Technical Report System Integration One

Riccardo Medici

INTRODUCTION

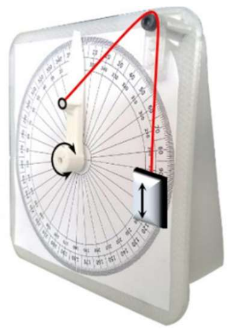
This project objective is to display a physical representation of the speed and the position in degrees of a signal using a hand connected to a DC motor.

To achieve this, the necessary components are:

- Arduino Due microcontroller
- Pololu Dual MC33926 Motor Driver Shield
- DC MOTOR BS138F
- Hall sensor encoder

On the software side a custom designed program to command the Arduino was coded in Visual Studio, whereas the actual code for the Arduino was written in the Arduino IDE.

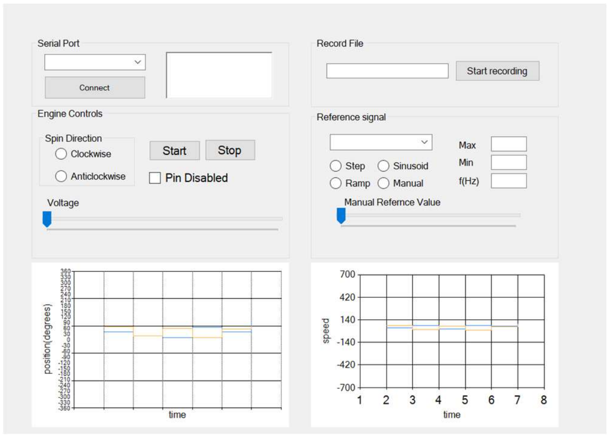
Inside the program it is possible to generate 3 different signals, a step signal, a ramp, and a sinusoid. Those signals are displayed in the two charts in the lower side of the window and are overlapped with the graphs representing the speed and the position of the DC motor.



0.1 figure of the hand connected to the DC motor

It is also possible to manually generate a signal whose speed and position can be changed using a trackbar.

The position, the speed and the direction of the DC motor are calculated using the integrated encoder, a magnetic sensor inside the DC motor with two hall sensors, capable of detecting the direction and velocity of a magnetic plate placed below the two hall sensors.



0.2 image of the visual studio designed program

INDEX

ELECTRONICS

Components

ARDUINO DUE
The Arduino Due is a microcontroller board based on the Atmel SAM3X8E ARM Cortex-M3 CPU. It is the first Arduino board based on a 32-bit ARM core microcontroller. It has 54 digital input/output pins (of which 12 can be used as PWM outputs), 12 analog inputs, 4 UARTs (hardware serial ports), a 84 MHz clock, an USB OTG capable connection, 2 DAC (digital to analog), 2 TWI, a power jack, an SPI header, a JTAG header, a reset button, and an erase button.

The Arduino DUE is the main electronic component of the entire project. The Arduino is responsible for generating the reference signals and controlling the DC motor and encoder using the MC33926 motor driver shield.

