Service-oriented control system for a differential wheeled robot

Alexander Mangin, Lyubov Amiraslanova, Leonid Lagunov, Yuri Okulovsky

Ural Federal University
Yekaterinburg, Lenina str. 51
Email: yuri.okulovsky@gmail.com

***Abstract*—Double-wheeled robot is a classical yet populair architecture for a mobile robot, and many algorithms are created to control such robots. The main goal of the paper is to decompose some of this algorithms into services in service-oriented system with an original messaging model. We describe data types that are common for robotics control, and ways to handle them in .NET Framework. We bring a list of various services' types, and each of them can be implemented in several ways and linked with other services in order to create flexible and highly adjustable control system. Service-oriented systems are scalable, can be distributed on many computers, and provides huge debugging capacities. The service-oriented representation is also very useful when teaching robotics, because each service is relatively simple, and therefore algorithms can be presented to students gradually. In this paper, we also focus on a particular services' types, which provides the correction of the robot by the feedback, decribe the original algorithm to do so, and compare it with several others.**

***Index Terms*—robotics, service-oriented approach, double-wheeled robots**

INTRODUCTION

A differential wheeled robot is a mobile robot whose movement is based on two separately driven wheels, placed on either side of the robot body. Examples of this architecture are Roomba vacuum cleaner [2], Segway vehicle [6], various research and educational robots (e.g., [7]).

Differential wheeled robot is a very simple and effective architecture, both in mechanic and control means, and many various algorithms were developed to control it. In this paper we decompose some of these algorithms within the service-oriented approach. In SOA, the functionality of the program is decomposed into a bunch of services, which communicate by TCP/IP protocol, or by shared memory, or by other means. Each of the services performs a single and simple task, and provides some result in response to an input in a contract-defined format.

Service-oriented approach is widely used in robotics [12], [17]. Its main advantages are as follows. The system can be distributed among several computers, which is important, because the real robotics is very resource-expensive. The system is also decentralized, which allows it to operate even if some auxiliary parts stop working due to errors. The service-oriented approach also has a great value in education. The service-oriented decomposition allows making a step-by-step acquaintance with complex algorithms, by dividing them to small and well-understandable parts, therefore simplifying teaching the algorithms to students. The decomposition also facilitates research and development. While running the algorithm, all the information that passes between services can be stored in logs and then viewed, which offers a great debugging feature. Also, modern development techniques, like agile development, become more applicable, because the parts that the algorithm is divided into can be distributed between developers, can evolve gradually, and can be thoroughly tested with unit and functional tests.

Overall, service-oriented approach to robot's control is one of the most populair and promising. Many control system are founded on it, and most prominent are Microsoft Robotics Developer Studio [3] and Robotic Operating System [10], [5]. In [11] we propose RoboCoP, a Robotic Cooperation Protocol, which introduces an innovative messaging model into service-oriented robotics. In RoboCoP, services have inputs and outputs, which are interconnected in a strict topology. For example, when analyzing images, a Camera services output is plugged in to a Filter services input, and the Filter in turn is connected to a Recognizer service in the save way. So the signal propagates along the control system from service to service, and is subsequently processed by them. This messaging model is used in LabView [18], DirectShow [19] and other software, but is new for robotics. For example, in MRDS, services exchange messages via a central switch, in ROS they use broadcast messaging model, etc. [11].

In [11], we implemented this new messaging model for interconnection of independent applications with an open and simple protocol. We also built a control system for a manupulator's control, and therefore assert the effectiviness of our approach. In this paper, we bring another example of decomposition into RoboCoP services, this time for the control system for a differential wheeled robot. All the services and algorithms, described in the paper, are implemented. The system was tested on the real differential-wheeled robots during the Intertational contest on autonomous robots control "Eurobot" [1].

In section I, we describe the data types that are important for control of the differential wheeled robot. We also describe an innovative LINQ-style [15] approach to their processing. In section II, we bring the service-oriented control systems for differential wheeled robots, and in section III we explore the peculiarities of some used algorithms.

