# Mobile Robot Odometry Using High Resolution Incremental Encoders and Single Board Computers

Vladimir Sibinović, Vladimir Mitić, Miloš Petković, Darko Todorović, Goran S. Đorđević

***Abstract*—Mobile robots are used in various applications, some of which have specific requirements, like precise localization. An odometry may be used for localization, or as a supporting system for improving accuracy of a global localization system. This often requires a lot of computing power and therefore is not suitable for embedded systems. Although there are a lot of small computers available, suitable for mobile robots, with enough processing power for ordinary odometry algorithms, rarely a suitable interface for reading a high pulse train signal of the high resolution incremental encoder is available. In this paper we will present one efficient solution, tested through experiments, for bridging Raspberry Pi 3 and magnetic incremental rotary encoders.**

***Index Terms*— Odometry, Encoders, RaspberryPi**

## I. Introduction

In recent years single-board computers (SBC) have become ubiquitous and are a preferred choice for many researchers. Currently there are a myriad of different SBC-s each with its strengths and weaknesses, and it can be sometimes difficult to choose the right one. The key features of a SBC are computational power, number of inputs and outputs, communication interfaces, cost and ease of use. Another important part to consider is the popularity of the SBC and its community.

For our project we have chosen the Raspberry Pi (RPi) as a platform for our mobile robot because of its low cost and computational power [1], also it is very popular and has a large community with a lot of documentation. The latest version of the RPi has also Wi-Fi communication, which is beneficial for wireless programing, control and data transfers.

RPi has a logic level of 3.3V which in some cases can cause some inconveniences, but can be solved using level shifters.

As for the programing of the RPi we can use different languages including Scratch, Python, and C++. That gives us an opportunity to develop complex programs and use the platform for different types of research.

Vladimir Sibinović is with the Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: vladimir.sibinovic@elfak.ni.ac.rs)

Vladimir Mitić is with the Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: vlada.m.mitic@gmail.com

Miloš Petković is with the Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: milos.petkovic@elfak.ni.ac.rs)

Darko Todorović is with the Faculty of Electronic Engineering, Universiy of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: darko.todorovic@elfak.ni.ac.rs)

Goran S. Đorđević is with the Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia (e-mail: goran.s.djordjevic@elfak.ni.ac.rs)

## II. Interfacing the Incremental Encoder

Our mobile robot is based around a differential drive system with Faulhaber motors shown on figure 1. These motors have a planetary gearhead with a reduction of 1:28, and a magnetic incremental encoder with 512 increments per revolution. Therefore, we had 14336 counts per revolution. At the maximum speed of 460 rpm which is 7.667 rev/s the number of counts would be around 110000 counts per second. This gives the signal frequency of around 220 kHz.

We tried connecting the encoder pins directly to the RPi, but the frequency of the signal is two large. It was possible to obtain some information from the readings, but the data was not reliable, and the processor engagement was significant. The solution to this problem we saw in two possibilities: to use a microcontroller for encoder reading, or to use a specialized IC for reading incremental encoders.

Fig. 1. Motor *Faulhaber 1516.A0210* with planetary gearhead and magnetic incremental encoder

We wanted to avoid using a microcontroller, because it requires software development for reading the signals and forwarding the information to the RPi. Ensuring that the data is correct would be challenging because of the large number of counts, so we decided to use a reliable solution. For

counting the encoder pulses we had at hand a specialized IC from Broadcom/Avago, model number *HCTL-2032*. This is a Quadrature Decoder IC that interfaces encoders to microprocessors. It has a 33 MHz clock, and two channels for reading encoders. Encoder increments are counted using 32-bit binary up/down counters.

Parallel reading from the IC can at one-point transfer one byte, and there are 2 pins for selection of which byte is being read. The counter is incremented even during reading, so that reading must be done from high to low byte. The circuit also has a reset pin that resets each counter.

