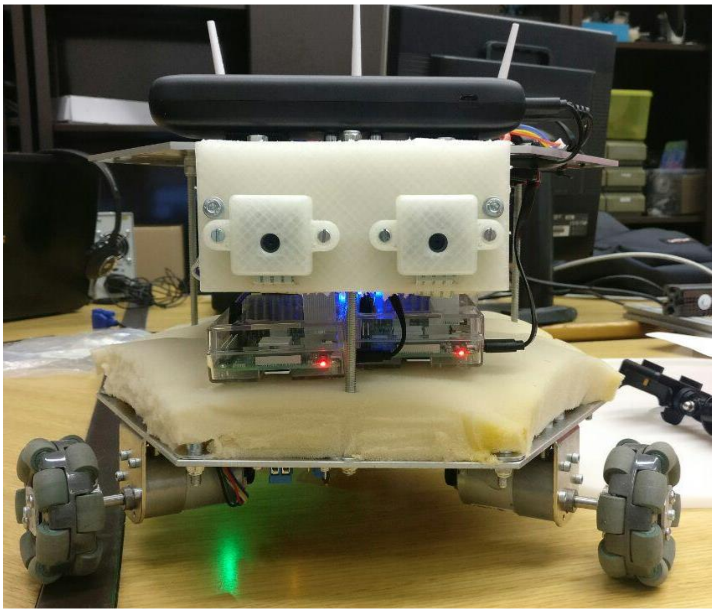
ViRHaS

ViRHaS is an omnidirectional robot, based on Triskar model, equipped with two Raspberry Pi Cameras. These cameras can be used either to watch video streaming from multiple devices, either to control robot in 3D mode: this mode is available both for smartphone (inside a VRBOX) and for PC plugged to a 3D-ready screen.

Website: http://airwiki.ws.dei.polimi.it/index.php/ViRHaS

Source code: https://github.com/michele-bertoni/ViRHaS

Video demo: --SOON AVAILABLE



Structure

ViRHaS is structured into 3 parts:

- Raspberry Pis are the hub of the entire robot: they handle both video and control streaming: one raspberry, the one with local address 192.168.1.3 is the master, while the other (192.168.1.4) is a slave device and only handles video transmission;

-Arduino Mega is used to control the motors: it's a slave device, controlled by master Raspberry Pi: Raspberry sends over USB serial to the Arduino a byte vector, which contains all speed components; then Arduino computes PID and control motors over PWM;

- User device: it may be any modern electronical device, for example laptop, smartphone, desktop computer, etc... ; the device receives video streaming and sends controls.

Software components

All software components may be divided into four parts:

- Video capture: performed only by Raspberry Pis, uses v4l2 libraries to get MJPG streaming from cameras; all data is stored in ByteArrays;

- Communication: performed by all the parts of the robot, in particular using WebSockets from Raspberries to user device (and vice versa) and via USB Serial communication from Raspberry to Arduino Mega;

- Video visualization: obviously, only user device can display video streaming: once received a new ByteArray containing a frame, the device loads the JPG image and displays it in the right position of the screen (user might change settings) using OpenGL and hardware acceleration of the GPU, if present;

- Control handling: each device, except the slave Pi, handle control:

    - user device is directly connected to the controller, which can be any kind supported by user device; the gamepad is handled transparently from the user device, which combines a key to

    a default command (for example throttle, direction, etc...); this association may be changed by the user;

    - Master Raspberry receives default commands, and from these calculates strafe, forward and angular components of the speed vector, then simplifies each component to a single byte for

    a simpler parsing by the Arduino;

    - Arduino receives a 4 bytes string, composed by the 3 values and the string terminator 'n'; after parsing, it computes inverse kinematics and estimates each wheel speed; finally, target

    speed enters the PID loop and the real speed is sent to each wheel via PWM.

Development

The first part of the project I developed was User Application: written in C++, using Qt libraries, this application is multiplatform, reliable and user-friendly: Qt compiler can generate executable file from the source code for almost every device.

Meanwhile, I developed "controller", which is the software running on Raspberry Pi: this is a simple command line interface which handles both communication and video capture. This software works in any device with a camera and v4l2 libraries installed. This is also written in C++, using Qt libraries.

