Institutionen för systemteknik
Department of Electrical Engineering

Examensarbete

Development of a Wireless Video Transfer System for Remote Control of a Lightweight UAV

Examensarbete utfört i Elektroteknik
vid Tekniska högskolan vid Linköpings universitet
av

Thomas Axelsson, Joakim Tosteberg

LiTH-ISY-EX--YY/XXXX--SE

Linköping 2012

Linköpings universitet
TEKNISKA HÖGSKOLAN

Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings tekniska högskola
Linköpings universitet
581 83 Linköping

Development of a Wireless Video Transfer System for Remote Control of a Lightweight UAV

Examensarbete utfört i Elektroteknik
vid Tekniska högskolan i Linköping
av

Thomas Axelsson, Joakim Tosteberg

LiTH-ISY-EX--YY/XXXX--SE

Handledare: **Andreas Ehliar**
ISY, Linköpings universitet
Marcus Eliasson
Epsilon AB
Tobias Antonsson
Epsilon AB

Examinator: **Ola Dahl**
ISY, Linköpings universitet

Linköping, 1 June, 2012

Avdelning, Institution
Division, Department

Division of Computer Engineering
Department of Electrical Engineering
Linköpings universitet
SE-581 83 Linköping, Sweden

Datum
Date

2012-06-01

Språk
Language

☐ Svenska/Swedish
☒ Engelska/English

☐ ________________

Rapporttyp
Report category

☐ Licentiatavhandling
☒ Examensarbete
☐ C-uppsats
☐ D-uppsats
☐ Övrig rapport
☐ ________________

URL för elektronisk version

http://www.da.isy.liu.se
http://www.ep.liu.se

ISBN

—

ISRN

LiTH-ISY-EX--YY/XXXX--SE

Serietitel och serienummer
Title of series, numbering

ISSN

—

Titel
Title

Utveckling av ett trådlöst videoöverföringssystem för fjärrstyrning av en minimal obemannad luftfarkost

Development of a Wireless Video Transfer System for Remote Control of a Lightweight UAV

Författare
Author

Thomas Axelsson, Joakim Tosteberg

Sammanfattning
Abstract

Abstract goes here.

And it can consist of several paragraphs.

Nyckelord
Keywords

key1, key2

Abstract

Abstract goes here.

And it can consist of several paragraphs.

Sammanfattning

Svenskt abstract kan man placera här.

Acknowledgments

I would like to thank a lot of people…

Contents

1 Introduction **1**

1.1 Background 1

1.2 Purpose 1

1.3 Disposition 1

2 Original System **3**

2.1 System Overview 4

2.1.1 Quadcopter Radio 4

2.1.2 Pin Headers 5

2.1.3 USB Radio Dongle 7

2.1.4 Computer Application 7

2.2 Performance Calculations 8

2.2.1 Theoretical throughput of radio channel 8

2.2.2 Delay calculations 9

2.2.3 Packet loss 12

2.3 Measurements 13

2.3.1 Flight Time and Lift 13

2.3.2 Radio Data Rate 15

2.3.3 USB Measurements 17

2.3.4 Image transfer measurements 18

3 Video **21**

3.1 Important Factors 21

3.1.1 Physical Factors 22

3.1.2 Software Factors 23

3.2 Parrot AR.Drone Experiments 24

3.2.1 Testing and Results 25

3.3 Raw formats 26

3.4 Theoretical Requirements 26

3.4.1 Frame rate 26

3.4.2 Image file size 27

4 Cameras 31
4.1 Summary of Requirements . . . 31
4.2 General SOMETHING . . . 33
4.2.1 Sensor Types . . . 33
4.2.2 Image Sensor Hardware Interfaces . . . 33
4.3 Candidate Chips . . . 34
4.3.1 Image Sensors . . . 34
4.3.2 Image Signal Processors . . . 37
4.4 Requirements Follow-up . . . 39
4.5 Design Ideas . . . 39
4.5.1 Data Transferred Directly to Quadcopter . . . 39
4.5.2 Data Processed by Helper Chip . . . 40
4.6 vald kamera/lÃ¶sning/Chosen Solution? . . . 41

5 System Design 43
5.1 Hardware . . . 43
5.1.1 Selection . . . 43
5.2 Design . . . 44
5.3 Software . . . 45
5.3.1 Camera module . . . 46
5.3.2 CrazyFlie . . . 46
5.3.3 CrazyRadio . . . 46
5.3.4 Computer . . . 47

6 Results 49

7 Conclusions 51

8 Future Work 53

A Radio rate measurements 57

B AR.Drone Test Data 65
B.1 Resolution Tests . . . 65
B.2 Angle of View Tests . . . 65
B.3 Angle Calculations . . . 65

C Schematics 72

D Bill of Materials 81

E UML diagrams 83

List of Figures

2.1 The CrazyFlie quadcopter 3
2.2 Quadcopter wireless packet format 4
2.3 Pinout for the CrazyFlie JTAG and EXP connector 5
2.4 Pinout for the Cortex Debug connector 5
2.5 The CrazyRadio dongle 7
2.6 Python control application for CrazyFlie 8
2.7 Parallel timings 11
2.8 Serial timings 12
2.9 FPS vs retransmissions 13
2.10 Tested weight positions 14
2.11 ACK data rates 17
2.12 USB data rates 18
2.13 Image data rates 19

3.1 Parrot AR.Drone 24
3.2 Images from AR.Drone 25
3.3 Maximum image size 27
3.4 Required compression 28

4.1 Pin configurations of some common image sensor interfaces 34
4.2 Design in which data is transferred directly to the Quadcopter . . 39
4.3 Design in which data is processed by a helper chip 40

5.1 CrazyCam PCB 45
5.2 Soldered CrazyCam 45
5.3 The CrazyCam module 46

B.1 AR.Drone resolution test images 66
B.2 AR.Drone angle of view test images 68
B.3 Measurements used to calculate the angle of view 70

C.1 Schematics for camera module 73
C.2 Schematics for debug board 75
C.3 Top layer copper 77
C.4 Top layer silkscreen 78
C.5 Bottom layer copper 79
C.6 Bottom layer silkscreen 80

E.1 Activity diagram for computer side USB code 83
E.2 Activity diagram for CrazyRadio 84
E.3 Activity diagram for CrazyFlie 85
E.4 Activity diagram for camera module 86

List of Tables

2.1 Pin descriptions for the JTAG and EXP connectors 6
2.2 ESB cycle times 9
2.3 Effective ACK data rates 9
2.4 15

3.1 Table of resolutions 21

4.1 Image sensor candidates 36
4.2 ISP candidates 38

A.1 ACK data rate measurements at 2 Mbps 59
A.2 ACK data rate measurements at 1 Mbps 61
A.3 Image data rate measurements 63
A.4 Image data rate measurements through door 64
A.5 Image data rate measurements through door and wall 64

B.1 Drone front camera angle test data 71

D.1 Debug board bill of materials 81
D.2 Camera module bill of materials 82

Todo list

how much from each? too much info on quad? 4

Motor type and rotation speed? 4

Look at this 10

More data points. Time data does not say anything 14

insert more text when more data points have been collected 15

Insert plot 15

Area outside would be interesting. Same as for 90 degree angles?. Refer to the placement figure mentioned earlier 15

Measure height 16

Cleanup of and reference to used code here (appendix or something else?) . 16

Code references 19

longer text? :) 21

camera aim. we have some data in drone tests. refer to that? 22

Better heading? 23

Address each factor and provide a “solution”. Separate section? Put in different part of design? Some discussion in video_factors.txt 23

code/modifications in appendix? 25

nice pic that explains angle of view and cropping? at least figure that shows how to measure it. can show both sides of lens :) 25

move requirements summary to video? 31

current consumption? 31

explain 4 in header. is this from the non-center weight? update when new flight data acquired 32

put px/degree calculations in video factors and refernce here? 32

follow up requirements were applicable 33

comment on un-fulfilled requirements 34

include info on existing products? 808, bluetooth based camera, mostly larger cameras, kkmulticopter, liu copter? 34

list package dimensions instead of package type? 37

short discussion on interesting sensors/interesting features? 37

switch dated galaxycore camera to new model? it is important that it has a flexible flat cable. gc0309 datasheet: http://www.docin.com/p-240246436.html 37

explain what factors make these sensors interesting. give more info on the most interesting ones 37

open source is a goal 37
explain that lenses are often acquired separately. try to find angle values for sensors with lenses 39
mpeg, frame drop 39
section on possibility to fulfill matching requirements? 39
comment on cameras with radios 39
source. even possible? preferably find test on stm32 39
mipi. differental. too fast? 40
explain why a fifo chip was not chosen. maybe in the next chapter 40
list physical size in table. 40
Having all processing done in hardware - low power consumption. source. jpeg chips have small mcus, need to compare with regular mcu running jpeg? 41
software jpeg, e.g. http://www.diydrones.com/profiles/blog/list?user=JackCrossfire . uses stm32f4 41
more info 41
analog camera? discuss somewhere. adc-conversion. direct radio to separate receiver without using quad radio 41
describe mentioned sensors below sensor table 41
See if we can get images with same orientation and size 45
split hardware section into subsections? 45
Update code/UML/text to actually match each other 46
Some stuff here possibly belongs in a discussion section 49
Check values 49
not that bad.. from url in comment: software jpeg for a 32-bit mcu with dual issue 51
super-nice figure from notebook 65
My reasoning seems validated by http://www.panohelp.com/lensfov.html but maybe get a proper source. It does feel like common knowledge though :) 71

Glossary

CCI Camera Control Interface. 33

CPI Camera Parallel Interface. 34

CSI Camera Serial Interface. 34

DVP Digital Video Port. 34

ESB Enhanced ShockBurst. 4, 8, 11

FPS frames per second. 7, 9, 12, 24, 26, 29, 32, 35, 49, 51

ISP image signal processor. 34, 37, 41

MCU microcontroller. 3, 5, 9–11, 39–41, 43–46

MIPI Mobile Industry Processor Interface. 33, 34

PCB printed circuit board. 14, 31, 33, 44, 45

SCCB Serial Camera Control Bus. 33

Chapter 1

Introduction

1.1 Background

Epsilon AB has, through an internal competency development program, developed a small quadcopter, which can be controlled wirelessly by a user, using a computer. They now want to extend it with functionality to display an image stream from the perspective of the quadcopter to the user, which should be possible to use to control the quadcopter also when the user doesn't have visual contact with it.

The problem to be solved is then a combination of a system design and a system integration problem where a system for capturing and transfering images shall be designed and then integrated with the existing quadcopter system. This to enable that the quadcopter can be controlled by an operator with the images acquired from the quadcopter as the only source of information.

There exists similar system today for larger UAVs [5, 8]. What is new with this project is the small size and limited resources of the target system with which the camera system is to be integrated.

1.2 Purpose

The purpose of this master thesis project is to first evaluate what requirements must be fulfilled for a camera system that is to be mounted on a lightweight quadcopter for the use of remote control of such a system and then to design a prototype of such a system and evaluate its real world performance.

1.3 Disposition

The report begins with an overview of the current CrazyFlie system followed by a theoretical analysis of its capabilities. The analysis is then followed up by a section on measurements done to verify and supplement the theoretical calculations.

After that follows a short study of the necessary quality and performance requirements for the image stream from the CrazyFlie to the computer, the theoretical material is then complemented by some practical tests using a Parrot AR.Drone. After this is a short overview on the different raw image formats that is commonly used as input for JPEG compression. This is then followed up with calculations on what will be possible to achieve given the constraints that was found in the calculations on the existing CrazyFlie system.

The report then goes on evaluating different camera systems for use on the CrazyFlie given the limitations and requirements found earlier. After the comparison is finished a camera system is selected as the one to use.

With a camera system selected, the report then describes how the necessary hardware and software for integrating the camera system with the CrazyFlie system is designed and then finally evaluates how well the camera system is working.

Chapter 2

Original System

The CrazyFlie quadcopter is shown in figure 2.1. It weighs about 20 grams with battery, measures 3.8cm along the side of the base and 9.8 cm between opposing engines.

The quadcopter is controlled wirelessly from a nearby computer, via a radio on a USB dongle. The pilot uses a gamepad to control the quadcopter. Accelerometers and gyros in three dimensions allows the quadcopter to determine its orientation and keep itself from overturning.

Power is delivered from a 170 mAh battery when the power button is pressed. The quadcopter is also on, and charging, when connected by USB to the computer. When on, the blue LED is lit and the red LED indicates data traveling through the wireless link. The green LED indicates the calibration status, when blinking slowly the quadcopter has not been calibrated its sensors and will not lift. The sensors will be calibrated when then quadcopter is put on a flat surface and the green LED will start to blink faster.

Add-ons can be connected to the quadcopter through the use of the expansion pin header on the top side.

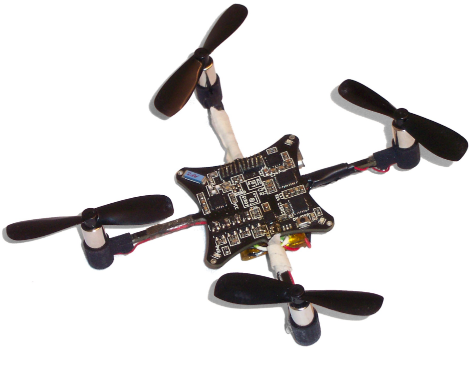


Figure 2.1: The CrazyFlie quadcopter

Preamble 1 byte	Address 3-5 bytes	Packet Control Field 9 bit	Custom Protocol Header 1 byte	Payload 0-31 bytes	CRC 1-2 bytes

Figure 2.2: Quadcopter wireless packet format

2.1 System Overview

The heart of the CrazyFlie is the STM32F103CB [35] Cortex-M3 microcontroller (MCU), running FreeRTOS. It uses several tasks to control the motors, communicate via the radio and read the sensors. Energy comes from a Li-Po battery with 3.7 V nominal voltage. Power consumption with the motors off is about 250 mW and about 5 W when the quadcopter is hovering. The motors are supplied with the battery voltage directly after the charging chip, while the rest of the system runs on two regulated 2.8 V power rails. Each of the 2.8 V regulators is capable of supplying 150 mA[18] of which 70 mA are used in total.

how much from each? too much info on quad?

Motor type and rotation speed?

2.1.1 Quadcopter Radio

Nordic nRF24L01 [3] is the radio chip mounted on the CrazyFlie. It communicates with the microcontroller using SPI.

Wireless communication with the USB dongle is performed at 2.4 GHz using a custom protocol on top of the Nordic Enhanced ShockBurst (ESB) protocol. ESB is a master-slave protocol which supports error detection and retransmission through the use of ACK and CRC checks. The transmission channel is half-duplex, which means that the master and the slave has to alternate between sending and receiving. Only the master can initiate a transfer and data from the slave is added as payload to ACK packets sent back to the master. The maximum payload is 32 bytes per packet in both directions and only one packet is in the air at the same time. The quadcopter is the slave and the USB dongle the master, in the current system. [3]

The custom quadcopter protocol uses one byte of the payload for its header, which contain priority and target module data, effectively giving a maximum packet payload of 31 bytes as seen in figure 2.2.

The radio can communicate with another radio at 250 kbps to 2 Mbps and the maximum output power is 0 dBm (1 mW). Note that the rate is the rate of whole packets and that the rate of the payload will be lower (refer to section 2.2.1). The radio is set to communicate with the USB dongle at 2 Mbps. Power consumption is maximally 38 mW when transmitting or receiving and about 1mW in standby. [3]

The radio chip is connected to an omni-directional ANT-2.4-CHP SMD-format antenna with 0.5 dBi maximum gain [36].

JTAG	VCC	1	2	TMS
	I2C_SCL	3	4	TCK
	I2C_SDA	5	6	TDO
	NC	7	8	TDI
	DGND	9	10	$\overline{\text{SNRST}}$
EXP	SPI1_MOSI	11	12	SPI2_MOSI
	SPI2_SCK	13	14	SPI2_MISO
	VCC	15	16	VCOM
	DGND	17	18	DGND
	AGND	19	20	VCCA

Figure 2.3: Pinout for the CrazyFlie JTAG and EXP connector

VCC	1	2	SWDIO/TMS
GND	3	4	SWDCLK/TCK
GND	5	6	SWO/TDO
KEY	7	8	NC/TDI
GNDDetect	9	10	$\overline{\text{RESET}}$

Figure 2.4: Pinout for the Cortex Debug connector[12]

2.1.2 Pin Headers

The CrazyFlie is equipped with two 2×5 pin header connectors, called EXP and JTAG respectively. JTAG is designed from the Cortex debug connector pinout[12] and is used for programming and debugging the microcontroller, while EXP allows for connection of add-ons. Both connectors share the same numbering. Pin descriptions can be found in table 2.1 and the connectors are shown in figure 2.3 and 2.4.

The EXP header contains 3 pins connected to an SPI bus shared with the radio. These pins must therefore be used as an SPI bus. The last MCU I/O pin on the EXP header can be used without limitations, for example as a chip select for the SPI bus. The rest of the pins provide access to all the supplies on the board.

The JTAG header is mainly intended for JTAG programming/debugging, however it should be possible to use most pins for general I/O if designed carefully. Looking at figure 2.3 and 2.4 one can see that the I2C connections correspond to the ground pins on the Cortex debug connector and since I2C nodes only set pins low, there is no electrical conflict. USART usage may however render shorts with

Header	Pin	Connected to	Description
JTAG	1	VCC	Digital supply (2.8 V)
	2	PA13/TMS	MCU GPIO/JTAG
	3	PB10/I2C_SCL/TX	MCU GPIO/I2C/USART
	4	PA14/TCK	MCU GPIO/JTAG
	5	PB11/I2C_SDA/RX	MCU GPIO/I2C/USART
	6	PB3/TDO	MCU GPIO/JTAG
	7	NC	Not connected
	8	PA15/TDI	MCU GPIO/JTAG
	9	DGND	Digital ground
	10	$\overline{\text{SNRST}}$	MCU reset
EXP	11	PA7/SPI1_MOSI	MCU GPIO/SPI1
	12	PB15/SPI2_MOSI	SPI2[1]
	13	PB13/SPI2_SCK	SPI2[1]
	14	PB14/SPI2_MISO	SPI2[1]
	15	VCC	Digital supply (2.8 V)
	16	VCOM	Common supply (5.5 V)
	17	DGND	Digital ground
	18	DGND	Digital ground
	19	AGND	Analog ground

[1] SPI bus shared with radio chip

Table 2.1: Pin descriptions for the JTAG and EXP connectors. Pins are either connected to the supply voltages or pins on the microcontroller.

the JTAG ground if the pins are set high.

2.1.3 USB Radio Dongle

The radio dongle is based on the nRF24LU1+[4] chip, which combines an nRF24L01+ (very similar to the one used on the CrazyFlie, see 2.1.1), an 8051 microcontroller with full speed USB 2.0 capability and a 16 KiB flash memory [4]. Figure 2.5 shows the USB dongle.

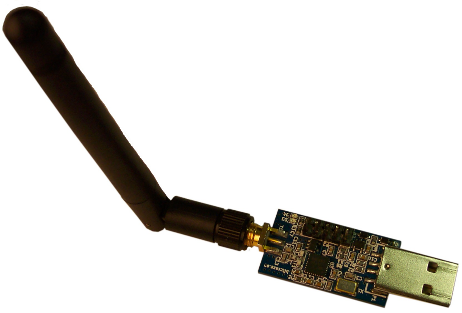


Figure 2.5: The CrazyRadio dongle

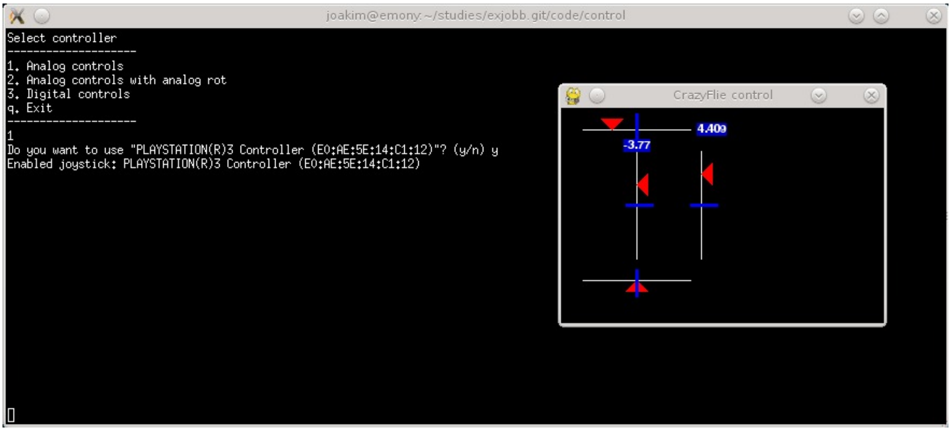
The dongle acts as an USB device and relays packets between a computer (the USB host) and the quadcopter.

When using USB, the device establishes a number of endpoints, which can be seen as communication channels. Each endpoint support transfer of data in one direction and can be optimized for different kind of data, for example control messages or bulk data transfers. [6]

USB 2.0 full-speed supports a transfer rate, including USB headers, of up to 12 Mbps, half-duplex. Data is sent and received in transfers, which are split into several packets. Each kind of transfer gets a certain amount (sometimes guaranteed) of, what is called, a frame. These frames are 1 ms each, that is, 1000 frames per second (FPS) are communicated. The host acts as a master and initiates all transfers. [6]

2.1.4 Computer Application

A Python script translates inputs from the gamepad and sends them to the CrazyFlie via the USB dongle. The script also shows the values of the different inputs, as shown in figure 2.6. The script runs on both Microsoft Windows and various Linux distributions.



```
Select controller
--------------------
1. Analog controls
2. Analog controls with analog rot
3. Digital controls
q. Exit
--------------------
1
Do you want to use "PLAYSTATION(R)3 Controller (E0:AE:5E:14:C1:12)"? (y/n) y
Enabled joystick: PLAYSTATION(R)3 Controller (E0:AE:5E:14:C1:12)
```

Figure 2.6: Python control application for CrazyFlie

2.2 Performance Calculations

2.2.1 Theoretical throughput of radio channel

As the computer acts as master in the radio communication it is the bandwidth for the ACK packets that is important for the transfer of image data. To maximize the bandwidth all ACK packets will carry their maximum payload of 32 bytes.

