3 Servomotors & PWM

During this second session, we will focus on the actuation of the two-wheeled robot through the regulation of each wheel speed.

3.1 Servomotors

The two-wheeled robot can move along a flat surface. Two servomotors can independently drive the two side wheels of the robot, to make it advance along a straight line or to make it turn. When the two wheels rotate at exactly the same angular velocity ($\omega_1 = \omega_2$, see Fig. 1) and direction, the robot moves straight forward (or backward, according to the rotation direction). Instead, when one wheel rotates faster than the other one, the robot turns to the side of the slower turning wheel (see Fig. 1, for a complete overview).

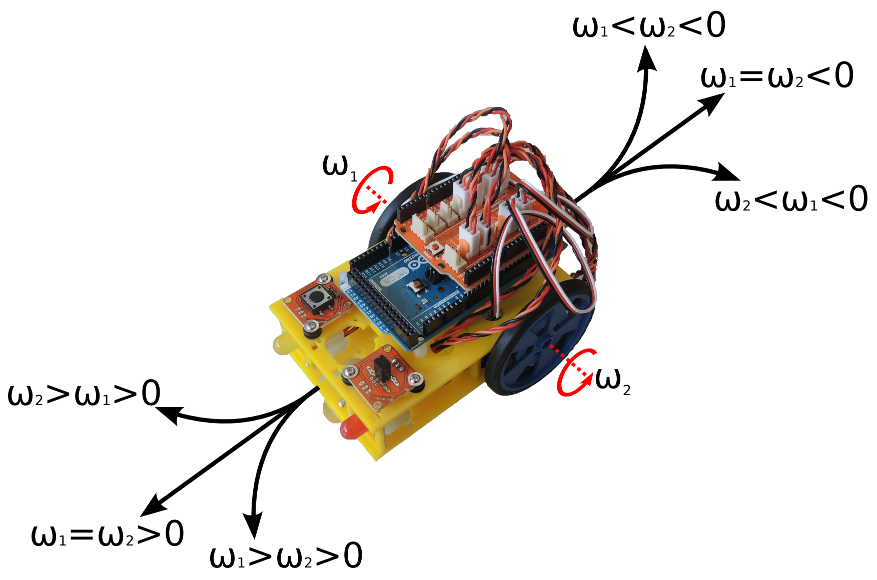

Figure 1: *Driving directions.*

A servomotor[1] is a rotary actuator that combines in a single device a motor, an encoder and a feedback loop control unit. The encoder provides a position and/or speed feedback and the control unit compares this feedback

[1]http://en.wikipedia.org/wiki/Servomotor and http://www.princeton.edu/~mae412/TEXT/NTRAK2002/292-302.pdf

with the incoming signal from the receiver thus controlling the current flowing through the motor.

These three components together aim to ensure a precise control of the output shaft, in terms of angular position and/or velocity, depending on the specific application. The control of the servomotor, in general, can rely on a single wire that carries the signal with encoded control parameters.

Each servomotor is connected to the Arduino board by means of a three-wire ribbon cable. The black wire is the reference ground, the red wire is the power supply ($V_{cc} = 5V$), while the white wire is the control signal that determines the target angular velocity. The control signal is periodic with a period of $20\,ms$ ($50\,Hz$). Within each period a pulse of variable duration encodes magnitude and direction of the target angular velocity[2]. Specifically, a 1.5 ms pulse corresponds to an output velocity of zero (the servomotor shaft holds the current position). A longer pulse indicates a positive angular velocity (forward rotation), while a shorter pulse indicates a negative angular velocity (backward rotation). The magnitude of the angular velocity is proportional to the absolute difference between the pulse duration and the reference value 1.5 ms. It is important to notice that the pulse duration has to be in the range $[1.0, 2.0]$ ms.

Fig.2 summarizes the different cases.

