Using ARM7 and µC/OS-II to Control an Autonomous Sailboat

Michael Koch and Wilhelm Petersen

Abstract. This paper presents some aspects of an autonomous sailboat project, we started at our university. An interdisciplinary group of students of mechanical engineering, electrical engineering and computer science are building the boat and are developing the control hard- and software. Our first attempt to control the boat combines an ARM7 microcontroller and the real time operating system µC/OS-II. Beside our first boat design, we will talk about the used control hard- and software, especially the mentioned controller and operating system, and the interaction between the components. We show the system architecture of the control system, the power consumption of the components and the different possibilities to process sensor data with the controller and the operating system.

1 Introduction

Working at an university located at the seaside and sailers since many years, we are searching for a student project that combines practical experience in mechanical and electrical engineering and computer science with water and sailing. In September 2010 we discussed the idea to build an autonomous sailboat with some students and now – in the actual semester – we have a small group of students, who are building the boat.

The interdisciplinary team consists of four students of mechanical engineering and four students of electrical engineering and computer science. The mechanical

Michael Koch
FH Stralsund, Electrical Engineering and Computer Science,
Zur Schwedenschanze 15, 18435 Stralsund, Germany
e-mail: michael.koch@fh-stralsund.de

Wilhelm Petersen
FH Stralsund, Mechanical Engineering, Zur Schwedenschanze 15,
18435 Stralsund, Germany
e-mail: wilhelm.petersen@fh-stralsund.de

engineering group is building the boat and the mechanical parts, while the sensors, the controller and the software are built by the computer science group. The students are supported by the workshops of both departments. They use this project as practical work within their studies. The team is completed by a student of business administration and engineering. He is organizing the needed materials and components.

The students began their work in February 2011. In May 2011, the mold of the hull of the boat is built. We hope, we can finish building the boat in July. In parallel, we are building the controller hardware and the sensors. The aim is to have a first version of the whole software in July.

2 The Boat

Our first sailbot design – called FHsailbot – is based on an AMYA[1] one meter class specification. The size of the hull is limited by the rig, especially the size of the available masts. We use a profiled aluminum mast with a groove for the sail. The maximum size of the mast (2 m) leads to a boat length of 1.52 m. The second limiting factor for our first design are the transport possibilities of our cars and trailers. To test the control system during the building the boat, we equipped an old model boat – called Saudade – with the controller, sensors and actuators. So we can test the components and the software independent of the new boat design. Figure 1 shows the hull of the FHsailbot and our test system Saudade. Table 1 compares the parameters of the boats.

The mold for the hull was shaped from a solid block of closed cell foam using an industrial robot equipped with a milling tool.

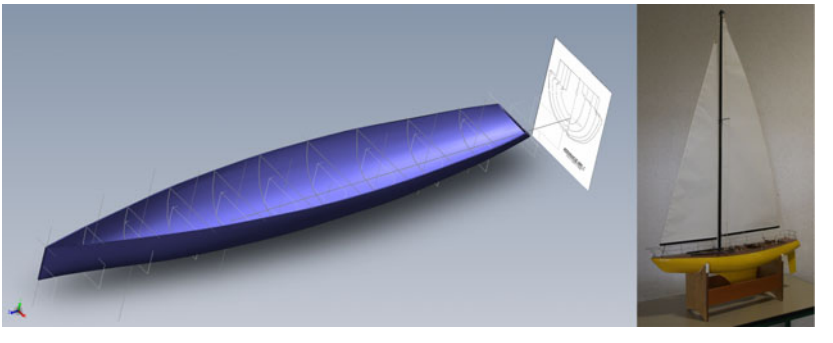

Fig. 1 The hull of the FHsailbot (left) and the test system for the control components (right)

[1] American Model Yachting Association.

