Final report
Controller for Hogger^2 HOG wheel robot

Eryk Możdżeń

January 27, 2025

1 Intro and goals

Hogger^2 is a mobile robot with two HOG (Hemispherical Omnidirectional Gimbaled) wheels belonging to KCiR. In the past, many diploma theses have been written on modeling and controlling a robot with such a drive [9], but none of them have tested them on a real robot, among other things due to the lack of efficient hardware.

The outcome of the project will be a robot with implemented simple algorithm for movement without slippage on the flat ground. The robot should be controlled wirelessly from PC.

2 Completed tasks

2.1 Hardware controller design

Hardware design of version 1.0 (first one) of the controller board was developed using KiCad [2]. Hardware should fulfill requirements shown on picture 1. The most important features are:

- two 3-phase inverters capable of controlling BLDC motors with sensorless method
- four PWM signals with appropriate voltages that allows controlling standard servos
- on–board MEMS accelerometer and gyroscope and support for external optical sensor
- wireless communication capabilities

Comparison between design and final working circuit and custom casing is available in picture 2.

After manual assembly, the MOSFET transistors driving the BLDC coils were found to burn out constantly. In order to fix this issue I added additional resistors, visible in picture 2b. Of course, some traces also need small fixes due to not using ERC (Electrical Rule Checker) during the development phase. From this reason some small wire is visible in picture 2d.

As on–board MEMS IMU chip MPU6050 was selected due to simplicity, cost and proven reliability in previous robotics projects. It consist of 3–axis accelerometer and 3–axis gyroscope that will be very helpful for estimating pose of the robot on the ground in near future. Additionally external optical sensor PMW3901 was added to obtain direct measurements of velocity of the robot in X and Y direction.

In previous thesis in field of mobile robots [10] network interface were used to communicate with the robot. As a continuation of this idea I decided to use ESP32C3 microcontroller as a slave device connected to the main unit. To handle potential electromagnetic interference from MOSFETs and motors, a backup radio module (nRF24L01+) was included. The fears turned out to be unfounded, the ESP32 RF interface performed very well even near working motors. It turns out that most problematic are servos, which generates current spikes capable of reset the ESP. Solution was to solder additional bulk capacitors (5x 100uF) "in the air" to power pads of the ESP32, also visible in picture 2b.

Enclosure was designed using claud CAD platform Onshape [12] and then 3D–printed from PLA. Mounting holes ware design to match existing mount points on the robot's frame.

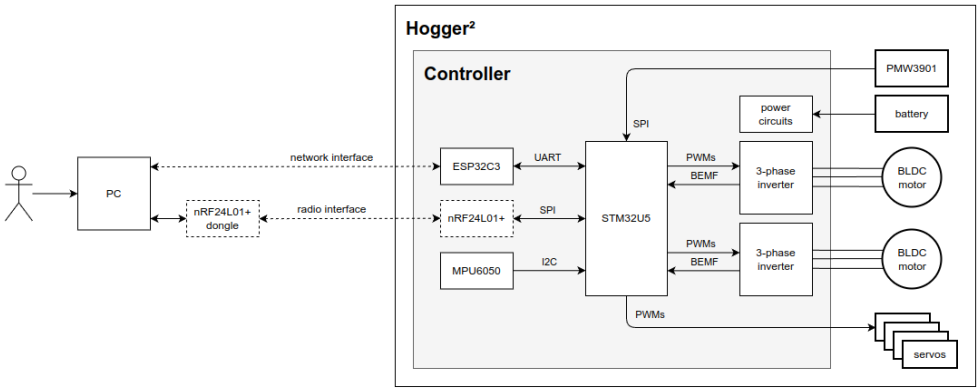


Figure 1: Electronic architecture

2.2 BLDC motor velocity control

In the task of controlling HOG–wheeled mobile robot wheels must spin at relatively high speeds. From this reason sensorless method was chosen to control BLDC motor without need of additional sensors mounted on each motor, like encoders or hall sensors. In short, the six step algorithm consists of 6 different motor commutations (one phase is connected to ground, one to supply voltage and is floating) are driven one after another in specific order. In each commutation floating phase is inducing voltage due to back electromotive force (BEMF) while rotor spin. Shape of this voltage should be similar to trapezoidal wave, while zero–cross point (ZC) should be in the middle of each commutatation to obtain maximum torque (using this method). To obtain stable angular velocity of the rotor, PI controller control PWM duty cycle (related with motor torque) of powered phases in each commutation. In order to detect ZC event external hardware comparators were used. Side effect of this algorithm is knowledge about rotor angular velocity that can be used in sensor fusion as odometry input in the future.

To obtain sufficient BEMF amplitude, startup procedure is needed to ramp up motor velocity (open–loop) at the begin. After that smooth transition between open–loop and closed–loop was needed.