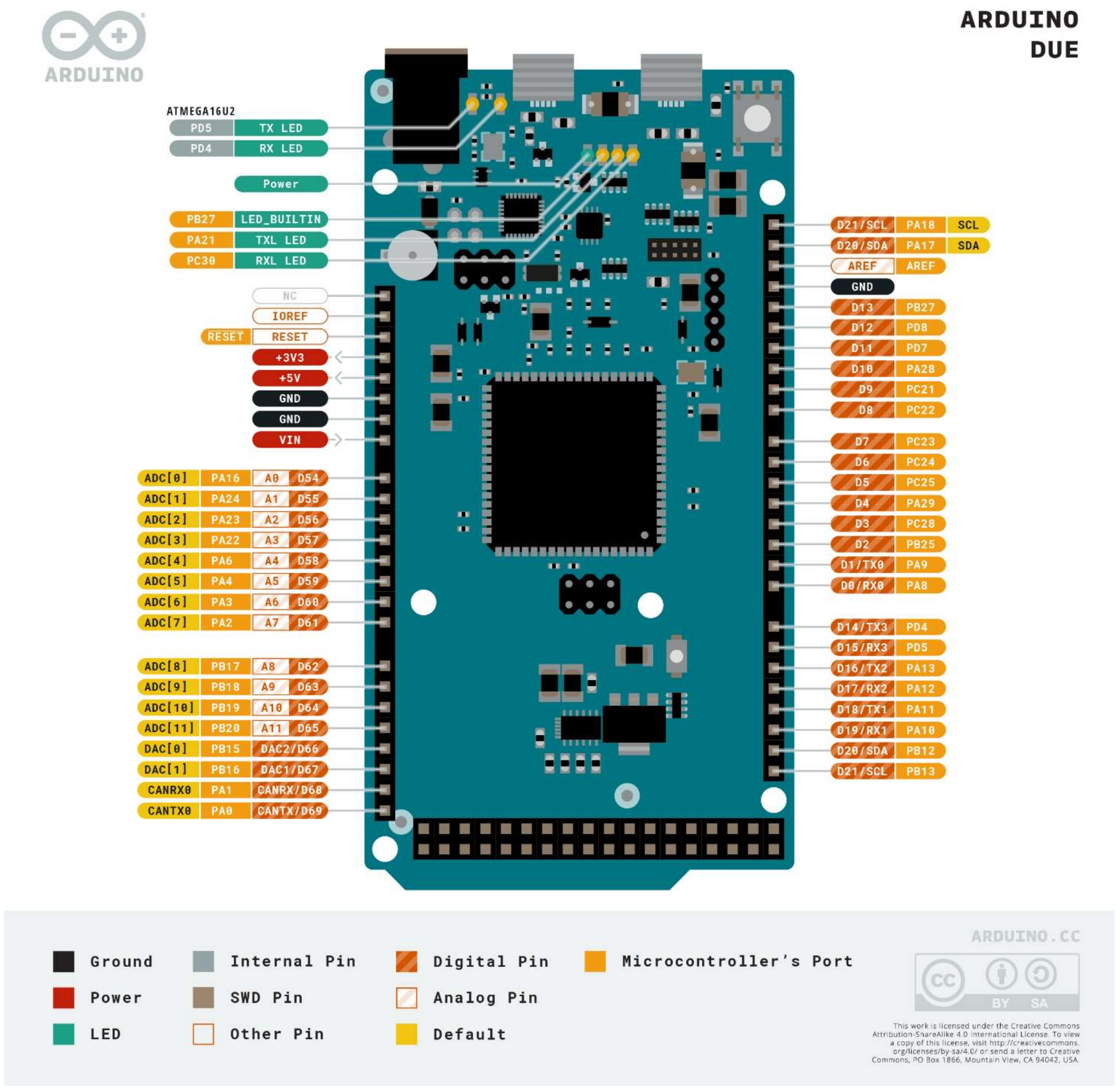
Each of the 54 digital pins on the Due, both digitals and analogs, can be set to be used as an input or an output using the PinMode function inside the setup part in the code. The PinMode function receives two arguments, the number of the pin and its mode (input or output).
They operate at 3.3 volts. Each pin can provide (source) a current of 3 mA or 15 mA, depending on the pin, or receive (sink) a current of 6 mA or 9 mA, depending on the pin. They also have an internal pull-up resistor (disconnected by default) of 100 KOhm.

Using the function analogWriteResolution(), it is possible to set the resolution of the analog pins. It defaults to 8 bits (values between 0-255) for backward compatibility with AVR based boards. However in this project the value is set to 12 bits, in this way it is possible to use the function analogWrite() with values between 0 and 4095 to exploit the full DAC resolution or to set the PWM signal without rolling over.

The Arduino Due can be powered via the USB connector or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery.

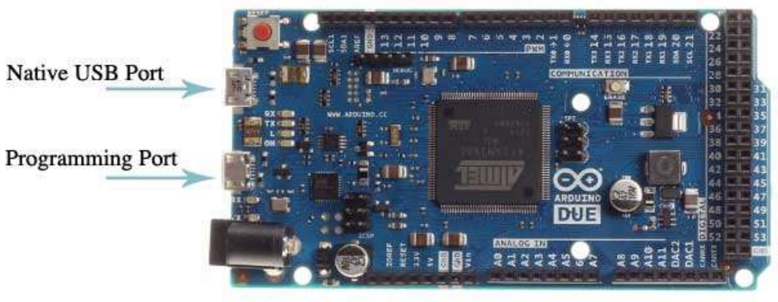
The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

1.0 the arduino due microcontroller



1.1 arduino due pinout diagram

The Arduino communicates with the computer through his two micro usb-type b ports. One called the programming port and the other the native port. Both ports can be used for programming, but it is recommended to use the Programming port due to the way the erasing of the chip is handled.



1.3 Arduino due usb ports

Pololu Dual MC33926 Motor Driver Shield for Arduino

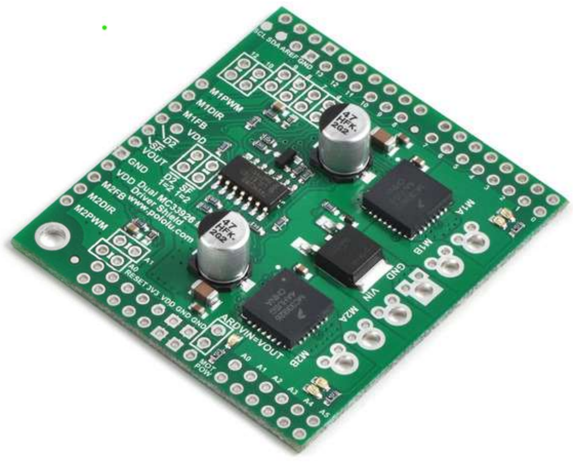
The Pololu Dual MC33926 Motor Driver Shield, connected with a compatible Arduino board, is used to control two brushed DC motors. Its dual MC33926 motor drivers operate from 5 to 28 V and can deliver a continuous 3 A per motor.