The time T_{ESB} for an ESB cycle can be expressed according to equation 2.3. [3] Two different border scenarios will be considered, one where all headers and the data payload is of maximum size, and one scenario where the header is of minimum size and there is one byte of data payload. This gives the parameters in 2.1 and 2.2 respectively. [3] For each of these scenarios all three speeds that the radio can run at are considered.

$$\begin{cases} \text{data packet length} = 329 \text{ bits} \\ \text{ACK packet length} = 329 \text{ bits} \end{cases} \tag{2.1}$$

$$\begin{cases} \text{data packet length} = 57 \text{ bits} \\ \text{ACK packet length} = 297 \text{ bits} \end{cases} \tag{2.2}$$

Furthermore some timing constants is defined directly in [3] and here specified in equation 2.4.

$$\begin{cases} T_{ESB} = T_{UL} + 2 \cdot T_{stby2a} + T_{OA} + T_{ACK} + T_{IRQ} \\ T_{OA} = \frac{\text{data packet length}}{\text{data rate}} \\ T_{ACK} = \frac{\text{ack packet length}}{\text{data rate}} \\ T_{UL} = \frac{\text{payload length}}{\text{SPI data rate}} \end{cases} \tag{2.3}$$

$$\begin{cases} T_{stby2a} = 130\ \mu\text{s} \\ T_{IRQ} = 6\ \mu\text{s} \end{cases} \tag{2.4}$$

	Radio data rate		
	250 kbps	1 Mbps	2 Mbps
Worst case	2830.32 µs	946.77 µs	634.01 µs
Best case	1648.62 µs	612.93 µs	440.31 µs

Table 2.2: ESB cycle times

	Radio data rate		
	250 kbps	1 Mbps	2 Mbps
Worst case	85.57 kbps	255.53 kpbs	381.99 kpbs
Best case	146.90 kbps	395.13 kbps	550.04 kpbs

Table 2.3: Effective ACK data rates

From this can then first the total time of an ESB cycle be calculated, and the results is summarized in table 2.2. The effective ACK transfer rate is then calculated using equation 2.5. The effective payload is one byte less than the real payload due to one byte being overhead of the protocol used in the CrazyFlie. The effective ACK data rates is summarized in table 2.3.

$$r_{ACK} = \frac{\text{effective payload}}{T_{esb}} \tag{2.5}$$

2.2.2 Delay calculations

As the images is to be used for remote control of the quadcopter it is important that there isn't too much delay from the point where the image is taken until it can be displayed to the user. The delay that can be directly calculated is the delay from when the readout of the image from the camera can be started until the point when it has been successfully transferred to the computer.

It will be assumed that the camera module outputs compressed JPEG data either over a parallel interface where the camera module acts as master and dictates data rate and timings. Or over a serial SPI/UART interface where instead the MCU is master and has control over timings and data rate. Depending on this two slightly different situation arises, with a serial interface the module can interface directly with the MCU that is present on the quadcopter while with the parallel interface a second MCU will be necessary, both due to pin constraints and timing problems so that the primary MCU isn't held up receiving data at critical moments.

In both cases some variables are in common, and those are defined in 2.6. The value used for r_{ACK} was chosen according to the results for 2 Mbps link speed in section 2.3.2 for 16 bytes of data payload with some extra margin. The average size s of the transferred image was chosen according to the calculations in section 3.4.2 as the maximum allowed size to achieve 15 FPS with 200 kbps average link speed.

These selections ensures that there are ample of margin in the calculations so that if it works for this case the real world values could be worse without that being a problem.

$$\begin{cases} r_{SPI} = 4.5 \text{ Mbps} - \text{SPI rate used in the quadcopter} \\ r_{ACK} = 300 \text{ kbps} - \text{Effective ACK bandwidth of radio channel} \\ s = 1.67 \text{ KiB} - \text{The average size of an compressed image frame} \\ n_p - \text{Number of radio packets required in average for a frame} \\ t_d - \text{Total delay for a frame from start of readout until last packet has been received} \end{cases} \tag{2.6}$$

Delay will be calculated from the point when the first bit of data for a frame is available for transmission to the MCU until the last bit has been received and acknowledged over the radio channel. This is as the delay up to this point is highly dependent on camera module solution. Delay for processing and rendering the image on the computer at receiving side is also ignored.

Camera module connected to separate MCU over parallel bus

A setup where a camera module is connected to an separate MCU over a parallel DVP interface. The MCU is then connected as a secondary master to the same SPI bus as the radio, and is allowed write access by the main MCU over some separate communication channel.

Delay from the point where the image taken until the first data is available over the parallel bus is not included in these calculations.

As the parallel interface outputs a frame at a continuous rate the MCU should have space to buffer an entire frame, 3 KiB of memory available for this is more than enough.

All variables used in the calculations is described in 2.7 and 2.6. Initial calculations will be pessimistic assuming full buffering of a frame and no pipe-lining.

$$\begin{cases} f_{pclk} - 36 \text{ MHz} - \text{Pixel clock frequency} \\ t_{cc} - \text{Transfer time for a frame from camera to MCU} \\ t_{ecc} - \text{Effective transfer time for a frame from camera to MCU} \end{cases} \tag{2.7}$$

With a 8-bit interface 1 byte is transferred from the camera module to the MCU each cycle of the pixel clock, and the time needed to transfer the entire image is given by equation 2.8. This does assume that the JPEG data comes at a continuous stream, which isn't the case in reality. No data is currently available about the real transfer time, so for now it is assumed that only 1/10 of the time is spent doing actual transfer

Look at this

. The total transfer time is then given by equation 2.9.

$$t_{cc} = \frac{s}{f_{pclk}} = \frac{1.67 \text{ KiB}}{36MHz} \approx 47.5 \text{ µs} \tag{2.8}$$

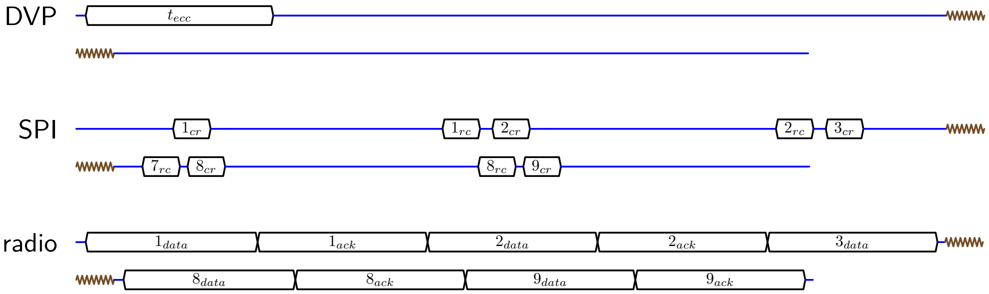


Figure 2.7: Parallel timings

$$t_{ecc} = t_{cc} \cdot 10 \approx 61.85 \cdot 10 = 475 \text{ µs} \tag{2.9}$$

$$t_{SPI} = \frac{\text{data size}}{r_{SPI}} \tag{2.10}$$

From equation 2.10 the time needed to transfer 32 bytes of data over the SPI-bus is calculated as 54.25 µs.

Now enough data to derive the timing diagram in figure 2.7 is available. The timing diagram shows that the transfer from the camera module to the MCU can done fully in parallel with the radio transmissions, so it's

Using equation 2.11 it is found that the number of packets, n_p, required to transmit a frame is 430. From this does equation 2.12 give the delay, t_d, as 43.4 ms.

$$n_p = \frac{1.67 \text{ KiB}}{31 \text{ bit}} \approx 430 \text{ pcs} \tag{2.11}$$

$$t_d = \frac{31 \text{ bit}}{300 \text{ kbps}} \cdot 430 \approx 43.4 \text{ ms} \tag{2.12}$$

Camera module connected to main MCU over serial bus

A setup where a camera module is directly connected to the main MCU over a serial bus. Both SPI and UART is considered, and as in the case of SPI the bus is shared with the radio the bus usage will be calculated to ensure that there is enough margin remaining.

The time to transfer 32 bytes over the SPI-bus was earlier derived from equation 2.10 and found to be 54.25 µs. The time for the data and ACK parts of the ESB cycle can be calculated according to equation 2.13 [3], where maximum header and payload sizes is used. This gives about 290 µs for each part. From this the timing diagram in figure 2.8 is derived.

$$\begin{cases} \text{data time} = T_{OA} + T_{stby2a} \approx 286.88 \text{ µs} \\ \text{ack time} = T_{ACK} + T_{stby2a} + T_{IRQ} \approx 292.88 \text{ µs} \end{cases} \tag{2.13}$$

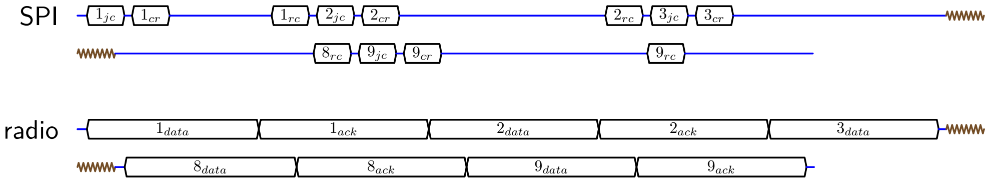


Figure 2.8: Serial timings

As seen in figure 2.8 all SPI transfers on the MCU side will be done in parallel with data being transmitted over the radio. So the delay will be identical to the parallel case, that is 44.3 ms.

Using USART instead of SPI to communicate with camera module wouldn't affect delay as long as it is fast enough so that the transfer is finished so that the SPI bus has time to transfer the data to the radio before it is needed. To accomplish this it has a maximum of $634.01 - 54.25 = 579.76$ µs to transfer 248 bits. So the required effective speed is $\frac{248 \text{ bit}}{579.76 \text{ µs}} = 427.76$ kbps, corresponding to an UART speed, in 8n1 mode, of 534.7 kbps. So the UART would need a baudrate of at least 534700 baud to work under optimal conditions where date can be continuously transferred without any extra delay, in reality extra margins would be needed so at least 1 Mbaud would probably be necessary.

Delay for USB transfer

Some extra delay is also added by the transfer from the radio dongle to the host computer over USB. In section 2.3.3 it was found that at least three 32 byte packets could be sent every millisecond. Furthermore it was found in section 2.2.1 and summarized in table 2.2 that the minimum time for an ESB cycle is 440 µs, that is at maximum about 2.3 packets will arrive every millisecond. So no extra buffering must be done and the extra delay added by the USB transfer will be less than a millisecond.

2.2.3 Packet loss

All above calculations has assumed that no packet loss will occur, in reality this is not the case, so here the effects of packet loss will be studied. Calculations is done on a 2 Mbps link. According to the datasheet with 32 bytes of payload is the minimum time to wait before retransmission, in this case 500 µs if no ACK has been received. Adding to this the TX time does each lost packet add 656.88 µs to the total time.

Without packet loss the total time to transmit a packet was 44.3 ms, which equals 22.57 FPS. Factoring in that n retransmissions happens per frame the new frame rate is given by equation 2.14. The result is plotted in graph 2.9.

$$FPS = \frac{1}{44.3 \text{ ms} + n \cdot 0.65688 \text{ ms}} \tag{2.14}$$

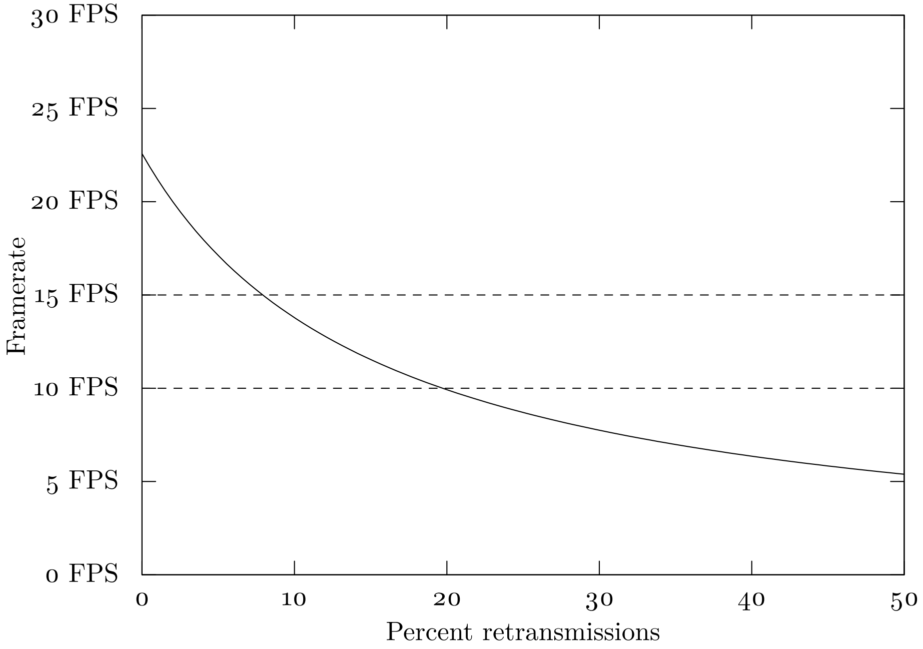


Figure 2.9: FPS vs retransmissions

2.3 Measurements

To find out the capabilities of the CrazyFlie, several tests and measurements were performed. Firstly, mounting a camera on the quadcopter will effectively increase its weight, which in turn may decrease its maneuverability and battery life. Secondly, the radio link will probably not perform as good as indicated by the theoretical calculations. Physical obstructions, external noise sources and the distance between the radio dongle and the quadcopter will affect the radio link performance.

2.3.1 Flight Time and Lift

A simple test scheme was devised to find out how much weight the quadcopter could lift and for how long it could stay airborne loaded with different weights.

Test Setup and Execution

The quadcopter was charged for at least 30 minutes before each test to make sure that the battery was fully charged. All tests were performed indoors at the same location, with no obstacles. There were 10 to 15 wireless networks covering the area. In each test, a weight was attached to the quadcopter using a rubber band. The rubber band was left in place when testing the lift without any added weight.

At the start of a test, the quadcopter was disconnected from the USB cable used for charging, and placed on the floor. Thereafter it was turned on and the

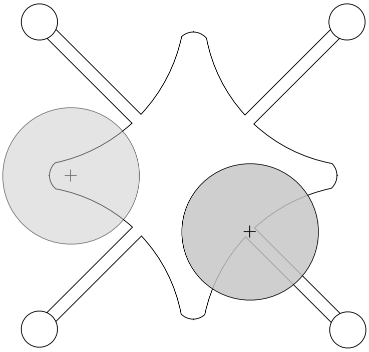


Figure 2.10: Illustration of the different weight positions tested (cross indicates center of gravity)

computer control application was started. The pilot then kept the quadcopter in the air, as much as possible, during the rest of the test. The test was terminated when the quadcopter no longer reacted to the commands from the pilot, due to battery drain.

Extra lift force needed was measured by slowly increasing the thrust, using the control application, until the quadcopter overtook the force of gravity and moved continuously upwards.

Any extra weight was placed under and in the center of the quadcopter printed circuit board (PCB) and the full test performed. The value of the weights were measured 3 times each on a letter-scale, with 1 grams resolution, and the average value used.

Tests with unbalanced extra weight were also performed. The 7.5 grams weight was placed at 45 degree increments offseted from a line through to opposing motor arms. This is indicated by figure 2.10.

Results

More data points. Time data does not say anything

The CrazyFlie was able to take off in all tests except with the 20 grams extra weight. The pilot felt that handling started to suffer considerably when using the 7.5 grams weight. The measured data is shown in table 2.4.

The time from turning the quadcopter on to losing control is listed as battery time. Flyable time begins at the same time but ends when the quadcopter cannot generate enough lifting force to keep itself in the air (this force is less than the force required for taking off). No times are listed when the quadcopter did not take off or when the test could not be completed due to damages.

Test number	Added weight [g]	Takes off	Battery time	Flyable time	Lift	
0	20	no	–	–	–	
1	4	yes	–	–	−0.31	
2	0	yes	11:06	11:06	−0.27	
3	7.5	yes	9:40	5:20	−0.35	
4	4	yes	7:36	7:36	–	
5	4	yes	8:12	8:12	–	
6	7.5	yes	8:00	4:48	–	
7	2	yes	9:01	9:01	−0.29	
8	2	yes	6:20	6:20	–	Battery proble
9	2	yes	6:38	6:38	–	
10	0	yes	7:23	7:23	–	
11	6	yes	8:16	6:50	−0.37	Changed to a
12	4	yes	–	–	−0.34	

Table 2.4

Thrust indicates the value given by the control application and is thus mostly of interest for comparing the different measurements against each other.

insert more text when more data points have been collected

Insert plot

Different positions of the 7.5 grams weight affected the quadcopter's ability to stay flat and its maneuverability. The quadcopter performed well as long as the weight was placed on a line along one of the motor axes with its center of gravity inside the area of the quadcopter's main body. Moving the weight such that the center of gravity ended up outside the base made the quadcopter tilt and drift in the direction of the weight.

Placing the 7.5 grams weight in-between to motor axes, the quadcopter was not able to lift when the weight's center of gravity was at a corner of the base. Not until moving the weight, such that it was only 1/4 of its diameter outside the base, was the quadcopter able to take off, albeit with a large drift in the direction of the weight.

Area outside would be interesting. Same as for 90 degree angles?. Refer to the placement figure mentioned earlier

2.3.2 Radio Data Rate

To complement the theoretical calculations of the achievable data rate for the ACK packets over the radio channel several practical measurements were done under different conditions.

Test Setup

The code for the CrazyRadio and the CrazyFlie was modified to keep the TX-FIFO:s in their respective radio chip full all the time.

For the quadcopter this was done by adding a new task which just added packets to an existing queue in the CRTP layer. The code in the radio layer was also changed to not disable the radio during the readout of packets as this was found to be quite deteriorating on the radio performance.

In the radio dongle it was just added a loop which continuously polled the radio for events and at each event ensured that the TX-FIFO was filled. The code in the radio dongle was further more modified to count the number of packets successfully transmitted and the number of packets which reached the maximum number of retransmissions. When 1 Mibit of data have been received from the quadcopter the value of a timer is read out and send to the host over USB together with the packet counts, and the timers and counter is then reset and the process restarted.

The initial plan was also to measure the number of retransmissions, but as this counter was automatically reset at the start of each new transmission it would have had to be read between one transmission was completed successfully and the next stared. And as the TX-FIFO is kept full at all times the next transmission is started automatically when the previous one is finished, leaving no time for readout of the retransmission counter.

The Auto Retransmit Delay (ARD) is set to 500 µs for link speeds of 1 Mbps and 2 Mbps is the minimum values required to ensure proper operation with 32 byte ACKs.[3] The maximum number of retransmissions for a packet is set to three for all cases.

The data rates is measured with data payloads of 1, 16 and 32 bytes for 2 Mbps and 1 Mbps, initially the plan was to measure also for 250 kbps but it turned out that mistakenly a nRF24L01 which doesn't support 250 kbps link speeds [3] instead of a nRF24L01+ had been mounted on the quadcopter. The ACK payload is always set to 32 bytes. For the calculations a value of 31 bytes is used as the data transferred with each ACK to compensate for that 1 byte is occupied by the CRTP header in the real communication. All tests is done at several distances first at about 0.15 meters from the radio dongle, then at 5 meters and from there at 5 meters steps up to 15 meters. The tests were executed in an open indoor room and there are several wireless LAN:s active in the building. Both the computer and the quadcopter is placed on top of XXX meter high chairs.

Measure height

.

Cleanup of and reference to used code here (appendix or something else?)

Results

Some attempts to count the number of retransmission by setting the maximum number of retransmissions per packets to 0 was made, under these conditions not a single packet was delivered successfully, even when increasing the ARD.

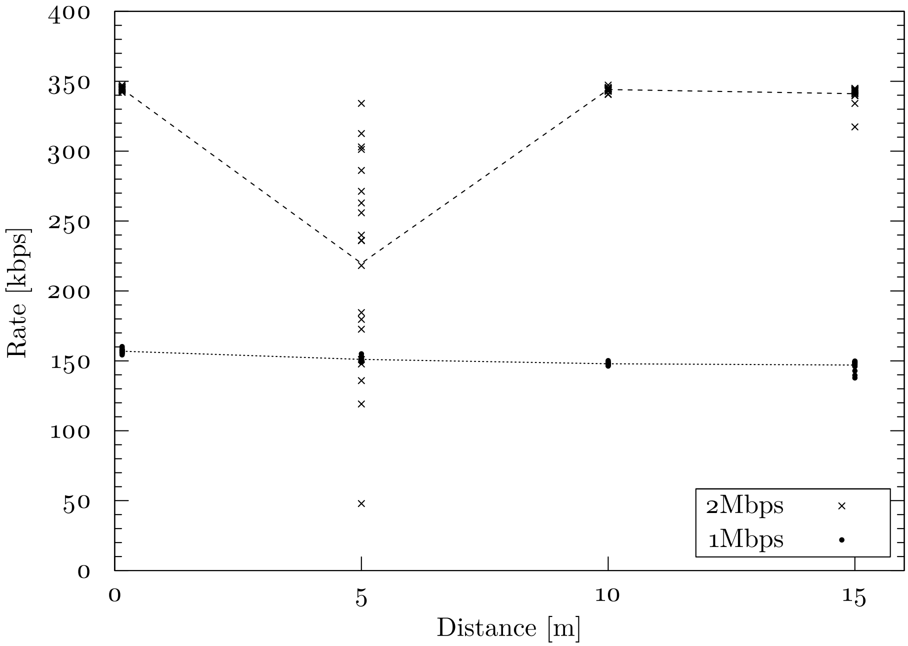


Figure 2.11: ACK data rates

The effective ACK data rates with 16 bytes of payload in the data packets with the CrazyFlie at different distances from the transceiver is plotted in figure 2.11. For both link speed the ACK data rate was relatively stable and only slightly affected by increased distance. The exception being the test at 5 meter distance for 2 Mbps link speed where large fluctuations in speed was present, possibly due to some kind of signal interference.

Varying the data payload at 2 Mbps link speed had the expected result, except at 5 meter distance where the fluctuations is to large to give reliable values, of that and increased payload decreased the ACK data rate and vice versa. At 1 Mbps link speed the behavior was not as expected, here an increase in the data payload actually increased the ACK data rate.

The raw data from the measurements is available in table A.1 and A.2.

2.3.3 USB Measurements

To verify that the radio dongle will be able to deliver enough performance over the USB channel some simple measurements was done.

Test setup

The CrazyRadio is configured with EP2 used for both IN and OUT transaction, and to maximize performance EP2 is paired with EP3 to achieve double buffering.