Results of voltage phases as well as comparator output is shown on picture 3. In the middle of each t_{ON} of PWM signal comparator is sampled. To reject jitter near ZC, software filter was implemented. During tests presented algorithm proofs that can drive BL2215/25 motor (without load) up to theoretical maximum ($950\ \text{KV} \cdot 12.4\ \text{V} \approx 1232\ \frac{rad}{s}$), so it can be considered that algorithm is implemented well.

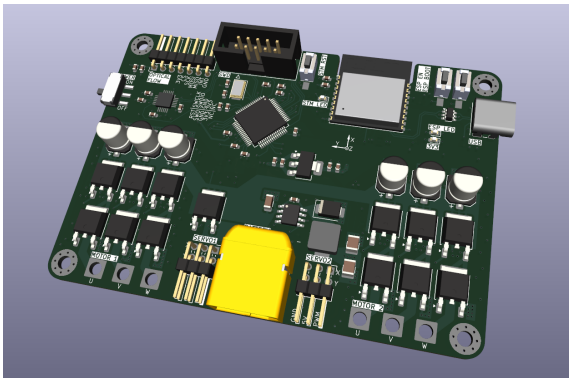
Test on the ground using two PropDrive 28–30 (750 KV) motors showed, that robot is able to spin the motors under load sufficient for movement. After tilting hemisphere too much, rotor stops in place and startup procedure need to be fired once again. In order to improve performance and remove high frequency noise, audible for humans, PWM frequency was setup to 50kHz.

2.3 Servo control

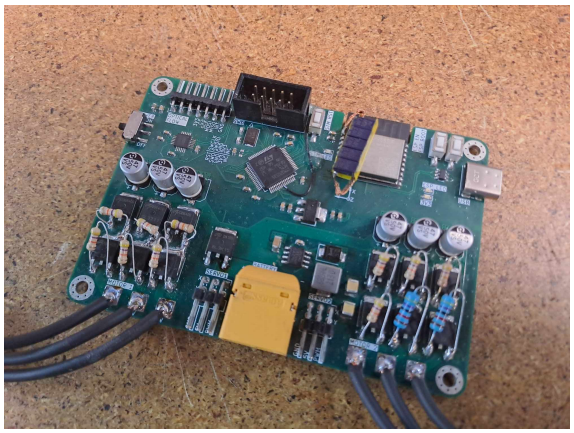
Currently, robot's frame uses standard old avionic servos that are powered directly from 5V. Control comes directly from the microcontroller in form of PWM signal, with frequency 50Hz and duty cycle from 1ms (-90°) to 2ms (+90°). Each of servos have its own offset due to not perfect gimbal mechanism, so additional calibration on station app site was needed. Future iterations may use servos with reduced gearbox backlash and feedback capabilities.

2.4 Reading on–board sensors

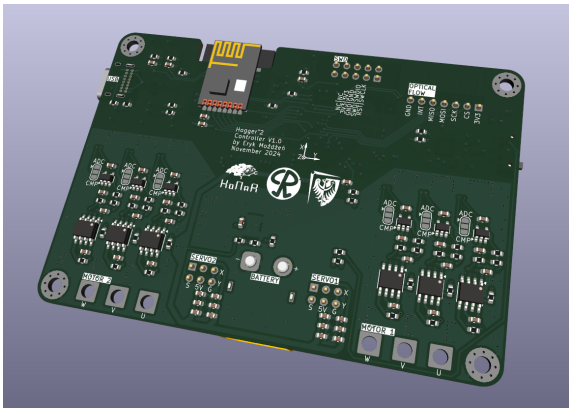
Presented version of the controller contain IMU chip MPU6050 [8] (integrated 3–axis accelerometer and 3–axis gyroscope) and support for external optical flow sensor PMW3901 [7]. Scope of this project focus on



