Fachhochschule Dortmund

Master Thesis for M. Eng. in Embedded Systems for Mechatronics

---

Development of a real-time software architecture for AMiRo robot based on the operator controller-module

---

Author:
Hector Gerardo Munoz Hernandez

Supervisors:
Prof. Dr. Carsten Wolff
Uwe Jahn

August 16, 2018

Contents

Abstract

This work covers most of the internal controller required functionality assigned to the DA_AMiRo project. The architecture of the AMiRo is briefly discussed and it is then explained how the three boards of the AMiRo talk to each other. Some other sensors are explained, and the Communication Area Network module. All these functionalities are incorporated in the sensor data handler which is explained later. Every piece of code described in the present report is part of the DA_AMiRo project and the reader can go to the gitlab repository and the Wiki page for more information. One of the purposes of the DA_AMiRo project is to compare its functionality with the DAEbot project conducted by Uwe Jahn. The reader can read all about the DAEbot's internal controller in the previous student's work [19]. This work will finalize with a comparison on both systems.

Chapter 1

Introduction

1.1 Real-time Operating Systems

The necessity to meet deadlines in today's projects is every time more demanded. Most applications consist of different tasks that can be executed in parallel, have different priority, have different period repetition, and frequently use shared resources. Using a multi-threading approach, however, can create a challenge in terms of performance, stability and time of implementation.

An operating system is a software component of a computer system that is responsible for the management and coordination of activities and the sharing of the resources of the computer [8]. "A real-time operating system perform these tasks, but is designed to run applications with a very precise timing and a high degree of reliability. This can be especially important in measurement and automation systems where downtime is costly or a program delay could cause a safety hazard" [11].

In other words, a real-time operating system is an operational system that provides certain capability according to preset timing constraints [2]. This is achieved through a scheduler that is designed to provide a deterministic execution pattern. And so, the RTOS are used in systems with critical timing requirements, when each task should be executed within a restricted time interval [2].

When it can be guaranteed that a task will never exceed a maximum amount of time in being completed, it can be said that the operative system is "hard real-time". If it can be guaranteed that a task will most of the time

not exceed a maximum amount of time in being completed, it can be said that the operative system is "soft real-time". An airbag system is an example of a "hard real-time" system, where the task has to be guaranteed to execute always within a time limit. For a "soft real-time" example, a video streaming can be considered, where an occasional loss of data can be accepted because it does not compromise the whole functionality [11].

In the real-time operating systems the managing of the tasks is crucial. To this end, programmers have to decide which task is running at certain time and how other tasks can preempt the processing resources if their priorities are higher. In other words, manage the schedule of the tasks so that they meet with their deadlines.

Some of the main characteristics of a RTOS are the following:

- Determinism: An application timing can be guaranteed within a certain margin of error.

- Soft and Hard Real-Time: The validity of data after meeting a deadline. In soft real-time the deadline is not that crucial but in hard real-time the meeting of the deadline is very important.

- Jitter: Which is the amount of error in the timing of a task over subsequent iterations of a program or loop [2].

1.2 Operator Controller Module

The OCM was introduced by a group of Professors in the University of Paderborn. The concept aimed to structure and design a reconfigurable controller systems [7]. As mentioned before, there are three different levels in the OCM. The first one is the lowest level of the OCM also simply known as the Controller. This level interacts directly with the plant of the system, which means that the control signal is produced and measured here. It is also necessary that the software processing in this layer is quasi-continuous so that the measured values are processed in real-time conditions [7].

The second layer is called the reflective operator. It is in this layer where the monitoring and the controlling routines are executed [7]. This layer does not have access to the actuators directly, because the Controller is already doing this. The expected result is that the Reflective Operator modifies the

Controller, sometimes also changing between different pre-established Controller configurations. This level also requires a quasi-continuous operations like a continuous adaptation algorithm or a watchdog. This operator also has to operate under hard-time constraints because its relation with the Controller. In short, this layer serves as a interface between the Cognitive Operator and the Controller [7].

