ECAM Strasbourg-Europe — Promotion 2018

Technical Project Final Report

Final Report
IR 3D Scanner Robot

Project #26 : IR 3D Scanner Robot

Group:
Nicolas Douard
Pauline Bosc
Jean-Felix Boyer
Romain Fredelisy
Antoine Toulemonde
Promotion 2018

Supervisor:
Mr. Arnaud Vivé

Client:
Mr. Yves Gendrault

Project beginning:

September 13, 2016

Project end:

April 9, 2016

Version 2 – April 9, 2017

This page intentionally left blank.

Contents

Special thanks

Special thanks to Mr. Gendrault and Mr. Vivé for their confidence, their clear explanations and the time they offered us during tutored sessions.

This page intentionally left blank.

Table of acronyms and abbreviations

ECS Embedded Control System

GCS Ground Control Station

PCB Printed circuit board

PWM Pulse Width Modulation

RC Radio Control

SSH Secure Shell

UGV Unmanned Ground Vehicle

US Ultrasonic

List of Figures

List of Tables

This page intentionally left blank.

1 Introduction

Our goal is to perform 3D reconstruction using mobile embedded systems. The developed solution should be capable to generate the 3D model of indoor spaces. We'll be using an infrared (IR) sensor embarked on a vehicle in order to do so.

This technical project is carried out jointly with a research and development project (PRD) aiming to develop a portable solution for a 3D scanner. Both projects will be using the same vehicle. The vehicle we'll develop should therefore be capable to embark a module developed by Mr. Motta Calderon as well.

The vehicle's specifications were established by Mr. Motta Calderon. They fulfil both its client's requirements and ours. Our main tasks are the following:

- Functional definition, conception and realization of the IR 3D scanner module.
- Conception and realization of the vehicle in accordance with Mr. Motta Calderon's specifications

In the two main upcoming sections, we'll develop **embedded system design** and current results for both the vehicle and the IR 3D scan module.

This page intentionally left blank.

2 Vehicle

2.1 Design problem

The vehicle should offer several navigation options which are as follows:

- Manual remote control
- GPS-based autonomous navigation outdoors
- Software override using input from external embarked module (autonomous navigation indoors for 3D reconstruction with 3D IR module, *autonomous car* control module)

Communication should be possible:

- From radio control unit to main navigation board
- Between a computer or a tablet/smartphone and the main navigation board for configuration and telemetry as well as waypoints definition
- Between the main navigation board and embarked modules

The vehicle's power source should be easy to remove and charge from a practical standpoint. As the laser 3D scan module is going to be a lot heavier and bigger than the IR 3D scan module, dimensioning will be done based on its specifications as it is quite obvious that a vehicle able to carry it properly will be able to carry our lighter and smaller IR 3D scan module.

The main software stack used for navigation has to be open enough to allow future implementations and aforementioned communication between the main navigation board and external modules.

2.2 Design solution evaluation

2.2.1 Mechanical hardware choice

The vehicle has to be suitable for both projects it will be used for. The IR 3D scan module is lightweight and small compared to the laser 3D scanner and associated stabilization system it should carry.

From that, dimensioning will be done based on the the requirements of the laser 3D scanner. Several solutions were proposed.

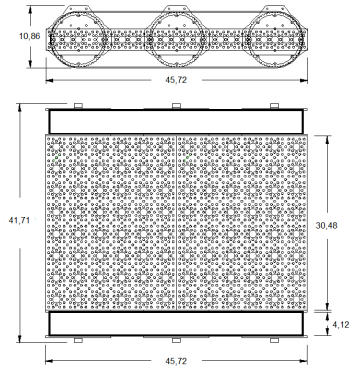

Figure 1: Configuration #1: top view

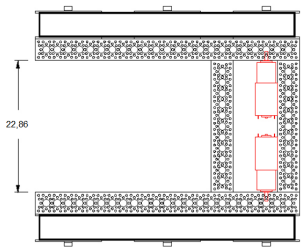

Figure 2: Configuration #1: bottom view

The above figures show configuration #1. As the laser 3D scanner with its stabilization system weights about 20 kg, doubts were expressed regarding the stability of the vehicle on 12 % slopes (maximum defined in specifications). This led to configuration #2, as shown below.

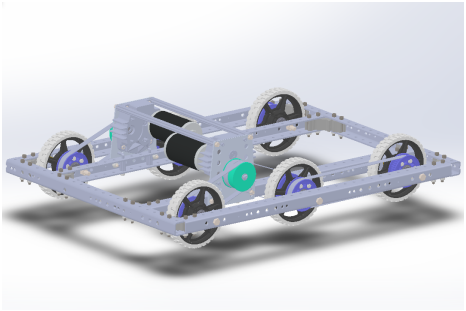

Figure 3: Configuration #2

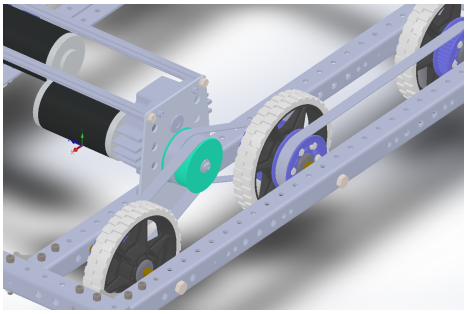

Figure 4: Configuration #2: transmission

This configuration uses a way larger chassis which would have had the following implications:

- Larger, heavier and more expensive motors
- Larger, heavier and more expensive batteries
- Leaded or equivalent less convenient batteries instead easy to charge LiPo batteries
- Reduced maneuverability in indoors environments making it less appropriate for our reconstruction project

Calculations performed by Mr. Motta Calderon and Mr. Benureau under the supervision of Mr. Altmeyer highlighted configuration #1 was appropriate regarding our needs. Detailed characteristics are listed in *mechanical characteristics* section.

2.2.2 Electronic hardware choice

Based on the design problem formulated above, an appropriate solution was proposed. The vehicle electronics will be constituted of the following baseblocks:

- A Rapsberry Pi board associated to a Navio 2 shield running APM/ArduPilot for navigation control.
- Two DC motor controllers for skid-steer drive.
- A 433 MHz radio module for communication with the ground station (PC/tablet) enabling telemetry and waypoints setup.
- A 2.4 GHz radio system for radio control unit communication
- A GPS module (coaxial wiring) for autonomous navigation outdoors

To that adds a 2.4 GHz DX8 G2 radio module and another 433 MHz telemetry module for the ground station.

2.2.3 Radio setup

The Spektrum DX8 G2 can't be directly bound to the Navio2. The radio signal is first received by the bound Spektrum DSXM Satellite which sends it to the Spektrum AR7700 which encodes it into PPM which Navio2 can accept as in input. The 433 MHz UART module on the other hand is directly connected to the Navio2 board.

This leads to the following:

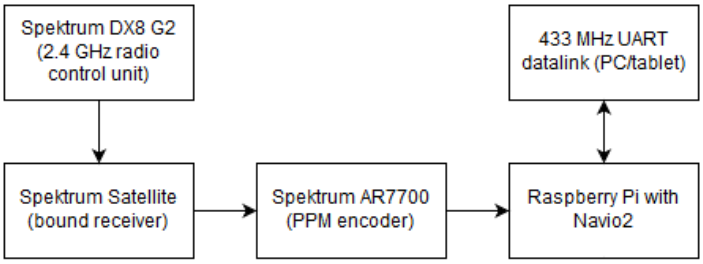

Figure 5: Radio setup overview

2.2.4 Batteries choice

We opted for two lithium polymer (LiPo) batteries each having a capacity of 5.6 Ah. Associating these in parallel leads to a total capacity of 11.2 Ah. In this section, we'll demonstrate why this is an appropriate choice.

We have:

$$T = \frac{11.2}{C} \times 60 \tag{1}$$

Where:

- T is the running time in minutes
- C is the the average consumption in A.

The highest power draw will come from the engines and will largely overpass the power consumption of other components. It is therefore meaningful to neglect other component's consumption and focus on the engines.

The vehicle will use a track system and will therefore be driven by two DC engines. Considered engines have the following specifications:

- Brand and model: Actobotics #638280
- Type: DC planetary gear brush motor
- No-load speed: 313 rpm
- Max. stall current: 20 A

Based on the low amount of meaningful information we have, it is hard to perform precise calculations at this stage. It is however very safe to do a first calculation based on the absolute worst case scenario assuming stall current is reached all the time:

$$T = \frac{11.2}{2 \times 40} \times 60 \simeq 20 \tag{2}$$

In this extreme scenario, the vehicle would have an autonomy of about 20 minutes. Data available for similar engines shows it is quite safe to assume a consumption about 5 A for each engine under load.

This leads to:

$$T = \frac{11.2}{2 \times 5} \times 60 \simeq 70 \tag{3}$$

We would therefore reach an autonomy above one hour which would be satisfactory. Another aspect should be considered: these calculations assume the vehicle is going to run continuously for one hour without stopping. This is indeed not what is going to happen from a practical standpoint which makes it even safer.

If the operating life is proven insufficient, it would be quite easy to associate a third LiPo battery in parallel increasing the autonomy furthermore. Flexibility goals are reached with this setup.

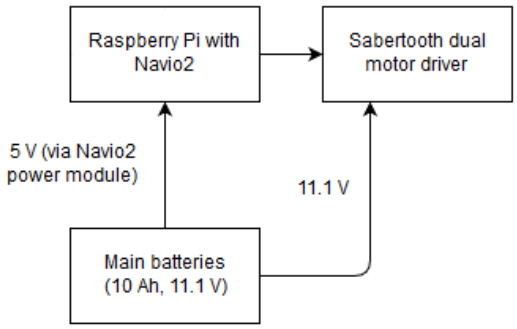

Figure 6: Batteries setup overview

2.2.5 Motor driver choice

It was decided we'd use a Sabertooth Dual 25 A 6 $V - 24$ V Regenerative Motor Driver. This piece of equipment can handle two engines with currents up to 25 A continuously and voltages up to 24 V.

The engines we opted for have the following specifications:

- Brand and model: Actobotics #638280
- Type: DC planetary gear brush motor
- No-load speed: 313 rpm
- Max. stall current: 20 A
- Operating voltage range: 6 $V - 12$ V (DC)

We therefore have:

$$I_{stall} < I_{max} \tag{4}$$

$$V_{upper} < V_{max} \tag{5}$$

Where:

- I_{stall} is the engine maximum stall current
- I_{max} is the maximum rated current chosen motor driver can handle per engine
- V_{upper} is the upper voltage in the engine operating voltage range
- V_{max} is the maximum input voltage chosen motor driver can handle

This motor driver has many built-in safety features such as:

- Overcurrent protection
- Thermal protection

This means accidental peaks won't damage the driver which makes it even more reliable for long term use.

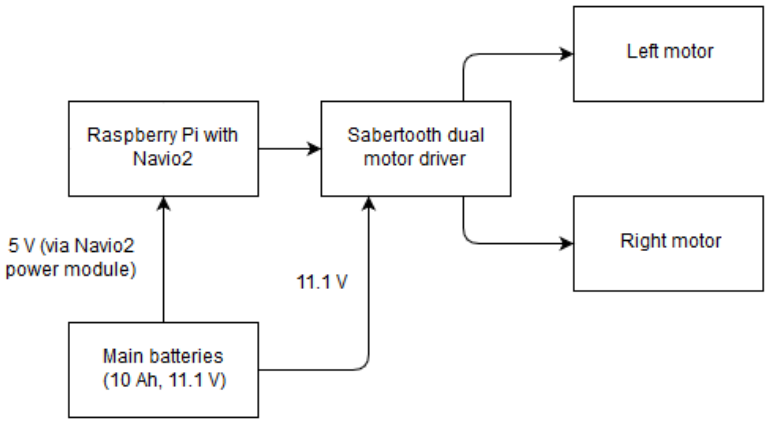

Figure 7: Motors and motor driver setup

The Navio2 board generates a PWM signal for each engine. This signal goes the motor controller via servo wires. The engines are then directly wired to the motor controller.

2.2.6 Mechanical characteristics

The vehicle uses Actobotics standard components. An Actobotics tracked robot parts set known as *Agent 390* was used. This base offers great traction and climbing ability (37°approach angle) and can easily handle heavy loads (over 50 kg).

The parts used in the vehicle chassis are as follows:

- 585462 18” Aluminum Channel
- 585450 9” Aluminum Channel
- 555174 Motor Mount
- 545360 Hub Mount C
- 625102 1/4” Clamp Collars (2pk)
- 535198 1/4” x 1/2” Flanged Ball Bearing (2pk)
- 625100 6mm-1/4” Clamping Coupler
- 634164 1/4” Shaft, 4” long
- 634162 1/4” Shaft, 3” long
- 633140 6-32 x 1.5” Standoff (4pk)
- 633126 6-32 x 0.75” Standoff (4pk)
- 545388 Hub Adaptor
- 615394 60T Hub Pulley
- 545588 1/4” Bore Clamping Hub
- 545532 Channel Attachment Plates (2pk)
- 585006 9” x 12” Channel Plate
- 632106 6-32 x 1/4” Socket Head Cap Screws (25pk)

- 632108 6-32 x 5/16" Socket Head Cap Screws (25pk)
- 632110 6-32 x 3/8" Socket Head Cap Screws (25pk)
- 632112 6-32 x 7/16" Socket Head Cap Screws (25pk)
- 632116 6-32 x 9/16" Socket Head Cap Screws (25pk)
- 632144 6 Standard Washers (25pk)
- 1.5" wide track
- 585580 Triple Channel Brackets
- 633104 1/4" Plastic Spacers (12pk)

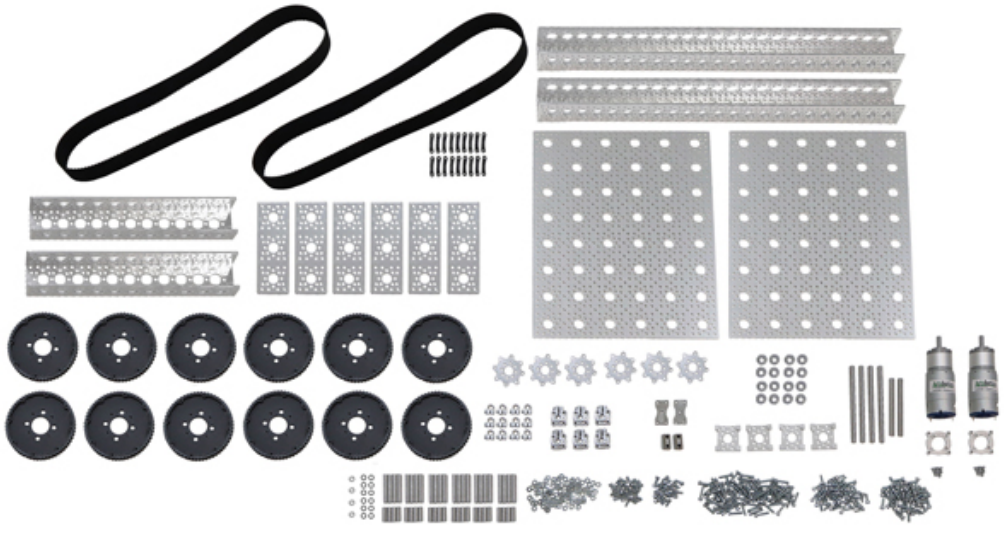

Figure 8: Vehicle chassis parts

Vehicle general dimensions

- Plate dimensions: $12" \times 18"$ / $30,48\ cm \times 45,72\ cm$
- Vehicle dimensions: $16,42" \times 18" \times 4,29"$ / $41,71\ cm \times 45,72\ cm \times 10,90\ cm$
- Mount plate height: $2,65"$ / $6,7\ cm$
- Pulleys diameter (x12) : $3,80"$ / $9,65\ cm$
- Aluminum channels: $1,5" \times 1,5"$ / $3,81\ cm$

