



TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies ■

Remote setting up the PID controller parameters based on visual observation of controlled motion

Diploma thesis

Study program: N2612 Electrical Engineering and Informatics

Specialization: Mechatronics

Author: **B. Sc. Dmitry Kochubey**

Supervisor: Doc. Ing. Petr Tůma, CSc.

Consultant: Doc. Ing. Denis A. Kotin, CSc.



DIPLOMA THESIS ASSIGNMENT

(PROJECT, ART WORK, ART PERFORMANCE)

First name and surname: **B.Sc. Dmitry Kochubey**
Study program: **N2612 Electrical Engineering and Informatics**
Identification number: **M13000295**
Specialization: **Mechatronics**
Topic name: **Remote setting up the PID controller parameters based on visual observation of controlled motion**
Assigning department: **Institute of Mechatronics and Computer Engineering**

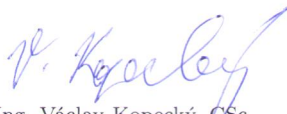
R u l e s f o r e l a b o r a t i o n :

1. Studying the basic principles and properties of controlled motion systems.
2. Studying the microcontrollers and sw equipment for controlling of motion systems.
3. Development of PID motion controller with internet/USB/RS232 interface for setting up PID parameters and desired motion position.
4. Construction based on development from previous topic.


Scope of graphic works: **In respect to the documentation needs**
Scope of work report
(scope of dissertation): **c. 40–50 pages**
Form of dissertation elaboration: **printed/electronical**
Language of dissertation elaboration: **English**
List of specialized literature:

- [1] **Atmel microcontroller manuals**
- [2] **Rodić A. D., AUTOMATION & CONTROL - Theory and Practice, India, In-Teh, 2009, 360 pages.**
- [3] **Encyclopedia of Automated process control systems - URL:
<http://www.bookasutp.ru>**
- [4] **Vostrikov A. S., Frantsuzova G.A. The theory of automatic control. Moscow, Higher School, 2006, 365 pages.**
- [5] **Pankratov V. V., Automatic control of electric drives: DC drives coordinate regulation. Part 1. Novosibirsk, NSTU, 2013, 200 pages.**

Tutor for dissertation: **doc. Ing. Petr Tůma, CSc.**
Institute of Mechatronics and Computer Engineering
Dissertation Counsellor: **doc. Ing. Denis Kotin, CSc.**
NSTU Novosibirsk, Russia
Date of dissertation assignment: **29 November 2013**
Date of dissertation submission: **16 May 2014**


prof. Ing. Václav Kopecký, CSc.
Dean




doc. Ing. Milan Kolář, CSc.
Department Manager

Liberec, dated: 29 November 2013

Declaration

I hereby certify that I have been informed the Act 121/2000, the Copyright Act of the Czech Republic, namely § 60 - Schoolwork, applies to my master thesis in full scope.

I acknowledge that the Technical University of Liberec (TUL) does not infringe my copyrights by using my master thesis for TUL's internal purposes.

I am aware of my obligation to inform TUL on having used or licensed to use my master thesis; in such a case, TUL may require compensation of costs spent on creating the work at up to their actual amount.

I have written my master thesis myself using literature listed therein and consulting it with my thesis supervisor and my tutor.

Concurrently I confirm that the printed version of my master thesis is coincident with an electronic version, inserted into the IS STAG.

Date:

Signature:



Acknowledgments

I would like to express my sincere appreciation to my advisory committee: my supervisor Doc. Ing. Petr Tůma, CSc, from Technical University of Liberec and my consultant Doc. Ing. Denis Kotin, CSc, from Novosibirsk State Technical University for their guidance, support and patience. They helped me solve most of challenges in my master thesis by giving their precious advice.

My gratitude also goes to Mr. Pavel Herajm. This person gave me technical support during the plant design and the assembling process.

Furthermore, I express my strong appreciation to MPAM Tempus program and European commission for the financial support of this beneficial project.

My special thanks go to my mother, my wife, and my daughter for their continuous support during the time I wrote my diploma thesis.

Abstract

The main project goal was to develop a laboratory stand for visual observation of changes in the work part motion caused by different settings of PID controller parameters.

The whole scope of work was divided into several steps in order to reach the goal. The first step was to learn the information about the topic in the field of automation control theory and propeller theory. The second step was to select components for hardware realization of the drive system and develop corresponding printed circuit board (PCB). In the step three, I wrote firmware for the developed electric drive. As step four I created a program for a personal computer (PC) that can control the drive by a set of commands and perform drive remote control using internet.

Keywords

Electric drive, pulse-width modulation, microcontroller, AVR, Atmega8, UART, SPI, USB, RS232, AS5045, magnetic rotary encoder, remote control, position control, PID, PCB, internet, Skype.



Table of contents

Declaration	3
Acknowledgments	4
Abstract	5
Keywords	5
Table of contents	6
List of used figures	8
List of abbreviations	10
1. Introduction	11
2. Plant description	12
2.1. Mechanical part overview	12
2.2. Pulse-width modulation	14
2.3. DC motors	15
3. Control system description	18
3.1. PID and PI controllers	18
3.2. System overview	23
3.3. Description of components used in the project	28
3.3.1. Atmel AVR ATmega8 microcontroller	28
3.3.2. Synchronous-buck mosfet driver TPS2834D	34
3.3.3. Dual operational amplifier LM358N	35
3.3.4. Magnetic rotary encoder AS5045	37
3.3.5. FT232RL communication module	38
4. Software description	40
4.1. Microcontroller firmware	40
4.2. Remote control via ProJet	41



Table of contents

4.3. Remote control via Skype.....	43
5. Conclusion.....	45
6. Literature.....	46
7. CD ROM.....	48
Appendix 1.....	49
Appendix 2.....	52
Appendix 3.....	53
Appendix 4.....	72



List of used figures

Figure 1. Plant overview	12
Figure 2. Mechanical part	13
Figure 3. Working principle.....	13
Figure 4. Pulse width modulation [3]	14
Figure 5. Brushed DC motor parts	15
Figure 6. Two-pole DC motor operation [4, p. 2]	16
Figure 7. Torque-speed and current-torque relations [4, p. 3]	16
Figure 8. DC motor block diagram	17
Figure 9. DC motors MIG 400.....	17
Figure 10. Basic block of PID controller.....	18
Figure 11. Effect of P term [6].....	19
Figure 12. Effect of I term [6].....	19
Figure 13. Effect of D term [6]	20
Figure 14. Basic block of PI controller.....	22
Figure 15. Drive board.....	23
Figure 16. Encoder board.....	24
Figure 17. Power supply converter.....	25
Figure 18. General view of the laboratory stand.....	25
Figure 19. Control system block diagram.....	26
Figure 20. Control closed loop of the armature current	26
Figure 21. Closed loop of the work part position control.....	27
Figure 22. Microcontroller ATmega8 in DIP package [1]	28
Figure 23. ATmega8 block diagram [1]	30
Figure 24. ATmega8 pin configuration [1].....	30
Figure 25. UART interface [7].....	32



List of used figures

Figure 26. UART message frame.....	33
Figure 27. SSI(SPI) interface [8].....	33
Figure 28. SSI(SPI) shift registers [8].....	34
Figure 29. TPS2834D synchronous-buck MOSFET drivers.....	34
Figure 30. Electric circuit of the motor power unit.....	35
Figure 31. Dual operational amplifiers LM358N.....	36
Figure 32. Electric circuit of the current measurement node.....	36
Figure 33. Magnetic rotary encoder AS5045.....	37
Figure 34. Encoder operation [11, p. 1].....	37
Figure 35. Synchronous serial interface [11, p. 13].....	38
Figure 36. PWM output signal [11, p. 16].....	38
Figure 37. FT232RL communication module.....	38
Figure 38. Electric circuit of the communication module.....	39
Figure 39. Propeller project firmware in CodeVisionAVR.....	40
Figure 40. ProJet interface.....	41
Figure 41. Skype logo.....	43
Figure 42. ProJet-Skype platform.....	43
Figure 43. Drive electric circuit.....	49
Figure 44. Top copper of the drive PCB.....	50
Figure 45. Bottom copper of the drive PCB.....	50
Figure 46. Encoder electric circuit.....	51
Figure 47. Encoder PCB.....	51



List of abbreviations

USB – Universal Serial Bus

PC – Personal Computer

RISC – Reduced instruction set computing

MIPS – Microprocessor without Interlocked Pipeline Stages

IC – Integrated Circuit

LED – Light-Emitting Diode

MOSFET – Metal–Oxide–Semiconductor Field-Effect Transistor

UART – Universal Asynchronous Receiver Transmitter

SPI – Serial Peripheral Interface

MCU – Multipoint Control Unit

1. Introduction

The main goal of my master thesis is to develop and assemble the control system of electric motors of the laboratory stand with a possibility of remote control via internet. In order to achieve this goal, I had to design an electrical system, a mechanical system, microcontroller firmware and a remote control application.

The electrical system was built in the trial version of Proteus software. I took all background information about microcontroller based systems from [1]. Finally, the electrical part consists of two printed circuit boards (PCBs). One of them is a drive board and another one is the board of magnetic rotary encoder. The plant has an external power supply in a range of seven volts of direct current. Communication with supervisory-application that is located on the PC is realized by a USB-UART bridge. A special set of commands was developed for carrying out remote control function via internet. I have decided to use Skype instant messaging client that is a freemium and the most popular service as a client application. For this purpose I added several Skype monitoring algorithms into the supervisory-application.

Mechanical part of the project is based on DC motors with propellers, MayTec profiles and accessories. I selected DC motors due to the easy methods of control, power supply voltage and low price.

Software is divided into two parts. The first part works on the PC, created in the professional version of Visual Studio 2013. The second part works on the microcontroller side that is controlling DC motors and position of the stand work part. Firmware was written in the evaluation version of CodeVisionAVR environment.

2. Plant description

2.1. Mechanical part overview

In this chapter I would like to describe the elements of mechanical part and work principle of the laboratory stand. Figure 1 illustrates common view of the plant. It based on industrial profiles and accessories produced by MayTec Company which are often used for machines and installations prototyping.

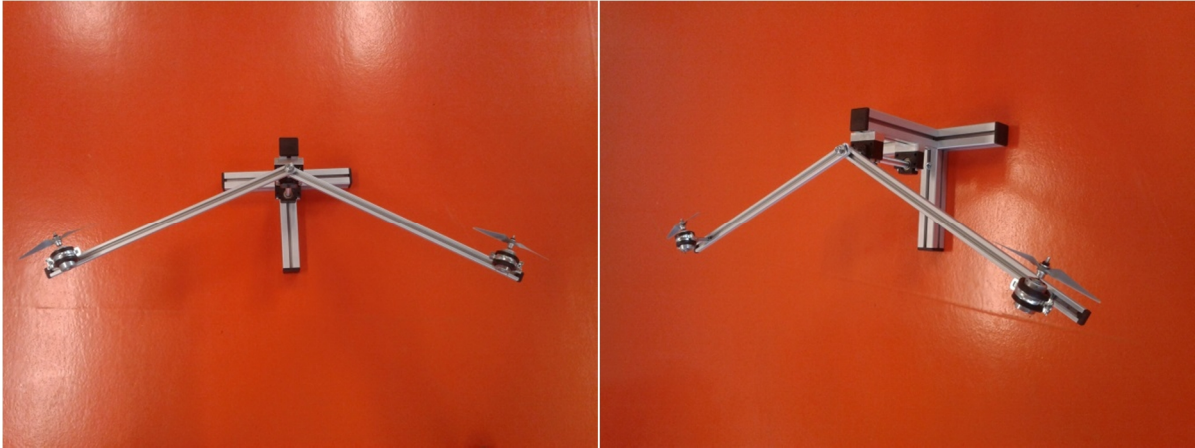


Figure 1. Plant overview

As it could be seen the stationary part consists of three E4-slot profiles. Two of them are used as support. At the fourth profile one pair of bearings with moving shaft is installed. This place is a point of interaction between stationary and working part. The working part includes two rails which are connected to the moving shaft. The angle between rails can be changed by adjusting a female screw in the point of rail-to-shaft connection. Motor mounting units with installed motors are located in the end of each rail. Motors are equipped with APC 6x4E propellers. Components of the plant are shown in the figure 2.

