Development of a Cyber-Physical System for Mobile Robot Control using Erlang

Szymon Szomiński, Konrad Gądek, Michał Konarski, Bogna Błaszczyk, Piotr Anielski, Wojciech Turek
AGH University of Science and Technology
Krakow, Poland
Email: szsz@agh.edu.pl

Abstract—**Design of mobile robot control systems is a huge challenge, which require solving issues related to concurrent hardware access and providing high availability. Existing solutions in the domain are based on technologies using low level languages and shared memory concurrency model, which seems unsuitable for the task. In this paper a different approach to the problem of building a cyber-physical system for mobile robots control is presented. It is based on Erlang language and technology, which support lightweight processes, fault tolerance mechanisms and uses message passing concurrency model with built-in inter-process communication. Created system used a new, open-source robotic platform, which had been designed for scientific and educational purposes. Integrated system has been tested in several scenarios, proving flexibility, durability and high performance.**

I. Introduction

RECENT decades brought impressive growth in capabilities of autonomous mobile robots. Each year new wheeled, walking, swimming and flying devices are being created overcoming another limitations of mechanical devices. However, despite theoretical opportunities, mobile robots are still not widespread in industry or other commercial applications.

Each robotics system consists of two main layers: the hardware platform, which provides certain capabilities and has certain limitations, and the software controlling the hardware in order to fulfill particular tasks. It has been shown many times that even relatively low quality hardware can be used for solving sophisticated tasks if the software controls it correctly. This fact encourages further research on methods for building software systems for managing mobile robots.

Building a cyber-physical system responsible for controlling a robot is a huge challenge. Advanced robotic hardware platform is typically equipped with various sensors and effectors for determining the state of the environment and being able to modify it. The system has to interconnect all hardware devices and manage its functioning concurrently. The requirement of reliable, concurrent hardware access with real-time constraints makes the design and implementation of such systems an extremely hard task. It seems that further popularization of mobile robots in real-life applications depends on finding proper technologies and defining methods for rapid development of high quality cyber-physical systems for robotics applications.

Research in the domain of mobile robotics, which have been evaluated using real robots, has always been considered more valuable than the results obtained in simulation. Even solutions to relatively simple problems, like cooperative box pushing [1] or formation control [2], required a huge amount of work to verify using hardware. It seems that development of a control system dedicated for a particular robot and a particular task is definitely an inefficient approach.

Reusability of high level control software components can be achieved by using the software agents paradigm. Defining several layers of abstraction in a solution of a complex problem makes it possible to implement high level algorithms without depending on the specific hardware [3], [4].

Recent years brought some attempts to build an abstraction layer over hardware components, which should accelerate robot control software development. The Player/Stage [5] platform succeeded by the Robot Operating System [6] share the same idea: to provide a set of drivers for particular devices and a uniform method for accessing the hardware. Using a typical hardware with the ROS is definitely far simpler than creating hardware drivers from scratch.

There is no doubt, that the C++ language used by the ROS for implementing hardware drivers is the appropriate choice. However using the same language and technology for providing reliable and concurrent access to the hardware layer and for writing control applications may raise doubts. Shared memory concurrency model, used in C++, is based on pthreads library [7], which does not provide any high level constructs or high availability mechanisms. Error in any fragment of such application causes whole system failure which is an extremely undesirable feature in real-time mobile robot control programs.

In this paper a different approach to the problem of building cyber-physical system for mobile robots control is proposed. It is based on a message passing concurrency model adopted from the software agent paradigm. The model is applied in the hardware management layer in order to make the system more resistant to hardware failures. The implementation of the presented system has been created using Erlang language and technology [8], which provides built-in inter-process communication and failure recovery mechanisms.

The system has been tested on a new wheeled robotic platform developed in the Department of Computer Science, AGH University of Science and Technology. The platform will hopefully become very popular in educational and scientific applications. It is fully open-source, built of relatively cheap and common components, powerful and extendible. Documentation is available at address http://capo.iisg.agh.edu.pl/.

In the following section the details on the hardware platform are presented. In the following section the complete Erlang-based cyber-physical system for controlling the robot is described. Finally results of preliminary experiments are provided.

II. Mobile Platform Hardware Design

The wheeled robotic platform required for testing the Erlang-based cyber-physical system for mobile robot control has to meet several requirements. It has to be equipped with relatively powerful on-board computer, capable of running Erlang virtual machine and Linux operating system. It has to provide precise velocity control and long lasting power source. One of the key features was extensibility – hopefully the system can be used in many different scientific applications, which may require different sensors and effectors. More over it should be relatively inexpensive.