The third layer is the Cognitive Operator. On this layer the system gathers information about itself and its environment to improve itself. This recollection of information can be done by applying various methods such as learning and model-based optimization among others. In other words, it is in this layer where the self-improvement takes place [7].

This project focuses on the Controller and Reflective Operator of the AMiRo. The controller is the responsible for commanding the internal sensors of the STM32F32 boards inside AMiRo in real-time. This sensor information is then retrieved by the Reflective Operator with a demo application that will be explained later in this document inside section 5.2.

1.3   AMiRo

The AMiRO project, which stand for Autonomous Mini Robot was started in the University of Bielefeld by Stefan Herbrechtsmeier and worked by Thomas Schöpping in collaboration with few others. The motivation for this project was to use small embedded systems in order to create a small robot that is easy to transport, usable on a table or the floor, appealing for young people, that is created with a modular approach is able to be extendable and customizable. With all of these aspects in mind, the result should be also a powerful tool for research and education [5].

The preliminary work was conducted in the University of Paderborn in the form of the BeBot mini robot [5]. The intended architecture of both robots is a modular one, which means that it has more than one board functioning as a unity. In AMiRo's case, said architecture consisted on three boards, DiWheelDrive, Power Management, and Light Ring that will be discussed in the next chapter, and two main extension boards: the Cognition board and the Image Processing board. The former will be explained in section 2.1.4, but the latter will not be discussed in this work, as it was not needed for the current reach of the project. If the reader is interested in the Image Processing board, he or she can refer to the project presentation given by

Prof. Herbrechtsmeier [6].

Physically AMiRo has a cylindrical shape with a diameter of 100mm. It is covered by a chassis that does not cover the DiWheelDrive board's bottom as this board has the proximity sensors pointing towards the ground. There are also two wheels with two separate motors directly connected to the DiWheelDrive board and two metal sliders for the robot stabilization in flat surfaces [25].

The chassis has three more openings for the following connections: USB serial that goes directly into the DiWheelDrive board, the power supply, and the charging pins. Additionally, when the Cognition board is attached, there are three more possibilities to physically interact with the robot: USB serial, USB dongle for wireless connection with the Linux OS, and a micro USB cable for connecting also with the Linux OS which is used in this board. The Cognition board will be later discussed in section 2.1.4.

These characteristics make AMiRo a very compact and portable robot. Every board is designed to be modifiable, stackable and exchangeable with custom elements. All boards communicate with each other with the Controller Area Network protocol [25]. This way every board can read any message from the CAN bus or write into it every time it is necessary.

Chapter 2

Architecture of AMiRo

The AMiRo project was started in 2015 and ever since, the repository supporting the AMiRo project has had several updates [26]. It is worth mentioning that the version used as the basis for the present work was the version 1.0 stable. The AMiRo's repository consists of three basic folders: AMiRo bootloader, AMiRo Operating System, and ChibiOS.

Although this software architecture will be explained with more detail in section 2.2, it is worth mentioning that AMiRo's bootloader was used as it is in version 1.0 from the original repository in order to configure the computer for programming the AMiRo. The ChibiOS was also used as it is in version 1.0 from the repository [26]. The AMiRo Operating System was hugely modified to meet the requirements of the present work, which can be found under the current project's repository *DA_AMiRo* [18].

2.1 Hardware Architecture of AMiRo

As mentioned in the previous chapter, the AMiRo is equipped with three basic board modules and two extension boards. In this section each board will be briefly discussed with a special emphasis in the functionalities that are of interest for this work. For a more complete discussion about the boards or the AMiRo in general, the reader is invited to read the documentation referenced in this work, specially [23], [25], [24], [5], [6] which can be found on the CITEC website or on the repository of this work on gitlab [18].

The functionality of the sensors used in the project will be briefly explained in this chapter. For the explanation on how to read the output of

5

the sensors that are sent via the Controller Area Network protocol, the reader is invited to jump directly into section 3.1 for reviewing the CAN convention used in the entire project and section 3.2 for understanding each sensor's output.

