



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

29 mai 2019

Table des matières

Table des matières	1
I Partie de groupe	4
1 Cahier des charges	5
1.1 Introduction	5
1.2 Nécessité du traçage d'astre	5
1.3 Nécessité d'une centrale inertuelle et d'un GPS	6
1.4 Aperçu d'un logiciel de traitement d'images en astronomie	7
1.5 Spécifications techniques	8
1.5.1 Fonctionnalités obligatoires	8
1.5.2 Fonctionnalités envisagées	8
1.5.3 Matériel	9
1.5.4 Résumé des exigences	9
2 Optique	10
2.1 Explications sur le télescope de Dobson	10
2.2 Choix de l'oculaire	11
3 Structure du télescope	15
3.1 Projet d'origine	15
3.2 Étude de la robotisation du télescope	16
3.3 État actuel de la structure	18
4 Architecture	19
4.1 Architecture système	19
4.2 Architecture logicielle	20
5 Organisation	21
5.1 Planning et répartition du travail	21
5.2 Accessibilité du projet sur internet	22
5.2.1 Dépôt principal	22
5.2.2 Dépôt du système d'exploitation de la Raspberry-Pi	23
5.2.3 Dépôt du plugin de Stellarium	23
6 Travail effectué par Thibaud LE DOLEDEC et Clément AILLOUD	24
6.1 Driver de la centrale inertuelle (IMU)	24
6.2 Interface utilisateur (Plugin de Stellarium)	24
7 Travail restant à faire	27
7.1 Structure du télescope	27
7.2 Hardware	27
7.3 Système d'exploitation	27

7.4	Support de la caméra	28
7.5	Transfert du flux vidéo sur le réseau	28
7.6	Daemon du GPS	28
7.7	Driver des contrôleurs moteur	29
7.8	Driver de la centrale inertielle (IMU)	29
7.9	Plugin de Stellarium	29
7.10	Logiciel principal du télescope (Autoscope-core)	29
II	Thomas LEPOIX	30
8	Hardware	31
8.1	Architecture	31
8.2	Conception du circuit	32
8.3	Contraintes de design	33
8.3.1	Contraintes électromagnétiques	33
8.3.2	Contraintes mécaniques	33
8.4	Bon de commande	36
8.5	Fichiers de fabrication	37
8.6	Modélisation 3D	38
8.6.1	Allure générale de la carte	38
8.6.2	Connecteurs des moteurs	39
8.6.3	Emboîtement des cartes	39
8.6.4	Repères des connecteurs	40
8.6.5	Intégration du modèle 3D au télescope	41
8.7	Fabrication et connectique	41
8.8	Validation	43
8.8.1	Tests de continuité et de fuite de courant	43
8.8.2	Test de l'environnement des interrupteurs de butée	43
8.8.3	Test de l'alimentation	44
9	Système d'exploitation	46
9.1	Introduction	46
9.2	Support de la caméra	47
9.3	Étude du transfert du flux vidéo sur le réseau	48
9.4	Splash screen	48
9.5	Hotspot Wifi	51
9.6	Serveur FTP	54
9.7	Driver helloworld	56
9.8	Programme helloworld	57
9.9	Daemonisation du programme helloworld	59
9.10	Support de la liaison UART et communication avec le GPS	61
9.11	Support du bus I2C et communication avec l'IMU	62
10	Daemon d'interface avec le GPS	65
10.1	Fonctionnement du GPS	65
10.2	Fonctionnement du daemon	67

11 Stellarium	69
11.1 Installation depuis les sources	69
11.2 Création d'un paquet binaire	70
11.3 Installation manuelle d'un paquet binaire	71
11.4 Lancement de Stellarium	71
 III Thomas ABGRALL	 72
12 Hardware moteur	73
12.1 Cahier des charges du mouvement	73
12.2 Les moteurs	73
12.3 Le contrôleur	74
12.4 Validation des moteurs et du câblage	74
 13 Driver commande des moteurs	 76
13.1 Introduction	76
13.2 Fonctionnalités du driver	76
13.3 Interactions avec le driver	78
13.4 Intégration du driver	79
13.5 Validation des fonctionnalités de bases	79

Première partie

Partie de groupe

Chapitre 1

Cahier des charges

1.1 Introduction

Le but de ce projet est de réaliser un télescope électronique. C'est-à-dire un télescope doté d'une caméra et dont les mouvements sont pilotables via une interface homme-machine.

Ce projet se base sur un projet existant : Un télescope de type Newton conçu pour être imprimable à l'imprimante 3D. (Le projet semble avoir récemment disparu d'internet).

Lien du projet : <https://blog.dagoma.fr/telescope-imprime-en-3d/>



FIGURE 1.1 – Photo du télescope imprimé

Concernant la politique du projet, nous le souhaitons libre et accessible. C'est pourquoi il sera disponible sur internet sous licence de type copyleft (GPL) et nous travaillerons avec des outils de production libres également, que quiconque peut utiliser.

1.2 Nécessité du traçage d'astre

Le traçage d'astre, vu au départ comme une fonctionnalité intéressante, s'est imposé comme une fonctionnalité nécessaire sur laquelle repose beaucoup de l'intérêt que porte

le projet. En effet il peut être particulièrement difficile pour une personne non initiée à l'astronomie de positionner le télescope vers un astre précis ou de reconnaître un astre que l'on observe. Il nous est donc apparu primordial que le télescope permette d'affranchir l'utilisateur de la nécessité d'avoir des bases en astronomie pour observer le ciel.

En étudiant les solutions disponibles nous avons trouvé des logiciels de simulation du ciel, comme par exemple le logiciel libre Stellarium.

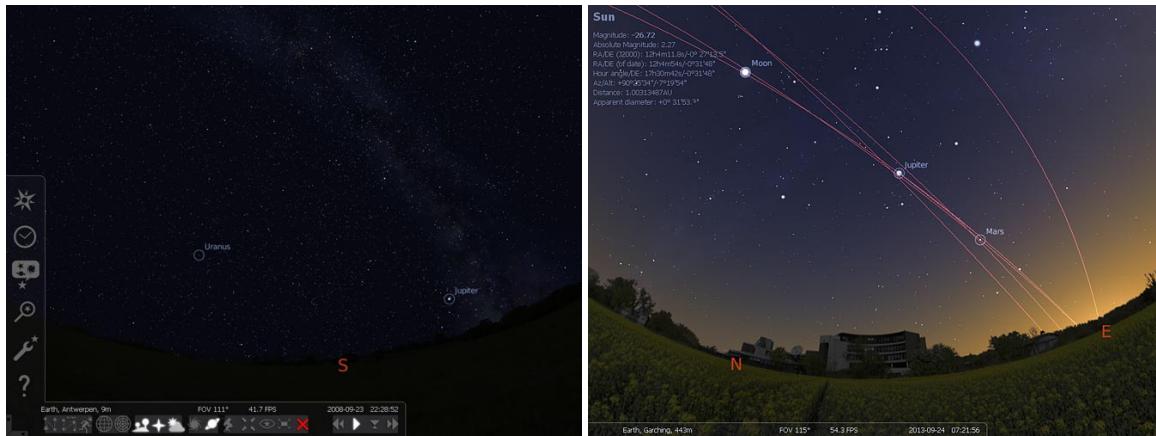


FIGURE 1.2 – Captures d'écran de Stellarium

Celui-ci permet notamment :

- De connaître les coordonnées d'un astre par rapport à l'endroit sur terre où se situe l'observateur.
- De s'orienter dans le ciel selon des coordonnées.
- De s'interfacer avec d'autres systèmes logiciels et/ou matériels

Nous avons donc décidé dans un premier temps de développer une interface pour piloter le télescope depuis un ordinateur distant doté de Stellarium. Puis éventuellement d'embarquer Stellarium dans l'ordinateur du télescope. Ainsi Stellarium fera partie intégrante de son interface utilisateur.

Celle-ci pourrait être alors un menu discret permettant de passer de l'exploration virtuelle du ciel à la vue correspondante à travers le télescope à d'autres élément comme un dispositif d'amélioration de la qualité des images prises.

1.3 Nécessité d'une centrale inertielle et d'un GPS

L'utilisation d'un logiciel de traçage d'astre tel Stellarium nécessite la compatibilité du télescope avec les coordonnées d'azimut et d'élévation couramment utilisées en astronomie. Il est également nécessaire pour cela de savoir de quel endroit sur terre le télescope observe le ciel, d'où l'utilisation d'un GPS.

Pour connaître l'azimut et l'élévation, il faut avoir des repère dans les deux dimensions. Un magnétomètre permet de déterminer la direction du nord et un accéléromètre permet de connaître la direction du sol, c'est à dire la verticale.

Une centrale inertuelle est un composant intégrant un magnétomètre, un accéléromètre et un gyroscope. Elle permet de connaître directement les coordonnées absolues de son orientation dans l'espace.

1.4 Aperçu d'un logiciel de traitement d'images en astronomie

Le traitement d'images en astronomie est un domaine particulier de la retouche photo puisqu'il ne s'agit pas simplement de créer de jolies choses mais de faire ressortir certaines choses d'une image ou d'une série d'images. Il s'agit donc de garder une pertinence physique lors des retouches.

Certains logiciels sont spécialisés dans le traitement d'images du ciel. Parmi les logiciels libres Siril semble être l'un des plus aboutis. Il dispose de nombreux outils et permet de faire de nombreux types de retouches. Un atout intéressant est qu'il permet d'utiliser des scripts pour traiter des images.

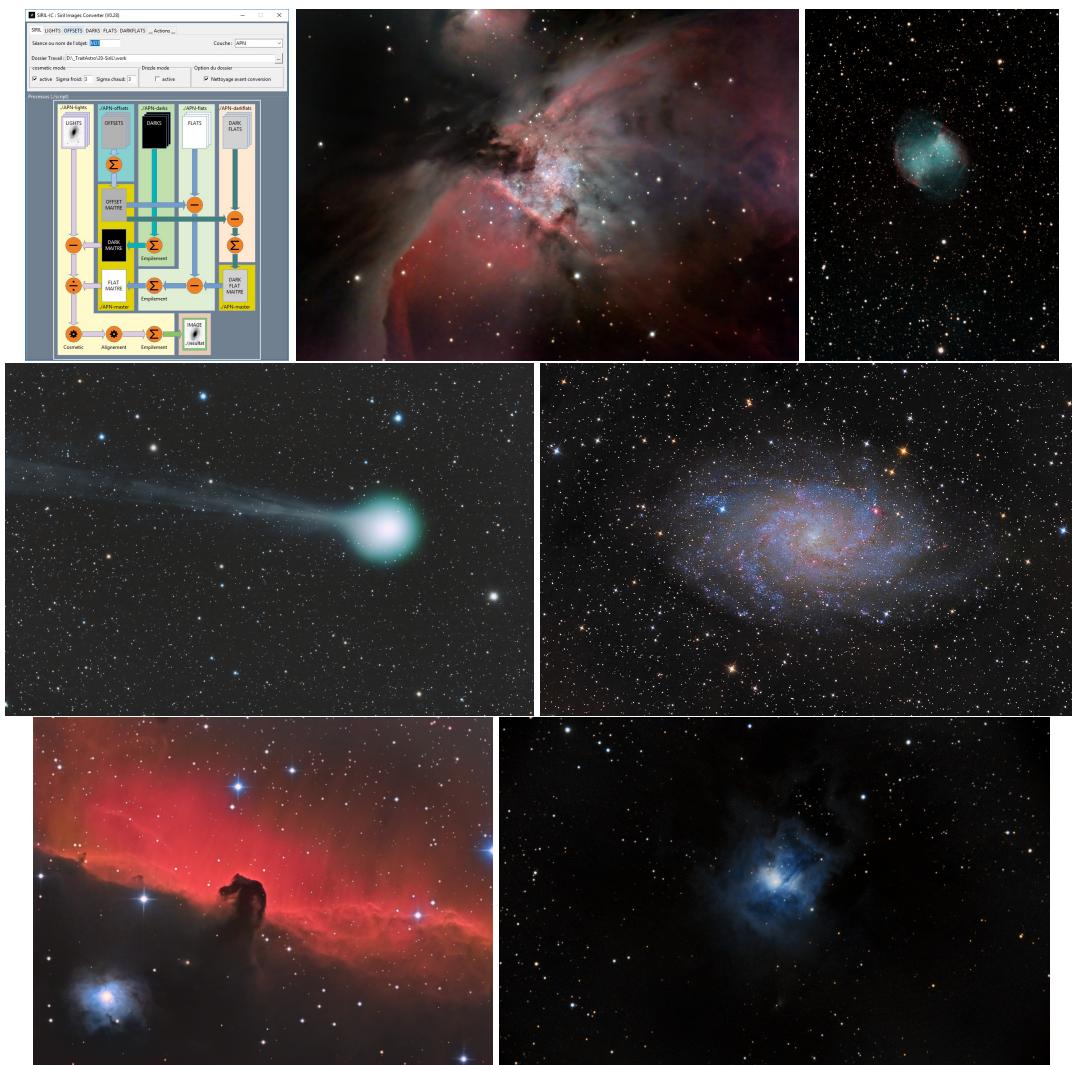


FIGURE 1.3 – Aperçu de Siril et de quelques clichés améliorés par son utilisation

1.5 Spécifications techniques

Certaines fonctionnalités devront impérativement être implémentées pour que le télescope soit validé. D'autres sont envisagées et seront implémentées dans la mesure du possible avant l'évaluation de ce projet. Certaines le seront éventuellement passé cette date, d'autres ne le seront peut être jamais.

De plus nous nous imposons dès le départ l'utilisations de certains matériels.

1.5.1 Fonctionnalités obligatoires

Le télescope devra être capable d'effectuer des mouvements d'azimut à 360°et des mouvements d'élévation dont l'amplitude dépend de la structure du télescope utilisé comme point de départ.

Il disposera d'une caméra permettant de prendre des clichés.

Pouvoir suivre les astres et se positionner dans le ciel est une fonctionnalité primordiale, elle devra être implantée en priorité. De fait Stellarium jouera un rôle central dans l'interface utilisateur du télescope. Une solution où un ordinateur distant équipé de Stellarium pilote le télescope sera d'abord développée.

Idéalement le télescope devrait être accessible à un ordinateur par Wifi.

1.5.2 Fonctionnalités envisagées

Une seconde interface utilisateur où Stellarium serait embarqué dans le télescope qui disposerait d'un écran tactile est envisagée. Ainsi le télescope se suffirait à lui même.

Il existe également une version de Stellarium disponible pour smartphone, il devient donc envisageable de piloter le télescope avec un téléphone. La version Android pourrait d'ailleurs être une piste à explorer pour embarquer ce logiciel.

L'ajout d'une fonctionnalité permettant d'améliorer la qualité des images de façon automatique ou simplifiée serait intéressant. Siril semble être une direction prometteuse à ce sujet.

Le traitement des photos est une fonctionnalité n'ayant de sens que si le télescope fonctionne pleinement, cette fonctionnalité ne peut donc être prioritaire.

L'ajout d'une batterie permettant l'autonomie du télescope est également envisagé mais n'est pas vu comme une priorité.

Une solution de zoom optique pourrait également être intéressante s'il n'est pas trop compliqué ou onéreux d'en mettre une en place. La valeur ajoutée de cette fonctionnalité serait assez grande puisqu'elle permettrait de rendre le télescope polyvalent quant à ce qu'il est capable d'observer.

1.5.3 Matériel

Nous avons choisi d'utiliser comme élément central un SoC (System on Chip) Raspberry-Pi qui, en dépit de ses faibles capacités d'industrialisations, a l'avantage d'être populaire dans le milieu de l'électronique amateur, c'est-à-dire le public le plus susceptible d'être intéressé par ce genre de projet.

Nous avons au départ envisagé l'usage d'une carte PICO-PI-IMX7D de NXP pour nous familiariser à un environnement de travail plus professionnel, que nous avons ensuite délaissé dans l'optique d'embarquer Stellarium dans le télescope. En effet la carte PICO-IMX7D ne dispose ni du GPU de la Raspberry-Pi, ni de suffisamment de RAM.

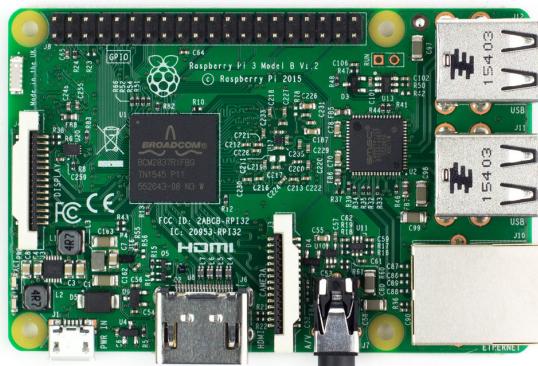


FIGURE 1.4 – Raspberry-Pi 3 B

La caméra utilisée sera celle fournie avec la Raspberry-Pi, à savoir le module Raspicam v2.1 intégrant une caméra IMX219 de 8Mpx. L'écran sera choisi le moment venu s'il est intégré.

1.5.4 Résumé des exigences

Niveau d'exigence	Élément / fonctionnalité	Valeur / référence
Obligatoire	Mouvement d'azimut	360°
Obligatoire	Mouvement d'élévation	
Obligatoire	SoC	Raspberry-Pi 3 B
Obligatoire	Caméra	IMX219
Obligatoire	Interface réseau	Wif , Ethernet ?
Obligatoire	Interface PC	
Obligatoire	Suivi / reconnaissance d'astre	Stellarium
Optionnel	Amélioration des images	
Optionnel	Écran tactile	Siril ?
Optionnel	Interface de pilotage via l'écran	
Optionnel	Autonomie énergétique	
Optionnel	Mouvement de zoom	Secteur

FIGURE 1.5 – Tableau récapitulatif des exigences du projet

Chapitre 2

Optique

2.1 Explications sur le télescope de Dobson

Le télescope est un instrument permettant d'observer le ciel en amplifiant la lumière captée des astres et en agrandissant l'image vue. Il réalise cela par un jeu de miroirs, à la différence de la lunette astronomique qui utilise un jeu de lentilles.

Il existe plusieurs types de télescope, celui de Newton est composé d'un miroir parabolique " primaire" concentrant la lumière vers un miroir plan "secondaire" la déviant vers l'oculaire, où l'utilisateur place son œil.

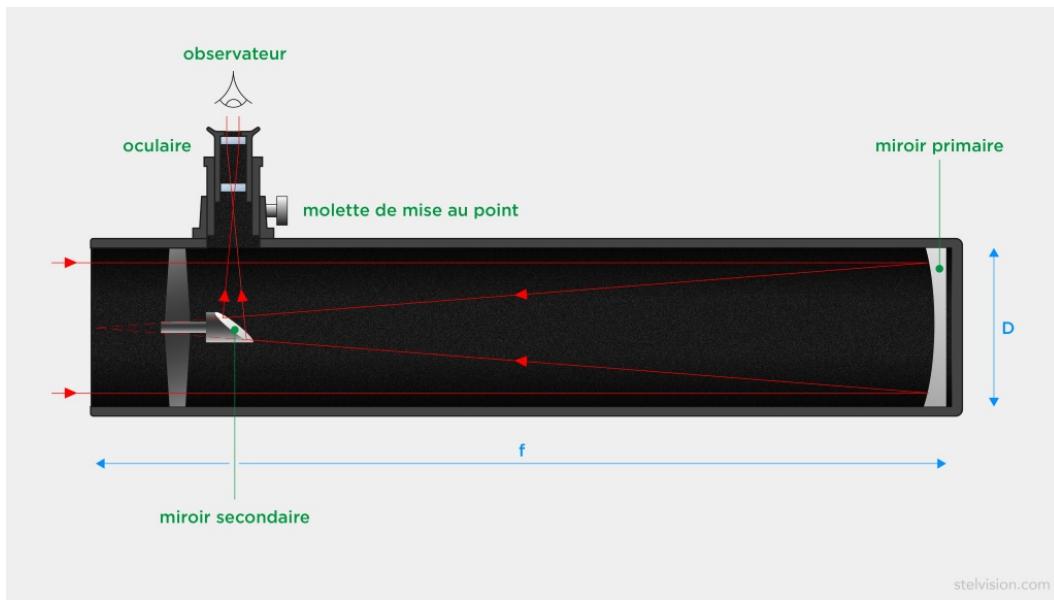


FIGURE 2.1 – Schéma du télescope de Newton
(Schéma ©Stelvision)

L'amplification lumineuse dépend du diamètre D du miroir parabolique. Le grossissement dépend de la distance focale de la parabole f . Le diamètre détermine la luminosité et la finesse de l'image. La distance focale détermine le grossissement. Trop agrandir l'image par augmentation de la distance focale assombrît et dégrade sa qualité.

