Embedded Visual Control
Final report

Martijn van den Dungen(#0647403)
Vlad Spiridonescu(#0788600)
Alexandru But(#-)

September 24, 2012
# Contents

1 Project description .................................................. 5
   1.1 Introduction ......................................................... 5
   1.2 What is a quadcopter ............................................. 6

2 Design considerations ................................................. 7
   2.1 System overview .................................................... 7
   2.2 Needed resources .................................................. 8

3 Hardware ........................................................................ 10
   3.1 PCB .......................................................................... 10
   3.2 Microcontroller ....................................................... 12
   3.3 Motor control ........................................................... 12
   3.4 Motors ....................................................................... 13
   3.5 Battery ....................................................................... 13
   3.6 IMU ........................................................................... 14
   3.7 Propellers ................................................................. 14
   3.8 AVR programmer ....................................................... 14
   3.9 Camera ...................................................................... 15
   3.10 Processing nodes ...................................................... 15

4 Global control ................................................................. 17
   4.1 Ball detection ......................................................... 18
   4.2 Distance estimation ............................................... 20
   4.3 Trajectory generation .............................................. 22
   4.4 Alternative approaches ......................................... 24
      4.4.1 Petalinux ............................................................ 24
      4.4.2 Microblaze ......................................................... 24

5 Local control ................................................................. 26

6 Conclusion .................................................................... 30
A  Work distribution  31
B  Course organization  32
Abstract

In order to manage the building of an entire robot from scratch, a great deal of various abilities and knowledge is involved.

The project revolves round the building of a quadcopter (a multicopter lifted and propelled by four motors). Design decisions were taken according to certain requirements: small price, flexibility, maintainability and modifiability. Also the physical magnitude represented an important fact since the multitude of very small and big quadcopters already built unleashes the curiosity of trying a relatively medium size. Another important reason for our decision has been the maximum weight that can be lifted by the propellers associated with the overall size.

Elements of embedded systems, vision and control have been taken into consideration. Two processing nodes are used: a microcontroller and an FPGA. The vision aspect demands the possibility of the quadcopter to acquire images/keypoints of the environment through a specific sensor. This was attained by attaching a camera to the FPGA board. A program needs to be made that controls the quadcopter and stabilizes it.

The project has been divided into different tasks, considering the multitude of areas that are addressed. One tough job was to handle the time disproportion between some parts for which the research/development track was entirely different. Hence, there were moments when physical parts were long-expected to be delivered while, for another task, research was made to find the optimal solution that would satisfy the proposed requirements. At the time, practically no work was done and this cannot be truly estimated in advance but just by supposition.

During the project we focused on obtaining results that are significant for those interested in building such a robot, hence various approaches and ideas were welcomed and tried extensively to be implemented.

Methods and results

In the beginning, the proposed method constitutes the designing from scratch of the PCB on which the microcontroller and motor controllers will be mounted. Different problems occurred while working on this idea but the result is satisfactory. The choice of the FPGA over the other available solutions is related to its small price/size/weight and a great configurability.
Further details about the project will be discussed later in the report at their designated chapters.

Since everything comes with a price, a small and cheap FPGA means small processing power. The chosen application is a motion detection one, namely, finding the ball in an image and afterwards follow it.

Various videos/designs found on the internet give a very good view of the possibilities of the construction and also related to its programming / controlling. Since the time was insufficient to finish the quadcopter due to different delays that appeared in the beginning and also the discontinuity of the electronic PCB creation, we infer that some of the results are prototypes which will help in future implementations. Hence, the requirements of the project changed in time because of these issues.
Chapter 1

Project description

1.1 Introduction

For the course Embedded Visual Control we decided to build a quadcopter from scratch. Designing, building and programming of such a system allows us to deal with all aspects of the course (embedded systems, vision and control).

There is a large variety of both hobby projects as well as more professional versions of quadcopters, varying in size, speed and features. Reviewing these we made the decision to build a rather small quadcopter, as it has larger speed and is more flexible in movement. A downside to this approach is that the control is more difficult to achieve.

Our final purpose is to have the quadcopter detect a ball of a specific color and have it follow it at a certain distance.

