An Architecture with Integrated Image Processing for Autonomous Micro Aerial Vehicles

Christian Dernehl,* Dominik Franke, Hilal Diab, Stefan Kowalewski
RWTH Aachen University, Germany

ABSTRACT

This paper presents an overall MAV design with an integrated camera system. It shows the integration of the camera into the hardware and software architecture and how camera information can be used within the logical design for improving flight control. The presented architecture will be tested and evaluated during the International Micro Aerial Vehicle Conference and Competition 2011 (IMAV 2011)[1].

1 Introduction

During the last decade research and development of micro aerial vehicles (MAV) have increased and new application areas have been discovered. With the improvement of smaller and better cameras, camera systems found their way as additional and important components in MAVs. By having a camera system available on a MAV the efficiency of this air vehicle increases and new fields of applications become available. For example, this is needed in military operations, where targets have to be identified. Such an identification is often done by a human on ground, to reduce the probability of mistakes. But a camera system is also helpful if a MAV shall autonomously fly through an arch. In such a scenario camera image evaluation can be integrated into the flight control system to help navigation. In addition tiny cameras today are not only getting cheaper, but also capable of high resolution pictures. Together with improved computing power of embedded systems on-board video processing becomes possible. By having video processing on-board of a MAV the reliability of the video processing system increase, since it is not necessary to send video data via a network connection to some other device (e.g. base station) for evaluation. Further, such an approach improves the performance of the overall system, because the time and effort of sending video data to another device and receiving results can be omitted.

In this work we present an autonomous MAV system design with an integrated camera system. On the one hand we show the integration of the camera system in the hardware architecture. On the other hand we also show how to integrate the camera system into the MAVs software architecture. The camera is integrated into the logical system by introducing *sensor weighting* in the MAVs decision making process. We also indicate, how one can use the additional, and often powerful, camera system for load balancing of the overall system.

This architecture is implemented in a tilt wing MAV, which takes part in the International Micro Aerial Vehicle Conference and Competition (IMAV) 2011. During this competition MAVs have to perform specific tasks, described in the section below.

The remainder of the paper is organized as follows. Chapter 2 presents functional and non-functional demands on the presented MAV system. In Chapter 3 we list some of the related work on this area. Chapter 4 introduces the MAVs hardware architecture, which is the basis for the software architecture, presented in Chapter 5. Chapter 5 further explains the integration of the camera system into hardware and software. Chapter 6 concludes this work.

2 Requirements

There are several missions which the MAV has to accomplish in the IMAV 2011 competition. These include take-off, landing, flying through an arch, hitting a balloon, drop a listening device, record audio from the listening device, identify a vehicle and observe a group of humans. In addition to these tasks, there is an endurance mission, focusing on energy consumption and speed of the MAV. In all tasks, extra points are given for autonomy. In fact, autonomy can win the competition, since the factor 12 is multiplied to the total score if full autonomy is given, i.e. the MAV needs no instructions from humans. From these missions, requirements on the MAV can be derived.

There are various functional requirements. One goal is to have a MAV, which is capable to control itself with respect to environmental interferences, such as wind. In addition autonomy needs to be implemented to fulfill the missions. For instance autonomy means, that the MAV is capable to observe humans and depending on their actions, the MAV shall react appropriately without human interaction. A camera system is crucial with respect to the observation missions. But the camera can also be used to improve the flight control system during other tasks, such as landing. Finally safety regulations, including a safe landing in case of GPS loss, have to be fulfilled. Furthermore, an emergency system is required, allowing a human operator to operate the MAV in case the autonomous flight control does not work properly.

In addition to functional requirements , some non-functional requirements can also be derived. Only small ve-

*Email address(es): {dernehl, franke, diab, kowalewski}@embedded.rwth-aachen.de
[1]See www.imav2011.org .

hicles of a limited size are allowed. Furthermore, vehicles, which are longer than $1m$, have a disadvantage in the rating process. This implies a limited size of the MAV. Moreover, there is a maximum weight of $25kg$ and a momentum, defined as mass times speed, of $20kg\frac{m}{s}$ is allowed. As our goal is to achieve high points at the endurance mission with a high air speed, we have strict limitations to the mass of the MAV. The computation hardware components of the MAV shall only have a total weight of less than $150g$.

3 Related Work

The related work are separated into two parts. In the first part we present work related to the architecture of MAV. The second part presents related work done in the area of object recognition.

