



estei
BORDEAUX

ÉCOLE SUPÉRIEURE DES TECHNOLOGIES
ÉLECTRONIQUE, INFORMATIQUE, INFOGRAPHIE

PROJET DE MASTER 2

Autoscope Compte rendu final

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

17 avril 2019

Table des matières

Table des matières	1
I Partie de groupe	2
II Thomas LEPOIX	3
1 Hardware	4
1.1 Architecture	4
1.2 Conception du circuit	5
1.3 Contraintes de design	6
1.3.1 Contraintes électromagnétiques	6
1.3.2 Contraintes mécaniques	6
1.4 Bon de commande	9
1.5 Fichiers de fabrication	10
1.6 Modélisation 3D	11
1.6.1 Allure générale de la carte	11
1.6.2 Connecteurs des moteurs	12
1.6.3 Emboîtement des cartes	12
1.6.4 Repères des connecteurs	13
1.6.5 Intégration du modèle 3D au télescope	14
1.7 Fabrication	14
1.8 Validation	14
1.8.1 Tests de continuité et de fuite de courant	14
1.8.2 Test de l'environnement des interrupteurs de butée	14
1.8.3 Test de l'alimentation	15
2 Système d'exploitation	17
2.1 Support de la caméra	17
2.2 Étude du transfert du flux vidéo sur le réseau	18
2.3 Splash screen	18
2.4 Hotspot Wifi	21
2.5 Serveur FTP	24
2.6 Driver helloworld	25
2.7 Support de la liaison UART	27
2.8 Support du bus I2C	28

Première partie

Partie de groupe

Deuxième partie

Thomas LEPOIX

Chapitre 1

Hardware

1.1 Architecture

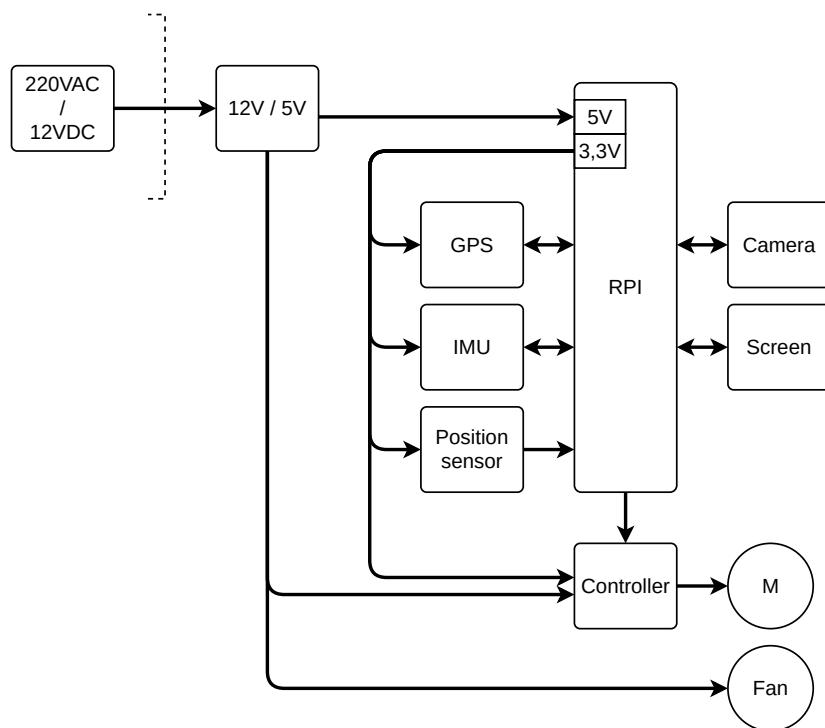


FIGURE 1.1 – Schéma structurel de premier niveau du télescope

La première question de l'étude structurelle du télescope est celle de l'alimentation.

Le télescope sera raccordé au secteur par un module externe 220VAC/12VDC. Ensuite les moteurs seront alimentés en 12V via leurs contrôleurs respectifs et la Raspberry-Pi sera alimentée en 5V. Un convertisseur 12V/5V est donc à ajouter.

Le télescope est également doté d'un ventilateur situé sous le miroir primaire et servant à le maintenir à température constante. Celui-ci sera alimenté en 12V.

Tous les autres éléments seront alimentés en 3,3V par la Raspberry-Pi. Il est toutefois important de s'assurer que la consommation maximale en courant de ces éléments ne dépasse pas ce que peut fournir la Raspberry-Pi, à savoir 500mA.

- Contrôleurs des moteurs ($\times 3$) : $8mA$
- GPS : $25mA$
- IMU : $3,7mA$
- Capteurs de position des moteurs ($\times 5$) : $33\mu A$

La consommation en courant totale des périphériques de la Raspberry-Pi est largement inférieure à $500mA$, le montage est donc réalisable sans risque de dysfonctionnement ou de dommages.