Tracks

- Width: $1,5"$ / $3,81\ cm$
- Pitch: $1/5"$ / $0,508\ cm$
- Material: Neoprene with fiberglass filaments

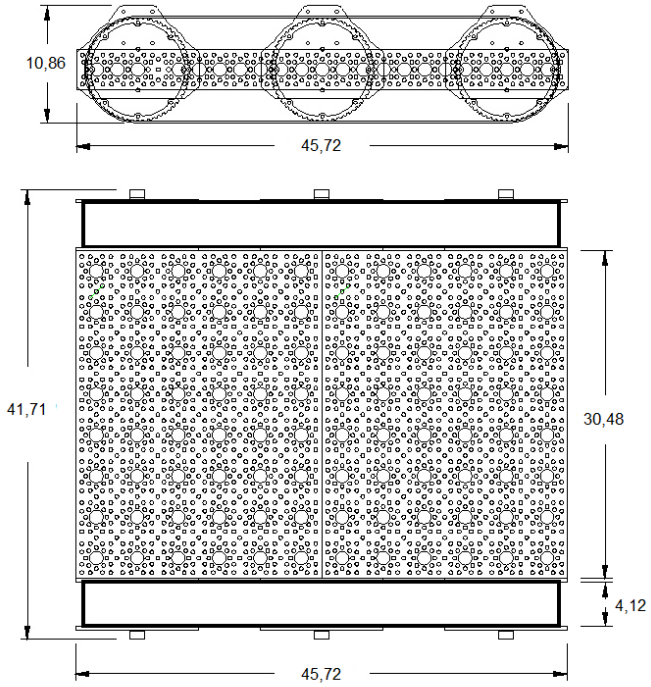

Figure 9: Vehicle chassis overall dimensions (cm)

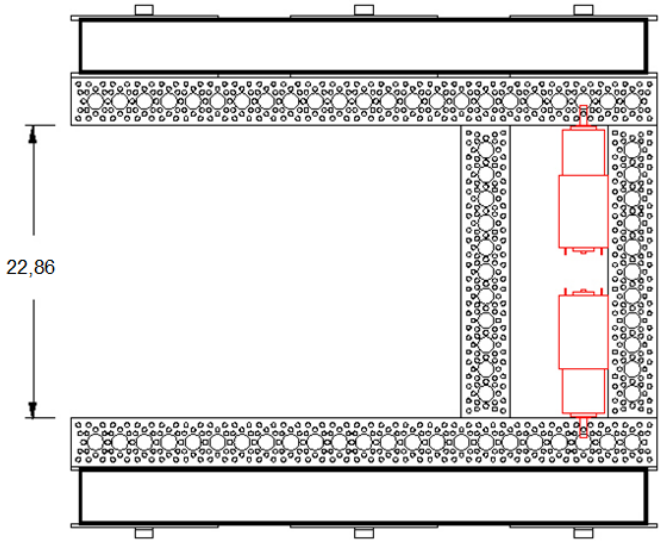

Figure 10: Vehicle chassis with motors dimensions (cm)

2.2.7 Custom mechanical parts

Vehicle electronic components are mounted underneath the plate to leave the space above available for additional modules, in our case, the IR 3D scan system.

The components located under the vehicle include:

- The AR7700 & Spektrum DSMX radio system
- The navigation Raspberry Pi/Navio2 combo

- The Sabertooth Dual 25 A 6 $V - 24$ V Regenerative Motor Driver
- The 433 MHz 3DRobotics telemetry module
- The two DC motors

A box will be designed for each component except for the 433 MHz 3DRobotics telemetry module which already comes with an adequate enclosure. These boxes are made to protect the components and to make them easier to access and recognize. Additionally, a laser cut PMMA plate was designed in order to protect the robot components against possible water splashes.

These boxes and mounts, being all under 20 $cm \times 20$ $cm \times 20$ cm, which is the printing volume of the printer we use, were perfect candidates for 3D printing which is very fast and costs virtually nothing.

The box designed to protect the Raspberry Pi includes holes for the power supply and for the electrical outputs. Boxes mounting holes have to be aligned with the vehicle plate (see figure 15).

A similar box was designed for the AR7700 & Spektrum DSMX combo. The only exception is the motor driver. It wasn't appropriate to put it in a closed PLA box as this component is likely to produce a lot of heat. A mount leaving the heatsinks uncovered was designed (**see annexes for drawings**).

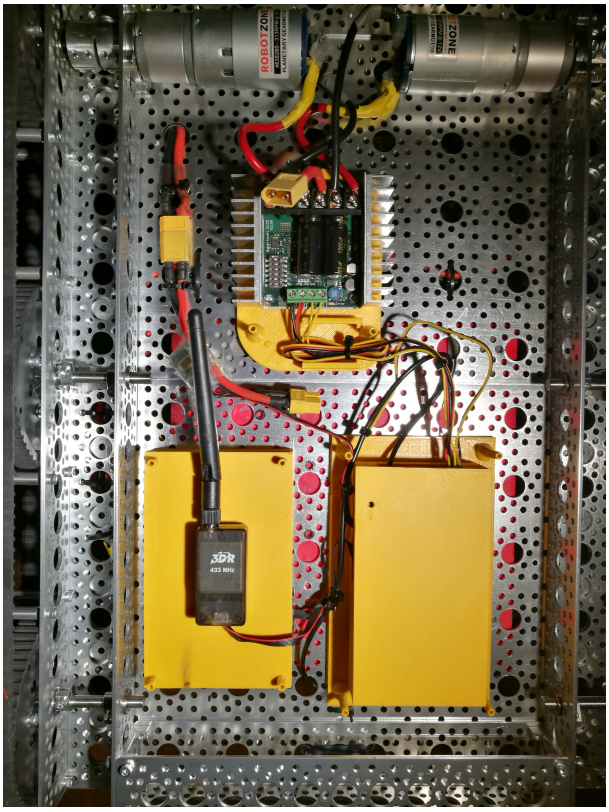

Figure 11: Actual vehicle component boxes implementation

2.2.8 Parking brake system

This feature was developed exclusively for the laser 3D scanner module. The specifications require the vehicle to be physically locked while acquiring data. A first solution we discussed was to use Nema 23 stepper motors with a belt system to lock the pulleys. This was eventually dropped in favor of a phyisical lock system using sevormotors for the following reasons:

- It would consume too much power when idle.

- The motors would be too big to fit *inside* the chassis and would be attached on the exterior increasing collision risk and making it unsightly.

The second iteration led to the final solution that was implemented. In this design, a servomotor is used as an actuator to physically push a 3D printed lock thanks to a 3D printed wheel.

In order for the lock to be properly inserted, the vehicle has to move foarward *very slowly* while deploying it.

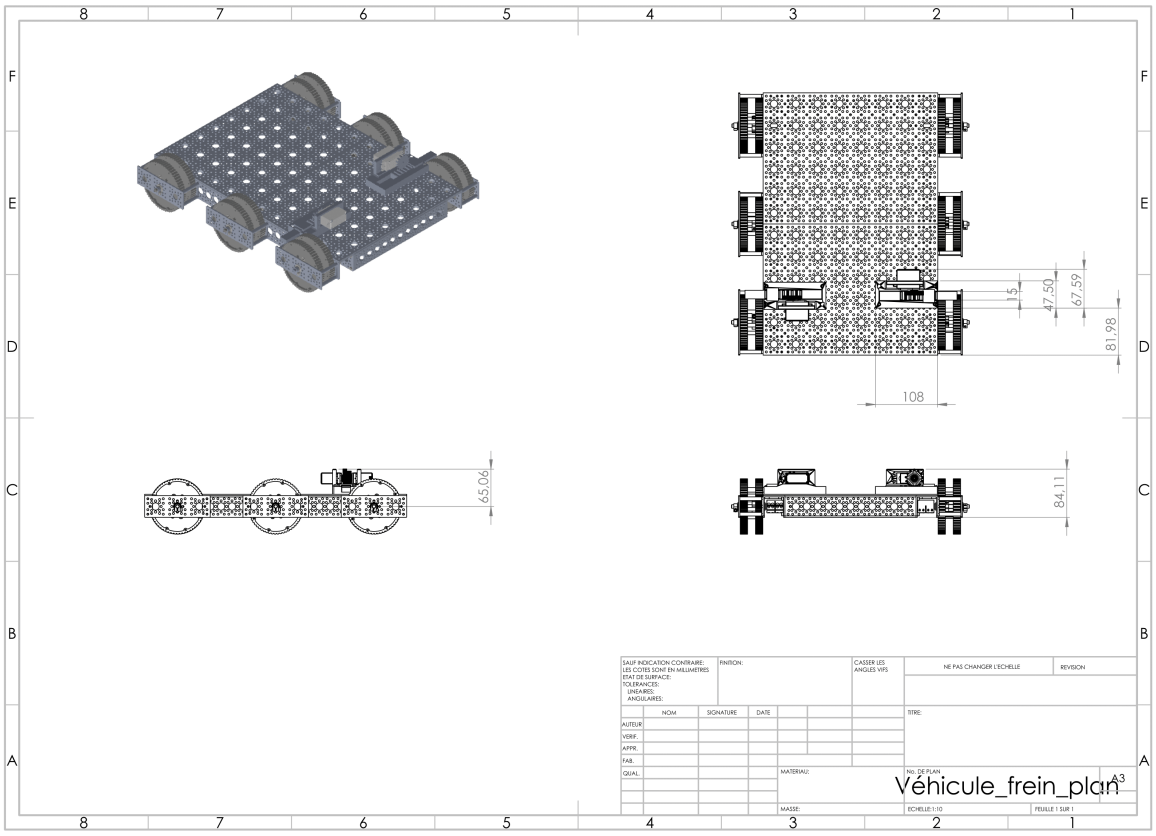

Figure 12: Parking brake system assembly

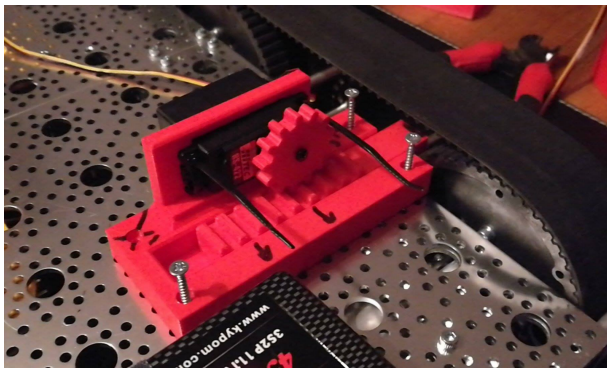

Figure 13: Parking brake system implementation

2.3 Final design overview

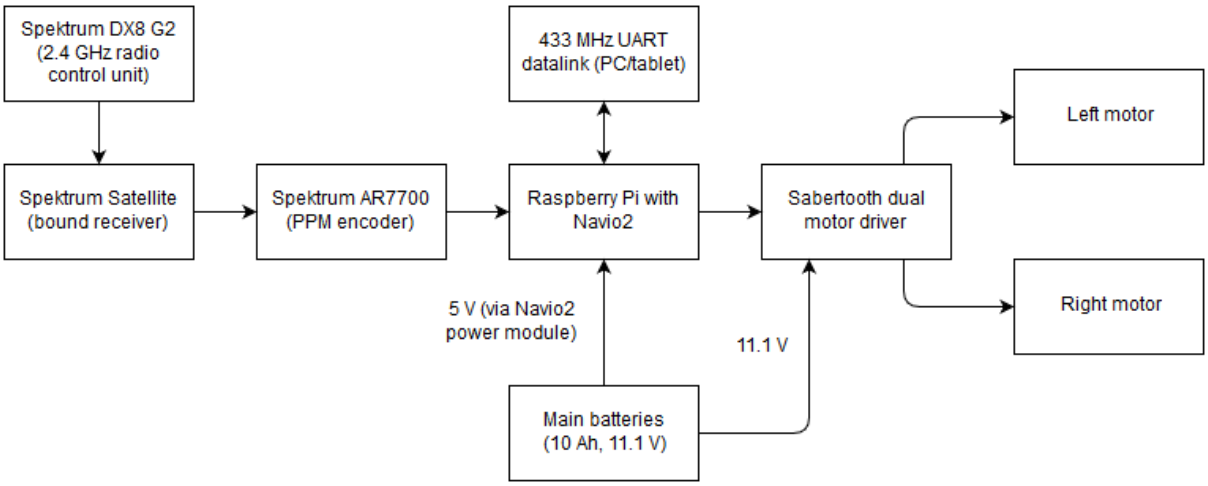

Figure 14: Vehicle electronics hardware setup

In this design, the Navio2 board powers the Raspberry Pi as well as other attached modules thanks to the 5 V power regulator which also measures battery voltage and current.

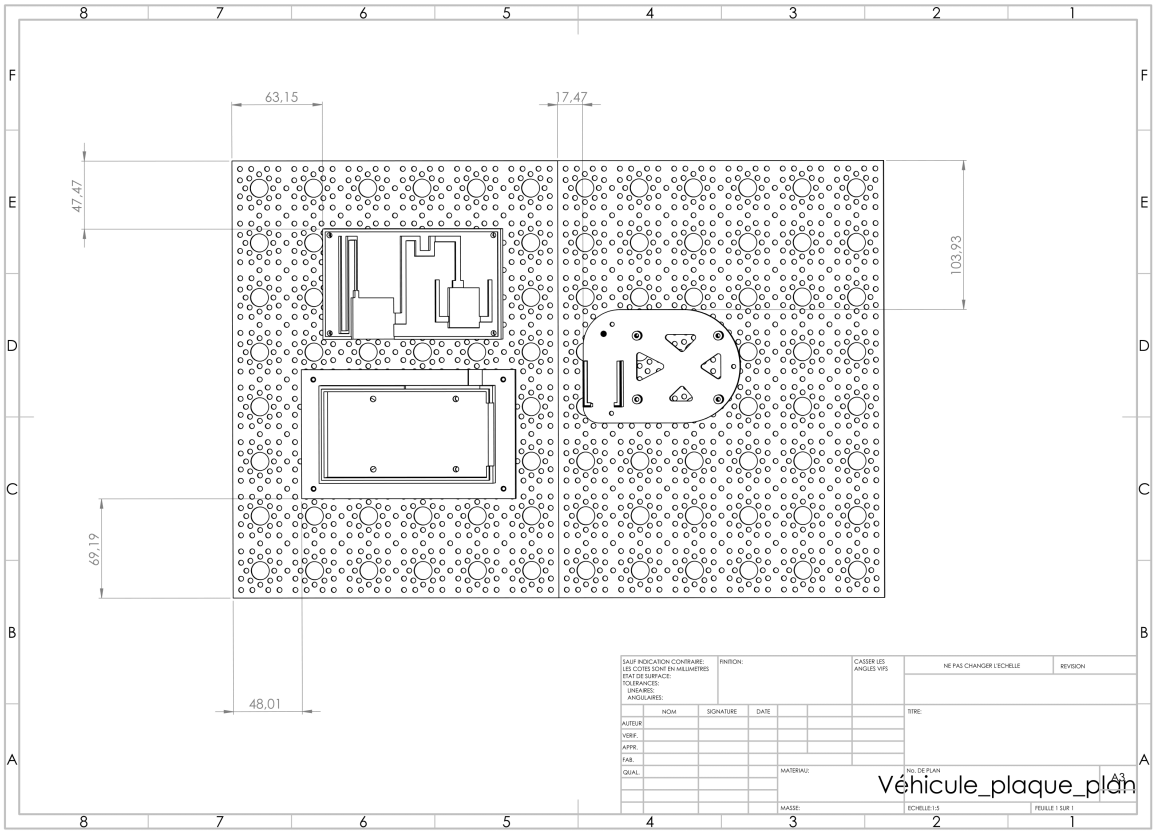

Figure 15: Vehicle component boxes assembly

Figure 16: Vehicle in operation

2.4 Instructions

2.4.1 ArduPilot installation

1. Flash Emlid Raspbian image as detailed here

2. Set Wi-Fi by editing the /boot/wpa_supplicant.conf file located on SD card. To add your network simply add the following lines to it [4]:

```
network={
  ssid="yourssid"
  psk="yourpasskey"
  key_mgmt=WPA-PSK
}
```

3. Download ArduRover

```
wget http://firmware.eu.ardupilot.org/Rover/stable/navio2/ardurover
```

4. Set execution permissions

```
chmod +x arducopter-quad
```

5. To automatically start ArduPilot on boot add the following (change -A and -C options to suit your setup) to /etc/rc.local file on your Raspberry Pi:

```
sudo nohup /home/pi/ardurover -A udp:192.168.1.2:14550 -C /dev/ttyAMA0 > /home/pi/startup_log &
```

2.4.2 Running ArduPilot

Run it as:

```
sudo ArduRover -A udp:ip:14550
```

Where "ip" is the IP address of the Ground Control Station device [4].