Le télescope de Dobson est un télescope de Newton doté d'une monture azimutale simplifiée. Il a été inventé par un astronome amateur et a les avantages d'être simple de fabrication et relativement peu onéreux pour un télescope.



FIGURE 2.2 – Télescopes de Dobson

2.2 Choix de l'oculaire

L'oculaire d'un télescope est un jeu de lentilles placé au plan focal de l'image produite par celui-ci. Il sert avant tout à faire de cette image présente au plan focal une image "à l'infini", c'est-à-dire nette sans accommodation de l'œil. Il permet également, par sa propre distance focale, d'ajuster le grossissement du télescope.

$$G = \frac{f_{\text{miroir primaire}}}{f_{\text{oculaire}}}$$

Un zoom est un objectif à lentilles mobiles et à distance focale variable. Il permet donc d'ajuster l'angle d'ouverture de l'appareil et de modifier son grossissement. Il existe des oculaires zoom pour télescope.

Le site internet Stelvision met à disposition un simulateur d'oculaires. Il permet, pour un télescope de caractéristiques f et D données, de visualiser l'image produite avec des oculaires de distances focales différentes. Quelques astres sont observables.

<https://www.stelvision.com/simulateur-telescope/simulateur-telescope.php>

Les caractéristiques de l'Autoscope sont les suivantes :

$$D = 254\text{mm}$$

$$f = 1,27\text{m}$$

Simulation détaillée : choisissez vos oculaires

Diamètre de l'instrument D = mm (10.00")

Focale de l'instrument F = mm ou rapport F/D =

Focales des oculaires

Je choisis mes oculaires

focale oculaire n°1 = mm

focale oculaire n°2 = mm

focale oculaire n°3 = mm

focale oculaire n°4 = mm

focale oculaire n°5 = mm

utiliser une barlow 2X

Proposez-moi une gamme d'oculaires optimisée

Nombre d'oculaires:

Champ apparent des oculaires ° (entre 30 et 110)

Option : autre champ pour comparaison ° (entrez une valeur si vous voulez comparer des oculaires de champs différents)

Catalogue d'oculaires La Clef des Etoiles

Astres à observer

Simuler

Caractéristiques

Diamètre de l'instrument: 254 mm

Oculaires:

- La magnitude atteignable est de **14.0** environ 11.8 millions d'étoiles visibles (contre 6000 à l'œil nu).
- Le pouvoir séparateur est de **0.5** secondes d'arc équivalent à 1 pièce de 1€ vue à 10.2 km, ou à des détails de 0.9km sur la Lune.

n°	focale	grossissement	G/D	Champ app.	Champ réel	pupille de sortie
1	21 mm	60 fois	0.2	50°	0.8°	4.2 mm
2	7 mm	181 fois	0.7	50°	0.3°	1.4 mm

Focale de l'instrument: 1270 mm

FIGURE 2.3 – Simulateur de télescope Stelvision.com



FIGURE 2.4 – Résultats de simulation pour un oculaire de focale $f = 21\text{mm}$
Lune, Saturne, les Pléiades,
galaxies M42, M31, M51

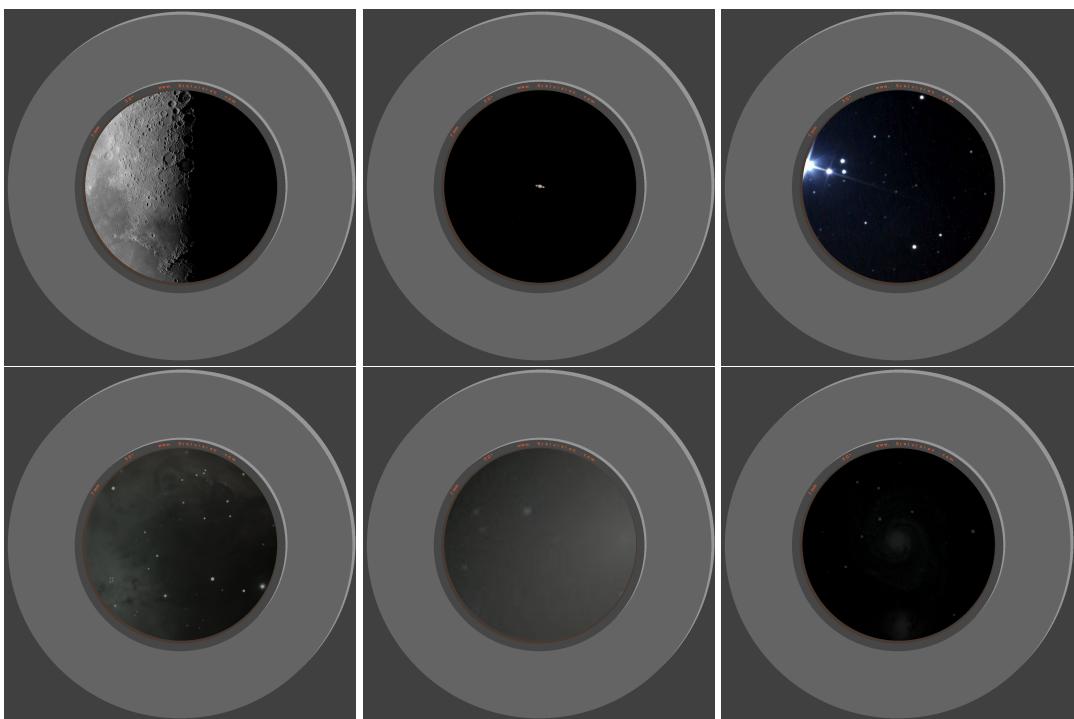


FIGURE 2.5 – Résultats de simulation pour un oculaire de focale $f = 7\text{mm}$
Lune, Saturne, les Pléiades,
galaxies M42, M31, M51

L'oculaire zoom choisi est polyvalent et permet un compromis satisfaisant entre grossissement et luminosité / vue d'ensemble.

<https://www.astroshop.de/fr/oculaires/omegon-7-21mm-super-ploessl-oculaire-zoom-apo-1-25-/p,5084>



FIGURE 2.6 – Oculaire zoom 21mm – 7mm

Chapitre 3

Structure du télescope

3.1 Projet d'origine

La structure du télescope est imprimable à l'imprimante 3D et provient d'un projet open source entre temps disparu d'internet : <https://blog.dagoma.fr/telescope-imprime-en-3d/>

Voici l'allure du télescope monté. Les barres métalliques raccordant le support du miroir secondaire au reste du télescope sont des barres de camping achetées en magasin de sport.

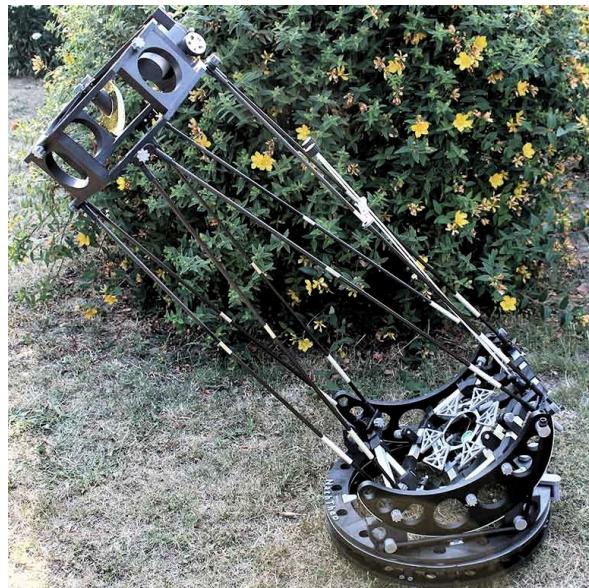


FIGURE 3.1 – Photo du télescope

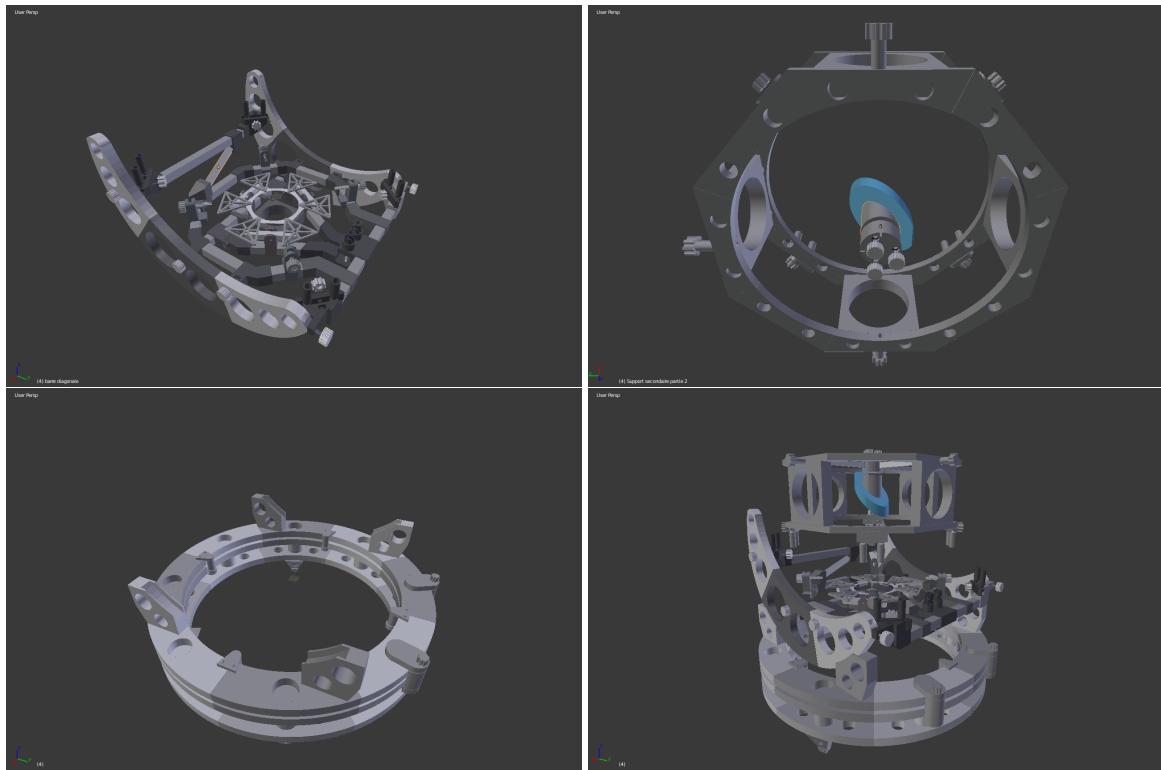


FIGURE 3.2 – Aperçu des fichiers de production du télescope

3.2 Étude de la robotisation du télescope

Ci-dessous quelques schémas représentant les modifications, telles qu'imaginées, à apporter à la structure du télescope pour l'automatiser.

Le socle, immobile, sera doté d'une courroie ainsi que d'un cran permettant d'activer le capteur de position. Le système électronique sera solidaire du support du miroir primaire, mobile, à l'étage supérieur. Un moteur doté d'une roue crantée fixé sur la plateforme tournante permettra de la mettre en mouvement par rapport au socle.

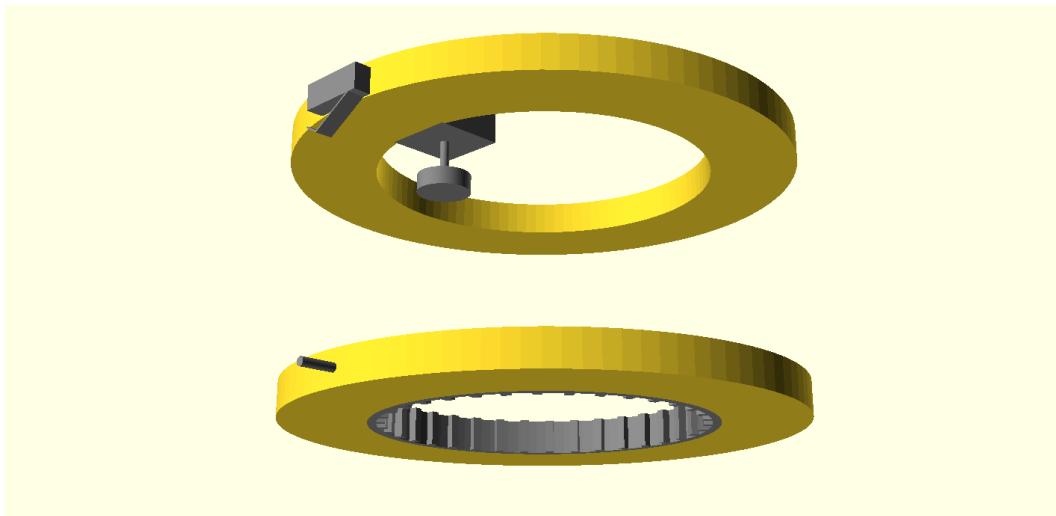


FIGURE 3.3 – Schéma du mécanisme permettant un mouvement azimutal

Pour le mouvement d'élévation, un système de tringlerie et de courroie a été imaginé pour mouvoir le support du miroir primaire par rapport à la plateforme tournante. Deux capteurs de position judicieusement placés permettront de détecter lorsque le mouvement atteint l'un de ses maximums.

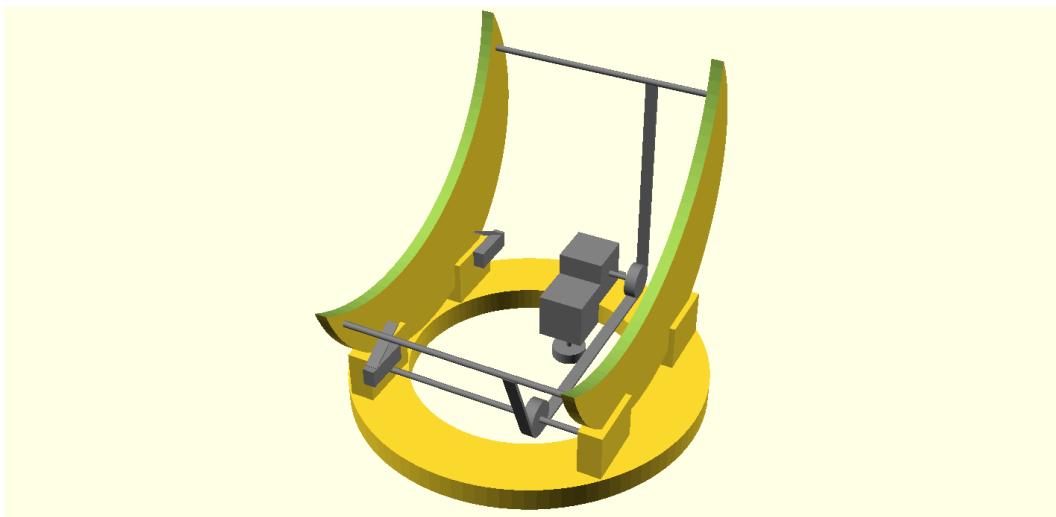


FIGURE 3.4 – Schéma du mécanisme permettant un mouvement d'élévation

Quant au zoom, il faudra créer un ensemble de pièces pour maintenir le zoom lui même, la caméra (non représentée), le moteur permettant d'actionner le zoom, les capteurs de positions extrêmes, ainsi qu'un système permettant de régler finement la position de l'oculaire et de la caméra.

Un cran pour activer les capteurs de positions extrêmes pourrait être fixé à l'oculaire avec un collier de serrage par exemple.

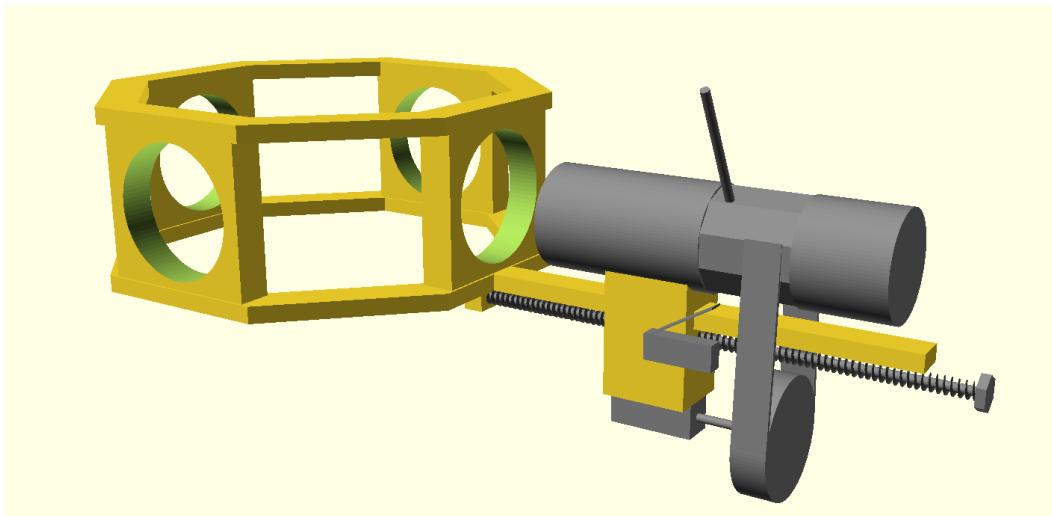


FIGURE 3.5 – Schéma du mécanisme permettant un mouvement du zoom

3.3 État actuel de la structure

Pour l'instant nous avons imprimé toutes les pièces fournies par le projet original. Certaines ont subi une détérioration de la qualité lors de l'impression pour une raison mal connue et demeurent à réimprimer. Le Socle, la plateforme tournante et le support du miroir secondaire ont été montés. Aucune modification de la structure en vue d'y intégrer le système électronique n'a été faite.