1.2 Conception du circuit

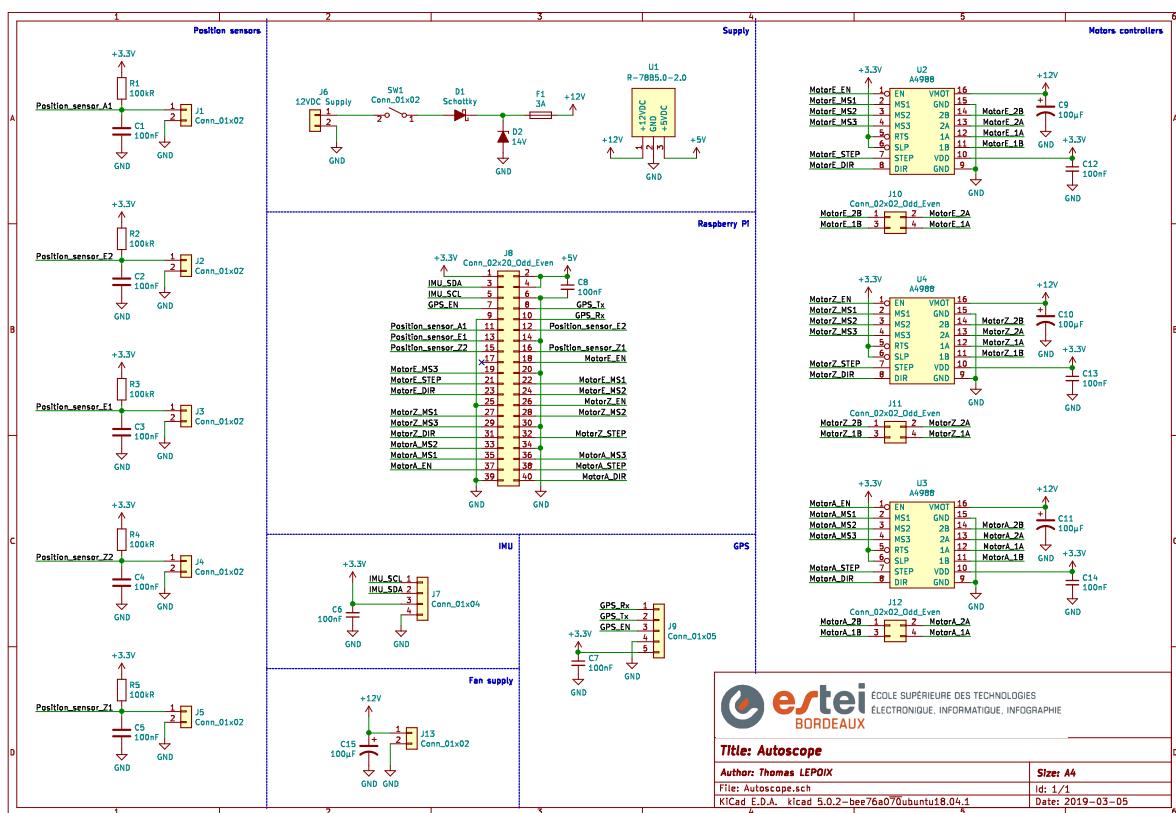


FIGURE 1.2 – Schéma structurel de la carte du télescope

Ce schéma ne présente pas de subtilité particulière, la plupart des composants étant des connecteurs.

L'alimentation est composée de :

- Un interrupteur d'allumage
- Une diode polarisante

- Une diode zener (TVS) protégeant des surtensions
- Un fusible protégeant des surintensités
- Un convertisseur DC/DC intégré

L'environnement des boutons poussoirs servant de capteurs de butée aux mouvements du télescope est le suivant :

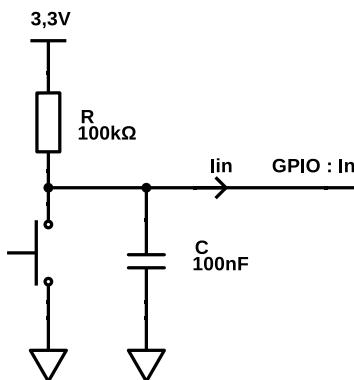


FIGURE 1.3 – Schéma de l'environnement des capteurs de butée

La valeur élevée des résistances de pullup $100\text{k}\Omega$ a pour but de réduire au maximum le courant consommé lors de l'appui, à $33\mu\text{A}$. Le courant prélevé par l'entrée GPIO de la Raspberry-Pi est de l'ordre de $0,5\mu\text{A}$.

Les condensateurs de 100nF permettent de filtrer les parasites générés par les rebonds propres aux boutons ainsi que les perturbations électromagnétiques.

1.3 Contraintes de design

1.3.1 Contraintes électromagnétiques

La première contrainte vient de la proximité du système électronique de deux moteurs, ceux-ci générant d'importantes perturbations électromagnétiques. Cela peut être particulièrement dérangeant pour le fonctionnement de la centrale inertuelle et du GPS.

La solution la plus simple et efficace est de déporter ces deux modules le long de la structure du télescope.

1.3.2 Contraintes mécaniques

Ensuite viennent les contraintes mécaniques de l'association de la carte à la Raspberry-Pi.

Tout d'abord, l'emplacement du connecteur et des fixations sont à prendre en compte.

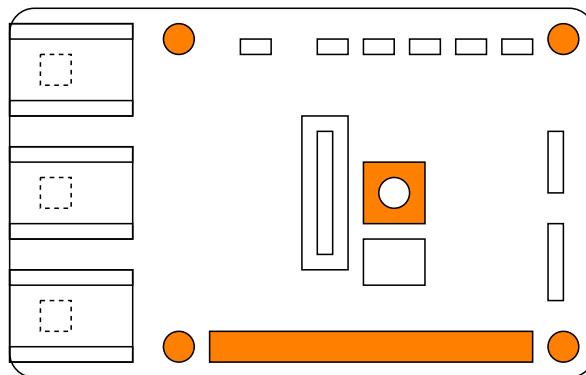


FIGURE 1.4 – Schéma mécanique de la carte vue de dessus

Le connecteur d'alimentation, centré sur la carte, est un connecteur cylindrique comme ceux des ordinateurs portables. Le télescope étant amené à tourner sur lui-même, ce connecteur devrait permettre le mouvement tout en empêchant le câble d'alimentation de s'emmêler ou de se détériorer.