I. Presenting and processing robotics data

A. Data types

Service-oriented control system consists of services, which transform the information from one type to another. In this section we describe the important data types in our system. The most fundamental structure is a differential wheeled movement (DWM), which is a tuple $(v_{0,l}, v_{0,r}, v_{1,l}, v_{1,r}, T)$ where $v0, s$ are linear speeds of left ($s = l$) and right ($s = r$) wheels at the beginning of movement, $v1, s$ are speeds at the end of movement, and T is a time the movement lasts. One DWM describes the movement with constant acceleration of wheels, which is a reasonable physical model of real engines.

Let a_s be an acceleration of the corresponding wheel, $a_s = \frac{v_{1,s}-v_{0,s}}{T}$. Let a be the linear acceleration of the robots coordinate system, $a = \frac{a_r+a_l}{2}$. Similarly, $v_s(t)$ is a speed of the corresponding wheel at the time t, so $v_s(t) = v_{0,s} + a_s t$, and $v(t)$ is the linear speed of the robot, $v(t) = \frac{v_l(t)+v_r(t)}{2}$. We can now compute a direction $\alpha(t)$ as follows

$$\alpha(t) = \frac{B_r(t) - B_l(t)}{\Delta}$$

where Δ is the distance between wheel, and $B_s(t)$ is the total path covered by s-th wheel at the time t, $B_s(t) = v_{0,s}t + \frac{a_s t^2}{2}$. The curvature $R(t)$ of the robot trajectory can be obtained as $R(t) = \frac{\Delta}{2}\frac{v_r(t)-v_l(t)}{v_r(t)+v_l(t)}$.

Let $L(t)$ be the offset of the robot at time t along the tangent of the direction at time $t = 0$, and $h(t)$ be the offset at the normal to the initial direction. It can be shown that

$$L(t) = \int_0^t v(\tau)\cos(\alpha(\tau))d\tau$$

$$h(t) = \int_0^t v(\tau)\sin(\alpha(\tau))d\tau$$

Depending on DWM, the robot covers trajectories of different shape.

1) The straight line when $v_{0,l} = v_{0,r}$ and $v_{1,l} = v_{1,r}$, and $L(t) = v(0)t + at^2/2$.
2) The turn at the spot when $v_{0,l} = -v_{0,r}$ and $v_{1,l} = -v_{1,r}$. In this case, $L(t) = h(t) = 0$, and $\alpha(t)$ gives the direction of the robot at the time t.
3) The circle arc when $v_{0,l}/v_{0,r} = v_{1,l}/v_{1,r}$, so $R(t) = R$ is constant, and $L(t) = R\cos\alpha(t)$ and $h(t) = R\sin\alpha(t)$.
4) The spiral arc when $a_l = a_r$ and $v_l(0) \neq v_r(0)$. Let $q(t) = (v_r(t) - v_l(t))/\Delta$, and in this case

$$L(t) = \frac{1}{q}\left[v(0)\sin tq + at\sin tq + \frac{a}{q}(1-\cos tq)\right]$$

$$h(t) = \frac{1}{q}\left[v(0)(1-\cos tq) - at\cos tq\frac{a}{q}\sin tq\right]$$

5) The clothoid segment otherwise.

Clothoid is the most general case, and one need Fresnel integrals $S(t) = \int_0^t \sin\pi\tau^2/2d\tau$ and $C(t) = \int_0^t \sin\pi\tau^2/2d\tau$ to compute the robots location. Let us consider some intermediate values:

$$X = \frac{a_r + a_l}{a_r - a_l}$$

$$v_c(t) = \frac{v_r(t) + v_l(t)}{2} + X\frac{v_r(t) - v_l(t)}{2}$$

$$\Delta_c = \Delta X/2$$

$$U(t) = \frac{v_r(t) - v_l(t)}{2\Delta|a_r - a_l|}$$

$$\delta = \mathrm{sign}(a_r - a_l)$$

With this definitions, we compute $L(t)$ and $h(t)$ as follows

$$L(t) = \Delta_c\sin\alpha(t) + V_c(C(t)\cos U(t) + S(t)\sin U(t))$$

$$h(t) = \Delta_c(\cos\alpha(t) - 1) + \delta V_c(S(t)\cos U(t) + C(t)\sin U(r))$$

To specify the shape the robot should cover, we use data structures, called geometries. Currently, there are three geometries available: the straight line, which is parametrized by its length; the turn on the spot, which depends on the desired angle; and the circle arc. We can set the arc by its radius and the total rotation angle, or by the radius and the total distance that should be covered. Spirals and clothoids are not implemented, because they are hard to define, and generally, by our opinion, the benefits of their use are greatly overwhelmed by the complexity of their handling.