2.1.1 DiWheelDrive

The DiWheelDrive is the bottom board of the AMiRo. It is equipped with an ARM Cortex-M3 based STM32F103 MCU from STMicroelectronics [25]. It is through the DiWheelDrive's UART interface that the AMiRo's three basic modules get programmed. Each board gets programmed at a time with help of the AMiRo bootloader which will be discussed in section 2.2.1.

This board has two motors of 1W DC. The purpose of having two motors was to be able to have a differential kinematic to allow flexible movements [25]. Among other sensors and actuators, the ones that are in this board and are being used in this work are: three axis gyroscope, accelerometer, and magnetometer, floor proximity sensors, the two motors, and optical motor encoders. The functionality and some specifications of the above mentioned sensors and actuators will be discussed in the next subsections.

As mentioned before, the DiWheelDrive board has a Cortex-M3 MCU for calculating such things as motion control, odometry, and dead reckoning algorithms at high sampling rates [25]. This board can be accessed by the other boards via the Controller Area Network protocol or via USB UART interface directly with the user. It is important to mention that every board has also a USB UART interface to communicate directly with the user but they are covered by the chassis except for the DiWheelDrive board [25].

Gyroscope

The gyroscope is a sensor that measures rotational motion. The units of the resulting measurement are revolutions per second or degrees per second. As a quick example, in a balancing robot a gyroscope can be used to measure how much the robot has rotated from the desired position and this information can be sent to an actuator to make corrections in order to reach the desired angle [30]. A gyroscope normally consists of a mass that moves with constant angular momentum, so when the gyroscope is tilted the axis of rotation of said mass reacts to the applied rotational movements making the axis of rotation tilt. This tilting leads to a displacement of the capacitance fingers

6

inside the sensor which changes the capacitance. Finally when compared to a reference capacitance level, the angular velocity can be determined [5].

The gyroscope used in the DiWheelDrive board is the l3g4200d from STMicroelectronics. This sensor communicates via SPI with the processor. This sensor is configured to have a resolution of +-500 dps in accordance with the user's manual [16] and it is possible to obtain the output in degrees per second, micro degrees per second, revolutions per second, and micro revolutions per second with the actual code. The actual units are degrees per second but the user can easily change the output selecting a different function from *updateSensorVal()* function inside DiWheelDrive.cpp. For a more clear explanation on the code structure, see section 2.2.3. The sensor communicates with the processor via SPI communication.

Accelerometer

The accelerometer is a sensor that measures the acceleration of an object. The units of the resulting measurement are meters per squared second or in g forces, where g as a unit equals $9.81m/s^2$ [31]. It is typically a system composed by a mass, a spring, and a damper. The mass moves according to the applied acceleration and changes the resulting capacitance that can be sent as an output for further reading and processing [5].

The accelerometer used in the DiWheelDrive board is the lis331dlh from STMicroelectronics. This sensor is configured be scaled with a +/-8g factor in accordance with the user's manual [17]. The output is in g units and the sensor is being communicated via SPI with the processor.

Magnetometer

The magnetometer is a sensor that measures the magnetic field for all three physical axes [34]. This sensor works thanks to three principles. The first one is called the Anisotropic magneto resistance effect discovered in 1857 by William Thomson, in which an external magnetic field can make the sensor change the electrical resistance of a ferromagnetic material. The electrical resistance will be at a maximum value when the direction of the electrical current is parallel to said magnetic field [5].

The second principle is called Fluxgate and it is a way of calculate the vectorial measurement of a magnetic field invented by Friedrich Förster in 1937. The method starts with two cores wrapped by two coils of wire. There

should be an electrical current driving the core through cycles of magnetic saturation. The resulting electrical current of the second coil will depend on the external magnetic field [5].

The last principle is called the Hall effect discovered in 1897 by Edwin Hall. In this principle can be visible when an electric current flows through a conductor in a magnetic field, the magnetic field will exert a transverse force on the moving charge carriers which tends to push them to one side of the conductor. The result is a voltage difference between both sides of the conductor [10].

The magnetometer used in the DiWheelDrive board is the hmc5883l from STMicroelectronics. This sensor is configured to have a field resolution of +/-5Gauss according to the user's manual [9] and the current output is set to be in Gauss units. The sensor is being communicated via I2C protocol with the processor.