Dimensions

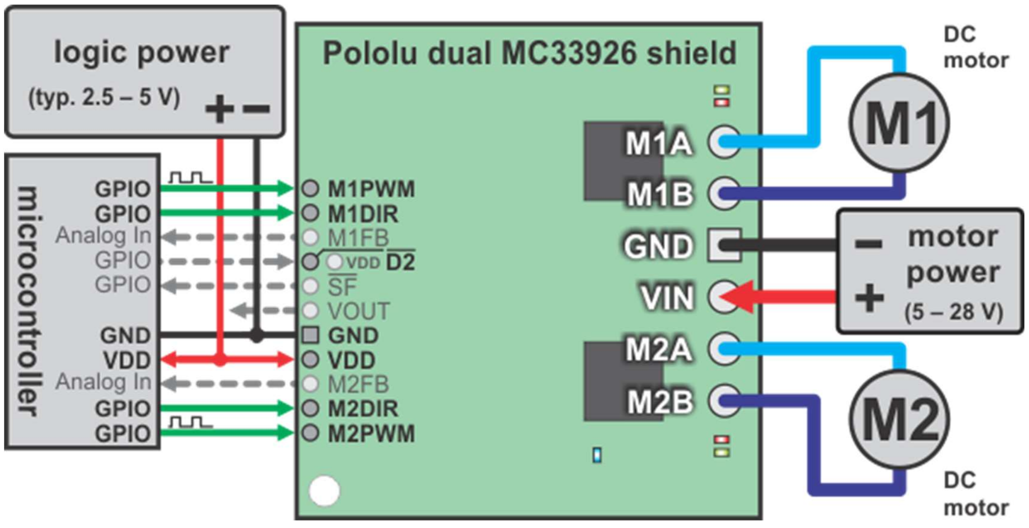
Size:	1.90″ × 2.02″ × 0.38″[1]
Weight:	10 g[1]

General specifications

Motor driver:	MC33926
Motor channels:	2
Minimum operating voltage:	5 V[2]
Maximum operating voltage:	28 V[3]
Continuous output current per channel:	3 A[4]
Current sense:	0.525 V/A
Maximum PWM frequency:	20 kHz
Minimum logic voltage:	2.5 V
Maximum logic voltage:	5.5 V
Reverse voltage protection?:	Y[5]



1.4 Image of the Pololu Arduino shield

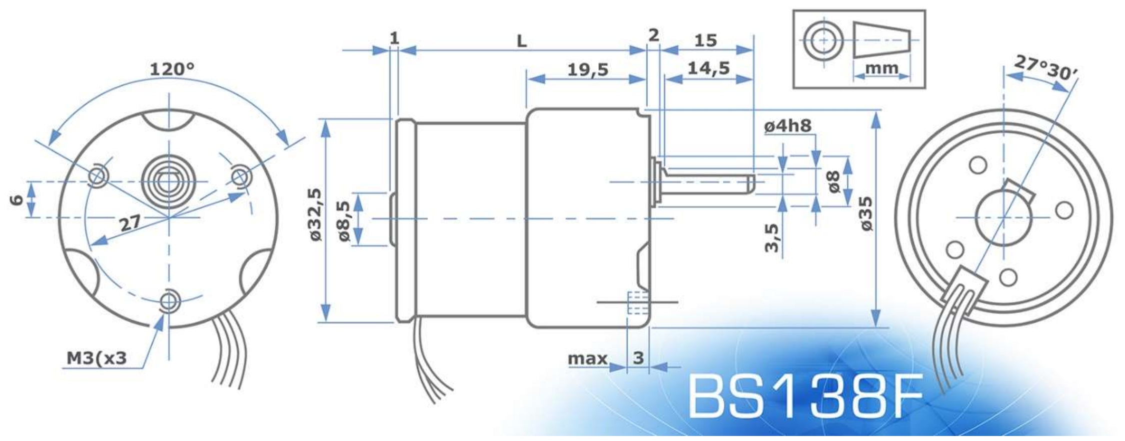


1.5 scheme of the connection in the Arduino shield

CONNECTIONS

ARDUINO DUE	**Pololu Arduino shield**	**DC motor**
5v pin	VDD	
GND	GND	
Digital Pin 52	D2	
Digital Pin 50	M1DIR	
PWM pin 2	M1PWM	
	M1A	Black cable
	M1B	Red cable
3.3V		Brown cable
Digital Pin 22		Yellow cable
Digital Pin 24		Blue cable
GND		Green cable

MOTOR BS138F



1.6 blueprints of the BS138f

General Characteristics at 20°C:

- VDR interference suppression on the collector
- Metal Brushes (Au – Ag – Cu)
- Direction of rotation depending on polarity
- Can be mounted in any position
- Maximum radial shaft load: 20N
- Maximum axial shaft load: 5N
- Temperature range: -20°C/60°C
- Approx weight: 90°C
  Tolerance (+/- 10%)

The BS138F version used in the project also disposes of an encoder, more information is provided in the Encoder part.

SERIAL PORT

The Serial Port is used for communication between the Arduino board and a computer or other devices.

In the case for the Arduino DUE the communication is handled through the USB port and different serial pins.

Due	SerialUSB (Native USB Port only)	0(RX), 1(TX)	19(RX), 18(TX)	17(RX), 16(TX)	15(RX), 14(TX)

2.0 table of the possible way of communications in the Arduino due

The Arduino IDE provides a built-in serial monitor to communicate with the Arduino board, however a specifically designed software was designed in order to provide a more user-friendly experience and to enhance the usability of the system.

