Hemispherical omnidirectional gimbaled wheel-based mobile robot Final Report

Maciej Bałuta
Artur Dróżdż

Intermediate Project
PhD Witold Paluszyński

Wrocław University of Science and Technology
Faculty of Electronics
Department of Cybernetics and Robotics

February 16, 2017

1 Overview

Mobile robots are commonly used nowadays. Taking a look at motor vehicles we may consider, that almost all of them have power-trains designed with the same idea. We can easily separate engine, transmission, drive shafts, differentials and the final drive. Torque is transferred from the engine through a drive-shaft to the wheels, hence the wheels' movement speed is proportional to their radius and angular speed of the motor. However, there exists a special kind of power-train. Its unusualness is caused by a hemisphere rotating with large angular speed. When the point of contact with the ground lies on the rotation axis then no movement is present. The driving force appears when rotating hemisphere leans out in one of the two axis parallel to the ground plane. The contact point has some linear velocity proportional to rotor's angular speed and trigonometric combination of flap angles in two axes. The hemisphere has a few functions: accumulates kinetic energy alike flywheel, has a role of the final drive components like usual wheels and changes seamlessly gear ratio in the same way as stepless transmission. Historically, the first known research was made in 1938 according to the article [1]. Some known research work was made by students from research group [2] and other people [3].

2 Construction

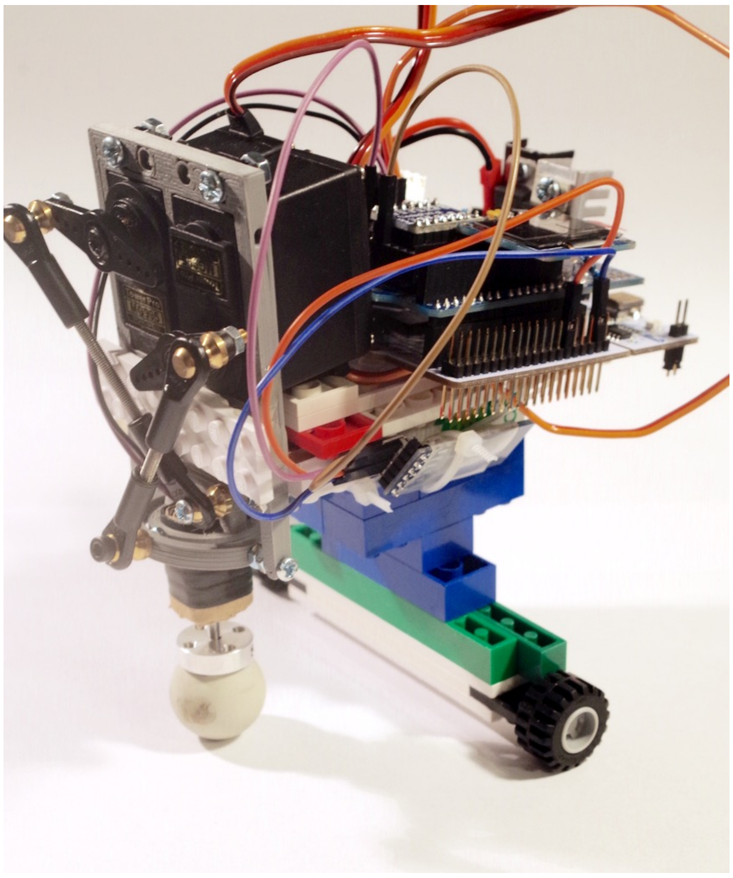


Figure 1: Final result — robot with HOG drive

Overview of the final construction is shown on fig. 1. Most important details are shown on fig. 2. There are 2 servos that control the axis of the hemisphere and a DC motor that rotates the hemisphere. We used popular *TowerPro MG995* servos with some cheap DC motor. We think they were good enough for construction of that size and we haven't encountered any problem with power or accuracy.

