs définis en début de projet :

- 1) réhabiliter la maquette avec Arduino ;
- 2) ajouter un électro-aimant pour accrocher et décrocher la charge ;
- 3) commander cet électro-aimant ;
- 4) générer la trajectoire permettant de déplacer la charge sans la faire balancer.

Les objectifs 1) et 2) sont remplis, ainsi que le début du 3). Le quatrième n’a pas pu être rempli.

Pour finir la maquette, il resterait à réaliser la commande et mettre en place la/les trajectoire(s) avec la charge. Il pourrait aussi être intéressant d’ajouter un contrôleur manuel avec par exemple comme entrées les commandes virtuelles u_1 et u_2 (les “gaz” et l’accélération angulaire sur le pitch, ou alors sur F_{front} et F_{back}) ainsi qu’un bouton pour l’électro-aimant (le choix de ces entrées est le plus simple au niveau de la commande, un réglage plus “haut-niveau” demanderait des calculs pour générer la trajectoire à 1/2 s du robot si on voulait garder la même loi de commande).

Du point de vu de la commande, il pourrait être intéressant d’implémenter autres lois, notamment un contrôleur PID, une commande par *modes glissants*, une commande du type *back stepping* et d’autres basées sur la linéarisation exacte (par difféomorphisme), et les comparer pour en étudier les prestations. Cependant il faut se rappeler que la synthèse de ces lois n’est pas faite sur papier vis-à-vis du modèle mathématique, ou sur des logiciels de simulation : elles requièrent d’être implémentées sur Arduino. De même la loi de commande que nous avons présentée dans ce rapport marche parfaitement en simulation car le modèle et la loi de commande sont exactes. Notamment en fait, dans les applications réelles, la loi de commande est synthétisée de façon numérique selon les paramètres du modèle obtenus par **identification** (IDIM-LS : identification par modèle inverse et moindres carrés), donc ils ne sont jamais connus avec une précision absolue. D’ailleurs, ensuite à les modifications que nous avons apporté à la maquette, est fortement conseillé de refaire des séances d’identification pour modifier les valeurs de paramètres.

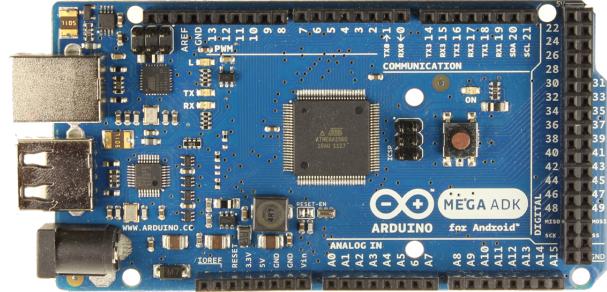
Ce projet nous a permis de développer certaines compétences qui nous serviront sûrement dans notre vie d’ingénieur. Nous avons pu ainsi développer notre autonomie, nos compétences pratiques en électronique, en informatique temps réel, en simulation et en commande non linéaire.

Annexes

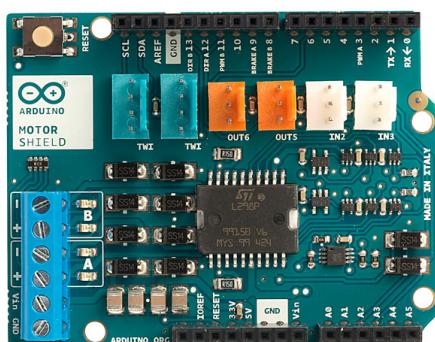
A Liste des cartes Arduino



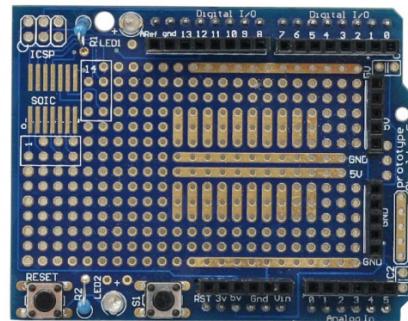
(a) Arduino Uno.



(b) Arduino Mega



(c) Arduino Motor Shield.



(d) Arduino Prototype Shield.

FIGURE 24 – Cartes et *shields* utilisés.

B Code C de la loi proportionnelle/dérivée

Ce code a été implémenté pour essayer de mettre en fonction la maquette et lire les valeurs du codeur *pitch*. De plus le but de ce code est aussi de stabiliser grossièrement l'angle *pitch* à travers une loi proportionnelle/dé-rivative.

```
#include <FlexiTimer2.h>

3 #include <digitalWriteFast.h>

double nbMillis = 10; //nb de millisecondes données à flexitimer
double dt = nbMillis/1000.0; //en ms c'est le temps entre deux appels de l'ISR
int compteurl = 0;
8 int limiteCompteur1 = 5; //nb de dt avant de mettre à jour les moteurs
volatile bool flag1;
int compteurn = 0;
int limiteCompteur2 = 50; //nb de dt avant d'afficher
volatile bool flag2;
13

// Définitions et déclarations pour le codeur incrémental du pitch
#define codeurPitchPinA 19
#define codeurPitchPinB 29
18 volatile long ticksPitch = 0;
volatile long ticksPitchml = 0;
volatile double pitchi=0;
volatile double dpitchi=0;
volatile double dpitchiml=0;
23 volatile double ddpitchi=0;
volatile double tpitch=0;
volatile double tpitchml=0;
volatile double dtpitch=0;

28 // Définitions et déclarations pour le codeur incrémental du elevation
#define codeurElevationPinA 20
#define codeurElevationPinB 30
volatile long ticksElevation = 0;
33 volatile long ticksElevationml = 0;
volatile double elevationi=0;
volatile double delevationi=0;
volatile double delevationiml=0;
volatile double ddelevationi=0;
38 volatile double telelevation=0;
volatile double telelevationml=0;
volatile double dtelelevation=0;

43 // Définitions et déclarations pour le codeur incrémental du travel
#define codeurTravelPinA 21
#define codeurTravelPinB 31
volatile long ticksTravel = 0;
volatile long ticksTravelml = 0;
48 volatile double traveli=0;
volatile double dtraveli=0;
volatile double dtraveliml=0;
```

```

53   volatile double ddtraveli=0;
54   volatile double ttravel=0;
55   volatile double ttravelm1=0;
56   volatile double dttravel=0;

57
58 //variables des moteurs
59
60   int Bdirpin = 13;
61   int Fdirpin = 12;

62   int Bbrapin = 8;
63   int Fbrapin = 9;

64   int Bspepin = 11;
65   int Fspepin = 3;

66
67 //variables électroaimant

68 #define pinAimant 5
69 bool aimantActive = false;
70
71 //++++++ FIN DECLARATION VARIABLES +++++++

72
73 //++++++ FONCTIONS AUXILIAIRES ++++++
74
75
76 //++++++ AFFICHAGE ++++++
77
78 void delaySerialPrint(){
79
80   Serial.print("Ticks angles :           ");
81
82   Serial.print("ticksPitch = " );
83   Serial.print(ticksPitch);
84
85   Serial.print("  ");
86
87   Serial.print("ticksElevation = " );
88   Serial.print(ticksElevation);
89
90   Serial.print("  ");
91
92   Serial.print("ticksTravel = " );
93   Serial.print(ticksTravel);
94
95   Serial.println("  ");
96
97
98   Serial.print("Angles :           ");
99
100  Serial.print("pitch = " );
101  Serial.print(pitchi);