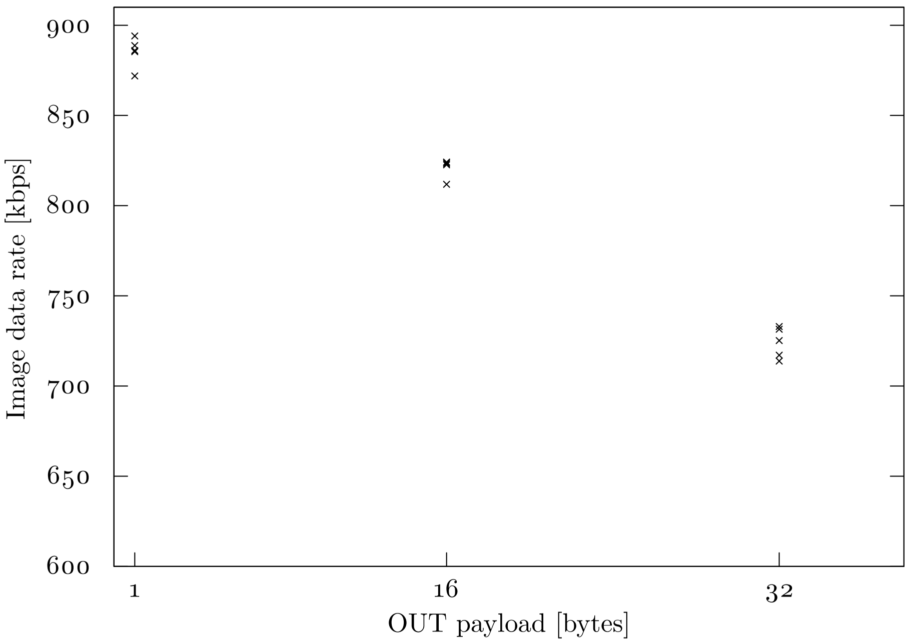


Figure 2.12: USB data rates

The dongle runs a loop which writes 33 bytes of data, which simulates sending a full radio packet + current radio status, to the the EP2 IN buffer if there is any space available there and just clears the EP2 OUT buffers if any data appears in them. The radio parts of the dongle is not utilized at all during these test.

On the client side a loop first writes dummy data to EP2 OUT and then reads data from EP2 IN. Both operations is done with the minimum timeout of 1ms supported by PyUSB 0.4.2. The test is performed with dummy data sized of 1, 16 and 32 bytes.

Results

The results of the measurements is plotted in figure 2.12. As seen the rate stays above 700kbps for all payloads in the test, and as it in section 2.3.2 was found that the peak radio rate was about 350kbps this speed is more than enough to not be the limiting factor.

2.3.4 Image transfer measurements

After verifying that the individual links was able to deliver enough performance to satisfy the requirements some measurements was also done on the entire chain, transferring data from the CrazyFlie through the radio dongle to the computer, and at the same time allowing control data to go through from the computer to the quadcopter.

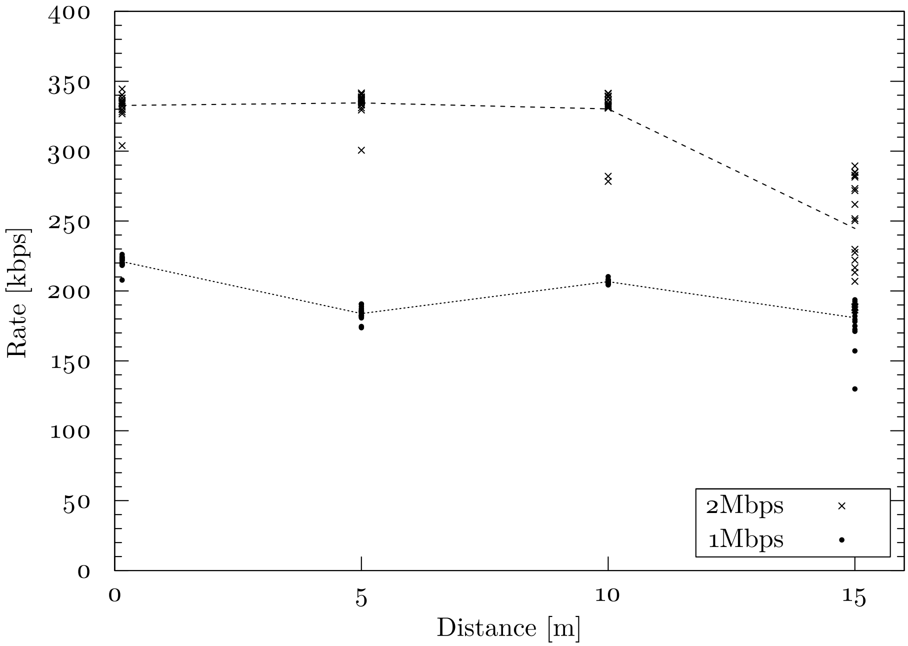


Figure 2.13: Image data rates

Test setup

For the quadcopter the same code as in the radio rate tests in section 2.3.2 is used, the code in the main loop of the radio dongle was rewritten to transmit 1 byte dummy packets to the quadcopter when there is nothing else to transmit. This to allow image data to be sent on ACKs from the quadcopter to the radio dongle as often as possible.

The physical setup and execution of the test is the same as in section 2.3.2. Also some extra tests to see the effects of obstacles is executed.

Code references

Results

The results for the tests at different ranges is summarized in figure 2.13, the raw results is available in table A.3. Up to 10 meters the results was stable for both link speeds, but at 15 meters the bandwidth started to drop and fluctuate more, especially at 2 Mpbs.

For 1 Mbps the results from this test does actually outperform the results from section 2.3.2 which is strange as these tests adds complexity by also transferring the data from the dongle to the computer over USB. For 2 Mbps the results are slightly lower than in section 2.3.2.

The results when putting obstacles between the quadcopter and the radio dongle is available in table A.4 and A.5. For both speeds the effect of having a closed door with a window in it between did not affect the results very much, but when also moving the quadcopter so that both the door and a wall was in between the performance at 2 Mbps dropped sharply while only a small drop was present at 1 Mbps

Chapter 3

Video

This chapter discusses different aspects and requirements on acquiring the video and transferring it from the CrazyFlie to the client application. Please refer to table 3.1 for image resolutions used throughout the text.

3.1 Important Factors

There are several factors that will be important when controlling the CrazyFlie using video as visual feedback. The following text describes each of them briefly. Much of the information relates to remote controlled cars, not copters, but the information should still have some relevance. After listing the factors, some experimental tests of the Parrot AR.Drone follows.

longer text? :)

Abbrevation	Dimensions [pixels]	Megapixels
VGA	640×480	0.31
QVGA	320×240	0.08
QQVGA	160×120	0.02
QQQVGA	80×60	0.00
QQQQVGA	40×30	0.00
CIF	352×288	0.10
QCIF	176×144	0.03

Table 3.1: Resolutions used in the report together with their correspondence to megapixels

3.1.1 Physical Factors

Angle of View

What determines how much of the surroundings that the driver sees is the angle of view. For a human, the angles are more than 140° horizontally and 60° vertically [14, 1]. This translates to more than 150° diagonally. A typical wide angle lens (35 mm camera with 50 mm lens) captures 90° horizontally [1]. In tests, with the subjects remote controlling a car using video feedback, it was found that a diagonal angle of 50° is required while 100° is preferable [14]. This can hopefully be roughly translated to a flying vehicle, although a bit larger angle could be required due to the possibility to move vertically.

Magnification

A magnification factor of less than 1.0 (zoomed out) may give the driver a better view, but he will perceive movements as smaller than they are and therefore have a harder time to control the vehicle. A magnification factor greater than 1.0 (zoomed in) may also confuse the driver, because the relation between camera rotation and image translation is different as compared to a magnification factor of 1.0. [14]

Camera Placement

camera aim. we have some data in drone tests. refer to that?

Placing the camera such that part of the vehicle is visible increases the driving performance and the situational awareness of the driver. It could also be mounted such that it can move and provide variable viewing direction, giving the driver a larger angle of view. However, this forces the driver to keep track of the camera versus vehicle direction without any body feedback (such as turning the head to look sideways). This could partly remedied by direction indicators or by controlling the camera direction using head-movement. [14]

Monoscopic/Stereoscopic View

Two cameras can be mounted side-by-side to provide stereoscopic (3D) vision and it was found that this increase the ability, for a person driving a car, to more easily avoid obstacles and ditches within 10 meters. For driving a car on flat or paved terrain it is however sufficient with a monoscopic (one camera) view. [14]

Vibration

A slow shutter speed will lead to wavy pictures due to vibration. Vibrations from the rotors are larger in helicopters than airplanes, due to their larger main rotor [15]. However, the quadcopter has small, albeit fast, moving rotors.

3.1.2 Software Factors

Resolution

Better heading?

When remotely driving a car on a flat surface the image resolution can be as low as 64×60 px pixels with a 80°×60° angle of view. For driving in terrain, the resolution must be at least 20 times larger. Increasing the resolution per degree by shrinking the angle of view hampers the driver's ability to get a correct picture of the environment and may lead to disorientation. [14] It is easily seen that image cropping will affect the angle of view, since outer parts of the image is removed, while scaling will lower the resolution per degree.

Color Depth

A higher image color depth increases the driver's ability avoid obstacles. Color images are needed when driving a car in rough terrain, black and white images are sufficient when driving on a paved surface. [14]

Frame Rate and Latency

A low frame rate hampers perception of motion, speed and heading. For driving a car through a curve, the driver needs a frame rate of at least 10 Hz. Delayed feedback through video latency degrades manual control performance and may, with considerable time delays, lead to the driver employing a go-and-wait or bang-bang (e.g. max left - max right - max left) control strategy. Lower frame rate and longer latencies seem to affect the driving performance in the same ways. [14, 13] Tests performed in a helicopter flight simulator show that a pilot's ability to hover at a fixed position was hampered when the video latency reached 134 ms [19]. The authors believe that increased velocity and acceleration will require a higher frame rate, due to the larger changes between to frames.

Control Aid

At the client application side, the pilot can be aided by a grid projected on the ground, indicating the UAV's relative movements and orientation. A computer-generated world can also be projected around the visible image, effectively enlarging it. It is believed that this kind of enhancement allows the pilot to acquire the information directly by the visual system, without having to manually interpret overlay data. [14]

Address each factor and provide a "solution". Separate section? Put in different part of design? Some discussion in video_factors.txt

3.2 Parrot AR.Drone Experiments

To better understand the requirements of the new camera system some tests were performed with the Parrot AR.Drone[33] (figure 3.1). The AR.Drone is a quadcopter that can easily be controlled from a smartphone or a computer through a WiFi connection. It weighs 420 grams and measures 52.5 cm × 51.5 cm with its protective indoor frame mounted. Contrary to the CrazyFlie, the AR.Drone has sensors that allows it to maintain its position in the air. [34]

The AR.Drone streams video from two cameras in a format called UVLC, which very closely resembles a stream of JPEG images. One camera is mounted on the front, and streams images with 93° diagonal angle of view and QVGA resolution. The other camera is mounted downwards, and streams images with a 64° diagonal angle of view and QCIF resolution. Both streams provide images at 15 FPS. [34, 30]

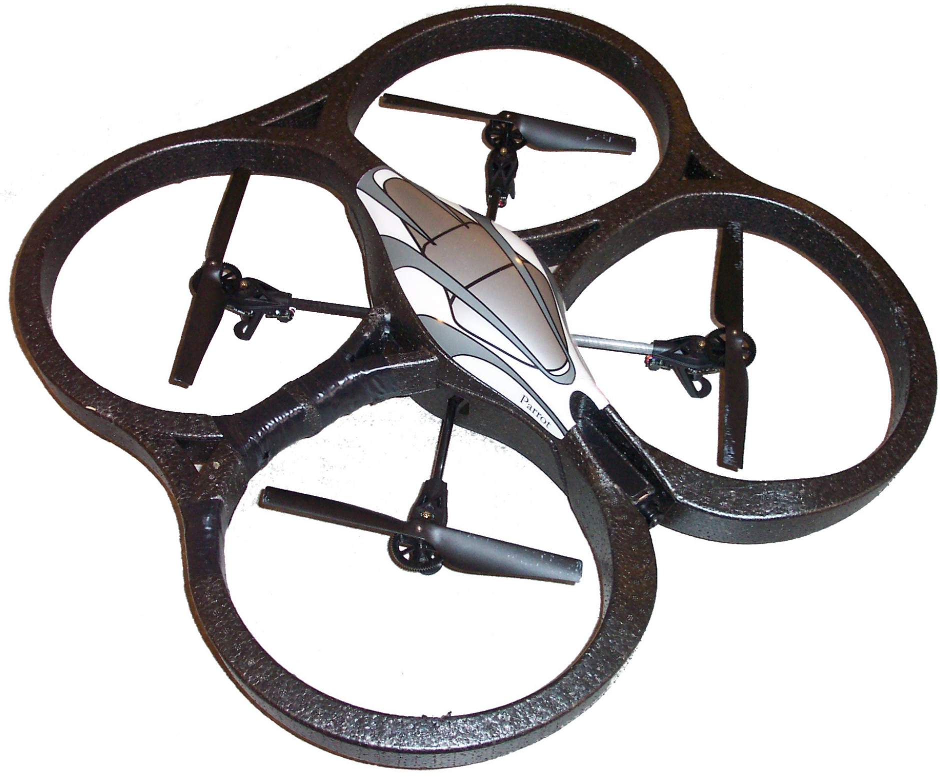


Figure 3.1: Parrot AR.Drone

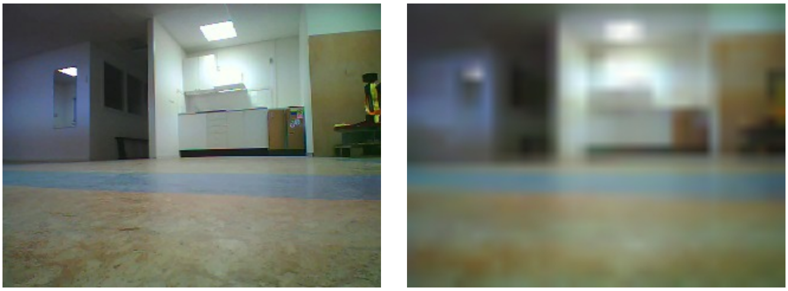
3.2.1 Testing and Results

To test how the various video factors (section 3.1) affected the performance of the pilot, the AR.Drone computer client software was modified in several ways. Support for changing the resolution (by scaling), the angle of view (by cropping), the frame rate (by dropping frames), the video delay (by buffering frames) and the color depth (color/grayscale) was implemented. The AR.Drone firmware version was 1.7.11 and client software version was 1.8.

code/modifications in appendix?

Initially a test course was set up to test how the time to complete the course was affected by different video settings, however, lack of experienced pilots made it impossible to get repeatable results. Some observations made by the authors will be listed instead.

The retrieved image was viewed at QVGA resolution in all tests, by performing a suitable scaling after any previous image modifications. It is worth to note that the AR.Drone performs lossy compression on the image data before sending it to the client [30]. This will most certainly make subsequent transformations appear to degrade the images more than if no lossy compression was used. When looking at the results one should also remember the very good stabilizing features of the AR.Drone, making it easier to control than the CrazyFlie.



(a) QVGA

(b) QQQQVGA

Figure 3.2: AR.Drone image at QVGA (a) and QQQQVGA (b) resolution (larger versions can be found in figure B.1)

To test the effect of different camera resolutions, the received image was scaled in several steps to down QQQQVGA. The pilot had no problems navigating the AR.Drone at QQQQVGA (figure 3.2), but felt that the image was much more blurry compared to QQQVGA. All images can be found in section B.1.

nice pic that explains angle of view and cropping? at least figure that shows how to measure it. can show both sides of lens :)

Angle of view tests was performed by cropping the image with a diagonal cropping factor from 1 to 2 (area factor of 1 to 4). Measurements of the physical distance between the endpoints in view was taken and the angles calculated as described in appendix B. It was found that the pilot had no problems navigating the AR.Drone even with a diagonal angle as small as 50° (this corresponds well to the research described in section 3.1.1). Please refer to section B.2 for the complete set of images.

The frame rate was easily halved by dropping every other frame, resulting in 7.5 FPS. This did not have a large impact on navigating the AR.Drone. Neither did turning the color image into grayscale.

Introducing delay was done by buffering the retrieved images in a FIFO. However, due to the client image pipeline not being timed there was no way to control the exact time delay with a few simple software modifications. Also, the delay from taking the picture to receiving it in the client was unknown. The authors concluded that large delays resulted in go-and-wait and bang-bang control, as mentioned in section 3.1.2.

It was also observed that, using the front camera, it was very hard for the pilot to determine the altitude of the AR.Drone. Not being able to see the edges of the AR.Drone made the pilot unable to avoid hitting the walls while flying close to them. This could probably have been remedied by the ideas in section 3.1.1.

Using the bottom camera was found to be of no use, since indoors the AR.Drone cannot cover much of the ground at an altitude of about 2 meters. Further, a floor without distinct features made it very hard for the pilot to navigate.

3.3 Raw formats

The data output from an image sensor after image processing can be in a few different formats, when the intended end format is a JPEG compressed image the $Y'C_bC_r$ format is most commonly used. This due to that when luma and chroma is separated the JPEG algorithms is able to achieve a better compression ratio [21].

Three different $Y'C_bC_r$ formats are usually available, $Y'C_bC_r$ 4:4:4, $Y'C_bC_r$ 4:2:2 and $Y'C_bC_r$ 4:2:0 where the difference is the amount of chroma subsampling. For all three versions the Y' channel is always represented with 8 bits per pixel. For 4:4:4 no chroma subsampling is done, so the chroma channels are represented with 8 bits each, giving a total for 24 bits per pixel. With 4:2:2 subsampling both C_b and C_r is subsampled with a factor 2 in the horizontal direction, giving an average of 4 bits for each chroma channel per pixel and a total average of 16 bits per pixel. Finally for 4:2:0 the C_b and C_r channels is subsampled with a factor 2 in both the horizontal and the vertical direction giving an average of 2 bits per pixel for each of the chroma channels and a total average of 12 bits per pixel.[21, 32]

3.4 Theoretical Requirements

3.4.1 Frame rate

Using information from section 3.1.2 and the frame rate tests of the AR.Drone in section 3.2 the target frame rate of the image stream is set to 15 FPS, but may

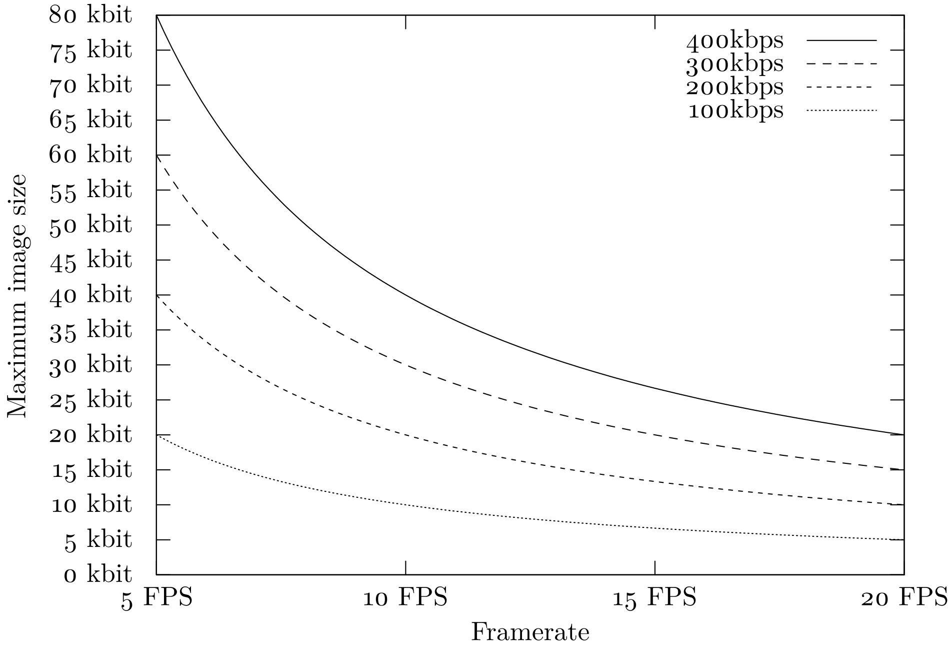


Figure 3.3: Maximum image size

if necessary be adjusted down towards 10 FPS. Even if it worked well to control the AR.Drone down at 7.5 FPS a higher target is chosen as the movement of the CrazyFlie is much quicker than those of the AR.Drone.

