



**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

6 mai 2019

# Table des matières

<b>Table des matières</b>	<b>1</b>
<b>I Partie de groupe</b>	<b>3</b>
<b>1 Cahier des charges</b>	<b>4</b>
1.1 Introduction . . . . .	4
1.2 Nécessité du traçage d'astre . . . . .	4
1.3 Nécessité d'une centrale inertuelle et d'un GPS . . . . .	5
1.4 Aperçu d'un logiciel de traitement d'images en astronomie . . . . .	6
1.5 Spécifications techniques . . . . .	7
1.5.1 Fonctionnalités obligatoires . . . . .	7
1.5.2 Fonctionnalités envisagées . . . . .	7
<b>II Thomas LEPOIX</b>	<b>8</b>
<b>2 Hardware</b>	<b>9</b>
2.1 Architecture . . . . .	9
2.2 Conception du circuit . . . . .	10
2.3 Contraintes de design . . . . .	11
2.3.1 Contraintes électromagnétiques . . . . .	11
2.3.2 Contraintes mécaniques . . . . .	11
2.4 Bon de commande . . . . .	14
2.5 Fichiers de fabrication . . . . .	15
2.6 Modélisation 3D . . . . .	16
2.6.1 Allure générale de la carte . . . . .	16
2.6.2 Connecteurs des moteurs . . . . .	17
2.6.3 Emboîtement des cartes . . . . .	17
2.6.4 Repères des connecteurs . . . . .	18
2.6.5 Intégration du modèle 3D au télescope . . . . .	19
2.7 Fabrication et connectique . . . . .	19
2.8 Validation . . . . .	21
2.8.1 Tests de continuité et de fuite de courant . . . . .	21
2.8.2 Test de l'environnement des interrupteurs de butée . . . . .	21
2.8.3 Test de l'alimentation . . . . .	22
<b>3 Système d'exploitation</b>	<b>24</b>
3.1 Support de la caméra . . . . .	24
3.2 Étude du transfert du flux vidéo sur le réseau . . . . .	25
3.3 Splash screen . . . . .	25
3.4 Hotspot Wifi . . . . .	28
3.5 Serveur FTP . . . . .	31

3.6	Driver helloworld . . . . .	32
3.7	Programme helloworld . . . . .	34
3.8	Daemonisation du programme helloworld . . . . .	35
3.9	Support de la liaison UART et communication avec le GPS . . . . .	38
3.10	Support du bus I2C et communication avec l'IMU . . . . .	39
<b>4</b>	<b>Stellarium</b>	<b>41</b>
4.1	Installation depuis les sources . . . . .	41
4.2	Création d'un paquet binaire . . . . .	42
4.3	Installation manuelle d'un paquet binaire . . . . .	43
4.4	Lancement de Stellarium . . . . .	43

**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.

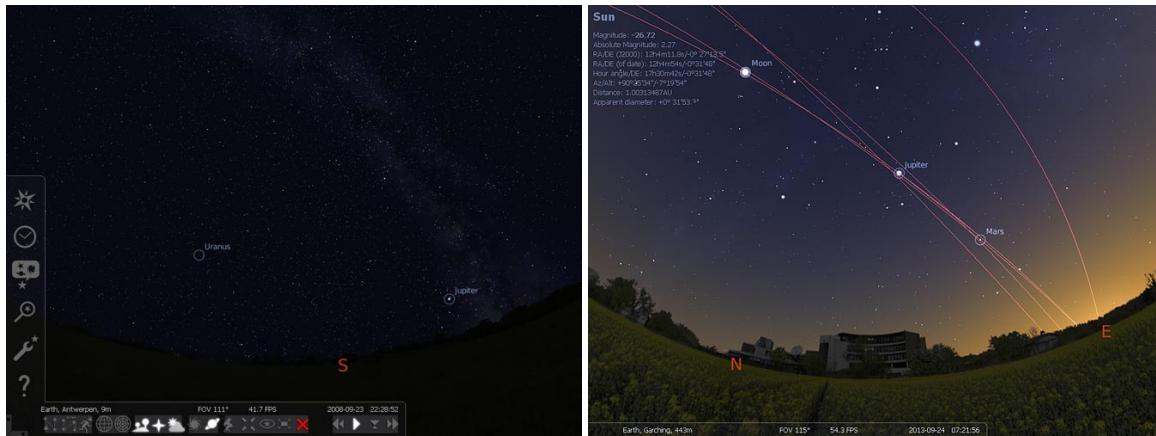


FIGURE 1.2 – Captures d'écran de Stellarium

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.

