Siddharth Patel

Application of PID on Differential Drive Robot

HTW Berlin - University of Applied Sciences

(Hochschule für Technik und Wirtschaft Berlin)

Treskowallee 8, 10318 Berlin

WS 2023/2024 - Electrical engineering

Preface 5
1 Differential Drive...6
1.1 Aim 6
1.2 Differential drive for our case 7
1.2.1 geometry_msgs/Twist Message 7
1.2.2 Differential drive equation for a two-wheeled robot 8
1.3 Implementing Equation 1.1 on Arduino and interfacing with the motor driver 9
1.3.1 Sabertooth 2x5 Motor Driver 10
1.3.2 Simplified serial mode and the I/O connections 10
1.3.3 SabertoothSimplified.h 11
(i) ST.motor( ) 11
(ii) Other functions and examples 12
1.3.4 Arduino code for a simple differential drive 12
(i) Adapting Equation 1.1 for Arduino code 12
(ii) Measuring Ѡmax , R (Radius) and L (Width) 14
(iii) Putting it all together and writing the final Arduino code 15
2 Magic Box 16
2.1 Breaking Down the Magic Box 17
2.2 Breaking Down Control Velocity Logic for One* DC Motor 18
3 Encoder Signals 20
2.1 Basic Understanding of the Need for Encoders 20
2.2 How to Control DC Motor Using Encoder 21
2.2.1 Exploring the datasheet of the encoder 21
Step 1: How to read from Encoder 23
Step 2: Learn How to Measure the Position of the Motor Shaft 25

Preface

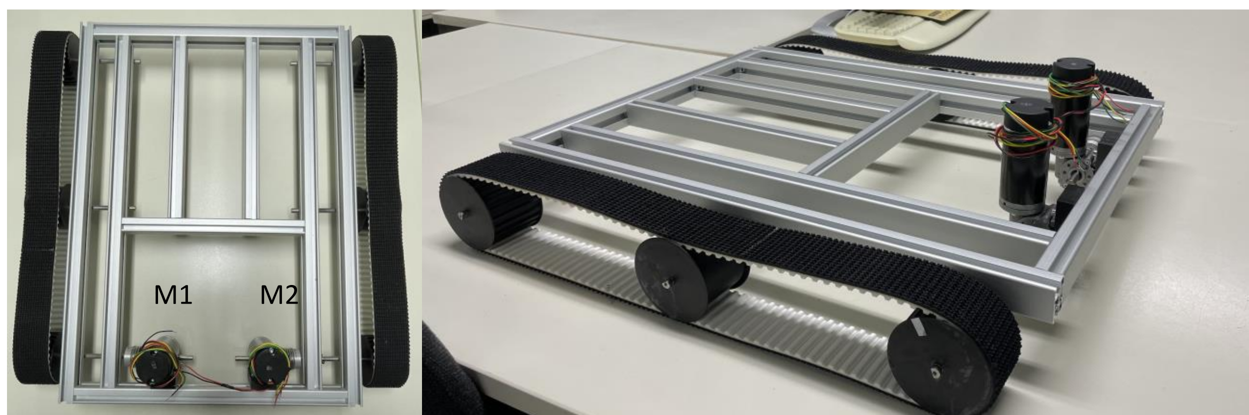
I am currently an Electrical Engineering student at HTW Berlin, actively engaged in a collaborative venture with Professor Jan Carsten's company. My primary focus involves the practical application of PID control loops on differential drive robots, a nuanced intersection of academic theory and real-world implementation. This document serves as a dual-purpose tool—systematically tracking my progress and breaking down the overarching challenge into manageable components. Collaborating with Professor Jan Carsten provides a valuable opportunity to bridge the gap between theory and practice. As I navigate the complexities of PID control for differential drive robots, this document serves as a transparent record of my contributions and progress in the project.

Chapter 1:

1 Differential Drive

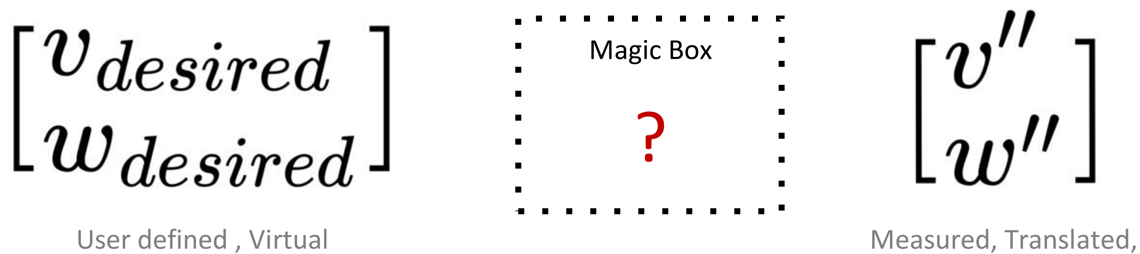
1.1 Aim

Let's begin by identifying the problem, or, in other words, clarifying our aim. Defining our objective is crucial, as it provides a clear direction for determining how to achieve it. This clarity not only simplifies our thought process but also enhances understanding for other readers, allowing them to grasp the context more easily. Our objective revolves around a robot equipped with two motors connected to wheels, each motor featuring an encoder (refer to Picture 1.1 below).



Picture (1.1)

Assigned with the task of creating Arduino logic, my goal is to establish a framework where the desired setpoint velocity—let's call them ($V_{desired}$, $W_{desired}$)—serves as user inputs. Here, $V_{desired}$ represents the linear resultant velocity of the rover, while $W_{desired}$ signifies the angular resultant velocity. As outputs, we anticipate V'' (actual linear velocity) and W'' (actual angular velocity) in real-world scenarios. The aim is to ensure that [V'', W''] closely approximates or is nearly equal to the desired setpoint velocity.



Picture 1.2

1.2 Differential drive for our case

Now that we have a clear understanding of the problem, which is to figure out what logic, algorithm, or perhaps magic we are going to put in the box (refer to picture 1.2), so that we get the output as we discussed.

