Basic Project Intelligent Systems
Cognitronics and Sensor Systems
Faculty of Technology
Bielefeld University

Implementation of a Communication Interface for a Flight Controller and two Example Applications for the JeVois Camera with an AMiRo

Marc Rothmann, Victoria Buchholz
Supervised by Timo Korthals

Bielefeld, June 2018

Contents

1 Introduction

This project report documents our implementation of a communication interface for a flight controller (FC) as well as the development of two example applications for a mini camera with a small mobile robot. The original idea of the project was to develop the example applications with the camera and a drone. The first step consisted in optimizing the communication interface of the FC of the drone. This is described in detail in section 3. Due to a delay of another project with the drone, we were not able to continue the project with the drone and therefore switch to the mobile robot. This was easily possible as the drone is based on the modular structure of the mobile robot.

In the following, we first describe the mobile robot, called AMiRo, shortly. After that, we explain the optimization of the communication interface of the FC and the recording of data with it. Then, we outline the implementation of the two example applications with the mini camera and the mobile robot. This part also contains an evaluation of the second application. Finally, we give a short conclusion of the whole project.

2 Autonomous Mini Robot

The Autonomous Mini Robot (AMiRo) is a small mobile robot developed for research and education at Bielefeld University. The robot itself is round with a diameter of 10 cm and it has a height of7.6 cm without extensions. Its structure is modular. Thereby, the basic modules are the DiWheelDrive, the PowerManagement and the LightRing. In this project we also used the cognition board as an extension module. The cognition board is equipped with a linux operating system. The AMiRo modules communicate via a Controller Area Network [2].

3 Flight Controller

The Seriously Pro Racing F3 EVO (SPRacingF3EVO) Flight Controller (FC) is built into our drone. It uses an STM32F3 microcontroller and an MPU9250 inertial measurement unit, among other peripherals.

To work with the FC we used an open source firmware called *BetaFlight* [1]. This FC software is used to fly different types of multi-rotor aircrafts, e. g. drones, and fixed wing aircrafts. It is based on *Cleanflight*.

3.1 Character and Byte Interface

At the beginning of the project only a character interface for the communication between the FC and the AMiRo existed. It was assumed that this kind of communication interface is quite slow and an evaluation confirmed the assumption. Thus, we implemented a byte interface in order to improve the system's real-time capabilities. For the character interface, each command is encoded as a string. In contrast to that, the byte interface encodes the commands as single bytes. Hence, messages sent via the character interface are longer and the transmission takes more time. An evaluation revealed that the new interface enables a data transfer that is 15 times faster as the old one. With the byte interface we are able to send one request within one microsecond.

For the sending of bytes we used the already existing function *cliPrintf* of the operating system *Betaflight*. However, it is only possible to send strings ending with a $\backslash n$. Additionally, strings are not allowed to contain zeros. In order to circumvent this problem, we decided to use a Base85 encoding. That way, we can clearly mark the end of a message and trigger the transmission at the same time.

The first byte of the command message specifies the type of command. The last two bytes contain the value to be sent in big endian format. To be able to use Base85 encoding we added an empty byte between the bytes for the command and the data.

```
message format: cmd | 0x00 | data (high byte) | data (low byte)

typedef enum {
    CMD_ROLL        = 0x0,
    CMD_PITCH       = 0x1,
    CMD_YAW         = 0x2,
    CMD_THROTTLE    = 0x3,
    CMD_AUX1        = 0x4,
    CMD_AUX2        = 0x5,
    CMD_AUX3        = 0x6,
    CMD_AUX4        = 0x7,
    CMD_AUX5        = 0x8,
    CMD_AUX6        = 0x9,
    CMD_AUX7        = 0xA,
    CMD_AUX8        = 0xB,
    CMD_GET         = 0xC,
    CMD_GET_VALUES  = 0xD
}
```

The disadvantage of this encoding process is that we need to send approximately 25% more bytes. This, however, can be neglected since it is still possible to send 100 bytes per message with the current baud rate and a transmission frequency of 100 Hz.