Table 1 Basic parameters of the boats

Boat	FHsailbot	Saudade
Length overall	152 cm	112 cm
Length waterline	148 cm	103 cm
Beam	33 cm	26 cm
Draft	81 cm	26 cm
Displacement	∼15 kg	9 kg
Sail area	∼0.65 m^2	0.52 m^2

3 Control Hardware

At the beginning of the project we have a long discussion about the controller needed to control the vessel. Beside microcontrollers, we use microprocessor for mobile devices like notebooks in different multimedia projects (e.g. Pentium M (Intel, USA) and similar processors) and for small PC-like systems with 386/486-cores. On the other hand, we use microcontrollers from Texas Instruments (TI, USA) like the MSP430 for many small systems. As a compromise between performance and power needs, for this project, we decided to use an ARM7 microcontroller Stellaris Cortex-M3 from TI as the main controller. Only its use is discussed in this paper (see details later in this chapter). The controller gets data from different sensors (GPS, 3-axis compass, wind direction), generates PWM signals for sail winch and rudder servo and communicates over Bluetooth. An USB interface is used for programming and debugging. Status information may be displayed on a small OLED display. Beside the internal FLASH memory, additional data may be stored on a SD-card.

Figure 2 shows the system architecture of the control system. Beside the controller board with display and SD-card, the sensors (GPS, compass, wind direction) are providing the necessary environmental data, rudder servo and sail winch are directed by the controller. The failsafe module is able to overwrite the controller signals of rudder and winch using a remote control to avoid problems (e.g. collisions with other boats), if the controller does not work as intended.

3.1 Controller

All ARM7 microcontrollers of the Stellaris Cortex-M3 family from TI use the same core running at 50 MHz with different peripherals. Due to the three UART, we decided to use the LM3S6965 controller. If a separation of some peripherals from the main controller is necessary, e.g. due to limited wire length of some sensors, a controller with CAN bus may be used to solve the problem. Figure 3 shows a possible solution: LM3S8962 combined with the smaller LM3S2110. The sensor on the I2C port may be placed as far away as possible from metal components of a boat. The ports of the LM3S2110 may be controlled over the CAN bus from the LM3S8962.

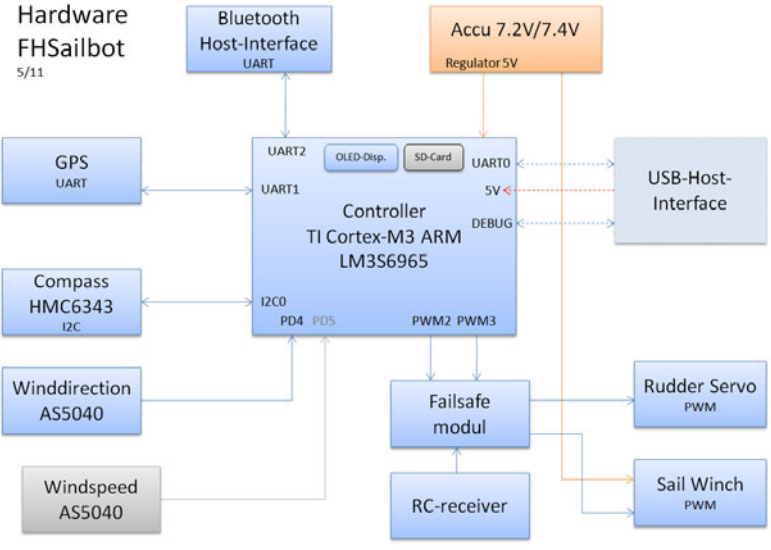

Fig. 2 System architecture of the control system: Sensors (left), actuators (right bottom), Bluetooth and USB communication and power supply

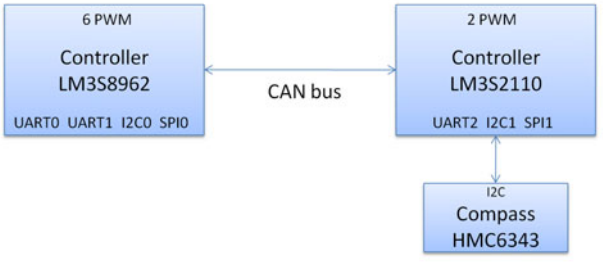

Fig. 3 System architecture with CAN bus, compass separated from the main controller

Table 2 shows the main characteristics of the controllers and the number of the different interfaces.