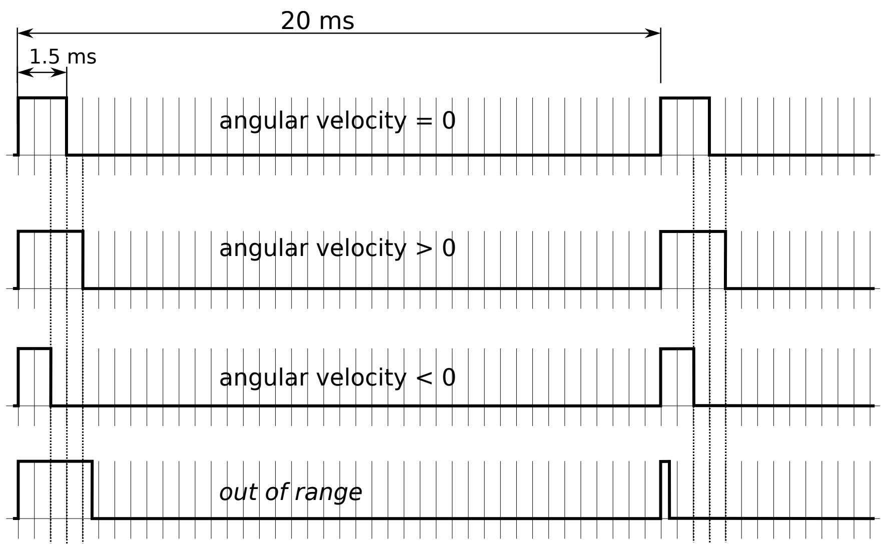

Figure 2: *Pulse duration and corresponding angular velocity.*

[2]see http://en.wikipedia.org/wiki/Servo_control for further information about the control. The web page explains how to control the angular position of a servomotor. However, same considerations are also valid for controlling the angular velocity

3.1.1 Preparation Task: PWM signal to control the servomotor

A way to generate a suitable control signal for the servomotor exploits the internal timer/counters of the MCU to output a Pulse-Width Modulation (PWM[3]) signal to an external pin. Even though the servomotor control signal is not a pure PWM, as the pulse width is limited within a range shorter than the base period and can not be null, using the internal timer/counter of the MCU reduces the overhead on the main processor that can hence work in parallel to the PWM generation.

Chapter 20 of the $ATMega2560$ datasheet explains how to program the 8-bit Timer/Counter2 to generate a PWM signal and which different settings are available.
The Timer/Counter2 requires an input clock to work that determines the basic counting frequency and that can be connected to different sources, as shown in Fig.20-1 of the datasheet. For our purpose, we select the MCU clock ($clk_{I/O}$ in the datasheet) as source. As very often the MCU clock has a frequency higher than required (the Arduino MCU works at $16\,MHz$). That can be scaled down with a prescaler of a power of two (1, 8, 32, 64, 128, 256 or 1024). The scaled down frequency will not have exactly the needed value of $50\,Hz$ (period of $20\,ms$), due to integer divisions. Nevertheless, the servomotors work fine also with periodic signals that approximate the target value.
The modes of operation describe the behaviour of the Timer/Counter: it is possible to set whether the output pin works in inverted mode, whether the timer gets cleared on a compare match (see output compare register), and whether the timer counts up or down.

3.1.2 Task: program the MCU to generate PWM signals

Create a new C-module and call it "servo". The module should implement the following three functions:

- $servo\_init()$, to initialize the timer/counter2;
- $servo\_left(int\, v)$, to set the angular velocity of the left wheel;
- $servo\_right(int\, v)$, to set the angular velocity of the right wheel.

There are four registers to be set during the initialization: TCCR2A, TCCR2B, OCR2A, OCR2B. Refer to the datasheet on how to configure them. While configuring TCCR2A and TCCR2B, it is better to temporary disable the interrupts to prevent accidental generation of unwanted signals: two functions $cli()$ and $sei()$ are available with the AVR-libc to disable and enable the global interrupts.

[3] `http://en.wikipedia.org/wiki/Pulse-width_modulation`

The two functions $servo\_left(int v)$ and $servo\_right(int v)$ set the corresponding Output Compare Register to control the pulse duration of the output control signal. The input parameter v of each function is an integer in the range $[-100, 100]$ whose absolute value is proportional to the angular velocity of the corresponding servomotor and whose sign defines the direction of rotation. As overall, these functions map the input range $[-100, 100]$ to the output range $[1.0, 2.0]$ *ms* which corresponds to the pulse duration of the control signal, thus the angular velocity and rotation direction of the servomotor. Notice that, as the two servomotors are symmetrically mounted, one of the two functions has to invert the direction of rotation.

To test your code initialize the servo module from *main.c* and call $servo\_left(int\, v)$ and $servo\_right(int\, v)$ with different values. Check whether the servos work as expected. You can even implement a function in main.c, which would let you change the speed of the servomotors via the serial terminal Putty after pressing the push button.
It could happen that rapid changes of wheel speed result in the reset of the MCU: this is due to *inrush current* of the servomotors. To mitigate this effect you can program a proportional control to slowly switch on the servos to progressively reach the target speed. This can be done, for example, by slightly incrementing (or decrementing, depending on the turning direction) the speed of the servo of a small amount ΔS at every time step Δt. You can use a delay function to define the duration of a time step. You should empirically find the two parameters ΔS and Δt which prevent the robot from resetting but which also minimize the *responsiveness* of the servos, as this will be crucial in the last assignment, where the robot will be controlled via the joystick.
The following table reports the connections between the servomotors and the Arduino board:

Servomotor	**ATMega2560 Pin**	**Output Compare Register**	**TinkerKit Conn.**
Right wheel (ω_1)	Port B pin 4 ($PB4$)	OC2A	O1
Left wheel (ω_2)	Port H pin 6 ($PH6$)	OC2B	O2

Table 1: *Connection between the ATMega2560 and the two servomotors.*