Architecture Autonomous aerial vehicles are being developed since multiple decades. In 1996, Johnson et al. [1] built an aerial vehicle capable of taking off, landing and discovering certain items with an attached camera system. In their work, the UAV transmits video data to the base station, which processes the images and sends the results back to the UAV. There is no automatic image processing component on-board of the MAV. Instead images are processed at the ground. In our approach video processing is performed on-board of the MAV, which we expect to be faster and more reliable (no transmission delays and faults).

Pastor et al. [2, 3] developed an architecture for UAVs, which includes a base station, an on-board camera system, a communication subsystem and a mission controller. The authors use the IEEE 802.3 standard[2] (Ethernet) for communication between their components. On top of the IEEE 802.3, application layer protocols are used, such as web services [4]. This architecture allows a high degree of flexibility, since components are easily exchangeable by other components implementing the suitable application protocol. In contrast to our work, an implementation of such a complex protocol stack is not necessary, since we face strong hardware constraints and do not make use of high-level protocols like Ethernet. This is due to the fact that in our project MAVs are considered, whereas the work of Pastor was applied to UAVs in general. Compared to their work our approach is on one side more low-level, but on the other side also more lightweight.

Maranhão et al. [5] designed in their work a hardware and software architecture for UAVs. The main idea is to connect components with the *Universal Serial Bus (USB)*, in which a normal PC, performing fast image processing, is the master while other microcontrollers and the camera act as slaves. Compared to our work, Maranhão focuses on the USB connection between the PC and the microcontroller, controlling the UAV. In our work we use the USB only to connect the camera module to the camera system and use the I^{2}C bus for other purposes. Our focus is on the integration of the camera system into a flight control system.

Handling multiple UAVs at the same time is the focus of the work of Tisdale et al. [6]. The authors developed an architecture containing a communication system via which the UAVs transmit data to each other. With respect to the communication, a hybrid approach was chosen, featuring communication between UAVs and data transmission to a base station. The base station assigns tasks to the UAVs. These tasks are then processed by one of the UAVs, being capable of accomplishing this mission. Our work, however, focuses on camera evaluation within one single MAV, so no other sensor data from other MAVs are available and, for video evaluation, no connection to a base station is necessary.

[2]See www.ieee802.org/3 .

Object Recognition In 2011 Chiu and Lo [7] developed a system, in which a camera module is attached to an UAV, transmitting data to a base station. The base station evaluates the video and is connected to a adapted RC receiver in a way the base station can control the UAV. With the detection of the skyline, flight control only by video evaluation is possible. Similar research has been done by Bao et al. [8], focusing primary on horizon extraction. In our work, we incorporate other sensors as well and perform video evaluation on board.

Tisdale et al. [9] used in 2008 multiple UAVs to identify and locate objects in a certain area. In their work, the authors focus on data fusion, which arises from the different available data sources and implement a particle filter, performing in their scenario better than a default Kalman filter.

Chen and Dawson [10] analyzed the tracking problem with UAVs. In their scenario, one UAVs has an attached camera and follows another UAV. The main focus of their work is, however, based on the coordinate system transformations between real world coordinates and the image planes of the UAVs.

4 Hardware Architecture

A flight control system is responsible for the stability of an aerial vehicle by reading and interpreting sensor data and controlling the engines and other actuators. Since the behavior of the environment, i.e. wind and other interferences, cannot be modeled perfectly, a closed loop control model is chosen. In this model the impact of the environment on the aerial vehicle is measured and fed back into the controller. Basically, there are three types of components: controllers, sensors and actuators. The main component is the controller itself, described in the next subsection. In the second subsection sensors are introduced, measuring the environment and transmitting the measurement data to the controller. Thereafter actuators, executing the actions of the MAV are presented. Finally in the last paragraph a summary of the architecture is provided.

4.1 Microcontroller

In our work an Arduino[3] based platform was chosen consisting of a software library, providing hardware abstraction, and hardware, the *ArduPilot Mega* board. The board can be programmed via USB, e.g. by a desktop computer. A *(FTDI)*-chip, converting USB data streams to other data streams like RS232, connects the hardware to the USB. Besides, the ArduPilot Mega board features two *Microcontroller Units (MCUs)*, an ATmega1280 and an ATmega328 from Atmel. The ATmega1280 offers 16MHz frequency and 128K RAM, while the ATmega328 provides up to 20MHz frequency and 32K RAM. The less powerful ATmega328 increases reliability by working independent of the ATmega1280 as an emergency system. If the ATmega1280 is not available due to heavy overload, the ATmega328 is still capable of controlling the MAV via forwarding incoming RC signals in a well-specified way to the corresponding actors. The ATmega1280 is used for autonomous flight control, which includes reading data from the sensors and telemetry devices and the evaluation of this data for controlling purposes. All devices within the MAV are connected by wires and various bus systems are used for communication. The ATmega1280 features four universal asynchronous receiver and transmitter (UART) devices and offers registers to attach the I^2C bus to the MCU. The ATmega328 provides one single UART.

