

## PROJET DE MASTER 2

---

# Autoscope

## Quatrième compte rendu

---

*Auteurs :*

Thomas ABGRALL  
Clément AILLOUD  
Thibaud LE DOLEDEC  
Thomas LEPOIX

MASTER Systèmes Embarqués

E.S.T.E.I.

École Supérieure des Technologies Électronique, Informatique, et Infographie  
Département Systèmes Embarqués

# Table des matières

<b>Table des matières</b>	<b>1</b>
<b>I Partie de groupe</b>	<b>3</b>
<b>1 Avancement général</b>	<b>4</b>
<b>II Thomas LEPOIX</b>	<b>5</b>
<b>2 Architecture</b>	<b>6</b>
2.1 Organisation des interfaces logicielles du télescope . . . . .	6
<b>3 Hardware</b>	<b>7</b>
3.1 Schéma structurel . . . . .	7
3.2 Bon de commande . . . . .	8
3.3 Fichiers de fabrication . . . . .	9
3.4 Modélisation 3D . . . . .	10
3.4.1 Allure générale de la carte . . . . .	10
3.4.2 Connecteurs des moteurs . . . . .	11
3.4.3 Emboîtement des cartes . . . . .	11
3.4.4 Repères des connecteurs . . . . .	12
3.4.5 Intégration du modèle 3D au télescope . . . . .	13
<b>4 Système d'exploitation</b>	<b>14</b>
4.1 Splash screen . . . . .	14
4.2 Hotspot Wifi . . . . .	16
4.3 Serveur FTP . . . . .	19
4.4 Driver helloworld . . . . .	20
4.5 Étude du transfert du flux vidéo sur le réseau . . . . .	21
<b>5 Travail restant</b>	<b>22</b>
5.1 Hardware . . . . .	22
5.2 Système d'exploitation . . . . .	22
5.3 Software . . . . .	22
<b>III Thomas ABGRALL</b>	<b>23</b>
<b>6 Driver de la commande des moteurs</b>	<b>24</b>
6.1 Rappel . . . . .	24
6.2 Fonctionnalités . . . . .	24
6.3 Câblage . . . . .	25
6.4 Avancement du programme . . . . .	25

6.5	Procédure de test . . . . .	25
6.6	Interactions avec le driver . . . . .	27
<b>7</b>	<b>Fabrication du télescope</b>	<b>28</b>
<b>8</b>	<b>Travail à venir</b>	<b>29</b>

**Première partie**

**Partie de groupe**

## Chapitre 1

# Avancement général

Nous avons commencé à monter le télescope. Le socle circulaire, les deux croissants ainsi que le support du miroir secondaire.

Plusieurs parties indépendantes seront bientôt terminées et le projet approche une phase d'intégration.

**Deuxième partie**

**Thomas LEPOIX**

## Chapitre 2

# Architecture

### 2.1 Organisation des interfaces logicielles du télescope

Il s'agit ici de définir les différents canaux de communication entre le télescope et un ordinateur client (équipé de Stellarium), ainsi que le rôle de chaque.

- Accès d'administration : Protocole SSH (port 22), réservé à l'utilisateur `admin`
- Transfert de photos : Protocole FTP (ports 20 et 21), réservé à l'utilisateur `autoscope`
- Transfert des directives de Stellarium : Port arbitraire (actuellement 4444), à étudier.
- Transfert du flux vidéo : À étudier

Définir ces différents canaux de communication et les associer à des utilisateurs particuliers permet de renforcer la sécurité du système en le partitionnant davantage. De plus cela permettra ensuite de mettre en place un pare-feu robuste.

Il sera judicieux d'utiliser pour les directives de Stellarium et éventuellement le flux vidéo des ports de la plage 49152 – 65535 car ceux-ci sont dédiés à des usages de ce type, qui ne sont pas voués à être standardisés. Cela diminuera le risque d'entrer un jour en conflit de port avec un autre protocole réseau.

# Chapitre 3

## Hardware

### 3.1 Schéma structurel

Ci-dessous le schéma structurel de la carte électronique du télescope. Un accès d'alimentation +12V pour un ventilateur a été ajouté, celui-ci étant normalement présent sous le miroir primaire et servant à le garder à température constante. Il avait été oublié jusqu'à présent.

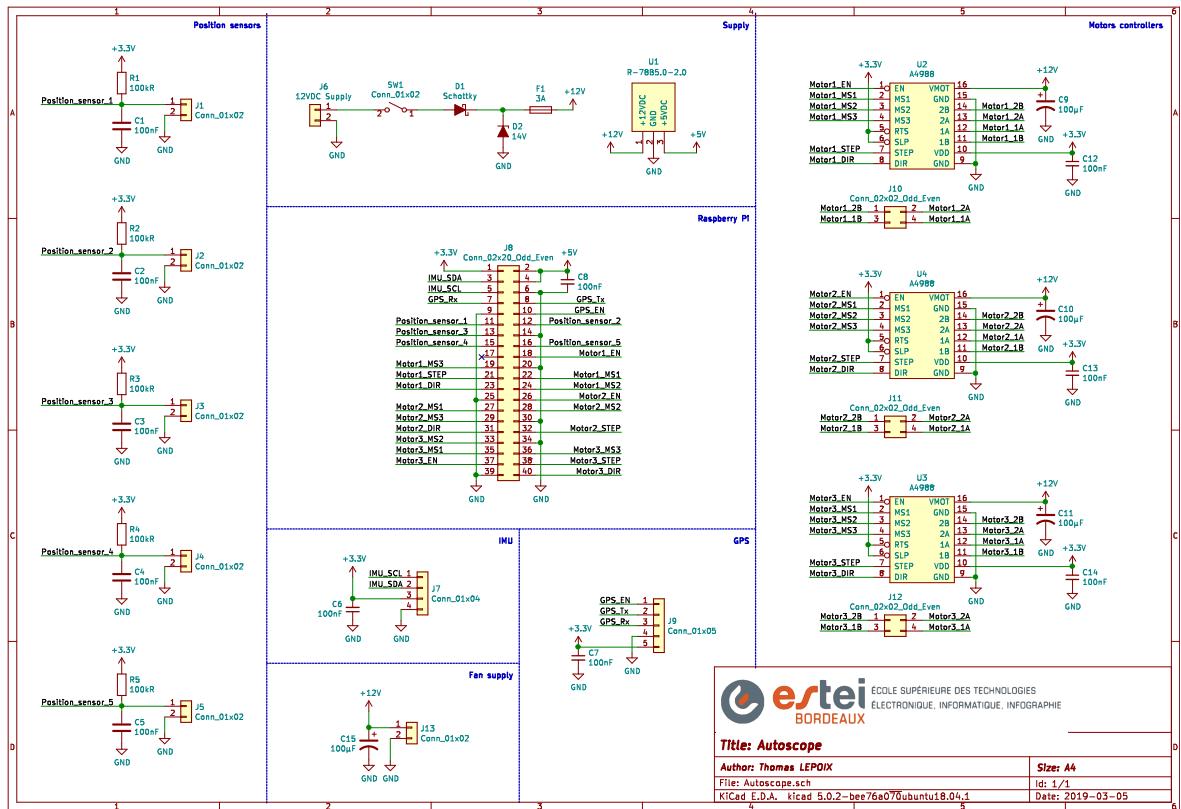


FIGURE 3.1 – Schéma structurel de la carte du télescope

### 3.2 Bon de commande

Figurent sur le bon de commande tous les éléments faisant partie du télescope, on observe ainsi le prix de chaque partie :