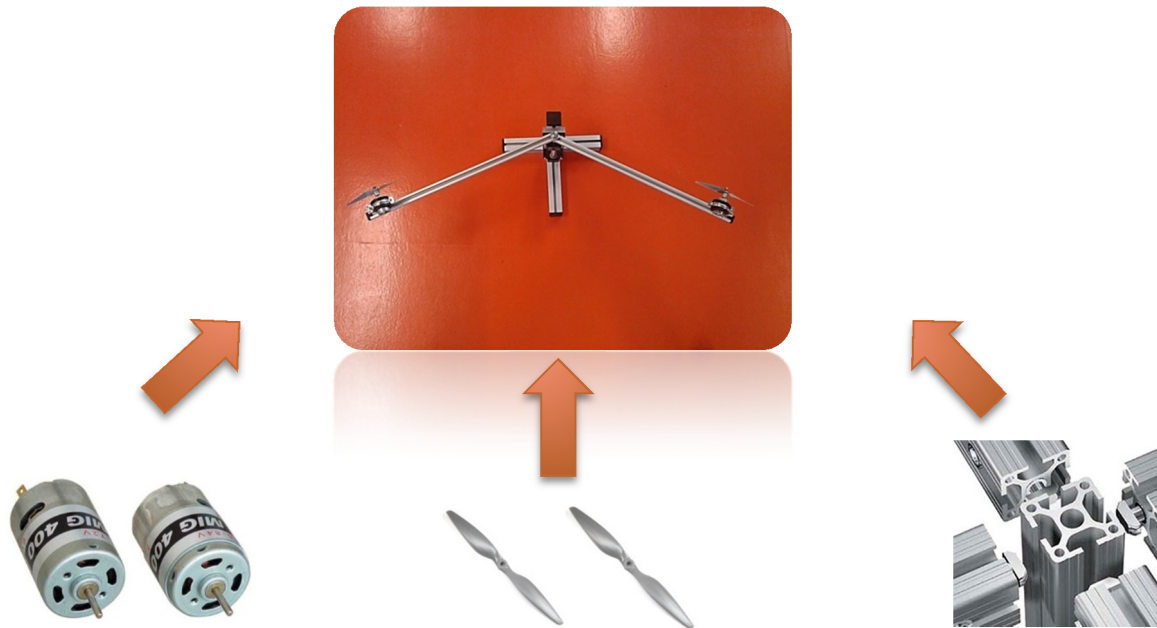


Figure 2. Mechanical part

Working principle of the laboratory stand is described as follows. Propellers during the motor shaft rotation produce the aerodynamic force. It enables the work part to move clockwise or counterclockwise relative to the axis of rotation. Therefore, by adjusting the speed of the motors we can control the position of the working part of the plant. The illustration of work is provided in the figure 3.

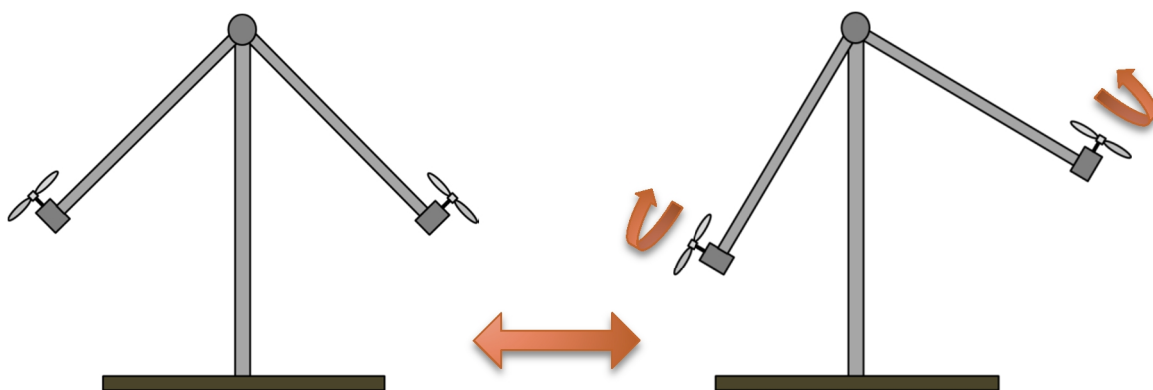


Figure 3. Working principle

2.2. Pulse-width modulation

To control the rotating speed of the plant's DC motors I used a pulse-width modulation (PWM) method. A pulse-width modulation technique is kind of a modulation technology which uses a rectangular pulse, whose pulse width during the period of the signal modulates an average waveform value [2].

The duty cycle is the percentage of the positive state compared to the period of the signal. It is calculated by the following formula:

$$Period(T) = \frac{1}{Frequency(F)}. \quad (2.1)$$

The idea of the method can be described in simple words. A PWM signal (with pulse width the same as period of a duty cycle) would modulate the maximal value of the voltage. If we decrease this width, the power delivered to the load will be also decreased as illustrated in the figure 4.

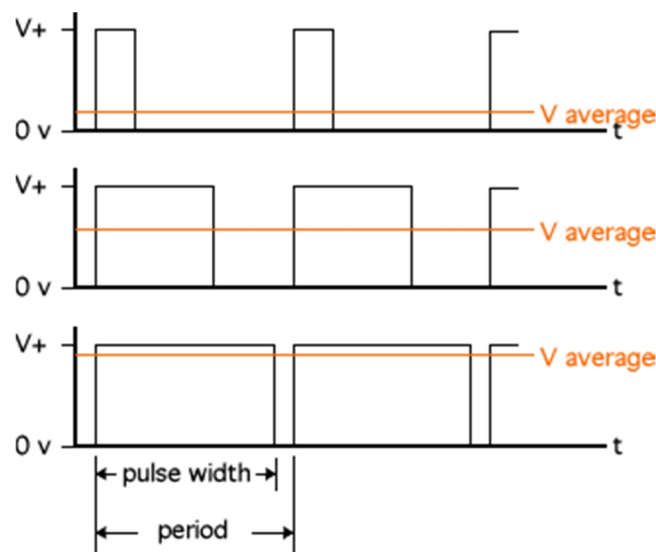


Figure 4. Pulse width modulation [3]

The total power delivered to the connected load each time, is the area under the positive state of the PWM. It is clearly seen that by altering the pulse width, we can alter the power delivered by the supply. Due to the fact that the wave form is a square wave, the power supplied each time is calculated by:

$$P_{delivered} = P_{supplied} \cdot pulse\ width; \quad (2.2)$$

The PWM principle applying to the DC motor looks as follows. If we supply a DC motor with nominal voltage, the motor will run at its maximal speed. However, fast switching between on and off state by applying a periodic signal does not let the motor reach the nominal voltage and, as a result, the maximal speed. It means the motor speed will be proportional to the average time of the motor on state.

In my control system speed control is carried out using ATmega8 PWM channels. This microcontroller is equipped with three timers to generate PWM signals at three channels. For my project I used only two of them as signal generators. The flexible setting of channels allows the user to generate a wide variety of PWM signals to control the speed of a DC motor.

2.3. DC motors

A Direct Current (DC) motor is a DC electric machine which converts electrical energy of direct current into mechanical energy. In my project I used brushed DC motors that consist of several parts such as rotating shaft with windings (armature), bearings, stator with magnet, brushes, commutator, stator case and motor wires (see figure 5).

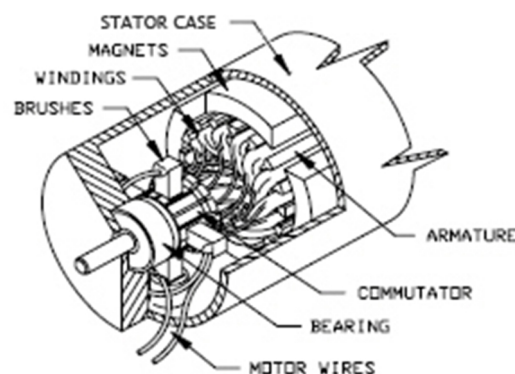


Figure 5. Brushed DC motor parts

Principle of two-pole DC motor operation is shown in the figure 6. The torque is produced when the coil is powered and generates a magnetic field around the armature. The same polarity poles are pushed away and opposite polarity poles are attracted. This phenomenon causes rotation.

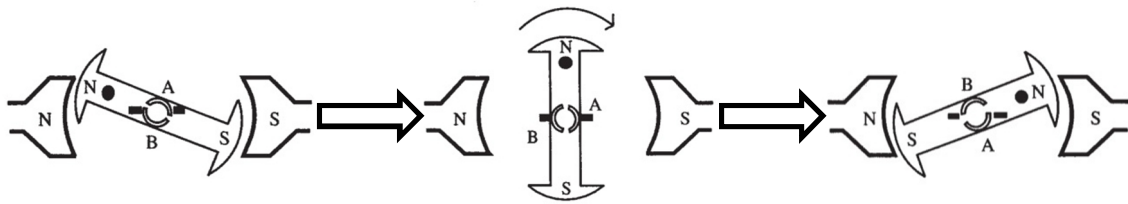


Figure 6. Two-pole DC motor operation [4, p. 2]

Brushed DC motors with permanent magnets have linear torque-speed and current-torque relations as illustrated in the figure 7. It makes them perfect control objects.

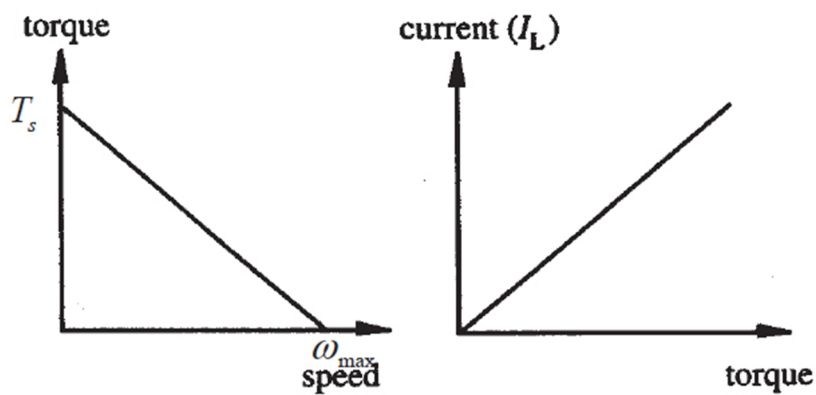


Figure 7. Torque-speed and current-torque relations [4, p. 3]

T_s – The starting torque is the maximum torque produced by motor at zero speed.

ω_{max} – The no-load speed is the maximum speed that the motor can reach without load.

Differential equations of the DC motor are provided below:

$$U = RI + L\dot{I} + U_m; \quad (2.3)$$

$$U_m = k\Phi\omega; \quad (2.4)$$

$$m_n = k\Phi I; \quad (2.5)$$

$$m_n = m_\omega + J\dot{\omega}. \quad (2.6)$$

Based on these equations we can obtain DC motor block diagram that is shown in the figure 8.

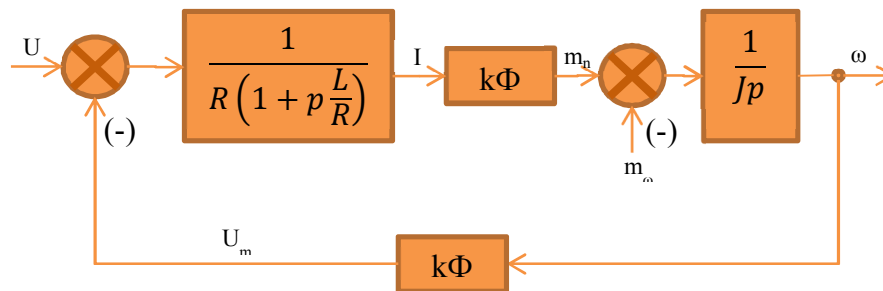


Figure 8. DC motor block diagram

MIG 400 is a brushed DC motor with permanent magnets. General view of the motor is provided in the figure 9.



Figure 9. DC motors MIG 400

It has the following technical parameters [5]:

- Recommended voltage: 3.6 to 9.6 V;
- Nominal current: 0.55 A;
- Armature resistance: 0.286 Ohm;
- No-load speed: 15,700 rev / min at 7.2 V;
- Max speed: 17,000 rev / min;
- Maximum efficiency of 73 %;
- Motor velocity constant Kv: 2189;
- Shaft diameter: 2.3 mm;
- Motor diameter: 28.9 mm;
- Length: 38mm;
- Weight: 80g.

3. Control system description

3.1. PID and PI controllers

A proportional-integral-derivative controller (PID controller) is a control loop feedback mechanism (controller) widely used in industrial control systems (Programmable Logic Controllers, SCADA systems, Remote Terminal Units, etc.) [6].