Significant development and miniaturization of electronic components over last decades resulted in creation of several advanced commercial and open-source mobile robotic platforms. The platforms are designed with different applications in mind, like transportation, exploration of unknown or unsafe area, inspection of inaccessible places such as water pipes or in the buildings security.

Most of these solutions are designed to solve a particular problem. The most widespread group, which can be found in our homes, are cleaning robots, like Roomba, Scooba and Myrrh [11]. Although these robots are quite advanced, they are hardly extendible and it is hardly possible to use the platforms to other purposes.

Avatar III is an advanced platform, which belongs to a group of robots whose main task is to detect potential intruder[12]. This type of robots is characterized by very good parameters in comparison to experimental projects, but it still does not support flexibility in adding extensions.

There are several more flexible mobile platforms available, like Komodo [13] or much bigger Husky [14], which provide large spectrum of available extensions. However, it is hard to determine what kind of on-board computer do they use. Moreover, these solutions definitely do not meet the inexpensiveness requirement.

Designing a robotic hardware platform is a very complex task. Many unexpected problems have to be solved, making the process surprisingly slow. Final solution has to include designing a robot body, its components, power management, physical and electrical interfaces, integration with remote devices etc. Building individual components, like motor drivers or power distribution systems, requires a lot of time and expenses along with further design problems and delays.

In the presented platform simplicity was the key factor. For this reason the robot was built from off-the-shelf components integrated within suitable chassis. Beside of time saving, the most important advantage of this approach is the simplicity of building new units – off-the-shelf components are relatively easy to assemble. Another advantage of this solution is the possibility of independent testing of individual parts of the system which greatly simplifies failure diagnosis. On the other hand, ready-made parts often cause problems during integration because they were designed for other purposes. Selecting, testing and integrating proper components is probably the most important outcome of the presented work.

Designed mobile platform is composed of the following components:

- chassis,
- power supply
- control unit,
- motor drivers,
- sensors and other peripherals.

Selected chassis is a Lynxmotion A4WD1 four-wheel body, 30 cm long and width. The platform is shown in figure 1.

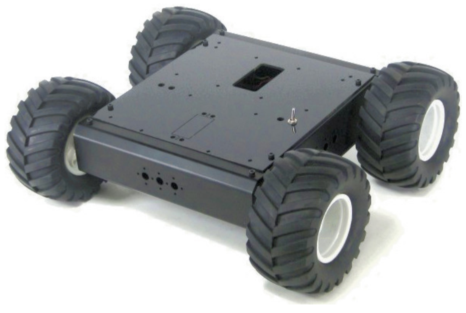

Fig. 1. Designed robotic platform based on A4WD1 chassis.

Power is supplied by two LiPo batteries connected in parallel with the nominal voltage level 14.8 V and single battery capacity 5000mAh, which is sufficient for several hours of continuous operation. Once the robot runs out of energy, batteries can be replaced without restarting the control unit.

The main robot control unit is Pandaboard [9]. Pandboard is a low-power low-cost single board computer based on the OMAP4430 dual core processor. Platform gives access to many of the powerful features of the multimedia processor while maintaining low cost. This will allow the user to develop software and use available peripherals in many configurations. The major components available on the PandaBoard, which can be used in the robot, are as follows:

- Power Management Companion Device,
- Audio Companion Device,
- Mobile LPDDR2 SDRAM Memory,
- HDMI Connector,
- SD/SDIO/MMC Media Card Cage,
- UART via RS-232 interface via 9-pin D-Sub Connector,
- LS Research Module 802.11b/g/n, Bluetooth, FM,
- Camera Connector,
- LCD Expansion Connectors,
- Generic Expansion Connectors,
- Composite Video Header.

The device runs Linux kernel with either popular distribution. The most basic task of the Pandaboard is to control the motor drivers – the RoboClaws [10].

The RoboClaw 2X15 Amp is an extremely efficient, versatile, dual channel synchronous regenerative motor controller. It supports dual quadrature encoders and can supply two brushed DC motors with 15 amps per channel continuous and 30 amp peak. With support for dual quadrature decoding it get greater control over speed and velocity is automatically maintains speed even if load increases. RoboClaw uses PID calculations with feed forward in combination with external quadrature encoders to make an accurate control solution. RoboClaw is easy to control with several built in modes. It can be controlled from a standard RC receiver/transmitter, serial device, microcontroller or an analog source, such as a potentiometer based joystick.