3.4.2 Image file size

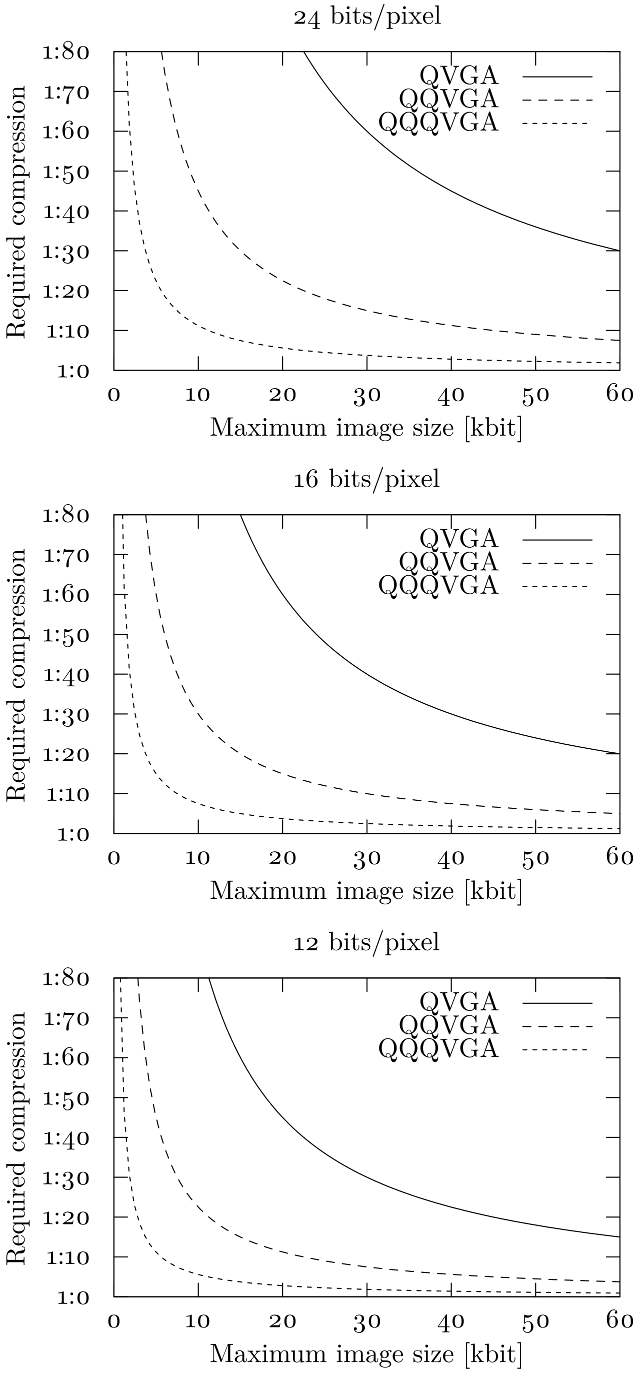
From the radio bandwidth calculations the maximum size of an image to meet a certain frame rate can be calculated according to equation 3.1. From this the graph in figure 3.3 is acquired for a few different image transfer rates.

$$\text{maximum image size} = \frac{\text{transfer rate}}{\text{frame rate}} \tag{3.1}$$

The size of a raw image is given by equation 3.2 and the required compression given a maximum allowed size of the final image is then given by equation 3.3. From this the plots in figure 3.4 is derived.

$$\text{raw image size} = \text{width} \cdot \text{height} \cdot \text{bits per pixel} \tag{3.2}$$

$$\text{required compression} = \frac{\text{raw image size}}{\text{max image size}} \tag{3.3}$$



24 bits/pixel

16 bits/pixel

12 bits/pixel

Figure 3.4: Required compression

To determine how much compression that could be applied to the images without losing too much details so that it would become to hard to navigate [41] was used to see the effects of different compression ratios. It was determined that up to 1:40 compression would work, but going above that would begin distorting the image to much.

Cross-referencing figure 3.3 and figure 3.4 it is directly seen that some compression must be applied to the raw image if the target frame rate is to be met. Even with compression QVGA resolution will probably be to large, requiring an JPEG compression ratio of 1:60 of a $Y'C_bC_r$ 4:4:4 image or 1:30 of a $Y'C_bC_r$ 4:2:0 image if the data rate can be sustained at 300 kbps. Moving down to QQVGA instead a compression ratio of just 1:33.75 for $Y'C_bC_r$ 4:4:4 or 1:22.5 for $Y'C_bC_r$ 4:2:2 is required to sustain 15 FPS with a data rate of 200 kbps.

Looking back at the measurements of the achievable data rates for image data over the radio link in section 2.3.2 the available speed should be enough for QQVGA images transferred at 15 FPS over a 2 Mbps link. Moving further down to QQQVGA would then also enable the use of 1 Mbps link speed.

Chapter 4

Cameras

This chapter first provides a summary of the requirements and restrictions discovered in Chapters 2 and 3. Then general information on image sensors (chip cameras) and the process of investigating different sensors and the findings is described. In the end, a few design alternatives are presented and one is selected for use in the final product.

4.1 Summary of Requirements

move requirements summary to video?

Summarizing the results from Chapters 2 and 3, the following list of requirements and guidelines can be compiled:

Power consumption of maximally a few hundred milliwatts

The original system consumes 250 mW not accounting for the motors and 5 W including them. A camera module consumption of 500 mW would then correspond to a 10% increase in power consumption and doing a rough calculation, assuming that the battery can deliver $E = P_{quadcopter}t_{flight}$, the flight time would be reduced by as much as 10%. Therefore the power consumption should not be higher than a few hundred milliwatts.

Supply voltage of 2.8 V

2.8 V is the only regulated voltage available in the original system. If other voltage levels are required, more regulators must be added. In that case, the regulators must be able to use the 3.7 V (nominal) battery voltage or the regulated 2.8 V as input. There should be as few extra regulators as possible to reduce waste of energy and PCB area.

current consumption?

SPI, I2C, USART or other interface requiring few I/O pins

These are the interfaces that the quadcopter provides. As described in section 2.1.2 the SPI interface cannot be replaced and one has to choose one of I2C, USART or direct I/O for the other pins.

Maximum weight of 4 to 7 grams mounted in the center

As found earlier, the maneuverability of the quadcopter was heavily reduced when the attached weight reached 7 g. Therefore this is the maximum weight allowable. As moving the weight off-center also reduced maneuverability, the module should preferably be mounted with its center of gravity in the center of the quadcopter.

explain 4 in header. is this from the non-center weight? update when new flight data acquired

Camera mounted forwards with quadcopter visible

A forward-directed camera provided much better information for navigation than a downwards-directed camera, though the exact angle has to be determined. Previous research also showed that it is beneficial for the pilot to see part of the vehicle through the camera. This might however be hard to fulfill due to the small and compact design of the quadcopter.

Minimum diagonal angle of view of $50°$, preferably $100°$

Researchers have found this to be the minimum value for adequate navigation of a car. A quadcopter can move in one more dimension, and should therefore at least require this angle of view.

No magnification

Both zoomed in and out images makes it harder for the pilot to control the vehicle, therefore the magnification factor should be 1.0.

Resolution

put px/degree calculations in video factors and refernce here?

Minimum resolution of QQQVGA, preferably QQVGA

The experiments performed indicates that QQQVGA is a viable resolution. Research indicates that, for controlling cars, it can be even lower.

Minimum 10 FPS and maximum 130 ms latency

A frame rate of at least 10 FPS is needed for remote controlling a car, which most probably means that at least 10 FPS is required for flying the quadcopter. Using the value acquired for hovering real helicopters, the maximum latency is set to 130 ms.

Compression factor of 1:30

The calculations on image size and radio bandwidth indicates that, for a QQVGA image, a compression factor of 1:30 is needed.

follow up requirements were applicable

4.2 General SOMETHING

4.2.1 Sensor Types

There are two dominating image sensor techniques: CCD and CMOS [24]. CMOS and CCD are area-scan sensors (capturing a 2-dimensional picture, as opposed to line-scan sensors which only captures 1 line). CCD sensors have traditionally performed better with regard to shutter leakage and noise but CMOS has started to reach the level of quality of CCD. CMOS have the advantages that it can support a higher frame rate and it consumes one-tenth as much power as CCD per frame, in the worst case. One disadvantage of the CMOS sensors is that most have rolling shutters, but there now exists some with global shutters. [37]

Sensors using rolling shutters expose one line at a time to the incoming light, this results in images of moving objects appearing distorted. CMOS sensors also suffer from bad image quality when the lighting is low and have smaller dynamic range compared to CCD sensors. However, CMOS has inherent antiblooming (suppressing overexposure spreading to neighbouring pixels) and windowing (reading out only a portion of the image sensor) capabilities. In CCD sensors the outputs are basically the electrical charges obtained in the sensor, which means that extra circuitry is needed on the PCB. CMOS on the other hand usually has the extra circuitry on the same chip and can output digital signals. [23]

CMOS sensors had 88.6 % of the area-scan market share in 2009, while CCD had 11.4 %. [24]

4.2.2 Image Sensor Hardware Interfaces

Most image sensors have two communication interfaces, one for control messages and one for data. This section gives a brief description of these interfaces. Note that pin naming is not the same in all datasheets, but the names can easily be mapped. Figure 4.1 gives a graphical overview of the pinouts.

All sensors found (section 4.3.1) has an I2C interface for control and setup. The Mobile Industry Processor Interface (MIPI) organization refers to this as Camera

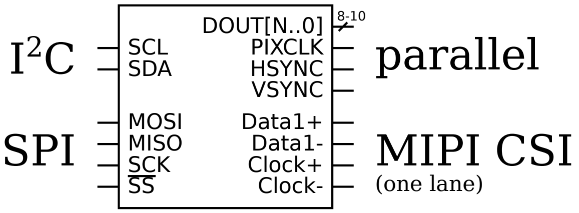


Figure 4.1: Pin configurations of some common image sensor interfaces

Control Interface (CCI) [25] and OmniVision refers to it as Serial Camera Control Bus (SCCB) [29].

The most common data output interface, for the found sensors, is an 8-10-bit parallel bus together with a few control pins. At each rising edge of the PIXCLK signal, 8-10 bits of image data is latched on the bus. HSYNC and VSYNC signals the end of row (line) and frame (image), respectively [20]. OmniVision calls the interface Digital Video Port (DVP) [26], while the MIPI organization, excluding the sync signals, refers to it as Camera Parallel Interface (CPI) [25].

The OmniVision OV3640 also has a MIPI Camera Serial Interface (CSI). This high speed interface is commonly seen in high-resolution image sensors and has one or several (CSI-2) *lanes* that transport image data serially. The physical signaling is either differential or single-ended. [26, 22, 25]

The PixArt PAS6167 is also able to output its image data via SPI.

comment on un-fulfilled requirements

4.3 Candidate Chips

Two kinds of chips were found in the camera chip investigation. Image sensors capture an image and output it. Some image sensors have extra image processing capabilities and are then usually referred to as SOCs. Image signal processors (ISPs) processes an image taken by an image sensor and outputs the result. The processing may include scaling, compression and different output format.

The following sections describe the chips found that was deemed to be most interesting the camera solution.

4.3.1 Image Sensors

Image sensors were sought in existing products, papers and directly on the Internet. Modules, with both sensor and lens were also sought, both in an individual search and modules containing the previously found image sensors. All sensors found were of the CMOS type.

include info on existing products? 808, bluetooth based camera, mostly larger cameras, kkmulticopter, liu copter?

Table 4.1 lists the most suitable sensor canditates. The notation for *Type* is as follows:

- SOC - System on Chip
- Module - Sensor with mounted lens
- w/ cable - Cable extending from the package
- Wafer-level camera - Chip with integrated lens

Note that the information from most vendors is very sparse and power consumption is usually given only for the maximum resolution and the most favorable supply voltages. Only sensors that have packages, that is, not only wafer/bare die, are included. When ranges of supply voltages are valid, the values best matching the CrazyFlie are chosen. All the found sensors operate with an input clock in the range of $6 - 80$ MHz and can output images at a rate of at least 15FPS when outputting VGA or, if maximum the resolution is less than VGA, at their maximum resolution.

Manufacturer	Part	Type	Output	Compr.	Power	Supply	Resolution		Package	Dimensio
							Min	Max		(*w*×*h*×*d* m
Aptina	MT9V111	SOC	parallel	no	<80mW	2.8V	QVGA	VGA	ICSP-44	
Aptina	MT9D131	SOC	parallel	JPEG	348mW	1.8V, 2.8V		UXGA	CLCC-48, iCSP-64	14×14 (CI
GalaxyCore	GC0306	Module w/cable		no	[<12mA]	1.8V, 2.8V	CIF	VGA	PLCC, CSP	
Omnivision	OV3640	SOC	MIPI, parallel	JPEG	[70mA]	1.5V, 1.8V, 2.8V		QXGA	CSP2-56	6.3×6.1
Omnivision	OVM7690	Wafer-level camera	parallel	no	100mW	2.8V	QCIF	VGA	CSP-20	2.5×2.5×2
PixArt Imaging Inc.	PAS6167	SOC	parallel, SPI	no	[16mA]	1.8V, 2.8V		QCIF+	CSP-20	5×5×3
PixelPlus	PO5010K	SOC	parallel	no	36mW	1.8V, 2.8V	QQCIF	CIF	CSP-31	
Sony	MCB770	SOC Module	parallel	JPEG	170mW	1.2V, 1.8V, 2.8V		UXGA		14.5×17.4×
Toshiba	TCM8240MD	Module	parallel	JPEG[1]	[120mA]	1.6V, 2.8V	Sub-QCIF	SXGA		
Toshiba	TCM8230MD	Module	parallel	no	[40mA]	1.5V, 2.8V	Sub-QCIF	VGA		6×6×4.5

[1] The datasheet seems to indicate that the TCM8240MD does not support JPEG compression when not using full resolution. No sources contradicting this have b

Table 4.1: Image sensor candidates [10, 9, 16, 26, 28, 17, 31, 11, 39, 38]

list package dimensions instead of package type?

short discussion on interesting sensors/interesting features?

switch dated galaxycore camera to new model? it is important that it has a flexible flat cable. gc0309 datasheet: http://www.docin.com/p-240246436.html

explain what factors make these sensors interesting. give more info on the most interesting ones

Image sensors

4.3.2 Image Signal Processors

ISPs provide a way to post-process the captured images by adding a chip in-between the image sensor chip and the image destination. Features important for the design of the camera system is functionality to shrink the image by scaling and compression, and reducing the number of I/O pins required by transforming the data into serial form.

Table 4.2 lists the ISPs found to be best suited for the camera system. All the listed chips perform JPEG compression. Only the supply voltages best matching the CrazyFlie are shown.

Out of the listed chips, OV529 is the chip with the smallest power consumption able to provide an image scaled to less than QVGA. Except for the ADV121, data may be communicated through SPI or UART, which is readily available on the CrazyFlie. However, for these chips, information and datasheets can only be acquired after signing non-disclosure agreements with the respective vendor and, in some cases, ordering thousands of components.

open source is a goal

Manufacturer	Part	Output	Power	Supply	Resolution	
					Min	Max
Analog Devices	ADV212	parallel, DMA, SRAM, JDATA	[400mA]	1.5V, 2.5V		4096×4096
Conexant	CX93510	SPI, UART, I2C	[12mA]	1.8-3.6V	QVGA	VGA
Omnivision	OV529	SPI, UART, parallel	100mW			VGA
Vimicro	VC0706	SPI, UART, parallel, composite	[65mA]	1.2V, 3.3V	CIF	VGA

Table 4.2: ISP candidates [2, 7, 27, 40]

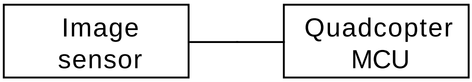


Figure 4.2: Design in which data is transferred directly to the Quadcopter

all have jpeg compression chips that do too much 808 mpeg

explain that lenses are often acquired separately. try to find angle values for sensors with lenses

mpeg, frame drop

beskrivning av alternativ, kort om de olika produkterna?

section on possibility to fulfill matching requirements?

4.4 Requirements Follow-up

4.5 Design Ideas

In this section follows a number of design proposals for the camera system.

Any compression is performed in hardware since it is feared that software processing will be too slow and consume too much energy. No sources indicating the opposite was found during the research.

comment on cameras with radios

4.5.1 Data Transferred Directly to Quadcopter

Transferring the image data directly from the image sensor to the quadcopter will create the smallest possible hardware design (figure 4.2). To be able to do this, the sensor will need to have an interface compatible with the quadcopter's available interfaces (section 2.1.2).

Having a sensor which can do all image processing will make the overhead in the quadcopter as small as possible. Unfortunately, the authors have not been able to find such a chip which has an interface compatible with the quadcopter MCU.

The second alternative is to have the quadcopter MCU do image compression and utilize an image sensor with SPI, such as the PAS6167. This will however result in a large computation overhead, leading to delayed images, higher power consumption and probably lower frame rate.

source. even possible? preferably find test on stm32

Replacing the MCU with a more powerful or specialized DSP could be one way to overcome the previously mentioned problems, however, this will most

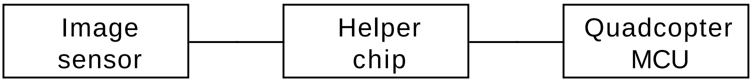


Figure 4.3: Design in which data is processed by a helper chip

certainly require redesign of the quadcopter PCB, since even the more powerful MCUs in the same family have larger footprints than the quadcopter MCU http://www.st.com/internet/mcu/class/1734.jsp. Also, the add-on pin header would need to be extended to accept parallel data.

mipi. differental. too fast?

4.5.2 Data Processed by Helper Chip

The following design ideas utilize an external helper chip placed in-between the image sensor and the quadcopter MCU (figure 4.3). If the helper chip is more advanced it may be possible, through handshakes, to let it communicate directly with the radio chip using the SPI bus.

External Data Serialization

Using an image sensor capable of image processing and compression, one could use a very simple helper chip that only translates parallel data from the sensor to use the quadcopter's SPI connection. Control of the image sensor would be done directly by the quadcopter MCU through the I2C interface.

There exists, what is called, FIFO chips that have this functionality, but an MCU could also be used, possibly also controlling the image sensor.

explain why a fifo chip was not chosen. maybe in the next chapter

Looking at image sensors, any of the JPEG capable sensors should be usable. At a first glance, the TCM8240MD looks to be the perfect choice, however, the datasheet seems to indicate that JPEG compression can only be performed at SXGA resolution [39]. As discussed earlier (REF to prev section or merge sections) it is preferable to not perform image processing in the quadcopter MCU. The choice is then limited to the MT9D131, OVM7690, MCB770, where the MT9D131 is the only sensor with a free datasheet.

list physical size in table.

mt9d131: 348mW tcm8240md: 120mA@2.8V 1.6V?

External Image Processing

Using an extra chip for image processing frees the quadcopter MCU from data processing and allows a greater choice of image sensors, since they do not need to contain any extra processing capabilities.

Having all processing done in hardware - low power consumption. source. jpeg chips have small mcus, need to compare with regular mcu running jpeg?

As described in section 4.3.2, there exists several ISP chips optimized for providing this functionality together with an interface matching one of the interfaces of the quadcopter (section 2.1.2). Of the chips listed, only the ADV212 has a public datasheet, but compared to the other chips has a lot more features and also uses much more power.

ov529: 100mW active, 40mW standby vc0706: 65mA@3.3V 1.2V? cameracube: 100mW tcm8230 mt9v111: <80mW sources in spreadsheet

software jpeg, e.g. http://www.diydrones.com/profiles/blog/list?user=JackCrossfire . uses stm32f4

An alternative, requiring more development effort, is to use an FPGA performing the same operations as an ISP chip.

more info

It is possible to use any one of the sensors, with parallel data output, in this design. It is therefore the most flexible design with regard to image sensor selection.

analog camera? discuss somewhere. adc-conversion. direct radio to separate receiver without using quad radio

4.6 vald kamera/lÃ¶sning/Chosen Solution?

describe mentioned sensors below sensor table

The most compact solution would be using the OVM7690 wafer-level camera or the TCM8240MD camera module. Both cameras require separate compression, since the OVM7690 has no built-in compression and the TCM8240MD only compresses SXGA images. Since the compression can't be done in the quadcopter it has to be done externally.

Due to the difficulties of acquiring documents for the ISP chips and the resulting difficulties in publishing all results, it was decided that the design should not use any ISP chip.

The alternatives remaining was either to create custom firmware to do encoding, using an FPGA, or having the image sensor do the compression and then serialize the data. Creating custom compression firmware would require wasting extra development and testing, as compared to using a pre-made JPEG encoder, and thus it was decided that an image sensor with built-in JPEG compression should be used. Of the found JPEG image sensors, the MT9D131 was the only one with a publicly available datasheet and it was therefore chosen.

The proposed solution looks as follows: The MT9D131 captures, scales and compresses images. The parallel data from the image sensor is then input into a chip for serialization and finally sent to the quadcopter MCU.

Chapter 5

System Design

5.1 Hardware

5.1.1 Selection

In section 4 the MT9D131 was found to be the camera most suited to use in the design. As this camera has a wide parallel interface some intermediate module must be added to do serial to parallel conversion of the data before it can enter the CrazyFlie microcontroller. The requirements of this module is that it must be small and as power-efficient as possible. It was also preferable if this module could contain all logic necessary for controlling the camera module as fewer changes then would be necessary in the CrazyFlie code and it would also enable debugging of the camera module without the need of having the quadcopter connected.

Three different choices for this module was found, FIFO chips supporting parallel in and serial out, an FPGA chip or an MCU.

The FIFOs had the disadvantage that either another module with control logic would have had to be added if the control of the camera was to be made outside the quadcopter. Also a problem was that the power consumption of all such FIFOs found was to high.

An FPGA chip would be able to satisfy all requirements, the main problem here was development time and ease of change. Developing control logic and interface logic for the FPGA would probably take considerably longer time than implementing the same solution in software and it would be much harder for someone else to make changes to the camera module in the future as it then would require not only knowledge of C-programming but also knowledge of VHDL/Verilog.

The final solution was to use an MCU responsible for interfacing with and controlling the camera. This enables easy testing and debugging of the whole camera module, simplifies changing and updating the module in the future and gives quicker development than an FPGA solution as standard functionality of the MCU can be used in many cases.

For an MCU to be feasible for use it must at least support using DMA to read data from GPIO pins, or alternatively have native camera interface, otherwise too much CPU time will be spent on reading data from the camera, consuming extra power and possibly leaving no time for other operations that also must be done to transfer the data on to the quadcopter.

A couple microcontrollers supporting this design but not having more features than needed were considered. For this project we decided on using a STM32F103CB MCU which is identical to the one used on the quadcopter. It sports a DMA unit which can be triggered by an external signal to transfer data from GPIO pins to SRAM which would be perfect for reading the video data from the camera module. It also has I^2C and SPI interfaces which can be used to communicate with the camera and the quadcopter. There were also some STM32 MCUs which had a camera interface, but those had a larger footprint, higher clock frequencies and would consume more power and was therefore not chosen.

5.2 Design

Using the MT9D131 camera and the STM32F103CB the schematic in figure C.1 was created using KiCad, a serial port was added for debugging purposes and a 1.8V regulator for supplying the core voltage to the camera. Initially decoupling was added so that each supply pin had it's own decoupling capacitance, but due to space constraints this was later changed so that neighboring supply pins was allowed to share the same decoupling.

To allow programming and debugging the camera module both standalone and while connected to the quadcopter a schematic for a debug card was also created in figure C.2. It consists of one JTAG connector that a JTAG dongle can be connected to, a connector for the camera module and a connector that can be either connected to a power supply part of the board or to the quadcopter. The internal connections was made so that when it is connected to the supply part of the board the camera module gets its power from there and the JTAG pins is connected so that there is a correct JTAG chain with only the camera module connected. When instead the quadcopter is connected the camera module is supplied from it and the JTAG pins is now connected so that there is a JTAG chain consisting of the camera module and the quadcopter.

The initial plan had been to just connect the different components for the camera module together on a off-the-self prototype board, but due to the package of the camera module this was not an option and instead a PCB had to be designed, the tool used to design the PCB was also KiCad.

