# CoopRobo Documentation

***Release 1.0.0***

**Gabriel Araujo**

**Aug 05, 2020**

# Contents:

Collaborative Robotics project


- *Robots*

CHAPTER 1

# Aerial Robots


## 1.1 VR-01

This work seeks the study, documentation and development of cooperative control techniques for unmanned aerial vehicles in order to enable three fixed-wing aircraft to fly, autonomously, in squadron formation.

The use of multiple unmanned aerial vehicles in missions such as surveillance, monitoring and searches can make them faster and more likely to succeed. For this, it is necessary that the UAVs communicate with each other and have a control strategy that determines what each one should do. This project includes three fixed-wing UAVs available at the Laboratory of Aerial Robotics at the University of Brasilia, which should be studied and implemented cooperative control techniques in the embedded hardware and software.

The complete system is constituted by four sub-systems: three aircraft and a base station. The communication between those sub-systems is carried out through communication radios. Each subsystem has its own communication modem and has the ability to communicate directly with any other subsystem.

Each aircraft has two main components that are worth mentioning. One of them is the autopilot, a device responsible for the acquisition, conditioning and processing of signals from the aircraft's sensors and for controlling the aircraft's actuators. The other is the embedded computer, responsible for data processing, control of the planes


Fig. 1: Aircraft Ranger EX Volatex RC

(individual control of each plane or cooperative control between the three planes) and for communication with the others sub-systems through the communication radios.

### 1.1.1 Autopilot

**Note:** This topic consists of a summary of the information available on the page Getting Started - PX4.

#### Introduction

PX4 is the professional open source autopilot, developed by both world-class developers from industry and academia, and supported by the active world wide community. The PX4 can run on multiple flight controll boards. Deserving highlight flight open hardware controllers of the PixHawk series, running PX4 on NuttX OS[1].

Given the options available on the market, Pixhawk 1 was chosen as an autopilot for its best cost benefit for the project.

The Pixhawk operate in several types of vehicle, from racing drones and cargo to land and submersible vehicles. In this article we will focus on aircraft application, where the Pixhawk operate as a general use flight controller, responsible for the acquisition, conditioning and processing of signals from the aircraft's sensors and for controlling the aircraft's actuators.

Pixhawk offers a development environment compatible with Unix and Linux systems, favoring the development of software applications. The Pixhawk system has multithreading capabilities, that is, it can perform several tasks simultaneously without interfering with the other through the sharing of process resources. Moreover, it has integrated autopilot functions with detailed logs missions and flight behavior[2].

Given the options available on the market, Pixhawk 1 was chosen as an autopilot for its best cost benefit for the project.

#### Basic Concepts

This topic introduces some basics concepts about unmanned aerial vehicles (UAV) and the use of PX4 platform.

[1] PX4 Autopilot User Guide. docs.px4.io
[2] Para mais informações a respeito de Multithreading (arquitetura computacional).

### Unmanned Aerial Vehicle (UAV or drone)

A UAV is any types of aircraft that can be controlled on the 3 axes of freedom and do not need a pilot on board to be guided, being able to be controlled remotely or autonomously.

The "brain" of a drone is called autopilot, an instrument responsible for controlling the aircraft's flight path. Referring to UAVs, the autopilot consists of flight control software running on specific hardware for the same function.

### Ground Control Station (GCS)

A Ground Control Station (GCS) is a control platform, usually a software application running on a computer on the ground, which communicates with UAVs for wireless telemetry and provides human operators control of the aircraft.

The ground station delivers to the controller several data in real time on the performance and position of UAVs and can even serve as a "virtual cockpit", providing many of the same instruments that a pilot would have if he were flying an airplane. However, Ground Control software is normally used for planning, uploading flight missions and defining flight parameters.

There are more than ten different ground control stations. In the area of UAV control, the main controllers are Mission Planner, APM Planner 2, MAVProxy, QGroundControl e UgCS. For Tablet / Smartphone, there is Tower (DroidPlanner 3), MAVPilot, AndroPilot e SidePilot.[4]

### Dronecode Platform

PX4 is part of the Dronecode Plataform, a complete platform for drone development, under an open source license the community. It includes, among other things, the PX4 flight stack, QGroundControl ground control station, the Dronecode SDK and the Dronecode Camera Manager.[3]

### Sensors

The PX4 based system uses several sensors to determine vehicle state (these being essential for stabilization and to enable autonomous control). The vehicle states include: position, heading, speed, airspeed, orientation (attitude), rates of rotation in different directions, battery level, etc.

The system minimally requires a gyroscope, accelerometer, magnetometer (compass), barometer and an airspeed sensor for the case of fixed wing (project case). A GPS or other positioning system is also required to activate all automatic modes and some assisted modes.

Pixhawk series flight controllers come with a minimum set of sensors incorporated. Additional/external sensors can be connected to the controller.

### GPS & Compass

The PX4 supports several receivers and compasses (magnetometers) of the Global Navigation Satellite System (GNSS). It also supports Real Time Kinematic (RTK) GPS Receivers, optimizing GPS systems to centimeter level accuracy.

---

**Note:** Pixhawk series controllers include an *internal* compass, but we recommend using an external Compass + GPS module, which is mounted as far as possible from power supply lines to reduce electromagnetic interference.

---

[4] Choosing a Ground Station - Conter documentation. ardupilot.org

[3] Eduardo Moura Cirilo Rocha. 2017. Desenvolvimento de um sistema com veículos aéreos não-tripulados autônomos, Universidade de Brasília, Brasil.

The PX4 supports the connection of up to 4 internal or external magnetometers, although only one will actually be used as a heading source. The system automatically chooses the best available compass based on their internal priority (external magnetometers have a higher priority). If the primary compass fails in-flight, it will failover to the next one. If it fails before flight, arming will be denied.

More information and the list of supported GPS/Compass can be found at GPS/Compass.

### Airspeed

---

**Tip:** Airspeed sensors are highly recommended for the safe operation of a fixed wing UAV or VTOL (Vertical Take-Off and Landing).

---

The flight of a fixed wing UAV depends on the airspeed, since this guarantees its support in flight and not the speed in relation to the ground. The autopilot has no other means to detect stall (loss of lift of the aircraft), for this reason air speed sensors are highly recommended.

More information and the list of supported Airspeed sensors can be found at Airspeed sensors.

### Distance (telemeter)

Distance sensors provide real-time distance measurement. It can be optical, when based on a focusing mechanism, or ultrasonic (ecotelemeter or acoustic rangefinder), when using sound reflections. They are used for improved landing behaviour, terrain following, collision prevention, warning of regulatory height limits, etc.

The PX4 supports a wide variety of distance sensors, using different technologies and supporting different features. More information and the list of supported distance sensors can be found at More information and the list of supported distance sensors can be found at Distance sensors.

### Optica Flow

PX4Flow is an optical flow smart camera that can track motion, and has as integrated sonar sensor. PX4 blends the sensor output with information from other position sources (e.g. GPS) to provide a more accurate position lock. This sensor can be used indoors, when no GPS signal is available.

Most of its applications are directed to rotary-wing aircraft.

### Pixhawk Specifications

- **Processor**
  - 32-bit ARM Cortex M4 core with FPU
  - 168 Mhz/256 KB RAM/2 MB Flash
  - 32-bit failsafe co-processor
- **Sensors**
  - MPU6000 as main accel and gyro
  - ST Micro 16-bit gyroscope
  - ST Micro 14-bit accelerometer/compass (magnetometer)
  - MEAS barometer
- **Power**
  - Ideal diode controller with automatic failover

  - Servo rail high-power (7 V) and high-current ready
  - All peripheral outputs over-current protected, all inputs ESD protected
- **Interface**
  - 5x UART serial ports, 1 high-power capable, 2 with HW flow control
  - Spektrum DSM/DSM2/DSM-X Satellite input
  - Futaba S.BUS input (output not yet implemented)
  - PPM sum signal
  - RSSI (PWM or voltage) input
  - I2C, SPI, 2x CAN, USB
  - 3.3V and 6.6V ADC inputs
- **Dimensions**
  - Weight 38 g (1.3 oz)
  - Width 50 mm (2.0")
  - Height 15.5 mm (.6")
  - Length 81.5 mm (3.2")
- **Included items**
  - 1 x SanDisk Ultra micro SD Card (8GB)
  - 1 x MRC0225- Cable [3-Pins DF-13] to Switch+LED
  - 1 x MRC0224- Cable [2-Pins DF-13] to Buzzer
  - 1 x I2C Splitter
  - 2 x MRC0213- Cable [6-Pins JST-GH] to [6-Pins DF-13], (Telemetry Radio, Power module and Extra)
  - 1 x MRC0216- Cable [6-Pins DF-13] to [6-Pins DF-13], (For legacy products)
  - 4 x Damping Foams
  - 3 x Decals "APM Rover", "APM Copter" and "APM Plane"

#### More information

- Dronecode Platform, Basic Concepts, PX4 Autopilot User Guide. docs.px4.io

### 1.1.2 Autopilot Configuration

The Pixhawk's preparation for flight essentially consists of installing the firmware on the device, connecting and calibrating the sensors and mounting on the aircraft structure.

This section contains essential configuration topics:

#### QGroundControl

QGroundControl is one of the main Ground Control Station (GCS) currently available to work with autopilots compatible with MAVLink, including PX4 and ArduPilot.

To implement this project, QGroundControl was chosen as GCS because it provides easy and direct use for beginners, in addition to offering experienced users support for advanced features in complete flight control and vehicle configurations with PX4.

Moreover, QGroundControl is one of the most stable ECSs in relation to the others, has a simple and efficient interface and is available in several operating systems, such as Windows, Mac OS X, Linux, Android and iOS.

### System Requirements

QGroundControl can run normally on most modern computers. A computer with an i5 and at least 8 GB of RAM will perform well for all applications in the program. For a better experience, it is advisable to have the operating system in its latest stable version.

### Installation

- **Windows**
  - Install QGroundControl for 64-bit versions of Windows Vista or later:
    1. Download the installer QGroundControl.exe
    2. Double-click the downloaded executable to open the installer
- **Mac OS X**
  - Installing QGroundControl on Mac OS 10.10 or later:
    1. Download QGroundControl.dmg
    2. Double-click the downloaded .dmg file
    3. Move your the $ QGroundControl application to your *Application* folder
- **Ubuntu**

Ubuntu has a **serial modem manager** that interferes with applications involving robotics that use a serial (or USB serial) port. Before installing **QGroundControl**, is necessary to remove such **serial modem manager** and grant its user the permissions to access the serial port. You also need to install GStreamer to enable video streaming.

- Before installing QGroundControl for the first time:

1. On the command prompt type:

```
sudo usermod -a -G dialout $USER
sudo apt-get remove modemmanager -y
sudo apt install gstreamer1.0-plugins-bad gstreamer1.0-libav -y
```

2. Logout and login again to enable the change to user permissions.

- To install QGroundControl for Ubuntu Linux 16.04 LTS or later:

1. Download QGroundControl.AppImage
2. Install and run Install using the terminal commands:

```
cd Downloads
chmod +x ./QGroundControl.AppImage
./QGroundControl.AppImage
```

---

**Tip:** The last command line is not necessary if the user goes to the file manager, search for the downloaded QGroundControl file and double-click it.

---

- **Android**
  - QGroundControl is available from the Google Play Store on QGroundControl - play.google.com .
- **IOS**
  - QGroundControl is available from the App Store.

### Firmware

The installation of the firmware on the flight controller hardware can be done in two ways, by using an Ground Control Station (GCS) program or directly by using developer tools without using an auxiliary program. An GCS is a software application that runs on a ground computer and communicates with the vehicle through the use of wireless telemetry[1].

The main GCS's available are Mission Planner, APM Planner 2, MAVProxy, QGroundControl and UgCS. To implement this project, QGroundControl was chosen because it provides easy and direct use for beginners, good documentation, a more stable program in relation to the others and also offers advanced features for experienced users.

**Tip:** Before starting this section, it is recommended to download and install QGroundControl on your computer.

**Note:** The official QGroundControl documentation is available at QGroundControl.

### Stable Installation

We recommend using the latest version of PX4 in order to benefit from bug fixes and get the best and latest features.

**Note:** Before installing the firmware, all USB connections of the vehicle must be disconnected and the vehicle must not be powered by a battery.

1. Select the *Gear* icon (**Vehicle Setup**) on the top toolbar, then select **Firmware** on the sidebar.
2. Connect the flight controller directly to your computer via USB (do not connect through a USB hub).
3. Select the PX4 Flight Stack X.x.x Release option to install the latest stable version of PX4 for your hardware (autodetected).
4. Click **OK** to start the installation.

The firmware will then proceed a several number of upgraee steps (download the new firmware, erasing old firmware version, etc.). The overall progress is displayed in a progress bar.

Once the firmware has completed loading, the device will reboot and reconnect.

### More Information

- PX4 user guide > Firmware.
- QGroundControl user guide > Firmware.
- PX4 Setup Video (Youtube)

### Airframe

After installing the firmware, is necessary to configure the firmware parameters for the specific structure of your vehicle.

[1] Eduardo Moura Cirilo Rocha. 2017. Desenvolvimento de um sistema com veículos aéreos não-tripulados autônomos, Universidade de Brasília, Brasil.

### Set the Airframe

1. With QGroungControl booted and the controller connected to the computer, select the gear icon (**Vehicle Setup**) in the top toolbar and then select **AirFrame** in the sidebar.
2. Select the vehicle group that corresponds to the aircraft structure and then use the dropdown menu within the group to choose the aircraft that best matches your vehicle.
3. Click **Apply and Restart**. Click **Apply** in the following prompt to save the settings and restart the vehicle.

### More Information

- PX4 user guide > Airframe.
- QGroundControl user guide > Airframe.

### Connections

The image below shows the connections of the sensors and other items included in the Pixhawk. Each part will be analyzed in more detail in the following sections.

### Buzzer and Safety Switch

The buzzer provides audible signals that indicate the situation of the UAV. While the switch operates in the safety of the aircraft, locking and unlocking the engines.

---

**Note:** The safety switch is activated by default and when activated, it does not allow flight, blocking the engines. To disable safe mode, press and hold the switch for 1 second. You can activate safe mode again by pressing the switch.

---

To connect the buzzer and the safety switch (required items), simply connect them to the Pixhawk as shown below.


### Slitter I2C

The I2C slitter expands the number of I2C ports allowing the connection of up to four peripherals to the Piwhawk. Use a 4-wire cable to connect the I2C slitter and to power an external compass, an LED display, a digital air speed sensor and/or any other peripheral compatible to the vehicle.

### Airspeed Sensor

Em edição...

### GPS + Compass

The GPS, another indispensable device, must be connected to the GPS port (6-pin) using the 6-wire cable provided in the kit. The compass connection is optional, but strongly recommend its use. To connect it, connect a 4-wire cable to an I2C port on the I2C slitter, as shown below.

---

**Note:** The GPS/Compass should be mounted on the aircraft's chassis as far away from other electronic components as possible, with the indicator arrow facing forward and as aligned as possible with the Pixhawk.

---

### Radio Control (RC)

The radio control (RC) system is necessary if you want to manually control your vehicle, as the Pixhawk does not require a radio system for autonomous flight modes.

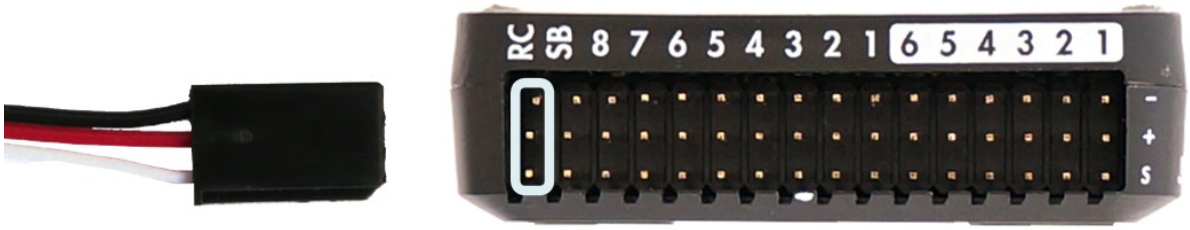
To connect the radio control system, is necessary need to select a compatible transmitter/receiver and then link it up so that they can communicate.

**Tip:** Read the instructions that came with your transmitter/receiver.

The following instructions show how to connect the different types of receivers to the Pixhawk:

- Spektrum and DSM receivers connect to the SPKT/DSM input.


- PPM-SUM and S.BUS receivers connect to ground, power and RC signal pins, as shown.

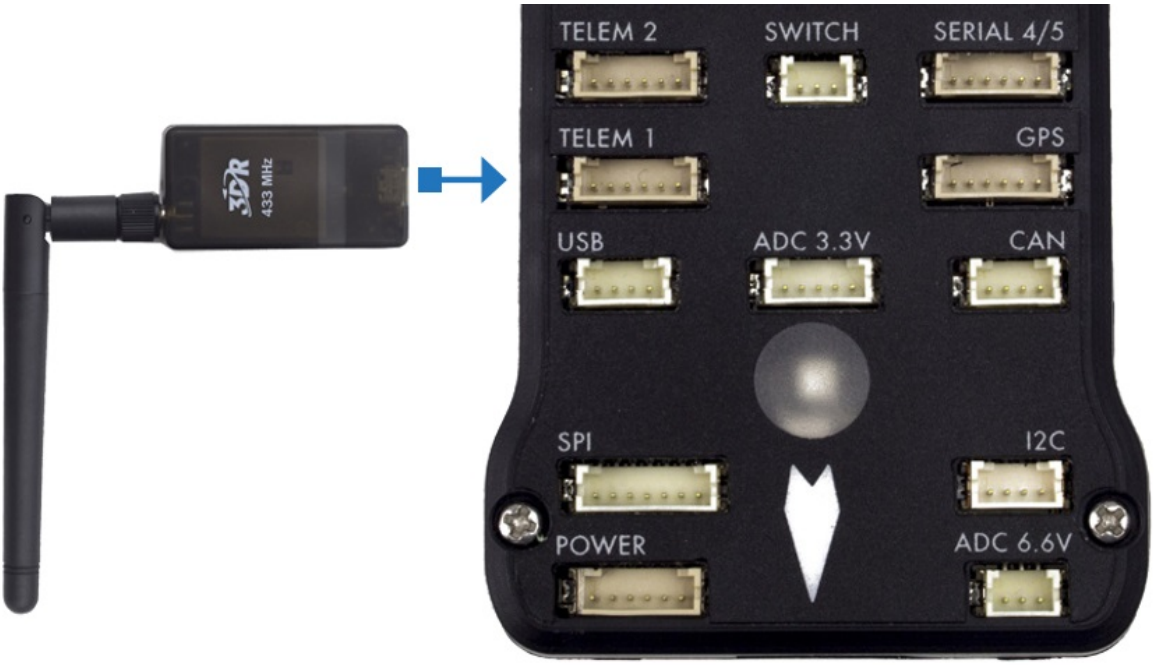



- The PPM and PWM receivers that have an individual wire for each channel must connect to the RC port via a PPM encoder (PPM-Sum receivers use a single signal wire for all channels).

For more information on selecting a radio system, receiver compatibility, and connecting your transmitter and receiver pair, see: Remote control transmitters and receivers.

### Telemetry

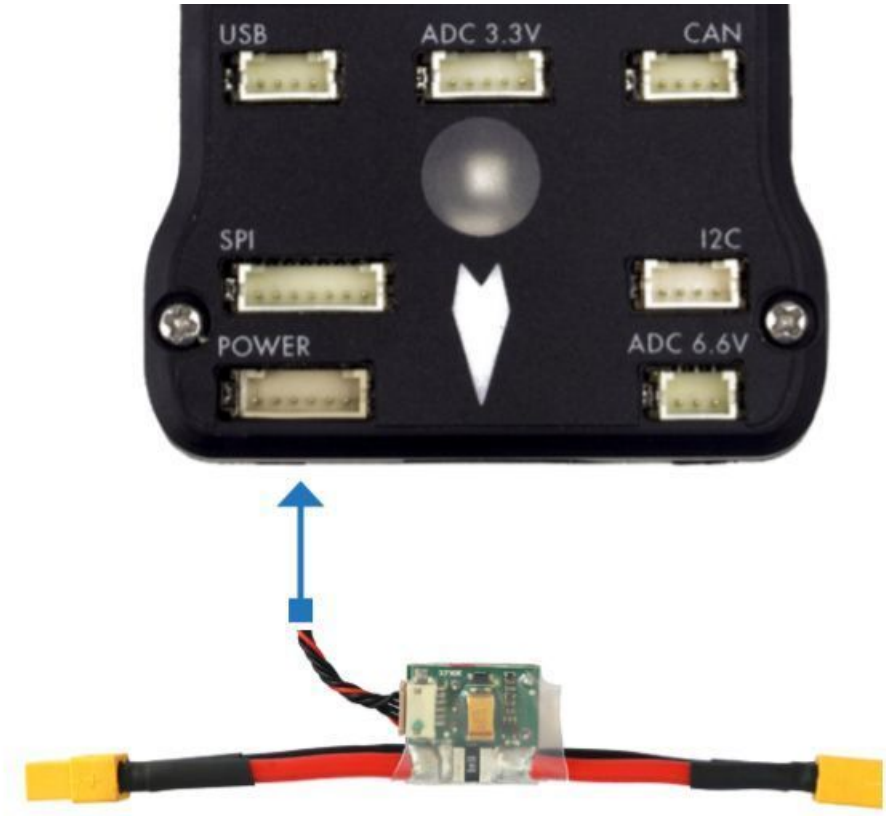
Telemetry modems can be used to communicate and control a vehicle in flight from an ground station (for example, you can direct the UAV to a specific position or load a new mission). A modem must be connected to your vehicle, as shown below. Another modem must be connected to the ground station computer or mobile device (usually via a USB port).




### Power Module

The **Power module** (PM) supplies power to the battery flight controller and also sends information about the analog current and voltage supplied by the module (including power to the flight controller and motors, etc.).

The output of the power module (PM) must be connected to the Pixhawk ** POWER ** port using a 6-wire cable, as shown in the image. The input module must be connected to a battery Po, while the main output will be responsible for supplying power to the ESCs and the aircraft engine (possibly through a power distribution board, depending on the aircraft).




### Distance Sensor

Pixhawk supports several different distance sensors, including Lidars (which use lasers or infrared rays for distance measurements) and Sonars (which use ultrasonic waves), and also include the Maxbotix Sonar and Pulsed Light LED range finders. Therefore, the installation varies from device to device. More information about the sensor configuration can be seen in Rangefinders.


Fig. 2: Example of some compatible distance sensors

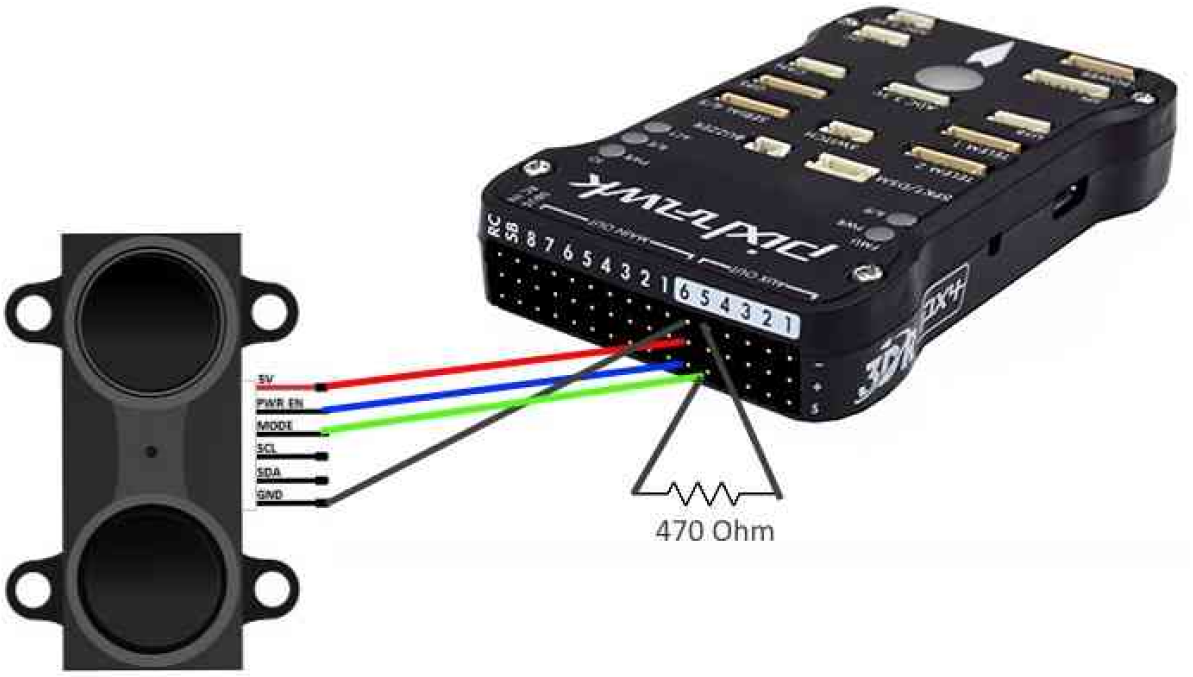
To implement the project, the Lidar sensor was chosen to enable the automatic landing function due to its greater accuracy compared to the others. Lidar sensor can be connected to the Pixhawk in two ways, via the I2C protocol on the I2C port (or I2C slitter) or by pulse-width-modulation (PWM) on the PWM track.

According to the Pixhawk documentation, the Lidar used presents interference problems with other devices when connected to the I2C port. Thus, the PWM connection was chosen. A connection diagram can be seen in the table below and the assembly diagram can be seen in the following figure, where the value of the resistor can vary between 200Ω and 1kΩ[1].

[1] Eduardo Moura Cirilo Rocha. 2017. Desenvolvimento de um sistema com veículos aéreos não-tripulados autônomos, Universidade de Brasília, Brasil

Table 1: Connection diagram between Lidar and Pixhawk

| Sinal LIDAR-Lite | Sinal Pixhawk |
|---|---|
| J1 | CH6 Out - V+ |
| J2 | CH6 Out - Signal (sinal interno 55) |
| J3 | CH5 Out - Signal (sinal interno 54) |
| J4 | |
| J5 | |
| J6 | Ch6 Out - Ground |




More details about the connection can be found in LIDAR-Lite Rangefinder.

### More information

- Pixhawk Wiring Quick Start - PX4 User Guide
- Basic Assembly - PX4 User Guide
- Pixhawk Series - PX4 User Guide
- Peripheral Hardware - Ardupilot Docs

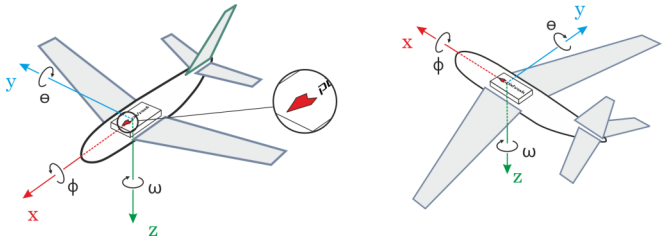
### Mounting the Pixhawk

### Autopilot Orientation

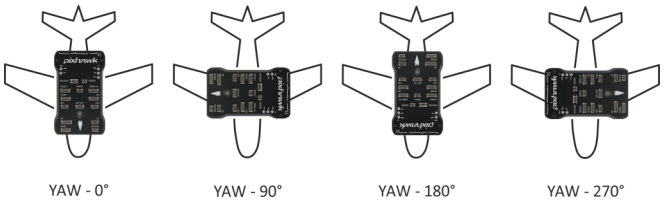
By default, the flight controller and the external compass should be placed on the aircraft frame oriented so that the arrow points towards the front of the vehicle. If the card or external compass are in any other direction, you need to correct the orientation in firmware.

### Calculating Orientation

The compensations for the rotation angles YAW, PITCH and / or ROLL are calculated in relation to the vertical orientation pointing forward (clockwise rotation around the Z, Y and X axes, respectively). This diagram is called body frame and the default orientation is given by `ROTATION_NOME`.




For example, the image below shows aircraft rotations around the z axis (YAW), corresponding, respectively, to: `ROTATION_NONE`, `ROTATION_YAW_90`, `ROTATION_YAW_180`, `ROTATION_YAW_270`.




### Setting the Orientation

To set the orientations on firmware:

---

**Note:** Before setting the orientation, QGroundControl must be started, connected to the vehicle and the firmware must have already been installed on the flight controller board.

---

1. Select the **Gear** icon (Vehicle Setup) in the top toolbar and then Sensors in the sidebar.
2. Select the **Set Orientations** button.
3. Select the autopilot orientation, as calculated above.
4. Select the **External Compass Orientation** in the same way (this option will only be displayed if your vehicle has an external compass).
5. Press **OK**.

---

**Tip:** Complete documentation on how to adjust the autopilot orientation is available in Autopilot Orientation.

---

### Vibration Isolation

Pixhawk boards have built-in accelerometers and gyroscopes, being sensitive to vibrations. High levels of vibration can cause a number of problems, including reduced flight performance, shorter flights and increased vehicle wear. In extreme cases, vibration can lead to sensor failures, resulting in estimation errors or even flight interruption.

For this reason, the Pixhawk comes with *vibration damping foams*.

The Pixhawk must be mounted on the aircraft using the anti-vibration foams included in the kit. It should be positioned as close as possible to the vehicle's center of gravity.

**Tip:** To determine whether the vibration levels are too high and use some techniques to improve the vibration characteristics, recommended to the topic PX4 user guide > Vibration Isolation.

### More information

- Advanced Orientation Tuning.
- PX4 user guide > Sensor Orientation.
- QGroundControl user guide > Sensors.
- PX4 user guide > Vibration Isolation.

### AutoPilot Calibration

This section contains the main autopilot calibration topics:

### Compass Calibration

All internal and external magnetometers that are connected to the Pixhawk will be configured in the process of calibrating the compass

### Calibration Steps

1. Open the app QGroundControl and connect the vehicle by the wire to the computer's usb.
2. Select the **Gear** icon (Vehicle Setup) in the top toolbar and then **Sensors** in the sidebar.
3. Click the **Compass** sensor button.
4. Click **OK** to start the calibration.
5. Place the vehicle in the indicated position of any of the images shown in red (not calibrated) let it stand and wait a moment in the chosen position. Once prompted (the orientation-image turns yellow) rotate the vehicle around the specified axis in either/both directions. Once the calibration is complete for the current orientation the associated image on the screen will turn green.
6. Repeat the calibration process for all vehicle orientations.

### Futher Information

- PX4 user guide > Compass.
- QGroundControl user guide > Sensors.
- PX4 Setup Video (Youtube)

### Gyroscope Calibration

Through the QGroundControl application you will be guided to position the vehicle in a location with a flat surface and keep it immobile in the determined position.

### Calibration Steps

1. Click the **Gyroscope** sensor button.
2. Position the vehicle on a flat surface and keep it in position.
3. Click **OK** to start the calibration.
4. When finished, the QGroundControl application will show a complete green bar and a green outline around the vehicle image.

---

**Tip:** If there is any movement in the vehicle, the QGroundControl application will automatically restart the gyroscope calibration process.

---

### More Information

- QGroundControl user guide > Gyroscope.

### Accelerometer Calibration

The QGroundControl app will show you all the steps to place and hold your vehicle in different orientations.

### Calibration Steps

---

**Tip:** It is necessary to have the autopilot orientation set to proceed with the accelerometer calibration steps. Otherwise, enter the **Mountig the Pixhawk** tab and perform the steps in the **Setting the Orientation** subtab.

---

1. Open the app QGroundControl and connect the vehicle by the wire to the computer's usb.
2. Select the **Gear** icon (Vehicle Setup) in the top toolbar and then **Sensors** in the sidebar.
3. Click the **Accelerometer** sensor button.
4. Click **OK** to start the calibration.
5. Place the vehicle in the positions shown by the image that appear on the application screen. When the position is requested, the orientation image will turn yellow, leave it in that position (without moving the vehicle) until the calibration is completed. The image will turn green when the calibration is complete.
6. Repeat the calibration process for all vehicle orientations.

### Futher Information

- PX4 user guide > Accelerometer.
- QGroundControl user guide > Sensors.
- PX4 Setup Video (Youtube)

### AirSpeed Calibration

The airspeed calibration needs to read a stable baseline with 0 airspeed in order to determine an offset. Cup your hands over the pitot to block any wind (if calibrating the sensor indoors this is not needed) and then blow into the tube using your mouth (to signal completion of the calibration).

### Calibration Steps

1. Open the app QGroundControl and connect the vehicle by the wire to the computer's usb.
2. Select the **Gear** icon (Vehicle Setup) in the top toolbar and then **Sensors** in the sidebar.
3. Click the **AirSpeed** sensor button.
4. Shield the sensor from the wind (i.e. cup it with your hand). Take care not to block any of its holes.
5. Click **OK** to start the calibration.
6. Blow on the tip of the pitot tube to indicate that the calibration is complete.
7. Wait two to three seconds before removing the cover, remembering that the calibration will be completed in a few seconds.

---

**Tip:** Blowing into the tube is also a basic check that the dynamic and static ports are installed correctly. If they are swapped then the sensor will read a large negative differential pressure when you blow into the tube, and the calibration will abort with an error.

---

### More Information

- QGroundControl user guide > AirSpeed.

### Radio Setup

The Radio Setup option that is present in QGroundControl is used to configure the mapping of your remote control unit's main attitude control sticks (rotation, pitch, yaw, throttle) fot the channels and to calibrate the minimum, maximum, trim and reverse settings for all other controls / RC channels.

### Binding the Receiver

Before calibrating the radio system, the receiver and transmitter must be connected. The connection process of a transmitter and receiver pair is hardware specific.

- QGroundControl user guide > Spektrum receiver.
- FrSky receiver (Youtube)

### Calibration Steps

In the calibration process it will be indicated that the handles are moved in a specific pattern that is shown in the transmitter diagram in the upper right corner of the screen.

1. Turn on your RC transmitter.
2. Open the app QGroundControl and connect the vehicle by the wire to the computer's usb.
3. Select the **Gear** icon (Vehicle Setup) in the top toolbar and then **Radio** in the sidebar.
4. Press **OK** to start the calibration.
5. Set the transmitter mode radio button that matches your transmitter (this ensures that QGroundControl displays the correct stick positions for you to follow during calibration).
6. Move the sticks to the positions indicated in the text (and on the transmitter image). Press Next when the sticks are in position. Repeat for all positions.

7. When prompted, move all other switches and dials through their full range (you will be able to observe them moving on the Channel Monitor).
8. Press **Next** to save the settings.

#### More Information

- Radio Setup Video (Youtube)

#### Automatic Landing

After the connection of LIDAR to the system via PWM, some parameters of the autopilot must be changed so that it recognizes the sensor. These parameters can be easily changed using QGroundControl. It's them:

- RNG_FND = 5, indicates that the connection occurs via PWM.
- RNDFND_MAX_CM = 4000, represents the maximum distance the sensor is reliable.
- RNDFND_STOP_PIN = 55, indicates the pin connected to the Lidar activation signal. Allows the device to reset the sensor if it stops providing data.
- The parameters RNDFND_SCALING and RNDFND_OFFSET must be adjusted in order to calibrate the sensor (they are usually approximately 0 and 1, respectively).

The sensor can be tested by QGroundControl, where readings can be observed in **Sonar Range** tab. After setting up the sensor, the autopilot will be able to land the aircraft much more quickly and accurately. The landing takes place by sending the command **Land** to the controller, but for it to occur correctly, the landing strip position must be defined and the landing parameters must be adjusted, such as the speed at which the airplane must land .

---

**Note:** Detailed documentation on automatic landings can be found at Automatic Landing.

---

### 1.1.3 Embedded System

An embedded computer must be added to each aircraft in order to communicate with the Pixhawk autopilot and with the computers embedded in the other aircraft through a modem responsible for receiving and transmitting information. The microprocessor must process the information obtained through the modem and pass on instructions to the Pixhawk autopilot.

Embedded computers are digital microprocessed systems in which the computer is entirely encapsulated or dedicated to the system it controls. Unlike general-purpose computers, such as the personal computer, an embedded system performs a set of predefined tasks, usually with specific requirements. The predefined activities to be carried out allow the embedded computers to have a smaller size, weight, price and processing capacity than ordinary computers that perform similar functions.

However, there are disadvantages in its application due to the same reason that make them attractive. Because they are very specific, these devices cannot change their initial function without changing a good part of their software or hardware structure, in some cases these devices do not even have a user interface.

This chapter will explain the first steps and procedures that had to be performed to operate the microprocessor.

#### Embedded Computer

#### Overo WaterStorm

After considering the most specific applications to be performed, the embedded computer chosen was the Overo WaterStorm Computer-On-Module (COM), this embedded system features a DM3730 processor with ARM Cortex-A8 architecture and a processor base clock of up to 1 GHz. In addition, this computer is attached to a Tobi expansion card that adds DVI display, Ethernet, USB Host, USB OTG, USB console, stereo console and a

segment with 40 pin-headers that can be used to the embedded computer. for the most diverse functions, such as PWM modulation, GPIO, power, analog digital conversion and serial communication.


Fig. 3: Gumstix system with Overo WaterSTORM computer, Tobi expansion card and Caspa VL camera

A Caspa VL camera, capable of capturing color images with dimensions of 752 x 480 pixels at a frequency of 60 images per second, was also attached to the system. These three components are produced by the company Gumstix, a manufacturer of hardware specialized in small computers of the type computer-in-module (COM - Computer-On-Module), widely used for embedded systems.

Despite its small size, the combination of Overo COM with the TOBI extension card has the same performance as a full-size Linux computer, larger than other systems of this type found on the market, such as the Raspberry Pi computer.

### Specifications

- **Camera** - Camera Connector: 27-Pin (OMAP ISP)
- **Mechanical** - Length: 58 mm - Width: 17 mm
- **Memory** - Flash Memory (NAND): 512
- **Processor** - Graphics Acceleration: PowerVR SGX530 with OpenGL - Digital Signal Processor: C64x+ - Processor: Texas Instruments OMAP3730 - Processor Architecture: ARM Cortex-A8 - Processor Base Clock: 800 MHz - Processor Max Clock: 1 GHz
- **Power** - Power Management: Texas Instruments TPS65950
- **Storage** - Storage Expansion via microSD Card Slot

**Note:** The complete specifications for the computer, extension card and camera are available in the datasheets below.

- `Datasheet – Overo Waterstorm COM`

- `Manual – Overo Waterstorm COM`
- `Datasheet – Placa de extensão TOBI`
- `Datasheet – Câmera Caspa VL`

### References


- PITA, H. C. Desenvolvimento de sistema de comunicação multiplataforma para veículos aéreos de asa fixa. Faculdade de Tecnologia, Universidade de Brasília, 2018.
- Overo® WaterSTORM COM - Gumstix Store


### Operational System

A digital computer with a certain complexity that requires the management of system resources and such primary functions require an operating system. The kernel is the most important and lowest level part of an operating system, it has the function of defining which program receives attention from the processor, managing memory, creating a file system, managing the communication system, etc.

The first step in using this computer is to create and configure an operating system image that meets the requirements of the project. They are: compatibility with the computer used, *Overo WaterStorm COM*, and support for real-time applications.

An Real Time Operating Systems (**RTOS**) is an operating system destined to the execution of multiple tasks with response time to a pre-defined event (external or internal). There are two approaches to running real-time applications on Linux, using tools that implement a dual kernel or using RTL (Real-time Linux).

### RT-Mag

Initially, it was decided to use the RT-MaG tool as an operating system for the embedded system.

The RT-MaG project (*Real-Time - Marseille Grenoble Project*) is a project developed by Gipsa-Lab (Grenoble, France) and the Institute of Mouvement Sciences (ISM, Marseille, France). The aim of this project is to provide efficient tools for rapid prototyping of robots for research and academic applications. RT-MaG provides a toolbox for Matlab and Simulink to program Linux-COM systems. With the tool, you can easily generate a standalone application in real time from a Simulink model for a robot using a Linux system.

These tools consist of a set of simulink blocks that provide direct access to the computer's inputs and outputs. Simulink models are automatically converted into real-time applications. The use of these tools is completely free. In addition, Gumstix Overo COM is currently fully compatible with the RT-MaG system.

Fig. 4: RT X4-MaG, first robot developed using the RT-Mag system

However, the RT-MaG tool takes on many of the operations necessary for the operation of our system, which makes it impossible to use it in the way it was designed, as a result, too much simplification of the stage could harm future applications. With this tool, it would be impracticable to use the autopilot *MAVLink* communication protocol for communication between devices or aircraft, for example.

Also noteworthy is the outdated documentation, which made it difficult to install the tool's components, such as the Matlab toolbox, which never worked, and the operating system of the embedded computer. The complexity in using the system increased with each step while even the simplest initial steps still did not work properly.

**Note:** More details of the RT-MaG project can be found at Projet RT-MaG.

### Linux

Linux is an operating system popularly used in embedded systems. In addition to providing support for more computational architectures than any other system, it is still lightweight and open source, minimizing implementation costs. Of the different operating systems supported by Gumstix Overo cards, Linux-based systems stand out. Being **Ubuntu** and **Yocto Project** the main ones, besides being recommended by the manufacturer itself.

Fig. 5: Tux, the mascot of Linux

### Projeto Yocto

The Yocto project is an open source collaboration project from the Linux Foundation, whose goal is to produce and provide metadata, tools and processes to help its users create Linux-based distributions for embedded software, regardless of the system architecture.

One element to be highlighted among the components of the Yocto Project is the build system based on the OpenEmbedded architecture, which allows developers to create their own Linux distributions specific to their environment, according to their own needs. These Project Yocto configurations provided by hardware vendors generally include kernel configurations, kernel modules, kernel firmware and base system packages.

Another important tool of the Yocto Project is the Poky reference build system. It contains the BitBake tool, which allows cross-compilation regardless of the platform. In addition, BitBake manages all configuration files and data, and tries to reduce compilation time using all available processing resources.

Unfortunately, with the wide versatility of the Yocto Project, the complexity of the process of creating a customized distribution is also increasing.

**Note:** More details of the Yocto project can be found at yoctoproject.org.

### Ubuntu

Ubuntu is an open source operating system, developed from the Linux kernel, based on Debian. Ubuntu is developed by Canonical and the community in a model of meritocratic governance. Canonical provides free security updates and support for each version of Ubuntu. All versions are available at no cost.

The advantage of using the Ubuntu system is that it is an operating system from the very popular Linux core that already contains several software that may be useful for some future applications, it contains, for example, a compiler which facilitates the creation and execution of simple codes for rapid tests.

The disadvantage of using this operating system is that many unnecessary parallel tasks can be performed that decrease the specificity and performance of the embedded computer.