Formula of analog PID controller is as follows:

$$u(t) = P(t) + I(t) + D(t) = K_P \cdot e(t) + K_I \cdot \int e(\tau) d\tau + K_D \cdot \frac{de(t)}{dt}; \quad (3.1)$$

where $e(t)$ represents the difference between the desired and the actual position.

Block diagram which is shown in the figure 10 describes the formula of analog PID controller.

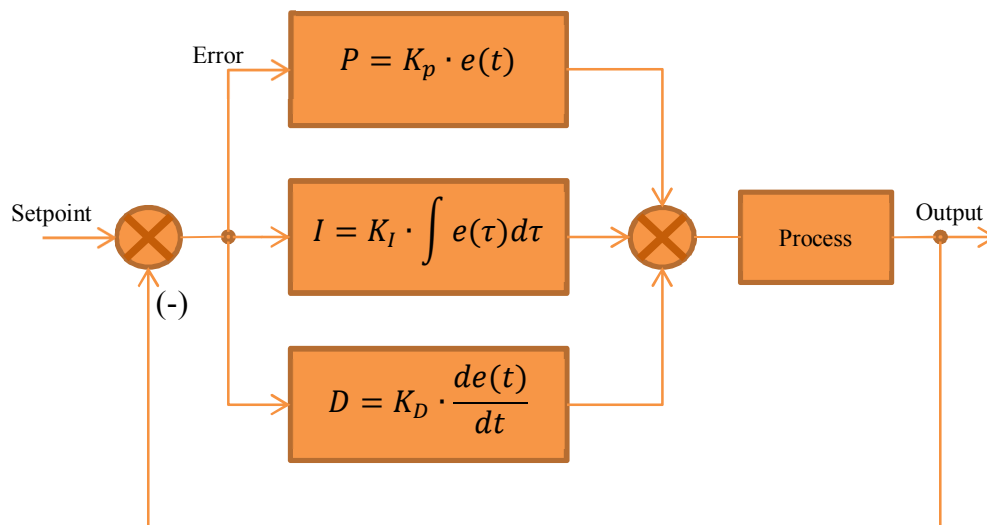


Figure 10. Basic block of PID controller

Formula of digital PID is:

$$\begin{aligned} u(k) &= P(k) + I(k) + D = \\ &= K_P \cdot e(k) + K_I \cdot \sum_{i=0}^k h \cdot e(i) + K_D \cdot \frac{e(k) - e(k-1)}{h}; \end{aligned} \quad (3.2)$$

where h is the sampling time.

The controlled value is held at the desired position by applying a restoring force to the process (plant) that is proportional to the position error, adding the integral part of the error and the derivative of the error.

Physical meanings of PID terms are described as following. The proportional term provides a restoring force proportional to the position error, just as a spring obeying Hooke's law does. The integration term provides a restoring force that grows with time, and thus ensures that the static position error is zero. The derivative term provides a force proportional to the rate of change of position error. It acts just like viscous damping in a damped spring and mass system. Effect of each PID term is respectively shown in figures 11, 12 and 13.

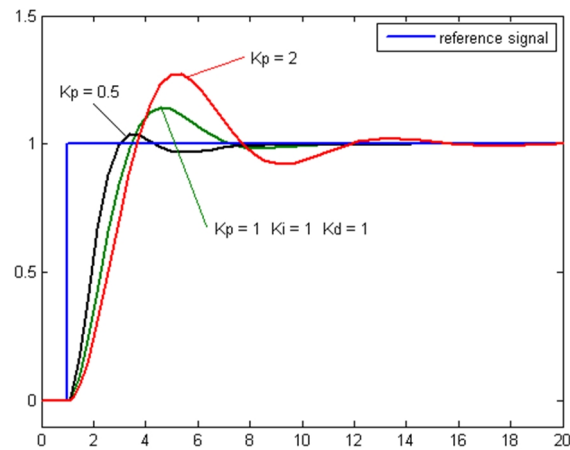


Figure 11. Effect of P term [6]

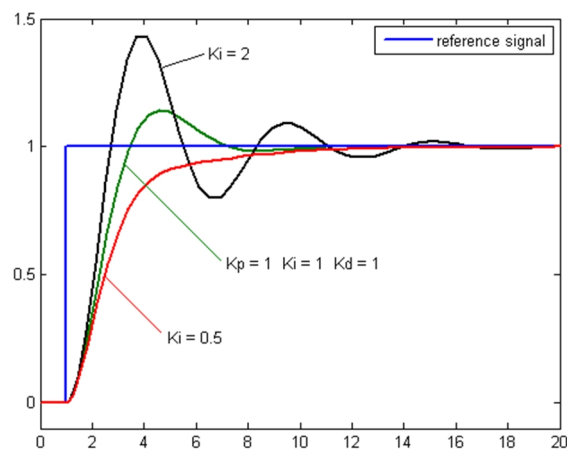


Figure 12. Effect of I term [6]

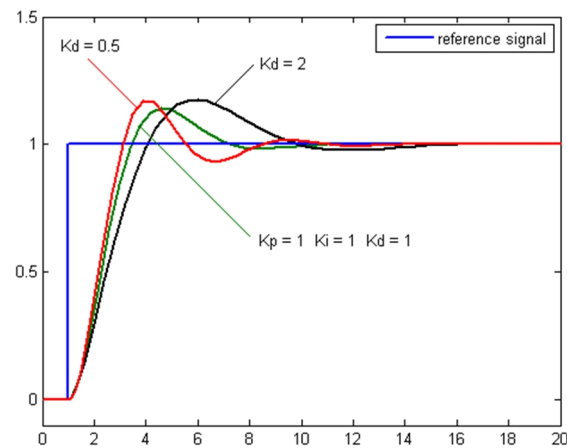


Figure 13. Effect of D term [6]

Tuning a control loop sets the control parameters to their optimum values in order to obtain desired control response. In this case, the main requirement to the system is stability, but beyond that, wide variety of systems leads to different demands and behaviors and these might be incompatible with each other.

From the first look, tuning of three parameters of the PID controller seems to be easy, however, in practice it is a difficult task. This situation occurs because the complex criteria at the PID limit should be satisfied. Tuning is mostly a heuristic concept, but there are many other goals to be reached such as short transient process and high stability increases complexity of this problem.

As an example, systems might include nonlinearity which means that while the parameters work properly for full-load operation, they might not work well for no-load operation. Moreover, wrong choice of PID parameters affects the control process. Input might be unstable and oscillating. It might lead to diverges of the output until it encounters saturation or mechanical breakdown.

Nowadays there are too a lot of various methods PID controller tuning, but most popular of them are as follows [6]:

- Manual Tuning Method;
- Ziegler-Nichols Tuning Method;
- Software Tools Method.

The idea of manual tuning method is to select PID parameters in accordance to the system response. Proportional (P), integral (I) and differential (D) coefficients are changed until the desired response is obtained.

Setting algorithm has the following sequence of actions. At the beginning we set “I” and “D” values to zero. “P” is increased until the loop output is oscillated; then “P” value should be changed to approximately half of the previous value for a "quarter amplitude decay" type response. Increase I until any offset is corrected in sufficient process time. However, an excessive value of integral coefficient will cause instability. Finally, change “D”, if required, until the loop is acceptably quick to reach its reference after a load disturbance. However, it is crucial to remember, an excessive value of a differential coefficient will cause excessive response and overshoot. Sometimes real systems require a “P” value significantly less than the half of the value that was causing oscillation because the system cannot accept large overshoot caused by fast setting of a PID controller. As it could be noticed, the successful outcome of manual PID tuning depends on experience and requires a lot of time.

Ziegler-Nichols tuning method is the method based on the neutral heuristic principle. This Method works as follows. First of all we have to check polarity of the desired proportional control gain. For this purpose, we manually increase step input by a small value and look at the behavior of the steady state output. If its value increases as well, it means the polarity is positive in the opposite case it is negative. As the next step, “I” and “D” coefficients are set to zero and “P” coefficient is increased until periodic oscillation appears at the output response. This critical value of the proportional coefficient is called “ultimate gain” and denoted as K_u . The period where the oscillation occurs is called “ultimate period” and denoted as P_u . As a result, the whole process depends on two variables K_u and P_u . Other coefficients are calculated according to the table 1 [6].

Table 1

Control type	P	I	D
P	$0.5 \cdot K_u$	-	-
PI	$0.45 \cdot K_u$	$1.2 \cdot \frac{P}{P_u}$	-
PID	$0.6 \cdot K_u$	$2 \cdot \frac{P}{P_u}$	$\frac{P \cdot P_u}{8}$

Most modern industrial plants no longer tune loops using the manual calculation methods shown above. Instead, PID tuning and loop optimization software are used to ensure consistent results. These software packages will gather the data, develop process models, and suggest optimal tuning.

A proportional-integral controller or PI Controller is a particular case of the PID controller which does not use the (D) derivative term. It is mainly used to remove the steady state error resulting from P controller. However, in terms of the response speed and overall stability of the system, it has a negative effect. This controller is mostly used in areas where speed of the system is not an issue. Since PI controller has no ability to predict the future errors of the system it cannot decrease the rise time and remove the oscillations.

Formula of analog PI controller is listed below:

$$u(t) = P(t) + I(t) = K_p \cdot e(t) + K_I \cdot \int e(\tau) d\tau; \quad (3.3)$$

where $e(t)$ represents the difference between the desired and the actual position.

Block diagram shown in figure 14 describes the formula of analog PI controller.

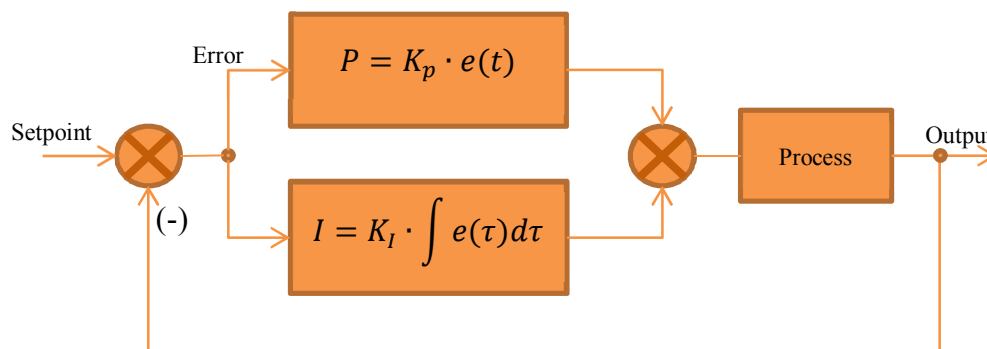


Figure 14. Basic block of PI controller

Formula of digital PI is:

$$u(k) = P(k) + I(k) = K_p \cdot e(k) + K_I \cdot \sum_{i=0}^k h \cdot e(i); \quad (3.4)$$

where h is the sampling time.

3.2. System overview

During the work on my master project I developed the control system of the laboratory stand. It consists of three main parts. The first part is the drive board which is illustrated in the figure 15. This element is based on the microcontroller ATmega8 and plays the main role in the control system. ATmega8 is responsible for command processing, motors controlling, current measurement, encoder data exchange and calculations of control impact.

