Differential drive robots: An Arduino-based motion control system

Università degli Studi di Bergamo

Lorenzo Nava

15 July 2018

Abstract

This paper illustrates the process, functions and tools used in order to achieve a core base for future sensor implementation in laboratory ausiliary robots.

The code reported throughout this paper is made for a specific set of components, it is not a general library, it is meant be used as a **base** for general purpose robots that share similar characteristics with the robot used here or it could be used to replicate the same robot by using the code and the components reported here.

It was decided not to create a library in order to make things as simple as possible both to read and integrate. Previous projects [reference], created general purpose libraries, but any correction and/or modification to the code is painful and generate loads of errors.

The robot is able to:

- send (remotely) its position (relative to a starting point)
- receive (remotely) a twist command where its linear and angular speed are set

In order to achieve the results mentioned here it were used odometry calculation...

Contents

1 Introduction **3**

2 Hardware composition **4**

2.1 Componentry overview 4

2.1.1 LabMate 4

2.1.2 Control unit 4

2.2 Components in detail 5

2.2.1 Motors 5

2.2.2 Encoders 6

2.2.3 Wheels 6

2.2.4 Arduino Due 7

2.2.5 PC 8

3 Software **9**

3.1 Position tracking Arduino 9

3.1.1 How it works 9

3.1.2 Code in detail 11

3.2 Wheels driving Arduino Due 13

3.2.1 How it works 13

3.2.2 Code in detail 14

Chapter 1

Introduction

The whole concept behind this project is to make a robust core base for the LabMate in order to easily implement sensors. The robot can be controlled remotely by another computer via **ssh** and it is also possible to get its position by **ssh**. So if, for example, a future application needed to implement cameras so the robot can deviate obstacles, the only thing that should be managed is the image processing for the camera and, based on it, sending twist comands to the robot.

Here, it will be exposed the composition of the LabMate in techinical terms. There will be no explanation whatsoever about how the components work.

Chapter 2

Hardware composition

In this chapter it will be firstly illustrated an overview of the components, so the reader can create an idea of the whole robot. Secondly, every component will be examplained in detail. Both sections explaining first the components to be controlled and then the control units.

2.1 Componentry overview

In this section the components of the LabMate will be exposed in a general way, without much detail.

2.1.1 LabMate

The robot itselft is composed by:

- Two brushed motors working in parallel
- A wheel for each motor with a reduction of 1:10 (wheel:motor)
- Two encoders attached to each motor

2.1.2 Control unit

The control unit is composed by:

- An Arduino Due for tracking the position

- An Arduino Due for driving the wheels
- A PC that gets/sends information from/to the Arduinos via USB and gets/sends information from/to a more powerfyl remote PC via SSH

The heart of the robot is the PC mounted onboard as it controls and gets information from the Arduinos. The brain of the robot would be a remote PC connected to the onboard PC, that would process different types of data and send control information to the onboard PC. The onboard PC could also be used as the brain of the robot, but the whole concept presented here is to process large amounts of data in a more powerful remote PC and use the onboard PC to drive the robot.

The Arduino tracking the position gets the distance traveled by each wheel and, based on it, retrieves to the PC the coordinates and the direction of the robot via USB.

The Arduino driving the wheels is always listening for a new speed for each wheel (setpoint), sent by the PC over USB.

The choice of using two Arduinos was taken as just one Arduino couldn't handle calculating the coordinates and driving the wheels at the same time.

2.2 Components in detail

In this section all the LabMate components will be exposed in detail.

2.2.1 Motors

Technical specifications

Property	**Value**
Type	DC/Brushed
Voltage	22V
Current	9A
Maximux speed	1300 rpm
Maximux power	0.13 KW
Wheel radius	75 MM
Reduction wheel:motor	1:10

2.2.2 Encoders

The encoder is attached to the motor and not the wheel, so it is necessary to take in consideration the wheel reduction to get the correct number of impulses for each rotation.

Channels A and B of the encoders were split in two in order to provide the signals for both Arduinos.

Gear ratio issue

During the test phase a problem was found in the readings of the encoders, probably as the wheel reduction is not exactly 1:10, which leads the encoders to have a 40440 IPR and not a 40000 IPR, which is the theoretical IPR for the wheel ($IPR_{wheel} = 4 \cdot CPR_{encoder} \cdot G_{ratio}$). [reference]

Technical specifications

IPR indicates "Impulses Per Revolution".