For the camera system we decided to chose a dedicated embedded system. Our choice is the *BeagleBoard xM*[4], including a 1GHz ARM A8 MCU and 512 megabyte of RAM. Further, the Beagle Board xM features an I^2C port and an USB host chip. Having its own floating point unit the ARM A8 is wide spread for image processing.

Concluding, the airborne computer consists of three microcontrollers, an ATmega1280, ATmega328 and an ARM A8 processor. Each microcontroller is responsible for specific tasks. The ATmega1280 is the main airborne computer and designed for controlling and the decision making process. The decision making process evaluates the current sensor data, e.g. camera data, with respect to the current mission goals and chooses an action. The ATmega328 is used for backup in case the ATmega1280 fails. While both ATmega microcontrollers are responsible for controlling the MAV, the ARM A8 microprocessor has to perform different tasks, i.e. evaluating data from the camera.

4.2 Sensors

Choosing the appropriate sensors is important especially for flight control, in order to get as many reliable information about the environment as needed to follow the current mission. Sensor data is critical for the closed control loop. They include air speed, acceleration, altitude, attitude and position of the aerial vehicle. Each of these *environment variables* can be measured by a suitable sensor. A very important sensor unit is the *Inertial Measurement Unit (IMU)*, which consists of *gyroscopes*, measuring the attitude, and an *accelerometer*, measuring the acceleration. Only with the IMU already all six degrees of freedom can be determined. Additional sensors are an *air speed sensor* with an attached *pitot tube* and a *pressure sensor*, which can be used for height sensing. With respect to low altitudes, an *ultrasonic sensor* might also be used in combination with the pressure sensor to improve the height estimation. Finally a *GPS receiver* locates the aerial vehicle in space.

For this scenario the *ArduPilot Oil Pan IMU Shield*, which has also been developed within the ArduPilot project, has been chosen. This board contains gyroscopes, accelerometer, pressure sensor and a 12-bit *Analog Digital Converter (ADC)*. For instance, ADC is used to convert the sensors on the IMU Shield. The IMU board is located on top of the ArduPilot Mega board.

4.3 Actuators

After the controller has evaluated data from the sensors, the appropriate actions are performed. Data is transmitted to actuators in order to change the MAV's fly route accordingly. With respect to the type of the MAV, available actuators differ. For example, a rotary based aircraft has two actuators, motors for the main rotor and the tail rotor. On the contrary a fixed wing aircraft has two motors, a rudder, an elevator and the aileron. One result of the composition of the rotary and fixed wing concept is the *tilt wing aircraft*, which can flip its wings up to 90 degree during flight. Compared to the fixed-wing concept, there exist additionally one tail rotor and another motor, regulating the degree of the wings. Furthermore, there might be actuators for specific mission tasks, for example a chute to drop specific items.

In our scenario, only electric motors are considered, even for the main propulsion. This fact allows the usage of *Pulse Width Modulation (PWM)* to control all actuators. PWM signals can be generated by the ATmega1280 MCU. In this way the controller can directly interact with the actuators without an external, additional engine controller.

4.4 RC System and Telemetry

There are two ways to control the actuators of the MAV, the first is via the airborne computer and the second is via the RC system. The RC system operates in Europe on either 35MHz or 2.4GHz and consists of a remote and a receiver. In most cases data transmission on these frequencies is analogous. When using an airborne computer in autonomous flight mode, the RC system is used as an emergency system, allowing the operator to control the MAV in case the airborne computer is not available (e.g. due to a failure).

Since the RC system serves for manual control of the MAV and as an emergency system, other data (e.g. video data) have to be transmitted to the base station in a different way. Video streaming requires a high transmission rate and

[3]See www.arduino.cc .

[4]See beagleboard.org/hardware-xM .

