A proposed software framework aimed at energy-efficient autonomous driving of electric vehicles

Jose-Luis Torres[1], Jose-Luis Blanco[1,⋆], Mauro Bellone[2], Francisco Rodríguez[3], Antonio Giménez[1], and Giulio Reina[2]

[1] Dept. of Engineering, Universidad of Almería, E-04120 Almería, Spain
[2] Dept. of Engineering for Innovation, Universitá del Salento, I-73100 Lecce, Italy
[3] Dept. of Computer Sciences, Universidad of Almería, E-04120 Almería, Spain

Abstract. This paper describes the development of an electric car prototype, aimed at autonomous, energy-efficient driving. Starting with an urban electric car, we describe the mechanical and mechatronics add-ons required to automate its driving. In addition, a variety of exteroceptive and proprioceptive sensors have been installed in order to obtain accurate measurements for datasets aimed at characterizing dynamic models of the vehicle, including the complex problem of wheel-soil slippage. Current and voltage are also monitored at key points of the electric power circuits in order to obtain an accurate model of power consumption, with the goal of allowing predictive path planners to trace routes as a trade-off between path length and overall power consumption. In order to handle the required variety of sensors involved in the vehicle, a MOOS-based software architecture has been developed based on distributed nodes that communicate over an onboard local area network. We provide experimental results describing the current stage of development of this platform, where a number of datasets have been already grabbed successfully and initial work on dynamics modeling is being carried on.

Keywords: Autonomous vehicles, Mobile robotics, Software architecture

1 Introduction

While autonomous driving in realistic situations remains a challenging problem, the DARPA challenge [1] and the Urban Challenge in 2004 and 2007 [2] have clearly shown that such a challenge could reasonably be addressed according to the recent progresses in the field of perception and autonomous navigation for unmanned vehicles. The Carnegie Mellon University *Tartan Racing* team won the urban challenge in 2007 using a hierarchical control system for planning and sensing [3]. The keystone of their winning approach is the convenient combination between on-board mechatronic system and software architecture. Their

⋆ Corresponding author, email: jlblanco@ual.es

Draft version: To appear in 2014 International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR 2014). The final publication is available at Springer website.

vehicle incorporates a variety of lidar, radar and visual sensors to safely navigate urban environments, as well as a software architecture decomposed into five broad areas: mission planning, motion planning, behavior generation, perception and world modeling. Another team, the VisLab group, achieved 15,926 km of autonomous driving in 2010, driving from Parma (Italy) to Shangai (China) using a Piaggio Porter Electric Power van [4, 5]. The sophisticated vision system that equips the VisLab's van including cameras and laser scanners allowed the autonomous driving even in critical scenarios and challenging roads [6].

Although autonomous vehicle technology is not completely mature yet, it has been attracting economic and industrial interest for years, and commercial cars include increasing levels of autonomy year after year. On one hand, social implications of such a huge revolution will change our way to see the transportation systems increasing the quality of our life. On the other hand, vehicles will have to be equipped with a large number of sensors that are still expensive and, most importantly, safety and reliability are mandatory, still open requirements.

Current experimental autonomous vehicles require sophisticated control algorithms as well as a large number of sensors, hence the need to employ a number of embedded computers and specific software architectures aimed at distributed sensing and processing capable of real-time performance. A variety of such architectures can be found in the literature and the industry. For example, the MIT DARPA Urban Challenge Team developed a set of libraries and tools for message passing and data marshalling called Lightweight Communications and Marshalling (LMC) [7]. Their work was targeted at real-time systems such as the experimental vehicle employed for the challenge. In such situations, high-bandwidth and low latency are critical issues. Recently, the Robotics Operative System (ROS) [8], a new software architecture for mobile robotics, is getting increasingly popular among research labs and industries. ROS provides both, a middleware for structured communications between processing nodes and a set of ready-to-use software nodes for many specific tasks usually found in robotics.