**Note:** More details about Ubuntu can be found at ubuntu.com.

### Chosen System

We even installed the RT-Mag on the embedded system, however, due to complications after the installation of the operating system, it was decided not to use this tool anymore.

It was then decided to use the core offered by the Yocto Project as it is specific to the embedded computer model. Choosing to install the Ubuntu 15.04 system on one of the computers in order to analyze the differences between the two operating systems and perform tests.

However, the Ubuntu system, despite being a stable version and adapted to the system in question, presented unresolved errors in the installation process, making it impossible to install the system on an SD card.

### References


- PITA, H. C. Desenvolvimento de sistema de comunicação multiplataforma para veículos aéreos de asa fixa. Faculdade de Tecnologia, Universidade de Brasília, 2018.
- ROCHA, E. M. C. Desenvolvimento de um sistema com veículos aéreos não-tripulados autônomos. Faculdade de Tecnologia, Universidade de Brasília, 2017.
- Phanuel Hieber. Yocto Project on the Gumstix Overo Board. Technische Universität München.
- RT-MaG Project - gipsa-lab.fr
- Yocto Project - yoctoproject.org


### Installing the Operating System

The installation of the operating system is not a trivial task, in addition there is a shortage of detailed and complete documentation explaining how to install the operating system on the embedded computer, soon the procedures necessary for installing an operating system will be documented in this section. In the current phase of the work, we installed both systems, so that we can decide later which of the two systems will be best for our application.

**Note:** Official tutorials can be found on the Gumstix website and on the GitHub repositories of the Yocto and Ubuntu project for Gumstix products.

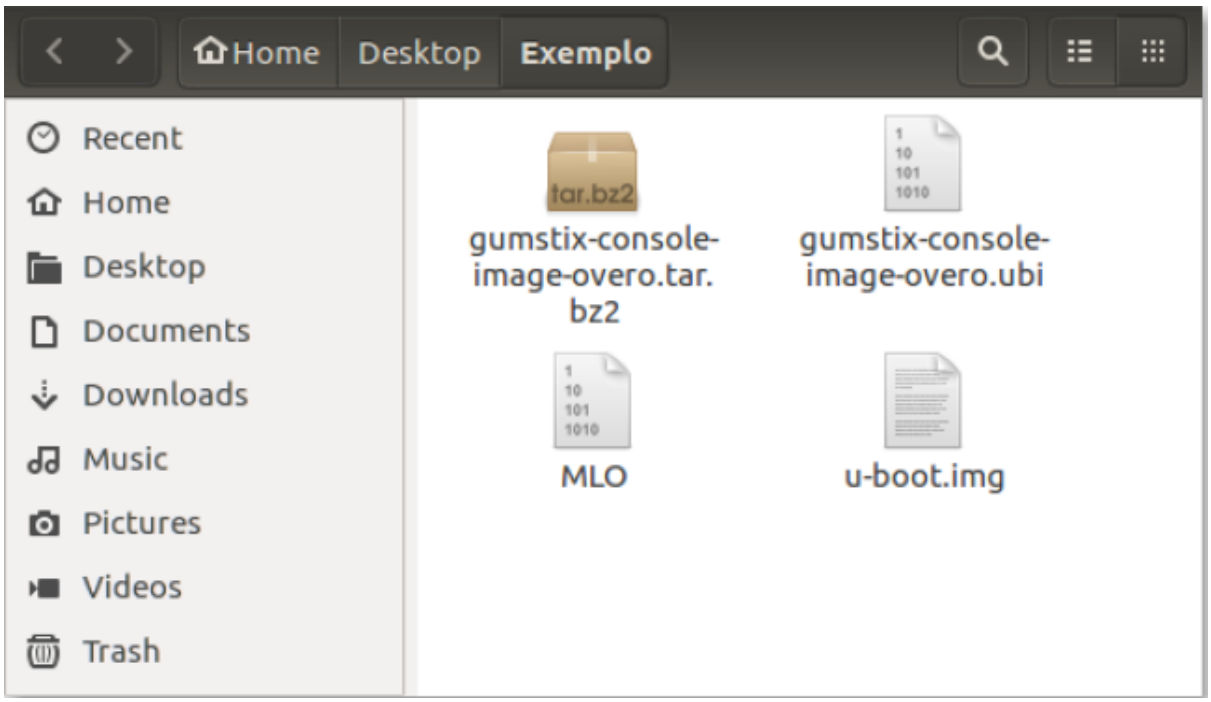
### Obtaining OS images

Essentially, the device only needs to run a small program, usually located in a non-volatile memory of the type *Read-Only Memory* (ROM), to access another non-volatile memory device that stores the operating system, and load the operating system on volatile quick access memory or *Random Access Memory* (RAM) where it can be executed. In more robust systems, there is, in fact, a chain of these small programs, called *bootloaders*, where a first stage executes a second stage that loads more complex programs and, in turn, executes a third stage and so on until the operating system is fully loaded and ready to run on its own.

There are two methods for obtaining the operating systems for Gumstix Overo. The first method is to download a precompiled image directly from Gumstix. The second method is to build the image on your computer yourself. Creating the operating system image manually has additional benefits, such as customizing or adding additional binary packages to your base image. The possibility of personalization will be very important in the development of the project.

### Obtaining images from the Yocto Project

In order to perform the Yocto Project operating system initiation process on the Gumstix embedded computer, we need three specific files, they are the first stage of the initiation system, the MLO (*Minimal Loader*) file, the second stage of the initiation system, the file *u-boot.img* (the acronym comes from *Universal Bootloader*), and the image of the system, which in our case will be Yocto 1.8.2 with Linux kernel.




The figure shows an example of the files described in the previous paragraph, note that, in this case, there is also a compressed folder that contains the root files of the operating system. The simplest way found to obtain these files and the operating system image is by following the steps in the README.md file of the Yocto project repository for Gumstix products. The advantage of using this method instead of simply obtaining the ready image of the operating system is that if necessary, we can modify it.

This tutorial explains how to manually build the image of the Yocto system and perform all procedures through command lines of the Linux terminal, with an emphasis on **Ubuntu 14.04 (LTS)**. However, to perform this step it is highly recommended to comply with the requirements indicated by the Yocto project.

### System Requirements

---

**Note:** For more information regarding system requirements, see System Requirements - Yocto Project Reference Manual.

---

The development of projects in the Yocto Project environment requires that some requirements are met, they are:

- A system with a minimum of 25 GB of free disk space running a supported Linux distribution. If the host system supports multiple cores and threads, you can configure the Yocto Project build system to significantly decrease the time required to build images.
- Appropriate packages installed on the system used to perform the builds.
- A distribution of the Yocto Project.

Project Yocto is currently supported on the following Linux distributions.

- Ubuntu 12.04 (LTS)
- Ubuntu 13.10
- Ubuntu 14.04 (LTS)
- Fedora release 19 (Schrödinger's Cat)
- Fedora release 20 (Heisenbug)
- CentOS release 6.4
- CentOS release 6.5
- Debian GNU/Linux 7.0 (Wheezy)
- Debian GNU/Linux 7.1 (Wheezy)
- Debian GNU/Linux 7.2 (Wheezy)
- Debian GNU/Linux 7.3 (Wheezy)
- Debian GNU/Linux 7.4 (Wheezy)
- Debian GNU/Linux 7.5 (Wheezy)
- Debian GNU/Linux 7.6 (Wheezy)
- openSUSE 12.2
- openSUSE 12.3
- openSUSE 13.1

---

**Note:** For a more detailed list of distributions that support the Yocto Project, see the Supported Linux Distributions section in the Yocto Project Reference Manual.

---

To build the operating system image, the build system must have the following versions of *software* Git, tar and Python.

- Git 1.8.3.1 or greater
- tar 1.27 or greater
- Python 3.4.0 or greater

---

**Note:** See the Required Git, tar, and Python Versions section in the Yocto Project Reference Manual for information.

---

In addition, it is recommended to update the Linux repositories. To do so, in the case of Ubuntu distribution, just run the following command:

```
$ sudo apt-get update && sudo apt-get upgrade
```

It is also necessary to install the essential host packages to build the image. The following command installs host packages based on systems with Ubuntu distribution.

```
$ sudo apt-get install gawk wget git-core diffstat unzip texinfo␣
↪gcc-multilib build-essential chrpath socat libsdl1.2-dev xterm␣
↪curl
```

---

**Note:** To install host packages on other supported Linux distributions, see the Required Packages for the Build Host section in Yocto Project Reference Manual.

---

### Configuring the Image

**Note:** The operating system used to test the commands was Ubuntu 14.04 (LTS).

Linhas de comando Linux para obtenção e montagem da imagem.

1. **Installing the repository**

To download Yocto images, we first need to install the **repo** command. In summary, the repo is basically a git wrapper, which provides a simple way to group several different git repositories into a single project. If you are interested in more information about the **repo** command, go to repo - gerrit.googlesource.com.

Download the scripts from the repository

```
$ curl http://commondatastorage.googleapis.com/git-repo-downloads/
↪repo > repo
```

Make files executable

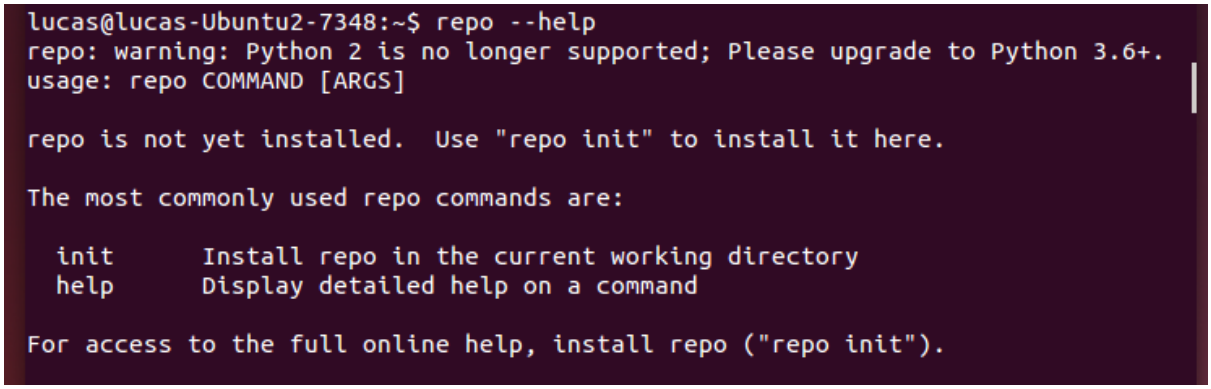
```
$ chmod a+x repo
```

Move files to the system path

```
$ sudo mv repo /usr/local/bin/
```

If everything goes well, a message similar to the image should appear when executing the following command. This command is not mandatory.

```
$ repo --help
```




2. **Creating a local repository**

Create a directory for the files and change the execution directory for the new repository.

```
$ mkdir yocto
$ cd yocto
```

Now with the repository already installed, we will download all the Yocto settings for our project. The **init** command can take some time, as it downloads all the git repositories associated with the project. The command **-b** specifies the branch to be used and the command **fido** selects the most stable branch of the repository.

```
$ repo init -u git://github.com/gumstix/yocto-manifest.git -b fido
```

A successful initialization will end with a message stating that **.repo** has been initialized in your working directory. Your directory should now contain a * .repo * folder where the repository control files are stored, but there is no need to open the directory.

```
lucas@lucas-Ubuntu2-7348:~/yocto$ repo init -u git://github.com/gumstix/yocto-manifest.git -b fido
repo: warning: Python 2 is no longer supported; Please upgrade to Python 3.6+.
Downloading Repo source from https://gerrit.googlesource.com/git-repo
remote: Finding sources: 100% (9/9)
remote: Total 9 (delta 0), reused 9 (delta 0)
Unpacking objects: 100% (9/9), done.
repo: Updating release signing keys to keyset ver 2.3
repo: warning: Python 2 is no longer supported; Please upgrade to Python 3.6+.
Downloading manifest from git://github.com/gumstix/yocto-manifest.git
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 534 (delta 6), reused 19 (delta 5), pack-reused 514

Your identity is: LucasCampos98 <[redacted]>
If you want to change this, please re-run 'repo init' with --config-name

Testing colorized output (for 'repo diff', 'repo status'):
  black     red       green     yellow    blue      magenta   cyan      white
  bold      dim       ul        reverse
Enable color display in this user account (y/N)? y

repo has been initialized in /home/lucas/yocto
```

3. **Downloading the files**

The following command is used to ensure that all of your repositories are up to date and is useful for updating your Yocto settings if you do a build later.

```
$ repo sync
```

**Note:** This step can take more than 20 minutes, depending on your internet connection.

Force all temporary files to be written to permanent devices using the command:

```
$ sync
```

4. **Starting the Yocto Project Build Environment**

**Warning:** If, for any reason, you cancel the activity before completing the Yocto compilation, you will need to execute this command each time before proceeding to the next steps. Keep in mind that this also applies to future builds.

Now that we have our basic Yocto settings, we will enter our build environment. Using the following command, we will copy the default configuration information into the **poky/build/conf** directory and set some environment variables for the image assembly system.

```
$ export TEMPLATECONF=meta-gumstix-extras/conf
$ source ./poky/oe-init-build-env
```

**Note:** This configuration directory is not under revision control, so you can edit these configuration files for your specific installation.

5. **Creating the image**

The Yocto project uses bitbake to compile the Yocto Linux image. Bitbake basically compiles only the OS, kernel, modules and all packages included in the target Linux OS.

**Tip:** (**OPCIONAL**) If you are familiar with compiling via make, you can speed up the compilation process by telling bitbake to compile with more threads. This step is not necessary, but if you are compiling on a system with a high-end CPU with many cores, it will speed up the compilation time. For example:

```
$ export PARALLEL_MAKE="-j 8"
```

The number "8" indicates the number of nuclei to be used in the matching. **It is worth mentioning that you should not specify a value -j greater than the amount of CPU cores present in your construction machine**.

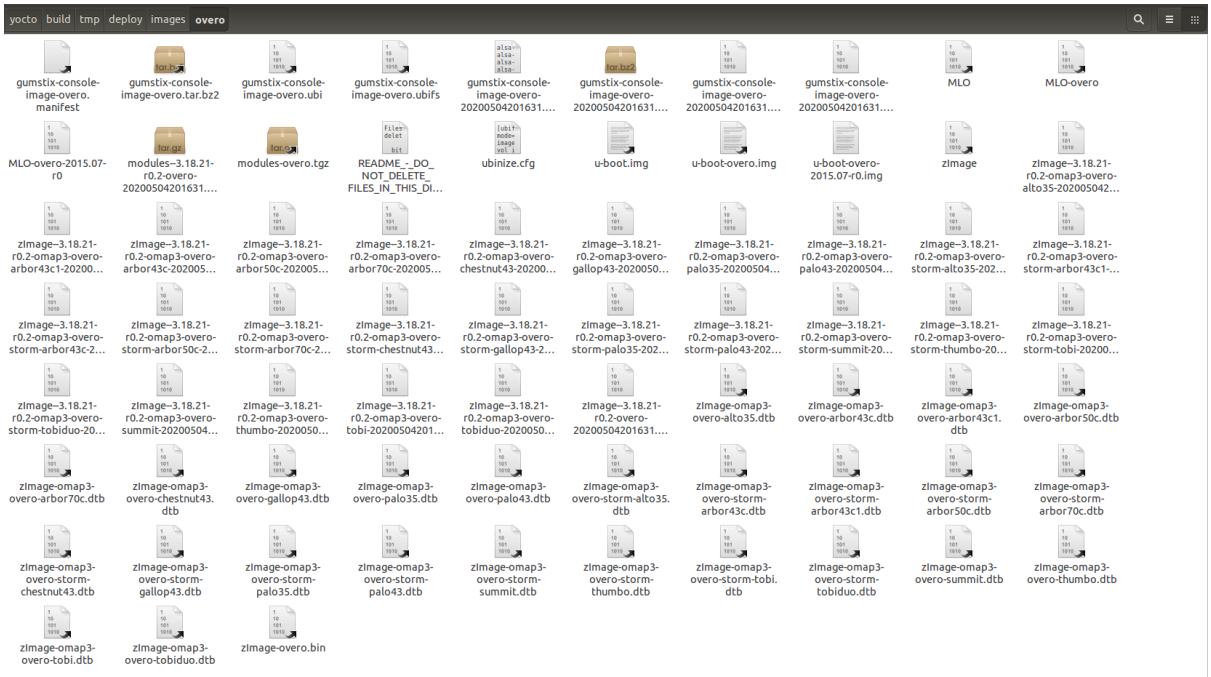
So, to download the source codes and compile the system images, run:

```
$ bitbake gumstix-console-image
```

**Note:** This process downloads several gigabytes of code and then makes a huge build. So, make sure you have at least 25GB of free space. This step may take a day or more to create the image, depending on your internet connection. Don't worry, it's just the first build that takes a while.

After completing the execution of all commands, it is recommended to check the folder **yocto/build/tmp/deploy/images/overo**, this folder must contain binary kernel files and bootloaders and root directory files in the format .tar.

The figure below shows an example of the contents of the described folder, this folder must be like the one obtained after performing the previous procedures.




In the figure we can find both the necessary bootloaders described previously and the binary (.ubi) and files in the root directory of some versions of the Yocto project.

**Warning:** Possible causes of failures are probably related to missing or outdated software, unsupported operating system or lack of free space.

#### References


- PITA, H. C. Desenvolvimento de sistema de comunicação multiplataforma para veículos aéreos de asa fixa. Faculdade de Tecnologia, Universidade de Brasília, 2018.
- Gumstix Repo Manifests for the Yocto Project Build System - github.com
- Yocto Project Quick Start - yoctoproject.org
- Yocto Project Reference Manual - yoctoproject.org
- Building Yocto Linux Images for the Gumstix Overo - hackgnar.com


### Preparing the Memory Card

Once the image of the operating system is obtained, we can transfer the files to the embedded computer in order to connect it. This task will be performed using an SD card that will act as the hard disk of the embedded computer. Therefore, the SD card will contain both the programs needed for boot, which will be used only when starting the computer, and the other programs can be used at any time and will make constant changes to the SD card. So, the best way to deal with this division is to partition the SD card into two partitions that will be called boot and rootfs.

The file management system defines the method that the operating system will use to store files and their information, or file metadata, in memory spaces, such as name, occupied memory space, dates of changes and last accesses. There is a wide variety of file management systems with the most diverse complexities. But what we may need in this work and in future works is the "FAT" system, an old system generally used in media and, usually, universal. "Ext" is a system designed specifically for Linux and it is not possible to access it from another operating system without a program for this purpose.

This is a very common procedure and there are numerous ways to do it, however, here we will use Linux's own disk manager to perform partitioning, as it is a simple, intuitive tool and allows for future changes without major difficulties. It is worth mentioning that this is not the method indicated by the manufacturer, since the procedures recommended by them on the page Create Bootable MicroSD Card presented the most diverse errors, however, the results obtained using the procedures of the topic below are the same.

#### Partitioning the SD Card

This guide describes the partitioning process, using a Linux system, of a microSD card in two parts, called **boot** and **rootfs** in order to generate a bootable SD card. The procedure described below is performed using the disk manager of Ubuntu itself, it is not necessary to install new *software*.

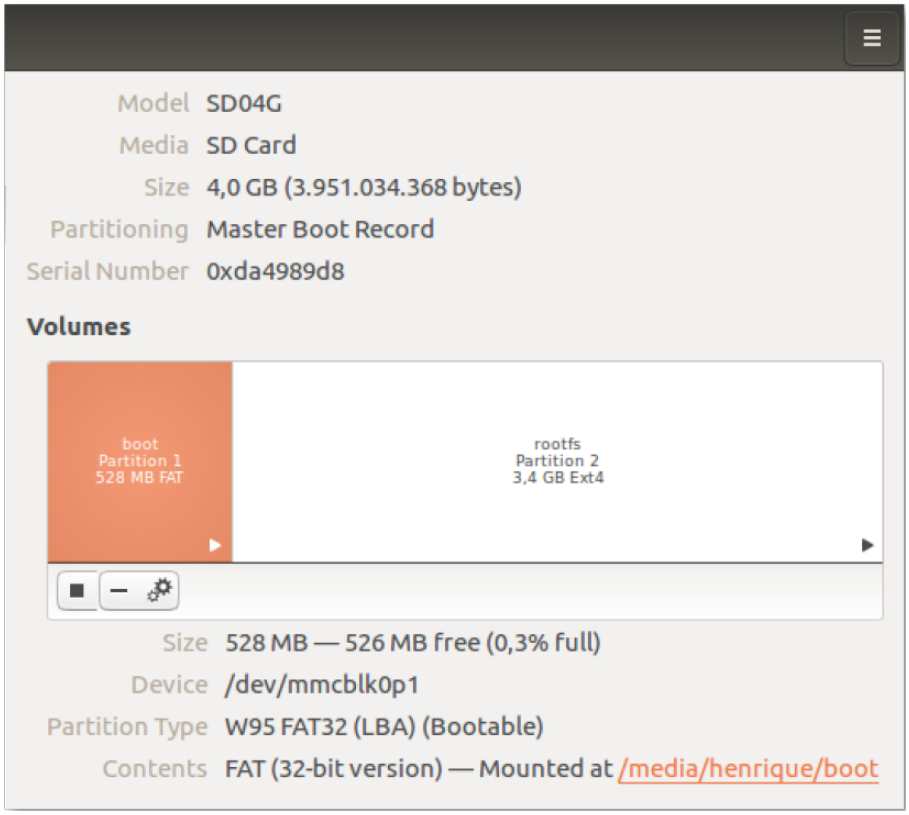
Usually, the microSD card is configured in a single partition formatted in the Windows FAT standard, a typical configuration found in cards purchased at retail. However, here we will partition the microSD card into two parts, which will be called **boot** and **rootfs**, with the file management system of the **boot** partition "VFAT" and the **rootfs** partition "ext4".

The figure below shows an example of a memory card with the partitions already defined, mounted and containing the operating system of the embedded computer. In the example the SD card has a total of 4 GB, however, for the Yocto project, a 2 GB memory card should be enough.

#### Procedures

> **Warning:** The operating system version used in the activities was Ubuntu 20.04 (LTS), however the commands are the same for older versions of Ubuntu, starting with Ubuntu 14.04 (LTS). The procedures may differ depending on the version and distribution of Linux being used.

1. Insert the microSD card or an adapter with it into an available port on your Linux computer.




2. Search your computer for an application called **Disks** and launch it. Upon opening, the application will display the memory devices connected to the computer.

3. In the **Disks** tab, select the microSD card you want to partition.

4. Click "**Unmount the file system**" below **Volumes** to enable modifications to the microSD card.

5. To create new partitions in different formats it is recommended to delete the partition on your microSD card, for this, click on "**Delete partition**".

> **Warning:** This step will format your microSD card, so all data on it will be permanently deleted.

6. Click "**Create a new partition** to create the first partition.

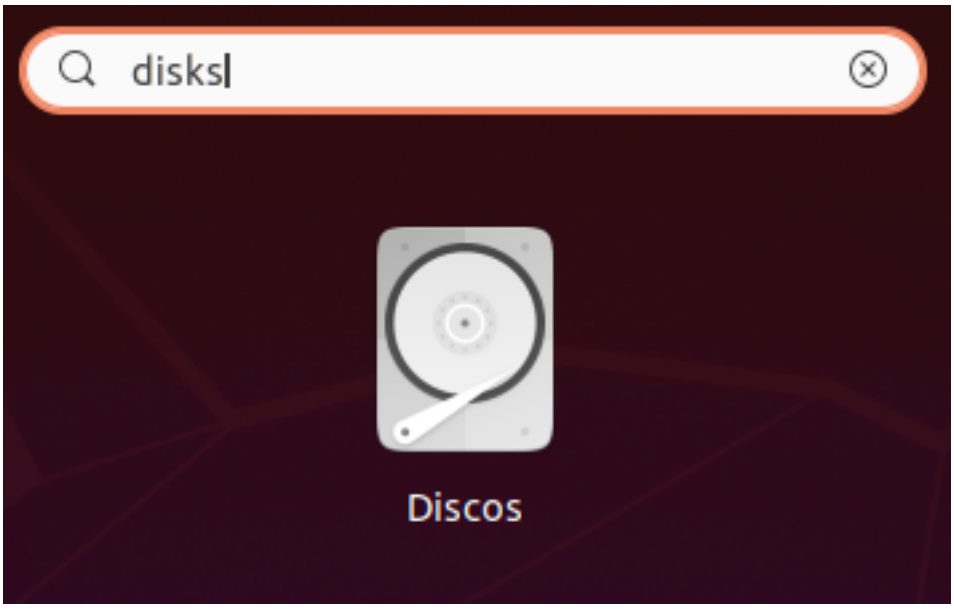
   This partition will be named "**boot**", will have a size of 528MB and will be configured with the file management type "FAT", as shown below. After configuring, click on "**Create**" to generate this new partition.

   Then, go to **More Actions> Edit partition**, set the **Partition type** to "**W95 FAT32 (LBA)**" and activate the option **Startable** to determine that this partition is where the operating system should be loaded.

---

**Tip:** In this example, 528 MB has been reserved for the boot partition, however, less than 100 MB is used for boot. Therefore, if there is a lack of space for data storage in the future, it will be possible to expand the roots partition by redoing this division.

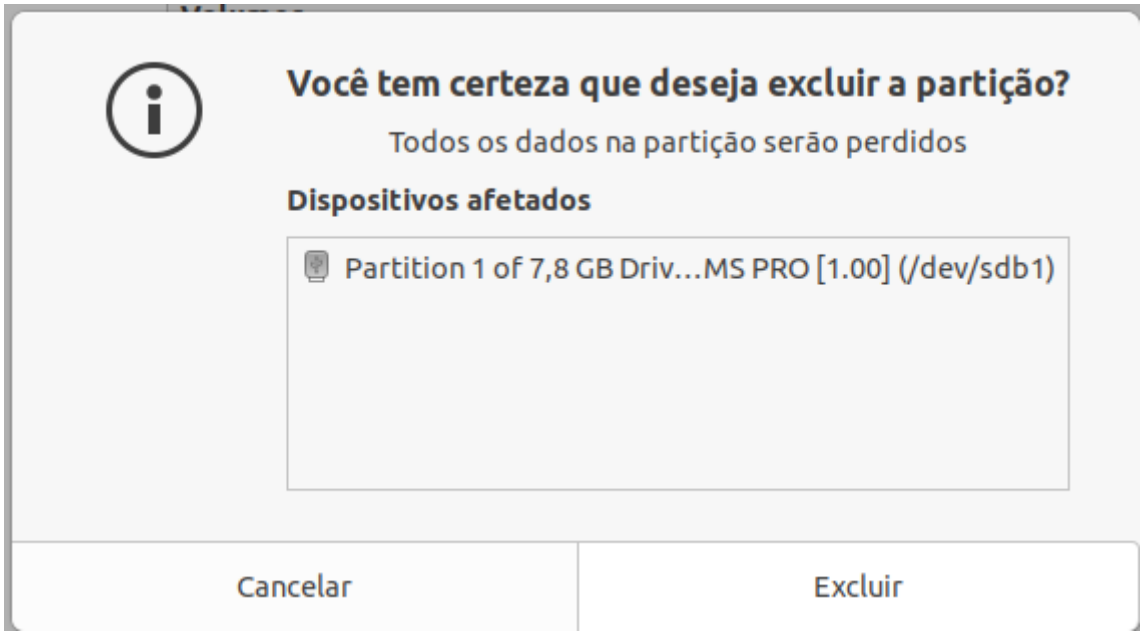
---

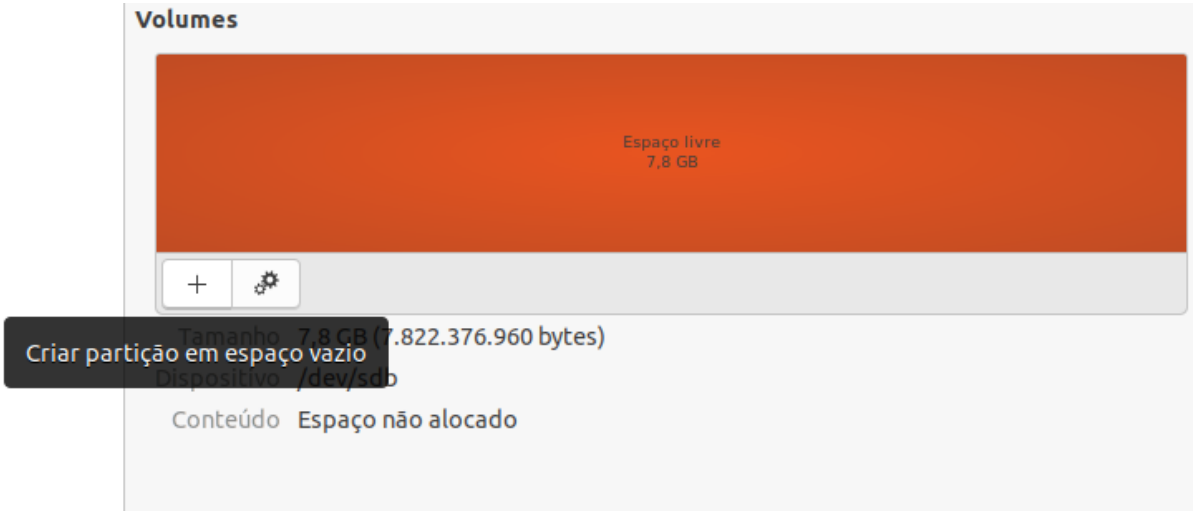

disks
Discos

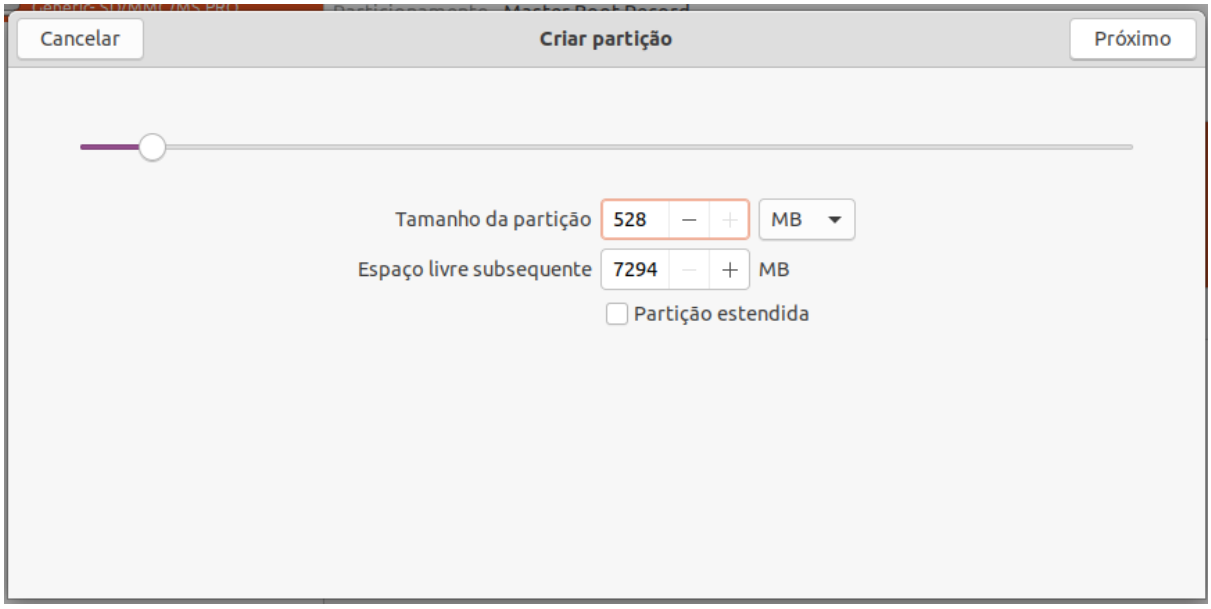

Discos
7,8 GB Drive
/dev/sdb
500 GB Hard Disk
ST500LM000-1EJ162
7,8 GB Drive
Generic- SD/MMC/MS PRO
Modelo Generic- SD/MMC/MS PRO (1.00)
Tamanho 7,8 GB (7.822.376.960 bytes)
Particionamento Master Boot Record
Número de série 20121112761000000
Volumes
SD 8GB
Partição 1
7,8 GB FAT
Tamanho 7,8 GB — 7,8 GB livres (0,2% ocupados)
Dispositivo /dev/sdb1
UUID 51AE-B32C
Tipo de partição W95 FAT32 (LBA)
Conteúdo FAT (32-bit version) — Montado em /media/lucas/SD 8GB

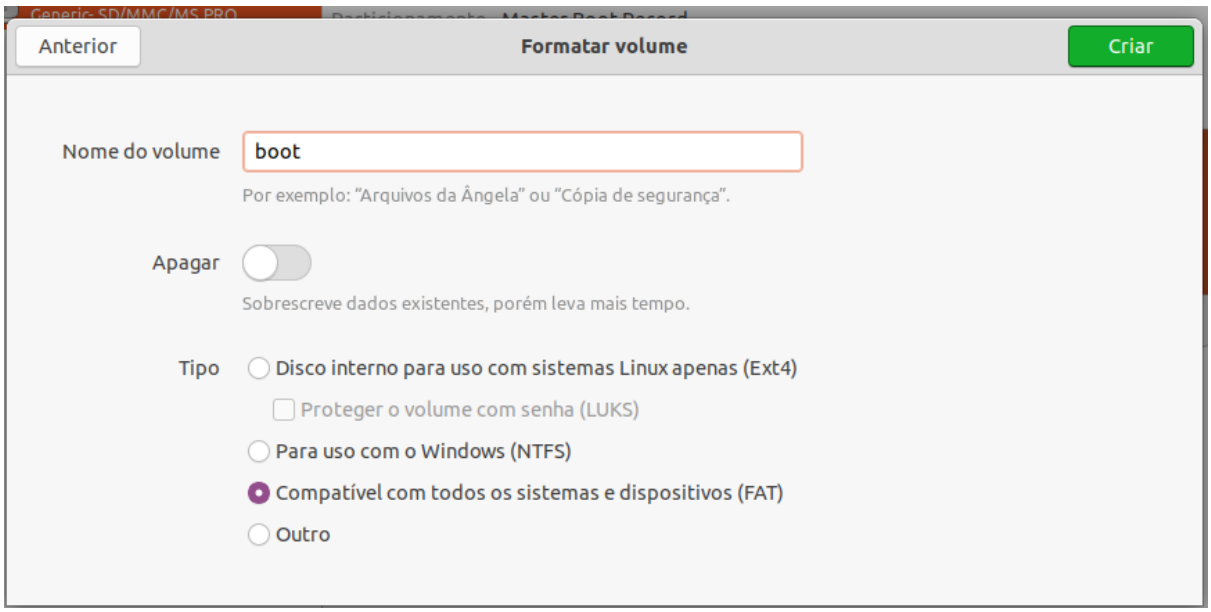

Volumes
SD 8GB
Partição 1
7,8 GB FAT
Desmontar partição selecionada
Tamanho 7,8 GB — 7,8 GB livres (0,2% ocupados)
Dispositivo /dev/sdb1
UUID 51AE-B32C
Tipo de partição W95 FAT32 (LBA)
Conteúdo FAT (32-bit version) — Montado em /media/lucas/SD 8GB

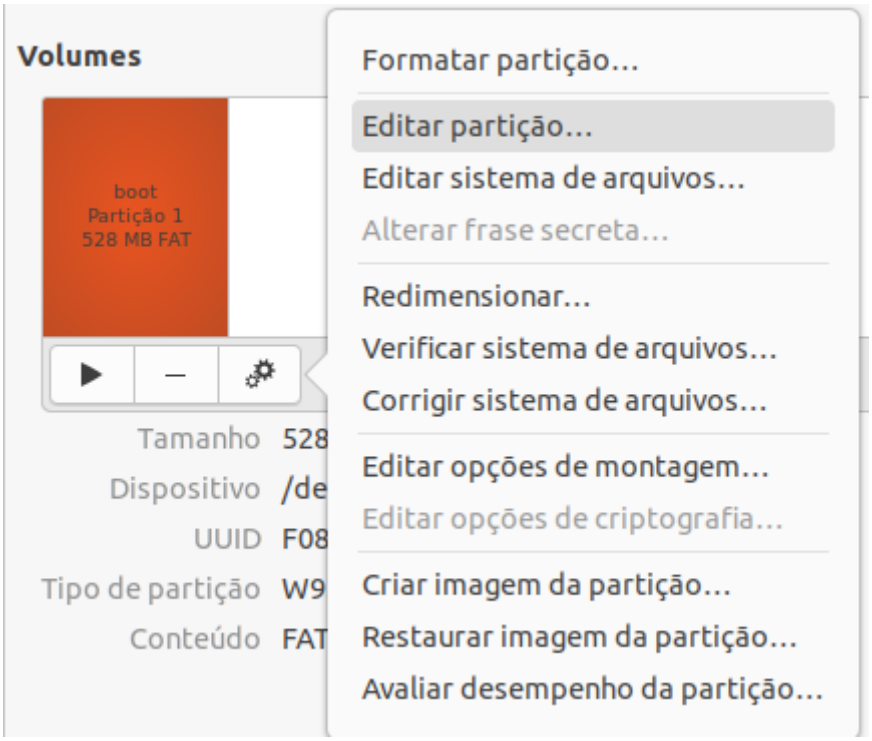

Volumes
SD 8GB
Partição 1
7,8 GB FAT
Excluir partição selecionada
Tamanho 7,8 GB (7.821.328.384 bytes)
Dispositivo /dev/sdb1
UUID 51AE-B32C
Tipo de partição W95 FAT32 (LBA)
Conteúdo FAT (32-bit version) — Não montado

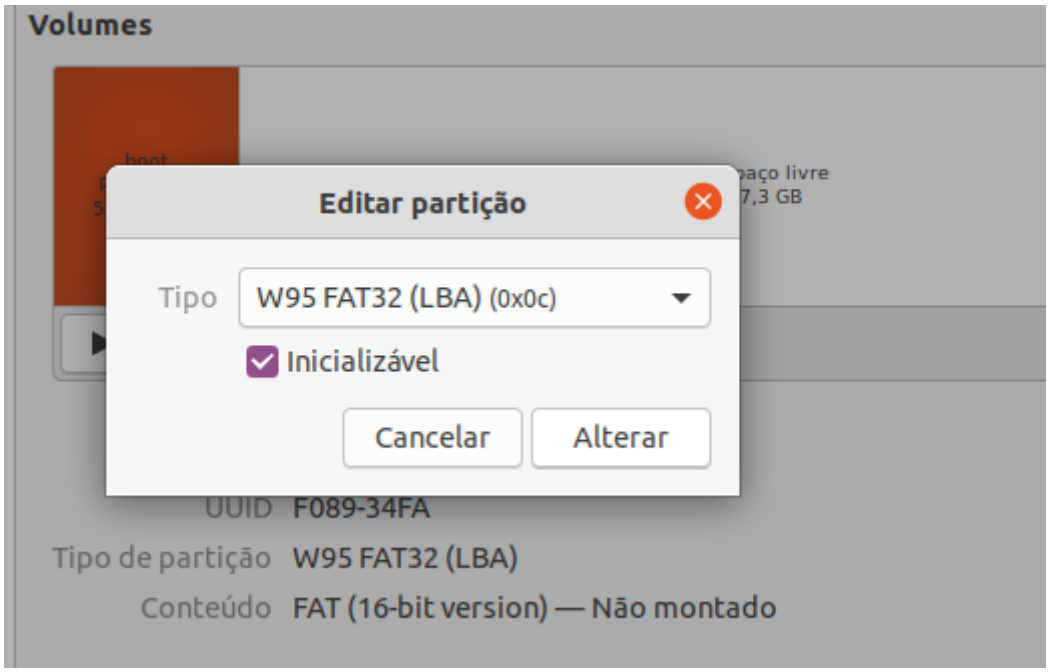

Você tem certeza que deseja excluir a partição?
Todos os dados na partição serão perdidos
Dispositivos afetados
Partition 1 of 7,8 GB Driv…MS PRO [1.00] (/dev/sdb1)
Cancelar
Excluir


Volumes
Espaço livre
7,8 GB
Criar partição em espaço vazio
Tamanho 7,8 GB (7.822.376.960 bytes)
Dispositivo /dev/sdb
Conteúdo Espaço não alocado

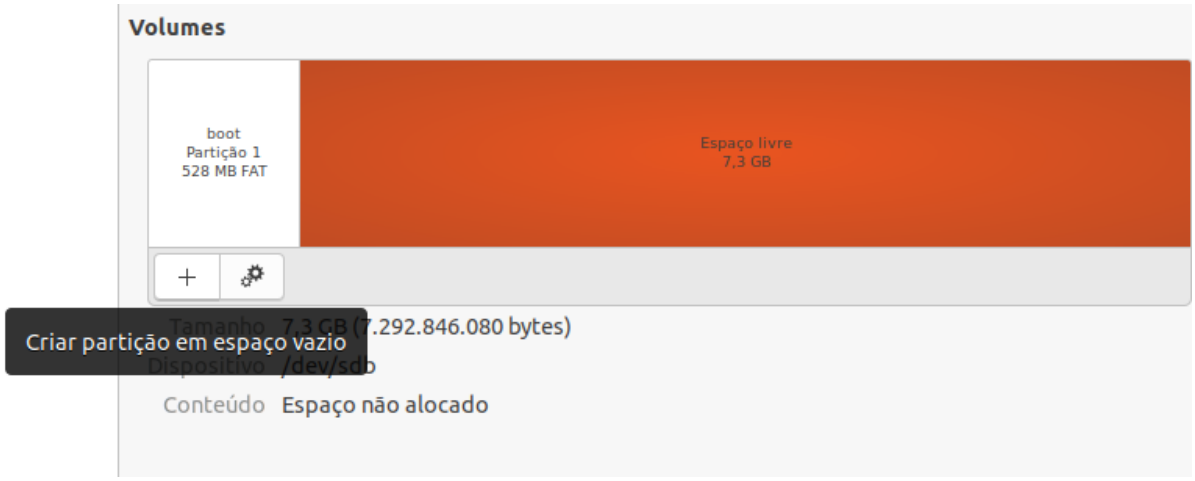

Cancelar
Criar partição
Próximo
Tamanho da partição
528
MB
Espaço livre subsequente
7294
MB
Partição estendida


Anterior
Formatar volume
Criar
Nome do volume
boot
Por exemplo: "Arquivos da Ângela" ou "Cópia de segurança".
Apagar
Sobrescreve dados existentes, porém leva mais tempo.
Tipo
Disco interno para uso com sistemas Linux apenas (Ext4)
Proteger o volume com senha (LUKS)
Para uso com o Windows (NTFS)
Compatível com todos os sistemas e dispositivos (FAT)
Outro


Volumes
boot
Partição 1
528 MB FAT
Tamanho 528
Dispositivo /de
UUID F08
Tipo de partição W9
Conteúdo FAT
Formatar partição…
Editar partição…
Editar sistema de arquivos…
Alterar frase secreta…
Redimensionar…
Verificar sistema de arquivos…
Corrigir sistema de arquivos…
Editar opções de montagem…
Editar opções de criptografia…
Criar imagem da partição…
Restaurar imagem da partição…
Avaliar desempenho da partição…


Volumes
Editar partição
Tipo
W95 FAT32 (LBA) (0x0c)
Inicializável
Cancelar
Alterar
UUID F089-34FA
Tipo de partição W95 FAT32 (LBA)
Conteúdo FAT (16-bit version) — Não montado

7. Now we are going to create the second partition, called **rootfs**. Therefore, select the free space of the SD card and click **Create partition in empty space**.