According to the professor, the first step is to determine what ($V_{desired}$, $W_{desired}$) is and why we even need them. In order to answer this question, I will take you one step back and provide information about the ROS function geometry_msgs/Twist Message.

1.2.1 geometry_msgs/Twist Message

According to the Wikipedia, *geometry_msgs* provides messages for common geometric primitives such as points, vectors, and poses. The robot is designed in a way where its computational power or decisions on where to move are derived from the Raspberry Pi. ROS is employed, and as an output from the Raspberry Pi, a vector with targeted velocities ($V_{desired}$, $W_{desired}$) is obtained. Refer to Picture 1.3 for a better understanding.

geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

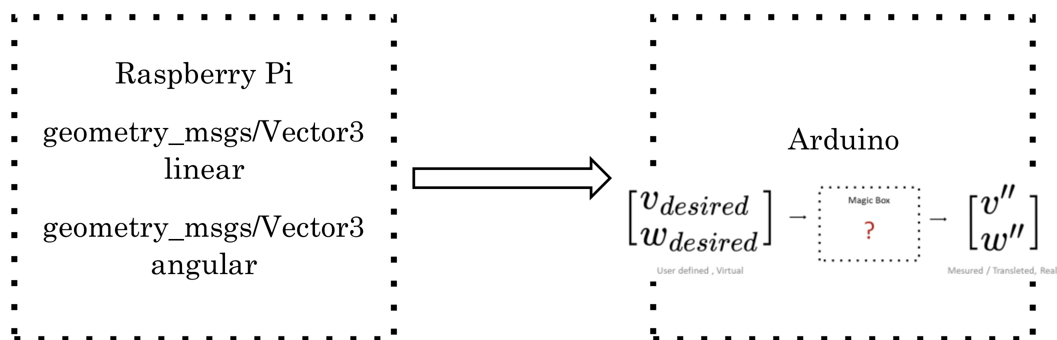
Compact Message Definition

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

autogenerated on Wed, 02 Mar 2022 00:06:53

Picture 1.3 (https://docs.ros.org/en/api/geometry_msgs/html/msg/Twist.html)

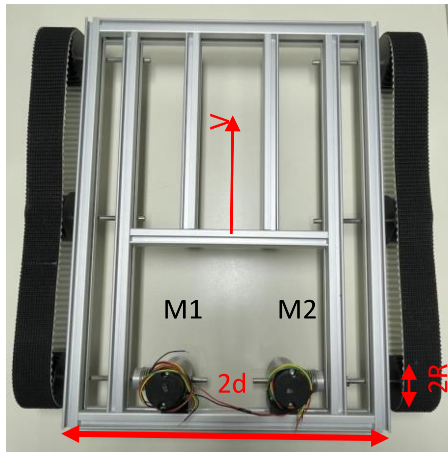
So, the calculation part is being done in Raspberry Pi, and the output of this calculation is transferred to the Arduino, upon which our system is based. I won't be discussing how we obtain the velocity vector from the Raspberry Pi in this document. We will simply assume that we somehow acquire the data about the setpoint velocity vector from somewhere.

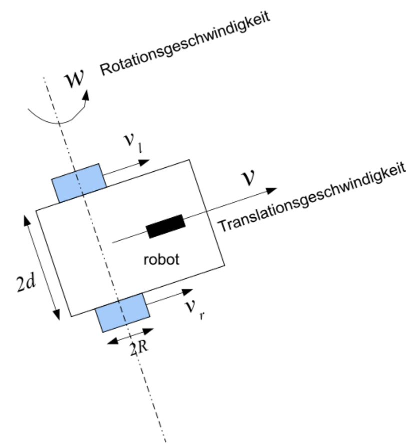


Picture 1.4

1.2.2 Differential drive equation for a two-wheeled robot

Now that we know the vector with linear and angular velocity [($V_{desired}$, $W_{desired}$)] is coming to the Arduino as an input, we first need to find the relation between the robot's parameters such as the radius of the wheels and the width of the robot. Fortunately, we already have the answer to this question. There is a transformation function matrix T that provides the relation between the desired velocity ($V_{desired}$, $W_{desired}$) and the angular velocity of each wheel (W_{right} , W_{left}). See Picture 1.5 for reference.





Picture 1.5

Given in Picture 1.5, if R is the radius of the wheels, and V and W are desired velocities, then the relationship between the target velocity and the angular speed of each wheel is shown below:

$$\begin{pmatrix} w_r \\ w_l \end{pmatrix} = \begin{pmatrix} \frac{1}{R} & \frac{d}{R} \\ \frac{1}{R} & \frac{-d}{R} \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix}$$

Equation 1.1

Sources: Picture 1.5 and Equation 1.1

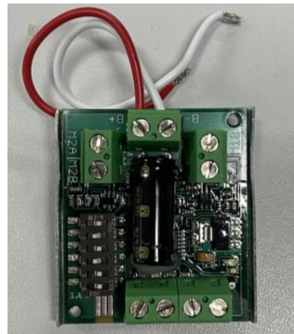
[https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui.inst.130/Mitarbeiter/oubbati/RobotikWS1113/OubbatiSkript.pdf]

1.3 Implementing Equation 1.1 on Arduino and interfacing with the motor driver

Now that we have the relationship between ($V_{desired}$, $W_{desired}$) and (W_{right}, W_{left}), we can begin by creating the Arduino code, considering a system with two motors, one motor driver module (specifically, the *Sabertooth 2x5*, more details on which will be provided later), one Arduino Mega, and a 4-cell LiPo battery (16.7V).

To develop the Arduino code, we can divide the task into several different parts. Let's begin by understanding the motor driver we are using and how it interfaces with the Arduino or any possible Arduino libraries that can be utilized.