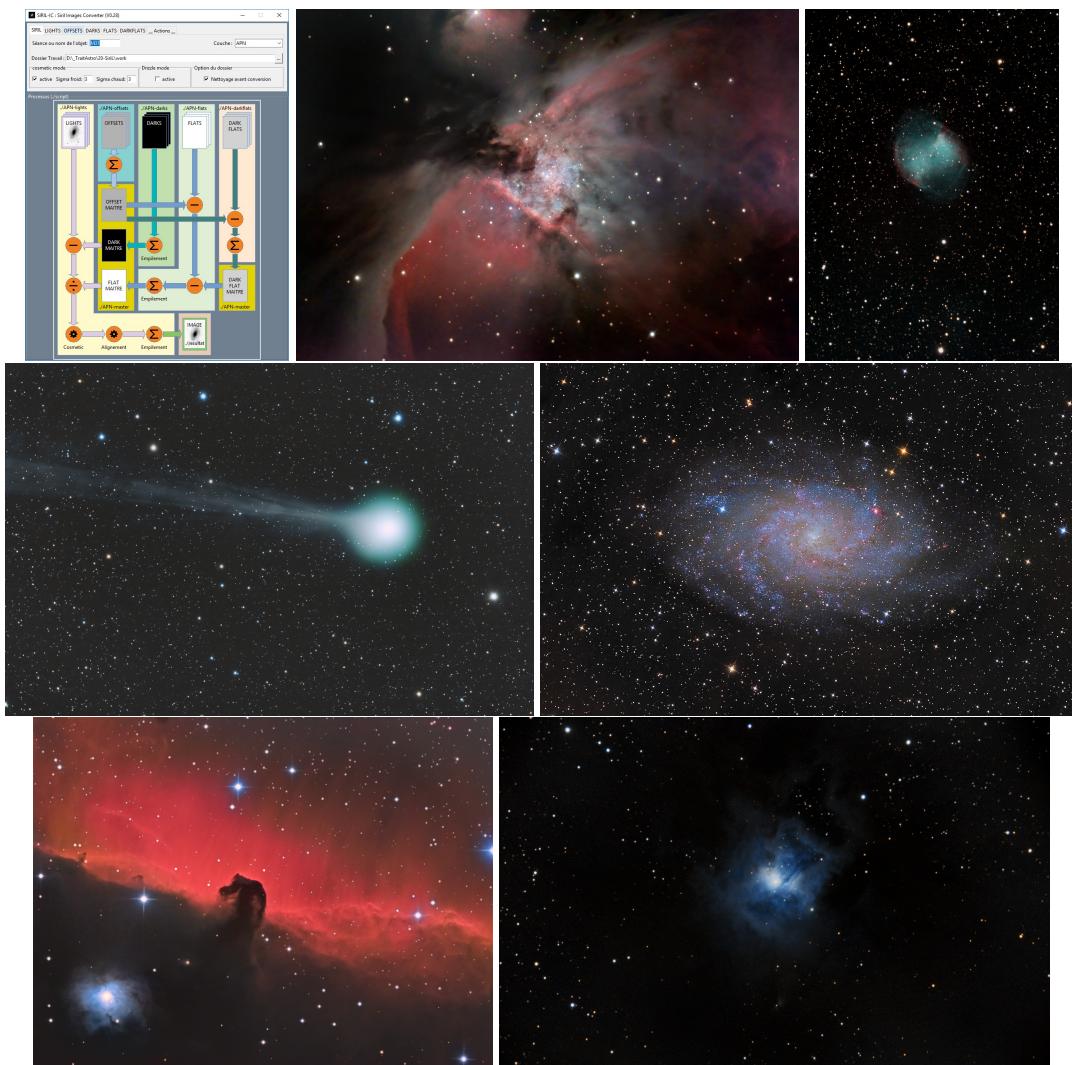


FIGURE 1.3 – Aperçu de Siril et de quelques clichés améliorés par son utilisation

## 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.

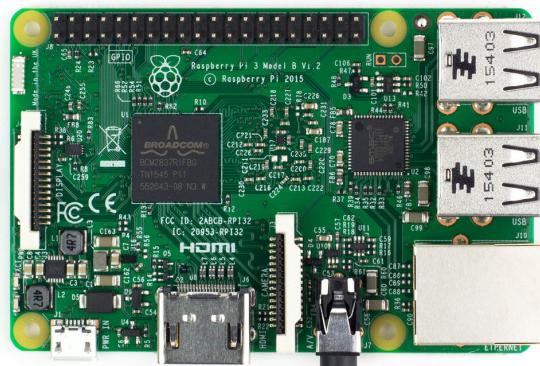


FIGURE 1.4 – Raspberry-Pi 3 B

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	Siril ?
Optionnel	Écran tactile	
Optionnel	Interface de pilotage via l'écran	
Optionnel	Autonomie énergétique	
Optionnel	Mouvement de zoom	Secteur

FIGURE 1.5 – Tableau récapitulatif des exigences du projet

**Deuxième partie**

**Thomas LEPOIX**

## Chapitre 2

# Hardware

### 2.1 Architecture

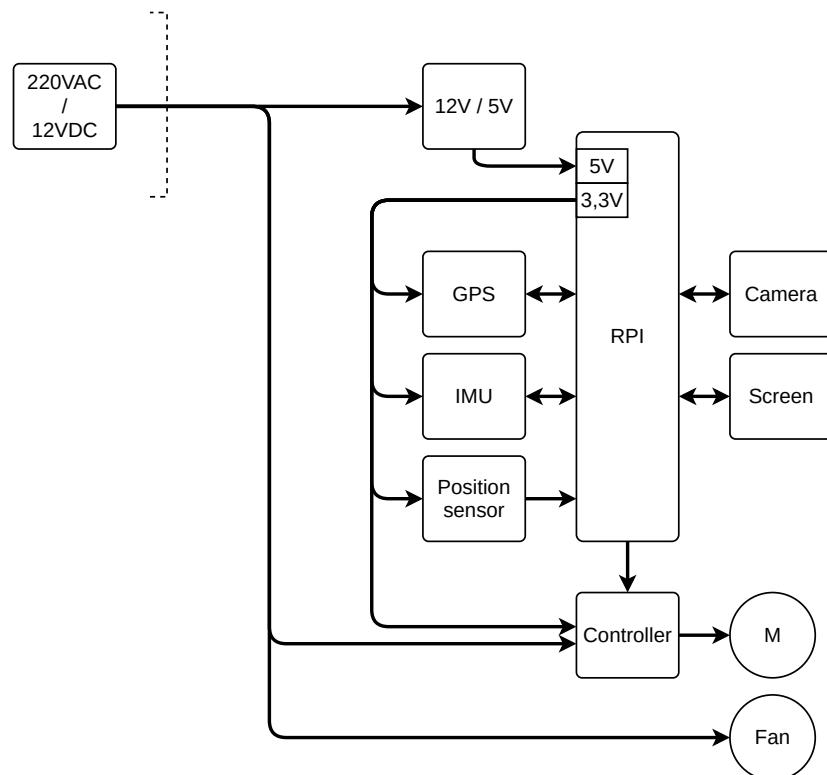


FIGURE 2.1 – Schéma structurel de premier niveau du télescope

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.