Then, I developed the physical robot, the hardware component of the project, which is composed by 3 motors, 3 omni-wheels, 2 drivers, 1 Arduino Mega, 2 Raspberry Pi 3.0, 2 Raspberry Pi Cameras (v2.1), 1 router, 1 LiPo 12V battery, 1 LiPo 5V powerbank and all necessary wires.

In the end, I wrote the code for Arduino: I developed library MR001004.h to interface drivers(C++), I adapted library Triskar.h, to maintain compatibility with drivers(C++) and then I wrote the ASrduino code for serial communication, data parsing and ViRHaS movement.

ViRHaS App

ViRHaS application is simple but complete; its strength is cross-compatibility, in fact it works fine on almost every operating system:

- Linux (any distribution): perfectly working

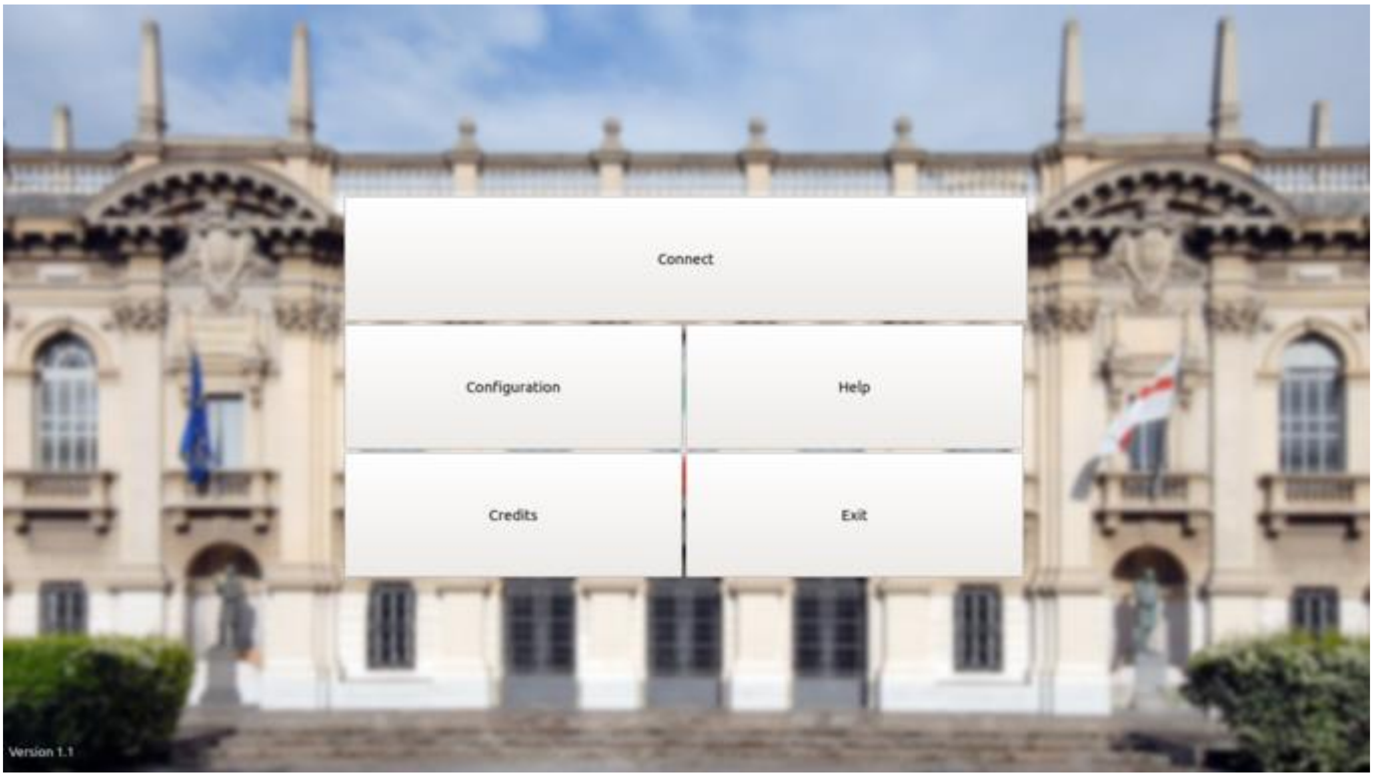
- MacOS: perfectly working