FIGURE 3.6 – Photos des parties montées du télescope

Chapitre 4

Architecture

4.1 Architecture système

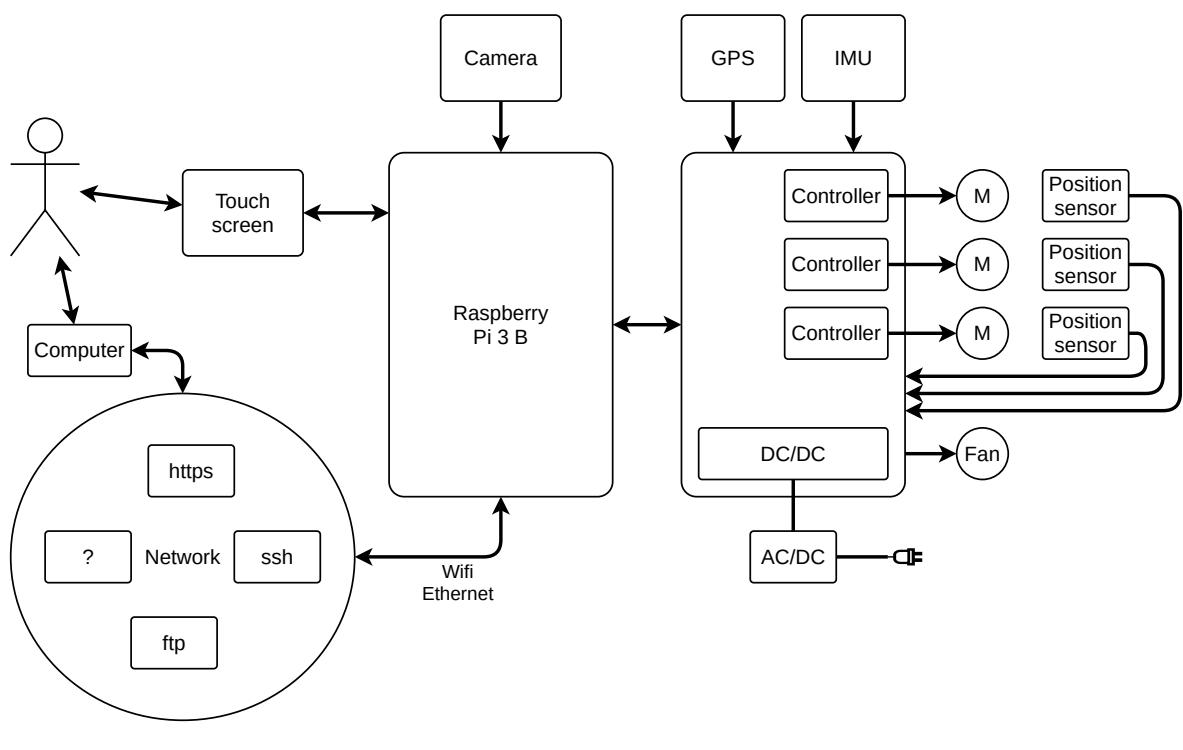


FIGURE 4.1 – Schéma fonctionnel du télescope

Le télescope sera composé d'une carte Raspberry Pi à laquelle sera connecté une carte de même format accueillant le module GPS, le module IMU (centrale inertie), les contrôleurs des moteurs ainsi que l'alimentation du système. Les modules GPS et IMU seront probablement déportés afin de les éloigner des moteurs et de leurs perturbations électromagnétiques.

La caméra et l'éventuel écran tactile seront directement connectés à la Raspberry Pi.

Trois moteurs permettront de mouvoir le télescope :

- Un pour l'azimut.
- Un pour l'élévation.
- Un pour le zoom.

Pour interagir avec le télescope, l'utilisateur aura le choix d'utiliser l'éventuel écran tactile embarqué ou bien un ordinateur connecté au télescope via le réseau. Cette solution étant la voie d'accès préférentielle aux contrôles du télescope.

4.2 Architecture logicielle

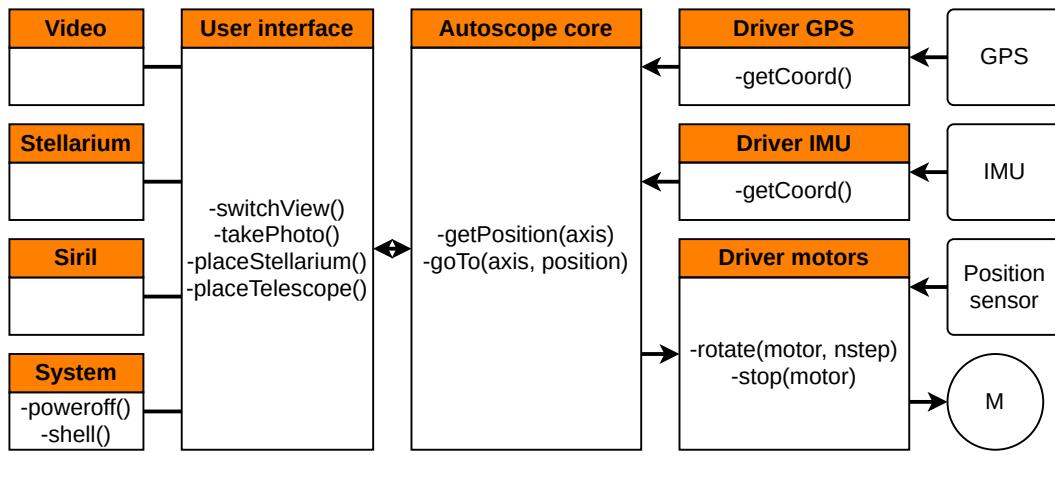


FIGURE 4.2 – Architecture logicielle du télescope

Le fonctionnement du télescope reposera sur plusieurs briques logicielles.

- Au plus bas niveau les drivers permettant de recueillir les données que fournissent l'IMU et le GPS, ainsi que d'actionner les moteurs, ceux-ci étant asservis par des capteurs de positions extrêmes.
- À l'étage suivant le logiciel principal du télescope permettant de connaître les coordonnées précises du télescope (position géographique, orientation spatiale) et d'ordonner au télescope d'adopter une orientation particulière.
- À l'étage supérieur l'interface utilisateur permettant de prendre des clichés du ciel, d'orienter le télescope selon Stellarium, de placer Stellarium selon l'orientation du télescope, mais avant tout de basculer entre les différents modes du télescope :
 - Observation du ciel via la caméra.
 - Exploration virtuelle du ciel avec Stellarium (local ou distant).
 - Amélioration d'images avec Siril.
 - Interaction avec le système d'exploitation du télescope.

Chapitre 5

Organisation

5.1 Planning et répartition du travail

En raison de l'abandon du projet par Thibaud LE DOLEDEC et Clément AILLOUD partis en stage, le projet ne pourra être mené à terme. Pour aborder la dernière ligne droite avant l'évaluation finale, un choix a donc été fait des tâches sur lesquelles travailler en priorité, au détriment d'autres qui demeureront inachevées.

Ci-dessous un diagramme représentant l'avancement des différentes tâches du projet. Le gris indique qu'une tâche est terminée ou à un niveau d'avancement satisfaisant et garantissant une maturité proche. Le orange indique les tâches en cours ou prioritaires au moment de l'évaluation finale. Les flèches représentent des liens de dépendance entre certaines tâches.

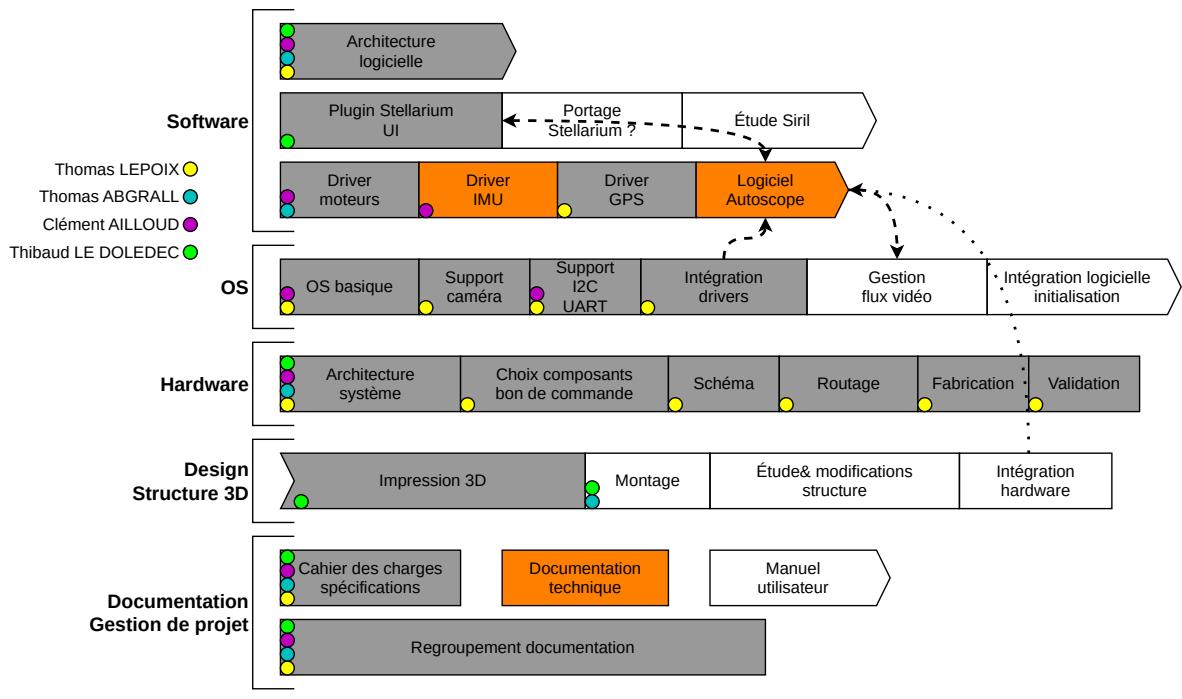


FIGURE 5.1 – Diagramme de l'organisation temporelle du travail sur le projet

5.2 Accessibilité du projet sur internet

Le projet étant libre, il est disponible sur Github sous licence GPL-2. Nous tachons d'accompagner nos différents dépôts d'une documentation claire permettant d'obtenir les informations suivantes :

- Une description brève et précise sur le contenu du dépôt et son rôle au sein du projet.
- Une explication de la procédure à suivre pour utiliser le contenu du dépôt.
- Une explication de la procédure à suivre pour travailler sur le dépôt.

5.2.1 Dépôt principal

<https://github.com/thibaudledo/Autoscope>

Le dépôt est organisé comme suit :

- Branche **master** : Sources du logiciel principal et explications sur le projet dans son ensemble.
- Release **alpha** : Paquets et fichiers binaires pour utiliser le projet "out of the box" (release expérimentale).
- Branche **hardware** : Fichiers Blender de la structure du télescope et fichiers KiCad de la carte électronique du projet.
- Branche **doc** : Documentations et datasheets des composants et éléments utilisés pour le projet.
- Branche **latex** : Fichiers LaTex et .pdf des comptes rendus sur le projet.
- Branche **hello_mod** : Sources d'un driver helloworld servant d'exemple.
- Branche **a4988_mod** : Sources du driver des contrôleurs moteur et des capteurs de fin de course des moteurs.
- Branche **mpu9250_mod** : Sources du driver de la centrale inertuelle.
- Branche **mtk3339d** : Sources du driver du GPS.

5.2.2 Dépôt du système d'exploitation de la Raspberry-Pi

<https://github.com/thomaslepoix/meta-autoscope>

Il s'agit de la couche de métadonnées utilisées par Yocto pour construire le système d'exploitation Linux utilisé par la Raspberry-Pi du télescope. Une image pré-compilée du système d'exploitation figurera sur la release **alpha** du dépôt principal.

Le dépôt est organisé comme suit :

- Branche **rpi** : Métadonnées Yocto et explications de comment compiler et installer le système d'exploitation.
- Branche **rpi-repo** : Données utilisées par Repo pour synchroniser le dépôt à d'autres dépôts de métadonnées Yocto utilisées pour construire l'OS.

5.2.3 Dépôt du plugin de Stellarium

<https://github.com/thibaudledo/Autoscope-Stellarium-plugin>

Il s'agit du plugin de Stellarium contenant l'interface par laquelle l'utilisateur interagira avec le télescope. Un paquet pré-compilé pour linux de Stellarium incluant le plugin figure sur la release **alpha** du dépôt principal.

Le dépôt est organisé comme suit :

- Branche **master** : Sources du plugin, patch des sources de Stellarium et explications de comment compiler une version de Stellarium intégrant le plugin.

Chapitre 6

Travail effectué par Thibaud LE DOLEDEC et Clément AILLOUD

6.1 Driver de la centrale inertie (IMU)

Clément AILLOUD a développé un driver Linux pour la centrale inertie en utilisant le framework `iio`. Celui-ci supporte actuellement le gyroscope et l'accéléromètre mais pas le magnétomètre. Il est compatible avec les bus I2C et SPI. Sur l'Autoscope, le bus I2C sera utilisé.

Un patch du device tree permettant le support du driver est également fourni.

Le driver a été intégré au système d'exploitation du télescope. `iio` permet d'accéder aux données issues de l'IMU via le dossier suivant :

```

1 root@autoscope ~ #
2   ls /sys/bus/iio/devices/iio:device0/
3     dev          in_accel_y_raw    in_anglvel_x_raw  in_temp_input  power
4     in_accel_scale  in_accel_z_raw  in_anglvel_y_raw  name          subsystem
5     in_accel_x_raw  in_anglvel_scale in_anglvel_z_raw  of_node      uevent
6
7   cat /sys/bus/iio/devices/iio:device0/name
8   mpu9250-i2c

```

6.2 Interface utilisateur (Plugin de Stellarium)

Le logiciel Stellarium offre la possibilité de développer des plugins aux fonctionnalités diverses. Nous avons décidé d'utiliser un plugin pour faire de Stellarium l'élément principal de l'interface utilisateur du télescope.

Thibaud LE DOLEDEC a développé un plugin de Stellarium permettant les actions suivantes :

- Rechercher un astre et le suivre tant à l'écran qu'avec le télescope (par l'envoi régulier de coordonnées spatiales à celui-ci).
- Demander au télescope ses coordonnées de position et de visée et afficher la vue du ciel simulé correspondante.
- Ordonner au télescope de prendre une photo. Il existe une interface destinée à configurer les paramètres de la caméra (temps d'exposition, etc.).
- Se connecter au serveur FTP du télescope pour récupérer les clichés du ciel.

- Afficher un aperçu de ce que voit le télescope à l'instant présent (retransmission d'un flux vidéo). Un espace actuellement matérialisé par un rectangle gris est destiné à cela.

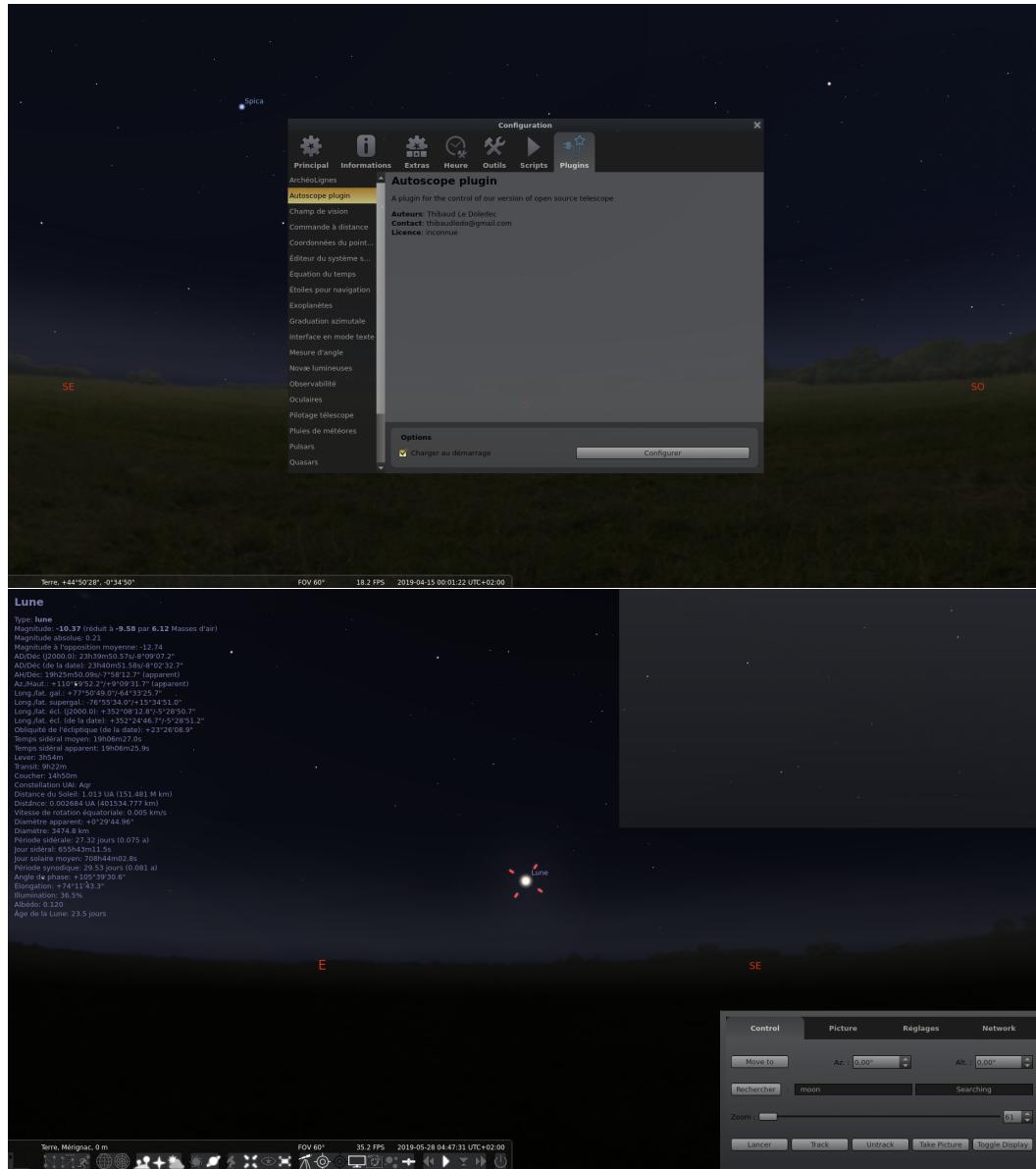


FIGURE 6.1 – Aperçu de l'interface du plugin de Stellarium

Le plugin fonctionne correctement, bien que certains éléments soient sûrement à re-travailler lors du développement du logiciel principal du télescope.

La gestion de la communication FTP repose sur l'utilisation d'une librairie dépréciée `qtftp` qui a semblé poser problème lors des essais de déploiement du travail réalisé par Thibaud. L'idéal serait de se passer de cette librairie.

Thibaud et Clément ont rédigé un document disponible à l'adresse suivante qui explique la procédure à suivre pour développer un plugin de Stellarium.

https://github.com/thibaudledo/Autoscope/blob/latex/Tutoriel_Stellarium.pdf

Chapitre 7

Travail restant à faire

Ce chapitre existe dans le but de récapituler le travail restant à faire sur le projet de façon un minimum exhaustive.

7.1 Structure du télescope

La plupart des pièces du télescope vierge de toute modification a été imprimée, quelques parties assemblées. Le travail restant à faire est conséquent et consiste en la modification du design du télescope en vue d'y intégrer :

- La carte électronique et la Raspberry-pi, face vers le sol, sous le miroir primaire.
- Les modules GPS et IMU, placés le long de la structure du télescope. L'interprétation des données émises par l'IMU dépendra de son positionnement.
- Les moteurs d'azimut et d'élévation ainsi que leurs courroies et tringles. Les roues crantées à fixer sur l'axe des moteurs n'ont pas été choisies. Leur diamètre reste à déterminer.
- Les boutons pousoirs, capteurs de fin de course des mouvements engendrés par les moteur. Leur position devra être déterminée avec précision.
- Tout le dispositif maintenant l'oculaire, le moteur permettant de l'actionner, les capteurs de fin de course associés (le mouvement de l'oculaire s'étend sur environ un quart de tour), ainsi que le module caméra. Ce dispositif devra être conçu avec le plus grand soin et la plus grande attention afin de permettre un réglage fin du télescope et d'obtenir une image nette. Il est conseillé de se renseigner la procédure de *collimation* des télescopes avant de se lancer dans la conception.

7.2 Hardware

Il reste uniquement à faire la connectique des éléments déportés de la carte électronique.

Un ventilateur de refroidissement du miroir peut aussi être choisi, nous ne nous sommes pas renseignés sur l'utilité avérée ou non de cet élément.

7.3 Système d'exploitation

- Sécurisation du système par la mise en place d'un pare feu avec `iptables` (et donc l'étude des ports utilisés, ainsi que le choix d'un port ne posant aucun conflit pour le transfert vidéo et la communication avec le plugin Stellarium), une gestion plus fine des droits de chaque utilisateur Linux, etc. L'abandon de FTP au profit de SFTP comme protocole de transfert d'images serait souhaitable (`openssh` déjà utilisé comme serveur SSH permet cela).

