Design and Implementation of a Quadcopter

By

Marcos Negreira

Felipe Soares

GROUP 18

**Date:** December, 2015

**Professor:** Dr. Bassem Alhalabi,

Table of Contents

1 Introduction 3

# 1 Introduction

In this project, a QuadCopter is designed and implemented to autonomously fly and demonstrate on-board GPS processing capabilities. The project is aimed to control the QuadCopter by virtue of two control systems; the local control, and the global control. The local control is intended for the stabilization of the QuadCopter, so that it is able to fly manually through the RC receiver and the global control is intended to make the QuadCopter to fly autonomously with the aid of control through the GPS. These two control loops are implemented on dedicated processors with appropriate capabilities.

# 2 System Architecture

In this section, we will discuss the used hardware and system architecture. First we will take a look at the local control, global control and used sensors, and after that, we will address communication used between these components.

###### 2.1 Local Control

The local control is powered by CC3D development board (figure 1), powered by the ~CC3D~ processor 16 MB flash memory. This board also contains a gyroscope, accelerometer and a magnetic sensor, all measuring three axis. This makes it an ideal board for a quadcopter’s local control, because it contains enough processing power, and has all the basic sensors available. A SPEKTRUM AR610 was used for testing manually.

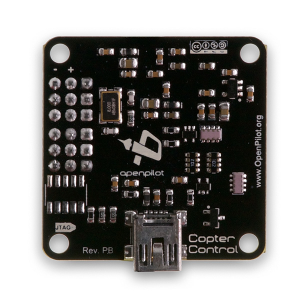


Figure 1: An CC3D.

###### 2.2 Global Control

For the global control, we have chosen for the Raspberry PI (figure 2). We chose for this platform because it was cheap, we already had it available, and the camera module was expected soon after the start of the project.



**Figure 2:** A Raspberry PI.

###### 2.3 Sensors

The local control board already contained the basic sensors required to fly a quadcopter: a gyroscope, an accelerometer and a magnetic sensor. To this list, we added a battery monitor.

