"A Real-Time Sailboat Controller Based on ChibiOS"

J. Cabrera-Gámez, A. Ramos-de-Miguel, A. C. Domínguez-Brito, J.D. Hernández-Sosa, J. Isern-González and L. Adler

Instituto Universitario SIANI (www.roc.siani.es),
Departamento de Informática y Sistemas (www.dis.ulpgc.es)
Universidad de Las Palmas de Gran Canaria (www.ulpgc.es), Spain

In: Morgan F., Tynan D. (eds) Robotic Sailing 2014. WRSC/IRSC 2014. Springer, Cham. DOI: 10.1007/978-3-319-10076-0_7

BibTeX:

```
@inproceedings{cabrera_gamez_2015_irsc_2014,
author="Cabrera-G{\'a}mez, Jorge
and de Miguel, Angel Ramos
and Dom{\'i}nguez-Brito, Antonio C.
and Hern{\'a}ndez-Sosa, Jose D.
and Isern-Gonz{\'a}lez, Jose
and Adler, Leonhard",
editor="Morgan, Fearghal
and Tynan, Dermot",
title="A Real-Time Sailboat Controller Based on ChibiOS",
booktitle="Robotic Sailing 2014",
year="2015",
publisher="Springer International Publishing",
address="Cham",
pages="77--85",
isbn="978-3-319-10076-0"
}
```


document created on: 31st March 2021
created from file: irsc_2014_camera_ready_first_page.tex
cover page automatically created with CoverPage.sty
(available at your favourite CTAN mirror)

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

A Real-Time Sailboat Controller based on ChibiOS *

J. Cabrera-Gámez[1,2], A. Ramos de Miguel, A.C. Domínguez-Brito[1,2], J.D. Hernández-Sosa[1,2], J. Isern-González[2] and L. Adler[1]

[1] Instituto Universitario de Sistemas Inteligentes y Aplicaciones Numéricas en Ingeniería (IUSIANI). E-mail address: jcabrera@iusiani.ulpgc.es
[2] Dept. Informática y Sistemas, Universidad de Las Palmas de Gran Canaria, SPAIN.

1 Abstract

This paper presents an ongoing work aimed at the development of a multithreaded open source sailboat controller based on cheap Arduino-compatible hardware and ChibiOS/RT, a small and agile real-time operating system.

The results achieved so far prove that this approach, that relies intensively on the programming resources provided by the real-time multithreaded operating system has produced a more stable, easy to modify and predictable controller, all of them valuable characteristics in the context of a sailboat and particularly in the case of competition environments..

2 Introduction

The Do-It-Yourself (DIY) movement organized around Arduino, Raspberry PI and equivalent platforms has gained a lot of momentum along the last years and it has become an influential movement catching the interest of main microcontrollers vendors. A significant part of this success derives from a smooth learning curve based on a simple programming environment, cheap hardware, free source philosophy and a large and enthusiastic community of users.

The successful utilization of Arduino boards or equivalent low power microcontrollers in the development of autonomous sailboats has been described in a number of papers [11][9][10] and other autonomous sailboat projects available on the web. In fact, features like low price, low power, simple hardware

*This work has been partially funded by Canary Government and FEDER funds under ACIISI ProId2010/0062

interface, powerful microcontroller, PWM outputs, ... make them nearly perfect tools for developing a low level sailboat controller.

Our group have also explored this approach with good results [12]. However, based on our previous experience, we have identified the single-threaded programming model as a main limitation of this approach. Consequently, in this paper we will describe our efforts to develop an autonomous sailboat controller within the programming framework provided by ChibiOS/RT [3], a compact, agile and open source real time operating system, capable of running on a large set of microcontroller boards, including AVR and ARM based Arduino boards. The underlying hypothesis is that the resulting system will win in elegance, simplicity, robustness, performance and scalability.

The organization of the paper is as follows. First, we will make a brief enumeration of the hardware components of the controller. Then, we will introduce ChibiOS, its basic features and the main resources it provides for multithreaded programming. The central section of the paper will describe the design of the software architecture and the main components of the sailboat controller. The final section will summarize the results achieved so far and it will provide some hints about future lines of development.

3 The hardware

The hardware part of the controller is based on the Arduino DUE board. This 110mm x 55mm board integrates an Atmel SAM3X8E ARM Cortex-M3, a 32-bit 84 MHz ARM microcontroller featuring 96KB of RAM, 512KB of Flash memory, 4 serial ports, CANBUS, SPI, 2 I^2C (TWI) buses, an in-chip real-time clock, watchdog and low power modes. Furthermore, the board integrates other interesting modules, such as 12 analog inputs, 54 DIO (12 of them can be used as PWM outputs), 2 DACs and 6 timers. Interestingly, the board includes also a USB OTG port and DMA controller. It operates at 3.3V and can provide 800mA through on board 3.3 and 5V regulators.

