IMPLEMENTATION OF A HOME APPLIANCE MOBILE PLATFORM BASED ON COMPUTER VISION: SYSTEM CONFIGURATION AND CALIBRATION

Florin-Dan SECUIANU[1], Ciprian LUPU[2]

This paper is a continuation of our previous studies on the implementation of a mobile platform based on computer vision and Raspberry Pi hardware, precise indoor navigation. Thus, we implemented a prototype version of a robot that can recognize targets using machine learning and computer vision based algorithms. Further, the platform was improved to achieve complete autonomy, indoor mapping and navigation, and self-charging. We showed that it is possible to create an affordable, completely autonomous robot that can navigate and detect targets. The present study has two main objectives. The first one is to build a platform that uses computer vision algorithms to detect objects, processes and aggregates the information acquired from various sensors, shares it with other devices, and has capabilities of remote and automated reconfiguration of the main program loop and logging results. The second objective is to improve the precision of the localization and navigation by adjusting the data received from the digital compass to the specifics of the environment and the hardware setup of the mobile unit.

Keywords: Raspbian, Raspberry Pi, Python, Wi-Fi, debugging, self-charging, digital compass calibration

1. Introduction

Lately, the interest in obtaining fully autonomous robots able to map, localize targets, self-charge with energy and perform various tasks indoors has increased. The use of autonomous and highly cooperative robots that are able to perform predefined tasks without human supervision is going to transform the future of almost all activities such as agriculture, mining, services and so on.

Such applications include search and find systems indoors, assisting humans with various tasks [1], inspecting areas and reading meters [2], delivery of products inside cafes, hotels, and even fully automated outdoor robots [3-8]. Other applications include collaboration between heterogeneous robots to perform coordinated search and rescue missions over a given area [9]. Many applications use Digital Magnetic Compass (DMC) as part of the Simultaneous Localization and Mapping (SLAM) functionality of the systems [10-14].

[1] PhD student, Dept. of Automatic Control and Systems Engineering, University POLITEHNICA of Bucharest, Romania, e-mail: dan.secuianu@gmail.com
[2] Prof., Dept. of Automatic Control and Systems Engineering, University POLITEHNICA of Bucharest, Romania, e-mail: ciprian.lupu@acse.pub.ro

Although there are numerous papers in the literature treating different topics on robots, our careful literature search revealed that currently our platform is the only one combining Raspberry Pi, digital compass, video camera, and ultrasonic sensors to map and navigate indoors using custom built navigation algorithms, as well as self-recharging using wireless charging technology. In addition, it should be stressed that not only the hardware configuration is original, being designed and assembled using various hardware components, but also the software procedures that achieve the described functions are custom built, and the machine learning models for object recognition were trained in-house. This paper is a continuation of our previous studies [15,16] where we presented the implementation of an autonomous robot that uses computer vision software and affordable Internet of Things (IoT) Raspberry Pi hardware and sensors to map an indoor area and navigate, as well as implementing energetic autonomy via self-charging configuration and software. The first version of the platform was built on a simple 3 wheels robot, based on a Raspberry Pi mini-computer. The software program that was written using the Python language was able to recognize specific targets (signs) using our own trained "Haar feature based cascade classifiers" and the functions provided by the OpenCV computer vision library (see Fig. 1). Thanks to one ultrasonic sensor, the robot was also able to detect physical boundaries and navigate pseudo-randomly in the environment, moving between several targets [15]. The prototype proved that it was possible to build an affordable robot with a lot of processing power that can use the recent advances in terms of hardware and software to intelligently navigate without much human supervision.

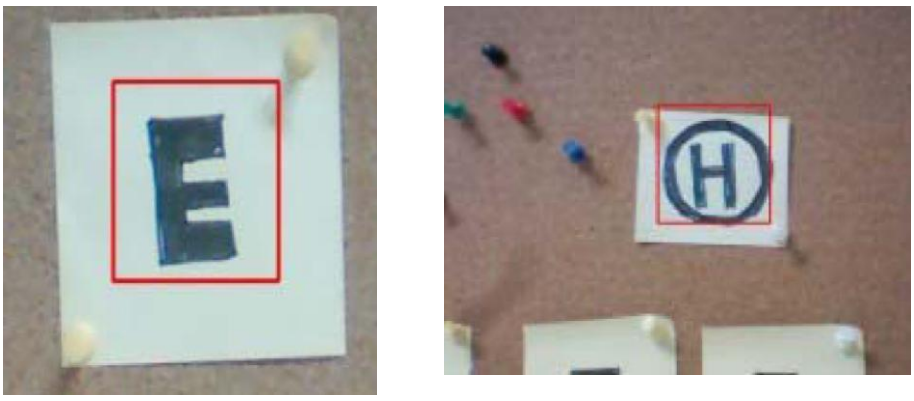

Fig. 1. Object detected using Haar Classifier [15]

The second version of the platform that was built [16] brought new software modules that were written to create, store, and update the map of the environment and to calculate the optimal paths between locations. Some other new software modules were implemented to perform the self-recharging

procedure. The platform built at this stage provides all the necessary functions required for complete autonomy: self-recharging with energy (wirelessly), improved navigation capabilities that combine the information provided by multiple sensors (see the evolution of the mobile robot platforms, Fig. 2).