FIGURE 1.5 – Connecteurs d'alimentation utilisés

Puis concernant la distance entre la carte et la Raspberry-Pi, la hauteur des plus hauts éléments de la Raspberry-Pi est à prendre en compte. Ainsi que la hauteur de certains condensateurs de la carte, ne pouvant donc être placés n'importe où.

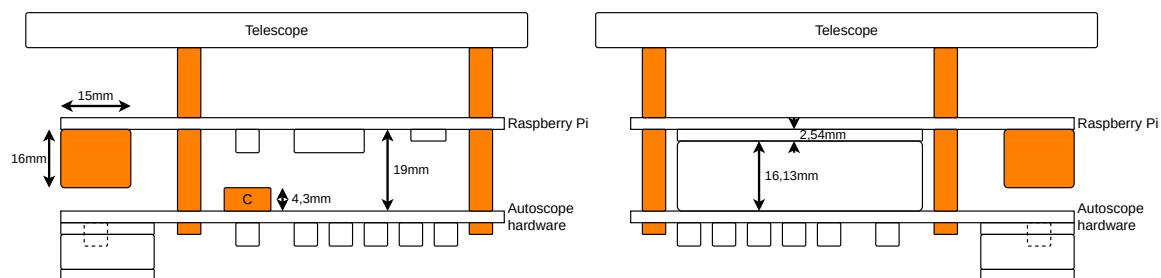


FIGURE 1.6 – Schéma mécanique de la carte vue de profil

Il faudra de plus utiliser un connecteur particulièrement haut (16,13mm) pour relier la carte à la Raspberry-Pi.

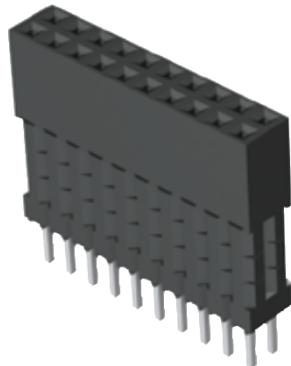


FIGURE 1.7 – Type de header utilisé

1.5 Fichiers de fabrication

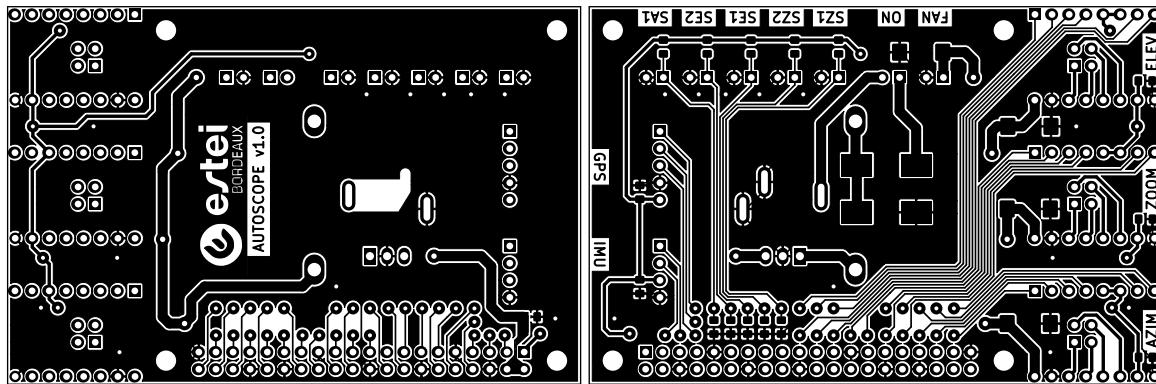


FIGURE 1.9 – Faces supérieure et inférieure du typon de la carte
(respectivement vues de dessus et de dessous)

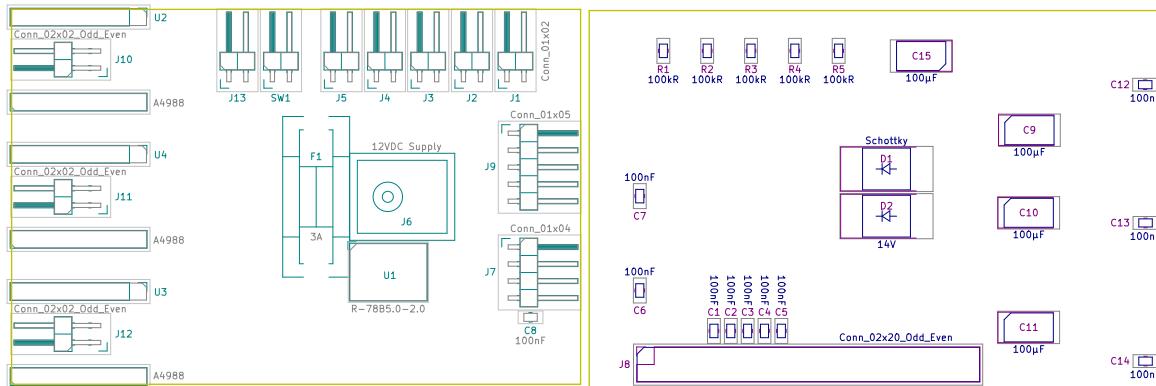


FIGURE 1.10 – Faces supérieure et inférieure de l’implantation des composants
(respectivement vues de dessus et de dessous)