Floor Proximity Sensor

The floor proximity sensor used is the "Fully Integrated Proximity and Ambient Light Sensor With Infrared Emitter", part vcnl4020 of Vishay [32]. Four of these sensors are in the bottom of the AMiRo, programmed as light sensors, while other eight of these sensors are being used as proximity sensors and ambient light detectors coordinated by the Power Management board, see section 2.1.2 for more information.

The ambient light sensor measures the intensity of light and its is measured in luminance. In order to do this the sensor usually consists of a photoresistor or a photo-sensitive material, outputting a resistance that can be measured and compared to a default value to estimate the luminance [15].

The goal of providing AMiRo with these sensors attached to its bottom is to detect changes in the color of the floor. An example application can be for the AMiRo to follow a line or path painted on the ground, by making the robot recognize when it is over one color and when it is not.

Motors

As previously described in section 2.1.1, the AMiRo has two DC motors of 1W each to offer a differential kinematic and to allow agile movements [25]. These motors are connected to the DiWheelDrive board which means that

this board controls the velocity of the motors and also knows the actual velocity of the motors.

The velocity of the motors can be set anytime from the Cognition board and the actual velocity of the AMiRo is one of the sensor values that can be asked from AMiRo's internal controller. The user can set and view the velocity in $\mu m/s$ or in $\mu rad/s$.

Encoder

AMiRo has optical encoders as well, that allow the user to know how much the motors have turned from a certain checkpoint. This data can be used to know how far the AMiRo has moved from a certain point. This sensor returns the x coordinate in μm, the y coordinate in μm, and the orientation of the wheels in μrad

2.1.2 Power Management

The Power Management board is the central basic module of AMiRo. This board is responsible for the power supply and elementary behavior of the system. This is why it has the most powerful microcontroller. This board has an ARM Cortex-M4 based STM32F405 from STMicroelectronics. The characteristics from this microcontroller were needed to ensure a fast reaction times on critical system events and still provide enough headroom to additionally perform basic behaviors like obstacle avoidance, or homing [25].

There are two lithium-ion batteries connected to the Power Management board that power the whole system, and there are also some sensors on this board. Among the sensors that this board has, the ones that are being used by this project are: eight proximity sensors that act as proximity sensors and also as ambient light sensors, sensors for knowing if the charging cable is connected, the remaining battery charge or the time for a complete battery charge, the time for this both events to be completed, and the actual current consumption [25].

Power Status

For the tracking of the power status, AMiRo is equipped with a bq27500 Impedance Track from Texas Instruments that allows the robot to obtain

information such as remaining battery capacity, state-of-charge, run-time to empty, battery voltage, and temperature [12].

Proximity Ring

As seen in section 2.1.1, the eight vcnl4020 sensors in connected to the Power Management board are functioning as ambient light sensors and also as proximity sensors. For this last functionality, the sensor has a built-in infrared emitter and photo-pin-diode, having a maximum relying measurement of 200mm [32]. These sensors are connected in a ring-fashion way so that the AMiRo has eight sensors throughout its entire circumference, making it able to detect obstacles in all horizontal directions. The units of the output are also in mm.

2.1.3   Light Ring

The Light Ring board is the top-most board in the AMiRo. This board has a STMicroelectronics ARM Cortex-M3 based STM32F103 MCU. It contains eight RGB LEDs arranged in a circular way corresponding to the Proximity Ring discussed in last section. These LEDs can be set from the Cognition board, setting each color of the RGB channels [25].

2.1.4   Cognition Board

This board was thought as an extension board which is no longer from the basic set of boards that make the AMiRo. The Cognition board in this project is being used as the Reflective Operator according to the OCM model. It provides all of the high-level tasks, gathering and coordinating the information generated in the three basic boards also known as the internal controller [25].

The most important feature of this board is the attached Gumstix Overo Computer-on-Module. In the current setup, the Overo TidalSTORM COM is being used, which combines an ARM Cortex-A8 based DaVinci DM3730 SoC from Texas Instruments [25]. This board has the same serial programming port as on the three basic boards. It also has a Type A, a micro-AB, and an internal pin header USB interfaces [25].