To control the speed of motor RoboClaw uses pulse width modulation (PWM). Pulse width modulation is a method of adjusting the current or voltage signals, which consists of changing the pulse width of constant amplitude, used in amplifiers, switching power supplies and systems control the operation of electric motors. PWM powers the system directly or through a low pass filter which smoothes the voltage waveform or current.

Because the Pandaboard and the RoboClaw works with different logic levels, a converter is required. For this purpose KAmodLVC [15] logic level converter has been used. KAmodLVC module is an 8-bit bi-directional converted voltage levels. The converter can be used to connect two digital systems operating with different voltages (like 1.8V and 5.0V in this case).

The basic orientation sensors embedded in the robot includes a gyroscope, accelerometer and magnetometer. The sensor can be used to determine the position of the robot in two planes. The diagram of components connections and relations is presented in Fig 2. The alignment of the components in the chassis is shown in Fig 3.

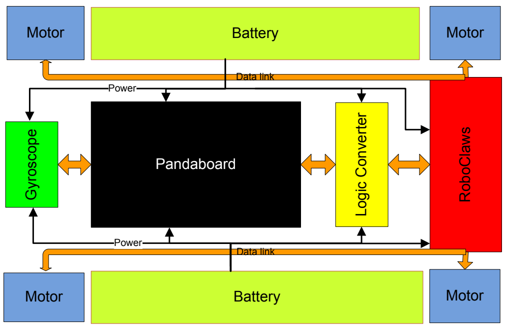

Fig. 2. The block diagram of the robot components.

The central point of control and communication is the Pandaboard. This board has several communication interfaces which are to control the robot effectors and to collect information from the sensors. Communication bus between Pandaboard and motor controller was realized using RS232 interface. For the purpose of control only lines RxD and TxD are used. There is no hardware flow control, because communication with the Pandaboard and RoboClaw is realized in inquiry respond method and it is always initiated by the Pandaboard. Therefore, if the control program waits for data from the controller it is not necessary to control rate. The data rate of this link is set to 38400bps.

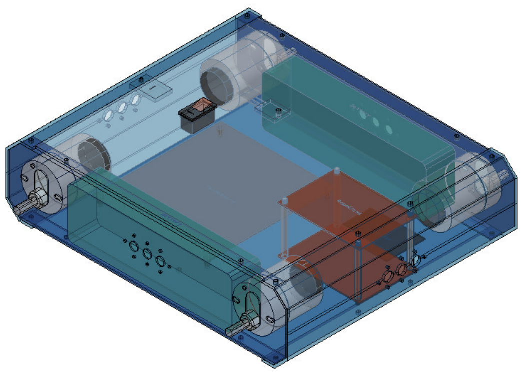

Fig. 3. Internal design of the robot components.

The orientation sensor uses serial I2C bus. To communicate with this bus the system uses duplex line Serial Data Line (SDA) and one-way line Serial Clock Line (SCL). Both lines are pull-up to power line so it is easy to detect transmissions collision using hardware. In robotic system this bus combines simplicity and functionality in one at a low investment of hardware and software to give the desired effect.

Robot communication with the surrounding environment is based on the built-in wireless card: Pandaboard WiFi. Each robot has its own unique MAC address so it is possible to communicate with the selected robot even if a group of robots is working in the same network.

Robot design provides an easy way for extending the range of sensors or effectors. It has been tested with ultrasonic sensors, laser rangefinders, cameras and Microsoft Kinect sensor. Further extensions are possible using various interfaces: USB, COM, I2C or SPI.

To determine the exact position of the robot can use the Global Positioning System (GPS) receiver or the more accurate indoor marker-based Hagisonic Stargazer [16] system. Stargazer uses markers placed on the ceiling and on the basis of their positions it can determine the location of the robot with high accuracy.

Ten units have been built so far for testing and further development purposes. The cost of all parts for a single unit does not exceed 900 USD, which is a very low price for the capabilities. The robot can develop speed of 3 m/s, it can put itself into vertical position by climbing a wall. It includes an on-board computer with 2-core CPU, running ordinary Linux OS and providing large variety of extension ports. It meets all defined requirements for testing the Erlang-based cyber-physical system for mobile robot control.