1.3.1 Sabertooth 2x5 Motor Driver



Picture 1.6 Sabertooth 2x5 Motor driver

Sources: [https://www.generationrobots.com/media/sabertooth2x5-user-guide.pdf]

Sabertooth 2x5 can be used with Arduino using several different methods, such as analog signals, radio signals, serial communication, and many more (feel free to explore them in the Sabertooth 2x5 user guide here). In our project, I have decided to work with the simplified serial mode using the SabertoothSimplified.h library. The reason for this choice is that the available public functions in the library make it easy for the user to write complex code.

1.3.2 Simplified serial mode and the I/O connections

The first step in using the simplified serial mode on the Sabertooth 2x5 is to turn on dip switches 1, 3, 5, and 6. This step is crucial as it is the only way to work with serial mode at (most probably) 9600 baud rate.

After setting the Sabertooth 2x5 to simplified serial mode, it's time to establish the I/O connections to the Arduino and the motors.

S1 ----->> TX0 Arduino

VCC ----->> +5v Arduino

GND ----->> GND Arduino

BAT + ----->> LiPo Positive

BAT – ----->> LiPo Negative

Picture 1.7 I/O Connections

Note: The connection for the motor to the Sabertooth can vary, as we need to calibrate the system for forward, backward, or turning motions. Your defined front and back of the robot may be different from mine. Therefore, for calibration, connect the motor in any manner you prefer. Once the code is completed, set $V_{desired} = 0$ and $W_{desired} = 1$. With these values, the robot is expected to move in the left direction because $W_{right} > 0$ and $W_{left} < 0$, causing a left turn. If this does not happen, change the motor terminals and calibrate it to take a left turn when ($V_{desired} = 0$, $W_{desired} = 1$).

1.3.3 SabertoothSimplified.h

Reference: [https://documentation.help/Sabertooth/documentation.pdf]

As discussed in section 1.3.1, we will be using the SabertoothSimplified.h library (refer to the information in the reference). From this library, we will utilize a function called ST.motor() responsible for controlling the motor using bytes. Please refer to the picture below to understand the function.

(i) *ST.motor()*

```
void SabertoothSimplified::motor ( byte  motor,
                                   int   power
                                 )
```

Parameters

motor The motor number, 1 or 2.

power The power, between -127 and 127.

Picture 1.7

Source: Picture 1.7 and section content
[https://documentation.help/Sabertooth/documentation.pdf]

As seen in Picture 1.7, the function accepts two parameters: one being the motor number, and the other being power, an integer ranging from –127 to 127.

−127 // maximum speed in the opposite direction

+127 // maximum speed in the regular direction

(ii) Other functions and examples

Under Classes >> Class Members >> Functions in the Sabertooth documentation PDF ([here]), you can find other useful functions. Additionally, under the section >> Examples, a list of examples can be found (Picture 1.8).

Sabertooth Simplified Serial Library for Arduino
Control your Sabertooth with Simplified Serial.
Main Page | Classes | Files | Examples
Class List | Class Index | Class Members
All | Functions

- drive() : SabertoothSimplified
- motor() : SabertoothSimplified
- SabertoothSimplified() : SabertoothSimplified
- stop() : SabertoothSimplified
- turn() : SabertoothSimplified

(a)

Sabertooth Simplified Serial Library for Arduino
Control your Sabertooth with Simplified Serial.
Main Page | Classes | Files | Examples

Examples

Here is a list of all examples:

- SimpleExample/SimpleExample.ino
- SoftwareSerial/SoftwareSerial.ino
- Sweep/Sweep.ino
- TankStyleSweep.ino

(b)

Picture 1.8 (a) Availabe functions ,(b) Examples

1.3.4 Arduino code for a simple differential drive

(i) Adapting Equation 1.1 for Arduino code

$$\begin{pmatrix} w_r \\ w_l \end{pmatrix} = \begin{pmatrix} \frac{1}{R} & \frac{d}{R} \\ \frac{1}{R} & \frac{-d}{R} \end{pmatrix} \begin{pmatrix} v \\ w \end{pmatrix}$$

Equation 1.1

Performing matrix multiplication in Equation 1.1, we get:

$$\begin{pmatrix} w_r \\ w_l \end{pmatrix} = \begin{pmatrix} (\frac{1}{R} \times v) + (\frac{d}{R} \times w) \\ (\frac{1}{R} \times v) + (\frac{-d}{R} \times w) \end{pmatrix}$$

Equation 1.1 (a)

After simplification...

$$w_r = (\frac{1}{R} \times v) + (\frac{d}{R} \times w)$$

$$w_l = (\frac{1}{R} \times v) - (\frac{d}{R} \times w)$$

Equation 1.1 (b)

$$w_r = \frac{v + dw}{R}$$

$$w_l = \frac{v - dw}{R}$$

Equation 1.1 (c)

Note: In Picture 1.5, the width of the rover is represented as 2d. Therefore, substituting L = 2d, we get:

$$w_r = \frac{v + \frac{l}{2}w}{R}$$

$$w_l = \frac{v - \frac{l}{2}w}{R}$$

Equation 1.2

When implementing Equation 1.2 to create Arduino logic, the resulting code might look like this:

```
float W_left = (v_liner - ( w_rover / 2.0)* w_angular)/r_wheel;
float W_right = (v_liner + ( w_rover / 2.0)* w_angular)/r_wheel;
```

(ii) Measuring Wmax , R (Radius) and L (Width)

To create the Arduino code, it is essential to specify each parameter. For instance, the formula depends on the radius of the wheels and the width of the rover, along with the user-input parameters $V_{desired}$ and $W_{desired}$.

in this scenario, the measurement of R radius and L width requires the use of a measuring tool. Personally, I utilized a digital vernier caliper to measure the radius of the wheels and a ruler for determining the width of the robot. This approach was chosen because the radius was on the scale of millimeters, while the width was on the scale of centimeters. Additionally, a digital RPM meter was employed to measure the maximum angular velocity of the wheels (at full-charged LiPo battery of 16.4V). These values were recorded in an Excel file, as illustrated below, and any necessary calculations were performed. You can access the Excel file here:

[Link to the Excel file]: Wmax_calculation.xlsx

	Lipo Voltage	Max (RPM)	Radius of the Wheels (mm)	Width of the rover (cm)	Vmax (m/s)	ωmax (rad/s)
Variable >	16,45	129	40	48		
In [SI] Units	16,45	2,15	0,04	0,48	0,54	13,51

Picture 1.9 Representation of example values

At this point, errors resulting from system or tool measurements are not being considered. I plan to address these once the logic is ready. It's also worth noting that there may be a possibility of using different wheels and a different size for the robot (varying R and L) towards the end of the project. However, my current focus is not on this aspect.