```

```

108 Serial.print( " " );
      Serial.print( "elevation = " );
      Serial.print( elevationi );
113 Serial.print( " " );
      Serial.print( "travel = " );
      Serial.print( traveli );
118 Serial.println( " " );

      Serial.print( "vitesses angulaires : " );
123 Serial.print( "dpitch = " );
      Serial.print( dpitchi );
      Serial.print( " " );
128 Serial.print( "delevation = " );
      Serial.print( delevationi );
      Serial.print( " " );
133 Serial.print( "dtravel = " );
      Serial.print( dtraveli );
      Serial.println( " " );
138 Serial.print( "accelerations angulaires : " );
      Serial.print( "ddpitch = " );
      Serial.print( ddpitchi );
143 Serial.print( " " );
      Serial.print( "ddtravel = " );
      Serial.print( ddtraveli );
148 Serial.print( " " );
      Serial.print( "ddtravel = " );
      Serial.print( ddtraveli );
153 Serial.println( " " );

      Serial.print( "Aimant activé : " );
158 Serial.println( aimantActive );
      }
163 //++++++ MAJMOTORFUNC ++++++

```

```

void majmotorspeed(double theta, double omega) {
168
    double Kp = 3; //gain proportionnel
    double Kd = 1; //gain dérivé
    double bregl=0.29;
    int valmoy=int(0.7*255);
173
    int Bspeval = 0; //shield track B back motor
    int Fspeval = 0; //shield track A front motor

178 //Motor A
    Fspeval = -Kp*theta-Kd*omega;
    if (Fspeval>bregl*255){
        Fspeval=int(bregl*255);
    }
183 if (Fspeval<-bregl*255){
    Fspeval=int(-bregl*255);
}
    Fspeval=int(0.92*(Fspeval+valmoy));

188 //Motor B
    Bspeval=Kp*theta+Kd*omega;
    if (Bspeval>bregl*255){
        Bspeval=int(bregl*255);
    }
193 if (Bspeval<-bregl*255){
    Bspeval=int(-bregl*255);
}
    Bspeval=int(Bspeval+valmoy);

198 analogWrite(Bspepin, Bspeval); //Spins the motor on Channel B with pwm value
                                Bspeval

    analogWrite(Fspepin, Fspeval); //Spins the motor on Channel A with pwm value
                                Fspeval
}
//+++++++
203

//++++++ FIN DES FONCTIONS AUXILIAIRES +++++

208
//++++++ SETUP +++++

void setup() {
    // put your setup code here, to run once:
213
    // Codeur incrémental Pitch
    pinMode(codeurPitchPinB, INPUT); // entrée digitale pin B codeur
    digitalWrite(codeurPitchPinB, HIGH); // activation de la résistance de pullup
    // A chaque changement de niveau de tension sur le pin A du codeur Pitch,
218 // on exécute la fonction GestionInterruptionCodeurPitchPinA (définie à la fin
        du programme)

```

```

attachInterrupt(digitalPinToInterruption(codeurPitchPinA),
    GestionInterruptionCodeurPitchPinA, CHANGE);

// Codeur incrémental Elevation
pinMode(codeurElevationPinB, INPUT);      // entrée digitale pin B codeur
223 digitalWrite(codeurElevationPinB, HIGH); // activation de la résistance de
    pullup
// A chaque changement de niveau de tension sur le pin A du codeur Elevation,
// on exécute la fonction GestionInterruptionCodeurElevationPinA (définie à la
// fin du programme)
attachInterrupt(digitalPinToInterruption(codeurElevationPinA),
    GestionInterruptionCodeurElevationPinA, CHANGE);

228 // Codeur incrémental Travel
pinMode(codeurTravelPinB, INPUT);      // entrée digitale pin B codeur
digitalWrite(codeurTravelPinB, HIGH); // activation de la résistance de pullup
// A chaque changement de niveau de tension sur le pin A du codeur Travel,
// on exécute la fonction GestionInterruptionCodeurTravelPinA (définie à la fin
// du programme)
233 attachInterrupt(digitalPinToInterruption(codeurTravelPinA),
    GestionInterruptionCodeurTravelPinA, CHANGE);

// initialisation des Compteur d'impulsions des codeurs
//(c'est à ce moment que le robot doit être à sa position de référence connue)
238 ticksPitch = 0;
ticksElevation = 0;
ticksTravel = 0;

243 //+++++ MOTORS ++++++
//Setup Channel A = MOTORFRONT
pinMode(Fdirpin, OUTPUT); //Initiates Motor direction Channel A pin
pinMode(Fbrapin, OUTPUT); //Initiates Brake Channel A pin
248 digitalWrite(Fdirpin, HIGH); //Establishes forward direction of Channel A
digitalWrite(Fbrapin, LOW); //Disengage the Brake for Channel A

//Setup Channel B
253 pinMode(Bdirpin, OUTPUT); //Initiates Motor direction Channel B pin
pinMode(Bbrapin, OUTPUT); //Initiates Brake Channel B pin

digitalWrite(Bdirpin, HIGH); //Establishes forward direction of Channel B
digitalWrite(Bbrapin, LOW); //Disengage the Brake for Channel B
258 //+++++ SERIAL ++++++
Serial.begin(57600);
Serial.flush();
263

//+++++ ISR ++++++
268 FlexiTimer2::set(nbMillis, isrt); // résolution timer = 1 ms
FlexiTimer2::start();