1.6 Modélisation 3D

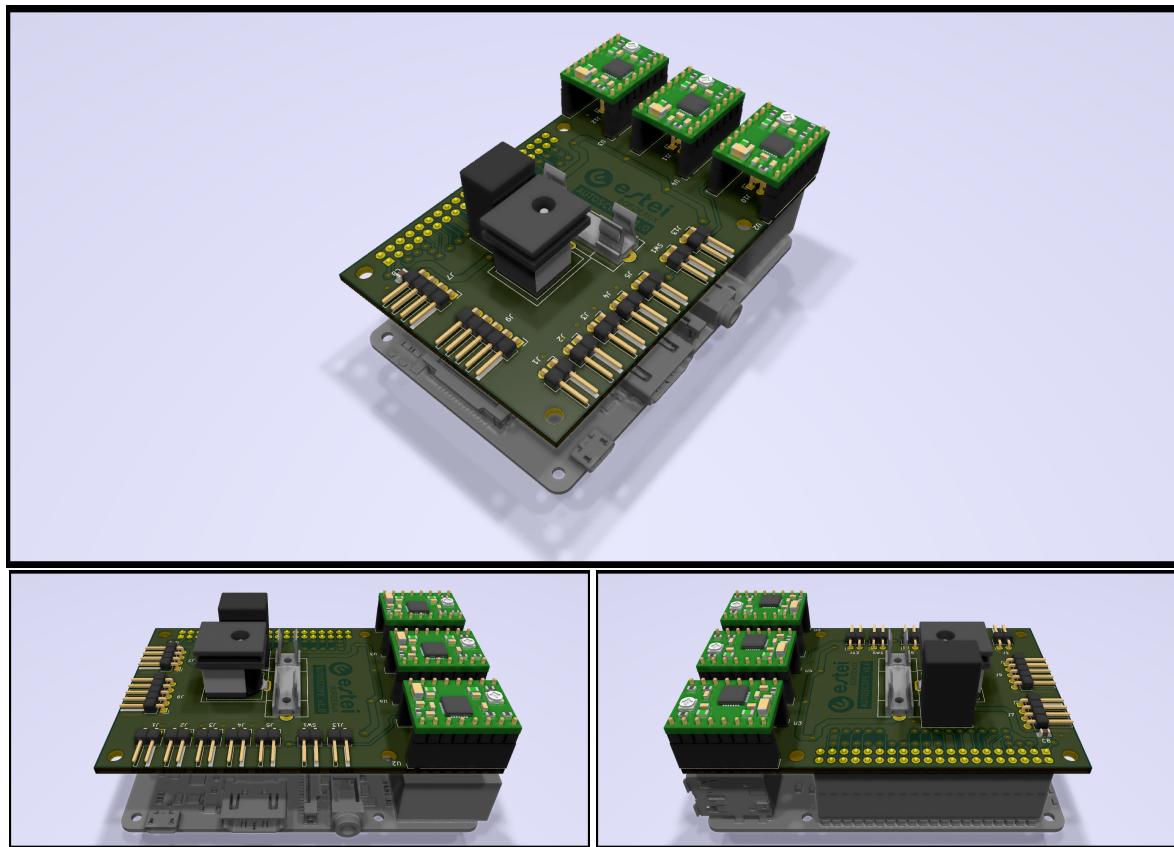


FIGURE 1.11 – Modélisation 3D de la carte pluggée sur la Raspberry-Pi

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.

1.6.1 Allure générale de la carte

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

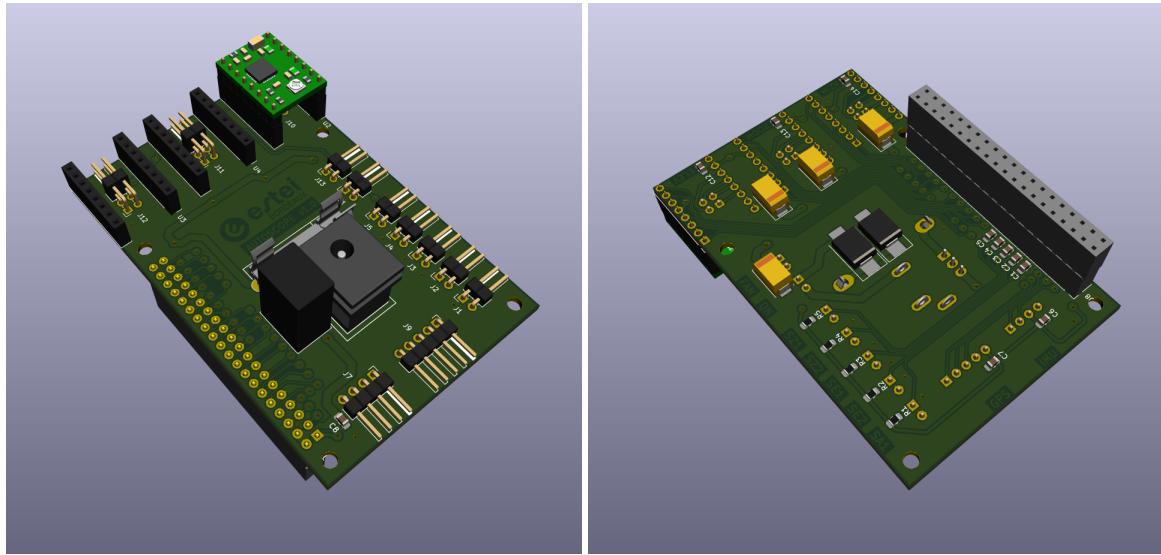


FIGURE 1.12 – Modélisation 3D de la carte avec et sans modules de contrôle moteur