<u>Arduino Serial Port setup</u>

Inside the setup function on the Arduino code, the serial port is initialized with the function Serial.begin(). This function Sets the data rate in bits per second (baud) for serial data transmission. It is a value going from 300 to 2000000 baud.

In this project the value chosen is 115200, this value must correspond to the value selected inside visual studio as explained in better detail later.

The other functions used to interact with serial port present in this project Arduino code are:

- if(Serial) – it indicates if the serial port is ready
- Serial.Read() – it reads the incoming serial data and returns the first byte of the available data (or -1 if no data is available). Data type: int.
- SerialAvailable –Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer
- SerialEvent() – Called when data is available. Serial.read() is used to capture this data.
- Serial.Print and Serial.Println – They Print data to the serial port as readable ASCII text, the Println has the difference that it is followed by a carriage return character ('\n')

<u>Visual studio Serial Port setup</u>

In visual Studio to use a Serial port it is necessary to search serial port inside the toolbox in the project window and drag and drop it inside the workspace. To set the serial port in order to read and write data, inside the properties set the Baud Rate so that matches the one used in the Arduino code (115200 in our case), set Dtr Enabled to true (it reset the board when connected).

When Arduino sends data to the computer, the information is handled by a function called ReadSerialDataFromArduino, this function reads the buffer, and it copies it inside a string called data.

The other functions used to communicate through the serial port are:

- serialPort1.read – it reads the buffer
- serailPort1.write – it writes in the buffer; this function is used to send commands to the Arduino.

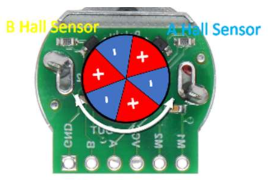
ENCODER

WHAT IS AN ENCODER

An encoder is a sensor that can convert motion to an electrical signal that can be read by some type of control device in a motion control system, such as a counter or PLC. The encoder sends a feedback signal that can be used to determine position, count, speed, or direction.

The encoder used in this project is a magnetic encoder located inside the BS138F motor. A magnetic encoder detects rotational position information as changes of the magnetic field, converts them into electrical signals, and outputs them. The encoder used, disposes of two Hall sensors, that is a magnetic sensor that uses the phenomenon of the Hall effect to output a voltage proportional to the strength of the magnetic field.
The presence of the second hall sensor permits to read not only the speed and the position but also the direction.

The ideal position of the magnetic encoder is that the centre of the rotary shaft, the permanent magnet, and the Hall element are aligned on the same line. Such a configuration is called "Shaft-End configuration".



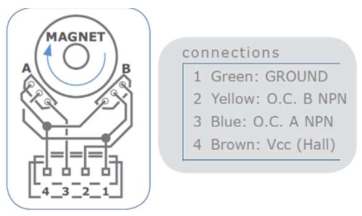
3.0 image of the encoder

The main advantages of this configuration are that it is robust against misalignment by combining the magnet magnetized in the radial direction and the Hall element that detects the strength of the horizontal magnetic field.

DETERMINING THE POSITION, THE SPEED AND THE DIRECTION

The encoder in The BS138F disposes of 4 cables connected as seen in the picture 3.1

The black and red cables are connected to the Pololu Dual MC33926 Motor Driver Shield, that can have two dc motors connected to itself. As explained before the encoder generates an electrical signal, more specifically a train of square pulses that are detected using the pins 22 and 24 in the Arduino.



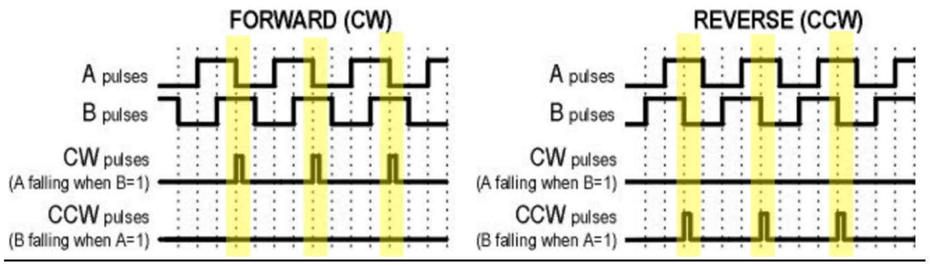
connections
1 Green: GROUND
2 Yellow: O.C. B NPN
3 Blue: O.C. A NPN
4 Brown: Vcc (Hall)

3.1 scheme of the encoder connections

When the motor spins clockwise the generated train of pulses A and B are synchronized in a way that when A is falling from 1 to 0, B is 1.

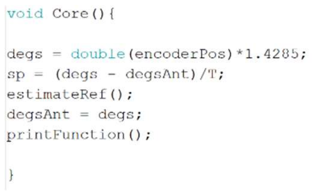
On the other hand when the motor spins Anticlockwise, the two trains of pulses are set up in a way so that when A is falling from 1 to 0, B is never 1.