```

```

//+++++ AIMANT ++++++
273 pinMode(pinAimant, OUTPUT);
if (aimantActive){
digitalWrite(pinAimant, HIGH);
}
else {
digitalWrite(pinAimant, LOW);
}

}

283 //+++++ FINDUSETUP ++++++
//+++++ LOOP ++++++
288 void loop() {
if (flag1){           //le flag1 s'active au bout de limiteCompteur1 passages dans l
  'isrt
majmotorspeed(pitchi, dpitchi);
293 flag1=false;
}
if (flag2){           //le flag2 s'active au bout de limiteCompteur2 passages dans l
  'isrt
delaySerialPrint();
flag2=false;
298 }
}

//+++++ FIN DU LOOP ++++++
303 //+++++ ISR ++++++
308 void isrt(){
//+++++ COMPTEURS ET AFFICHAGE ++++++
compteur1++;
313 compteur2++;

if (compteur1 >= limiteCompteur1){
compteur1=0;
flag1=true;
318 }
if (compteur2 >= limiteCompteur2){
compteur2=0;
flag2=true;
}
323 }

```

```

//++++ Routines des codeurs (on commenterà uniquement la première) +++
328 void GestionInterruptionCodeurPitchPinA() {
    // Routine de service d'interruption attachée à la voie A du codeur incrémental
    // droit
    // On utilise la fonction digitalReadFast2 de la librairie digitalWriteFast
    // car les lectures doivent ãtre très rapides pour passer le moins de temps
    // possible
    // dans cette fonction (appelée de nombreuses fois, à chaque impulsion de codeur
    // )
333 tpitch=millis();
dtpitch=tpitch-tpitchm1; //calcul du temps passé depuis le dernier appel (
    // changement de position)

if (digitalReadFast2(codeurPitchPinA) == digitalReadFast2(codeurPitchPinB)) {
    ticksPitch++;
338 }
else {
    ticksPitch--;
}

343 if (dtpitch>0){ //s'il s'est passé un temps non nul depuis le dernier
    // appel on calcule alors
    //les dérivées puis l'accélération angulaire
    pitchi=(ticksPitch%2048)*360/2048.;
    dpitchi=(ticksPitch-ticksPitchm1)/dtpitch*360./2048.*1000;
    ddpitchi=(dpitchi-dpitchim1)/dtpitch*1000;
348 }

    //mise à jour des variables
    tpitchm1=tpitch;
    dpitchim1=dpitchi;
353 ticksPitchm1=ticksPitch;
}

void GestionInterruptionCodeurElevationPinA() {
    // Routine de service d'interruption attachée à la voie A du codeur incrémental
    // droit
358 // On utilise la fonction digitalReadFast2 de la librairie digitalWriteFast
    // car les lectures doivent ãtre très rapides pour passer le moins de temps
    // possible
    // dans cette fonction (appelée de nombreuses fois, à chaque impulsion de codeur
    // )
    telelevation=millis();
    dtelevation=telelevation-telelevationm1;

363 if (digitalReadFast2(codeurElevationPinA) == digitalReadFast2(
        codeurElevationPinB)) {
    ticksElevation++;
}
else {
368 ticksElevation--;
}

if (dtelevation>0){
    elevationi=(ticksElevation%2048)*360/2048.;


```

```

373 delevationi=(ticksElevation-ticksElevationm1)/dtelevation*360./2048.*1000;
      ddelevationi=(delevationi-delevationim1)/dtelevation*1000;
    }

378 elevationm1=elevation;
delevationim1=delevationi;
ticksElevationm1=ticksElevation;
}

void GestionInterruptionCodeurTravelPinA() {
383 // Routine de service d'interruption attachée à la voie A du codeur incrémental
// droit
// On utilise la fonction digitalReadFast2 de la librairie digitalWriteFast
// car les lectures doivent être très rapides pour passer le moins de temps
// possible
// dans cette fonction (appelée de nombreuses fois, à chaque impulsion de codeur
)
ttravel=millis();
388 dttravel=ttravel-ttravelm1;

if (digitalReadFast2(codeurTravelPinA) == digitalReadFast2(codeurTravelPinB)) {
  ticksTravel++;
}
393 else {
  ticksTravel--;
}

if (dttravel>0){
398 traveli=(ticksTravel%4096)*360/4096.;
dtraveli=(ticksTravel-ticksTravelm1)/dttravel*360./4096.*1000;
ddtraveli=(dtraveli-dtravelim1)/dttravel*1000;
}

403 ttravelm1=ttravel;
dtravelim1=dtraveli;
ticksTravelm1=ticksTravel;
}