Another important kind of data is sensors measurements. Encoders are devices that count the rotations of wheels and store the measurements in EncodersData structure. Encoders are considered the most fundamental sensors, and many algorithms use them. Accelerometers and gyroscopes are sometimes used to collect data about acceleration and direction of the robot. This data is used in the processed format of NavigatorData, which consists of the robots basis in 2-dimensional space and of the time when measurements were taken. Aside from using in control algorithms, the series of NavigatorData is also used for drawing charts.

When measurements are considered, it is convenient to introduce spans between them. NavigatorDataSpan contains a time interval between two NavigatorData, and displacement of the last basis relatively to the first one. Encoders data span contains a time interval and differences of distances performed by the left and the right wheels.

B. Conversions

Let us examine some possible conversions between the described data types. This conversions are widely used in control algorithms, to answer the questions like “where a robot, driven by a given DWM, is situated”.

DWM can be converted to NavigatorDataSpan. Backwards conversion is impossible because some displacements, e.g. a shift to the right side, cannot be achieved by the differential wheeled robot at all. DWM and EncodersDataSpan are mutually convertible.

DWM and geometries are conventionally convertible. DWM can be converted to a geometry, but since spirals and clothoids

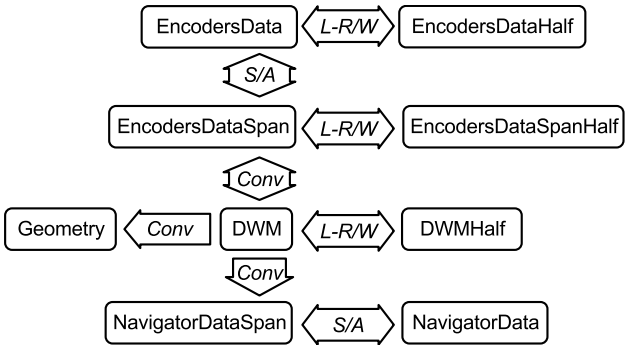

Fig. 1. A map of conversions between robotics data types

are not implemented, this conversion currently works only for a limited set of DWM. Many DWMs corresponds to one geometry, and for movement planning, we should consider velocity and acceleration limitations, the finishing speed of the robot at the previous path, and so on. Therefore, turning geometry into DWM can be done in several ways, and we should use a proper service to achieve it. However, converting geometry to some DWM is useful, and therefore we introduced a normalized DWM for lines, circles and turns. A normalized DWM is a DWM without acceleration, and has $\max(v_{0,l}, v_{0,r}, v_{1,l}, v_{1,r}) = 1$. This conversion is used only for further conversion to NavigatorDataSpan and NavigatorData, in order to draw charts for geometries.

Two measurements NavigatorData and EncodersData can be converted into a span between them of the corresponding NavigatorDataSpan and EncodersDataSpan types. A measurement and a corresponding span can be converted into the final measurement. We call this types of conversion spreading and accumulating, and they can be applied to arbitrary measurements.

Finally, we define symmetric data, i.e. data that can be naturally divided into "left" and "right" part. For example, DWM and EncodersData are symmetric. DWM can be subdivided into DWMHalf that describes the command for one wheel, and EncodersData can be subdevided into EncodersDataHalf that descbes the wheels state.

The map of these methods is shown in the Figure 1, where L-R/W arrows depicts transformations of symmetric data, S/A corresponds to spreading and accumulating, and Conv denotes conversions. In Figure 1 we may see, that any data type can be turned into NavigatorData, and therefore drawn at the chart.

C. Conversions of series

Control algorithms usually deal with the series of robotics data, and so we developed means to handle such series. .NET Framework provides an incredibly powerful and convenient tool for series processing, named language-integrated queries, LINQ. An example of LINQ is shown in the Listing 1. The code in the example processes IntArray, a collection of integers. The collection is filtered with the lambda `number=>number>=10` , which maps an integer into boolean value. With this lambda, all the integers less than 10 will be thrown off. Then the resulting collection is sorted, and converted into a collection of strings. Finally, the collection of strings is aggregated with the lambda `(stacker,str)=>stacker+","+str`, i.e. strings are subsequently accumulated in a stacker through commas. LINQ changes the very view on how the collections should be processed, and increases enormously the codes readability and reliability.

