Small Size Holland
Team Description Paper
RoboCup 2017

Álvaro Espinosa, Thijs Geurkink, Jelle Holtslag, Rogier Heeg,
Nico Kleinreesink, Mark Lefering, Thomas Hakkers, Jeroen de Jong,
Ryan Meulenkamp, Seif Megahed, Joost Overeem, Maarten Overeem,
Rimon Oz, and Joep Tool

Life Science, Engineering, and Design, Saxion University of Applied Sciences,
M.H. Tromplaan 28, 7513 AB Enschede, The Netherlands
robocup.saxion@gmail.com
http://www.SmallSizeHolland.com/

Abstract. This paper outlines the major design decisions, implementation, and test results by Team Small Size Holland since participating in Robocup 2016 in Leipzig, Germany. The technical issues which resulted in malfunctioning robots have been analyzed and solved in a redesign of the robot's motherboard and a change in motor alignment. Progress was made on a custom OS for the embedded system and the strategy software data processing engine has been replaced with $reactive - streams - commons$.

Keywords: Robotics, Easy Disassembly, Omni Wheels, Dribbler, CPU, Modular, Reactive, Streams

1 Introduction

Small Size Holland (Team SSH) from Saxion University, the Netherlands, is a Small Size League soccer team consisting of students who participate in the RoboCup for a single semester during their minor or specialisation as part of their bachelor studies. This research builds on the research of previous years. This paper outlines the changes made to the designs and implementations described in the team's previously published papers ([1] and [2]).

At the tournament in Hefei SSH played with the first generation of game-ready robots (G2). The team made the decision to create a new mechanical and electrical design for the robots based on the knowledge and experience gained while working with G2. Between September 2015 and July 2016 two teams worked on designing, prototyping, testing, and producing SSH's third generation of robots (G3). Due to both organisational and technical issues the G3 robots were not suitable (nor safe) to be used during games and despite the effort of the team the robots were not ready at the beginning of RoboCup 2016.

The new team, which started in September of 2016, has been working on a redesign and reimplementation of the mechanical and electrical systems; reverting to (improved) subsystems from G2 where necessary. The only remaining step to take is system integration tests before G4 is game-ready.

This paper discusses the major changes in hardware and the software which resulted in SSH's fourth generation of robots (G4). Chapter two discusses the mechanical layout of the robot. The third chapter describes the new (modular) motherboard. Chapter four describes *LegOS*: SSH's custom OS for the embedded system. Chapter five describes *zosma*: the next iteration of SSH's strategy software.

2 Mechanical Design

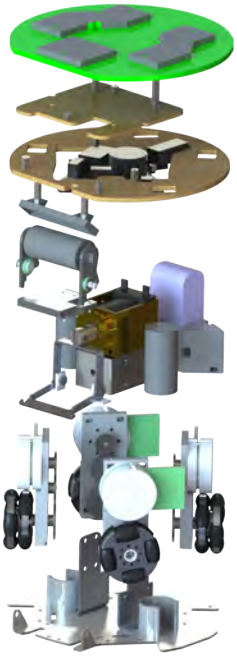


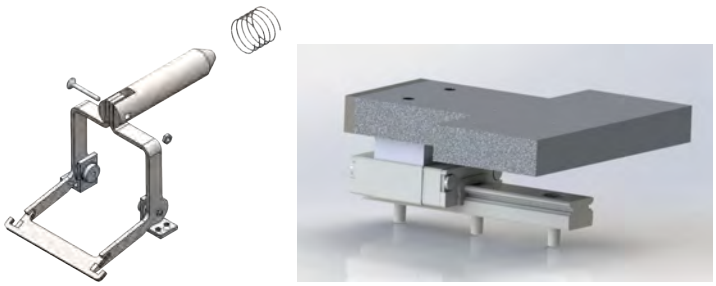
Fig. 1: Exploded view of the G4 robot

The addition of a fourth motor to G3 reduced the amount of space inside of the robot which is used for the chipping- and kicking-solenoids, the solenoid capacitors, and the dribbler. Because of this the decision was made to wind custom flat solenoids for G3 and to suspend the motors above the wheels. System tests indicated that these solenoids performed worse (ie. kicking and chipping velocity) than the solenoids in G2. Furthermore, the configuration of the four

wheels (ie. the angle and radial distance) proved incompatible with the wheels used resulting in a significant loss of energy to friction.