The operating system running on the Overo COM is based on the Linux Yocto reference distribution Poky [3], and the way it communicates with

the three basic boards is through CAN, using a simple SocketCAN software-wrapper that is offered by this distribution [25].

2.2 Software Architecture of AMiRo

As it has been discussed in previous sections, the AMiRo project was started back in 2015 and it is still being maintained. The version 1.0 of the repository is the one being used for this project [18].

2.2.1 AMiRo bootloader

The open-source bootloader for MCUs is the bootstrap for the three basic boards of the AMiRo. The AMiRo's bootloader is the responsible of configuring and setting up the computer for working with the AMiRo. The set-up that is provided in the bootloader is more than enough to successfully start working on the Operating System of AMiRo and programming it [33].

As said before, the DiWheelDrive is the only module of which the programming port can be accessed by the user. This is why new software is applied to the two other basic boards by remote flashing via OpenBLT. Using CAN communication the new binary is forwarded by the DiWheelDrive board so that the target board receives the data and stores it in its flash memory [25].

2.2.2 AMiRo real-time operating system kernel

For the real-time operating system the open-source ChibiOS was chosen. This project aims at creating a portable, light weight, and fast real-time operating system for embedded applications [27]. ChibiOS not only provides low-level drivers for all interfaces of the most common microcontrollers, it also gives an unified hardware abstraction layer, priority based preemptive scheduling, and high-performance event and message passing systems. ChibiOS can be configured statically and dynamically to exactly fit the application in order to maximize efficiency and performance [25].

ChibiOS

Some of the main characteristics from ChibiOS are:

- System time: ChibiOS has a 16 or 32 bits system time counter.

- Real tick-less mode: There is no periodic system tick for optimal power management.

- IRQ Management: ISRs abstraction.

- Preemption: Fully preemptive scheduling.

- Round robin scheduling: Round robin scheduling for threads at equal priority.

- Messages: Inter-thread synchronous messages.

- Mailboxes: Message queues.

- Counter semaphores: Semaphores with boolean state.

- Binary semaphores: Semaphores with boolean state.

- Mutexes: Mutexes implementing the priority inheritance algorithm.

- Events: Events, event flags, event sources.

- Dynamic services: Dynamic threading.

Note: All the above mentioned characteristics can be found and further explained in the ChibiOS homepage. [27]

A project containing ChibiOS consists on the Operating System source files, few configuration files and a HAL file for the peripherals handling.

2.2.3 AMiRo Operating System

AMiRo's Operating System or board specific abstraction layer from the original repository [26] includes two basic applications, the first one is a rudimentary command shell and the second one a hardware test. The first one is already integrated in ChibiOS and it is extended by module specific commands [25].

The structure of the AMiRo Operating System, including the files that are being used with the current project can be visualized in figure 2.1. The entire project is written in C++ language, with some inclusions of C code.