Analyzing the resulting signal the code shown in figure 3.2 is it possible to determine the position, the speed and the direction of the motor. This thanks to two functions that read the state of the pin 22 and 24 as shown in figure x.x and returns the integer called encoderPos, when the motor spins clockwise encoderPos increases, when anticlockwise decreases.



3.2 image of the comparison between the two train of pulses

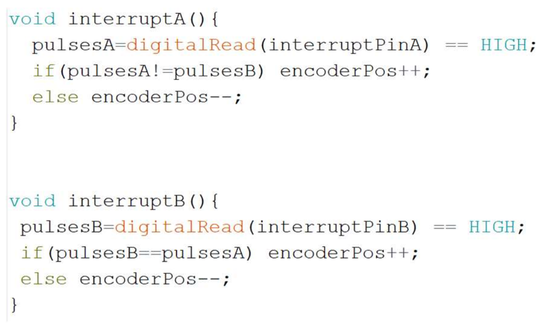
The encoderPos value and sign is enough to determine the position, the speed and the direction of the engine. In the Arduino code these calculations are handled by the Core function:



```
void Core(){

degs = double(encoderPos)*1.4285;
sp = (degs - degsAnt)/T;
estimateRef();
degsAnt = degs;
printFunction();

}
```

3.3 the code in the Core function

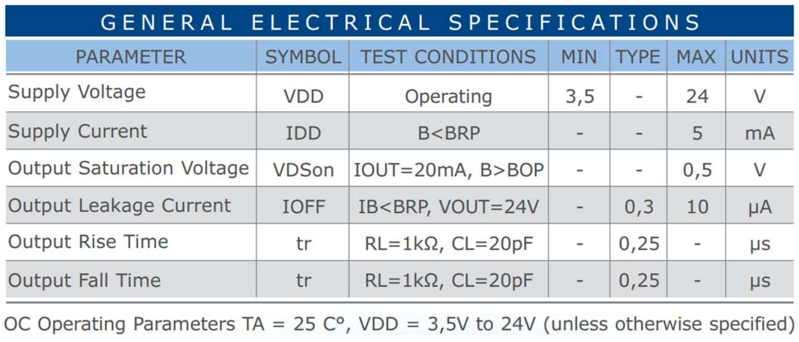
- degs = it is a variable that estimate the position, it does so multiplying the encoderPos value by 1.4285, a constant value of this specific encoder that converts the number of pulses to degrees.
- sp = it is a variable that calculates the speed. It does so by dividing for T, the differnce of the degrees at t = 0 and t = T.
- As said before the direction is determined by the sign of encoderPos, clockwise when positive, AntiClockwise when negative



```
void interruptA(){
  pulsesA=digitalRead(interruptPinA) == HIGH;
  if(pulsesA!=pulsesB) encoderPos++;
  else encoderPos--;
}


void interruptB(){
 pulsesB=digitalRead(interruptPinB) == HIGH;
 if(pulsesB==pulsesA) encoderPos++;
 else encoderPos--;
}
```

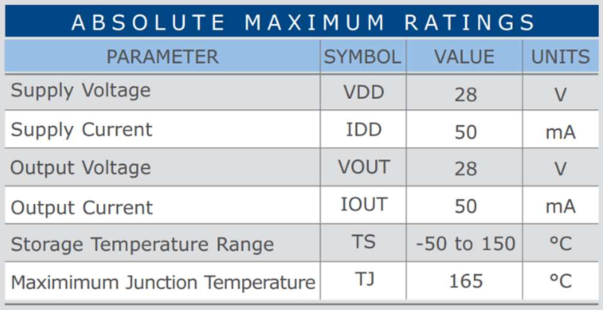
3.4 functions interruptA and interruptB

TECHNICAL DATA



GENERAL ELECTRICAL SPECIFICATIONS						
PARAMETER	SYMBOL	TEST CONDITIONS	MIN	TYPE	MAX	UNITS
Supply Voltage	VDD	Operating	3,5	-	24	V
Supply Current	IDD	B<BRP	-	-	5	mA
Output Saturation Voltage	VDSon	IOUT=20mA, B>BOP	-	-	0,5	V
Output Leakage Current	IOFF	IB<BRP, VOUT=24V	-	0,3	10	μA
Output Rise Time	tr	RL=1kΩ, CL=20pF	-	0,25	-	μs
Output Fall Time	tr	RL=1kΩ, CL=20pF	-	0,25	-	μs

OC Operating Parameters TA = 25 C°, VDD = 3,5V to 24V (unless otherwise specified)



ABSOLUTE MAXIMUM RATINGS			
PARAMETER	SYMBOL	VALUE	UNITS
Supply Voltage	VDD	28	V
Supply Current	IDD	50	mA
Output Voltage	VOUT	28	V
Output Current	IOUT	50	mA
Storage Temperature Range	TS	-50 to 150	°C
Maximimum Junction Temperature	TJ	165	°C

Exceeding the absolute maximum ratings may cause permanent damage. Exposure to all absolute-maximum-rated conditions for extended periods may affect device reliability.

3.5 general characteristics of the encoder in the BS138F motor

REFERENCE

The main objective of this project is to generate reference signals and to have the motor to follow them, so as to create a physical representation of the signal.

In order to do so, it is necessary to be able to generate in the microcontroller so that the motor attached to it can follow them.