Our electric car architecture is based on Open Mobile Robot Arquitecture (OpenMORA) [9], originally developed by MAPIR lab, University of Málaga, and at present also co-maintaned by the authors of this work. OpenMORA relies on two open-source frameworks: MOOS [10], and Mobile Robot Programming Toolkit (MRPT) [11]. MOOS is a middleware for distributed robotic architectures based on the publish-subscribe (pub/sub) pattern. It comprises a core C++ library and a set of tools for managing and monitoring so-called *communities* of distributed modules. Key advantages of MOOS against other alternatives are its simplicity and its suitability to attain sub-millisecond round-trip message passing. At the core of MOOS is the idea of a minimalist data-type middleware (i.e. only double numbers and text strings are allowed to be exchanged between modules), hence that transmitting complex data types (e.g. laser scans or images) over MOOS implies custom implementations of data marshalling. For that purpose we use MRPT, which provides data structures for the most common autonomous vehicle sensors along with efficient serialization and deserialization mechanisms. MRPT is also employed for low-level interfacing with most sensors,

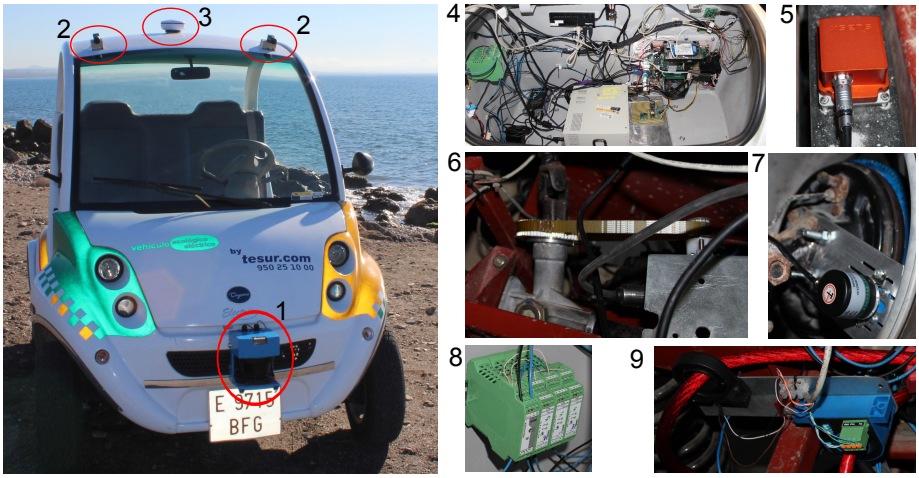


Fig. 1. (a) Autonomous car prototype, with visible sensors marked in red, and (b) close-ups of the rest of sensors. Labeled devices are: (1) Sick LMS-200, (2) PGR Flea3 USB3 cameras, (3) GPS antenna, (4) embedded computers in the car trunk, (5) IMU system, (6) steering-wheel actuator and encoder, (7) rear wheels encoders, (8) voltage and (9) current sensors.

performing sequential Monte-Carlo (SMC) vehicle localization [12] and providing autonomous reactive navigation control [13]. Together, MOOS and MRPT have a small footprint and allow modules to be written in C++ and run in a number of platforms and operative systems (i.e. Windows, OSX, GNU/Linux, GNU/kFreeBSD).

The rest of the paper is organized as follows. Section 2 describes our custom autonomous car prototype including its sensors system, then section 3 introduces the proposed software architecture. We finalize with Section 4 providing on-road experiments and drawing some conclusions.

2 Prototype description

An electric car prototype has been developed at the Automatic, Robotics and Mechatronics (ARM) research group at University of Almería with the ultimate scientific objective of studying novel localization, mapping and energy-efficient path planning techniques. The prototype, presented in Figure 1, consists of a urban electric car that has been adapted for automated control. Table 1 provides a summary of its most relevant mechanical and electric characteristics. The prototype features a 48 volt DC-motor and 8 gel-batteries, selected thinking of the future requirements of energy efficiency. Its batteries system ensures an autonomy of 90 km at a maximum travel speed of about 45 km/h. In its current form, the prototype features both: full manual control, or autonomous mode. For