- Android: perfectly working

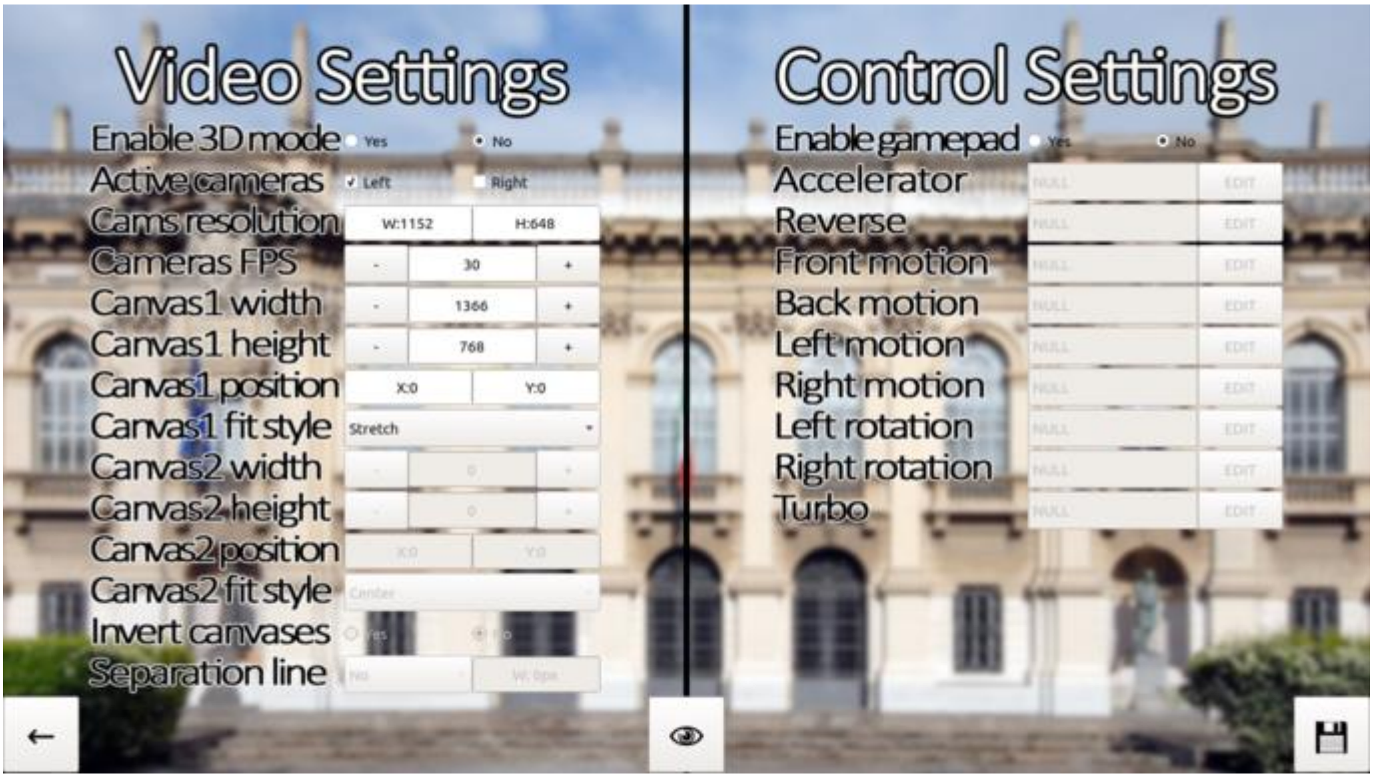
- iOS: theoretically, software should work, but, since this is a closed OS I could not test it

- Windows: unfortunately, the application on this OS has two known issues, both caused by Windows itself: the first is the incompatibility of the library QtGamepad with this OS, the second is a bug concerning window update immediately after stopping OpenGL.

Regarding the app structure, in the main menu, user can choose between some features: the most important ones are Connect and Settings.



Settings page may be used to edit almost every parameter, both view and control; favourite settings may be stored and automatically loaded when application starts. Preview button shows a 5 seconds preview of canvases size and position.