In this stage of the project the microcontroller can generate 3 different waveforms, a step signal, a ramp signal and a sinusoidal signal.

The waveforms are generated through the estimateRef function inside the code.

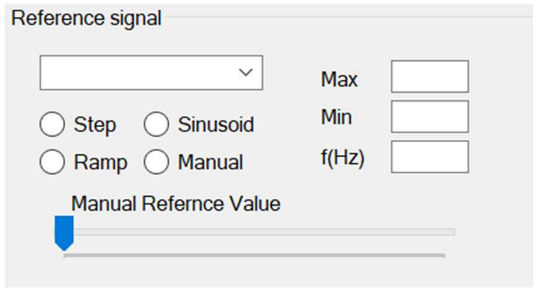
There are four switch cases, one for every different signal plus one for a manual reference as explained below.

In each switch case the time is initialized with the millis() function and the result is a double variable called ref. The ref value is determined using different variables, depending on the waveform, that can be changed inside the SerialEvent function and in the visual studio program as shown in fig 4.1

As briefly hinted before it is possible to generate a manual reference signal. The output value of the variable ref is determined by a trackbar in the visual studio program. In this way it is possible to have the motor follow both position and speed of a manually generated signal.

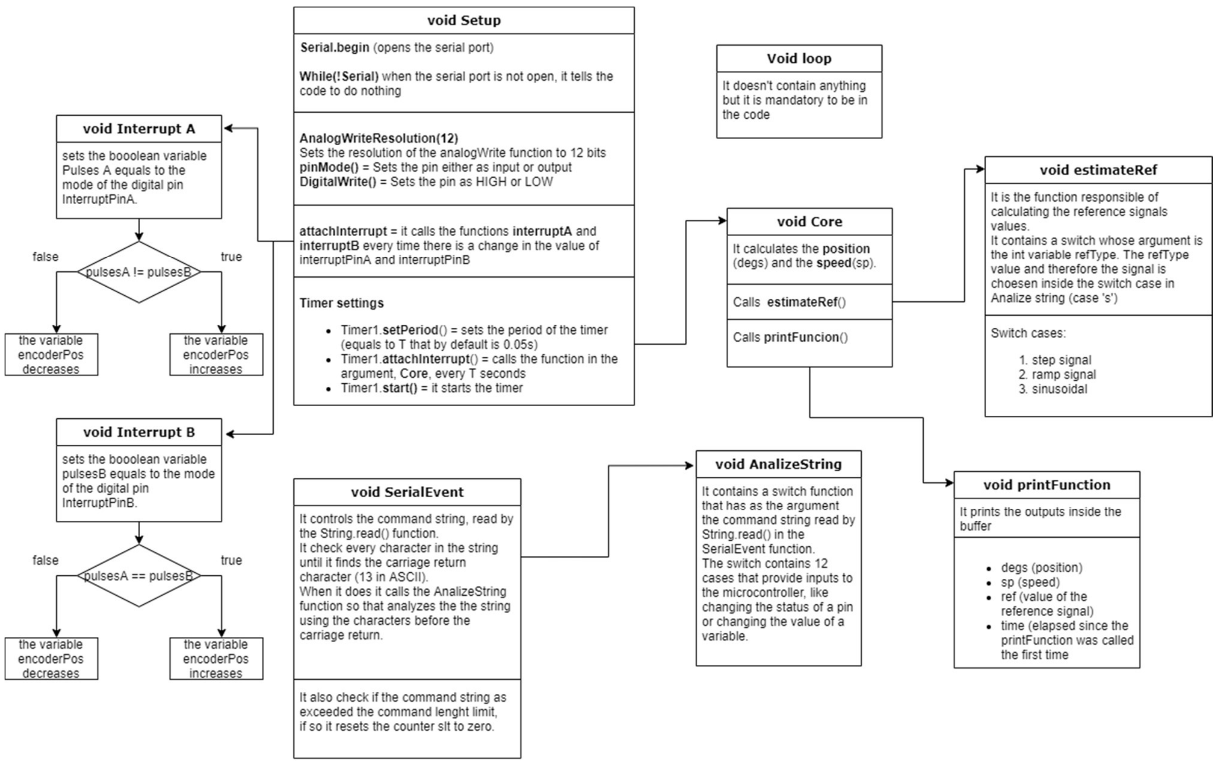
```
void estimateRef(){

  switch(refType){
     case 1: //step
      t = double(millis())/1000 - tIniSig;
      if (t>=1/f) {tIniSig = double(millis())/1000; t = 0;}
      if (t<(1/f)/2)          ref = vMin;
      else                    ref = vMax;
      break;
    case 2: //ramp
      t = double(millis())/1000 - tIniSig;
      if (t>=(1/f)) {tIniSig = double(millis())/1000; t = 0;}
      ref = ((vMax - vMin)/(1/f))*t + vMin;
      break;
    case 3: //sinusoid
      t = double(millis())/1000;
      Amp = vMax - vMin;
      offset = Amp/2 + vMin;
      ref = offset + sin(2.0*PI*f*t);
      break;
    case 4: //Manual
      break;
```