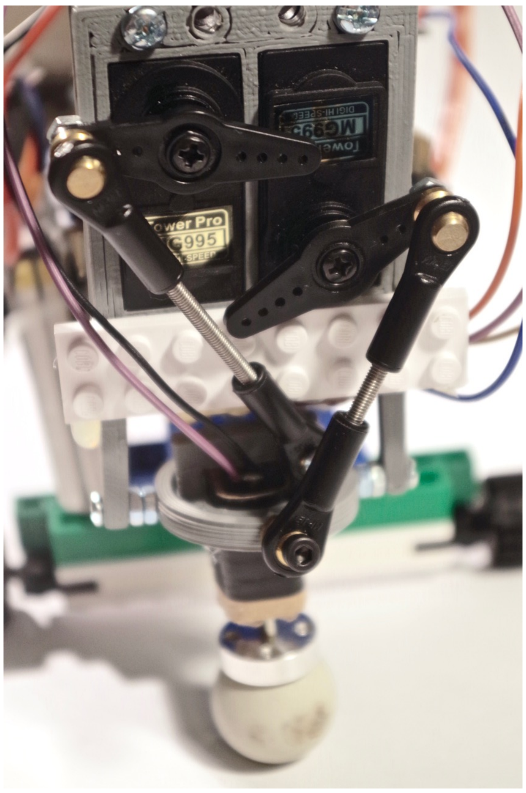


Figure 2: Detailed view of HOG drive

2.1 Frame model

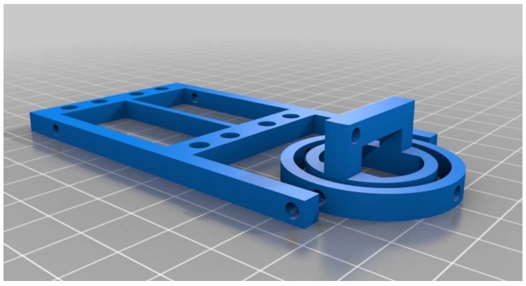


Figure 3: 3D model used for main part of drive

We used an open-source CAD model [4] created by *i-make-robots* (shown on figure 3). Unfortunately, this model has many disadvantages. Small parts caused problems with finding appropriate elements, f.e. rod ends should be smaller, but it was impossible to buy such ones. After creating the model on 3D printer it occured, that its stiffness is not sufficient, therefore there were some clearances on fastenings. The rest of the construction was made at first from aluminium profiles, then the LEGO blocks were used due to the fact, that they offered more possibilities to adapt for our needs. LEGO tyres made from rubber gripped well to the ground.

2.2 Inverse kinematics

Having such type of construction it was necessary to calculate inverse kinematics. Using Denavit-Hartenberg notation we created a function taking inclination of the hemisphere in one of the axes on the input and giving back servo's angular position as a result. Possible angular positions are restricted due to mechanical limitations caused by specific length of connectors between servo and rings surrounding the hemisphere and by other parts' dimensions. Changing coordinates system is done in the following way:

$$A = Rot(Z, -\frac{\pi}{2}) \cdot Trans(X, a_3) \tag{1}$$

$$B = Trans(Z, -d_1) \cdot Trans(X, -a_1) \cdot Rot(Z, -\frac{\pi}{2}) \cdot Trans(X, a_2) \tag{2}$$

$$R = Rot(X, \Theta_1) \cdot Rot(Z, \frac{\pi}{2}) \cdot Rot(X, \Theta_2) \cdot Rot(Z, -\frac{\pi}{2}) \tag{3}$$

$$P_1 = R \cdot A \tag{4}$$

$$P_2 = R \cdot B \tag{5}$$

where:
Θ_1, Θ_2 - angles of desired hemisphere's inclination

Having these equations we can calculate coordinates of the point defined as hole at the end of rudder bar which is connected by rod with the bar surrounding the hemisphere. Using calculated point's coordinates we can easily get the desired angle of servo. For the first servo we have:

$$(x - P_{2x})^2 + (y - P_{2y})^2 + (z - P_{2z})^2 = {R_2}^2 \tag{6}$$

$$y = -a_4 \tag{7}$$

$$(z + d_3)^2 + (x + a_5)^2 = r^2 \tag{8}$$

For the second servo:

$$(x - P_{1x})^2 + (y - P_{1y})^2 + (z - P_{1z})^2 = {R_1}^2 \tag{9}$$

$$y = -a_4 \tag{10}$$

$$(z + d_2)^2 + (x - a_5)^2 = r^2 \tag{11}$$

where:
P_1 and P_2 are the points calculated in previous step

a_3 - distance from the middle of the ring to the middle of the first rod's end
d_1 - height of the engine's holder box
a_1 - width of the engine's holder box
a_2 - distance between engine's holder box and the middle of the second rod's end
R_1 - length of the second servo push rod
R_2 - length of the first servo push rod
d_2 - distance between the middle of the ring and first servo rotation axis
d_3 - distance between servos rotation axes
Solving this system of equations is done in the following way:

$$a = -2P_{2x} \tag{12}$$
$$b = -2P_{2z} \tag{13}$$
$$c = {R_2}^2 - (-a_4 - P_{2y})^2 - {P_{2x}}^2 - {P_{2z}}^2 \tag{14}$$
$$d = 2a_5 \tag{15}$$
$$e = 2d_2 \tag{16}$$
$$f = r^2 - {a_5}^2 - {d_2}^2 \tag{17}$$
$$g = -\frac{a-d}{b-e} \tag{18}$$
$$h = \frac{c-f}{b-e} \tag{19}$$
$$i = g^2 + 1 \tag{20}$$
$$j = d + 2gh + eg \tag{21}$$
$$k = f - h^2 - eh \tag{22}$$
$$ix^2 + jx + k = 0 \tag{23}$$
$$\tag{24}$$

Solving the last equation with respect to x we get the result. Delta of this quadratic equation determines existence of the solution. Finally we have:

$$\Phi_1 = atan2(y_1, x_1) \tag{25}$$
$$\Phi_2 = atan2(y_2, x_2) \tag{26}$$

Φ_1 and Φ_2 are desired angles of servos.

3 Hardware

3.1 Electronics

We used ST Nucleo boards with STM32 F4 series microcontrollers with Arduino-style shield described below. We wanted to use as many ready-to-use components as possible to focus on mechanics and algorithms. We had to use separate power supplies, because servos operate with 5V and all logic components with 3.3V.

3.2 Shield

We needed to extend the Nucleo board with more components such as:

- power supply,
- DC motor driver,
- and OLED display.

We decided to make an Arduino-style shield with all these components and also external connectors for servos. This saved us a lot of problems caused by tangled, loose cables. It was also possible to change Nucleo board. The shield is shown on figure 4.

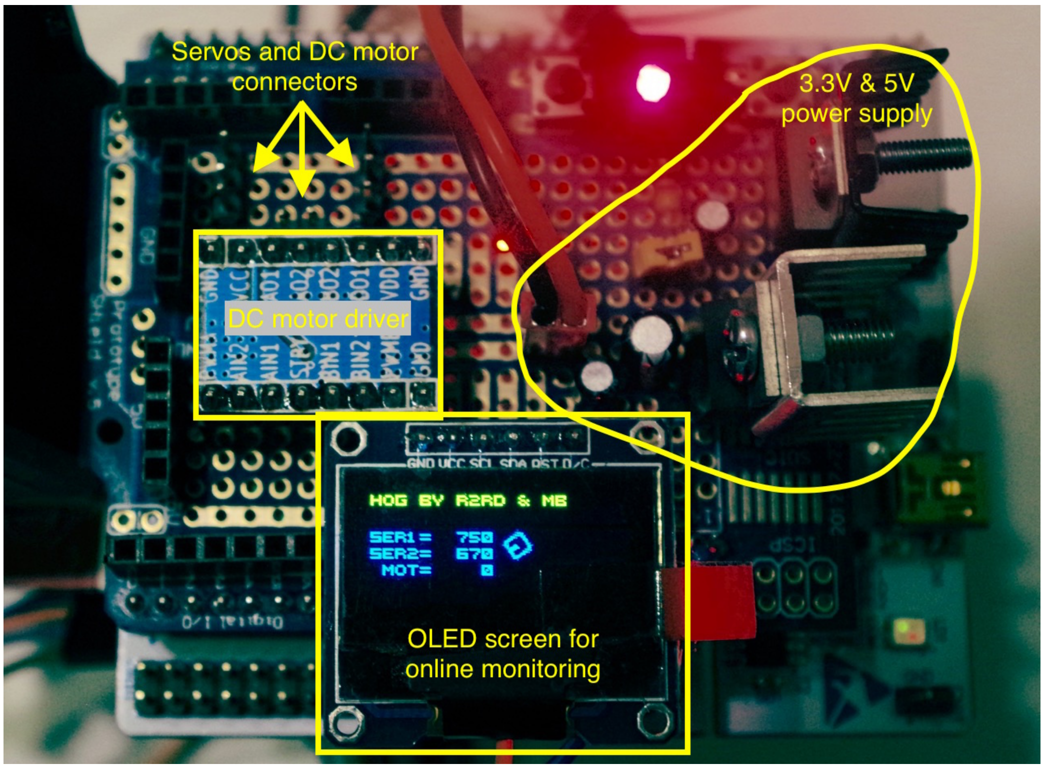


Figure 4: Shield connected to Nucleo board

Display shows current PWM settings of servos and DC motor and some rotating box that indicates if the system is working — everything happens in main loop, so if the program gets stuck, the box does not rotate.