In addition to this, in the Excel file, you will find the calculation to find the maximum linear and angular velocities ($V_{desired}$ and $W_{desired}$) that you can enter. To achieve this, I defined constants such as radius (R), width (L), and the maximum RPM of the motor at 16.7 V. Afterward, I wrote the Excel formula for Equation 1.2. I also implemented logic in the Arduino code that converts (W_{right}, W_{left}) to the byte format which the motor driver (*Sabertooth 2x5*) can use to interpret the velocity. The logic in the Arduino code is shown below:

```
// Normalize the motor velocities to the range -127 to 127.
int motor_left = round(255 * v_left / (2 * W_max));
int motor_right = round(255 * v_right / (2 * W_max));
```

(iii) Putting it all together and writing the final Arduino code

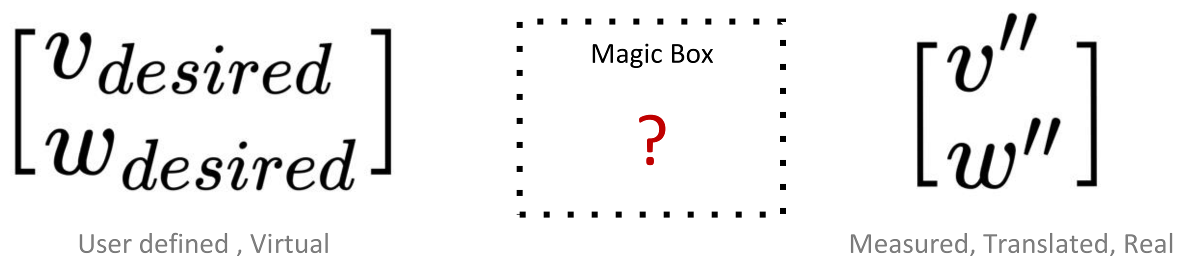
Reference: [Sabertooth Simplified Serial Library for Arduino][Sabertooth 2x5 User's Guide][Einfuhrung in die Robotik]

```
#include <SabertoothSimplified.h>

SabertoothSimplified ST; // We'll name the Sabertooth object ST.
// For how to configure the Sabertooth, see the DIP Switch Wizard for
// http://www.dimensionengineering.com/datasheets/SabertoothDIPWizard/start.htm
// Be sure to select Simplified Serial Mode for use with this library.
// This sample uses a baud rate of 9600.
//
// Connections to make:
// Arduino TX->1 -> Sabertooth S1
// Arduino GND -> Sabertooth 0V
// Arduino VIN -> Sabertooth 5V (OPTIONAL, if you want the Sabertooth to power the Arduino)
//
// If you want to use a pin other than TX->1, see the SoftwareSerial example.
void setup()
{
SabertoothTXPinSerial.begin(9600); // This is the baud rate you chose with the DIP switches.
}

void loop()

{
float v_liner = 0; // linear velocity of rover in m/s
float w_angular = 0; // angular velocity of rover in rad/s
float r_wheel = 0.04; // radius of the wheels in m
float w_rover = 0.48; // width of the rover in m
float W_max = 13.51;

// The calculations below convert the input parameters to motor velocities.
// These calculations assume the left and right wheels have the same speed.
float W_left = (v_liner - ( w_rover / 2.0)* w_angular)/r_wheel;
float W_right = (v_liner + ( w_rover / 2.0)* w_angular)/r_wheel;

// Normalize the motor velocities to the range -127 to 127.
int motor_left = round(255 * W_left / (2 * W_max));
int motor_right = round(255 * W_right / (2 * W_max));

// Apply the motor velocities.
ST.motor(1, motor_left); // Go forward at full power.
ST.motor(2, motor_right); // Go forward at full power.`

}
```

Chapter 2:

2 Magic Box

This chapter is here to remind us of our main goal from Section 1.1 – to figure out the logic behind the illustration in the last section. Remember that Magic Box with a big question mark? That's what we're trying to understand to control velocities like W''_{right} and W''_{left}. In this chapter, I will explain the logic I'm thinking of, or what logic you might consider. The next chapter will break down my plan and the Magic Box logic into basic parts so that, in the end, we can put it all together and get what we want. If it doesn't work out, we'll have to figure out where the logic went wrong and fix it. It's all about working towards our dream, step by step.



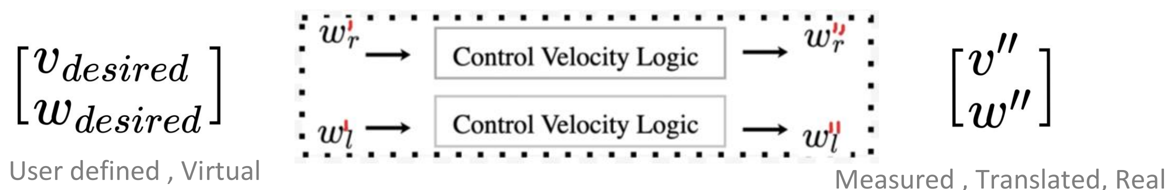
Picture 2.1

2.1 Breaking Down the Magic Box

In the first chapter, we learned a way to turn ($V_{desired}$ and $W_{desired}$) into (W_{right}, W_{left}), and I'll call these (W'_{right}, W'_{left}) with a 'prim' to show there's some error. Check out the illustration below.