- La mise en place d'un système de mise à jour de l'OS. **swupdate** semble être une solution plus intéressante que **Mender** puisque nettement plus souple et configurable, tant dans la façon de mettre à jour le système que dans la façon de déclencher la mise à jour. Dans tous les cas, ces systèmes de mise à jour nécessitent d'utiliser un bootloader en plus du firmware du GPU de la Raspberry-Pi, or l'overlay du device tree **pi3-disable-bt** (nécessaire pour la communication avec le GPS) semble être incompatible avec U-Boot. Une tentative de transformer l'overlay en patch n'a pas résolu le problème, les investigations n'ont pas été poussées davantage.
- L'optimisation de l'OS. Plusieurs choses peuvent être optimisées, la mémoire occupée, le temps de boot (qui peut être optimisé à plusieurs niveau, les plus faciles à travailler étant le niveau applicatif et le niveau de processus d'initialisation, ensuite viennent Linux et bootloader). Sur Yocto, le fichier **meta-autoscope/conf/distro/autoscope.conf** permet de se délester du support d'éléments inutilisés. Le lien suivant est une documentation intéressante :

<https://embexus.com/2017/05/16/embedded-linux-fast-boot-techniques/>

7.4 Support de la caméra

Il faut évaluer les possibilités de paramétrage de la capture photo que permettent les logiciels **raspistill**, **raspiyuv**, etc. Peut être cela est suffisant, peut être existe il une librairie de plus bas niveau utilisable dans un programme C/C++, peut être sera il nécessaire d'en développer une. Il semble peu probable qu'il faille modifier le driver de la caméra.

7.5 Transfert du flux vidéo sur le réseau

La communauté Raspberry-Pi fournit une librairie Python de contrôle de la caméra, il semble qu'elle permette un transfert sur le réseau du flux vidéo sans latence excessive. La recette Yocto pour intégrer ladite librairie a été écrite mais pas celle permettant l'ajout de toutes celles qu'elle même utilise.

Il serait préférable de trouver une librairie équivalente en C/C++, ou à défaut de l'écrire.

Il n'a pas été déterminé s'il est plus pertinent d'intégrer le logiciel de transmission du flux vidéo au logiciel principal du télescope ou s'il est plus pertinent d'en faire un autre logiciel.

Le serveur vidéo devra probablement s'interrompre le temps que le télescope utilise la caméra pour prendre une photo ponctuelle. Dans le cas où il serait un logiciel distinct du logiciel principal, les signaux seraient sûrement pour ce dernier un moyen de contrôle suffisant. Par exemple **SIGSTOP** et **SIGCONT** s'il suffit de mettre le serveur en pause, **SIGUSR1** et **SIGUSR2** si le serveur doit effectuer des actions de connexion/déconnexion au client ainsi qu'à la caméra.

7.6 Daemon du GPS

Le daemon du GPS n'a pas été testé dans un environnement où la connexion est suffisamment bonne pour que le GPS parvienne à calculer sa position géographique. Il suffit probablement de l'emmener à l'extérieur. Ce test permettrait de valider définitivement le fonctionnement du daemon ainsi que du GPS.

Une meilleure gestion des cas d'erreurs dans le code du daemon serait la bienvenue.

7.7 Driver des contrôleurs moteur

Le cœur du driver des contrôleurs moteur fonctionne. L'organe de communication avec le logiciel principal du télescope est le dernier à être en cours de développement bien que proche d'un état satisfaisant.

Quelques éléments légers resteront à modifier. Une phase de refactoring du code sera également nécessaire.

7.8 Driver de la centrale inertie (IMU)

- Le plus important du travail a été réalisé puisque le driver existe et supporte le gyroscope et l'accéléromètre de la centrale inertie. Il reste à programmer le support du magnétomètre.
- D'avantage de documentation sur le fonctionnement du driver serait également bien-venue.
- Enfin, une série de tests reste à concevoir et à mettre en œuvre pour valider le fonctionnement du driver.

7.9 Plugin de Stellarium

- La librairie FTP utilisée `qftp` pose problème et est dépréciée, il faudrait se passer de son utilisation. Il serait judicieux d'en profiter pour migrer vers SFTP, plus sécurisé.
- Une partie du code sera probablement à retravailler conjointement au développement du logiciel principal du télescope, ainsi que du serveur vidéo.

7.10 Logiciel principal du télescope (Autoscope-core)

- Intégration dans un programme unique des fragments d'interface avec les différents organes logiciels du télescope (drivers moteurs et IMU, daemon GPS, plugin Stellarium, etc). Ces fragments se trouvent sur la branche `master` du dépôt git principal et sont voués à disparaître une fois cette tâche effectuée.
- Développement de l'algorithme de déplacement du télescope. Celui-ci va dépendre des données qu'il est possible d'obtenir de la centrale inertie, des possibilités du driver des contrôleurs moteur, ainsi que de la structure du télescope et la façon dont sera intégrée le hardware.

Deuxième partie

Thomas LEPOIX

Chapitre 8

Hardware

8.1 Architecture

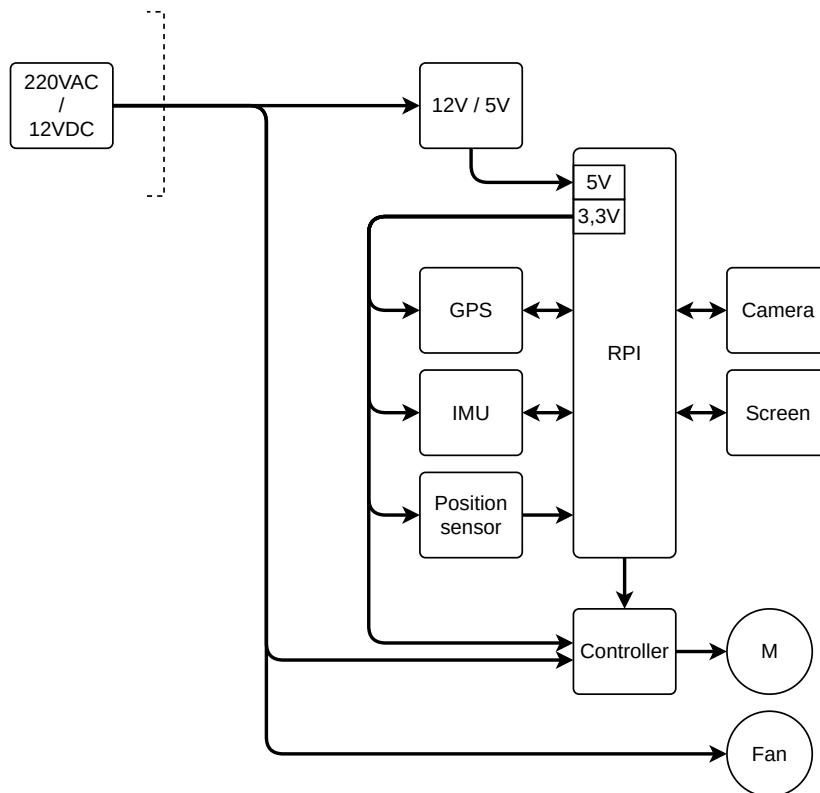


FIGURE 8.1 – Schéma structurel de premier niveau du télescope

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.

8.2 Conception du circuit

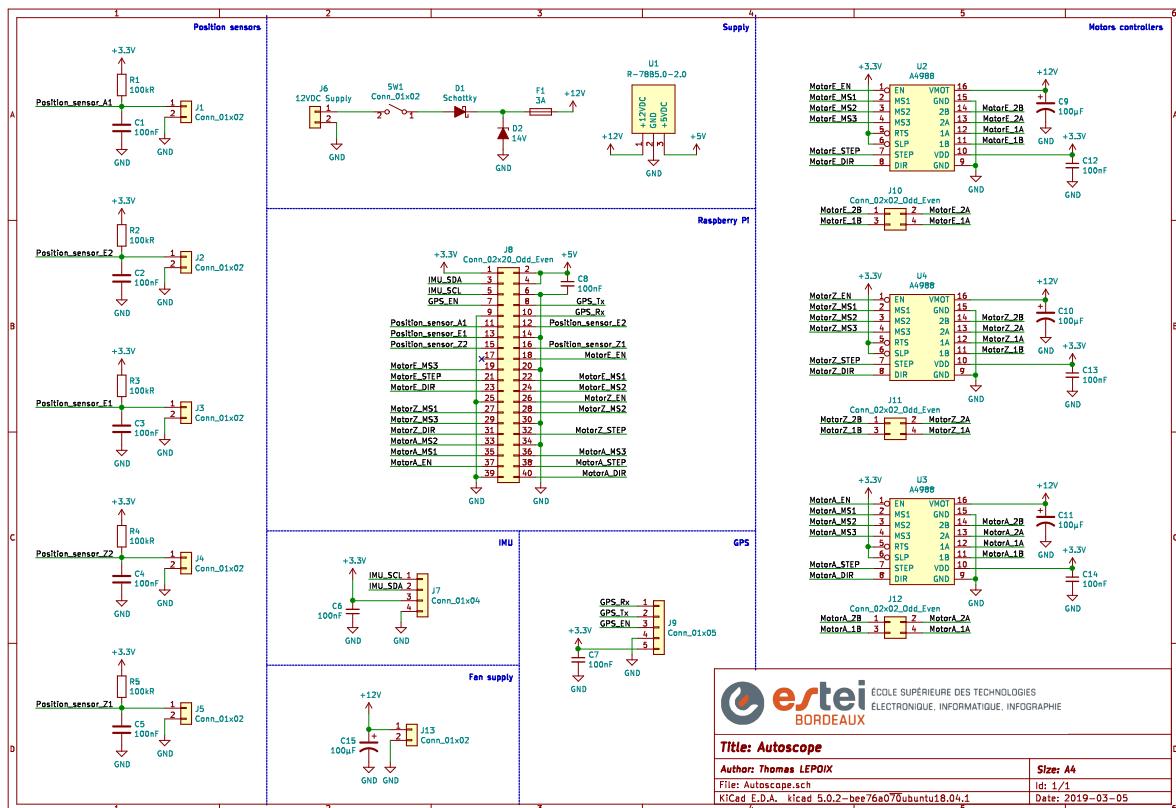


FIGURE 8.2 – Schéma structurel de la carte du télescope

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 :

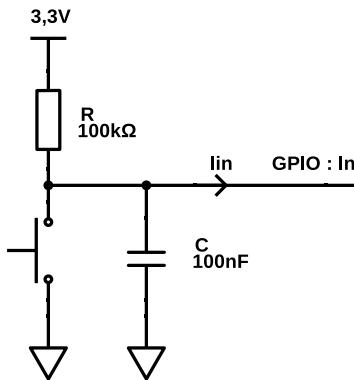


FIGURE 8.3 – Schéma de l'environnement des capteurs de butée

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

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

8.3 Contraintes de design

8.3.1 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.

8.3.2 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.

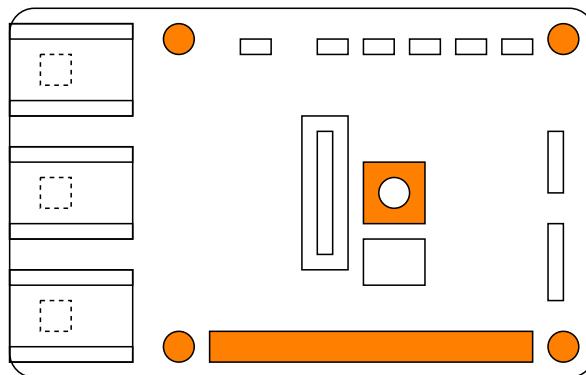


FIGURE 8.4 – Schéma mécanique de la carte vue de dessus

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 8.5 – 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.

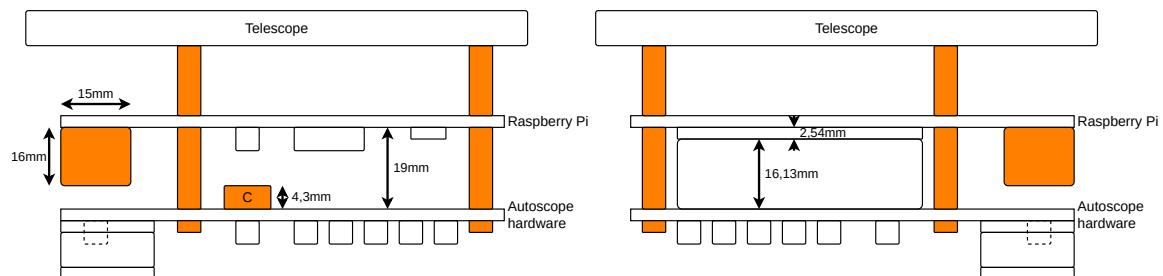


FIGURE 8.6 – Schéma mécanique de la carte vue de profil

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

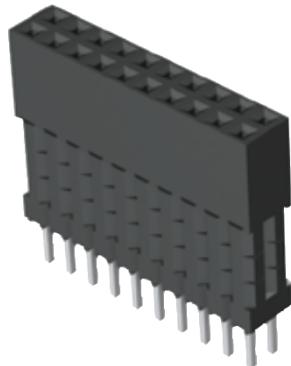


FIGURE 8.7 – Type de header utilisé

8.4 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, caméra et carte mémoire : 61,28€
 - Carte électronique Autoscope : 158,48€

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

	Référence	Valeur	Nom du produit	Nom du fabricant	Référence du fabricant	Nom du fournisseur	Référence du fournisseur	Quantité	Prix unitaire HT (€)	Prix TTC (€)	Total (€)
Optique	-	254mm ; F/D=5	Miroir parabolique			Optique Unterlinden	KE031	1	325	325	
	-	70mm	Miroir plan			Optique Unterlinden	KE035	1	70	70	
	-	7mm;1mm	Occulaire	Omegon		Astro-shop	5084	1	59	59	59
											454
Mécatronique	-	0,9° ; 5,4V ; 0,9A 1,8° ; 3,9V ; 0,6A	Motor pas à pas		17HM15-0904S S20TH30-0604A	Gotronic	31119	2	17,42	20,9	41,8
	-		Motor pas à pas			Gotronic	31107	1	19,08	22,9	32,9
SOC	-		Raspberry Pi 3 B	Raspberry Pi	RASPBERRYPI3-MODB-1GB	Farnell	2525225	1	29,29	29,29	29,29
	-	8Mp	Raspi Camera v2	Raspberry Pi	RPI 8MP CAMERA BOARD	Farnell	2510728	1	22,19	22,19	22,19
	-	16Gb	Carte micro SD	Sandisk	SDSDQAB-016G	Mouser	467-SD5DQAB-016G	1	9,8	9,8	9,8
	-	1,20m ?	Nappe CSI					1			0
											61,28
Carte Electronique	-		Contrôleur moteur	Pololu		Pololu	1182	3		5,23806	15,71418
	-	1,575GHz	Module GPS	Adafruit	Ultimate-GPS-746	Mouser	485-T46	1	34,89	34,89	
	-		Module IMU	Sparkfun	SEN-13762	Mouser	474-SEN-13762	1	13,06	13,06	
	U1	12V ; 5V ; 2A	Module DC/DC	Recom	R-78850-2.0	Mouser	919-R-78850-2.0	1	8,62	8,62	
	-	12V ; 5,41A	Alimentation AC/DC	XP Power	VEC65U512	Farnell	2524410	1	21,75	21,75	
	-	M;4mm;1,7mm	Connecteur alim	CUI inc	PP-013	Mouser	490-PP-13	1	0,987	0,987	0,987
	J6	F;4mm;1,65mm	Connecteur alim	CUI inc	PJ-043H	Mouser	490-PJ-043H	1	0,707	0,707	
	[J10; J12]	2x2 ; F ; 90° ; 2,54mm	Header	Amphenol	76382-302LF	Mouser	649-76382-302LF	3	0,987	0,987	2,961
	-	2x2 ; F ; 90° ; 2,54mm	Header	Amphenol	65239-002LF	Mouser	649-65239-002LF	3	0,241	0,241	1,023
	SW1 ; [J1; J5] ; J13	1x2 ; F ; 90° ; 2,54mm	Header	Amphenol	76382-302LF	Mouser	649-76382-302LF	7	0,498	0,498	3,486
	-	1x2 ; F ; Wire ; 2,54mm	Header	Amphenol	65240-002LF	Mouser	649-65240-002LF	7	0,201	0,201	1,407
	J7	1x4 ; F ; 90° ; 2,54mm	Header	Amphenol	76382-304LF	Mouser	649-76382-304LF	1	0,699	0,699	0,699
	-	1x4 ; F ; Wire ; 2,54mm	Header	Amphenol	65240-004LF	Mouser	649-65240-004LF	1	0,314	0,314	0,314
	J9	1x5 ; F ; 90° ; 2,54mm	Header	Amphenol	76382-305LF	Mouser	649-76382-305LF	1	0,882	0,882	0,882
	-	1x5 ; F ; Wire ; 2,54mm	Header	Amphenol	65240-005LF	Mouser	649-65240-005LF	1	0,402	0,402	0,402
	-	2,54mm	Contacts Header	Amphenol	76357-401LF	Mouser	649-76357-401LF	36	0,192	0,192	6,912
	JB	2x20 ; F ; 2,54mm ; 16,13mm	Header	Samtec	ESQ-120-33-T-D	Mouser	200-ESQ12033TD	1	4,54	4,54	
	[U2 ; U4]	1x8 ; F ; 2,54mm	Header	Amphenol	76341-308LF	Mouser	649-76341-308LF	6	0,996	0,996	5,976
	D2	14V ; DO-214AB	Diode TVS	Littlefuse	SMC1J12A	Mouser	576-SMC1J12A	1	0,54	0,54	
	D1	5A ; DO-214AB	Diode Schottky	MMC	SK54L-TP	Mouser	833-SK54L-TP	1	0,445	0,445	
	[C9 ; C11] ; C15	100µF ; 16V ; C2917	Condensateur	Kemet	T491X107K016AT	Mouser	80-T491X107K016	3	1,96	1,96	5,88
	[C1 ; C8] ; [C12 ; C14]	100nF ; C0808	Condensateur	Yageo	CC0805KRX7R9B104	Mouser	603-C0805KRX7R9B104	11	0,131	0,131	1,441
	[R1 ; R5]	100kΩ ; R0805	Résistance	Bourns	CR0805-FX-1003ELF	Mouser	652-CR0805FX-1003ELF	5	0,087	0,087	0,435
	-	OFF-(ON)	Bouton poussoir	Omron	SS-5GL111	Mouser	653-SS-5GL111	5	1,98	1,98	9,9
	-	OFF-ON	Interrupteur								0
	-	19mm ; M2,5 ; FF	Entretorse	RAF	M1266-2545-SS	Mouser	761-M1266-2545-SS	4	1,62	1,62	6,48
	-	10mm ; M2,5 ; MF	Entretorse	RAF	M2105-2545-SS	Mouser	761-M2105-2545-SS	4	1,37	1,37	5,48
	-	6mm ; M2,5	Vis	Keystone	29301	Mouser	534-29301	4	0,361	0,361	1,444
	-	7mm ; M2,5	Vis								0
	F1	5x20mm ; 6,3A	Porte fusible	Bulgin	FX0321	Mouser	422-FX0321	1	1,62	1,62	
	-	5x20mm ; 3A	Fusible	Bel Fuse	SMF 3-R	Mouser	530-SMF3-R	3	0,175	0,525	
	-	12V	Ventilateur								0
											158,52018
									Total (€)	738,50018	