Video Settings

Enable 3D mode ○ Yes ● No

Active cameras ☑ Left □ Right

Cams resolution W:1152 H:648

Cameras FPS - 30 +

Canvas1 width - 1366 +

Canvas1 height - 768 +

Canvas1 position X:0 Y:0

Canvas1 fit style Stretch

Canvas2 width - 0 +

Canvas2 height - 0 +

Canvas2 position X:0 Y:0

Canvas2 fit style Center

Invert canvases ○ Yes ● No

Separation line No W: 0px

Control Settings

Enable gamepad ○ Yes ● No

Accelerator NULL EDIT

Reverse NULL EDIT

Front motion NULL EDIT

Back motion NULL EDIT

Left motion NULL EDIT

Right motion NULL EDIT

Left rotation NULL EDIT

Right rotation NULL EDIT

Turbo NULL EDIT

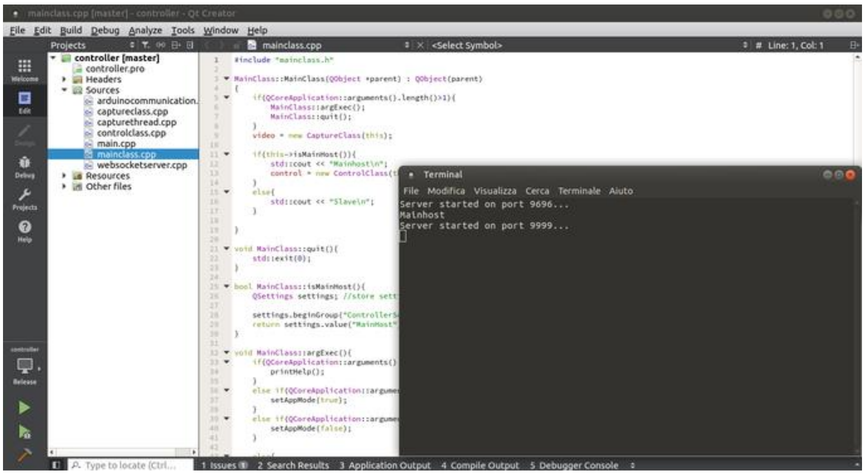
Connect page is the one used for playing with the robot: the entire page is filled with 2 canvases, inside which streaming is displayed. To return the main menu and close connections, user must click or tap any part of the screen, then a small arrow button is displayed in the bottom left angle for a few seconds: by clicking that button the game returns to the main menu and all open sockets are closed.

To compile and install ViRHaS application, you need QtCreator, which can be downloaded here: https://www.qt.io/download-open-source/ . During installation process, select all libraries, otherwise, something should not work fine when compiling project. To compile an apk file for Android, you need the Android version of QtCreator (https://www.qt.io/download-open-source/#section-2); then, after setting up Android compiler, plug the smartphone to the computer USB (debug mode) and run compiler.

ViRHaS controller

This Qt command line program is not as multiplatform as the application, because it uses some libraries available only on Linux: one of these libraries is v4l2 (Video4Linux2), which is not available, for example, on Windows .

When program starts with no parameters, it reads settings to understand if this Raspberry is the master or the slave one. This setting is automatically set permanently by running once the program with the parameter "-c" for the master and "-s" for the slave.



```
#include "mainclass.h"

MainClass::MainClass(QObject *parent) : QObject(parent)
{
    if(QCoreApplication::arguments().length()>1){
        MainClass::argExec();
        MainClass::quit();
    }
    video = new CaptureClass(this);

    if(this->isMainHost()){
        std::cout << "Mainhost\n";
        control = new ControlClass(t
    }
    else{
        std::cout << "Slave\n";
    }

}

void MainClass::quit(){
    std::exit(0);
}

bool MainClass::isMainHost(){
    QSettings settings; //store sett

    settings.beginGroup("ControllerS
    return settings.value("MainHost"
}

void MainClass::argExec(){
    if(QCoreApplication::arguments()
        printHelp();
    }
    else if(QCoreApplication::argume
        setAppMode(true);
    }
    else if(QCoreApplication::argume
        setAppMode(false);
    }
```

```
Server started on port 9696...
Mainhost
Server started on port 9999...
```

If Raspberry is slave, the program creates only one server for the video, while if it's master, the program creates one more server to handle controls reception. When a socket is opened between client and server, if it's a video socket, camera starts shooting and ByteArrays containing frames are sent over the socket; instead, on the control socket, ViRHaS controller receives default commands, then elaborates them and sends them to the Arduino, as described before.