Listing 1 The LINQ code for processing collections

```
IntArray
    .Where( number => number>=10 )
    .OrderBy( number => number)
    .Select( number=>number.ToString())
    .Accumulate(
        (stacker,str)=>stacker+","+str);
```

We have developed the following LINQ extensions for handling robotics data:

1) Conversion extensions. For example, `encSpans.ToDWM().ToNavigatorDataSpans()` gets a sequence of displacements that corresponds to initial encoders data.
2) Spreading and accumulating. For example, `navData.Spread().Accumulate(newBasis)` shifts the navData from initial basis to the new one.
3) SymmetricData handling. For example, `encData.Lefts()` and `dwms.Lefts()` give the states and commands correspondingly for the left wheel.

Our extensions are compatible with the original LINQ, for example,

```
encData.Lefts()
       .Select
         (spanHalf => spanHalf.Distace)
       .Sum()
```

gives the total distance, covered by the left wheel.

The tricky moment in our LINQ implementation is to support adequate type-inference. On other words, how the extension method `Spread` knows if it should return the sequence of EncodersDataSpan or NavigatorsDataSpan, depending on its arguments, EncodersData or NavigatorData? To do that, we developed the generic interfaces, presented in the Listing 2. The type TSpan in the method `Spread` is determined from the argument, and because NavigatorData implements interface `ISpannableDeviceData<NavigatorData, NavigatorDataSpan>`, TSpan is assign to `NavigatorDataSpan`. This implementation is extendable, because the method `Spread` will appear for any type that supports ISpannableDeviceData.

Listing 2 The hierarchy of interfaces for the correct type-inference

```
public interface
    ISpannableDeviceData<TData, TSpan>
            { ... }

public interface
    IDeviceDataSpan<TData, TSpan>
            { ... }

public class NavigatorData
     : ISpannableDeviceData
       <NavigatorData, NavigatorDataSpan>
            { ... }

public class NavigatorDataSpan
     : IDeviceDataSpan
        <NavigatorData,NavigatorDataSpan>
            { ... }

public class Helpers  {
public static IEnumerable<TSpan>
    Spread<TData, TSpan>
       (this IEnumerable
             <ISpannableDeviceData
              <TData,TSpan>> data)
            { ... }
}
```

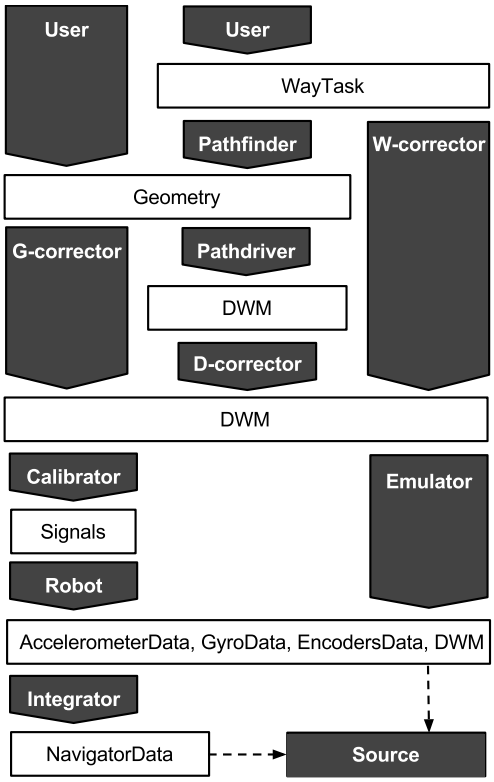

Fig. 2. Service-oriented decomposition of the control system for double wheeled robot

II. Service-oriented decomposition of control algorithms

In this section we offer the decomposition of a control system for a differential wheeled robot into services, which process data, described in the previous section. The whole control process is represented as a sequence of small and simple algorithms, each representing a single responsibility: creating a path for robot to go, processing data from sensors, etc. Such sequences can be best described in schematic way, as in the Figure 2. We should stress, that such decomposition is in fact half of the work while creating a service-oriented control system, because it is not easy to invent he design that looks naturally, is easy to expand and allows implementation of different kinds of control algorithms.