```

C Datasheet du moteur



9234S004

Lo-Cog® DC Motor



Assembly Data	Symbol	Units	Value	
Reference Voltage	E	V	12	
No-Load Speed	S _{NL}	rpm (rad/s)	6,151	(644)
Continuous Torque (Max.) ¹	T _C	oz-in (N-m)	6.1	(4.3E-02)
Peak Torque (Stall) ²	T _{PK}	oz-in (N-m)	41	(2.9E-01)
Weight	W _M	oz (g)	10	(286)
Motor Data				
Torque Constant	K _T	oz-in/A (N-m/A)	2.58	(1.82E-02)
Back-EMF Constant	K _E	V/krpm (V/rad/s)	1.91	(1.82E-02)
Resistance	R _T	Ω	0.83	
Inductance	L	mH	0.63	
No-Load Current	I _{NL}	A	0.33	
Peak Current (Stall) ²	I _P	A	14.49	
Motor Constant	K _M	oz-in/√W (N-m/√W)	3.01	(2.13E-02)
Friction Torque	T _F	oz-in (N-m)	0.60	(4.2E-03)
Rotor Inertia	J _M	oz-in-s ² (kg-m ²)	5.9E-04	(4.2E-06)
Electrical Time Constant	τ _E	ms	0.85	
Mechanical Time Constant	τ _M	ms	9.3	
Viscous Damping	D	oz-in/krpm (N-m-s)	0.039	(2.6E-06)
Damping Constant	K _D	oz-in/krpm (N-m-s)	6.7	(4.5E-04)
Maximum Winding Temperature	θ _{MAX}	°F (°C)	311	(155)
Thermal Impedance	R _{TH}	°F/watt (°C/watt)	62.8	(17.1)
Thermal Time Constant	τ _{TH}	min	12.0	
Gearbox Data				
Encoder Data				

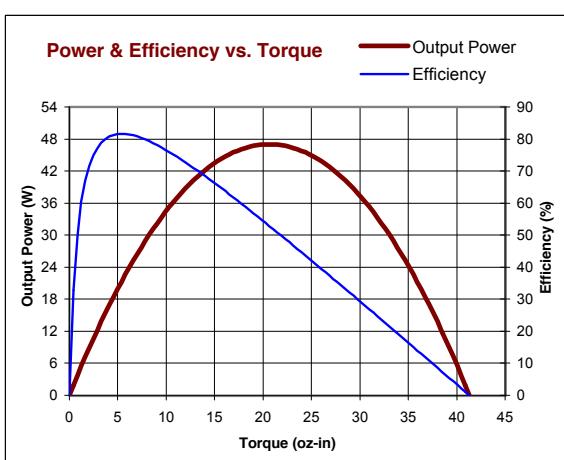
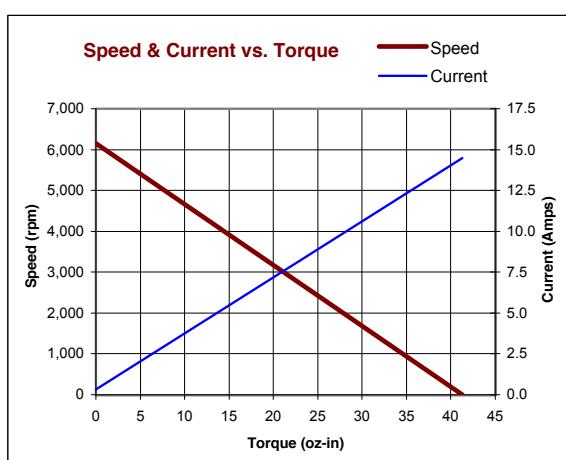
1 - Specified at max. winding temperature at 25°C ambient without heat sink. 2 - Theoretical values supplied for reference only.

Included Features

- 2-Pole Stator
- Ceramic Magnets
- Heavy-Gauge Steel Housing
- 7-Slot Armature
- Silicon Steel Laminations
- Stainless Steel Shaft
- Copper-Graphite Brushes
- Diamond Turned Commutator
- Motor Ball Bearings

Customization Options

- Alternate Winding
- Sleeve or Ball Bearings
- Modified Output Shaft
- Custom Cable Assembly
- Special Brushes
- EMI/RFI Suppression
- Spur or Planetary Gearbox
- Special Lubricant
- Optional Encoder
- Fail-Safe Brake



All values are nominal. Specifications subject to change without notice. Graphs are shown for reference only.

© 2001 Pittman.

D Datasheet du codeur

HEDS-9000/9100

Two Channel Optical Incremental Encoder Modules



Data Sheet



Description

The HEDS-9000 and the HEDS-9100 series are high performance, low cost, optical incremental encoder modules. When used with a codewheel, these modules detect rotary position. The modules consist of a lensed (LED) source and a detector IC enclosed in a small C-shaped plastic package. Due to a highly collimated light source and unique photodetector array, these modules are extremely tolerant to mounting misalignment.

The two channel digital outputs and the single 5 V supply input are accessed through five 0.025 inch square pins located on 0.1 inch centers.

Standard resolutions for the HEDS-9000 are 500 CPR and 1000 CPR for use with a HEDS-6100 codewheel or equivalent.

For the HEDS-9100, standard resolutions between 96 CPR and 512 CPR are available for use with a HEDS-5120 codewheel or equivalent.

Features

- High performance
- High resolution
- Low cost
- Easy to mount
- No signal adjustment required
- Small size
- -40°C to 100 °C operating temperature
- Two channel quadrature output
- TTL compatible
- Single 5 V supply

Applications

The HEDS-9000 and 9100 provide sophisticated motion detection at a low cost, making them ideal for high volume applications. Typical applications include printers, plotters, tape drives, and factory automation equipment.

Note: Avago Technologies encoders are not recommended for use in safety critical applications. Eg. ABS braking systems, power steering, life support systems and critical care medical equipment. Please contact sales representative if more clarification is needed.

ESD WARNING: NORMAL HANDLING PRECAUTIONS SHOULD BE TAKEN TO AVOID STATIC DISCHARGE.