The controller needs a supply voltage of 3.3 V. All peripheral pins are 5 V-tolerant, that means, we dont need to change the logic levels of 5 V-peripherals. For the boat we use an evaluation kit (EK-LM3S6965) from TI, which provides the following additional components: In-circuit debugging interface, USB interface

Table 2 Main characteristics of the Stellaris Cortex-M3 controllers and the available interfaces

Controller	LM3S6965 (without CAN)	LM3S8962/LM3S2110 (CAN)
Main clock	50 MHz	50 MHz/25 MHz
Flash/RAM	256 KB/64 KB	
Timers	4x 32-bit or 8x 16-bit, SysTick, Watchdog	
I2C	2	1/1
SSI/SPI	1	1/1
UART	3	2/1
PWM	2x3	2x3/2
Ethernet	1	1
CAN	-	1/1

(communication over UART0, programming of the device, debugging), SD-card slot, buttons, status led and an 128x64 OLED display. Figure 4 shows the controller board and some other components.

3.2 Sensors

The boat gets data from three sensors: GPS, compass and wind direction, similar to [1]. The GPS is a small module from NAVILOCK (Germany) based on an ublox5 chip set (u-blox, Switzerland). Under free air conditions, the module is able to generate differential GPS data (Fig. 4 b). With a free sight to the south, we try to

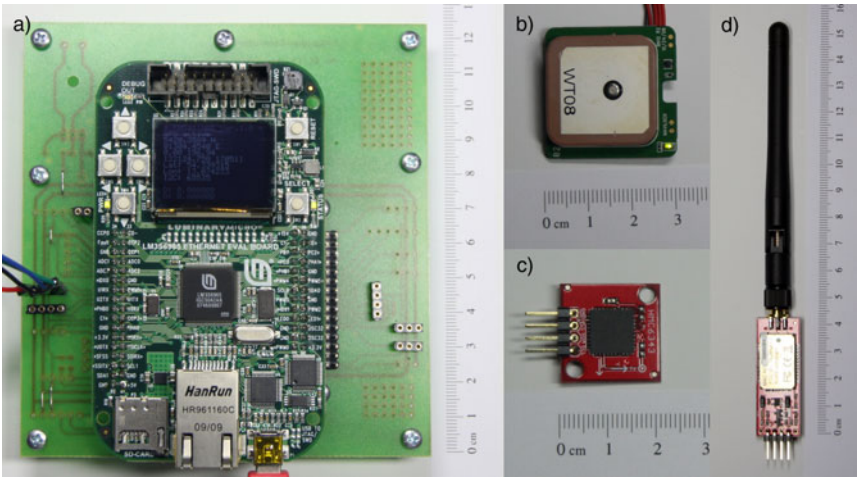

Fig. 4 Components: **a** Controller board EK-LM3S6965 with test adapter. **b** GPS. **c** compass. **d** Bluetooth modem.

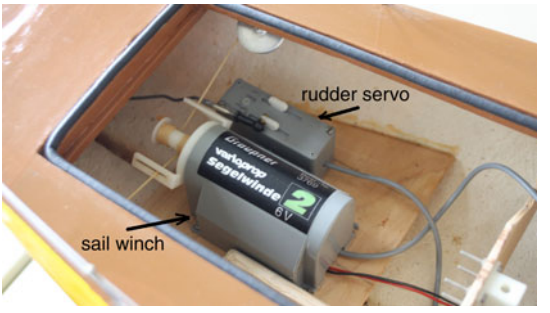

Fig. 5 Rudder servo and sail winch in the test system Saudade

receive the reports from the EGNOS[2] satellite ARTEMIS to increase the accuracy of the GPS. Unfortunately the connection to the EGNOS satellite is very unstable. The GPS module occupies one UART interface. As compass, we use the 3-axis compass HMC6343 (Honeywell, USA). It returns the tilt-compensated value for the heading and the heeling of the boat. It communicates over the I2C bus. To get the direction of the apparent wind, we use the PWM signal generated by the AS5040 (austriamicrosystems, Austria). To eliminate the tolerance of the PWM frequency, the controller gets the absolute value of the wind direction *w_dir* by the following calculation:

$$w\_dir = \left(\frac{w\_dir\_value \times 1025}{w\_dir\_period} - 1\right) \times \frac{360}{1024} \tag{1}$$

Where *w_dir* is the wind direction in degree, *w_dir_value* the active phase of the PWM signal and *w_dir_period* the whole period of the PWM signal.