FIGURE 8.8 – Bon de commande du matériel du télescope

8.5 Fichiers de fabrication

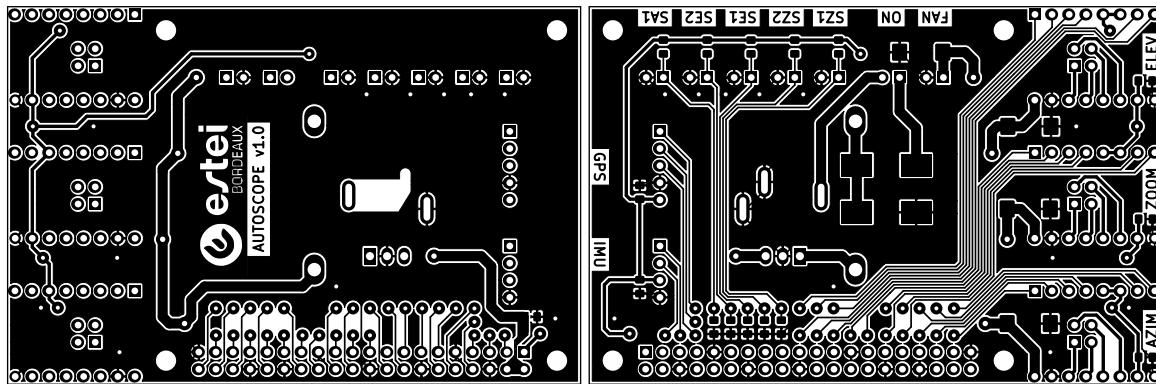


FIGURE 8.9 – Faces supérieure et inférieure du typon de la carte
(respectivement vues de dessus et de dessous)

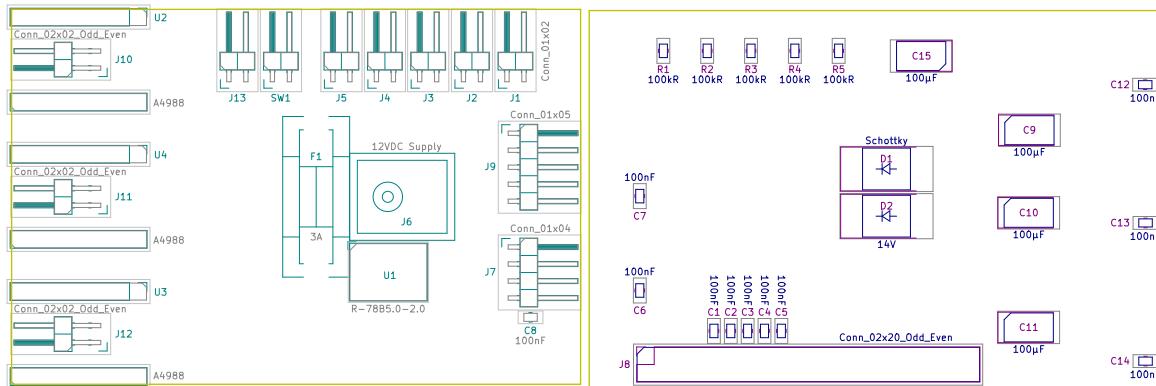


FIGURE 8.10 – Faces supérieure et inférieure de l’implantation des composants
(respectivement vues de dessus et de dessous)

8.6 Modélisation 3D

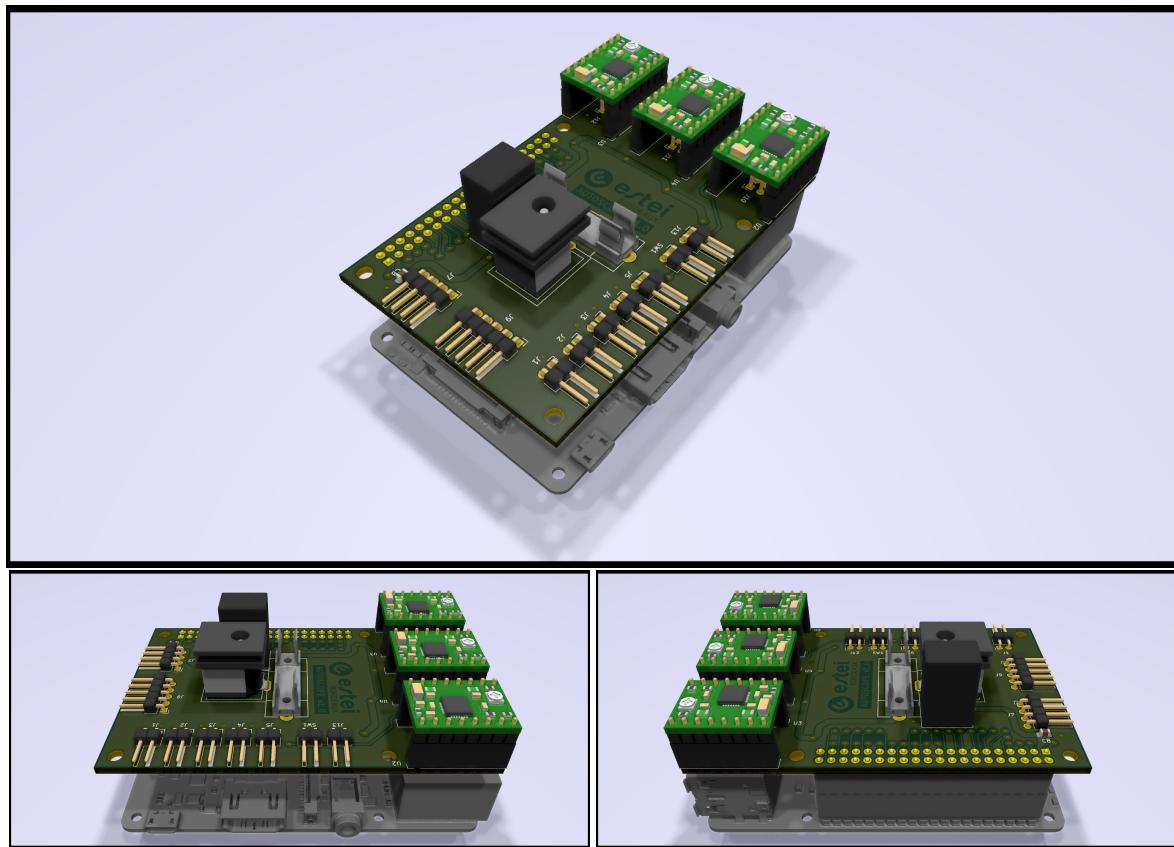


FIGURE 8.11 – Modélisation 3D de la carte pluggée sur la Raspberry-Pi

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.

8.6.1 Allure générale de la carte

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

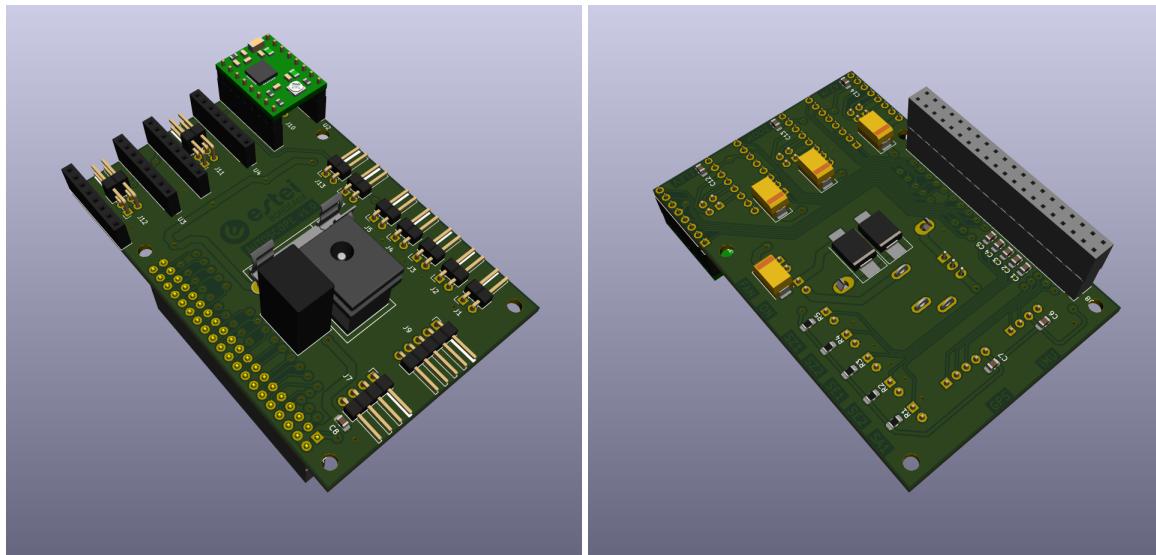


FIGURE 8.12 – Modélisation 3D de la carte avec et sans modules de contrôle moteur

8.6.2 Connecteurs des moteurs

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

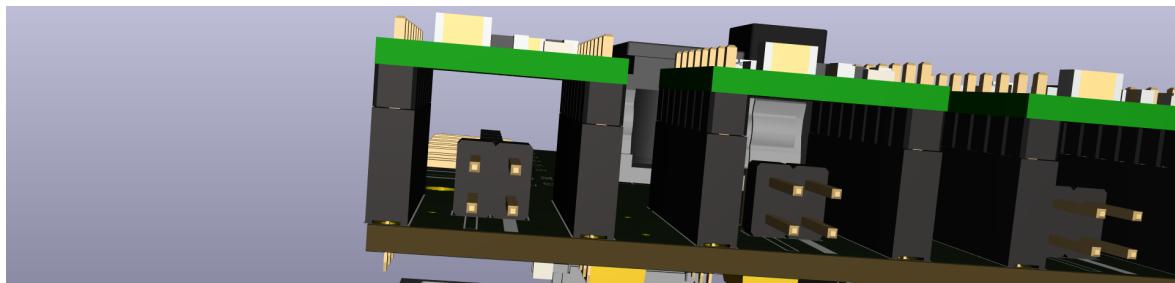


FIGURE 8.13 – Modélisation 3D de la carte : zoom sur les connecteurs moteurs

8.6.3 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 :

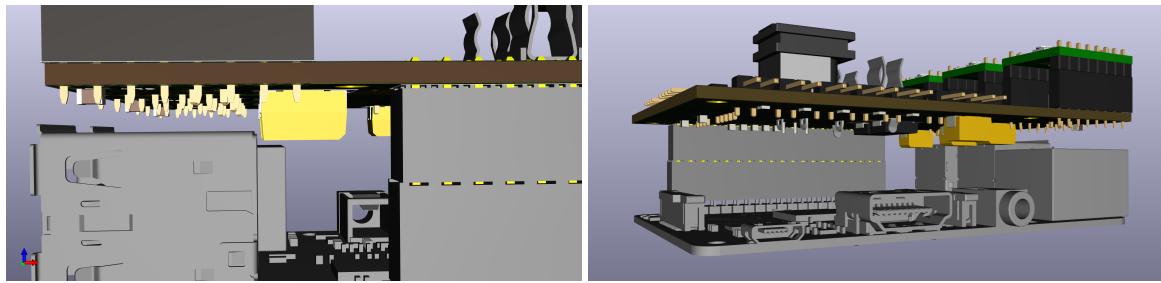


FIGURE 8.14 – Modélisation 3D de la carte : zoom sur l'emboîtement des cartes

8.6.4 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.

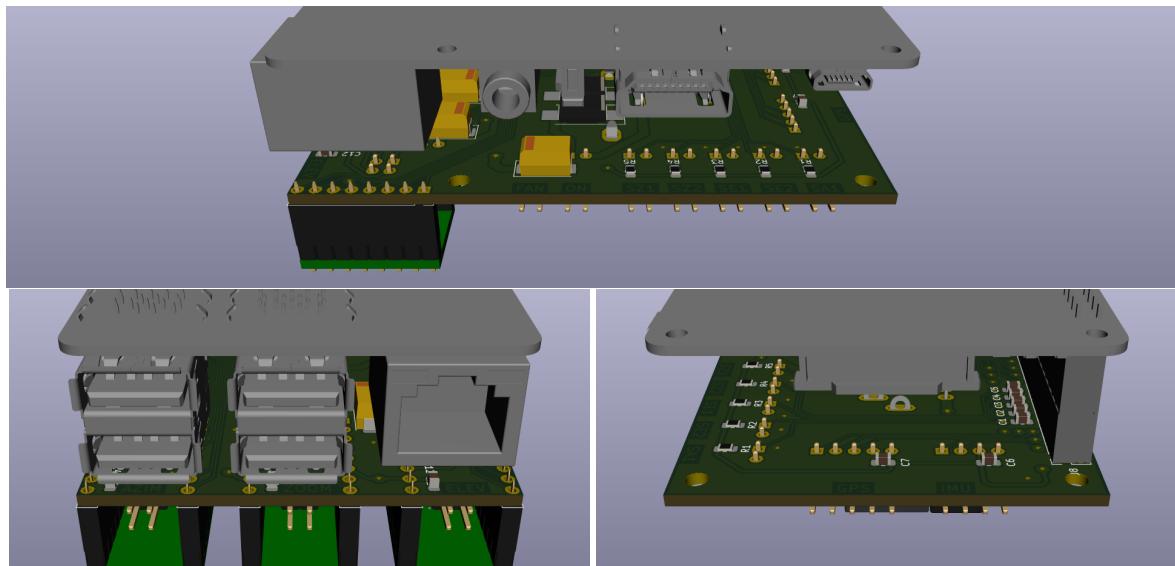


FIGURE 8.15 – Modélisation 3D de la carte : zoom sur les repères des connecteurs

- 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 inertie).

8.6.5 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 :

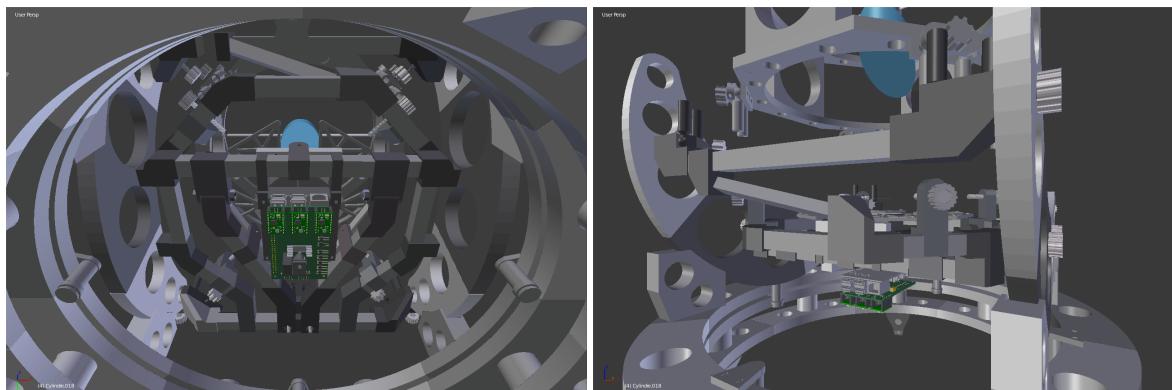


FIGURE 8.16 – Modélisation 3D du télescope grossièrement équipé de ses cartes électroniques

8.7 Fabrication et connectique

La fabrication de la carte ne présente pas de subtilité particulière. Il est toutefois conseillé d'être méthodique quant à l'ordre de soudure des composants pour ne pas être gêné par certains en en soudant d'autres.

Aussi il est conseillé de placer les contrôleurs moteur sur leurs connecteurs support (U2, U3 et U4) pour souder lesdits supports à la carte. Cela pour que les connecteurs supports restent parallèles et qu'emboîter les contrôleurs ne pose pas de problème.

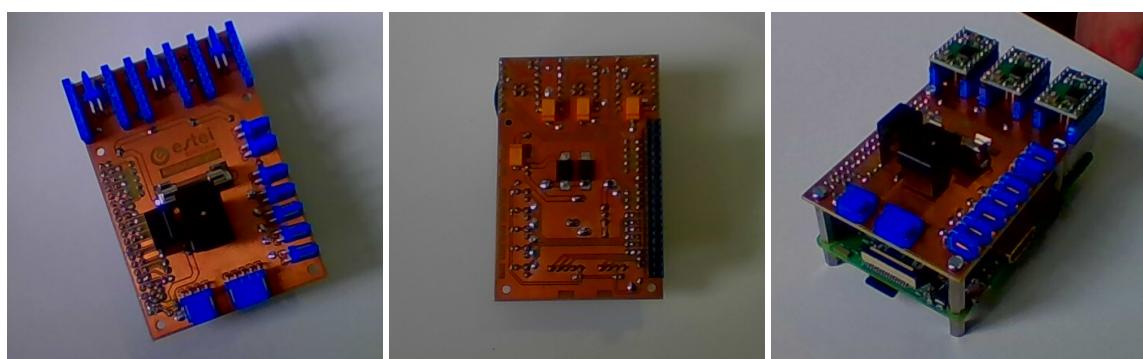


FIGURE 8.17 – Photos de la carte seule et pluggée sur la Raspberry-Pi

Quant au raccordement des différents périphériques, celui-ci devra être fait une fois la structure du télescope montée, ou du moins une fois le modèle 3D de la structure dans sa version définitive.

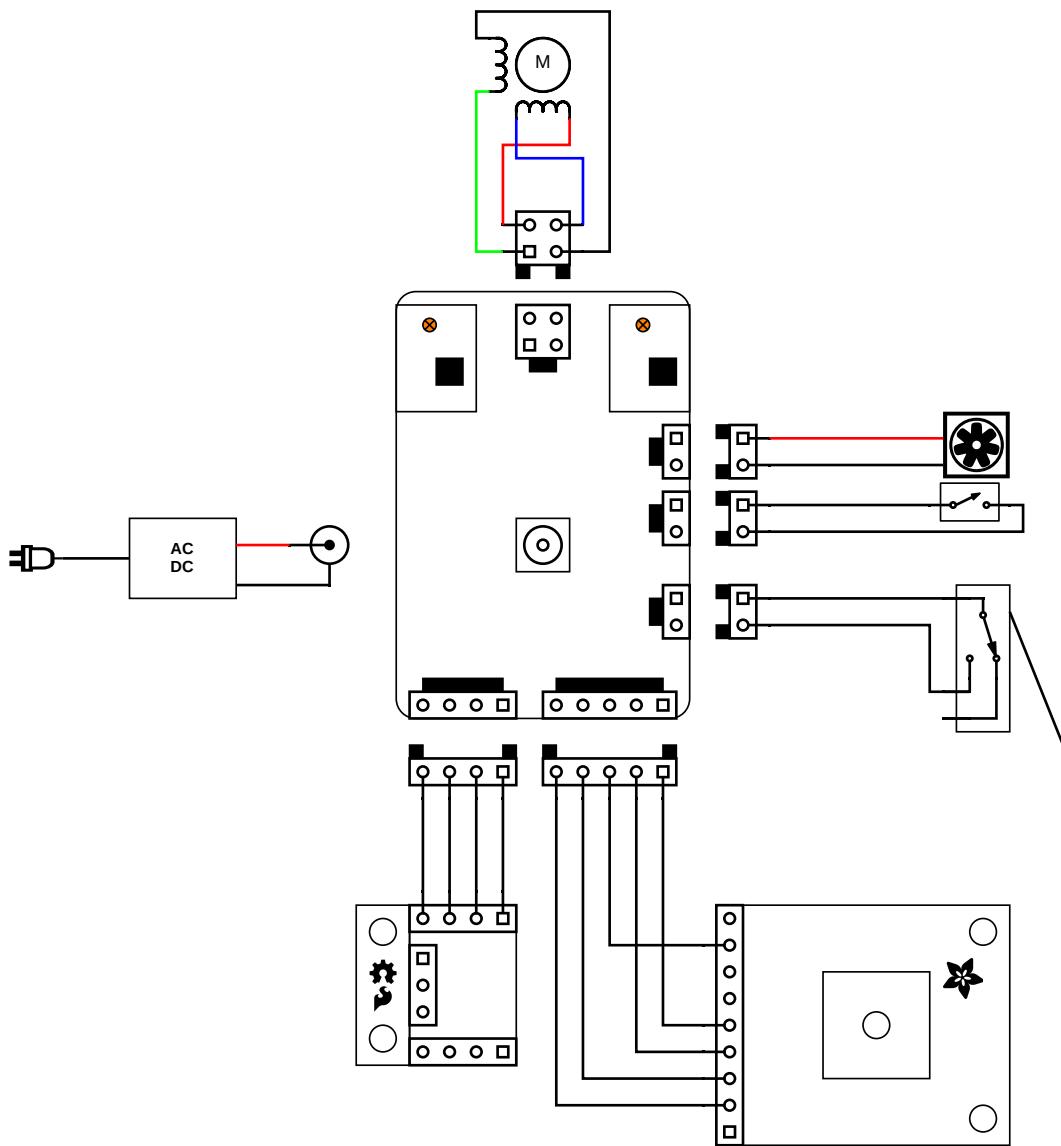


FIGURE 8.18 – Schéma de raccordement des périphériques de la carte

Remarque : Compte tenu de la proximité du connecteur de l'interrupteur **on/off** et du connecteur d'alimentation +12V du ventilateur, il est impératif d'être vigilant à ne pas brancher l'interrupteur à la mauvaise place sous peine de faire fondre le fusible ou pire, d'en-dommager la carte.

