Institute for Cognitive Systems
Technische Universität München
Prof. Gordon Cheng

Humanoid Sensors and Actuators - Tutorial 4

Brushless and Stepper Motors (120 points)

In this tutorial we will learn:

- How to operate a Brushless DC motor in Stepper mode
- How to operate a Brushless DC motor in two quadrants
- How to control the speed of a Brushless DC motor

Brief reminder about Brushless DC motors

A Brushless Direct Current (BLDC) motor consists of a permanent magnet rotor and of a set of wound stator poles. The electrical energy is converted into mechanical energy by the magnetic attraction forces between the permanent magnet rotor and a rotating magnetic field induced in the poles of the wound stator[1]. BLDC motors are widely used in robotics because of their high efficiency, mainly resulting from the absence of sliding contacts between the rotor and stator. The counterpart of this high yield lies in the complexity of their control strategy, as a state-dependent rotating magnetic field has to be generated using fixed stator coils. Most BLDC motors have a three-phase winding topology with star

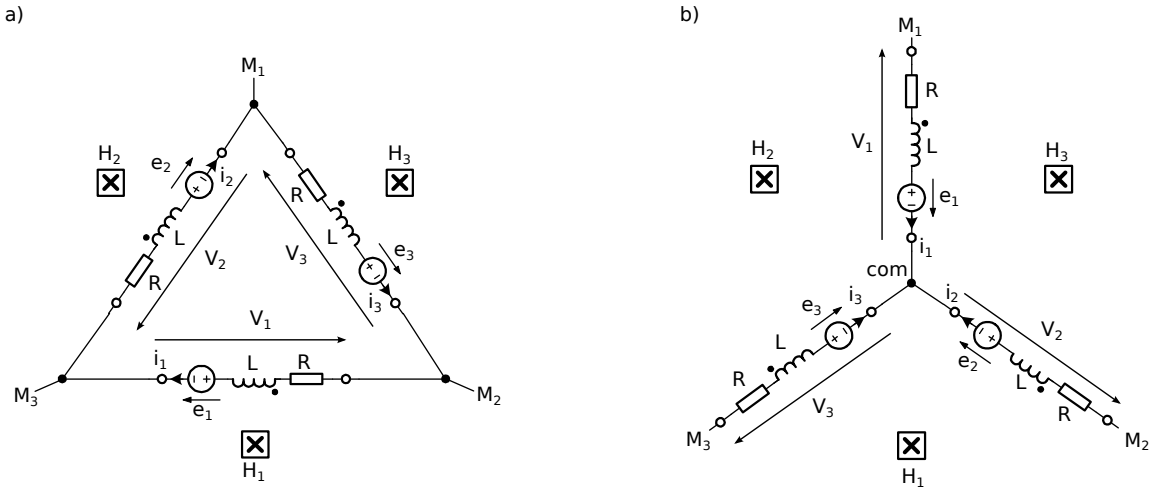

Figure 1: a: Electrical model of a BLDC motor in Delta (D) winding. b: Electrical model of a BLDC motor in Star (Y) winding. The Star topology favours high torques at the expense of the rotational speed.

[1] One could understand it in the same way as a compass placed in a rotating magnetic field: the magnetic needle (rotor) must rotate at the same speed as the outer field.

connection (c.f. Fig.1.b). A motor with this topology is driven by energizing two phases at the same time. The rotor can be made to rotate $\frac{60}{p}$ degrees (where p is the number of pole pairs of the rotor) by simply changing the current path into the coils, as illustrated in Fig.2. A complete rotation of the rotor with one pole pair therefore requires a sequence of six commutation steps, carried out in a well defined order.

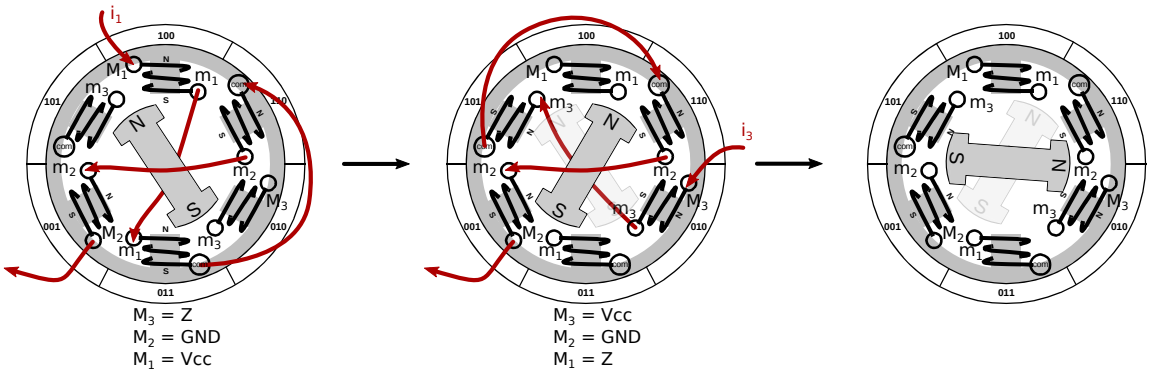

Figure 2: Two steps of a Star (Y) commutation sequence in a BLDC motor. The current circulating in the motor windings is depicted in red. Notice that two phases are energized in the same time, at each step of the commutation sequence.