In this figure, it can be seen that inside *AMiRo_OS* there is the *components* folder which contains the shared codes that the three boards are using. The other part of the structure worth mentioning now is the *devices* folder in which the three boards have their own set of files, always maintaining the same structure among them.
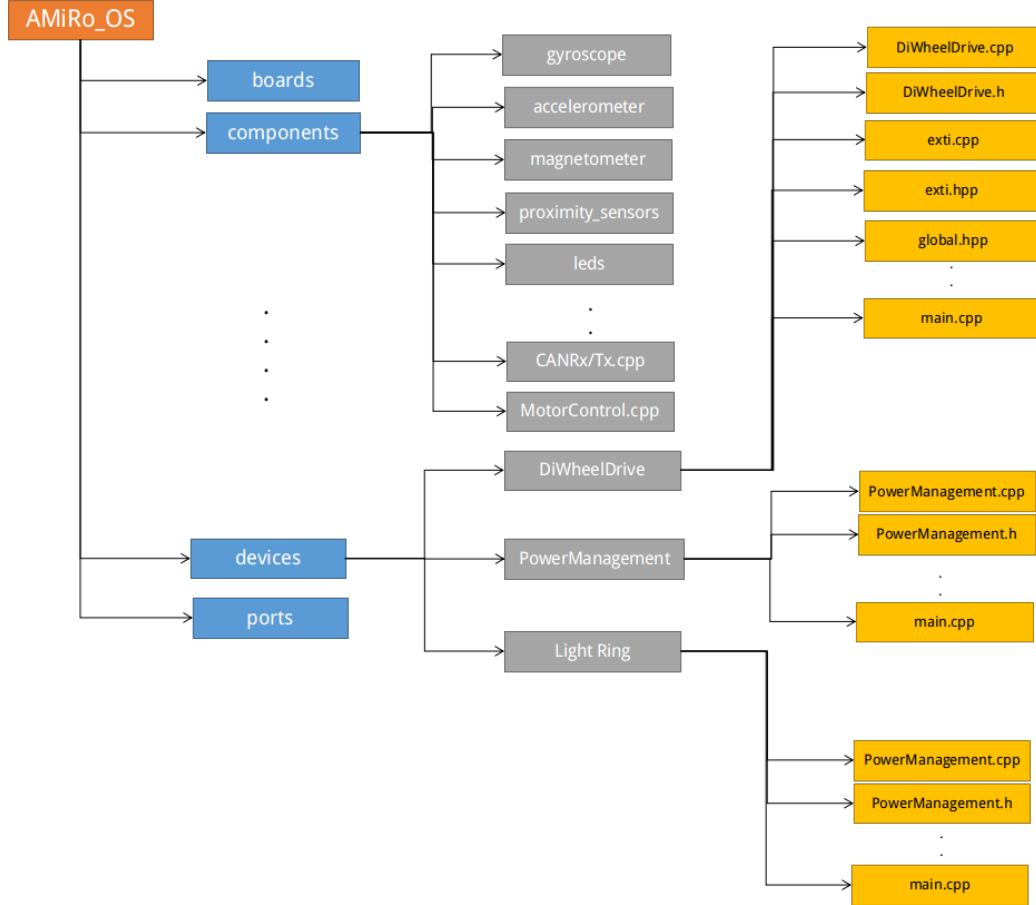

Figure 2.1: AMiRo_OS breakdown structure

2.2.4 Overall structure

From the past sections the hardware and software structure of the AMiRo was mentioned and briefly discussed. From this point forward the structure and functionality of the DA_AMiRo project will be discussed as it is the

central point of the master thesis. As a friendly reminder, the reader can obtain the source codes made for the DA_AMiRo project from the repository [18].

Taking into consideration once again figure 2.1, the

Chapter 3

Controller Area Network

This chapter explains how to receive and transmit CAN frames in the AMiRo. The functionality of the CAN module had to be merged with the current ControllerAreaNetworkRx.cpp and ControllerAreaNetworkTx.cpp of the existing repository of AMiRo [26]. The ControllerAreaNetwork.h however, is the same that the DAEbot project is using. The reason behind sharing the ControllerAreaNetwork.h among projects is to be able to use a single Relfective Operator for both systems AMiRo and DAEbot so it is easier to compare functionalities.

3.1   Convention used for sending and receiving frames

The format of the CAN frames was done accordingly with the pre-established convention for the DAEbot project [13]. See figure 3.1

For the transmission of a CAN frame, the id is coded according to figure 3.1 and when a CAN frame is received, the id is decoded into the same structure. The CAN frame pre-established structure can be seen in snip of code 3.1, which can be compared with the figure 3.1.

A Base Frame Format CAN Identifier (11 Bits) will be used to identify the messages in the system.

Bit	0	1	2	3	4	5	6	7	8	9	10
Subject	C/S	Priority		Identity							
Number of Bits	1	2		8							

- The first bit, the C/S Bit is used to identify if the message is:
  - 0 = Command (/Actuator)-Data (higher priority as Sensor-Data)
  - 1 = Sensor (/Status)-Data (lower priority as Command-Data)
  - The Operator sends a command message to a Controller with the Controller's ID (take a look below)

- The Bits two and three are used to set the priority of this CAN Frame to:

  - 0 (00) = Urgent
  - 1 (01) = High
  - 2 (10) = Medium
  - 3 (11) = Low

- A unique ID is used for every topic on every component (2^8 = 256 possibilities). Take a look below for the specific ID's and don't forget to write them down if you create new ones.