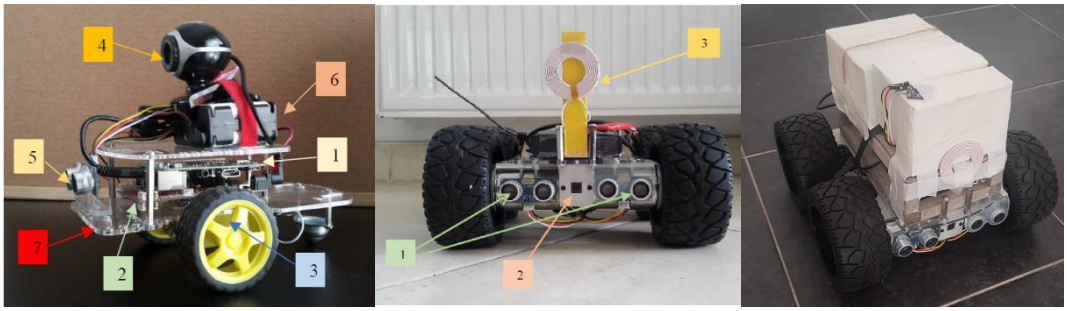

Fig. 2. Mobile robot versions. From left to right: 1 [14], 2 [15], and 3 [last version].

The software written makes use of the data from the camera and computer vision to detect targets, data from ultrasonic sensors to detect the boundaries of the environment, and the data from the newly added digital compass to save, load, and update the map (see Fig. 3) and to achieve precise localization. The objective was to build an affordable, mobile platform that is fully-autonomous that can operate and perform various tasks in an indoor environment.

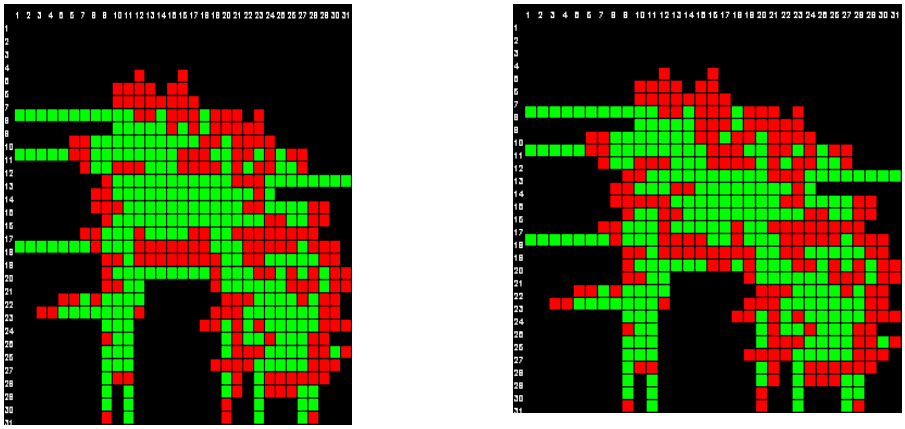

Fig. 3. Example of maps: initial (left side) and updated scan (right side)

The cost of the hardware for both versions combined was around EUR 400-500. The software algorithms were designed and implemented by the authors, and used available platforms and open-source libraries: Linux OS, Python, OpenCV. The platform does not carry maintenance costs, moreover constant upgrades of the connected software modules are available so that the platform can make use of performance improvements or bug fixes. Thanks to the modular architecture of the software that we implemented, the platform can be easily

adapted to perform as designed and described using other hardware components: different chassis, control boards, sensors, digital compass.

Here we present the results of our exhaustive study on building the complete platform of the autonomous robot, including the software configuration and the operating system, communication and control of the robot via Wi-Fi from the software developer point of view, configuration of the camera and computer vision software, the software tools and modules, information on the hardware used such as the extension board, sensors and motion, the self-charging feature, the calibration and use of the digital compass. We studied the calibration of the digital compass to the specific conditions of the testing environment and we propose an original solution for accurate positioning and movement irrespective of the observed distortion factors.