In order to prevent unnecessary polling, we implemented a thread that constantly sends sensor data from the FC to the AMiRo. This thread runs with a low priority to avoid impairing other functions.

```
typedef union {
    uint8_t data[44];
    struct {
        int32_t accSmooth[3];
        int32_t gyroADC[3];
```

```
        int32_t baroAlt;
        int32_t baroTemp;
        int32_t magADC[3];
    } values;
} sensorData_t;
```

3.2 Data Recording

We used this interface to record data from the FC together with the ground truth data from the Teleworkbench (TWB) with the drone. The FC data consisted of sensor values from the inertial measurement unit, barometer and magnetometer data. The TWB data consisted of translation and rotation values in x, y and z direction. We merged the data into a JSON file and assigned multiple FC data to one time stamp from the TWB as we recorded the data from the FC with a higher frequency. The recorded data is going to be used for the implementation of a Kalman filter and a controller for the drone in another project.

4 JeVois Camera

For the main part of our project we worked with an open source mini-camera called JeVois. It is equipped with a video sensor, a quad-core CPU and a dual-core GPU. The camera's output is streamed to a computer connected via USB. It provides visual output as well as text output via a serial port. The camera already comes with a number of open source machine vision algorithms ready to use [3].

We used the JeVois Camera, mounted on top of an AMiRo (see Figure 1), for two different test cases: 1. controlling the distance to ArUco markers, and 2. collision avoidance with optical flow. In both test cases we used the original AMiRo. We implemented both programs for the Linux operating system running on the AMiRo cognition board. Thus, we used the Controller Area Network to set the speed of the different wheels.

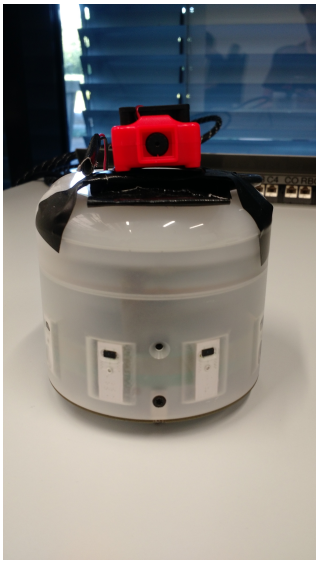


Figure 1: The JeVois camera mounted on top of the AMiRo.

4.1 ArUco Marker Tracking

To control the distance of an AMiRo to an ArUco marker, we used an existing JeVois module to track the position of the marker. To accomplish that, we implemented a program for the AMiRo that interacts with the JeVois camera to start the ArUco tracking module. We used the following sequence of commands:

```
streamoff
setpar serout USB
setmapping 4
setpar serstyle Normal
setpar dopose True
setpar markerlen 50
streamon
```

The received serial messages from the ArUco module of the JeVois camera have the following format:

```
N3 id x y z w h d
```

Figure 2: An example output image of the optical flow computation of the JeVois camera.

After parsing these messages, we first applied a low pass filter to the value of the z coordinate, which indicates the distance between the marker and the camera, to reduce noise in the tracking data. Subsequently, we used a P-controller to adjust the robot's speed according to the distance to the marker.

4.2 Obstacle Avoidance with Optical Flow

A variety of machine vision algorithms are included in the JeVois camera that are ready to run from the micro SD card image. Among them an algorithm that computes the optical flow using dense inverse search. The existing module for the optical flow only supported the output of the result as an image. As the output image contains the horizontal as well as the vertical motion map, it is twice as tall as the camera input image (see Figure 2). In both maps mid-grey values represent no motion. Furthermore, in the horizontal motion map lighter shades represent leftward and darker shades rightward movement. Accordingly, in the vertical motion map lighter shades represent upward and darker shades downward movement.

In order to be able to send the result of the computation of the optical flow via serial messages to the AMiRo, we added a text-based output. Moreover,

to facilitate further processing of the optical flow on the AMiRo, we already calculated the vector pointing towards the closest object on the camera. This vector and further calculations on the AMiRo were made according to the equations proposed by Meyer et al. [4].

Taking the output of the optical flow algorithm, we first create a relative nearness map μ_r. The values in this map represent the "contrast-weighted relative nearness to objects in the environment" [4, p. 172] (see Equation 1). In the Equation 1 the variables v and h stand for the vertical and horizontal optical flow at the position (x, y).

$$\mu_r(x, y) = \sqrt{v_{EMD}^2(x, y) + h_{EMD}^2(x, y)} \tag{1}$$

This map is then used to calculate the center-of-mass-average-nearness-vector ($COMANV$). As can be seen in Equation 2, the $COMANV$ is the sum of vectors dependent on the azimuth angles, weighted by the sum of average nearness values for the respective azimuth angle.

$$COMANV = \sum \left(\begin{pmatrix} \cos(\Phi) \\ \sin(\Phi) \end{pmatrix} \frac{1}{n} \sum \mu_r(\epsilon, \Phi) \right) \tag{2}$$

After receiving this $COMANV$ via serial messages on the AMiRo, we compute the collision avoidance direction (CAD, see Equation 3) and the collision avoidance necessity (CAN, see Equation 4).