2.4.3 ArduPilot firmware configuration

1. Start Mission Planner and connect via UDP or telemetry module
2. Go to Config/Tuning > Planner and set 'Layout' to 'Advanced'
3. Go to 'Full Parameter Tree' and search for 'RCMAP'. In RCMAP, set map entries as in the RCMAP table below. Click 'Write Params' to save your configuration.
4. Use the bind plug to bind the AR7700 Rx with the DX8 radio (see Spektrum instructions).
5. Go to Initial Setup > Mandatory Hardware > Radio Calibration.
   Click on 'Calibrate Radio'. Follow the instructions. Click one 'Done' once done.
   Note: make sure to actually put the throttle in middle position (50% on Tx) before clicking 'OK' when prompted.
6. Go to Initial Setup > Optional Hardware > Battery Monitor and set entries as in the below table. See this for more information.
7. Set throttle failsafe by doing as detailed here
   (a) Turn on the radio and go into servo setup.
   (b) Find the throttle travel setup (on the DX8 it's the first screen)
   (c) Scroll down to negative travel (Left hand 100%). Change that to as high as it'll go. 140
   (d) Confirm in Mission Planner that throttle input is now around 920uS
   (e) Turn off radio and receiver.
   (f) Insert Bind plug into RX and turn on.
   (g) Remove bind plug from rx. Light should continue blinking.
   (h) Ensure throttle stick is down.
   (i) Turn on radio while holding down bind button. Radio should rebind.
   (j) Turn everything off.
   (k) Turn radio back on and change the throttle throw back to 100%
   (l) Turn on quad and connection with MP
   (m) Confirm that minimum throttle is back to around 1100.
   (n) Turn off radio and throttle should now drop to around 920uS (or whatever it was beforehand)
   (o) If not, start again.
   (p) Now go to parameters in MP
   (q) Find THR_FAILSAFE and set it to 1.
   (r) Find THR_FS_ACTION and set it to 2 (1 = continue on Auto mission, 2 is RTL)
   (s) Click on the write params.
   (t) Click on Refresh to ensure it wrote it.

   From now one, failsafe should trigger only when Tx is off.
8. Set battery arming checks in general configuration

9. As a $3S$ 11.1 V battery is being used, set minimum failsafe voltage to 10.5 V
10. Set throttle warning on Tx powerup if not already enabled

Table 1: RCMAP parameters

Parameter	**Value**
RCMAP_PITCH	2
RCMAP_ROLL	1
RCMAP_THROTTLE	3
RCMAP_YAW	4

2.4.4 Spektrum radio configuration

1. Turn on the DX8 radio while pressing the select button to enter System Setup
2. Go to *Rx Port Assignments*
3. Map as in the table below
4. Select 'NEXT' to show *Channel Input Config*
5. Map as in the table below

Table 2: Rx port assignment

Port	**Assignment**
THRO	Throttle
AILE	Aileron
ELEV	Elevator
RUDD	Rudder
GEAR	Gear
AUX1	Aux1
AUX2	Aux2
AUX3	Aux3

Table 3: Channel input config

Channel	**Input**
THRO	N/A
AILE	N/A
ELEV	N/A
RUDD	N/A
GEAR	RKnb
AUX1	A
AUX2	B
AUX3	C

This page intentionally left blank.

3 IR 3D scan module

3.1 Design problem

The IR 3D scan module's primary goal is to log RGB and depth data which is to be used for 3D reconstruction. It should be able to reconstruct indoor environments and generate a proper 3D model.

The RGB-D camera should be mounted on top of the vehicle and its orientation and tilt angle should be controllable.

3.2 Design solution evaluation

3.2.1 3D scan software solution

ElasticFusion is, according to the words of Thomas Whelan, its creator, a real-time dense visual SLAM (Simultaneous Localization And Mapping) system capable of capturing comprehensive dense globally consistent surfel-based maps of room scale environments explored using an RGB-D camera. This happens to be exactly what we need.

The capture and the reconstruction processes can be separated thanks to dedicated loggers (see 3.4.3 for more details) which collect raw data that is packed into a KLG file. This file can later be processed by ElasticFusion in order to generate a 3D model.

3.2.2 3D scan hardware solution

RGB-D cameras are cheap and widely available. We're currently using a Kinect for Xbox 360 sensor which does the job correctly even though we're experiencing data link issues that might be hardware related which may push us to opt for a different RGB-D camera such as the Primesense Carmine 1.09 which is newer and widely supported as well.

The Kinect for Xbox 360 features an embedded motor allowing to tilt the camera up and down to a maximum of 45°. As we don't really need that motor, this is another argument in favor of the Primesense Carmine 1.09 which, unlike Kinect, doesn't require an additional 12 V power source.

3D reconstruction is resource expensive and requires a very powerful GPU/CPU combo. The hardware recommendations for the software stack we opted for are as follows:

- A very fast Nvidia GPU (at least 900 series - 3.5 $TFLOPS+$)
- A fast CPU - at least something like an Intel Core i7

This makes it impossible to run 3D reconstruction on cheap embedded hardware like a Raspberry Pi board. The data will therefore be logged on the vehicle and later transferred to a remote computer that is to perform resource expensive 3D reconstruction.

3.2.3 Remote computation station for RGB-D data processing

As discussed above, the logging part should be separated from the reconstruction part. A very powerful dedicated computer will be used. Its computation hardware is as follows:

- Nvidia GTX 1080 GPU (8192 MB)
- Intel Core i7 6800K CPU (6 cores, 3.40 GHz)

In the final version of the IR 3D scan module, it should be made so that the module would automatically transfer data to the remote computer via a wireless connection once logging is done. Said computer would then automatically start reconstruction once data is transferred.

Basic operations would follow this order:

1. Start logging process via user request
2. Manual or autonomous navigation and RGB and depth data collection
3. KLG log transfer to remove computation station
4. KLG log treatment and MeshLab ready 3D model generation

3.2.4 Post-reconstruction model treatment

ElasticFusion exports MeshLab compatible data. MeshLab cleans and optimizes meshes and can export reconstructed 3D models to more common formats like STL or 3DS.

An optimized model in this format can then be imported into 3D modeling software such as 3ds Max and manually modified to be, for example, suitable for 3D printing.

3.2.5 RGB-D camera mount hardware solution

A turret with modular height was designed. It offers two degrees of freedom: base rotation and tilt adjustment. Movement is enabled thanks to two identical servomotors. On at the top of the turret and one at the bottom.

Raspberry Pi boards offer a very limited GPIO and can't command more than one servomotor at once. For that reason, it was decided a slave Arduino Board will be used in association with a dedicated Raspberry Pi unit used as the module's main board.

This leads to the following setup:

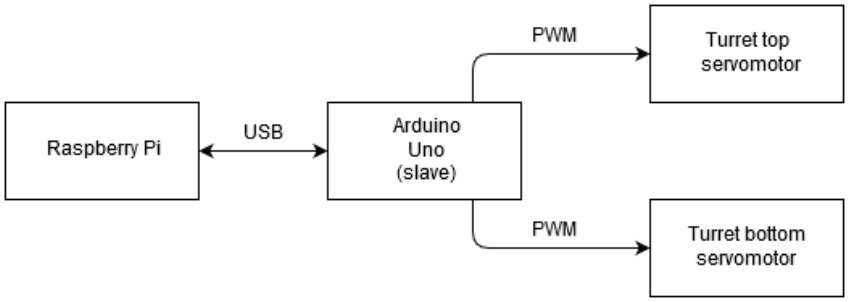

Figure 17: IR 3D scan module electronics setup

3.2.6 RGB-D camera mount software solution

The turret will be controlled either by the user via remote control or autonomously.

Manual control can be done:

- Using a knob on the Spektrum DX8 G2 radio unit for rotation
- Using a 3-positon switch on the Sprektrum DX8 G2 radio unit for camera tilt presets

We'll be using nanpy, a Python API allowing to use an Arduino board as a slave controlled from a master device (a Raspberry Pi board in our case) where scripts are run.

3.2.7 Automation

The first milestone to reach is a functional reconstruction setup based on manual user remote navigation control. Once done, it is envisioned to make the vehicle as autonomous as possible while scanning indoor environments. IR sensors as well as 2D camera modules will be used in order to achieve this.

As the RGB-D camera is mounted on top of 2DOF rotating turret, the vehicle would be able to automatically change the turret's orientation in order to cover all of its surroundings.

3.2.8 Power setup

The IR 3D scan module's power system will be separated from the vehicle's. It'll use its own independent battery and will rely on three adjustable step-down voltage regulator. The model we opted for offers a $1.25\ V - 35\ V$ range and a $3\ A$ maximum.

The main aspect we took into consideration is the maximum current draw for each device in our setup:

- The Kinect RGB-D camera draws up to $2\ A$ at $12\ V$ (DC).
- The Raspberry Pi board draws up to $2.5\ A$ at $5\ V$ (DC). The Arduino Uno board is powered through the Rapberry Pi via USB
- Both servomotors combined will never exceed $3\ A$ at $6\ V$ (DC).

From that, it was decided three regulators will be used:

- One regulator for the Kinect RGB-D camera
- One regulator for the Raspberry Pi board
- One regulator for both servomotors (at the top of the turret)

This leads to the following setup:

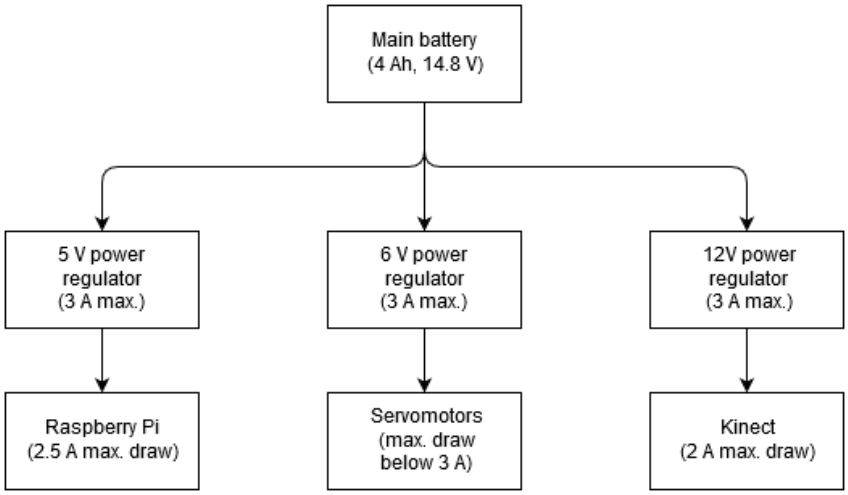

Figure 18: IR 3D scan module power setup

3.2.9 Embedded control system (ECS) software solution

Embedded control system (*ECS*) software handles interaction between the different boards and controls the overall behavior of the vehicle.

The master Raspberry Pi board, known as *master board* sends data to and receives data from other boards, including:

- Arduino Uno board for turret servo control using Nanpy - USB link
- Navigation Raspberry Pi/Navio2 board running ArduRover using custom server/client library - Ethernet link
- Logger/Minnowboard running custom Logger1 for RGB-D logging - Ethernet link

This led to the following architecture:

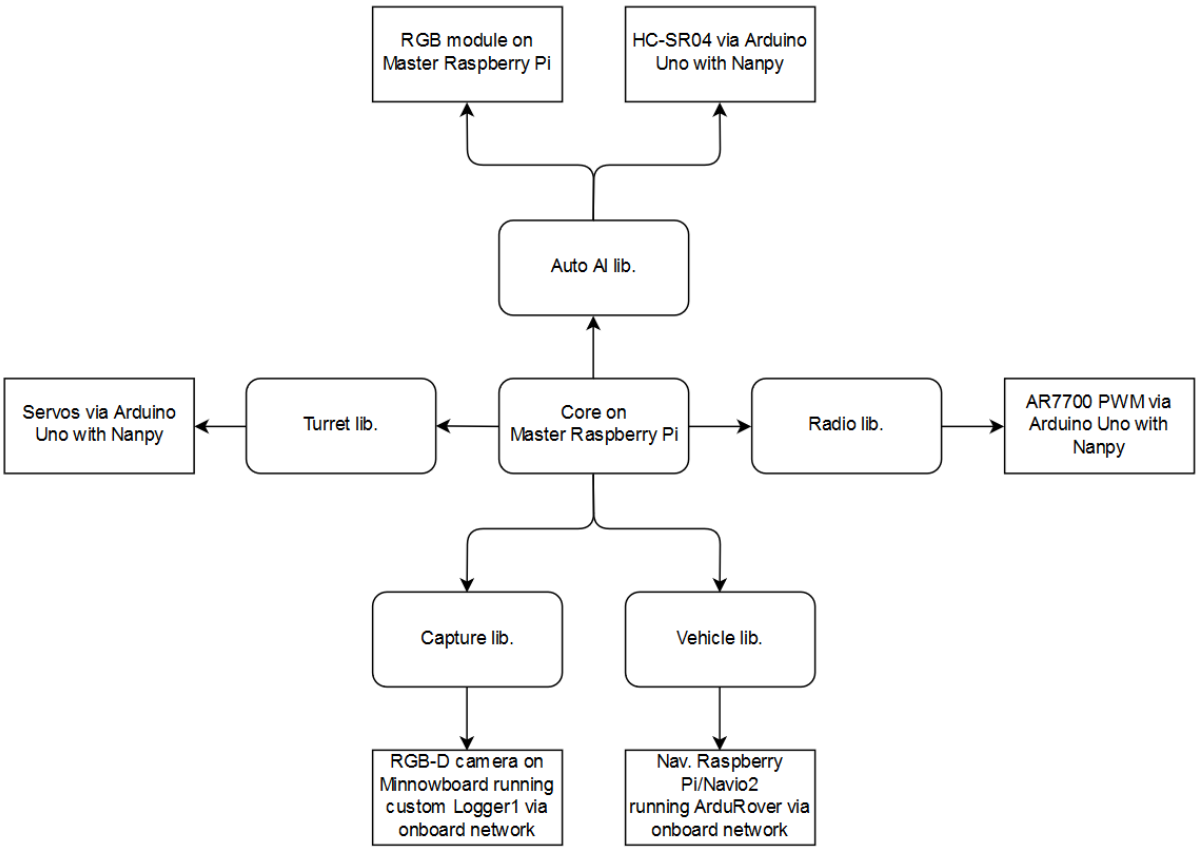

Figure 19: Initial ECS software architecture solution

In this first design, RC PWM data is read by the master board from the AR7700 via Arduino Uno using Nanpy. This was eventually dropped in favor of a server/client approach that would also be used for vehicle control in autonomous mode where servo data is sent from the master board to the navigation board.