Picture 2.2

Now, let's add more details to the picture by introducing two additional boxes for the "*controlling velocity of the one DC motor Logic.*" These will produce (W''_{right}, W''_{left}), representing controlled velocities. This implies that the error is reduced and becomes nearly equal to ≈ (W_{right}, W_{left}). If we translate this into V" and W", it can be expressed as (V", W") ≈ ($V_{desired}$, $W_{desired}$). See the updated illustration below.



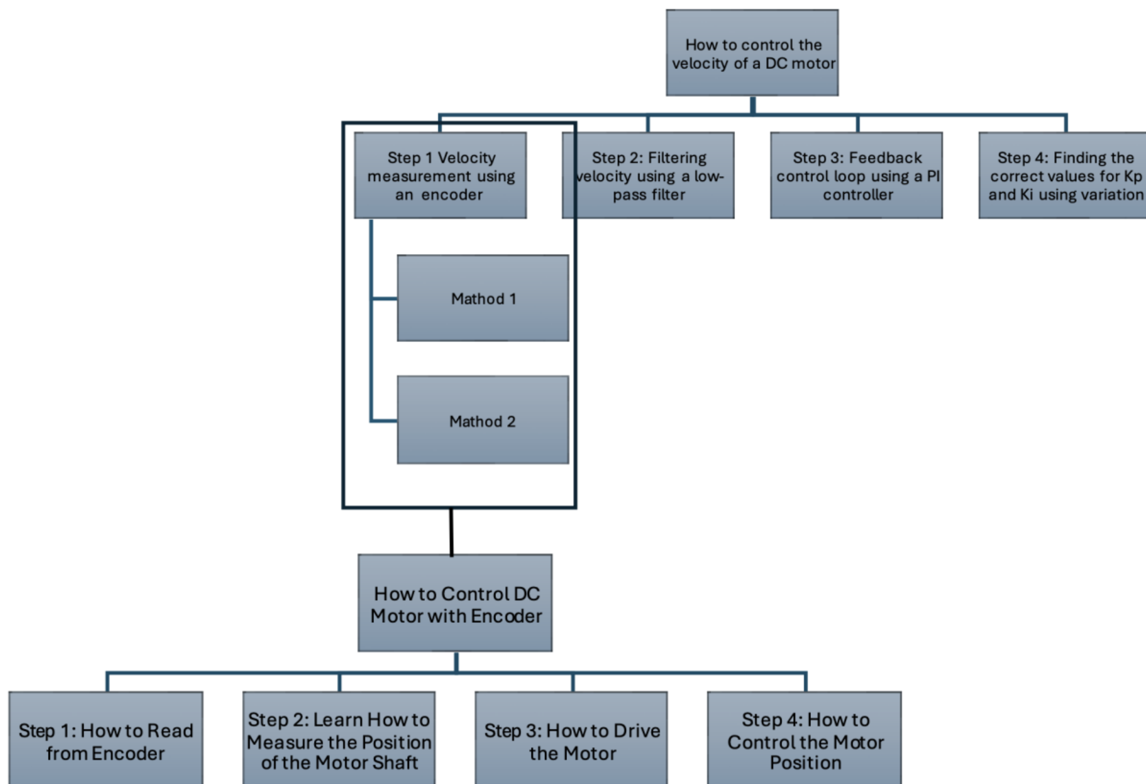
Picture 2.3

Referring to Figure 2.3, if we successfully develop a logic for controlling the velocity of one DC motor, we can eventually integrate all components. Let me break down what needs to be done to control the motor's velocity.

[**Note**: It's important to note that when I use the term "*Control*", such as in Controlling Velocity or Controlling Position, I mean achieving a target point (velocity or position) with minimal error after going through the feedback control loop. We label these as controlled velocity or controlled point as they've undergone a process of reducing error through sensor feedback. If I want to refer to the regular output, I might use the term "Drive." For example, controlling* the motor and driving* the motor represent different aspects in this context.]

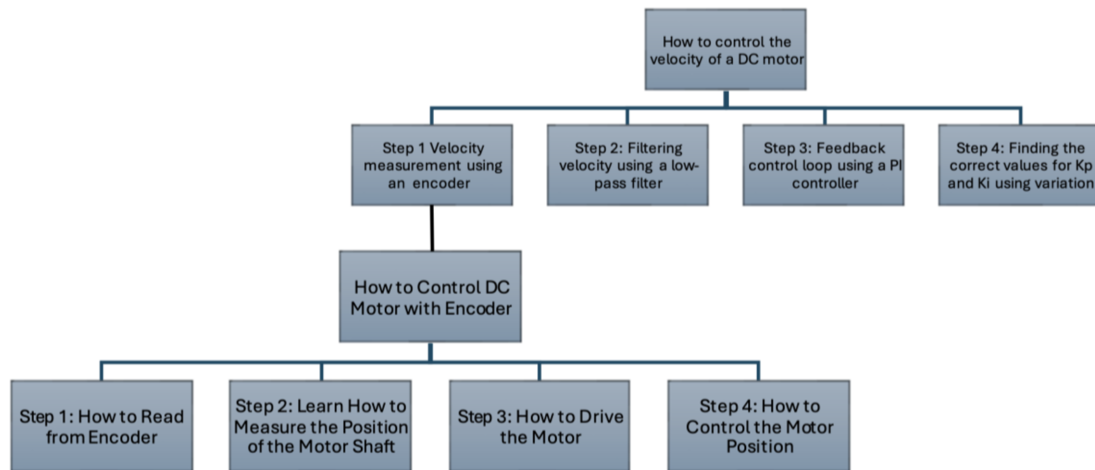
2.2 Breaking Down Control Velocity Logic for One* DC Motor

To better understand the Control Velocity logic, I've put together an illustration for you. Take a moment to review it, and I'll provide a detailed explanation of each step shortly.



Picture 2.4

The above-mentioned illustration (Picture 2.4) depicts the method or short planning of achieving the goal of controlling the velocity of a DC motor. As you can see, the first box at the top states "How to control the velocity of the DC motor," which leads to four steps. The first step involves velocity measurement using an encoder. The second step is filtering the velocity signals, the third involves applying feedback loop using a PI controller, and the fourth and final step is finding the kp and ki parameters (each step will be explained later in more detail).