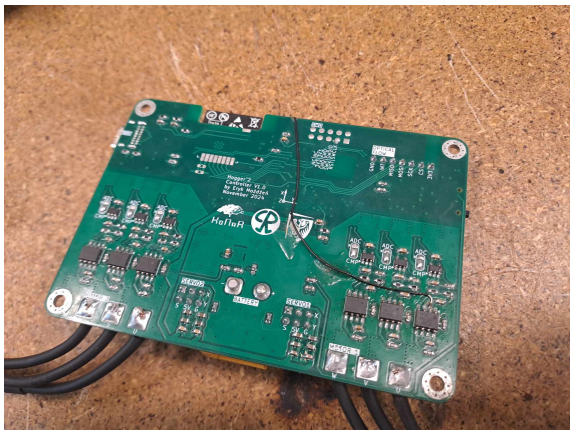
(a) front render



(b) front physical

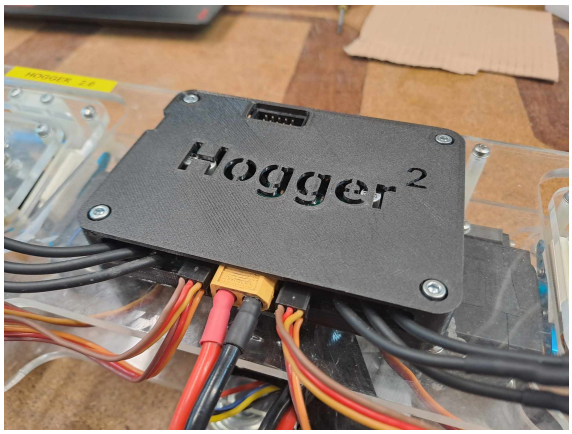


(c) back render



(d) back physical

(e) enclosure render



(f) enclosure physical

Figure 2: Comparison between design and physical realization

Figure 3: Motor phase voltage (yellow) and comparator output (blue) during closed–loop operation

reading measurements from those sensors continuously and converting them to physical units. For IMU I^2C peripheral was used with DMA (Direct Memory Access) to offload microcontroller core, interrupt configure as "data ready" trigger sensor readout on around 1kHz. For optic feedback SPI interface is periodically samples on 50Hz. Data from both sensors are converted from bytes to floating point values in SI units. In the future probably additional magnetometer chip will be places on the board to improve sensor fusion results.

2.5 Wireless communication

Selected host microcontroller STM32U5 [11] does not support wireless communication. Originally, there was two ides to resolve this issue. First of them was placing ESP32C3 [3] as a wireless network interface, this should allow simple communication to PC station app without additional hardware on PC side. In case of large influence of noise due to inverters, there was plan B: radio chip nRF24L01+ [13] connected to host microcontroller via SPI. In this setup PC should also have some USB–radio dongle with custom reception algorithm.

The ESP32 firmware implements serial communication with host STM32 via UART. Messages are formed in MessagePack [4] standard using open–source C implementation [1]. ESP32 converts received MessagePack message and convert it to JSON format using cJSON [5] library. All the time ESP32 is access point in which publish on broadcast address those JSON's using UDP protocol. After receiving any JSON from station app (also through UDP) convert it back to MessagePack and send to the host.

Watchdog mechanism are added to improve safety of the communication system. If host microcontroller is not transmitting any valid MessagePack's for more than 1 second, it broadcast JSON with error code. If station app is not transmitting and valid JSON's for more than 1 second it send message including stop command to the host.

Due to the satisfactory level of performance of ESP32 radio module was not even developed at all. In the future, radio module will be probably removed from controller board design.

2.6 Station app

The network interface for communicating with the robot and the form of messages have been designed to be easy to implement in any programming language. Simple Python script can be used to receive and

```python
import socket

IP = '0.0.0.0'
PORT = 4444

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP, PORT))

while True:
    data, _ = sock.recvfrom(1024)

    print(data.decode())

```

(a) Receive robot state

```python
import socket
import json
import time

IP = '192.168.4.1'
PORT = 3333

with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as sock:
    while True:
        data = {
            'command': 'manual',
            'ref_cfg': [
                1500,
                1500,
                500,
                1500,
                1500,
                0,
            ],
        }
        sock.sendto(bytes(json.dumps(data), encoding='utf-8'), (IP, PORT))

        time.sleep(0.05)

```

(b) Send desired robot configuration

Figure 4: Example Python communication scripts

visualize state of the robot (ASCII characters in form of JSON through UDP) and for sending control values (picture 4). However, for USB joystick it was simpler to use C++ GUI app written using Qt [6].

3 Summary

Goals		Realization
Must do	custom circuit for BLDC control and sensors	fully
	BEMF control algorithm on both motors	fully
	controlling of servos	fully
	reading IMU sensor	fully
	2-way communication using ESP32	fully
	robot movement in open-loop	fully
Hope to do	reading optical flow sensor	fully
	2-way communication using nRF24L01+	not needed
	EKF for state estimation	none
	robot movement in close-loop (using on-board sensors)	none