1.6.2 Connecteurs des moteurs

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

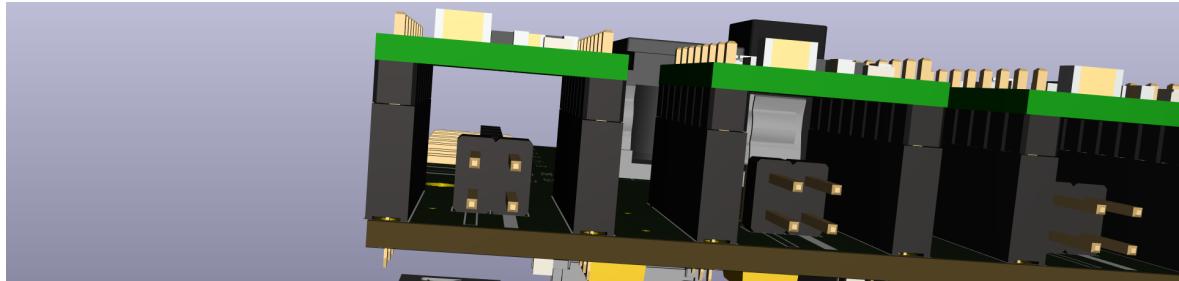


FIGURE 1.13 – Modélisation 3D de la carte : zoom sur les connecteurs moteurs

1.6.3 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 :

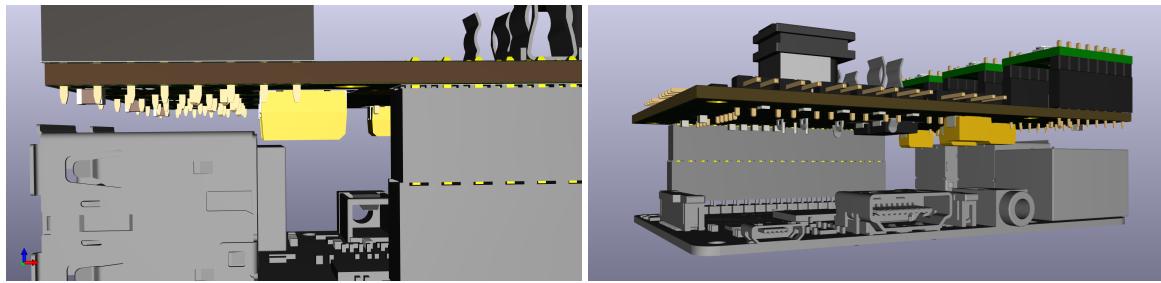


FIGURE 1.14 – Modélisation 3D de la carte : zoom sur l'emboîtement des cartes

1.6.4 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.

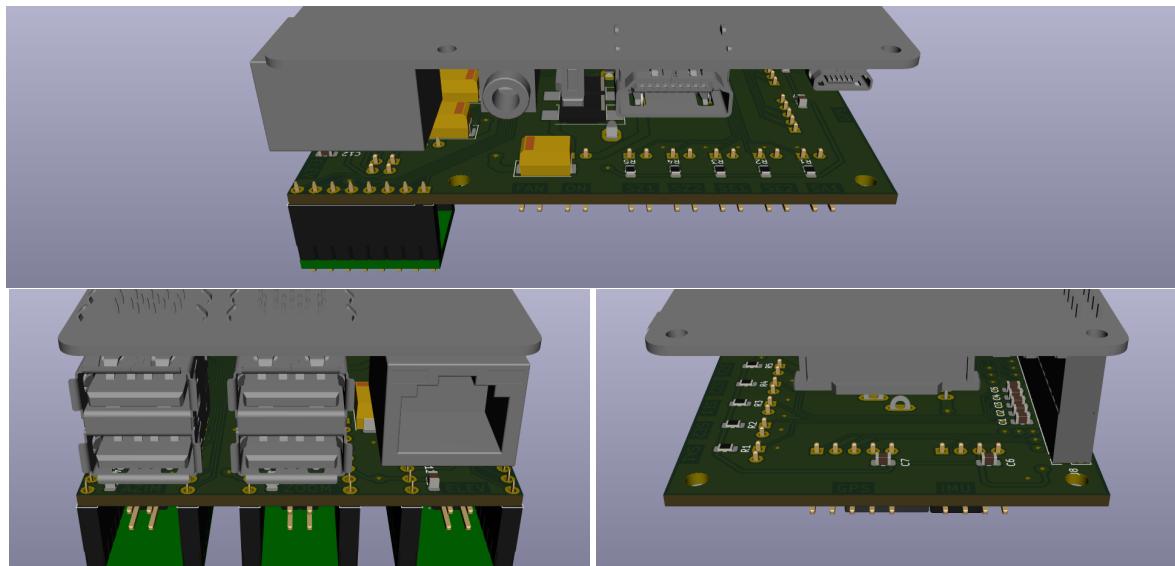


FIGURE 1.15 – Modélisation 3D de la carte : zoom sur les repères des connecteurs

- 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 inertielle).

1.6.5 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 :

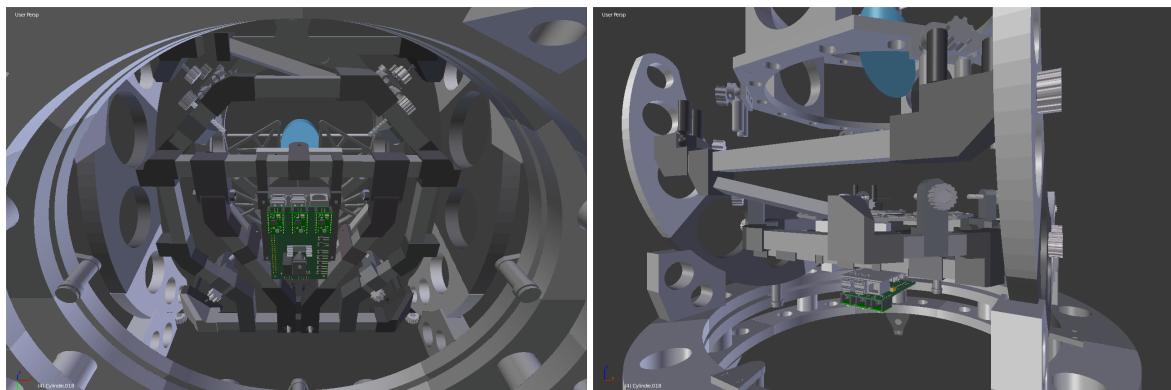


FIGURE 1.16 – Modélisation 3D du télescope grossièrement équipé de ses cartes électroniques

1.7 Fabrication

1.8 Validation

1.8.1 Tests de continuité et de fuite de courant

Ces tests, bien qu'élémentaires, permettent de valider la fonctionnement de la quasi-intégralité de la carte. Une relecture attentive du schéma de la carte, en particulier du cablage des connecteurs est impérative.