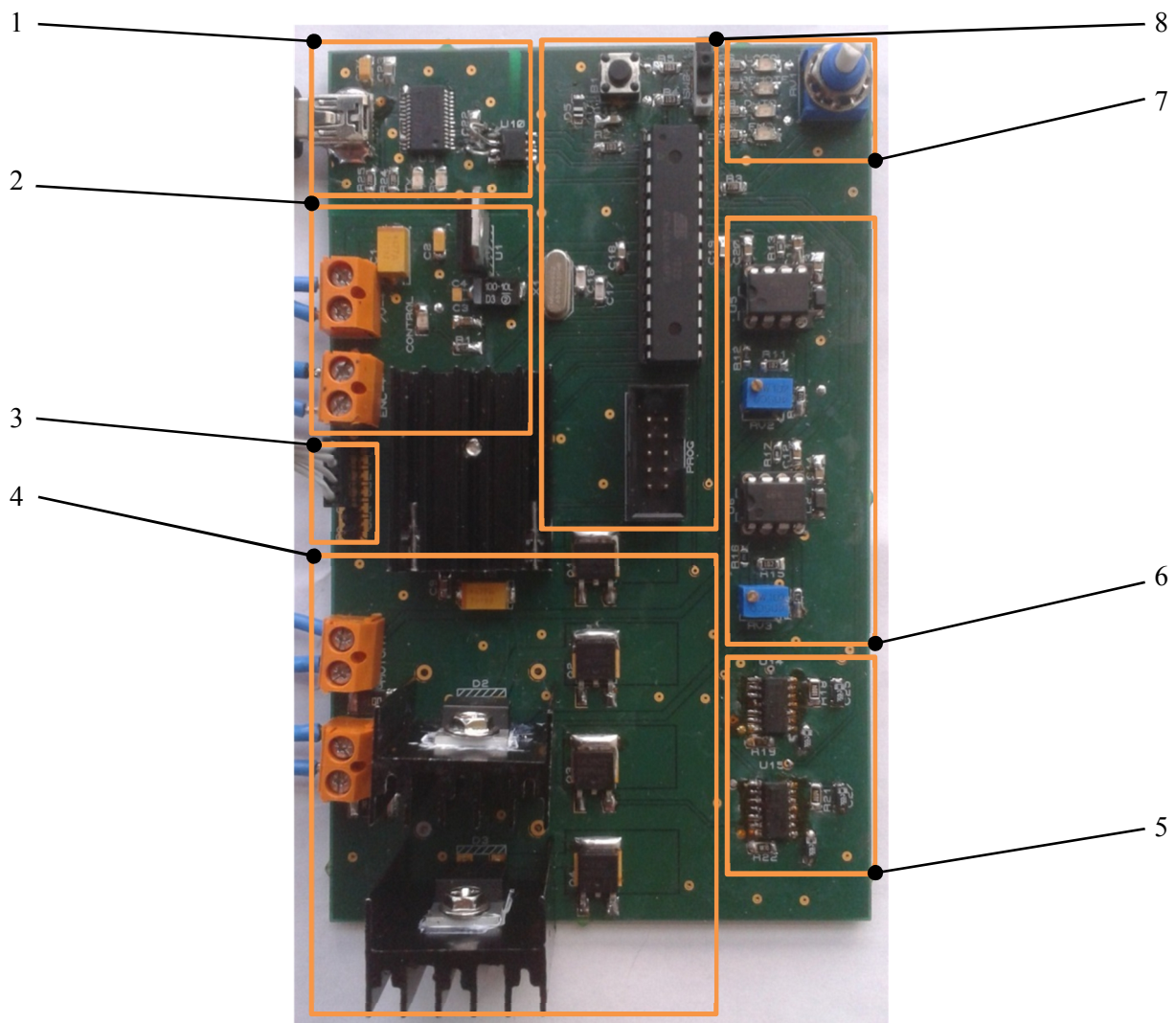


Figure 15. Drive board

We can split this board into several nodes which are given below:

1. USB-UART converter with digital isolator and indication LEDs;
2. Drive power supply input, encoder power supply output and circuit of a voltage regulator for controller power supply with indicator of the work;

Remote setting up the PID controller parameters based on visual observation of controlled motion

3. Socket of SSI (SPI) interface for data transfer between magnetic rotary encoder and controller;
4. Motors power supply unit consists of four power MOSFETs and three power diodes;
5. Control module of the motors power supply unit;
6. Current measurement node consists of two amplifiers with LC-filters and trimmer resistors;
7. Local setpoint potentiometer with drive state LED indicators;
8. Controller with peripheral, reset button, programmer socket and mode switch.

The second part is the encoder board which is shown in the figure 16. The main element of this board is a magnetic rotary encoder AS5045. It is used to control the position of the movable part relative to the fixed part.

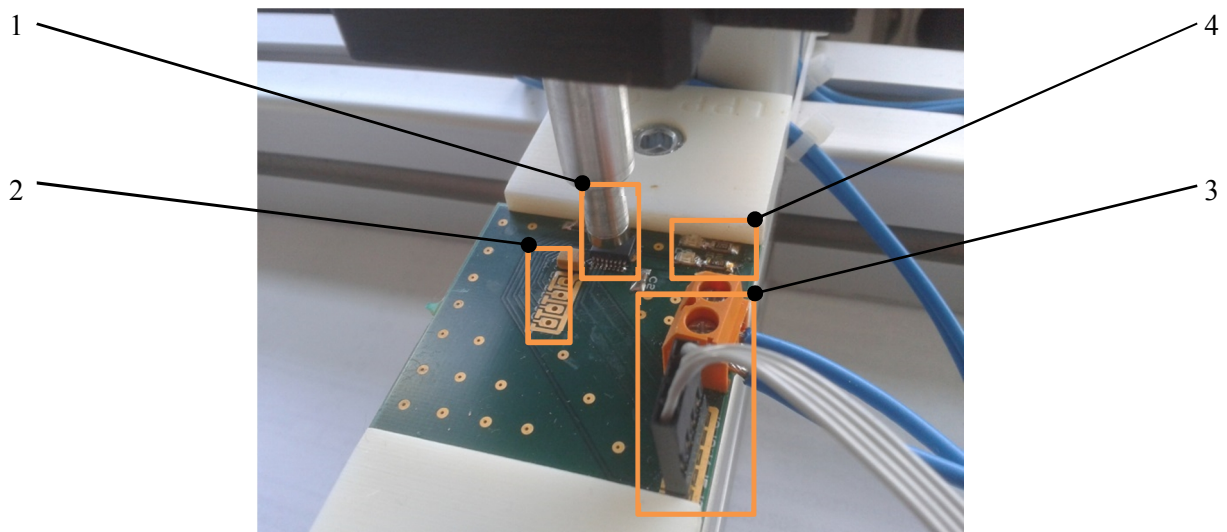


Figure 16. Encoder board

Nodes from the figure 16 are described as follows:

1. Encoder with permanent polar magnet and peripheral capacitors;
2. Contact points of unused signals such as +3.3V, A, B and increment;
3. Encoder power supply input and SSI(SPI) interface socket;
4. Indication LEDs with limiting resistors.

The third part is an external power supply converter from 110V or 220V of alternative current to 7V of the direct current which is presented in the figure 17.



Figure 17. Power supply converter

Figure 18 illustrates the general view of the assembled plant with all parts described above. For additional information about electric circuit and used components see appendix 1 and 2.

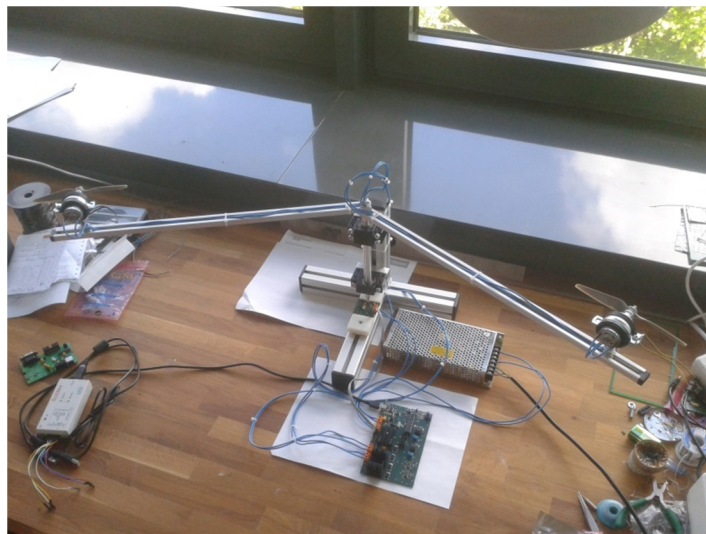


Figure 18. General view of the laboratory stand

Block diagram of the laboratory stand is illustrated in the figure 19. As you can see the system has two possible options to set up the angle setpoint. In the first option the value can be selected by adjusting of the potentiometer. In the second option the value can be specified by using UART interface. This interface is also used for setting up of controller's parameters and system monitoring. To control the speed of DC motors, system uses a power supply unit based on TPS2834D MOSFET drivers. Two sensors are used for armature current. A magnetic encoder is used to receive actual work part position.

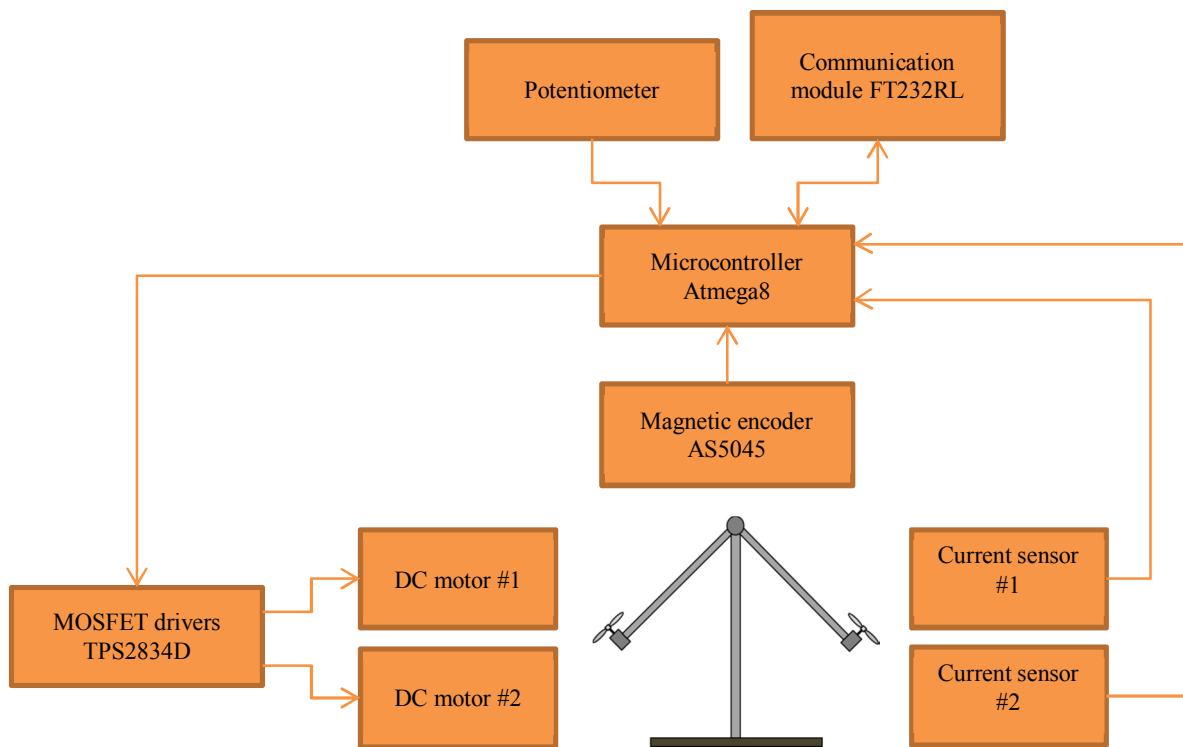


Figure 19. Control system block diagram

As a result, a multi control loop system was obtained. In my application I used cascaded control. The first loop is the control loop of the armature current based on PI controller. It is shown in the figure 20. The work principle of the loop could be described as follows. The current setpoint goes through the adder where its value is summarized with the negative feedback from the sensor. As a result of this action we obtain a regulating error which goes through the PWM generator and motor driver to the DC motor after multiplying with the coefficients of the PI controller.