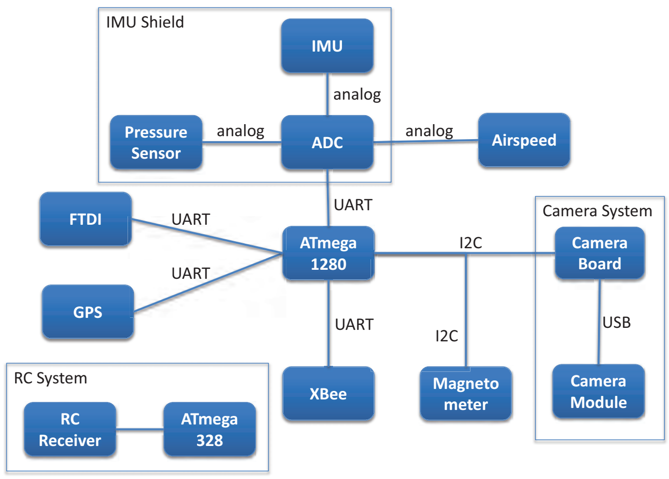

Figure 1: Hardware Architecture

is usually done via digital protocols. The *ZigBee specification* includes a protocol for digital data transfer and is comparable to other protocols like Bluetooth. In fact, Bluetooth and ZigBee reside both in the IEEE 802.15 working group. Unlike analogous RC transmission, the ZigBee specification is packet based and includes its own MAC layer, so multiple devices can be addressed. This is useful, when operating multiple MAVs with a single base station. ZigBee operates on the 868MHz, 902-928MHz, 2.4GHz frequencies.

In our scenario two *XBee modules*, operating at 2.4GHz, are provided. XBee modules implement the ZigBee specification. One module serves as the base station controlling the MAV, and one communicates with the airborne computer. The base station in this scenario is a laptop.

4.5 Overall Hardware Setup

The hardware architecture presented in Figure 1 describes the interactions and connections between the different hardware components. The essential part of the architecture is the airborne computer (ATmega1280) and the IMU board (IMU Shield). The ATmega1280, providing four UART modules, which are used for communication between two hardware components, is responsible for the manual as well as autonomous flight control. In the ArduPilot project the first UART module is assigned to communicate with the FTDI chip. This allows the developer to upload Arduino programs, i.e. programs written in the C language and linking the Arduino libraries, via USB to the ATmega1280. The second UART module is used to communicate with the GPS receiver. The third UART module is utilized to read data from sensors. Analogous sensor data is first transmitted to a 12-bit ADC, before processed by the ATmega1280. In comparison to the UART modules, supporting data transmission between two devices, the I^2C bus can handle multiple devices in a master-slave fashion. The ATmega1280 was chosen to be the master device, whereas the magnetometer and the camera system work as slave components. Note that the camera system is computationally more powerful than the airborne computer. However, the airborne computer with its access to all sensors and actuators is responsible for the decision making process. Therefore, the airborne computer was chosen as the master component.

5 Software Architecture

After providing an overview of the hardware architecture, this section describes the software architecture. The Ardupilot Mega project provides a general purpose flight control system, capable of navigating to given GPS locations. However, the software architecture needs to cover the additional camera system, as well.

5.1 Camera System

The camera system consists of two components, first a camera board for video processing and second a camera module, connected via USB to the camera board. In contrast to the ATmega MCUs, the powerful camera board is capable to execute a full Linux operating system. It has no hard disc drive, so Linux is installed on a flash card. Since a RS232 port is available, the console can be accessed via the serial interface to communicate with the operating system during the development process. The USB host chip on the camera can be accessed through the Linux USB drivers. So the developer can run software on the camera board, which reads RGB frames from the camera module. For image processing a software module is implemented, which takes advantage of the OpenCV library. Image processing in this context includes the recognition of objects and deriving from this a flight control suggestion, which is sent to the ATmega1280.

For this scenario a camera module with 30 frames per second with a resolution of 640x480 pixels is used. The computer vision algorithms of OpenCV can process 7 frames per second on our hardware, if color frames are evaluated. Instead of calculating color images, about 15 greyscale images can be calculated in a second. Evaluation means here that items, e.g. an arch, within a taken picture are recognized and their position in the picture determined (x- and y-coordinates in pixels relative to the image border). Assuming a cruise speed of about $10\frac{m}{s}$ this allows 0.7 frames per meter in color mode or 1.5 frames per meter in greyscale mode.

5.2 Airborne Computer

The software of the airborne computer is based on the Arduino Mega platform, which again is based on the open source Arduino platform. Next to the introduced hardware the Ardupilot Mega provides also various libraries, e.g. for autonomous stabilization and GPS navigation.