Remarque : On remarque un routage maladroit puisque la nappe jusqu'au GPS doit être croisée.

8.8 Validation

8.8.1 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 câblage 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.

8.8.2 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 :

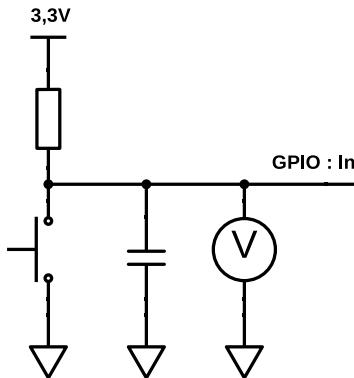


FIGURE 8.19 – Procédure de test de l'environnement des boutons

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 8.20 – Mesure de la tension de sortie des boutons

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

FIGURE 8.21 – 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.

8.8.3 Test de l'alimentation

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

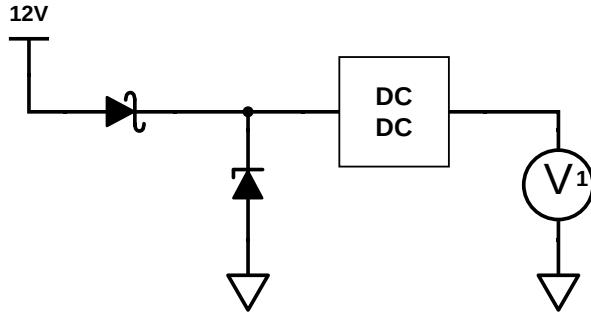


FIGURE 8.22 – Procédure de test du convertisseur DC/DC

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. Cette protection est destinée aux condensateurs de découplage de l'alimentation de puissance dont la tension de claquage est de 16V. Le convertisseur DC/DC et les contrôleurs moteur pouvant tolérer respectivement 32V et 35V et les moteurs étant généralement moins sensibles aux surtensions.

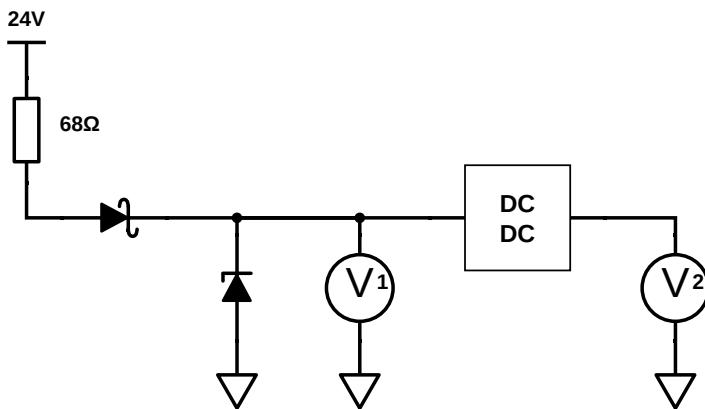


FIGURE 8.23 – Procédure de test de la protection aux surtensions

Pour une tension d'entrée $V_{in} = 23,77V$, l'on a des tensions $V1 = 13,92V$ et $V2 = 5,03V$. La diode adaptée à une tension de 12V a une tension de coupure comprise entre 13,3V et

14,7V pour un courant de 1mA. La protection fonctionne comme elle le devrait, la tension n'excédant pas 16V.

Remarque : En fonction de la puissance que la diode va absorber, celle-ci va chauffer et au fur et à mesure que sa température augmente, sa tension de coupure augmente également.

Concernant le connecteur d'alimentation, la solution semble bonne puisque la carte est capable de pivoter aisément autour du câble d'alimentation sans que celui-ci ne se décroche.

Chapitre 9

Système d'exploitation

9.1 Introduction

L'avantage d'utiliser une carte Raspberry-Pi pour automatiser le télescope est la possibilité de disposer d'un système d'exploitation, en particulier d'une distribution GNU/Linux.

Yocto est un outil qui permet de créer ses propres distributions. Son utilisation permet d'obtenir un système d'exploitation adapté à l'application à laquelle il se destine sans éléments superflus consommant inutilement de la mémoire, de la puissance de calcul ou du temps de réaction. Il facilite de plus la reproductibilité du système.

Yocto est un outils complexe dont il est difficile d'avoir une vue d'ensemble. Ce chapitre n'en expliquera pas le fonctionnement, en revanche pour permettre de comprendre et de se réapproprier le travail effectué, tous les éléments de code permettant de mettre en place des fonctionnalités seront détaillés.

Pour plus d'informations, le fonctionnement global de Yocto est décrit dans un précédent rapport de projet : https://github.com/thomaslepoix/ESTEI/blob/master/M1_Projet_Telecommande_Raspberry-Pi/Hardware_Middleware.pdf

Le travail sur le système d'exploitation de la Raspberry-Pi du télescope s'articule autour des points suivants :

- Mettre en place une configuration réseau permettant un accès simple au télescope depuis un ordinateur, via Wifi notamment.
- Configurer le noyau Linux de sorte que toutes les briques logicielles soient capables de fonctionner. Cela passe notamment par une configuration du device tree.
- Intégrer les différentes briques logicielles et assurer la coordination de leurs fonctionnements en un tout cohérent. Les interactions de ces briques logicielles sont un enjeu majeur.
- L'évaluation pré-développement de certaines solutions, comme par exemple des possibilités qu'offrent les logiciels Raspberry-Pi de gestion vidéo et photo (`raspistill`, `raspivid`, etc.).

Concernant les interfaces du système d'exploitation :

- Il existe une interface FTP destinée uniquement à récupérer les clichés du ciel stockés sur le télescope. L'utilisateur `autoscope` est le seul à pouvoir se connecter au serveur FTP et il ne sert qu'à ça.

- Il existe une interface SSH destinée uniquement à l'administration système du télescope. L'utilisateur `admin` est le seul à pouvoir se connecter au serveur SSH.
- L'utilisateur `root` n'est accessible que localement ou à distance via l'utilisateur `admin`.
- La console locale `tty1` est la console d'accueil du système. Elle est la seule à disposer d'un terminal.
- La console locale `tty2` affiche les messages générés par le système d'exploitation et notamment les messages du système d'initialisation.
- La console locale `tty3` affiche les messages générés par le daemon d'interface avec le GPS `mtk3339d`.
- La console locale `tty4` affiche les messages générés par le daemon `helloworld` s'il est présent.

9.2 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

```

9.3 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 ~ $ nc -l -p 5001 | /usr/bin/mplayer -fps 10 -cache 1024 -
2
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 (10fps), une latence particulièrement longue existe, de l'ordre de 2 à 10 secondes.

Confronté à une problématique similaire sur un projet différent, un autre étudiant a réussi à transférer le flux vidéo à un périphérique quelconque sans latence en utilisant la librairie Python **picamera** et le programme disponible ici :

<http://picamera.readthedocs.io/en/latest/recipes2.html##web-streaming>

Celui-ci semble être similaire mais dénué de l'interface web. Il est une piste intéressante à explorer et probablement la solution la plus simple pour mettre en œuvre le système. L'existence d'une librairie similaire ou d'une solution alternative en C/C++ serait tout de même préférable car elle affranchirait du besoin d'inclure dans le système d'exploitation toutes les librairies Python nécessaires à ce programme.

<https://picamera.readthedocs.io/en/latest/recipes2.html##rapid-capture-and-streaming>

9.4 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 1822×900 . 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-raspberrypi3/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` :

```
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 démarriages :



FIGURE 9.1 – 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 9.2 – Écran de démarrage du noyau Linux à droite et écran d'accueil à gauche

9.5 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   "
```

```
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   connmanctl enable wifi
3   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
7 meta-autoscope/recipes-connectivity/connman-conf/files/main.conf :
8
9 [General]
10 DefaultAutoConnectTechnologies=wifi
11 TetheringTechnologies=wifi
12 PersistentTetheringMode=true
13
14 meta-autoscope/recipes-connectivity/connman-conf/files/settings :
15
16 [global]
17 OfflineMode=false
18
19 [WiFi]
20 Enable=true
21 Tethering=true
22 Tethering.Identifier=Autoscope
23 Tethering.Passphrase=123456789
24
25 [Wired]
26 Enable=true
27 Tethering=false
28
29 [P2P]
30 Enable=false
31 Tethering=false
32
33 meta-autoscope/recipes-connectivity/connman-conf/connman-conf.bbappend :
34
35 FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
36
37 SRC_URI += " \
38   file://main.conf \
39   file://settings \
40 "
41
42 FILES_${PN} += "${sysconfdir}/*"
43
44 do_install_append() {
45   install -d ${D}${sysconfdir}/connman/
46   install -m 0755 ${WORKDIR}/main.conf ${D}${sysconfdir}/connman/main.conf
47   install -d ${D}${localstatedir}/lib/connman/
48   install -m 0755 ${WORKDIR}/settings ${D}${localstatedir}/lib/connman/settings
49 }
```

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

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

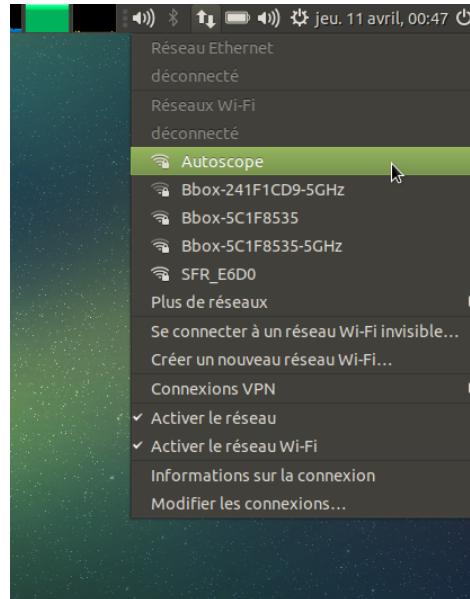


FIGURE 9.3 – Connexion à la Raspberry-Pi depuis un ordinateur distant

9.6 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.

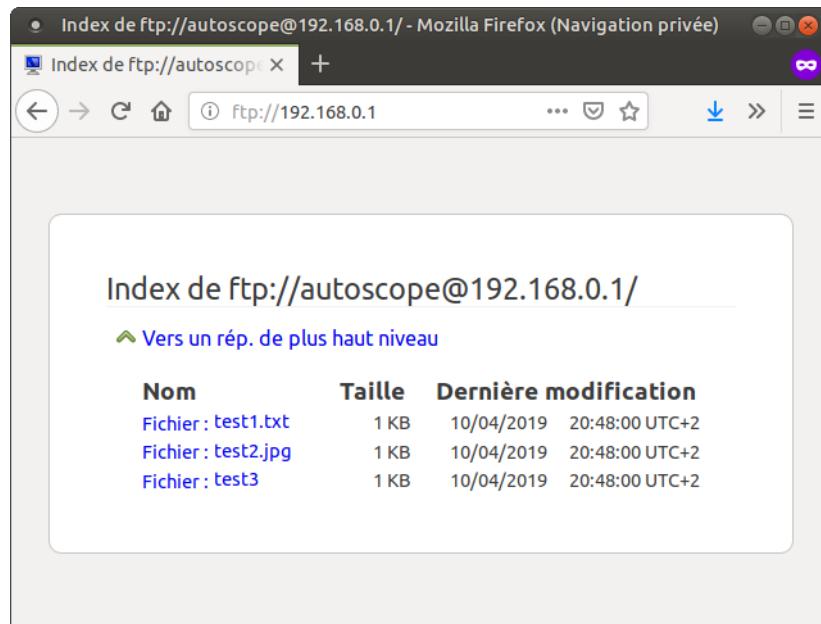


FIGURE 9.4 – Accès au serveur FTP de la Raspberry-Pi depuis un navigateur

- `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.

9.7 Driver helloworld

Ce driver minimaliste a pour but de préparer le terrain pour les drivers des moteurs 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://${COREBASE}/meta/COPYING.GPLv2;md5=751419260aa954499f7abaabaa882bbe"
5 inherit module
6
7 SRC_URI = "git://github.com/thibaudledo/Autoscope;protocol=git;branch=hello_mod"
8
9 SRCREV = "${AUTOREV}"
10 S = "${WORKDIR}/git"
11
12 RPROVIDES_${PN} += "kernel-module-hello"

```

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

```

1 HELLOWORLD = " \
2     hello-mod \
3 "
4
5 IMAGE_INSTALL += " \
6     ${CAMERA} \
7     ${HOTSPOT} \
8     ${FTP} \
9     ${HELLOWORLD} \
10 "

```

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!

```

Pour travailler localement, ce qui est utile en phase de développement, il suffit de cloner le dépôt du driver :

```

1 ~/yocto/sources/meta-autoscope/recipes-test/hello-mod $ 
2     git clone https://github.com/thibaudledo/Autoscope -b hello_mod files/

```

Et de créer une seconde recette où **SRC_URI** pointe vers les fichiers présents dans le dossier **files/**.

meta-autoscope/recipes-test/hello-mod/hello-mod-local.bb :

```

1 SRC_URI = " \
2     file://Makefile \
3     file://COPYING \
4     file://hello.c \
5 "

```

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

```

1 HELLOWORLD = " \
2     hello-mod-local \
3 "

```

9.8 Programme helloworld

Ce programme a pour but de préparer le terrain pour le programme principal du télescope ainsi que pour d'éventuels programmes de test.

meta-autoscope/recipes-test/helloworld/files/helloworld.c :

```

1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main(void) {
5     while(1) {
6         printf("helloworld : TEST MESSAGE\n");

```

```

7         usleep(2000000);
8     }
9     return(0);
10 }
```

En l'absence de **Makefile**, c'est à la recette de comporter les instructions de compilation du programme.

meta-autoscope/recipes-test/helloworld/helloworld.bb :

```

1 SUMMARY = "Helloworld program test"
2 SECTION = "base"
3 LICENSE = "GPLv2"
4 LIC_FILES_CHKSUM =
5   ↵ "file://${COREBASE}/meta/COPYING.GPLv2;md5=751419260aa954499f7abaabaa882bbe"
6
7 SRC_URI = " \
8   file://helloworld.c \
9   "
10
11 do_compile () {
12   ${CC} ${CFLAGS} ${LDFLAGS} ${WORKDIR}/helloworld.c -o ${WORKDIR}/helloworld
13 }
14
15 do_install () {
16   install -d ${D}${bindir}
17   install -m 0755 ${WORKDIR}/helloworld ${D}${bindir}/
```

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

```

1 HELLOWORLD = " \
2   hello-mod \
3   helloworld \
4   "
5
6 IMAGE_INSTALL += " \
7   ${CAMERA} \
8   ${HOTSPOT} \
9   ${FTP} \
10  ${HELLOWORLD} \
11  "
```

À l'issue de la compilation on observe la présence du programme dans le dossier **/usr/bin** du **rootfs** :

```

1 ~/yocto/build/tmp/work/raspberrypi3-poky-linux-gnueabi/autoscope-console-image/1.0-r0/
2   ↵ rootfs $
3     find . | grep helloworld
4       ./usr/bin/helloworld
```

Et à l'usage :

```

1 root@autoscope ~ #
2   helloworld
3     helloworld : TEST MESSAGE
4     helloworld : TEST MESSAGE
5     helloworld : TEST MESSAGE
6   ^C
```

9.9 Daemonisation du programme helloworld

Pour faire d'un programme un daemon, c'est-à-dire un programme se lançant au démarrage de la machine et fonctionnant en tache de fond (comme le serveur FTP par exemple), il faut avoir recours au système d'initialisation de l'OS, ici il s'agit de SysVinit.

On écrit d'abord un script d'initialisation qui sera installé dans le dossier `/etc/init.d/` doté d'un bandeau d'entête où figurent notamment les runlevels dans lesquels le daemon doit être invoqué et ceux dans lesquels il doit être tué. Généralement un daemon peut être invoqué dans les runlevels 2, 3, 4, 5 et peut être tué dans les runlevels 0, 1, 6. Généralement le runlevel 5 est celui de fonctionnement normal de la machine, 0 celui de l'arrêt et 6 celui du redémarrage.

```
meta-autoscope/recipes-test/helloworld/files/helloworld.sh :
```

```

1 #!/bin/sh
2 ##### BEGIN INIT INFO
3 # Provides:                      helloworld
4 # Required-Start:
5 # Required-Stop:
6 # Default-Start:                 5
7 # Default-Stop:                  0 1 6
8 # Short-Description:             Helloworld test
9 ##### END INIT INFO
10
11 NAME="helloworld"
12 DESC="Test program"
13 DAEMON="/usr/bin/${NAME}"
14 PIDFILE="/var/run/${NAME}.pid"
15
16 case "$1" in
17     start)
18         echo -n "Starting ${DESC}: ${NAME}... "
19         # start-stop-daemon -S -b -C -q -x ${DAEMON} > /dev/tty3 #-C don't work with busybox
20         ${DAEMON} > /dev/tty4 &
21         echo $! > ${PIDFILE}
22         echo "done"
23         ;;
24     stop)
25         echo -n "Stopping ${DESC}: ${NAME}... "
26         start-stop-daemon -K -q -x ${DAEMON}
27         rm -f ${PIDFILE}
28         echo "done"
29         ;;
30     restart)
31         $0 stop
32         $0 start
33         ;;
34     status)
35         if [ "`pidof ${DAEMON}`" ]
36         then
37             echo "Program ${NAME} is running (`pidof ${DAEMON}`)."
38         else
39             echo -n "Program ${NAME} is not running"
40             ls ${PIDFILE} &> /dev/null && echo -n " and the pidfile exists" ; echo "."
41             fi
42         ;;
43     *)
44         echo "Usage: $0 {start|stop|restart|status}"
45         exit 1
46         ;;
47     esac
48 exit 0

```

Les messages produits par le daemon sont redirigés vers la console `/dev/tty4` accessible par la combinaison de touches **[CTRL]**, **[ALT]**, **[F4]**.

La recette yocto est légèrement différente.

`meta-autoscope/recipes-test/helloworld/helloworld-daemon.bb` :

```

1 SUMMARY = "Helloworld daemon test"
2 SECTION = "base"
3 LICENSE = "GPLv2"
4 LIC_FILES_CHKSUM =
5     file://$(COREBASE)/meta/COPYING.GPLv2;md5=751419260aa954499f7abaabaa882bbe"
6
7 SRC_URI = " \
8     file://helloworld.sh \
9     file://helloworld.c \
10    "
11
12 inherit update-rc.d
13
14 do_compile () {
15     ${CC} ${CFLAGS} ${LDFLAGS} ${WORKDIR}/helloworld.c -o ${WORKDIR}/helloworld
16 }
17
18 do_install () {
19     install -d ${D}${sysconfdir}/init.d
20     cat ${WORKDIR}/helloworld.sh | \
21         sed -e 's,/etc/${sysconfdir},g' \
22             -e 's,/usr/sbin/${sbin},g' \
23             -e 's,/var/${localstatedir},g' \
24             -e 's,/usr/bin/${bindir},g' \
25             -e 's,/usr/${prefix},g' > ${D}${sysconfdir}/init.d/helloworld
26     chmod a+x ${D}${sysconfdir}/init.d/helloworld
27
28     install -d ${D}${bindir}
29     install -m 0755 ${WORKDIR}/helloworld ${D}${bindir}/
30 }
31
32 INITSCRIPT_NAME = "helloworld"
33 INITSCRIPT_PARAMS = "start 99 5 . stop 00 0 1 6 ."
```

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

```

1 HELLOWORLD = " \
2     hello-mod \
3     helloworld-daemon \
4   "
5
6 IMAGE_INSTALL += " \
7     ${CAMERA} \
8     ${HOTSPOT} \
9     ${FTP} \
10    ${HELLOWORLD} \
11  "
```

On observe ensuite dans le `rootfs` la présence du script d'initialisation ainsi que la présence d'un lien symbolique vers celui-ci dans les dossiers des runlevels 0, 1, 5, 6 avec des préfixes :

- `s` ou `k` indiquant si le daemon est invoqué (Start) ou tué (Kill).
- `00` à `99` indiquant le numéro de priorité de l'appel, `00` étant le plus prioritaire.