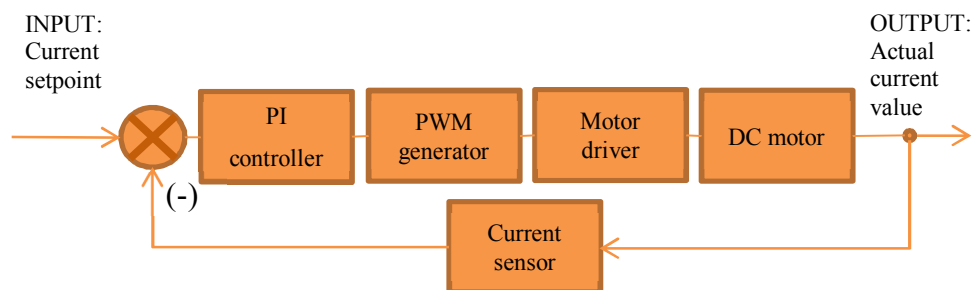


Figure 20. Control closed loop of the armature current

The second loop is the work part position control loop based on the PID controller. It is presented in the figure 21. The work principle of the loop might be described as follows. The position setpoint goes through the adder where its value is summarized with negative feedback from the encoder. As a result of this action we obtain a regulating error which goes to the motor control logic block where it is converted to current setpoint for each motor. Then the obtained setpoints go to the armature current control loop of the corresponding motor. The motors start to run. Propellers' rotation causes change of the work part position. The

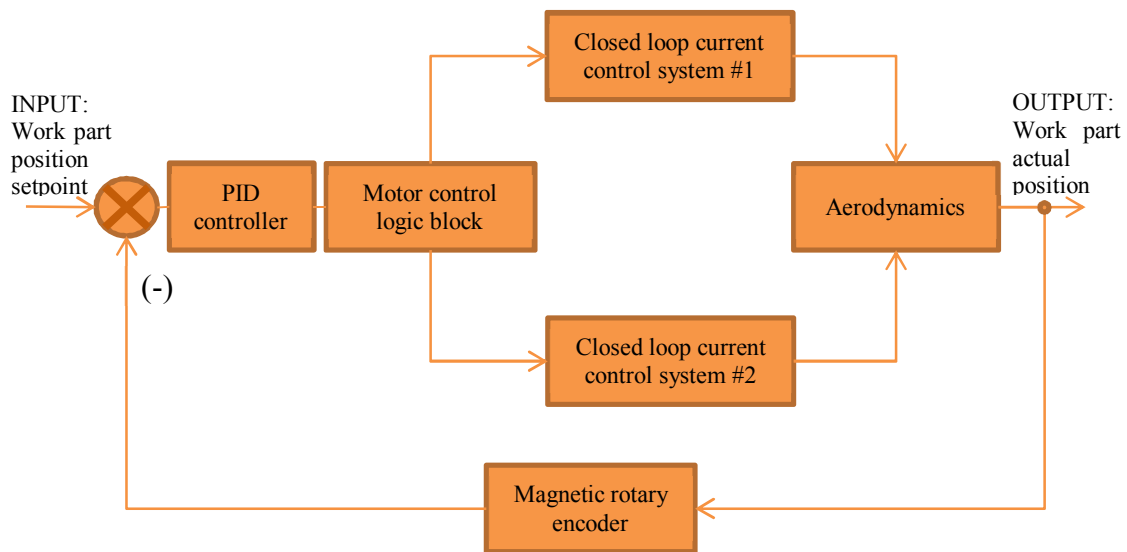


Figure 21. Closed loop of the work part position control

3.3. Description of components used in the project

3.3.1. Atmel AVR ATmega8 microcontroller

There are lot important features in selection of a right microcontroller. In my project I decided to use ATmega8 produced by Atmel Company (see figure 22). This is a low-power CMOS 8-bit microcontroller based on the AVR RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega8 achieves throughputs approaching 1 MIPS per MHz, allowing to optimize power consumption versus processing speed.

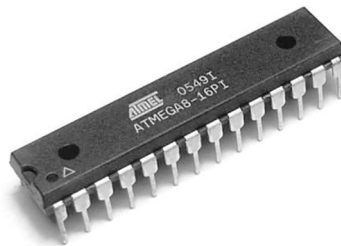


Figure 22. Microcontroller ATmega8 in DIP package [1]

Chosen microcontroller characterized by following features:

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
 - 130 Powerful Instructions – Most Single-clock Cycle Execution
 - 32 x 8 General Purpose Working Registers
 - Fully Static Operation
 - Up to 16 MIPS Throughput at 16 MHz
 - On-chip 2-cycle Multiplier
- Nonvolatile Program and Data Memories
 - 8K Bytes of In-System Self-Programmable Flash
 - Endurance: 10,000 Write/Erase Cycles
 - Optional Boot Code Section with Independent Lock Bits
 - In-System Programming by On-chip Boot Program
 - True Read-While-Write Operation
 - 512 Bytes EEPROM
 - Endurance: 100,000 Write/Erase Cycles
 - 1K Byte Internal SRAM
 - Programming Lock for Software Security
- Peripheral Features

Remote setting up the PID controller parameters based on visual observation of controlled motion

- Two 8-bit Timer/Counters with Separate Prescaler, one Compare Mode
- One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
- Real Time Counter with Separate Oscillator
- Three PWM Channels
- 8-channel ADC in TQFP and MLF package
 - Six Channels 10-bit Accuracy
 - Two Channels 8-bit Accuracy
- 6-channel ADC in PDIP package
 - Four Channels 10-bit Accuracy
 - Two Channels 8-bit Accuracy
- Byte-oriented Two-wire Serial Interface
- Programmable Serial USART
- Master/Slave SPI Serial Interface
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-chip Analog Comparator
- Special Microcontroller Features
 - Power-on Reset and Programmable Brown-out Detection
 - Internal Calibrated RC Oscillator
 - External and Internal Interrupt Sources
 - Five Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, and Standby
- I/O and Packages
 - 23 Programmable I/O Lines
 - 28-lead PDIP, 32-lead TQFP, and 32-pad MLF
- Operating Voltage
 - 4.5 - 5.5V
- Speed Grades
 - 0 - 16 MHz
- Power Consumption at 4 MHz, 3V, 25°C
 - Active: 3.6 mA
 - Idle Mode: 1.0 mA
 - Power-down Mode: 0.5 μ A

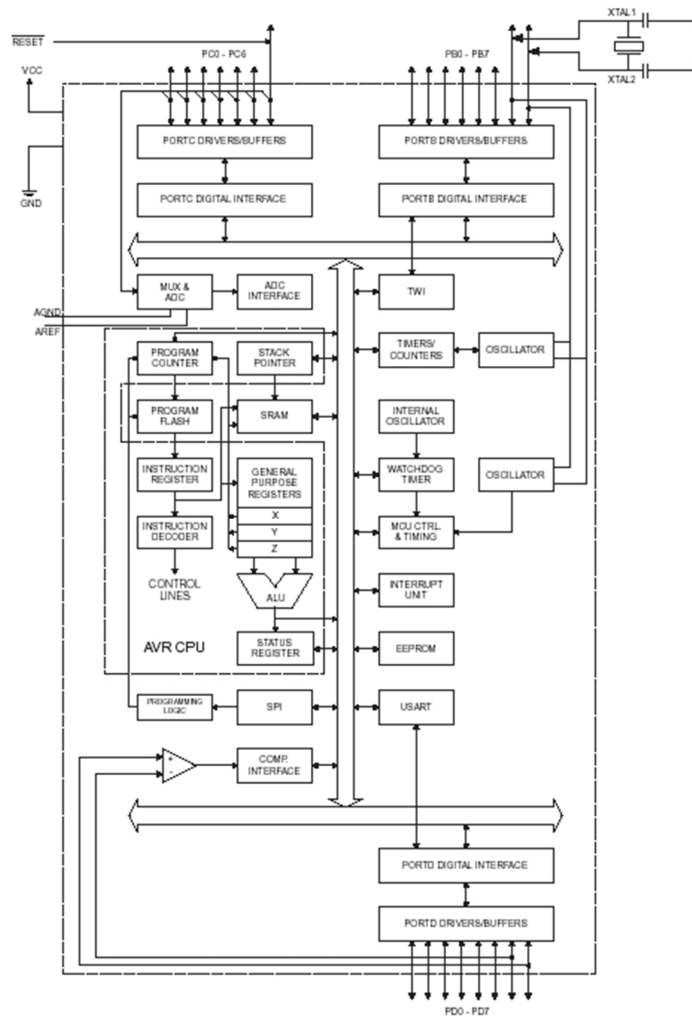


Figure 23. ATmega8 block diagram [1]

Figure 24 shows pin configuration of Atmega8 microcontroller.

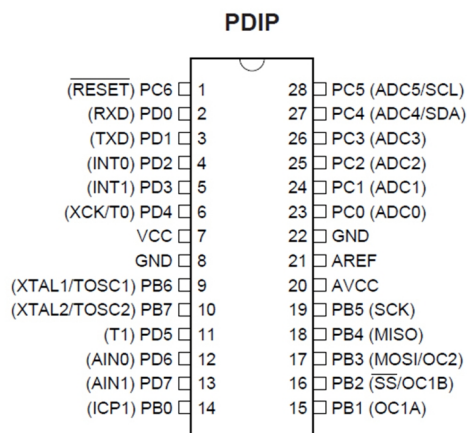


Figure 24. ATmega8 pin configuration [1]

Description of possible usage of the pins is provided below:

VCC	Digital supply voltage. (+5V).
GND	Ground.
Port B (PB7..PB0) XTAL1/XTAL2/ TOSC1/TOSC2	Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running. Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit. Depending on the clock selection fuse settings, PB7 can be used as an output from the inverting Oscillator amplifier. If the Internal Calibrated RC Oscillator is used as chip clock source, PB7.6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set. The various special features of Port B are elaborated in Atmega8 datasheet [1] on pages 23 and 56.
Port C (PC5..PC0)	Port C is an 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.
PC6/RESET	If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C. If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. The minimum pulse length is given on page 36 of the datasheet. Shorter pulses are not guaranteed to generate a Reset. The various special features of Port C are elaborated on page 59 of Atmega8 datasheet [1].
Port D (PD7..PD0)	Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running. Port D also serves the functions of various special features as listed on page 61 of Atmega8 datasheet [1].
RESET	Reset input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. The minimum pulse length is given on page 36 of Atmega8 datasheet [1].

Shorter pulses are not guaranteed to generate a reset.

AVCC AVCC is the supply voltage pin for the A/D Converter, Port C (3..0), and ADC (7..6). It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter. Note that Port C (5..4) use digital supply voltage, VCC.

AREF AREF is the analog reference pin for the A/D Converter.

3.3.1.1. UART

A Universal Asynchronous Receiver Transmitter (UART) is one of the most famous, reliable and widely used interfaces. It provides communication between devices by two wires connection. One wire plays the role of a transmitter (TXD) and another one is a receiver (RXD). This configuration makes connection possible to work in a full duplex mode as shown in the figure 25.

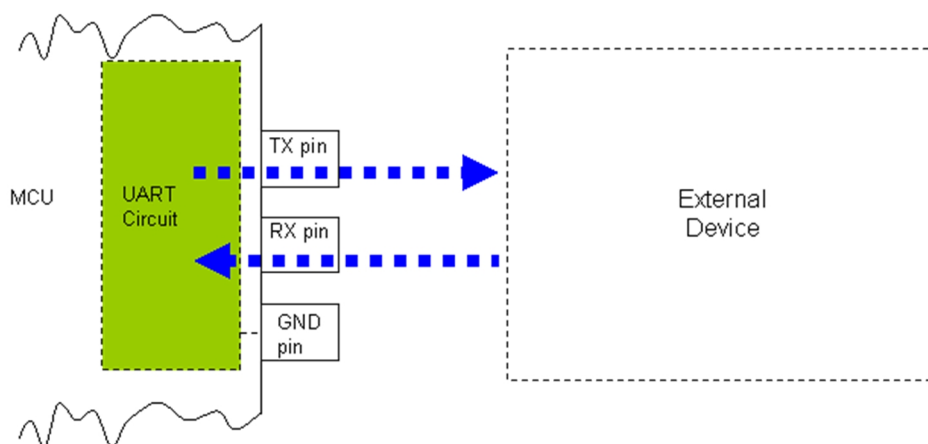


Figure 25. UART interface [7]

In some cases, we can use only one wire of this interface if it is required only to send (display) or receive the information (sensor). The microcontroller Atmega8 has integrated into a USART circuit. It is asynchronous serial interface. Its protocol has frame bits at the beginning and end of a data byte. These bits inform the receiver about new incoming data byte and also indicate that information was completely sent. The bit rate of asynchronous serial interface is relatively slow, but it only demands a single wire for communication. The UART is capable to transfer 7 or 8 data bits and 2 or 3 frame bits. Normal message consists of one start bit, eight bits of data and one stop bit. UART message frame is presented in the figure 26.

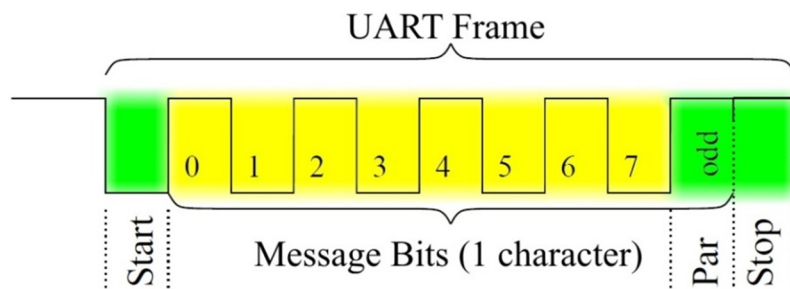


Figure 26. UART message frame

3.3.1.2. SPI (SSI)

The Serial Peripheral Interface (SPI) is a 4 wires bus which performs serial data transfer between Master and Slave circuits as it is shown in the figure 27. In most cases (but not always) MCU performs the Master function. This interface was developed by the Motorola Company, but it is currently used by all manufacturers. The main distinction of this interface is simplicity of use and implementation, high data rate and small working distance.