Figure 3.1: Pre-established CAN frame format

```
typedef struct can_frame_types {
    priority_can_id_t priority_id;
    c_s_bit_can_id_t c_s_bit_id;
    topic_can_id_t topic_id;
    uint8_t dlc;
    union{
        uint8_t      data8 [8];
        uint16_t     data16 [4];
        uint32_t     data32 [2];
    };
} can_frame_types_t;
```

Listing 3.1: CAN frame

In order to send a CAN frame, can_transmit_data_frame_pd(...) can be called. When sending a CAN frame, the id is set to be in standard mode. The CAN frame that is sent consists of three main parts: id, DLC, and the data. The id is therefore set with the can_code_identifier_pd(...) function before actually calling the transmit function. This function combines the c_s bit with the priority bit and then combines that value to the topic ID. The

16

CAN frame is sent with the function HAL_CAN_Transmit(...).

On the CAN reception side, the status of the reception is assigned when calling to the can_receive_data_frame_pd(...) function. The status of the CAN reception can be successful, timeout, or error. Then the id of the CAN frame gets decoded with the function can_decode_identifier_pd(...) and the data frames are passed into a global structure like in snip of code 3.1

Note that can_code_identifier_pd(...) and can_decode_identifier_pd(...) are functions that were already implemented beforehand, for this project it was necessary to adjust both functions to operate with the current hardware platform.

3.2 Interpreting outputs of the sensors

Chapter 4

Sensor data handler

4.1   Overall functionality

4.2   Implementing modes

4.3   Real-time implementation

Chapter 5

MuRoX

5.1 Compiling for MuRoX

5.2 Example application

Chapter 6

Matlab

6.1   Compiling from Simulink for MuRoX

Chapter 7

Results

7.1   AMiRo vs. DAEbot

7.2   Outlook

Bibliography

[1] *Amiro cheatsheet.*

[2] FreeRTOS. (2015). What is an rtos?, [Online]. Available: `http://www.freertos.org/about-RTOS.html`.

[3] T. Graydon and B. Flanagan. (Aug. 30, 2016). Poky linux, [Online]. Available: `http://www.pokylinux.org/`.

[4] C. Henke, M. Tichy, T. Schneider, J. Böcker, and W. Schafer, "System architecture and risk management for autonomous railway convoys", pp. 1–8, May 2008.

[5] S. Herbrechtsmeier, "Amiro – autonomous mini robot for research and education", Cluster of Excellence Cognitive Interaction Technology 'CITEC', Bielefeld University, Bielefeld, Germany, Project presentation. [Online]. Available: `http://www.ks.cit-ec.uni-bielefeld.de/`.

[6] ——, "Amiro – autonomous mini robot for research and education (extensions)", Cluster of Excellence Cognitive Interaction Technology 'CITEC', Bielefeld University, Bielefeld, Germany, Project presentation. [Online]. Available: `http://www.ks.cit-ec.uni-bielefeld.de/`.

[7] T. Hestermeyer, O. Oberschelp, and H. Giese, "Structured information processing for self-optimizing mechatronic systems", in *Proceedings of 1st International Conference on Informatics in Control, Automation and Robotics (ICINCO 2004), Setubal, Portugal,* (published), IEEE Computer Society Press, Aug. 2004.

[8] M. Hill, *Concise Encyclopedia of Engineering.* 2002.

[9] Honeywell. (). 3-axis digital compass ic hmc5883l, [Online]. Available: `https://cdn-shop.adafruit.com/datasheets/HMC5883L_3-Axis_Digital_Compass_IC.pdf`.

[10] HyperPhysics. (). Hall effect, Department of Physics and Astronomy, Georgia State University, [Online]. Available: `http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/Hall.html#c2`.