Table 1. Summary of vehicle mechanical and electrical characteristics

Mechanic characteristics	**Value**
length × width × height	2680 × 1525 × 1780 mm
Track	1830 mm
Front/rear wheelbase	1285/1260 mm
Weight without/with batteries	472/700 kg
Electric characteristics	**Value**
DC motor XQ 4.3	4.3 kW
Batteries (gel technology)	8 × 6 V −210 Ah
Autonomy	90 km

the implementation of such automatic control, the car required the installation of a large number of sensors and actuators as well as a computer architecture capable of properly handling all the dataflow in real time (refer to section 3). Figure 2 shows an illustrative scheme of the embedded computers and sensors architecture. Two embedded computers PC1 and PC2 constitute the processing core of the vehicle, gathering information from each sensor and giving control commands to each actuator.

During the prototype construction, the first task carried out was the automation of typically human-actuated signals, namely: steering and throttling. The former was addressed by implementing a so-called steering-by-wire system, which consists of a 12 volt DC-motor coupled to the steering column by means of an electromagnetic clutch. Such system is governed by a custom microcontroller-based steering controller which controls the motor by means of a PWM signal. Furthermore, the controller reads back the angular position of the steering column by means of an incremental optical encoder attached to it. In order to allow recovering the manual steering mode, the controller is capable of uncoupling the motor and the steering column by actuating over the electromechanical clutch.

Regarding the automatic propulsion, we generate a throttle signal equivalent to that one generated at the gas pedal. This is easily achieved by actuating on the propulsion DC-motor controller via an analog signal from the NI-DAQ board connected to PC2. As a way to watch and characterize the power supplied by this motor, we monitor the current and voltages at the rotor, the stator and the main battery terminals. The motor controller allowed manual selecting between forward and reverse drive and also between two working regimes (dubbed "sport" and "economy") by means of switches in the front panel of the vehicle. Now those connections have been replaced by electronics which is controlled via digital outputs from the DAQ system.

Once the problem of automating the vehicle controls is overcome, the rest of sensors aim to different control strategies, state observers, fault-tolerance systems or supervisory tasks. Among the kind of the control tasks which can be performed on the prototype, one can distinguish between low and high level controllers. The mission oriented philosophy of the implemented architecture makes possible to reuse some low-level controllers for higher supervisory tasks

as path planning or obstacle avoidance. Some examples of these low-level controllers comprise the cruise control system or the low-level steering controller running on the custom microcontroller-based platform.

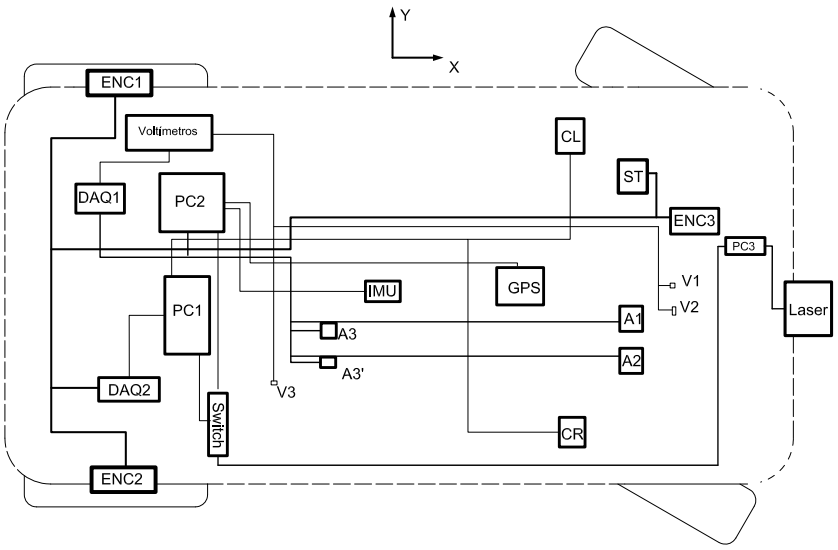


Fig. 2. Hardware equipment lay-out.