Theory of Operation

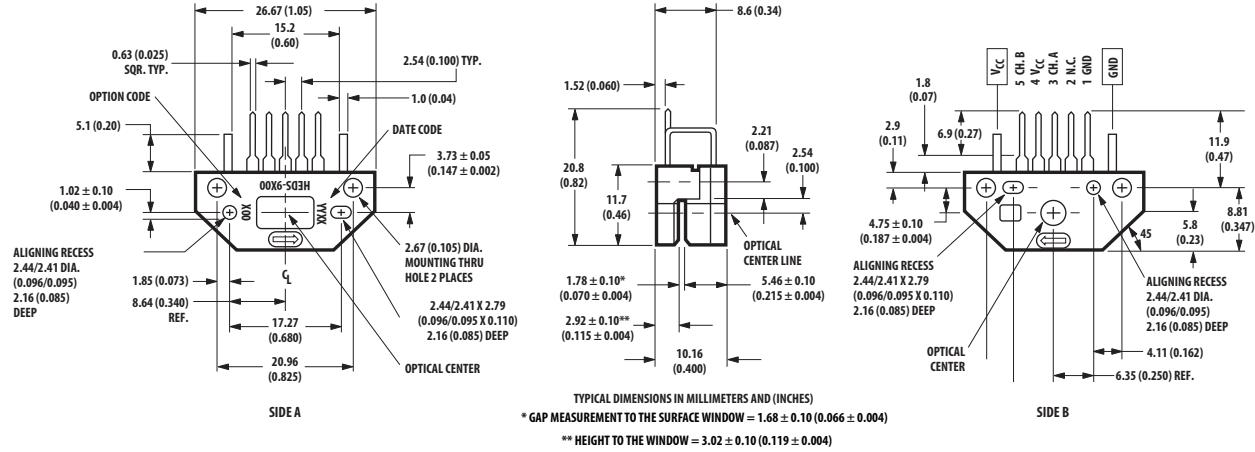
The HEDS-9000 and 9100 are C-shaped emitter/detector modules. Coupled with a codewheel, they translate the rotary motion of a shaft into a two-channel digital output.

As seen in the block diagram, each module contains a single Light Emitting Diode (LED) as its light source. The light is collimated into a parallel beam by means of a single polycarbonate lens located directly over the LED. Opposite the emitter is the integrated detector circuit. This IC consists of multiple sets of photodetectors and the signal processing circuitry necessary to produce the digital waveforms.

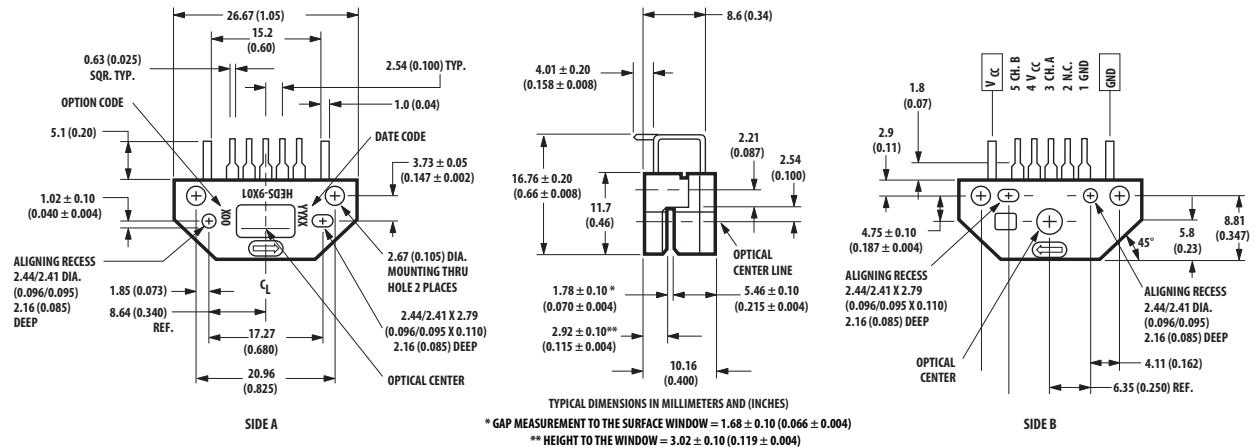
The codewheel rotates between the emitter and detector, causing the light beam to be interrupted by the pattern of spaces and bars on the codewheel. The photodiodes which detect these interruptions are arranged in a pattern that corresponds to the radius and design of the odewheel. These detectors are also spaced such that a light period on one pair of detectors corresponds to a dark period on the adjacent pair of detectors. The photodiode outputs are then fed through the signal processing circuitry resulting in A, \bar{A} , B, and \bar{B} . Two comparators receive these signals and produce the final outputs for channels A and B. Due to this integrated phasing technique, the digital output of channel A is in quadrature with that of channel B (90 degrees out of phase).

Package Dimensions

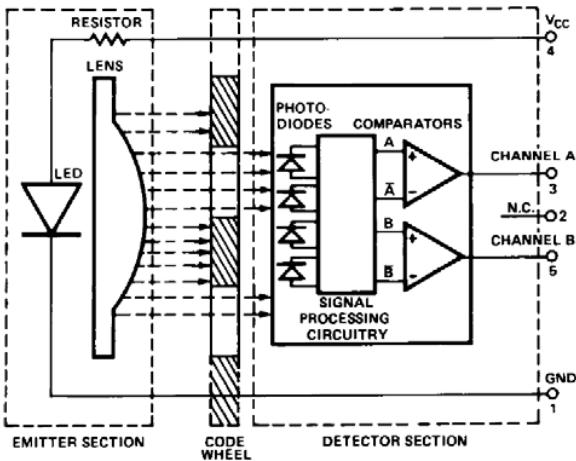
HEDS-9x00



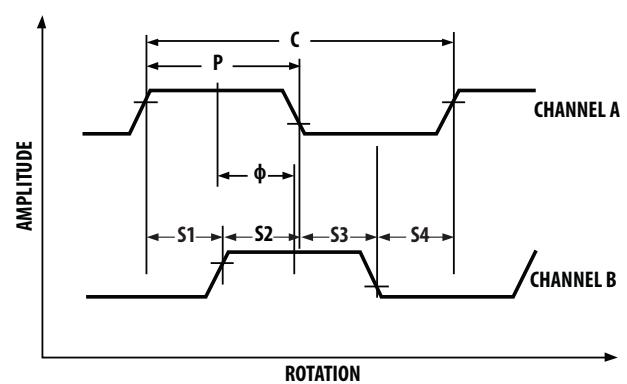
HEDS-9x01



Block Diagram



Output Waveforms



Definitions

Count (N): The number of bar and window pairs or counts per revolution (CPR) of the codewheel.