```

1 ~/yocto/build/tmp/work/raspberrypi3-poky-linux-gnueabi/autoscope-console-image/1.0-r0/
2     rootfs $
3     find . | grep helloworld
4         ./usr/bin/helloworld
5         ./etc/init.d/helloworld
6         ./etc/rc0.d/K00helloworld
7         ./etc/rc1.d/K00helloworld
```

```
7     ./etc/rc5.d/S99helloworld
8     ./etc/rc6.d/K00helloworld
```

Et à l'usage :

```
1 root@autoscope ~ #
2   /etc/init.d/helloworld status
3     Program helloworld is running (347).
4
5   # [CTRL] [ALT] [F3]
6     helloworld : TEST MESSAGE
7     helloworld : TEST MESSAGE
8     helloworld : TEST MESSAGE
9
10  # [CTRL] [ALT] [F1]
11  /etc/init.d/helloworld stop
12    Stopping Test program: helloworld... done
13  /etc/init.d/helloworld status
14    Program helloworld is not running.
```

9.10 Support de la liaison UART et communication avec le GPS

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 arguments 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`

De plus sur la Raspberry-Pi 3, il existe une UART classique attribuée par défaut au module Bluetooth de la carte et une "miniUART" qui peut poser problème à l'utilisation puisqu'elle n'a pas d'horloge interne et dépend de l'horloge du processeur qui varie selon sa charge. Quelques essais ont confirmé que cette UART posait problème.

Il existe une overlay du device tree permettant d'intervertir les deux UARTs `pi3-disable-bt-overlay.dts` et de relier l'UART classique aux pins `8` et `10` du connecteur. L'on active l'overlay ainsi :

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

```
1 RPI_EXTRA_CONFIG = "\n\
2 dtparam=act_led_trigger=off\n\
3 disable_camera_led=1\n\"
```

```

4 dtoverlay=pi3-disable-bt\n\
5 "

```

Remarque : Cette overlay ne semble pas fonctionner avec U-boot comme bootloader. Aucune solution n'a encore été trouvée pour les faire fonctionner ensemble.

Si l'on connecte le GPS à la carte, on peut lui envoyer une trame l'interrogeant sur les informations de son firmware **605 PMTK_Q_RELEASE**.

```

1 root@autoscope ~ #
2   echo -ne "$PMTK605*31\r\n" | microcom /dev/ttyAMA0 -s 9600
3     $PMTK705,AXN_2.31_3339_13101700,5632,PA6H,1.0*6B
4     $GPGGA,000700.800,,,,,0,00,,,M,,M,,*77
5     $GPGSA,A,1,,,,,,,*1E
6     $GPRMC,000700.800,V,,,,0.00,0.00,060180,,,N*4D
7     $GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
8     $GPGGA,000701.800,,,,,0,00,,,M,,M,,*76
9     $GPGSA,A,1,,,,,,,*1E
10    $GPGSV,1,1,00*79
11    $GPRMC,000701.800,V,,,,0.00,0.00,060180,,,N*4C
12    $GPVTG,0.00,T,,M,0.00,N,0.00,K,N*32
13 ^C

```

L'on observe la trame de réponse **705 PMTK_DT_RELEASE** indiquant le nom et la version du firmware **AXN_2.31_3339_13101700** accompagnés d'autres informations constructeurs. L'on observe ensuite une suite de quelques trames NMEA envoyée périodiquement toutes les secondes.

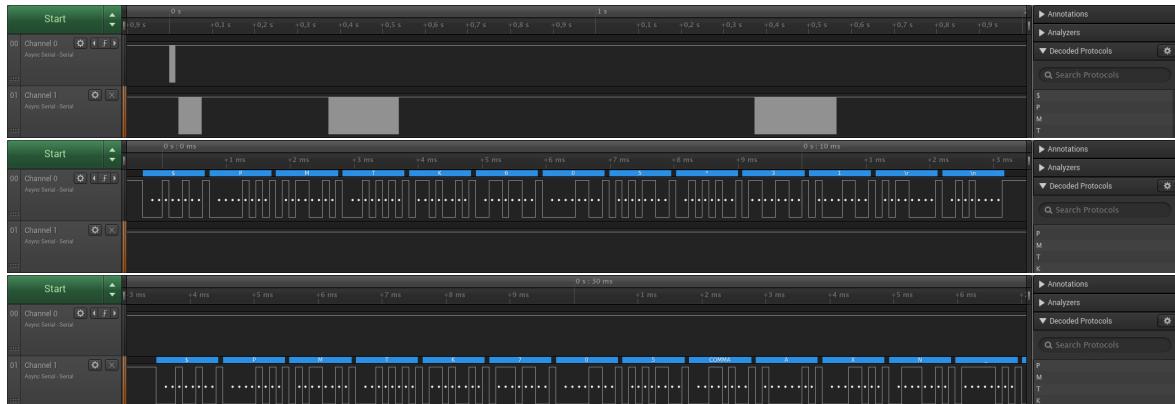


FIGURE 9.5 – Trame UART relevée à l'analyseur logique

9.11 Support du bus I2C et communication avec l'IMU

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   ${HELLOWORLD} \
6   i2c-tools \
7 "
```

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

```
1 root@autoscope ~ #
2   modprobe i2c_dev
3   [ xx.xxxxxx] 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 connecte la centrale inertuelle au bus I2C et l'on peut 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:          -- -- -- -- -- 68 -- -- -- -- --
11  70:          -- -- -- -- -- -- -- -- -- -- --
```

Une seule adresse est attribuée, il s'agit de la centrale inertuelle. Il est dit dans sa documentation que le registre `WHO_AM_I` à l'adresse `0x75` contient invariablement la valeur `0x71`. On peut alors essayer de lire ce registre.

```
1 root@autoscope ~ #
2   i2cget -y 1 0x68 0x75
3           0x71
```

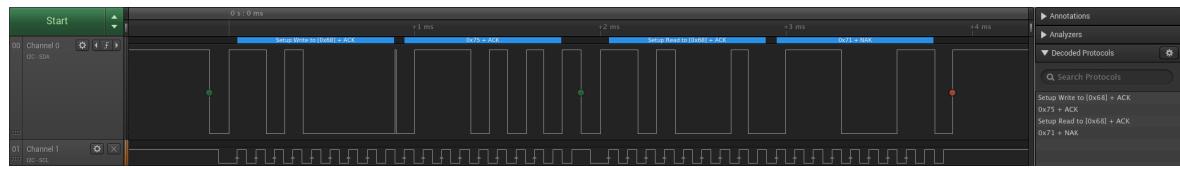


FIGURE 9.6 – Trame I2C relevée à l'analyseur logique

La communication entre la centrale inertuelle et la Raspberry-Pi fonctionne.

Chapitre 10

Daemon d'interface avec le GPS

Sur Linux, il ne convient généralement pas de gérer les périphériques utilisant l'UART avec des drivers. En effet à la différence des bus de communications comme I2C ou SPI, l'UART dispose déjà d'un driver permettant d'envoyer et de recevoir des trames depuis l'espace utilisateur. Les drivers résidant eux dans l'espace du système d'exploitation.

De plus l'UART est d'un niveau d'abstraction supérieur par rapport à ces bus puisqu'il ne s'agit pas d'envoyer des commandes de lecture ou d'écriture sur les registres du périphérique mais d'envoyer des trames de commande que le périphérique va interpréter.

On utilise donc généralement des daemons pour faire l'interface avec ce type de périphériques. Les daemons étant des programmes tournant en tâche de fond de l'espace utilisateur du système.

10.1 Fonctionnement du GPS

Le GPS utilise des trames NMEA, qui est une norme de communication américaine pour les équipements marins. Elle est largement utilisée par les GPS.

Il existe de nombreuses trames NMEA, toutes sont préfixées de `$` et suffixées de `\r\n`. Les deux premiers caractères après `$` désignent le type d'équipement qui émet la trame, une trame émise par un GPS commence par `GP`. Elles comportent également un CRC, calculé par XOR des codes ASCII des précédents caractères.

Ce GPS dispose également d'un jeu de trames propre au constructeur, composé de commandes et de réponses, commençant par `PMTK`.

Le GPS envoie périodiquement certaines trames NMEA et répond aux instructions que l'on peut lui envoyer.

Parmi les trames intéressantes qu'envoie le GPS, on trouve :

- **GPGLL** : Contient notamment la latitude et la longitude.
- **GPRMC** : Contient notamment la latitude et la longitude.
- **GPGGA** : Contient notamment la latitude, la longitude et l'altitude.
- **PMTK705** : Contient le nom et la version du firmware du GPS. Réponse à la trame de requête `PMTK605`.

La trame **GPGGA** est celle qui contient le plus d'informations intéressantes. À titre d'exemple, voici sa décomposition :

\$GPGGA, 204444.000, 4042.6250, N, 07400.5054, W, 1, 6, 1.45, 278.7, M, -34.2, M,, *6C\r\n

- **204444.000** : Heure de calcul de la position $20h44m44s$
 - **4042.6250, N** : Latitude $40^{\circ}42'62,50''$ Nord
 - **07400.5054, W** : Longitude $074^{\circ}00'50,54''$ Est
 - **1** : Indicateur de qualité 1 = GPS
 - **6** : Nombre de satellites utilisés
 - **1.45** : Dilution de la position horizontale (HDOP)
 - **278.7, M** : Altitude par rapport au niveau moyen de la mer $278,7m$
 - **-34.2, M** : Hauteur du géoïde $-34,2m$
 - **vide** : Age des données GPS différentiel
 - **vide** : Identifiant de la station GPS différentiel
 - **6c** : CRC de **GPGGA, 204444.000, 4042.6250, N, 07400.5054, W, 1, 6, 1.45, 278.7, M, -34.2, M, , ***

La trame **GPGLL** contenant toutes les informations utiles, il semble inutile de laisser le GPS envoyer périodiquement d'autres trames que celle-ci.

La trame de commande **314 PMTK_API_SET_NMEA_OUTPUT** permet de configurer les trames qu'envoie périodiquement le GPS :

- **GPGLL** : Trame jamais envoyée.
 - **GPRMC** : Trame jamais envoyée.
 - **GPVTG** : Trame jamais envoyée.
 - **GPGGA** : Trame envoyée une fois tous les 1 calcul de position. Peut aller jusqu'à une trame tous les 5 calculs de position.
 - **GPGSA** : Trame jamais envoyée.
 - **GPGSV** : Trame jamais envoyée.
 - Les autres champs doivent rester à 0.
 - **29** : CRC de **PMTK314,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0***

Le GPS doit alors répondre une trame 001 PMTK_ACK : \$PMTK001,314,3*36\r\n signifiant que la commande a été prise en compte.

10.2 Fonctionnement du daemon

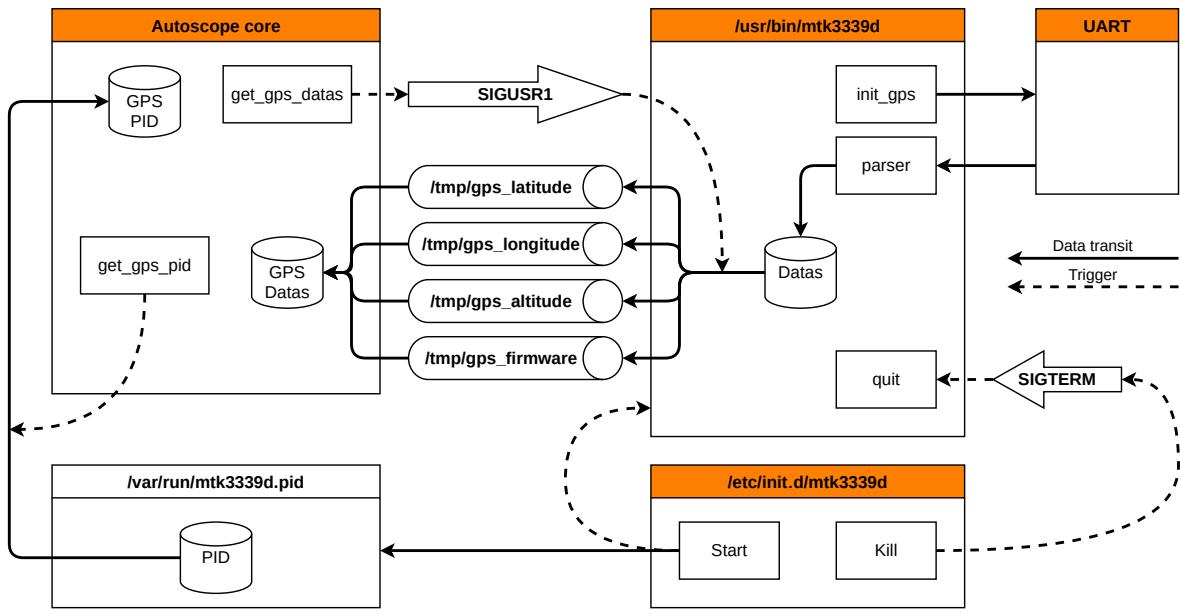


FIGURE 10.1 – Architecture logicielle du daemon d'interface avec le GPS

Le daemon `/usr/bin/mtk3339d` est invoqué et tué par son script d'initialisation `/etc/init.d/mtk3339d`.

Lors de son fonctionnement, il réceptionne les trames envoyées par le GPS sur la liaison série et en extrait les données intéressantes qu'il stocke dans sa mémoire. Ces données sont la latitude, la longitude, l'altitude, ainsi que le nom et la version du firmware, utilisés principalement comme débug.

La communication de ces données au logiciel principal du télescope se fait à la demande de ce dernier et par le biais d'un pipe FIFO pour chaque donnée.

Pour adresser sa requête des données au daemon (matérialisée par l'envoi d'un signal `SIGUSR1`), le logiciel principal a besoin de connaître le `pid` du daemon. Celui-ci est stocké dans le fichier `/var/run/mtk3339d.pid` créé par le script d'initialisation lors de l'invocation du daemon.

La fonction `init_gps` envoie la trame de configuration `PMTK314` décrite précédemment ainsi qu'une trame de requête des informations firmware `PMTK605`.

À l'usage, le daemon se lance automatiquement à l'allumage du système. On peut alors utiliser un programme de test **mtk3339d-test** contenant le fragment décrit ci-dessus du programme principal du télescope. Le test n'a pas eu l'occasion d'être fait dans un environnement où le GPS parvient à se connecter aux satellites, d'où les données nulles :

```
1 root@autoscope ~ #
2   mtk3339d-test
3     GPS daemon PID : 335
4     Recieved latitude : 0
5     Recieved longitude : 0
6     Recieved altitude : 0
7     Recieved firmware : AXN_2.31_3339_13101700
```

Chapitre 11

Stellarium

Ce chapitre n'a pas vocation à expliquer comment fonctionne le plugin de Stellarium développé par Thibaud LE DOLEDEC mais la procédure à suivre pour pouvoir utiliser ce plugin.

Ajouter un plugin à ce logiciel nécessite de modifier son code source et donc de compiler l'intégralité du logiciel.

11.1 Installation depuis les sources

Installation des dépendances (sur une distribution de type Debian/Ubuntu) :

```
1 ~ $ sudo apt-get install build-essential cmake zlib1g-dev libgl1-mesa-dev gcc g++
2                                     sudo apt-get install graphviz doxygen gettext git
3
```

Installation des dépendances Qt :

```
1 ~ $ mkdir DEV/
2 cd DEV/
3 wget http://download.qt.io/archive/qt/5.5/5.5.0/qt-opensource-linux-x64-5.5.0-2.run
4 chmod +x qt-opensource-linux-x64-5.5.0-2.run
5 ./qt-opensource-linux-x64-5.5.0-2.run
6 #enter a shitty email address (e.g. qt.qt@yopmail.com)
7 #install in /opt/Qt5.5.0/
8
9
10 git clone https://github.com/qt/qtftp
11 cd qtftp/
12 /opt/Qt5.5.0/5.5/gcc_64/bin/syncqt.pl --version 5.5.0
13 /opt/Qt5.5.0/5.5/gcc_64/bin/qmake
14 make
15 sudo make install
```

Téléchargement des sources de Stellarium et du plugin Autoscope :

```
1 ~/DEV $ wget https://github.com/Stellarium/stellarium/releases/download/v0.19.0/
2           stellarium-0.19.0.tar.gz
3 tar -xzf stellarium-0.19.0.tar.gz
4 cd stellarium-0.19.0/plugins/
5 git clone https://github.com/thibaudledo/Autoscope-Stellarium-plugin.git
6           plugins/Autoscope
```

Activation du plugin. Cela peut être fait en modifiant manuellement les sources de Stellarium ou en appliquant un patch de la modification fourni avec les sources du plugin :

```

1 ~/DEV/Stellarium-0.19.0 $
2   git init
3   git add CMakeLists.txt src/core/StelApp.cpp
4   git apply plugins/Autoscope/0001-enable-autoscope-plugin.patch
5
6   ### OR APPEND MANUALLY ###
7   line 369 : CMakeLists.txt
8       ADD_PLUGIN(Autoscope 1)
9   line 94 : src/core/StelApp.cpp
10      #ifdef USE_STATIC_PLUGIN_AUTOSCOPE
11          Q_IMPORT_PLUGIN(AutoscopeStelPluginInterface)
12      #endif

```

Export de l'emplacement de Qt :

```

1 export QTDIR=/opt/Qt5.5.0/5.5/gcc_64
2 export PATH=/opt/Qt5.5.0/5.5/gcc_64/bin:${PATH}
3 export LD_LIBRARY_PATH=${QTPATH}/lib:${LD_LIBRARY_PATH}

```

Compilation de Stellarium :

```

1 ~/DEV/Stellarium-0.19.0 $
2   mkdir -p builds/unix/
3   cd builds/unix/
4   cmake -DCMAKE_BUILD_MODE=Release -DCMAKE_INSTALL_PREFIX=/opt/stellarium ../../
5   make -j4

```

Installation de Stellarium :

```

1 ~/DEV/Stellarium-0.19.0/builds/unix $
2   sudo make install

```

11.2 Crédit d'un paquet binaire

Après la compilation il est possible d'installer le logiciel mais il est aussi possible de créer des paquets permettant de le distribuer. L'on peut créer :

- Un paquet contenant le code source de Stellarium et du plugin, prêt à être compilé.
- Un paquet contenant le logiciel compilé, à installer manuellement.
- Un paquet .deb contenant le logiciel compilé et destiné au logiciel de gestion de paquets des distributions GNU/Linux de la famille Debian.
- Un paquet .rpm contenant le logiciel compilé et destiné au logiciel de gestion de paquets des distributions GNU/Linux de la famille Red Hat.

```

1 ~/DEV/Stellarium-0.19.0/builds/unix $
2   make package_source
3   make package
4   cpack -G DEB
5   cpack -G RPM

```

11.3 Installation manuelle d'un paquet binaire

L'on choisit d'installer le logiciel dans le dossier `/opt` plutôt que `/usr/local` dans le but de l'isoler un peu et de permettre un développement moins risqué ainsi que la cohabitation d'une version Autoscope de Stellarium installée manuellement et d'une version classique installée depuis le gestionnaire de paquets.