- Éléments optiques : 454€
  - Motorisation : 64,7€
  - Système électronique : 197,49€ dont :
    - Raspberry-Pi et carte mémoire : 39,09€
    - Carte électronique Autoscope : 158,40€

Le vert représente les composants déjà reçus, le jaune ceux commandés et le orange ceux qui seront considérés plus tard.

FIGURE 3.2 – Bon de commande du matériel du télescope

### 3.3 Fichiers de fabrication

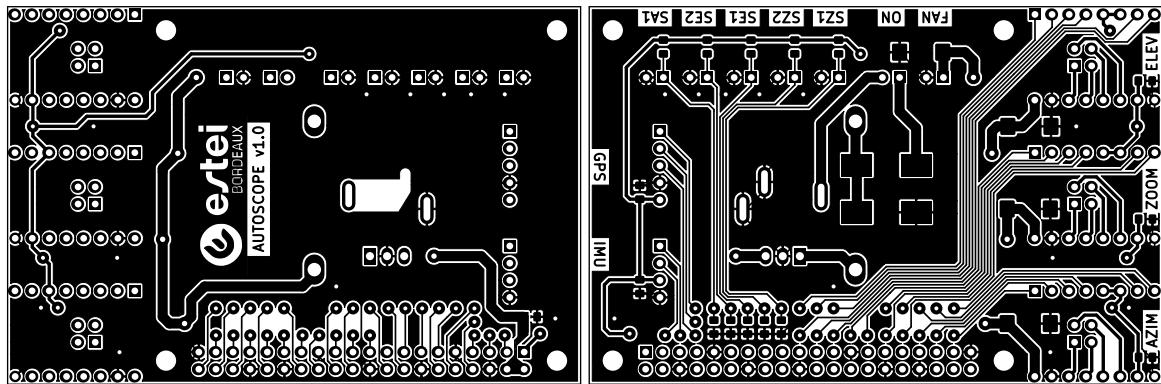


FIGURE 3.3 – Faces supérieure et inférieure du typon de la carte  
(respectivement vues de dessus et de dessous)

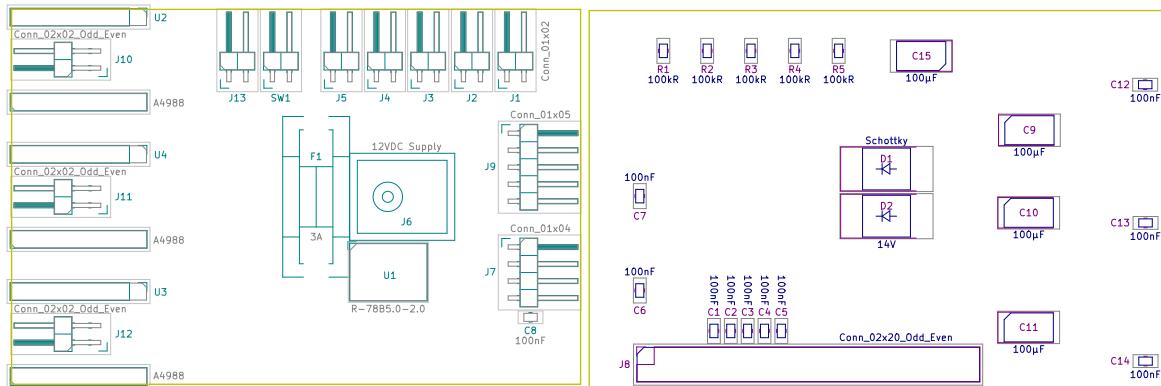


FIGURE 3.4 – Faces supérieure et inférieure de l’implantation des composants  
(respectivement vues de dessus et de dessous)