## 2.2 Conception du circuit

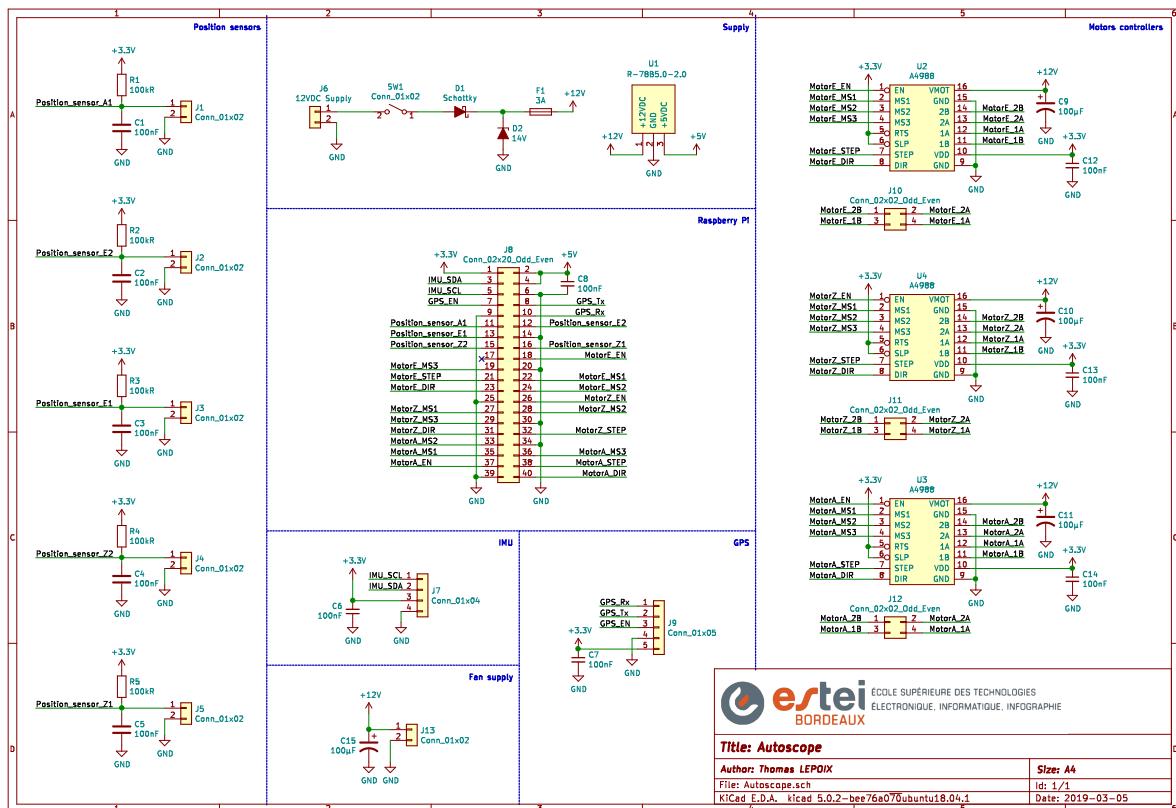


FIGURE 2.2 – Schéma structurel de la carte du télescope

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 :

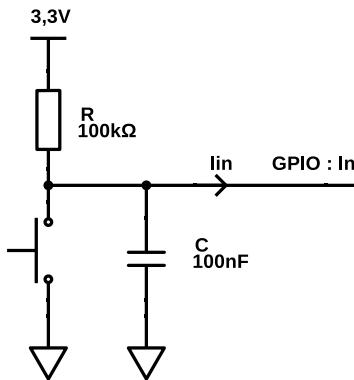


FIGURE 2.3 – Schéma de l'environnement des capteurs de butée

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.

## 2.3 Contraintes de design

### 2.3.1 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.

### 2.3.2 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.

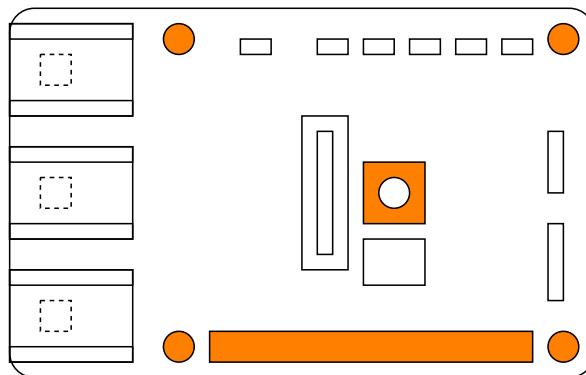


FIGURE 2.4 – Schéma mécanique de la carte vue de dessus

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 2.5 – 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.

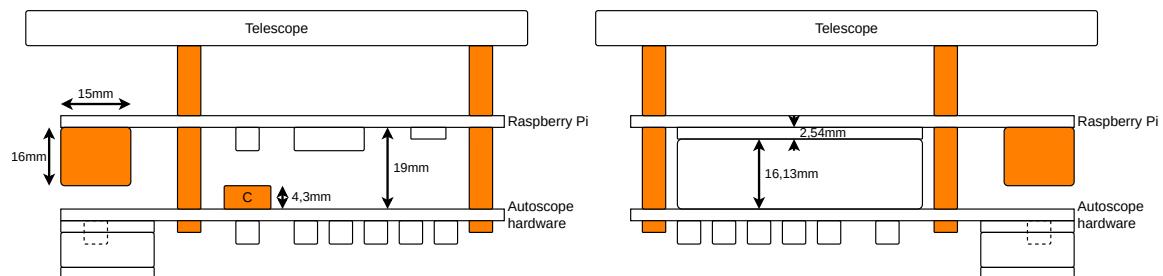
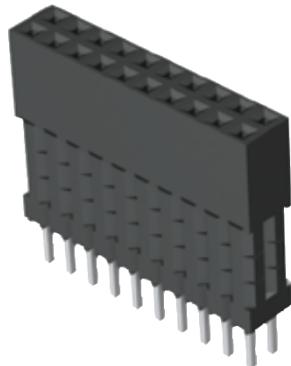


FIGURE 2.6 – Schéma mécanique de la carte vue de profil

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



---

FIGURE 2.7 – Type de header utilisé

## 2.4 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.

FIGURE 2.8 – Bon de commande du matériel du télescope

## 2.5 Fichiers de fabrication

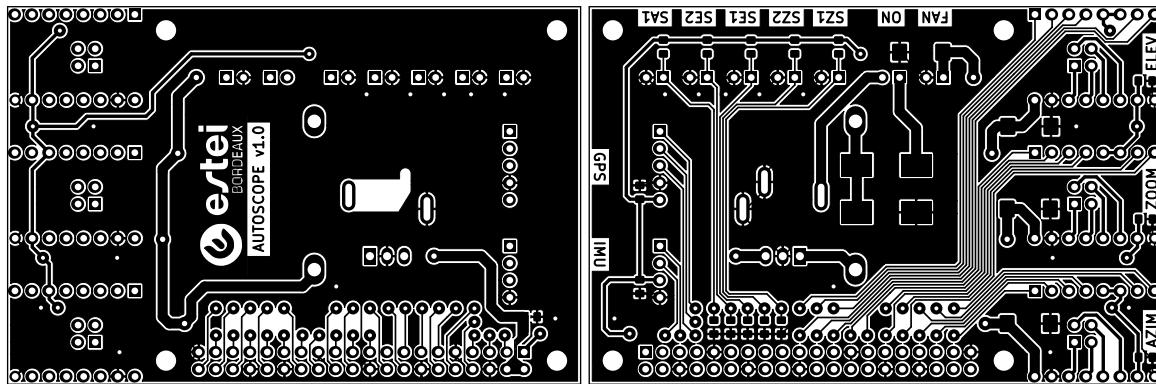


FIGURE 2.9 – Faces supérieure et inférieure du typon de la carte  
(respectivement vues de dessus et de dessous)