To solve these issues the decision was made to redesign the interior of the robot. First, the decision was made to make the system compatible with the G2 solenoid subsystem until a provably better subsystem has been built. Second, the decision was made to improve on the design of the G2 chipper-and-kicker in such a way that it remained compatible with the G2 solenoid subsystem. Third, the decision was made to redesign the wheel-and-motor subsystem to address the issue of space in the interior of the robot. The team currently has eight mechanically working prototypes weighing an average of $2.8kg$.

2.1 Chipper and Kicker



(a) G4 chipper design (b) G4 kicker design

Fig. 2: G4 chipper-and-kicker design

The chipper and kicker for G4 were redesigned to ensure compatibility with the G2 solenoid subsystem. This meant changing from a chipper with a push-action to a chipper with a pull-action as in G2. The chipper plate is attached to the lever by sliding it into position and securing it with a simple spot weld. The design was also modified to account for the reduction in space due to the addition of the fourth motor. The new kicker is attached to a linear guide in order to reduce the displacement of the kicker perpendicular to the force being applied to the ball; this ensures that the total force applied to the ball is maximized.

2.2 Wheels and Motors

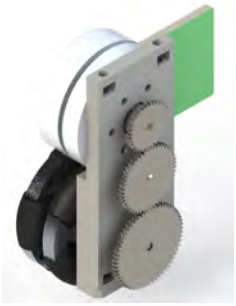


Fig. 3: G4 wheel-and-motor assembly

The addition of the fourth motor resulted in reduced space in the interior of the robot. The wheel-and-motor assembly of G4 differs to the assembly in G3 in that the wheel-and-motor holder was replaced with a 3D-printed and flatter equivalent. This resulted in enough space being freed up to align the motors properly and fit in the solenoid and chipper-and-kicker subsystems.

3 Electrical Design

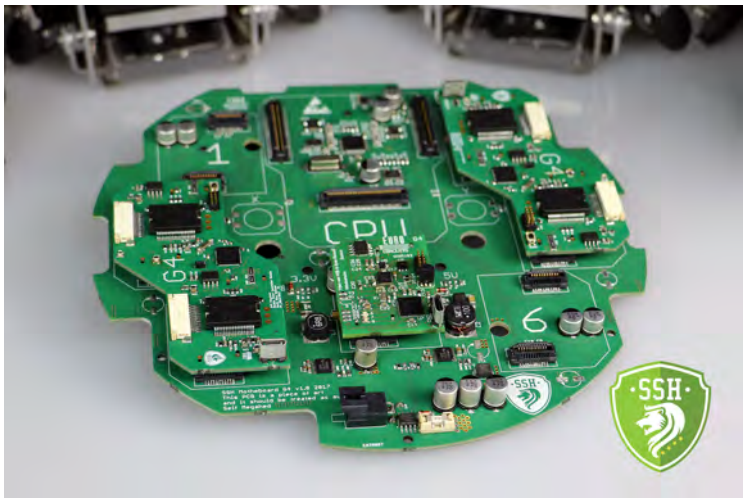


Fig. 4: G4 assembled motherboard

The G3 motherboard was designed with modularity in mind. This meant that the CPU-unit, WiFi-unit, and motor driver IC's were replaceable. Tests with the G3 motherboard indicated that the controlling circuit (based on the Reference Schematic in [3], p. 26), which is a speed control, contained noise which increased with continued usage of a motor driver IC. The cause of this noise was a defect in the separation between power and signal grounds which resulted in spikes on the signal lines which in turn burned the motor drivers.