To compile and install this software on Raspberry Pi, since Qt Creator IDE may be too heavy, I suggest installing the command-line compiler, which unfortunately comes only with default packages; the following packages must be installed with command "sudo apt-get install <library_name>":

- qt5-default: Qt compiler

- libv4l-dev: v4l2 libraries

- libqt5websockets5-dev: Qt libraries for WebSockets (if it doesn't compile, install also qtwebsockets5-dbg)

- v4l-utils: Camera information (usage: v4l2-ctl --list-formats-ext)

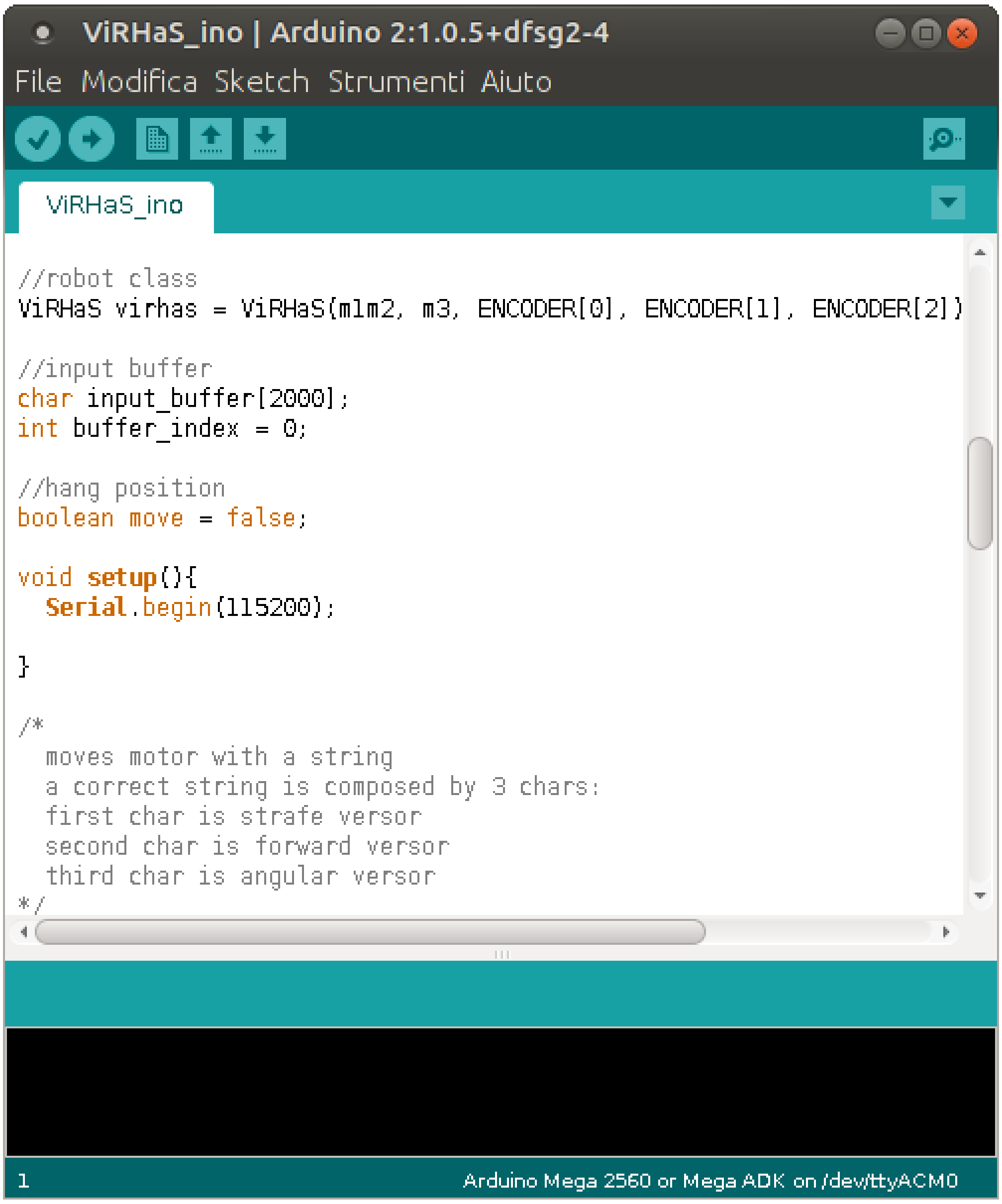
- libgl1-mesa-dev: openGL libraries

- libqt5serialport5-dev: serial communication libraries

In order to compile the project, you have to be in the same directory as .pro file, then digit the commands "qmake" and then "make": the executable will be in the same directory. Before launching the controller, a few more steps are necessary: on Raspberry Pi, camera and serial are not enabled by default, so type "sudo raspi-config", then enable both camera and serial; then add the module "bcm2835-v4l2" with "sudo nano /etc/modules" and, after reboot, everything should work fine. Now navigate to the directory where you compiled the controller executable and type "./controller -c" if this Raspberry is connected to the Arduino Mega, otherwise "./controller -s": this process is reversible and you can change mode in any moment. Finally type "./controller" to launch ViRHaS controller.