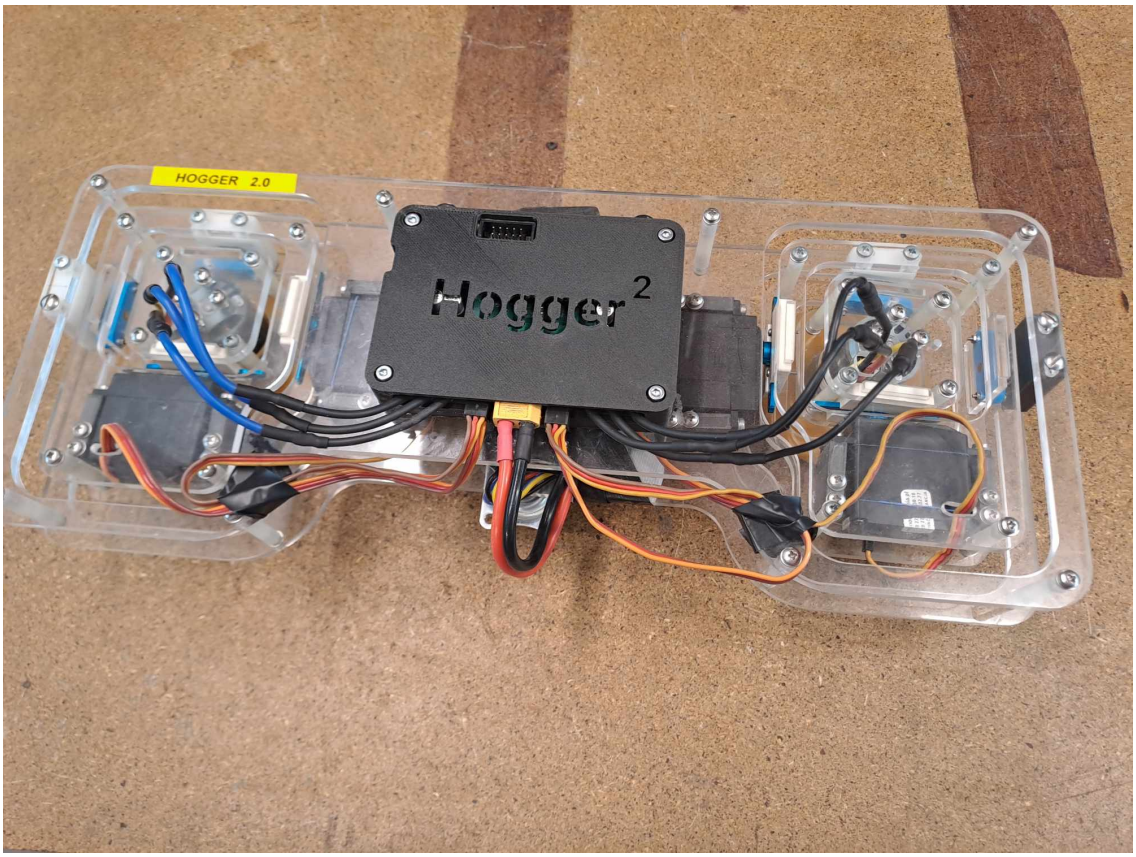


Figure 5: Controller mounted on the robot frame

4 Conclusions

Main goals of the project have been achieved. Robot is able to startup motors and start moving around on the flat ground. On–board sensors are working continuously on maximum possible frequency. Current robot state can be received using wireless network as well as reference configuration for the robot. Developed hardware allow to determine pros and cons of the originally proposed solutions and serve as a good starting point for the final controller board.

References

[1] camgunz. An implementation of the MessagePack serialization format in C. https://github.com/camgunz/cmp.

[2] J. P. Charras. KiCad. https://www.kicad.org/.

[3] Espressif. ESP32-C3 Series. https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf.

[4] S. Furuhashi. Message Pack It's like JSON. but fast and small. https://msgpack.org/index.html.

[5] Gamble, D. Ultralightweight JSON parser in ANSI C. https://github.com/DaveGamble/cJSON.

[6] Qt Group. Qt. https://www.qt.io/.

[7] PixArt Imaging Inc. PMW3901 datasheet. https://download.kamami.pl/p587092-pmw3901mb-txqt_-_productbrief_2451186_7.pdf.

[8] InvenSense. MPU6050 datasheet. https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf.

[9] P. Joniak. Control problem for two HOG wheel mobile robot. Master's thesis, https://kcir.pwr.edu.pl/~mucha/Pracki/Pawel_Joniak_praca_magisterska.pdf, Wrocław University of Science and Technology, 2017.

[10] T. Lubelski. Construction of small mobile laboratory robot of class (1,2). Bachelor's thesis, https://kcir.pwr.edu.pl/~mucha/Pracki/Tomek_Lubelski_praca_inzynierska.pdf, Wrocław University of Science and Technology, 2023.

[11] ST Microelectronics. STM32U535xx datasheet. https://www.st.com/resource/en/datasheet/stm32u535cb.pdf.

[12] PTC. Onshape. https://www.onshape.com/en/.

[13] Nordic Semiconductors. nRF24L01+ Single Chip 2.4GHz Transceiver Product Specification v1.0. https://cdn.sparkfun.com/assets/3/d/8/5/1/nRF24L01P_Product_Specification_1_0.pdf.