III. Control System Architecture

On the top of robot's hardware there is a need for a control software layer that allows users to interfere with it. Due to the fact that the robot was designed from scratch, control system was also chosen to be created from the ground up instead of using existing solution in order to fit the needs perfectly. Main aims of the software layer were to:

- provide high level, easy to use and consistent programming interface to low-level robot's peripherals,
- allow multiple client applications to run simultaneously on one robot, taking into account concurrency, timing, performance and other possible issues,
- allow users to write their client application in different programming languages,
- give an ability to put client application either on robot's on-board computer or on a separate network-reachable machine,
- ensure flexibility by allowing to add other external devices in the future.

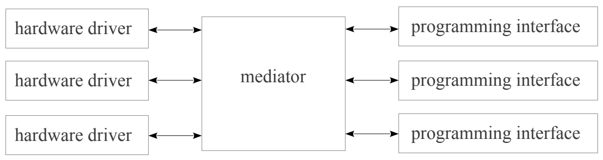

Fig. 4. Control system architecture schema

Control system has been divided into three parts (as shown on Fig 4):

- hardware drivers,
- mediator,
- programming interfaces.

Hardware drivers are standalone programs that interfere directly with robot's hardware. Being written in C++, they provide full compatibility with low-level Linux communication mechanisms.

Mediator connects hardware drivers and programming interfaces, handling communication between those two parts and controlling the whole system. It was developed in Erlang/OTP [8] due to the fact that Erlang was designed to be a solution for message passing and orchestrator applications.

Programming interfaces are libraries that end users include in their programs. They provide a consistent API to robot's hardware and can be implemented in virtually any language. There have been developed exemplary interfaces in Erlang and Java. This part of the system with be described in the next section.

System's internal communication has been based on Protocol Buffer [21] library, because it offers easy and reliable way of specifying and using custom binary protocols and has support for many popular programming languages.

A. Erlang in Embedded Systems

Erlang is a programming language created in 1986 at Ericsson Telecom AB to 'provide a better way of programming telephony applications' and "was designed for writing concurrent programs that 'run forever' " [17]. At that time telephony applications tackled atypical problems and so had unusual requirements. That applications were highly concurrent, had "soft real-time" constrains, had to be changed "on the fly" and—most importantly—had to be highly fault–tolerant, because "when the software that controls telephones fails, newspapers write about it".

Modern web servers have very similar requirements: high availability, ability to serve multiple concurrent clients, low latency and low downtime. As recent study shows [18], Erlang is well suited for such servers. It allowed writing Data Mobility server in $\frac{1}{3}$ of code and to obtain twice the throughput of C++ implementation. It's worth to note that the C++ server crashed when overloaded while Erlang just slowed down.

Since its birth, Erlang was designed as a practical tool. It is a dynamically typed, functional language with garbage collector to facilitate prototyping and ease programming. To greatly improve robustness, it implements language–level lightweight processes in shared–nothing architecture[23]. Communication is done exclusively with messages. Moreover, Erlang easily integrates with programs and libraries written in other languages. Finally it has a low memory footprint and people "successfully run the Ericsson implementation of Erlang on systems with as little as 16MByte of RAM. It is reasonably straightforward to fit Erlang itself into 2MByte of persistent storage"[24]. With all that in mind and with soft real-time characteristics of its scheduler, Erlang appears to be a perfect fit for modern embedded systems.

Embedded systems have to deal with hardware, but currently more and more sophisticated logic has to be implemented as well. Functional aspect of the language allows it to create great abstractions over hardware, algorithms and data structures. That is why it is considered that "Erlang programmers are not happy with design patterns as a convention, they want a solid abstraction"[20]. One positive effect of that is code reuse increases and the programmer can concentrate on the problem itself.

Interoperability is also very important in embedded world. Erlang has few methods for that. One of them is to write so–called "NIF"s – Native Implemented Functions. Another method is to use port drivers – communication method based on stdin/stdout streams. The latter has some advantages, most important of them is the separation of processes: even if the external program crashes for whatever reason (hardware failure or system bug), Erlang run–time is not harmed and can make attempt to recover.