ViRHaS Arduino

ViRHaS_ino software is simple: on boot, the Arduino instantiates 3 Encoder objects (library Encoder.h), 2 MR001004 objects (library MR001004.h, written by me for motor drivers MR001-004) and from these objects instantiates a ViRHaS object (library ViRHaS.h, created by me, modifying the Triskar.h library); setup() opens serial connection, then loop() reads serial connection and computes PID.


ViRHaS_ino | Arduino 2:1.0.5+dfsg2-4
File Modifica Sketch Strumenti Aiuto
ViRHaS_ino

```
//robot class
ViRHaS virhas = ViRHaS(m1m2, m3, ENCODER[0], ENCODER[1], ENCODER[2])

//input buffer
char input_buffer[2000];
int buffer_index = 0;

//hang position
boolean move = false;

void setup(){
  Serial.begin(115200);

}

/*
  moves motor with a string
  a correct string is composed by 3 chars:
  first char is strafe versor
  second char is forward versor
  third char is angular versor
*/
```

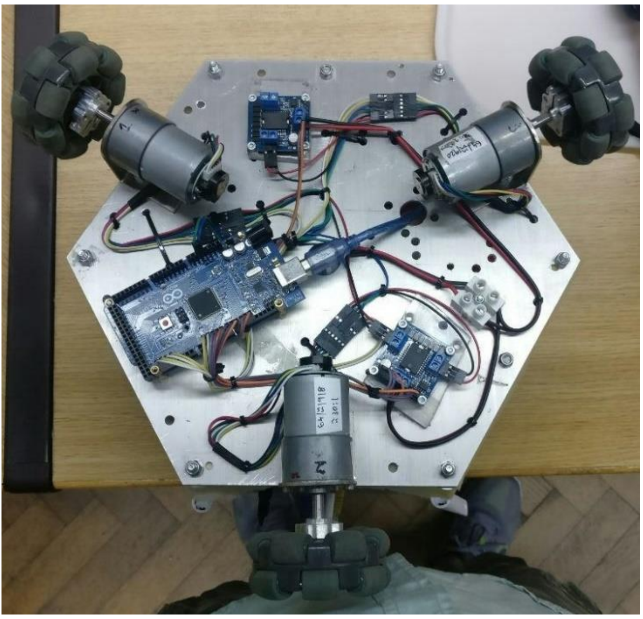
1 Arduino Mega 2560 or Mega ADK on /dev/ttyACM0

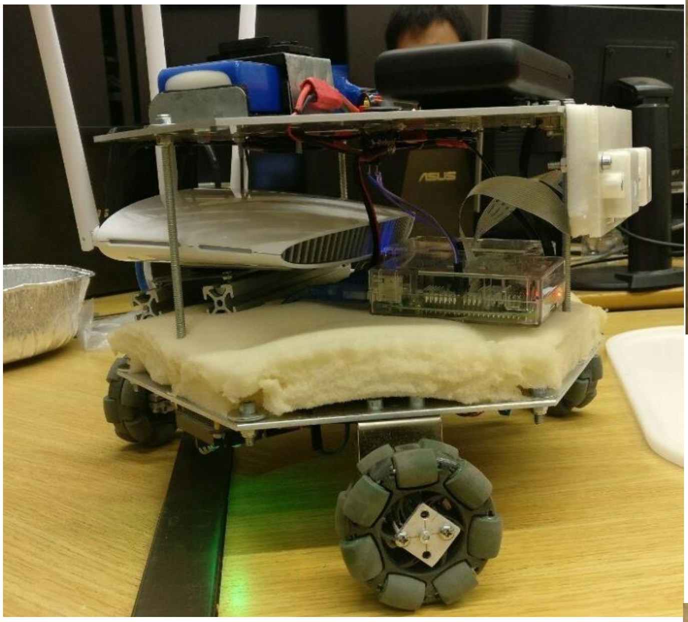
To compile ViRHaS_ino.ino, add all necessary libraries and then just compile for Arduino Mega 2560.