The calibration of the compass and the proposed algorithms resulted in a higher precision of pose and navigation. We studied and implemented a procedure that can adjust the readings of the digital compass to be as close as possible to the real orientation. Most importantly, our original method, which aggregates data from multiple sources, digital compass, video camera, ultrasonic distance sensors, is efficient in correcting the position on map and orientation of the mobile robot. Secondly, configuring the platform to allow remote transfer of executable code and data sharing at any time opens infinite possibilities, such as adding new software sub-routines on-the-fly, adding the capability to detect new objects via machine-learning algorithms trained on live images transmitted by the mobile robot, updating the live map with information added automatically by other devices or even human operators.

2. Software configuration

2.1. Operating system

The software programs that enable the functionality of our robot on the Raspberry Pi (R-Pi) board use high-level programming languages. The programs need the functionalities provided by an operating system that include access to hardware resources such as storage, memory, and CPU, and access via communication ports/interfaces to hardware components that can be attached to the board: video cameras, analog and digital sensors, Wi-Fi and Bluetooth enabled devices.We used the documentation provided by the manufacturer for the “Raspberry Pi 3 Model B Rev 1.2” board and installed "Raspbian GNU/Linux 8 (Jessie) 4.9.35-v7+" operating system, the minimum version to support R-Pi 3 [17]. We used a laptop with Windows OS and SD card writer capability to install the OS on a 6GB SD card via the R-Pi imager utility also provided by the manufacturer. Once the card was inserted in the R-Pi, we attached it to a TV via a HDMI cable, to a router via an Ethernet cable, to wireless keyboard and mouse

via a wireless receiver connected to the one of the USB ports and powered up the R-PI by connecting a USB power supply.

2.2. Wireless connection and control of the R-Pi

The next steps included changing the default password for user 'pi' and configuring access to Wi-Fi networks, so that the board can connect to the local Wi-Fi and other wireless access points (AP) that we used later on, e.g. AP enabled on mobile devices (smartphones) to allow wireless connections between our R-Pi and the development devices (laptop) without the need of a standard wireless router.

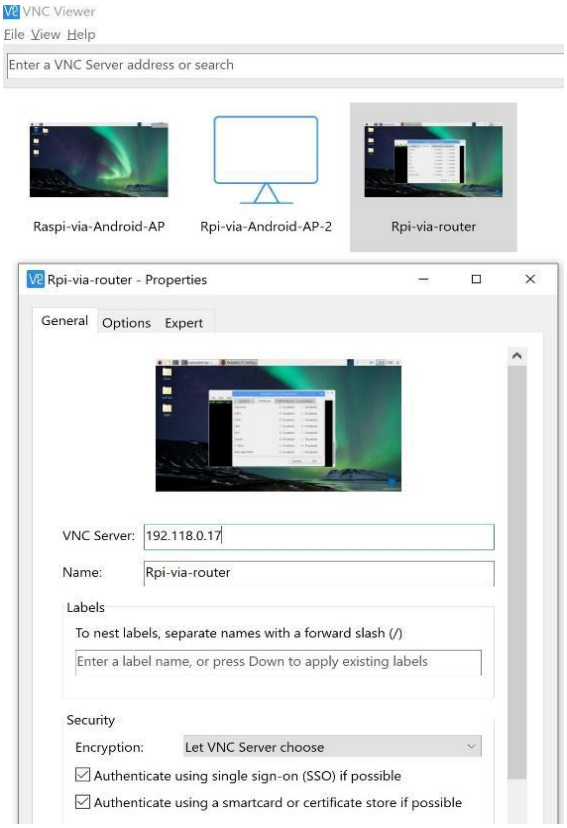

Fig. 4. VNC viewer connection between laptop and R-Pi

In order to be able to connect wirelessly to our R-Pi from a laptop, we enabled Secure Shell (SSH) and Virtual Network computing (VNC) interfaces via the R-Pi configuration utility and set-up a specific value for the private IP address the device requests when connecting to Wi-Fi networks. We used this documentation to configure the remote access via VNC [18].

The next step was to install a VNC viewer software on our development laptop, connect to R-Pi and continue with the next steps without the need of the monitor/TV, keyboard or mouse attached to it (Fig. 4).

Securely transferring files between computers can be done with the ‘scp’ command in terminal, e.g.

‘scp<path1>/<filename1> pi@<ip_address_rpi>:/home/pi/<path2>/<filename2>’.

