Design and Implementation Raspberry Pi-based Omni-wheel Mobile Robot

Kirill Krinkin
Saint-Petersburg State
Electrotechnical University
St. Petersburg, Russia
kirill.krinkin@fruct.org

Elena Stotskaya
Pascal Lyceum
St. Petersburg, Russia
elena.stotskaya@yutax.ru

Yury Stotskiy
EMC Corporation
St. Petersburg, Russia
yury.stotskiy@yutax.ru

***Abstract*—Nowadays simultaneous localization and mapping (SLAM) algorithms are being tested at least in two phases: software simulation and real hardware platform testing. This paper describes hardware design and control software for small size omni-directional wheels robot implemented for indoor testing SLAM algorithms.**

I. Introduction

Omni-wheel robots are widely used in mobile robotics. They have unique ability to move instantly in any direction from any pose [1], but they require more complex control algorithms and sensitive to underlying surface. Omni-wheel motion is not new but still actively discussed [2]. Our goal is create small mobile platform for testing SLAM algorithms in indoor environment, which will satisfy next requirements:

- based on common, replaceable and chip hardware;
- small sized and small weighted (not more than 2 kg and 20x20x20 cm);
- ability to move with relatively high speed – up to 1 m/s;
- provide good connection with different indoor surfaces (like carpets, wood, concrete,...);
- have enough space for sensor installation (up to two mini-RGB cameras and four infrared/ultrasound distance sensors).

The rest of paper is organized following way. In Section II we discuss the hardware design of robot. Section III is dedicated to motion algorithms implementation. Section IV describes calibration and tuning approaches. Sections V and VI present the robot software architecture and control protocol implementation. Evaluation of mobile robot design and future plans discussed in Section VII. The conclusion is in Section VIII.

II. Hardware design

According the requirements, all parts of robot should be relatively cheap and available widely on market, to be easily replaced. We used standard mechanics and controllers.

A. Mechanical part

We selected centrosymmetric design with four circumferentially spaced omni wheels as shown in Fig. 1. Wheels are directly put to four electric motors contained build-in speed reduction units. Motors are mounted at the bottom side of silumin box. The box provides constructional strength and flexibility for installing electronic components inside and outside.

Robot radius R_c is equal to 9.6 cm. Angle between motor 1 and motor n spindles D_n is equal to $90° \cdot (n-1)$.

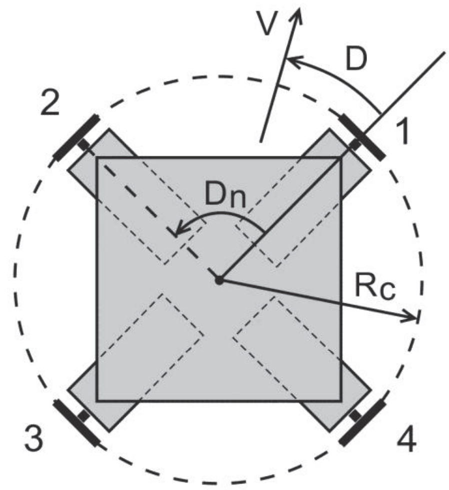

Fig. 1. Omni-wheel mobile robot structure

Initial motors mount design assumed fixed mounting all four motors and wheels to the box bottom plate. However, such design does not ensure all wheels touch floor constantly and first motion experiments demonstrated path instability. Then we added simple suspension to two adjacent wheels. It improved path stability dramatically even without any springs and absorbers. Other two wheels are kept fixed mounting.

The robot uses omni wheels shown in Fig. 2. Wheel radius r is 25 mm. The wheels have rubber rollers. This allows avoiding slipping wheels problem defined in [3] even at wood or plastic (linoleum) surface. However, this kind of wheels demonstrates big friction that requires robot calibration and applying compensating correction as described below.

B. Electronics

Electronic part of the system consists of two double-channel L298 motor drivers, Arduino based controller [4], Raspberry Pi [5] main onboard computer and Li-Pol 7.4 V battery as it shown

ISBN 978-952-68397-0-7 (paperback), ISBN 978-952-68397-1-4 (PDF)

in Fig. 3. The battery powers Arduino controller, motor drivers and motors directly. Raspberry Pi unit is powered by using 5 V regulator.

The controller provides pulse-width modulation (PWM) signal for motor drivers calculates and executes basic motion algorithms and processes sensors input. The controller is also responsible for buffering motors and sensors related command from onboard computer, maintaining commands execution timeline and issuing events and statuses back to onboard computer.