To solve these issues the decision was made to design a G4 motherboard with a tested signal and ground separation and in which entire motor drivers (ie. including the controlling circuit) could be replaced. Since the G3 chargeboard functioned properly there were no changes made to this subsystem. The team currently has two working motherboards including motor controller and CPU/WiFi units and is expected to complete eight sets more by the end of March 2017.

3.1 Motherboard

The G4 motherboard is a six-layer design which makes use of Molex SpeedStack connectors to allow up to seven modules (excluding the CPU/WiFi unit) to be attached. Each of these modules are on a common bus for I^2C, SPI, and USB. The motherboard features an USB hub which joins all the modules so that they can be programmed with ease. The connectors provide additional GPIO pins so that the CPU/WiFi unit can override the behavior of the processing units on the modules in case of technical failure. Furthermore, the motherboard contains three power supplies ($3.3V$, $5V$, and $16V$).

3.2 Modules

All of the modules have temperature and current sensors to determine the performance of these modules under load. Currently SSH has developed motor controller, sensor, and dribbler modules for the robot. All these modules feature an Atmel SAML21G18 microprocessor clocked at 48MHz. Currently work is being done on creating a chipper-and-kicker module which may potentially replace the G3 chargeboard before RoboCup 2017.

Communication The communication buses on the motherboard were designed using microstrip and stripline calculations. Because of this the buses all have controlled impedance and function properly at high frequencies. The communication bus has been completely isolated from the power planes which feed the inductive load (ie. the motors) so that no interference occurs. The six layer design of the motherboard allows for smooth ground return paths with a negligible loop area. The communication bus itself consists of SPI, I2C, and USB so that each module can make use of the most appropriate protocol. The modules which are currently on the motherboard make use of SPI exclusively, but future plans include USB-support for these modules so that they can be accessed as USB-devices.

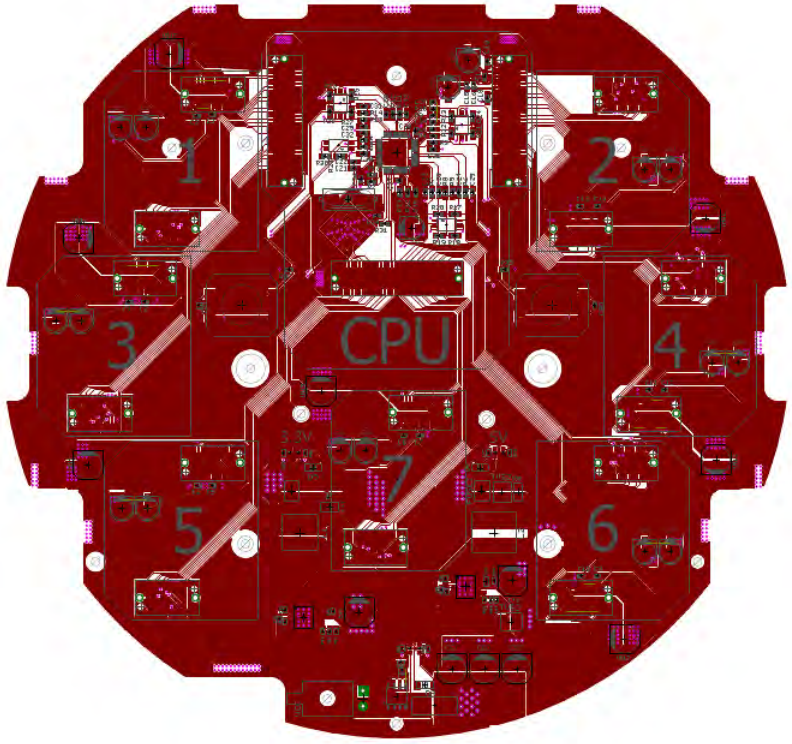


Fig. 5: G4 motherboard layout

3.3 Motor Controllers

The G4 motor controller is a dual-module four layer board which consists of one power, one ground, and two signal layers. The signal layers are the outer layers to ensure an equivalent impedance reference from both power planes. The motor drivers used in these controllers are the same drivers which were used in G2 and the speed control has been replaced with a direct control by the embedded system (via DAC).