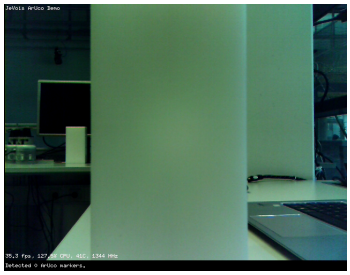
$$CAD_{fov} = \frac{-\arctan(COMANV_y, COMANV_x)}{\frac{2\pi}{\Theta}} \tag{3}$$

$$CAN = ||COMANV|| \tag{4}$$

Next, the heading direction γ (see Equation 5) is computed as the mean of the collision avoidance direction and the direction to a goal, weighted by the output of a sigmoid function based on the collision avoidance necessity (see Equation 6).

$$\gamma = W(CAN) \cdot CAD_{fov} + (1 - W(CAN)) \cdot \alpha \tag{5}$$

(a) Camera downwards



(b) Camera horizontal

(c) Camera upwards

Figure 3: Example images with three different camera angles.

$$W(CAN) = \frac{1}{1 + (\frac{CAN}{n_0})^{-g}} \tag{6}$$

Finally, as the computed values for γ were to small to have an effect on the AMiRo's heading direction, we multiply it with a constant value and send it via the Controller Area Network to the DiWheelDrive module.

4.2.1 Evaluation

At first, we tested different camera angles to see if different angles have an effect on the results of the optical flow computation. Hence, we directed the camera downwards, upwards and straight ahead (see Figure 3). We obtained the worst results when directing the camera downwards. A possible explanation is that light, shadow and line on the floor are interpreted as optical flow. To some extent this also happened when directing the camera straight ahead as the floor is still visible in this setting. Furthermore, we

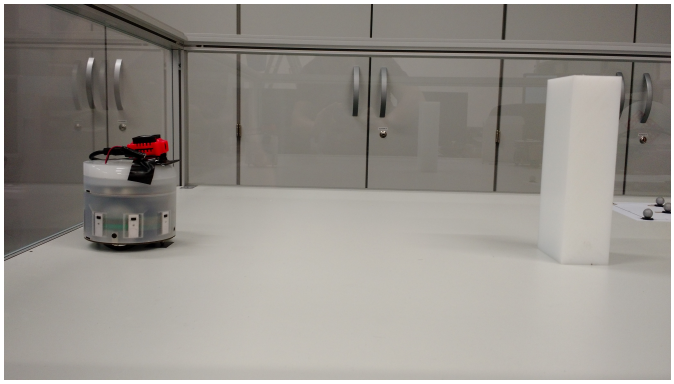


Figure 4: Setting for obstacle avoidance evaluation.

assume this to be the reason why directing the camera upwards provided the best results. Thus, in the evaluation of the parameters used in the collision avoidance algorithm we used this setting.

The algorithm includes two free parameters to be chosen, the gain g and the threshold n (see Equation 6). To find a suitable set of parameters, we did a grid search with five different values for each parameter. To be able to compare our results to the ones presented by Meyer et al. [4] we used values in the same range as in their evaluation. Our test setting consisted only of one block that was positioned in front of the robot in the same distance in every trial. At the beginning of each trial we set the robot at a specified point facing the block and observed its behavior (see Figure 4). As a measure of the algorithm's quality we counted the number of successes, that means no collision with the object, in 10 trials. As can be seen in Table 1 no significant differences were observed in the test trials, except for the fact that the robot's behavior was worse when the gain g was set to zero. Overall, the results have been good for the majority of the parameters.

Furthermore, we examined the energy consumption of the JeVois camera with and without the optical flow algorithm running as well as the energy consumption of the fan. The consumption of the camera while the OF algorithm was not running amounted to 69 mAh while it amounted to 129 mAh with the algorithm running. Furthermore, the fan consumed 30 mAh.

g/n	0.0	12.5	25.0	37.5	50.0
0.0	6	5	6	6	4
0.5	9	10	8	7	10
1.0	10	9	6	10	10
1.5	9	10	9	8	10
2.0	9	10	10	9	9

Table 1: Evaluation of different values for parameters n (threshold) and g (gain). Number of successes in 10 trials.