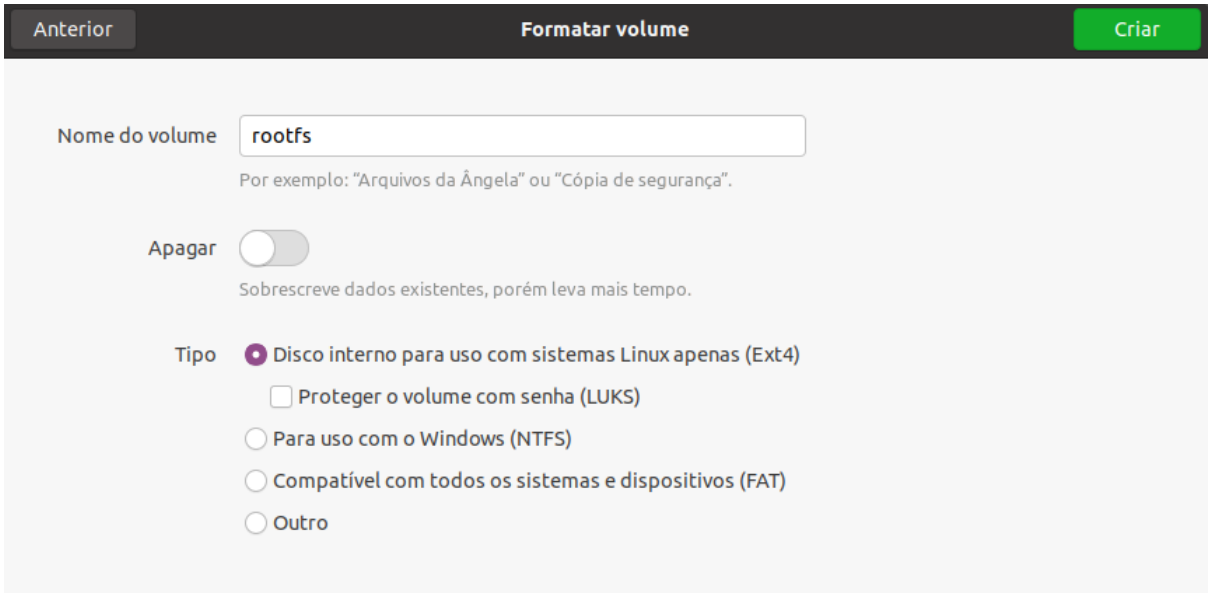

This partition will be named "**rootfs**" and we will allocate all the remaining memory on the SD card to it. This partition will be configured with the "Ext4" file management type, the default file system for current GNU / Linux systems. After configuring, click on "**Create**" to generate this new partition.


On a successful run, the result will be similar to the figure below, where the procedures were applied to an 8GB card.

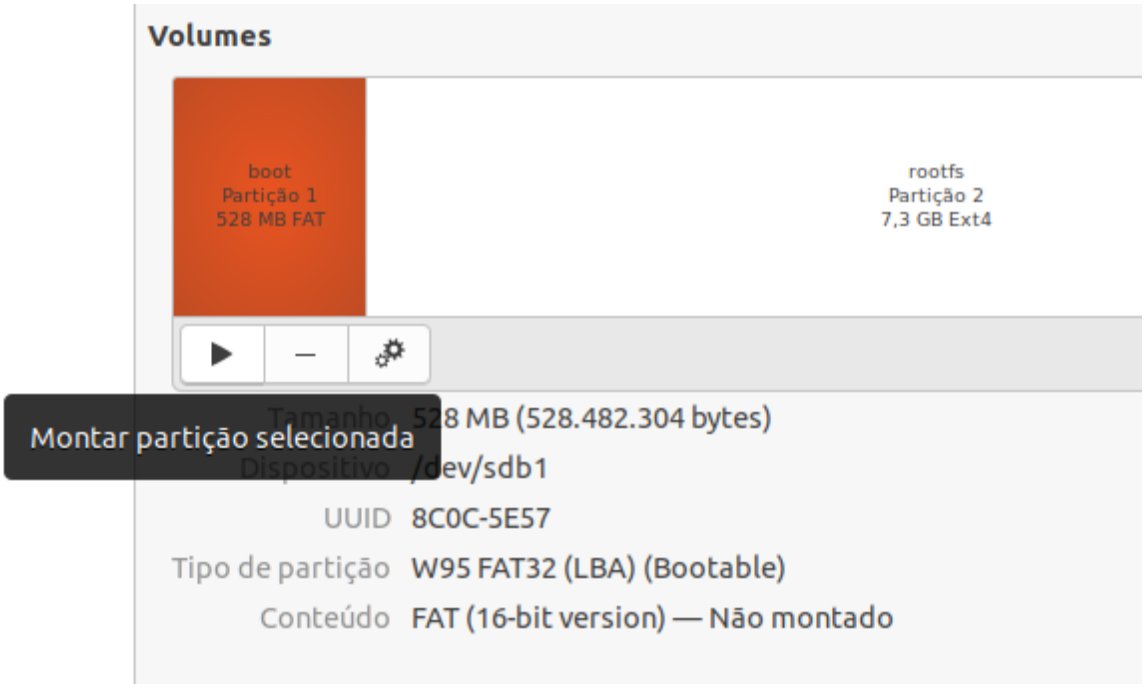
8. (Optional) To reassemble the partitions, just select the partition and click **Mount the selected partition**. This tool will automatically mount the selected partition to the file system /media/<User_Name>

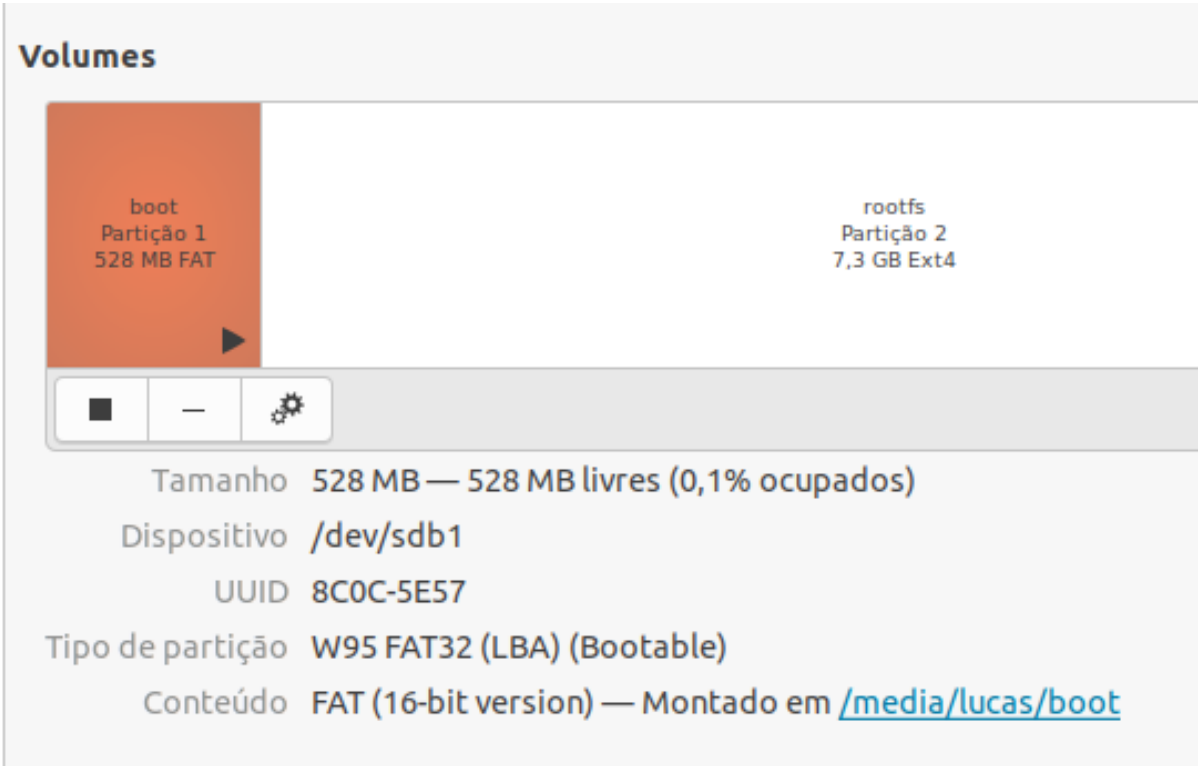
### References

- Create Bootable MicroSD Card - gumstix.com
- Script - Make 2 Partition SD Card - github.com
- How to Make 2 Partition SD Card - Texas Instruments Processors Wiki


Anterior
Formatar volume
Criar
Nome do volume
rootfs
Por exemplo: "Arquivos da Ângela" ou "Cópia de segurança".
Apagar
Sobrescreve dados existentes, porém leva mais tempo.
Tipo
Disco interno para uso com sistemas Linux apenas (Ext4)
Proteger o volume com senha (LUKS)
Para uso com o Windows (NTFS)
Compatível com todos os sistemas e dispositivos (FAT)
Outro


Modelo Generic- SD/MMC/MS PRO (1.00)
Tamanho 7,8 GB (7.822.376.960 bytes)
Particionamento Master Boot Record
Número de série 20121112761000000
Volumes
boot
Partição 1
528 MB FAT
rootfs
Partição 2
7,3 GB Ext4
Tamanho 528 MB (528.482.304 bytes)
Dispositivo /dev/sdb1
UUID F089-34FA
Tipo de partição W95 FAT32 (LBA) (Bootable)
Conteúdo FAT (16-bit version) — Não montado


Volumes
boot
Partição 1
528 MB FAT
rootfs
Partição 2
7,3 GB Ext4
Montar partição selecionada
Tamanho 528 MB (528.482.304 bytes)
Dispositivo /dev/sdb1
UUID 8C0C-5E57
Tipo de partição W95 FAT32 (LBA) (Bootable)
Conteúdo FAT (16-bit version) — Não montado


Volumes
boot
Partição 1
528 MB FAT
rootfs
Partição 2
7,3 GB Ext4
Tamanho 528 MB — 528 MB livres (0,1% ocupados)
Dispositivo /dev/sdb1
UUID 8C0C-5E57
Tipo de partição W95 FAT32 (LBA) (Bootable)
Conteúdo FAT (16-bit version) — Montado em /media/lucas/boot

- PITA, H. C. Desenvolvimento de sistema de comunicação multiplataforma para veículos aéreos de asa fixa. Faculdade de Tecnologia, Universidade de Brasília, 2018.

### Writing the image on the MicroSD Card

After dividing the SD card, we can proceed with the installation of the system by mounting its partitions and copying the files obtained previously, the two bootloaders files, to the folder where the boot partition was mounted and extracting the system directories to the folder where the rootfs partition has been mounted.

After that, the procedure for mounting a memory partition is an activity of the operating system to ensure that the transfer of information will be done correctly, basically the connected device is read in its entirety to identify the files stored in it and where new information can be written to without overlapping data. However more important than mounting the partition is to unmount the partition before disconnecting the peripheral, as it ensures that no write activity on the partition is taking place at the time the device is removed. This procedure also ensures that all requested changes have been made on the peripheral and are not saved in temporary files or system buffers.

The procedure described below is based on the manufacturer's recommendations and is specific for installing the Yocto Project system on Gumstix Overo devices.

**Tip:** Remember to unmount the partitions before removing the SD card.

### Mounting the Partitions

The procedure for mounting a memory partition is an activity of the operating system to ensure that the transfer of information will be done correctly, basically the connected device is read in its entirety to identify the files stored in it and where new information can be written to without overlapping data.

To do this, open the terminal and enter the commands below:

```
# Comando para montar a partição boot
$ sudo mount t vfat /dev/<mmcblk0p>1/  media/<UserName>/boot

# Comando para montar a partição rootfs
$ sudo mount -t ext4 /dev/<mmcblk0p>2/  media/<UserName>/rootfs
```

**Note:** The two names in <> must be changed. About *UserName* should be changed to the login name of the machine in use and about *mmcblk0p* should be changed to the name of the file that the system automatically creates when it recognizes the card for the first time and that file will be present in /dev with name similar to *mmcblk0p*. To display the media devices connected to your computer, run the command `df -hT`.

**Tip:** **Alternatively**, it is possible to carry out the process of assembling the microSD card partitions following the **procedure 8** of the topic Procedures - Partitioning the SD Card. The expected result is the same.

### Copy and Extraction Files

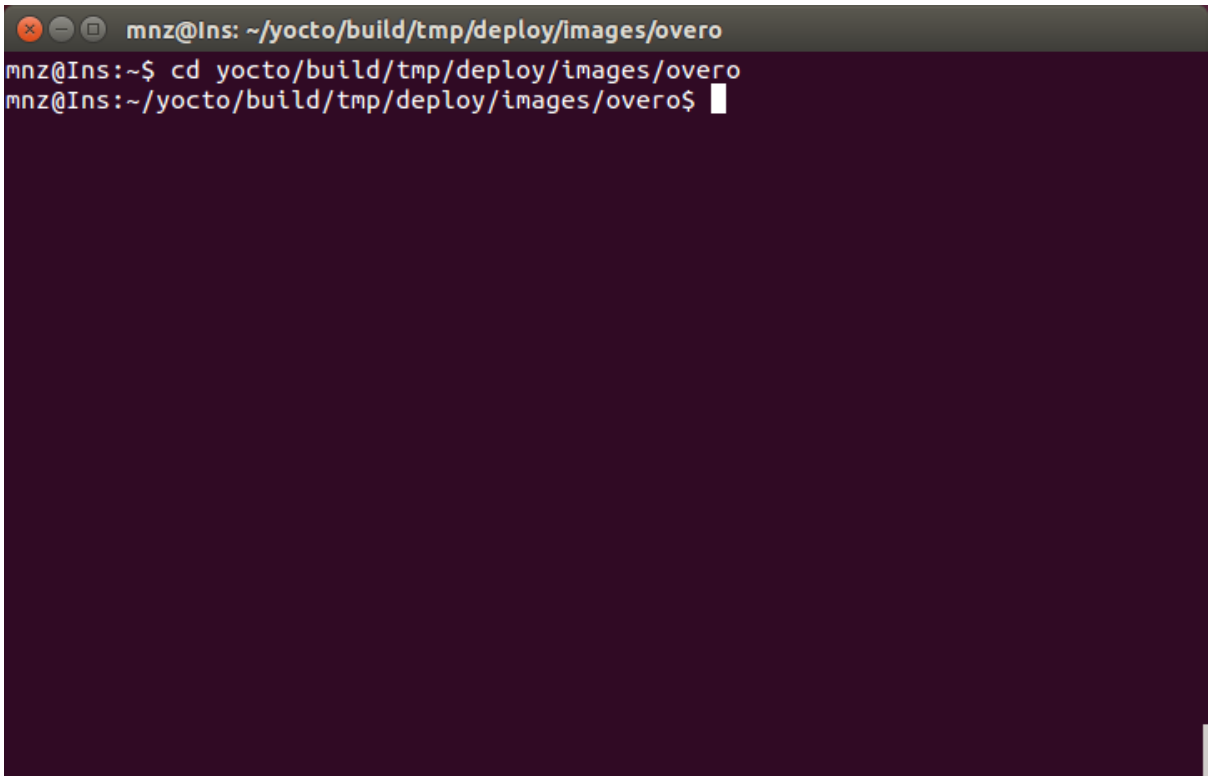
### Copy

Two files named MLO and u-boot.img must be copied to the boot partition of the memory card. For this, the following commands must be executed inside the folder that contains the images or correctly specifying the location of the files. We recommend that you enter the images folder, as it is possible to write the image more efficiently.

The folder containing the image files is located in `/yocto/build/tmp/deploy/images/overo` and can be accessed by the command:

```
$ cd /yocto/build/tmp/deploy/images/overo
```

After these commands, the terminal will be like the image below:




After the previous steps, enter the following commands:

```
$ sudo cp MLO /media/<UserName>/boot/
$ sudo cp u-boot.img /media/<UserName>/boot
```

### Extraction

With the terminal still in the folder `/yocto/build/tmp/deploy/images/overo`, the same where the file copy procedure was performed, enter the following command to extract the system directory:

```
$ sudo tar -xjvf gumstix-console-image-overo.tar.bz2 -C /media/<UserName>/rootfs
```

### Result on the Memory Card

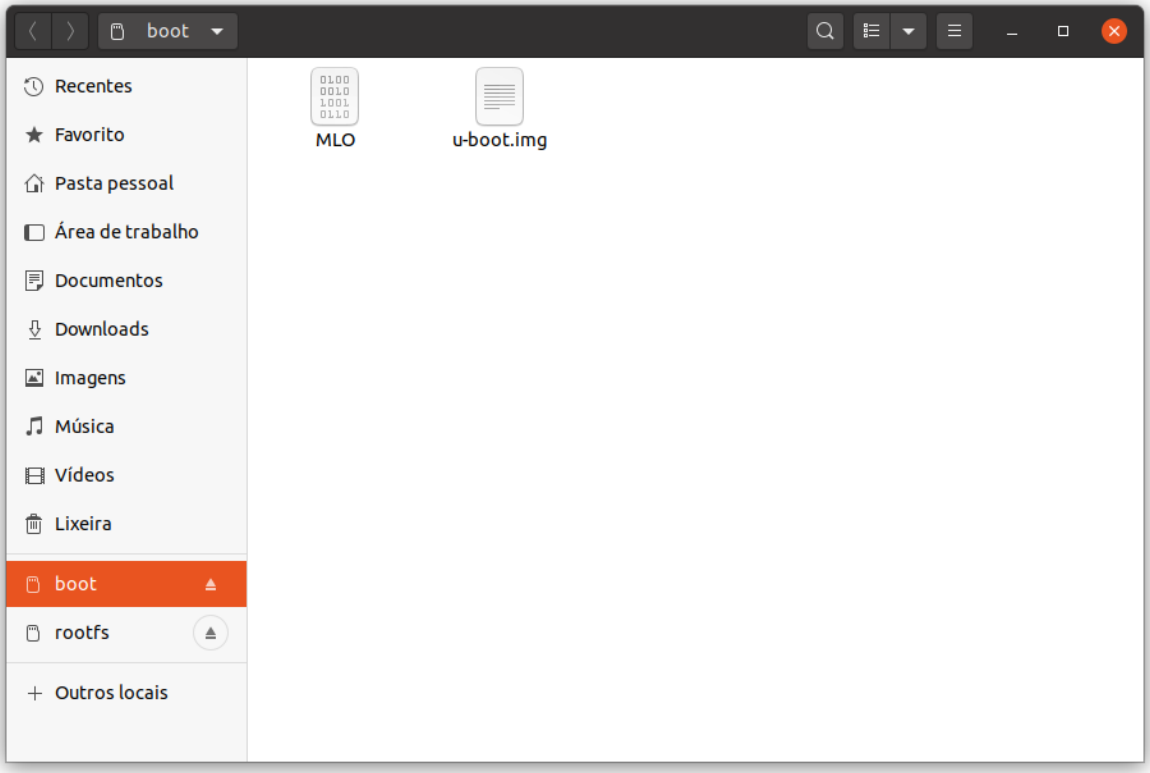
Boot partition:

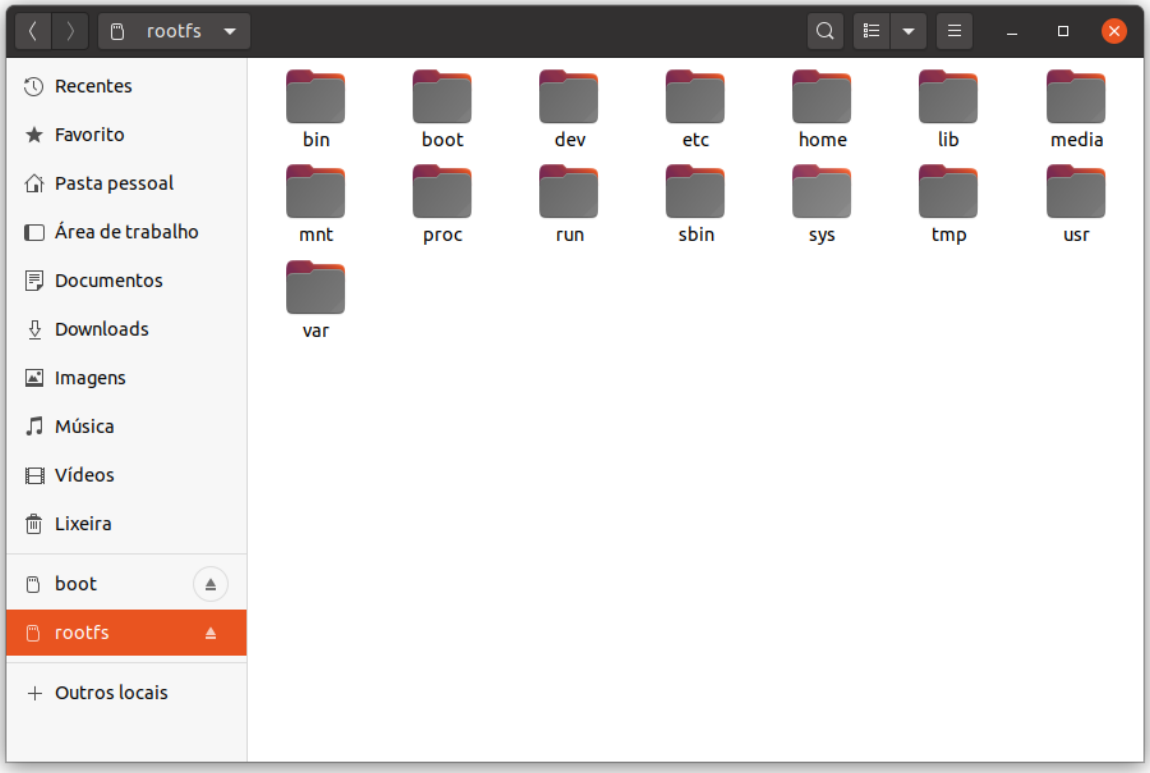
Rootfs partition:

**Tip:** Remember to unmount the partitions before removing the SD card.

### References

- Create Bootable MicroSD Card - gumstix.com


boot
Recentes
Favorito
Pasta pessoal
Área de trabalho
Documentos
Downloads
Imagens
Música
Vídeos
Lixeira
boot
rootfs
Outros locais
MLO
u-boot.img


rootfs
Recentes
Favorito
Pasta pessoal
Área de trabalho
Documentos
Downloads
Imagens
Música
Vídeos
Lixeira
boot
rootfs
Outros locais
bin
boot
dev
etc
home
lib
media
mnt
proc
run
sbin
sys
tmp
usr
var

- PITA, H. C. Desenvolvimento de sistema de comunicação multiplataforma para veículos aéreos de asa fixa. Faculdade de Tecnologia, Universidade de Brasília, 2018.

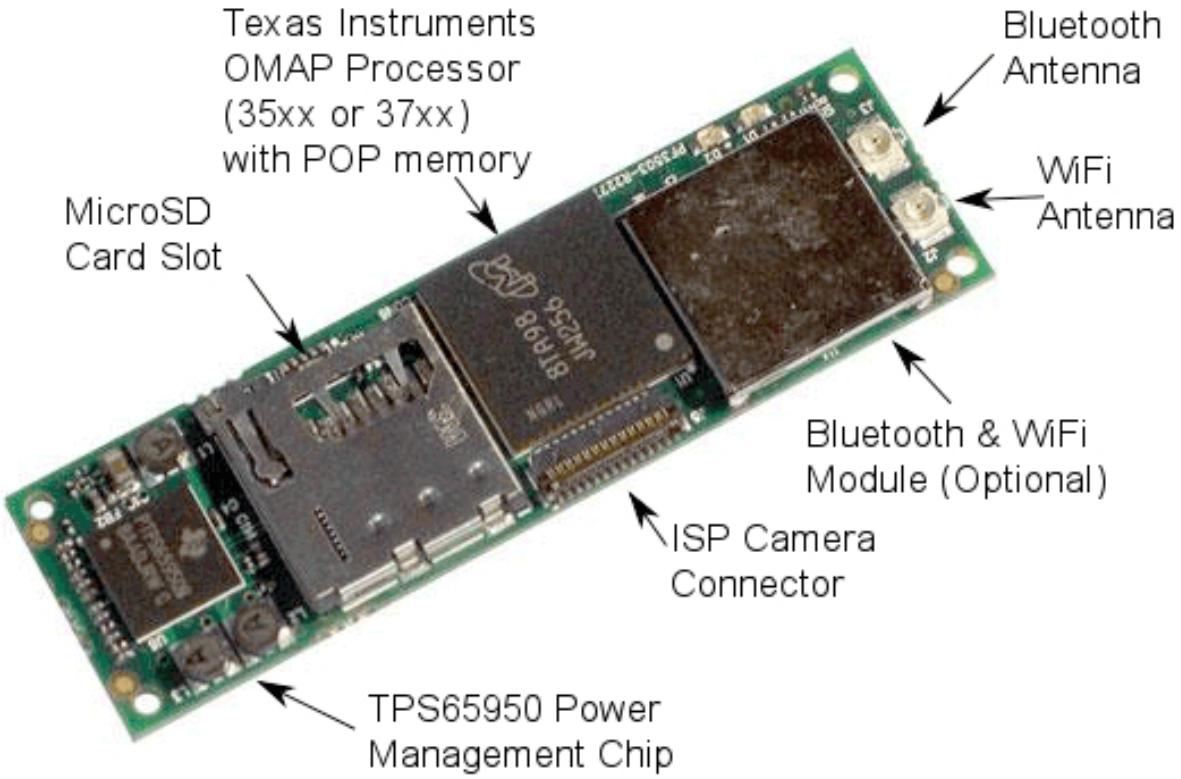
### References

- PITA, H. C. Desenvolvimento de sistema de comunicação multiplataforma para veículos aéreos de asa fixa. Faculdade de Tecnologia, Universidade de Brasília, 2018.

### Getting started with Gumstix Overo

#### Mounting Gumstix COM on the Tobi Expansion Board

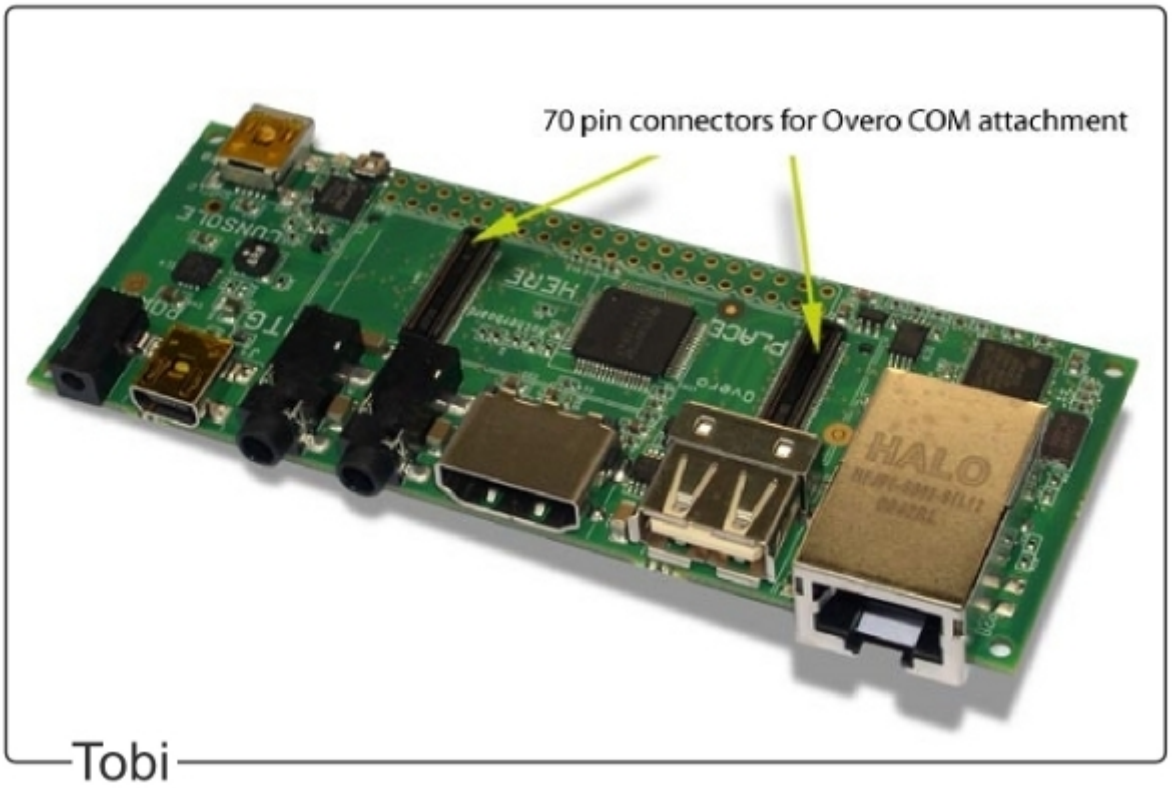
The configuration of Gumstix Overo computers consists of a module computer and an expansion board. The Overo module connects to a Tobi expansion board via the two 70-pin AVX connectors located on the bottom of the COM. Place the Tobi board on a flat, anti-aesthetic surface, align the COM with the white outline on the board above the connectors and gently press the COM until it snaps into place.

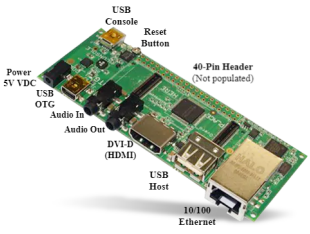



To use the camera, the camera board must be connected to the top of the Overo COM via a ribbon cable.

#### Overo Connections

The Tobi expansion card comes with a USB Host port and a USB On-the-Go (OTG). The USB Host port is used exclusively to connect peripherals to the system, while the USB OTG port can be used to connect peripherals via USB OTG cable or to connect the Gumstix system as a peripheral to a separate host system.


70 pin connectors for Overo COM attachment
Tobi


USB
Console
Reset
Button
40-Pin Header
(Not populated)
Power
5V VDC
USB
OTG
Audio In
Audio Out
DVI-D
(HDMI)
USB
Host
10/100
Ethernet

The USB Host port uses a current of 500 mA and accepts a **High-speed** (HS)signaling rate at 480 Mbit/s, while the USB OTG port has a current of 100 mA and supports three different signaling rates, **Low Speed** (LS) at 1.5 Mbit/s, **Full Speed** (FS) at 12 Mbit/s and **High Speed** (HS) at 480 Mbit/s.

---

**Note:** Many USB peripherals use a signaling rate of **Full-Speed** (FS) and do not work on the USB Host port, which is only **High Speed** (HS). If you are having trouble connecting USB peripherals directly to the Gumstix system, connecting the peripherals first to a powered USB hub and then connecting the powered hub to the Gumstix system will usually solve the problem.

---

For connecting more peripherals, in addition to the number of USB ports available on the Tobi expansion card, we recommend using a USB hub. The **powered USB hub** must be connected to the USB Host port and a **non-powered USB hub** must be connected to the USB OTG port with a USB On-the-Go cable.

---

**Tip:** The video Connecting Gumstix Tobi Expansion Board to Video Monitor demonstrates how to connect an Overo COM to a monitor and some peripherals via the Tobi board.

---

### Connecting to Overo

First, insert your microSD card with the operating system image in the card slot at the top of Overo COM. Make sure it fits securely in place.

The Overo computer can be accessed by connecting it to another Linux or Windows computer, or even be connected directly to a DVI monitor and connected to various peripherals, such as mouse, keyboard, monitor, sound output, among others, through the Tobi expansion board.

In this work, we will choose to connect it to a Linux computer and establish a serious connection via the USB Console port for simplicity.

### Establishing a serial connection via console

To connect the embedded computer to the host computer, connect a USB cable to the computer and to the USB console of the Tobi expansion board. Once this is done, a green light should come on indicating the correct connection. Then check which serial communication port Gumstix COM is connected to, in Windows this can be checked by accessing **Device Manager** and then **Ports (COM and LPT)**. On Linux, just run the command:

```
$ dmesg | grep tty
```

---

**Note:** The `dmesg` command is a command that prints the core messages which, in most cases, are messages from the device drivers. When we add ‘‘ grep tty‘‘we are performing a search on the outputs of the ‘‘ dmesg‘‘ function for the term `tty` and restricting its output to those messages that contain this term.

---

The Gumstix card should be the last entry to appear. For example:

```
user@Ubuntu:~$ dmesg | grep tty
[    0.000000] printk: console [tty0] enabled
[ 4214.120990] usb 2-1: FTDI USB Serial Device converter now attached to␣
↪**ttyUSB0**
```

Then it will be necessary to run a program to emulate the terminal, we recommend the Screen program for Linux OS. If you have not yet installed it, just run the command line `sudo apt-get install screen`. Or in the case of the Windows OS, PuTTY is recommended. These programs emulate terminals and perform only the task of printing the characters received through the serial port, or USB in this case, and sending the characters typed through that same port.

To start communication by terminal with Gumstix Overo, just run the following command line:

```
$ sudo screen /dev/<USB Device Name> 115200
```

In the case of the command line in the example presented above, the term `ttyUSB0` was the port found when using the command `dmesg` and "115200" is the communication speed in *baud*. At this point, communication between Gumstix and the computer must be established and as soon as the Gumstix is turned on, the characters must begin to be printed on the computer screen.

### Booting the Overo COM

Once connected to the console, Overo COM will be ready to be turned on. To boot the system, simply connect the 5 Volt power supply to your expansion card. The LED indicators on the COM should light up in blue and green. The boot process will be displayed on your host machine's terminal.

However, before turning it on, it is important to comment that the manufacturer recommends cleaning variables from flash memory whenever starting a new version of the operating system on the embedded computer for the first time. To do this, simply interrupt the boot process by pressing any key before it starts at the moment when the message "**Hit any key to stop autoboot**" and a countdown on the screen appears. The typical startup process will be similar to the following:

```
reading u-boot.img
reading u-boot.img


U-Boot 2012.04.01 (Jul 19 2012 - 17:31:34)

OMAP36XX/37XX-GP ES1.2, CPU-OPP2, L3-165MHz, Max CPU Clock 1 Ghz
Gumstix Overo board + LPDDR/NAND
I2C:   ready
DRAM:  512 MiB
NAND:  512 MiB
MMC:   OMAP SD/MMC: 0
In:    serial
Out:   serial
Err:   serial
Board revision: 1
Direct connection on mmc2
timed out in wait_for_pin: I2C_STAT=1000
I2C read: I/O error
Unrecognized expansion board
Die ID #2d3800229ff8000001683b060a00b012
Net:   smc911x-0
Hit any key to stop autoboot:  0
Overo #
```

Once the system boot is interrupted, just execute the command `nand erase 240000 20000` to clear the saved variables and `reset` to restart the boot process, as shown below:

```
# nand erase 240000 20000
# reset
```

**Note:** If the blue and green LEDs on the COM do not light up and nothing appears on your terminal, try pressing the reset button on the expansion card until you see a boot process. If the problem persists, the image may not have been installed successfully. It is recommended that you try to install again or use a different image.

The following figure illustrates this procedure. The characters are printed quickly, and the time count is only 1 second for the cores of the Yocto project, so it is necessary to be attentive to interrupt the process.

Once this is done, the boot process should start, and several messages will appear on the screen. It is important to check, the first time the operating system is started, that no error message appears and, if all goes well, a password

will be required at the end of the process; if the embedded computer has reached this point, everything is probably in order. The password to access the Yocto system is "**root**" and for the Ubuntu Gumstix system, if necessary, the password is the same as the user.

### Saving the OS image to flash memory

The Gumstix Overo WaterSTORM COM has a non-volatile internal memory of 1 GB of the Flash type, enough memory to store the operating system. Although the most recommended is to continue using the SD card because it has more memory and is easily transferred between devices, having the operating system saved in the flash memory of the embedded computer can be useful.

The manufacturer's website describes four different ways to perform this procedure. The way that presented the best result was the last of the options explained and it is summarized to install in the flash memory everything that was installed in the memory card added to the core binary through a script provided in its website. The desired script is available in Flashing with U-Boot - Write Images to Flash, however, the entire process will be described in detail below.

1. With the bootable SD card connected to your host computer, access the **/boot** directory on the ** rootfs ** partition. For example, if **rootfs** is mounted on **/media/<User_Name>/rootfs/**:

```
$ cd /media/<User_Name>/rootfs/boot
```

2. We must store in the **boot** folder of the **rootfs** partition the new **MLO**, **u-boot.img** and **core binary**. Note that these *bootloaders* that will be added to the **boot** folder are not the same as those in the **boot** partition, as these new *bootloaders* must be specific to operate from flash memory. These new files can be obtained with the following commands:

```
$ sudo wget https://s3-us-west-2.amazonaws.com/gumstix-yocto/2015-02-25/overo/
↪master/MLO
$ sudo wget https://s3-us-west-2.amazonaws.com/gumstix-yocto/2015-02-25/overo/
↪master/u-boot.img
$ sudo wget https://s3-us-west-2.amazonaws.com/gumstix-yocto/2015-02-25/overo/
↪master/gumstix-console-image-overo.ubi -O rootfs.ubi
```

3. Create a script to save the files in the flash memory with the name *flash-all.cmd*. To do this, simply execute the command:

```
$ sudo nano flash-all.cmd
```

Copy and paste the script:

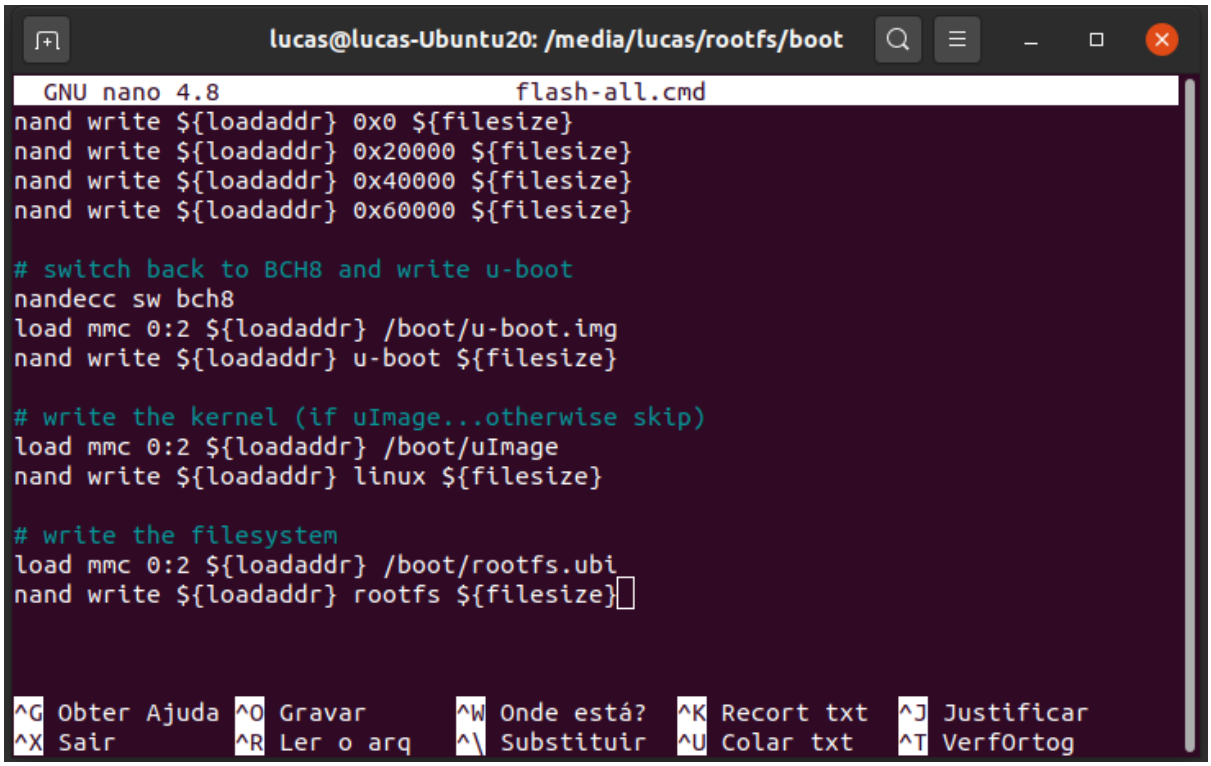
```
nand erase.chip

# switch to 1-bit ECC and write MLO
load mmc 0:2 ${loadaddr} /boot/MLO
nandecc hw
nand write ${loadaddr} 0x0 ${filesize}
nand write ${loadaddr} 0x20000 ${filesize}
nand write ${loadaddr} 0x40000 ${filesize}
nand write ${loadaddr} 0x60000 ${filesize}

# switch back to BCH8 and write u-boot
nandecc sw bch8
load mmc 0:2 ${loadaddr} /boot/u-boot.img
nand write ${loadaddr} u-boot ${filesize}

# write the kernel (if uImage...otherwise skip)
load mmc 0:2 ${loadaddr} /boot/uImage
nand write ${loadaddr} linux ${filesize}
```

(continues on next page)

(continued from previous page)

```
# write the filesystem
load mmc 0:2 ${loadaddr} /boot/rootfs.ubi
nand write ${loadaddr} rootfs ${filesize}
```

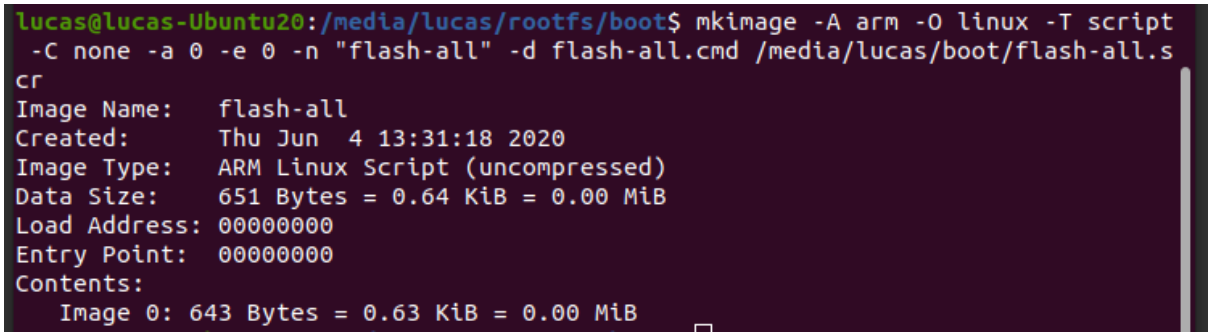
Then confirm the file name (**Ctrl+O**) and exit the text editor (**Ctrl+X**).