Picture 2.5

Also, in the first step, "velocity measurement using Encoder," we first need to understand how to control a DC motor with an encoder. That's why I expand the chart by adding one more branch on how to control a DC motor using an encoder, which also has four subparts. It starts with step 1, learning how to read from the encoder. Then, we progress to understanding how to measure the position of the motor shaft, and so on.

Let's start from the bottom and work our way up to achieve our goal of controlling the DC motor's velocity.

Chapter 3:

3 Encoder Signals

2.1 Basic Understanding of the Need for Encoders

Now that we understand the concept of differential drive and have defined our desired velocities, $V_{desired}$ and $W_{desired}$, we have examined how to convert them into angular velocities for each wheel, namely W_{right} and W_{left}. It is now time to proceed, keeping in mind that our goal is to develop a logic for a feedback loop for this system using PID. If you need a reminder of our aim, please refer to section 1.1 Aim.

In our case, a feedback loop involves user-defined setpoint velocities, represented as numerical values. When we apply these setpoint velocities using the code outlined in section 1.3, we observe that it directly translates the setpoint velocities into initial angular velocities. Subsequently, we use specific lines of code to convert them into bytes, which are then sent to the Sabertooth motor driver, resulting in real-life movement of the wheels.

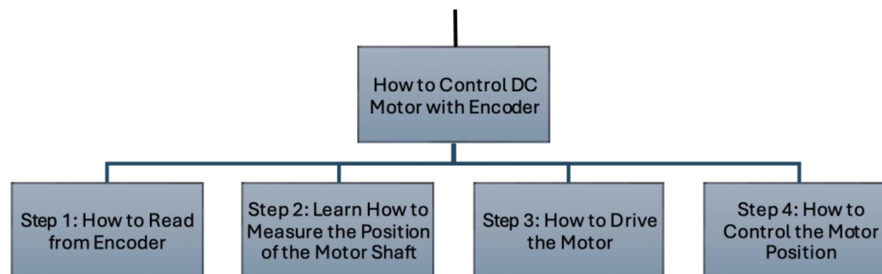
However, it is true that due to varying load situations on the rover, the translated velocities may differ. Consider the following example: if a command for $V_{desired} = 0.5$ and $W_{desired} = 0$ is sent, indicating a desire for the rover to move forward at a speed of 0.5 m/s, external factors such as a slope, rough surface, or excessive weight may cause the rover to only achieve a speed of, for instance, 0.35 m/s. The error caused by the load prevents the rover from reaching the desired speed, highlighting the need for a feedback loop.

Continuing with the same example, let's delve further into the feedback loop. We need a sensor that can determine the actual speed at which the rover is moving in real life. This sensor sends this data back to the logic, where the data is compared. For instance, if Vdesired = 0.5 corresponds to $W_{right} = x$ and $W_{left} = y$, the sensor may read $W_{right}^* = x^*$ and $W_{left}^* = y^*$. It is fair to say that $(W_{right}, W_{left}) \neq (W_{right}^*, W_{left}^*)$ due to the error.

Our objective is to minimize the difference between (W_{right}, W_{left}) and $(W_{right}^*, W_{left}^*)$ to achieve a resultant velocity equal to the desired parameter. This is accomplished through PID control, which stands for Proportional, Integral, and Differential control. However, detailed discussion of PID control is reserved for a different section. In this section, we will first explore what an encoder is and how we can read data from the encoder for later use in our final PID control loop.

2.2 How to Control DC Motor Using Encoder

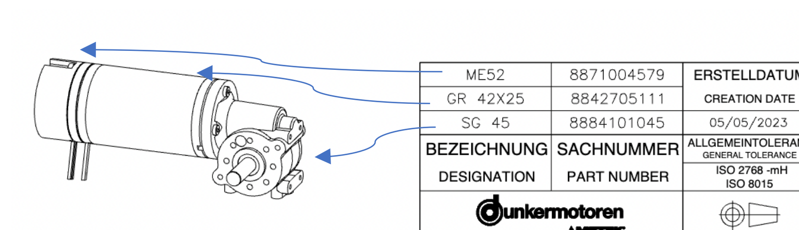
Source and Credit for the Section: [https://youtu.be/dTGITLnYAY0?si=GPbzjtLWgdOMSBbo]



Picture 3.1

2.2.1 Exploring the datasheet of the encoder

In our project, we are using a magnetic encoder that was previously connected to the motor shaft. Please take a look at the company-provided diagram of the encoder with the motor.



Picture 3.2

As you can observe in the diagram, the motor utilized in our project comprises three main components. The first part is the Magnetic Encoder, the second part is the Motor, and the third part is the Gearbox. (Datasheets for each component can be found here.)

Datasheet: Magnetic Encoder [https://cloud.htw-berlin.de/apps/files/?dir=/SHARED/Technik/Rover/Motor%2C%20Main%20Drive&openfile=140523162]

Datasheet: Motor [https://cloud.htw-berlin.de/apps/files/?dir=/SHARED/Technik/Rover/Motor%2C%20Main%20Drive&openfile=140514652]

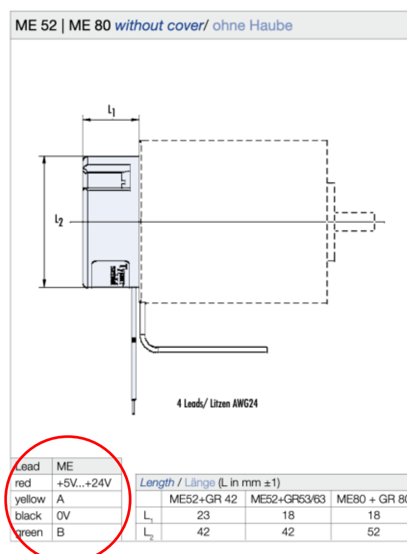
Datasheet: All Combined [https://cloud.htw-berlin.de/apps/files/?dir=/SHARED/Technik/Rover/Motor%2C%20Main%20Drive&openfile=140148932]

Important things to note from this datasheet include the PPR (pulses per rotation) for the encoder and the wire configuration of the encoder. Please refer to the picture below, which displays both pieces of information.