In 2008 Erlang gained a SMP scheduler that allows it to scale on multiple cores/CPUs. This is a great feature, as it allows to fully use modern hardware like 64–core Parallella platform. In conjunction with multiple independent processes and message passing, this is a great advantage over most programming languages. To compare briefly:

- Standard system processes are heavy – in practice it's not feasible to create more than few hundred of them. Erlang processes on the other hand are lightweight: each one occupies only 309 words of memory. Some tests showed that it's possible to run 136.000 Erlang processes on Raspberry Pi[19].
- Concurrency is *very hard* – while using low–level tools like locks, monitors and semaphores, programmer must deal with hard problems like deadlocks, process starvation, priority inversion. Using higher–level tools, exploiting scheduler that Erlang provides and using generic structures from standard libraries allows to avoid those problems most of the time and to facilitate reasoning about process' safety and liveness.

B. Mediator

Mediator is a central part of a system. It's a thin middleware that gives much flexibility:

- Abstracts messaging between components.
- Communicates with components using standard methods, so endpoints can be written independently in most modern languages.
- Supervises each component and takes actions in case of failures.
- It is a central part of a system – only one place where configuration needs to be done.

. During start, mediator reads configuration, creates supervision tree and spawns hardware drivers (Fig. 5). Next it runs a server for communication with, possibly remote, logic system.

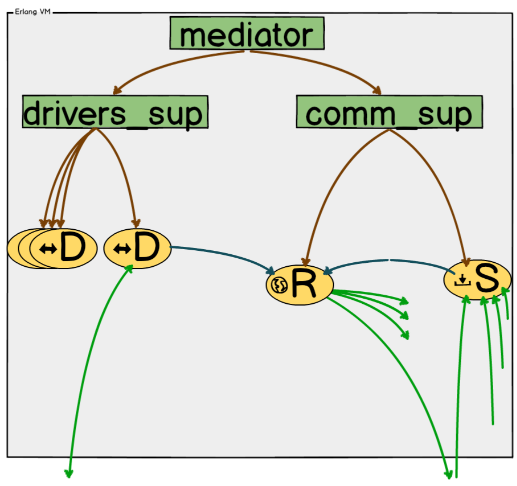

Fig. 5. Mediator is divided into: D – component mediating with software driver; R – central component, routing messages between components; S – server component, communicates with logic.

Communication with hardware drivers is done with Unix pipes. If a software or hardware has an error, mediator tries a simple yet effective tactic: restart and try again. After a number of failures in a row, it is assumed that such system is not recoverable. What is important, other subsystems are not affected and can continue to work, while defective system is turned off.

To communicate with logic, UDP protocol was chosen:

- Usually, if some part of transmission is lost, there is no need for retransmission as newer data will be available.
- UDP allows for communication locally and between computers in exactly the same way.
- When performed on localhost, packets could be lost only in case of UDP buffer overflow.
- UDP is fast and easier to use for programmers than TCP.
- Most modern programming languages have capability to communicate via UDP.

C. Hardware Drivers

As mentioned above hardware drivers are the part of the system that lays right next to robot's peripherals. There are different driver implementations for each type of supported device. They handle all low-level communication with external devices using hardware-specific protocols. This is also the only place were device logic is implemented.

Drivers are relatively simple programs spawned by the mediator and communicating with it using Unix pipes. Because software that interferes with hardware is always exposed to different kinds of failures, it is crucial to make the system as easy to recover from such issues as possible. Therefore drivers are designed as lightweight programs that can be quickly killed and restarted in case of any problems. This approach is, of course, not a perfect solutions for all types of possible exceptional situations (e.g. hardware malfunction), but makes system much more error-tolerant.

Due to the fact the drivers are separate and independent programs it is easy to add support for other devices, protocols and interfaces in the future.

All original requirements have been met in described robot's control system. The software is robust, error-tolerant, fast, flexible, perfectly suited to given hardware and ready to be used in future applications.

IV. Robot Programming Interfaces

Programming interfaces are the part of the system that end user uses directly. They communicate with the mediator using UDP sockets. Therefore client application can be run on any machine that has a network connection with robot, especially on the robot itself. UDP protocol has been chosen because it introduces small delays and low transmission overhead.

Programming interfaces can be implemented in any language that supports UDP sockets and has Protocol Buffer bindings. Thus it is possible to provide API in popular, easy to learn languages like Java or Python and allow less experienced users to work with the robot.

First implementation of programming interface was written in Java, which is a high-level, widely spread and well documented programming language that can be run on many different types of computers and other devices including mobile phones and tablets. This fact extends the number of possible applications in which robot can be used.