4 Software

4.1 Microcontroller program

The microcontroller was used in order to:

- control servos for gimbal system and DC motor of the hemisphere using PWM signal of desired duty and frequency,
- provide an interface for the user to modify the parameters online (via serial interface),
- show current state of the system to ease debugging.

The controller was programmed in C language using HAL libraries provided for STM32 platform. Hardware configuration was generated using

STM32CubeMX. Such approach allowed easy and fast prototyping. In our opinion the platform is stable enough to use for complex projects, even though it is relatively new on the market.

4.1.1 CMSIS

We discovered CMSIS library (Cortex Microcontroller Software Interface Standard) which contains *DSP toolkit*. It contains a lot of functions useful for robotic systems and we were especially interested in matrix manipulation. What is important is that they are well optimized for Cortex microcontrollers and they use features of ARM cores like SIMD operations. Despite being written in C with assembler, CMSIS has easy to use object oriented interface.

We used the library to implement compute inverse kinematics of the gimbal. The library worked well, even though we used software implementation of floating point numbers.

We implemented functions that create rotation and translation matrices that are later multiplied to solve kinematic chains. An example function looks in the following way:

```
void matrix_create_rotationX(arm_matrix_instance_f32 *mat,
     float32_t angle)
{
   float32_t ang_sin, ang_cos;
   arm_sin_cos_f32(RAD2DEG(angle), &ang_sin, &ang_cos);

   arm_fill_f32(0, mat->pData, 16);
   mat->pData[MAT_INDEX(0,0)] = 1.0;
   mat->pData[MAT_INDEX(1,1)] = ang_cos;
   mat->pData[MAT_INDEX(1,2)] = -ang_sin;
   mat->pData[MAT_INDEX(2,1)] = ang_sin;
   mat->pData[MAT_INDEX(2,2)] = ang_cos;
   mat->pData[MAT_INDEX(3,3)] = 1.0;
}
```

We compared the results from CMSIS based program with a GUI implemented in C# and the results were the same and during testing there were no problems with precision or overflows.

4.2 Serial communication protocol

The system communicates over Bluetooth serial link using HC-05 module. The protocol was text-based and the following commands were implemented:

`?` - returns current PWM settings for servos and DC motor:
`SER1=%d, SER2=%d, MOT=%d`

`r` - resets PWM settings to initial values and returns the state as in `?` command.

`a ROLL PITCH` - sets roll and pitch angles for the gimbal. `ROLL` and `PITCH` are given as angle * 100 without point, because standard implementation of `sprintf` does not support floating point numbers. Returns PWM settings as in `?` command if angles set properly, otherwise it may return "`Wrong value!`" if given angles are out of range or "`Forbidden SER1=%d, SER2=%d`" if the given angles lead to a position that is forbidden in software (to avoid damage).

`m MOT` - sets motor PWM value and returns all PWM settings as `?` command if successful, "`Wrong value!`" if out of range.

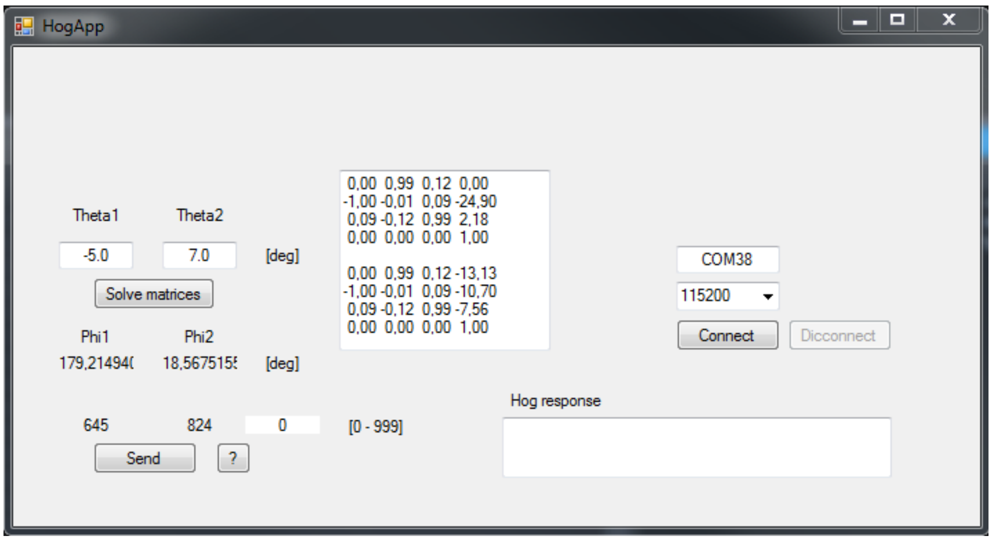
`SER1 SER2 MOT` - sets values of PWM for servos and DC motor. Return given settings as `?` command if successful, "`Forbidden setting`" if such config is forbidden in software, "`Wrong value!`" if the values are out of range.