In this document we outline our approach. We first discuss what a quadcopter actually is in the next section. In chapter 2 we give an overview of the system we have developed. Chapter 3 describes the hardware we have acquired. In chapters 4 and 5 we discuss global and local control respectively. We end with a conclusion in chapter 6 and show the work distribution in chapter A.
1.2 What is a quadcopter

We shall give some ideas about what a quadcopter represents and also its flying principles. We are talking about a multicopter that uses 4 rotors with propellers to lift itself. Compared to helicopters, quadrotors can fly vertically much more easily but the control is difficult. They are popular in UAV research (Unmanned Aerial Vehicle), where they are used to fly through challenging environments with lower risk of damaging the surroundings, since the rotors are much smaller than the one of a regular-sized helicopter. The four motors have two propellers rotating clockwise and two propellers rotating counter-clockwise.

As basic idea for the quadcopter to lift, all of the propellers have to rotate over a certain stabilizing threshold. For it to roll around an axis, M1 has to rotate faster. For an yaw action, M4 and M2 have to speed up. For a Pitch action, M4 has to rotate faster.

Figure 1.1 shows a basic illustration of roll, pitch, yaw and the four motors.

We refer to [1] for a complete overview of already built and available quadcopters and their specifications.
Chapter 2

Design considerations

2.1 System overview

To develop a system overview and resources needed, we identify tasks our system needs to perform.

Firstly, a ball needs to be detected. This gives the ball’s pose in 3-space and the pose of our quadcopter relative to it. As we are trying to have the quadcopter follow the ball at a specific distance a pose error can be determined from this information.

Knowing the pose error we can generate a trajectory. Having the quadcopter follow this trajectory the pose error should in time reduce to zero.
The tasks described above can be considered global control. The system this trajectory information is fed to, and which controls the actual hardware for flying is considered local control. The local control is concerned with driving the motors and getting the current orientation from sensors.

The reasoning above leads us to the global overview of the system, shown in figure 2.1.

In the next chapter we will discuss the hardware resources to compose this system.

### 2.2 Needed resources

Since we decided to make the PCB by ourselves, an etching machine is needed for it and also screws to put the boards together.

The table below shows the price and weight of the various components that are needed.
<table>
<thead>
<tr>
<th>Component</th>
<th>Price (€)</th>
<th>Weight (g)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motors</td>
<td>50</td>
<td>40</td>
</tr>
<tr>
<td>Propellers</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>Battery (4x)</td>
<td>12</td>
<td>80</td>
</tr>
<tr>
<td>Sensor Board</td>
<td>45</td>
<td>10</td>
</tr>
<tr>
<td>Microcontroller + electronic components</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>FPGA</td>
<td>80</td>
<td>70</td>
</tr>
<tr>
<td>Cables &amp; Screws</td>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>PCB</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>Camera</td>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>Remote control</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>254</strong></td>
<td><strong>350</strong></td>
</tr>
</tbody>
</table>

Also a RS232 to USB cable and an AVR programmer needed to be bought.
Chapter 3

Hardware

3.1 PCB

Having determined what our system needs to do, we can now determine the hardware resources. We need a frame to mount components on and since we develop some circuitry ourselves we etch a PCB with circuitry in the center and have some substrate protruding from the edges on which we can mount the motors and propellers. Figure 3.2 illustrates this with sizes indicated.

In order to both have a frame and a place on which to build circuitry a PCB was etched according to our design. The next picture shows a picture of our quadcopter with various components already attached.
Components are discussed in the following sections.
3.2 Microcontroller

A couple of options were analyzed in terms of what microcontroller to use. The difference being in the size/weight or capabilities. Initially a dsPIC was considered since image processing power was needed. We tried to have electronic components as small as possible so we chose between an ATMEGA16 in Plastic DIP and a Thin QFP. Because the difference in weight is very small compared to the overall weight of the quadcopter ease of soldering, the PDIP type was chosen.

3.3 Motor control

We develop our own circuitry for the motor control. Since our motors are 6-phase brushless we use 6 outputs of a shift-register where we make 2 consecutive outputs high and shift these outputs to drive all phases of the motor consecutively. Using a NAND gate with the shift-register outputs as inputs and feeding it back to the data input of the shift register we can achieve this. The clock input can be used to alter the speed of the motors. This is illustrated in the figure below.