3.3 Actuators

To control rudder and sail, we use standard components similar to the components used for the Saudade: a servo for the rudder and a sail winch with separate power supply to control the sails (Graupner, Germany). Figure 5 shows the components of the test system.

3.4 Power Consumption

For a small boat like our test system Saudade, the power consumption is a critical factor. To keep the system simple and cheap, we searched for a control system, which can be supplied by small NiMH oder LiPo (Lithium-polymer) accumulators. The chosen LM3S6965 needs less than 1 W. The power consumption and the cost of the components shows Table 3. Even if the computing power may not be sufficient for all algorithms, we think this is a good starting point.

[2] European Geostationary Navigation Overlay Service.

Table 3 Power consumption and cost of the control system components

Component	Current at Voltage	Cost
EKS-LM3S6965 at 50 MHz, display on	170 mA at 5 V	80 €
Bluetooth modem	30 mA at 5 V	54 €
GPS	80 mA at 5 V	50 €
Compass	5 mA at 3.3 V	130 €
Wind direction	20 mA 3.3 V or 5 V	
Rudder servo quiescent current	10 mA at 5 V	50 €
Rudder servo max. current	500 mA at 5 V	
Sail winch electronic	10 mA at 5 V	150 €
Sail winch motor max. current	3.5 A at 7.2 V	
Failsafe system incl. remote control receiver	30 mA at 5 V	50 €

The boat needs a minimum total current of about 360 mA and a maximum total current during the sail adjustment of about 4 A. To supply the necessary current, the boat is equipped with two accumulators: one for the control system and the sensors (regulated to 5 V) and another directly connected to the sail winch. The cost for the whole control system including the accumulators, sensors and actuators will be less than 1000 €.

3.5 Communication

Today there are two ways to communicate with the boat. During the development of the software, we use the USB interface to program and to debug the system and to exchange data using a virtual COM port on the host system. To get data from the boat and send data to the boat during sailing, we use a Bluetooth connection. The Bluetooth module connects to UART2 and transmits the serial data. The BlueSMiRF RP-SMA (Sparkfun, USA) connects to a standard 2.4 GHz antenna. On the host side, we use a PCMCIA-Bluetooth module with the same antenna. With this combination, we reach distances of about 100 m.

For a failsafe operation, the PWM signals of the controller may be overwritten using a standard remote control (RC). With a separate channel of the RC we can switch between the PWM signals of the RC and the controller signals.

Other communication modules will be included in future.

4 Software

The software of the control system is written in C. TI provides a driver library to encapsulate some of the hardware details, e.g. to program a timer or to communicate using an UART. To simplify the software development, there are different programming environments available. In this project, we use the IAR[3] Embedded

[3] IAR Systems, Sweden.

Workbench for ARM. An alternative are TIs Code Composer Studio development tools. Based on the in-circuit debugging interface of the evaluation kit, we are able to program and debug the controller from within the Embedded Workbench using the USB interface.

4.1 Operating System

Because of the different tasks the control system has to do during sailing (get the sensor data, calculate the course, generate the signals for actuators and communicate with the environment), we decided to use an operating system for the control system. There are different operating systems available for the ARM7 controller. We use the very small and stable operating system µC/OS-II. It has a small real-time kernel and has been certified within some commercial products to meet the requirements to be used in safety-critical systems. Details of the OS can be found in [2]. An alternative with compatibility to Linux is RoweBots Unison V5 RTOS (RoweBots, Canada).

4.1.1 Hardware Requirements

To use µC/OS-II as OS for a microcontroller, the controller needs to provide a system tick – a periodical interrupt (in this case with a period of 1 ms) as a basic clock for the scheduler. The LM3S6965 has a special timer to provide the system ticks.