A description of sensors layout can be found in Figure 2 while their properties are summarized in Table 2. Next, we give further details on the hardware components installed in our prototype:

Computers: *PC1* is the host computer, it features a quad core 2 GHz processor running a 64bit Ubuntu operating system. Its tasks comprise the execution of MOOS core and the acquisition of wheel encoders, the communication with the low-level steering controller and cameras images processing. *PC2* runs Windows 7 operating system and acquires current/voltage signals as well as actuates over the throttle setpoint via a NI-DAQ board as well as the throttling signals. Moreover, this computer is responsible for the management of the GPS device and the IMU.

The single board computer (SBC) *PC3* is a Raspberry Pi connected to the rest of computers in the architecture via Ethernet protocol. It has been prepared to execute several algorithms of control and state observers. Moreover, due to its emplacement in the front part, this is the computer which is connected to the laser scanner. The choice of a SBC for these task is based on the principle of delegating each process to the cheapest (in terms of both, economy and power consumption) hardware platform capable of executing it in a reliable way.

Table 2. List of sensors, I/O devices and computers

Label	Type	Model
IMU	Inertial sensor	Xsens MTI 300
GPS	GPS	Hemisphere R100
ENC1,2	Wheel encoders	2 × Phidgets Optical Rotary Encoder ISC3004
ENC3	Steering-wheel encoder	SICK Optical Rotary Encoder DFS61
CR,L	stereo camera	2 × Flea3 FL3-U3-13E4C-C
Laser	Laser Range Finder	SICK LMS 200
A1,2,3, 3'	Current transducer	4 × Hall-effect LEM sensors
V1,2,3	Voltage transducer	3 × Phoenix Contact
DAQ1	I/O device	NI-USB-6211
DAQ2	I/O device	Phidgets Encoder High-Speed
ST	Controller	Prototype
PC1	Ubuntu O.S.	Adapted PC
PC2	Windows 7 O.S.	Industrial PC
PC3	Raspbian	Raspberry Pi

Since we mostly use open source software for the cameras and vision systems, whereas the pre-existing sensors were acquired on the NI-DAQ board, we have chosen to preserve past architecture and, at the same time, add new features on the car as an independent architecture. This allows also to change every module independently at every advance of our research. The usage of the Windows OS in one of the computers comes after carefully evaluating the support of NI-DAQ C++ libraries under different driver versions and GNU/Linux distributions without finding any with an acceptable level of functionality and reliability.

Sensors: The Xsens MTI-300 *IMU* is connected to PC2 via USB, and collects the vehicle angular position and velocity along with its acceleration on x,y,z axes at a frequency up to 2 kHz. The differential *GPS* R100 by Hemisphere ensures the localization of the vehicle within a tolerance of 2 cm thanks to RTK correction techniques. It is connected to PC2 through two RS-232 ports, for both data communication and correction via 3G Internet access. The SICK LMS *Laser* attached to the front part of the vehicle scans up to 180 degrees at a frequency of 18 Hz and a range of 81 m. A specifically developed controller converts the RS-432 signal to USB aimed to the communications with the PC3. The *ENC1* and *ENC2* Phidgets Optical Rotary Encoder ISC3004 mounted at the rear wheels offer a resolution of 360 pules per revolution (ppr) which allows to implement a low-cost efficient odometry system. However, the *ENC3* encoder attached to the steering column consists of the SICK DFS61 model with a resolution of 10000 ppr since the requirements of precision of the steering angle determination. The *CL* and *CR* PGR Flea3 USB3 cameras attached to the upper front left and right part of the vehicle respectively comprises the computer vision system controlled by PC1 at a frequency up to 80 kHz. Regarding the power consumption of the vehicle, three voltmeters (*V1*, *V2* and *V3*) and ampere-meters (*A1*, *A2* and *A3*) measure the voltage and current in the rotor, the field and the batteries.

Additionally, the ampere-meter A3' detects the sense of the current from the batteries in order to identify when the regenerative braking system is actuating.