1 Shaft Rotation = 360 mechanical degrees,
= N cycles.

1 cycle (C) = 360 electrical degrees ($^{\circ}$ e),
= 1 bar and window pair.

Pulse Width (P): The number of electrical degrees that an output is high during 1 cycle. This value is nominally 180° e or 1/2 cycle.

Pulse Width Error (ΔP): The deviation, in electrical degrees of the pulse width from its ideal value of 180° e.

State Width (S): The number of electrical degrees between a transition in the output of channel A and the neighboring transition in the output of channel B. There are 4 states per cycle, each nominally 90° e.

State Width Error (ΔS): The deviation, in electrical degrees, of each state width from its ideal value of 90° e.

Phase (ϕ): The number of electrical degrees between the center of the high state of channel A and the center of the high state of channel B. This value is nominally 90° e for quadrature output.

Phase Error ($\Delta\phi$): The deviation of the phase from its ideal value of 90° e.

Direction of Rotation: When the codewheel rotates in the direction of the arrow on top of the module, channel A will lead channel B. If the codewheel rotates in the opposite direction, channel B will lead channel A.

Optical Radius (R_{op}): The distance from the codewheel's center of rotation to the optical center (O.C.) of the encoder module.

Absolute Maximum Ratings

Storage Temperature, T_s	-40°C to 100°C
Operating Temperature, T_A	-40°C to 100°C
Supply Voltage, V_{CC}	-0.5 V to 7 V
Output Voltage, V_o	-0.5 V to V_{CC}
Output Current per Channel, I_{out}	-1.0 mA to 5 mA

Recommended Operating Conditions

Parameter	Symbol	Min.	Typ.	Max.	Units	Notes
Temperature	T	-40		100	°C	
Supply Voltage	V_{CC}	4.5		5.5	Volts	Ripple < 100 mV _{p-p}
Load Capacitance	C_L			100	pF	3.3 kΩ pull-up resistor
Count Frequency	f			100	kHz	Velocity (rpm) x N 60

Note: The module performance is guaranteed to 100 kHz but can operate at higher frequencies.

Encoding Characteristics

Encoding Characteristics over Recommended Operating Range and Recommended Mounting Tolerances. These Characteristics do not include codewheel/codestrip contribution.

Description	Sym.	Typ.	Case 1 Max.	Case 2 Max.	Units	Notes
Pulse Width Error	ΔP	30	40		°e	
Logic State Width Error	ΔS	30	40		°e	
Phase Error	$\Delta \phi$	2	10	105	°e	

Case 1: Module mounted on tolerance circle of ±0.13 mm (±0.005 in.).

Case 2: HEDS-9000 mounted on tolerances of ±0.50 mm (0.020").

HEDS-9100 mounted on tolerances of ±0.38 mm (0.015").

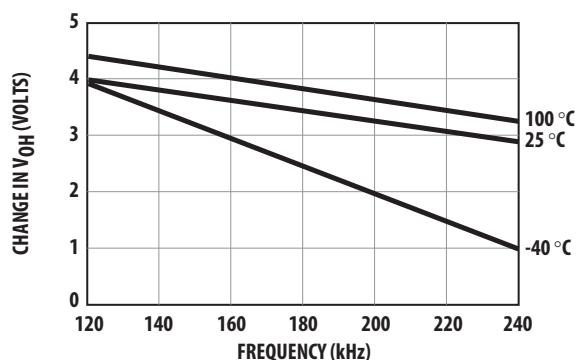
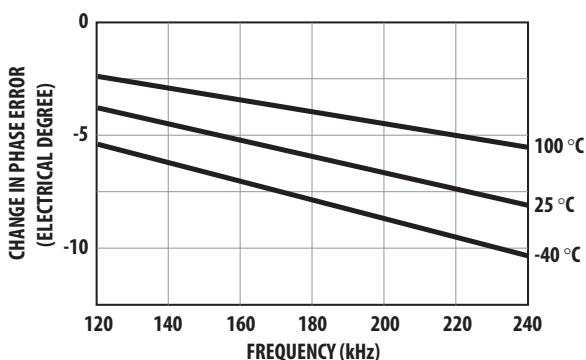
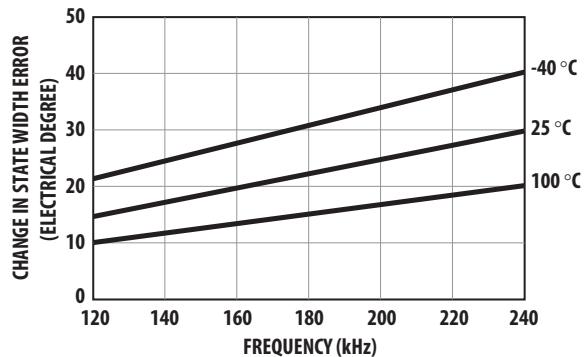
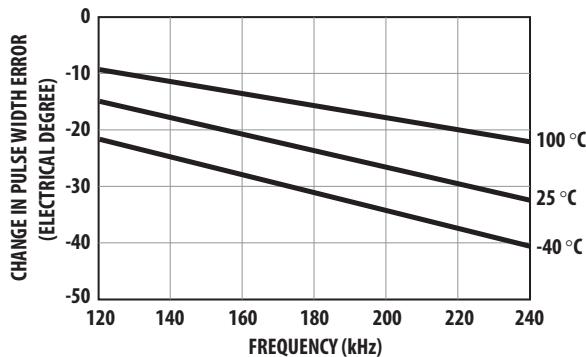
Electrical Characteristics

Electrical Characteristics over Recommended Operating Range, typical at 25°C.