To keep weight down and make debug probing easier it was decided to not use a PCB with four layers but instead stick to two layers. Making the PCB as small as possible, to be able to mount it on the quadcopter, did not add considerably much more effort. Therefore the main goal when designing the PCB was to keep the size down as much as possible. In addition to the camera and MCU, status LEDs and access to one of the MCU's USARTs was added to aid in debugging. A barometer was also included in the design for future use in height stabilization. In figure C.3, C.4, C.5 and C.6 is the result of the final PCB design.

The PCB was then manufactured by Gold Phoenix PCB and to further keep the weight down the PCB thickness was selected as half the standard thickness. The final PCB can be seen in figure 5.1.

See if we can get images with same orientation and size

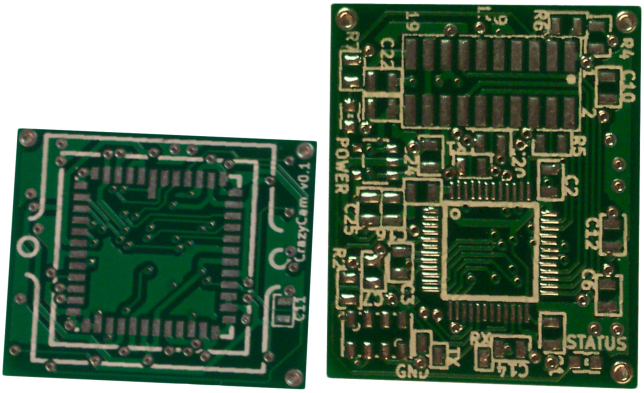


Figure 5.1: CrazyCam PCB

The components was then soldered on the PCB using a hot air station for the camera and the MCU and standard hand soldering for the rest of the components. Figure 5.2 shows the soldered CrazyCam module. Finally the lens was mounted, for the prototype a rubber band was used to allow for easy removal, the final result with lens and cable attached is show in figure 5.3.

split hardware section into subsections?

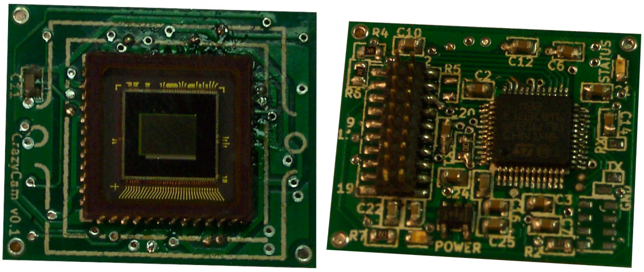


Figure 5.2: Soldered CrazyCam

5.3 Software

For the software design four main components that was to be implemented was identified, one for each part of hardware involved, and UML action diagrams was created for each part.

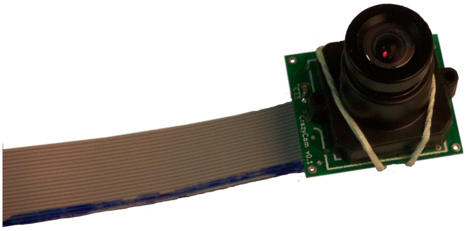


Figure 5.3: The CrazyCam module

5.3.1 Camera module

This is the code that is responsible for controlling the camera, reading the parallel data from it and then interfacing with and then transferring it in a serial manner to the quadcopter.

An overview for the design of this module is available in figure E.4. The image data is continuously transferred from the camera to a circular buffer in memory using DMA and the software just runs a loop which reads data from this buffer and transfers to the quadcopter over SPI.

To preserve power, the MCU power downs the camera and enters a low power mode when not in use. This power down mode is currently not implemented, but should be simple to add in the future.

Update code/UML/text to actually match each other

5.3.2 CrazyFlie

In the quadcopter some code must be added to receive data from the camera module and create a CRTP packet that can be sent over the radio. As the SPI bus in the current version of the CrazyFlie is shared between the radio and the camera module a mutex is used to synchronize the accesses to the bus. The main structure of the code is outlined in figure E.3

5.3.3 CrazyRadio

As it only is the radio dongle that can initiate transmissions on the radio link it has to be adapted to regularly poll the quadcopter for new image data. USB endpoint 2 is now used instead of the originally used endpoint 1, due to the ability of the chip being able to double buffer this endpoint. In figure E.2 the code flow is shown. The flow is almost the same as the one used when doing the image transfer measurements with the only addition being checks if video is enabled or not as to not send polling packets when they are not required.

5.3.4 Computer

On the computer side some changes has to be made to the code responsible for the communication with the CRTP dongle, firstly the endpoints used has to be changed to match with the crazyradio and then some changes of the code flow to handle more continuous transfers. The new flow in the communication of the radio dongle is outlined in figure E.1.

For the computer code the client software also has to be adapted so that the image data which is received is processed and then rendered in the GUI.

Chapter 6

Results

Some stuff here possibly belongs in a discussion section

The initial prototype of the camera module didn't work at first, not accepting JTAG commands at all. The problem was found to be the with the resistor R5 which was used to provide pull-down for the reset signal to the sensor, this signal was also connected to the JNRST pin on the MCU which caused the JTAG state machine to be continuously reset. This is easily solved by just removing R5 as it is not strictly needed. For further versions this resistor should either be removed completely from the design or it could be moved to pin 42 or 43 on the MCU which are easily accessible for the signal, and at those pins a pull-down wouldn't be a problem.

The target performance for the camera module was not achieved, depending on light conditions between 2 and 8 FPS can be acquired at a QQQVGA resolution with a JPEG quality setting of X giving compression ratio of about Y.

Check values

The limiting factor for the frame rate was the speed of pixel clock from the sensor, at speeds higher than 6 MHz the DMA in the MCU wasn't able to keep up any longer.

Also the exposure time currently doesn't change while the sensor is in video capture mode, so the frame rate is decided by the light conditions when the sensor is started.

The SPI communication between the camera module and the quadcopter was found to not work that well sporting problems with lost bytes. A more preferable solution would be to not have to handle a slave SPI interface in software, which should be possible on the next version for the CrazyFlie where a dedicated SPI interface will be available for external devices, allowing for the quadcopter to become the slave and just receive data over DMA.

The final weight of the camera module was Z g

Chapter 7

Conclusions

todo[inline]put this in here somewhere (about 80 FPS at 752x480 for a 100MHz RISC MCU only doing compression

not that bad.. from url in comment: software jpeg for a 32-bit mcu with dual issue

) http://www.diydrones.com/profiles/blogs/wifi-cam-complete 320x240 color 20fps stm32f4, 168MHz

Chapter 8

Future Work

References

[1] AECOM. *Exhibit 18.3: Physiology of the Eye*. 2009.

[2] Inc Analog Devices. *ADV212 Wavescale Video Codec*. Version B. 2010.

[3] Nordic Semiconductor ASA. *nRF24L01 Single Chip 2.4GHz Transceiver Product Specification*. Version 2.0. 2007.

[4] Nordic Semiconductor ASA. *nRF24LU1+ Single Chip 2.4GHz Transceiver with USB Microcontroller and Flash Memory Product Specification*. Version 1.1. 2010.

[5] Guowei Cai et al. "Systematic design methodology and construction of UAV helicopters". In: *Mechatronics* 18 (2008), pp. 545–558.

[6] Compaq et al. *Universal Serial Bus Specification*. Version 2.0. USB Implementers Forum. 2000.

[7] Conexant. *CX93510 JPEG Encoder with a BT.656 Camera Interface and Optional Microphone Input*. 2009.

[8] Gianpaolo Conte and Patrick Doherty. "Vision-Based Unmanned Aerial Vehicle Navigation Using Geo-Referenced Information". In: *EURASIP Journal on Advances in Signal Processing* 2009 (2009).

[9] Aptina Imaging Corporation. *MT9D131 1/3.2-Inch System-On-A-Chip (SOC) CMOS Digital Image Sensor*. Version F. 2006.

[10] Aptina Imaging Corporation. *MT9V11 1/4-Inch SOC VGA CMOS Active-Pixel Digital Image Sensor*. Version L. 2004.

[11] Sony Corporation. "2M-Pixel CMOS Camera Module with Autofocus Function for Mobile Equipment MCB770". In: *CX-NEWS* 41.8 (2005).

[12] *Cortex-M Debug Connectors*. ARM? URL: http://infocenter.arm.com/help/topic/com.arm.doc.faqs/attached/13634/cortex_debug_connectors.pdf.

[13] Philip N. Day, Patrik O'Brian Holt, and George T. Russell. "Modelling the Effects of Delayed Visual Feedback in Real-Time Operator Control Loops: A Cognitive Perspective". In: *Proceedings of the XVIII European Annual Conference on Human Decision Making and Manual Control* XVII (1999). Ed. by Professor James L Alty, pp. 70–96.

[14] Jan B.F. van Erp. *Controlling Unmanned Vehicles: the Human Factors Solution*. Tech. rep. RTO-MP-44. RTO SCI Symposium, 1999.

[15] Tom Frank. *The New #16 HD Key Cam (READ Posts#1-#5 BEFORE posting questions) - Post #740*. Jan. 27, 2012. URL: http://www.rcgroups.com/forums/showpost.php?p=20340906&postcount=740.

[16] GalaxyCore. *GC0306/GC0316*. Feb. 3, 2012. URL: http://www.gcoreinc.com/en/product_detail.asp_id=523.html.

[17] PixArt Imaging Inc. *PAS6167 CMOS QCIF+ DIGITAL IMAGE SENSOR*. Version 1.0. 2009.

[18] Texas Instruments Incorporated. *TPS76301, TPS76316, TPS76318, TPS76325, TPS76327, TPS76328, TPS76330, TPS76333, TPS76338, TPS76350 LOW-POWER 150-mA LOW-DROPOUT LINEAR REGULATORS*. 2004.

[19] S. Jennings et al. "The Effect of Visual System Time Delay on Helicopter Control". In: *Human Factors and Ergonomics Society Annual Meeting Proceedings* 44 (13 2000), pp. 69–72.

[20]Tareq Hasan Khan, Kahn A. Wahid, and Sandro Bartolini. "A DVP-Based Bridge Architecture to Randomly Access Pixels of High-Speed Image Sensors". In: *EURASIP Journal on Embedded Systems* 2011 (2011).

[21]Tom Lane. *[75] Introduction to JPEG*. Mar. 19, 2012. URL: http://www.cd.sc.ehu.es/DOCS/mice/compression-faq/part2/faq-doc-6.html.

[22]Kyusam Lim et al. "A Multi-Lane MIPI CSI Receiver for Mobile Camera Applications". In: *Ieee Transactions on Consumer Electronics* 56(3) (2010), pp. 1185–1190.

[23]Dave Litwiller. "CCD vs. CMOS: Facts and Fiction". In: *Photonics Spectra* January (2001).

[24]Carrie Meadows. *iSuppli: CCDs fall in image sensor market as CMOS surges*. Vision Systems Design. Jan. 27, 2012. URL: http://www.vision-systems.com/articles/2010/10/isuppli-image-sensors-market-2010.html.

[25]MIPI Alliance, Inc. *Camera Interface Specifications*. Jan. 23, 2012. URL: http://mipi.org/specifications/camera-interface.

[26]Inc. OmniVision Technologies. *OV3640 3.1 megapixel product brief*. Version 1.2. 2011.

[27]Inc Omnivision Technologies. *OV529-B64 Enhanced Single-Chip Camera Controller for Serial/Parallel bus Camera Phone Applications*. Apr. 27, 2012. URL: http://www.ovt.com/products/ip_detail.php?id=3.

[28]Inc. OmniVision Technologies. *OVM7690 640x480 CameraCubeChip product brief*. Version 1.4. 2012.

[29]OmniVision Technologies, Inc. *OmniVision Serial Camera Control Bus (SCCB) Functional Specification*. Version 2.2. 2007.

[30]Stephane Piskorski, Nicolas Brulez, and Pierre Eline. *AR.Drone Developer Guide*. Version SDK 1.7. Parrot S.A. 2011.

[31]Ltd Pixelplus Co. *PO5010K*. 2007.

[32]Charles Poynton. *Chroma subsampling notation*. Jan. 24, 2008. URL: http://www.poynton.com/PDFs/Chroma_subsampling_notation.pdf.

[33]Parrot S.A. *AR.Drone Parrot - First quadricopter that can be controlled by an iPhone/iPod Touch/iPad and Android*. Jan. 20, 2012. URL: http://ardrone.parrot.com/parrot-ar-drone/usa.

[34]Parrot SA. *AR.Drone Parrot - Technologies*. Jan. 31, 2012. URL: http://ardrone.parrot.com/parrot-ar-drone/usa/technologies.

[35]STMicroelectronics. *STM32F103x8, STM32F103xB*. Version 13. 2011.

[36]Linx Technologies. *Ultra Compact Chip Antenna Data Guide*. Nov. 30, 2011.

[37]Ann R. Thryft. "CCD vs. CMOS image sensors". In: *Test & measurement world* 31 (2011), pp. 44–45.

[38]Toshiba. *TCM8230MD (A)*. Version 1.20. 2004.

[39]Toshiba. *TCM8240MD*. Version 1.3. 2005.

[40]Vimicro. *VC0706 Digital Video Processor Datasheet*. Version 1.0. 2007.

[41]Visengi. *JPEG Encoder | VISENGI*. Feb. 19, 2012. URL: http://www.visengi.com/products/jpeg_hardware_encoder.

Appendix A

Radio rate measurements

Raw values for all measurements of the ACK data radio for the radio channel. Payload for data packets is varied between 1, 16 and 32 bytes while the ACK payload is 32 bytes in all cases. The data rate is calculated on 31 bytes to account for overhead from the CRTP header.

Distance	ACK data rate [kbps]		
[m]	1 B data	16 B data	32 B data
0.15	414.952126	343.439016	271.620429
0.15	416.730638	342.062627	270.800077
0.15	417.397466	344.309110	271.408873
0.15	412.896078	342.574685	271.483890
0.15	416.233120	345.054596	271.893704
0.15	412.788967	344.702292	270.877284
0.15	416.079125	344.194185	271.360377
0.15	411.226608	343.572717	271.579745
0.15	415.821623	342.169701	271.064760
0.15	412.989376	343.662485	271.410707
0.15	413.700427	343.874119	271.434989
0.15	414.397075	344.235667	271.534212
0.15	413.894319	345.618523	272.255289
0.15	409.699266	346.169462	272.101082
0.15	410.712359	347.294118	271.488586
0.15	409.052133	343.759051	271.277613
0.15	418.835942	345.963673	272.124457
0.15	412.316748	346.029962	271.834101
0.15	415.295319	345.427895	270.014781
0.15	421.692044	345.680828	268.114885
5	356.613338	235.954204	259.193601
5	395.767573	271.300361	270.523581
5	306.156963	236.215475	266.451924

Distance	ACK data rate [kbps]		
[m]	1 B data	16 B data	32 B data
5	421.857782	239.939908	270.527066
5	413.840121	184.558357	269.961924
5	415.704509	218.157686	271.178416
5	408.929859	172.609922	271.376343
5	409.103244	255.943350	271.494417
5	272.502537	147.777518	254.037858
5	417.460385	179.641986	271.501867
5	416.709015	119.149549	263.078300
5	395.952825	262.949705	237.979034
5	308.086124	135.864896	244.402790
5	343.703060	47.906497	233.017692
5	344.010325	150.865642	234.599862
5	221.850168	334.159048	266.234114
5	114.020723	286.190649	267.537064
5	325.221524	301.228167	241.781395
5	197.815492	312.620595	270.662376
5	212.919946	303.027646	238.538018
10	415.546352	344.728925	271.853154
10	410.255355	343.938483	271.717138
10	418.057342	344.179087	252.117830
10	420.835594	344.966452	271.319501
10	414.583815	344.486161	270.873414
10	409.096257	343.227917	271.599356
10	414.002757	343.377775	271.476818
10	421.513682	344.995654	270.892225
10	414.596279	345.704374	271.662423
10	413.442354	344.120531	271.409952
10	413.297173	343.390730	270.740524
10	414.766314	344.511196	272.114857
10	415.933987	347.257548	270.872017
10	411.938806	344.623810	271.816675
10	419.972151	342.706422	270.984489
10	413.206236	340.894412	271.857755
10	409.839346	344.957911	271.781828
10	414.783703	345.274148	271.222860
10	416.329075	343.364044	271.128333
10	414.755352	340.503155	271.269258
15	418.217673	342.639423	271.558624
15	415.757170	345.010911	271.044204
15	414.451671	343.236028	271.965359
15	417.255870	343.260968	271.422956
15	411.657727	334.130834	271.093125
15	411.001427	341.320954	271.598762

Distance	ACK data rate [kbps]		
[m]	1 B data	16 B data	32 B data
15	418.978741	339.971107	271.576233
15	415.276624	317.362817	270.418454
15	416.352817	339.565833	271.350777
15	414.917573	344.258321	271.281925
15	413.374131	343.741568	271.458898
15	411.407774	344.682363	270.791591
15	414.484384	344.315101	270.994493
15	406.399096	342.112427	271.775931
15	411.068003	343.118887	271.309041
15	409.298985	341.106314	272.435763
15	413.338964	344.115153	271.271145
15	422.717747	343.189352	271.833398
15	416.956285	343.490341	271.569265
15	416.151795	341.527150	271.667180

Table A.1: ACK data rate measurements at 2 Mbps

Distance	ACK data rate [kbps]		
[m]	1 B data	16 B data	32 B data
0.15	92.647879	154.273470	165.288608
0.15	86.687497	156.248331	165.256757
0.15	91.011630	157.833793	165.115098
0.15	91.171123	160.181766	164.991983
0.15	91.071161	158.286042	166.449881
0.15	87.849190	157.877797	165.447083
0.15	90.728153	155.777822	164.469471
0.15	90.532074	157.121838	166.812428
0.15	94.505049	155.010829	164.428330
0.15	96.294441	157.115834	165.564172
0.15	102.910444	156.227949	165.562004
0.15	97.242615	155.481702	164.541303
0.15	100.193385	157.243117	167.661100
0.15	97.339656	159.298832	166.507614
0.15	90.904414	156.092988	165.295511
0.15	91.121099	156.952898	165.177004
0.15	88.005843	155.589982	163.474036
0.15	92.725652	157.896877	164.008495
0.15	90.748934	156.305018	165.442191
0.15	98.561715	157.476447	164.830199

Distance	ACK data rate [kbps]		
[m]	1 B data	16 B data	32 B data
5	108.773256	154.999443	158.889402
5	116.993358	150.210538	160.922395
5	112.891998	151.674638	161.556536
5	114.067345	152.786524	161.581449
5	111.356723	151.784917	159.630866
5	108.417066	151.199662	159.571778
5	111.755736	153.051968	160.381317
5	108.265942	150.299022	160.773525
5	107.842458	151.443622	159.801732
5	111.240660	151.147771	159.642196
5	107.139000	149.664502	159.377976
5	108.022982	150.178004	158.467576
5	106.544739	153.155704	157.767204
5	109.457590	149.019472	157.046670
5	105.622338	150.607255	160.601161
5	104.505647	149.256948	159.577504
5	106.607917	151.461095	159.349627
5	103.083141	149.335426	160.000298
5	104.135903	149.018919	158.574236
5	107.882859	150.662682	158.773732
10	99.097683	149.243521	153.994812
10	100.440044	147.592701	155.575497
10	100.371152	146.795075	155.504866
10	99.710439	147.782349	152.895909
10	96.692236	147.403204	154.261268
10	99.230403	148.337354	158.428446
10	98.867644	148.076321	156.016234
10	98.886584	147.543208	146.486388
10	98.503042	148.779427	152.132720
10	102.286470	146.268389	155.350802
10	99.228153	148.135333	155.997498
10	102.488233	148.434295	154.192312
10	98.066222	150.150423	151.585419
10	99.514754	146.639049	153.180534
10	99.137648	147.024828	152.810738
10	98.142750	148.622117	150.985366
10	98.148197	147.952987	153.015329
10	102.554401	147.572520	133.454352
10	106.988372	147.969807	151.020171
10	99.958383	147.585681	153.122466
15	142.840159	147.570175	147.413295
15	143.900969	149.605757	148.124019
15	144.473398	148.991875	146.774764

Distance	ACK data rate [kbps]		
[m]	1 B data	16 B data	32 B data
15	144.654726	147.142575	146.803409
15	146.285283	148.462509	142.876633
15	143.683998	142.813653	145.088442
15	142.031717	137.795952	145.940333
15	144.125929	139.622275	143.390774
15	143.327731	147.659025	143.807312
15	144.996991	146.043663	148.224720
15	143.181689	146.012394	148.688123
15	141.961656	146.857350	145.472635
15	142.563048	148.353650	146.113496
15	146.159890	148.700107	144.261573
15	146.381349	148.899434	144.347504
15	149.362938	149.668948	143.280125
15	148.119567	149.208926	144.640612
15	145.217964	149.815872	146.385994
15	146.048756	149.273790	148.338966
15	149.435246	147.937725	147.576971

Table A.2: ACK data rate measurements at 1 Mbps

Distance	Image data rate [kbps]	
[m]	1 Mbps	2 Mbps
0.15	344.473818	218.333257
0.15	334.729597	218.566268
0.15	329.787866	218.741175
0.15	340.212663	219.409067
0.15	332.390113	220.022132
0.15	335.146696	222.338296
0.15	340.238450	220.335245
0.15	332.268791	224.674917
0.15	335.103481	221.796629
0.15	333.602702	220.240052
0.15	329.784246	207.775473
0.15	336.217607	223.582176
0.15	334.173908	223.198375
0.15	328.164491	221.616666
0.15	337.992192	219.432524
0.15	303.946490	224.022251
0.15	331.886382	222.103252

Distance	Image data rate [kbps]	
[m]	1 Mbps	2 Mbps
0.15	337.966426	226.112756
0.15	326.696116	224.034507
0.15	329.767563	222.902069
5	336.037688	182.050423
5	341.586398	182.380891
5	334.717991	182.745151
5	333.006777	187.505670
5	337.557852	185.343158
5	331.005635	173.689544
5	335.927585	185.572055
5	337.446513	180.762718
5	329.420439	182.015512
5	338.536571	181.225812
5	336.089953	183.848760
5	335.611342	174.596910
5	340.522497	186.655450
5	338.213509	185.913351
5	333.559547	188.984359
5	338.902114	184.756895
5	336.771592	185.049375
5	333.479406	190.624038
5	341.515440	182.328416
5	300.626209	190.651253
10	278.369548	207.848850
10	333.723341	206.274268
10	332.230323	206.020121
10	341.228813	206.299281
10	333.003938	210.194759
10	336.343911	207.801776
10	339.139671	205.995176
10	330.723487	207.291800
10	334.283410	206.284283
10	341.429016	205.539127
10	331.193916	204.344605
10	339.476094	206.788431
10	338.856027	205.539137
10	331.310263	207.234670
10	338.900965	204.978987
10	341.043167	204.641025
10	331.720288	206.777632
10	336.606772	206.885560
10	332.531908	208.129724
10	282.033285	208.064443

Distance	Image data rate [kbps]	
[m]	1 Mbps	2 Mbps
15	229.821122	187.260929
15	227.496218	193.687938
15	221.989547	190.549807
15	261.951014	187.561135
15	289.477217	129.888235
15	216.653121	184.531867
15	251.692717	189.991778
15	271.789547	190.700027
15	282.780741	189.443398
15	188.052652	192.387810
15	186.469465	189.627396
15	285.082784	174.954678
15	250.294469	157.109858
15	281.449607	178.110414
15	284.715873	171.103374
15	273.257299	181.991384
15	282.558250	186.825822
15	188.670592	179.623385
15	213.230380	189.172397
15	206.851371	172.399095

Table A.3: Image data rate measurements

Image data rate [kbps]	
1 Mbps	2 Mbps
325.622157	176.431194
340.348215	179.122392
329.479889	179.168741
322.722734	180.990881
285.500383	181.568652
310.834806	180.293306
317.164829	182.270798
327.331343	184.615492
327.401345	181.781361
331.750055	183.400960
323.294821	172.937909
323.475551	183.117363
333.117686	183.577583
335.680078	180.567708

Image data rate [kbps]	
1 Mbps	2 Mbps
324.111630	180.615347
329.587474	182.330722
326.304855	183.081058
329.098120	171.806304
335.204491	188.701629
325.285942	187.230976

Table A.4: Image data rate measurements at 10 meters through door with window

Image data rate [kbps]	
1 Mbps	2 Mbps
134.771917	171.603919
100.999835	179.594346
81.079130	176.888806
27.850033	179.272080
74.491319	167.461990
66.949518	147.978727
125.018896	163.826673
137.747370	170.885353
158.180882	175.039652
201.699235	169.534629
183.999038	177.561468
165.660632	180.383646
174.120238	157.336533
148.965221	175.817199
142.434430	170.593444
73.665554	173.241628
156.444772	175.707827
116.996759	177.266688
108.280773	176.040881
68.278899	166.057473

Table A.5: Image data rate measurements at 11 meters through door with window plus wall

Appendix B

AR.Drone Test Data

B.1 Resolution Tests

Figure B.1 shows the image from the AR.Drone scaled to different resolutions.