Here boxes are services that run simultaneously and processes data, which is depicted as arrows. The work of the control system starts, when it accepts a task from the user. The task can be expressed as a WayTask type, which is a collection of points that are to be visited by the robot. WayTask can be processed by the Pathfinder service to the collection of geometries that represents the desired path. The simplest strategy is to go from one point to another and then to turn the robot on the spot in the direction of the next point. More sophisticated versions were also developed [14].

In fact, more than one control system is depicted in the Figure 2, because each type of the data can be processed in many ways. Instead of going to the Pathfinder, WayTask can be used for the direct control of the robot, which is performed by the W-Corrector service. Note the important difference between W-Corrector and Pathfinder. Pathfinder completes its job at once, and we call such services “functional services”, because they are a program model of a single function. Unlike Pathfinder, when W-Corrector receives the task, it starts a continuous process of the robots control. Corrector performs an iteration for each 50-200 milliseconds, depending on its settings. At each iteration, it collects the sensors data from the Source service, which plays the role of a buffer for the measurements; then it determines the best command at the current time, and sends it further. The algorithm of W-Corrector is described in the section III.

The collection of Geometry can also be processed by its own G-Corrector. The idea of G-Corrector is that in order to follow the line, turn and circle geometries, the robot completes some given distance by one of its wheel, and travel along with the constant rate of its left and right speeds (1 for line, -1 for turn, another constant for a circle). The peculiarities of G-Correction is also described in the section III.

The collection of geometry may instead go to the Pathdriver, which converts geometries into DWM commands. Again, various strategies are possible: to stop completely after each geometry, which is very accurate but time-consuming, or to proceed to the next geometry maintaining non-zero speed. The

Pathdriver *is not* the corrector: it completes the job at once, taking a collection of geometry and producing a collection of DWM, therefore being a functional service. This collection is later processed by the D-Corrector, also described in the section III.

When the DWM is produced by some of the Correctors, it may be sent to the robot. In this case, the speeds should be converted into discrete signals, representing the duty cycles in the pulse-width modulation. This is done by the Calibrator service, which provides the correspondence between speeds and signals. The correspondence is established in the calibration process, when signals are sent to the robot, and then the rotation speed of engines is measured by encoders. Complex Calibrators may correct that correspondence while the robot is operating.

The program model of the real robot is a Robot service, which accepts discrete signals and produce the measurements of gyroscopes, accelerometers and encoders. The Robot *is not* the functional service: it does not return the measurements in response to the command. Instead, it produces measurements constantly and asynchronously. Robot service communicates with a controller board, Open Robotics [4] in our case. This board contains ATMega128 controller, slots for servos, I2C and analogous sensors, can be connected to PC via USB-UART adapter or Bluetooth adapter, and can be augmented with the amplifier board for ommutator motors. We have developed our own firmware for this board, which accepts commands in DWM format, manages servos and continuously examines sensors, collects the data and sends it to the computer in text format. It improves the built-in firmware, because sensors are monitored constantly, while built-in firmware requires sending a command to get sensors data. Other versions of Robot service can be developed for other boards and firmwares.

Instead of going to Robot, the initial DWM may be passed to Emulator, which is used for debugging. Emulator is a software that applies DWM commands to the robots current locations, computes due values of accelerometers, gyroscopes, and other sensors, adds noise into control action and feedback.

Measurements, generated by the Robot or the Emulator services, are used to get more programmer-friendly information about robots location, i.e. NavigatorData, by simple integration or Kalmans filter [8]. Alternatively, NavigatorData can be obtained directly from Emulator, but, of course, not from Robot. All measurements are stored in buffers of Source service, and when corrector starts the iteration, it reads the buffers.