Parameter	Symbol	Min.	Typical	Max.	Units	Notes
Supply Current	I_{CC}		17	40	mA	HEDS-9100 All Series, HEDS-9000 #A00, B00, and J00
Supply Current	I_{CC}		51	85	mA	HEDS-9000 #T00 and U00
High Level Output Voltage	V_{OH}	2.4			Volts	$I_{OH} = -40 \mu A$ max.
Low Level Output Voltage	V_{OL}			0.4	Volts	$I_{OL} = 3.2$ mA
Rise Time	t_r		200		ns	$C_L = 25$ pF
Fall Time	t_f		50		ns	$R_L = 11$ kΩ pull-up

Derating Curves over Extended Operating Frequencies (HEDS-9000/9100)

Below are the derating curves for state, duty, phase and V_{OH} over extended operating frequencies of up to 240 kHz (recommended maximum frequency is 100 kHz). The curves were derived using standard TTL load. -40°C operation is not feasible above 160 kHz because V_{OH} will drop below 2.4 V (the minimum TTL for logic state high) beyond that frequency.



Recommended Codewheel Characteristics

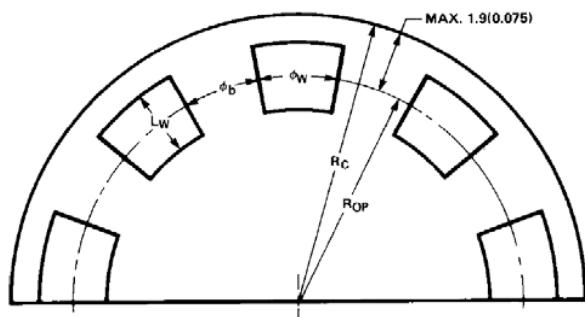


Figure 1. Codestrip Design

Codewheel Options

HEDS Series	CPR (N)	Option	Optical Radius mm (in.)
5120	96	K	11.00 (0.433)
5120	100	C	11.00 (0.433)
5120	192	D	11.00 (0.433)
5120	200	E	11.00 (0.433)
5120	256	F	11.00 (0.433)
5120	360	G	11.00 (0.433)
5120	400	H	11.00 (0.433)
5120	500	A	11.00 (0.433)
5120	512	I	11.00 (0.433)
6100	500	A	23.36 (0.920)
6100	1000	B	23.36 (0.920)

Parameter	Symbol	Minimum	Maximum	Units	Notes
Window/Bar Ratio	ϕ_w/ϕ_b	0.7	1.4		
Window Length	L_w	1.8 (0.071)	2.3 (0.09)	mm (inch)	
Absolute Maximum Codewheel Radius	R_c		$R_{OP} + 1.9$ (0.0075)	mm (inch)	Includes eccentricity errors

Mounting Considerations

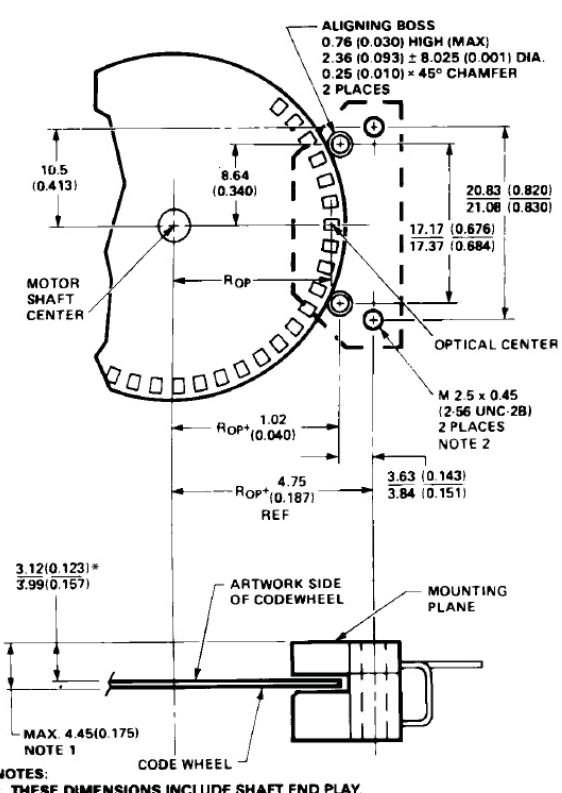


Figure 2. Mounting Plane Side A.

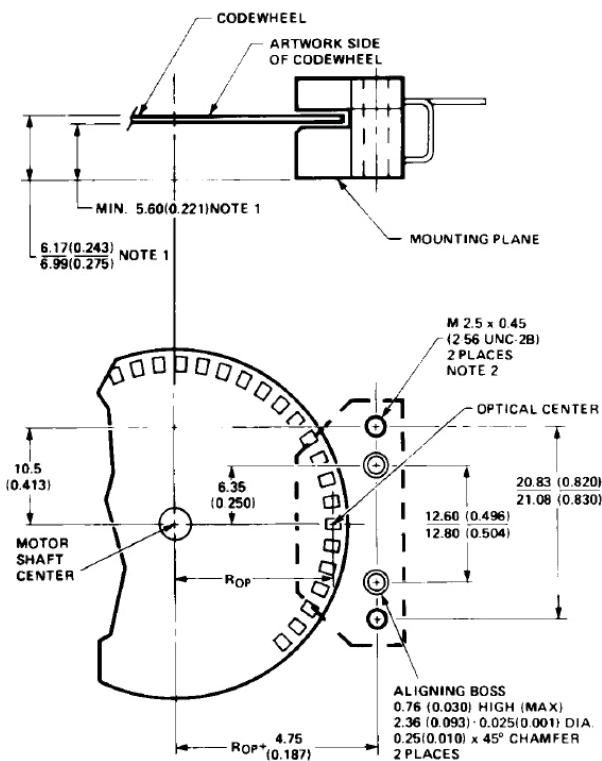


Figure 3. Mounting Plane Side B.