All arguments are checked for being valid. If the issued command syntax is invalid, microcontroller returns "`Wrong arguments`" message.

4.3 GUI

4.3.1 Inverse kinematics solver

The first GUI was implemented in C#. It provided possibility to calculate inverse kinematics, display rotation matrices, transform results to servo coordinates and send data through serial port directly to the driver board. It was replaced by other GUI written in C++ because of better portability.



Theta1: -5.0 [deg]
Theta2: 7.0 [deg]

```
 0,00  0,99  0,12  0,00
-1,00 -0,01  0,09 -24,90
 0,09 -0,12  0,99  2,18
 0,00  0,00  0,00  1,00

 0,00  0,99  0,12 -13,13
-1,00 -0,01  0,09 -10,70
 0,09 -0,12  0,99 -7,56
 0,00  0,00  0,00  1,00
```

Phi1: 179,21494(
Phi2: 18,567515! [deg]

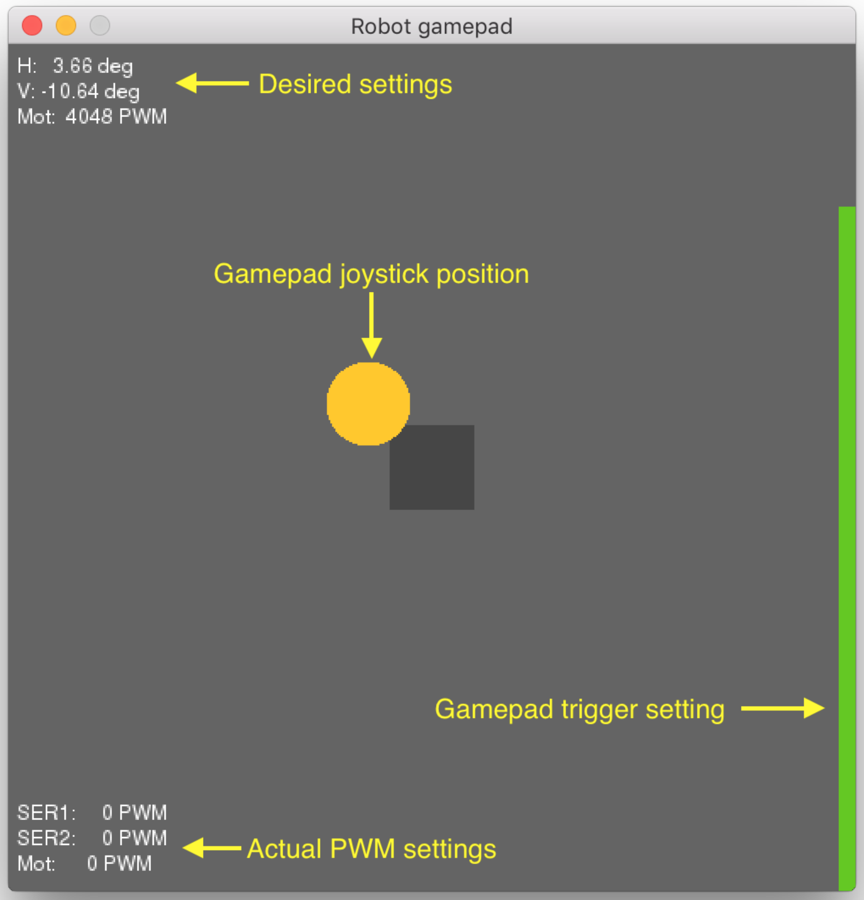
645 824 0 [0 - 999]

COM38
115200
Hog response

Figure 5: First version of GUI

4.3.2 Game controller interface

The seconds graphical program we developed was created in order to control our robot with a game controller. Joystick (2D analog controller) is used for setting gimbal position and trigger (1D analog controller) is used for setting DC motor speed. Such interface is intuitive and feels natural for such kind of robot.



```
H:  3.66 deg
V: -10.64 deg
Mot:  4048 PWM
```

```
SER1:    0 PWM
SER2:    0 PWM
Mot:     0 PWM
```

Figure 6: Gamepad interface

The program is written in C++. We used SDL 2.0 library for accessing game controller and drawing user interface. Program communicates with robot via Bluetooth serial link and sends desired angles and motor speed. Robot responds with set PWM values.