4. To make the script executable and add it to the boot partition of the bootable SD card, simply run the following command line (assuming the boot partition is mounted on /media/<User_Name>/boot):

**Warning:** Remember to edit the filenames in the script to match the filenames that will be added next.

```
$ mkimage -A arm -O linux -T script -C none -a 0 -e 0 -n "flash-all" -d flash-all.
↪cmd /media/<User_Name>/boot/flash-all.scr
```




**Note:** If the command `mkimage` is not found, just run the command `sudo apt install u-boot-tools` to install the tool package on your computer. The command `mkimage` is a command used to make images for use by **u-boot**. Command options and explanations are easily obtained by typing `man mkimage` in the Linux terminal.

5. Unmount the SD card and insert it into your embedded computer, start the system and wait for u-boot to load. Stop the startup process when you see "**Hit any key to stop autoboot**" and enter the command:

```
# mmc rescan 0; load mmc 0 ${loadaddr} flash-all.scr; source ${loadaddr}
```

This command line will execute the script passing the bootloaders, the core binary and the root files of the operating system to the flash memory of the embedded system and the messages shown in the figure below should be printed.

```
Overo # mmc rescan 0; load mmc 0 ${loadaddr} flash-all.scr; source ${loadaddr}
mmc - MMC sub system

Usage:
mmc info - display info of the current MMC device
mmc read addr blk# cnt
mmc write addr blk# cnt
mmc erase blk# cnt
mmc rescan
mmc part - lists available partition on current mmc device
mmc dev [dev] [part] - show or set current mmc device [partition]
mmc list - lists available devices
mmc hwpartition [args...] - does hardware partitioning
  arguments (sizes in 512-byte blocks):
    [user [enh start cnt] [wrrel {on|off}]] - sets user data area attributes
    [gp1|gp2|gp3|gp4 cnt [enh] [wrrel {on|off}]] - general purpose partition
    [check|set|complete] - mode, complete set partitioning completed
  WARNING: Partitioning is a write-once setting once it is set to complete.
  Power cycling is required to initialize partitions after set to complete.
mmc setdsr <value> - set DSR register value

reading flash-all.scr
715 bytes read in 5 ms (139.6 KiB/s)
## Executing script at 82000000

NAND erase.chip: device 0 whole chip
Erasing at 0x3ffe0000 -- 100% complete.
OK
57380 bytes read in 117 ms (478.5 KiB/s)

NAND write: device 0 offset 0x0, size 0xe024
 57380 bytes written: OK

NAND write: device 0 offset 0x20000, size 0xe024
 57380 bytes written: OK

NAND write: device 0 offset 0x40000, size 0xe024
 57380 bytes written: OK

NAND write: device 0 offset 0x60000, size 0xe024
 57380 bytes written: OK
397108 bytes read in 142 ms (2.7 MiB/s)

NAND write: device 0 offset 0x80000, size 0x60f34
 397108 bytes written: OK
** File not found /boot/uImage **

NAND write: device 0 offset 0x280000, size 0x60f34
 397108 bytes written: OK
99483648 bytes read in 6315 ms (15 MiB/s)

NAND write: device 0 offset 0xa80000, size 0x5ee0000
corrected bitflip 3249
corrected bitflip 607
 99483648 bytes written: OK
Overo #
```

Remove the SD card and restart your system. If everything went well, your system should start up normally.

### References


- PITA, H. C. Desenvolvimento de sistema de comunicação multiplataforma para veículos aéreos de asa fixa. Faculdade de Tecnologia, Universidade de Brasília, 2018.

- 4. Boot Your System - gumstix.com
- Write Images to Flash - gumstix.com

### 1.1.4 Using the embedded computer

In this chapter, basic knowledge necessary for handling the embedded computer Gumstix Overo, the Linux operating system and the specific operating system installed will be discussed and, later, tests of the GPIO and serial communication of the embedded computer will be carried out. The information described in these steps is essential for understanding different topics and will be used throughout the work.

#### Getting used to Linux

Having performed the procedures presented in the previous section correctly, the embedded computer will operate with a Linux operating system very similar to what we are used to on regular computers. Therefore, we can perform some simple procedures so that we can explore and get a little used to the environment in which we will work. What will be demonstrated in this step are standard procedures, commands and information for Linux systems running on the Gumstix Overo WaterStorm embedded computer.

---

**Tip:** If the reader is already used to the Linux desktop, it is recommended to skip to the next section.

---

#### Command Lines

We will begin the process of setting featuring basic command lines that will help you use the operating system without major problems. It is worth mentioning that there is no need for any special knowledge to use Linux commands, since the terminal is a program like any other.

---

**Tip:** For more details on any commands listed here, simply execute the command followed by `--help` or preceded by `man`.

The `--help` command prints a brief description of the commands followed by instructions for use, for example:

```
uname --help
```

To reduce the amount of printed content can use `less`, for example:

```
ls --help | less
```

While the `man` command displays the instruction manual for the requested command, for example:

```
man cd
```

The main commands are simple, the arrows for **up and down** go up and down the page, respectively, as well as the arrow keys for **left and right** make the movement for reading the texts, the **Enter** also makes the page go down. The "**h**" key shows the **help** of the `man` command, showing all the keys and shortcuts used. And the "**q**" key leaves the manual navigation.

---

1. `cat` **comand**

```
cat [OPTION] [FILE]
```

Its name is a derivation of the word *concatenate* and allows you to create, merge and display files in standard screen format or in another file, among other things.

If the option is not specified, the `cat` command reads the content present in the indicated file and prints it on the screen. For example, when executing the command `cat/etc/issue` in the Overo terminal, the branch and the version of the operating system used are printed.

```
root@overo:/# cat /etc/issue
Poky (Yocto Project Reference Distro) 1.7.1 \n \l

root@overo:/#
```

**Note:** If you want to know more features, access Comand Cat Linux - man7.org.

2. `uname` **comand**

```
uname [OPTION]
```

The command `uname`, a name derived from the term "Unix Name", displays detailed information about your Linux system, such as the name of the machine, the operating system, the kernel and so on.

For example, the option `-a` prompts you to print all the information available by the program.

```
root@overo:/# uname -a
Linux overo 3.17.8-custom #1 SMP Wed Feb 25 04:05:20 PST 2015 armv7l GNU/Linux
root@overo:/#
```

3. `echo` **comand**

```
echo [OPTION] [STRING]
```

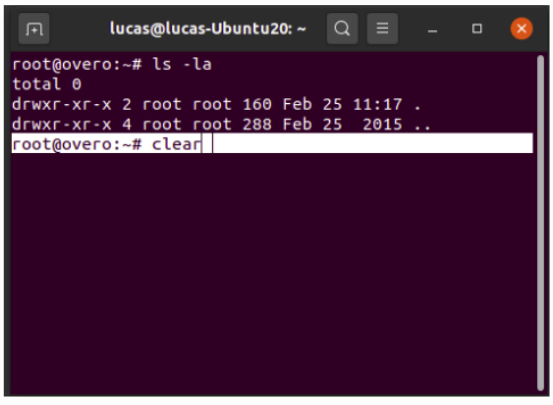
The `echo` command is a command used to display messages on the screen or in a file. When using the command followed by a *string*, the text of the *string* is printed on the terminal screen. For example:

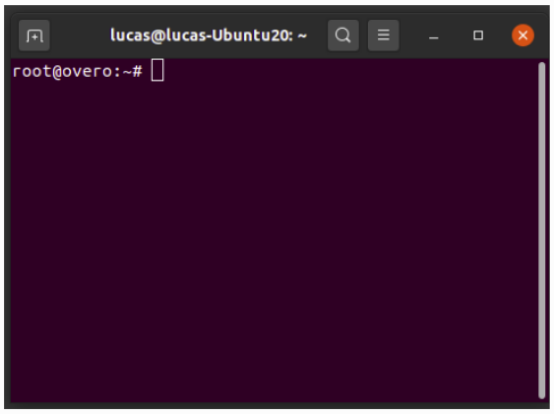
```
root@overo:~# echo "Aerolab"
Aerolab
root@overo:~# echo Aerolab
Aerolab
```

4. `clear` **comand**

```
clear [OPTION]
```

Use the `clear` command to clear the contents of your terminal screen. The command doesn't require parameters, it uses variables from the current desktop to determine how to clear the screen.




5. `pwd` **comand**

```
pwd [OPTION]
```

The `pwd` command is used to find the path to the current directory (folder) where you are. The command will return a full path, which is basically a path that begins with a forward slash (/). For example:

```
root@overo:~# pwd
/home/root
```

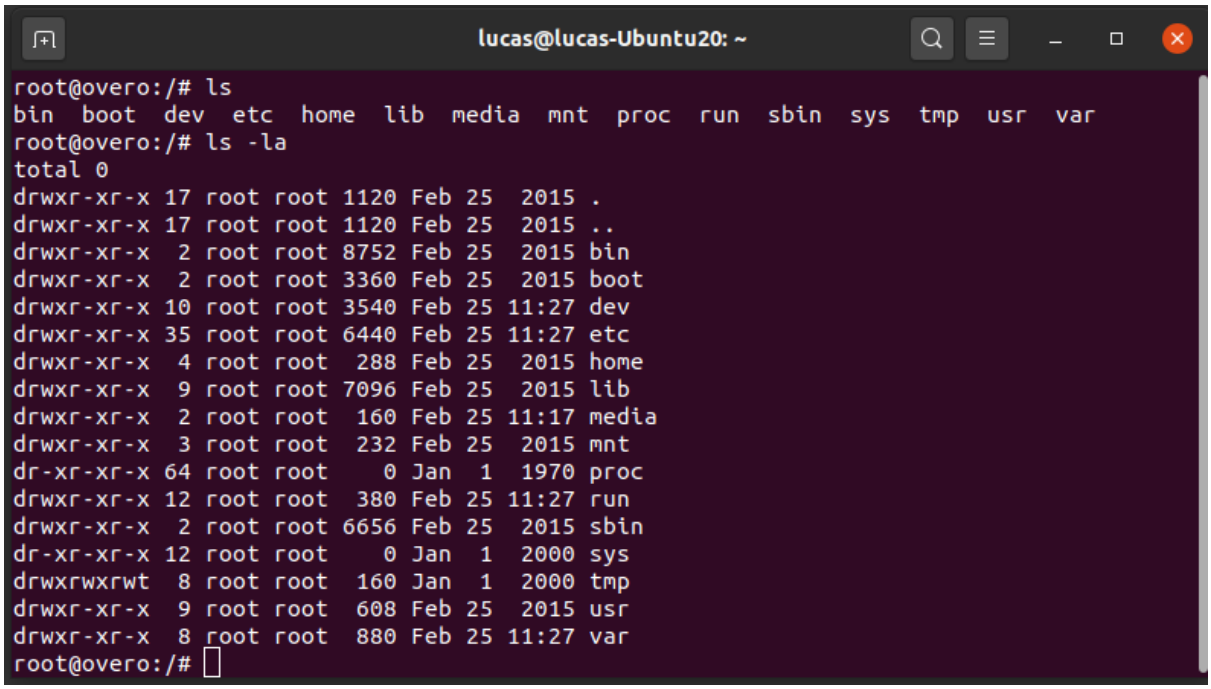
6. `ls` **comand**

```
ls [OPTION] [FILE]
```

Its name derives from the first consonants of the English word "list". The `ls` command is used to list content within a directory. By default, this command will show only the contents of the current directory where you are.

By using the command `ls` without specifying any options or directory, the terminal will print the contents of the current directory. However, if you want to see the contents of other directories, type **ls**, and then the path to the directory. For example, type `ls/home/username/Documents` to see the contents of **Documents**.

A useful option of the `ls` command is the `ls -la` option which, in addition to listing all files and folders in the current directory, also prints some useful information about each one.




The figure shows an example of the output of the command `ls -la`, in it we can see that for each file a line with several columns of information is printed. Explaining what each column means is unnecessary, however it is important to know what the first letters mean, as this is often the cause of some problems.

The first 10 columns that are composed of "**-**" and varied letters indicate the file type and the users' permissions for those files. In the figure, the first column, which is always indicated by the letter "**d**", showing that the file is a directory, if the file was a program or a regular text file it would be indicated by a "**-**". The next nine letters can be separated into groups of 3 indicating the permissions of the owner, group and others, respectively. The letters "**r**", "**w**" and "**x**" indicate **reading**, **writing** and **execution**, respectively. Therefore, if we analyze the data in the "usr" folder, we will see that the owner of the folder has permission to read, write and execute, but his group and other users will only be allowed to read and execute.

7. `cd` **comand**

```
cd [OPTION] [DIRECTORY]
```

Its name is an acronym of the English expression "change directory" and its purpose is, as your name suggests, to change from the current working directory, the directory you are in, to another directory. For example, if you are in **/home/user** and want to go to **Documents**, a user subdirectory, just type `cd Documents`.

```
root@overo:/# ls
bin  boot  dev  etc  home  lib  media  mnt  proc  run  sbin  sys  tmp  usr  var
root@overo:/# cd home/
root@overo:/home#
```

In addition, there are some shortcuts that can be used to navigate quickly. Some of them are:

```
cd ..    # (with two points in a row) move to a directory above (previous).

cd       # moves directly to the home folder.

cd-      # (with a hyphen) moves to previous directories.
```

**Note:** The Linux terminal is sensitive to character types. Therefore, you need to type the name of the directory exactly as it is written (using lowercase or uppercase letters).

8. `cp` **comand**

```
cp [OPTION] <DIRECTORY SOURCE>
```

This command is used to copy files or directories to a specific directory. For example, the command `cp Document.txt /home/username/Documents` will create a copy of **Document.txt** in the **Documents** directory, if this document exists. While the command `cp -R /home/user/project /home/user/new_project` will copy the directory **project**, with all its files, subdirectories and files from the subdirectories to the directory **new_project** .

9. `mv` **comand**

```
mv <HOME DIRECTORY> <DESTINATION DIRECTORY>
```

The name of the command `mv` is derived from the first consonants of the English word "move" and its usual use is to move files, although it can also be used to rename files. That is, this command copies and changes the path of the original file to the desired path and, in this way, deletes the original file (it is also possible to rename and change the directory of a file simultaneously).

The syntax in this command is similar to the `cp` command. You need to type `mv`, the file name and the destination directory. For example: `mv file.txt /home/username/Documents`.

As for renaming files, the argument to be used is `mv Old_Name.txt Old_Name.txt`, with "**Old_Name.txt**" being the original file and "**New_Name.txt**" being the new file.

10. `mkdir` **comand**

```
mkdir [OPTION] DIRECTORY
```

The `mkdir` command creates a new directory, if it doesn't already exist. For example, execute `mkdir Test` will create a new directory called **Test**. Its name derives from the English term "Make Directory".

11. `rmdir` **comand**

```
rmdir [OPTION] DIRECTORY
```

The command `rmdir` has the function of deleting a directory and its syntax is similar to that of the command `mkdir`. However, this command only allows empty directories without content to be deleted. Its name comes from the English term "Remore Directory".

12. `rm` **comand**

```
rm [OPTION] [FILE]
```

The rm command is used to delete a specific file or directory with all the contents inside. For example, running the command `rm /home/user/Documents/text.txt` will delete the file *text.txt*.

If you want to delete a specific directory (as an alternative to rdmir) use `rm -r [DIRECTORY]`.

13. `chmod` **comand**

```
chmod [OPTION] MODE[,MODE FILE    #or
chmod [OPTION] MODE-OCTAL FILE    #or
chmod [OPTION] --reference=ARQREF FILE.
```

The `chmod` (short for "change mode") is a command that can change access permissions for system objects (files and directories) and flagging in a special way. The flags are a way to set options and pass arguments to commands that you run.

Usually, the command `chmod` is used in the form:

```
chmod <OPTION> <PERMISSIONS> <FILE NAME>
```

If none option is specified, `chmod` command modifies the file's permissions to the specified permissions. There are two ways to represent the possible permissions: with symbols (alphanumeric characters) or with octal numbers (the digits 0 to 7). Here we will stick to just explain the symbolic method.

As seen earlier, the characters **r, w and x** represent three types of permissions: read, write and execute, respectively. However, to specify the user group when granting or removing a permission, the command uses a few more symbols. To visualize it more clearly, imagine that these symbols are in two lists, and the combination of them generates the permission:

**User Group:**

```
u: user owner of the file
g: file owner user group
O (capital letter 'o'): all other users
a: all types of user (owner, group and others)
```

**Permission type:**

```
r: refers to read permissions
w: refers to writing permissions
x: refers to execution permissions
```

**In order to be able to combine the symbols of these two lists, operators are used:**

```
+ : adds permission
- : removes permission
= : defines permission
```

To show how combinations can be made, note the examples below:

```
chmod u+w teste.exe     # adds write permission to the file for a user

chmod g+rw teste.exe    # adds read and write permission to your group

chmod g=rwx teste.exe   # adds all available permissions to the group
```

**Tip:** Since this command is relatively complicated, more information can be obtained from Linux chmod Command.

14. `sudo` **comand**

The `sudo` command allows ordinary users to perform tasks that require permissions from another user, usually the super user, to perform specific tasks within the system in a safe and controllable manner by the administrator. However, it is not very advisable to use it on a daily basis because it may be an error if you do something wrong. The name is an abbreviated way of referring to "Substitute User Do" or "Super User Do".

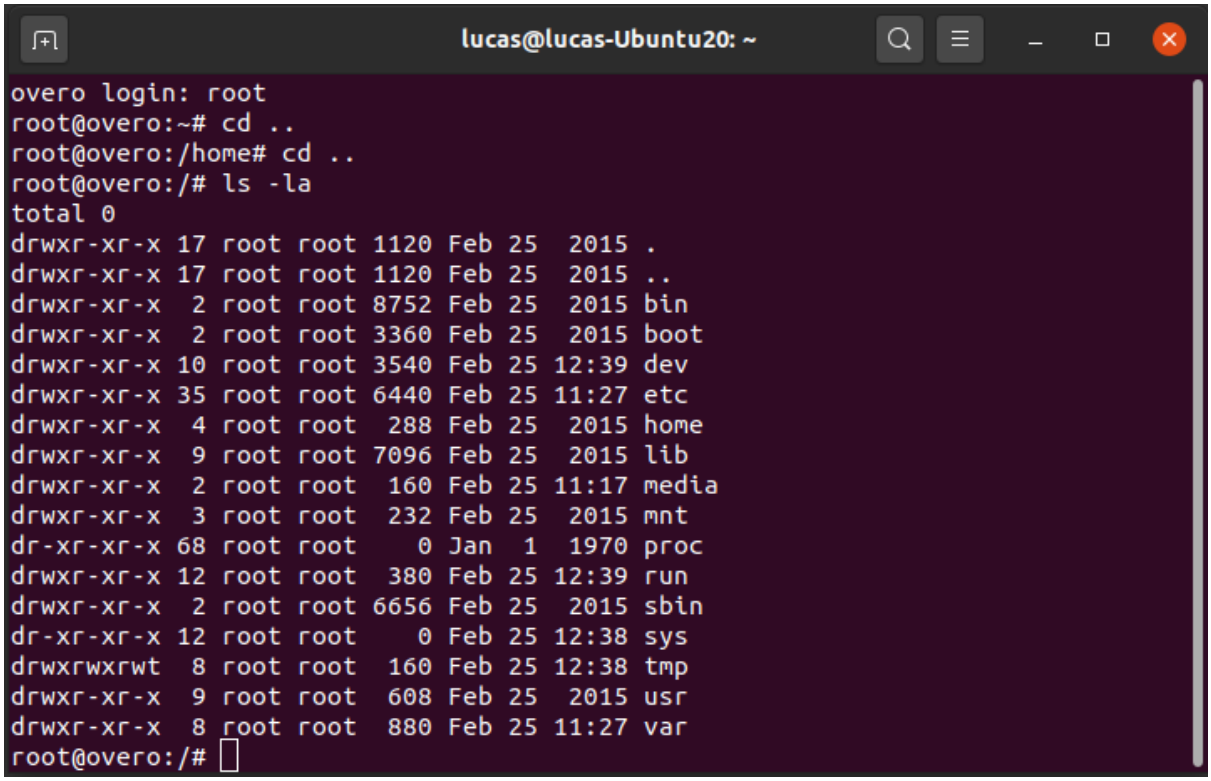
Generally, the command `sudo` is executed in the form:

```
sudo [-u user] <command>
```

Where <command> is the command you want to execute. The [-u user] option is used to specify which user should be used to execute the command, if omitted, the command `sudo` assumes the root user and asks for the login password to confirm.

### Exploring the System Files

Once this information and basic commands are passed, we are able to explore the system files. Therefore, we will migrate to the first directory of the system by running `cd ..` twice. And then run the command `ls -la` so that we can view the system folders. If everything is done as explained we should get something as shown in the following figure.




Of the various directories present in the figure, the directories "**/bin**", "**/boot**", "**/dev**", "**/lib**" and "**stand out/sys**".

The directory "**/bin**" is where the binaries of essential Linux commands are stored, such as the commands presented previously, so if it becomes necessary to add any more software to the microprocessor that is necessary, it must be added to this folder so that it can be found by the operating system when requested.

The directory "**/boot**" has already been used in this work and is the place where the bootloaders and other programs that are part of the system boot must be stored.

The "**/dev**" directory is the directory where system device files are stored. Device file is a way that the Linux system uses to generate a communication interface with device drivers. It will be used a lot later on during serial communication, for example.

The "**/lib**" directory is the directory that contains the essential libraries for the binaries contained in the "/bin" directory, so if new software installation is necessary, we will probably also need to add some library to this

directory.

Finally, the "**/sys**" directory is the directory that contains device and driver information. This folder will be used a lot if it is necessary to use functions such as General Purpose Input/Output (**GPIO**), **I2C** and Direct Memory Access (**DMA**).

### References


- PITA, H. C. Desenvolvimento de sistema de comunicação multiplataforma para veículos aéreos de asa fixa. Faculdade de Tecnologia, Universidade de Brasília, 2018.
- Linux man pages online - man7.org
- 30 Comandos Linux Que Todo Usuário Deve Conhecer - hostinger.com.br


### Cross-compilation

Cross-compilation occurs when a device compiles source code for a different platform than the one that compiled the code, we usually have a development environment that is in an architecture different from the architecture of the board on which we will run our program. So that our notebooks or desktops can generate a binary, a program that runs on another architecture, there is the Cross Compilation. For example, in our case, a computer with Linux Ubuntu, using an Intel or AMD processor with x86 architecture, will compile code for the embedded computer running an adapted Linux system running on a completely different architecture.

Using cross-compilation to work with embedded systems is very common, especially when they do not have the processing capacity to support a compiler. For us, it would be possible to compile in the embedded system itself, however, although the resources are not so limited, we do not want to waste them. In addition, programming on a regular computer with several different IDE types is much better than programming on an embedded computer with limited interface capabilities.

To accomplish this cross-compilation process must first obtain a Software Development Kit, also known as SDK or devkit. The SDK is nothing more than a set of software development tools that make it possible to create and compile software for a different system.

### Installing the SDK

---

**Note:** The Yocto project offers a tutorial on how to obtain and install the SDK for your system on the GitHub Cross Compile with Yocto SDK page. However, all the procedures performed and the results obtained in the installation will be described in this topic.

---

To obtain the SDK for the image of OS we are using, simply perform the following procedures:

1. **Creation of directories**

We will create a directory called **workspace** where you will install the SDK and compile the codes in the future. Choose a location convenient for you and execute the commands.

```
$ mkdir workspace
$ cd workspace
$ wget http://gumstix-yocto.s3.amazonaws.com/sdk.sh
```

---

**Note:** The `wget` command will download the SDK for Gumstix systems. This process may take a while, depending on your download speed.

---

2. **Checking permissions**

```
lucas@lucas-Ubuntu20:~$ mkdir workspace
lucas@lucas-Ubuntu20:~$ cd workspace/
lucas@lucas-Ubuntu20:~/workspace$ wget http://gumstix-yocto.s3.amazonaws.com/sdk
.sh
--2020-06-15 12:24:24--  http://gumstix-yocto.s3.amazonaws.com/sdk.sh
Resolvendo gumstix-yocto.s3.amazonaws.com (gumstix-yocto.s3.amazonaws.com)... 52
.218.245.106
Conectando-se a gumstix-yocto.s3.amazonaws.com (gumstix-yocto.s3.amazonaws.com)|
52.218.245.106|:80... conectado.
A requisição HTTP foi enviada, aguardando resposta... 200 OK
Tamanho: 1028162585 (981M) [text/x-shellscript]
Salvando em: “sdk.sh”

sdk.sh                      0%[                                    ]   3,13M   41
sdk.sh                0%[                    ]   7,41M   343KB/s    TED 47m 32s
```

After the download is complete, check the file permissions by running the command `ls -la sdk.sh`. If the terminal returns that you are allowed to run this file, skip to the next procedure. Otherwise, you will need to change the file's permissions using the `chmod` command.

---

**Note:** Recalling, the first three spaces indicate the owner's permissions and, basically, we have 3 basic permissions, **r**, **w** and **x**, indicating permission to read, white and execute, respectively.

---

For example, in the image below, the file has read and white permission, but cannot be executed.

```
lucas@lucas-Ubuntu20:~/workspace$ ./sdk.sh
bash: ./sdk.sh: Permissão negada
lucas@lucas-Ubuntu20:~/workspace$ ls -la sdk.sh
-rw-rw-r-- 1 lucas lucas 1028162585 mai 23 05:35 sdk.sh
lucas@lucas-Ubuntu20:~/workspace$
```

To add the permission to run, just execute command `chmod +x sdk.sh`.

```
lucas@lucas-Ubuntu20:~/workspace$ chmod +x sdk.sh
lucas@lucas-Ubuntu20:~/workspace$ ls -la sdk.sh
-rwxrwxr-x 1 lucas lucas 1028162585 mai 23 05:35 sdk.sh
lucas@lucas-Ubuntu20:~/workspace$
```

3. **Installation**

To start the installation, run the *sdk.sh* file. Once prompted, enter the directory you want to install the SDK in (we recommend creating a repository called *sdk*) and confirm.

```
$ ./sdk.sh
```

4. **Setting up the environment**

Installing the SDK will generate a folder with the content shown in the figure below.

---

**Note:** The "sysroot" directory contains the root files of the two systems, both the system that will compile the code and the system that will run the program, and the first file in the list imports the adrdresses and variables important for compiling the code.

---

In order to proceed with the compilation of the code it, is necessary to execute the following command line:

```
source sdk/environment-setup-cortexa8hf-neon-poky-linux-gnueabi
```

```
lucas@lucas-Ubuntu20:~/workspace$ ./sdk.sh
Poky (Yocto Project Reference Distro) SDK installer version 2.7.3
==================================================================
Enter target directory for SDK (default: /opt/poky/2.7.3): sdk
You are about to install the SDK to "/home/lucas/workspace/sdk". Proceed [Y/n]?
y
Extracting SDK......
```

```
lucas@lucas-Ubuntu20:~/workspace$ ls
sdk  sdk.sh
lucas@lucas-Ubuntu20:~/workspace$ cd sdk/
lucas@lucas-Ubuntu20:~/workspace/sdk$ ls
environment-setup-cortexa8hf-neon-poky-linux-gnueabi
site-config-cortexa8hf-neon-poky-linux-gnueabi
sysroots
version-cortexa8hf-neon-poky-linux-gnueabi
lucas@lucas-Ubuntu20:~/workspace/sdk$
```

### Compiling Hello World

Once the installation is done, we can test the cross-compilation by creating a simple script. Let's create a file in the text editor called "helloworld.c", paste the code below and save it in the **workspace directory**.

```c
#include <stdio.h>
int main(void)
    {
    printf ("Hello World!\n");
    return 0;
    }
```

We can now run the command `make <code_name>` to create an executable binary file from *helloworld.c*.

---

**Note:** The `make` command is actually a simplification of an extensive command line that calls a **arm-poky-Linux-gnueabi-gcc** compiler and gives it the parameters contained in the SDK folder. All this thanks to the "source" command previously used.

---

Once the code executable is obtained, just copy it to one of the memory card folders and transfer it to Overo and execute it. Remember that the main directory is the `/home/root/` directory, so if the file is placed inside this directory it will be very easy to find it.

After inserting the memory card in Overo, we can start it normally. When started, we go to the directory where the program was saved and run it with the command `./helloworld`. If everything goes well, the program should be executed, similar to the figure below.

### References


- PITA, H. C. Desenvolvimento de sistema de comunicação multiplataforma para veículos aéreos de asa fixa. Faculdade de Tecnologia, Universidade de Brasília, 2018.
- Cross Compile with Yocto SDK - github.com/gumstix


```
lucas@lucas-Ubuntu20:~/workspace$ source sdk/environment-setup-cortexa8hf-neon-p
oky-linux-gnueabi
lucas@lucas-Ubuntu20:~/workspace$
```

```
lucas@lucas-Ubuntu20:~/workspace$ ls
helloworld.c  sdk  sdk.sh
lucas@lucas-Ubuntu20:~/workspace$ make helloworld
arm-poky-linux-gnueabi-gcc  -mfpu=neon -mfloat-abi=hard -mcpu=cortex-a8 --sysroo
t=/home/lucas/workspace/sdk/sysroots/cortexa8hf-neon-poky-linux-gnueabi  -O2 -pi
pe -g -feliminate-unused-debug-types   -Wl,-O1 -Wl,--hash-style=gnu -Wl,--as-nee
ded  helloworld.c   -o helloworld
lucas@lucas-Ubuntu20:~/workspace$
```

```
Poky (Yocto Project Reference Distro) 1.8.2 overo ttyO2

overo login: root
root@overo:~# ls
helloworld
root@overo:~# ./helloworld
Hello World!
root@overo:~#
```

### Registers

**Warning:** Este tópico precisa ser testado e reeditado.

Following the procedures of the previous sections we are able to start the system and generate programs to be executed by the operating system. The next step, therefore, is to control the signals that can be sent to other devices by the embedded computer to establish communication between the devices.

Communication between devices is done by changing the voltage levels of the embedded computer's pins. These pins are, in summary fashion, connected to **system memory space** and, when we change the bit stored in this memory space, also changed the pin voltage level, allowing the encoding of a message and its transmission to another device.

Subsequently, the communication between devices will be more discussed, but at this moment what most matters to us are the **memory spaces**, mentioned in the previous paragraph. These memory spaces are actually volatile digital circuits that are capable os stoting voltage levels, the acess to the contents of these memory spaces is extremely fast and these memory spaces are called **register**. **Registers** are at the top of the memory hierarchy, making them the fastest type of memory in a central processing unit.

So, in order to implement communication between two devices, a modem and the embedded computer, for example, we first need to perform a simpler task of changing the voltage levels of a pin. This process of changing the voltage levels of a pin has several applications, ranging from simple ON / OFF control of an LED to serial communication between devices. Pins for this purpose are called General Purpose Input/Output (**GPIO**).

**General Purpose Input/Output** (**GPIO**) are, basically, communication pins for input and output of digital signals, of an integrated circuit or electronic circuit board, with no pre-defined purpose, thus being able to have functions defined by the designer or user to provide an interface between other devices (peripherals, modems, microcontrollers, microprocessors, etc.).

As previously mentioned, we are using the Overo embedded computer next to a Tobi expansion board. One of the functions of Tobi board is to provide user access to the pins of the embedded computer, so the pins of the embedded computer that we can access physically are the pins of the Tobi expansion board. In the figure below we can see a diagram that contains, in summary form, which functions or pins of the embedded computer are connected to each pin of the Tobi expansion board. Note that some of these pins have more than one function.

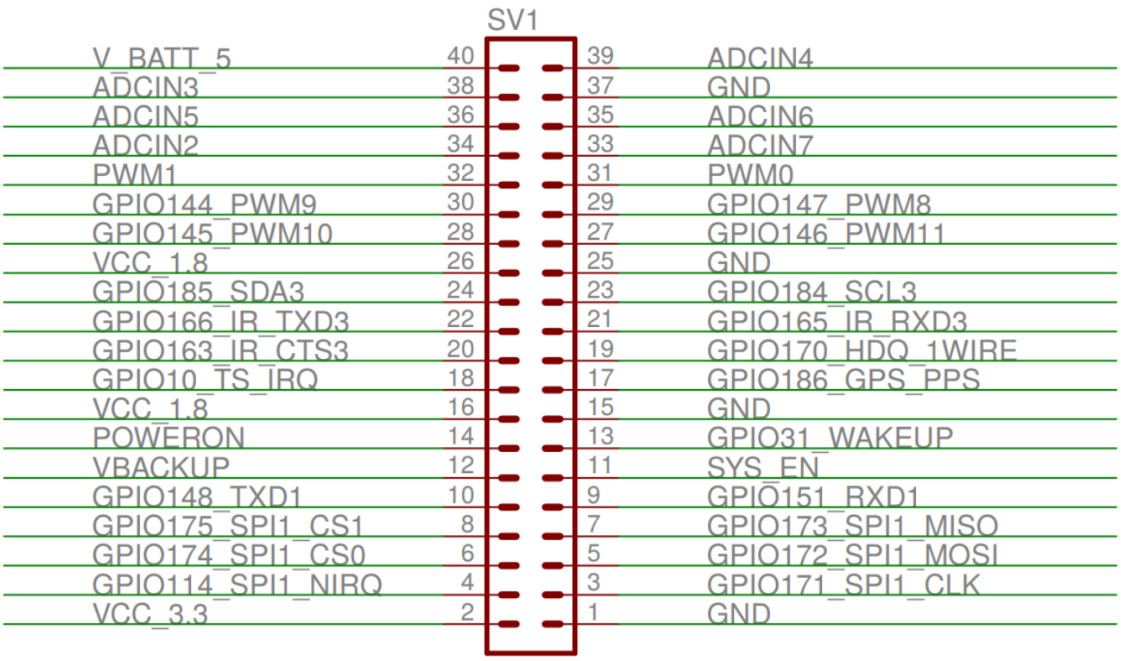



Fig. 6: Tobi expansion board pin diagram.

### GPIO control via terminal

The simplest, but least efficient way to control the GPIO is described on the manufacturer's own website, available at Control Overo GPIO. They indicate to control the GPIO by the Linux system terminal itself through a *sysfs system*. The *sysfs system* is a system of files offered by the Linux kernel for control and communication with devices and drivers through the Linux terminal.

If, for example, we want to control **GPIO10** output using this method to flash an LED, we need to export **GPIO10** to the user space by typing `10` in the file */sys/class/gpio/export*, which will generate a directory with other files for **GPIO10** manipulation. Next, we must define its direction as outgoing by writing `out` in */sys/class/gpio/gpio10/direction* and define its value as high or low by writing `1` or `0` in */sys/class/gpio/gpio10/value*.

---

**Tip:** The interrupt configuration function is also accessible from the terminal.

---

This process can be done by the user's terminal with the command `echo`, or by a program that opens this file and writes to it for us. For example, to control **GPIO146** via the terminal, we can execute the following commands (example used on the Gumstix website):

---

**Note:** Remembering that the command `echo test > folder/file` will overwrite the entire file with the word 'test' and the command `cat folder/file` will display the contents of the file.

---

```
root@overo# echo 146 > /sys/class/gpio/export
root@overo:/sys/class/gpio# cat gpio146/direction
in
root@overo# echo out > /sys/class/gpio/gpio146/direction
root@overo:/sys/class/gpio# cat gpio146/direction
out
root@overo# cat /sys/class/gpio/gpio146/value
0
root@overo# echo 1 > /sys/class/gpio/gpio146/value
```

(continues on next page)

(continued from previous page)

```
root@overo# cat /sys/class/gpio/gpio146/value
1
```

This command will control pin 27 on the Tobi board.

**Tip:** If you don't have a meter, a 1.8V LED can be used. Use pin 1 as the ground.

However, as already mentioned, this method is very slow and cannot be used for communication between devices. Thought, for activities over 100 milliseconds, this method can be used smoothly.

Another approach, using the same method, is to use code similar to the code shown below, which writes directly to **GPIO** files. This approach has been tested and has considerably improved, through a simple code, the GPIO response time.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>

int main()
{
    int arq = open("/sys/class/gpio/export", O_WRONLY);
    write(arq, "10", 2);
    close(arq);

    arq = open("/sys/class/gpio/gpio10/direction", O_WRONLY);
    write(arq, "out", 3);
    close(arq);

    arq = open("/sys/class/gpio/gpio10/value", O_RDWR);

    for (int i = 0; i < 10000; i++)
    {
        write(arq, "1", 1);
        //usleep (500000);
        write(arq, "0", 1);
        //usleep (500000) ;
    }
    close(arq);

    return 0;
}
```

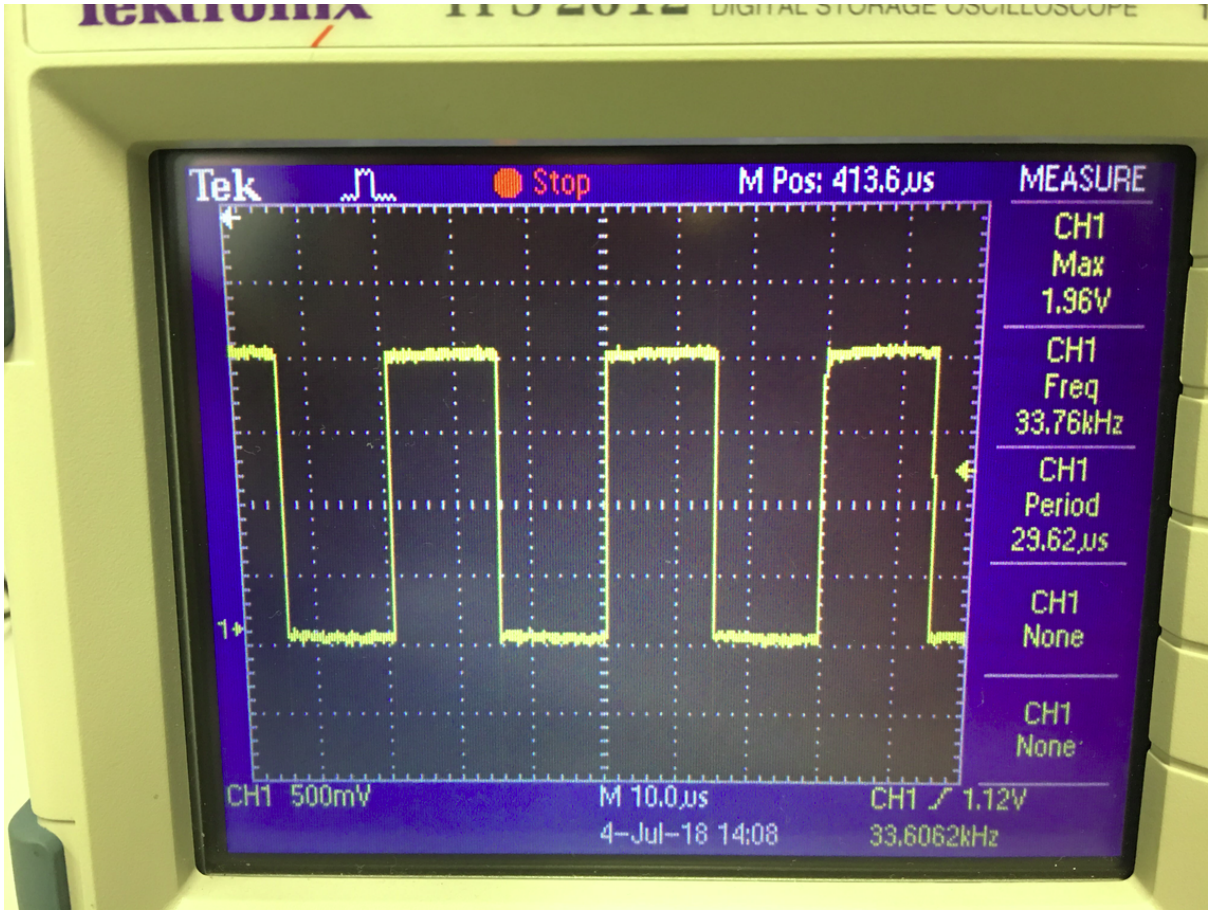
Downloaded commented code 1

To test the code, pin 18 (**GPIO** pin 10) was connected to an oscilloscope in order to measure the period of the waveform. The result of this measurement can be seen in the figure below, in which we can see the amplitude of the wave form of 1.96 V, frequency of 33.76 kHz and period of 29.62 microseconds. For most applications we can use this method.

### GPIO control via registers

Another way to control the GPIO is to write directly to the system registers. Although the procedure is a little more complex, this is actually the most common and recommended way to perform this procedure, offering much faster results.

To use this method, we first need to define which registers to write to and what to write to. This information




can only be found at `Technical Reference Manual (TRM)` of the DM3730 processor, available on the Texas Instruments website.

As explained in section **25** of the DM3730 processor TRM, starting on page 3477, the control interface combines six GPIO banks. Each GPIO module provides 32 pins, totaling 192 pins that can be used as input and/or output. In our case, only some of these 192 pins are physically accessible, as can be seen in the figure shown below. Each GPIO bank has 26 registers distributed from a base address, each register having a length of 32 bits or 4 bytes.

---

**Note:** The figure was taken from the Technical Reference Manual of the DM3730 processor and shows a little more detail how these pins are distributed among the GPIO modules. A detailed explanation of each of these registers can be found in the DM3730 processor manual.

---

In this work, only two of the registers will be commented in order to illustrate the functioning of these registers.

The register **GPIO_OE** is the register that defines the direction of the pin being configured. The abbreviation "OE" comes from "Output Enable". This register has an address offset equal to "0x034", that is, its address will be the base address of the GPIO module plus 34 in hexadecimal. This register has 32 bits of type "Read/White", or so, if the pin corresponding to the GPIO port is storing the value **0**, this GPIO port will be configured to operate as an output, if this pin is the value **1** the port will be configured as an input.

The register **GPIO_SETDATAOUT** is the register that has the function of setting the bit corresponding to the register **GPIO_DATAOUT** to 1. That is, if everything is set correctly, the voltage value equivalent to bit 1 will appear on the physical pin. This register has an offset address equal to "0x094". Like the register mentioned above, this register consists of 32 bits of type "RW". Reading any of the bits in this register returns the value of the corresponding bit in **GPIO_DATAOUT**.

In addition to the registers shown in section 25 of the Technical Reference Manual, it's also necessary to configure a register for the **System Control Module** (**SCM**). SCM is a module that allows control through software of various functions of the device. For our application, the SCM is the primary control point for the GPIO function

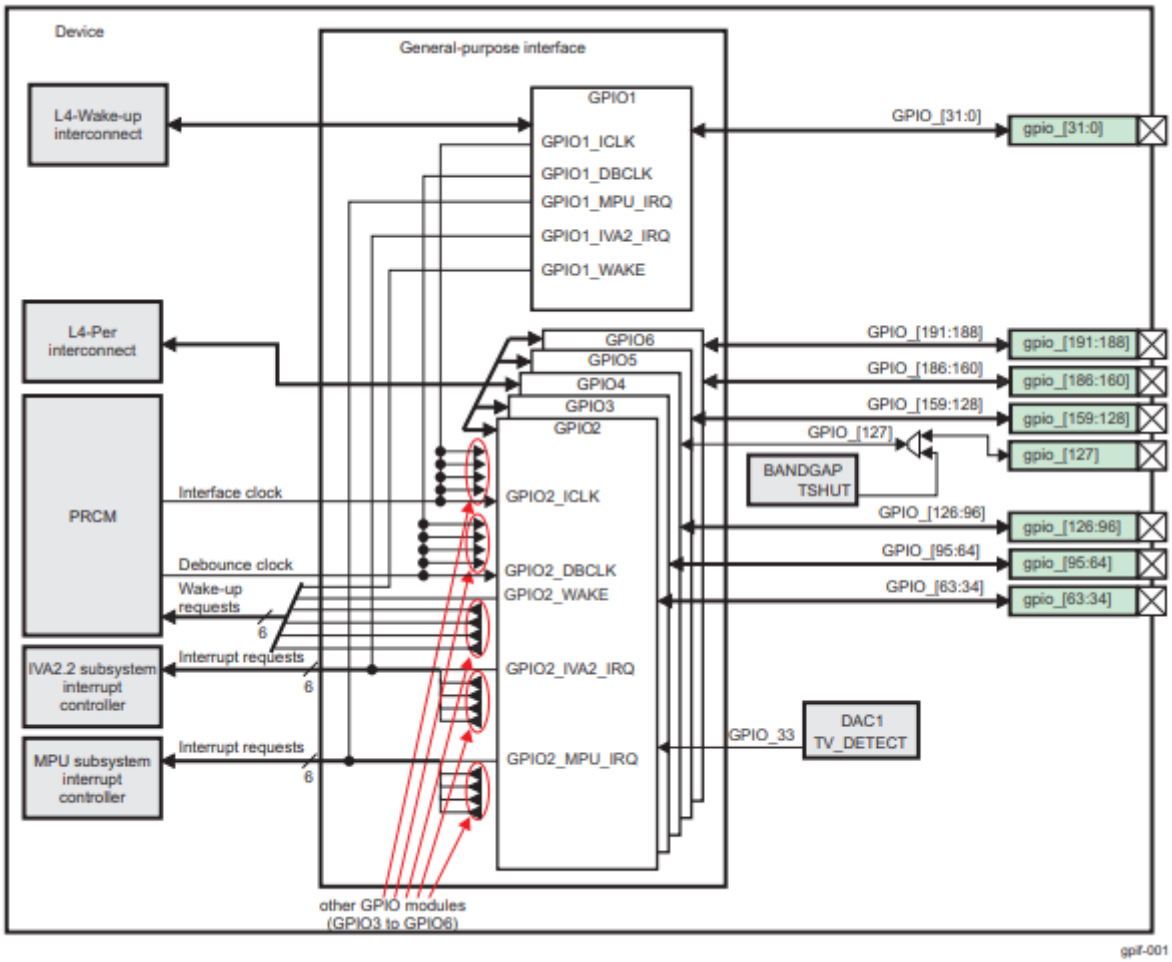



Fig. 7: Diagram of the GPIO interface.

and it is where we will perform the multiplexing, which determines whether the pin will operate in the GPIO function or in its specific function, and we will define whether the GPIO will be of the type pullup or pulldown, for example.

SCM registrars are divided into five classes. However, for our application we will use only one, the block of configuration and multiplexing registers. This block is a set of 32-bit registers, which configures 2 pins and defines, in addition to the two parameters mentioned above, the **wakeup** function. Registrars belonging to this block are called **Configuration Register Functionality**.

---

**Note:** More information about SCM can be found in section 13 of the Technical Reference Manual.

---

To find the address of each register of this type we can look in table 13-4 of the TRM. This table will be given the exact physical address of each register (base + offset). In this case, the base address is the address of the "PADCONFS" registers of the SCM interface, found in section 13.6.1 of the TRM and the offset address of each register in this block can be found in table 13-73 of the same document.

After the identification of the registrars, we can start the elaboration of a code to modify them. So we face yet another challenge, operating systems work with two concepts of memory, physical memory and virtual memory. Physical memory is the memory of the hardware, the one that we know the address and because we checked in the TRM. However, if we create a pointer that points to the memory "0x4800000", for example, it will not point to the physical memory that has this address because the operating system maps a different physical memory space for each program with the main objectives of increasing the security and avoid data conflicts between programs.

However, to have access to the physical memory of the system, we need to ask the operating system to map this memory space to the application. One way to do this is through the ´´mmap ()´´ function.

---

**Note:** Details on how this function works and its parameters can be found at mmap(2) — Linux manual page.

---

Let's assume that we want to map the physical memory space from **0x45000000** to **0x45001000** and for that we decided to use the `mmap ()` function. Therefore, we call the function as follows, for example, `mmap(NULL, 0x1000,PROT_WRITE || PROT_READ,MAP_SHARED,fd,0x45000000)`, by executing this the function will return a pointer that points to a virtual memory address addressed to the physical memory address **0x45000000**. Where, to access the physical memory of the device, "**fd**" is the **file descriptor** directed to "/dev/mem".

With this information, we have everything that is necessary to implement tests on this operating mode. Below is a code that applies the method described in this section to toggle the voltage level of pin **186**. This code was implemented to perform the same test as the section "GPIO control via terminal".

---

**Note:** The code below was obtained from the Gumstix Discussion Forum and minor changes were made to avoid excessive information and facilitate its understanding.

---