The airborne computer in our setup consists of a telemetry module, the flight controller, mission controller and is influenced by the camera system. Further, it has access to all sensors and actuators (see Figure 2). The camera system transmits flight control suggestions to the mission controller. This

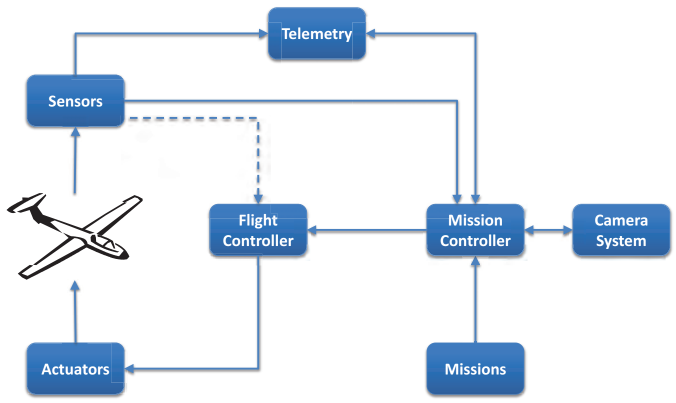

Figure 2: Software Architecture

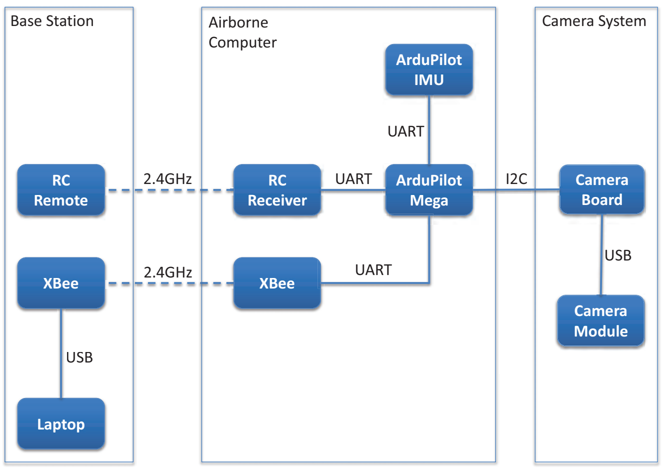

Figure 3: Overall Architecture

is explained in detail in the next subsection. Consider the MAV approaching a balloon. The coordinates of the balloon are fixed and known, stored in the mission database. Then together with the IMU and GPS the flight route can be set accordingly. However, when the MAV is very close towards the balloon the frequency of data measured from the IMU and GPS might not be sufficient to hit a balloon with a diameter of 60cm flying in about 10 meter height. From then on, the MAV can fly assisted by the camera system. Additionally, plausibility tests have to be performed on the mission controller to check the reliability of the camera suggestions or to detect a failure in the camera system. For instance, the camera system suggests to fly lower to hit the balloon. Then the mission controller checks, if this can be confirmed by the current IMU and GPS data (e.g. flight direction and last GPS coordinates indicate that the MAV is flying towards the expected GPS position of the balloon). Furthermore, the mission controller is capable of turning the camera system on or off, e.g. for energy efficiency reasons if the camera is not needed in a mission.

Missions are stored in a database, attached to the mission controller. In this way delays, resulting from loading mission data, are minimized and do not interfere or depend on the current communication load. Mission data is also used by the mission controller for further improvements of the flight control system. For example, in some missions certain sensors are more important to flight control than others. The sensor importance can be expressed by a weighting function, assigning a weight to each sensor before the decision making process (e.g. which actuator is activated) starts. Therefore, the airborne computer can behave differently in each mission. Consider a mission to land the MAV and another one to observe an area. During landing the altitude is crucial, whereas during observation, the exact altitude is of lower importance, but camera data becomes significant. Therefore, in a landing mission, the altimeter and ultrasonic sensors have a higher weight than the camera.

In the autonomous mode the mission controller, knowing what the next task is, works together with the flight controller, which has access to all actuators and sensors (details in next subsection). Since the flight controller implements a closed control loop, current state values are read from the sensors and output data is transmitted to the actuators. The telemetry module allows the user to set the current state of the mission controller, e.g. skip mission x and proceed with mission y.

The decoupling of the mission controller from the flight controller increases, by encapsulation, the reusability of the software modules. Consider the type of the MAV changes from fixed wing aircraft to rotary wing aircraft. In this case the flight controller module needs to be replaced with an according module. The mission controller, however, does not need any additional changes.

In Figure 3 the overall structure is depicted. On the left side, the system used for the base station is illustrated. Signals are transmitted either with the RC remote or the XBee module to the airborne computer. The airborne computer is attached via the I^2C bus to the camera system. For interconnections within the airborne computer UART is used, except for the magnetometer, which is not included in the illustration.