Second implementation was written in Erlang. It's conceptually similar to Java's implementation, but it's written idiomatically to allow programmer fully benefit features of Erlang/OTP platform. Moreover, if mediator and logic are to be both running on the same unit, they can be run on one virtual machine, thus reducing memory usage. Finally, this allows fast prototyping and experimenting using REPL (Read–Eval–Print Loop, interactive environment with command line shell).

Apart from running programs on the robot there is a possibility of testing them in a simulation. The platform was integrated with ROBOSS simulation framework [22]. A model of a physical robot is described in XML and its visual representation in ROBOSS is shown on Fig 6.

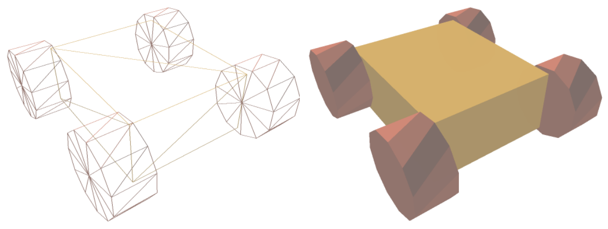
Fig. 6. Visualization of a robot model in ROBOSS simulation framework.

The use of custom Erlang module behaviour allowed to expose a simple interface which is implemented by specific Erlang modules (one for the simulation and one for the robot). As a result, the program containing logic can be run both in simulation and on the physical robot without any changes. The decision what target driver module should be used is made with regard to the configuration files.

V. Examples and Tests

In order to test the concept of building the Erlang-based cyber-physical platform and to prove that it can be used in solving real world problems, the system has been tested in a number of different applications. There were two basic groups of examples:

- on-board - when controlling program is running on robot's on-board computer,
- remote - when robot is controlled from other machine.

A. Basic Tests

In the first example robot was remotely controlled by user moving a joystick plugged into a standalone laptop computer connected to local wireless network. Moreover, real-time data read from 9DOF sensor was constantly transmitted back to the laptop and visualised on the screen as charts (see Fig 7). Therefore the user is able to see the immediate change of data charts while robot is moving.

This test showed that control software itself generates very small delays and provides enough performance to control robot manually. Actual latency is mostly dependent on WiFi

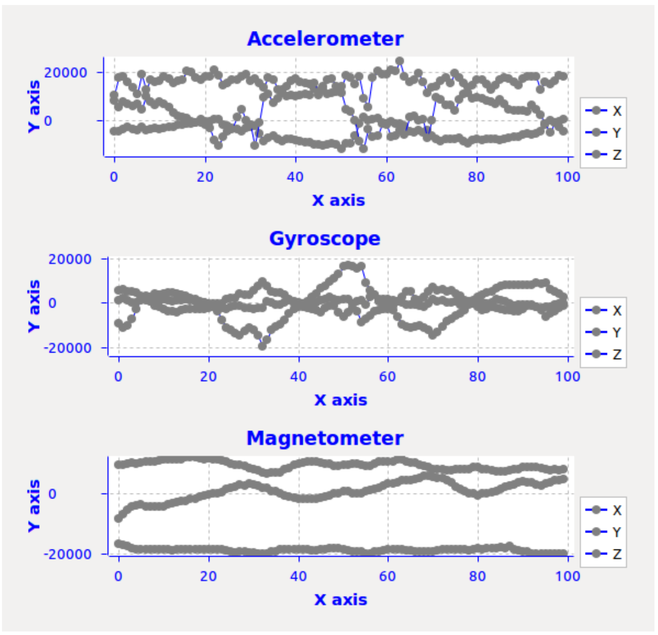

Fig. 7. Accelerometer, gyroscope and magnetometer sensors reading received from the robot during motion tests.

connection quality – some noticeable delays were observed on wireless router. This suggests that all time-critical decisions should be made on the on-board computer, while robot management or monitoring can be performed remotly.

To verify autonomous control algorithms using localization and motors controllers an advanced Trajectory Follower algorithm has been designed and implemented in Erlang. The algorithm was supposed to control robot's movements in order to reach specified locations in particular moments in time. A marker-based localization system (Hagisonic Stargazer) was used for finding current localization.

The algorithm is fully reactive. In an infinite loop it calculates most suitable control using localization and specified trajectory. The movements of the robot are smoothed according to specified algorithm parameters.