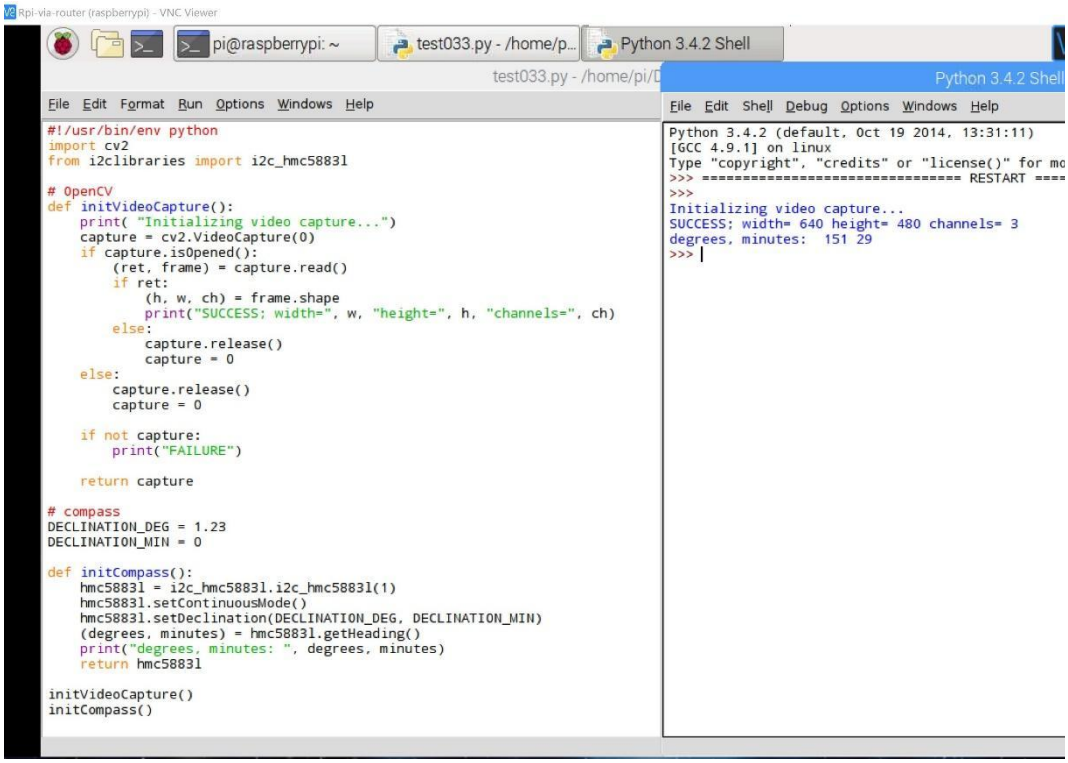

Fig. 5. Python 3 software development framework

2.3. Connecting the camera and installing the computer vision software

One of the main strengths of our robot is the use of computer vision for detection of objects. We use an original R-Pi camera module V2. There is an option in the R-Pi configuration panel to enable the module. Note that image can be also acquired by attaching to R-Pi any type of camera via the USB interface.

2.4. Software tools and modules

The choice for high-level programming languages was Python. This is an interpreted, general purpose language, very popular in the scientific community. The decision was based on the fact that we could easily use open-source software libraries for reading data from sensors and for running computer vision algorithms. We used Python 2.7 in the early phase of development and then upgraded to version 3.7.2. Writing, running and troubleshooting our own software package was straightforward (Fig. 5). Data from sensors is read via open source Python libraries.

3. Hardware configuration

3.1. Extension board

The robot that we built was created by combining hardware parts from various manufacturers, some open source software (OS - Raspbian, computer vision libraries, libraries for serial port access), and our in-house software code that controls the robot and enables the required functionalities.

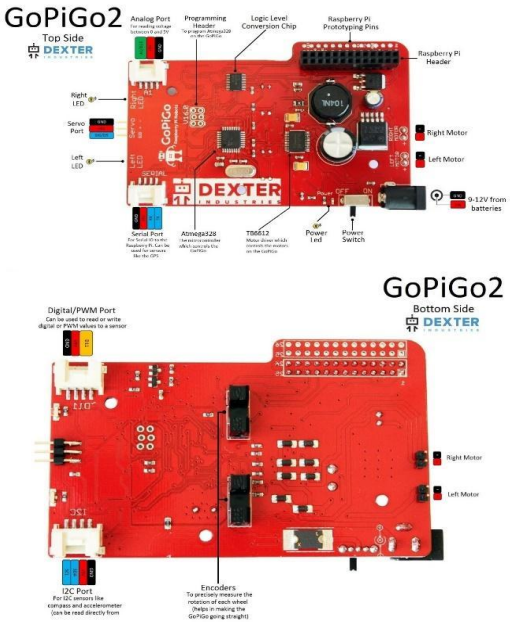

Fig. 6. Hardware configuration of GPG-2 board [19]

The R-Pi on our robot does not allow by itself to transfer power to electrical motors. This is done with the help of an extension board connected to R-Pi via the serial port. The extension board we used is GoPiGo 2 (GPG-2) and is produced by Dexter Industries. The board is based on an ATMEGA328 microcontroller which handles the communication between the board and the R-Pi. [19], as can be seen in Fig. 6. The documentation provided by the manufacturer contains information about the hardware and high-level programming code language e.g. Python to help communication with the ports.