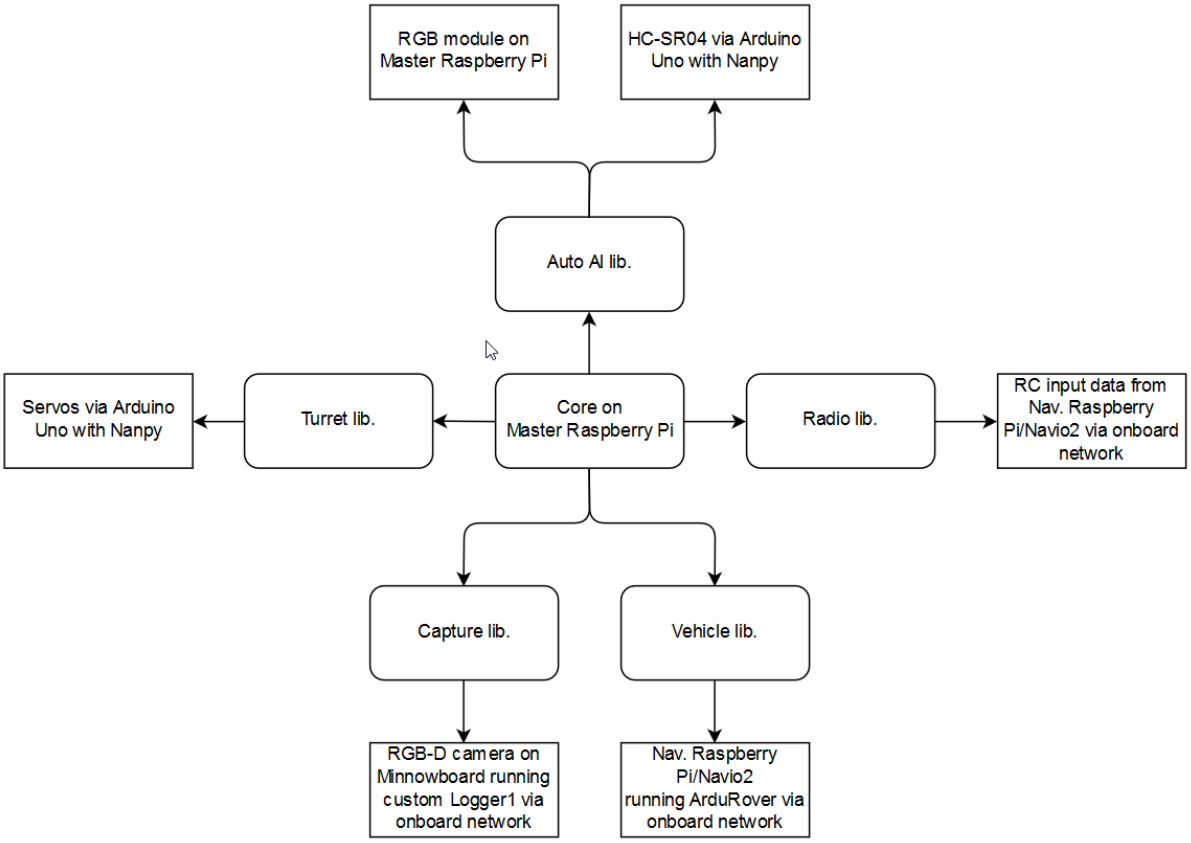

Figure 20: Final ECS software architecture solution

In the server/client approach, the navigation board client sends RC input data to the master board server via TCP/IP which uses this data to control the turret in manual mode. These packets consist in a stream of integers.

ECS software is invoked through remote GCS via SSH. On startup, the user has to choose between two capture modes: manual or autonomous. Once done, the user should specify how long capture will last so the logger board can be configured.

In manual mode, this system is used to retrieve 3 position switch RC input data used to control turret orientation (left, right, center). Navigation is done via RC control. In autonomous mode on the other hand, the vehicle navigates using US sensors and swipes the room in order to get as many points as possible. It then stops when capture duration is reached.

This behavior can be summarized as follows:

Manual mode

1. Write capture duration on logger board
2. Start server on master board
3. Start client on navigation board
4. Write servo position based on client data while running

Autonomous mode

1. Write capture duration on logger board

2. Start server on navigation board
3. Start client on master board
4. Initialize US sensors (via Nanpy)
5. Send PWM data from master to navigation board to control vehicle and turret behavior while running

3.2.10 Modified Logger1

Logger1, a C++ program written by Thomas Whelan, handles RGB-D frames capture from Primesense Carmine and other RGB-D cameras.

According to its creator, it grabs RGB and depth frames which are then compressed (lossless ZLIB on depth and JPEG on RGB) and written to disk in a custom binary format. Multiple threads are used for the frame grabbing, compression and GUI. A circular buffer is used to help mitigate synchronisation issues that may occur.

It uses the following open source libraries:

- CMake
- Boost
- Qt4
- OpenNI
- ZLIB
- OpenCV.

The original Logger1 app is designed to run with a GUI as sole user input. This was not suitable for what was needed. We needed a console app we can run via SSH from the master board with launch arguments. These arguments include:

- Capture duration in seconds
- KLG file destination path

To that adds the ability to stop capture before duration delay is reached through keyboard event.

This is exactly what we did. Our custom Logger1 app should be started like this:

```
./Logger --duration delay_in_seconds --destination '/absolute/path/to/file.klg'
```

Any keyboard event during execution leads to capture termination.

This custom branch is available here on GitHub: https://github.com/QBitor/Logger1_CLI

3.2.11 Mechanical parts

The turret is made of five 3D printed parts including:

- The turret bottom mount (see figure 33)
- At least one turret structural cylinder (adjustable height) (see figure 34)
- Three turret top parts including two Hitec HS-422 servo slots (see figures 35, 36, 37)

Electronic boards as well as buck converters are mounted on a dedicated 3D printed plate including holes for M3 screws. This plate also includs space for the turret base. See figure 32.

3.3 Final design overview

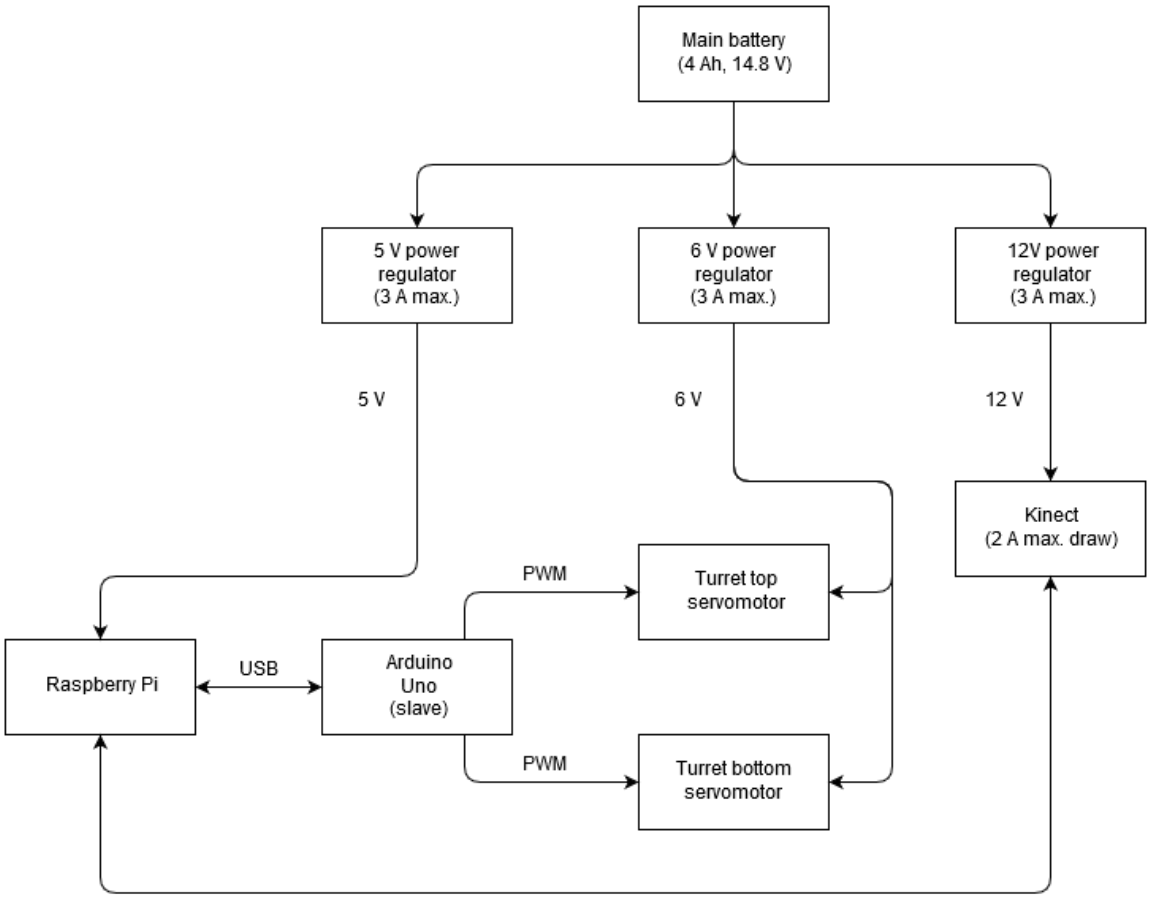

Figure 21: IR 3D scan module hardware setup

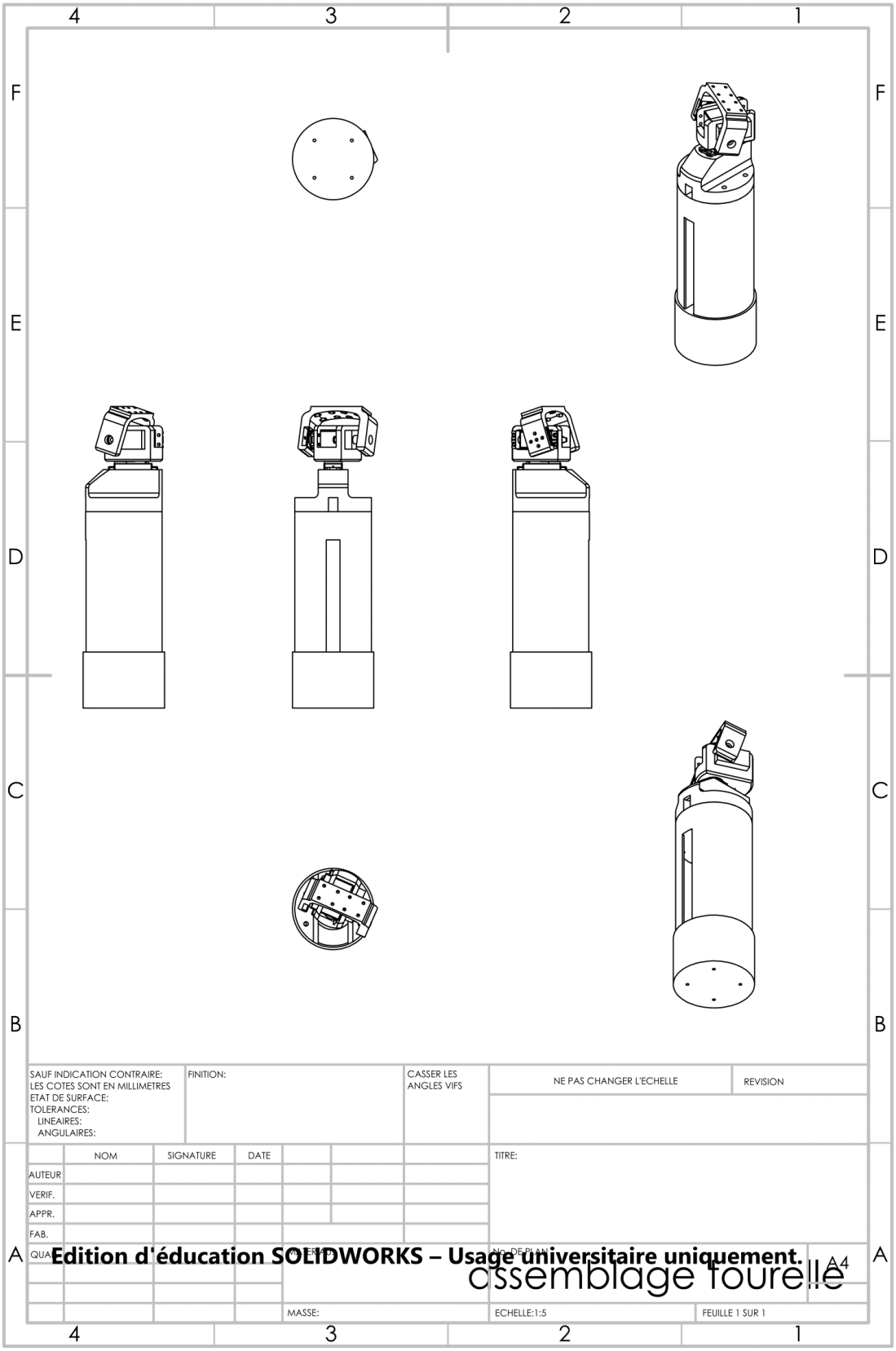

Figure 22: IR 3D scan module turret

3.4 Instructions

3.4.1 ElasticFusion build

ElasticFusion can be built through our installation script.

Requirements are:

- Ubuntu 14.04
- A very fast Nvidia GPU (at least 900 series - 3.5 $TFLOPS+$)
- A fast CPU - at least something like an Intel Core i7

Download it here: https://goo.gl/BBbHOL. Once done, place it where you want and run:

```
sudo ./install_elasticfusion.sh
```

It will download and install the following dependencies:

- CMake
- OpenGL
- CUDA >= 7.0
- OpenNI2
- SuiteSparse
- Eigen
- zlib
- Pangolin

It will build the following:

- ElasticFusion Core
- ElasticFusion GUI
- ElasticFusion GPUTest

It will also install Nvidia drivers for GPUs wihtin the requirements. After the installation, restart your system. If you can't start your system anymore, do the following:

1. Select one of the recovery entries in GRUB (under advanced)
2. Select root prompt
3. Remount filesystem with read/write access: mount -o remount,rw /

Once you have root terminal access, do:

```
apt-get purge nvidia-*
reboot
```

After that, select your normal Ubuntu entry in GRUB. You should be able to login in low resolution. Once done, reinstall the drivers by running:

```
sudo apt-get update
sudo apt install nvidia-367
```

3.4.2 ElasticFusion use

Get a dataset in KLG format, go to GUI/build and run:

```
./ElasticFusion -l /path/to/dataset.klg
```

You should see something similar to this:

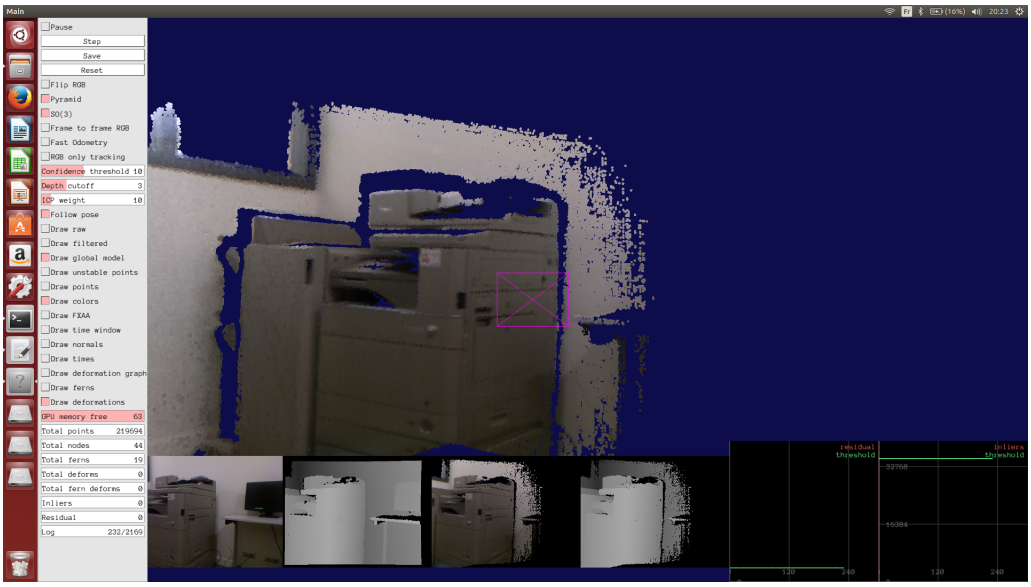

Figure 23: ElasticFusion running with GUI

Additional arguments include:

- -cal : Loads a camera calibration file specified as fx fy cx cy.
- -l : Processes the specified .klg log file.
- -p : Loads ground truth poses to use instead of estimated pose.
- -c : Surfel confidence threshold (default 10).
- -d : Cutoff distance for depth processing (default 3m).
- -i : Relative ICP/RGB tracking weight (default 10).
- -ie : Local loop closure residual threshold (default 5e-05).
- -ic : Local loop closure inlier threshold (default 35000).
- -cv : Local loop closure covariance threshold (default 1e-05).
- -pt : Global loop closure photometric threshold (default 115).
- -ft : Fern encoding threshold (default 0.3095).
- -t : Time window length (default 200).
- -s : Frames to skip at start of log.
- -e : Cut off frame of log.
- -f : Flip RGB/BGR.