Reading needs 8 General Purpose input/output (GPIO) pins for transferring the data only. The minimal number of pins needed for interfacing the IC was 15. This was without using the full potential of the IC. This pose a problem for the RPi because of the 40 pin header that, besides the encoder reading had a couple of peripherals connected as well. Also, because of the header pinout that goes without any particular order, the wiring diagram was very complex. In spite all this issues we managed to connect the 15 pins to the RPi, via level shifters, and developed an algorithm for interfacing the IC. After testing the algorithm on a microcontroller in order to confirm that it is working we came to a conclusion that it was not possible to interface this IC from the RPi. There were a couple of reasons for this. Firstly, because the IC is still counting while we are obtaining the data, the data transfer speed should be as fast as possible in order to avoid data loss. The problem here was that RPi pinout is defined in a way that every pin is read separately, where on microcontrollers we have 8-pin ports. Besides that we hat the issue with the Linux operating system, which like most operating systems isn't real-time. We couldn't ensure fast switching of the bytes that are being read is fast enough as well so, as a result, the read data was corrupted.

The conclusion was that there isn't a reliable way to read information from encoders with a RPi, without substantial loss of data. To overcome this problem, we have used a microcontroller which is used to read the encoders and it connected to the RPi using serial communication. Because we had tested the algorithm for interfacing the IC, and we knew that the IC is reliable we decided to connect the encoders to the IC, we interfaced the IC with the microcontroller and then passed the information to the RPi using serial communication.

The microcontroller used is PIC18F4431, which was also something that we had at hand, and different microcontroller can be used. We have connected the 15 lines from HCTL-2032 to the microcontroller. Because of the different logic level between the RPi and the microcontroller, we must use a level shifter for serial communication.

A custom two-layer PCB was designed, shown on figure 2, to hold the HCTL-2032, microcontroller, level shifter and a voltage regulator. The 8-pin parallel interface from the IC was connected to port D of the microcontroller so we could have one cycle reading of a byte. The constraints of the board ware the dimensions of the robot and the space required for all of the components.

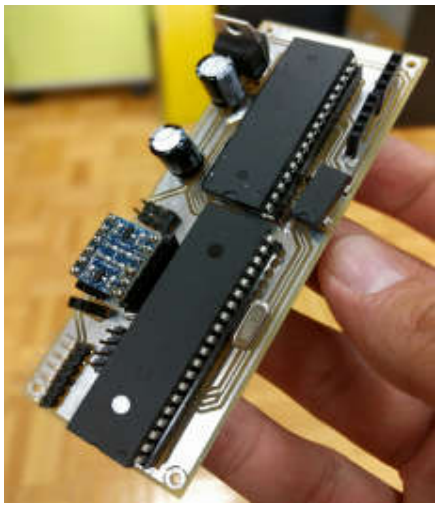

Fig. 2. Custom made two-layer PCB for the encoder reading part of the system

The serial communication is two-way, where the RPi acts as a master and the microcontroller is the slave. There are 7 defined commands that the RPi can issue to the microcontroller:

- Enable – the microcontroller start reading the number of counts for two encoders periodically and sends the information to RPi.
- Disable – the microcontroller stops reading and waits for the next command.
- Reset X – asynchronous reset of the counter for the first encoder.
- Reset Y – asynchronous reset of the counter for the second encoder.
- Reset All – asynchronous reset of the counters for both encoders.
- Acknowledge X – synchronous reset of the counter for the first encoder.
- Acknowledge Y – synchronous reset of the counter for the second encoder.

Enable and disable commands are defined if for some reason we want to pause the reading of the encoders. The asynchronous reset commands are present in case data corruption is detected. They also should be issued before the enable command to ensure a 0 starting point. The synchronous reset commands are sent every time the RPi receives the number of pulses, so that the microcontroller knows it is safe to reset the counter. Resetting is important in order to avoid counter overflow which would lead to significant data loss. When in reading mode, every 50 ms, the microcontroller reads the data from the IC for each encoder and if acknowledgment was received, resets the counter for the specified encoder immediately. The reset should be executed right after the read in order to minimize the number of pulses that are lost between the read and the reset. The microcontroller doesn't store the value because it is being stored in the IC, it

immediately forwards it to the RPi. This way the possibility of having data synchronization problems is minimized. Because each time, the RPi obtains the data, the counter is being reset, the data represents an increment of the travelled distance.