The rest of the elements that conforms the hardware of the controller includes a 5Hz Telit JN3 GPS receiver, a XBee 868 Pro radio module, a 10 DOF IMU (3-axis magnetometer, 3-axis accelerometer, 3-axis gyro and a barometer) and a solar power recharging board. For interfacing the XBee radio module a XBee wireless Arduino shield containing a micro SD reader is used.

The cost of the whole system (an Arduino Due, an Arduino XBee Wireless shield, a XBee 868 Pro module, a Lipo Rider Pro, a Waspmote GPS board and a DFRobot 10 DOF MEMS IMU) is approximately 185€. It weighs less than 80 grams, excluding batteries, and fits within a volume of 100 x 55 x 60 mm^3.

This prototype also includes a legacy electronic compass board [1], which includes a temperature sensor and pitch and roll inclinometers. This board

has been conserved to compare its performance with the performance of the low cost 10 DOF IMU.

This hardware is intended to replace a similar system that has been used on board the ATIRMA sailboat, a carbon fiber One Meter class vessel with mainsail and foresail (LOA[3]: 100 cm; beam: 24.5 cm; draft: 14 cm; sail area: 0.61 m^2; displacement: 4.3 kg; mast height: 1.6m). A complete description about the ATIRMA can be found in [12] and there are also some videos available [6].

4 ChibiOS/RT

The programming model normally used with Arduino platforms follows a single threaded iterative programming model based in an initial setup stage followed by an iterative function. While this approach may be convenient for solving simple tasks, it imposes serious limitations for programming real-time controllers that must execute several control loops at different frequencies, and programming a sailboat controller is a good example of this.

Programming a sailboat controller locates in that context, where the availability of a real-time multithreaded programming infrastructure constitutes a definite advancement in this context. ChibiOS/RT [3] is a very compact, open source real-time operating system (RTOS) for embedded microcontrollers that supports an ample set of architectures (ARM, AVR, MSP430, ...) and platforms. This RTOS offers a very compact preemptible kernel with a very fast context switching capability. The API includes the set of primitives normally found in other multithreaded programming environments, like threads management functions, virtual timers, semaphores, mutexes, condition variables, messages, mailboxes, event flags or queues.

4.1 ChibiOS/RT scheduler

A main component of ChibiOS/RT is its scheduler, which implements a round robin scheduling strategy combined with priority levels. A round robin rotation may happen in any of these cases [4]:

- The thread in execution invokes chThdYield(). This permits the next thread at the same priority level to execute.
- The thread in execution goes into a sleep state. When this thread is awaken it is scheduled behind any other ready thread at the same priority level.
- The thread holding the CPU is preempted by a higher priority thread. When this happens the thread is reinserted in the ready list behind any other thread at the same priority level.

[3] Length Over All

- Only when the ChibiOS/RT configuration parameter CH_TIME_QUANTUM is set to a value greater than zero, the thread in execution has exhausted its time quantum and there is another thread ready for execution at the same priority level, the executing thread is preempted and reinserted in the ready list behind any thread at the same priority level.

In the default configuration, CH_TIME_QUANTUM parameter is set to 20ms. ChibiOS/RT recommends to set it to zero and design the application to perform the so called "cooperative scheduling", i.e. make the threads go to sleep or invoke chThdYield() voluntarily to schedule a new thread for execution. This strategy makes the kernel faster and smaller, while it does not prevent the utilization of multiple threads at the same priority level.

5 Software Architecture

The software architecture is made up of several threads running at different frequencies, each one implementing a specific functionality or service, and it is roughly organized into three levels attending the characteristic frequency of operation of each thread.

- Low level threads that cope with the low level services, like attending the reception of radio packets, logging data on the micro SD, reading sensors or adjusting the rudder and sails, that demand high frequencies.
- Medium level threads. This set includes threads that execute with periods in the range of 1 second, like the bearing selection component, or the fuzzy controller in charge of sails and rudder.
- High level threads. There is only one thread at this level and is the thread that monitors the execution of a specific activity along time.

The sensors onboard the sailboat produce data at different rates. For example, the wind vane is read at a high frequency, typically 20Hz, and then filtered using a median filter. The TCM2-50 inclinometers and compass produce data at a programmable rate, currently set at 8Hz. The GPS receiver can provide NMEA messages at a maximum rate of 5Hz, even though it can be programmed at 1Hz for low power operation.