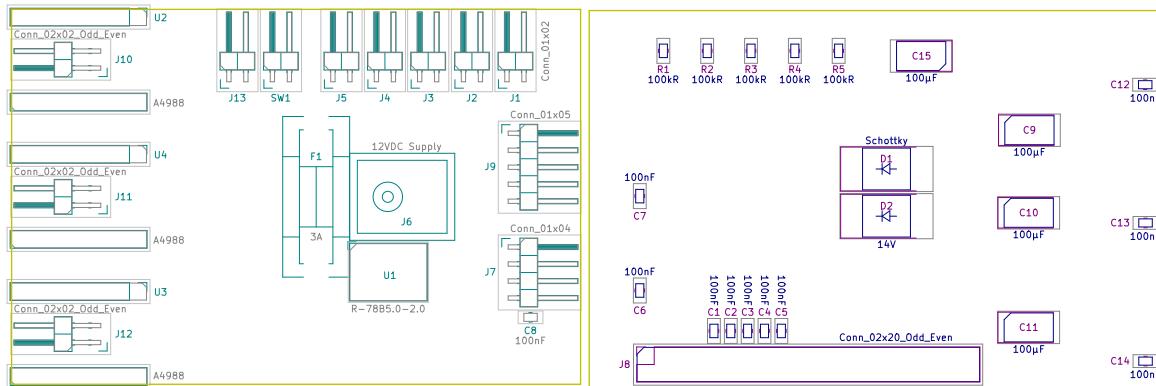


FIGURE 2.10 – Faces supérieure et inférieure de l’implantation des composants  
(respectivement vues de dessus et de dessous)

## 2.6 Modélisation 3D

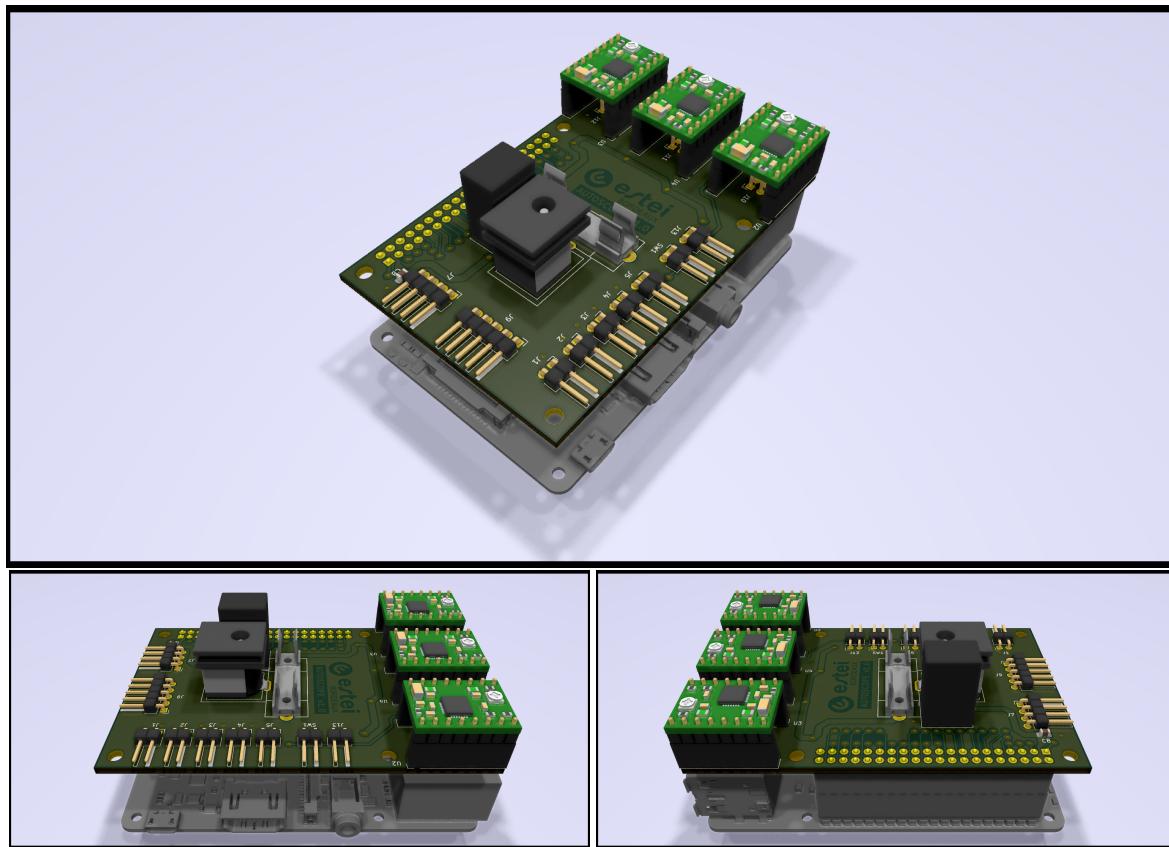


FIGURE 2.11 – Modélisation 3D de la carte pluggée sur la Raspberry-Pi

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.