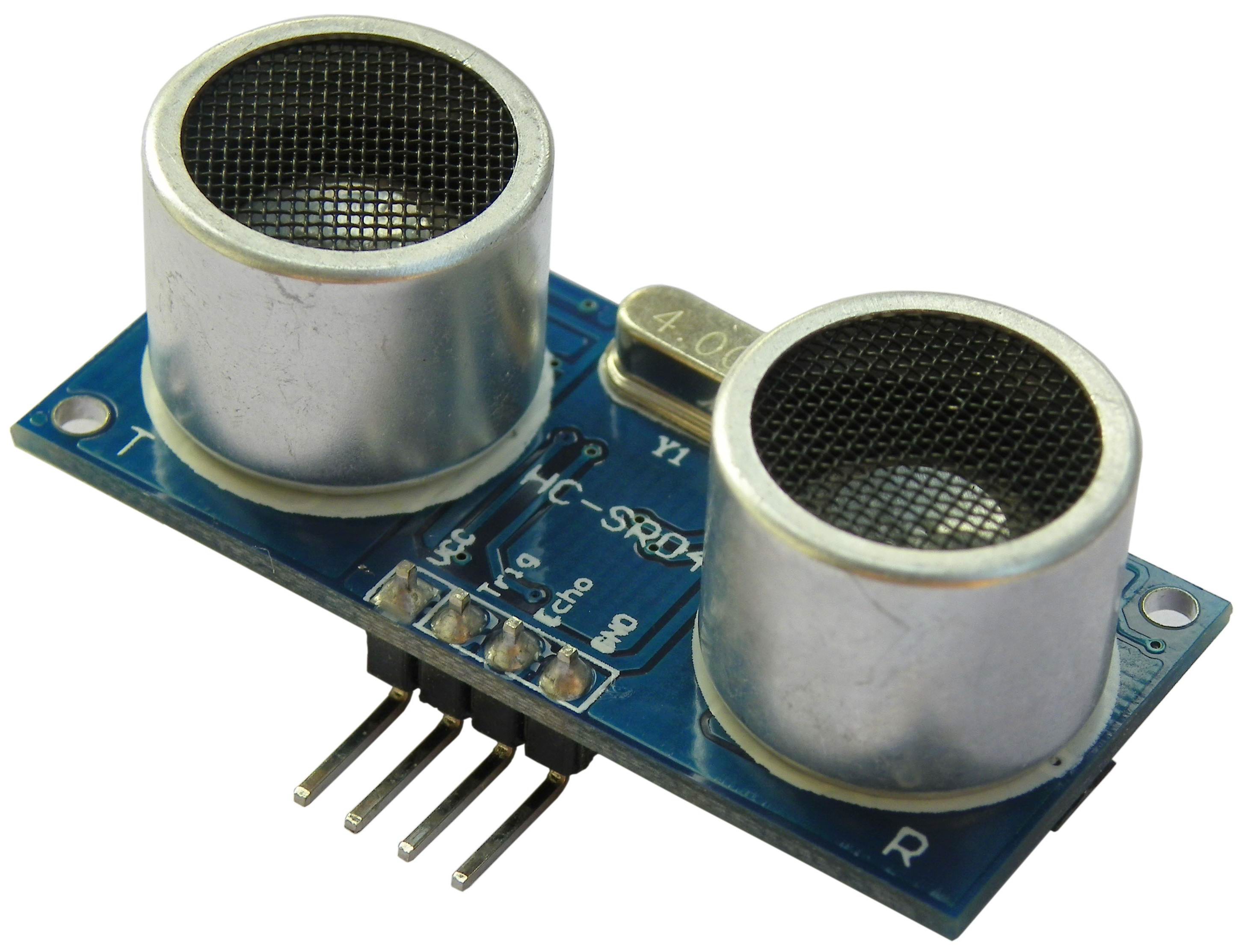
###### 2.3.1 Battery monitor

The battery monitor was a simple voltage divider that connected the battery voltage to one of the ADC pins of the CC3D board. The voltage divider was necessary to convert the voltage to the 0 – 3V domain.

###### 2.3.2 Ultrasound

Initially the plan was to use the ultrasound sensor (figure 3) for tracking the altitude of the quadcopter. The HCSR04 sonar driver was requested from Taulabs development team and merged to our code base. The ultrasound driver implementation could be obtained in Github

link1. As we further realized, the ultrasound sensor suffered propeller sound interference which made the sensor values noisy and inaccurate. This happened because there were high frequency components in the sound waves emitted by the propellers and it fell in the band of 40 KHz which was the operating frequency of the sound waves used for ultrasound sensor, hence this interference resulted in changing our design decision to use alternative altitude tracking sensor such as barometer instead of the ultrasound sensor.



**Figure 3:** The HCSR04 sensor.

1 <https://github.com/mnegreira/TauLabs>

###### 2.3.4 Cameras

We have chosen for the Raspberry PI cameras, because they are affordable, of relative good quality, small and light, and they uses few system resources on the Raspberry PI. Because each Raspberry PI can only interface with one camera. (This was not part of the Scope of the project)

These cameras are able to film in 1080p resolution with 30 fps, and take still images with

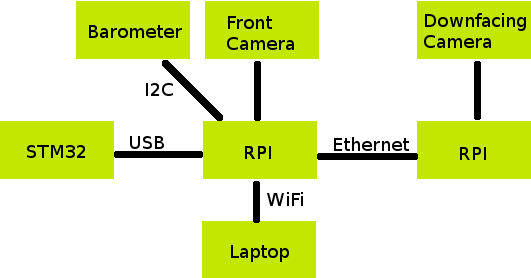
2592 x 1944 resolution.



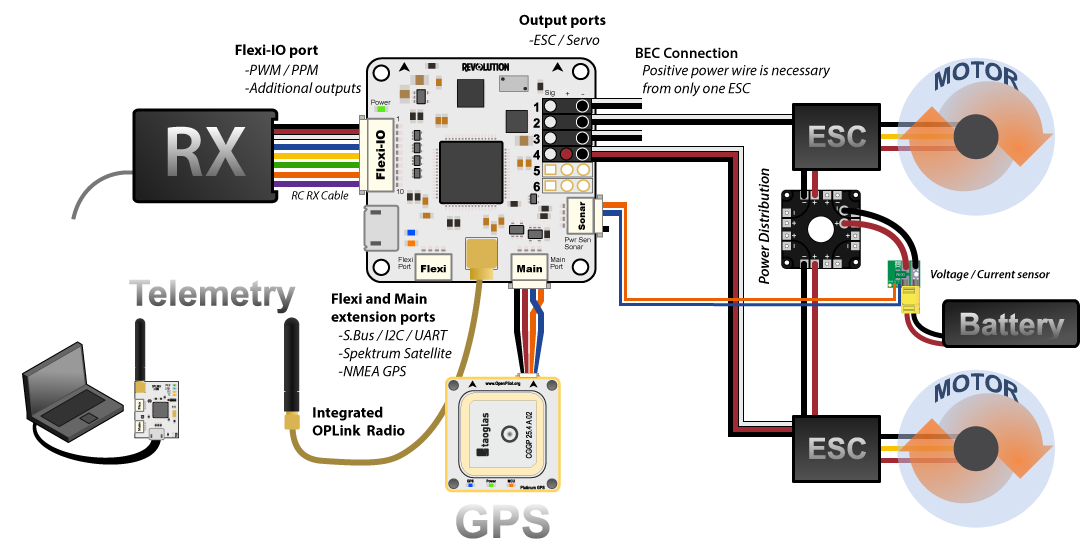
**Figure 5:** Raspberry PI with camera module.