I/O Devices: *DAQ1* is a National Instrument USB-6211 acquisition board with 16 analog inputs (16-bit, 250 kS/s), 2 analog outputs (16-bit, 250 kS/s), 4 digital inputs and 4 digital outputs and 2 32-bit counters. It is connected to PC1. On the other hand, *DAQ2* is a Phidgets Encoder HighSpeed 4-Input board which acquires the signals from the encoder of the rear wheels and communicates with the PC2 via USB. Finally, the *ST* device refers to the steering controller box which is connected to the PC1 via USB.

3 The MOOS-based software architecture

This section introduces the MOOS-based [10] distributed software architecture that has been developed for interfacing all vehicle's sensors and actuators. The fundamental processing element in MOOS is a *module*, an independent process that publishes and subscribes to *variables*. Several modules typically subscribe to the same variable whereas one normally finds only one publisher for each variable (although the latter is not a strict rule and the opposite makes sense in some situations), forming processing *pipelines*. A central hub called *MOOSDB* is in charge of assuring that all publishers reach all subscribers, which may be running on different computers across the local network. At present our distributed system comprises two embedded computers and one single-board-computer (SBC) (Raspberry PI-B), all of them interconnected through 1000Mbs Ethernet. Submillisecond communication delays and the usage of a local Network Time Protocol (NTP) server assures the accurate synchronization between timestamps of data gathered in different computers, a crucial issue for either grabbing datasets for offline processing or for closing control loops online.

Upon this MOOS middleware, we have designed a set of C++ modules which communicate to each other uniquely by means of message-passing under a pub/sub pattern. As customary in software engineering, our approach comprises a *layered* structure, beginning at the lowest level where modules directly interface hardware, and up to the higher levels where modules become more platform-independent. In fact, our guiding idea is allowing the transparent replacement of all the modules that interface the real vehicle with a physics (multibody dynamics) simulator, thus easing and boosting development of high-level algorithms and controller prototypes. At present such simulator is under development, thus in the following we focus on the specific modules designed for the vehicle prototype.

A sketch of the lower layers of our architecture is provided in Figure 3. Starting our description from the bottom up, we find a first layer containing all physical devices (labeled "hardware" in the figure). Next, we have the two lower layers of the software architecture itself, namely: (i) Drivers layer and (ii) the vehicle-abstraction layer (VAL). In the former layer we find software modules, each one in charge of interfacing a specific device. Modules in this