```
// Local includes definition
#include <stdio.h>    // for lprint instruction
#include <stdlib.h>
#include <fcntl.h>    // ok for mmap
#include <sys/mman.h> // ok for mmap
#include <unistd.h>

// Defines local parameters (from TRM)
#define SCM_INTERFACE_BASE 0x48002000
#define SCM_PADCONFS_BASE 0x48002030
#define CONTROL_PADCONF_SYS_NIRQ (*(volatile unsigned long *)0x480021E0)
#define CONTROL_PADCONF_SYS_NIRQ_OFFSET 0x1B0

#define GPIO6_BASE 0x49058000
#define GPIO6_SYSCONFIG_OFFSET 0x10
```

(continues on next page)

(continued from previous page)

```
#define GPIO6_CLEARDATAOUT_OFFSET 0x90
#define GPIO6_SETDATAOUT_OFFSET 0x94
#define GPIO6_OE_OFFSET 0x34
#define GPIO6_CTRL_OFFSET 0x30

#define MAP_SIZE (volatile unsigned long)4 * 1024
#define MAP_MASK (volatile unsigned long)(MAP_SIZE - 1)

// Defines "volatile unsigned long" how "u32"
#define u32 volatile unsigned long

// Defines commom variables
u32 *A;
u32 *B;

int main() // Local functions definition
{
    // Defines local variables
    unsigned long i;
    int fd;
    int j;

    fd = open("/dev/mem", O_RDWR | O_SYNC); // "O_RDWR" opens the file for reading␣
↪and writing & "O_SYNC" guarantees that the call will not return before all data␣
↪has been transferred to the disk

    A = (u32 *)mmap(NULL, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED, fd, SCM_
↪INTERFACE_BASE & ~MAP_MASK); // creates a new mapping in the virtual address␣
↪space

    *(u32 *)((u32)A + 0x30 + CONTROL_PADCONF_SYS_NIRQ_OFFSET) |= (0x00040000); //
↪set mode 4 on the pad 186 configuration register; enables digital pin use

    close(fd);
    /********/

    fd = open("/dev/mem", O_RDWR | O_SYNC);

    B = (volatile unsigned long *)mmap(NULL, MAP_SIZE, PROT_READ | PROT_WRITE, MAP_
↪SHARED, fd, GPIO6_BASE & ~MAP_MASK); // COM1 0x4806A000

    //gpio_186 handling
    *(u32 *)((u32)B + GPIO6_SYSCONFIG_OFFSET) |= 0x00000004; // bit2=1 enable/wake␣
↪up, free running clock

    //*(u32 *)((u32)B + GPIO6_CTRL_OFFSET) &= 0xfffffffe; // bit0=0 module enabled,
↪ clock not gated , clock=interface clock divided by 8

    *(u32 *)((u32)B+GPIO6_CTRL_OFFSET)&= 0xfffffff8;  // bit0=0,bit1=0,bit2=0␣
↪module enabled, clock not gated , clock=interface clock not divided

    *(u32 *)((u32)B + GPIO6_OE_OFFSET) &= 0xfbffffff; // bit26=0, gpio_186 output

    // generate a pulse stream on gpio_186 pin output

    for (j = 0; j < 1000000; j++)
    {
        *(u32 *)((u32)B + (GPIO6_CLEARDATAOUT_OFFSET)) |= 0x04000000;
        //printf("Saida = 0\n");
        //usleep(1000000);
```

(continues on next page)

(continued from previous page)

```
        *(u32 *)((u32)B + (GPIO6_SETDATAOUT_OFFSET)) |= 0x04000000;
        //printf("Saida = 1\n");
        //usleep(1000000);
    }
    close(fd);
    return (0);
}
```

Downloaded commented code 2

The code above was tested in the same way as the code presented in the previous section. In the following figure, you can see the result of this test. Note that the time obtained was 720.3 nanoseconds, that is, approximately 42 times faster than the result of the other method. Moreover, we can observe that the waveform is no longer an exact rectangular signal, the presence of a capacitive effect slowing the process is evident, therefore, it is possible that this is the maximum speed at which the signal of a pin can be changed .

Very hardly any application involving GPIO will not be satisfied by any of the methods presented here.

### Problems writing to registers

To conclude this last topic, it is necessary to highlight some recently encountered problems involving writing in registers.

The first problem encountered occurs whenever we try to change the value of registers "**0x49050030**", "**0x49056030**" and "**0x49058030**", responsible for controlling the clock of the entire block of "**GPIO_2**","**GPIO_5**" and "**GPIO_6**", respectively.

---

**Note:** `devmem2` is a command that executes a simple program to read or write in any memory space. More information can be found in devmem2 - Ubuntu Manual.

---

What happens is that moments after changing the value of the register, its returns the value it had before being changed. As the test in this section had a very high frequency, it was not interrupted by this effect, but the phenomenon occurs even when we change values of the registers by terminal commands, such as `devmem2`. This problem is exemplified in the figure below, where we execute the command `devmem2 0x49058030 w 0x2` to modify the register **0x49058030** which is the register that controls the clock of the entire **GPIO_6** block.

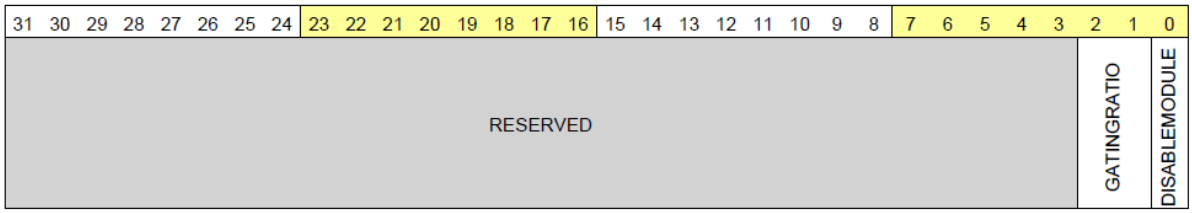
```
root@overo:~# devmem2 0x49050030
/dev/mem opened.
Memory mapped at address 0xb6ff5000.
Read at address  0x49050030 (0xb6ff5030): 0x00000002
root@overo:~# devmem2 0x49050030 w 0x1
/dev/mem opened.
Memory mapped at address 0xb6fef000.
Read at address  0x49050030 (0xb6fef030): 0x00000002
Write at address 0x49050030 (0xb6fef030): 0x00000001, readback 0x00000001
root@overo:~# devmem2 0x49050030
/dev/mem opened.
Memory mapped at address 0xb6fad000.
Read at address  0x49050030 (0xb6fad030): 0x00000002
root@overo:~#
```

Such modification should reduce the clock speed by dividing it by 2, as indicated in the Technical Reference Manual (TRM) of the DM3730 processor, in table 25-29, page 3528, where it is explained that the **GPIO_CTRL** may have its clock divided by certain pre-registered values, as shown in the following figure.

However, immediately after executing the command, a reading procedure is performed that ensures that everything was written in the register as expected. Though, the same command, executed moments later in read mode,

**Table 25-29. GPIO_CTRL**

| | | | |
|---|---|---|---|
| **Address Offset** | 0x030 | | |
| **Physical Address** | 0x4831 0030 | **Instance** | GPIO1 |
| | 0x4905 0030 | | GPIO2 |
| | 0x4905 2030 | | GPIO3 |
| | 0x4905 4030 | | GPIO4 |
| | 0x4905 6030 | | GPIO5 |
| | 0x4905 8030 | | GPIO6 |
| **Description** | This register controls the clock gating functionality. | | |
| **Type** | RW | | |




| Bits | Field Name | Description | Type | Reset |
|---|---|---|---|---|
| 31:3 | RESERVED | Read returns 0 | RW | 0x00000000 |
| 2:1 | GATINGRATIO | Gating Ratio | RW | 0x1 |
| | | 0x0: Functional clock is interface clock. | | |
| | | 0x1: Functional clock is interface clock divided by 2. | | |
| | | 0x2: Functional clock is interface clock divided by 4. | | |
| | | 0x3: Functional clock is interface clock divided by 8. | | |
| 0 | DISABLEMODULE | Module Disable | RW | 0x0 |
| | | 0x0: Module is enabled, clocks are not gated | | |
| | | 0x1: Module is disabled, clocks are gated | | |

always returns to the previously stored value, the value existing in the register before the modification. It's worth mentioning that this problem doesn't occur for the GPIO control method via terminal, this method operates until it receives a stop order from the user.

The second problem found occurs when we try to change the value of the registers **0x49052030** and **0x49054030**, responsible for controlling the clock of the entire block of **GPIO_3** and **GPIO_4**, respectively. In these specific registers, when trying to execute the command `devmem2` to change the clock of a given GPIO block or just perform a reading, the system returns the error "*bus error*" as shown in the figure below, where we execute the same command in register **0x49054030**.

```
root@overo:~# devmem2 0x49054030
/dev/mem opened.
Memory mapped at address 0xb6f67[  834.972076] Unhandled fault: external abort on non-linefetch (0x101
8) at 0xb6f67030
[  834.983886] In-band Error seen by MPU  at address 0
[  834.989044] ------------[ cut here ]------------
[  834.993896] WARNING: CPU: 0 PID: 1857 at /home/lucas/yocto/build/tmp/work-shared/overo/kernel-sourc
e/drivers/bus/omap_l3_smx.c:162 omap3_l3_app_irq+0xdc/0x130()
[  835.008972] Modules linked in: snd_soc_omap_twl4030 snd_soc_omap_mcbsp snd_soc_omap snd_pcm_dmaengi
ne snd_soc_twl4030 snd_soc_core snd_compress snd_pcm snd_timer snd soundcore twl4030_madc industrialio
 mt9v032 v4l2_common videodev edt_ft5x06 media usb_f_acm u_serial usb_f_ecm g_cdc u_ether libcomposite
 ipv6
[  835.037780] CPU: 0 PID: 1857 Comm: devmem2 Tainted: G        W      3.18.21-custom #1
[  835.046051] [<c00159b4>] (unwind_backtrace) from [<c001213c>] (show_stack+0x10/0x14)
[  835.054199] [<c001213c>] (show_stack) from [<c05f7b4c>] (dump_stack+0x84/0x9c)
[  835.061828] [<c05f7b4c>] (dump_stack) from [<c00427f4>] (warn_slowpath_common+0x6c/0x90)
[  835.070343] [<c00427f4>] (warn_slowpath_common) from [<c00428b4>] (warn_slowpath_null+0x1c/0x24)
[  835.079559] [<c00428b4>] (warn_slowpath_null) from [<c0335ef4>] (omap3_l3_app_irq+0xdc/0x130)
[  835.088562] [<c0335ef4>] (omap3_l3_app_irq) from [<c008fedc>] (handle_irq_event_percpu+0x3c/0x1d8)
[  835.097991] [<c008fedc>] (handle_irq_event_percpu) from [<c00900b4>] (handle_irq_event+0x3c/0x5c)
[  835.107330] [<c00900b4>] (handle_irq_event) from [<c0092c84>] (handle_level_irq+0xb4/0x144)
[  835.116119] [<c0092c84>] (handle_level_irq) from [<c008f5b4>] (generic_handle_irq+0x28/0x3c)
[  835.124969] [<c008f5b4>] (generic_handle_irq) from [<c008f898>] (__handle_domain_irq+0x64/0xc8)
[  835.134124] [<c008f898>] (__handle_domain_irq) from [<c00086c0>] (omap_intc_handle_irq+0xb4/0xc4)
[  835.143463] [<c00086c0>] (omap_intc_handle_irq) from [<c05fed24>] (__irq_svc+0x44/0x5c)
[  835.151885] Exception stack(0xdd14df50 to 0xdd14df98)
[  835.157196] df40:                                     00000001 00000001 00000000 de31c2c0
[  835.165802] df60: 00000001 dd14c018 00000000 dd14dfb0 dd14c000 dd14c000 b6f67030 bee80db4
[  835.174377] df80: 00000000 dd14df98 c0083434 c0011aec 200d0113 ffffffff
[  835.181335] [<c05fed24>] (__irq_svc) from [<c0011aec>] (do_work_pending+0x5c/0xc4)
[  835.189300] [<c0011aec>] (do_work_pending) from [<c000e6c0>] (work_pending+0xc/0x20)
[  835.197448] ---[ end trace d18b1a8ec835e19d ]---
000.
Bus error
root@overo:~#
```

Thus, it was only possible to change the **GPIO_1** block clock, as can be seen in the image below.

```
root@overo:~# devmem2 0x48310030
/dev/mem opened.
Memory mapped at address 0xb6fa6000.
Read at address  0x48310030 (0xb6fa6030): 0x00000000
root@overo:~# devmem2 0x48310030 w 0x1
/dev/mem opened.
Memory mapped at address 0xb6fa9000.
Read at address  0x48310030 (0xb6fa9030): 0x00000000
Write at address 0x48310030 (0xb6fa9030): 0x00000001, readback 0x00000001
root@overo:~# devmem2 0x48310030
/dev/mem opened.
Memory mapped at address 0xb6faa000.
Read at address  0x48310030 (0xb6faa030): 0x00000001
root@overo:~#
```

We do not know why these phenomena are occurring with blocks 2 to 6, but suspect that some processes of the operating system are preventing the clock of such blocks from being changed, probably by some internal circuit or operation depends on such pre-defined values or even some restriction on energy consumption.

### References


- PITA, H. C. Desenvolvimento de sistema de comunicação multiplataforma para veículos aéreos de asa fixa. Faculdade de Tecnologia, Universidade de Brasília, 2018.
- TEXAS INSTRUMENTS. AM/DM37x Multimedia Device Technical Reference Manual. 12500 TI Blvd, Dallas, TX 75243, EUA, 2012. Version R. Disponível em: ti.com.
- Direct register access control of GPIO ARM interface on Overo Water +TOBI - Gumstix Discussion Forum


### Serial Communication

The main characteristic of serial communication is the process of sending data one bit at a time, sequentially, through a communication channel or bus. Unlike parallel communication, where all bits of each symbol are sent together.

In order to enable the conversion, transmission and reception of data in a serial way, which were originally arranged in parallel, the **UART** format emerges, an acronym for Universal Asynchrounous Receiver/Transmiter.

The term "Universal" refers to the characteristic of the data format and speed being configurable, while "Asynchronous" refers to the way in which serial communication occurs, in which the devices are not continuously synchronized by a common clock signal.

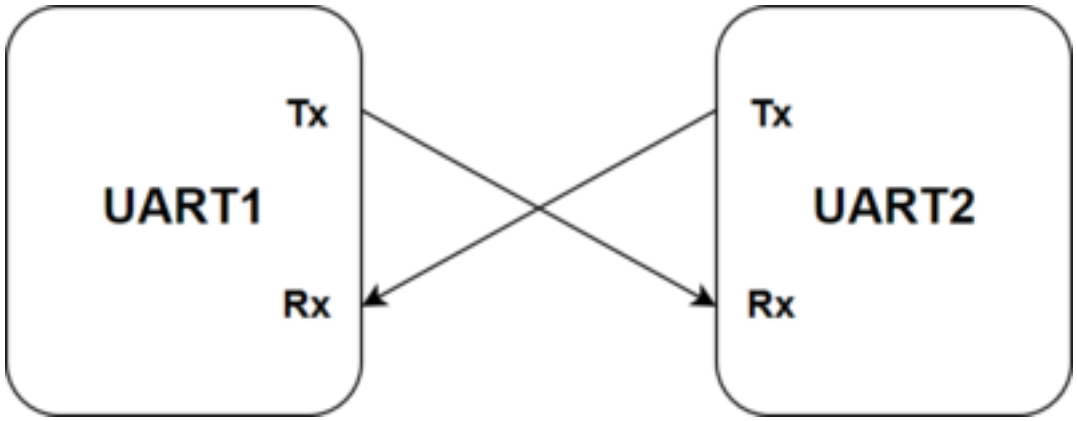
---

**Note:** More details on UART communication can be seen in Introduction to UART Communication - microcontrollerslab.com.

---

In the case of Overo, we have three Communication Systems **UART** implemented by hardware at our disposal. What makes any manual implementation through software using GPIO unnecessary.

Now let's understand how the UART communication protocol works. This communication works by connecting the transmitter (**TX**) of one device to the receiver (**RX**) of another device, in this case, only **TX** makes changes to the line voltage level , therefore, communication is, for each connection, a one-way street. Therefore, to perform a two-way communication we will use two connections, one will be the **RX** connection from device 1 with **TX** from device 2 and the other connection will be the opposite, **RX** from device 2 with **TX** from device 1, similar to the image below.



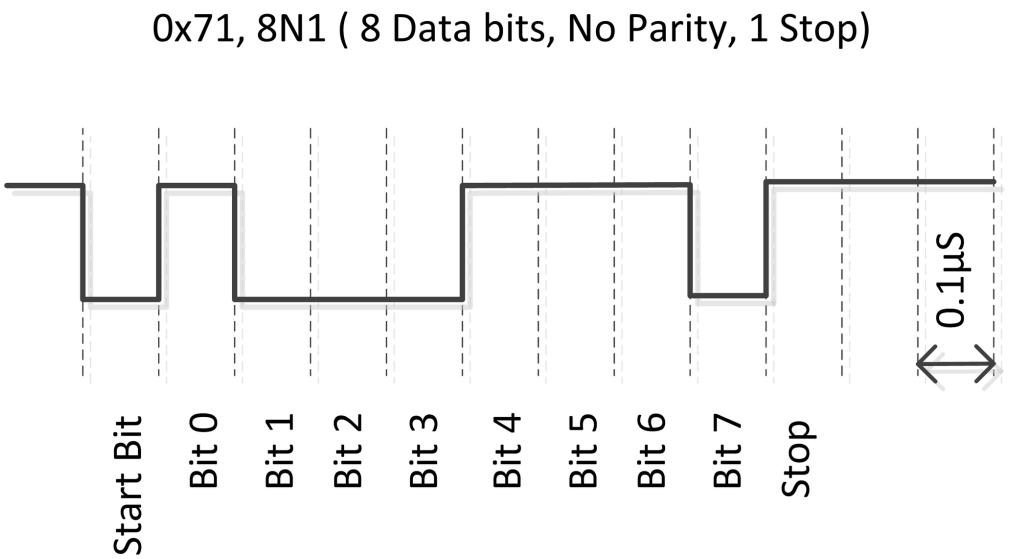

We can analyze the communication situation at bit level. For example, for a UART communication type "8N1" (8 data bits, 0 parity bits and 1 stop bit) we will have the channel in **IDLE** state, which means "not operating", represented by the static voltage level high. Therefore, when the intention is to start the communication, a low level pulse is sent and then the eight data bits are sent, which will be accompanied by a stop bit in a high state.

It's important to note that the function of the stop bit is to pause the transmission for some internal processing of the devices, therefore, there is no need for any additional time between the transmitted data.

As this is an asynchronous communication, it is essential that the speed of the communication is predetermined. This speed is usually given in Baud rate, a unit of measurement for the number of signal units sent per second.

**Note:** The term "Baud rate" is used to measure the speed of data transmission between devices. A baud is a measure of signaling speed and represents the number of changes in the transmission line (whether in frequency, amplitude, phase, etc.) or events per second.

The following figure shows a schematic illustrating how UART communication works. In the illustration, the communication speed is 10,000,000 baud/s.




### Particularities of Gumstix Overo

By default, the Tobi expansion card provides only two of the UARTs available on Overo module computer for use via its pins. As can be seen in the image below, UART1 is connected to pins 10 and 9 and UART3 is connected to pins 22 and 21. It's also important to say that the UART3 serial port is the same pin used by the **USB console**, ie , it's the same pin that we use to control Gumstix from the computer, that is, in standard configuration system messages and messages to the system are sent through this port.

If the two serial ports already mentioned are not enough, exist also the UART2 serial port. However, by default, it's not available on any of the pins on the Tobi board for our use. In fact, it was reserved to communicate with Bluetooth, yet only later versions of the Overo embedded computer used by us have Bluetooth, so we can, if necessary, export UART2 to the pins and use them. To use this serial port, it is necessary to modify *u-boot* in order to multiplex its function to GPIO 146/147 which, as shown in the previous figure, are connected to pins 29 and 27. Therefore, to do this it is necessary to modify the file "overo.h" removing the command lines for the GPIO mode from pins 146 and 147, removing the lines that disable UART2 and adding the lines that enable serial communication over UART2.

To understand, in detail, what needs to be done and which registers will be changed, consult the serial communication section of the processor's Technical Reference Manual (TRM). However, there is a topic in the Gumstix Discussion Forum that directly indicates what changes must be made to the " u-boot" in order to use UART2, although the solution presented in that forum was not tested during this work.

### UART configuration

As previously mentioned, the embedded computer has specific hardware for UART communication, in other words, it's not necessary to perform a manual implementation to use UART communication, just write in some registers to send the message.

In fact, in our case it is even simpler because the installed operating system already has configured drivers for the application of serial communication. Therefore, it isn't necessary to access the physical memory of the device, we

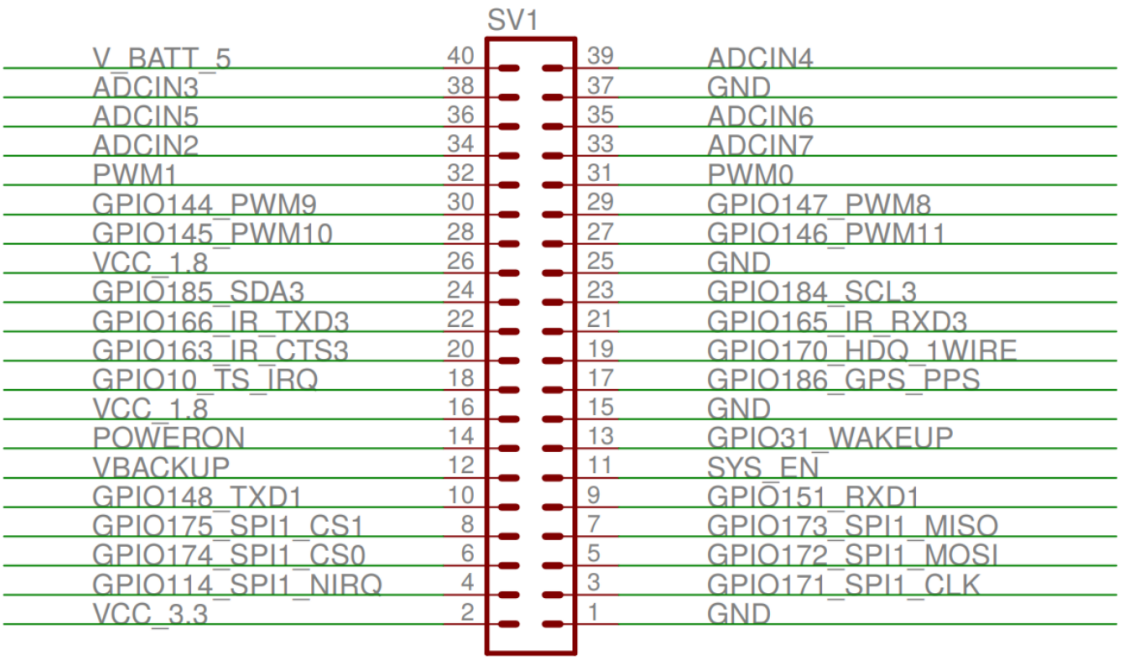



Fig. 8: Pin diagram of the tobi expansion card.

just need to write in the driver what should be transmitted.

The serial communication drivers are files of type character, named "**ttyOx**", where "**x**" represents the unique number of each UART. These drivers are located in "**/dev**" and function as a communication terminal.

For example, the "**ttyO2**" driver is the serial communication driver for the **USB Console** port, the same one we connect to the computer, that is, when writing to this port we will write on the computer connected to Gumstix and when reading this port we will be reading the computer. In other words, writing or reading in this driver will have the same final result of calling, respectively, the function `printf ()` or `scanf ()`, when a computer is connected to that port with the terminal open.

The configuration of the serial ports can be done in two ways, by command lines in the Linux terminal or by a code that changes the hardware settings. The simplest and, again, the most limited or least efficient is the configuration through command lines, the configuration in this way is usually used only when done by a human user in real time.

To perform the configuration through the Linux terminal, we must use the command `stty`, since this command has a huge number of parameters that allows to establish serial communication in the desired way.

---

**Note:** To view all parameters of the `stty` command, just run `stty --help` in the terminal.

---

If, for example, the command line `stty -F / dev / ttyO0 -a` is executed, all the serial communication settings **UART1** of the device will be printed. To print only the main settings, you must delete the `-a`, the last option of the command. If changing the speed is desirable, it can be changed simply by adding the desired speed to the end of the command line.

The figure below shows an example of configuring UART1 using the Linux command terminal.

The other way to configure the serial communication made by these drivers without manually changing the contents of the physical address of the memory is with the aid of the "**termios.h**" library. This library has a wide variety of functions that configure serial communication based on the parameters of a "**termios**" structure, also defined in this library.

---

**Note:** More information about the **termios.h library** can be found at termios.h - Linux manual page.

---

```
root@overo:~# stty -F /dev/ttyO0
speed 9600 baud; line = 0;
-brkint -imaxbel
root@overo:~# stty -F /dev/ttyO0 raw 115200
root@overo:~# stty -F /dev/ttyO0
speed 115200 baud; line = 0;
min = 1; time = 0;
-brkint -icrnl -imaxbel
-opost
-isig -icanon
root@overo:~#
```

There are two parameters of UART communication, in addition to those mentioned above, which stand out, the minimum number of bits that are expected to be read at each reading attempt and the maximum time to wait for a new character after the transmission of the last character and after the minimum number of characters to be reached.

The minimum number of bits expected to be read and the maximum wait time for the next bit in tenths of a second can be configured with the following commands `termios.c_cc [VMIN] =` and `termios.c_cc [VTIME] =`, where `termios` is the name of its structure. For the speed setting, it is recommended to use the function `cfsetspeed ()`, while the function `cfmakeraw ()` configures, in addition to other parameters, the operation without parity bit and with 8 data bits. After making the adjustments to the structure, it is still necessary to execute the function `cfsetattr ()` for the changes to be made in the UART.

Below is the code used to configure the serial communication of Overo computers. Note that in this configuration function the "O_NONBLOCK" flag was not used in the "open ()" function and the minimum number of characters to be returned after an attempt to read was set to 1, so if the code is executed and no information is entered sent to this channel the processor will wait forever for that character. The time count, set to 0.1 second, does not start until the minimum number of characters has been reached.

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>

void main()
{
    struct termios cUART1;
    int UART1 = open("/dev/ttyO0", O_RDWR);

    if(tcgetattr(UART1,&cUART1))
        printf("Erro tcgetattr");
    cfmakeraw(&cUART1);
    cfsetspeed(&cUART1,B115200);
    cUART1.c_cflag &= ~CSTOPB;

    cUART1.c_cc[VMIN] = 1;
    cUART1.c_cc[VTIME] = 1;
    if (tcsetattr(UART1, TCSANOW, &cUART1))
        printf("Erro tcsetattr");

}
```

Download commented code

The following figure shows an example of UART1 configuration using the configuration code above.

**Note:** In order to simplify the configuration of the UART within another code, some modifications were made to the previous code to convert it into a function for configuring serial communication, as shown below:

```
root@overo:~# stty -F /dev/ttyO0
speed 9600 baud; line = 0;
-brkint -imaxbel
root@overo:~# stty -F /dev/ttyO0 raw 115200
root@overo:~# stty -F /dev/ttyO0
speed 115200 baud; line = 0;
min = 1; time = 0;
-brkint -icrnl -imaxbel
-opost
-isig -icanon
root@overo:~#
```

```
int configUART1()
{
    struct termios cUART1;
    int UART1 = open("/dev/ttyO0", O_RDWR);

    if(tcgetattr(UART1,&cUART1))
        printf("Erro tcgetattr");
    cfmakeraw(&cUART1);
    cfsetspeed(&cUART1,B115200);
    cUART1.c_cflag &= ~CSTOPB;

    cUART1.c_cc[VMIN] = 1;
    cUART1.c_cc[VTIME] = 1;
    if (tcsetattr(UART1, TCSANOW, &cUART1))
        printf("Erro tcsetattr");

    return UART1;
}
```

Once the configuration was made, the following code was also implemented in order to test the communication between two computers. In the test, one device sends a message to the other device that responds with a similar message to the first device, then both devices print the received message.

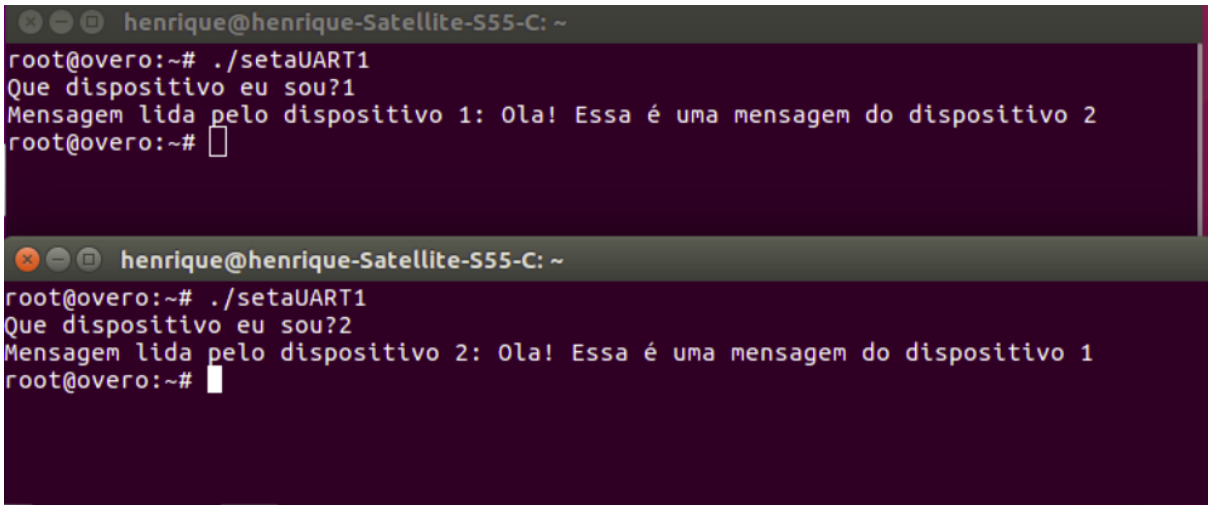
```
int main()
{

    int UART1 = configUART1();  // call the UART configuration function
    char dis[2], out[100], string[100];

    printf("What device am I?");
    scanf("%c", &dis[0]);
    dis[1] = 0;
    string[0] = 0;
    strcat(string, "Hello! This is a message from the device");
    strcat(string, dis);

    // testa UART
    write(UART1, string, strlen(string));
    sleep(1);
    read(UART1, out, 100);
    printf("Message read by the device %s: %s\n", dis, out);
    close(UART1);
    return 0;
}
```

Download the complete code

Since the two devices are identical, it will be necessary to connect pin 10 of one device with pin 9 of the other

device and vice versa. Using this code as a basis it is possible to send any message of up to 100 characters from one device to the other.

The following figure shows the result of testing the codes presented. In this figure we can see two Linux terminals, each linked to an embedded computer, and both call the same function, right after that we see the message read by each of the devices.




#### References


- PITA, H. C. Desenvolvimento de sistema de comunicação multiplataforma para veículos aéreos de asa fixa. Faculdade de Tecnologia, Universidade de Brasília, 2018.
- Universal asynchronous receiver-transmitter - wikipedia.org
- Asynchronous serial communication - wikipedia.org
- Como funcionam as UARTs - newtoncbraga.com.br
- UART Basics - ece353.engr.wisc.edu
- termios.h(0p) - Linux manual page - man7.org
- cfsetspeed(3) - Linux man page - linux.die.net

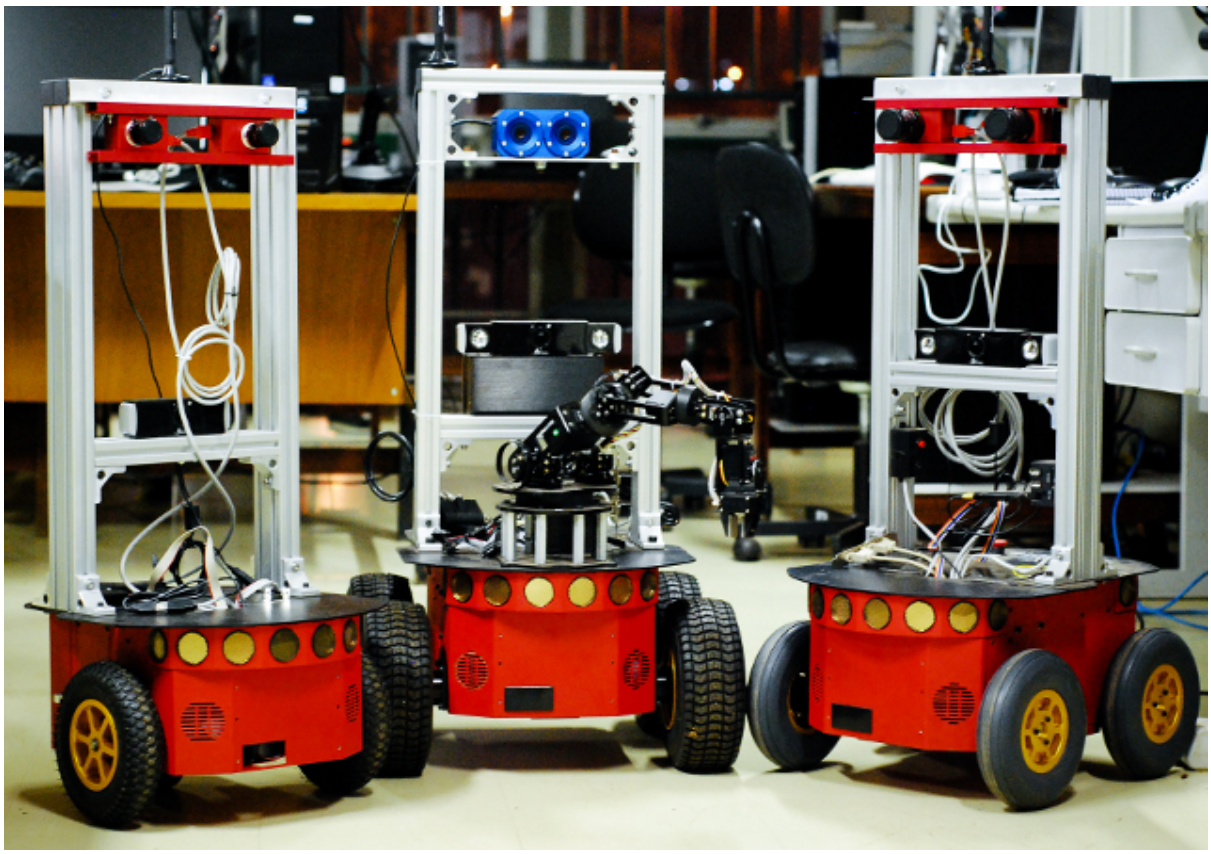

### 1.1.5 References


- ROCHA, E. M. C. Desenvolvimento de um sistema com veículos aéreos não-tripulados autônomos. Faculdade de Tecnologia, Universidade de Brasília, 2017.
- CORDEIRO, T. F. K. Desenvolvimento de um sistema com veículos aéreos não-tripulados autônomos. Faculdade de Tecnologia, Universidade de Brasília, 2018.
- PITA, H. C. Desenvolvimento de sistema de comunicação multiplataforma para veículos aéreos de asa fixa. Faculdade de Tecnologia, Universidade de Brasília, 2018.
- PX4 Autopilot User Guide - docs.px4.io
- QGroundControl User Guide - qgroundcontrol.com
- Ardupilot Docs - ardupilot.org
- RT-MaG Project - gipsa-lab.fr
- Yocto Project - yoctoproject.org
- Getting Started - Gumstix COM - gumstix.com
- Gumstix, Inc - GitHub - github.com

CHAPTER 2

# Mobile Robots

## 2.1 Pioneer



### 2.1.1 Kinematics

Kinematics, in classical mechanics, study the description of the motion of points, bodies (objects), and groups of bodies without considering the forces that cause them to move. In mobile robotics, kinematics helps us to understand and quantify the constraints about the robot de-

sign, which implies restrictions in its movement. Also, we can draw the paths and trajectories that the robot can do. Here, the text focus on wheeled robots, which can only move in 2D space.

Fig. 2: Pioneer P3-AT

The P3-DX is a two-wheel-drive and P3-AT is a four-wheel-drive. The Omron Adept MobileRobots considers in its manual the P3-DX as a differential drive robot and the P3-AT as a skid/slip drive robot. For our sake, both are modeled as differential drive vehicles because the exact center of rotation in a skid/slip drive is unpredictable[1].