- -icl : Enable this if using the ICL-NUIM dataset (flips normals to account for negative focal length on that data).
- -o : Open loop mode.
- -rl : Enable relocalisation.
- -fs : Frame skip if processing a log to simulate real-time.
- -q : Quit when finished a log.
- -fo : Fast odometry (single level pyramid).
- -nso : Disables SO(3) pre-alignment in tracking.
- -r : Rewind and loop log forever.
- -ftf : Do frame-to-frame RGB tracking.
- -sc : Showcase mode (minimal GUI).

Click 'Save' to save a PLY (*.klg.ply*) 3D model file in the same folder as the KLG file. This PLY file can later be imported in MeshLab which is available here.

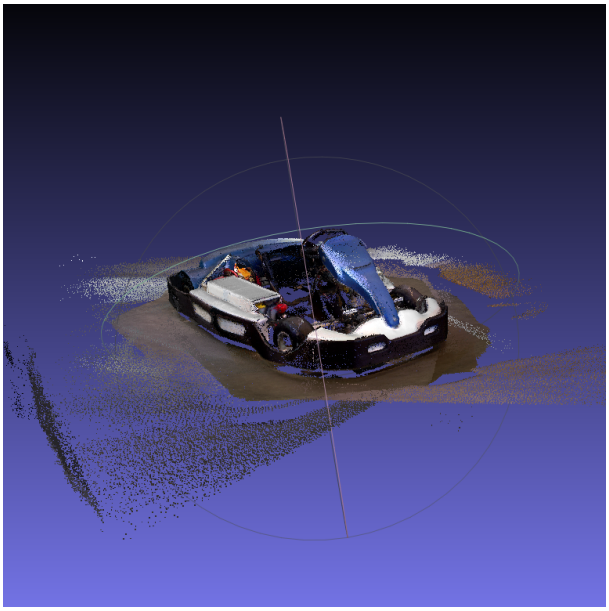

Figure 24: ECAM electric kart reconstructed point cloud opened in MeshLab

3.4.3 Logger1 installation

Loggers are used for logging RGB-D data. There are two possible options: Logger1 and Logger2. Both grab RGB and depth frames which are then compressed (lossless ZLIB on depth and JPEG on RGB) and written to disk in a custom binary format. Multiple threads are used for the frame grabbing, compression and GUI [2].

The Kinect RGB-D camera can only be used with Logger1. The Primesense Carmine 1.09 on the other hand is supported by Logger2. The main difference is that Logger1 uses OpenNI1 while Logger2 uses OpenNI2 which should offer better results.

Logger1 can be installed on a 64-bit (x64) computer running Ubuntu 14.04 thanks to our installation script. Download it here: https://goo.gl/BBbHOL. Once done, place it where you want and run:

```
sudo ./install_logger1.sh
```

It should download all dependencies and build everything.

3.4.4 Logger1 use

To use Logger1 go to the build folder and do:

```
./Logger
```

You should see something similar to this:

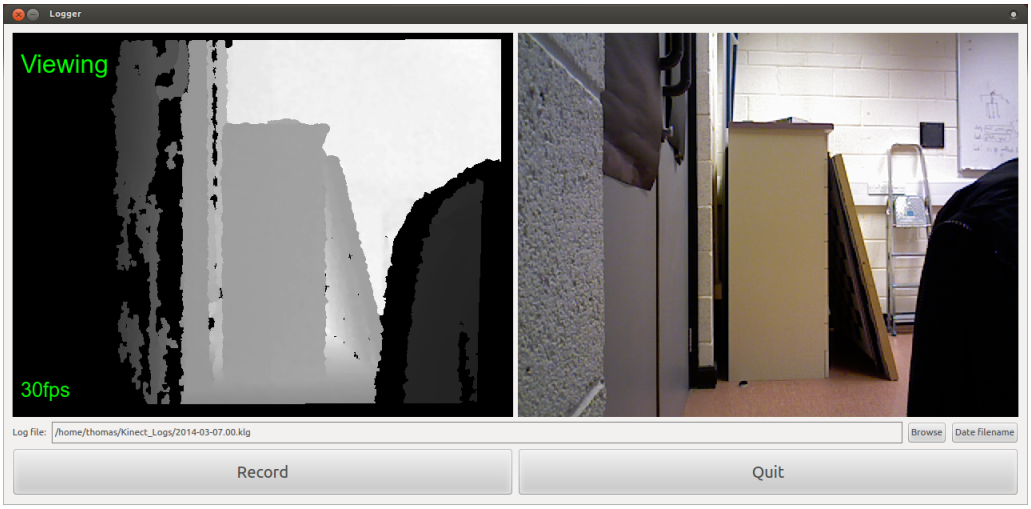

Figure 25: ElasticFusion running with GUI

3.4.5 ECS software use

Connect to to "ScanBot" Wi-Fi network. Start any SSH client and connect to the master board (192.168.0.1). Once connected, do:

```
sudo python ./ScanBot_ECS/core.py
```

This will start the main application. From there, follow onscreen instructions.

For reference, ScanBot_ECS is located into /home/pi/

4 Conclusion

This project was something we never did on such a large scale before and allowed us to put many skills into practice. After having performed a state of the art regarding 3D reconstruction, we chose an appropriate technical solution regarding the specifications our client requested. We gave priority to long term reliability and precision by using some of the most reliable systems.

Once initial design was done, the vehicle parts were ordered and arrived as expected. From there, assembly and electronics setup was conducted properly. The IR 3D scan module was then installed onto the vehicle and manual turret control as well as post-scan wireless data transmission to the ground station were implemented successfully.

At this point, our first goal was met: we ended up with a working manually controlled rover capable or logging and transmitting data for 3D reconstruction. From there, the automation subproject began. The rover was automatized for indoors automated 3D reconstruction as much as possible.

Now that a strong technical base is established, this project could lead to a new one aiming to automatize it furthermore. Improvements could include developing more advanced interactions relying, for example, on computer vision (Raspberry Pi with camera module running OpenCV for example). This could allow the robot to recognize patterns or even people and react according to it [5].

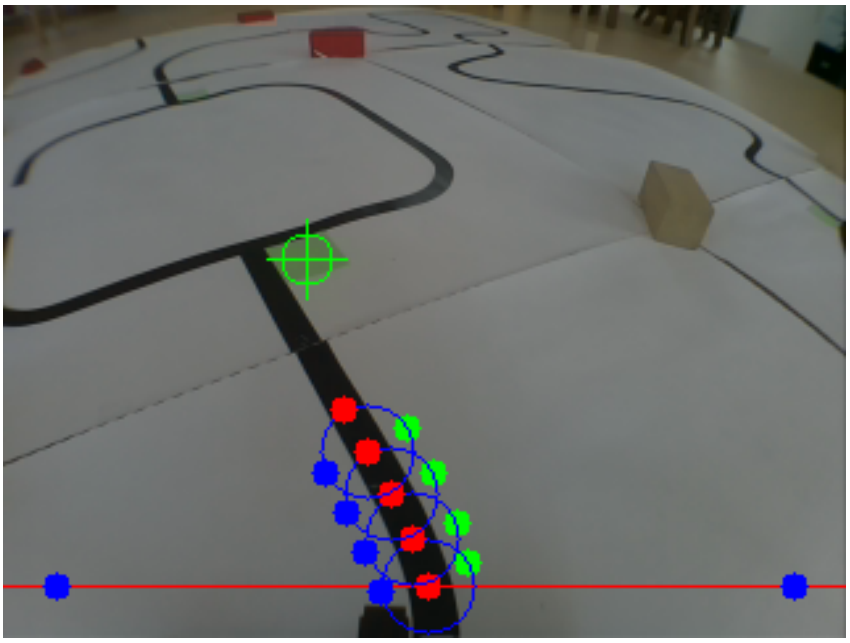

Figure 26: OpenCV example

This page intentionally left blank.

References

[1] Github. *ElasticFusion*. [Online]. https://github.com/mp3guy/ElasticFusion.

[2] Github. *Logger1*. [Online]. https://github.com/mp3guy/Logger1.

[3] Github. *nanpy*. [Online]. https://github.com/nanpy/nanpy.

[4] Emlid. *Navio2 docs*. [Online]. https://docs.emlid.com/navio2/.

[5] Raspberry Pi. *An image-processing robot for RoboCup Junior*. [Online]. https://www.raspberrypi.org/blog/an-image-processing-robot-for-robocup-junior/.

This page intentionally left blank.

Annexes

Orders

Orders for both the IR 3D scan module and the vehicle are referenced here: http://tinyurl.com/gryrt7c

User accounts summary

Each board running a Linux distribution has a specific user account. Usernames and passwords as well as board names are listed here.

Interaction between ECS and GCS is done by connecting GCS to ECS master board access point.

- **Master board**
  - Name: master
  - Username: pi
  - Password: aqw741zsx
- **Logger**
  - Name: minnowboard
  - Username: logger
  - Password: aqw742zsx
- **Navio**
  - Name: navio
  - Username: pi
  - Password: aqw743zsx
- **Router**
  - Username: admin
  - Password: aqw740zsx