Dans le dossier `/opt`, chaque logiciel installé dispose d'un dossier lui étant propre, tandis que dans `/usr` et `/usr/local`, l'arborescence est partagée à tous les logiciels (les binaires avec les binaires, les sources avec les sources, les icônes avec les icônes, etc.). Le dossier `/opt` n'est généralement pas utilisé par les gestionnaires de paquets.

Un paquet binaire est disponible sur le dépôt du projet, à l'adresse suivante :

<https://github.com/thibaudledo/Autoscope/releases/tag/alpha>

```
1 /opt #
2   wget https://github.com/thibaudledo/Autoscope/releases/download/alpha/
3     ↳ Stellarium-0.19.0-Linux.tar.gz
4   tar -xzf Stellarium-0.19.0-Linux.tar.gz
```

11.4 Lancement de Stellarium

La commande suivante peut être utilisée pour créer un raccourci dans un dock, un tableau de bord ou un menu :

```
1 ~ $
2   /opt/Stellarium-0.19.0-Linux/bin/stellarium
```

Troisième partie

Thomas ABGRALL

Chapitre 12

Hardware moteur

12.1 Cahier des charges du mouvement

Pour mouvoir le télescope dans des conditions optimales, il y a des contraintes spécifiques à respecter. Le télescope est un appareil qui se doit d'être très précis. Les mouvements étant assurés par les moteurs, il faut que ces derniers soient précis. Les critères à contrôler sont l'angle par pas et le nombre de pas.

Dans un souci de confort d'emploi le télescope doit pouvoir atteindre sa cible sans trop faire patienter l'utilisateur. Il a fallu faire un compromis entre la vitesse et la précision.

12.2 Les moteurs

Nous avons retenu deux type de moteurs :



FIGURE 12.1 – Moteur 17HM15-0904S et moteur S20STH30-0604A

Le premier a été choisi pour l'azimut et l'élévation car il a suffisamment de couple pour mouvoir le télescope, et avec précision. À l'origine, nous en avons trouvé un exemplaire sur les imprimantes 3D et nous avons vérifié s'il répondait à nos contraintes.

Le second est plus léger que le premier, moins puissant. Il est plus adapté pour changer le zoom au bout de la flèche du télescope. Il est indispensable de limiter le poids au bout de la flèche du télescope pour faciliter ses déplacements et surtout ne pas le déséquilibrer.

12.3 Le contrôleur

Le contrôleur est une carte qui à partir de fronts montants en entrée fait tourner un moteur pas à pas d'un angle précis. Cette carte A4988 opto-isolée permet l'entraînement de moteur pas à pas à micropistes Allegro A4988. Il est doté d'une protection contre les surintensités et la surchauffe, ainsi que de cinq résolutions différentes en micropas (jusqu'à 1/16 pas). Il fonctionne de 8V à 35V et peut fournir jusqu'à environ 1A par phase sans dissipateur de chaleur ni flux d'air forcé.

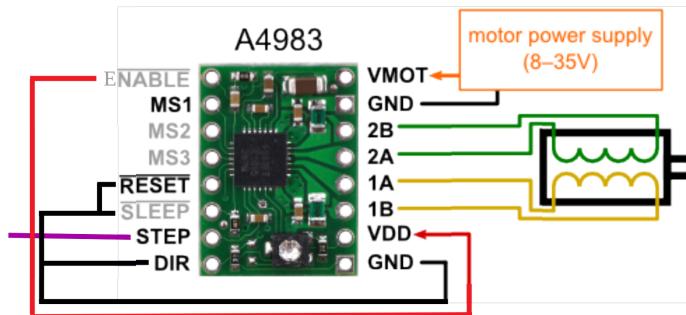


FIGURE 12.2 – Photo du contrôleur A4988

12.4 Validation des moteurs et du câblage

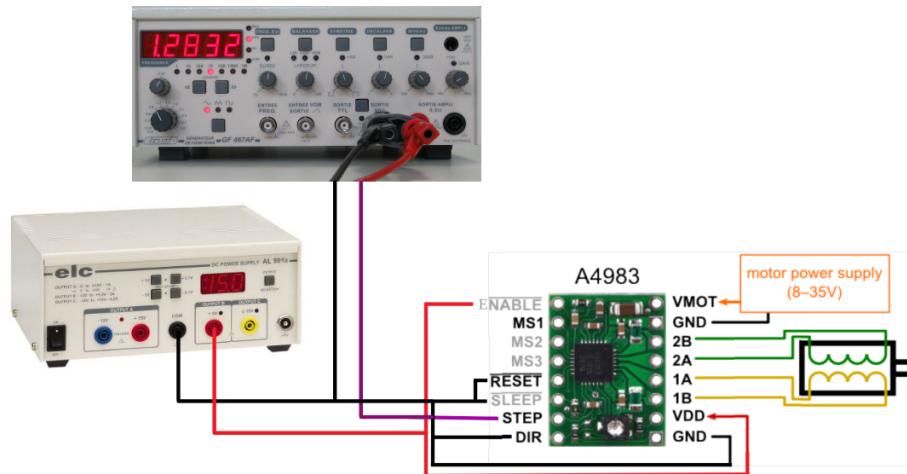


FIGURE 12.3 – Schéma de câblage d'un contrôleur moteur sur la Raspberry Pi

Durant ce test nous avons vérifié le bon fonctionnement du moteur et contrôleur moteur.

Pour nous assurer que les moteurs de rotation et inclinaison sont suffisamment coupleux et résistant nous avons placé une courroie autour de l'axe du moteur et au bout de cette dernière nous avons attaché une masse d'environ 10kg. Ce test fut un succès et nous avons validé le choix des moteurs.



FIGURE 12.4 – Test de la puissance du moteur

Nous avons profité du test pour manuellement utiliser les modes de pas en appliquant aux entrées MS1, MS2, MS3 du 5V en suivant la documentation ci-dessous :

MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

FIGURE 12.5 – Documentation microStep a4988

Chapitre 13

Driver commande des moteurs

13.1 Introduction

Un driver est un programme développé avec un langage reconnu pour sa proximité avec le matériel, le langage C. Et c'est la raison d'être des drivers. Ces derniers font la passerelle entre les applications qui se trouvent dans la partie utilisateur (user-space) et le matériel.

Le plus simple des exemples est celui du clavier. Lorsque j'appuie sur une touche pour écrire ce rapport un contact électrique est émis et informe le micro-processeur qu'une touche a été appuyée. Puis un programme, le driver du clavier, viens contrôler les entrées pour savoir quelle touche a été pressé. Et envoie cette information au système d'exploitation pour que la lettre soit affichée.

Le driver moteur a pour mission de permettre de contrôler les moteurs d'azimut, d'élévation et de zoom du télescope. D'informer le logiciel principal de l'accomplissement d'une manœuvre, théoriquement (explication chapitre 13.2). Il assure la sécurité du système de pilotage à bas niveau. Le système de sécurité consiste à interrompre le mouvement d'élévation ou de zoom si ils atteignent leur fin de course, le mouvement d'azimut n'étant pas soumis à cette contrainte.

13.2 Fonctionnalités du driver

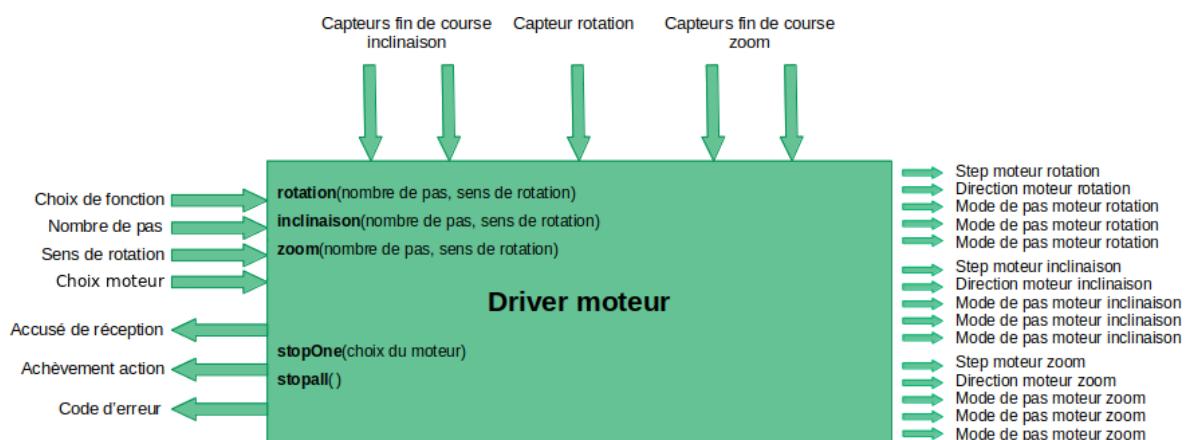


FIGURE 13.1 – Schéma des entrées / sorties du driver moteur

La figure ci-dessus représente le driver contrôlant les moteurs. À gauche du bloc il y a les entrées et les sorties du driver d'un point de vue informatique.

Les entrées regroupent l'ensemble des données qu'attend le driver pour entreprendre une action, nous retrouvons de ce fait, le choix de l'action à réaliser, le nombre de pas à parcourir, et le sens de rotation et le choix du moteur. Les choix des actions à faire sont en gras à l'intérieur du bloc Driver. Comme vous pouvez le voir toutes les actions n'ont pas besoin de toutes les informations demandées en entré (comme l'action `stopone`), dans ces cas seules les entrées nécessaires seront prises en compte.

Concernant les sorties informatiques, il y a l'acquittement de réception qui a pour but de rassurer l'application en lui indiquant ce qu'il a reçu, ainsi l'application peut vérifier que les données envoyées sont correctes. Si il y a une erreur de communication l'application demande l'arrêt des moteurs et renvoi l'ordre ajusté (en prenant en compte le déplacement qui a été effectué dans l'erreur).

Nous trouvons ensuite l'achèvement d'action, lorsque une action arrive à son terme (mouvement d'azimut, d'élévation, etc.), le driver l'indique à l'application, ainsi si jamais il y a un ajustement à effectuer elle renverra un ordre pour bien être placé. Et pour finir ce côté informatique, il y a le retour de code d'erreur qui survient lorsqu'un incident arrive dans le driver (ressource inaccessible (pin, timer, interruption, problème de fermeture driver ou d'ouverture, etc.).

Sur le dessus du driver nous trouvons 5 entrées (matérielles) capteurs. Chacun d'eux prévient de l'arrivée en butée d'un axe, excepté pour le capteur rotation azimutale qui indique juste la position du télescope en rotation lorsqu'il est activé.

Ce capteur a deux rôles, durant la phase de développement il nous permettra de déterminer le nombre exact et réel du pas pour un tour complet du télescope (ce qui permettra de développer l'algorithme de positionnement du télescope dans l'application principale). Le second rôle, plus tard, sera de faire du recalibrage de la centrale inertie en cas de décalage entre les résultats de ce capteur et ceux de la centrale inertie.

Au final à droite du bloc nous trouvons les sorties (matérielles) du driver. Ces dernières repètent le même chose pour chacun des 3 moteurs : une pin pour le front montant pour lancer un pas moteur, le sens de rotation (0 ou 1), et 3 pins pour le choix du mode pas de rotation du moteur (pas complet, demi-pas, quart de pas, etc.).

Deux exemples de fonctionnement du driver avec ces entrées/sorties :

- Faire tourner le zoom de 50 pas dans le sens horaire :
 1. Réception des nombres de l'application [3 ; 50 ; 0] pour choisir la fonction zoom, nombre de pas, sens de rotation horaire.
 2. Envoi de l'accusé de réception, à l'application, contenant les nombres [3 ; 50 ; 0].
 3. Exécution de la fonction zoom avec les données [3 ; 50 ; 0] en argument.
 4. Écrit dans la sorte du moteur zoom sens de rotation : 0.

5. Écrit en sortie de mode de pas du moteur zoom la configuration pour faire des pas complet.
 6. Créer un timer qui va permettre de créer un front montant, tant que l'on n'a pas atteint le nombre de pas à parcourir.
 7. Si l'on a parcouru tous les pas complet et qu'il faut réaliser des pas plus fins, pour atteindre au plus près l'angle demandé, on change le mode de pas en 16ème de pas.
 8. Une fois tous les 16ème de pas effectués on envoi de l'achèvement d'action contenant le nombre 1.
 9. Le driver réinitialise ses variables et ressources (timer, etc.).
 10. Mise en attente d'un nouvel ordre, la boucle est bouclée.
- Arrêter tout les moteurs
 1. Envoi au driver le nombre 5 pour choisir la fonction `stopall`.
 2. Réception de l'accusé de réception contenant le nombre 5.
 3. Réinitialise les variables et ressources.
 4. Réception de l'achèvement d'action contenant le nombre 1.

Une fois une manœuvre terminée le driver informe l'application que l'action demandée est arrivé à son terme. Cependant d'un point de vue driver moteur il n'y a pas de possibilité de vérifier si le télescope est bien placé là où on lui a demandé de se placer. Si une personne ou objet bloque le mouvement du télescope ou des moteurs ces derniers ne renverront pas d'avertissement. C'est donc la centrale inertie qui contrôlera le bon placement du télescope.

Les capteurs de fin de course servent de sécurité, cette partie est indépendante du reste. Elle a pour but d'arrêter un moteur qui à atteint sa position maximale, hormis pour le capteur de rotation qui a pour utilité dans un premier temps de faire une interruption lorsque un tour complet.

Avec cette information on pourra déterminer le nombre de pas moteur qu'il faut faire pour faire un tour complet de télescope sur lui même. Et dans un second temps de donner un point de repère pour les recalibrages de la centrale inertie.

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). Augmenter la division de pas permet d'affiner la rotation afin d'atteindre l'angle le plus exact possible compte tenu de ce qui a été commandé. Ainsi lorsqu'un moteur a accompli environ 90% de la distance à parcourir, on change le mode de pas afin d'arriver au plus proche de l'angle exact.

13.3 Interactions avec le driver

Le driver est utilisé à partir d'une application situé dans le user-space du Linux embarqué. Pour cela j'ai étudié et testées plusieurs solutions.

La 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 peu sécurisée.

La troisième solution qui m'a été proposée par Vincent POULAILLEAU est d'utiliser la fonction `iocctl`. 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.

13.4 Intégration du driver

Pour que le driver soit utilisable dans le Linux embarqué Yocto, j'ai créé deux recettes. L'une viens chercher dans le Github du projet les sources du driver, cette recette permet l'importation du driver lors du build du Yocto (avec la commande `bitbake`).

La seconde recette permet d'ajouter l'application d'interface avec le driver dans le user-space du Yocto. Pour lancer l'application dans le Yocto dans la Raspberry-Pi il suffira d'écrire dans le terminal (de la cible) le nom de l'application : `a4988-test`

13.5 Validation des fonctionnalités de bases

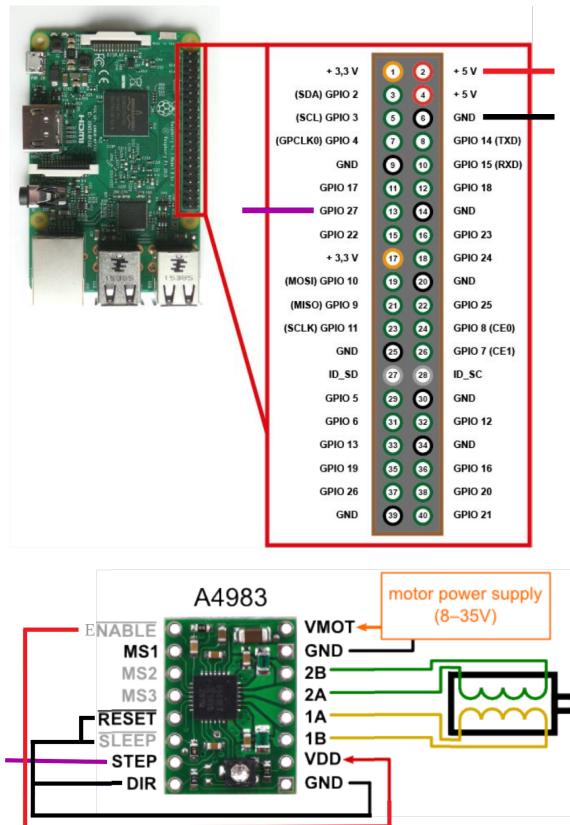


FIGURE 13.2 – Schéma de câblage d'un contrôleur moteur sur la Raspberry Pi

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

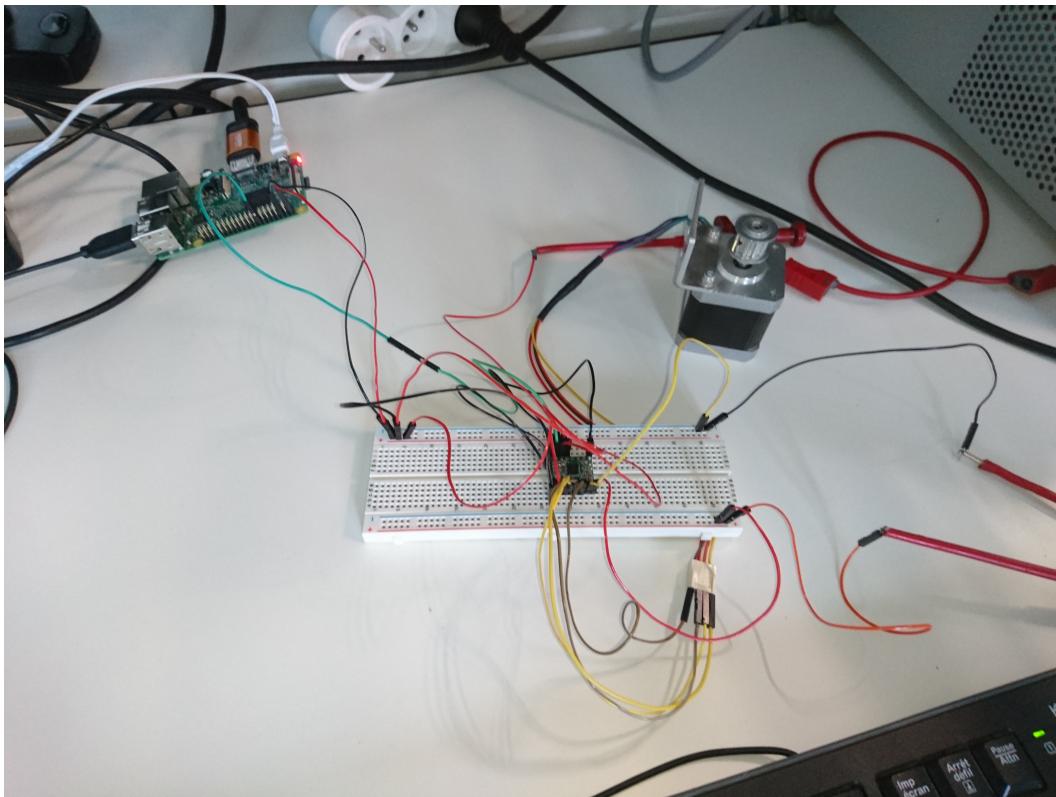


FIGURE 13.3 – Photo du banc de test

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.

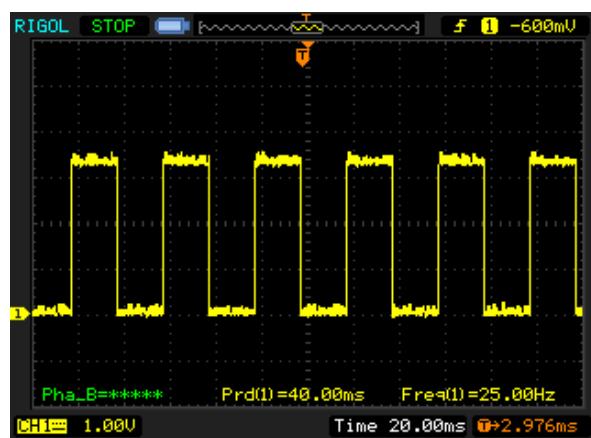


FIGURE 13.4 – Mesure à l'oscilloscope de la commande moteur générée

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.