### 2.6.1 Allure générale de la carte

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

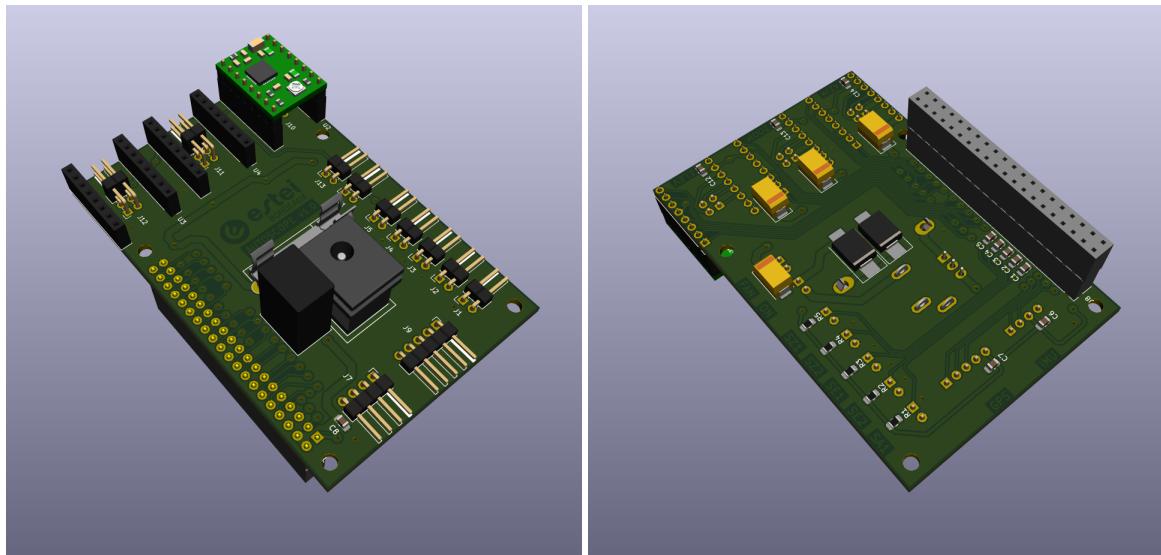


FIGURE 2.12 – Modélisation 3D de la carte avec et sans modules de contrôle moteur

### 2.6.2 Connecteurs des moteurs

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

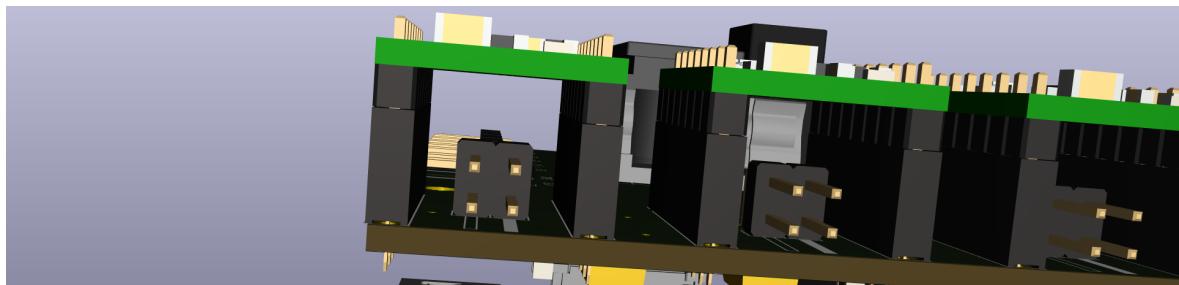


FIGURE 2.13 – Modélisation 3D de la carte : zoom sur les connecteurs moteurs

### 2.6.3 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 :

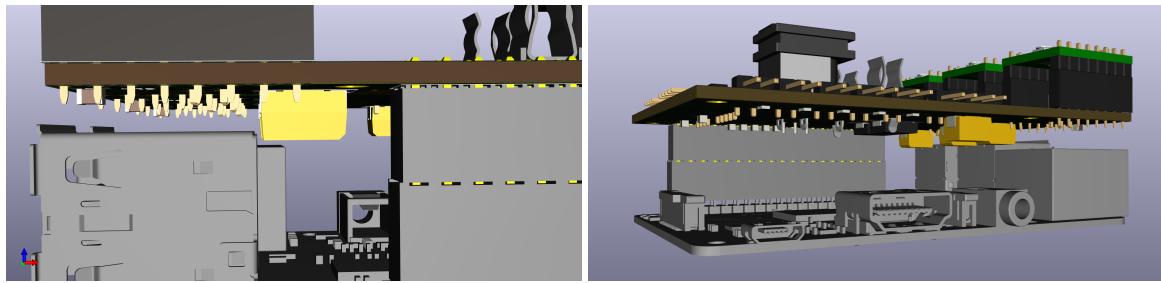


FIGURE 2.14 – Modélisation 3D de la carte : zoom sur l'emboîtement des cartes

#### 2.6.4 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.

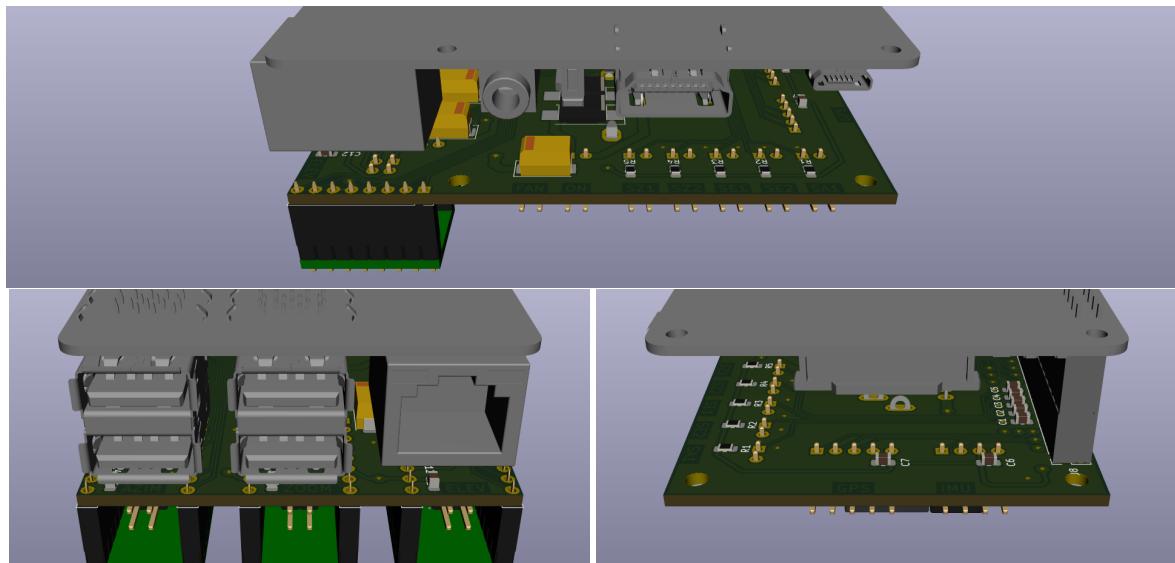


FIGURE 2.15 – Modélisation 3D de la carte : zoom sur les repères des connecteurs

- 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).