Vehicle custom parts drawings

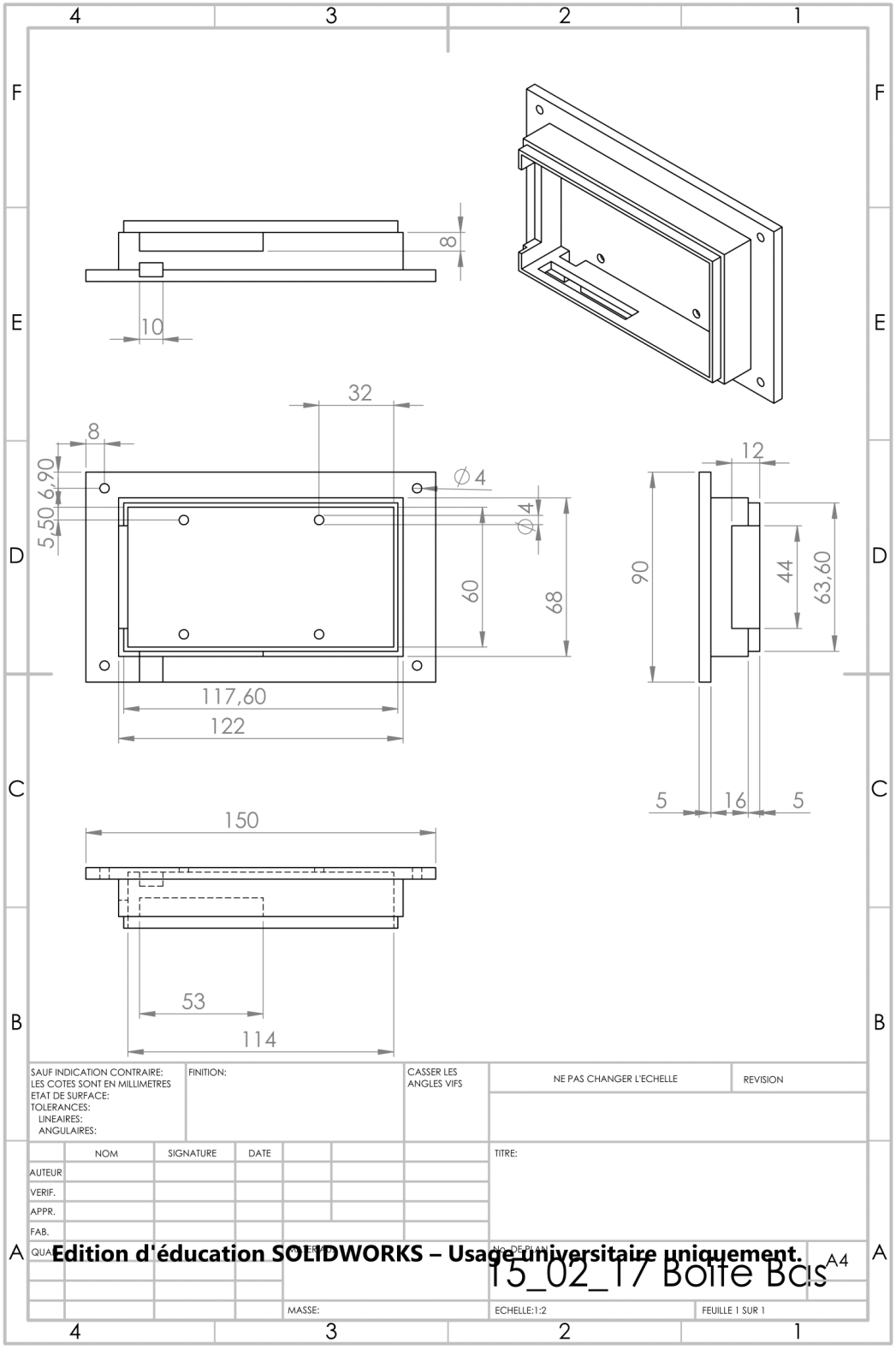

Figure 27: Nav. Raspberry Pi/Navio2 box

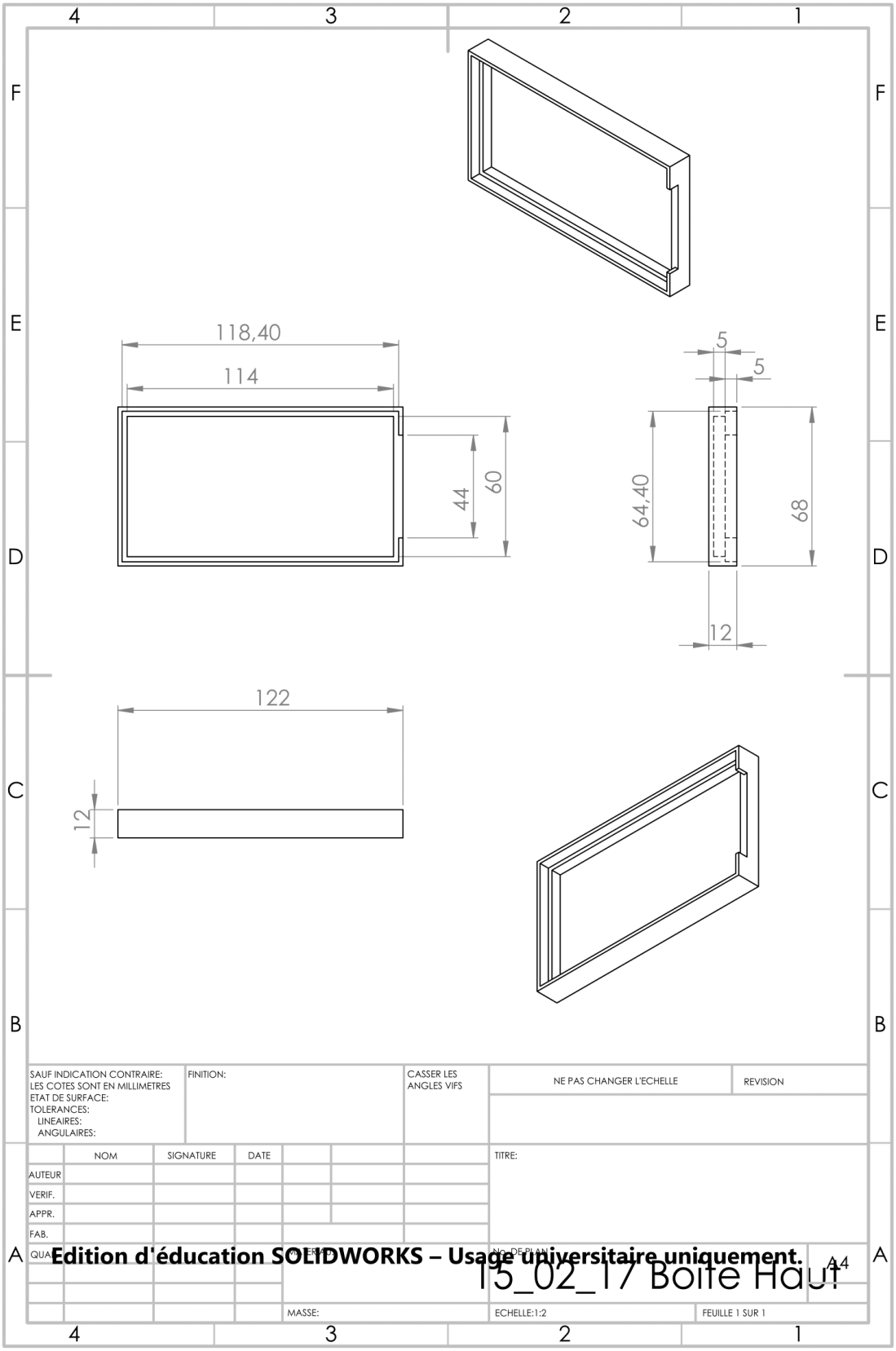

Figure 28: Nav. Raspberry Pi/Navio2 box lid

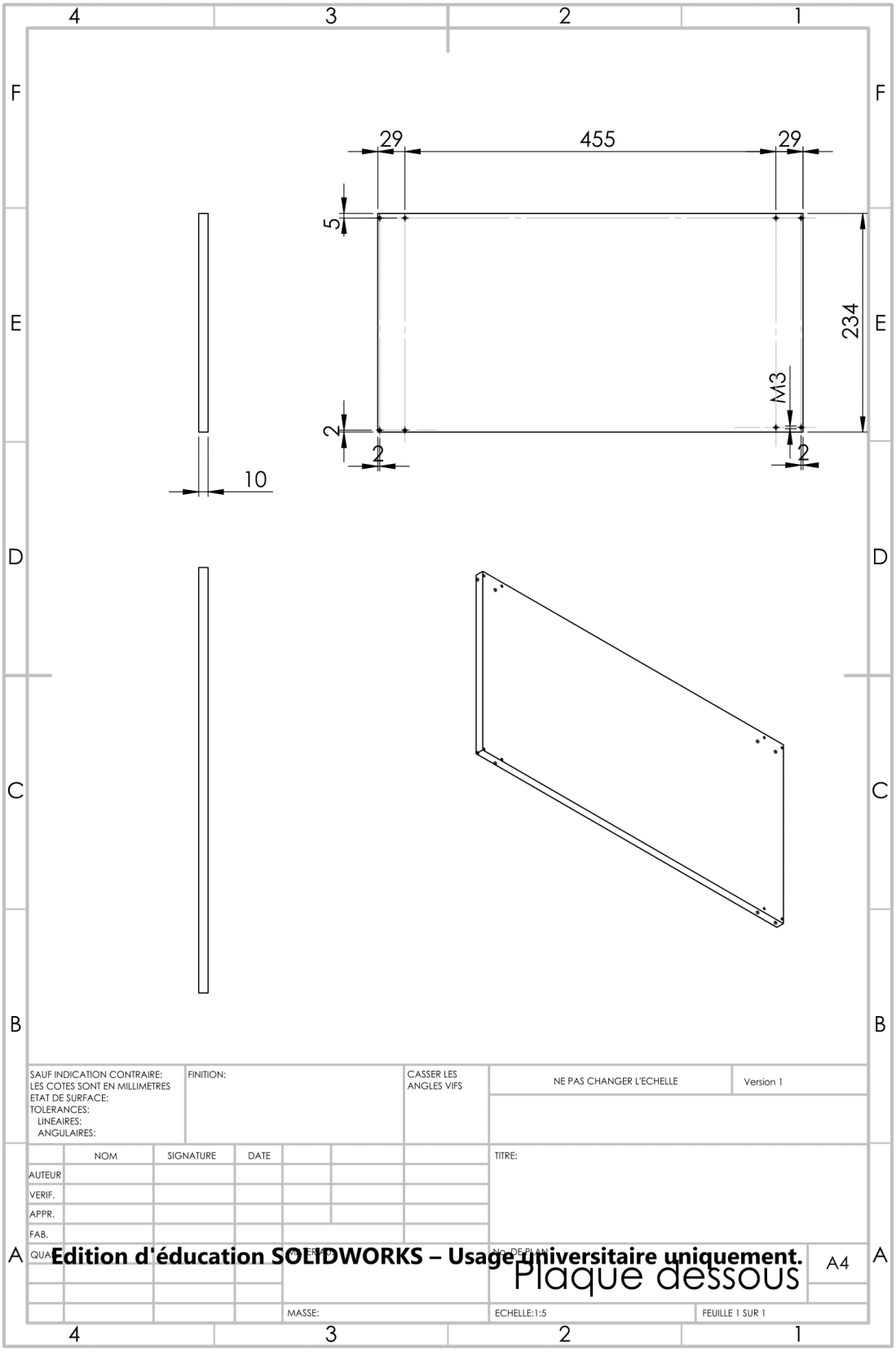

Figure 29: Vehicle bottom plate

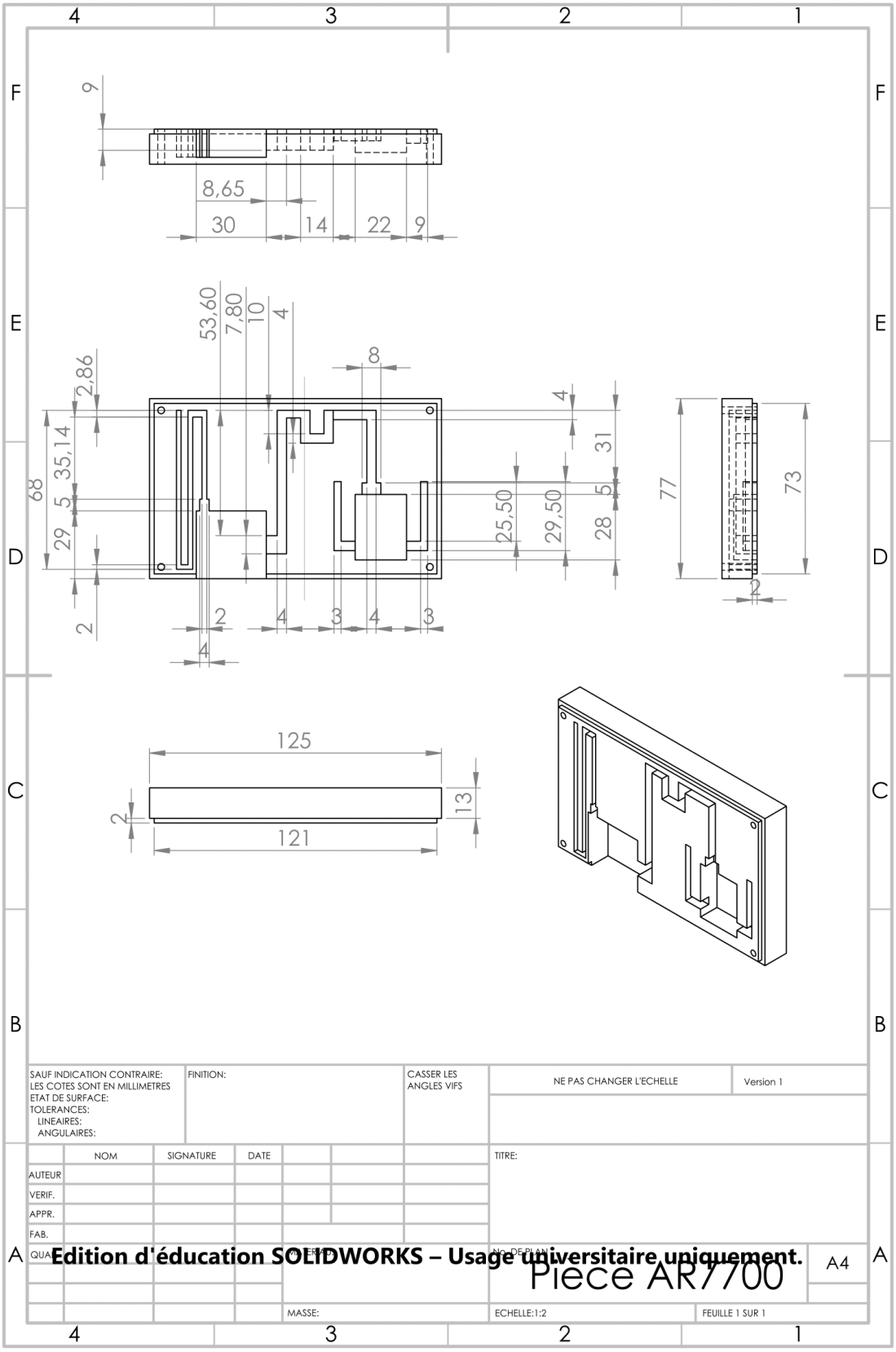

Figure 30: AR7700 Spektrum DSMX box

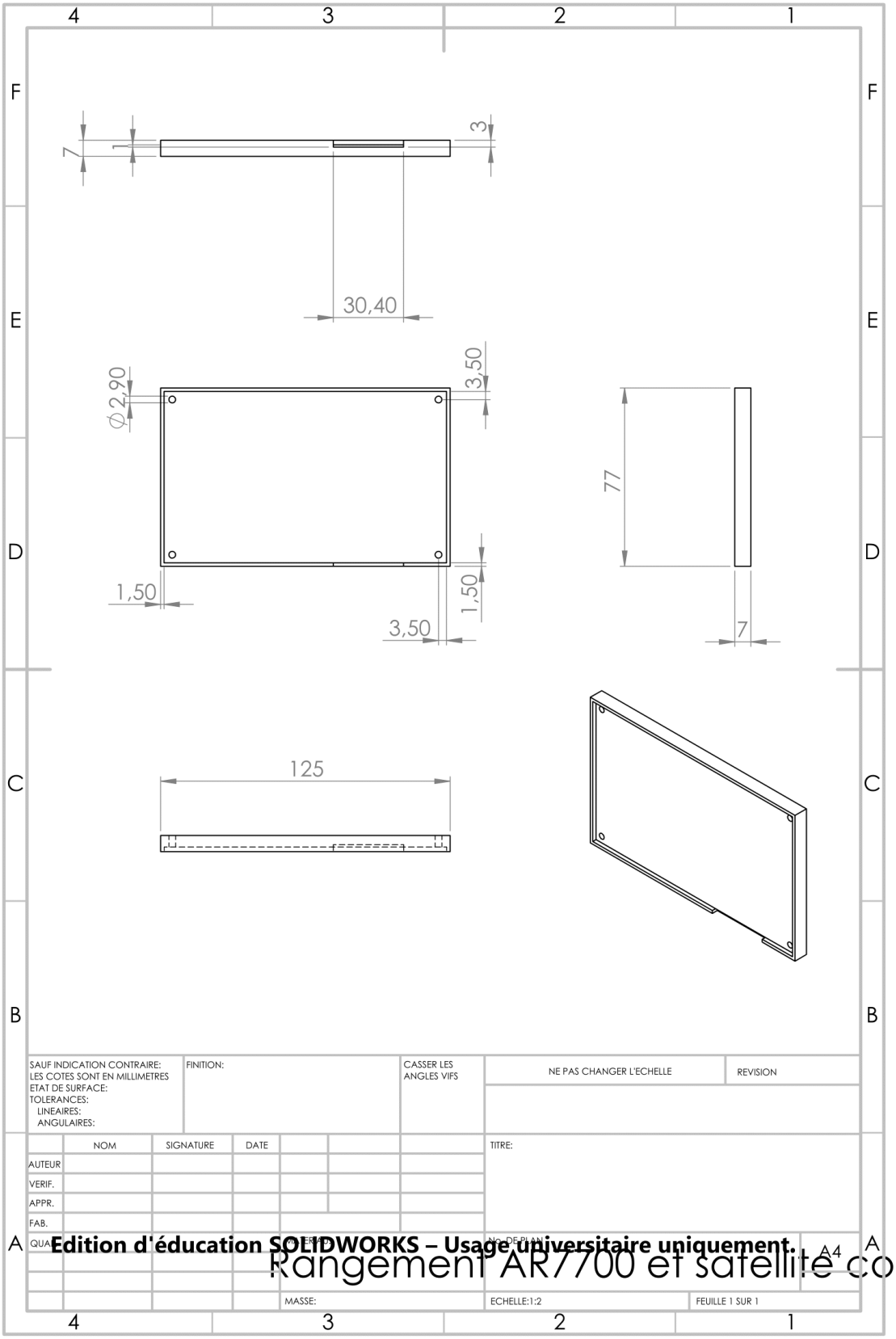

Figure 31: AR7700 & Spektrum DSMX box lid