5.3 *Integration of the Camera System into the Airborne Software Architecture*

In this chapter we explain the interaction of the camera system with the airborne computer in detail. If the current mission, chosen by the mission controller, needs the camera system, it will be switched on by the mission controller. Further, the mission controller tells the camera system, which object shall be recognized by sending an object ID. For this purpose a set of corresponding objects with IDs are already specified in the flash storage of the camera system. The camera system is then responsible for recognition, processing and evaluation of the object (e.g. with OpenCV, as ex-

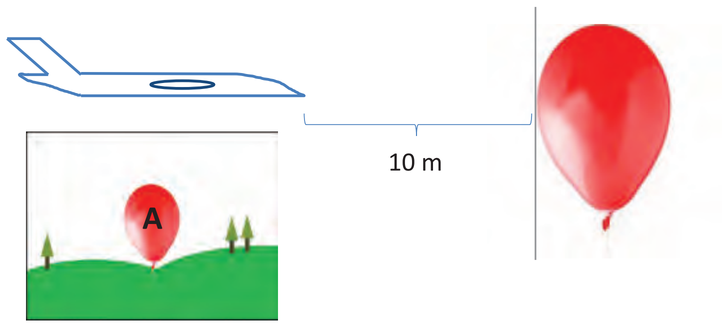

Figure 4: Calibration of Object Size

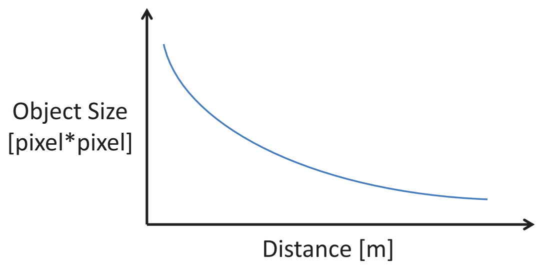

Figure 5: Relation between Object Size and Distance

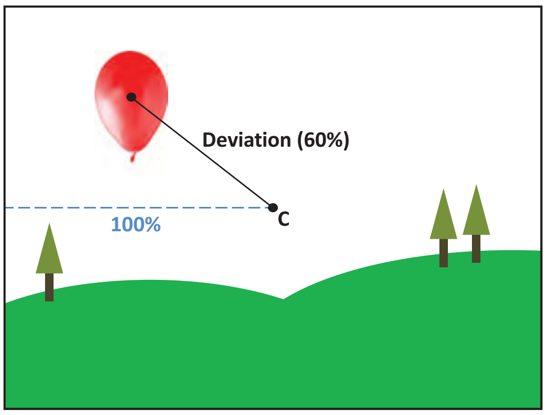

Figure 6: Computation of Deviation

plained above). Finally evaluation results has to be send to the mission controller, which is capable of interpreting them as flight suggestions. The interpretation in the mission controller is dependent on the current flight mode (manual or autonomous), the current mission (does camera play a role?) and reliability and relevance of the camera system information (do GPS and IMU also confirm, that the MAV is approaching the balloon position?).

To get a better idea of this concept, we take a close look on the communication protocol between the camera system and the airborne computer. Our communication protocol between these two systems contains the following elements:

Status Bit This bit is set to 0, if the camera system does not recognize the wanted object. It has the value 1, if the object is recognized by the camera system. Only in this case, the other three parameters are available to the mission controller.

Size of Object This parameter contains the size of the recognized object, which is needed in the mission controller to estimate the distance to the object and derive from this the intensity of future actions, like adjusting the route.

The size is given in a relative manner. In a calibration process on the ground each object is placed exactly 10 meters ahead of the MAV. The object is then recognized by the MAV and the area that the object takes on the camera images during this recognition is defined as 100%. Figure 4 shows the setup during calibration and recognition of a balloon. In our project the balloon recognition is based on color and shape features. The balloon area is labeled *A*. We chose area as measurement unit, since it can be fast computed with our camera system. A distance of 10 meters is chosen with respect to the flight speed and size of the available objects during the IMAV competition. If an object is closer than 10 meters to our flying MAV, then reacting on corresponding images becomes difficult as less than one second is left to pass the object in fixed wing mode. But values greater than 100% are possible as well. This is the case, if the object is closer than 10 meters.

We chose a relative measure to keep the encapsulation of our software and hardware modules high. For instance, if a camera module is replaced, then no adjustments have to be made to the mission controller, since the relative value will not be changed. In case of a camera replacement only the calibration on ground has to be repeated to define the size of 100% area of each object and store this value in the camera processing hardware.

Concluding, the size of the recognized object, as area A on the camera picture, is passed as a relative value to the mission controller. The mission controller has a model containing the size of objects relative to distance. The model is sketched in Figure 5. This model is based on the fact that size of objects recognized on camera images, relates exponential to the distance of the objects. The current size of the object together with the size of the object during calibration, are used by the mission control system to derive the current distance to the object. Based on this knowledge the strength of the necessary action (e.g. steer hard or soft left) is computed by the mission control algorithm.