### 3.4 Modélisation 3D

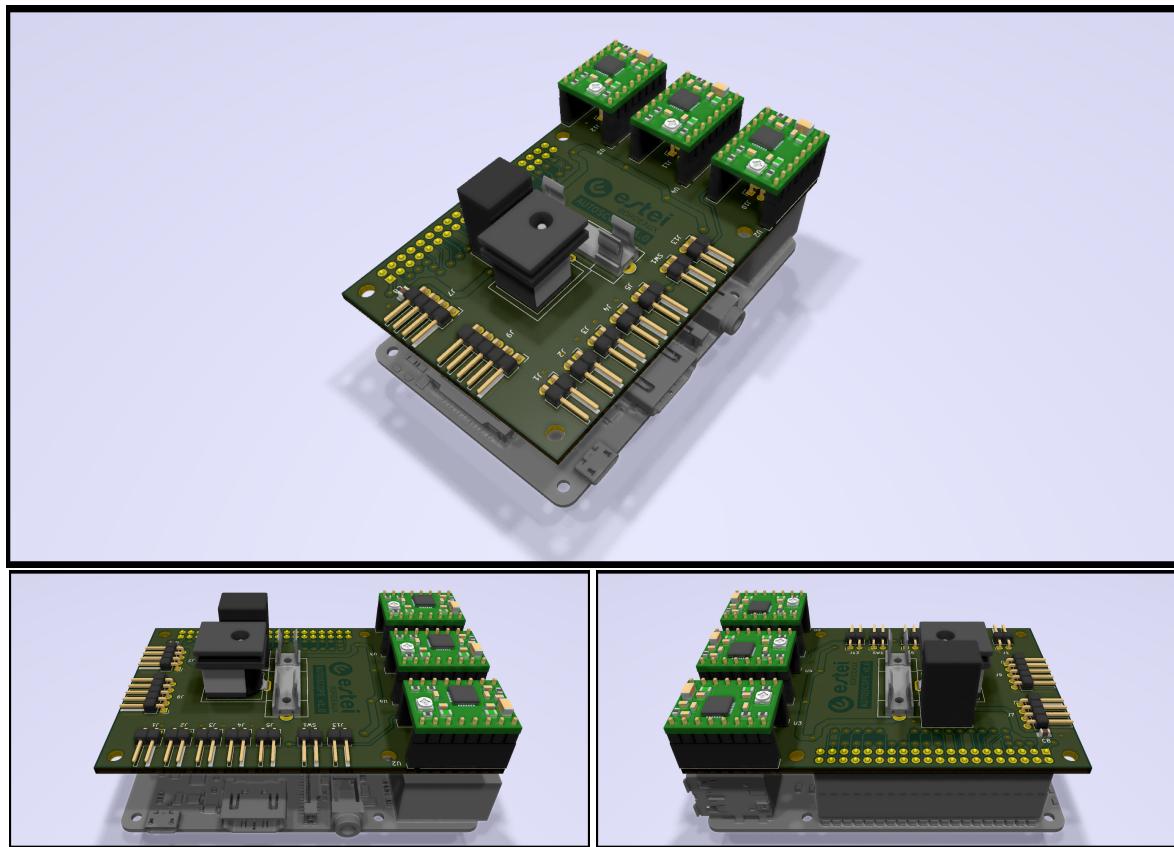


FIGURE 3.5 – Modélisation 3D de la carte pluggée sur la Raspberry-Pi

Au delà de l'aspect esthétique, la modélisation 3D permet d'avoir un aperçu du produit fini et de valider ou non le respect de certaines contraintes de design. Ou encore de repérer des vices que l'on ne voit pas forcément lors de la réalisation du typon, voire du schéma.

#### 3.4.1 Allure générale de la carte

Ainsi l'on observe d'abord l'allure générale de la carte :

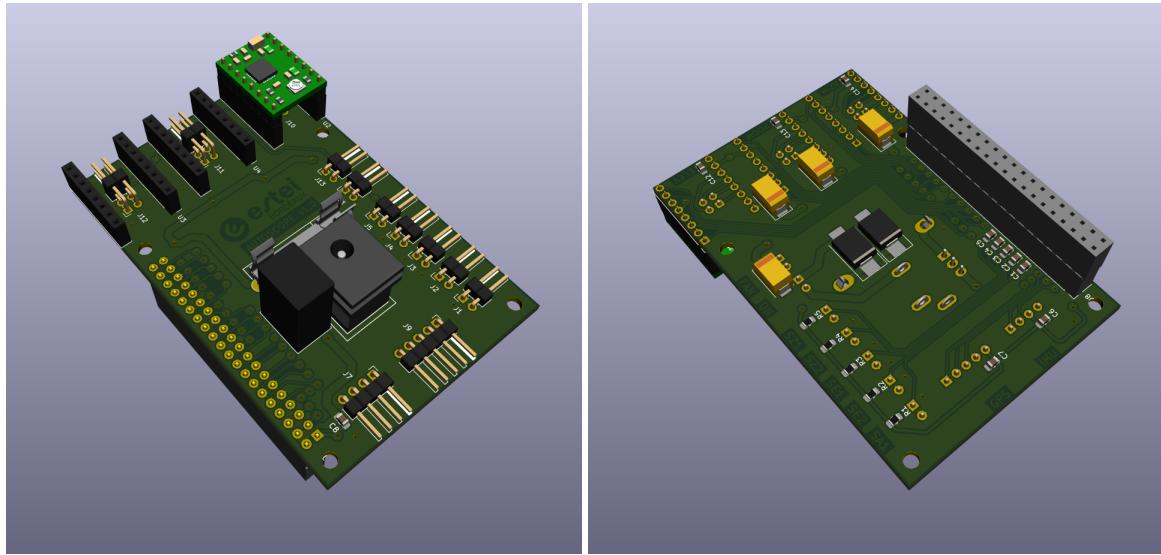


FIGURE 3.6 – Modélisation 3D de la carte avec et sans modules de contrôle moteur

### 3.4.2 Connecteurs des moteurs

Ensuite l'on peut s'assurer de la pertinence de disposer les connecteurs des moteurs sous leurs contrôleurs :

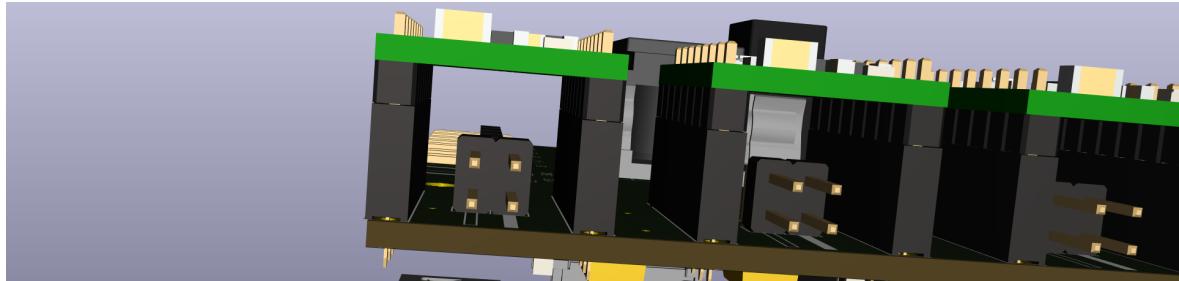


FIGURE 3.7 – Modélisation 3D de la carte : zoom sur les connecteurs moteurs

### 3.4.3 Emboîtement des cartes

Puis on peut vérifier le correct emboîtement des cartes pour un espace les séparant de 19mm, la longueur des entretoises utilisées. En particulier au niveau des condensateurs les plus volumineux :

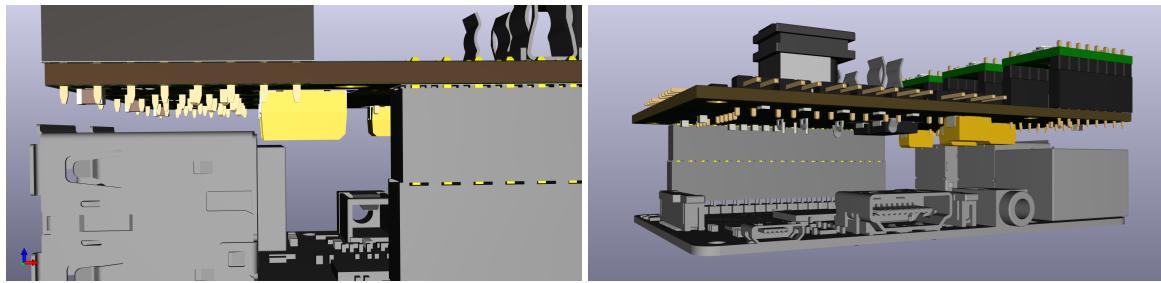


FIGURE 3.8 – Modélisation 3D de la carte : zoom sur l'emboîtement des cartes

### 3.4.4 Repères des connecteurs

Enfin, question de commodité, des repères ont été ajoutés pour indiquer le rôle de chaque connecteur. Ceux-ci se trouvent au niveau du connecteur associé, sur la face opposée du PCB, c'est-à-dire, sur la face côté Raspberry-Pi.