C. Communications

Onboard computer and the controller communicate each to other by UART at 115200 baud rate using level shifter between units that utilize 3.3 V and 5 V TTL levels respectively. High communication rate makes it possible to send more than 100 commands per second that allows changing motion parameters every centimeter of the robot route. Actual limitations of commands flow are amount of Arduino controller memory and the controller CPU clock rate. These hardware limitations define maximum buffer length and ability to execute buffered commands on time. We found that Arduino controller takes up to 2 ms to process an individual command. Therefore UART communication rate is enough for selected design.

Another reason of using exactly 115200 baud rate selection is ability to upload motor controller software over UART without UART reconfiguration because this baud rate is used by Arduino boot loader.

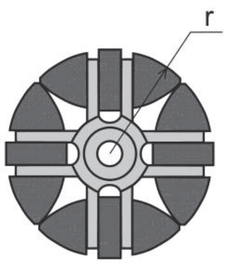

Fig. 2. Omni wheels used in the construction

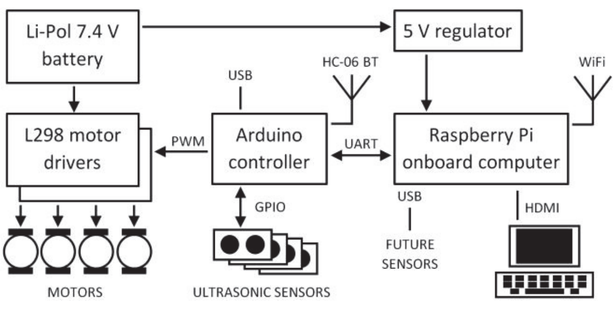

Fig. 3. Electronic components

WiFi dongle is installed to one of two available USB ports at Raspberry Pi. It provides connectivity between robot and local network and Internet. Communication with robot could be made over ssh.

For debugging and uploading initial Arduino controller software we utilized external computer connected to the controller by using Arduino USB connector or HC-06 Bluetooth module connected to the controller UART instead of Raspberry Pi.

We also kept ability to connect remote USB keyboard and mouse and HDMI based display to Raspbery Pi.

C. Sensors

Ultrasonic HC-SR04 distance sensor is installed at one of the robot box side and connected to Arduino GPIO pins. Based on our experiments the sensor can detect distance from 3 to 500 cm. We are going to extend number of ultrasonic sensors in future.

There are plans to mount other sensors like video camera and laser locator but exact sensors configuration depend on planned research and still under discussion.

III. Simple Motion Algorithms

At the first stage, we implemented two motion algorithms: Move and Rotate. They are enough to perform arbitrary motion in a room that has complex shape and some interior. They also allow orienting robot and its sensors in required direction.

These algorithms are based on known formulas as defined below. Exact software implementation is created from scratch by the article authors.

A. Move

This universal motion defines the robot movement along a curve (including straight-line motion) at specified velocity as shown in Fig 4.

A motion specified by the following parameters:

1) *Distance L* is requested distance to move the robot for.

2) *Velocity V* is linear speed of the robot geometric center.

3) *Curve C* is the robot route curvature that is reciprocal of the route radius (see Fig. 4).

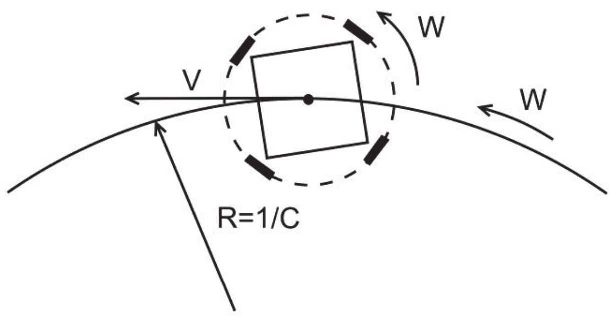

Fig. 4. Motion along a curve at constant course relative to the robot body

4) *Course D* is direction followed by the robot relative to the robot “head” as shown in Fig 1. Course is defined as

angle between wheel number 1 motor spindle direction (robot “head”) and the robot motion line.