B.2 Angle of View Tests

Table B.1 describes the different angle of views found with different image cropping. The angles was found by pointing the AR.Drone straight towards a wall and measuring the dimensions of the resulting triangle, spanning from the camera to the endpoints visible on the wall. According to section 3.2 the front camera has a 93° diagonal angle of view, which does not match the value found in this table. It is believed that the reason being due to measurement errors and the sensor area being smaller than the lens, effectively cropping the image.

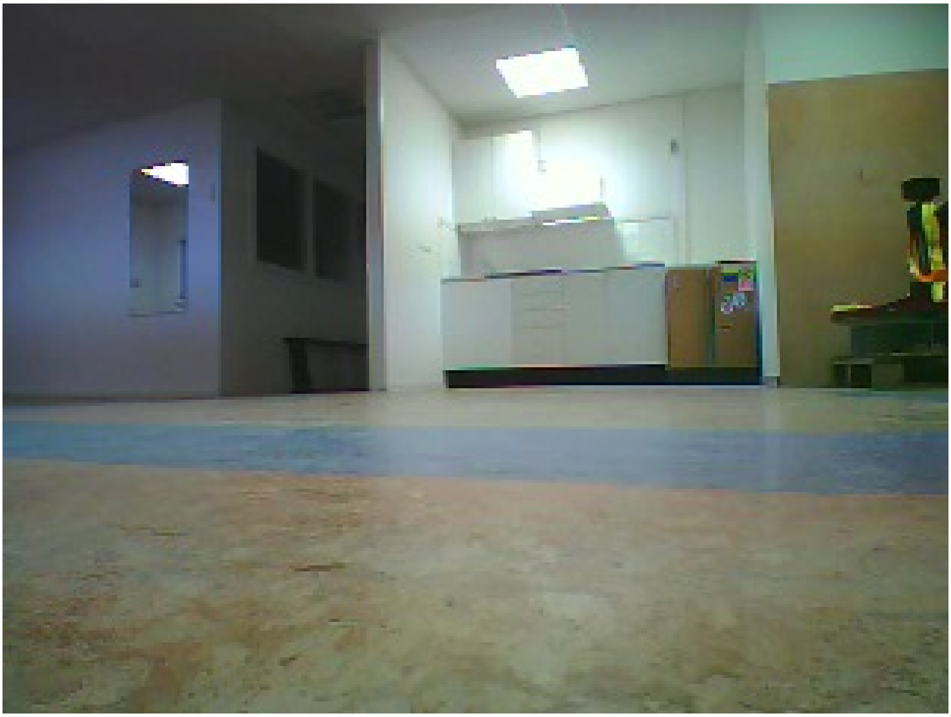
super-nice figure from notebook

The theory behind the calculation of the data can be found in section B.3

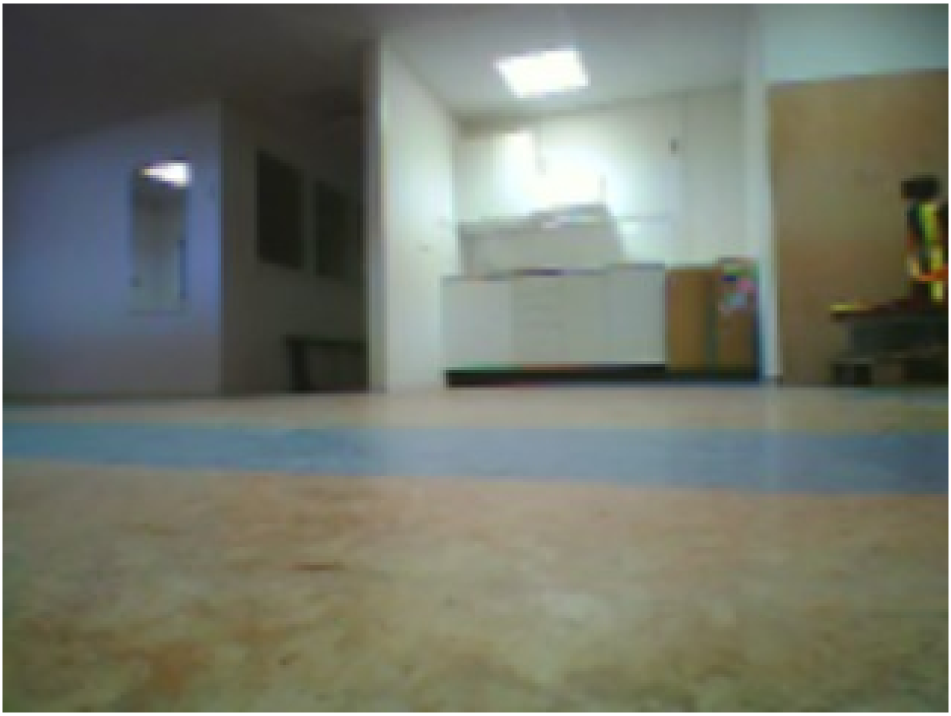
B.3 Angle Calculations

Following is an explanation of the formulas used to find the angle of view for the AR.Drone camera. The optics are very simplified and no attention has been paid to lens diffraction.

To find the angle of view (α_x) simple trigonometry is used. Take for example the calculation of the horizontal angle: Looking at the bottom triangle formed by the light visible by the sensor, in figure B.3, it can be split into two equally shaped right triangles. The distance S from the focal point to the object and half of the width W of the object inside the view makes up the catheti of these triangles. Using the trigonometric relation and doubling the result, the angle a_h is found as $\alpha_h = 2 \arctan \frac{W}{2S}$. The same calculation can be used to find the vertical (α_v) and diagonal (α_d) angles from the height (H) and the diagonal (D).

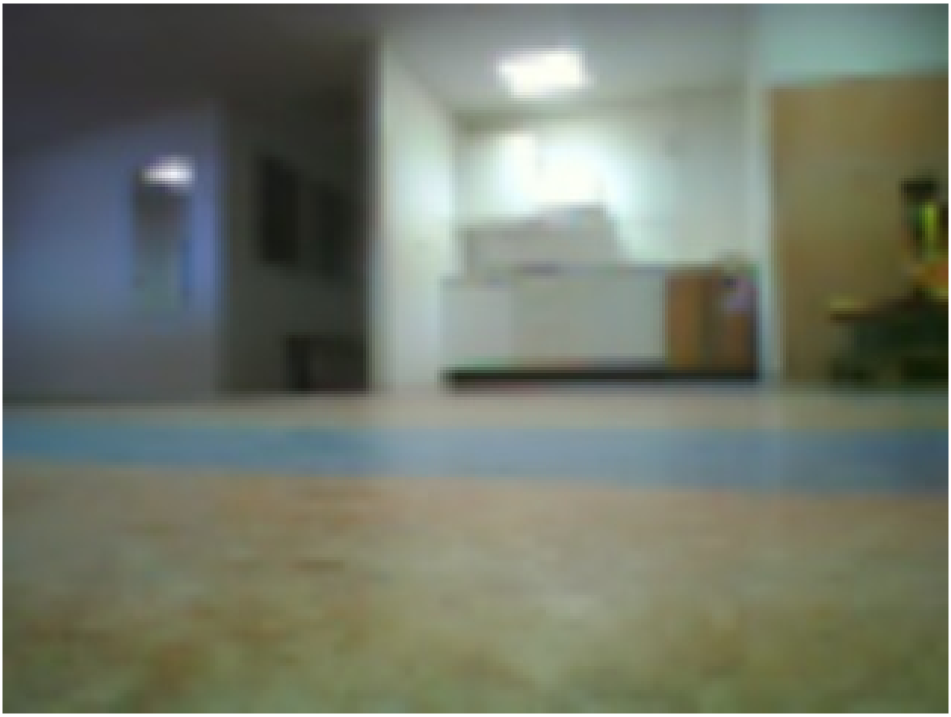


(a) QVGA

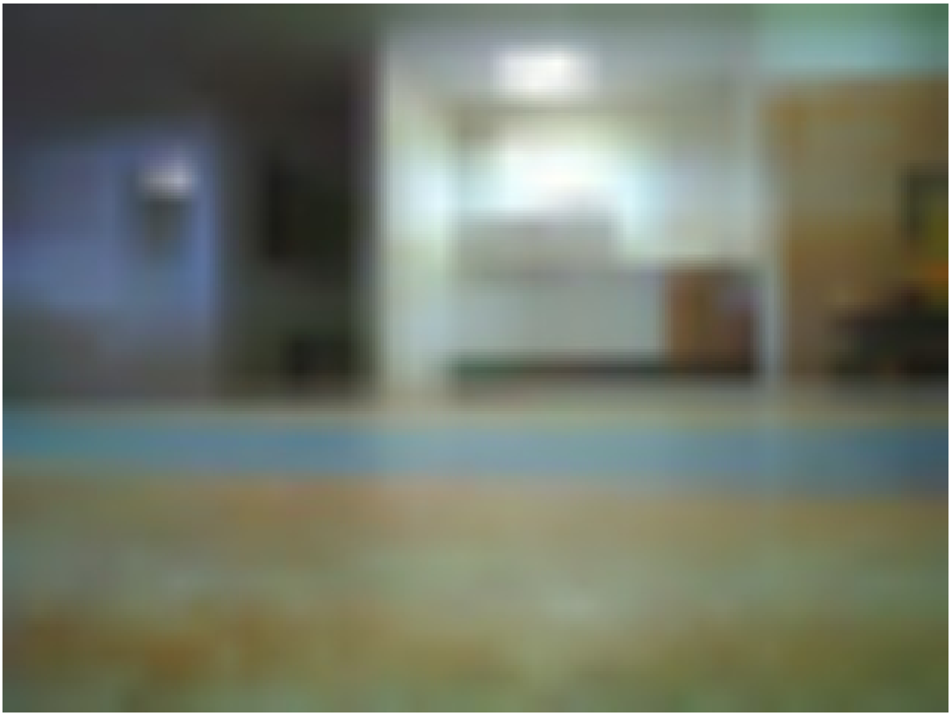


(b) QQVGA

Figure B.1: AR.Drone resolution test images, continues on next page

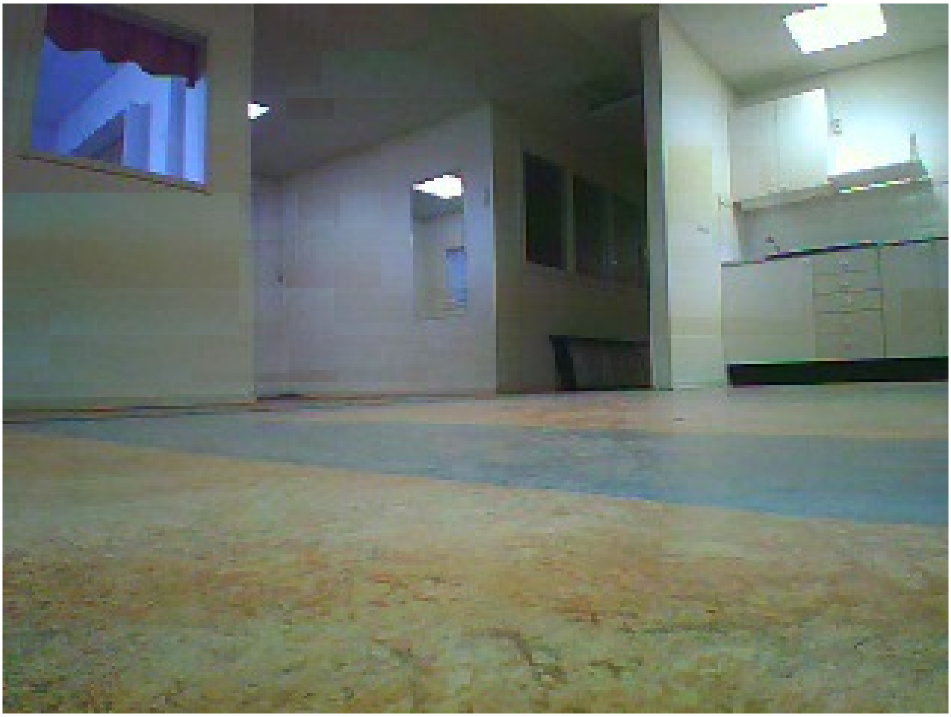


(c) QQQVGA

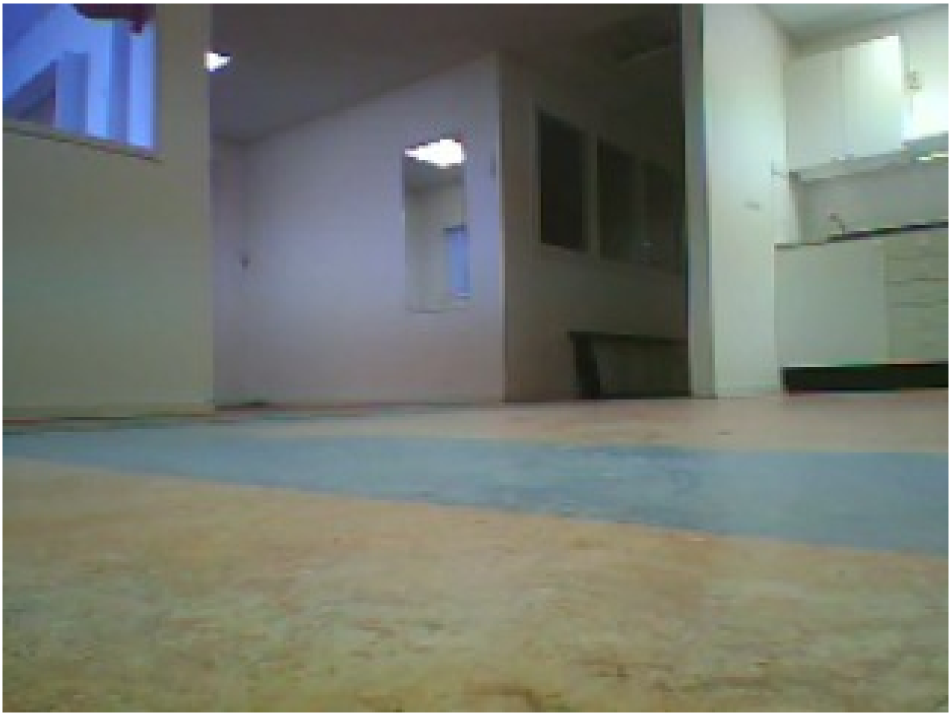


(d) QQQQVGA

Figure B.1: AR.Drone resolution test images, continued from previous page

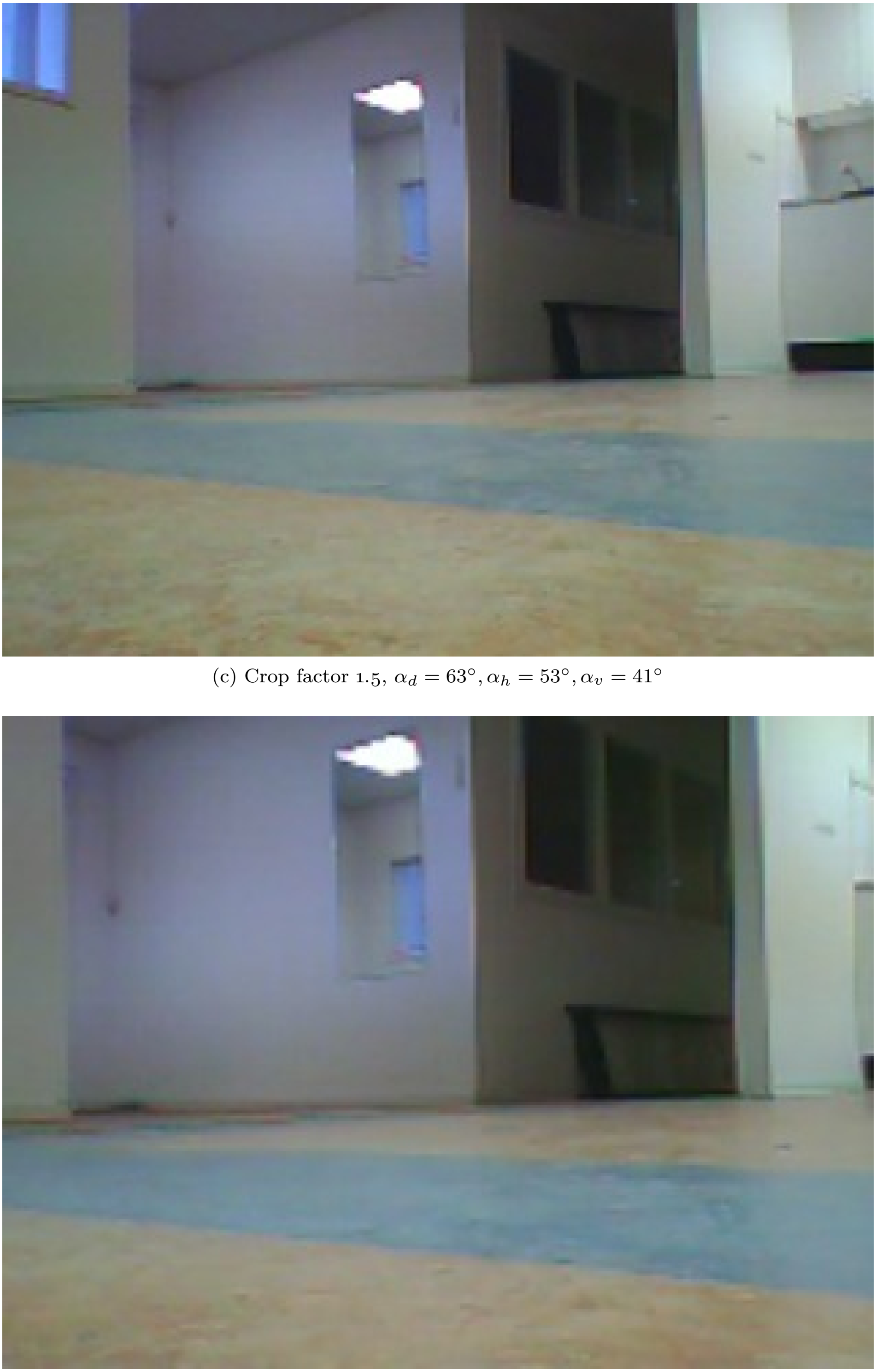


(a) Crop factor 1, $\alpha_d = 87°, \alpha_h = 75°, \alpha_v = 58°$



(b) Crop factor 1.2, $\alpha_d = 76°, \alpha_h = 64°, \alpha_v = 49°$

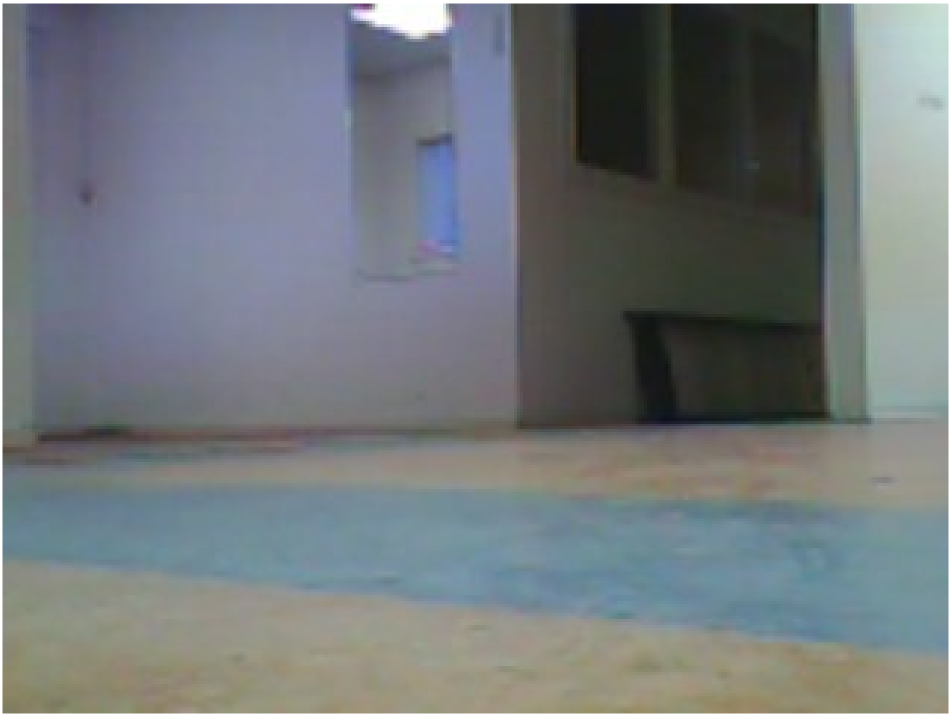
Figure B.2: AR.Drone angle of view test images, continues on next page