4 LegOS

LegOS is the custom real-time OS which runs on the modules SSH has currently developed for the G4 motherboard, and in specific the SAML21G18. The base of the OS contains a model for generic drivers and applications. Furthermore, several drivers have been written for peripherals, such as SERCOM (serial communication such as SPI and I^2C), USB, ADC and DAC, and PWM.

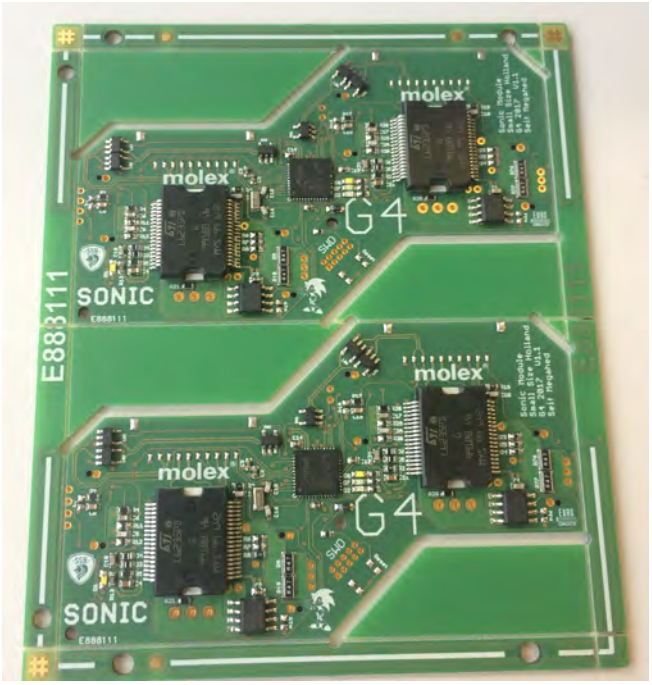


Fig. 6: G4 motor controllers

The communications module on the motherboard makes use of an ESP12 for communicating with the server running the strategy software. SSH has worked on a port of *LegOS* for the ESP12 which contains drivers for 802.11b/g, UDP and TCP, and transcoding protobuf.

4.1 Applications

One of *LegOS*' major design decisions was to abandon a threading model in favor of an event-system based on callbacks. This means that applications written for *LegOS* consist of a single-threaded application which chains functionality using callbacks. Applications built for *LegOS* are purposely kept small and consist of mapping data received from one of the communication buses to a format which is understood by peripherals, such as the motor driver IC, and subsequently transmitting that data to the peripheral. Applications which have been built for *LegOS* so far include *Django*, the chipper-and-kicker application, *Sonic*, the motor controller application, and *Aewop*, the WiFi application. Progress is being made on a separate application for dribbler functionality.