4.0 estimateRef function



4.1 portion of the software for generating and controlling the reference signal

DIAGRAMS


false
pulsesA != pulsesB
true
the variable encoderPos decreases
the variable encoderPos increases
false
pulsesA == pulsesB
true
the variable encoderPos decreases
the variable encoderPos increases

void Setup

Serial.begin (opens the serial port)

While(!Serial) when the serial port is not open, it tells the code to do nothing

AnalogWriteResolution(12)
Sets the resolution of the analogWrite function to 12 bits
pinMode() = Sets the pin either as input or output
DigitalWrite() = Sets the pin as HIGH or LOW

attachInterrupt = it calls the functions **interruptA** and **interruptB** every time there is a change in the value of interruptPinA and interruptPinB

Timer settings

- Timer1.**setPeriod**() = sets the period of the timer (equals to T that by default is 0.05s)
- Timer1.**attachInterrupt**() = calls the function in the argument, **Core**, every T seconds
- Timer1.**start()** = it starts the timer

void Interrupt A

sets the booolean variable Pulses A equals to the mode of the digital pin InterruptPinA.

void Interrupt B

sets the booolean variable pulsesB equals to the mode of the digital pin InterruptPinB.

Void loop

It doesn't contain anything but it is mandatory to be in the code

void Core

It calculates the **position** (degs) and the **speed**(sp).

Calls **estimateRef**()

Calls **printFuncion**()

void estimateRef

It is the function responsible of calculating the reference signals values.
It contains a switch whose argument is the int variable refType. The refType value and therefore the signal is choesen inside the switch case in Analize string (case 's')

Switch cases:

1. step signal
2. ramp signal
3. sinusoidal

void SerialEvent

It controls the command string, read by the String.read() function.
It check every character in the string until it finds the carriage return character (13 in ASCII).
When it does it calls the AnalizeString function so that analyzes the the string using the characters before the carriage return.

It also check if the command string as exceeded the command lenght limit, if so it resets the counter slt to zero.

void AnalizeString

It contains a switch function that has as the argument the command string read by String.read() in the SerialEvent function.
The switch contains 12 cases that provide inputs to the microcontroller, like changing the status of a pin or changing the value of a variable.

void printFunction

It prints the outputs inside the buffer

- degs (position)
- sp (speed)
- ref (value of the reference signal)
- time (elapsed since the printFunction was called the first time

Flowchart made using draw.io (https://app.diagrams.net/)

CONTROLLER

To successfully control our DC motor is it necessary to implement a controller, among all the different strategies to control our system, one of the simplest and used is a feedback control system.

A feedback control system is a system whose output is controlled using its measurement as a feedback signal. This feedback signal is compared with a reference signal to generate an error signal which is filtered by a controller to produce the system's control input.

In our case the input signal can refer both to the position *pos(t)* or the velocity *v(t)*, the same statement is true also for the output.

COMPONENTS OF THE FEEDBACK CONTROL SYSTEMS

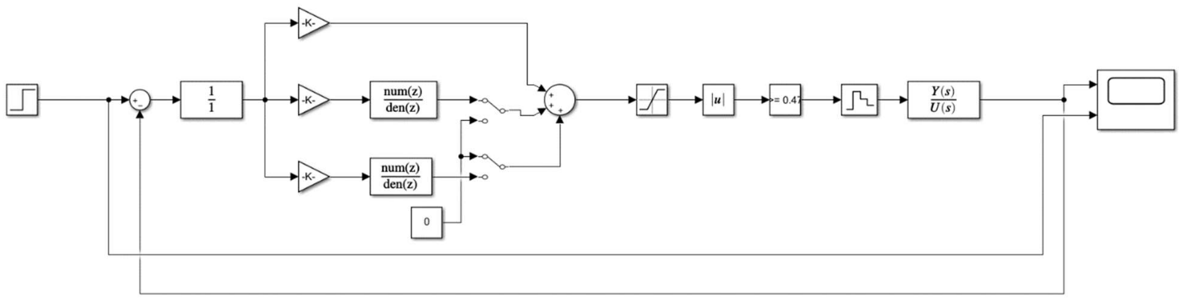


Fig 6.0 Simulink model of the system using a PI controller

The input represents the desired behaviour of our system, this input in our case, is a continuous time signal, and for this reason it needs to be sampled in order to be sent to the DC motor.

To do so a sampler is implemented that captures the value of the signal every Tc (sample time), in our case 0.05 seconds.

The sampled signal then goes through the controller. There are different types of controllers the one used in this system are the PI (Proportional Integral) and the PID (Proportional Integral Derivative) controllers.

$$u(t) = K_P e(t) + K_I \int e(t)dt$$

Fig 6.1 The PI controller produces an output, which is the combination of outputs of the proportional and integral controllers.

$$u(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{\mathrm{d}e(t)}{\mathrm{d}t}$$

Fig 6.2 The PID controller produces an output, which is the combination of the outputs of proportional, integral and derivative controllers.

To have the system behave as desired is it important to tune the controller correctly. In order to do so we need to find the values of the constant **Kp**, **Kd** and **Ki**.