ViRHaS Robot

The ViRHaS robot is built on the base of a Triskar model, a holonomic three-wheeled robot. ViRHaS is bigger than Triskar, because of the space occupied by Raspberry Pi, cameras and router. ViRHaS has three layers:

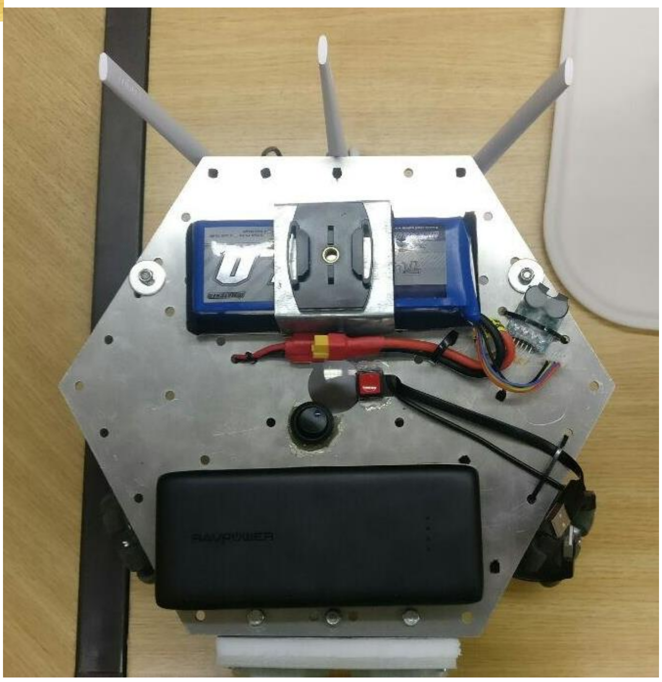
- the bottom layer is the actual robot base and hangs motors, drivers, Arduino and all wires between them





- the middle layer is made of foam rubber and is fixed to the layer 0: foam is necessary for isolation and is also good for hiding cables and bolts under it; this layer hangs 2 Raspberries and the router

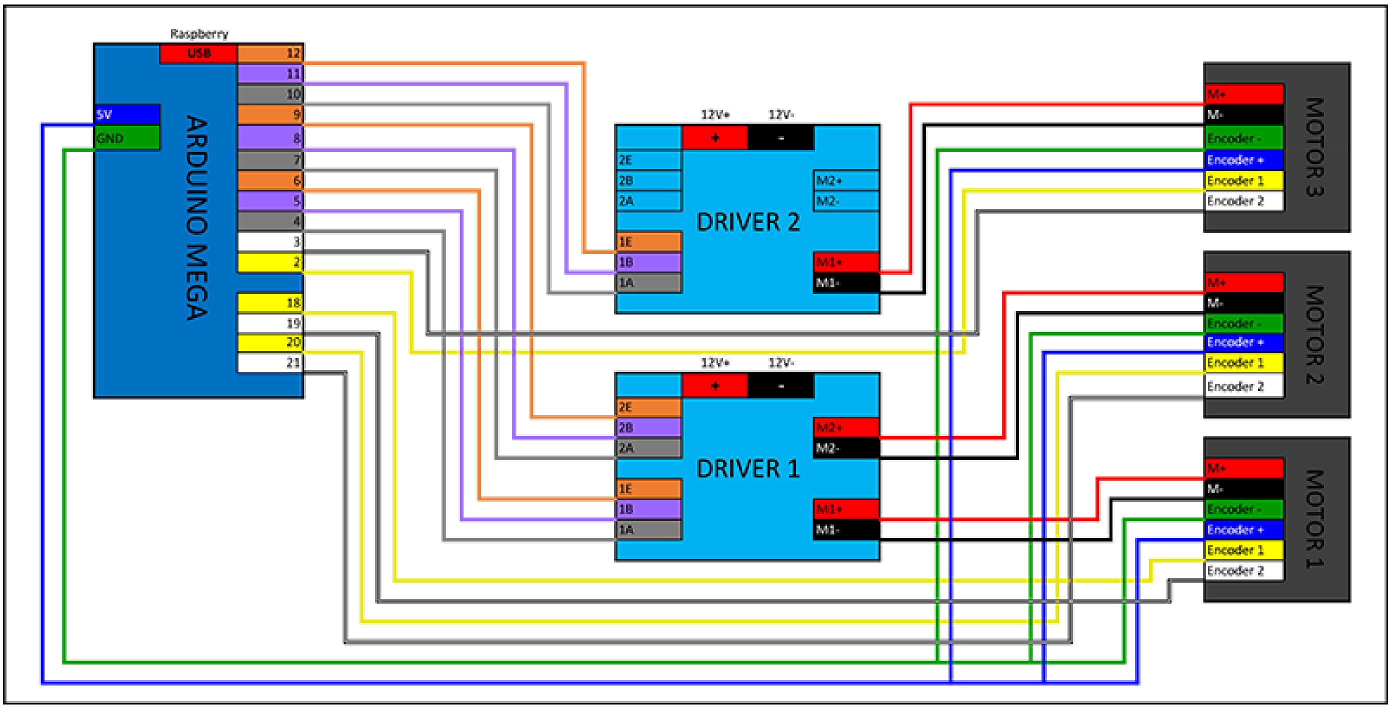
- top layer is fixed to bottom layer with three threaded rods and some dice; the bottom face of this layer hangs cameras support and hides power cables, while the top face hangs the two batteries, the LiPo buzzer and two buttons (motors and raspberries' switches).