These commutations are usually achieved using a three phase H-bridge (c.f. Fig. 3), each arm of the bridge being connected to a phase of the motor:

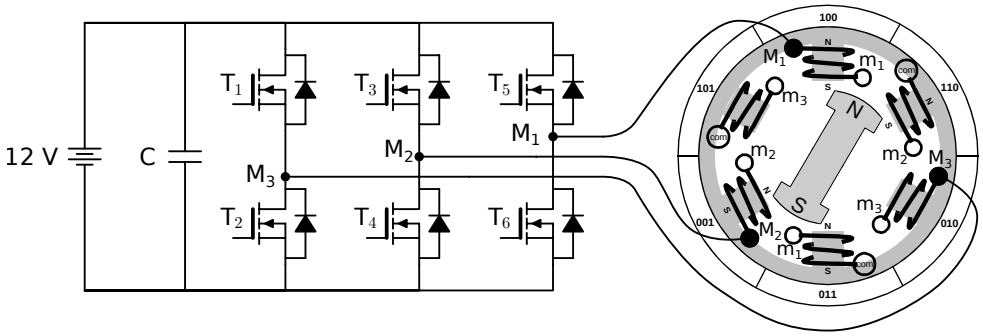

Figure 3: BLDC Motor connected to a three-phase half-H-bridge driver

1 Control of a BLDC Motor in Stepper mode (20 points)

The aim of this part is to investigate the control of a BLDC electric motor in **stepper mode**. Using an integrated triple half-H-bridge driver – as illustrated in figure 3 – we start controlling the rotation of the motor stepwise by changing the phases of the motor according to the following table in **fixed time steps**:

Phase	M1	M2	M3	Wait during
1	−	+	Z	`step_delay()`
2	−	Z	+	`step_delay()`
3	Z	−	+	`step_delay()`
4	+	−	Z	`step_delay()`
5	+	Z	−	`step_delay()`
6	Z	+	−	`step_delay()`

Table 1: Stepper Control

Here "+" means the motor input is connected to the driving voltage, "–" means the motor output is connected to GND and "Z" means the motor output is in high impedance state.

a) Setup

- Download the documentation of the micro-controller and of the motor drivers:
  `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T4/Material/atmega32.pdf`
  `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T4/Material/drv8313.pdf`
- Download the documentation of the motors:
  `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T4/Material/bldc.pdf`
- Download the template project for micro-controller C programs: `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T4/Material/Atmega32Template.tar.gz`

b) Experimental protocol (15 points)

- Read the technical datasheets of the motor and of its controller. Wire up the motor driver to the micro controller (see table 4).
- Connect the motor to the motor driver. So far, you do not need to connect the hall effect sensors.

T.1.1 (10 points) Use an infinite while loop to drive the `INx` and `ENx` input pins of the motor controller with the values from the phase table [2]. You may use the "`_delay_ms()`" routine from the `<util/delay.h>` library.

T.1.2 (5 points) Change the delay between each phase step. What do you observe ?

c) Report (5 points)

R.1.1 (5 points) How reliable is this feed forward commutation ? Discuss in particular the case of small delays.

[2] ***Hint:*** setting `ENx` to zero will result in a high impedance state on the corresponding branch of the bridge.

2 Hall-Sensor Based Control of a BLDC Motor (50 points)

Let's now improve the performance of the current setup by precisely controlling the switching sequence based on the actual rotor position. For this purpose the motor is equipped with three hall-effect sensors. Every state of the rotor corresponds to one of six possible hall sensor configurations in the following table:

Phase	H1	H2	H3	M1	M2	M3
1				$-$	$+$	Z
2				$-$	Z	$+$
3				Z	$-$	$+$
4				$+$	$-$	Z
5				$+$	Z	$-$
6				Z	$+$	$-$

Table 2: Hall-Sensor based Control of a BLDC motor

a) Experimental protocol (35 points)

T.2.1 (15 points) Connect the hall effect sensors to the ATmega32 (see table 4) and configure your software to read the corresponding states:

T.2.1.1 (3 points) The sensor outputs are "open-drain". You should therefore activate pullups on the `PORTC` of the microcontroller, by simply setting the corresponding "`DDRC`" inputs to the correct value.

T.2.1.2 (2 points) Use "`PINC`" to recover the state of the hall effect sensors[3].

T.2.1.3 (10 points) Run the motor in stepper mode using the previous commutation sequence. While the motor is running, capture the state of the Hall effect sensors and fill the table 2 accordingly.

T.2.2 (10 points) Switch the motor phases depending on the current hall state. The motor should now run smoothly in one direction.

T.2.3 (10 points) Adjust the commutation sequence so that the motor runs in reverse direction. Both commutation sequences should be clearly visible within your code.

b) Report (15 points)

R.2.1 (5 points) Compare the obtained results with the feed-forward commutation.