III. The correction algorithms

A. D-Correction

D-correction is the most trivial correction algorithm, and is a variation of the PID-controller [16]. At each iteration i, the due state of robot is d_i vector, which is a pair of distances, covered by the left and the right wheel. Due vectors are computed by D-corrector from the input, which is a collection of DWM, and therefore can be converted into a collection of EncoderDataSpan. Real state of the robot can be obtained from encoders, and, assuming the work of the encoders is synchronized with iterations of D-Corrector, a serie r_i of real states on each iteration can be achieved. PID controller computes the next control action as a weighted sum of three terms. Let $e_i = r_i - d_i$, i.e. the error at the iteration i, and proportional term at the iteration k is $T_P = e_k$. Integration term T_I is defined as $\sum_{i=0}^{k} e_k \Delta T$, where ΔT is the time between correction iterations, and derivative term $T_D = \frac{e_k - e_{k-1}}{\Delta T}$. The resulting value $c_k = d_k + g_P T_P + g_I T_I + g_D T_D$, where g_P, g_I and g_I are the weights of corresponding terms. So $c = (c_l, c_r)$ is the state of the robot that should be achieved by the next iteration, and consists of the desired distances for both wheels. D-Corrector should now construct DWM. Let $v_{i,l}$ and $v_{i,r}$ be the end velocities of DWM, assigned at $(i-1)$-th iteration, $v_{0,l} = v_{0,r} = 0$. Therfore, at i-th iteration D-corrector should construct DWM with starting speed $v_{i-1,r}$ and $v_{i-1,l}$ such that this DWM covers distances c_l and c_r within the time ΔT, and therefore $v_{i,s} = \frac{2c_s}{\Delta T} + v_{i-1,s}$ for $s \in \{l, r\}$.

B. G-Correction

In D-correction, DWMs are used as a source of control actions. In G-correction, the geometries are. Suppose we need to travel along a circle, or line, or turn on the spot. The only thing needed to be done is to cover some distance L by one of the wheels, e.g. the left one, while keeping a constant rate k between the speeds of the left and right wheels. For the line $k = 1$, for the turn on the spot $k = -1$, for an arc of a circle k depends on circles radius. We have implemented G1-Correction, using encoders to get the current speed values, and PID-controller to maintain the proper value of k.

C. W-Correction

Way task is a set of points with coordinates (x_i, y_i) for $i = 1, \ldots, n$. For each i we construct a vector field F_i, which indicates the proper vector of robots speed in point (x, y) while it heads toward (x_i, y_i). Let $F_i(x, y) = (w_{i,x}(x, y), w_{i,y}(x, y))$, and

$$w_{i,x}(x, y) = k(x, y)(x - x_i)$$

$$w_{i,y}(x, y) = k(x, y)(y - y_i)$$

and $k(x, y) > 0$ is a normalizing coefficient such that

$$||(w_{i,x}(x, y), w_{i,y}(x, y)|| =$$

$$= \min \left\{ \begin{array}{l} v_{max}, \\ \sqrt{2||(x - x_i, y - y_i)||a_{max}}, \\ \sqrt{2||(x - x_{i-1}, y - y_{i-1})||a_{max}}, \end{array} \right\}$$

where v_{max} and a_{max} are maximum allowed speed and acceleration of the robot. This definition of F assures that if the robot is heading to the point(x_i, y_i), and is found in the point (x, y), it should direct to (x_i, y_i) with allowed speed, and also should be able to stop with the allowed acceleration.

When W-Corrector drives the robot to $i - th$ point and constructs the next DWM, it uses the sensors measurement to determine the current state: location (x, y), direction ϕ,

speeds of the left and the right wheels v_l and v_r, linear speed $v = \frac{v_l+v_r}{2}$ and torsion $q = \frac{v_l}{v_r}$. If (x, y) is close enough to (x_i, y_i), W-Corrector increments i. Then it calculates the due speed vector $(w_x, w_y) = F_i(x, y)$, its module w and direction ψ. The W-Corrector asserts new linear speed to v, which equals to vc_v if $v < w$, and to v/c_v otherwise. Similarly, new torsion q is increased by c_q if direction ψ is on the left side from ϕ, and decreased otherwise. Finally, using v and q, speeds v_l and v_r are obtained from it, and new DWM is constructed.

D. Comparison of correction algorithms

We have implemented aforementioned algorithms and tested them to choose the best one for Eurobot competitions [1]. The preliminary results of comparison are as follows.