Il s'agit de vérifier pour chaque broche du connecteur 40 broches que le courant se propage correctement jusqu'à la broche correspondante d'un autre connecteur, à l'autre extrémité de la piste. Vérifier également qu'aucune fuite de courant ou court-circuit n'existe vers les pistes voisines ou la masse.

1.8.2 Test de l'environnement des interrupteurs de butée

Pour valider le fonctionnement des boutons, on alimente la carte et on mesure la tension de sortie à l'état enfoncé et relâché de chaque bouton :

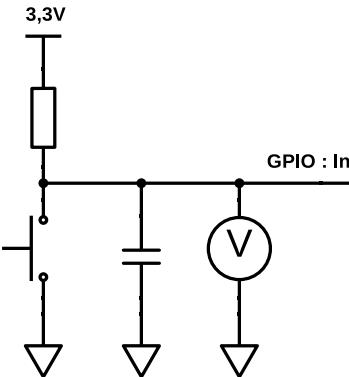


FIGURE 1.17 – Procédure de test de l'environnement des boutons

Button	Vin (V)	Vout ↑ (V)	Vout ↓ (V)
Azimut 1	3,33	3,26	0,00
Elevation 2	3,33	3,26	0,00
Elevation 1	3,33	3,26	0,00
Zoom 2	3,33	3,26	0,00
Zoom 1	3,33	3,26	0,00

FIGURE 1.18 – Mesure de la tension de sortie des boutons

Logique	CMOS (V)
0	0 – 1,1
1	2,2 – 3,3

FIGURE 1.19 – Niveaux de tensions de la logique CMOS

Les niveaux électriques haut et bas sont largement inclus dans les plages de tolérances CMOS, les boutons sont fonctionnels.

1.8.3 Test de l'alimentation

Pour tester l'alimentation, on vérifie d'abord le fonctionnement du convertisseur DC/DC 12V/5V par le montage suivant :

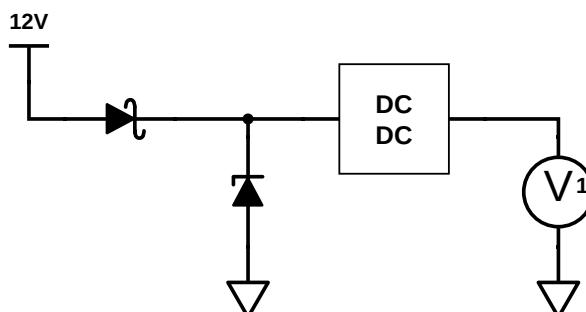


FIGURE 1.20 – Procédure de test du convertisseur DC/DC

Pour une tension d'entrée $V_{in} = 12,10V$, l'on a une tension de sortie $V_1 = 5,02V$. Le convertisseur fonctionne.

Ensuite, pour tester le système de protection aux surtensions, on ajoute une résistance pour limiter le courant et on augmente la tension d'alimentation. Pour rappel, cette protection est davantage destinée au ventilateur alimenté en 12V qu'au convertisseur DC/DC pouvant tolérer des entrées jusqu'à 32V.

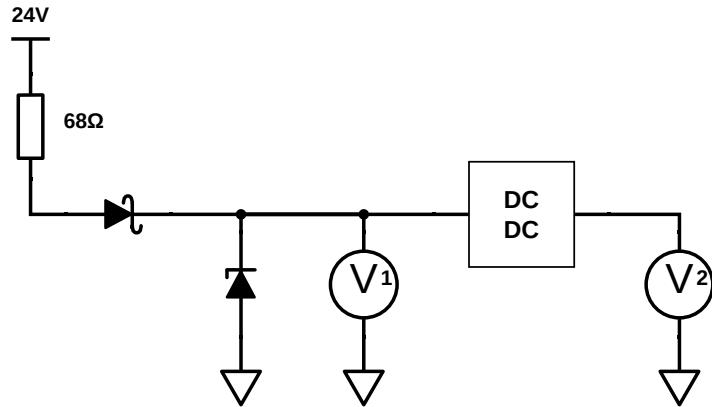


FIGURE 1.21 – Procédure de test de la protection aux surtensions

Pour une tension d'entrée $V_{in} = 23,3V$, l'on a des tensions $V_1 = 17,51V$ et $V_2 = 5,03V$. La diode adaptée à une tension de 14V a une tension de coupure comprise entre 15,6V et 17,2V. La protection fonctionne comme elle le devrait. Celle-ci étant relativement large pour permettre une certaine souplesse quant au convertisseur AD/DC et au ventilateur utilisés.

Chapitre 2

Système d'exploitation

2.1 Support de la caméra

Pour gérer les différentes configurations matérielles de façon plus "userfriendly", la Raspberry-Pi dispose d'un fichier de configuration `config.txt` que le *bootloader* interprétera pour sélectionner les *overlays* correspondants et composer le *devicetree* qui convient à l'architecture matérielle utilisée. Celui-ci étant ensuite passé au *kernel* lors du démarrage.