Placées sur le télescope, la Raspberry-Pi étant vouée à être sur le dessus, les repères sont sensés être visibles par l'utilisateur.

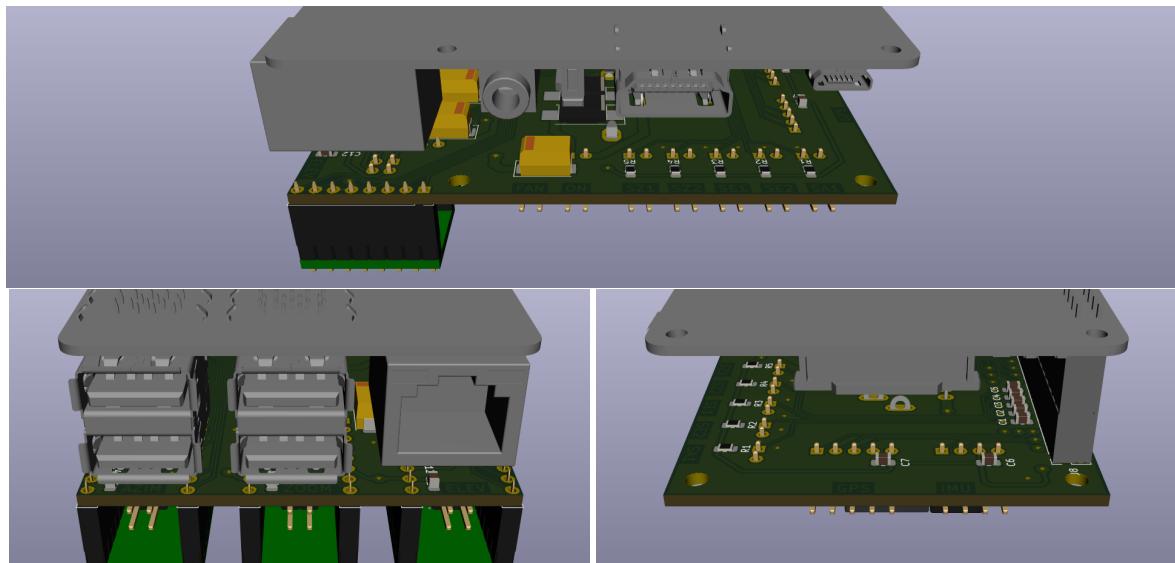


FIGURE 3.9 – Modélisation 3D de la carte : zoom sur les repères des connecteurs

- AZIM : Connecteur du moteur d'azimut.
- ZOOM : Connecteur du moteur de zoom.
- ELEV : Connecteur du moteur d'élévation.
- FAN : Connecteur d'alimentation du ventilateur de refroidissement du miroir primaire.
- ON : Connecteur d'un interrupteur On/Off d'alimentation du télescope.
- SZ1 : Connecteur du premier interrupteur de butée du moteur de zoom.
- SZ2 : Connecteur du second interrupteur de butée du moteur de zoom.
- SE1 : Connecteur du premier interrupteur de butée du moteur d'élévation.

- SE2 : Connecteur du second interrupteur de butée du moteur d'élévation.
- SA1 : Connecteur de l'unique interrupteur du moteur d'azimut.
- GPS : Connecteur du module GPS.
- IMU : Connecteur du module IMU (centrale inertie).

### 3.4.5 Intégration du modèle 3D au télescope

Il est possible d'exporter le modèle 3D des deux cartes emboîtées et ainsi de l'utiliser comme élément lors de la modélisation du télescope. Ci-dessous un aperçu des cartes grossièrement placées sur le modèle du télescope :

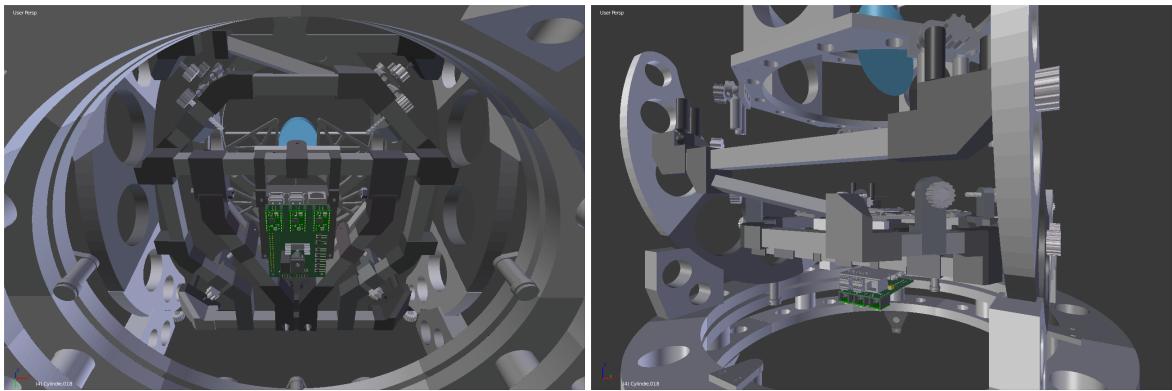


FIGURE 3.10 – Modélisation 3D du télescope grossièrement équipé de ses cartes électroniques

## Chapitre 4

# Système d'exploitation

### 4.1 Splash screen

Le système d'exploitation disposait déjà d'un écran de démarrage aux couleurs de l'école survenant lors de la phase d'initialisation du système.

Un second a été ajouté lors de la phase précédente, le démarrage du kernel, c'est à dire de Linux. Cela n'a pas vraiment d'intérêt pour le télescope si ce n'est un peu d'esthétique lorsqu'il démarre avec un écran branché. Cependant cela met en œuvre un processus qui sera utile pour d'autres choses, la configuration du kernel.