2.2.3 Wheels

The only information available about the wheels is its radius, which is 75 mm. [ref]

Property	**Value**
Maker	Avago Technologies
Model	HEDS 9000
Type	Incremental
Voltage	4.5V $\sim$ 5.5V (DC)
CPR on motor	1000 (4000 impulses in quadrature)
IPR on wheel	40000
Real IPR on wheel	$\sim$ 40400
Output	Dual channel/Digital

To be correct, it should be mentioned that the robot has other four wheels on its four corners to provide stability and support, nothing more. Also, the two driving wheels are attached to a spring to absorb the irregularities present in the field.

2.2.4 Arduino Due

Of all Arduinos tested, the Arduino Due was the only one capable of reading all the impulses generated by the encoders.

Technical specifications

Property	**Value**
Maker	Arduino
Model	Due
Microcontroller	AT91SAM3X8E
Clock Speed	84 MHz

Position tracking Arduino Due

The Arduino is connected to both encoder's channels A and B for a total of 4 pins used. It is also connected to the onboard PC via USB.

Wheels driving Arduino Due

The Arduino is connected to both encoder's channels A and B. Also, for each wheel the Arduino dedicates two pins: one for the wheel direction and one for the pwm signal. The number of pins used by the Arduino in total is 8. It is also connected to the onboard PC via USB.

2.2.5 PC

Chapter 3

Software

In this chapter it will shown the software for each hardware component exposed in the Hardware chapter.

3.1 Position tracking Arduino

This Arduino takes as input channels A and B for both left and right encoders. By using a technic of odometry calculation it is able to determine the current coordinates and direction of the robot relative to a starting point. These coordinates and direction are transmitted via USB.

3.1.1 How it works

In this implementation both the timer and void loop() are used. The timer handles the reading of the encoders while the void loop() is used to send the coordinates over USB periodically. The use of the timer, or rather of timed interrupts is handled by the ¡Timer.h¿ library which calls the function assigned to one of Arduino's (Arduino Due) timers once in a period defined by the code.
The reading of the encoders is handled by the library ¡Encoder.h¿ [ref], which updates the impulses of the encooders every time the function Encoder.read() is called. As it was decided on not to use interrupts to read the encoders as it caused issues

due to impulse overlapping, the function Encoder.read() needs to be called periodically so library can compare channel A and channel B values to detect an impulse and its direction (backwards or forward). This can be a limitation depending on the wheels speed, in this application the wheels speed is not relatively high so the timer mounted on the Arduino can easily handle the operation of reading all the impulses without any misreadings.

The odometry calculation is obtained by trigonometry. The resulting formulae are:

$$\theta_{i+1} = \theta_i + \frac{(\Delta d_{left} - \Delta d_{right})}{d_{wheels}} \tag{3.1}$$

$$x_{i+1} = x_i + \Delta d_{right} \cdot \sin(\theta) \tag{3.2}$$

$$y_{i+1} = y_i + \Delta d_{left} \cdot \cos(\theta) \tag{3.3}$$

θ_i: direction of the robot compared to x axis
$\Delta d_{left} = d_{left_{i+1}} - d_{left_i}$: linear distance traveled by left wheel from last reading i
$\Delta d_{right} = d_{right_{i+1}} - d_{right_i}$: linear distance traveled by right wheel from last reading i
d_{wheels}: distance between the two driving wheels
x: x axis coordinate
y: y axis coordinate

There are three functions that send data relative to the position of the robot. Each function sends a specific set of data (coordinates, direction and/or impulses), since not all information is useful all the time. The data is sent over USB by concatenating each piece of data to a string separated by a space. The order of the data must be known a priori by the receiver as there is no identifier for each piece of data. The only way of identifying the arriving data is by knowing the order by which the data was sent.

3.1.2 Code in detail

Here some pieces of code will be exposed in detail. Only some pieces of code are shown here, for the entire source code please refer to [appendix x].

void setup()

Here it is shown the assignment of the function read_encoders() to Timer3. In other words, at every READ_PERIOD defined by the program the function read_encoders() will be called.

```
#define READ_PERIOD 100                // timer period in microseconds

void setup()
{
    Serial.begin(115200);
    Timer3.attachInterrupt(read_encoders).setPeriod(READ_PERIOD).start();
}
```

void loop()

The loop allows the odometry computation only when the odometry flag is set to true, that is when the distance traveled by each wheel has been updated. This is necessary as the odometry cannot be computed without the correct detection of distance traveled by the wheels.