IR 3D scan module parts drawings

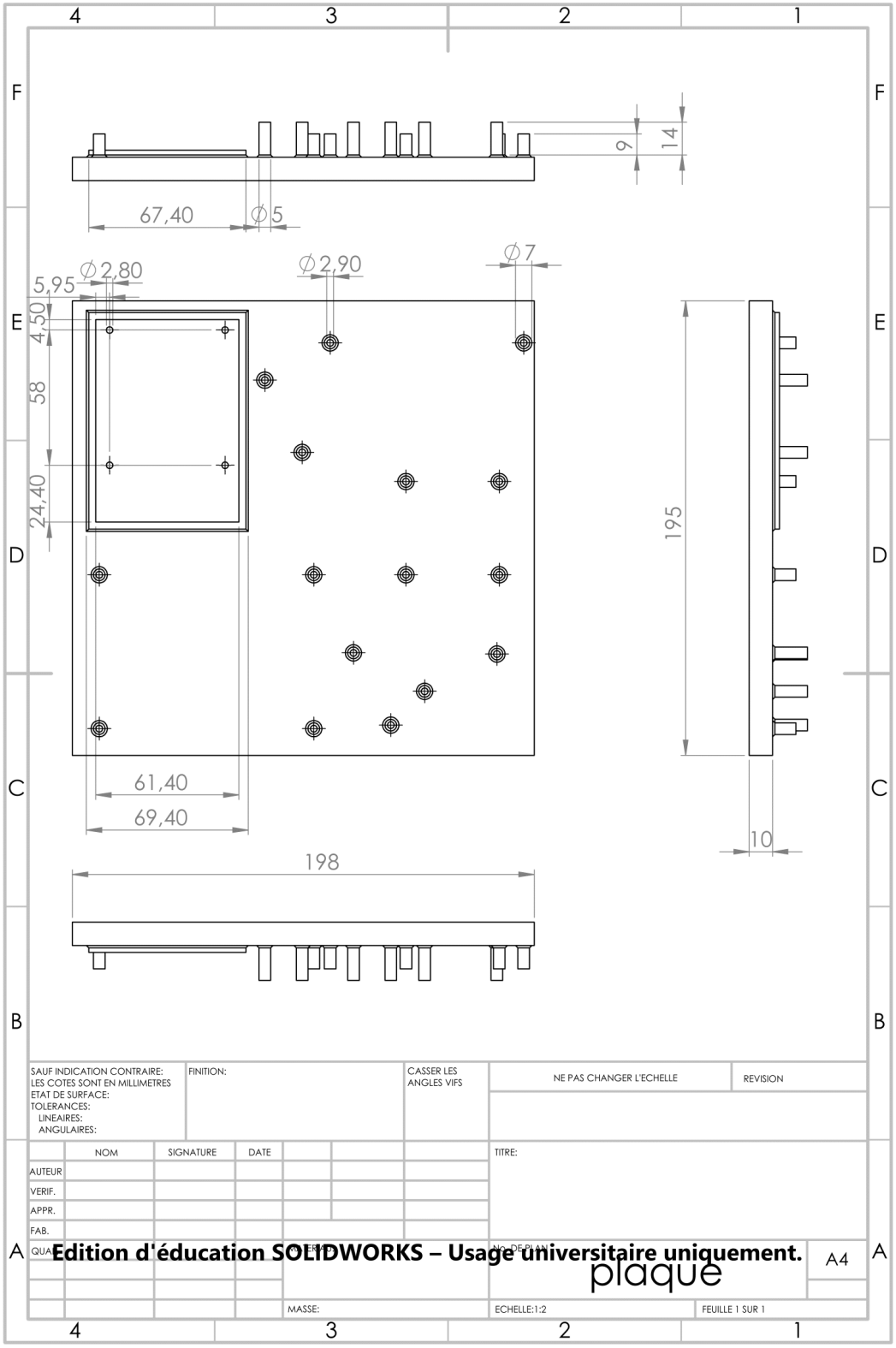

Figure 32: Module main board

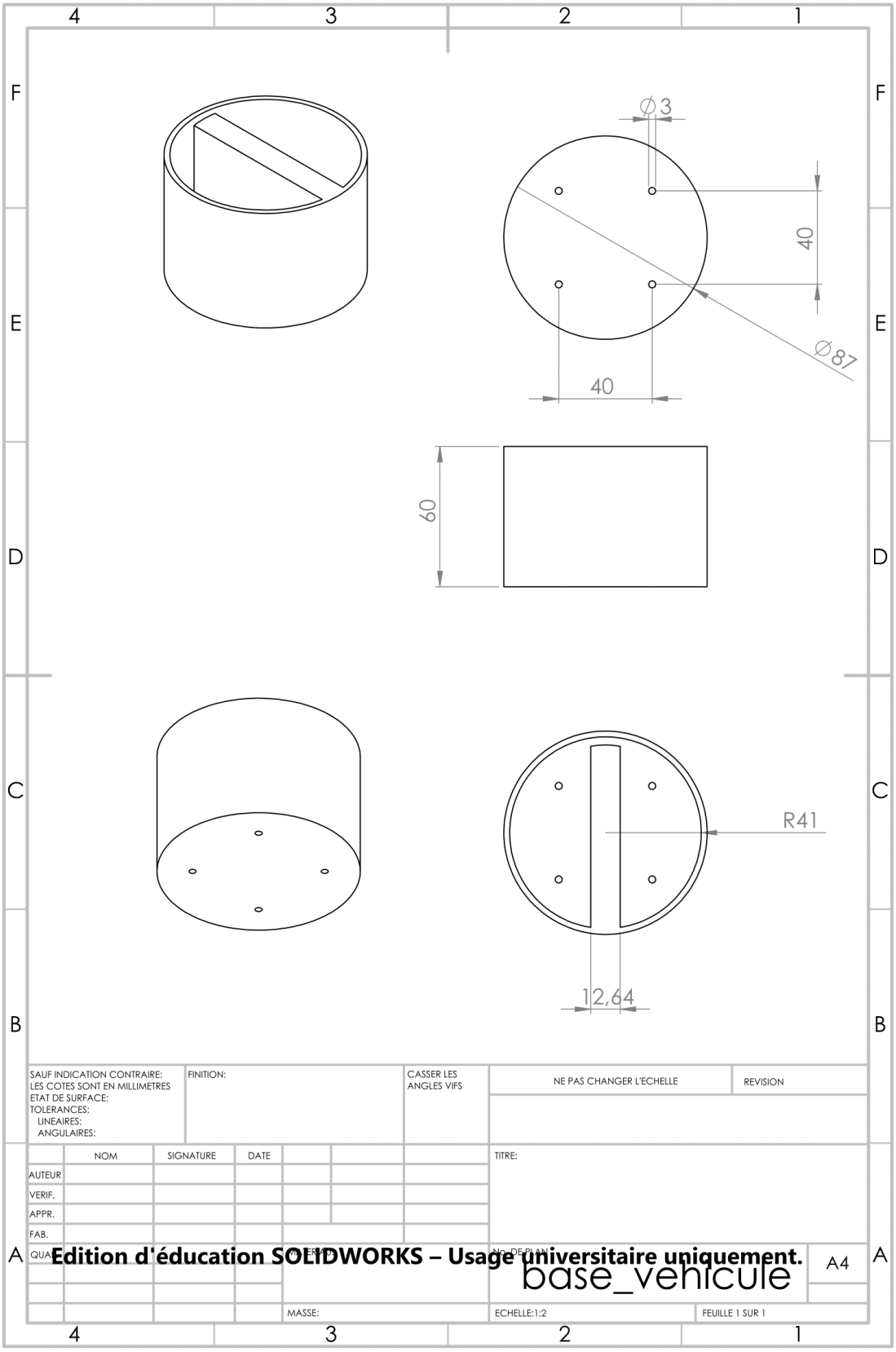

Figure 33: Turret bottom mount

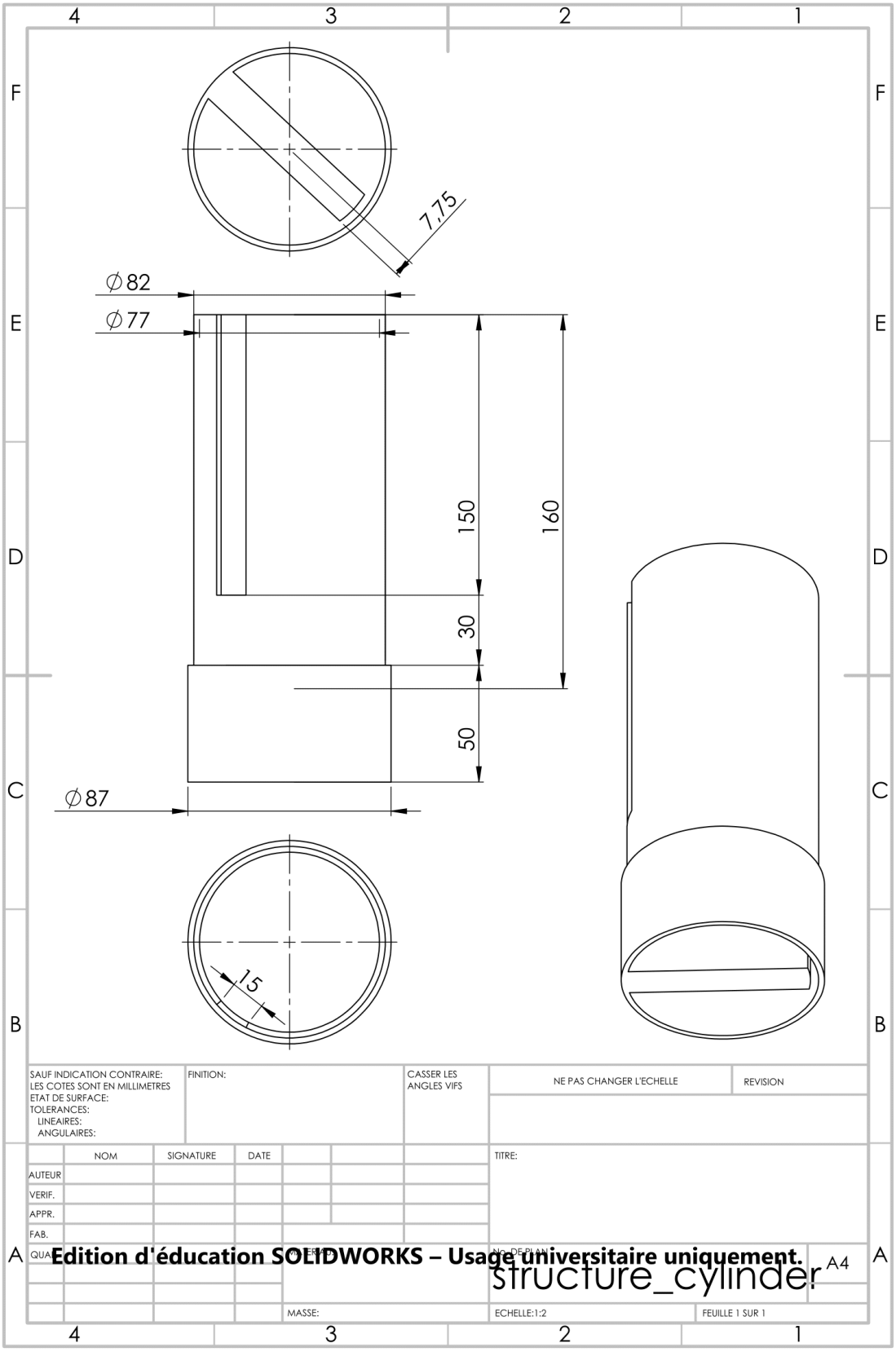

Figure 34: Turret structural cylinder

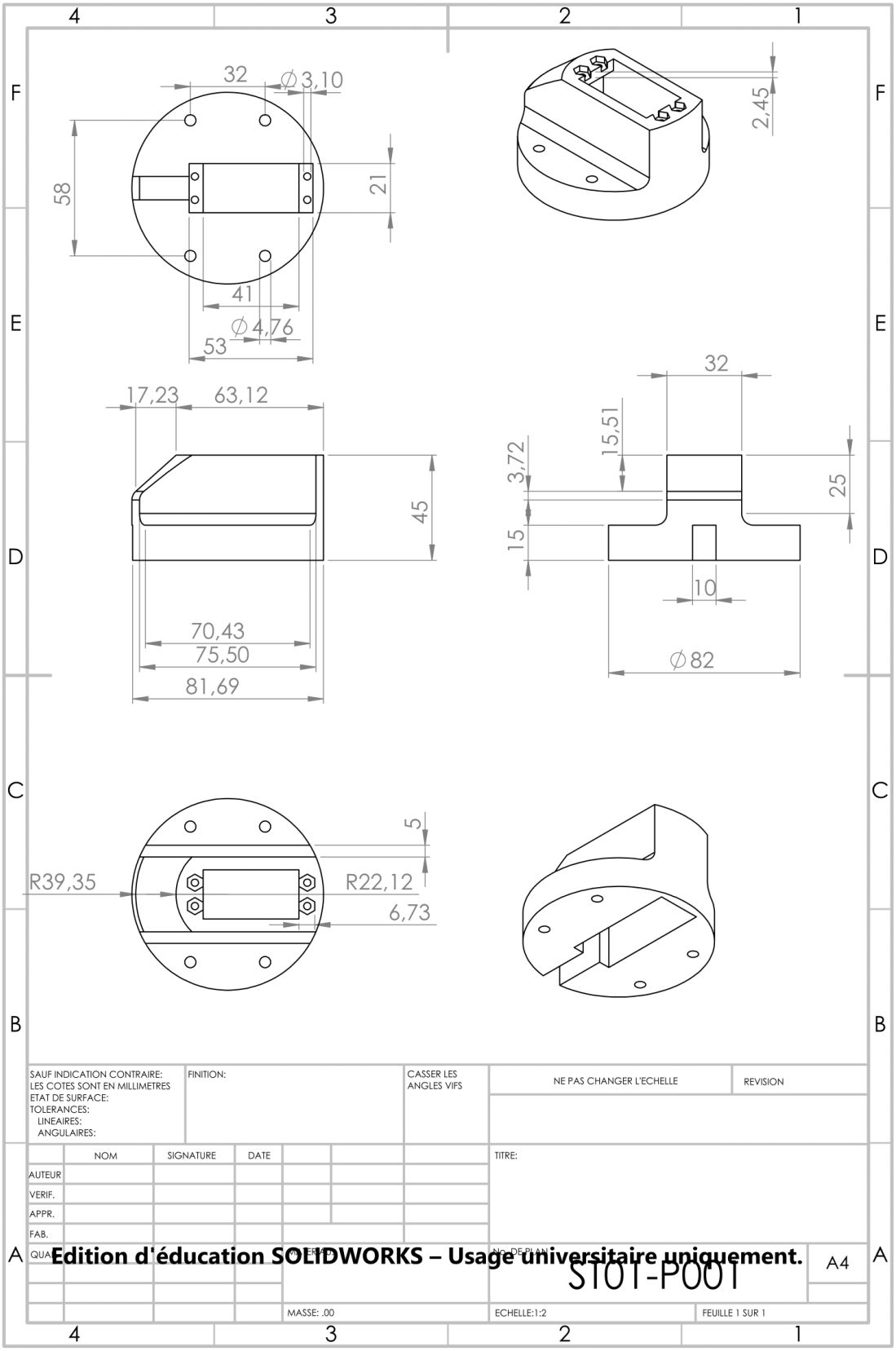

Figure 35: Turret top (part 1/3)

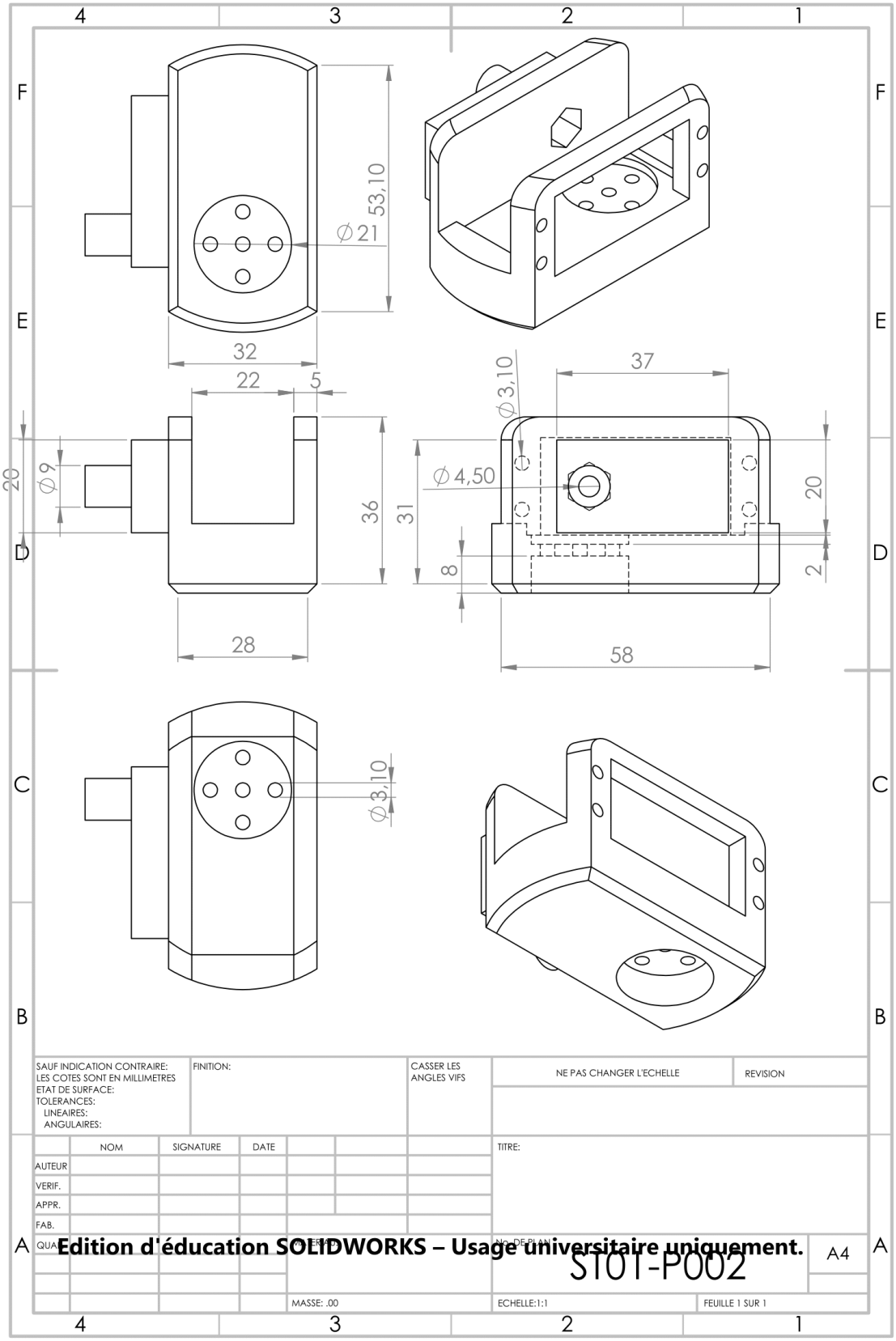

Figure 36: Turret top (part 2/3)