If Curve is equal to zero then Move is straight-line motion and corresponded motor number n power P_{nl} can be defined by formula (1) in accordance with [6].

$$P_{nl} = K \cdot K_l \cdot V \cdot \sin(D+D_n) \quad (1)$$

Here K is power per cm/s factor defined by motor characteristics. K_l, (at this formula), Kt, and K_r (at next formulas) are calibration factors connected to wheels friction and motors resistance and depend on velocity, course and curve values. These factors discussed below.

To implement motion at Curve different from zero we have turning the robot for full circle relative to the floor as soon as the robot traveled $2\pi/C$ path length. This is equivalent to adding the robot rotation around its geometrical center at circular frequency $W = V \cdot C$ as shown in Fig. 4.

Robot geometry defines $W \cdot R_c = w \cdot r$ equity for the rotation, where r is wheel radius and w is the wheel rotation circular frequency. Required rotation means adding supplement velocity $V_t = w \cdot r$ to each motor perpendicularly to the motor axis.

Using V_t in formula (1) and considering $D + D_n = 90°$ we can calculate additional motors power P_{nt} required for turning.

$$P_{nt} = K \cdot K_t \cdot V \cdot (R_c \cdot C) \quad (2)$$

Total motor power for Move motion is $P_n = P_{nl} + P_{nt}$. Time to apply calculated motor power is $T = L/V$.

B. Rotate

Move motion type cannot ensure rotation around the robot geometrical center. Rotate motion used instead and specified by:

1) Angle A to turn.

2) Frequency F of rotations in angle units per second.

For this case motors power P_{nr} and time T can be calculated as

$$P_{nr} = K \cdot K_r \cdot 2\pi \cdot F \quad (3)$$

$$T = A/F \quad (4)$$

The benefit of these two simple motion types is motor power permanence in time for each individual motion. Motors power is a constant of time at equations (1) to (3). This reduces motor controller algorithms complexity. However, such useful motions as, for example, linear travel and simultaneous robot rotation cannot be executed by applying static motor powers. Such motion implementation requires dividing travel path to short sections (about 2 cm in the robot path length) and issuing separate Move or Rotate command to motor controller per each section. At maximum robot speed 100 cm /s, it requires sending up to 50 commands per second from onboard computer to motor controller that may overload UART communication channel and the controller command queue.

IV. CALIBRATION

Equations (1) to (4) describe motion for ideal case. Real algorithms have to take motor resistance, minimal motor start power, wheel friction and wheel slipping into consideration. That means K_l, K_t and K_r coefficients are not constant but functions of velocity and course. These functions are different for each of three power entries: P_{nl}, P_{nt} and P_{nr}.

We did three series of experiments to measure linear movements at 0° and 45° courses and rotation (where distance corresponds to wheel path along circumference R_c). Fig 5 shows experimental distance in cm passed by the robot in one second for different motors power.

For 0° motion case, the power is applied to two opposite motors in positive and negative rotate direction respectively. Two other motors are unpowered. For 45° motion case the same power is applied to all motors but even and odd motors (as numbered in Fig. 1) were powered in opposite directions. For rotation case, all motors are powered to turn in the same direction.

We found that floor surface material affects wheel friction and slipping and introduces a difference up to 20% for the motion distance and curvature if the same power applied to motors in experiments. Therefore, calculated and calibrated P_{nl}, P_{nt} and P_{nr} cannot ensure precise motion without feedback from some sensors. We get calibration data for two reference surfaces (wood and carpet floors) and applied piecewise linear approximation for calibration functions based on average data obtained at different surfaces.

Such approach provides ability to control the robot motion with reasonable error and use simple linear approximation algorithms at the same time. This allows reducing motor controller CPU load.

Fig. 6 shows used calibrations graph for K_t and K_r. Initially we expected K_t to be equal to K_r because both P_{nt} and P_{nr} are motor power to perform the same robot motion type (rotation around robot center). Experimental data proved that rotation without movement requires more power than turning adjustment.

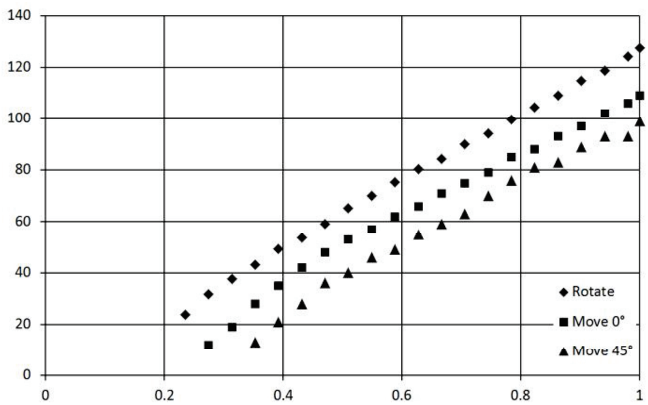

Fig. 5. Dependence of rate (cm/s) on motors power (PWM pulse ratio)

We assume rotation spends additional power to get the robot moving on but turning adjustment applied to already moving robot and therefore this power addition is included to P_{nl} part of the motion calculation.