The SPI has the following work principle. The selection line (CS or SS, Chip Select or Slave Select) is used to select a device and activate its communication. Every data block or command consists of 8 bits and aligned on the CS signal. The data is transferred in clock cycles synchronized by the SCLK line (generated always by the Master). The two remaining lines allow full duplex data transferences: MOSI (Master Output Slave Input) and MISO (Master Input Slave Output).

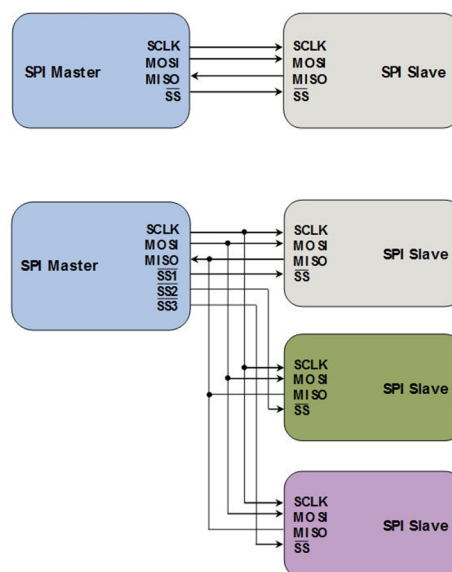


Figure 27. SSI(SPI) interface [8]

The SPI is physically realized on the shift register basis which performs transmitter and receiver function. The principle of the data exchange via SPI is illustrated in the figure 28.

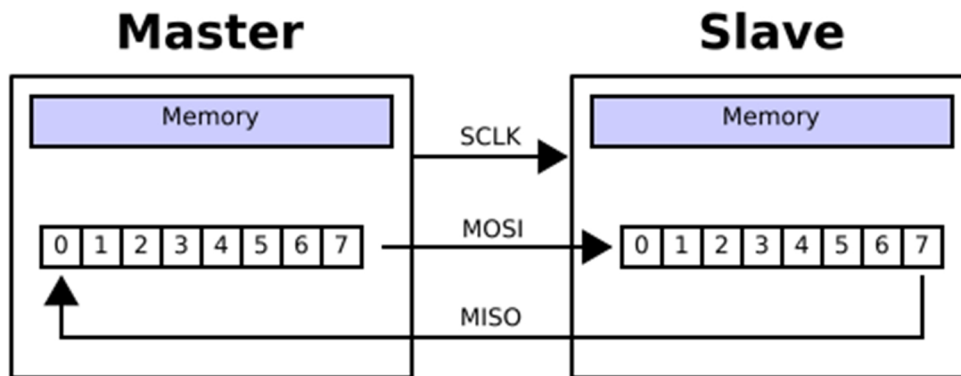


Figure 28. SSI(SPI) shift registers [8]

3.3.2. Synchronous-buck mosfet driver TPS2834D

The TPS2834 is a MOSFET driver for synchronous-buck power stages. It is shown in the figure 29. This device is ideal for designing a high-performance power supply using switching controllers that do not include on-chip MOSFET driver. The driver is designed to deliver at least minimum 2-A peak currents into large capacitive loads. The high-side driver can be configured as ground-reference or as floating-bootstrap. An adaptive dead-time control circuit eliminates shoot-through currents through the main power FETs during switching transitions, and provides high efficiency for the buck regulator. The TPS2834 has additional control functions: ENABLE, SYNC, and CROWBAR. Both high-side and low-side drivers are off when ENABLE is low. The driver is configured as a nonsynchronous-buck driver disabling the low-side driver when SYNC is low. The CROWBAR function turns on the low-side power FET, overriding the IN signal, for overvoltage protection against faulted high-side power FETs [9, p. 1].

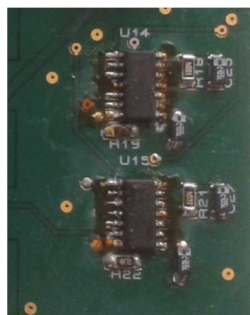


Figure 29. TPS2834D synchronous-buck MOSFET drivers

In my application these devices perform a controller function of a motor power supply unit. Figure 30 illustrates electrical circuit of the unit. As it is shown, the unit includes two half H-bridges. Each of them consists of two MOSFETs, one diode, several capacitors and resistors which perform accurate work of the drivers.

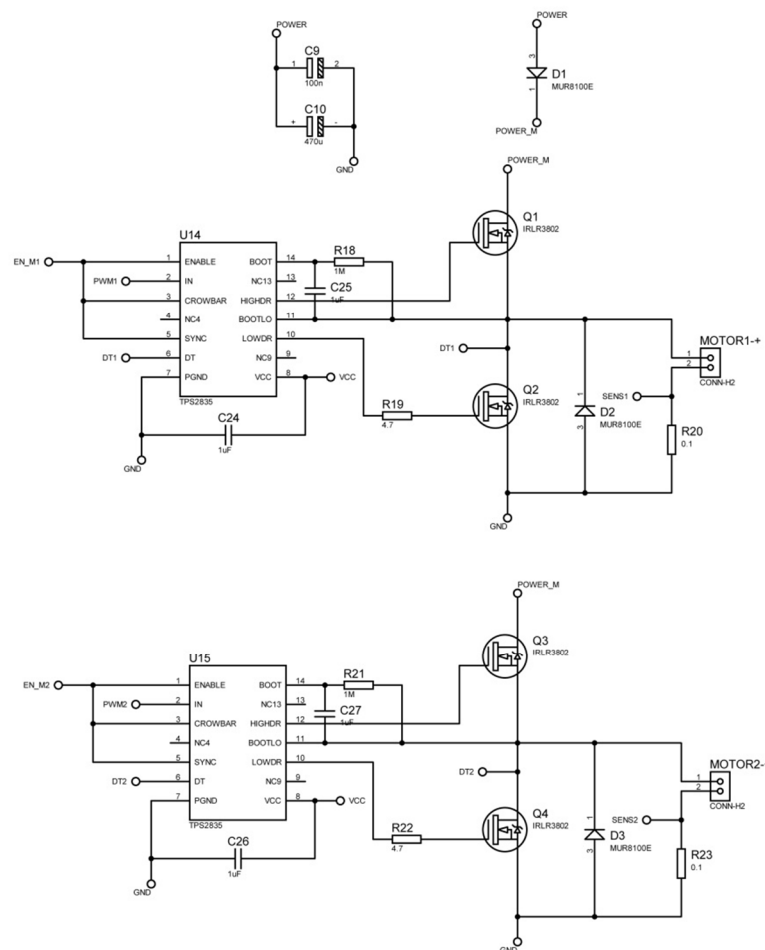


Figure 30. Electric circuit of the motor power unit

3.3.3. Dual operational amplifier LM358N

These circuits consist of two independent, internally frequency-compensated high-gains which were designed specifically to operate from a single power supply over a wide range of voltages [10, p. 1]. The low power supply drain is independent of the magnitude of the power supply voltage. The general view of the device is illustrated in the figure 31.



Figure 31. Dual operational amplifiers LM358N

Application areas include transducer amplifiers, DC gain blocks and all the conventional op-amp circuits which now can be more easily implemented in single power supply systems. For example, these circuits can be directly supplied with the standard +5V which are used in logic systems and easily provide the required interface electronics without any additional power supply. In the linear mode, the input common-mode voltage range includes ground and the output voltage can also swing to ground, even though it is operated from only a single power supply voltage.

I used this device for amplification of the measured armature current. It consists of several resistors (which help to tune the gain coefficients) and two filters. One filter is a LC-filter and another one is an RC-filter. They are used to obtain smoothed value for armature current feedback. The electric circuit of the current measurement node is presented in the figure 32.

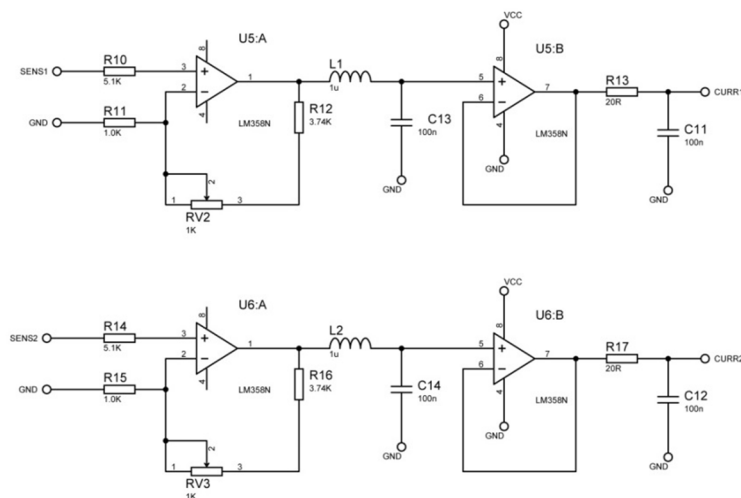


Figure 32. Electric circuit of the current measurement node

3.3.4. Magnetic rotary encoder AS5045

The AS5045 is a contactless magnetic rotary encoder for an accurate angular measurement over a full turn of 360°. It is a system-on-chip, combining integrated Hall elements, analog front end, and a digital signal processing in a single device. General view of the encoder is illustrated in the figure 33.

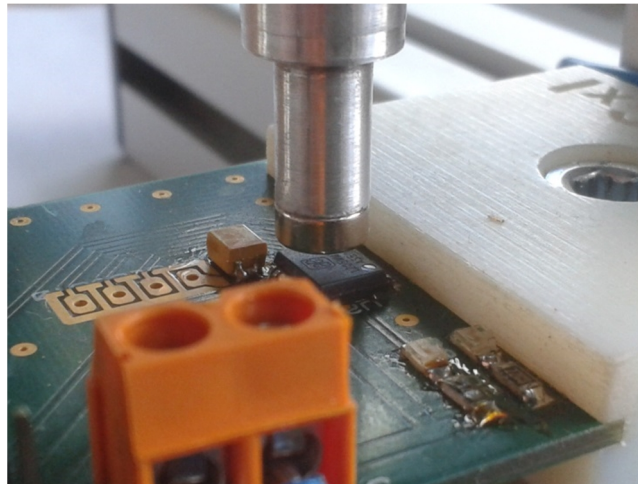


Figure 33. Magnetic rotary encoder AS5045

To measure the angle, only a simple two-pole magnet, rotating over the center of the chip, is required. The magnet may be placed above or below the IC (see figure 34).

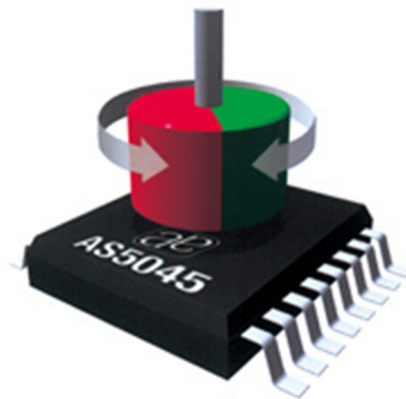


Figure 34. Encoder operation [11, p. 1]

The absolute angle measurement provides instant indication of the magnet's angular position with a resolution of $0.0879^\circ = 4096$ positions per revolution. These digital data are available as a serial bit stream and as a PWM signal. Both diagrams are provided in the figures 35 and 36 correspondingly [11].

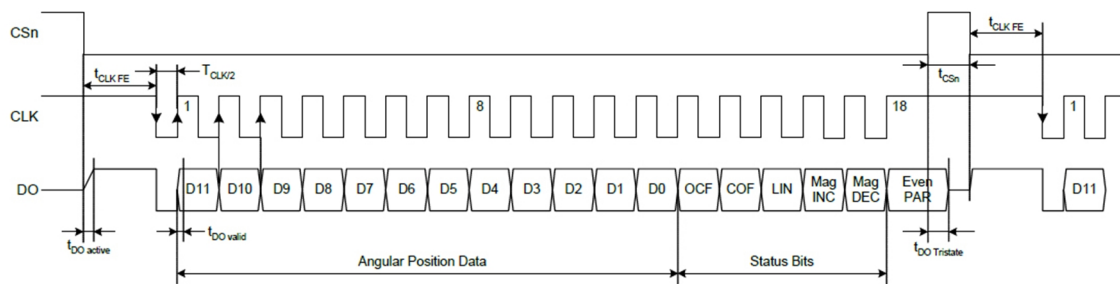


Figure 35. Synchronous serial interface [11, p. 13]

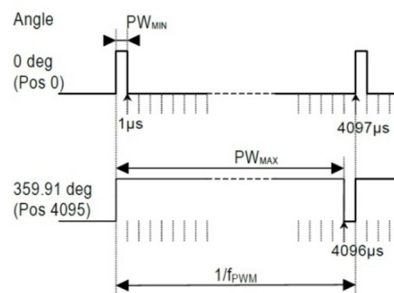


Figure 36. PWM output signal [11, p. 16]

3.3.5. FT232RL communication module

The FT232RL chip is a bridge between USB and serial UART interfaces with the following advanced features. The chip is defined as COM-port at the PC side in case of the default settings. From the device side it has an interface UART. This circuit simplifies the process of connection your handmade device to PC via USB. It is very reliable, stable and supported by all operating systems [12].