[11] N. Instruments. (2018). What is a real-time operating system (rtos)?, [Online]. Available: `http://www.ni.com/white-paper/3938/en/`.

[12] T. Instruments. (2018). Bq27500 (nrnd) system-side impedance track™ fuel gauge, [Online]. Available: `http://www.ti.com/product/BQ27500#`.

[13] U. Jahn. (2017). Daebot wiki, [Online]. Available: `https://gitlab.idial.institute/DAEbot/DAEbot/wikis/home`.

[14] S. M. ltd. (2018). Stm32f3discovery overview, [Online]. Available: `http://www.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-discovery-kits/stm32f3discovery.html`.

[15] C. Mathas. (). Ambient-light sensors mimic the human eye, [Online]. Available: `https://www.digikey.com/en/articles/techzone/2014/feb/ambient-light-sensors-mimic-the-human-eye`.

[16] S. mems and sensors. (2011). L3g4200d: Three axis digital output gyroscope, [Online]. Available: `https://www.elecrow.com/download/L3G4200_AN3393.pdf`.

[17] ——, (). Mems digital output motion sensor ultra low-power high performance 3-axes "nano" accelerometer, [Online]. Available: `https://www.st.com/resource/en/datasheet/cd00213470.pdf`.

[18] H. G. Munoz Hernandez. (2018). Da_amiro repository, [Online]. Available: `https://gitlab.idial.institute/DAdevs/DA_AMiRo.git`.

[19] ——, "Daebot, internal controller", Fachhochschule Dortmund, Research project, 2014.

[20] (2016). Murox wiki, [Online]. Available: `http://www.multirobotix.de/dokuwiki/doku.php?id=start`.

[21] Plantuml.com. (2018). Plantuml home page, [Online]. Available: `http://plantuml.com/`.

[22] R. Queinell, "Embedded market study", *In ARM TechCon*, p. 34, 2015.

[23] U. Rückert, "Amiro: A mini robot as versatile teaching platform", Cluster of Excellence Cognitive Interaction Technology 'CITEC', Bielefeld University, Bielefeld, Germany, paper, 2018. [Online]. Available: `http://www.ks.cit-ec.uni-bielefeld.de/`.

[24] ——, "Amiro: A mini robot for scientific applications", Cluster of Excellence Cognitive Interaction Technology 'CITEC', Bielefeld University, Bielefeld, Germany, paper, 2015. [Online]. Available: `http://www.ks.cit-ec.uni-bielefeld.de/`.

[25] ——, "Amiro: A modular & customizable open-source mini robot platform", Cluster of Excellence Cognitive Interaction Technology 'CITEC', Bielefeld University, Bielefeld, Germany, paper, 2016. [Online]. Available: `http://www.ks.cit-ec.uni-bielefeld.de/`.

[26] T. Schöpping. (2018). Amiro-os, [Online]. Available: `https://openresearch.cit-ec.de/projects/amiro-os/wiki`.

[27] G. D. Sirio. (2018). Chibios home page, [Online]. Available: `http://www.chibios.org/dokuwiki/doku.php`.

[28] ——, (2018). Chibios rt manual, [Online]. Available: `http://chibios.sourceforge.net/docs3/rt/index.html`.

[29] C. Sondermann-Wölke and W. Sextro, "Integration of condition monitoring in self-optimizing function modules applied to the active railway guidance module", Jan. 2010.

[30] Sparkfun. (). What is a gyroscope, [Online]. Available: `https://learn.sparkfun.com/tutorials/gyroscope/all`.

[31] ——, (). What is an accelerometer, [Online]. Available: `https://learn.sparkfun.com/tutorials/accelerometer-basics`.

[32] Vishay. (). Fully integrated proximity and ambient light sensor with infrared emitter, i2c interface, and interrupt function, [Online]. Available: `https://www.vishay.com/docs/83476/vcnl4020.pdf`.

[33] F. Voorburg. (Aug. 30, 2016). Openblt, [Online]. Available: `http://feaser.com/en/openblt.php`.

[34] W3C. (). Magnetometer, [Online]. Available: `https://www.w3.org/TR/magnetometer/`.