4.1.2 Board Support Package

To hide more of the hardware details, a board support package (BSP) provided as a part of the port of µC/OS-II for the Cortex-M3 is used. The BSP may be extended corresponding to the systems requirements. Figure 6 shows the initialization of the AS5040 to get the wind direction. Commands with a leading *BSP_* are special commands of the µC/OS-II BSP, while the other commands are part of the driver lib.

4.1.3 Tasks

To create a task, µC/OS-II needs some initialization steps (see Fig. 7). After *BSP_Init*, *CPU_Init* and *OS_Init* the main task is created. In this task, all needed user tasks are created. Then the OS may be started (line 12). After this step the multitasking environment is running.

4.2 Interrupt Service Routine vs. Task

To show the different handling, e.g. to get sensor data, we compare the wind direction using the AS5040 with the heading using the compass (see Fig. 8). The wind direction is a PWM signal which needs to be analyzed by the controller. After the initialization, an interrupt service routine (ISR) is called whenever the logic level on the input pin changes. As result we got two values (see Fig. 9): The whole

```
01: // Timer Initialization
02: // Configure 16-bit periodic timer for wind direction
03: SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
04: TimerConfigure(TIMER0_BASE, TIMER_CFG_16_BIT_PAIR
                              | TIMER_CFG_A_PERIODIC);
05: // Set the prescaler to 50; Timer Tick is 1 MHz
06: TimerPrescaleSet(TIMER0_BASE, TIMER_A, 50);
07: // Enable the timer
08: TimerEnable(TIMER0_BASE, TIMER_A);
09: // Load start value
10: TimerLoadSet(TIMER0_BASE, TIMER_A, 0xFFFF);
11: // GPIO Port D Initialization
12: SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
13: // Use Pin 4 as input for the wind direction
14: GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_4);
15: // Generate an interrupt on both edges of the signal
16: GPIOIntTypeSet(GPIO_PORTD_BASE, GPIO_PIN_4,
                                    GPIO_BOTH_EDGES);
17: // Init the ISR for port D pin 4
18: BSP_IntVectSet(BSP_INT_ID_GPIOD, BSP_GPIOD_IntHandler);
19: // Enable interrupt
20: BSP_IntEn(BSP_INT_ID_GPIOD);
```

Fig. 6 Initialization of the port reading data from the wind direction sensor AS5040

```
01: BSP_Init(); // Initialize BSP.
02: CPU_Init(); // Initialize CPU.
03: OSInit();   // Initialize "uC/OS-II"
04: // Create the Semaphores
05: LCDSem = OSSemCreate(1); //LCD
06: UARTSem = OSSemCreate(1); //UART
07: // Create the start task
08: // The start task creates all other tasks
09: os_err = OSTaskCreateExt(...);
10: ...
11: // Start multitasking (give control to uC/OS-II)
12: OSStart();
```

Fig. 7 Initialization steps of μC/OS-II

period of the PWM signal (*W_Dir_Period*, line 21) and the time, the signal is '1' (*W_Dir_Value*, line 15). The wind direction variable is updated every 1025 μs (Fig. 8 right). This variable may be read by different tasks, which need the wind direction for internal calculations. Other tasks read sensor values over different interfaces when needed (Fig. 8 left).

Corresponding to (1) we can calculate the error-corrected value of the wind direction in degree. On the other hand, the compass HMC6343 is able to deliver