Cette subtilité propre aux Raspberry-Pi permet à l'utilisateur de ne pas avoir besoin d'avoir affaire au *devicetree* quand il s'agit de configuration. Par exemple l'activation du support d'un élément courant sur les Raspberry-Pi comme le module Raspicam.

le logiciel `raspi-config` n'est autre qu'une interface à ce fichier de configuration.

Pour activer le support matériel de la caméra, il faut ajouter les lignes suivantes à ce fichier :

```
1 start_x=1
2 gpu_mem=128
```

À travers Yocto, le fichier `config.txt` est généré par la recette `meta-raspberrypi/recipes-bsp/bootfiles/rpi-config_git.bb`.

On la surcharge donc, dans la *layer* dédiée au projet, de la recette `meta-autoscope/recipes-bsp/bootfiles/rpi-config_% .bb` :

```
1 VIDEO_CAMERA = "1"
2 GPU_MEM = "128"
```

Quant à l'utilisation de la caméra, il existe des logiciels tels que `raspivid` pour filmer ou `raspistill` pour prendre des clichés. Tous deux font partie de la suite `userland` que l'on ajoute à notre image via la ligne ci-dessous dans le fichier `meta-autoscope/recipes-autoscope/images/autoscope-console-image.bb` :

```
1 IMAGE_INSTALL += "userland"
```

À l'usage, la commande suivante permet d'afficher en plein écran le flux vidéo jusqu'à ce qu'on le stoppe :

```
1 root@autoscope ~ #
2     raspivid -t 0
```


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

```
1 jpegtopnm lune-1822x900.jpg | ppmquant 224 | pnmmoraw > logo_autoscope_clut224.ppm
2 mv logo_autoscope_clut224.ppm
   ↳ ~/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 2.1 – Écrans de démarrage du noyau Linux à droite et du processus d'initialisation à gauche

Les messages provenant du kernel s'affichent par dessus l'écran de démarrage et "polluent" l'écran d'accueil. Il est possible de les masquer en passant l'argument `quiet` au kernel lorsque le bootloader l'invoque. L'argument `console=tty2` permet de rediriger les messages provenant du processus d'initialisation vers la console `/dev/tty2` accessible par la combinaison de touches **[CTRL]**, **[ALT]**, **[F2]**.

Sur Raspberry-Pi, les arguments passés au kernel sont partiellement stockés dans le fichier `cmdline.txt`. Pour le modifier depuis Yocto, on ajoute la ligne suivante à la recette

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

```
1 CMDLINE_append = "quiet console=tty2"
```



FIGURE 2.2 – Écran de démarrage du noyau Linux à droite et écran d'accueil à gauche

2.4 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-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   "
```

```
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=wifi
4 PersistentTetheringMode=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**.

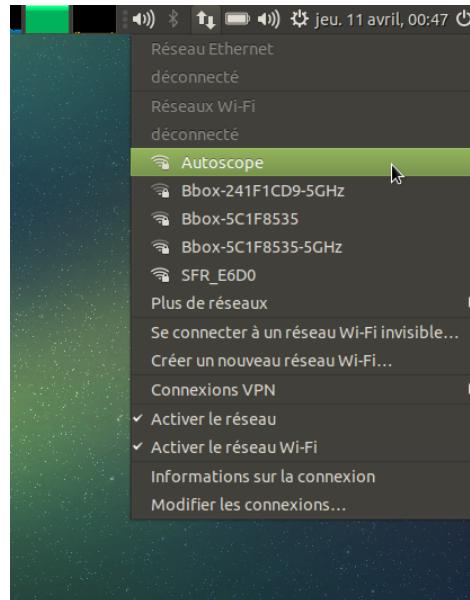


FIGURE 2.3 – Connexion à la Raspberry-Pi depuis un ordinateur distant

2.5 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.

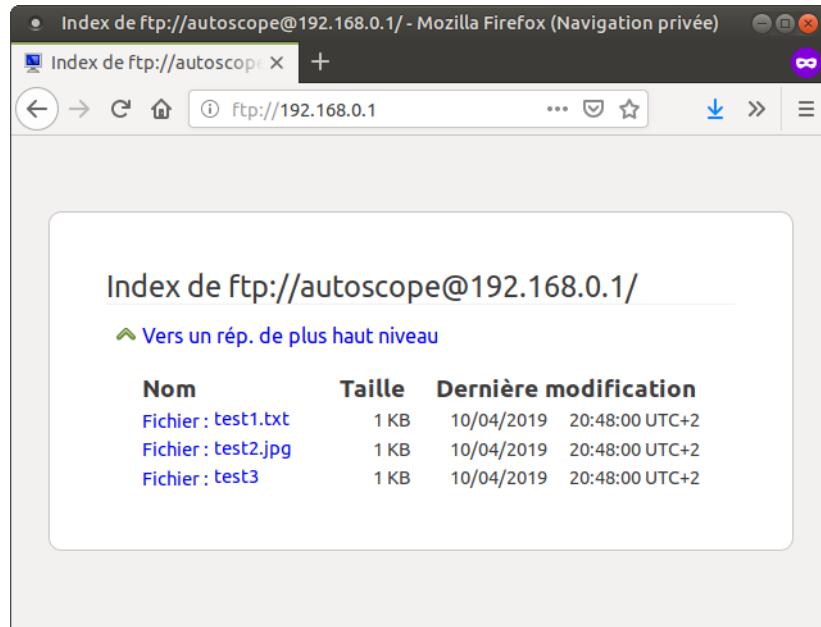


FIGURE 2.4 – Accès au serveur FTP de la Raspberry-Pi depuis un navigateur

- `autoscope` est le seul utilisateur autorisé à accéder au serveur FTP.
- Le serveur se lance automatiquement lors de la phase d'initialisation du système.
- L'option `chroot_local_user=YES` du fichier `vsftpd.conf` empêche l'utilisateur de remonter dans l'arborescence du système au delà du dossier défini par l'option `local_root=/home/$USER`, c'est-à-dire `/home/autoscope`. Il s'agit là d'une mesure élémentaire de sécurité.

2.6 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.

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://${COREBASE}/meta/COPYING.GPLv2;md5=751419260aa954499f7abaabaa882bbe"
5
6 inherit module
7
8 SRC_URI = "git://github.com/thibaudledo/Autoscope;protocol=git;branch=hello_mod"
9
10 SRCREV = "${AUTOREV}"
11 S = "${WORKDIR}/git"
12 RPROVIDES_${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         [ xx.xxxxxx] hello: loading out-of-tree module taints kernel.
4         [ xx.xxxxxx] Hello World!
5 rmmod hello
6         [ xx.xxxxxx] Goodbye Cruel World!
```