Quite surprisingly the trickiest part of this program was serial communication. At first we used serial interface built in Nucleo. We tried to use some serial port library, but after sending some commands the communication was stuck and it was needed to unplug the Nucleo board and plug it again. After hours of debugging it turned out that the communication channel simply blocks when the input buffer gets full. When we changed the code to read data sent by the microcontroller everything started to work flawlessly.

5 Results and conclusions

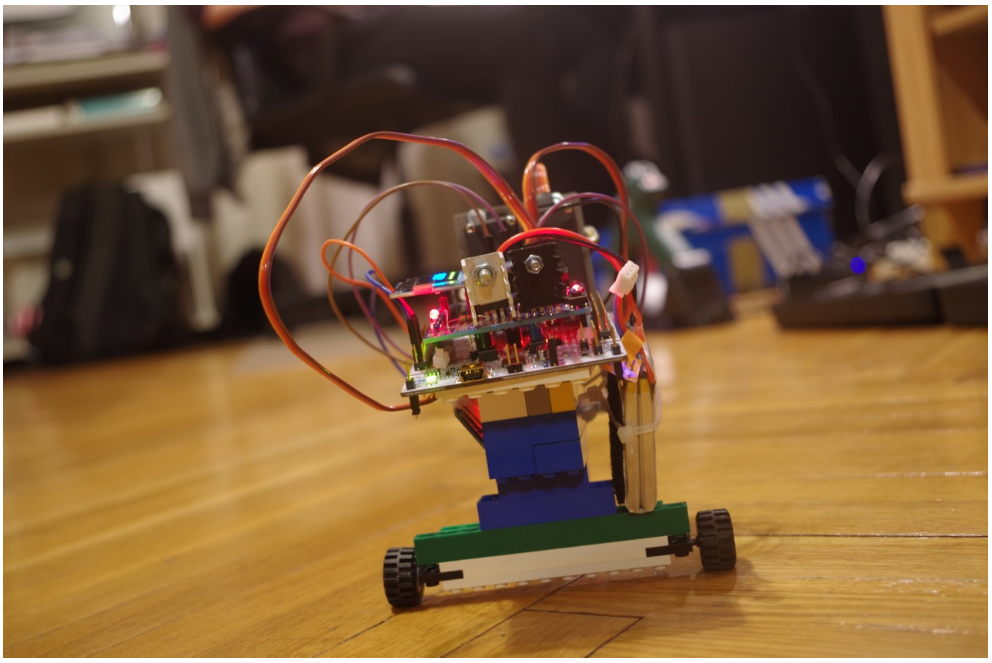


Figure 7: Final result

The robot is able to move in the desired direction and with desired speed, however there is no simple relation between position of the joystick and linear velocity of the robot, so it was not possible to design deterministic trajectory control, especially without feedback mechanism which is not present in our construction. However in real construction, like f.e. cars, we have also nonlinear relation between accelerator and speed of the car. In case of nonlinear friction between dissimilar materials it is not possible to design the control for every type of the ground, hence we get wide range of different behaviours.

Probably the most challenging part of the project was mechanical construction. The size of our construction makes it impossible to find suitable parts such as joints and rods. It was hard to make construction stiff and stable enough. The most critical task is to attach hemisphere to drive-shaft coaxially and rigidly. Fortunately, it was realized with good effect.

Analysis of energy efficiency could be interesting area of research for engineers who are looking for alternative drive systems. It seems even more attractive, because of very small amount of articles and materials discussing this topic.

References

[1] Hemisphere Drive Speedster. `http://blog.modernmechanix.com/hemisphere-drive-speedster/`. Access: 13.02.2017.

[2] Model of mobile robot with HOG wheel. `http://konar.usermd.net/wordpress/wp-content/uploads/2016/11/hog.pdf`. Access: 13.02.2017.

[3] E. Ackermann. You've Never Seen a Robot Drive System Like This Before. `http://spectrum.ieee.org/automaton/robotics/diy/youve-never-seen-a-drive-system-like-this-before`. Access: 13.02.2017.

[4] HOG Drive. `https://www.thingiverse.com/thing:52097`. Access: 13.02.2017.