3.2 Sensors

The GPG-2 board contains one Inter-Integrated Circuit (I2C) port, which we used to connect a digital compass, one digital, and one analog port to which we attached two ultrasonic distance measuring sensors. The manufacturer provided the scripts necessary to install all the necessary dependencies for the libraries provided, as well as tests for troubleshooting and debugging. Using the

software modules with Python is straightforward by importing them into our software.

3.3 Motion

Controlling the electrical motors is possible via two power outputs of the extension board.

Fig. 7. Robot chassis by PiBorg [20]

The motors and the chassis of the robot were provided by another manufacturer, PiBorg. The model used is a rugged, aluminum based, with the most powerful motor version available at the time [20], as can be seen in Fig. 7. We used only the chassis, the four electric motors and the custom location to insert the R-Pi video camera.

3.4. Self-recharging feature

To solve the problem of energetic autonomy, we took the approach of wireless charging. We used a rechargeable battery pack (RBP) manufactured by Romoss, with one input (mini-USB) and two outputs (USB). The input is connected to a wireless power receiver. When the receiver coil on the robot is connected or in the proximity of the transmitter coil placed on the fixed base charging station, the battery pack is charging. One of the outputs is connected to the R-Pi board via a USB to mini USB cable, and the other output is connected to the GPG-2 board via a DC booster cable (USB to 2.1mm). The cable contains an integrated boost converter from 5V to 12V, thus providing constant voltage to the board that controls the electrical motors. When the RBP's input is connected to an energy source, because of the internal switch, there is a short drop in power output to R-Pi. Due to the fact that R-Pi is sensible to voltage fluctuation, this will cause a restart of the operating system. In order to resume the operation, the script that contains the main loop of the program executed by our robot is added as one of the programs to be run automatically by the OS on restart, using crontab and @reboot. The main program script saves the current state and time in a file. On start, data is loaded from the file so that the program can switch from 'locking to charger' state to 'charging'. Since we use a booster to provide constant voltage to the GPG-2 board and there is no information about the current charging percentage of the battery, a constant of 8 hours is used to decide when the robot is

fully charged and can resume other activities. The electrical scheme of the power flow is represented in Fig. 8.

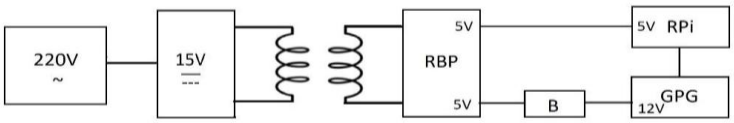

Fig. 8. Electrical scheme of the power flow

3.5. Digital compass calibration.

The digital compass we used is model HM5883L.

In the following section we describe the calibration process, the data and the results. The values of the magnetic field on the three axes *x*, *y*, and *z* can be read from the registers of the digital compass, via the serial bus. The heading is calculated using the values for *x* and *y* axes, and a correction is applied to take into account the local declination where the measurement takes place. The source code for calculating the heading is:

```
headingRad = math.atan2(value_y, value_x)
headingRad += self.declination
if (headingRad < 0):
        headingRad += 2*math.pi
if (headingRad > 2*math.pi):
        headingRad -= 2*math.pi
```

Consequently we created an experiment and the robot was programmed to measure and store the calculated heading in steps of 5 degrees, between 0 and 360, in triplicates and also stored the corresponding measurements taken with a compass.

Table 1

Sample of calculated heading – *C(x)*, real heading – *R(x)*, and difference (error) – *D(x)*

***C(x)* [deg]**	***R(x)* [deg]**	***D(x)* [deg]**
70	60	10
73	65	8
76	70	6
80	75	5
83	80	3
87	85	2
90	90	0
94	95	-1
96	100	-4
100	105	-5
103	110	-7
105	115	-10
109	120	-11
112	125	-13

We were particularly interested in orthogonal movements in the '*xy*' plane so that the robot keeps its heading precisely on two orthogonal axes and their four directions: front, back, left, and right. For each entry in the table (*C(x)* – the calculated value of the angle for every *x* degrees; *R(x)* – the real value corresponding to calculated *C(x)*; *D(x)* – the difference between *C(x)* and *R(x)*), we calculated the deviation from the real values and observed the sum of deviation squares in steps of 90 degrees:

$$Dev = \sqrt{D(0)^2 + D(90)^2 + D(180)^2 + D(270)^2} \quad (1)$$

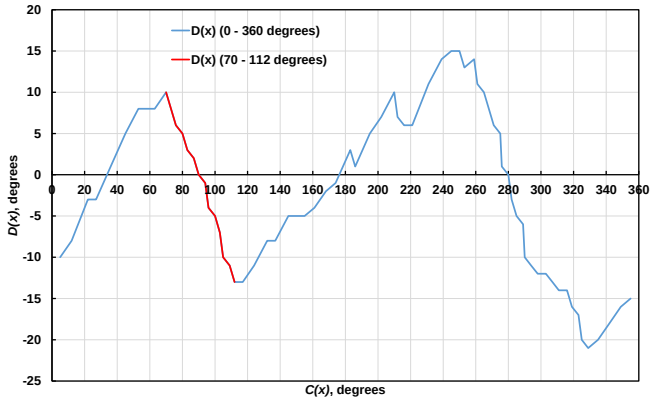

Fig. 9. The error between the calculated and real value of the heading

Sample of calculated data (72 data measurements repeated three times) are presented in Table 1 and plotted in Fig. 9 (red line). We then placed the robot on the initial starting position, which is the origin of the map, aligned it perpendicular to the base, and rotated the digital compass with 5 degrees, so that *Dev* will be kept to the minimum value while the robot is moving along the orthogonal directions. This allows the robot to measure correctly its orientation in the four interesting directions (Fig. 10).

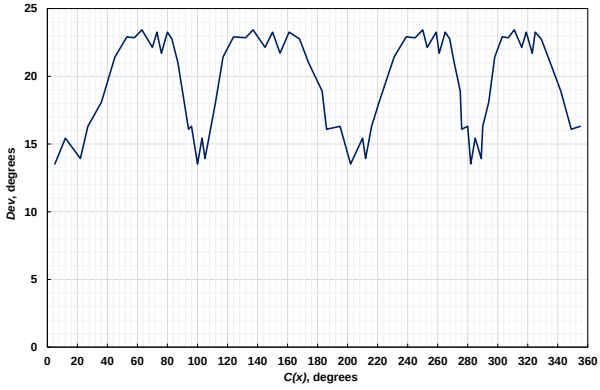

Fig. 10. *Dev* function

We observed there are multiple positions where the deviation calculated above is minimum, at 5, 100, 202, and 282 degrees. The software program that we wrote to enable precise positioning and navigation contains a procedure that rotates the robot until a desired heading is reached. Because of the time difference between the moment when the robot reads the heading and the moment it actually stops rotating, there is an inherent difference between the desired heading and the real one. In order to minimize this error of alignment, we control the rotation speed based on the difference between the current and the desired heading. When the difference is higher than a threshold, the robot rotates at maximum speed. When the difference is lower than the threshold, we decrease the speed so that the program can stop the rotation as close as possible to the desired heading. In order to have a more precise measurement at any given angle, we measured and eliminated the soft-iron errors.

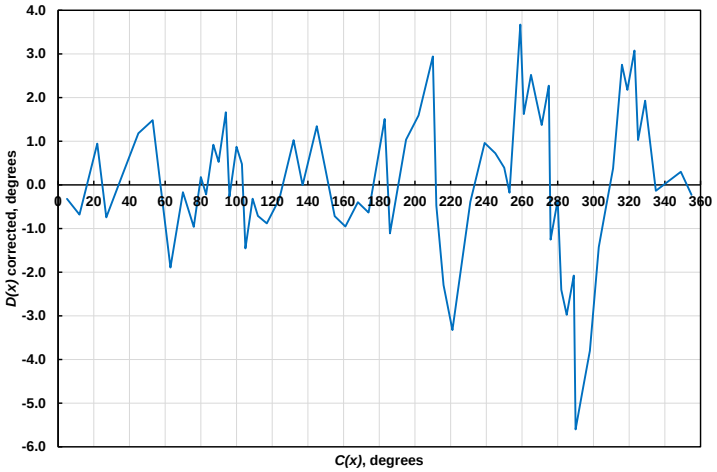

Fig. 11. Absolute error deviation after soft-iron correction

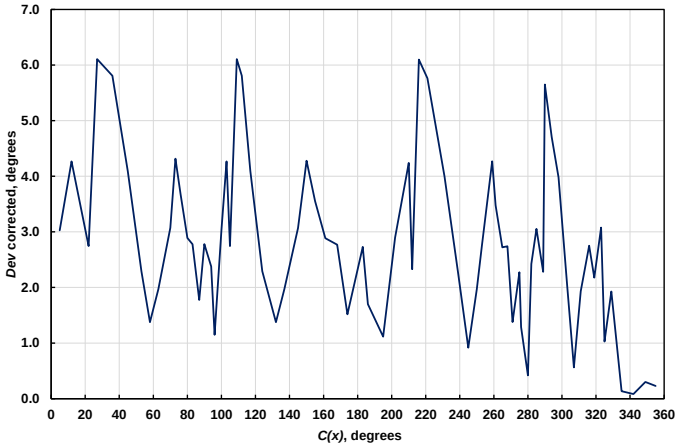

Fig. 12. *Dev* function after soft-iron correction

As described in [10], the soft iron distortion arises from the interaction of the earth's magnetic field and any magnetically soft material surrounding the compass. We applied a correction to the calculated heading and reduced the heading errors, as seen in Figs. 11 and 12 above.