The Trajectory Follower is designed to be used both with physical robots and in a simulation. It is also desired to run on both remote and onboard nodes. Used simulation framework [22] is .NET based and this is why testing and running the trajectory follower in a simulation requires it working correctly on Windows operating system. Nonetheless, Windows OS is not required to run this component on the physical robot.

B. Trajectory Follower Algorithm

The entry point to the algorithm is a desired path to follow, expressed as a list of line segments and time constraints. On this basis, for each cycle of a control loop invoked by localisation update, desired robot speed is calculated. To preserve abstraction over the physical layer of robot, output of the algorithm is expressed as a pair of desired angular and linear velocity. Those values are later converted to velocities on respective motors by a dedicated Erlang module, called driver.

In general, it is transparent to the driver whether it communicates with physical device or simulated robot, but it is responsible for translating control and localization. It must adapt abstract values to actual robot configuration: number of independent wheels or tracks, wheels distance and radius.

Velocity calculation algorithm is based on PD controller. Considered parameters – robot's angular distance d_α from the desired robot orientation and linear distance d_{track} from the followed trajectory, with respectful weights w_α, w_{track}, are used to obtain turn radius R:

$$\frac{1}{R} = w_\alpha d_\alpha + w_{track} d_{track} \tag{1}$$

The value of d_{track} can be treated as the P term while d_α can be treated as the D term in PD controller.

The input trajectory consists of successive path segments. The final R value is a weighted average of radiuses for corresponding segments. The number of segments taken into account and weight depend on the distance to them. Maximal cut-off distance is specified by $lookahead$ parameter in the configuration file.

Behaviour of the algorithm depends on two sets of settings: description of a robot and algorithm parameters. First one defines the name of the dedicated driver, robot's physical dimensions, localization update interval (in case of polling type of driver) and path to the simulation agent, if one is used. The latter one allows to modify weights of respective factors of PD controller, $lookahead$ parameter, maximal centripetal acceleration and maximal linear velocity.

The input can be also defined as a list of control points of Bézier splines which will result in much smoother path with no rapid turns. In this case the number of segments sampled from smoothed Bézier curve has to be defined. If the $lookahead$ parameter is too small, the robot can sometimes perform tougher turns. To ensure robot stability, centripetal acceleration of the robot must stay below certain limit.

C. Results

Example run performed in a simulation is shown in Fig 8.

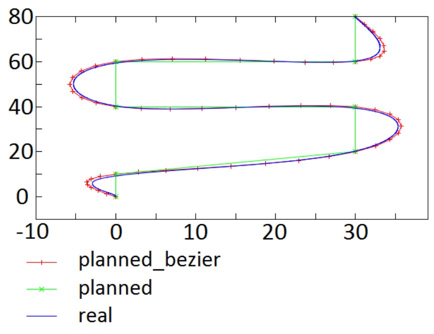

Fig. 8. Visualization of a run performed by a simulated robot. The input trajectory was smoothed with Bézier splines. Units in meters.

There was no difference or time overhead observed in communication while running the application from the remote and onboard node.

During the tests on a real robot, an issue with marker-based localization systems occurred. There were several strong light sources in the testing room. As a result, the robot tended to perform better runs with lights turned off. Exemplary run is presented in Fig 9. The robot managed to successfully read the destination within specified time, however, there is place for improvements. It is possible to reduce the noise and make measurements of localizer more precise by introducing a dead reckoning technique, i.e., applying Kalman filter.

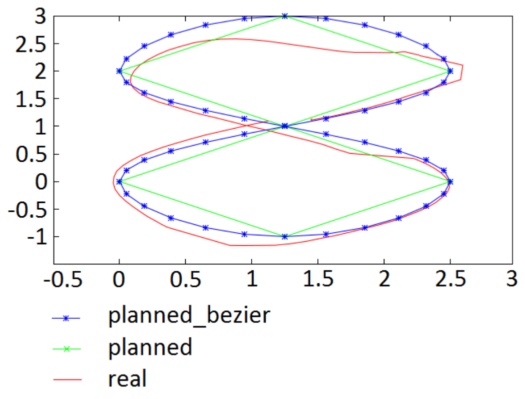

Fig. 9. Visualization of a run performed by a real robot. The input trajectory was smoothed with Bézier splines. Units in meters.

The system has demonstrated stability and performed well during the tests. It generated very small (unnoticeable) and constant delays even when several applications were using hardware components simultaneously. Erlang's functional nature seems to match high level abstraction what could be experienced during the design and implementation of the applications.