R.2.2 (10 points) Justify the six steps of the reverse switching sequence with a rotor/stator scheme, similar to Fig.2. You can find the *.svg* template in `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T4/Material/motor.svg`

[3] In this case it is useless to use the ADC since the Hall effect sensors only provide digital 0-1 outputs !

3 Speed Control of a BLDC Motor (50 points)

a) Experimental protocol (40 points)

a).1 Open loop speed control

Controlling the rotation speed of a BLDC motor requires the voltage provided by the driver on each phase to be properly adjusted[4]. The most simple and effective way to do so is to use PWM signals as input `INx` of your motor driver (see table 3). The objective of this part is to perform open loop speed control of the BLDC motor by changing the duty cycle of the driver PWM signal with a potentiometer.

- Download the template project for the PC program which uses the FtdiUart Library and creates CSV files using the following link: `http://ics.ei.tum.de/~flo/hsa-lecture/Tutorials/T2/Material/FtdiAdcTemplate.tar.gz`
- You may use the AVR UART library introduced in the "`Atmega32Uart512BitAdderTemplate`" project.

T.3.1 (5 points) Initialize the ADC MCU Block with the following settings[5]:

- Use the AVCC with a capacitor on pin AREF (in our case the capacitor is not strictly needed)
- Use an ADC prescaler of 2 (the clock frequency of the ADC is half of the CPU frequency)
- Use the free running mode
- Use the ADC in the 8 bit mode to read the potentiometer input

T.3.2 (20 points) You are here asked to follow exactly the state table 3. **Be careful:** generating PWM on the `OCx` pin – *as achieved in the previous tutorials* – and connecting the `INx` to this pin will cause the driver to malfunction. You have to modulate each input signal based on the Timer **compare-match** and **overflow** interrupts:

T.3.2.1 (2 points) Create a global `volatile uing8_t` variable. You will use it to change the state of the active `INx` inputs. Here the `volatile` keyword prevent the compiler optimizer to do crazy things.

T.3.2.2 (2 points) Set the timer of your choice in **normal mode**, with prescaler 1

T.3.2.3 (3 points) Enable the timer **Compare Match interrupt** and the timer **Overflow interrupt**

T.3.2.4 (3 points) Write an ISR for the Timer Overflow interrupt in which you set the global state variable to 0

T.3.2.5 (3 points) Write an ISR for the Timer Compare Match interrupt in which you set the global state variable to 1

T.3.2.6 (3 points) Send the value read from the potentiometer, to the output compare register `OCRx` of the timer.

T.3.2.7 (4 points) Changing the state of the global variable in the compare match and overflow ISR will cause it to be modulated as a PWM[6] whose duty cycle will be tuned by the value of the OCR0 register. Use this variable to change the state of your active `INx`.

[4] Don't forget that the rotation speed of a BLDC motor changes linearly with the voltage applied on its phases.
[5] cf. AVR datasheet p.201-218
[6] Changing the state of a PIN in the same ISR will allow you to generate PWM signals on any GPIO of your microcontroller !

T.3.3 (5 points) Using the hall-sensor feedback and a proper timer, estimate the actual rotation speed of your motor[7]. Send the result to your computer using UART. You may want to use a digital low-pass filter in order to obtain a smoother signal.

T.3.4 (10 points) Execute an open-loop speed measurement for PWM duty cycles of 10%, 25%, 50%, 75% and 100%[8]. Plot the results on Matlab in a single graph. The rotation speed should be expressed in [`rpm`]. Please include this graph within your report.

Phase	H1	H2	H3	M1	M2	M3
1				–	PWM	*Z*
2				–	*Z*	PWM
3				*Z*	–	PWM
4				PWM	–	*Z*
5				PWM	*Z*	–
6				*Z*	PWM	–

Table 3: Hall-Sensor based Control of a BLDC motor

b) Report (10 points)

R.3.1 (5 points) In **T.3.2**: what happens when the `OCRx` register reaches extremal values ? How can you solve this problem ?

R.3.2 (5 points) In **T.3.4**: does the rotation speed changes linearly with the duty cycle ? Justify.

[7] Do not forget to take the number of poles of the motor into account.
[8] Note that for each measurement, you may give the motor enough time to reach a steady state.

Motor Driver	Microcontroller	Motor	Other
IN 1	PD 0	-	-
EN 1	PD 1	-	-
IN 2	PD 2	-	-
EN 2	PD 3	-	-
IN 3	PD 4	-	-
EN 3	PD 5	-	-
OUT 1	-	Phase 1	-
OUT 2	-	Phase 2	-
OUT 3	-	Phase 3	-
Hall 1	PC 0	-	-
Hall 2	PC 1	-	-
Hall 3	PC 2	-	-
Reset	-	-	+5V
Sleep	-	-	+5V
VM (2x)	-	-	+12V
-	+5V	+5V	+5V
GND	GND	GND	GND
PGND1-3	GND	GND	GND

Table 4: Connections between the microcontroller, the driver circuit and the motor