The output of the shift register is then fed to 6 MOSFETs which are used to actually drive the motor, as shown in the next figure.
3.4 Motors

For our motors we have chosen the Suppo A1504 Brushless. These have light weight (10g) which is an advantage in our relatively small quadcopter. Its rpm is 2900KV which means that at 1V applied, it turns at 2900 revolutions per minute. This number is really important in determining the best motors and also propellers for the design. The value increases with the decrease of the motor size which makes it harder to control.

3.5 Battery

We decided to use 4 E-Fly Li-Poly 3.7V 1200mAH batteries in a 2x2 array. This should give us about 8 minutes of continuous flight. 

![E-Fly Battery Image]
3.6 IMU

As for an Inertial Measurement Unit, a Drotek board with 9 degrees of freedom was used.

It contains a digital triaxial accelerometer BMA180 from Bosch, a 3-axis compass HMC5883L from Honeywell and a gyroscope, namely the ITG3200 which is a 3-axis one from Invensese. All offer data through an I2C interface with the microcontroller board.

3.7 Propellers

GWS propellers of size 4X2.5 were chosen since these are adequate for the motors and the size of the quadcopter.

3.8 AVR programmer

As for the AVR programmer, a USB one has been considered and the USBASP V2.0 is a very good one for its price. Also it can be used with the
3.9 Camera

We have chosen the Omnivision 7670. This camera uses the SCCB (Serial Camera Control Bus), which is a modified version of I2C. As output it can give both YUV and RGB in various resolutions. It also has numerous other settings. It’s also very inexpensive at 5 €. It already comes on a breakout board, which makes it easy to connect to the FPGA.

3.10 Processing nodes

Various options are available for processing of images from the camera for visual control, retrieving data from the IMU, implementing the control loop and controlling the hardware. We have chosen an Atmel Atmega16 for local control and the Xilinx Spartan-6 LX9 Microboard FPGA for global control.

The reason for choosing two devices is that they can perform their own tasks independently. A disadvantage to this approach is that the FPGA will need
to send some trajectory generation to the Atmega when the ball has moved. However, as we already need to develop I2C communication to control the camera we can use this same module to communicate between the FPGA and Atmega.

The Atmega and FPGA combination has as advantages over other boards, such as the Beagleboard and Arduino, that they are more power-efficient, smaller in size and cheaper. The FPGA also allows us to do parallel processing, this gives us the option to do processing on one image as we are fetching the next from the camera. A disadvantage is that programming them is more complex.
Chapter 4

Global control

For the global control part of our system we use an FPGA for processing. We fetch camera data from the camera over a serial connection (I2C) and once we have retrieved the required trajectory information from an image we again use a serial (I2C) connection to update the local control with the new information. Additionally we use a serial (RS232) connection between the FPGA and PC for debug and testing purposes. The next figure gives an overview.

![Global control overview](image)

Figure 4.1: Global control overview

As we can do parallel processing with the FPGA, one module can be busy fetching image data from the camera while another is doing processing of the previous image. The next figure illustrates this.
We implemented the ball detection algorithm as described in [2] and extended this with our own method for approximating the distance of the ball. This information is then used for trajectory generation. This algorithm is easier to implement and computationally less expensive than other methods such as SIFT. The next sections give more details on these parts.

4.1 Ball detection

Since the HSV colour space is more suitable for colour separation than RGB, we use it for thresholding values that conform to our ball colour. So we find the hue (H) and saturation (S) of the ball we want to follow and binarize the image using a threshold function, i.e.:

\[ F_{x,y} = \begin{cases} 1, & \text{if } H_{x,y} < H_{\text{MAX}} \cap (S_{x,y} > S_{\text{MIN}}) \\ 0, & \text{else} \end{cases} \]

Figures 4.3 and 4.4 show the original input image and the binarized result.