In my application, FT232RL allows developed PWM drive to communicate with the supervisory application which is located on the PC side. Figure 37 presents hardware implementation of the communication module.

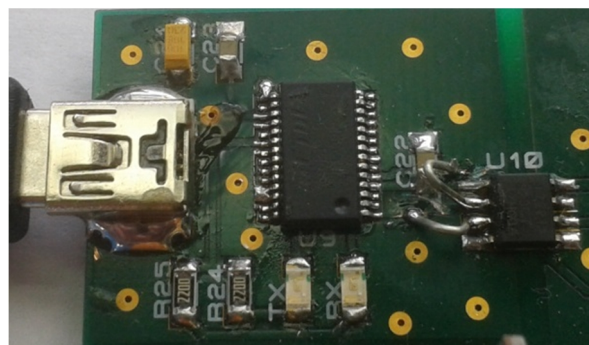


Figure 37. FT232RL communication module

Remote setting up the PID controller parameters based on visual observation of controlled motion

As it could be noticed, the module consists of one FT232RL chip, two indication LEDs with limiting resistors, one mini-USB socket, three capacitors and one digital isolator. Indicators allow us to know how the process of data exchange is going and the digital isolator allows adapting the voltage between the PC and PWM drive. The module electric circuit is illustrated in the figure 38.

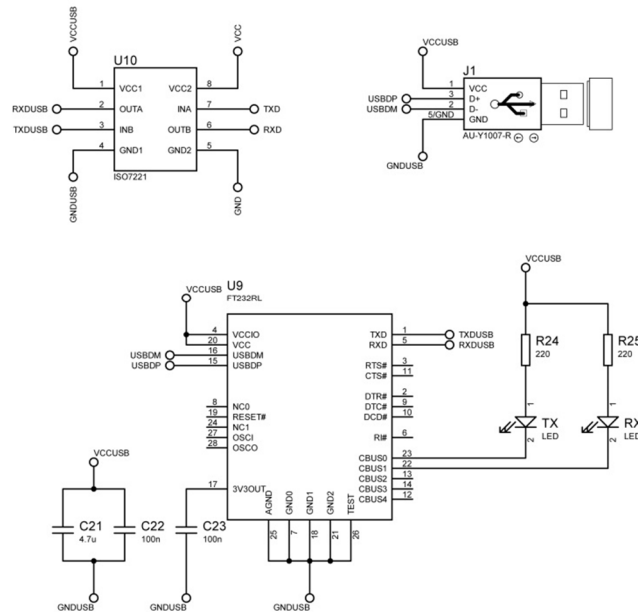


Figure 38. Electric circuit of the communication module

4. Software description

4.1. Microcontroller firmware

Microcontroller firmware was written in the evaluation version of CodeVisionAVR environment. Project structure is shown in the figure 39. It consists of six source files. Each file includes a set of functions.

Code from “adc.c” performs interaction between analog-digital converter of Atmega8 and the main program of controller. “as5045.c” source communicates with magnetic rotary encoder AS5045. Due to this file we can select zero setpoint position for encoder or receive information about the actual position of the work part. Code of the main file collects all functions together. It incorporates a few basic algorithms and instructions. Here you can find the realization way of PI and PID controller and their implementation in the cascade closed loop. Commands received from PC are processed by functions from “parser.c”. Control of PWM A and B of Atmega8 is realized by “pwmab.c”. “usart.c” contents set of data transfer functions. Using this set you can change parameters PC to drive connection such as baud rate, data bits and parity. Content of all source files is shown in the appendix 3.

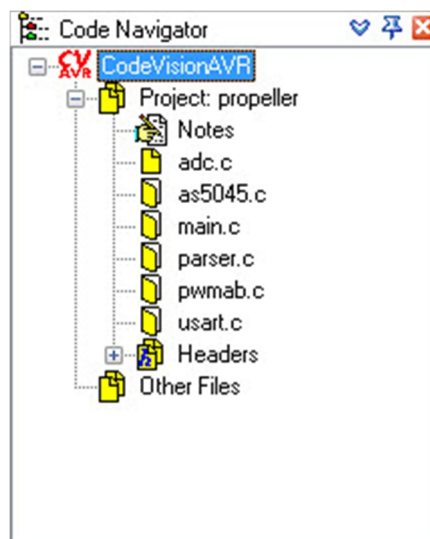


Figure 39. Propeller project firmware in CodeVisionAVR

4.2. Remote control via ProJet

ProJet is my own developed software application which allows us to control the plant via PC or via Internet. This program is written in Visual Studio 2013 by using C# programming language. It performs the following functions: command processing, PWM drive control, and system monitoring. The program interface is shown in the figure 40.

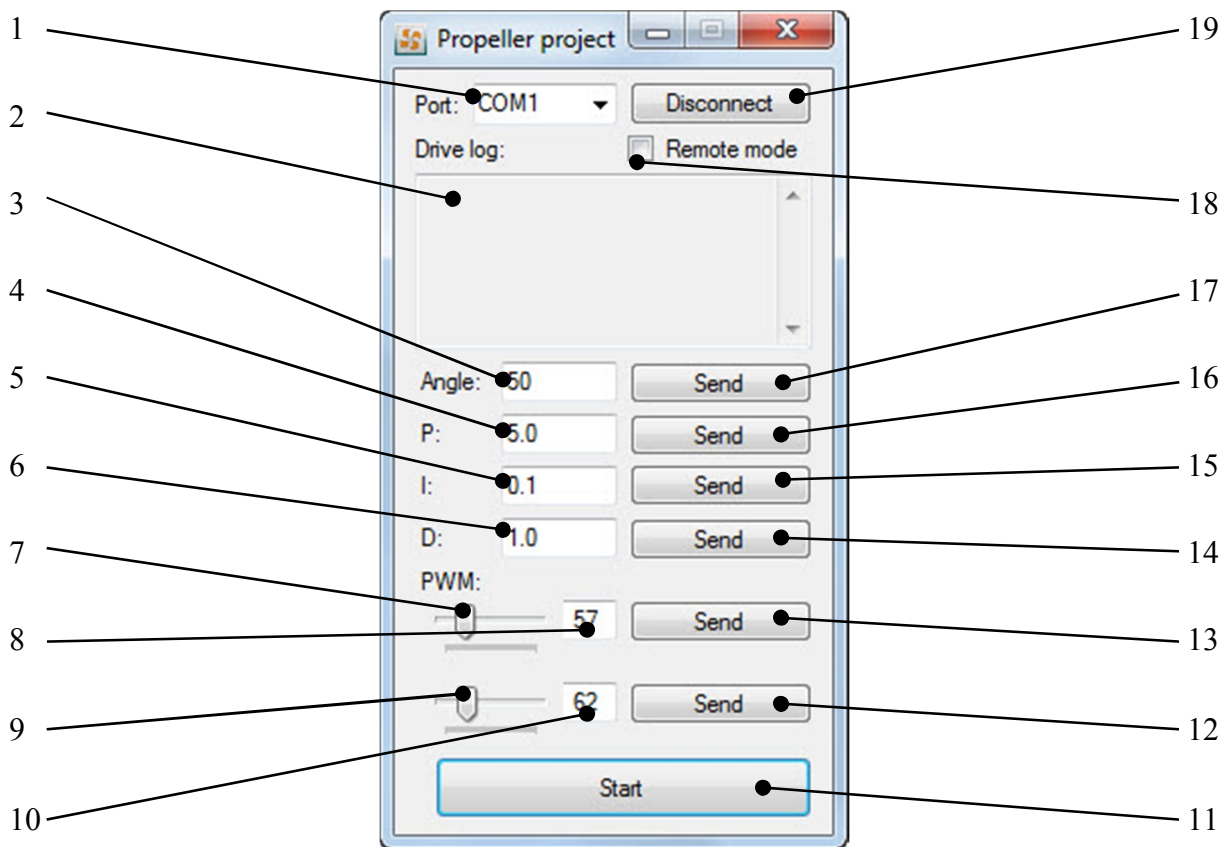


Figure 40. ProJet interface

The purpose of corresponding elements from figure 40 is described below:

1. Port. This ListBox shows the list of all available COM-ports. User should select a port which is connected to the plant;
2. Drive log. TextBox shows information about the plant and status of command processing;
3. Angle. User should type desired angle setpoint to this field;
4. P. User should enter desired value of proportional coefficient of PID controller to this field;
5. I. User should enter desired value of integral coefficient of PID controller to this field;
6. D. User should enter desired value of differential coefficient of PID controller to this field;



Remote setting up the PID controller parameters based on visual observation of controlled motion

7. PWM TrackBar1. Runner of TrackBar1 selects setpoint value for PWM1;
8. PWM TextBox1. This field shows selected value from TrackBar1;
9. PWM TrackBar2. Runner of TrackBar1 selects setpoint value for PWM2;
10. PWM TextBox2. This field shows selected value from TrackBar2;
11. START/STOP Button. This button click starts or stops the experiment. If the button was in the START state, motors begin to rotate and work part moves to desired position, otherwise the experiment is interrupted.
12. PWM2 SEND Button. By clicking this button user will immediately set the value from PWM TextBox2 as a setpoint for PWM generator #2 of the drive;
13. PWM1 SEND Button. By clicking this button user will immediately set the value from PWM TextBox1 as a setpoint for PWM generator #1 of the drive;
14. D SEND Button. Clicking this button leads to setting of the value from D field to differential coefficient of drive PID controller;
15. I SEND Button. Clicking this button leads to setting of the value from I field to integral coefficient of drive PID controller;
16. P SEND Button. Clicking this button leads to setting of the value from P field to proportional coefficient of drive PID controller;
17. Angle SEND Button. Clicking this button leads to setting of the value from Angle field to angle setpoint of drive;
18. Remote mode CheckBox. Check of this CheckBox means the start of the work in the remote control mode via Skype. In this mode control of the plant it is possible only by set of command words. See chapter 4.3 for extra information;
19. CONNECT/DISCONNECT Button. After clicking this button in the "CONNECT" state ProJet try to connect to the drive system and start to receive information from drive. If the button was in "DISCONNECT" state, click leads to connection interrupting and the data exchange stops.

For additional information about application source code see appendix 4.

4.3. Remote control via Skype

Nowadays Skype is the most popular and famous free voice-over-IP service and instant messaging client. This cross platform application is currently being developed by the Microsoft Skype Division. The name was derived from "sky" and "peer". Skype logo is presented in the figure 41.



Figure 41. Skype logo

The service allows users to communicate with peers by voice using a microphone, video by using a webcam, and instant messaging over the Internet. These three criteria and free license of the program influenced my decision to use Skype as internet transport to deliver commands from remote client to the plant.

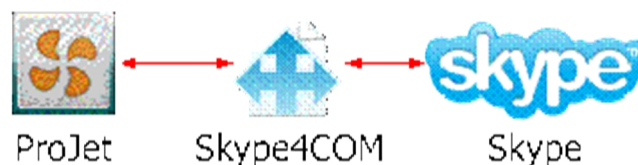


Figure 42. ProJet-Skype platform

Implementation of this idea becomes possible thanks to Skype4COM library. The platform structure is shown in the figure 42. This component gives access to various Skype functions and we can use it in external application. Operating range is quite impressive and can be seen in the library manual [13]. Despite the procedures variety I used only three of them in my application - message send, message receive and video call.



During the work process I developed a set of command words. Each command is written as a usual text message in Skype, but it has to include “+” sign in the beginning. Plus in this case plays role of command identifier.

Words with short description are shown below:

- **+drive** – drive status check and signal to start controlling via Skype;
- **+start** – start the experiment;
- **+angle %value%** - set deviation angle for work part of the plant;
- **+p %value%** - set proportional coefficient of PID controller;
- **+i %value%** - set integral coefficient of PID controller;
- **+d %value%** - set differential coefficient of PID controller;
- **+stop** – stop the experiment;
- **+help** – commands description.

In our case **%value%** is any numeric value in corresponding range which has float or integer format.