In opposite to K_t and K_r that depend on power only K_l is a function of power and course. Dependency on course is caused by two reasons. The first one is wheel friction variation and another one is power reduction by sinus function in (1) if course is close to 45°.

Therefore, K_l calibration function provides both experimental and theoretical adjustments. Our motor controller logic calculates K_l calibration function for courses from 0° to 45° as shown in Fig. 7. To calculate K_l for courses over 45° we use angle between course and nearest motor axis to convert the course to 0° – 45° range for calibrating.

During experiments, we found that straight-line motion at courses close to any motor axis direction is less predictable because two wheels (almost) do not rotate in such cases. These two wheels are located across motion path and make largest contribution to friction. Friction of wheel we use (see Fig. 1) depends very much on wheel angle position in the plane of the wheel rotation. This angle position can vary K_l as much as twice. For calibration measurements at course 0°, we locate front and back wheels to predefined position (with less possible friction) in advance.

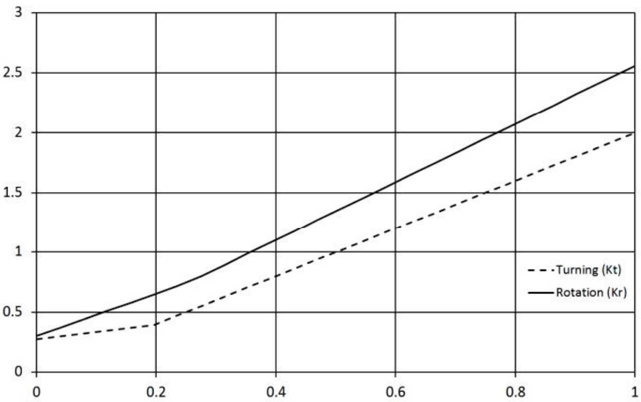

Fig. 6. Power (PWM pulse ratio) dependence of K_t and K_r calibration factors

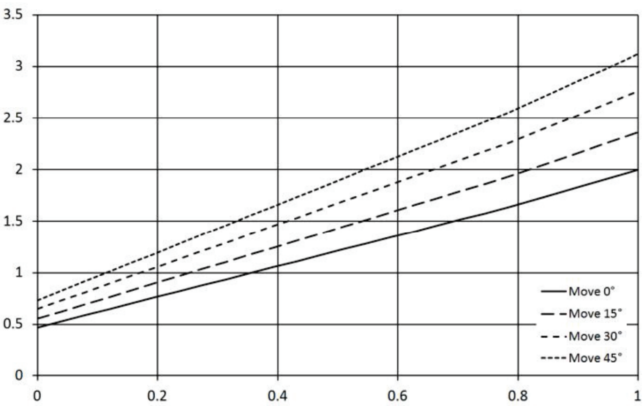

Fig. 7. K_l factor for different courses dependence on power

Considering wheels angle position cannot be set at real robot movements we recommend to use courses close to 45° for straight-line motion. Motion with significant curvature or complex motions discussed below are not so sensitive to wheel friction instability because all wheels are constantly rotating.

V. SOFTWARE DESIGN

This section describes control software for omni-wheel robot including basic principles, communication, and sensors. High level robot control algorithm is shown in Fig. 8.

Considering Arduino CPU performance limitation we avoided using any third party Arduino libraries and implemented all required logic inside our own code to control execution timing.

A. Motor controller

Arduino runtime library executes main application loop infinitely. Any motor controller motion and communication algorithms are initiated by some action in this loop. This architecture applies such limitation as inability interrupting an action and as a result the requirement to keep any action execution time below some reasonable time. We keep this time below 20 ms that corresponds to maximum motion execution inaccuracy about 2 cm at the maximum robot speed 100 cm/s because of possible delaying motion command execution.