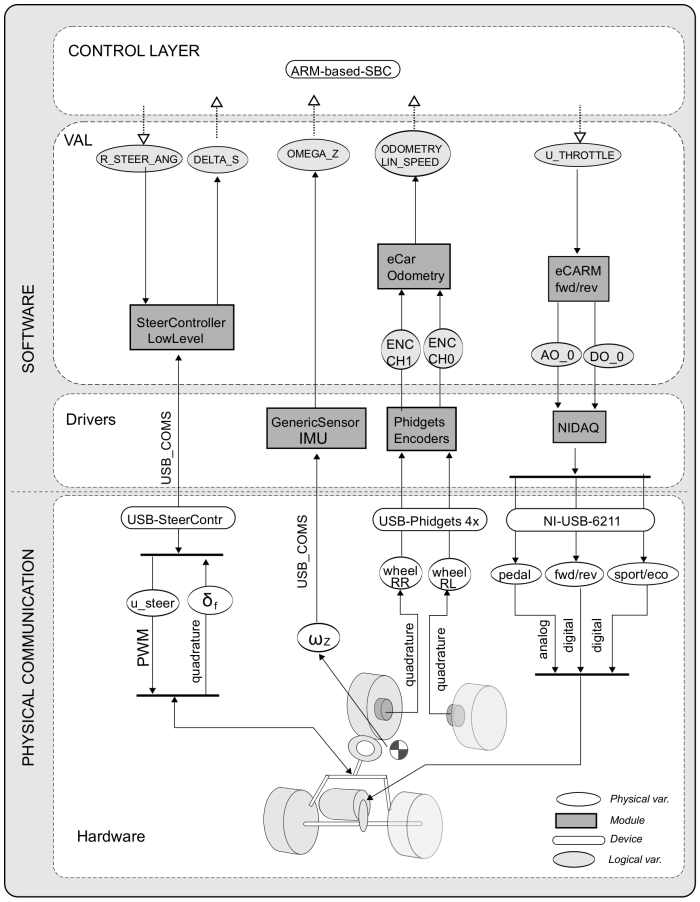


Fig. 3. Layered structure of the proposed software architecture. Refer to section 3 for details.

layer are generic and reusable, in the sense of being agnostic about the semantic or relevance of each sensed or output signal. In our present implementation this layer contains: (i) a module to read from the XSens MT4 Inertial Measuring Unit (IMU) placed at the vehicle center of mass, whose more relevant output for this work is the instantaneous rate of change in yaw (ω_z); (ii) an interface to the two rear wheels quadrature encoders, which ultimately provide wheel odometry (dead reckoning) and the linear and angular velocities of the vehicle (disregarding slippage); and (iii) the interface to a National Instruments DAQ providing several analog and digital inputs and output, from which only the outputs directly related to throttle control are displayed in Figure 3 for conciseness.

Upwards in the architecture we reach the VAL. The goal of this layer is providing a uniform interface (i.e. set of MOOS variables) that isolate the specific vehicle being used from high-level controllers. Therefore, we find two kinds of

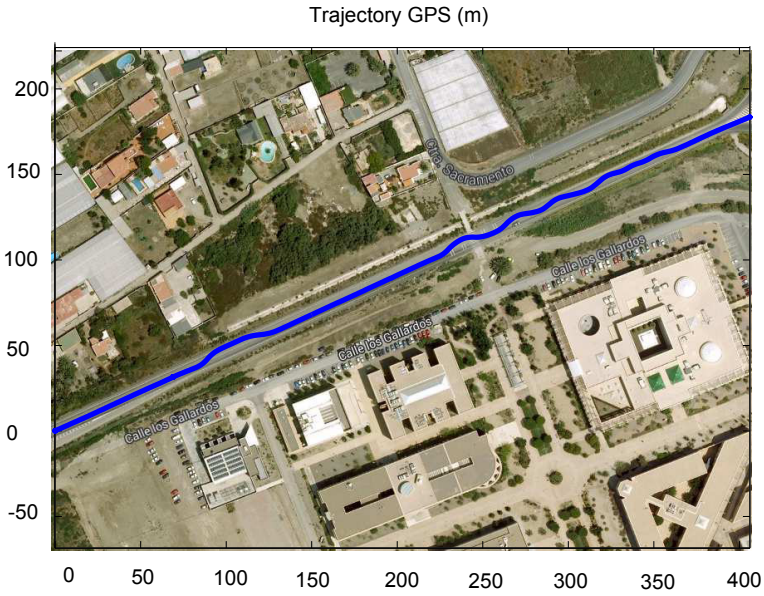


Fig. 4. GPS registered signal

modules in this layer: (i) non-generic hardware drivers (e.g. the "SteerController" module which interfaces our custom microcontroller-based steering wheel controller), whose output have a semantic closely related to the robotic platform, and (ii) converters between generic sensor signals and meaningful variables (e.g. from encoder velocity in ticks per second to vehicle linear velocity).

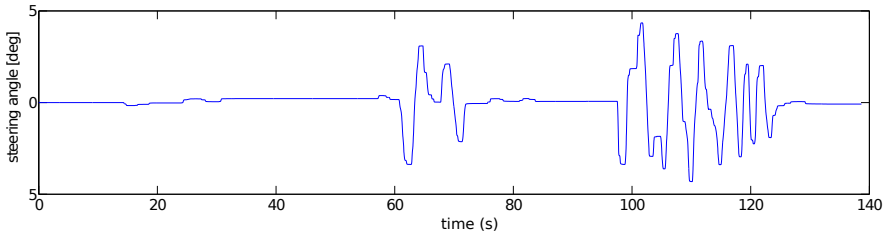
We decided that the following variables are sufficient for modeling and controlling an Ackermann-like vehicle:

- `R_STEER_ANG` (Input): Desired reference or setpoint for the steering angle.
- `U_THROTTLE` (Input): Desired normalized throttle in the range $[-1, 1]$, with negative values implying reverse gear.
- `DELTA_S` (Output): The actual instantaneous steering angle.
- `OMEGA_Z` (Output): Instantaneous yaw rate of the vehicle, as measured by the IMU.
- `ODOMETRY_LIN_SPEED` (Output): Current linear speed of the vehicle, according to rear wheels encoders.

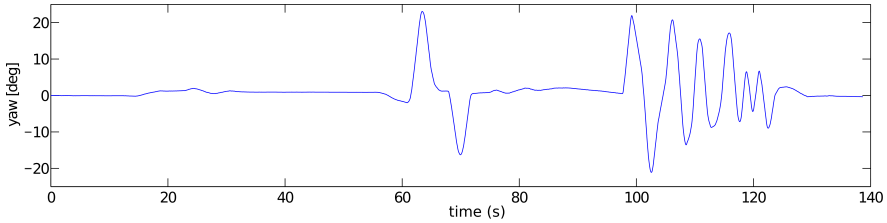
Upon these layers there also exist other high-level modules in charge of, for example, robust vehicle localization and reactive navigation, which lie outside of the scope of the present paper.

4 Preliminary experimental results and discussion

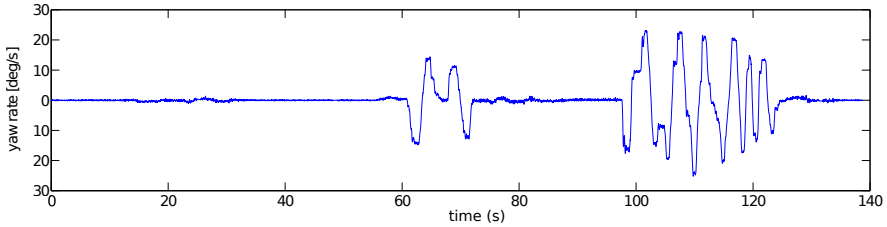
So far, experiments carried out with the proposed prototype consist of grabbing datasets as the car is driven along fixed paths. As an example of such datasets,



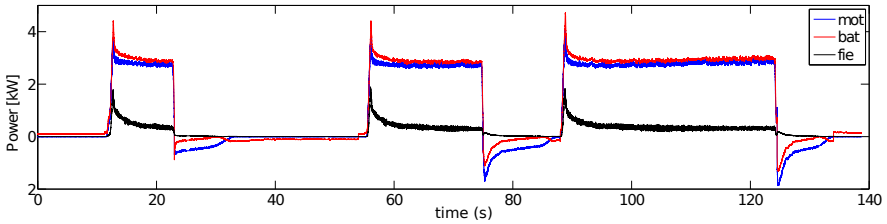
(a) Steering angle



(b) Vehicle yaw position



(c) Vehicle yaw rate



(d) Power consumption/regeneration

Fig. 5. Experimental results

Figure 4 shows the GPS signal regarding one of the paths followed in a particular experiment. This experiment consisted in the drive of the vehicle through a straight road of about 500 m, car speed ranges between 0 and of 10 m/s. The first part of the experiment consists of a straight line maneuver. The vehicle accelerated from a state of repose, and then was stopped again. The second part belongs to a double lane change, which reproduces typical real driving conditions. Finally, a slalom maneuver is performed in the last part of the experiment. Figure 5(a) represents the steering angle as the manual input signal as long as Figures 5(b)-(c) show the vehicle response in terms of its yaw angle and rate respectively. As can be seen, even when the steering angle attends to severe driving conditions as those of the third part of the experiment, the signals from the IMU fit perfectly to such a fast transient response. Regarding the power consumptions Figure 5(d) represents the energy consumption of rotor and field of the motor as well as the power supplied by the batteries. It can be seen the high requirements when the vehicle accelerates from an initial state of repose, whereas some part of the kinetic energy is returned to the batteries when the vehicle decelerates.