Configurations

Robot, for working fine, needs some additional settings:

- Arduino and drivers' wirings - Arduino has 9 output, 6 input and 2 power pins: outputs are pins from 4 to 12, inputs are all interrupt pins (2, 3, 18-20) and power pins are 5V and GND; the following schema describes completely all wirings



(higher resolution here: https://github.com/michele-bertoni/ViRHaS/blob/master/pins_connections.pdf)

- Arduino must be connected to the master Raspberry Pi via USB: this both gives power to Arduino and allows serial communication

- Motor drivers must be directly connected to the 12V battery: the two drivers must be connected in parallel, with a switch between battery and drivers

- Raspberry Pis and router must be connected to the 5V USB powerbank

- Raspberries must be connected to the router with ethernet cables: IP address of the master Pi must be set to 192.168.1.3, while the one of the slave Pi must be set to 192.168.1.4; for doing this, open router settings and in LAN page pair each MAC address with the right LAN IP

- Raspberries must have a power-off button: since unplugging before shutting them down could cause errors, a switch is necessary; I recommend following instructions in this video: https://www.youtube.com/watch?v=Ns0JdZrXwdk

- Raspberry may be set to CLI Autologin mode to speed up boot: type "sudo raspi-config", then edit the settings from that menu

- ViRHaS controller must be executed on boot: type "crontab -e", then add a line on the bottom of the file as follows (controller executable is inside /home/pi/controller directory)

```
@reboot /home/pi/controller/controller
```

- Cameras must be fixed tightly so that they can't move: the focal distance between the two cameras must be about 6.5cm for a medium 3D effect; if this distance is increased 3D effect will be higher for farther distances, but nearer objects will be split by eyes into two different objects; conversely, if focal distance is decreased, nearest object will be always seen fine, but 3D effect will be lower

Conclusions

ViRHaS is only a simple game, but at the same time is very amusing. Unfortunately, low price hardware constrains the user experience: VRBox 3D is mediocre, smartphone is way slower than a PC (so cameras resolution must be decreased to keep FPS stable) and router connection radius is about 20 meters. In my opinion with better hardware, an improved version of ViRHaS can be used even for military purposes: 3D vision and virtual reality headset helps a lot when driving any robot. On the other hand, ViRHaS software would perfectly work with better hardware: when, in a few years, we'll have faster and cheaper hardware, ViRHaS can be upgraded with no effort.

If only used for ludic purposes, ViRHaS is already great: in fact, latency is low (<0.1 s) and video streaming with one camera can be over 720p 30FPS (480p each with two cameras). Also, high application customization is helpful: even if they're the same model, the two cameras may be lightly different, but changing a few settings, can immediately solve this problem.

I'm really satisfied for my work on ViRHaS and I would like to continue working on it: improvements I would like to do are:

- add servo motors to rotate cameras when rotating head

- use H264 protocol instead of MJPG

- find a way to use UDP without losing many packets so that I can use multicast and access the same camera from different devices at the same time

. improve robot appearance

- add sensors for collision avoidance

Michele Bertoni