The motor controller software is created from scratch. No external additional Arduino libraries are used. This allowed predicting code execution timing and memory usage.

B. UART communication

String based command protocol is implemented to send commands from onboard computer to motor controller and receive controller states and events.

The protocol defines any command or event as ASCII characters string consists of key word followed by up to five signed integer numeric parameters separated by comma and string terminator that can be ";" symbol or any combination of LF and CR symbols. Maximum number of command parameters is a trade-off and limited by the controller memory allocated for command queue.

One of the infinite loop actions reads UART stream and calls command parser as soon as complete command string detected in the stream. The parser executes the command immediately or put it to circular buffer (command queue). All motion control commands except immediate stop instruction are queued to the buffer to be run one by one as soon as previous command execution time T expired.

C. Commands execution

Another action of the application loop monitors active motion execution time and reads next command from the queue as soon as current motion completed. Command parameters can instruct controller to run corresponded motion infinitely. Such motion continues to be active while command queue is empty (no any motion command queued by parser after current one). As soon as next motion command detected in the queue the infinite command interrupted immediately to run next motion instead.

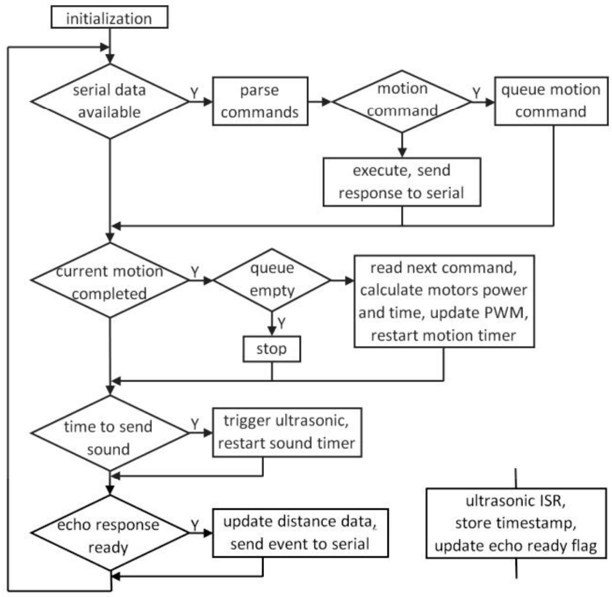

Fig. 8. Arduino based motor controller firmware stack

D. Pulse-width modulation

Standard Arduino runtime library supports build-in ability to generate PWM modulated TTL signal at controller output pin by single function call executed by the application. As soon as the signal duty factor assigned the controller continues issuing the PWM signal while another duty factor not assigned. This allowed us to connect motor drivers to PWM output pins and apply required P_n or P_{nr} power to a motor permanently by single call in the motor controller application.

E. Ultrasonic sensor

Third action of the motor controller application loop is periodical echolocation by using ultrasonic sensor. Command protocol provides ability switching echolocation on or off, defines distance detection rate and configuring two thresholds. As soon as detected distance to remote object is inside (or outside) of defined thresholds the controller issues event notification to onboard computer.

The echolocation can be configured to initiate immediate stop if an object detected too close to the robot.

Implemented echo processing algorithm uses GPIO pin edge interrupt subroutine to detect ultrasonic echo response time and do not blocks main loop execution while sensor is waiting for echo.

Our experiments proved stable echo detection at distances up to 500 cm.

F. Onboard computer

Raspberry Pi onboard computer contains full functional Linux package installed at SD card. Network connection over WiFi allows updating and installing required libraries. Raspberry Pi Linux is configured to avoid using UART as kernel logging console and TTY.

We additionally installed Git and Arduino IDE for ARM. As a result full software development cycle can run using the robot computers only. This includes motor controller software development and uploading it to Arduino, onboard computer software development, building, testing and committing it to external project repository.

Considering most hardware specific logic is encapsulated into controller software we assume external powerfull computer with cross-compiler can be used to develop and debug complex algorithm.

Wiringpi2 for Python and wiringpi for C libraries are used to access Raspberry Pi based GPIO pins and communication channels. Minicom is installed to debug motor controller command protocol.

Onboard computer application implements a parser similar to the parser at motor controller. The application shares header file contained communication protocol definitions and the robot limits with motor controller software. This guarantees communication protocol compatibility.

The computer is going to be used for implementing advanced robot motion and positioning, SLAM and image processing algorithms, and robot navigation tasks. However, simple test applications only are available for today.