- D-correction is a very accurate algorithm, but is hardly compatible with electronics we possess. The problem is that in the end of movement, the small but frequent oscillations occurs, driving wheels backwards and forwards to achieve the requested position. Such oscillations puts out of action the motors amplifiers.
- W-correction is a great way to understand the control of the robot, and to visualize the correction algorithms. Still, further researches are needed to ascertain its effectiveness and obtain optimal values for its coefficients.
- G-correction is currently our best solution to correction.

IV. Conclusion and future works

In this paper, we presented a decomposition of a control system for a double wheeled robot into a bunch of services. We have developed the architecture of services, as well as the services themselves, and were able to use this system for control of an autonomous double wheeled robot in Eurobot 2013 competitions.

The primary direction of our future works is to introduce more correction algorithms. For example, we are developing the G-correction algorithm, which uses gyroscope data, and the services for elimination of gyroscope noise. Also, we are developing more sophisticated services for conversion of accelerometers and gyroscope measurements into NavigatorData for using it in D-Correction algorithms. Also, we plan the further decomposition of D-correction into fields generator and fields driver, and work on different sources of vector fields, such as geometries.

Other planned works in the area of double wheeled robots control includes the following topics.

- Shifting the current SOA Framework, RoboCoP, to a better solution, based on Redis [9] common memory service.
- Integrating emulator, described in [13], into the system for better visual feedback about robots location, and for getting emulated images from camera.
- Publishing the solution and some of the developed algorithms for an open access of community.
- Thorough statistical comparison for correction algorithms.

V. Acknowledgments

We thank Pavel Egorov for valuable commentaries and suggestions, which help us design W-correction algorithm.

The work is supported by RFFI grant 12-01-31168, “Intelligent algorithms for planning and correction of robot’s movements”.

References

[1] Eurobot competitions. *http://eurobot.org*.
[2] The irobot company. irobot roomba. *http://www.irobot.ru/aboutrobots.aspx*.
[3] Microsoft robotics developer studio. *http://msdn.microsoft.com/en-us/robotics/default.aspx*.
[4] Open robotics. *http://roboforum.ru/wiki/OpenRobotics*.
[5] Robotics operating system. *http://www.ros.org*.
[6] Thesegway company. segway. *http://www.segway.com/*.
[7] The willow garage company. turtlebot. *http://turtlebot.com/*.
[8] A. V. Balakrishnan. *Kalman filtering theory*. Optimization Software, Inc., Publications Division, 1984.
[9] J. L. Carlson. *Redis in Action*. Manning, 2012.
[10] J. L. Foote, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. *http://www.robotics.stanford.edu/ ang/papers/icraoss09-ROS.pdf*, 2009.
[11] D. O. Kononchuk, V. I. Kandoba, S. A. Zhigalov, P. Y. Abduramanov, and Y. S. Okulovsky. Robocop: a protocol for service-oriented robot control system. In *Proceedings of international conference on Research and Education in Robotics - Eurobot 2011*. Springer, 2011.
[12] J. Kramer and M. Scheutz. Development environments for autonomous mobile robots: A survey. *Autonomous Robots*, 22:132, 2007.
[13] M. Kropotov, A. Ryabykh, and Y. Okulovsky. Eurosim - the robotics emulator (russian). In *Proceedings of the International (43-th Russian) Conference ”The contemporary problems of mathematics”*, 2012.
[14] A. Mangin and Y. Okulovsky. The implementation of the control system for double wheeled robot (russian). In *Proceedings of the International (44-th Russian) Conference ”The contemporary problems of mathematics”*, 2013.
[15] F. Marguerie, S. Eichert, and J. Wooley. *LINQ in Action*. Manning, 2008.
[16] R. C. Panda, editor. *Introduction to PID Controllers - Theory, Tuning and Application to Frontier Areas*. InTech, 2012.
[17] M. Somby. Software platforms for service robotics. *http://www.linuxfordevices.com/c/a/Linux-For-Devices-Articles/Updated-review-of-robotics-software-platforms/*, 2008.
[18] J. Travis. *LabVIEW for Everyone*. Prentice Hall, 2001.
[19] P. Turcan and M. Wasson. *Fundamentals of Audio and Video Programming for Games (Pro-Developer)*. Microsoft Press, 2004.