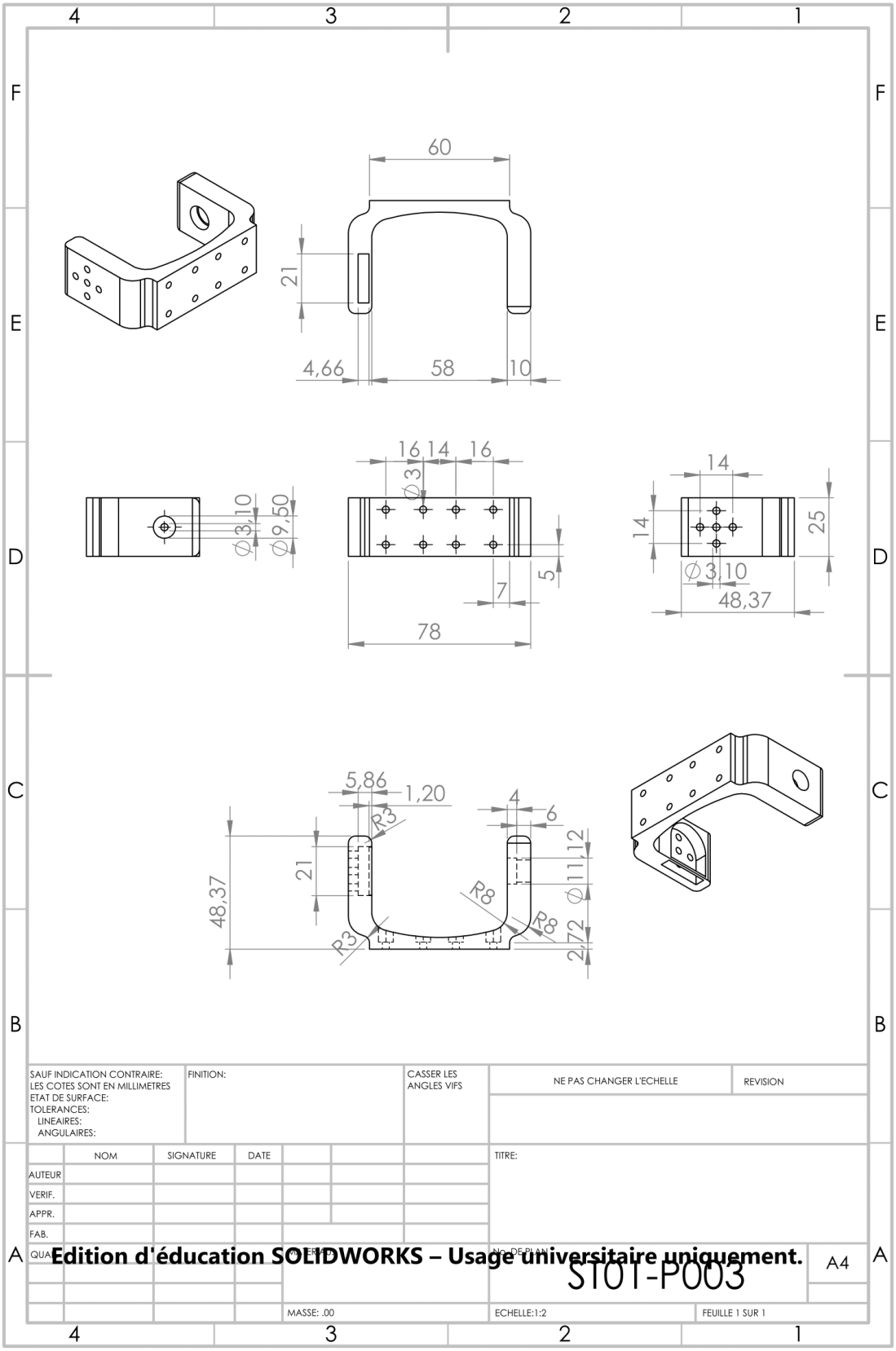

Figure 37: Turret top (part 3/3)

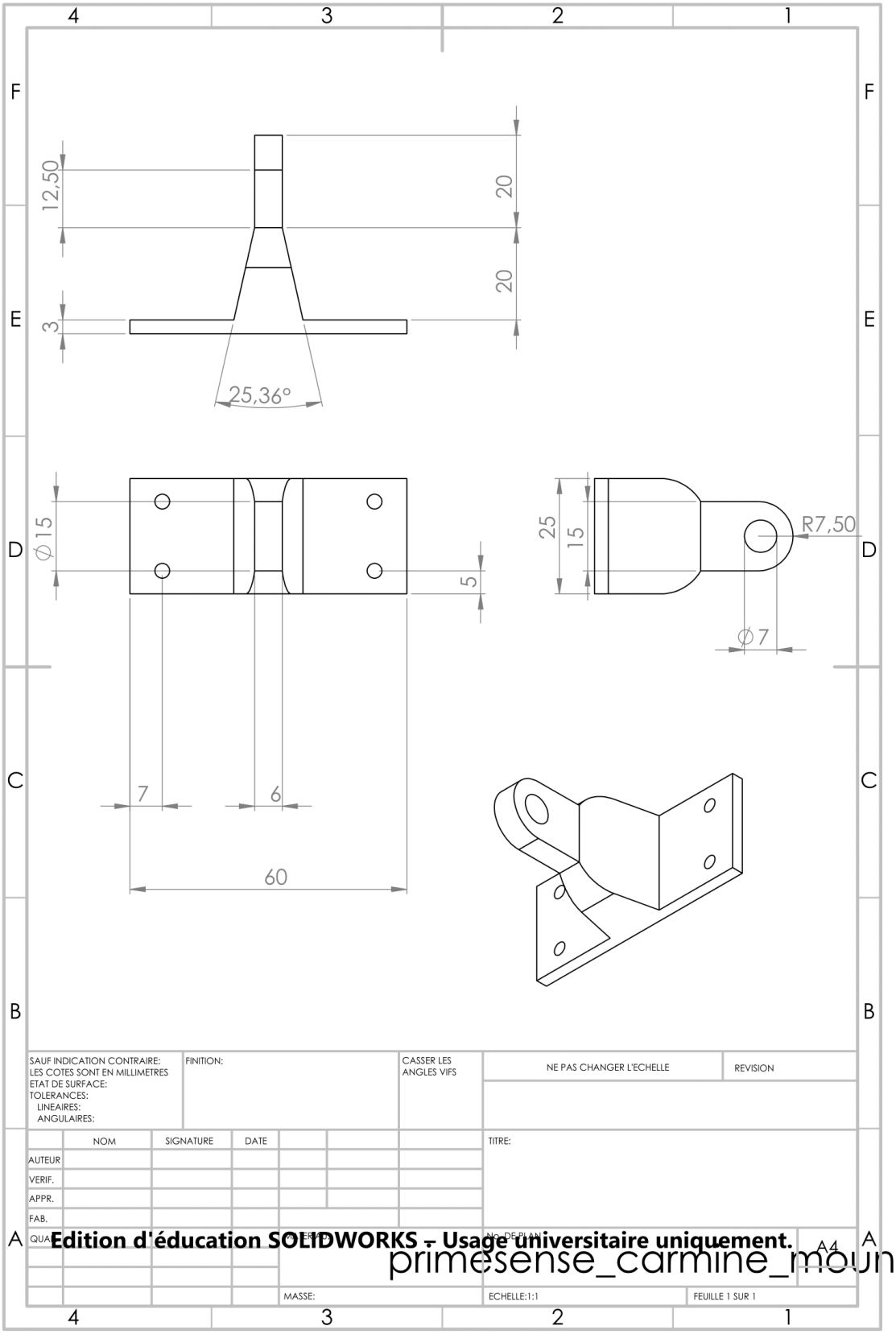

Figure 38: Primesense Carmine RGB-D camera turret mount

Vehicle chassis assembly drawings

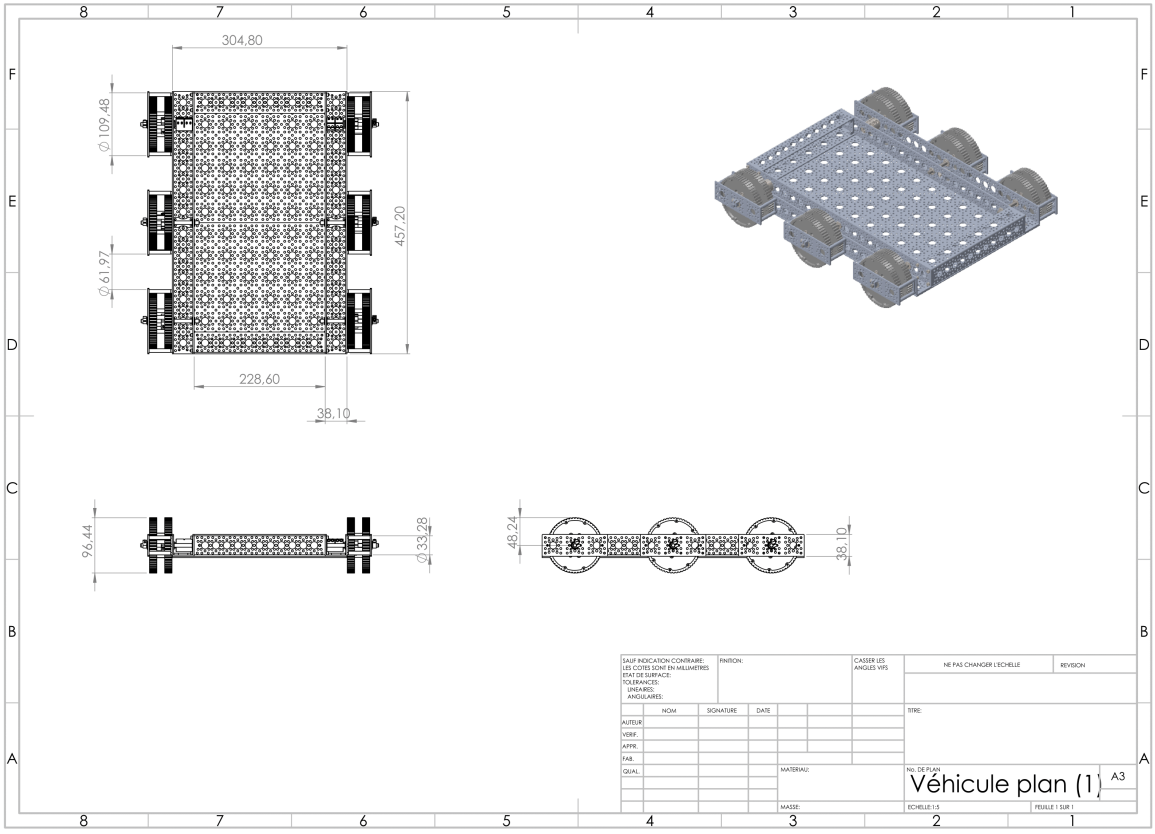

Figure 39: Vehicle assembly (1/2)

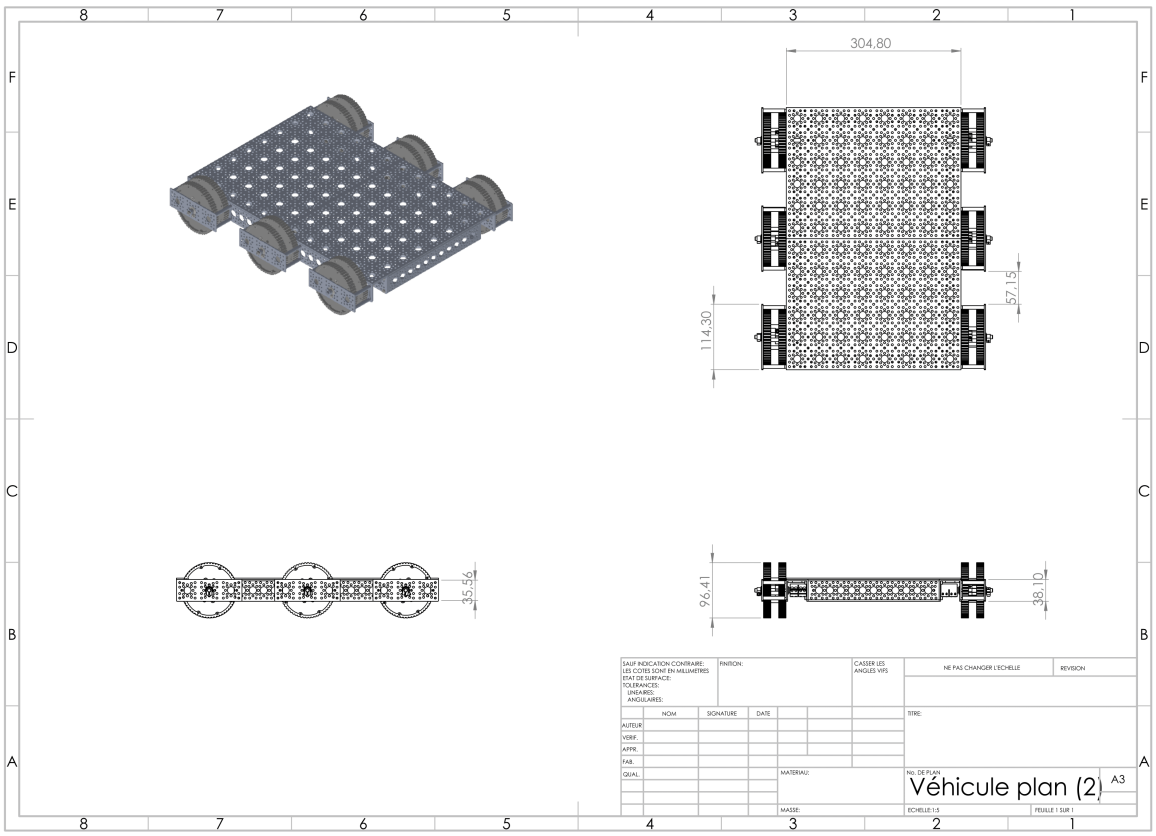

Figure 40: Vehicle assembly (2/2)