Les éléments des sources de Linux impliqués dans le splash screen sont les suivants :

- Dans le dossier `kernel-source/drivers/video/logo/`
  - `Makefile`
  - `Kconfig`
  - `logo.c`
  - Différents fichiers image, ceux nous intéressant sont au format `logo_*_clut224.ppm`
- Dans le dossier `kernel-source/include/linux/`
  - `linux_logo.h`

Tout d'abord l'on crée une recette yocto de surcharge de la recette kernel utilisée par la `meta-raspberrypi`. Voici l'allure du dossier associé à la recette. À noter que l'arborescence utilisée par une recette de surcharge du kernel est plus sensible qu'une recette de surcharge quelconque.

```

1 ~/yocto/sources/meta-autoscope $ tree recipes-kernel/
2     recipes-kernel/
3         └── linux/
4             ├── linux-raspberrypi/
5                 └── raspberrypi3/
6                     └── linux-raspberrypi_%.bbappend

```

Ensuite l'on crée l'image qui servira d'écran de démarrage, en commençant par lui donner les dimensions souhaitées, ici  $1822 \times 900$ . Il faut alors la convertir au format qui convient :

```

1 jpegtopnm lune-1822x900.jpg | ppmquant 224 | pnmmnoraw > logo_autoscope_clut224.ppm
2 mv logo_autoscope_clut224.ppm
3   ~/yocto/sources/meta-autoscope/recipes-kernel/linux/linux-raspberrypi/raspberrypi3/

```

Puis des lignes sont à ajouter dans les sources du kernel dont la localisation dans l'arborescence yocto est la suivante : `~/yocto/build/tmp/work-shared/raspberrypi3/kernel-source/`

```

include/linux/linux_logo.h :

1  extern const struct linux_logo logo_autoscope_clut224;

    drivers/video/logo/logo.c :

1  #ifdef CONFIG_LOGO_AUTOSCOPE_CLUT224
2      /* Autoscope Linux logo */
3      logo = &logo_autoscope_clut224;
4  #endif

    drivers/video/logo/Makefile :

1  obj-$ (CONFIG_LOGO_AUTOSCOPE_CLUT224)      += logo_autoscope_clut224.o

    drivers/video/logo/Kconfig :

1  config LOGO_AUTOSCOPE_CLUT224
2      bool "224-color Autoscope Linux logo"
3      default y

```

On réalise un patch de ces modifications :

```

1  ~/yocto/build/tmp/work-shared/raspberrypi3/kernel-source $ 
2      git add drivers/video/logo/{Kconfig,Makefile,logo.c} include/linux/linux_logo.h
3      git commit -m "Autoscope logo"
4      git format-patch -1
5      mv 0001-Autoscope-logo.patch
       ↳ ~/yocto/sources/meta-autoscope/recipes-kernel/linux/linux-raspberrypi/raspberrypi3/

```

On configure ensuite le kernel afin d'activer l'écran de démarrage et de choisir celui souhaité. On réalise pour cela un *fragment* de configuration :

`meta-autoscope/recipes-kernel/linux/linux-raspberrypi/raspberrypi3/logo.cfg` :

```

1  CONFIG_LOGO=y
2  CONFIG_LOGO_LINUX_MONO=y
3  CONFIG_LOGO_LINUX_VGA16=y
4  # CONFIG_LOGO_LINUX_CLUT224 is not set
5  CONFIG_LOGO_AUTOSCOPE_CLUT224=y

```

La dernière étape est d'écrire la recette pour prendre en compte toutes ces modifications :

`meta-autoscope/recipes-kernel/linux/linux-raspberrypi_\%.bbappend` :

```

1  FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"
2
3  SRC_URI_append_raspberrypi3 += " \
4      file://0001-Autoscope-logo.patch \
5      file://logo.cfg \
6      file://logo_autoscope_clut224.ppm \
7      "
8
9  do_patchprepend() {

```

```

10 cp ${WORKDIR}/logo_autoscope_clut224.ppm ${S}/drivers/video/logo/
11 }
```

Voici donc l'arborescence associée à la recette :

```

1 ~/yocto/sources/meta-autoscope $ tree recipes-kernel/
2     recipes-kernel/
3         └── linux/
4             ├── linux-raspberrypi/
5                 └── raspberrypi3/
6                     ├── 0001-Autoscope-logo.patch
7                     ├── logo_autoscope_clut224.ppm
8                     └── logo.cfg
9             └── linux-raspberrypi_%.bbappend
```

Après une longue phase de compilation puisque le kernel est recompilé, on observe au démarrage la succession des deux écrans de démarrages :



FIGURE 4.1 – Écrans de démarrage du noyau Linux à droite et du processus d'initialisation à gauche

## 4.2 Hotspot Wifi

Configurer le télescope en Hotspot Wifi est une solution intéressante car ainsi un appareil client peut s'y connecter sans qu'il n'y ait besoin d'une infrastructure existante, un réseau local par exemple.

Pour configurer la Raspberry-Pi en Hotspot, deux choses sont nécessaires :

- Configurer le kernel pour qu'il supporte le mode `tether`.
- Le logiciel `connman`

`meta-autoscope/recipes-kernel/linux/linux-raspberrypi/raspberrypi3/hotspot.cfg`:

```

1 CONFIG_BRIDGE=m
2 CONFIG_IP_NF_TARGET_MASQUERADE=m
3 CONFIG_NETFILTER=m
4 CONFIG_NF_CONNTRACK_IPV4=m
```

```

5 CONFIG_NF_NAT_IPV4=m
6
7 CONFIG_IP_NF_IPTABLES=m
8 CONFIG_IP_MULTIPLE_TABLES=m
9 CONFIG_NETFILTER_NETLINK_ACCT=m
10 CONFIG_NETFILTER_XT_MATCH_NFACCT=m
11 CONFIG_NETFILTER_XT_CONNMARK=m
12 CONFIG_NETFILTER_XT_TARGET_CONNMARK=m
13 CONFIG_NETFILTER_XT_MATCH_CONNMARK=m

meta-autoscope/recipes-kernel/linux/linux-raspberrypi_%.bbappend :

1 SRC_URI_append_raspberrypi3 += " \
2     file://0001-Autoscope-logo.patch \
3     file://logo.cfg \
4     file://hotspot.cfg \
5     file://logo_autoscope_clut224.ppm \
6     "
7

meta-autoscope/recipes-autoscope/images/autoscope-console-image.bb :

1 HOTSPOT = " \
2     connman \
3     connman-client \
4     iptables \
5     "
6
7 IMAGE_INSTALL += " \
8     ${CAMERA} \
9     ${HOTSPOT} \
10   "

```

Si l'on effectue les commandes suivantes, la Raspberry-Pi devient visible sur le réseau :

```

1 root@autoscope ~ #
2     sysctl -w net.ipv4.ip_forward=1
3     connmanctl enable wifi
4     connmanctl tether wifi on Autoscope 123456789

```

Pour automatiser le processus au démarrage, il faut utiliser le paquet `connman-conf` et remplir quelques fichiers de configuration :

`meta-autoscope/recipes-autoscope/images/autoscope-console-image.bb :`

```

1 HOTSPOT = " \
2     connman \
3     connman-client \
4     connman-conf \
5     iptables \
6     "