In order to represent the motion of a mobile robot, we must define the reference frames and determine their position. There are two essential frames to a robot if we consider the robot as a rigid body. They are the global reference frame, that is world fixed, and the local reference frame, which is robot fixed.

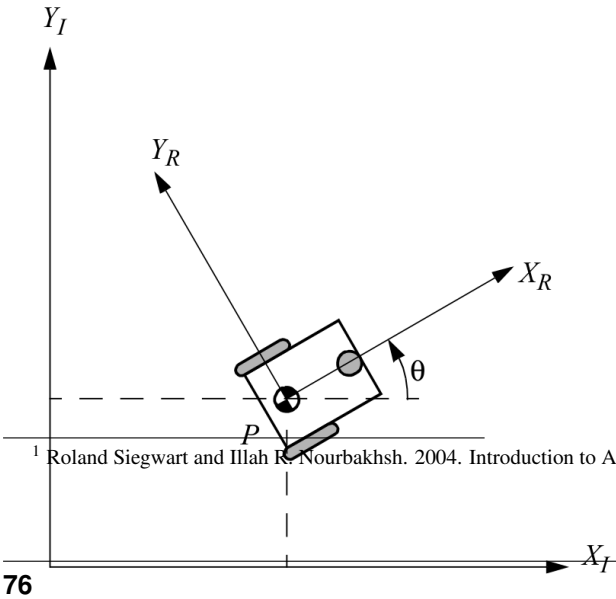



Fig. 3: The global reference frame and the robot local reference frame. Figure from[1].

The figure shows a robot and its reference frames. Where the $X_I$ and $Y_I$ defines the global reference frame, also known as the inertial frame, and $X_R$ and $Y_R$ defines the local reference frame or the robot frame. The coordinates $x$ and $y$ represent the robot's position in the global reference frame, point P, whereas $\theta$ is the angular difference among the global and the local reference frames. Thus, we represent the robot's pose as the vector with these three components.

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

[1] Roland Siegwart and Illah R. Nourbakhsh. 2004. Introduction to Autonomous Mobile Robots. Bradford Company, USA.

**Note:** We could still assume the robot as a rigid body free in the space, with 6 degrees of freedom, $[x, y, z, \phi, \psi, \theta]^T$. However, as the robot is moving subject to gravity, which keeps it confined to a euclidian plane, $x$ and $y$ describe the robot's position, and $\theta$ describes the orientation in the plane. The 2D space in which the robot lies is called the *C-space*[5], firstly formalized in[6], also called the configuration space. A configuration is a complete specification of the position of every point in the system. The space of all configurations is the *C-space*.

We can now utilize this definition to describe elements represented in the local frame in the global frame and vice-versa. For example, we can map the motion calculated in the global frame to motion in the robot's local frame. Or, we can map obstacles sensed by the robot in terms of the global reference frame.

$$R(\theta) = \begin{bmatrix} cos\theta \\ sin\theta \\ 0 \\ -sin\theta \\ cos\theta \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

The *orthogonal rotation matrix* does the map between these frames as a function of the robot's current pose.

$$\dot{\xi_R} = R(\theta)\dot{\xi_I}$$

**Note:** It does not make sense to talk about the robot's pose in the local frame. Because, the point P, the origin of the coordinates, moves around with the robot. So, the robot's pose in the local frame is, always, $\xi_R = [0, 0, 0]^T$. That is why the relationship is between the derivative of the pose in the frames.

### Wheels and its constraints

"Wheeled Mobile Robots (WMR) constitute a class of mechanical systems characterized by kinematics constraints that are not integrable and cannot, therefore, be eliminated from the model equations"[2]. If we want to study and describe a robot motion, we also must specify which are the hypotheses and constraints of the wheels. There are three essential hypotheses about the kinematics model of the wheeled robot during the motion; they are:

- Each wheel remain perpendicular about its plane;
- There is only one contact point between plane and wheel;
- There is only rolling without slipping;

And two constraints:

- About rolling: the motion component along the wheel plane is equal to the rotation velocity of the wheel;

[5] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki and S. Thrun, "Principles of Robot Motion: Theory, Algorithms, and Implementations," MIT Press, Boston, 2005.

[6] Lozano-Perez, "Spatial Planning: A Configuration Space Approach," in IEEE Transactions on Computers, vol. C-32, no. 2, pp. 108-120, Feb. 1983.

[2] G. Campion, G. Bastin and B. Dandrea-Novel, "Structural properties and classification of kinematic and dynamic models of wheeled mobile robots," in IEEE Transactions on Robotics and Automation, vol. 12, no. 1, pp. 47-62, Feb. 1996.

- About slipping: the motion component along the orthogonal direction is equal to zero;

Some authors may call these constraints as “the pure rolling and rotation condition”.

### Differential Drive Model

Now we delve into modeling of a differential drive kinematic. Dudek et al. say that the differential drive consists of two wheels mounted in the same axis with separated motors[3]. Each wheel contributes to the robot motion, so to fully describe the robot motion, we must compute each contribution.

The image shows the robot, the wheel velocities, and the local frame. $\dot{\phi}_1$ and $\dot{\phi}_2$ is the spin speed of the left wheel and right wheel. $r$ is the wheel radius, while the distance between the two wheels is $l$. While $v_1$ is the left wheel velocity along the ground and $v_2$ the right wheel velocity. As the wheels contribute independently to the robot motion, we can analyze each contribution separately.

$$v_i = \frac{\dot{\phi}_i r}{2}$$
$$\omega_i = \frac{\dot{\phi}_i r}{2l}$$
$$\text{where } i = \{1, 2\}$$

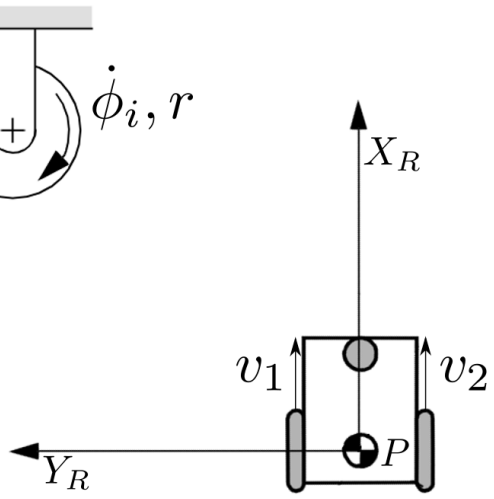



Fig. 4: Wheel velocities and the robot frame.

Point $P$ is halfway between the two wheels, so each wheel contributes with half of the linear speed of the robot in the direction of $X_R$. Each wheel also adds a new component to the angular speed of the robot. $v_1$ moves the robot clockwise around point $P$ while $v_2$ moves it counter-clockwise. That is why they differ in their sign. And, using the equation which relates the angular speed of disk with its linear speed, we have the above equations.

Using the superposition theorem, we have the equations for the linear velocity in the direction of $X_R$ and the angular velocity in the direction of $Z_R$:

$$v = v_1 + v_2$$
$$\omega = -\omega_1 + \omega_2$$

[3] Gregory Dudek and Michael Jenkin. 2010. Computational Principles of Mobile Robotics (2nd. ed.). Cambridge University Press, USA.

In the local frame, we have the following kinematic equation:

$$\dot{\xi_R} = \begin{bmatrix} \frac{r}{2} \\ \frac{r}{2} \\ 0 \\ 0 \\ -\frac{r}{2l} \\ \frac{r}{2l} \end{bmatrix} \begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \end{bmatrix}$$

**Note:** In the robot frame, there is no velocity in the direction of $Y_R$. Because we assumed the pure rolling and rotation condition. And yet he can reach any point in the global frame.

### Forward Kinematics

The forward kinematics problem tries to solve the problem when we have the control inputs, and we must know where the robot goes in the global frame. As we have seen, to solve this question, we should know five parameters of the robot — two parameters about the robot geometry, $l$ and $r$, the current robot orientation, $\theta$, and, at least, the two inputs, $\dot{\phi}_1$ and $\dot{\phi}_2$.

$$\dot{\xi}_I = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = f(l, r, \theta, \dot{\phi_1}, \dot{\phi_2})$$

$f$ is the function that solves the forward kinematics problem. To map between the parameter vector, $\{l, r, \theta, \phi_1, \phi_2\}$, and the state of the robot in the inertial frame. We should use the matrix, which links the spin speed and the derivative of the robot state in the local frame. Then, we can transform the robot velocities in the local frame to the global frame utilizing the inverse of the rotation matrix.

$$R(\theta)^{-1} = \begin{bmatrix} cos\theta \\ -sin\theta \\ 0 \\ sin\theta \\ cos\theta \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix},$$

$$\dot{\xi}_I = R(\theta)^{-1}\dot{\xi}_R,$$

$$\dot{\xi_R} = \begin{bmatrix} \frac{r}{2} \\ \frac{r}{2} \\ 0 \\ 0 \\ -\frac{r}{2l} \\ \frac{r}{2l} \end{bmatrix} \begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \end{bmatrix}$$

$$\dot{\xi}_I = R(\theta)^{-1} \begin{bmatrix} \frac{r}{2} \\ \frac{r}{2} \\ 0 \\ 0 \\ -\frac{r}{2l} \\ \frac{r}{2l} \end{bmatrix} \begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \end{bmatrix}$$

Then,

$$f(l, r, \theta, \dot{\phi}_1, \dot{\phi}_2) =$$

$$\begin{bmatrix} cos\theta \\ -sin\theta \\ 0 \\ sin\theta \\ cos\theta \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} \frac{r}{2} \\ \frac{r}{2} \\ 0 \\ 0 \\ -\frac{r}{2l} \\ \frac{r}{2l} \end{bmatrix} \begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \end{bmatrix}$$

$$f(l, r, \theta, \dot{\phi}_1, \dot{\phi}_2) =$$

$$\begin{bmatrix} \frac{rcos\theta}{2} \\ \frac{rcos\theta}{2} \\ \frac{rsin\theta}{2} \\ \frac{rsin\theta}{2} \\ -\frac{r}{2l} \\ \frac{r}{2l} \end{bmatrix} \begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \end{bmatrix}$$

Or

$$\dot{\xi}_I =$$

$$\begin{bmatrix} \frac{rcos\theta}{2} \\ \frac{rcos\theta}{2} \\ \frac{rsin\theta}{2} \\ \frac{rsin\theta}{2} \\ -\frac{r}{2l} \\ \frac{r}{2l} \end{bmatrix} \begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \end{bmatrix}$$

**Note:** The matrix which maps spin speed to the robot velocities is commonly known as **Jacobian Matrix**. "The Jacobian maps configuration velocities to workspace velocities"[5].

Well, we know the relationship between spin speeds and robot velocities, but what about the robot pose in the global frame?

$$\xi_I = \int_0^t \begin{bmatrix} \frac{rcos\theta}{2} \\ \frac{rcos\theta}{2} \\ \frac{rsin\theta}{2} \\ \frac{rsin\theta}{2} \\ -\frac{r}{2l} \\ \frac{r}{2l} \end{bmatrix} \begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \end{bmatrix} dt$$

Or

$$\begin{cases} x(t) = \frac{r}{2} \int_0^t (\dot{\phi}_1(t) + \dot{\phi}_2(t)) cos(\theta(t)) dt \\ y(t) = \frac{r}{2} \int_0^t (\dot{\phi}_1(t) + \dot{\phi}_2(t)) sin(\theta(t)) dt \\ \theta(t) = \frac{r}{2l} \int_0^t (\dot{\phi}_2(t) - \dot{\phi}_1(t)) dt \end{cases}$$

### Inverse Kinematics

The inverse kinematics problem is the opposite of the forward problem. The problem aims to solve the following question: "Given the desired pose, which are the controls needed to reach the desired pose?". We already know the relationship between the velocity and

$$\begin{bmatrix} \dot{\phi}_1 \\ \dot{\phi}_2 \end{bmatrix} = g(\dot{\xi}_I)$$

The function $g$ is the mathematical inverse of the function $f$.

$$g = f^{-1} = \begin{bmatrix} \frac{rcos\theta}{2} & \frac{rcos\theta}{2} \\ \frac{rsin\theta}{2} & \frac{rsin\theta}{2} \\ -\frac{r}{2l} & \frac{r}{2l} \end{bmatrix}^{-1}$$

As we can see, the matrix which represents the function $f$ is not invertible. The forward kinematics is an easy problem because we have one and only one solution. Nevertheless, the inverse kinematics is often not analytically solvable; commonly, we have more than one solution or none. However, we can try to solve the problem, limiting the possibles solutions like $\dot{\phi}_1 = \dot{\phi}_2$ or $\dot{\phi}_1 = -\dot{\phi}_2$.

### Straight Line

If we limit the solution to $\dot{\phi}_1 = \dot{\phi}_2 = \dot{\phi}$, with $\dot{\phi} > 0$, the robot should move along a straight line. Then, the robot motion simplifies to:

$$\xi_I = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + vcos(\theta)\delta t \\ y + vsin(\theta)\delta t \\ \theta \end{bmatrix}$$

### Rotaion in place

Similarly, if we limit the solution to $-\dot{\phi}_1 = \dot{\phi}_2$, with $\dot{\phi}_2 > 0$, the robot should rotate in the place around the point P.

$$\xi_I = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta + \frac{2v}{l}\delta t \end{bmatrix}$$

### Motion Composition

If we would like to drive the robot from any pose to some other pose in the global frame, we can decompose the motion in two rotations in place and one translation along a straight line. The robot can turn in the place aligning its orientation aiming the goal position, $(x_d, y_d)$, then move forward to the goal position, and then turn in the place again to reach the goal orientation, $\theta_d$.

The image above tries to illustrate the proposed motion. The robot starts with the $\xi_I = [x, y, \theta]^T$. Then it spun around the point $P$ and aim the desired position $P' = (x_d, y_d)$ reaching the pose $\xi'_I = [x, y, \theta_1]^T$. To reach the position, it moves forward to $P' = (x_d, y_d)$ and reaches $\xi''_I = [x_d, y_d, \theta_1]^T$. And then the robot spun again to from $\theta_1$ to $\theta_d$. The final robot state should be $\xi'''_I = [x_d, y_d, \theta_d]^T$.

### The Unicycle Model

So far, we saw the kinematics of a two-wheeled robot. But now we talk about a more general and simple model. The previous model tells us how a robot with two wheels can reach a specific pose in the world, acting in the wheel speeds. But, we do not care about how the wheel is turning; we care about the pose of the robot. The unicycle model represents a robot with only one wheel. If the wheel complies with our pure rotation and rolling condition, the wheel has two control inputs, the linear velocity, $v$, in the axis $X_R$ and the angular velocity, $\omega$, around $Z_R$. So,

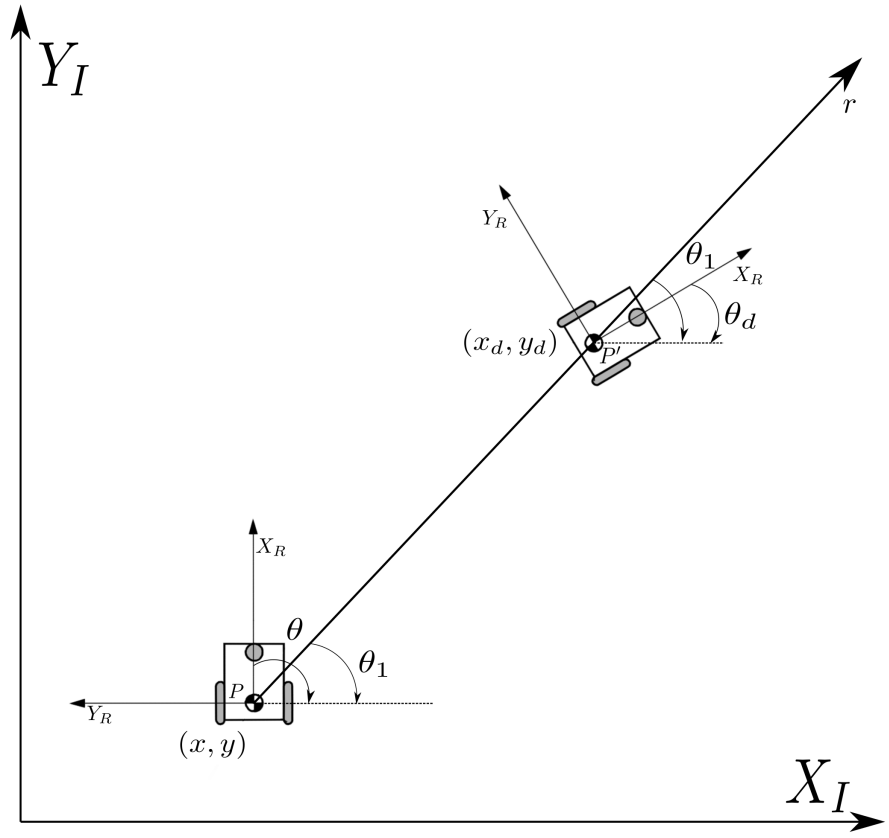



Fig. 5: A robot is moving around with the proposed motion framework.

the kinematics of a unicycle robot described in the inertial frame $\{X_I, Y_I, \}$ is given by

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\theta & \\ 0 & \\ \sin\theta & \\ 0 & \\ 0 & \\ 1 & \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

Where $x$, $y$ and $\theta$ are the coordinates of the robot in the global frame and $u = (v, \omega)$ is the control vector.

Commercial robots usually provide an interface to translate from a desired unicycle control input to desired wheel velocities. And a lower level dedicated microcontroller, which aims to control the wheel velocities.

### Notes on Control

So, we should be able to build a system or software capable of, using the maths showed, move a robot to any reachable goal. The control theory is the branch of maths dedicated to this problem. A control system sends inputs to the system and leads the variables of the system to the desired goal. Our system is a mobile robot. And, using the previous equations, the inputs are the spin speed of each wheel, and the output is the pose of the robot.

A controller should give the system the inputs necessary to perform the desired action. As in the image below:

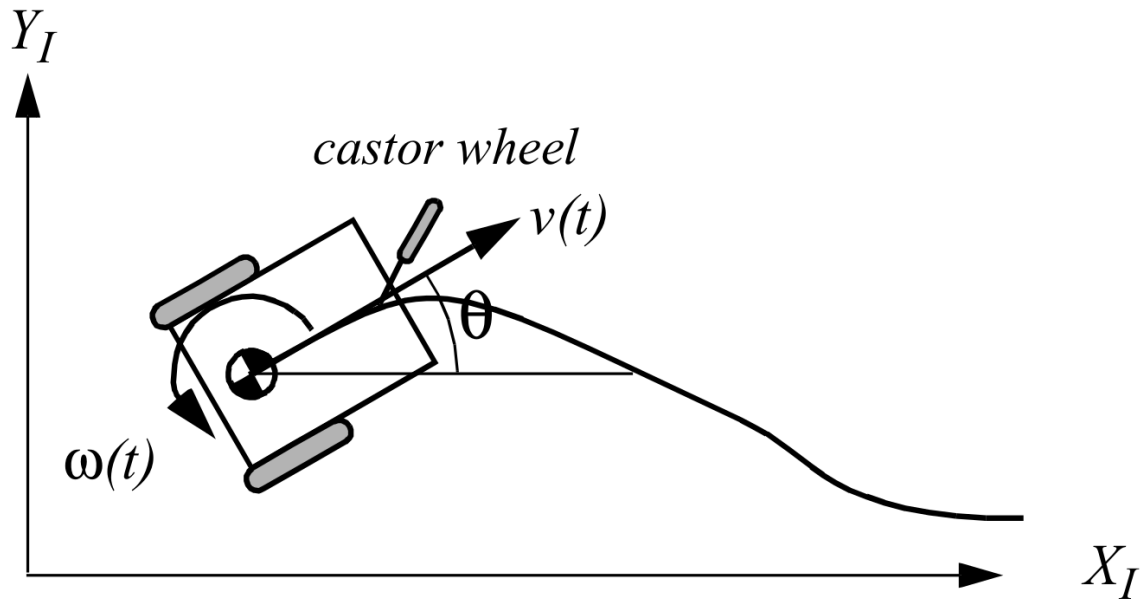
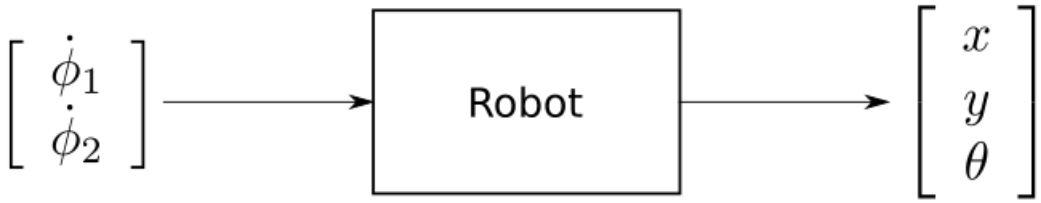
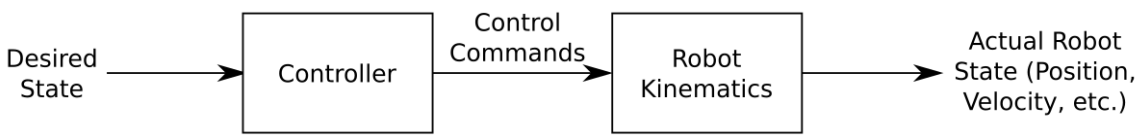
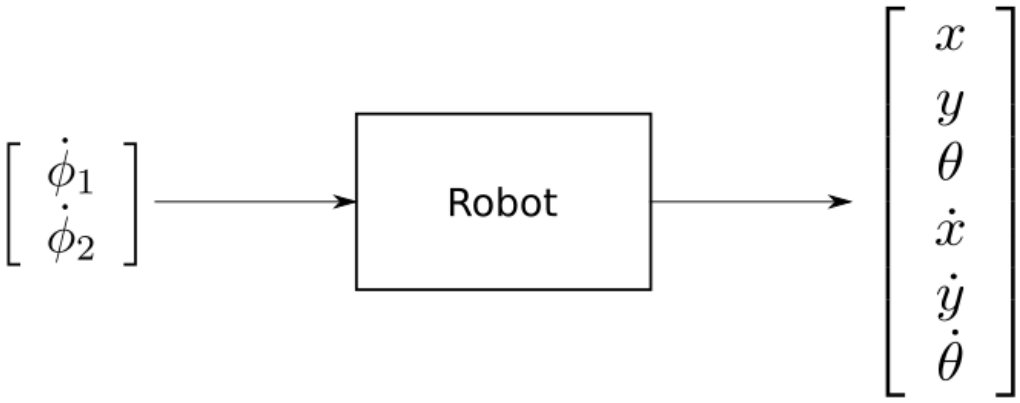



Fig. 6: A differential-drive robot in its global reference frame. Figure from[1].



If we see the controller and the robot as a single system, we can have another system with the desired state as input and the robot state as output. Then we can build a new controller which deals with choosing the desired state. In the same manner, if we would like to control the velocities of the robot and not only the pose, to be able to control how the robot moves. We can add the velocities to the robot state vector and control them with the equations related.




**Note:** A differential drive robot has a major problem which is... Feng et al.[4] develops in 1993 a motion controller which...

### 2.1.2 Probabilistic Kinematics

In the previous section, we try to model the movement of a mobile robot mathematically. As we have seen, we made assumptions to get a simple model capable of describing the movement without loss of generality. Nevertheless, if the assumptions are incorrect, we have to put in the trash all our work? Or, we can make adjustments to continue to describe the actions with the same simplicity of the previous model?

Well, the purpose of this section is to delve into the assumptions. Try to predict when they are incorrect. Furthermore, study a way that could deal with this kind of problem.

In the previous sections, we build a mathematical model looking at a mobile robot modeled with just two wheels. However, our robots have more than two wheels. The P3-DX has three wheels, two motored, and a castor wheel. The P3-AT has four wheels and two motors, one for each side. None of them fits in our model. The castor wheel adds a new problem in the differential drive model. Moreover, the four wheels violate the pure rolling and rotation condition.

#### The castor wheel problem

The castor wheel, unlike fixed wheels, is an off-centered wheel that is orientable to the robot frame.

The castor wheel does not add any additional constraints to the robot motion, because the robot motion will steer the wheel to a new orientation. However, in a castor wheel, the steering action itself moves the robot chassis because of the offset between the ground contact point and the vertical axis of rotation.

The steering action adds uncertainty to the robot motion. As we can not predict the additional action, we can not guarantee that the control action will drive the robot to the desired pose.

Fig. 7: Castor wheel.

[4] L. Feng, Y. Koren and J. Borenstein, "Cross-coupling motion controller for mobile robots," in IEEE Control Systems Magazine, vol. 13, no. 6, pp. 35-43, Dec. 1993.

Fig. 8: Castor wheel producing a unpredictable behavior.

The above image shows the simulation of rotation in place of the P3-DX. The castor wheel begins not aligned to the rotation. Then the robot starts to move; this movement steers the castor wheel. Although, the castor wheel pushes the robot away from its original position when steering. We can try to parameterize the steering action, adding a new translation and expand the previous model to add the new change, as we see in[1]. Nonetheless, here we try to generalize this problem as a violation of the motion hypothesis since the robot will not move as the equations tell anymore.

#### The assumption violation

#### Velocity Motion Model

#### Odometry Motion Model

### 2.1.3 Robots

**Note:** All robot's name were referencies to The Three Musketeers

#### Athos

Athos is Pioneer 3-AT

#### Aramis

Aramis is Pioneer 3-DX

#### Porthos

Porthos is Pioneer 3-AT

### 2.1.4 Computer

All the three robots have the same onboard computer, to run its system.

#### Computer Specs

- Model IPX1800G2
- Processor Intel® Celeron Dual-Core J1800, 2.41GHz
- RAM 4 GB DDR3 SODIMM
- SSD 120 GB
- 6 USB 2.0
- 1 USB 3.0

[1] Roland Siegwart and Illah R. Nourbakhsh. 2004. Introduction to Autonomous Mobile Robots. Bradford Company, USA.

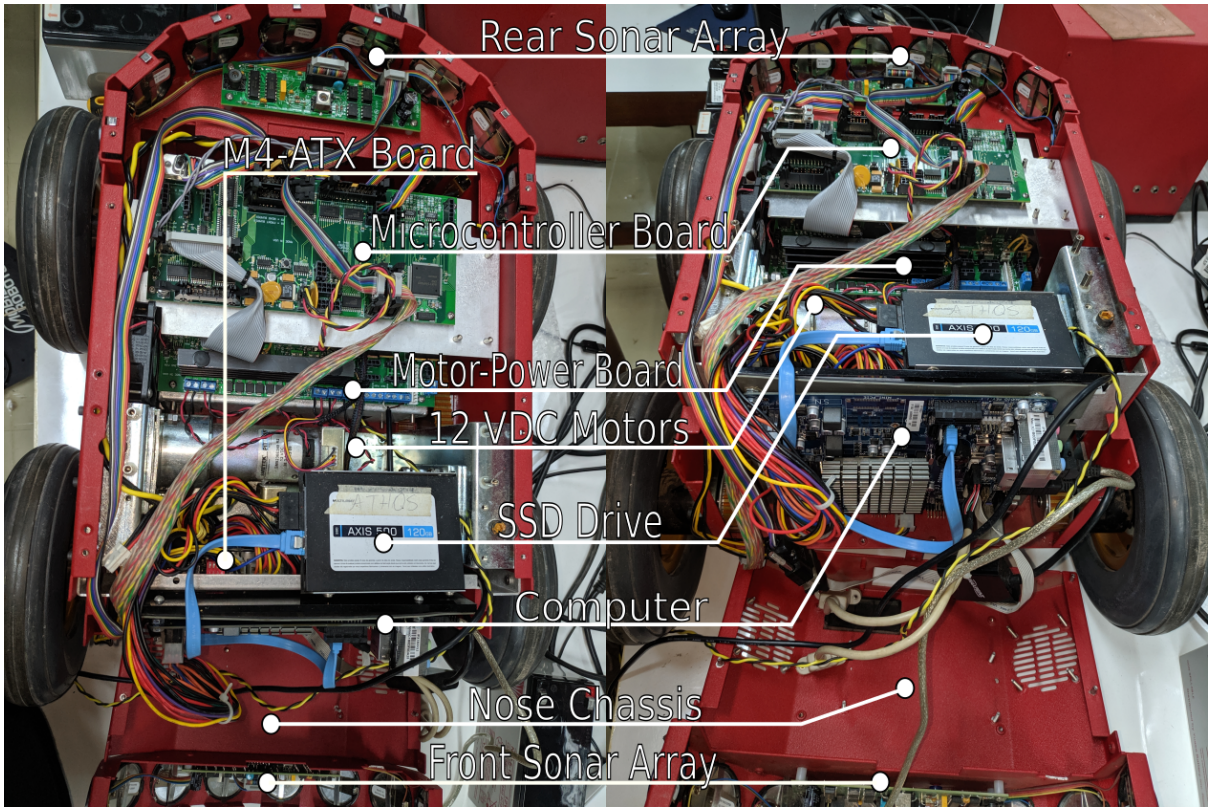



Fig. 9: Athos inside.

- 1 RS232, already used for robot's microcontroller
- WiFi
- VGA and HDMI out

### Power Supply

The Computer is powered by a DC-DC Circuit. Athos and Aramis have an M4-ATX and Porthos have a Pico PSU-160-XT. They are only used to power up the computer using the 24 pin ATX connector. Others accessories, such as Kinect and cameras, use the Motor Power Board.

> **Attention:** The Pioneer family has a board which all the batteries are connected, there is a 20 Amp car fuse in this board. Please be sure that this fuse is connected and working. The robot should not be able to power up without it.

### M4-ATX

The M4-ATX Power Supply is manufactured by Minibox.

- M4 ATX Manual

### PSU-160-XT Specs

- PSU 160 XT Manual

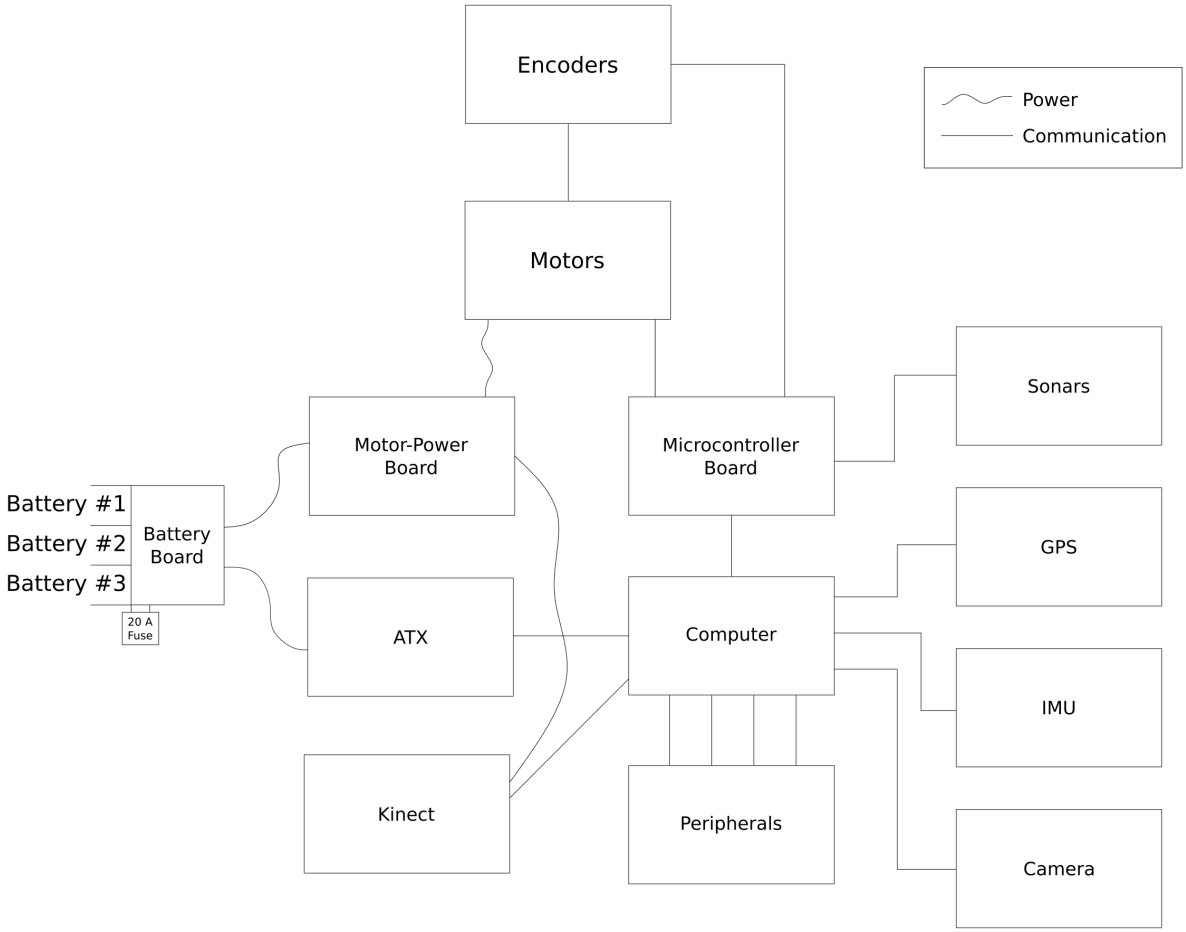



Fig. 10: Hardware Diagram.

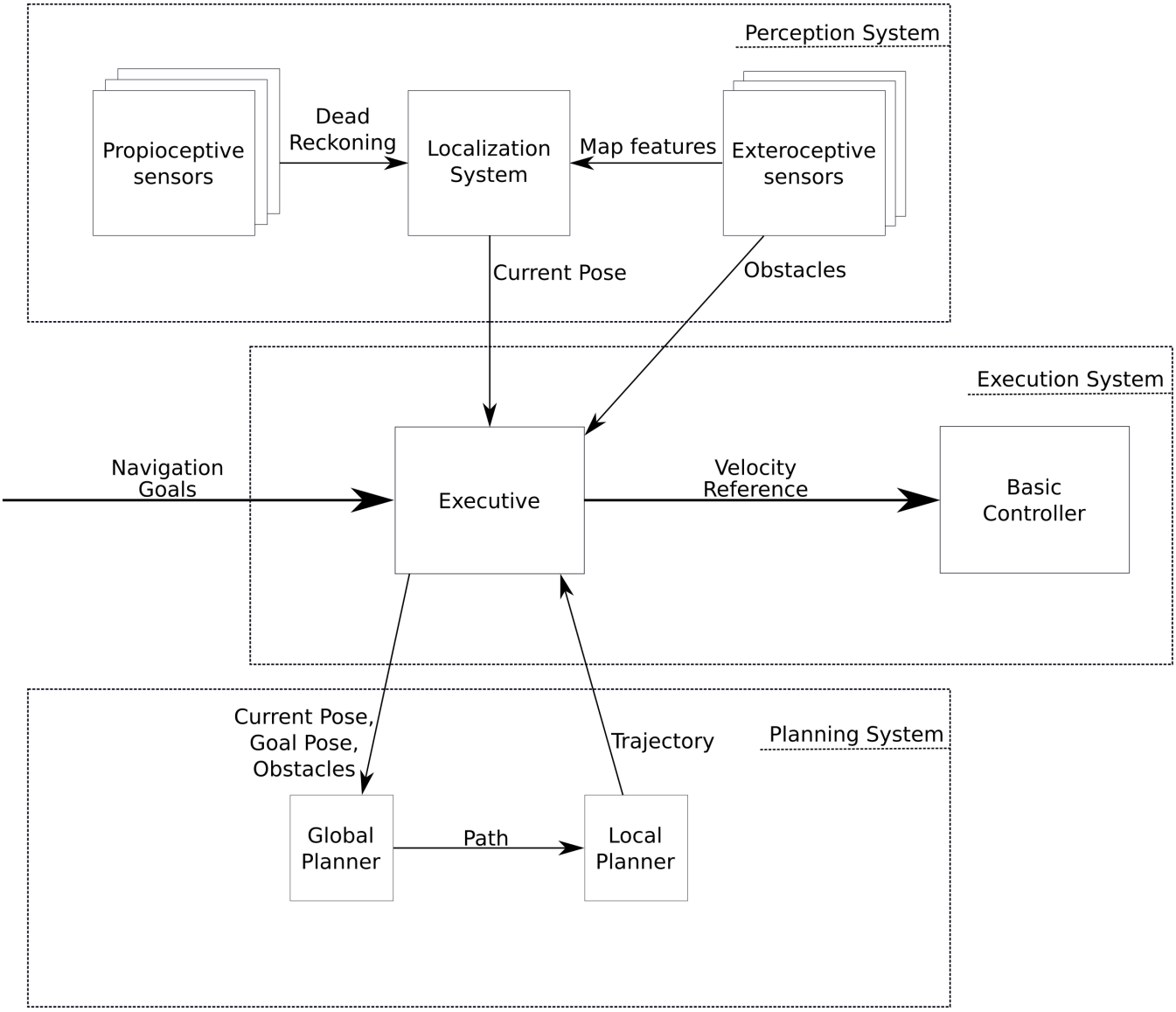



Fig. 11: Navigation System Diagram.

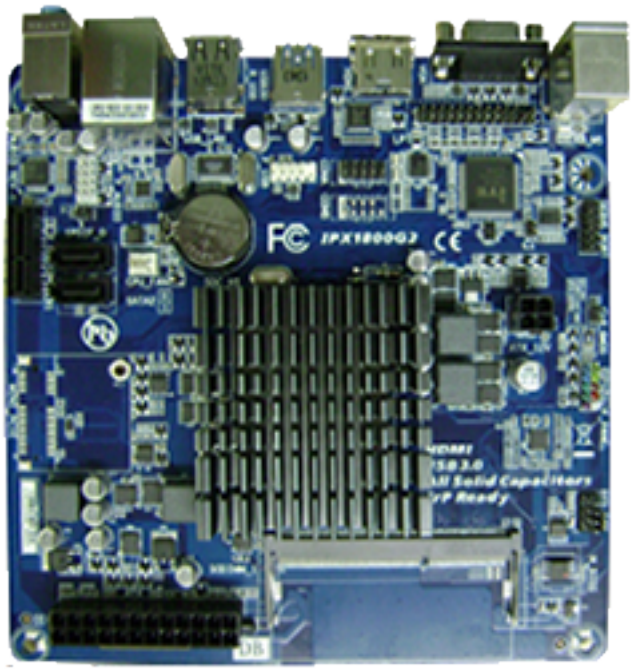
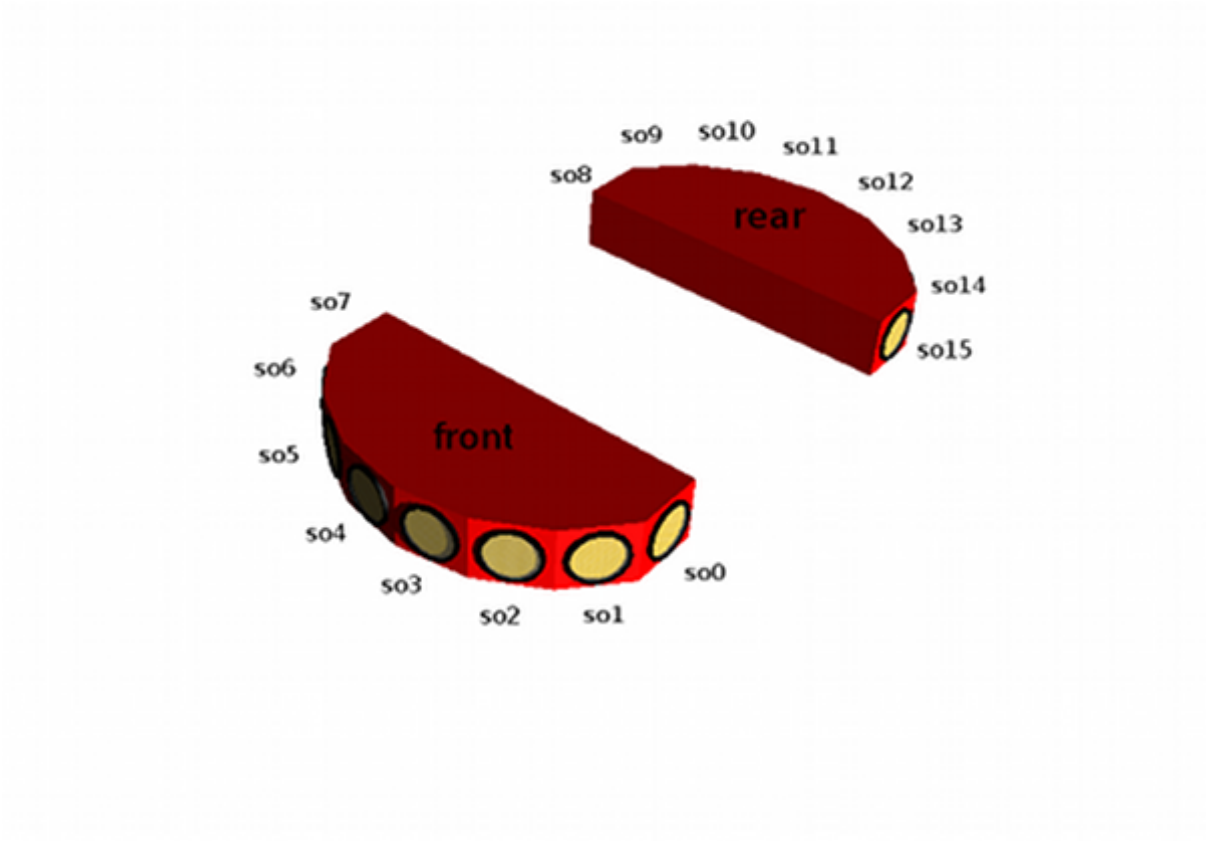


Fig. 12: IPX1800G2 from PCWARE

### 2.1.5 Sonars

Athos and Porthos has two sonar arrays (front and rear), while Aramis have only one (front) sonar array, sonar or ultrasoud is a sensor that uses sound wave for detect obstacles and range information for collision avoidance.