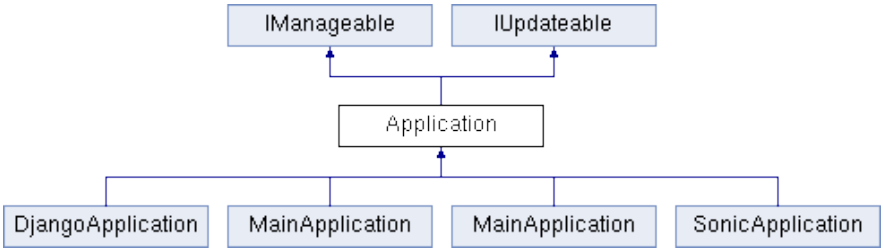


Fig. 7: *LegOS* class diagram for applications

4.2 zosma

Since RoboCup 2016 work has been progressing on SSH's strategy software. The decision was made to extend *zosma* in such a way that it could be reused as a generic strategy system for playing games. The software has been split up into several modules to achieve this goal. $zosma - game$ is the module which is responsible for modelling games. $zosma - ssl$ is an implementation of $zosma - game$ for Small Size League Soccer. *zosma* makes extensive use of $reactive - streams - commons$[4] to allow for high data throughput. In comparison to the previous iteration of the software, which was developed using the functional reactive paradigm, *zosma* is a purely reactive design. Furthermore, *zosma* contains a module named $zosma - math$ which, in contrast with $apache - commons - math$, is built using $nd4j$[5] to allow for GPU accelerated computations.

4.3 zosma-game

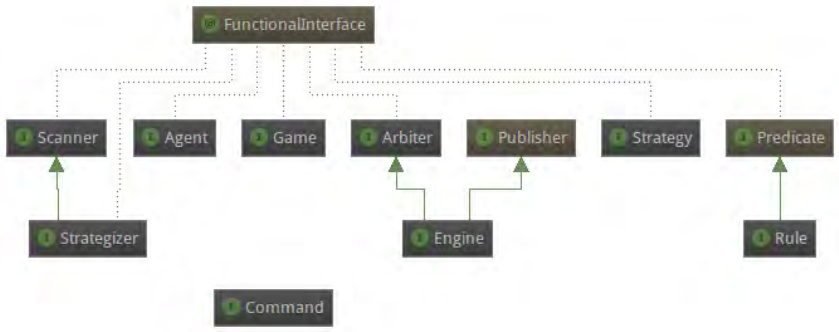


Fig. 8: $zosma - game$ class diagram

zosma − game consists of several interfaces to make programming games easier. In the context of the Small Size League the implementations of *Agent*, *Command*, and *Game* correspond to immutable state representations of the robot, a command which is used to control it, and the game itself (including detection and referee data). A *Strategizer* is a *Scanner*, ie. a bifunction which behaves as a reducer. For every incoming piece of data to a *Strategizer* the bifunction will take as its argument the previously returned value and the incoming data object. As a result of this all data objects in *zosma* are immutable. The *Arbiter* is a functional interface which takes as its argument a set of objects and returns a single object of the same type. The *Arbiter* is used to arbitrate between competing strategies.

Rules A *Game* is defined to be a functional interface which returns a set of game rules. As a result of this it is possible for the system to detect when rules are being violated and allows *Strategizer*s to act on rules. Currently work is being done to integrate the autoref with this system and this is expected to be completed before RoboCup 2017.

Engines Engines, within the context of *zosma*, are *Arbiter*s which contain a set of *Strategizer*s. Engines are used to group *Strategizer*s which perform similar tasks such that a more optimal strategy can be calculated. The arbitration functionality is then used to determine which strategy (produced by which *Strategizer*) is the best solution. The implementation of the *Arbiter* is purposely left open, and so programmers may decide whether they use discrete, probabilistic, or custom methods to determine the relative quality of computed strategies and decide on a final strategy before it is broadcast to the robot.

4.4 zosma-ssl

zosma − ssl is an implementation of *zosma − game* specifically for Small Size League soccer. This module contains implementations of the (extended) Kalman filter(s) and particle swarm filters for state estimation. Furthermore, *zosma − game* contains several engines dealing with state estimation, computing strategies, and detecting hotspots on the field (ie. spots where the measurement covariance is significant).

A *SoccerAI* is a program which can play a Small Size League soccer game and implements a *SoccerEngine*. A *SoccerEngine* is an engine which maps *Strategizer*s to *Agent*s controlled by those *Strategizer*s. Furthermore, a *SoccerEngine* has the functionality of *SoccerOperations* which is a group over the Euclidian plane (the domain of the state space) allowing the *SoccerEngine* to compute distances and other metrics.