5 Conclusion

An efficient and easy to use communication interface for a flight controller has been implemented, as well as two small test cases (marker tracking and obstacle avoidance with optical flow) combining the JeVois camera with the small mobile robot AMiRo. The JeVois camera together with its existing machine vision modules enabled easy development of the test cases. Furthermore, the variety of the existing modules provides many possibilities for future projects. The tracking of the ArUco markers was problem-free. Moreover, an evaluation of the obstacle avoidance with optical flow also showed good results. The original idea of the project was to use the mini camera together with the drone. Due to difficulties we needed to switch to the mobile robot. As the drone, however, is based on the AMiRo architecture, we conclude that it will be easily possible to extend the use of the JeVois camera to drone based projects.

References

1. Betaflight (Version 2.9.1). Retrieved from https://github.com/betaflight/betaflight

2. Herbrechtsmeier, S., Korthals, T., Schöpping, T., & Rückert, U. (2016). AMiRo: A Modular & Customizable Open-Source Mini Robot Platform. In *IEEE*.

3. JeVois Smart Machine Vision (Version 1.5.2). Retrieved from https://www.jevoisinc.com/

4. Meyer, H. G., Bertrand, O. J. N., Paskarbeit, J., Lindemann, J. P., Schneider, A., & Egelhaaf, M. (2016). A bio-inspired model for visual collision avoidance on a hexapod walking robot. In N. F. Lepora, A. Mura, M. Mangan, P. F. Verschure, M. Desmulliez, & T. J. Prescott (Eds.), *Biomimetic and biohybrid systems* (pp. 167–178). Cham: Springer International Publishing.

A How to Start

This section describes how to start the two different applications. The source code and videos of the JeVois demos can be found here: *https://github.com/mrothmann/flymiro*. The changes for the byte interface for the FC can be found here: *https://projects.cit-ec.uni-bielefeld.de/projects/murox_dev*.

A.1 ArUco Marker Tracking

1. Connect JeVois camera to AMiRo
2. Using the command-line interface one can find out which number the desired videomapping on the camera has by using the command:

   ```
   listmappings
   ```

3. The number of the mapping

   ```
   NONE 0 0 0.0 YUYV 320 240 30.0 JeVois DemoArUco
   ```

   and the number in the line

   ```
   system("echo \"setmapping 4\" > /dev/ttyACM0");
   ```

   in *amiro_dist.cpp* need to be identical
4. Build binary of *amiro_dist.cpp* using CMake and Poky
5. Copy binary to AMiRo
6. Execute binary

A.2 Obstacle Avoidance with Optical Flow

1. Add videomapping to file *jevois/Config/videomappings.cfg* to disable video output and only use serial ports for communication:

   ```
   NONE 0 0 0.0 YUYV 176 144 100 JeVois OpticalFlow
   ```

2. Flash modified JeVois software to the SD card
3. Connect JeVois camera to AMiRo
4. Using the command-line interface one can find out which number the desired videomapping on the camera has by using the command:

   ```
   listmappings
   ```

5. The number of the mapping

   ```
   NONE 0 0 0.0 YUYV 176 144 100 JeVois OpticalFlow
   ```

   and the number in the line

   ```
   system("echo \"setmapping 9\" > /dev/ttyACM0");
   ```

   in *jevoisOpticalFlow.cpp* need to be identical
6. Build binary of *jevoisOpticalFlow.cpp* using CMake and Poky
7. Copy binary to AMiRo
8. Execute binary and supply n_0, g and k as command line parameters

A.3 Flight controller data recording

1. Put jumper on FC pins to enable flashing mode
2. Add the following to your bashrc: export TARGET=SPRACINGF3EVO
3. Clone Betaflight repository (http://projects.cit-ec.uni-bielefeld.de/git/murox_dev.0.git)
4. Build the repository: make hex
5. Connect FC to PC via USB, open Betaflight (the top right corner should say "Port: DFU")
6. Choose "firmware flasher"
7. Local firmware: choose the hex-file built in step 4
8. Flash (After flashing "Port: DFU" should change to "Port: dev/ttyACM0")
9. Remove USB and jumper
10. Be careful not to move the FC when inserting the USB cable again (otherwise the gyroskope will not work)
11. Set permissions

    ```
    chmod u+x /dev/ttyACM0
    ```

12. Clone flymiro github repository and compile *read_tty.c*
13. Execute the resulting binary
14. The recorded data will written to stdout