2.7 Support de la liaison UART

Pour activer le support de la liaison UART, il faut ajouter la ligne suivante au fichier `config.txt`

```
1 enable_uart=1
```

C'est-à-dire la ligne suivante à la recette

```
meta-autoscope/recipes-bsp/bootfiles/rpi-config_% .bbappend :
```

```
1 ENABLE_UART = "1"
```

Remarque : Ce genre de variables de configurations propre aux Raspberry-Pi est généralement utilisé via les fichiers `local.conf` ou `distro.conf`. La raison en est que certaines, comme `ENABLE_UART` justement, ont un effet dans plusieurs recettes. Celle-ci, par le biais de la recette `linux-raspberrypi_% .bb` ajoute également `console=serial0,115200` au fichier `cmdline.txt`. Ce fichier contient les argument qu'U-Boot passe à Linux lorsqu'il l'appelle. Dans notre cas cette ligne n'est pas souhaitable, on peut donc placer la variable dans une surcharge de la recette `rpi-config_% .bb`

On observe que désormais le fichier `/dev/ttys0` existe, avant cela seul `/dev/ttyAMA0` existait, autre ligne UART utilisée par le module Bluetooth de la Raspberry-pi.

Si l'on boucle la ligne `Tx` sur `Rx` et que l'on observe à l'analyseur logique, voici la trame observée pour une écriture quelconque sur la ligne.

```

1 root@autoscope ~ #
2     echo "5A" | microcom /dev/ttys0 -s 9600
3     5A
```

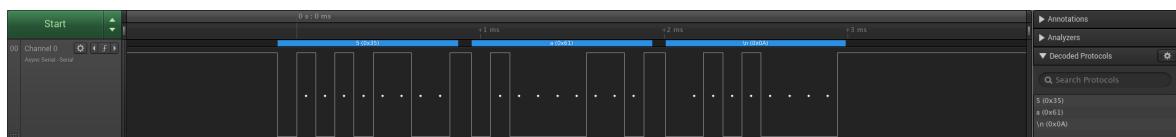


FIGURE 2.5 – Trame UART relevée à l'analyseur logique

On retrouve les trois caractères ASCII envoyés par la commande `echo`. Ceux-ci réapparaissant également à l'écran, démontrant le fonctionnement de la liaison en lecture et en écriture.

2.8 Support du bus I2C

Pour activer le support du bus I2C, il faut ajouter les lignes suivantes au fichier `config.txt`

```
1  dtparam=i2c1=on
2  dtparam=i2c_arm=on
```

C'est-à-dire la ligne suivante à la recette

`meta-autoscope/recipes-bsp/bootfiles/rpi-config_% .bbappend :`

```
1  ENABLE_I2C = "1"
```

On ajoute également la suite de test `i2c-tools`:

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

```
1  IMAGE_INSTALL += " \
2      ${CAMERA} \
3      ${HOTSPOT} \
4      ${FTP} \
5      hello-mod \
6      i2c-tools \
7  "
```

Ensuite, pour pouvoir utiliser `i2c-tools`, il faut activer un driver :

```
1  root@autoscope ~ #
2      modprobe i2c_dev
3      [ xx.xxxxxxx] i2c /dev entries driver
```

Remarque : Il est possible de le charger dès le démarrage en utilisant la variable `KERNEL_MODULE_AUTOLOAD`. Celle-ci est utilisable dans la recette kernel ou dans la recette dudit module.

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

```
1  KERNEL_MODULE_AUTOLOAD_append = "i2c_dev"
```

On peut alors observer les périphériques présents sur le bus :

```
1  root@autoscope ~ #
2      i2cdetect -y 1
3          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
4      00:   -- -----
5      10:   -- -----
6      20:   -- -----
7      30:   -- -----
8      40:   -- -----
9      50:   -- -----
10     60:   -- -----
11     70:   -- -----
```

Aucune adresse n'est attribuée, aucun périphérique n'est présent. On peut tout de même tenter une écriture d'une donnée quelconque à une adresse quelconque et observer la trame émise.

```
1 root@autoscope ~ #  
2     i2cset -y 1 0x5a 0x00  
3     Error : Write failed
```



FIGURE 2.6 – Trame I2C relevée à l’analyseur logique

La trame d’écriture à l’adresse **0x5A** est émise mais n’est acquittée par aucun périphérique, ce qui est normal. La seconde trame, la donnée à écrire **0x00** n’est donc pas émise.

```
1 TetheringTechnologies=true  
2 → TetheringTechnologies=wifi  
3  
4 PersistantTetheringMode=true  
5 → PersistentTetheringMode=true
```