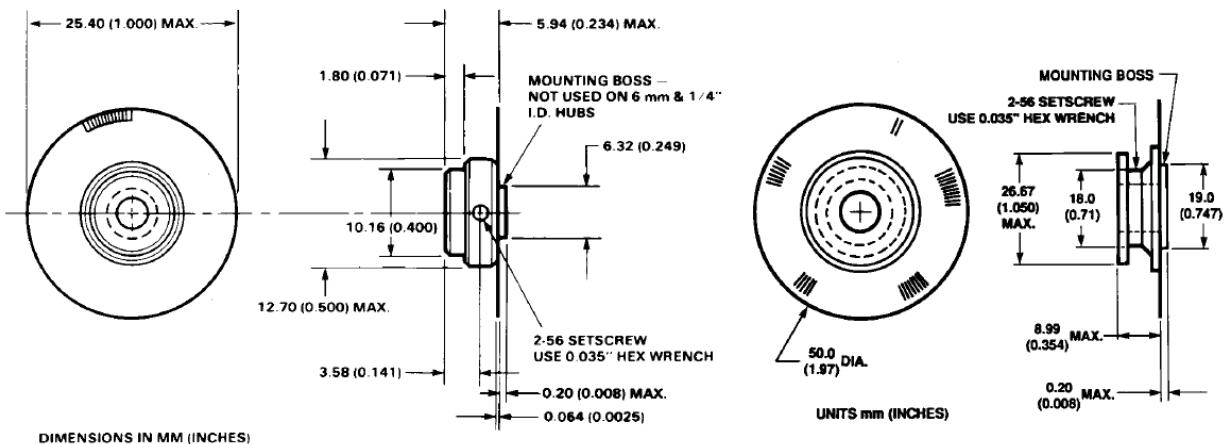


Figure 4. Mounting as Referenced to Side A.

Figure 5. Mounting as Referenced to Side B.

Connectors

Mounting Manufacturer	Part Number	Surface
AMP	1203686-4	Both
	640442-5	Side B
DuPont	65039-032 with 4825X-000 term.	Both
HP	HEDS-8902 with 4-wire leads	Side B (see Fig. 6)
Molex	2695 series with 2759 series term.	Side B

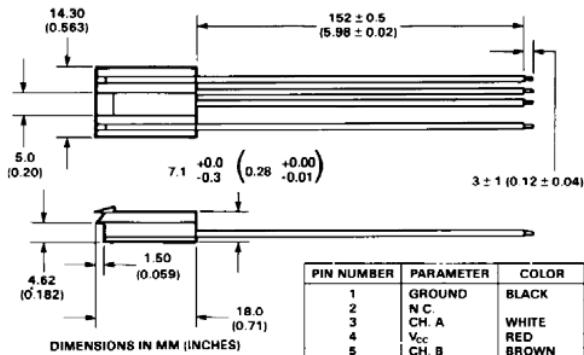
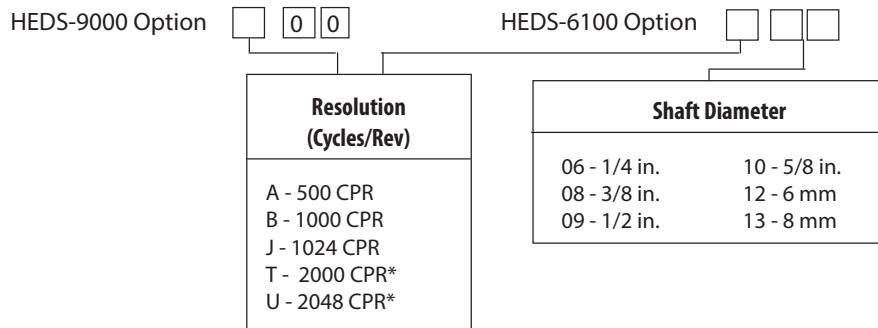


Figure 6. HEDS-8902 Connector.

Ordering Information



	A	B	C	D	E	F	G	H	I	J	K	L	S	T	U
HEDS-9000	*	*								*				*	*

	01	02	03	04	05	06	08	09	10	11	12	13	14
HEDS-6100	A							*				*	*
	B						*	*	*	*			

HEDS-910	<input type="checkbox"/>	0	Option	<input type="checkbox"/>	0	0	HEDS-5120 Option
Lead				Resolution (Cycles/Rev)			
0 - Straight Leads				S - 50 CPR	H - 400 CPR		
1 - Bent Leads				K - 96 CPR	A - 500 CPR	01 - 2 mm	11 - 4 mm
				C - 100 CPR	I - 512 CPR	02 - 3 mm	14 - 5 mm
				E - 200 CPR	B - 1000 CPR*	03 - 1/8 in.	12 - 6 mm
				F - 256 CPR	J - 1024 CPR*	04 - 5/32 in.	13 - 8 mm
				G - 360 CPR		05 - 3/16 in.	06 - 1/4 in.

	A	B	C	D	E	F	G	H	I	J	K	S	T	U
HEDS-9100	*	*	*		*	*	*	*	*	*	*	*		
HEDS-9101	*		*		*		*							

		01	02	03	04	05	06	08	09	10	11	12	13	14
HEDS-5120	A	*	*	*	*	*	*				*	*		*
	C		*				*				*	*	*	*
	D					*								
	E						*					*		
	F					*								
	G		*	*		*	*				*			*
	H		*				*				*	*		*
	I		*		*		*				*	*	*	
	K		*										*	

For product information and a complete list of distributors, please go to our website: www.avagotech.com

Avago Technologies and the A logo are trademarks of Avago Technologies in the United States and other countries.
Data subject to change. Copyright © 2013-2016 Avago Technologies. All rights reserved. Obsoletes 5988-6712EN.
AV02-1867EN - August 1, 2016

Avago
TECHNOLOGIES
A Broadcom Limited Company

E Spécifications de l'hélice *Graupner* 1315.20.15



FIGURE 25 – Hélice *Graupner* utilisée sur la maquette.

Description Il s'agit d'une hélice tripale SUPER NYLON. Les pales sont fait en nylon renforcé avec les fibres de verre et elle sont solides.

Spécifiques Les informations peuvent être trouvés au lien :

<https://www.graupner.com/3-blades-Prop-20x15-cm-/1315.20.15/>

Sort :	Rigid props
Rotating direction :	R
Colour :	grey
Diameter / bore [mm] :	6,5
Diameter / pitch [inch] :	8 x 6
Piece / Pack :	1
Type :	3/9,9

TABLE 4 – Spécifiques de l'hélice *Graupner*.