(c) Crop factor 1.5, $\alpha_d = 63°, \alpha_h = 53°, \alpha_v = 41°$

(d) Crop factor 1.82, $\alpha_d = 54°, \alpha_h = 44°, \alpha_v = 34°$

Figure B.2: AR.Drone angle of view test images, continues on next page



(e) Crop factor 2, $\alpha_d = 49°, \alpha_h = 41°, \alpha_v = 31°$

Figure B.2: AR.Drone angle of view test images, continued from previous page

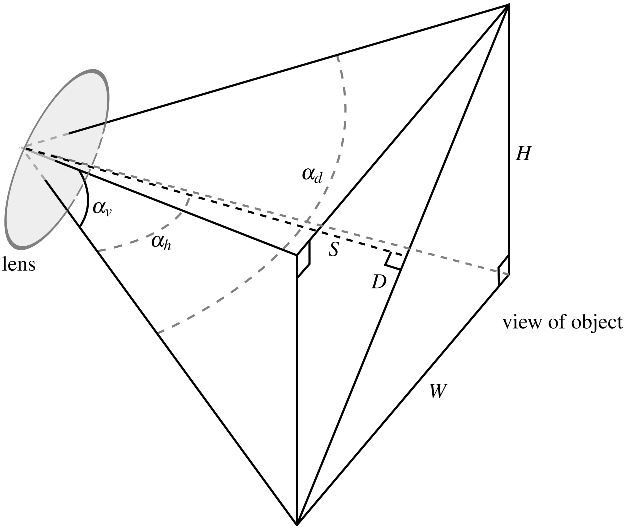


Figure B.3: Measurements used to calculate the angle of view

Distance to wall [m]	Distance between endpoints [m]	Direction	Crop Factor	Resolution [px]	Angle [°]
3.81	5.83	horizontal	1.00	320.0	75
3.81	4.74		1.20	266.7	64
3.81	3.76		1.50	213.3	53
3.81	3.08		1.82	175.8	44
3.81	2.83		2.00	160.0	41
3.80	4.20	vertical	1.00	240.0	58
3.80	3.47		1.20	200.0	49
3.80	2.82		1.50	160.0	41
3.80	2.32		1.82	131.9	34
3.80	2.13		2.00	120.0	31
3.79	7.18	diagonal	1.00	400.0	87
3.79	5.91		1.20	333.3	76
3.79	4.69		1.50	266.7	63
3.79	3.83		1.82	219.8	54
3.79	3.47		2.00	200.0	49

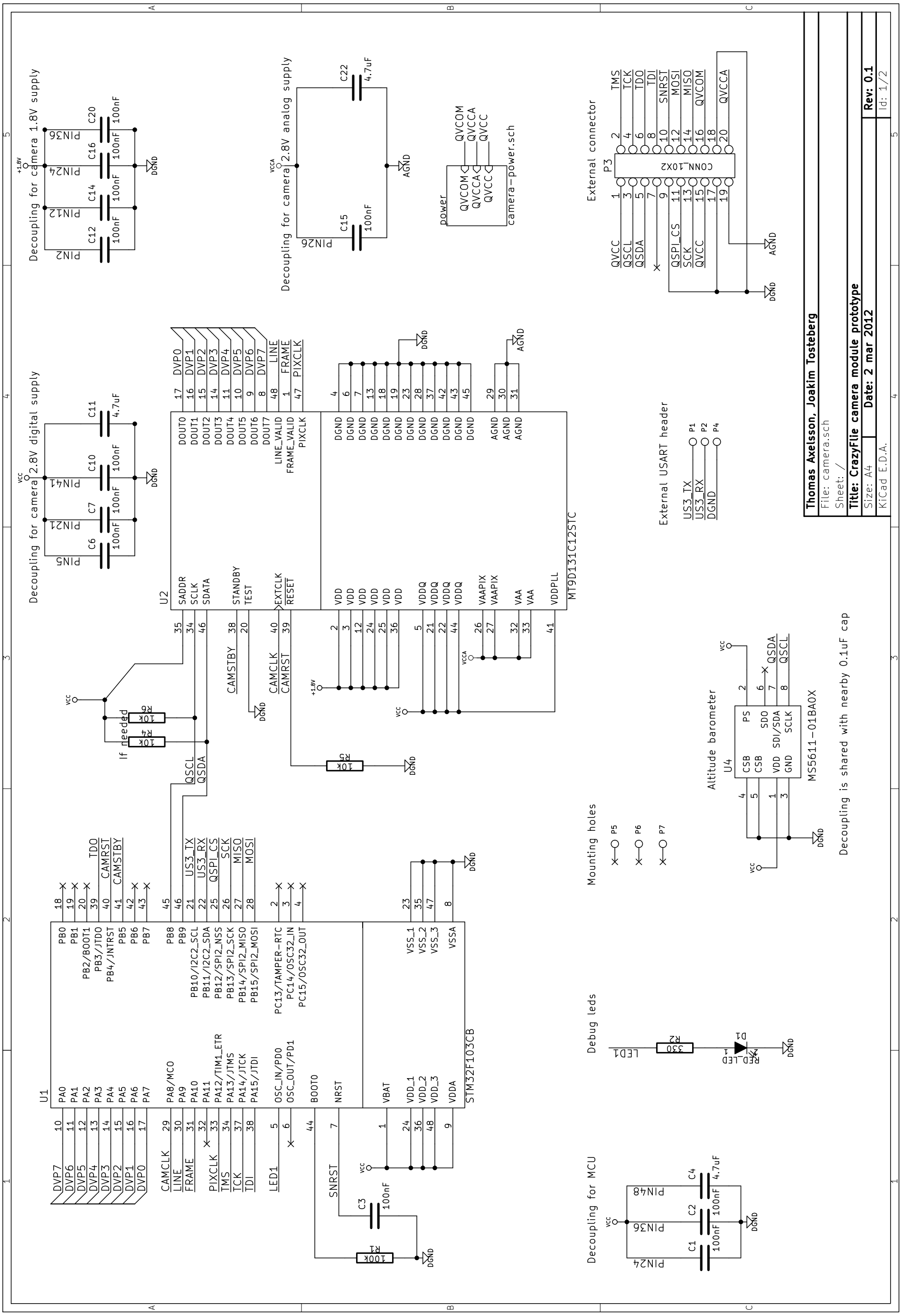
Table B.1: Drone front camera angle test data

Due to the exact location of the focal point being unknown, the distance to the camera lens was used instead. The error is less than 1 cm because the focal point must reside inside the small camera and should not affect the results considerable due to comparatively large distance to the object (several meters). Ignoring the diffraction probably adds a few degrees of error.

My reasoning seems validated by http://www.panohelp.com/lensfov.html but maybe get a proper source. It does feel like common knowledge though :)