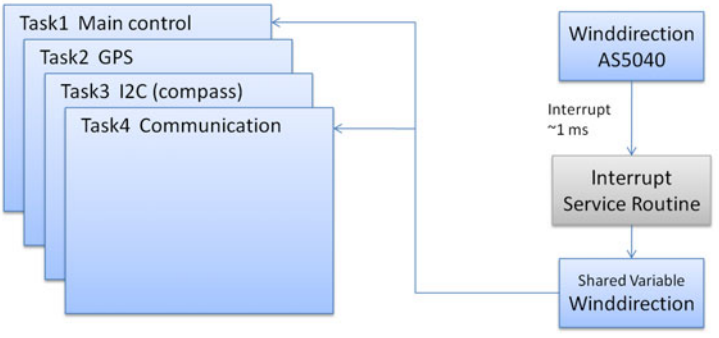

Fig. 8 Tasks reading sensor values (left) vs. interrupt driven calculations (right)

```
01: static  void  BSP_GPIOD_IntHandler (void)
02: {
03:   CPU_INT64U intstat;
04:   // Get GPIO interrupt source
05:   intstat = GPIOPinIntStatus(GPIO_PORTD_BASE, true);
06:   // Clear the GPIO interrupt.
07:   GPIOPinIntClear(GPIO_PORTD_BASE, GPIO_PIN_4);
08:   // Interrupt on Pin 4 (Wind direction)
09:   if (intstat & GPIO_PIN_4)
10:     if (GPIOPinRead(GPIO_PORTD_BASE, GPIO_PIN_4) == 0)
11:     {
12:       // Falling edge
13:       // Save the time since the rising edge
14:       W_Dir_Value =
               0xFFFF-TimerValueGet(TIMER0_BASE, TIMER_A);
15:     } else {
16:       // Rising edge
17:       // Save the time since the last rising edge
18:       W_Dir_Period =
               0xFFFF-TimerValueGet(TIMER0_BASE, TIMER_A);
19:       // Reload the timer
20:       TimerLoadSet(TIMER0_BASE, TIMER_A, 0xFFFF);
21:     }
22: }
```

Fig. 9 Interrupt handler for port D. Detect changes on BIT4 (PWM signal of the AS5040)

the heading directly over the I2C bus with a default rate of 5 Hz. So we can get new values every 200 ms. Figure 10 shows the code for a task reading the data of the HMC6343 once a second. *OSTimeDly* ensures that the task sleeps for 1 s (see line 13). The tasks may be prioritized depending on the requirements. The

```
01: static  void  App_TaskI2C (void *p_arg)
02: {
03:   CPU_INT08U err;
04:   CPU_INT16U heading;
05:
06:   while (DEF_TRUE) {
07:     // get the heading of the boat and display it
08:     heading = BSP_I2C_Read_HMC6343();
09:     OSSemPend(LCDSem, 0, &err);
10:     sprintf(App_LCDLine9, "I2C: %d       ", heading);
11:     OSSemPost(LCDSem);
12:
13:     OSTimeDly(OS_TICKS_PER_SEC);
14:   }
15: }
```

Fig. 10 Task reading the compass data over the I2C bus once per second

combination *OSSemPend*/*OSSemPost* locks the display for this task to ensure a correct viewing of the value (line 9/11). The exclusive access to a shared variable is possible in the same way. Shared components in this system are the display and the Bluetooth UART. They are locked with semaphores.

4.3 Control Software

For our first tests, we used a modified fuzzy control algorithm based on [3]. Currently, only the low-level functions for capturing the sensor data and controlling the actuators are embedded software modules running on the LM3S6965, whereas all high-level software components are implemented on a laptop computer connected via Bluetooth. The laptop gets the sensor data and sends commands back to the boat. Depending of the distance, the laptop is also used as remote control. In the final set-up, all software components are running on the LM3S6965, the laptop is only used to show the status of the boat.

5 Conclusion

We have shown our approach building an autonomous sailboat. In June, the hull of the boat is laminated. The low-level software components are ready, the high-level software components are work in progress. The work on both parts (the boat and the controlling software) gets on, but we will still have to do many tests and improvements. Our goal is the participation of our FHsailbot at the WRSC2011 in Lübeck.

References

1. Ammann, N., Hartmann, F., Jauer, P., Bruder, R., Schlaefer, A.: Design of a robotic sailboat for WRSC/SailBot. In: IRSC 2010, Canada (2010)
2. Labrosse, J.J.: MicroC/OS-II, The Real-Time Kernel. CMP Books, San Francisco (2002)
3. Stelzer, R., Pröll, T., John, R.: Fuzzy logic control system for autonomous sailboats. In: IEEE International Conference on Fuzzy Systems (2007)