Now the binarized image is downscaled to a coarse matrix (\( M_{x,y} \)). This means we divide the binary image into 8x8 windows and accumulate all the values in it:

\[
M_{x,y} = \sum_{\nu=0}^{7} \sum_{\mu=0}^{7} F_{8x+\nu,8y+\mu}
\]
In order to weigh the region of interest we take the product of the binarized image \((F_{x,y})\) with corresponding elements of the coarse matrix \((M_{\nu,\mu})\).

\[
A_{x,y} = F_{x,y} \cdot M_{\nu,\mu}; \nu = \frac{x - x\text{mod}8}{8}; \mu = \frac{y - y\text{mod}8}{8}
\]

The result of these steps are shown in figures 4.5 and 4.6. We can clearly see that regions that are not of interest are filtered out in this process.

Using the center of mass principle we find the coordinates of the center ball. Figure 4.7 shows the coordinates found for this example input image.
4.2 Distance estimation

In order to find the third coordinate (distance) from the quadcopter to the ball we use the pinhole camera model[3]. The pinhole camera model describes the mathematical relation between the coordinates of a 3D point and its projection onto the image plane. This model is not perfect as it does not take into account distortions such as blurring due to unfocused objects, making it a first-order model. Despite its imperfection it will suite our purpose just fine.

In figure 4.8 the geometry of a pinhole camera is shown. An image plane where the 3D world is projected through the aperture (at the origin \(O\)) of the camera is shown in this figure. The image plane is located at a distance \(f\) from the aperture along the optical axis. For more details of this model we refer to [3].
We now want to find the coordinates \((x_1, x_2, x_3)\) of an interest point \(R\) on the image plane. We have already found \(x_1\) and \(x_2\) previously. In order to find the third coordinate we calibrate our camera by taking a picture of the ball and measure the distance between camera and ball manually. Using the triangle similarity we find the distance.

We place a ball with known diameter \(D\) at a distance \(F\) from the camera and measure the diameter in pixels, \(d\). The focal length of the camera is then:

\[
f = \frac{d \times F}{D}
\]

When we next capture an image and its width is \(d'\) pixels we know that:

\[
\frac{f}{d'} = \frac{F'}{D}
\]

and

\[
F' = \frac{D' \times f}{d'}
\]

Where \(F'\) is the balls current distance from the camera. The next table shows the results for some tests with pictures taken at various distances.
<table>
<thead>
<tr>
<th>Actual distance</th>
<th>Estimated distance</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>250cm</td>
<td>225cm</td>
<td>-10.0%</td>
</tr>
<tr>
<td>500cm</td>
<td>469cm</td>
<td>-6.2%</td>
</tr>
<tr>
<td>1000cm</td>
<td>1001cm</td>
<td>0.1%</td>
</tr>
<tr>
<td>2000cm</td>
<td>1918cm</td>
<td>-4.1%</td>
</tr>
<tr>
<td>4000cm</td>
<td>5756cm</td>
<td>30.5%</td>
</tr>
</tbody>
</table>

When the ball is either very close or very far away the error gets large, in these cases we can choose to ignore the detection of the ball and hover the quadcopter instead.

The next table gives an approximation of processing times for the various parts of the algorithm on a modern CPU.

<table>
<thead>
<tr>
<th>Process</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binarization / Coarse matrix</td>
<td>0.23 ms</td>
</tr>
<tr>
<td>Product of matrices</td>
<td>0.20 ms</td>
</tr>
<tr>
<td>Center of mass</td>
<td>0.33 ms</td>
</tr>
<tr>
<td>Distance estimation</td>
<td>0.00 ms</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>0.76 ms</strong></td>
</tr>
</tbody>
</table>

Pictures tested have a resolution of 320x240. Using a larger resolution will increase computation time but decrease distance error.

## 4.3 Trajectory generation

We derive equations for trajectory generation from [4].

We want to find a function $q(t)$ that describes a trajectory from the current position $(x_0, y_0, z_0)$ to an end position $(x_f, y_f, z_f)$. Having found the position of the ball in 3-space $(x_b, y_b, z_b)$ previously we can find the end position, noting that we want to remain a distance $d$ from the ball.

$$
\begin{align*}
x_f &= x_b - x_0 \\
y_f &= y_b - y_0 \\
z_f &= z_b - z_0 - d
\end{align*}
$$

At time $t_0$ we suppose the function satisfies:

$$
\begin{align*}
q(t_0) &= q_0 \\
\dot{q}(t_0) &= \nu_0
\end{align*}
$$
And at time $t_f$ we wish to attain the values:

\[ q(t_f) = q_f \]
\[ \dot{q}(t_f) = \nu_f \]