***Algorithm of automatic correction of digital compass data*.** We studied and implemented a solution to calibrate the digital compass of the robot in order to minimize the positioning errors. The algorithm uses data from multiple sources: the digital compass, the video camera, and the ultrasonic sensor. Here we describe the procedure that runs when the robot is positioned at the charging base, or in any location that is labelled with one of the known images that the computer vision subroutine can recognize. The digital compass is attached in a fixed position on the top side of the robot, at the height empirically determined to reduce the electromagnetic influence of the battery pack, motors and chassis. The robot is rotated in small steps, with consecutive commands, around his vertical axis, and the number of rotation commands is counted. Since the friction coefficient between the surface and the robot's wheels is not known, and in this step the algorithm does not control the value of the rotation angle in one rotation step, this is a value that can vary and has to be determined. With the help of the subroutine that is also used for determining if a target image is in the center of the snapshot acquired from the video camera, we can identify the approximate moment when the robot has performed a complete 360 rotation. In step 1, the algorithm stores the number of rotation commands per 360 degrees and repeats the cycle a few times, e.g. 72 steps will result in a 5 degree rotation per step. In step two, the robot performs several rotation cycles, storing the data from the digital compass, mapping the rotation angle and the data from the digital compass. In step three, the read data is converted accordingly by adding or subtracting the offset, so that the direction of the base becomes the new 'North' and data for all the other angles will be adjusted accordingly to match the real orientation of the robot. As described earlier, the robot will navigate in the environment using 90 degree rotations, straight forward and backward movements so we need these to be precise. In step four of the calibration, the robot performs a few cycles of 90 degree rotations, analyzing the position of the target with respect to the center of the image. If it is not centered, the calibration loop returns to step 1 to acquire new data, and saves the average values. The calibration process ends when the robot can perfectly align with the 'North' after a few 360 rotation cycles.

4. Conclusions

This paper presents how we achieved full autonomy of a mobile robot platform using affordable materials and in-house built software. A detailed description of the software and hardware modules of the proposed and implemented mobile platform was provided. It proposes solutions to completely

configure and build an autonomous mobile robot. A novel procedure for very precise indoor positioning and navigating was designed and implemented, using the DMC technology. The platform advanced from the stage where the robot was able to move and locate targets using a pseudo-random search algorithm, to the current stage where the platform has the memory of the previously acquired sensor data stored as a map, and is able to calculate optimal paths for moving around between objectives. One of the key factors is the calibration of the digital compass to reduce the reading errors for any direction, so the precision of positioning and navigation is dramatically improved. This opens the possibility of various applications such as substitution of humans in hazardous environments, active monitoring and surveillance of indoor and outdoor spaces, guidance of humans in offices, museums or storage areas, delivery of goods, vacuum cleaning, lawn mowing, farming. Due to the versatility of the platform supported by high processing power and advanced connectivity, the main program loop can be upgraded on the fly to include artificial intelligence based functionalities such as scene, object, and face recognition. It is also independent of the hardware used for navigation, so it can be easily transferred to bigger and more powerful hardware.

Future research lines will be along description of the current navigation algorithms that were implemented, analyzing and improving the navigation in terms of complexity, execution speed and scalability, 'live' collaboration between multiple devices in achieving fast mapping and optimal navigation solutions, calculating the paths for devices in a swarm of e.g. vacuum cleaners in order to cover every 'square' of a shared map in the most optimal way. We will study and implement a novel precise and orthogonal positioning on target waypoints by comparing multiple solutions. We will add new functionalities based on machine learning algorithms and artificial intelligence e.g. automatic labelling of scanned areas on the map, automatic generation of waypoints and the corresponding ML detection algorithms, and correction of position based on waypoints.

REFERENCES

[1]. *J. Mišeikis, P. Caroni, P. Duchamp, A. Gasser, R. Marko, N. Mišeikiene, F. Zwilling, C. de Castelbajac, L. Eicher, M. Früh, H. Früh*, "Lio-A Personal Robot Assistant for Human-Robot Interaction and Care Applications", IEEE Robot. Autom. Lett. 5, 2020, pp. 5339–5346.

[2]. *C. Wang, L. Yin, Q. Zhao, W. Wang, C. Li, B. Luo*, "An intelligent robot for indoor substation inspection", Ind. Robot: Int. J. Robot. Res. Appl. **47**, 2020, pp. 705–712.