### 2.6.5 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 :

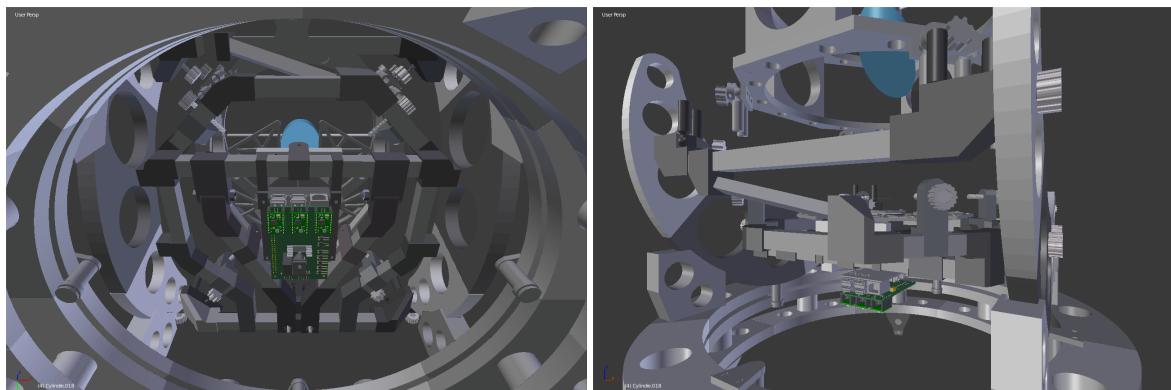


FIGURE 2.16 – Modélisation 3D du télescope grossièrement équipé de ses cartes électroniques

## 2.7 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 le contrôleur ne pose pas de problème.



FIGURE 2.17 – 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.

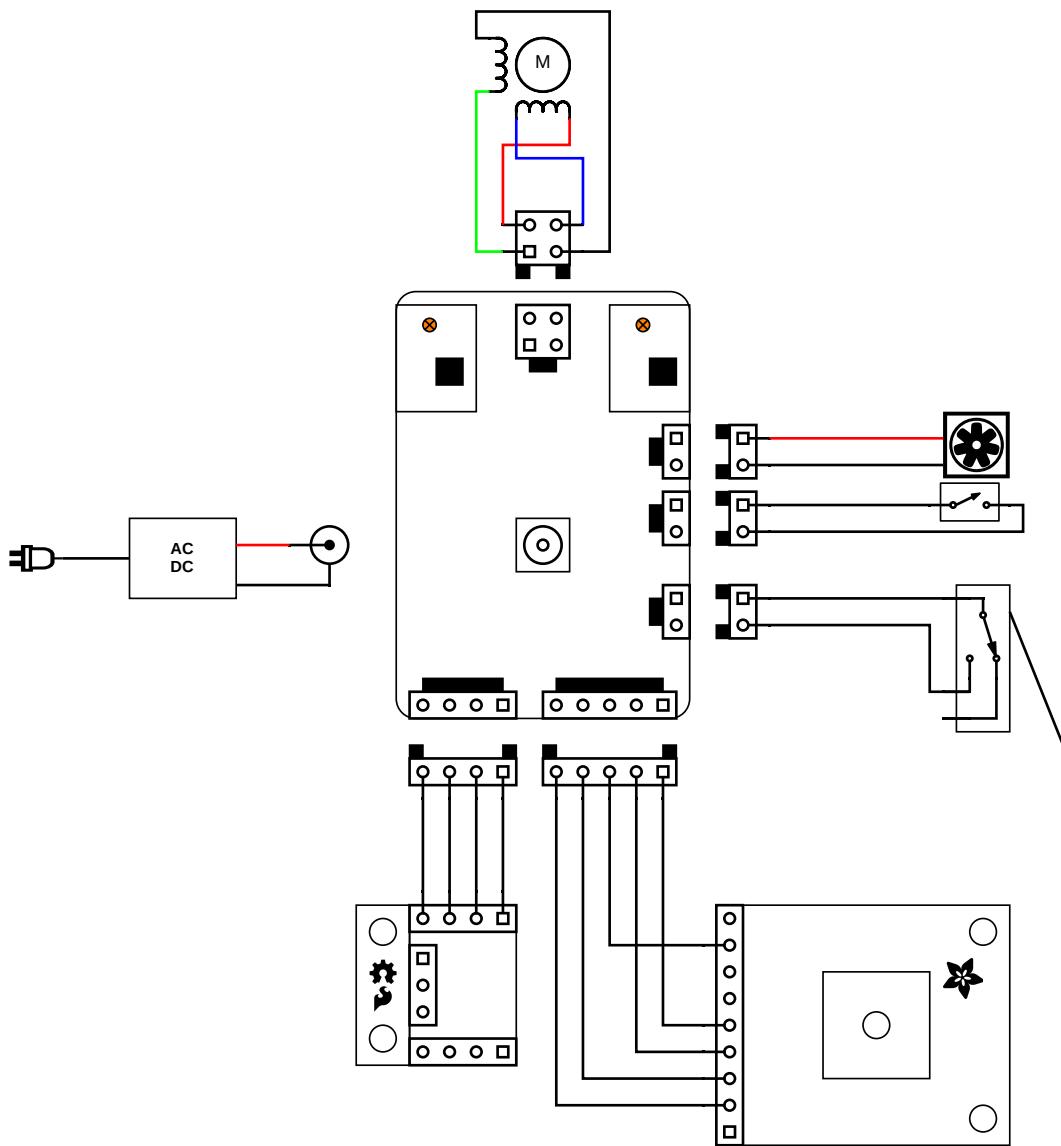


FIGURE 2.18 – Schéma de raccordement des périphériques de la carte

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.

## 2.8 Validation

### 2.8.1 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.

### 2.8.2 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 :

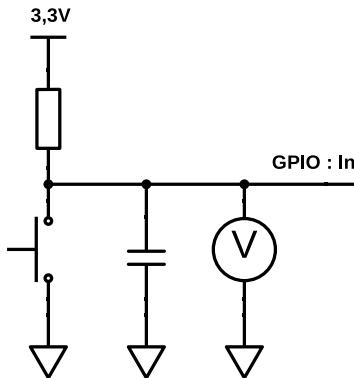


FIGURE 2.19 – Procédure de test de l'environnement des boutons

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 2.20 – Mesure de la tension de sortie des boutons

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

FIGURE 2.21 – 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.