Anbau	
Vorzugsreihe	S
Standardprodukt	J
Encoder Auflösung	12 ppr
Encoder Kanäle	2
Encoder Versorgungsspannung	5
Schutzhaube	N
detailsModal.table.SCHUTZART_A	IP30

Data/ Technische Daten		MG 2			ME 52		ME 80
For motor/ Für Motor		G 30	GR 42	GR 42 mit Haube	GR 53	GR 63	GR 80
Pull-up resistor integrated/ Ausgangsschaltung	-	*open collector*	*open collector + pullup*	*open collector*	*open collector + pullup*		*open collector + pullup*
Signals per rotation/ Signale pro Umdrehung	ppr	2			2 / 12		2 / 12
Output signal/ Ausgangssignale	-	*2 square wave signals, in phase quadrature*/ 2 Rechtecksignale, 90° phasenversetzt					

Picture 3.3



Lead	ME
red	+5V...+24V
yellow	A
black	0V
green	B

Length / Länge (L in mm ±1)			
	ME52+GR 42	ME52+GR53/63	ME80 + GR 80
L_1	23	18	18
L_2	42	42	52

Picture 3.4

From pictures 3.3 and 3.4, it is evident that the PPR (pulses per rotation) value for our encoder is 12, with the yellow wire representing output A and the green wire representing output B. Let's explore where we will apply this information.

Step 1: How to read from Encoder

So, a *magnetic encoder* operates by detecting changes in the magnetic field caused by the magnet attached to the motor shaft. In theory, if output A triggers first, then the motor is moving in a clockwise direction, and if output B triggers first, then the motor is moving in a counterclockwise direction. Let's attempt to interpret this through Arduino code.

```
#define ENCA 2 // Yellow
#define ENCB 3 // Green

void setup() {
Serial.begin(9600);
pinMode(ENCA,INPUT);
pinMode(ENCB,INPUT);
}

void loop() {
int a = digitalRead(ENCA);
int b = digitalRead(ENCB);
Serial.print(a*5);
Serial.print(" ");
Serial.print(b*5);
Serial.println();
}
```

Pin connections:

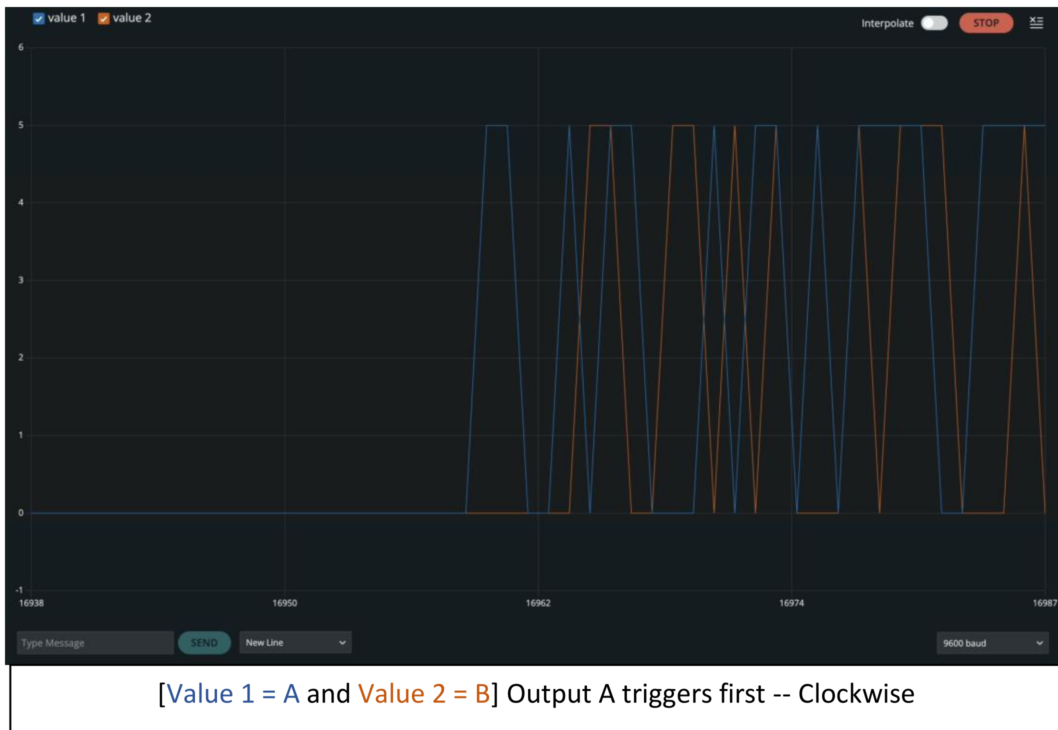
Yellow (A) Encoder --> Digital pin 2 Arduino