```

`meta-autoscope/recipes-connectivity/connman-conf/files/main.conf :`

```

1 [General]
2 DefaultAutoConnectTechnologies=wifi
3 TetheringTechnologies=true
4 PersistantTetheringMode=true

```

`meta-autoscope/recipes-connectivity/connman-conf/files/settings :`

```

1 [global]
2 OfflineMode=false
3
4 [WiFi]
5 Enable=true
6 Tethering=true
7 Tethering.Identifier=Autoscope
8 Tethering.Passphrase=123456789
9
10 [Wired]
11 Enable=true
12 Tethering=false
13
14 [P2P]
15 Enable=false
16 Tethering=false

```

`meta-autoscope/recipes-connectivity/connman-conf/connman-conf.bbappend :`

```

1 FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
2
3 SRC_URI += " \
4   file://main.conf \
5   file://settings \
6 "
7
8 FILES_${PN} += "${sysconfdir}/*"
9
10 do_install_append() {
11   install -d ${D}${sysconfdir}/connman/
12   install -m 0755 ${WORKDIR}/main.conf ${D}${sysconfdir}/connman/main.conf
13   install -d ${D}${localstatedir}/lib/connman/
14   install -m 0755 ${WORKDIR}/settings ${D}${localstatedir}/lib/connman/settings
15 }

```

Les lignes ajoutées à `do_install()` permettent de déplacer les fichiers à leur place dans l'arborescence du système :

- `/etc/connman/main.conf`
- `/var/lib/connman/settings`

À noter que sans la ligne `FILES_${PN} += "${sysconfdir}/*"`, Bitbake est incapable de connaître cette variable et donc de déplacer le fichier. La question ne se pose pas pour la variable `localstatedir` puisque l'on trouve la ligne suivante

`FILES_${PN} = "${localstatedir}/* ${datadir}/*"` dans la recette originale :  
`poky/meta/recipes-connectivity/connman-conf/connman-conf.bb`

Quant à la commande `sysctl -w net.ipv4.ip_forward=1` qui n'est pas à effectuer à chaque démarrage, on peut l'automatiser ainsi :

`meta-autoscope/recipes-autoscope/images/autoscope-console-image.bb :`

```

1 hotspot() {
2   echo 'net.ipv4.ip_forward = 1' >> ${IMAGE_ROOTFS}/etc/sysctl.conf
3 }
4
5 ROOTFS_POSTPROCESS_COMMAND += " hotspot; "

```

Ainsi, dès le démarrage de la Raspberry-Pi, le Hotspot **Autoscope** est visible depuis un équipement Wifi. Le mot de passe est **123456789**.

Cela a fonctionné jusqu'à la mise en place de l'écran de démarrage qui passe également par une configuration du kernel. Un conflit doit exister mais n'a pas encore été repéré et arrangé.

### 4.3 Serveur FTP

Plusieurs serveurs FTP sont disponibles dans Poky, **vsftpd** semble être une bonne solution pour sa légèreté, sa fiabilité et ses possibilités de configuration.

**meta-autoscope/recipes-autoscope/images/autoscope-console-image.bb** :

```

1  FTP = " \
2    vsftpd \
3  "
4
5 IMAGE_INSTALL += " \
6   ${CAMERA} \
7   ${HOTSPOT} \
8   ${FTP} \
9 "

```

**meta-autoscope/recipes-connectivity/vsftpd/vsftpd\_\%.bbappend** :

```

1 FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
```

**meta-autoscope/recipes-connectivity/vsftpd/files/vsftpd.user\_list** :

```

1 autoscope
```

**meta-autoscope/recipes-connectivity/vsftpd/files/vsftpd.conf** :

```

1 listen=YES
2 anonymous_enable=NO
3 local_enable=YES
4 write_enable=YES
5 local_umask=022
6 dirmessage_enable=YES
7 xferlog_enable=YES
8 connect_from_port_20=YES
9 xferlog_std_format=YES
10 ftpd_banner=Welcome to Autoscope FTP service.
11 ls_recurse_enable=YES
12 pam_service_name=vsftpd
13 userlist_deny=NO
14 userlist_enable=YES
15 use_localtime=YES
16 chroot_local_user=YES
17 allow_writeable_chroot=YES
18 tcp_wrappers=YES
19 user_sub_token=$USER
20 local_root=/home/$USER
```

Note : Des commentaires explicatifs figurent dans le fichier **vsftpd.conf** mais ne sont pas affichés ici.

Le serveur est alors accessible depuis un navigateur web via l'URL **ftp://<ip.raspberry.pi>** ou plus simplement la commande **ftp autoscope@<ip.raspberry.pi>**. Le mot de passe de l'utilisateur **autoscope** est demandé.

**autoscope** est le seul utilisateur autorisé à accéder au serveur FTP.  
De plus le serveur se lance automatiquement lors de la phase d'initialisation du système.

## 4.4 Driver helloworld

Ce driver minimaliste a pour but de préparer le terrain pour les drivers des moteurs, du GPS et de l'IMU.

**hello.c** :

```

1 #include <linux/module.h>
2
3 int init_module(void)
4 {
5     printk("Hello World!\n");
6     return 0;
7 }
8
9 void cleanup_module(void)
10 {
11     printk("Goodbye Cruel World!\n");
12 }
13
14 MODULE_LICENSE("GPL");

```

**Makefile** :

```

1 obj-m := hello.o
2
3 SRC := $(shell pwd)
4
5 all:
6     $(MAKE) -C $(KERNEL_SRC) M=$(SRC)
7
8 modules_install:
9     $(MAKE) -C $(KERNEL_SRC) M=$(SRC) modules_install
10
11 clean:
12     rm -f *.o *~ core .depend .*.*.cmd *.ko *.mod.c
13     rm -f Module.markers Module.symvers modules.order
14     rm -rf .tmp_versions Modules.symvers

```

Un fichier **COPYING** contient une licence GPL.

Tous ces fichiers figurent sur une branche dédiée **hello\_mod** du dépôt :  
[github.com/thibaudledo/Autoscope](https://github.com/thibaudledo/Autoscope).