Access to data registers is protected using specific mutexes. Data registers are marked as volatile and only have one writer or producer, the thread that updates it, and one or several readers or consumers. Producers systematically timestamp the data, so that the consumer always knows the age of the datum.

The fuzzy control thread acts as a skipper in charge of monitoring the navigation state of the sailboat in order to make it safe and respect the marked course. Normally, it runs at a specific frequency, typically around 4Hz or lower but it can be awaken by any of the data providers whenever a significant is detected, e.g. excessive heeling or any out-of-range data event. This functionality is implemented using a condition variable with timeout (denoted as

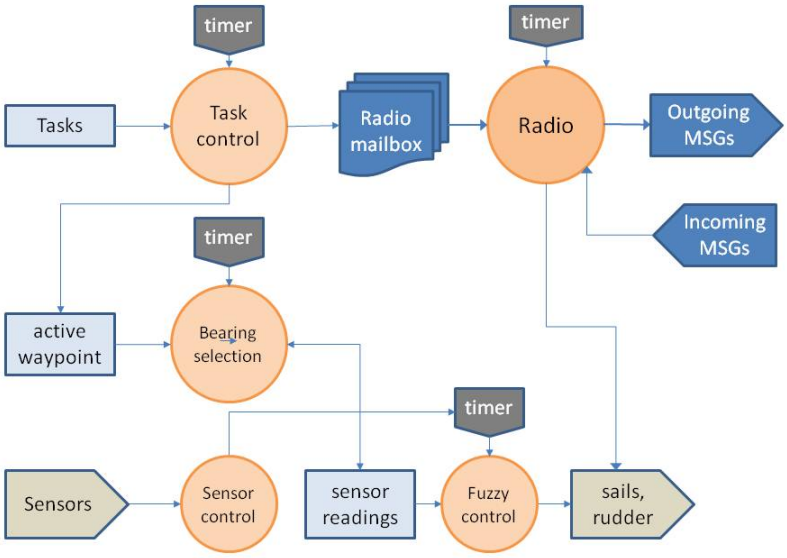

Fig. 1: Main components of the software architecture.

"timer" in Fig.MainComps. The timeout is set to make this thread run with the predetermined period, but the condition variable can be signaled by data producers to accelerate the corrective action. The skipper implements a fuzzy controller [8] using EFLL[2] to control the sails and the rudder.

The pilot is the thread in charge of selecting the optimal bearing given a destination (waypoint coordinates) and wind conditions[7]. Normally, it executes every 5 seconds when sailing in a stable course, but its execution can be inhibited if the boat is turning, until she completes the maneuver. When suspended at a timed condition variable, it can be signaled to execute immediately whenever a new destination has been set.

The radio thread is one of the most important components of the system, being in charge of the communications between the sailboat and the base station. Its operation is driven by outgoing messages, that arrive at this thread's mailbox, and incoming messages. Incoming messages are used to set the system into two possible control modes: remote and autonomous. In the remote control (R/C) mode the rudder and sail servos are operated remotely and the radio packets are directly translated into servo commands. The other mode is the autonomous mode, in which the sailboat navigates under its own control. The transition between these two modes is done upon the reception of a specific radio packet. The radio thread cycles continuously between the transmission of outgoing messages, received through the mailbox, and the pro-

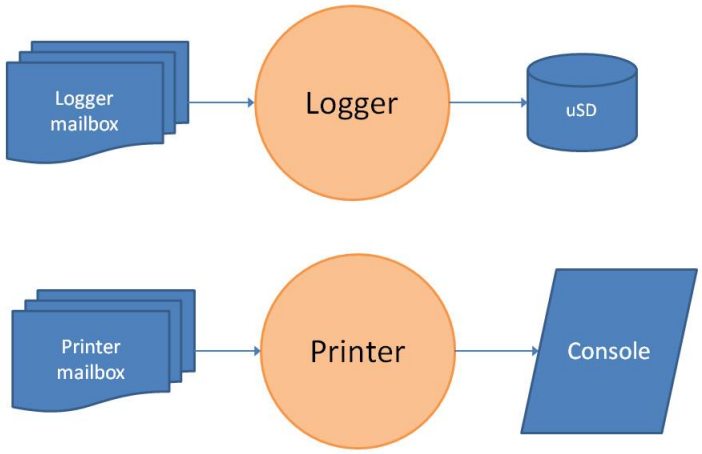

Fig. 2: Printer and Logger components.

cessing of incoming messages. The reading of the mailbox is non-blocking and it only yields the processor if there are not pending incoming packets. Under the R/C mode, this thread has the highest priority to guarantee an efficient radio communication service. But, even in this mode, packet dispatching is done very fast and the system does not block because the thread performs periodic yields of its processor quantum.