Those calculations were handled using MATLAB PID tuner app. The app to operate correctly needs the open-loop digital transfer function, the type of controller we are using (PD or PID) and the derivative and integral digitalization method. The chosen digitalization method was the Trapezoidal form for the integrator and the Backwards Euler for the derivative.

The PI and PID controller are used to decrease the steady state error without affecting the stability of the control system.

Once the constant values have been found the signal that has been multiplied by the PI or PID controller functions, the signal passes through a saturator and a dead zone block that handle the non-linearities of the system.

The saturator purpose is to assure that the signal is between two fixed values, in our case **+6** and **-6** since the motor operates with a range of 6V.

This is necessary so that, in the case of a malfunction, the saturation prevents damaging the components if a higher voltage input was applied.

The **dead zone** block instead returns zero if the signal value is inside the set interval two (0.47 and -0.47V), it is important because those are the minimum values at which the motor operates.

In the Simulink model the dead zone block is represented like in fig 6.3 because not only the engine has a minimum operating voltage, but it also has a minimum operating rpm value.

Without this ulterior tuning the Simulink model would show a significant delay.

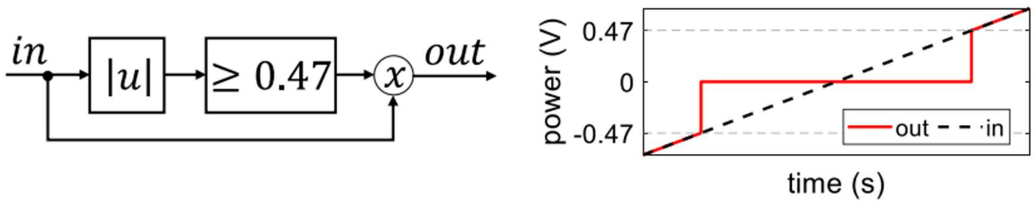


Fig 6.3 correct representation of the dead zone block that takes into account the minimum rpm value of the motor

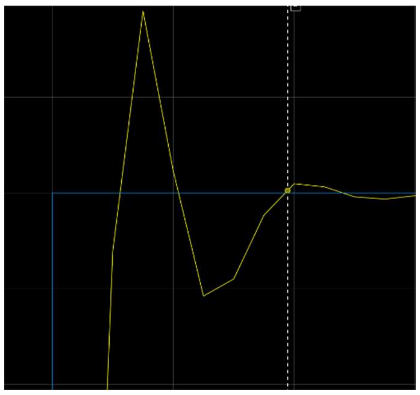
COMPARISON

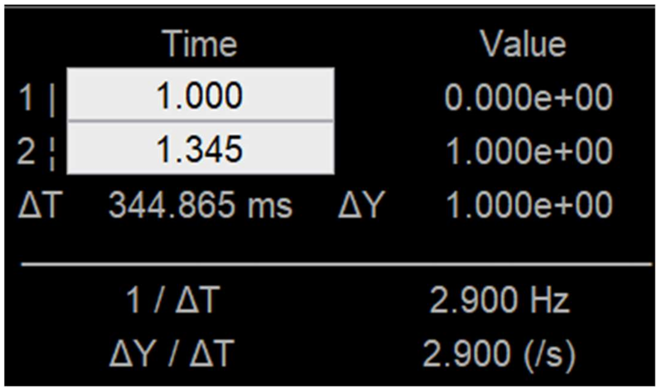
To validate the Simulink model accuracy, a step signal representing the wanted velocity was introfuced as an input both in the Simulink and in the real system.

In the Simulink were simulated two different scenarios, with two different controllers, one a PI controller and one a PID controller.

In both simulated scenarios the delay time measured is around **352ms** a value comparable to the delay in the real system as shown in the table fig 6.5.

This result validates the accuracy of the Simulink theoretical model.





	Time		Value
1	1.000		0.000e+00
2	1.345		1.000e+00
ΔT	344.865 ms	ΔY	1.000e+00
1 / ΔT			2.900 Hz
ΔY / ΔT			2.900 (/s)

fig 6.4 chart and measurements using Simulink shows a delay time of 350ms so that the signal reaches the input step-signal value.

VEL(deg/s)	**REF(signal)**	**TIME(sec)**	**ERROR**
0	0	19.802	0
0	100	19.852	0
28.57	100	19.902	100
114.28	100	19.952	71.43
142.85	100	20.002	-14.28
85.71	100	20.102	-42.85
114.28	100	20.152	14.28
114.28	100	20.202	14.28

Fig 6.5 table showing the real value of the system.
From the moment the ref signal is 100 (t=19.852) to the moment the engine gets to the desired value (t=20.152), <u>the elapsed time is 350ms</u>.

REFERENCES

Courses slides	“SistemIntegrationOne2021”
Arduino main page	https://store.arduino.cc/products/arduino-due?selectedStore=eu
Pololu motor shield	https://www.pololu.com/product/2503
DC motor	https://www.micromotors.eu/en/gear-motors/series-bs138f/
Encoder	https://www.akm.com/us/en/technology/technical-tutorial/basic-knowledge-encoder/magnetic-encoder/
	https://www.encoder.com/
Controller	Bhattacharya, S. K. (2013). Control Systems Engineering, 3/e, 3rd Edition. Pearson.