**meta-autoscope/recipes-test/hello-mod/hello-mod\_git.bb** :

```

1 SUMMARY = "Example of how to build an external Linux kernel module"
2 LICENSE = "GPLv2"
3 LIC_FILES_CHKSUM =
4     file://${COREBASE}/meta/COPYING.GPLv2;md5=751419260aa954499f7abaabaa882bbe"
5 inherit module
6
7 SRC_URI = "git://github.com/thibaudledo/Autoscope;protocol=git;branch=hello_mod"
8
9 SRCREV = "${AUTOREV}"
10 S = "${WORKDIR}/git"
11
12 RPROVIDES_${PN} += "kernel-module-hello"

```

```
meta-autoscope/recipes-autoscope/images/autoscope-console-image.bb :
```

```
1 IMAGE_INSTALL += " \
2   ${CAMERA} \
3   ${HOTSPOT} \
4   ${FTP} \
5   hello-mod \
6 "
```

Ainsi avec les commandes suivantes on observe les messages d'init et de cleanup s'afficher :

```
1 root@autoscope ~ #
2   modprobe hello
3   rmmod hello
```

## 4.5 Étude du transfert du flux vidéo sur le réseau

Pour l'instant seul un test a été réalisé. Il s'agit d'afficher sur un ordinateur distant le flux vidéo filmé par la Raspberry-Pi. On utilise pour cela **netcat**.

```
1 ~ $ 
2   nc -l -p 5001 | /usr/bin/mplayer -fps 10 -cache 1024 -
3 
4 root@autoscope ~ #
5   raspivid -fps 10 -t 0 -o - | nc <ip.de.l'ordinateur> 5001
```

La procédure fonctionne, le lecteur **mplayer** s'ouvre et lit la vidéo, toutefois malgré le taux de rafraîchissement très faible ( $10\text{fps}$ ), une latence particulièrement longue existe, de l'ordre de 2 à 10 secondes.

D'autres solutions plus performantes existent sûrement, en particulier des solutions de plus bas niveau intégrables à un programme C ou C++.

## Chapitre 5

# Travail restant

### 5.1 Hardware

Il reste pour cette partie la fabrication et la validation de la carte, ainsi que le câblage sur le télescope. Ces taches à défaut d'être difficiles sont assez chronophages.

### 5.2 Système d'exploitation

Le plus urgent est sans doute de réparer la configuration du Hotspot Wifi ainsi que de mettre en place une solution fiable pour transférer le flux vidéo de la caméra.

Ensuite, l'intégration des drivers et du logiciel principal pourront demander quelques configurations au niveau du kernel et du device tree.

Enfin, d'autres taches moins urgentes sont à réaliser, la mise en place d'un pare-feu et la sécurisation du système, la mise en place d'un système de mise à jour, sans doute **Mender** ou **Swupdate**. Puis diverses optimisations du système d'exploitation, retirer des choses inutiles, comme le daemon gérant le Bluetooth.

### 5.3 Software

Des travaux de réflexion sur les algorithmes à mettre en place pour mouvoir le télescope et calibrer ses actionneurs ont été réalisés. Ceux-ci seront à approfondir conjointement aux autres membres du groupe pour écrire le logiciel principal du télescope dont des briques de base apparaissent par différentes extrémités.

**Troisième partie**

**Thomas ABGRALL**

## Chapitre 6

# Driver de la commande des moteurs

### 6.1 Rappel

Le driver moteur a pour mission de permettre de contrôler les moteurs de rotation, inclinaison et zoom du télescope. Mais également assurer la sécurité du système de pilotage à bas niveau. Le système de sécurité consiste à interrompre le mouvement d'inclinaison ou de zoom si ils atteignent leur fin de course.

### 6.2 Fonctionnalités

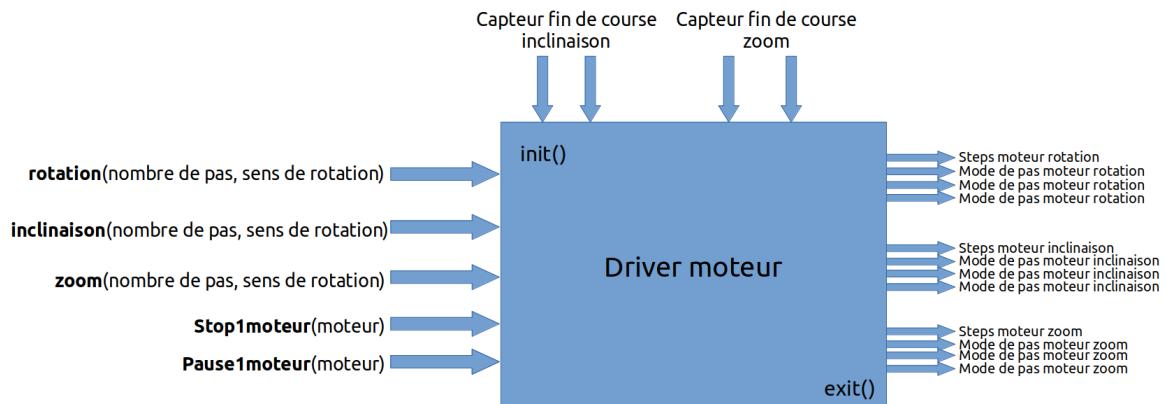


FIGURE 6.1 – Schéma des entrées / sorties du driver moteur

Le driver va permettre l'utilisation des fonctions qui se trouvent à gauche du bloc "driver moteur". Chacune des fonctions agit sur les sorties. Les capteurs de fin de course servent de sécurité, cette partie est indépendante du reste et arrête le moteur qui a atteint sa position de butée. Les sorties "mode de pas moteur..." permettent de choisir dans quel mode de rotation doit tourner le moteur (pas complet, demi-pas, quart de pas, huitième de pas, seizième de pas). Réduire la division de pas permet d'affiner la rotation afin d'atteindre l'angle le plus exact possible à ce qui a été commandé.

## 6.3 Câblage

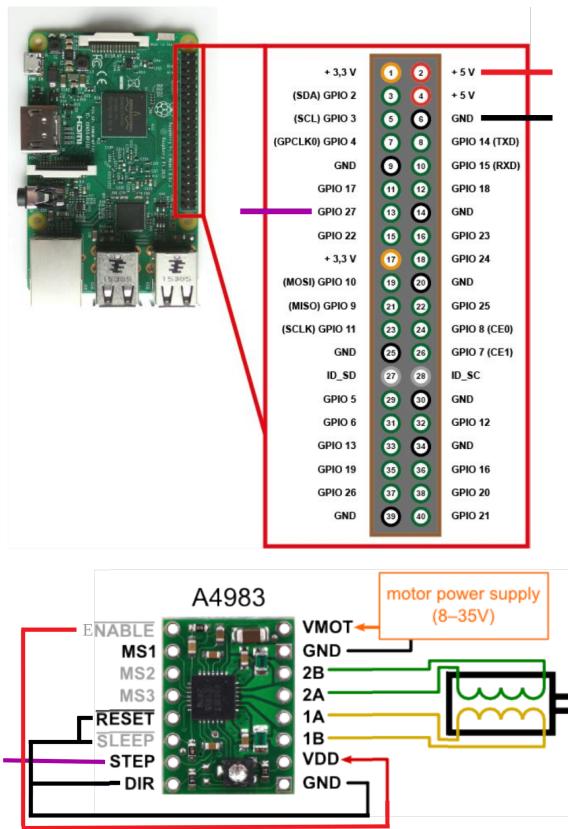


FIGURE 6.2 – Schéma de câblage d'un contrôleur moteur sur la Raspberry Pi

## 6.4 Avancement du programme

La première version du driver qui est validé depuis fin janvier permet :

- Initialiser le driver
- Lancer la rotation (à 5ms max par période de step)
- Lancer l'inclinaison (à 5ms max par période de step)
- Stopper les moteurs grâce aux capteurs de fin de course

À suivre :

- La partie qui consiste à changer de mode de pas est codé mais pas testé.
- Étendre le système pour gérer le zoom.