4.5 zosma-torch

While the previous iteration of the software had a 3D-field it is impractical (and unnecessary) to do 3D-computations on the same system which is calculating

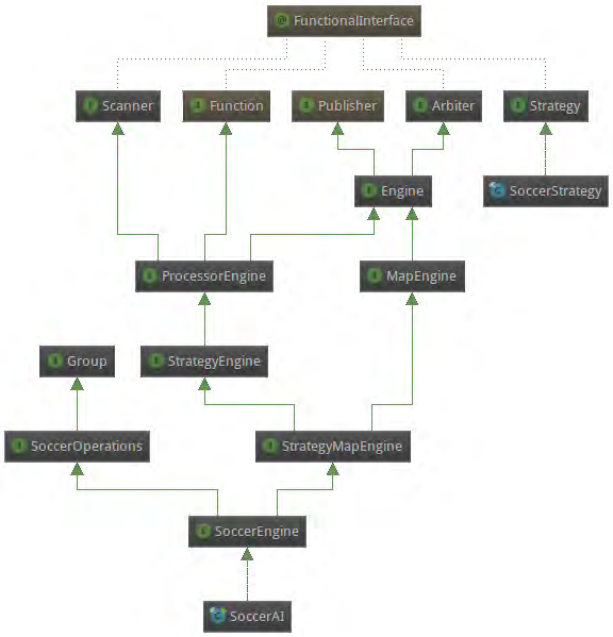


Fig. 9: *zosma − ssl* class diagram

strategies. Because of this the team decided to create a command-line interface to *zosma* which can be used to configure the application and display system feedback. *zosma − torch*, the module which accomplishes this, is built using *lanterna*[6].

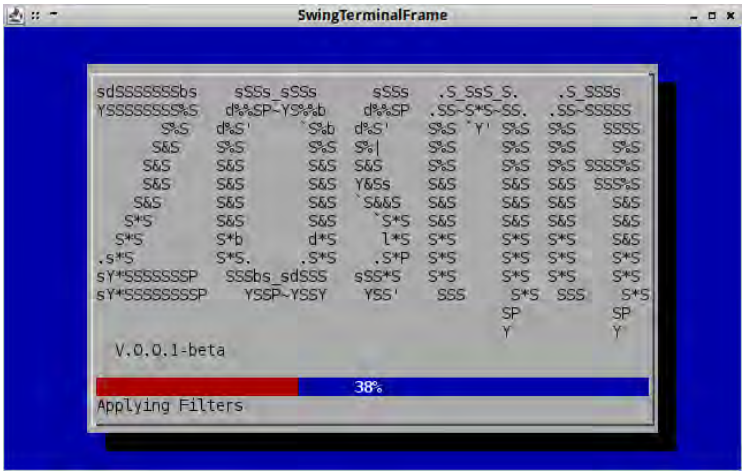


Fig. 10: *zosma − torch* splash screen

References

1. Emmerink, T., Berg, R. van den, Overeem, J.W., Hakkers, T., Jong, J. de, Van Ommeren, J., Meulenkamp, R.:SSH Team Description Paper for RoboCup 2015. Enschede (2015)
2. Overeem, J., Oz, R., Schrijver, N., Jong, J. de, Hakkers, T., Meulenkamp, R., Meijerink, J., van Deth, N., Teterissa, M., Dickers, S., Elmas, E., Lefering, M., and Berg, R. van den :SSH Team Description Paper for RoboCup 2016. Enschede (2016)
3. L6235 - DMOS driver for 3-phase brushless DC motor. STMicroelectronics NV. (2014)
4. *reactive − streams − commons*, reactor. Retrieved February 20, 2017, from https://github.com/reactor/reactive-streams-commons
5. *nd4j*, deeplearning4j. Retrieved February 20, 2017, from https://github.com/deeplearning4j/nd4j
6. *lanterna*, mabe02. Retrieved February 20, 2017, from https://github.com/mabe02/lanterna