Because of possible power instability or battery discharge SD card file system can be corrupted or SD card can be damaged physically. We recommend backup the card image periodically to avoid data lost. Monitoring battery power and automatic system shutdown can be implemented to protect SD card file system if battery discharged below critical level.

VI. MOTION CONTROL IMPLEMENTATION

A. Motion commands

Following protocol commands used to implement Move and Rotate motions:

1) MOVE,L,V,D,C; command requests the robot movement. Special distance value used to request infinite motion.

Motor controller remembers parameters of last executed MOVE command. This allows issuing motion command as a derivative of last executed Move motion or in other words a derivative of current robot movement. A usual robot application processes some sensor feedback and adjusts current motion to return the robot to planned route. Therefore, ability to apply incremental adjustment is important feature of motion control algorithm.

2) DELTA,L,V,D,C,R; command is used to request Move motion with parameters calculated as sum of last Move motion values and provided DELTA parameters. In other words, DELTA command defines derivatives to current motion parameter values.

Repeat (R) parameter specifies how many times repeating the DELTA command before processing next command from the queue. Infinite reiteration can be requested by special value of the Repeat parameter. As soon as next command is available in the queue, the infinite reiteration interrupted just after current repetition completed.

3) ROTATE,A,F; command requests rotation around the robot center. Similar to distance case there is special angle value to request infinite rotation. ROTATE command does not change last Move motion values. Therefore, DELTA command is able to use the values even if ROTATE command issued in between.

4) STOP; command interrupts current motion, stops all motors, clear the queue and resets last MOVE parameters to all zeroes. The command is not queued but executed by the parser immediately.

B. Parameter limits

Parameter limits are defined by the robot geometry and characteristics (such as maximum motor power, wheel friction) and Arduino platform maximum integer value limit. Any command is checked by the parser against limits before writing the command to the queue. We apply the following limits for MOVE and ROTATE command parameters:

–500 cm < L < 500 cm.

30 cm/s < |V| < 100 cm/s. Minimal possible velocity defined by minimal motors power required to overcome mechanical and electrical resistance and get robot moving. Combination of distance and velocity signs specifies forth or back motion along given course.

–360 < D < 360 degrees.

–100 < C < 100 where C is measured as 1000/cm. Curve sign specifies route bend to the left or right relative to the given course. Curve equal to 100 corresponds to the robot motion around some point at the robot outer radius.

–30000 < A < 30000 degrees. A value more than 360° is used to request several turnovers.

–900 < F < 900 degrees/s. Combination of angle and frequency signs specifies clockwise or counter clockwise rotation direction.

DELTA parameter limits are twice more than corresponded MOVE limits to provide ability switching MOVE parameters in full available range by single DELTA command.

0 < R < 30000. If R equal to zero DELTA command is executed ones.

C. Complex motion algorithms

Basic motion algorithms Move and Rotate provide ability to relocate the robot to any position and orient the robot sensors. However, planned missions require moving the robot along complex path and turning it simultaneously.

Developed DELTA command allows realizing various complex motions.

As an example two subsequent commands MOVE,2,40,0,0; DELTA,0,0,10,0,36; initiate 37 movements to 2 cm at 40 cm/s speed. Each next movement change course to 10 degrees compare to previous one. As a result, the robot moves around a circle but keeps its orientation relative to ground.

An even more powerful feature is ability to specify infinite DELTA command repeat sequence by assigning special value (30000 in our implementation) to R.

The following two commands MOVE,3,60,0,–40; DELTA,0,0,10,0,30000; initiate infinite robot movement along straight line and simultaneous robot rotation around the robot center because first command specify 2 cm motion along the left bend curve but subsequent DELTA adds permanent course switching in opposite side that compensate bend.

Such complex movement considers frequent changing motor powers. The key feature is executing this motor power regulation by motor controller internal algorithms and avoiding huge command flow from onboard computer to the controller.

D. Additional commands

The command protocol implements additional commands to provide the following operations and notifications:

- Apply exact motor power to each motor for defined time. The command is mainly used for debug and calibration purposes.
- Switch operation modes like simulating motion without applying actual power to motors.
- Request current motor controller status including applied motor powers, last MOVE parameters, and command queue usage.
- Report errors and warnings happened in the controller parser and algorithms.
- Ultrasonic sensor configuration and reporting detected distance.