###### 2.4 Communication

There are many communication-paths, as can be seen in figure 6. In this section, these will all be discussed.



**Figure 6:** Overview of different communication protocols.



**Figure 6.1:** Overview of different communication protocols.

###### 2.4.1 Local control and global control

The local control on the CC3D communicates with the first Raspberry PI using an USB-cable. They use the UAVTalk2 protocol, and thus communicate high level objects. Because the UAVTalk protocol is already seamless integrated in the local control (it is also used for communication with the Ground Control Station), all useful pieces of data are avail- able as a UAVObject.

This makes it an extremely powerful interface. Global control algorithms have access to all the information that it available to the local control, and it can influence and override all aspects of the local control by writing new values to these objects.

The interface is also flexible. Currently it uses USB, but UART or other protocols are also possible. The UAVObjects on the local control are also accessible on a laptop and on the other Raspberry PI, using a WiFi or an ethernet connection, thus eliminating the need for a separated (bluetooth) telemetry module.

###### 2.4.2 Laptop

On the Raspberry PI runs a normal Linux installation, and that gives the option to add a WiFi adaptor so that it can communicate with the outside world using WiFi. We did this, and even shared the internet connection to the second Raspberry PI, so both could easily download and install new software.

2 <http://wiki.openpilot.org/display/Doc/UAVTalk>

###### 2.4.5 Camera

The cameras are attached to the Raspberry PIs, and use a flatcable for the connection to a

CSI3-interface.

**3 Software**

To get all this hardware flying, we need some software to control everything. We have software for local and global control, and both will be discussed in this section.

###### 3.1 Local control

For the local control, we chose for the open source project TauLabs4 . This project is a fork from the OpenPilot project. We choose for this project, because is was already ported to the hardware we where using. Also a SPEKTRUM AR610 was used for control.

This project provided not only software for the local control, there was also a “Ground Control Station” (GCS). This program runs on your computer and communicated with the local control to provide configuration options and means of debugging and monitoring.

###### 3.2 Global control

We have written our own global control program. This program runs on a Raspberry PI, and is a stripped down version of the GCS provided by TauLabs. This allows it to communicate with the local control using UAVObjects, as is discussed in section 2.4.1. Our global control code is also available on GitHub5.

The global control has a fixed core, and allows functionality to be added by modules. We did not have enough time to write really interesting modules, but we do have some “proof- of-concept” modules, which will be discussed next. These plugins show that it is possible to read and write data from/to the local control.

###### 3.2.1 Recorder plugin

This plugin monitors the “Armed” flag in the “FlightStatus” UAVObject, and as soon as the quadcopter gets armed, it starts recording video on both Raspberry PIs. On disarming, the recording stops. This allows us to easily create in-flight footage with the cameras.

3 Camera Serial Interface

4 <https://github.com/TauLabs/TauLabs>

5 https://github.com/mnegreira/TauLabs

**4 Objectives**

Here we will discuss the objectives given in the course, and how we handled them.

###### 4.1 Building Quadcopter and flying

We have build our quadcopter, and made fly. See figure 7 for a photo.



Figure 7: Our quadcopter.

###### 4.2 Tuning Local control

We spend a lot of time on PID tuning, but only in the end we found that some of the settings where completely wrong; the direction that the motors where spinning was configured wrong, so yaw corrections resulted in more yaw, crashing the quadcopter. Also one of our speed controllers was not functioning correctly, which led to an unstable quadcopter.

###### 4.3 Altitude hold

Altitude hold should have been possible with the barometer installed, but there was no time to actually test this.

###### 4.4 Autonomous flight

We did not manage to get the quadcopter fly autonomously, because it was not stable enough and still needed some manual corrections, and we didn’t have altitude hold working.

###### 4.5 Drift correction in horizontal axis

We where unable to implement this, because we didn’t have altitude hold and autonomous flight.

**B Partlist**

Here follows a list of all the parts and components that we have used while building this quadcopter.