When we want to generate a trajectory between these points and specify a start and end velocity, a cubic polynomial can give a smooth curve.

\[ q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 \]

And the velocity is then given by:

\[ \dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2 \]

Combining these two polynomials with the constraints imposed gives a linear system in four dimensions, in matrix form:

\[
\begin{bmatrix}
1 & t_0 & t_0^2 & t_0^3 \\
0 & 1 & 2t_0 & 3t_0^2 \\
1 & t_f & t_f^2 & t_f^3 \\
0 & 1 & 2t_f & 3t_f^2 \\
\end{bmatrix}
\begin{bmatrix}
a_0 \\
a_1 \\
a_2 \\
a_3 \\
\end{bmatrix}
= 
\begin{bmatrix}
q_0 \\

\nu_0 \\
q_f \\
\nu_f \\
\end{bmatrix}
\]

Solving this linear system will give a smooth trajectory. Numerical constraints can be found by heuristically determining the maximum velocity of the quadcopter.

The next figures show an example with start and end velocities set to zero, which is what we want when we consider a stable hovering quadcopter that moves towards the ball and stops when it is at the right position. The figure shows the position, velocity and acceleration respectively.

![Figure 4.9: Quadcopter trajectory](image)

(a) Position (m) vs. Time (sec)  
(b) Velocity (m/sec) vs. Time (sec)  
(c) Acceleration (m/sec^2) vs. Time (sec)

Figure 4.9: Quadcopter trajectory
4.4 Alternative approaches

Some other approaches were explored. These are discussed now.

4.4.1 Petalinux

Petalinux is a an Embedded Linux system targetting FPGA-based Systems on Chip. Until recently Petalinux, including the tools used to build, develop, test and deploy Embedded Linux on FPGAs, was freely available. Now only some pre-built images and cross-compilers are available.

The advantage of using Embedded Linux is that kernel drivers, such as for I2C communication, are already available as well as ordinary system calls used in C programming. Software can be developed on a PC and later deployed on the FPGA by cross-compiling it.

We tested this approach by cross-compiling a corner detection application and running it on the FPGA. Even for a very small image (30x20) the execution time exceeded a minute.

Although a cross-compiler is available, the kernel sources and configuration needed in order to develop kernel modules were not supplied, limiting the usability of this approach.

4.4.2 Microblaze

Another solution that still allows us to write software in C is by using the Microblaze softcore processor. The speed of this processor is still very much limited, much like using Linux, but it allows for offloading parts to custom IP cores, such as the I2C communication with the camera and the Atmega.

We use the same corner detection program as for the Linux test. After removing system calls and dynamic memory management, a test image is placed in memory. Again the execution time is very high (>10 minutes).

In order to reduce this we eliminated floating point operations and use fixed-point arithmetic instead. This vastly improved execution time but not enough to be feasible. Using compiler optimizations we again gained speedup but not enough for feasibility. The next table gives execution times for our experiment.
<table>
<thead>
<tr>
<th>Method</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>C code</td>
<td>&gt;10 minutes</td>
</tr>
<tr>
<td>Fixed-point arithmetic</td>
<td>18 seconds</td>
</tr>
<tr>
<td>Compiler optimizations</td>
<td>13 seconds</td>
</tr>
</tbody>
</table>

Approximating the number of instructions executed and the clockspeed of the processor and taking memory latency into account these results make sense. Therefore we abandon this approach.
Chapter 5

Local control

Since we cannot rely on the assembled quadcopter, testing of a possible algorithm for local control and also visual servoing has to be done using a mathematical model. In "Robotics, Vision and Control" by Peter Corke a Matlab toolbox together with the theory behind it and also some applications are provided. One of them is a quadcopter model controlled by various algorithms in different situations. We will consider these applications in order to study the local control that is made with it.