Deviation Another parameter passed by the camera system to the mission controller is the deviation of the recognized object in the camera image from the center of the image.

Figure 6 sketches the computation of the deviation parameter. It is computed as the distance between the center of the camera image and the center of the recognized object. We

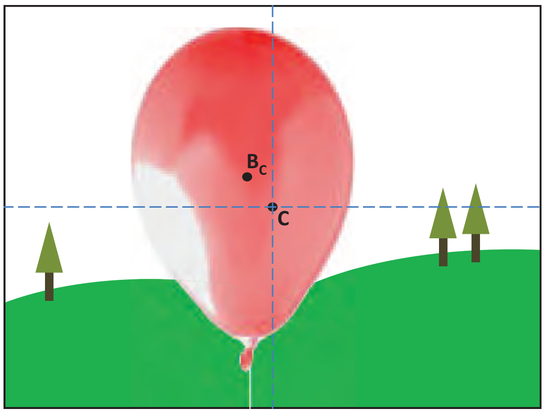

Figure 7: Close to Object

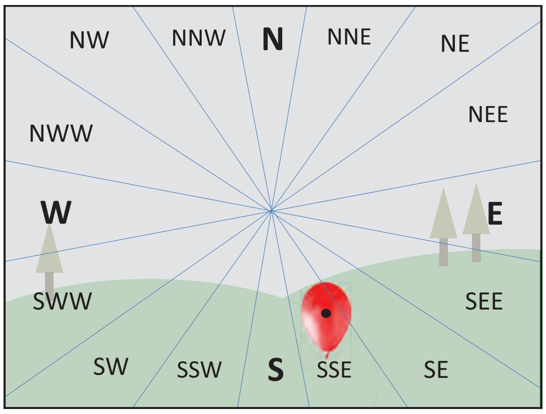

Figure 8: Identification of Direction Parameter

again pass a relative value to the mission controller, for the same reason as with the size of the recognized object. For the deviation 100% are defined as the distance from the center of the image to the left and right border of the image (see Figure 6). The deviation parameter is larger than 100%, when the center of a recognized object is located one of the corners of the image, since the diagonal of a rectangle is larger than its width. Together with the parameter *size of object*, the deviation parameter is used within the mission control algorithm to compute how hard a corresponding reaction of the flight controller has to be executed. For instance, if the mission controller detects with the parameter *size of object*, that it has a large distance to the recognized object (as explained above), and the deviation parameter has a value of 80%, then, although deviation is large, the mission control algorithm instructs the flight controller for a soft reaction, since the distance to the object is large.

If the center of the camera image is within the recognized object (see Figure 7), deviation is likely to be still greater than zero (figure 7, distance between center of image C and center of recognized balloon object B_C). This is due to the fact, that it is within fixed wing flight mode and some certain speed improbable that the center of the recognized object remains exactly on the center of the camera image. Therefore, the mission controller additionally has for each stored object, depending on its size, a certain deviation tolerance. This means, if it for example approaches the balloon, for which it computes that the distance is about 12 meters, and the deviation is 5%, no reaction is necessary. As another example, if it approaches orthogonally an arc, with a width of 10 meters and a height of 5 meters, and computes out of the previous parameters a left distance of 12 meters, then even a deviation of 30% does not imply any reaction, since the arc is large enough to pass properly through with the current route.

Direction The fourth parameter passed by the camera system to the mission controller is a suggestion, to which direction the plane should adjust its route. Therefore, we divide the camera image to different areas, as presented in Figure 8. The center of this pattern complies with the center of the camera image. The different areas are named by cardinal points (N = north, NNE = north north east, NE = north east, NEE = north east east, ...). To keep the protocol between the camera system and airborne computer short and high-performance these abbreviations are passed from the camera system to the mission controller.

For instance, if a balloon is recognized in the bottom right corner of the camera image (see Figure 8), then depending on the area, in which the center of the balloon is placed, the corresponding abbreviation is passed to the mission controller. The mission controller interprets this direction parameter as a flight suggestion. In case of the balloon, the corresponding flight suggestion would be SSE, meaning that the flight route has to be adjusted down and a bit right.

With the first three parameters the plane knows if an object is recognized (status bit), what the distance to the object is (derived from the size of the object) and the deviation of the object from our current flight route. So the mission controller can compute if a reaction is needed and the strength of corresponding actions. However, it has no information about the direction, in which the action shall be applied. With the direction parameter the mission controller gets a suggestion, in which direction the flight route has to be adjusted. Depending on the current speed of the MAV and the distance to the target, the mission controller decides, if an action is performed, or not.