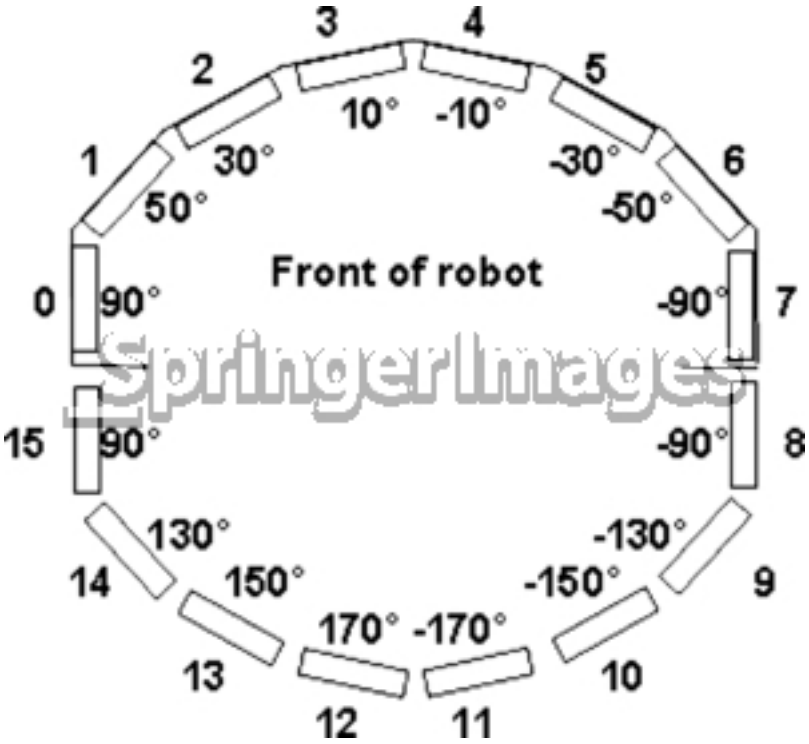
### Sonar Specs

- Range of view: 0.1 m ~ 5 m
- Aquisition rate: 25 Hz

**Warning:** If the sonar doesn't view anything in its cone of view, it will send to the software the max range.

### Geometry

The position of each sonar is showed in the image below.




**Important:**

- All these locations are fixed in the robot.
- There're a URDF file, describing these locations to ROS. Please see the section description of the robot.

### Sensitivity Adjustment

The driver electronics for each array is calibrated at the factory. However, you may adjust the array's sensitivity and range to accommodate differing operating environments. The sonar gain control is on the underside of the sonar driver board, which is attached to the floor of each sonar module.

Sonar sensitivity adjustment controls are accessible directly, although you may need to remove the Gripperto access the front sonar, if you have that accessory attached. For the front sonar, for instance, locate ahole near the front underside of the array through which you can see the cap of the sonar-gain adjustment potentiometer. Using a small flat-blade screwdriver, turn the gain control counterclockwise to make the sonar less sensitive to external noise and false echoes.

Low sonar-gain settings reduce the robot's ability to see small objects. Under some circumstances, that is desirable. For instance, attenuate the sonar if you are operating in a noisy environment or on uneven or highly reflective floor,

a heavy shag carpet, for example. If the sonar are too sensitive, they will "see" the carpet immediately ahead of the robot as an obstacle.

Increase the sensitivity of the sonar by turning the gain-adjustment screw clockwise, making them more likely to see small objects or objects at a greater distance. For instance, increase the gain if youare operating in a relatively quiet and open environment with a smooth floor surface.

By Mobile Robots©

---

**Hint:** See more in the manual.

---

#### Software

We use the p2os to read the sonars readings and the sonar_viz to transform the readings in standart data to better maniplation.

### 2.1.6 Cameras

Athos have a Stereo Camera, Aramis a Kinect and a USB Camera, Porthos only Kinect.

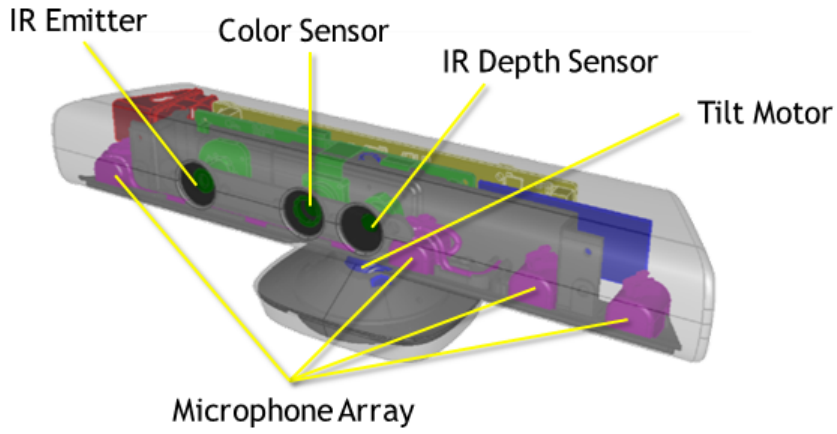
#### Stereo Cameras

#### Specs

- Aquisition rate: 30 fps

#### Kinect


Kinect is a motion sensor device by Microsoft©




### Specs

- Color Camera Resolution 640 x 480
- Depth Camera Resolution 320 x 240
- Max Depth Distance ~4.5 m
- Min Depth Distance ~0.04 m
- Viewing angle 43° vertical by 57° horizontal field of view
- Vertical tilt range +-27°
- Frame rate (depth and color stream) 30 fps
- Audio format 16-kHz, 24-bit mono pulse code modulation (PCM)
- Audio input characteristics A four-microphone array with 24-bit analog-to-digital converter (ADC) and Kinect-resident signal processing including acoustic echo cancellation and noise suppression
- Accelerometer characteristics A 2G/4G/8G accelerometer configured for the 2G range, with a 1° accuracy upper limit.

### USB Camera

The USB Camera in Aramis is a FMVU-03MTM-CS from Point Grey.

### Specs

- Resolution 752 x 480
- Frame Rate 60 fps
- Megapixels 0.3 MP
- Chroma Mono
- Sensor Type CMOS

- Readout Method Global shutter
- Interface USB 2.0

### 2.1.7 Batteries

All the three musketeers contain three sealed lead-acid batteries accessible through a hinged and latched rear door. The batteries charge life typically ranges from two to three hours.

---

**Important:** Batteries have a significant impact on the balance and operation of your robot. Under most conditions, we recommend operating with three batteries. Otherwise, a single battery should be mounted in the center, or two batteries inserted on each side of the battery container.

By Mobile Robots©

---

#### Batteries Specs

- Lead-acid
- Sealed
- 12 VDC
- 4 Ah
- With 3 batteries, 252 Wh
- Hot-swappable

#### Battery Indicators and Low Voltage Conditions

The User Control Panel[1] has a bi-color LED labeled BATTERY that visually indicates current battery voltage. From approximately 12.5 volts and above, the LED glows bright green. The LED turns progressively orange and then red as the voltage drops to approximately 11.5 volts.

Arually, the buzzer will sound a repetitive alarm if the battery voltage drops below 11.5 VDC. If the battery voltage drops below 11 VDC the microcontroller automatically shuts down a client connection and notifies the computer to shut down.

---

**Note:** The batteries voltage is monitored by a own package, this package if necessary notifies the user and shut down automatically the operating system. See more in robot monitor.

---

**Attention:** The Pioneer family has a board which all the batteries are connected, there is a 20 Amp car fuse in this board. Please be sure that this fuse is connected and working. The robot should not be able to power up without it.

#### Recharging

#### Standart Charger

This accessory recharge the batteries in the fast-charge mode (4A maximum current). The fast-charge mode is showed with an orange LED and trickle mode by a green LED, which the batteries are given only enough current to remain at full charge.

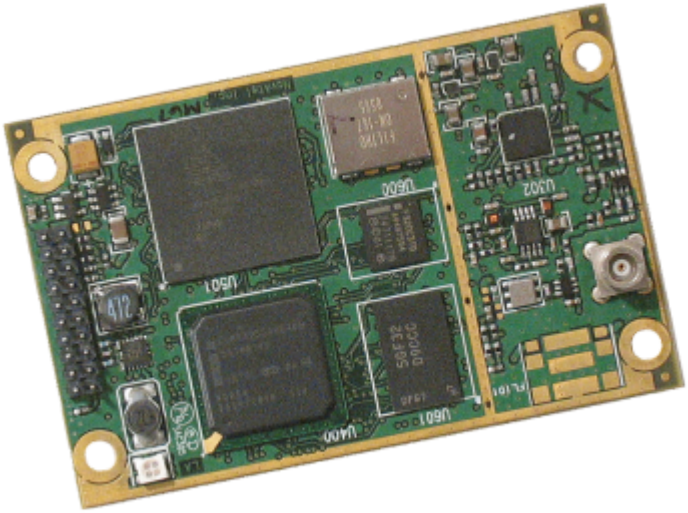
[1] See in the manual.

> **Warning:** In the fast-charge mode, care must be taken to charge at least two batteries at once. A single battery may overcharge and thereby damage both itself and the robot.

#### Power Cube

This accessory allows simultaneous recharge of three batteries outside the robot.

### 2.1.8 GPS Receiver



The robots have each one a Novatel GPS receiver model OEMV-1. We have a custom software to publish the GPS readings in ROS and a custom hardware to power the receiver.

#### Specs

- GPS tracking
- L1, L-Band and SBAS signal tracking
- Low power consumption for longer operating time
- Single frequency

### Antenna


"The ANT-35C1GA-TW-N is an active GPS antenna, 88.9 mm (3.5") in diameter, and designed to operate at the GPS L1 frequency of 1575.42 MHz. Its mechanical configuration is a spherical radius molded radome which provides enhanced protection against rain and ice."[1]

### Power Board

A circuit board was built to integrate GPS and IMU readings, to communicate with onboard pc and to power the GPS receiver and the IMU.

## 2.1.9 IMU


[1] See more in the antenna user guide.

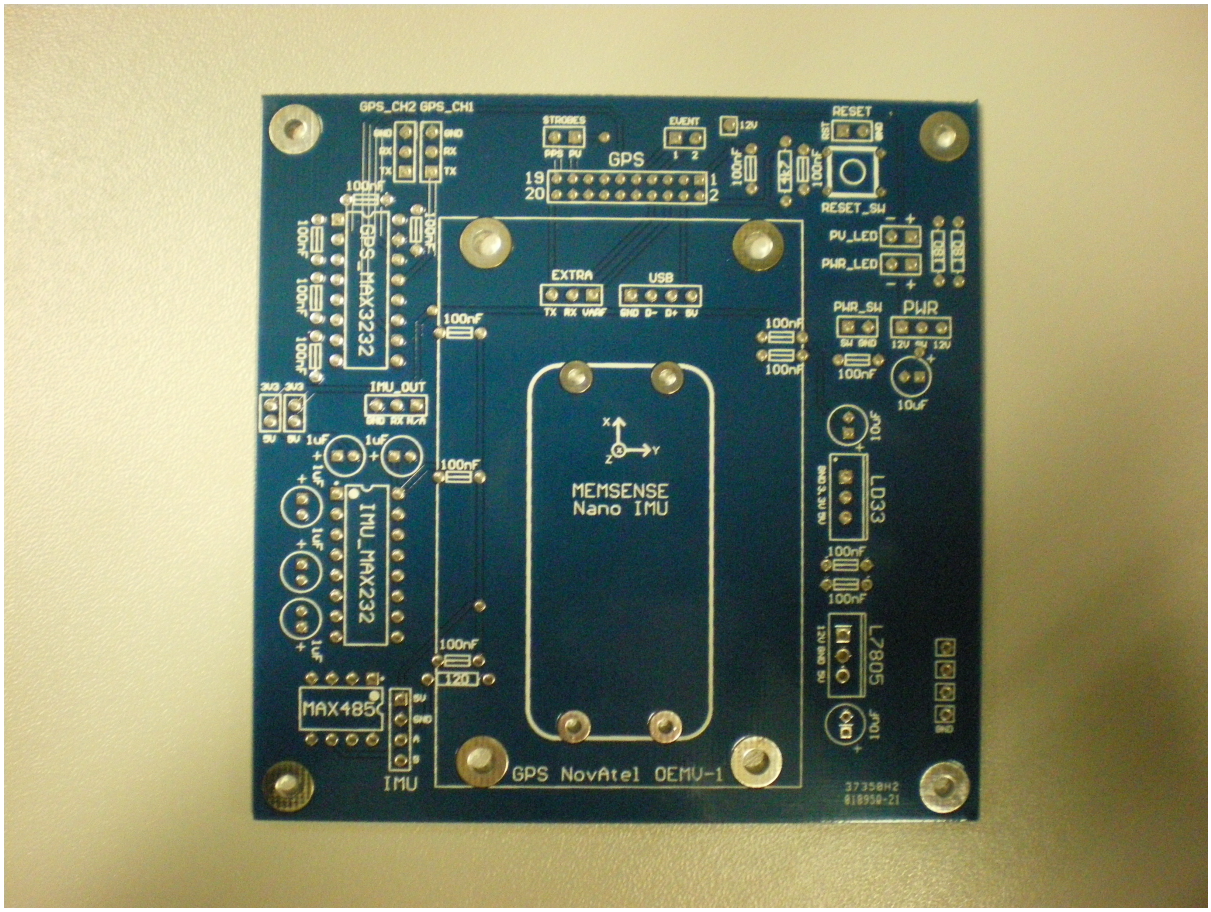



Fig. 13: Power Board

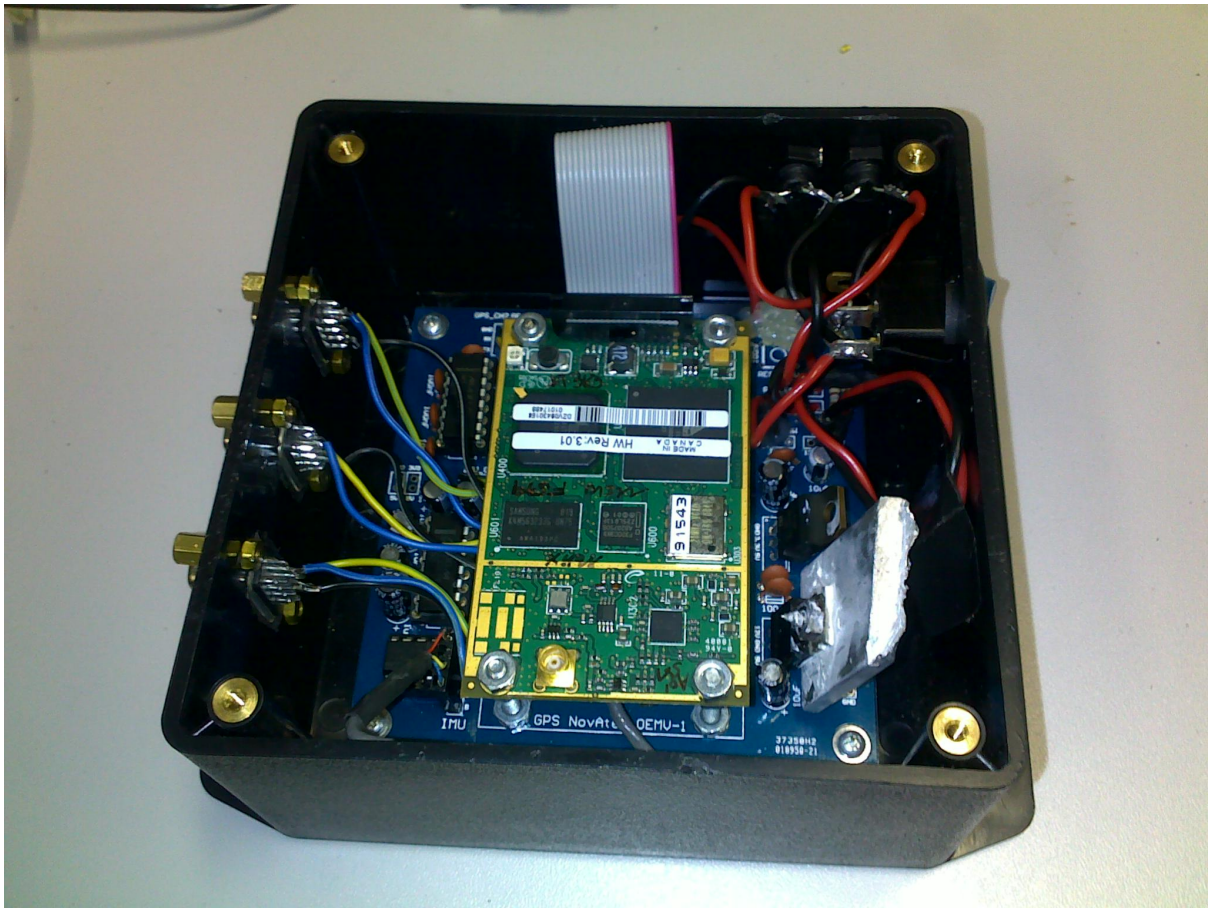



Fig. 14: Power Board with GPS Receiver and IMU (under GPS) in its case.

The MEMSense NanoIMU is a 9DOF IMU with thermometer used in the robots together with GPS receiver. We have a custom software and a hardware to power and send the readings to the computer.

### Specs

#### Gyrometer Specs

- Dynamic Range +-300 º/s
- Offset +-1.5 º/s
- Cross-axis sensitivity +-1 %
- Nonlinearity +-0.1 % of FS (Best fit straight line)
- Noise 0.56 (max 0.95) º/s, sigma
- Digital Sensitivity 1.3733E-2
- Bandwidth 50 Hz

#### Accelerometer Specs

- Dynamic Range +-2 g
- Offset +-30 mg
- Nonlinearity +-0.4 (max +-1.0) % of FS
- Noise 0.6 (max 0.8) mg, sigma
- Digital Sensitivity 9.1553E-5
- Bandwidth 50 Hz

#### Magnetometer Specs

- Dynamic Range +-1.9 gauss
- Drift 2700 ppm/ºC
- Nonlinearity +-0.5 % of FS (Best fit straight line)
- Noise 0.00056 (max 0.0015) gauss, sigma
- Digital Sensitivity 8.6975E-5
- Bandwidth 50 Hz

#### Thermometer Specs

- Digital Sensitivity 1.8165E-2

#### Power Board

The circuit board is the same for GPS receiver, more informations here.

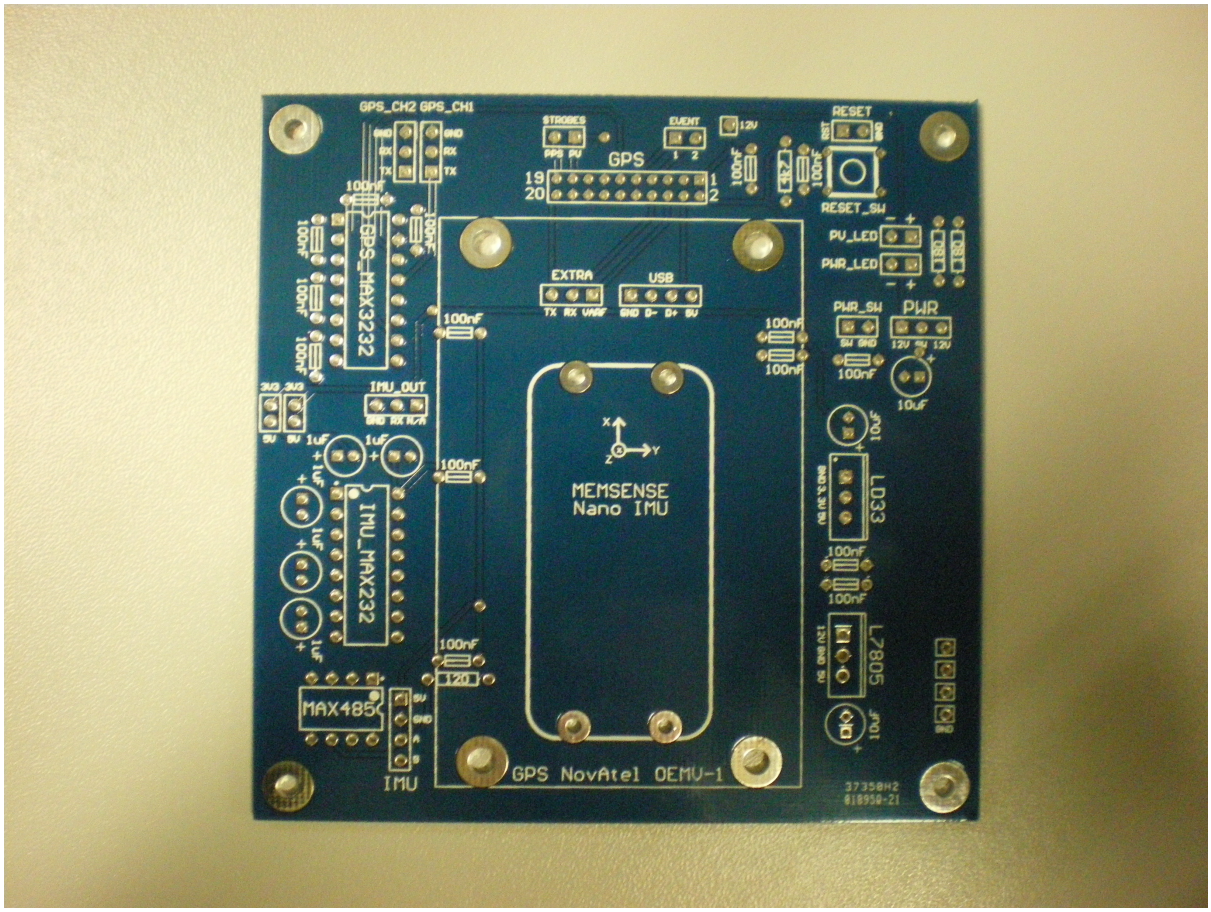



Fig. 15: Power Board

### 2.1.10 ROS

### 2.1.11 Gazebo Simulator

### 2.1.12 P2OS Package

### 2.1.13 Move Base Package

### 2.1.14 System's Architecture

### 2.1.15 Mobile Robotics Lab 1

#### Objetivos

Aprender a criar um pacote no ROS.

#### Introdução

Todo o software produzido no ROS é organizado em pacotes. Um pacote pode conter nós do ROS, bibliotecas independentes, datasets, arquivos de configuração ou qualquer material que possa constituir um módulo. Para que um pacote seja visto pelo ROS ele deve ter no mínimo os seguintes requisitos:

- Deve conter um arquivo chamado package.xml com código catkin;
- Deve conter um arquivo CMakeLists.txt;
- Cada pacote deve ter seu próprio diretório;

O arquivo package.xml reúne todas as informações de autor, nome do pacote, versão do código e dependências. Já o CMakeLists.txt reúne as regras de compilação do pacote.

#### Procedimentos

- **No terminal, configure o ros com o comando:**
  - $ bash.sh
- **No novo terminal aberto, entre na pasta catkin_ws. Esse será o ambiente de trabalho a ser usado neste curso:**
  - $ cd catkin_ws
- **Entre na pasta src. Para o ROS essa é pasta onde fica o código fonte dos pacotes:**
  - $ cd src
- **Use o script catkin_create_pkg para criar um novo pacote. Esse pacote será chamado de 'beginner_tutorials' e terá co**
  - $ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
- **Este comando irá criar uma pasta chamada de beginner_tutorials que conterá um package.xml e um CMakeLists.txt**
  - Abra o CMakeLists.txt em um editor de texto e procure onde as dependências são chamadas.
  - Abra o package.xml em um editor de texto e procure onde as informações de autor, versão e dependências são informadas.
- **Para compilar o workspace volte para catkin_ws:**
  - $ cd ~/catkin_ws
- **Atualize as variáveis de ambiente do ROS:**

  - $ cd catkin_ws
  - $ . devel/ros_custom.sh
- **Compile o workspace:**
  - $ catkin_make

#### Exemplo

#### Referências

Esse tutorial foi baseado no tutorial Creating a ROS Package do ROS:

http://wiki.ros.org/ROS/Tutorials/CreatingPackage

### 2.1.16 Mobile Robotics Lab 2

#### Objetivos

Aprender sobre mensagens e tópicos do ROS.

#### Procedimentos

**Abra o terminal e inicie o nó central do ROS:**

- $ roscore

**Em outro terminal inicie o turtlesim:**

- $ rosrun turtlesim turtlesim_node

**O ROS usa um sistema de comunicação distribuída em que os executáveis, chamados de nós, usam tópicos para a troca de n**

- $ rostopic list

**Este comando irá listar todos os tópicos ativos. Para ver o nós ativos use:**

- $ rosnode list

**Para ver as informações de um nó, use rosnode info /nome_do_no:**

- $ rosnode info /turtlesim

**Este comando irá mostrar os tópicos que são publicados e ouvidos pelo nó turtlesim. Para mover a tartaruga do turtlesim d**

- $ rostopic info /turtle1/cmd_vel

**Este comando mostrará o tipo de mensagem, os 'publishers' e os 'subscribers' deste tópico. O tipo 'geometry_msgs/Twist' r**

- $ rosmsg show geometry_msgs/Twist
- Obs.: Todas as unidades no ROS estão no Sistema Internacional de Unidades (SI), portanto a velocidade se dá em m/s;

**Para publicar uma mensagem no tópico /turtle1/cmd_vel, use o comando:**

- $ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist "{linear:[0.0, 0.0, 0.0], angular:[0.0, 0.0, 0.0]}"
- Obs.: a tecla TAB pode ser usada para completar os comandos;

**O primeiro argumento da mensagem representa a velocidade linear e suas componentes x, y e z. Já o segundo argumento re**

- $ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist "{linear:[1.0, 0.0, 0.0], angular:[0.0, 0.0, 0.0]}"

**Para enviar uma mensagem em loop, deve-se adicionar o argumento '-r <frequência_do_loop>'. Para enviar o mesmo coma**

- $ rostopic pub /turtle1/cmd_vel geometry_msgs/Twist "{linear:[0.0, 0.0, 0.0], angular:[0.0, 0.0, 1.0]}" -r 10

**Para ver o que está sendo publicado em um nó pode se usar 'rostopic echo':**

- $ rostopic echo /turtle1/cmd_vel
- $ rostopic echo /turtle1/pose

Usando os comando acima faça a tartaruga desenhar uma linha. Usando os comando acima faça a tartaruga desenhar um quadrado. Usando os comando acima faça a tartaruga desenhar um círculo.

#### Referências

Para mais informações sobre tópicos, mensagens e nós visite:

- http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes
- http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics
- Referência sobre geometry_msgs: http://wiki.ros.org/geometry_msgs
- Referência sobre o turtlesim: http://wiki.ros.org/turtlesim

### 2.1.17 Mobile Robotics Lab 3

#### Objetivos

O objetivo desse exercício é programar a movimentação de um robô diferencial usando os comando no terminal aprendidos no laboratório passado. Este exercício usará uma versão simulada do robô Pioneer.

#### Procedimentos

**Na pasta src do seu workspace (catkin_ws/src/) baixe o pacote fcr2018**

- $ git clone https://github.com/Gastd/fcr2018

**Atualize as variáveis de ambiente do ROS**

- $ cd ~/catkin_ws
- $ source devel/ros_custom.sh

**Abra a simulação do pioneer**

- $ roslaunch fcr2017 pioneer3at.gazebo.launch

Use o 'rostopic list' para descobrir quais mensagens o pioneer publica e quais ele recebe. Use o 'rostopic echo' para olhá-las.

Descubra qual tópico o robô usa para publicar as informações do laser

Descubra qual tópico o robô usa para publicar as informações da odometria

Descubra qual tópico é usado para enviar velocidades para o robô

Use o 'rostopic pub' para publicar velocidades. Escolha números que façam o robô andar em uma linha reta

Faça o robô andar em um círculo

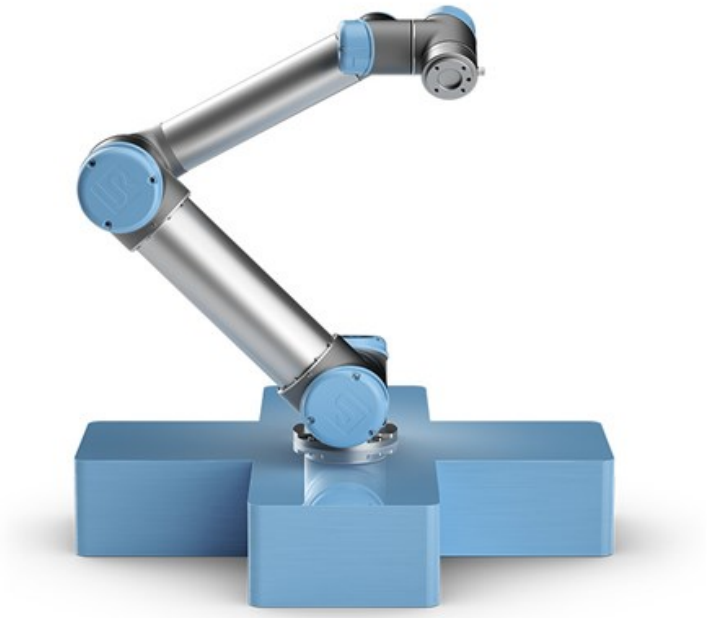
Faça o robô andar em um retângulo

### Referências

CHAPTER 3

# Manipulators

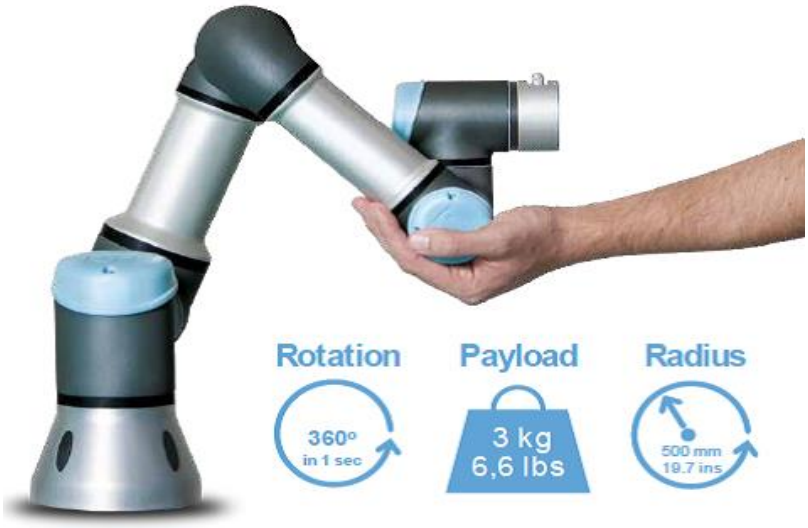
## 3.1 UR3



### 3.1.1 System Description

#### Overview

The UR3 is a table-top collaborative robot. With its 3 kg payload it is very capable and its small footprint makes it suitable for limited workspace situations. With its infinite turn on the end joint, several activities can be perfomed




with grippers attached at robot tool connector.

Some of its applications:

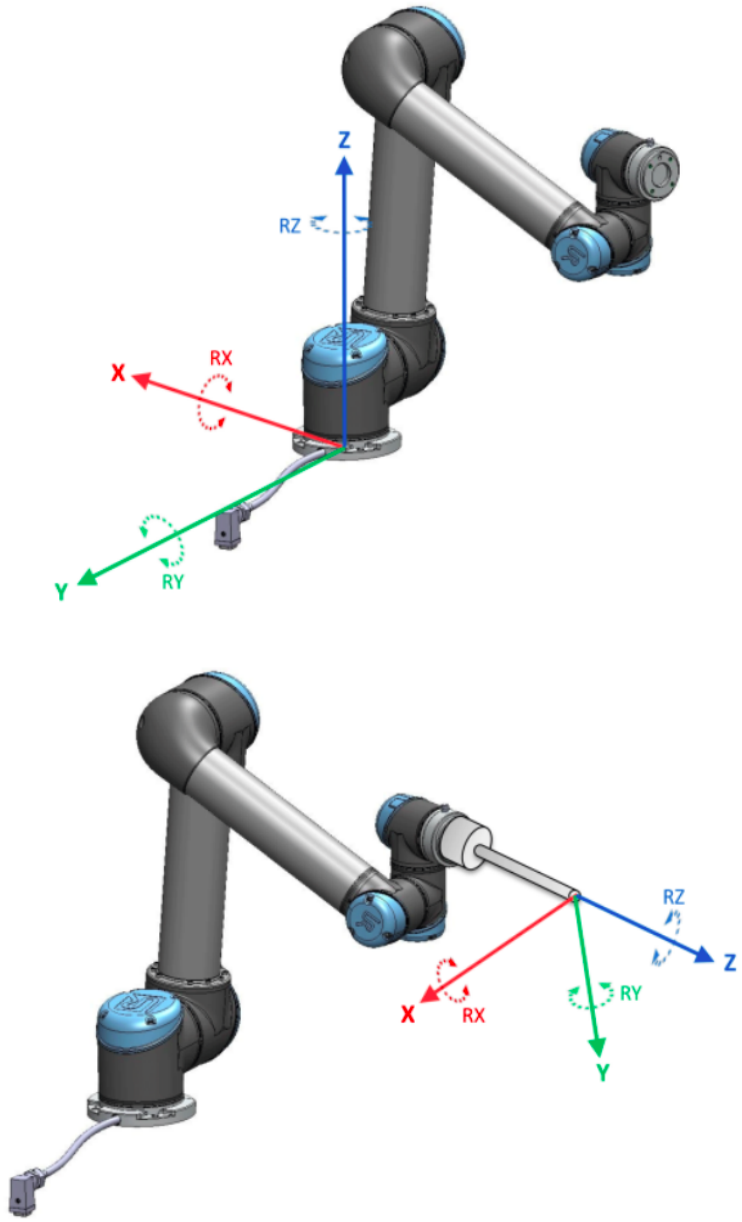
- Laboratory work
- Assembly tasks
- Polishing
- Soldering
- Gluing
- Screwing
- Painting
- Pick and place
- Operating hand tools
- Fume hood tasks

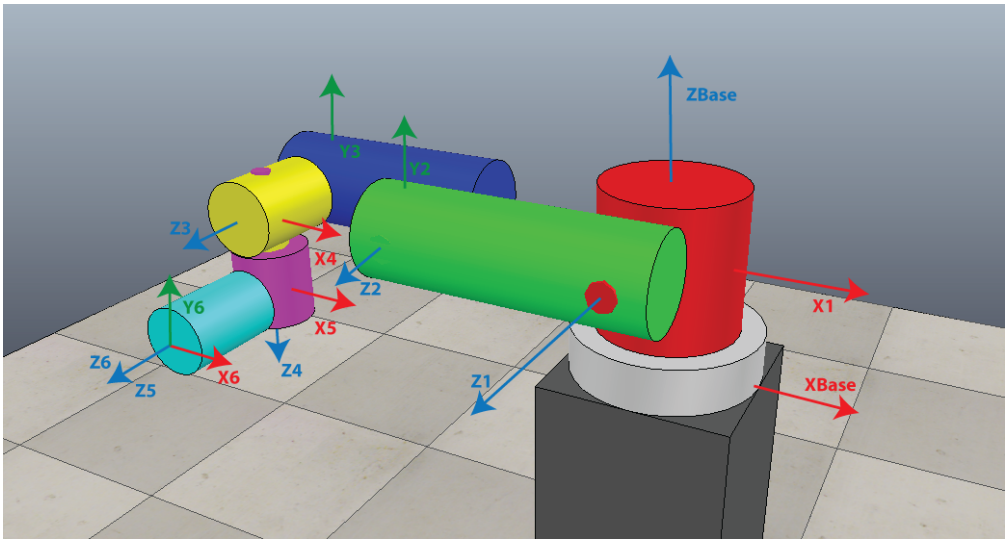
### Specification

Table 1: UR3 Specification

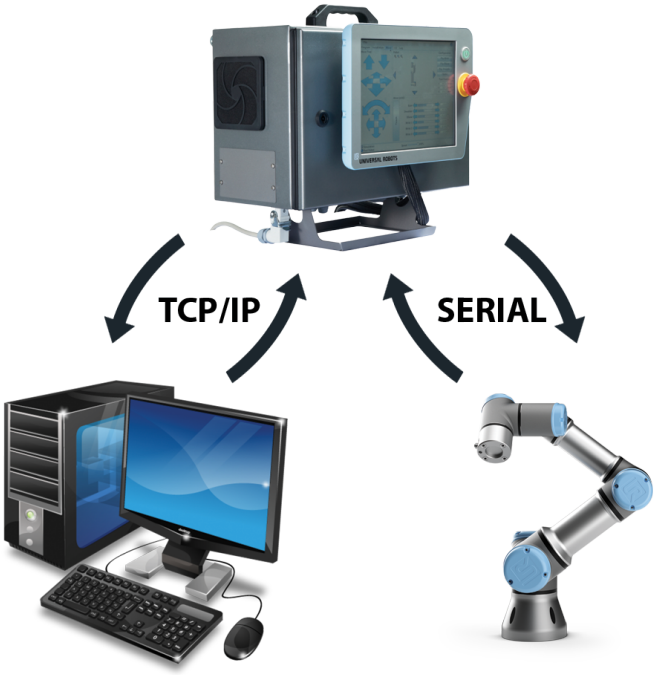
| | |
|---|---|
| Weight | 11.2 kg |
| Payload | 3 kg |
| Reach | 500 mm |
| Footprint | Ø 128 mm |
| Degrees of freedom | 6 rotating joints |
| Joint ranges | +/- 360°, infinite rotation on end joint |
| Speed wrist joints | 360 degrees/sec |
| Other joints | 180 degrees/sec |
| Noise | Comparatively noiseless |
| IP classification | IP64 |

### Coordinate System




ZBase
Y3
Y2
Z3
X4
Z2
Y6
X5
X1
Z6
Z5
X6
Z4
Z1
XBase

### Communication




There are two main communication gateway in this system.

Foremost, there is a TCP/IP communication port between a LinuxPC and the UR3 Control Box. Basically, it is possible to send ROS commands directly from from Linux Terminal or specialized simulation softwares.

Next, there is a serial communication port between the Control Box and the UR3 Robot. It is responsible for send all the position and velocity commands to the robot.

Some of its specification:

- TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX
- Ethernet socket & Modbus TCP

#### Control Box


The Controller Box contains both digital and analog input and output sockets which can be used for interfacing other components or system components itself. The teach pendant can be used to program the robot as per the requirement of user and can be based on inputs and outputs.

Using this Controller Box, the robot can be set up quickly without programming experience using patent technology and can be operated with an 3D intuitive visualization. It requires a simple movement of the robotic arm by giving waypoints or from the controls given on the touchpad.

#### Linux PC

Alternatively to Control Box, it is possible to control the robot system from a Linux PC. Using ROS and ensuring that your hardware, computer and robot, are properly configured to talk to each other, it is feasible to perform any movement or path directly from your PC, as good, or better, as from Control Box.

### 3.1.2 Kinematics

### 3.1.3 Dynamics

### 3.1.4 Robots

### 3.1.5 ROS

### 3.1.6 Control System

### 3.1.7 Lab 1: First Contact

#### objective

The new user contacts the ur3 platform and performs a first experiment.

#### Introduction

This lab teaches how to run a code to make a simple move on the ur3 robot using ROS.

### Procedures to setup the ur3 robot

First, let's turn on the UR3 robot by pressing the POWER button shown in figure 1.

Fig. 1: Figure 1: Power button

After pressing the power button, the screen like the figure below will appear.


Fig. 2: Figure 2: Loading screen

After loading the robot system, the screen as the figure below will appear.

Now, turn on the control box by clicking **Go to the initialization screen**, which is in the center of figure 3.

now, we have a screen like figure 4.

The turn on the robot, press the **ON** button. The screen that will appear will be like as in figure 5.

The robot is now on. To unlock the joints, press the **START** button. The robot will make an unlocking sound, which is expected, and will enter the normal operating mode that can be seen in figure 6.

Depois disso, aperte **OK** que fica localizado no canto inferior direito da tela (figura 6). Now, a new screen will appear (figure 7) with some options for robot functionality. Let's click on **Program Robot**.

Depois de apertar **Program Robot** aparecerá uma tela semelhante a figura 8. Aperte **Move**.

Ready. We have completed our robot setup and the final screen is similar to Figure 9.

### Procedures to setup the Linux PC

Now let's turn on the computer that controls the robot. To do this, connect the LARA computer designated for the UR3 robot. The computer name is **ur3** and password **ur3**.

One more thing, the communication will be done using the TCP/IP protocol using cable, so check if the network cable of the Lara-robots router is connected to the ur3 computer.

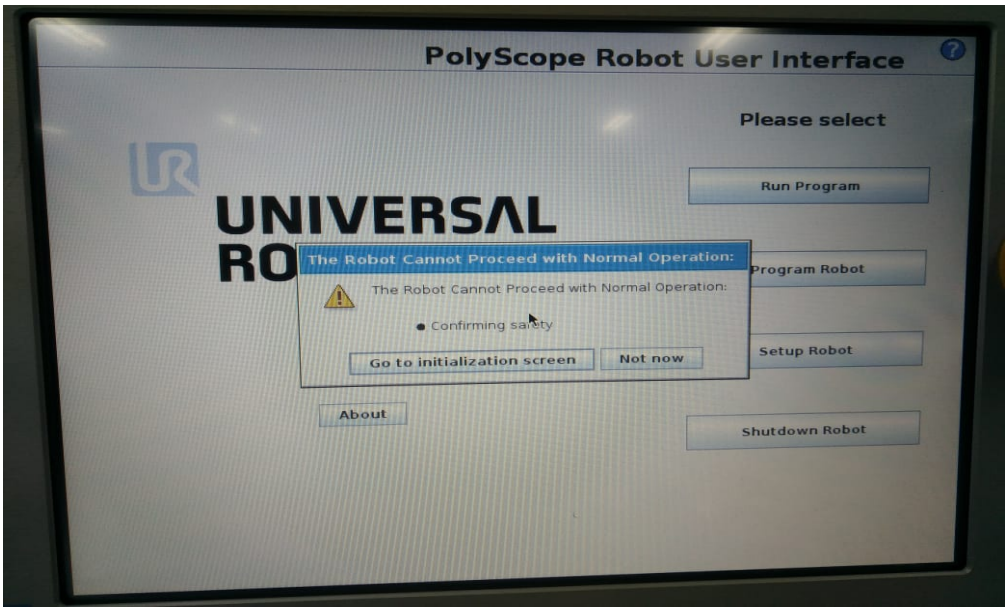



Fig. 3: Figure 3: Apresentation screen

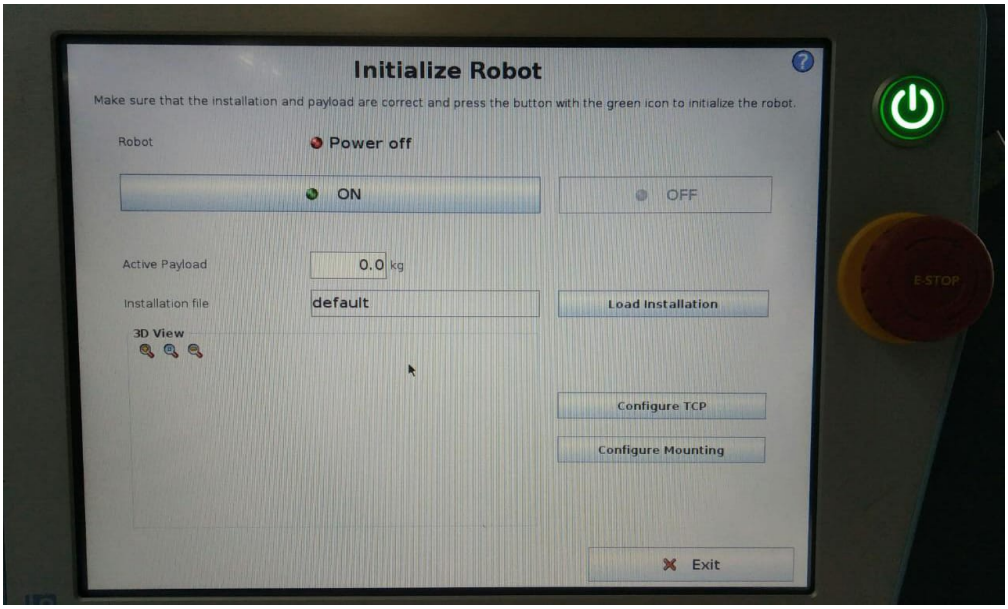



Fig. 4: Figure 4: screen to turn on the robotic arm.