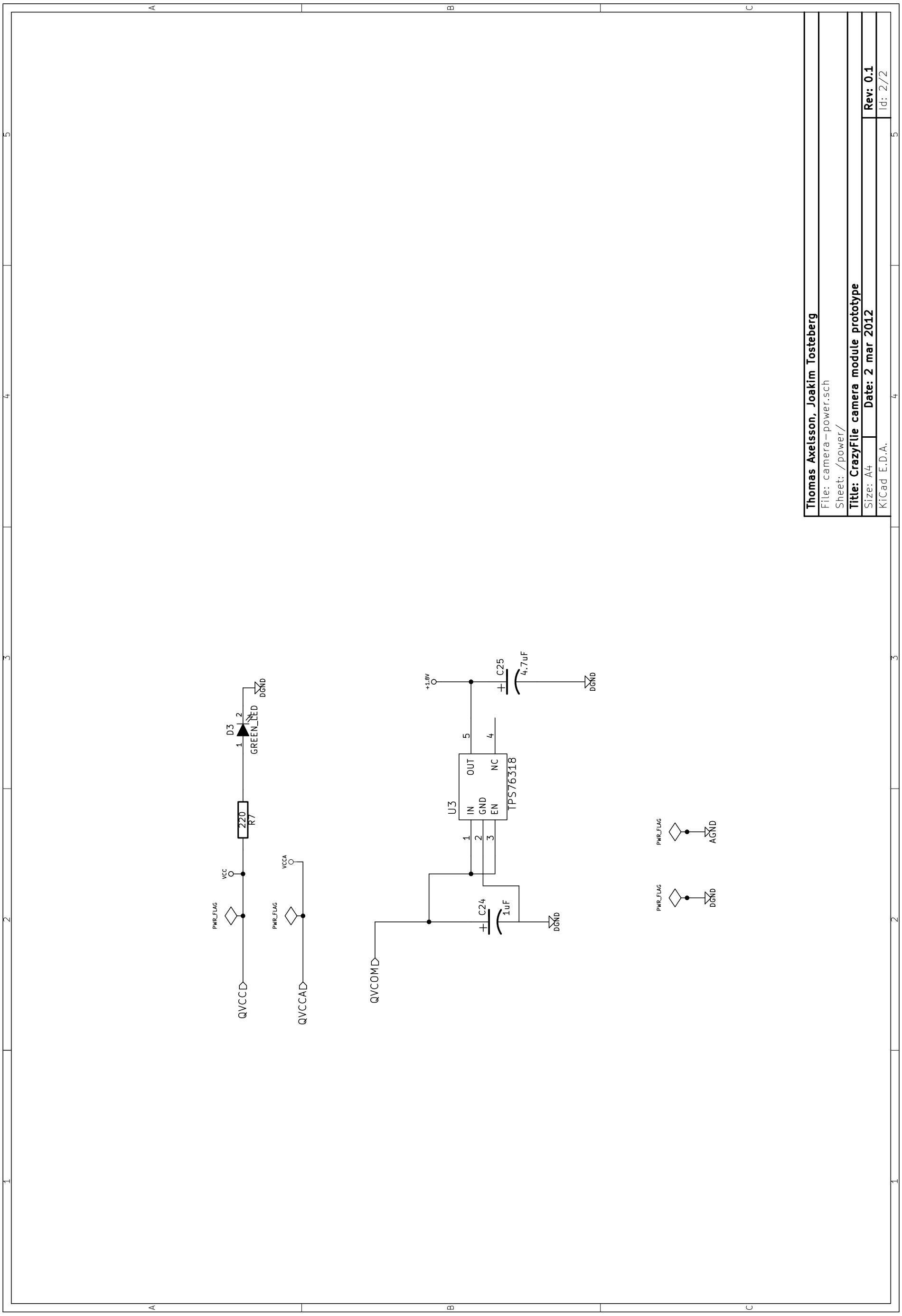
Appendix C

Schematics



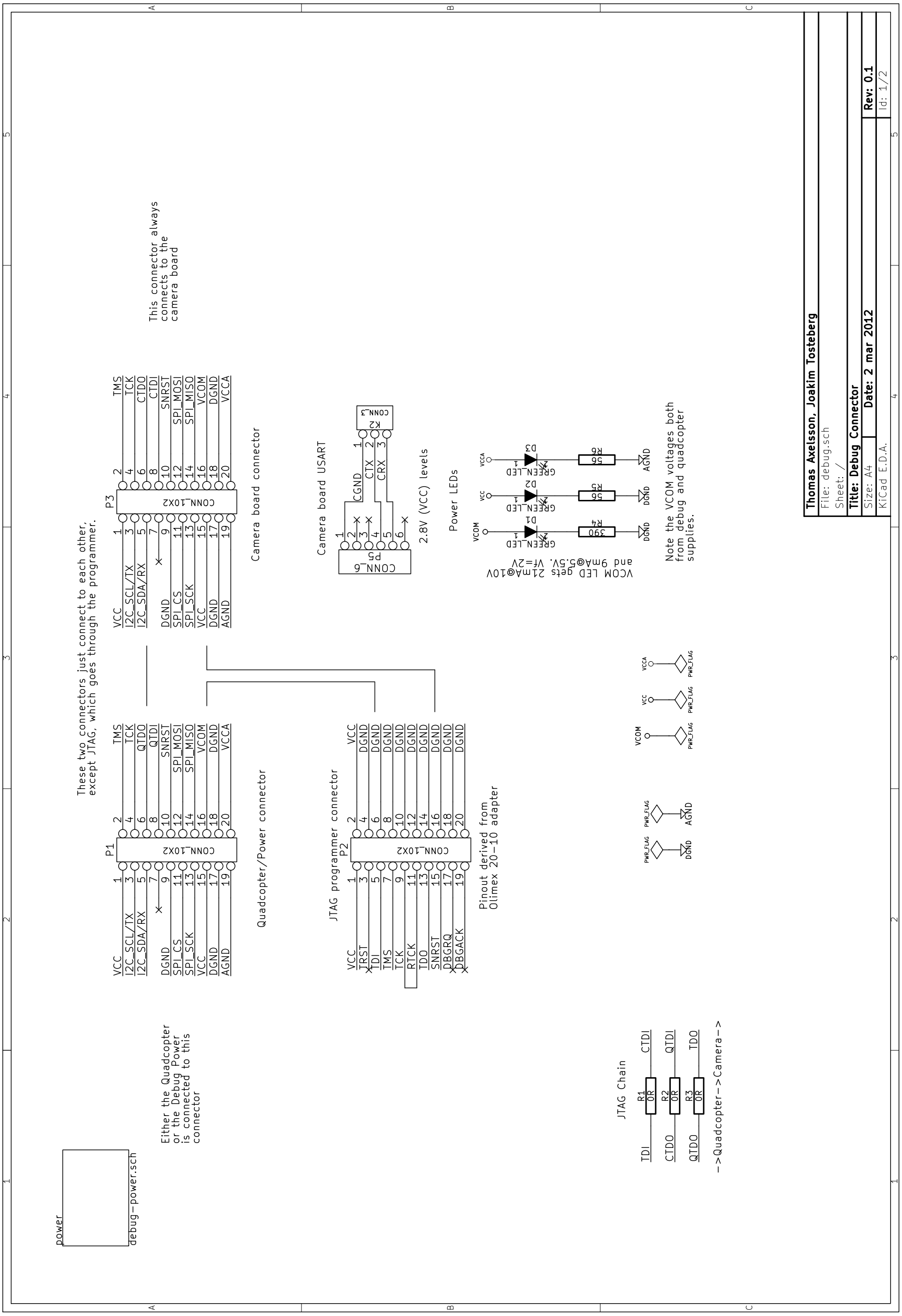
(a) Main part

Figure C.1: Schematics for camera module



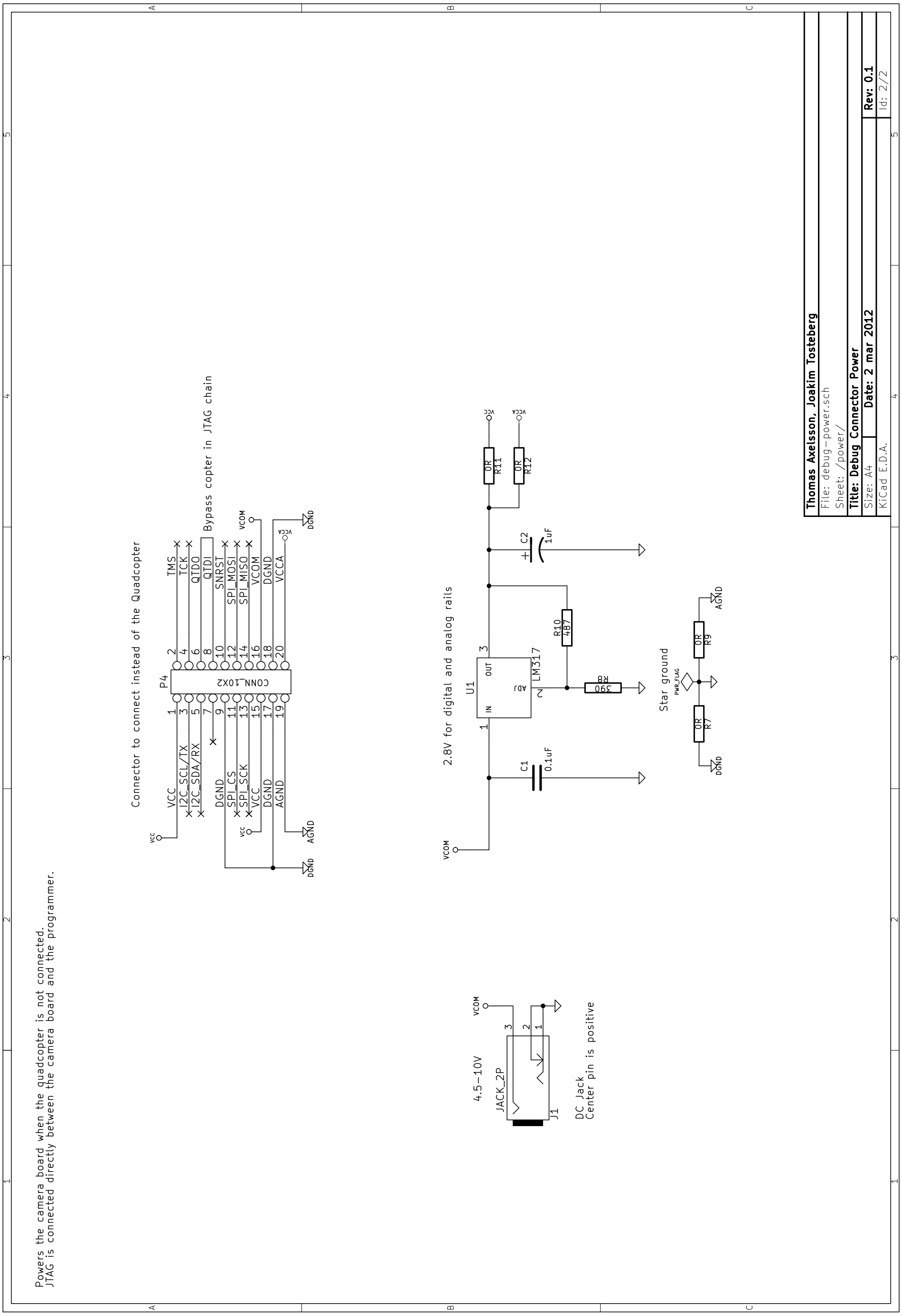
(b) Power supply

Figure C.1: Schematics for camera module



(a) Main part

Figure C.2: Schematics for debug board



(b) Power supply

Figure C.2: Schematics for debug board

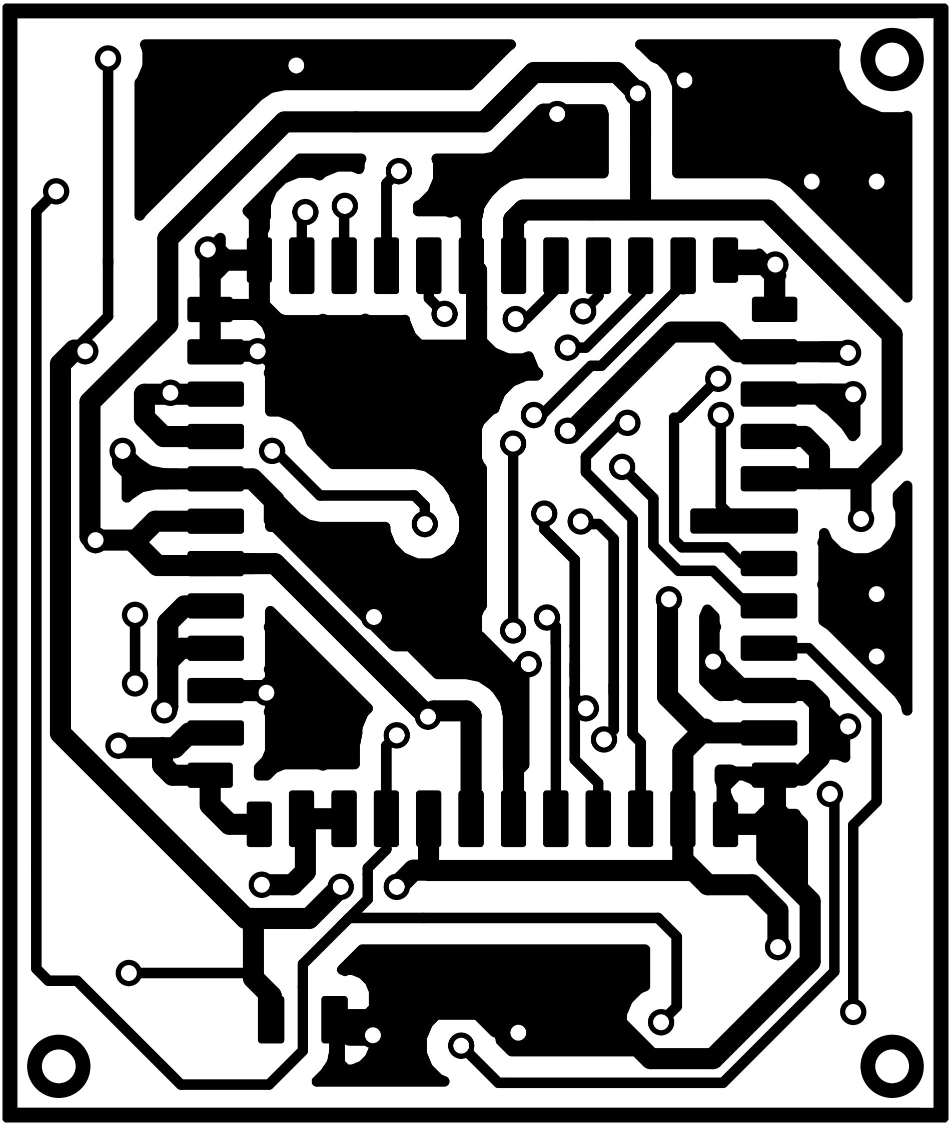


Figure C.3: Top layer copper

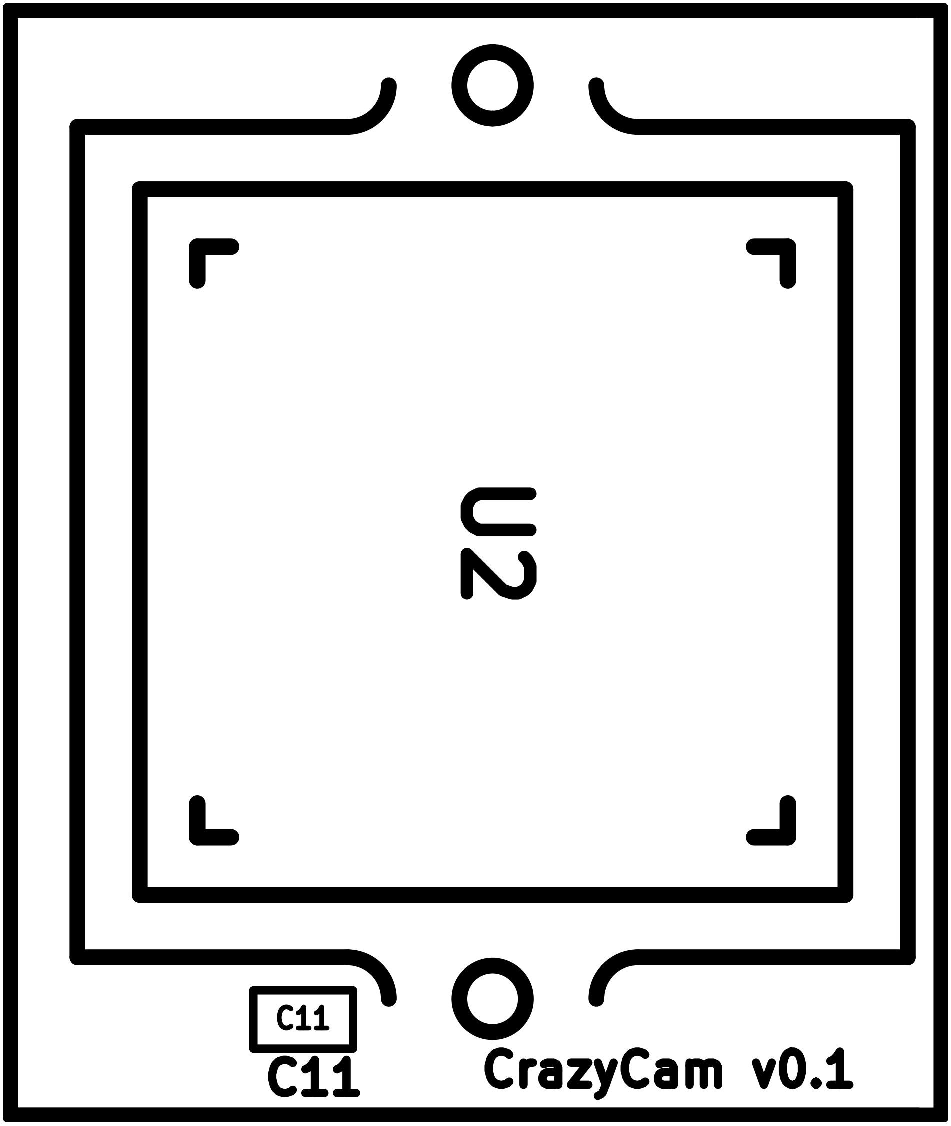


Figure C.4: Top layer silkscreen

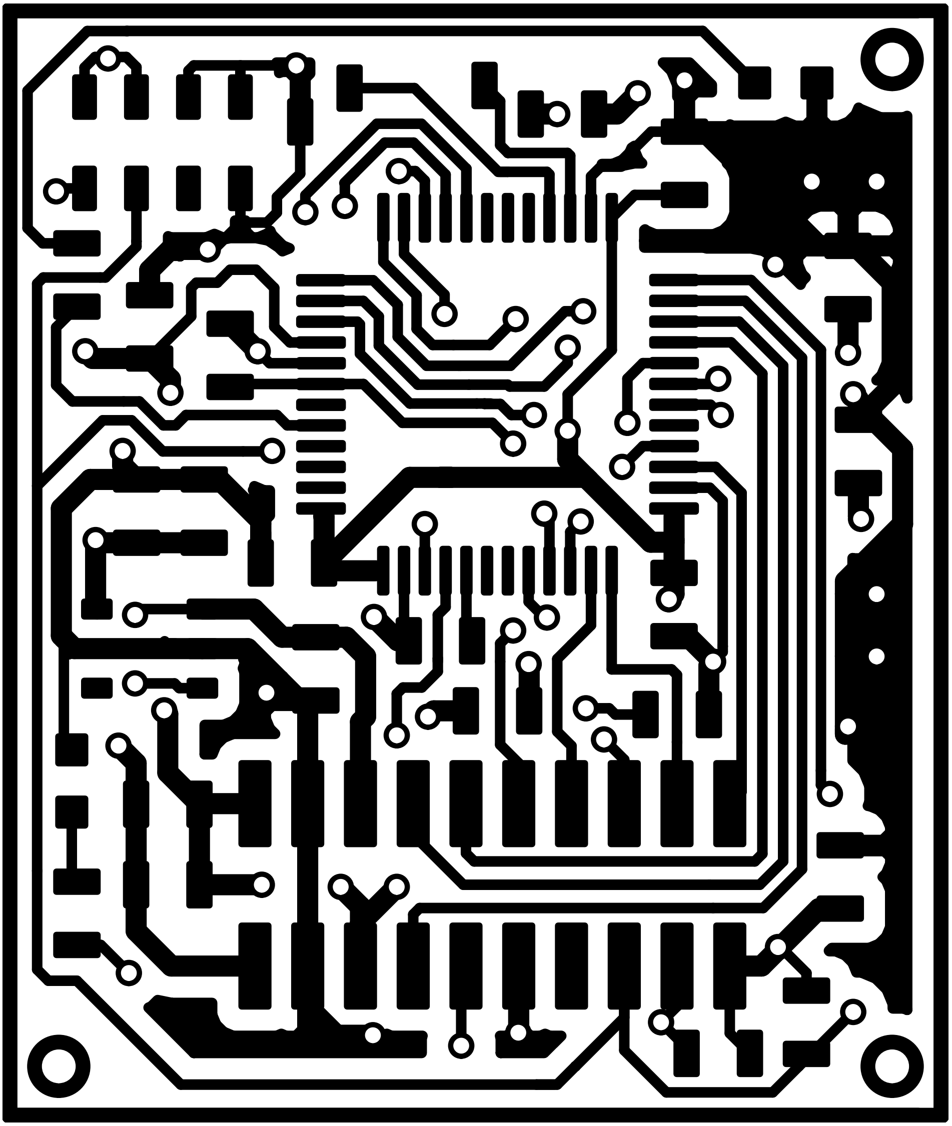


Figure C.5: Bottom layer copper

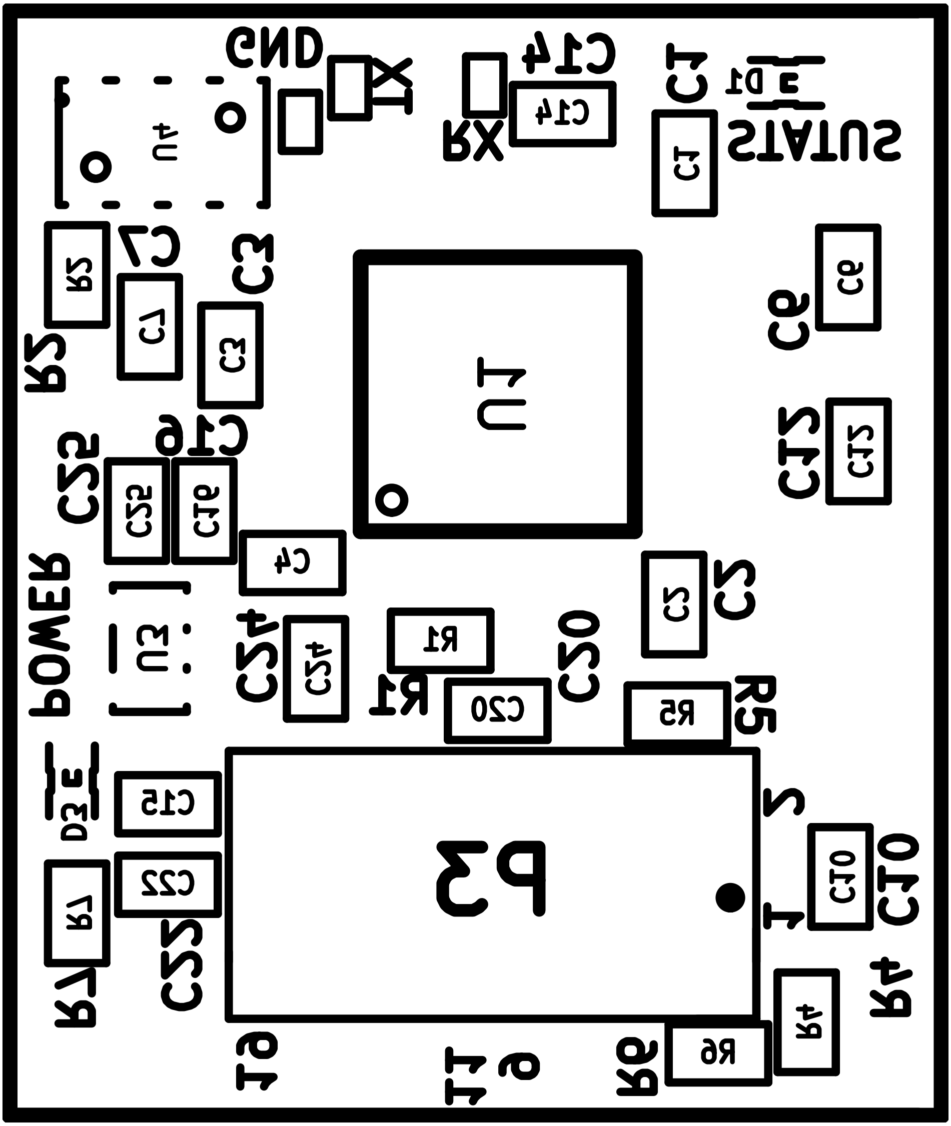


Figure C.6: Bottom layer silkscreen

Appendix D

Bill of Materials

Reference	Value	Package	Comment	Quantity
P1,P2, P3, P4		pin-header-2x10	2.54 mm pitch	4
R5, R6	56 Ω	axial		2
R4, R8	390 Ω	axial	0.25 W	2
D1, D2, D3	green	axial	LED	3
K2		pin-array-1x3		1
P5		pin-array-1x5		1
U1		TO-220	LM317AHVT	1
R10	487 Ω	axial		1
C1	100 nF	axial		1
C2	1 µF	axial		1
JACK_2P				1
		IDC-socket-2x10	2.54 mm pitch	4
		IDC-socket-2x10	1.27 mm pitch	4
		ribbon-cable	0.635 mm pitch	1
		ribbon-cable	1.27 mm pitch	1
		prototype board	1.27 mm pitch	1

Table D.1: Debug board bill of materials

Reference	Value	Package	Comment	Quantity
R1	100 kΩ	SM0603		1
R2	330 Ω	SM0603		1
R4, R5, R6	10 kΩ	SM0603		3
R7	220 Ω	SM0603		1
C1, C2, C3, C6, C7, C10, C12, C14, C15, C16, C20	100 nF	SM0603		11
C24	1 µF	SM0603		1
C4, C11, C22, C25	4.7 µF	SM0603		4
D1	red	LED-0603		1
D3	green	LED-0603		1
P3		pin-array-10x2	1.27 mm pitch	1
U1		LQFP48	STM32F103CB	1
U2		CLCC-48	MT9D131C12STC	1
U3		SOT23-5	TPS76318	1
U4		QFN8	MS5611-01BA0X	1
		cable 20x2	1.27mm pitch	1

Table D.2: Camera module bill of materials

Appendix E

UML diagrams

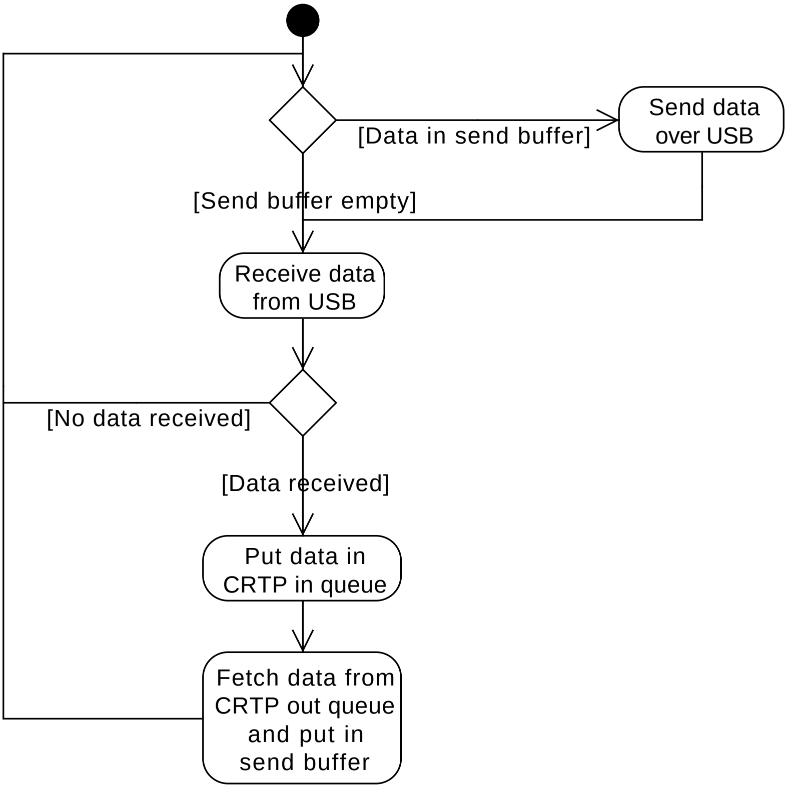


Figure E.1: Activity diagram for computer side USB code

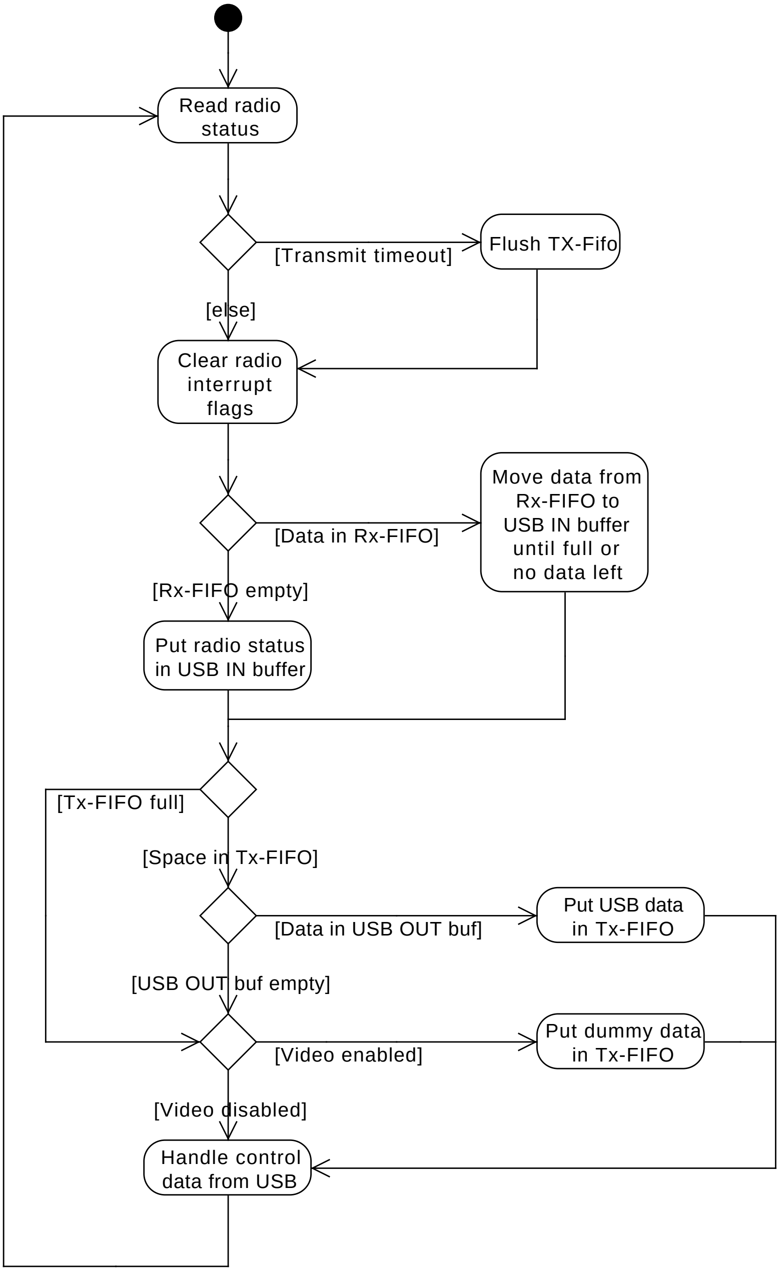


Figure E.2: Activity diagram for CrazyRadio

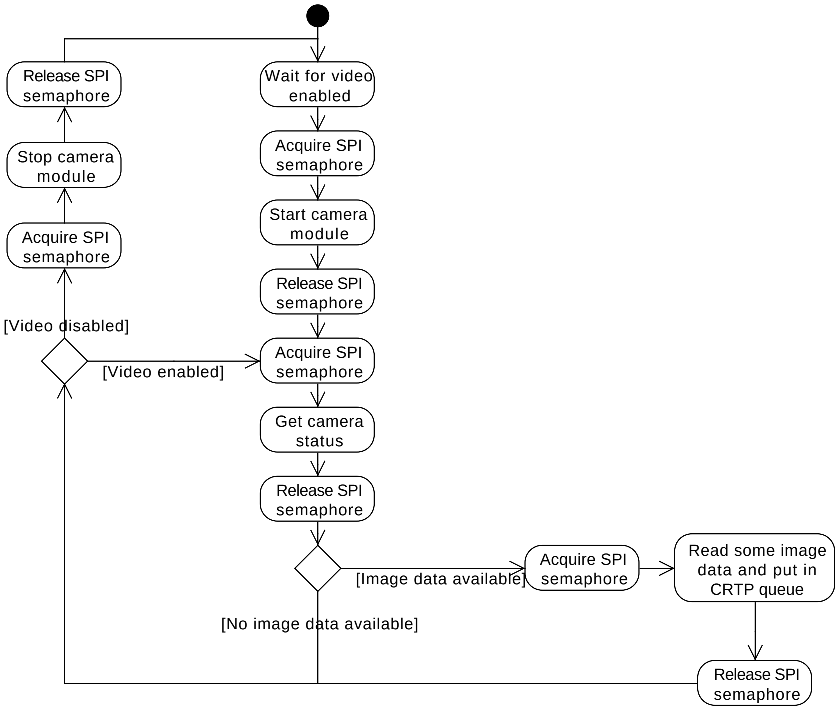


Figure E.3: Activity diagram for CrazyFlie

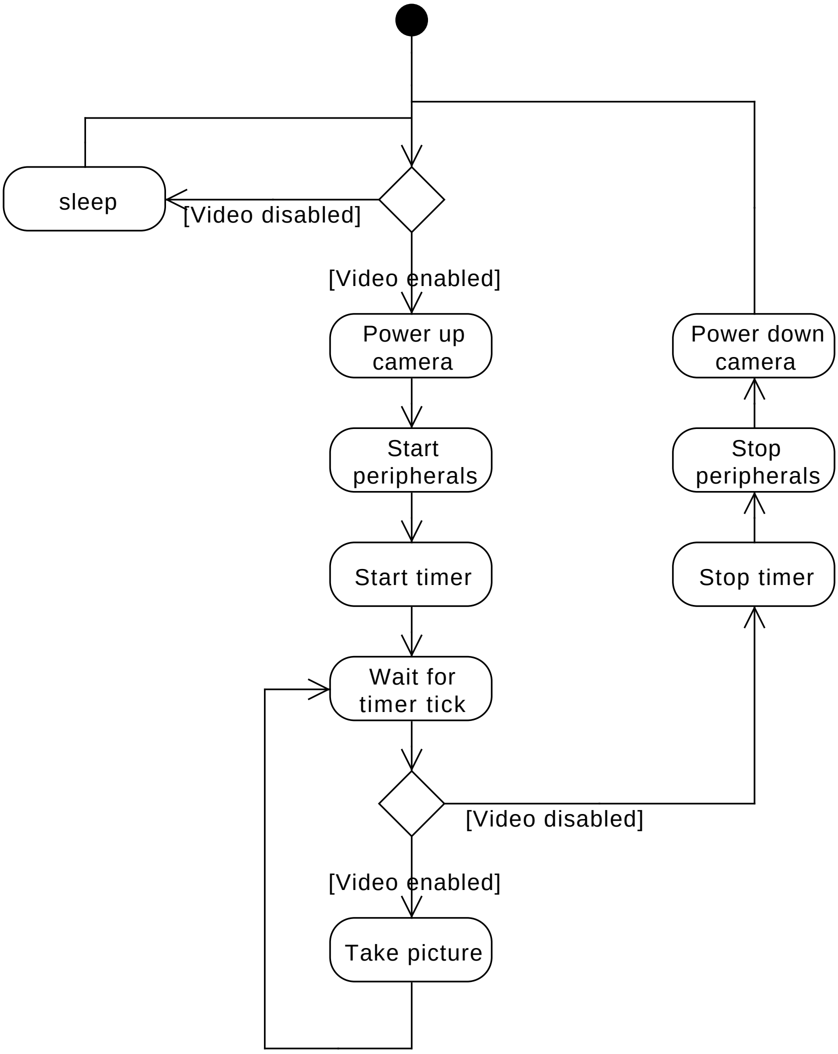


Figure E.4: Activity diagram for camera module