The computed odometry data is only sent once every 1000 iterations of the void loop(), in such a way that the transmission of data over serial do not slow down the odometry computation.

```
void loop()
{
  // Odometry can be computed only if the distance traveled has been updated
  if (odometry == true)
  {
    odometry == false;
    compute_odometry();
    odometry_cnt++;
    if (odometry_cnt >= 1000) {
      // send_coordinates();
      // send_impulses();
      send_all();
    }
    odometry_done = true;
  }
}
```

compute_odometry()

The odometry is calculated using the formulae mentioned before.

```
void compute_odometry()
{
    // transform the distance traveled from impulses to MM
    r_distance = delta_r_enc * R_STEP_LENGTH;
    l_distance = delta_l_enc * L_STEP_LENGTH;
```

```
6
7       // Compute coordinates and direction
8       theta = theta + (l_distance - r_distance) / double(WHEELS_DISTANCE_MM);
9       coordinates[0] = coordinates[0] + double(r_distance) * sin(theta);
10      coordinates[1] = coordinates[1] + double(l_distance) * cos(theta);
11  }
```

read_encoders()

The distance traveled by each wheel is calculated no more than once at every 100 times the impulses are read, it may be read less than one time every 100 if the last odometry compuation have not finished yet. This is achieved by using the flag odometry_done that is set to true every time the odometry computation is done. The distance traveled by the wheels is computed less than once every 100 times the impulses are read by the only fact that if it was calculated at every impulse it would lead to value of θ too small to be accurate enough for the odometry computation.

```
1  void read_encoders()
2  {
3    r_enc = R_ENC.read(); // Reads right encoder's impulses
4    l_enc = L_ENC.read(); // Reads left encoder's impulses
5
6    cnt++;
7
8    // Updates the odometry variables every 100 iterations and only if the last odometry
       computation has finished
9    if (cnt >= 100 && odometry_done)
10   {
11     l_enc_cpy = l_enc * CPR_CORRECTION; // by multiplying l_enc by CPR_CORRECTION it is taken in
         consideration the gear ratio error
12     r_enc_cpy = r_enc * CPR_CORRECTION;
13
14     // Calculate the distance in impulses traveled by each encoder
15     delta_l_enc = l_enc_cpy - l_enc_old;
16     delta_r_enc = r_enc_cpy - r_enc_old;
17
18     odometry = true; // Odometry permission flag. If set to true, void loop() can compute the
         odometry
19     odometry_done = false; // Resets odometry completion flag
20     cnt = 0;
21
22     l_enc_old = l_enc_cpy;
23     r_enc_old = r_enc_cpy;
24   }
25 }
```

Sending functions

The different types of data that can be sent over USB:

send_coordinates()

```
1  /*
2    Send coordinates separated by space to serial port.
3  */
4  void send_coordinates()
5  {
6    String str;
7    str.concat(coordinates[0]);
8    str.concat(" ");
9    str.concat(coordinates[1]);
10   str.toCharArray(b, 200);
11   Serial.write(b);
12   Serial.write("\n");
13 }
```

send_impulses()

```
/*
  Send left and right impulses separated by space to serial port.
*/
void send_impulses() {
  String str;
  str.concat(l_enc_cpy);
  str.concat(" ");
  str.concat(r_enc_cpy);
  str.toCharArray(b, 200);
  Serial.write(b);
  Serial.write("\n");
}
```

send_all()

```
/*
  Send left/right impulses, coordinates and rotational angle separated by space to serial port.
*/
void send_all() {
  String str;
  str.concat(l_enc_cpy);
  str.concat(" ");
  str.concat(r_enc_cpy);
  str.concat(" ");
  str.concat(coordinates[0]);
  str.concat(" ");
  str.concat(coordinates[1]);
  str.concat(" ");
  str.concat(theta*180/PI);
  str.toCharArray(b, 200);
  Serial.write(b);
  Serial.write("\n");
}
```

3.2 Wheels driving Arduino Due

This Arduino takes as input channels A and B for both left and right encoders. Also, it takes as input the setpoints for left and right motors via USB. The goal of this Arduino is to bring the wheels to the received setpoints. This is achieved by a PID controller that sends to the motors PWM signal.

3.2.1 How it works

The the Arduino has two timers, one for reading the encoders, just like the other Arduino and one timer triggers a function responsible for computing the PID, which changes the outputs associated with the motors (PWM signals), that is changing the wheels speed. Despite using two timers, the Arduino also takes advantage of the void loop() function, this time to get the setpoints from the serial port.

The readings of the encoders, just as the other arduino, is handled by the library <Encoder.h>, and againg just like the other Arduino, the timer is handled by the library <Timer.h>.

The PID controller is handled by the library <PID.h>, in which by given a set-point and the encoders impulses it returns an integer ranging from 0 to 255 wich corresponds the duty cycle value taken by the PWM object that will activate the pin in PWM mode for controlling the motors. The PWM class is given by the <PWM.h> library.

3.2.2 Code in detail

Here some pieces of code will be exposed in detail. Only some pieces of code are shown here, for the entire source code please refer to [appendix x].

void setup()