### 2.8.3 Test de l'alimentation

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

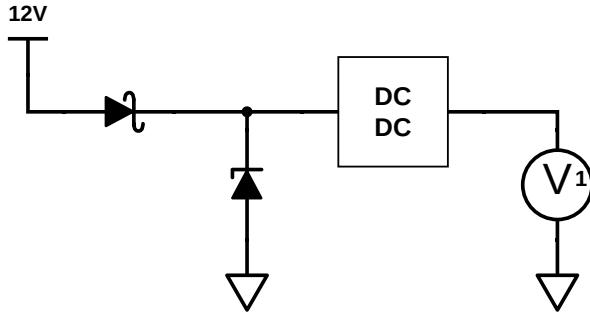


FIGURE 2.22 – Procédure de test du convertisseur DC/DC

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.

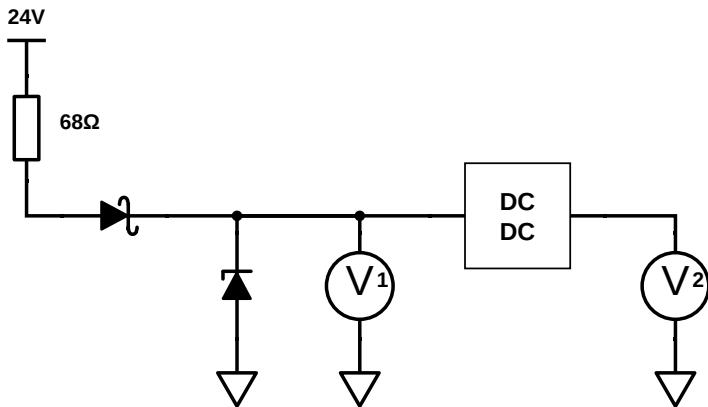


FIGURE 2.23 – Procédure de test de la protection aux surtensions

Pour une tension d'entrée  $V_{in} = 23,3V$ , l'on a des tensions  $V_1 = V$  et  $V_2 = 5,03V$ . 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.

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 3

# Système d'exploitation

### 3.1 Support de la caméra

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

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

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

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

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

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

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

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

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

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

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

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



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

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

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

```
include/linux/linux_logo.h :

1 extern const struct linux_logo logo_autoscope_clut224;

drivers/video/logo/logo.c :

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

drivers/video/logo/Makefile :

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

drivers/video/logo/Kconfig :

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

On réalise un patch de ces modifications :

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

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

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

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

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

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

```

1 FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:" 
2 
3 SRC_URI_append_raspberrypi3 += " \
4     file://0001-Autoscope-logo.patch \
5     file://logo.cfg \
6     file://logo_autoscope_clut224.ppm \
7     "
8 
9 do_patchprepend() {
10     cp ${WORKDIR}/logo_autoscope_clut224.ppm ${S}/drivers/video/logo/
11 }

```

Voici donc l'arborescence associée à la recette :

```

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

```

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



FIGURE 3.1 – 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 3.2 – Écran de démarrage du noyau Linux à droite et écran d'accueil à gauche

### 3.4 Hotspot Wifi

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

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

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

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

```
1 CONFIG_BRIDGE=m
2 CONFIG_IP_NF_TARGET_MASQUERADE=m
3 CONFIG_NETFILTER=m
4 CONFIG_NF_CONNTRACK_IPV4=m
5 CONFIG_NF_NAT_IPV4=m
6
7 CONFIG_IP_NF_IPTABLES=m
8 CONFIG_IP_MULTIPLE_TABLES=m
9 CONFIG_NETFILTER_NETLINK_ACCT=m
10 CONFIG_NETFILTER_XT_MATCH_NFACCT=m
11 CONFIG_NETFILTER_XT_CONNMARK=m
12 CONFIG_NETFILTER_XT_TARGET_CONNMARK=m
13 CONFIG_NETFILTER_XT_MATCH_CONNMARK=m
```

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

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

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

```

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

```

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

1 FILESEXTRAPATHS_prepend := "${THISDIR}/files:"
2

```

```

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

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

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

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

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

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

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

```

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

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

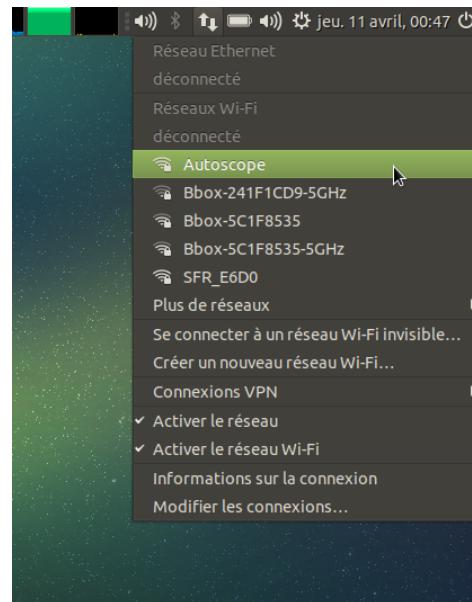


FIGURE 3.3 – Connexion à la Raspberry-Pi depuis un ordinateur distant

### 3.5 Serveur FTP

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

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

```

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

```

`meta-autoscope/recipes-connectivity/vsftpd/vsftpd_\%.bbappend`:

```

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

`meta-autoscope/recipes-connectivity/vsftpd/files/vsftpd.user_list`:

```

1 autoscope
```

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

```

1 listen=YES
2 anonymous_enable=NO
3 local_enable=YES
4 write_enable=YES
5 local_umask=022
6 dirmessage_enable=YES
7 xferlog_enable=YES
8 connect_from_port_20=YES
9 xferlog_std_format=YES
10 ftpd_banner=Welcome to Autoscope FTP service.
11 ls_recurse_enable=YES
```