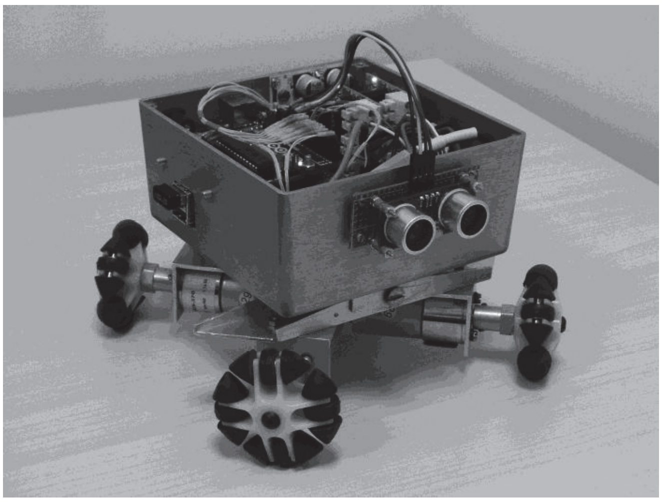

Fig. 9. The robot picture

The command protocol detailed description is out of this article scope.

VII. Evaluation and Further Work

Currently we have simple and robust platform for real indoor experiments and extensions. Mechanical design demonstrated strength. The robot was tested indoor and at some outdoor surfaces at up to maximum speed and acceleration without any damage. The robot is small enough to be used in rooms with limited space and still provides ability to mound additional sensors and electronic equipment.

Today we can formulate the following possible improvements to the current functionality:

- Add sensors feedback to motion control and implement motion primitives in onboard computer application library.
- Install additional ultrasonic sensors and calibrate them to use their output as reference distance to interior items.
- Design and test universal connection for fast installing sonars and cameras outside of robot body.
- Add binary command communication protocol over UART to reduce communication channel overhead and optimize parser performance.
- Install Robot Operating System on Raspberry Pi computer, test performance and tune ROS stack for running on low power devices.
- Test different omni wheel types.

In a long term perspective we plan the following activities:

- Install additional sensors and develop an application that creates interior maps and navigates around the room.
- Use the platform to research SLAM algorithms and test available SLAM solutions at real situation.

VIII. Conclusion

In this paper we discussed problems and solutions discovered during small robust omni-wheel robot implementation. The robot is designed for SLAM algorithms indoor research. Full stack from hardware implementation up to high level software was presented. Real prototype is shown in Fig. 9.

Robot software is available in project repository https://github.com/OSLL/omnibot/.

Acknowledgment

Authors would like to thank Saint-Petersburg Computer Science Center, for provided equipment. Some parts of this paper have been prepared within the scope of project part of the state plan of the Board of Education of Russia (task # 2.136.2014/K).

References

[1] M.O. Tatar, C. Popovici, D. Mandru, I. Ardelean, and A. Plesa, “Design and development of an autonomous omni-directional mobile robot with Mecanum wheels”, *2014 IEEE International Conference on Automation, Quality and Testing Robotics (AQTR)*, DOI: 10.1109/AQTR.2014.6857869.New York: Wiley, 1991.

[2] I. Doroftei, V. Grosu, and V. Spinu, “Omnidirectional mobile robot - design and implementation”, Book-Chapter, Bioinspiration and Robotics - Climbing and Walking Robots, Editor Maki K. Habib, Published by Advanced Robotic Systems International (Vienna) and ITech, 2007, pp. 511-529.

[3] Raul Rojas, “Omnidirectional control”, Freie Universitat Berlin, Web: http://www.inf.fu-berlin.de/lehre/WS04/Robotik/omnidrive. pdf.

[4] Arduino / Genuino UNO official website, Web: https://www. arduino.cc/en/Main/arduinoBoardUno, access date 29.09.2015.

[5] Raspberry Pi 2014 website, Web: https://www.raspberrypi.org/ documentation/hardware/raspberrypi/schematics/Raspberry-Pi-B-Plus-V1.2-Schematics.pdf, access date 29.09.2015.

[6] L. Huang, Y.S. Lim, David Li, and Christopher E.L. Teoh, “Design and analysis of a four-wheel omnidirectional mobile robot”, *Second International Conference on Autonomous Robots and Agents*, December 13-15 2004, Web: http://www-ist.massey.ac.nz/ conferences/icara2004/files/Papers/Paper74_ICARA2004_425_428. Pdf.