With these parameters from the camera system together with sensor data (position, orientation, speed), the model of size-distance relation (see Figure 5) and the current mission objectives the mission control system is able to autonomously control the MAV via the flight control system.

6 Conclusion

This work presents an overall MAV architecture with a completely integrated camera system. We present the integration on hardware as well as on software layer. While the integration on hardware layer is often given by the available connection types, the software architecture integration can become pretty challenging.

For the communication between the camera system and the airborne computer, we introduce a modular and lightweight protocol. The structure of this protocol consists of four elements: status bit, size of object, deviation and direction. All four values are computed within the camera system. We show how this parameters can be used in the mission control system together with the flight control system, to integrate image processing into autonomous flight control.

By introducing a modular and lightweight protocol for the communication between the camera system and airborne computer, we also clearly separate the tasks of these two modules. We separate the following tasks (on software layer):

1. image recognition
2. image processing
3. image evaluation
4. deriving flight suggestions based on the evaluated images
5. sensor weighting
6. and decision making

Such an encapsulated approach has different benefits on hardware as well as on software layer. On hardware layer, if the camera itself has to be replaced, then only the modules 1 and possibly 2 have to be adjusted. Furthermore, image processing hardware, like the BeagleBoard xM, is often more powerful than the flight control hardware (here Atmega1280). This is why we execute tasks 1-4 on the BeagleBoard xM and tasks 5 and 6 on the Atmega1280. Such a load balance can even be enhanced by outsourcing even more tasks from the flight control hardware to the video hardware (e.g. sensor fusion). This depends on the overall architecture of each single MAV system.

Next to load balancing our approach also increases the safety and reliability of the MAV. If the BeagleBoard xM crashes due to complex tasks or heavy load, this has just a small effect on the separated flight control system. In our case the Atmega1280 will simply notice, that one sensor, namely the camera system, is not available any more.

As future work of this project the evaluation of this presented approach in different flight competitions like the IMAV 2011 is planned.

Acknowledgements

This work has been supported by the UMIC Research Centre, RWTH Aachen University, Germany.

References

[1] E.N. Johnson, P.A. DeBitetto, C.A. Trott, and M.C. Bosse. The 1996 MIT/Boston University/Draper Laboratory Autonomous Helicopter System. In *Digital Avionics Systems Conference, 1996., 15th AIAA/IEEE*, pages 381 –386, oct 1996.

[2] E. Pastor, J. Lopez, and P. Royo. An Embedded Architecture for Mission Control of Unmanned Aerial Vehicles. In *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*, pages 554 –560, 0-0 2006.

[3] E. Pastor, J. Lopez, and P. Royo. UAV Payload and Mission Control Hardware/Software Architecture. *Aerospace and Electronic Systems Magazine, IEEE*, 22(6):3 –8, june 2007.

[4] Francis McCabe, David Booth, Christopher Ferris, David Orchard, Mike Champion, Eric Newcomer, and Hugo Haas. Web Services Architecture. W3C Note, W3C, February 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[5] D. Maranhao and P. Alsina. Project of a Hardware and Software Architecture for an Unmanned Aerial Vehicle. In *Robotics Symposium (LARS), 2009 6th Latin American*, pages 1 –6, oct. 2009.

[6] J. Tisdale, A. Ryan, M. Zennaro, Xiao Xiao, D. Caveney, S. Rathinam, J.K. Hedrick, and R. Sengupta. The Software Architecture of the Berkeley UAV Platform. In *Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE*, pages 1420 –1425, oct. 2006.

[7] C.-C. Chiu and C.-T. Lo. Vision-only automatic flight control for small uavs. *Vehicular Technology, IEEE Transactions on*, 60(6):2425 –2437, july 2011.

[8] Gui-Qiu Bao, Shen-Shu Xiong, and Zhao-Ying Zhou. Vision-based horizon extraction for micro air vehicle flight control. *Instrumentation and Measurement, IEEE Transactions on*, 54(3):1067 – 1072, june 2005.

[9] J. Tisdale, A. Ryan, Zu Kim, D. Tornqvist, and J.K. Hedrick. A multiple UAV System for Vision-based Search and Localization. In *American Control Conference, 2008*, pages 1985 –1990, june 2008.

[10] Jian Chen and D.M. Dawson. UAV Tracking with a Monocular Camera. In *Decision and Control, 2006 45th IEEE Conference on*, pages 3873 –3878, dec. 2006.