## 6.5 Procédure de test

Pour valider mon driver, je l'ai testé avec une Raspberry Pi, un contrôleur moteur et un moteur.

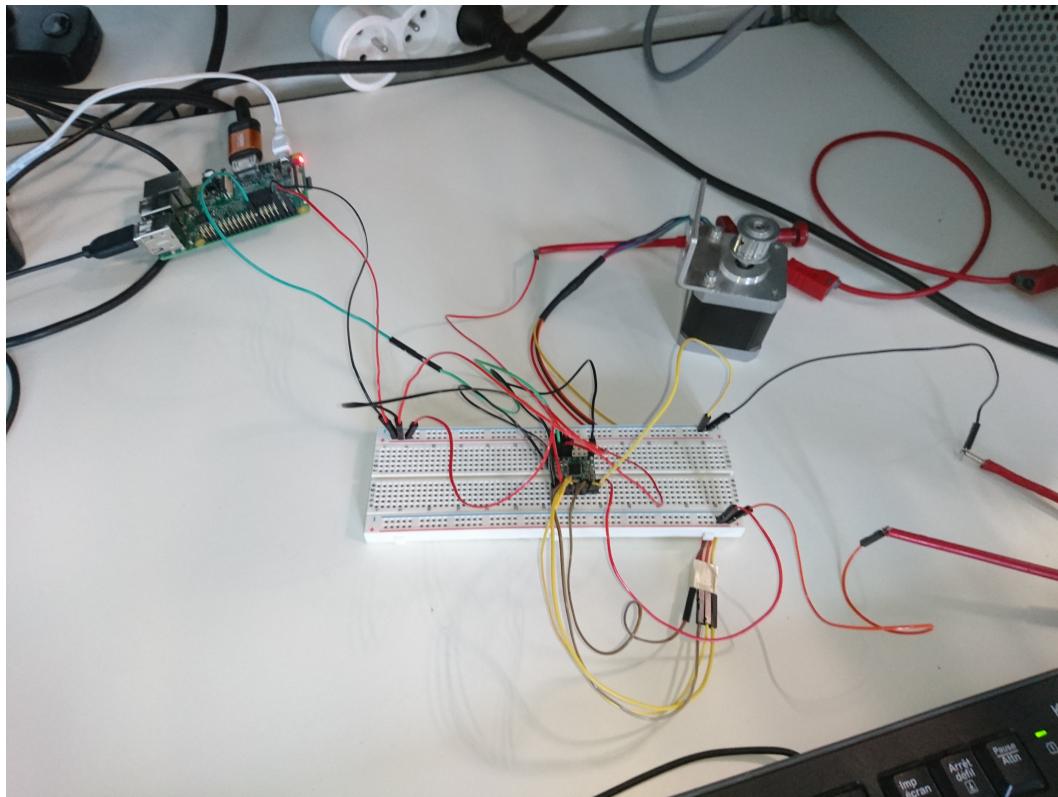


FIGURE 6.3 – Photo du banc de test

J'ai pu valider les fonctionnalités développées, cependant la vitesse max atteinte par le moteur contrôlé par le driver est inférieure à celle que l'on peut atteindre en contrôlant le moteur avec un générateur basse fréquence. Voici le résultat de la commande envoyée par la Raspberry Pi par la pin Step au contrôleur moteur pour faire avancer à chaque front montant le moteur de 1 pas.

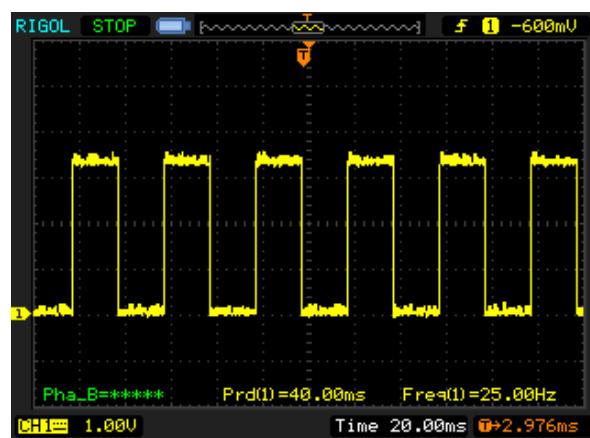


FIGURE 6.4 – Mesure à l'oscilloscope de la commande moteur générée

Après avoir étudié les résultats avec d'autres professeurs, j'ai conclu que la période minimale que je puisse atteindre soit de  $20ms$  est normal. Cette limitation est due au fait que mon driver n'a pas la priorité des ressources dans le Linux. Et que le Linux a un cycle lecture/écriture des entrées/sorties limité. Pour améliorer cela il faudrait passer sur un Linux temps réel.

## 6.6 Interactions avec le driver

Le driver devra pourvoir être utilisé à partir d'une application situé dans le user-space du linux embarqué. Pour cela j'ai étudié et testé plusieurs solutions.

Le première qui m'est venu à l'idée était d'utiliser l'IPC (inter process communication).

Une seconde plus archaïque m'est venu à l'idée, celle de partager un fichier entre le programme dans le user-space et du driver. Pour que le programme dans le user-space puisse transférer ses ordres. Cependant je trouve cette solution pas très sécuritaire.

La troisième solution qui m'a été proposé par Vincent Poulailleau est d'utiliser la fonction ioctl. C'est cette solution que j'ai retenu car elle est une technique très rodé dans le domaine car elle était déjà utilisé à la version 7 d'UNIX.

Je suis maintenant en phase de développement de cette fonction pour l'ajouter au driver existant.

## Chapitre 7

# Fabrication du télescope

Nous avons commencé à réaliser l'assemblage du télescope. Nous avons réalisé les couronnes et croissants. Nous avons été confronté à la mauvaise qualité du pistolet à colle qui en faisait débordé partout lorsque je m'en servais (correctement).

## Chapitre 8

# Travail à venir

- Rendre le driver compatible device tree
- Réalisation du programme (user-space) qui utilisera le driver
  - Algorithmie des mouvements
  - Calibration du télescope