Green (B) Encoder --> Digital pin 3 Arduino

Vcc --> 5V

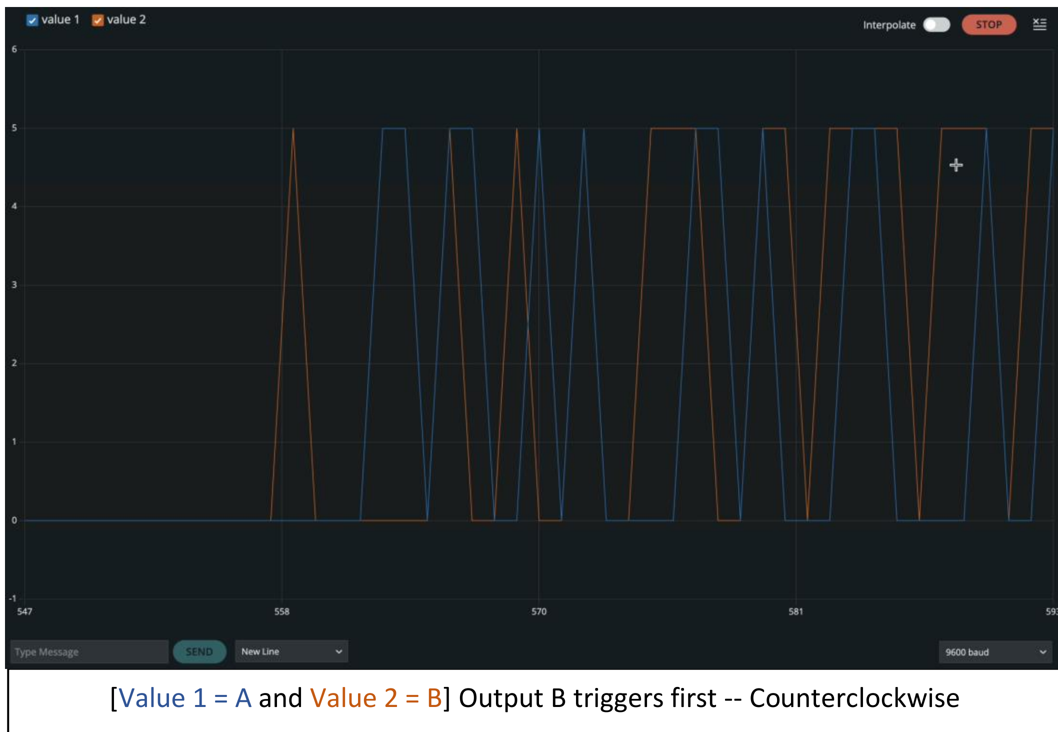
Gnd --> Gnd

Result: When the motor was turned in the clockwise direction, we obtained the following graph with the first trigger on output A. Conversely, when the motor was turned in the counterclockwise direction, we observed the first trigger on output B.



[Value 1 = A and Value 2 = B] Output A triggers first -- Clockwise

Picture 3.5



[Value 1 = A and Value 2 = B] Output B triggers first -- Counterclockwise

Picture 3.6

Step 2: Learn How to Measure the Position of the Motor Shaft

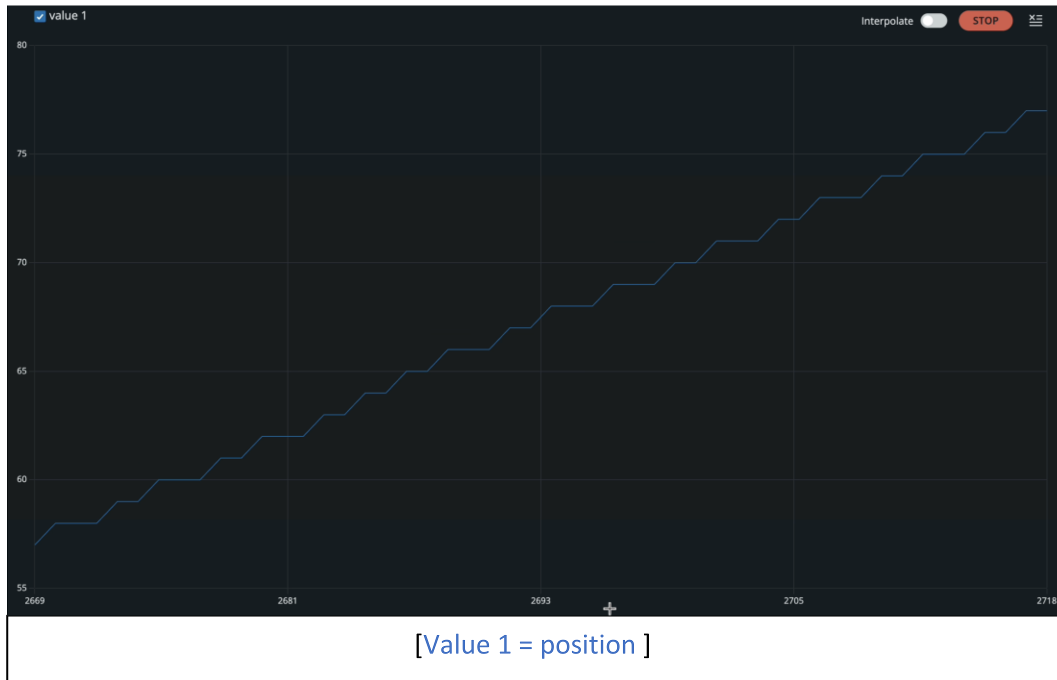
Now that we know how to read from the encoder, we can attempt to measure the position of the motor shaft. This can be achieved by storing values in a variable each time the motor turns. For instance, if it turns in a clockwise direction, we add 1 to the variable, and if it turns in an anticlockwise direction, we subtract 1 from the variable. The starting value will be zero.

```
#include <util/atomic.h> // For the ATOMIC_BLOCK macro

#define ENCA 2 // YELLOW
#define ENCB 3 // WHITE

volatile int posi = 0; // specify posi as volatile:
https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/

void setup() {
Serial.begin(9600);
pinMode(ENCA,INPUT);
pinMode(ENCB,INPUT);
attachInterrupt(digitalPinToInterrupt(ENCA),readEncoder,RISING);
}

void loop() {
// Read the position in an atomic block to avoid a potential
// misread if the interrupt coincides with this code running
// see: https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/
int pos = 0;
ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
pos = posi;
}

Serial.println(pos);
}

void readEncoder(){
int b = digitalRead(ENCB);
if(b > 0){
posi++;
}
else{
posi--;
}
}
```

1.

If the motor is turned with a constant speed, it will exhibit a linear graph. The code is structured to store the points by which the motor shaft has turned. When we plot a

graph of the change in position versus time, it will be linear with a positive slope if the motor turned in a clockwise direction with constant speed and a negative slope if the motor was turned anticlockwise with constant speed. Please take a look at the resulting graph.



[Value 1 = position]