## III. ODOMETRY

Microcontroller sends the read values to the RPi, and then the values are decoded and tested. If the data is valid, RPi signals a successful read so that the microcontroller can reset the counters.

The read value of the counted encoder pulses is transfer to mm and added to the current value of traveled path. The conversion from number of pulses to mm is done using equation (1). Where $d$ is the length of path traveled in mm, $n$ is the number of pulses from the encoder, $R$ is the wheel dimeter, $N_m$ is the ratio of the gearhead and $n_o$ is the number of the impulse per revolution of the encoder [2].

$$d = \frac{nR\pi}{N_m n_o} \tag{1}$$

With the combination of the encoder and gearhead, and with 58.5 mm diameter of the wheel, we can calculate the resolution of traveled path which is 0.012819mm.

To calculate robot's location, we use two different set of equations, one for straight line and one for curved movement. We differentiate the two movements based on the difference of path traveled by the two wheels. If it's grater then zero then the movement is on a curved line, and we use equations (2) for calculating the increments in angle, x and y coordinates. Where $l$ represents the distance between the wheels, $d$ is the traveled path of right or left wheel, and $k$ a time point.

$$\begin{aligned}
\Delta\theta_{(k)} &= \frac{d_{r(k)} - d_{l(k)}}{l} \\
\Delta x_{(k)} &= \frac{1}{2}\frac{d_{r(k)} + d_{l(k)}}{d_{r(k)} - d_{l(k)}}\left(\sin\left(\theta_{(k-1)} + \Delta\theta_{(k)}\right) - \sin\left(\theta_{(k-1)}\right)\right) \\
\Delta y_{(k)} &= \frac{1}{2}\frac{d_{r(k)} + d_{l(k)}}{d_{r(k)} - d_{l(k)}}\left(\cos\left(\theta_{(k-1)}\right) - \cos\left(\theta_{(k-1)} + \Delta\theta_{(k)}\right)\right)
\end{aligned} \tag{2}$$

Then we add these increments to the previous angle and coordinates and get the current values. This means that we form the referent coordinate system based on the position and orientation of the robot when the application was started.

We can see that for the straight movement if we use equations (2) zero will be the divider and we must use different equations (3).

$$\begin{aligned}
\Delta\theta_{(k)} &= 0 \\
\Delta x_{(k)} &= \frac{d_{r(k)} + d_{l(k)}}{2}\cos\left(\theta_{(k-1)}\right) \\
\Delta y_{(k)} &= \frac{d_{r(k)} + d_{l(k)}}{2}\sin\left(\theta_{(k-1)}\right)
\end{aligned} \tag{3}$$

Only equations for straight movement (3) could be used for calculating, which would mean that a curved path is approximated with short straight segments. This can be done if the traveled path between two readings of the encoders is small enough, which they are in our case, but this would add an approximation error that could over time accumulate.

By using both sets of equations we lower the error in calculating the robot's location. The setup was experimentally tested and proven that it can provide reliable data about robot position. The test has been done on a small workspace, and on a larger workspace we will probably accumulate a substantial error.

## IV. CONCLUSION

Although the RPi is a cheap platform intended for teaching basic concepts, and as a development platform, it can, with the right combination of additional electronics, be used for research purposes. There some limitations to this.

High resolution encoders can be used if specialized ICs are used with a microcontroller. This provides a low cost reliable solution with a high resolution distance measurement.

We have built a mobile robot platform using RPi that can be used for different types of research. The positioning of that robot relies on odometry implementation described in this paper.

Further work may include determining the accumulation error of the system, using some kind of external reference points, which the system can detect.

## REFERENCES


[1] Wirth, M., Weichmann, F., Schaeffel, F., Zrenner, E. and Strasser, T. (2013), Keep an eye on the Pi – Using the Raspberry Pi as inexpensive, yet powerful platform for vision research. Acta Ophthalmologica, 91: 0. doi:10.1111/j.1755-3768.2013.T028.x.

[2] Borenstein J., Everett H. R., Feng L. „Where am I? Sensors and Methods for Mobile Robot Positioning“, 1996