```

12 pam_service_name=vsftpd
13 userlist_deny=NO
14 userlist_enable=YES
15 use_localtime=YES
16 chroot_local_user=YES
17 allow_writeable_chroot=YES
18 tcp_wrappers=YES
19 user_sub_token=$USER
20 local_root=/home/$USER

```

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

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

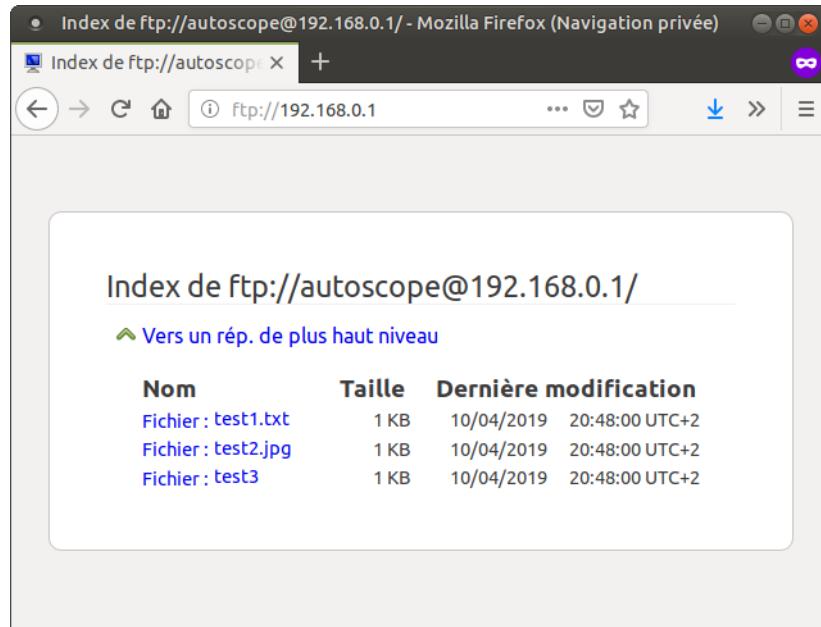


FIGURE 3.4 – Accès au serveur FTP de la Raspberry-Pi depuis un navigateur

- `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.

## 3.6 Driver helloworld

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

**hello.c :**

```

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

```

**Makefile :**

```

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

```

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

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

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

```

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

```

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

```

1 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   "

```

## 3.7 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.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

```

## 3.8 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 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
15 case "$1" in
16     start)
17         echo -n "Starting ${DESC}: ${NAME}... "
18         # start-stop-daemon -S -b -C -q -x ${DAEMON} > /dev/tty3 #-C don't work with busybox
19         ${DAEMON} > /dev/tty3 &
20         echo "done"
21         ;;
22     stop)
23         echo -n "Stopping ${DESC}: ${NAME}... "
24         start-stop-daemon -K -q -x ${DAEMON}
25         echo "done"
26         ;;
27     restart)
28         $0 stop
29         $0 start
30         ;;
31     status)
32         start-stop-daemon -T -q -x ${DAEMON}
33         case "$?" in
34             0)
35                 echo "Program ${NAME} is running (`pidof ${NAME}`)."
36                 ;;
37             1)
38                 echo "Programm ${NAME} is not running and the pid file exists."
39                 ;;
40             3)
41                 echo "Programm ${NAME} is not running."
42                 ;;
43             4)
44                 echo "Unable to determine ${NAME} status."
45                 ;;
46         esac
47         ;;
48     *)
49         echo "Usage: $0 {start|stop|restart|status}"
50         exit 1
51         ;;
52     esac
53 exit 0

```

Les messages produits par le daemon sont redirigés vers la console `/dev/tty3` accessible par la combinaison de touches **[CTRL]**, **[ALT]**, **[F3]**.

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/${bindir},g' \
23       -e 's,/var/${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
8       ./etc/rc5.d/S99helloworld
9       ./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.

```

### 3.9 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 confirmés 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.

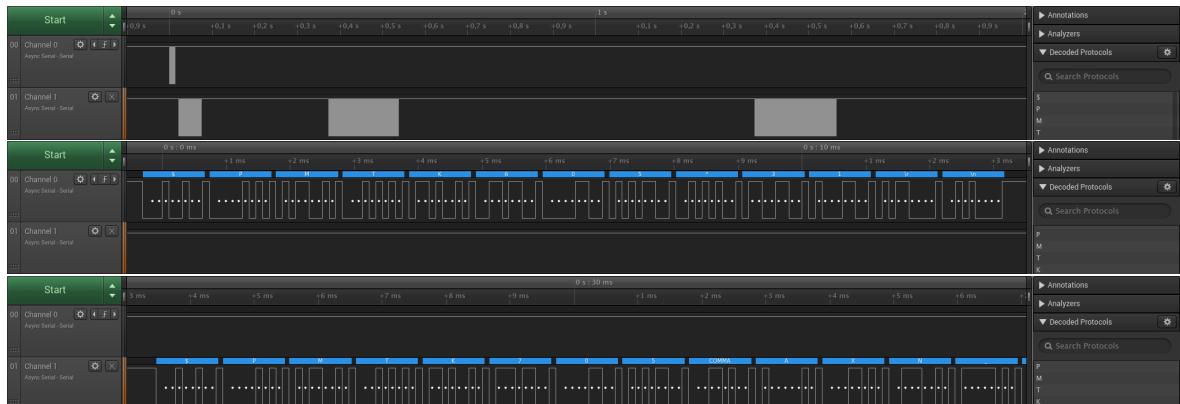


FIGURE 3.5 – Trame UART relevée à l'analyseur logique

### 3.10 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:      - - - - - - - - - - - - - - - - - - - - - - - 
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

```

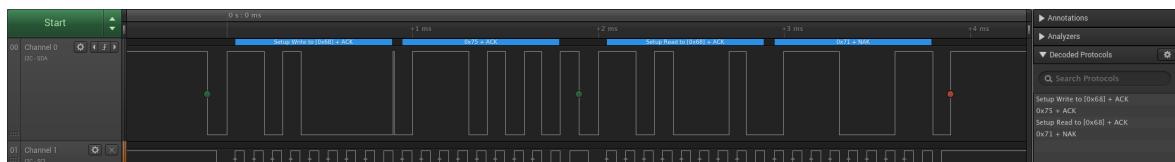


FIGURE 3.6 – Trame I2C relevée à l'analyseur logique

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

## Chapitre 4

# 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.

### 4.1 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
       ↳ 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

```

## 4.2 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

```

## 4.3 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
```

## 4.4 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
```