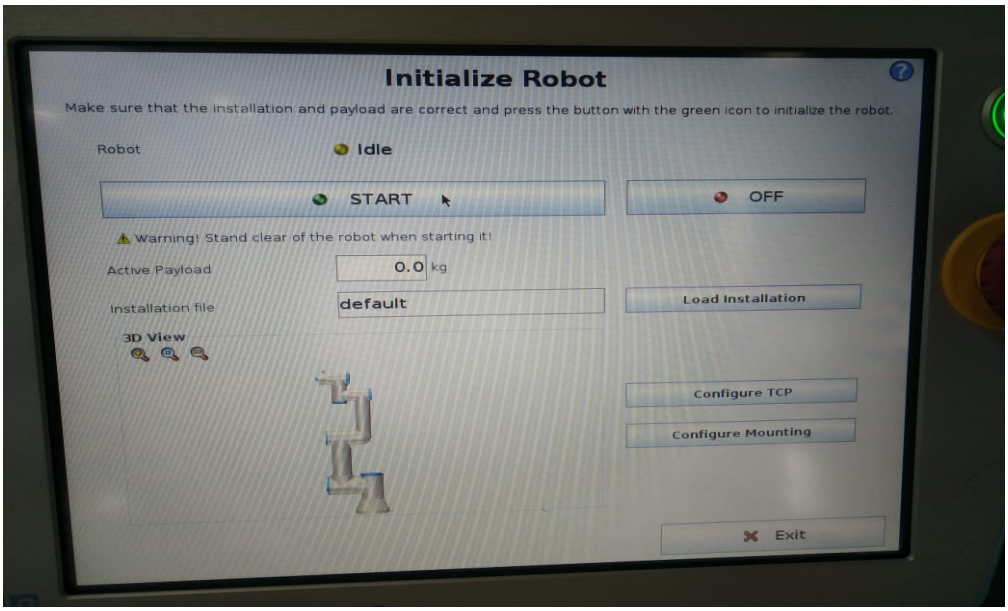



Fig. 5: Figure 5: start screen.

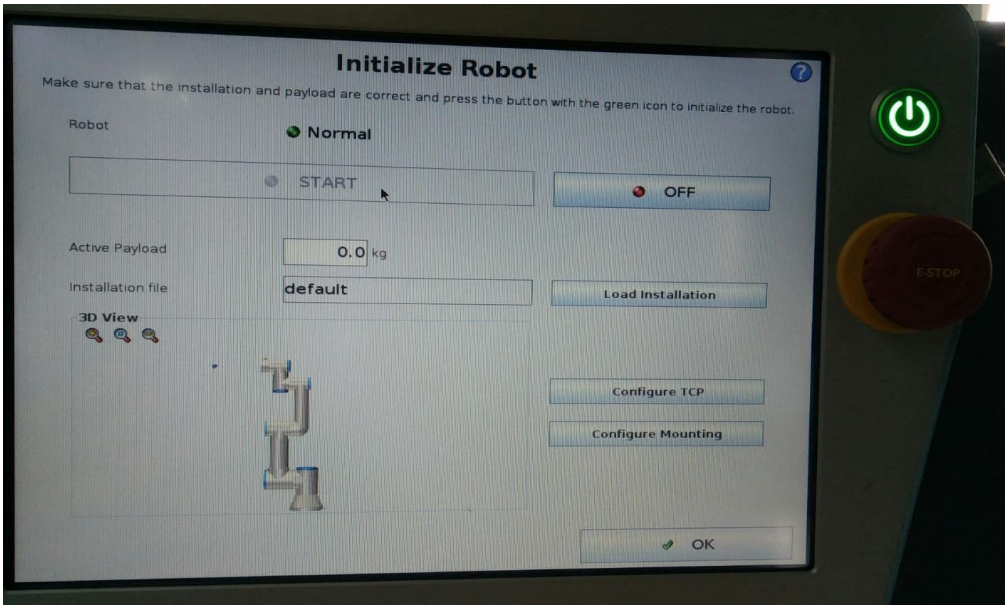



Fig. 6: Figure 6.

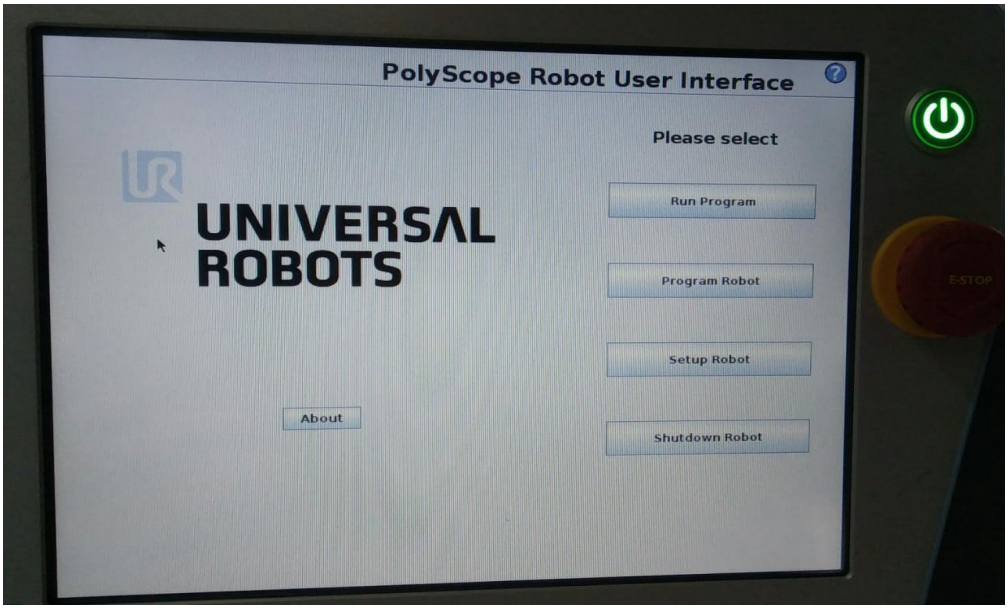



Fig. 7: Figure 7.

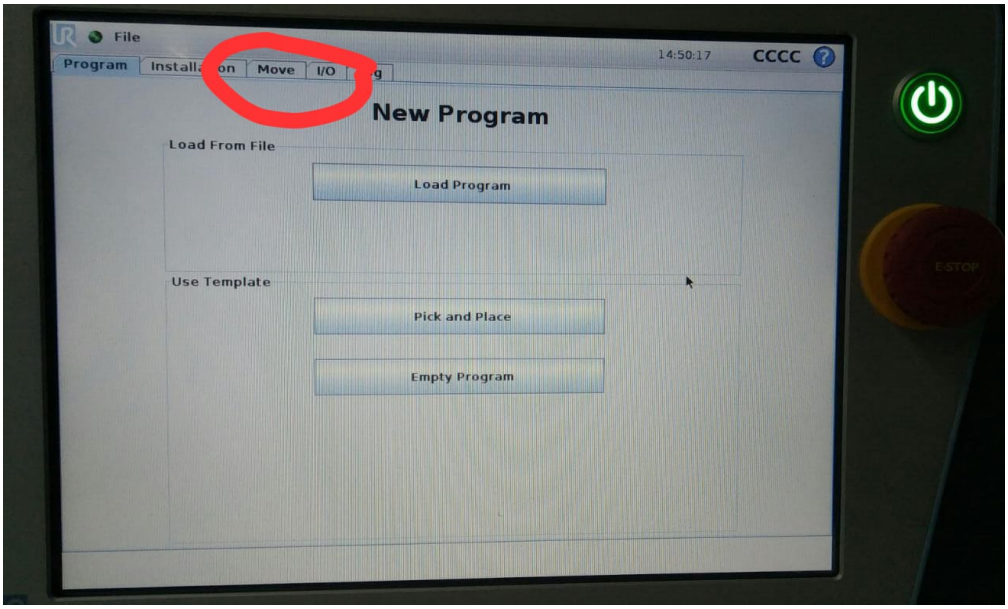



Fig. 8: Figure 8.

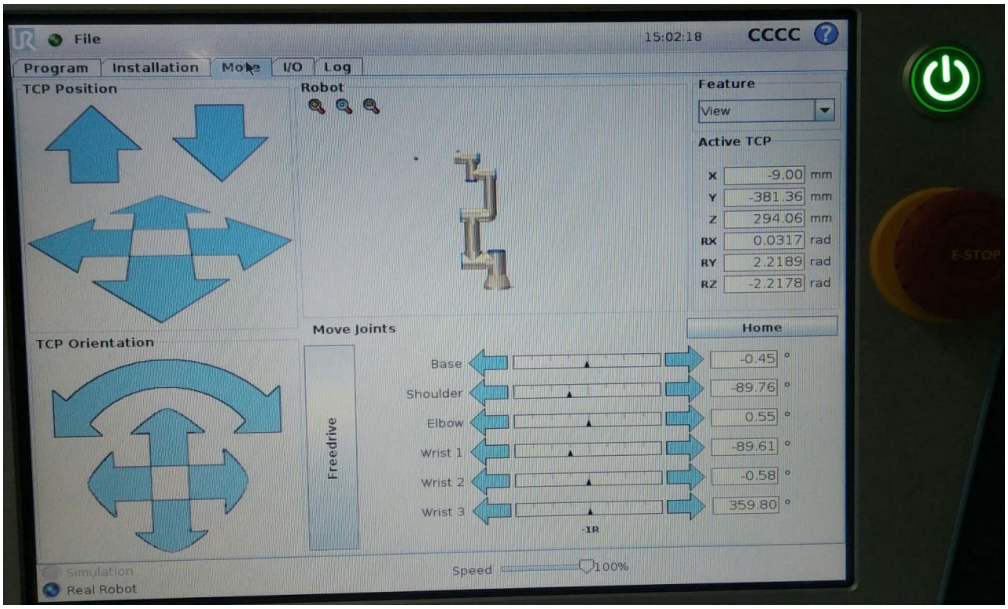



Fig. 9: Figure 9.

After making the instructions described in the paragraph above, we will download the files that contain the codes for our experiment. To do this, open the application called **Terminator** on the ubuntu 18.04 taskbar, which is exactly like figure 10.

Fig. 10: Figure 10: Terminator

After opening **Terminator**, we will use the command **git clone** to download the file. Then, type in the Terminator the command (If the file already exists, skip this step):

- $ git clone https://github.com/lara-unb/UR3_interface.git

The next step is to compile the code to "run" on the ur3 computer. To compile the code we will enter the directory, using Terminator, where the executable folders are.

With the command shown below you can enter the interface code workspace.

- $ cd UR3_interface/catkin_ur3

To compile the code use the command:

- $ catkin_make

You will receive a message like the one in figure 11 below.

The computer setup is ready. In the next step we will learn how to start the ROS (Robot Operating System).

### ROS initialization

The demo_1 interface and executables are ready (done in the previous step), now let's start ROS. For that, we will open more sections in Terminator.

Open 4 (four) sections as shown in figure 12 below.

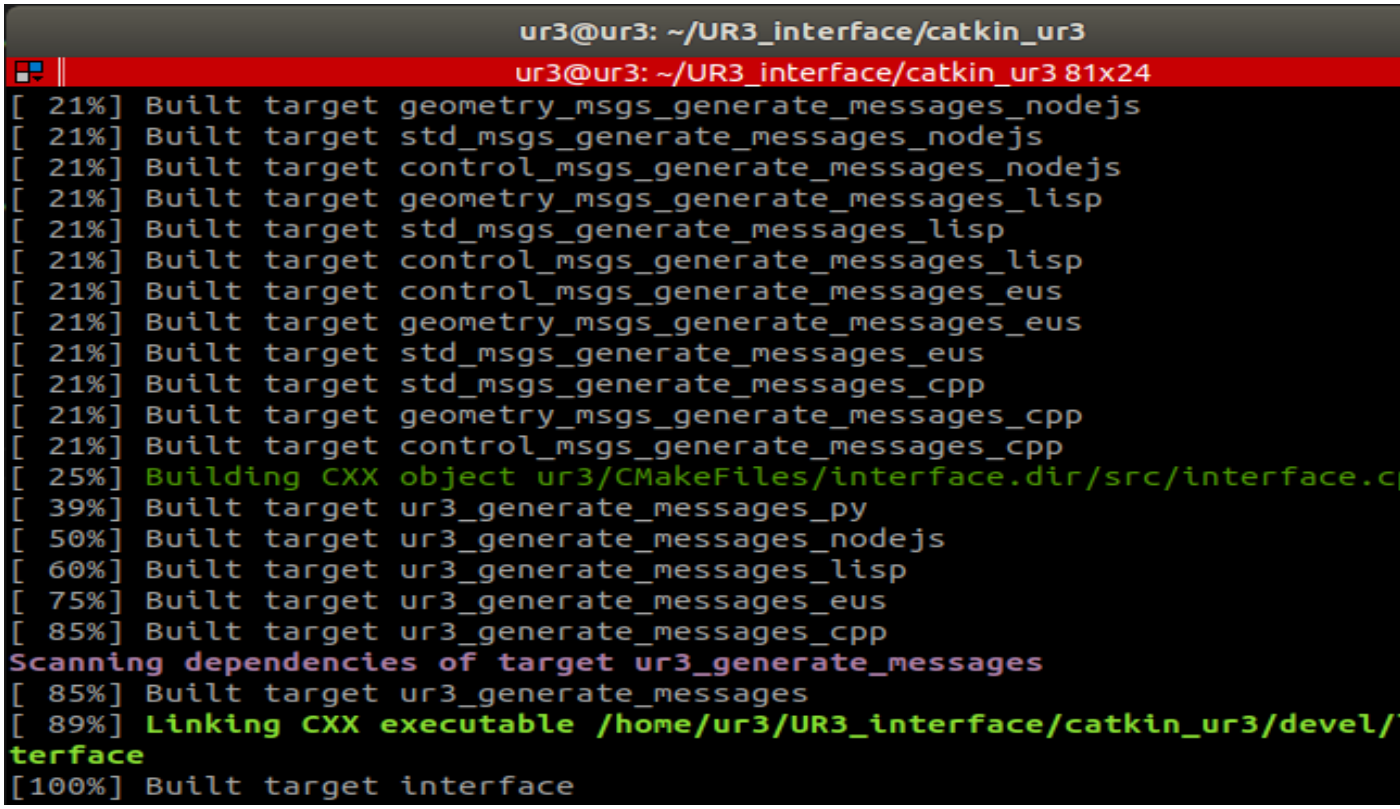



Fig. 11: Figure 11.

To split the Terminator you can use Ctrl + shift + O to split horizontally or Ctrl + shift + E to split vertically. Remembering that the new sections have to be in the same workspace.

Para iniciar o ROS use o comando:

- $ roscore

When ROS is ready, you will receive a message in one of the sections as shown in figure 13 shown below.

NOTE: Don't touch the section that ROS is running.

### Running Demo_1

To run **demo_1**, we will first run the **interface** (Use another section of Terminator). For this, we will configure setup of the ROS package with the following command:

- $ source devel/setup.bash

Do this for all sections of the Terminator except for the one running ROS.

Now let's "run" the interface application. To do this, type the command shown below and press ENTER on one of the Terminator sections.

- $ rosrun ur3 interface

If everything is working as expected, the message we will get in return for this command is shown below in figure 14.

When you have confirmation that the robot is ready (UR3 is ready) we can run demo_1.

Now, in another section of Terminator, we will "run" the demo-1 application. To do this, type the command shown below and press ENTER.

- $ rosrun ur3 demo_1

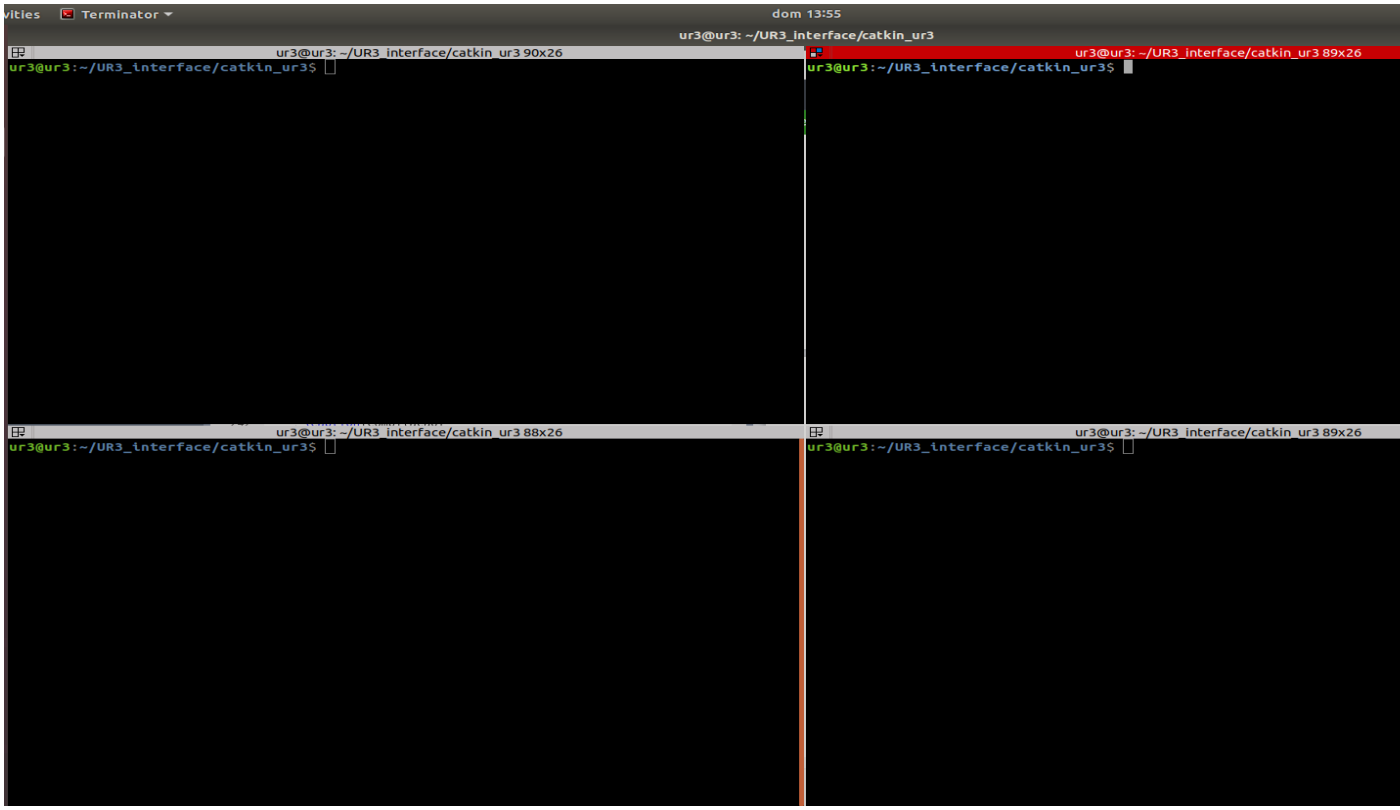



Fig. 12: Figure 12.

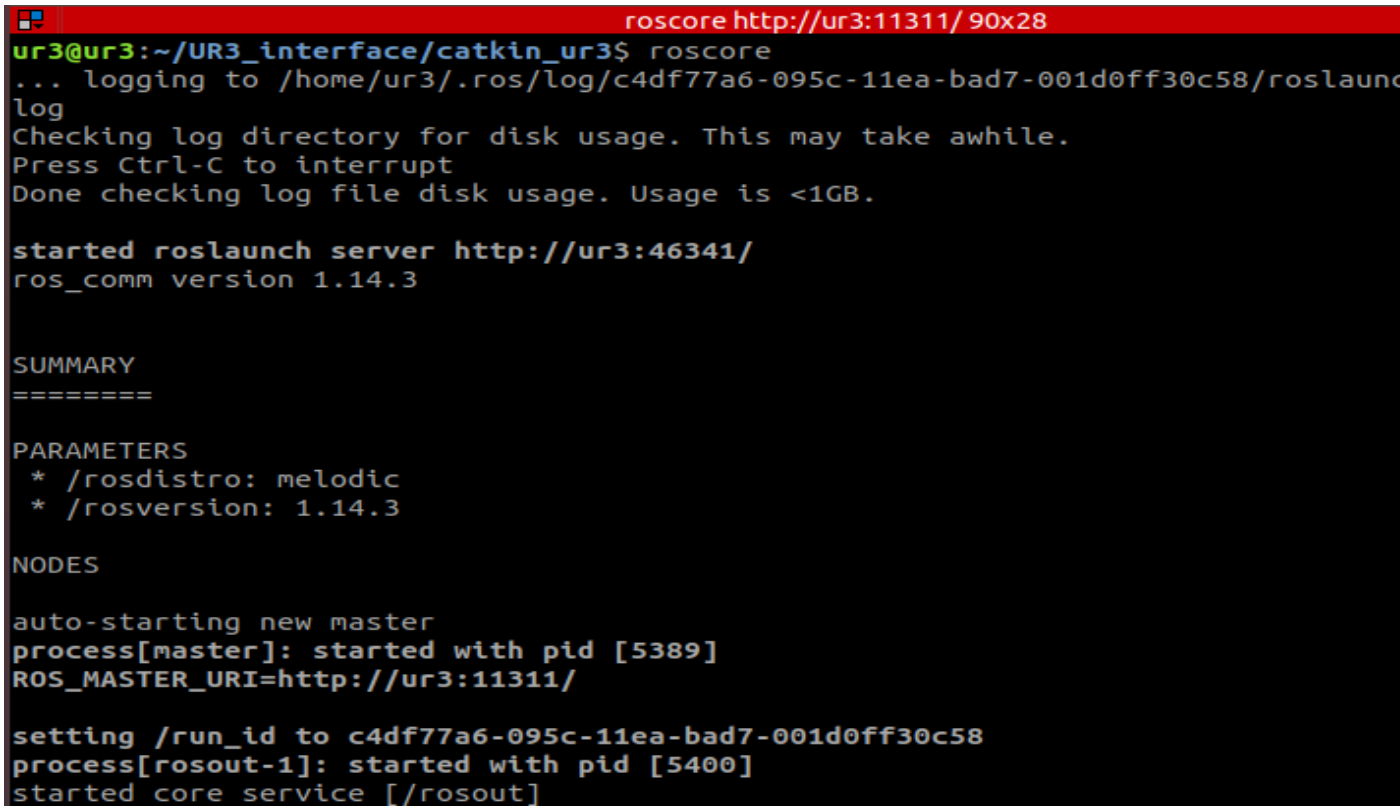



Fig. 13: Figure 13.

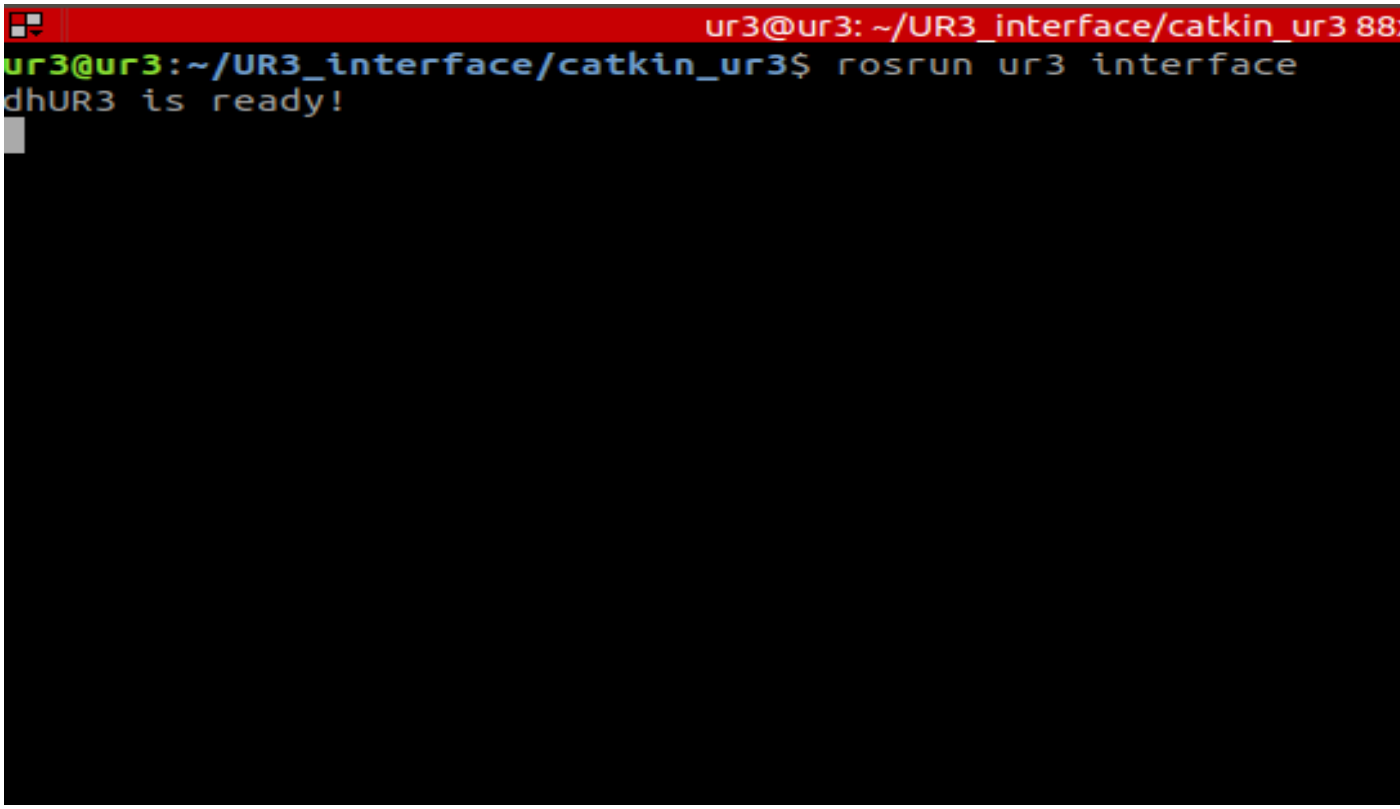



Fig. 14: Figure 14.

At this point, the robot should start making a repetitive movement and after a few seconds it will stop. After that movement for the robot will have executed demo_1.

### View topics

Topics are data structures in which the variables of the robot joints are published. In this robot, there are two topics, but for this experiment, we will focus on just one that will be the topic **arm**.

In order to observe the data present in the topic **arm**, we will type, in another section of **Terminator** the following command:

- $ rostopic echo /arm

Figure 15, shown below, shows the result of this command with its respective variables.

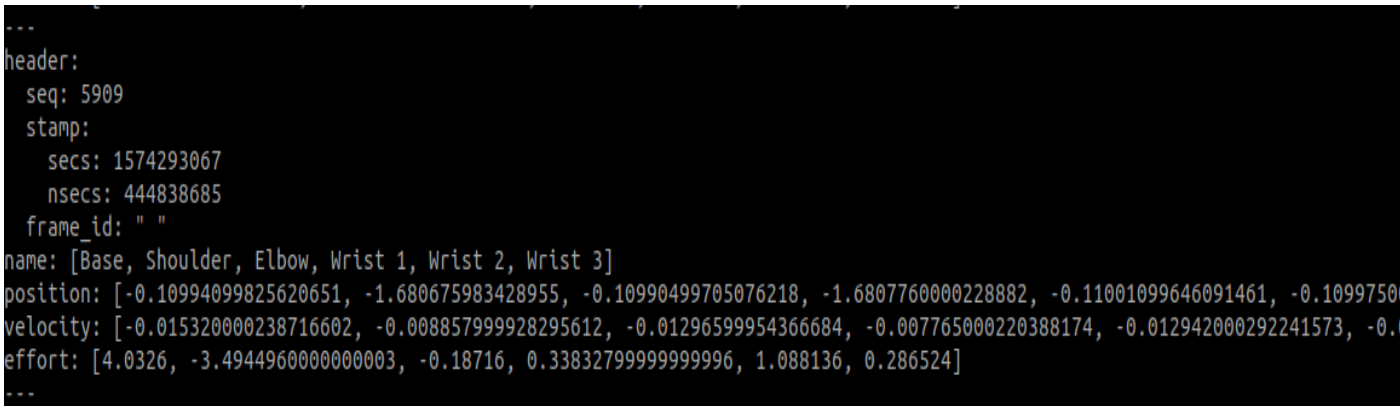



Fig. 15: Figure 15.

You can “kill” this command by typing ctrl + C.

### rqt_plot: Graphical data visualization interface

To have a visualization of the joint data in a graphical interface, we will use the application rqt_plot. This application allows the user to observe that of a variable over time.

Given that explanation, let’s use the tool. To do this, type the command shown below.

- $ rqt_plot

Figure 16, shown below, shows the application rqt_plot with one of the variables (joint speed 0 (zero)) being shown over time.

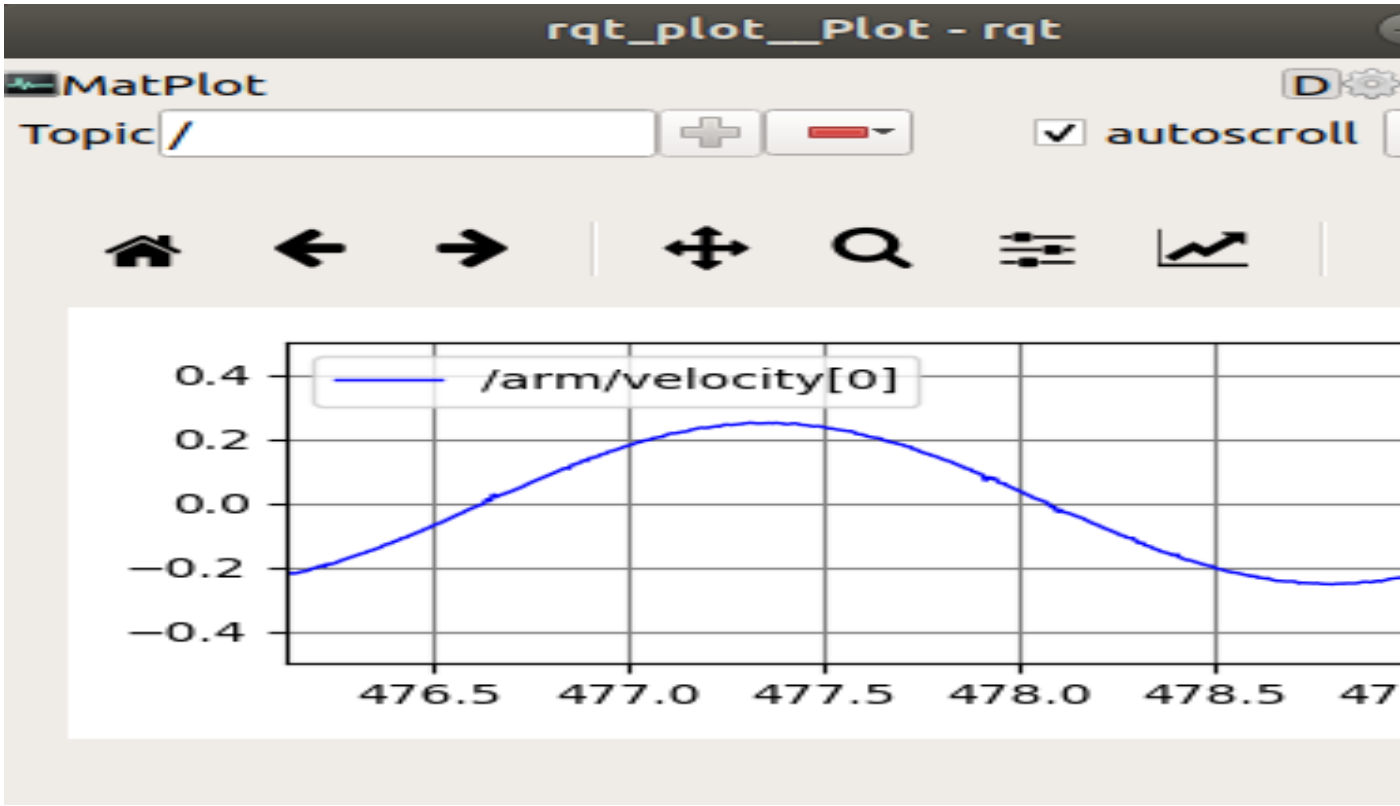



Fig. 16: Figure 16.

To see a particular variable, type/arm/variable[joint number] in Topic below MatPlot.

### Saving the experiment to a file

To save the data, just type the following command:

- $ rosbag record /arm

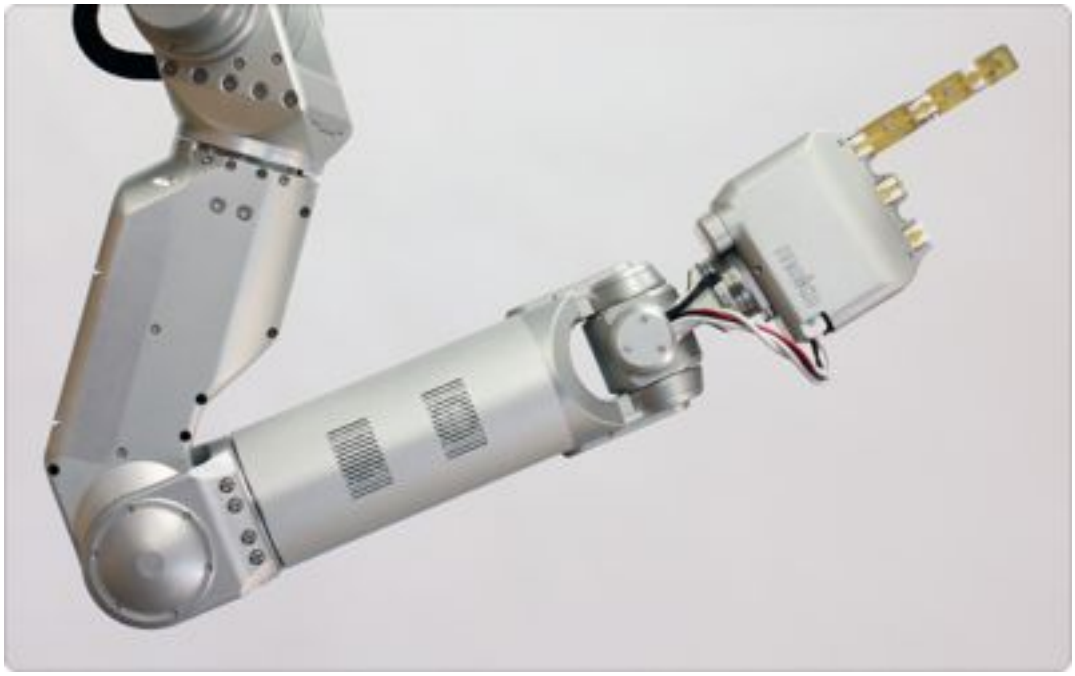
With this command you will write the data for all topics in a bag file.

### Turning off Robot and Computer

To turn the robot off, press the **POWER** button, figure 1, and then click **Power Off**.

To shut down the computer, close all **Terminator** tabs and shut down Ubuntu normally.

## 3.2 Meka



## 3.3 Schunk

CHAPTER 4

# Documentation

Hi,

The official site for Cooperative Robotics is https://cooprobo.readthedocs.io/ or https://cooprobo.rtfd.io/

This site should contain all documentation towards the hardware, software, modelling and tutorial. This site is builded using the official project repo https://github.com/lara-unb/CoopRobo/

## 4.1 References

**About this site**

- https://readthedocs.org/
- https://github.com/RobInLabUJI/ROSLab

**About ROS programming**

- https://github.com/ethz-asl/programming_guidelines/wiki

**Best maintainer practices**

- https://github.com/leggedrobotics/ros_best_practices/wiki
- https://github.com/leggedrobotics/ros_best_practices/blob/master/ros_package_template/README.md

**Programming tips**

- https://lara-unb.github.io/dicas-programacao/#/

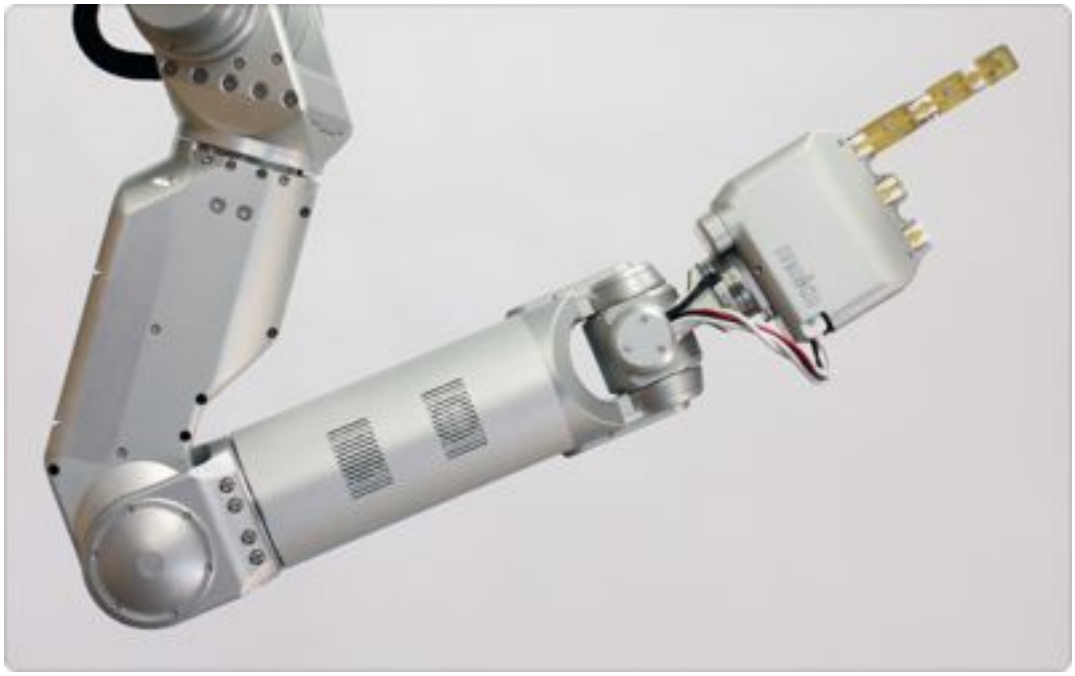
**ReadMe Exemple**

- https://github.com/Gastd/consensus

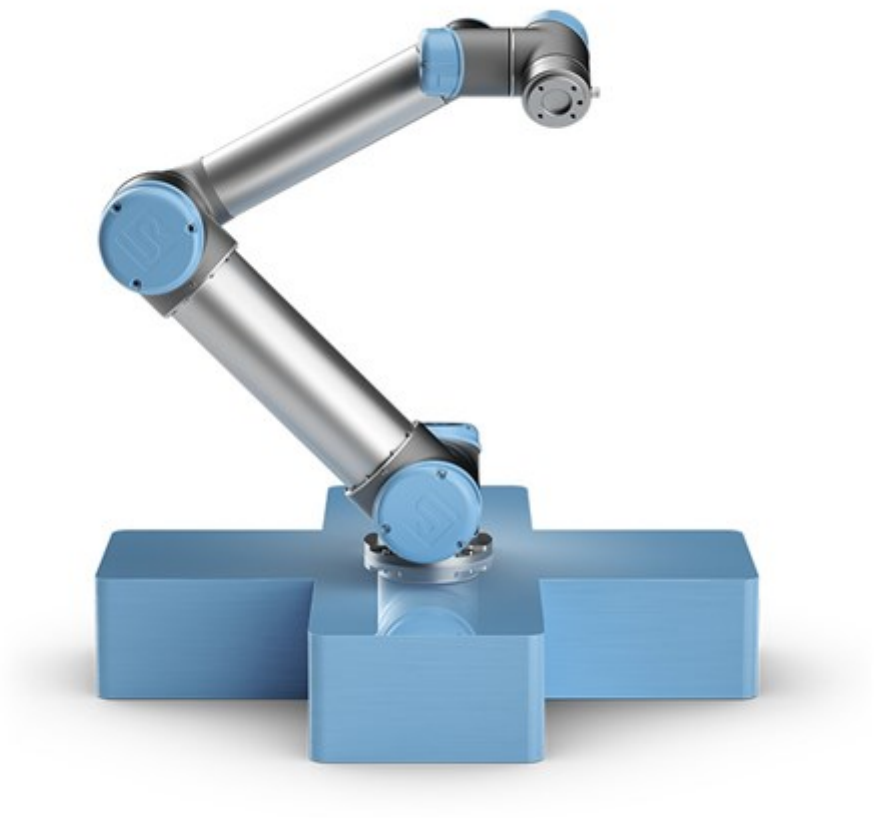
CHAPTER 5

# Camera

## 5.1 x



## 5.2 y



### 5.2.1 System Description

#### Overview

The UR3 is a table-top collaborative robot. With its 3 kg payload it is very capable and its small footprint makes it suitable for limited workspace situations. With its infinite turn on the end joint, several activities can be perfomed with grippers attached at robot tool connector.

Some of its applications:

- Laboratory work
- Assembly tasks
- Polishing
- Soldering
- Gluing
- Screwing
- Painting
- Pick and place
- Operating hand tools

- Fume hood tasks

### Description

Table 1: UR3 Description

| | |
|---|---|
| Weight | 11.2 kg |
| Payload | 3 kg |
| Reach | 500 mm |
| Footprint | Ø 128 mm |
| Degrees of freedom | 6 rotating joints |
| Joint ranges | +/- 360°, infinite rotation on end joint |
| Speed wrist joints | 360 degrees/sec |
| Other joints | 180 degrees/sec. |
| Noise | Comparatively noiseless |
| IP classification | IP64 |

### Communication

- TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX
- Ethernet socket & Modbus TCP

### Control Box

#### Linux PC

### 5.2.2 Kinematics

### 5.2.3 Dynamics

### 5.2.4 Robots

### 5.2.5 ROS

### 5.2.6 Lab 1

## 5.3 z