Additionally, mailboxes are used to interface some real-time components to other non real-time subsystems. A clear example is the logger thread Fig.PrinterLogger, responsible of saving messages delivered through the mailbox into the micro SD card. Another example is the "printer" service, used for debugging by any thread that wants to print out coherent messages in the serial monitor console. Any thread that wants to print any message may deliver it at the printer mailbox. This service warranties that messages produced concurrently at different threads are not mangled at the console. This thread is executed only during debugging.

In the current implementation all threads share the same priority level. The only exception is the one managing the radio (see Fig.MainComps). This particular thread promotes itself to a higher priority level whenever in R/C mode. Typically, threads adjust their frequency of execution using a simple scheme like this:

```
msg_t any_thread(void *param)
{
  systime_t next_time = chTimeNow();
```

```
  while (TRUE)
  {
    // Compute the next deadline
    next_time += MS2ST(period);
    do_something();
    chCondWaitTimeout (condVar, next_time);
  }
}
```

6 Discussion and conclusions

This paper has described ongoing work for porting our former sailboat controller [12], developed and tested on a 1281-ATMega (8MHz, 8Kb RAM, 128Kb Flash) processor, to a more powerful ARM-based Arduino DUE board (84MHz, 96KB RAM, 512KB Flash). The availability of better computational resources allows not only a substantial improvement in processing speed, but also to use a real-time operating system. The ChibiOS real time operating system has been selected to provide improved programming facilities that include multithreading, virtual timers, mutual exclusion mechanisms and many other features commonly found in other real-time programming contexts. Specifically, we use the port of ChibiOS/RT 2.6.0 available for the DUE board from [5].

Within this new design it is quite simple to add or remove monitoring or control threads, that must execute at specific frequencies, without the need of interleaving them with the execution of other elements in a monolithic system. Threads can be ranked by priorities in accordance to its criticality, even though care must be taken to impede blocking the execution of lower priority threads.Once developed, the resulting controller is now simpler, more stable, robust, predictable and easier to modify than it was formerly.

The programming resources contributed by ChibiOS, specially the availability of mailboxes, have allowed the definition of services like the radio communication, the logging system of the so called "printer" services that can be used by any of the components of the system.

At this stage of development the sailboat controller uses some standard Arduino libraries that have been tuned to avoid the utilization of the delay() function, substituted by equivalent chThdSleep() calls, or abusive utilization of dynamic memory.

Next steps will be directed towards testing the performance of this controller on board the ATIRMA. On a medium term, work is planned to address a better exploitation of some unexplored ChibiOS/RT capabilities, like HAL drivers, low power optimizations or some ARM processor features.

References

1. PNI's legacy TCM2.5 electronic compass manual. http://www.pnicorp.com/download/347/99/TCM2.52.6Manualr09.pdf
2. EFLL fuzzy logic library. https://github.com/zerokol/eFLL
3. ChibiOS/RT Home page. http://www.chibios.org/dokuwiki/doku.php?id=start
4. ChibiOS tutorial on round robin scheduling. http://www.chibios.org/dokuwiki/doku.php?id=chibios:kb:round_robin
5. RTOS Libraries available for Arduino. https://code.google.com/p/rtoslibs/downloads/list
6. ATIRMA's Blog. http://velerorobot.blogspot.com.es/
7. Stelzer, R., Pröll, T.: Autonomous sailboat navigation for short course racing. Robotics and Autonomous Systems, **56**, 604-614(2008).
8. Stelzer, R., Pröll, T., John, R.I.: Fuzzy Logic Control System for Autonomous Sailboats. FUZZ-IEEE2007, pp 97–102(2007).
9. Alvira, M., Barton, T.: Small and Inexpensive Single-Board Computer for Autonomous Sailboat Control, Robotic Sailing 2012, pp 105-116, Springer, 2013.
10. Koch, M., Petersen, W.: Using ARM7 and C/OS-II to Control an Autonomous Sailboat Robotic Sailing 2011, pp 101-112, Springer 2012.
11. Bruget, K., Clement, B., Reynet, O., Weber, B.: An Arduino Compatible CAN Bus Architecture for Sailing Applications, in Robotic Sailing 2013, Proceedings of the 6th International Robotic Sailing Conference, F. Le Bars and L. Jaulin (Eds), Springer, 2013.
12. Cabrera-Gámez, J., Ramos de Miguel,A., Domínguez-Brito, A.C., Hernández-Sosa, J.D., Isern-González, J., Fernández-Perdomo, E.: An Embedded Low-Power Control System for Autonomous Sailboats, in Robotic Sailing 2013, Proceedings of the 6th International Robotic Sailing Conference, F. Le Bars and L. Jaulin (Eds), Springer, 2013.