The Simulink model is presented which contains blocks like the quadcopter (model), a block that transforms the pitch, yaw, roll and thrust values into the 4 motor speeds and gains that adjust the parameters for the control. Also a special function is made that plots the quadcopter in a 3D space for the purpose of showing its movement.
Figure 5.1: Simulink model

The way pitch, yaw, roll and thrust are mixed to obtain the motors’ speeds are expressed in the next figure:

Figure 5.2: Control mixer
\[
\begin{bmatrix}
T \\
\Gamma
\end{bmatrix} =
\begin{bmatrix}
-b & -b & -b & -b \\
0 & -db & 0 & db \\
db & 0 & -db & 0 \\
k & -k & k & -k
\end{bmatrix}
\cdot
\begin{bmatrix}
\omega_1^2 \\
\omega_2^2 \\
\omega_3^2 \\
\omega_4^2
\end{bmatrix}
= A \cdot
\begin{bmatrix}
\omega_1^2 \\
\omega_2^2 \\
\omega_3^2 \\
\omega_4^2
\end{bmatrix}
\]

Where \( T \) is the total upward thrust and \( \Gamma \) is the applied torque. \( A \) can be inverted, in that sense being possible to obtain the motors’ required speed.

A proportional - derivative controller based on the error between desired and measured pitch angle is used to compute the pitch, yaw, roll and thrust values. Altitude is controlled in the red loop (PD controller) that determines the average motor speed. In the hardware implementation the inertial measurement unit gives different types of errors that have to be filtered in order to obtain the appropriate values.

Figure 5.3: Local control
Low-pass filtering eliminates the short-term fluctuations of the accelerometer output. Also the gyroscope (ITG3200) does not have a high-pass filter so we need to implemented it in software. The high-pass filter helps in taking out the drift made by the gyro (which accumulates errors in time, making the measured value different from the real one) [7].
Chapter 6

Conclusion

A design was made for a small quadcopter from scratch. Hardware was acquired and the design was implemented on a PCB.

We divided our design in global and local control and continued to write and test the software needed to control the quadcopter.

Unfortunately we were not able to develop all the modules necessary to have a flying quadcopter. The local control part could not be tested since the hardware construction was not finished.
Appendix A

Work distribution

The table below shows the tasks that have been worked on during the project and which person(s) worked on that task.

<table>
<thead>
<tr>
<th>Task description</th>
<th>Martijn</th>
<th>Vlad</th>
<th>Alexandru</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research on quadcopter dynamics</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Global system design</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Determining hardware</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Circuitry schematics</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Hardware construction</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>FPGA Linux experiment</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>FPGA Microblaze experiment</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>FPGA RS232 communication</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Atmega testing</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Motor PWM control</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>IMU readout</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Camera readout</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Ball detection</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Ball distance estimation</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Trajectory generation</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>PD control</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Report</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
Appendix B

Course organization

While determining what exactly our project should be, we made decisions that resulted in a broad scope of skills and knowledge: hardware design and assembly, low-level programming, hard- and software control and vision. Some of these skills (FPGA programming, hardware assembly) we had little or no experience in, making things very difficult for ourselves. Although interesting, it made it practically impossible for us to reach our goals. We therefore recommend that prior knowledge and experience (such as passed courses) be taken into account when setting the final goals for a group’s project.

For future course iterations we also recommend not to include any hardware construction. Ordering of parts can delay the project and assembling components into a working machine is far from trivial. Even those with some experience in this will spend a lot of time making a complete system, time that could be spend on aspects that are more on par with a university level master course.

Finally we recommend that the scope of the project should not be too broad. Both local and global control have enough challenges, which is proven by the fact no group has completely succeeded in either one.

As for our design, some simplifications could have resulted in reaching more of our goals. We designed our own motor control where ESC’s would have done the trick. We used an FPGA / Atmega setup for global and local control respectively where using a single board with a CPU (such as the Arduino) would have simplified our design, make better use of our experience and include more information (coding examples, user experiences, higher-level coding languages) while sufficing in performance aspects. We think this
would have relieved us of a major burden and allow us to focus on more interesting things (algorithms and control).

Also the fact that there are already some Simulink models and functions for move and vision, it should be more relevant to really focus on improving and analysing the algorithms already built in terms of performance and not re-developing them.

Despite not being able to finish a working quadcopter we have enjoyed the course and hope it will continue since we know many other students would enjoy it as well.
Bibliography

[1] Multicopter wiki’s multicopter table http://multicopter.org/wiki/Multicopter_Table