VI. Conclusions and Further Work

The implementation and performed test suggest that the Erlang language and technology is a suitable basis for building cyber-physical systems for mobile robots control. The developed system has been intensively tested in several applications. The results are promising and further work on this approach is definitely justified.

Mobile platform, which has been used for testing the approach, met all specified requirements. Designed robot is relatively inexpensive to build, offers good performance and is very easy to extend. Optional sensors and effectors will be introduced to the systems in order to increase its abilities and the range of applications. Hopefully the platform will become popular among robotics researchers.

Acknowledgements

The research leading to this results has received founding from the Polish National Science Centre under the grant no. 2011/01/D/ST6/06146.

References

[1] C. R. Kube, H. Zhang, *Collective robotic intelligence.* Proceedings of: Simulation of Adaptive Behavior, Honolulu, Hawai USA, 1992, pp. 460–468.

[2] T. Balch, R. Arkin, *Behavior-based Formation Control for Multi-robot Teams.* IEEE Transactions on Robotics and Automation, 14, 1999, pp. 926–939.

[3] W. Turek. *Extensible Multi-Robot System.* In: Computational Science - ICCS 2008, Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 574–583.

[4] W. Turek, K. Cetnarowicz, and W. Zaborowski. *Software Agent Systems for Improving Performance of Multi-Robot Groups.* Fundamenta Informaticae, 112(1), 2011, pp. 103–117.

[5] B. Gerkey, R. T. Vaughan, A. Howard, *The player/stage project: Tools for multi-robot and distributed sensor systems.* In Proceedings of the 11th International Conference on Advanced Robotics, 2003, pp. 317–323.

[6] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, A. Ng, *ROS: an open-source Robot Operating System.* ICRA Workshop on Open Source Software, vol. 3 (2), 2009.

[7] B. Nichols, D. Buttlar, J. Farrell, *Pthreads programming: A POSIX standard for better multiprocessing.* O'Reilly Media, Inc. 1996.

[8] F. Cesarini, S. Thompson. Erlang Programming. A Concurrent Approach to Software Development. O'Reilly Media, 2009.

[9] PandaBoard Documentation, http://pandaboard.org/, 05.2013.

[10] RoboClaw Documentation, http://www.basicmicro.com/, 05.2013.

[11] iRobot Products, http://store.irobot.com/, 05.2013.

[12] Avatar III Security Robot, http://robotex.com/, 05.2013.

[13] Komodo Robot Specification, http://www.robotican.net/#!komodo/c9sa, 05.2013.

[14] Husky Robot Technical Specification, http://www.clearpathrobotics.com/husky/tech-specs/, 05.2013.

[15] KAmodLVC module technical documentation, http://www.kamami.pl/dl/kamodlvc.pdf, 05.2013.

[16] J. Lopez Fernandez, C. Watkins, D. Perez Losada, M. Diaz-Cacho Medina, *Evaluating different landmark positioning systems within the RIDE architecture*, Journal of Physical Agents, 7(1), 2013, pp. 3–11.

[17] J. Armstrong. *A history of Erlang*, Proceedings of the third ACM SIGPLAN conference on History of programming languages, San Diego, California, 2007, pp. 6–26.

[18] J. H. Nyström, P. W. Trinder, D. J. King, *High-level distribution for the rapid production of robust telecoms software: comparing C++ and ERLANG*, Concurr. Comput. : Pract. Exper. 20(8), 2008, pp. 941–968.

[19] O. Kilic, *136.000 Processes on a Pi.*, http://www.erlang-embedded.com/2012/05/episode-3-\%E2\%80\%93-136-000-processes-on-a-pi/, 05.2013.

[20] F. Hébert, *Learn You Some Erlang for Great Good!: A Beginner's Guide*, No Starch Press, Incorporated, 2013.

[21] Protocol Buffer library homepage, https://code.google.com/p/protobuf/, 05.2013.

[22] W. Turek, R. Marcjan, K. Cetnarowicz. *A Universal Tool for Multirobot System Simulation*, Knowledge-Driven Computing, Springer, 2008, pp. 289-303.

[23] J. Armstrong *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, ACM, 2007,pp. 6-26.

[24] Erlang FAQ, *Implementation and ports of Erlang*, urlhttp://www.erlang.org/faq/implementations.html, 05.2013.