[3]. *A. Barnawi*, "An Advanced Search and Find System (ASAFS) on IoT-Based Mobile Autonomous Unmanned Vehicle Testbed (MAUVE)", Arabian J. Sci. Eng. **45**, 2020, pp. 3273–3287.

[4]. *M. Shen, Y. Wang, Y. Jiang, H. Ji, B. Wang, Z. Huang*, "A New Positioning Method Based on Multiple Ultrasonic Sensors for Autonomous Mobile Robot", Sensors **20**, 17, 2020, pp. 1–15, doi: 10.3390/s20010017.

[5]. *Ł. Białek, J. Szklarski, M. M. Borkowska, M. Gnatowski*, "Reasoning with four-valued logic in multi-robotic search-and-rescue problem," in Challenges in Automation, Robotics and Measurement Techniques, **vol. 440**, no. 1. Cham, Switzerland: Springer, 2016, pp. 483–499.

[6]. *J. J. Acevedo, B. C. Arrue, I. Maza, A. Ollero*, "Cooperative large area surveillance with a team of aerial mobile robots for long endurance missions," J. Intell. Robotic Syst., **vol. 70**, nos. 1–4, 2013, pp. 329–345.

[7]. *M. A. Goodrich, B. S. Morse, D. Gerhardt, J. L. Cooper, M. Quigley, J. A. Adams, C. Humphrey*, "Supporting wilderness search and rescue using a camera-equipped mini UAV," J. Field Robot., vol. 25, no. 1, 2008, pp. 89–110.

[8]. *E. Zalama, J. G. Garcia-Bermejo, S. Marcos, S. Dominguez, R. Feliz, R. Pinillos, J. Lopez*, "Sacarino, a service robot in a hotel environment," in Proc. 1st Iberian Robot. Conf. (ROBOT), 2014, pp. 3–14.

[9]. *S. C. Mohamed, S. Rajaratnam, S. Tae Hong, G. Nejat*, "Person Finding: An Autonomous Robot Search Method for Finding Multiple Dynamic Users in Human-Centered Environments", IEEE Trans. Automat. Sci. Eng. **17** (1), 2020, pp. 433–449.

[10]. *M. J. Caruso*, "Applications of Magnetic Sensors for Low Cost Compass Systems", in Proceedings of the IEEE Position, Location and Navigation Symposium, San Diego, CA, USA, 13–16 March 2000, pp. 177–184.

[11]. *Y.-Y. Jung, D.-Y. Lim, Y.-J. Ryoo, Y.-H. Chang, J. Lee*, "Position Sensing System for Magnet Based Autonomous Vehicle and Robot Using 1-Dimensional Magnetic Field Sensor Array", in SICE-ICASE International Joint Conference 2006, Oct. 18-2 1, 2006 in Bexco, Busan, Korea, pp. 187–192.

[12]. *J. M. Yun, J.-P. Ko, J. M. Lee*, "An Inexpensive and Accurate Absolute Position Sensor for Driving Assistance", IEEE Trans. Instrum. Meas. **vol. 5**7, no. 4, April 2008, pp. 864–873.

[13]. *I. Boniolo, S. M. Savaresi, M. Prandini, G. Borghi, B. Garavelli, S. Bittanti*, "Performance analysis of a digital compass for the heading estimation in nautical application", in Proceedings of the 15th IFAC Symposium on System Identification Saint-Malo, France, July 6-8, 2009, pp. 1399–1404.

[14]. *B. Livada, S. Vujic, D. Radic, T. Unkaševic, Z. Banjac*, "Digital Magnetic Compass Integration with Stationary, Land-Based Electro-Optical Multi-Sensor Surveillance System", Sensors, **19**, 4331, 2019, pp. 1–18, doi:10.3390/s19194331.

[15]. *F. Secuianu, C. Mihai, A. Vulpe, C. Lupu*, "Implementation of an Autonomous Mobile Platform Based on Computer Vision", in 18th International Carpathian Control Conference (ICCC), 2017, pp. 246–251.

[16]. *F.-D. Secuianu, C. Lupu*, "Implementation of a home appliance mobile platform based on computer vision: self-charging and mapping", in 22nd International Conference on System Theory, Control and Computing (ICSTCC), 2018, pp. 464–468.

[17]. https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up, accessed on 2020-06-28.

[18]. https://www.raspberrypi.org/documentation/remote-access/vnc/, accessed on 2020-06-28.

[19]. https://www.dexterindustries.com/GoPiGo/learning/hardware-port-description/, accessed on 2020-06-28.

[20]. https://www.piborg.org/robots-1/monsterborg, accessed on 2020-06-28.