To conclude, the experience gained with this work demonstrates that the middleware MOOS, together with the proposed architecture built upon it, perfectly fits the demanding requirements of a distributed control architecture for an electric vehicle. To end this section, a word is in order regarding the several lines of work that remain open at present. One of them consists of contrasting theoretical models for slippage with experimental results, given different kinds of terrain and driving conditions. Another open challenge is exploring robust data fusion techniques to achieve a ground-truth 6D vehicle localization (position and attitude) as it moves at high speed. We are experimenting with fusion of stereo visual odometry, inertial data from the IMU and the wheels encoders. Finally, estimating the state of charge of the vehicle batteries is also a goal related to energy-efficient driving, a process that will require grabbing datasets that include currents, voltages and temperature measurements for the gel batteries. It is clear that all these open fronts deserve much future work after the design and construction of the presented vehicle prototype.

ACKNOWLEDGMENT

This work has been partially funded by the Spanish "Ministerio de Ciencia e Innovación" under the contract "DAVARBOT" (DPI 2011-22513) and the grant program JDC-MICINN 2011, as well as by the Andalusian Regional Government grant programs FPDU 2008 and FPDU 2009, co-funded by the European Union through the European Regional Development Fund (ERDF).

References

1. Buehler, M., Iagnemma, K., Singh, S.: The 2005 darpa grand challenge. Springer Tracts in Advanced Robotics **36**(5) (2007) 1–43

2. Leonard, J., How, J., Teller, S., Berger, M., Campbell, S., Fiore, G., Fletcher, L., Frazzoli, E., Huang, A., Karaman, S., et al.: The darpa urban challenge (2009)
3. Urmson, C., Bagnell, J.A., Baker, C.R., Hebert, M., Kelly, A., Rajkumar, R., Rybski, P.E., Scherer, S., Simmons, R., Singh, S., et al.: Tartan racing: A multi-modal approach to the darpa urban challenge. Technical Report 967, Robotics Institute (2007)
4. Bertozzi, M., Broggi, A., Cardarelli, E., Fedriga, R.I., Mazzei, L., Porta, P.P.: VIAC Expedition Toward Autonomous Mobility. Robotics and Automation Magazine **18**(3) (September 2011) 120–124 ISSN: 1070-9932.
5. Broggi, A., Medici, P., Zani, P., Coati, A., Panciroli, M.: Autonomous vehicles control in the VisLab Intercontinental Autonomous Challenge. Annual Reviews in Control **36**(1) (2012) 161–171 ISSN: 1367-5788.
6. Broggi, A., Buzzoni, M., Felisa, M., Zani, P.: Stereo obstacle detection in challenging environments: the VIAC experience. In: Procs. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, San Francisco, California, USA (September 2011) 1599–1604
7. Huang, A., Olson, E., Moore, D.: LCM: Lightweight communications and marshalling. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). (October 2010)
8. Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software. (2009)
9. MAPIR lab (University of Málaga), ARM group (University of Almería): Open Mobile Robot Arquitecture (OpenMORA). http://sourceforge.net/projects/openmora (June 2014)
10. Newman, P.M.: MOOS-mission orientated operating suite. Technical Report 2299, Massachusetts Institute of Technology (2008)
11. Blanco, J.L., et al.: Mobile Robot Programming Toolkit (MRPT). http://www.mrpt.org/
12. Blanco, J.L., González, J., Fernández-Madrigal, J.A.: Optimal filtering for non-parametric observation models: applications to localization and slam. The International Journal of Robotics Research **29**(14) (2010) 1726–1742
13. Blanco, J.L., González-Jiménez, J., Fernández-Madrigal, J.A.: Extending obstacle avoidance methods through multiple parameter-space transformations. Autonomous Robots **24**(1) (2008)