5. Conclusion

In this work I have developed and described the laboratory stand, including hardware, firmware and software aspects of the design. The main goal of my master thesis was fulfilled. Obtained plant can visually show the effect of the PID parameters change. Not a very complex and user friendly application ProJet allows any user to interact with the stand easily. For example, it became possible to make the setpoint or parameters change. The set of command and functions can be easily extended.

Using Skype service is an innovative step that allows us to control the plant from any modern mobile phone, tablet or PC in any point of the world. For instance, students from any country will be able to make laboratory work with the equipment which is located in the Czech Republic and vice versa.

All algorithms were written in the very popular and well know programming languages such as C# and C++. Making software in the form of separate libraries lets us use its functions in new projects without any difficulties. Devised solutions can be implemented in a wide range of areas from education to industry.

6. Literature

- [1] Atmel, 8-bit AVR with 8k bytes in-system programmable flash. ATmega8 ATmega8L datasheet., 2011, p. 303.
- [2] "Pulse-width modulation," [Online]. Available: http://en.wikipedia.org/wiki/Pulse-width_modulation. [Accessed 12 Apr 2014].
- [3] "Pulse Width Modulation," [Online]. Available: <http://pctuning.tyden.cz/ilustrace3/urbanek/lcdoc/PWM.gif>. [Accessed 3 May 2014].
- [4] M. Ghioni, "DC Motor Fundamentals.pdf," [Online]. Available: <ftp://ftp.elet.polimi.it>. [Accessed 12 Apr 2014].
- [5] "MIG 400 7.2V," [Online]. Available: http://homepages.paradise.net.nz/bhabbot/MIG400_72V.html. [Accessed 15 Apr 2014].
- [6] "PID controller," [Online]. Available: http://en.wikipedia.org/wiki/PID_controller. [Accessed 12 April 2014].
- [7] "UART BASICS," [Online]. Available: http://www.layadcircuits.com/layad_articles/UART_Basics.htm. [Accessed 20 May 2014].
- [8] "Serial Peripheral Interface Bus," [Online]. Available: http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus. [Accessed 10 Apr 2014].
- [9] "TPS2834 Non-inverting Fast Synchronous Buck MOSFET Drivers with TTL Inputs and Enable," TI, [Online]. Available: <http://www.ti.com/product/tps2834>. [Accessed 24 May 2014].
- [10] "LM158, LM258, LM358 Low-power dual operational amplifiers," ST, [Online]. Available: <http://www.farnell.com/datasheets/1718567.pdf>. [Accessed 24 May 2014].



- [11] "AS5045 Rotary Sensor," ASM, [Online]. Available:
<http://www.ams.com/eng/Products/Position-Sensors/Magnetic-Rotary-Position-Sensors/AS5045>. [Accessed 10 May 2014].
- [12] "FT232RL USB UART IC Datasheet Version 2.10," FTDI, [Online]. Available:
<http://www.farnell.com/datasheets/1647396.pdf>. [Accessed 24 May 2014].
- [13] Skype, Desktop API Reference Manual, SkypeSDK.pdf, p. 100.
- [14] P. V., Automatic control of electric drives: DC drives coordinate regulation. Part 1., Novosibirsk: NSTU, 2013, p. 200.
- [15] F. G. Vostrikov A. S., The theory of automatic control, Moscow: Higher School, 2006, p. 365.



7. CD ROM

The CD ROM contains the following components:

1. PDF file of the diploma thesis;
2. PDF file of the drive and encoder electrical circuits;
3. PDF file with top and bottom copper of the drive PCB;
4. PDF file of encoder PCB;
5. GBR and EXC files for the boards' production;
6. HEX file of the microcontrollers' firmware;
7. Source files of the firmware;
8. EXE file of the ProJet software;
9. Source file of the ProJet software;
10. Manuals and datasheets.

Appendix 1

This appendix presents whole electric circuit of the drive which is shown in the figure 43 and electric circuit of encoder (see figure 46). It also includes printed circuit boards of both devices. Top and bottom copper of the drive PCB are presented in the figures 44 and 45 correspondingly. Figure 47 illustrates the encoder PCB.

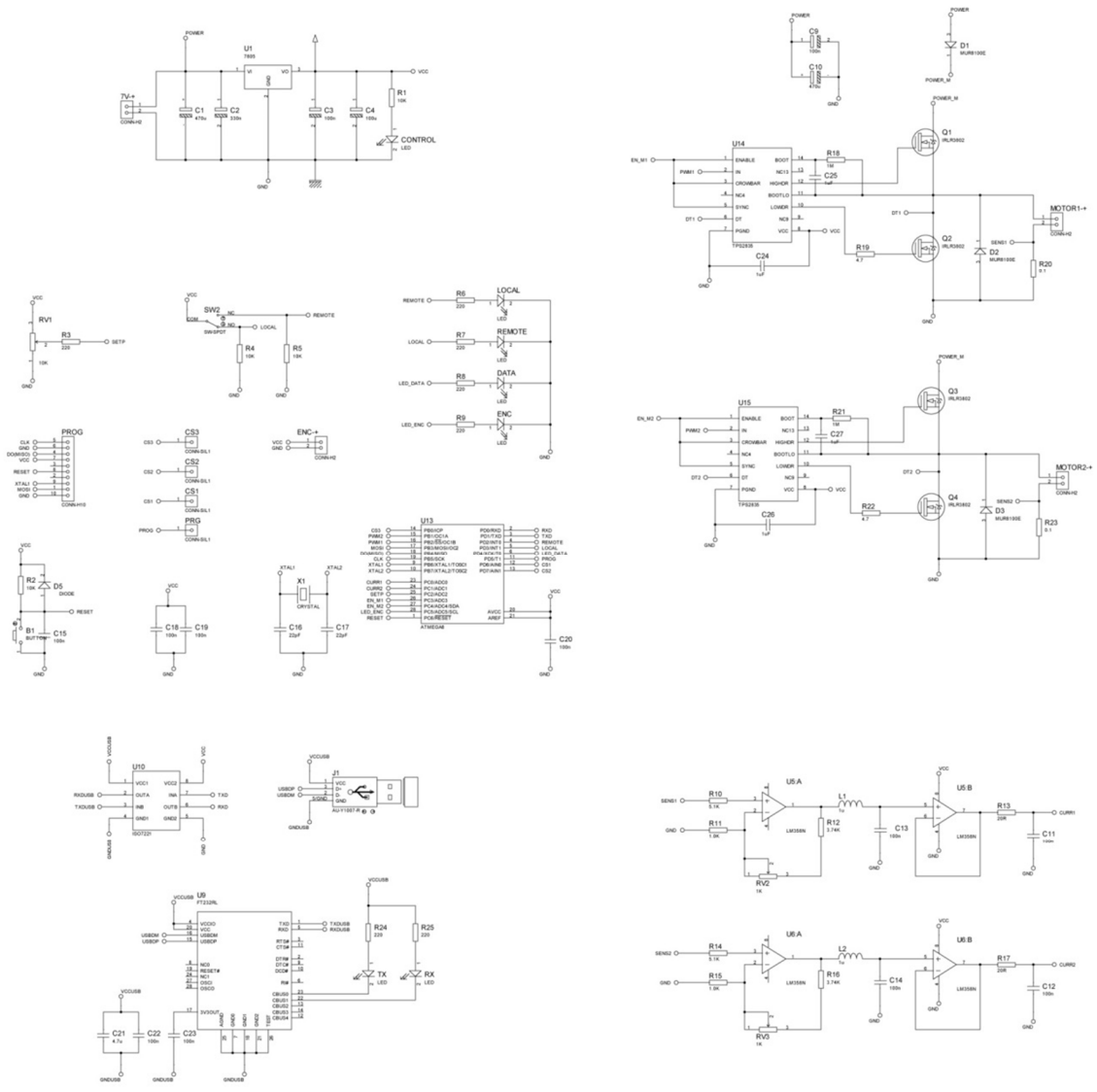


Figure 43. Drive electric circuit

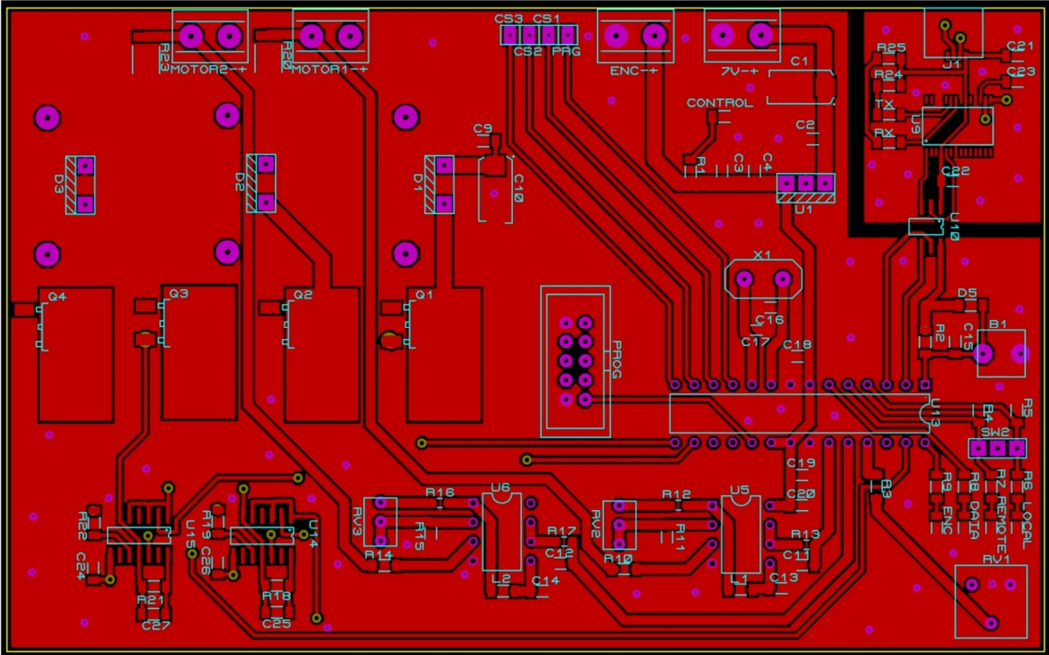


Figure 44. Top copper of the drive PCB

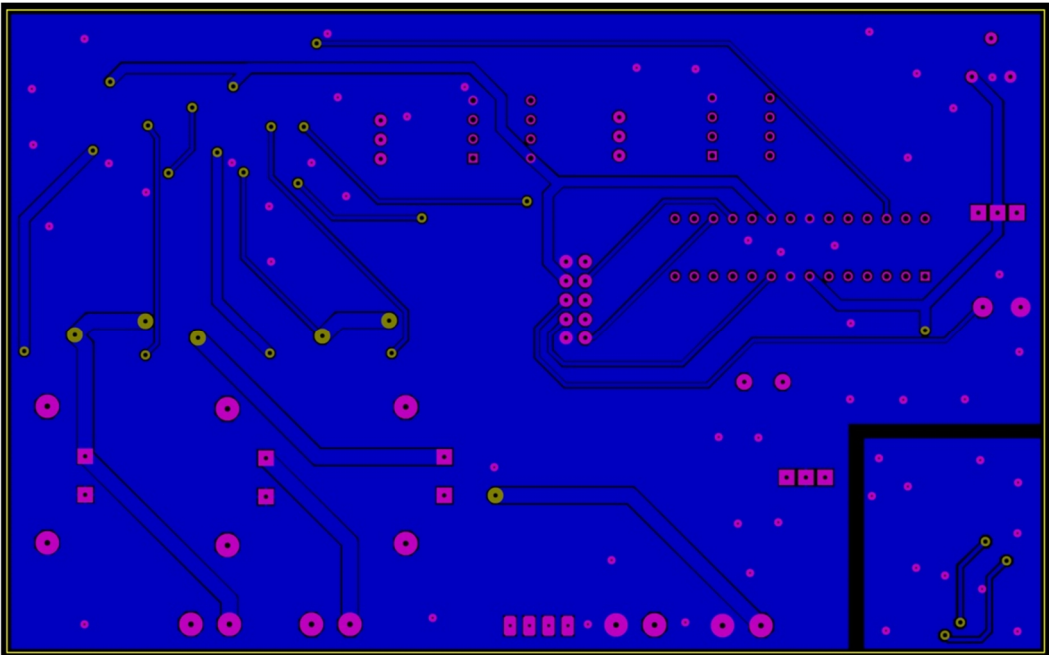


Figure 45. Bottom copper of the drive PCB

Remote setting up the PID controller parameters based on visual observation of controlled motion

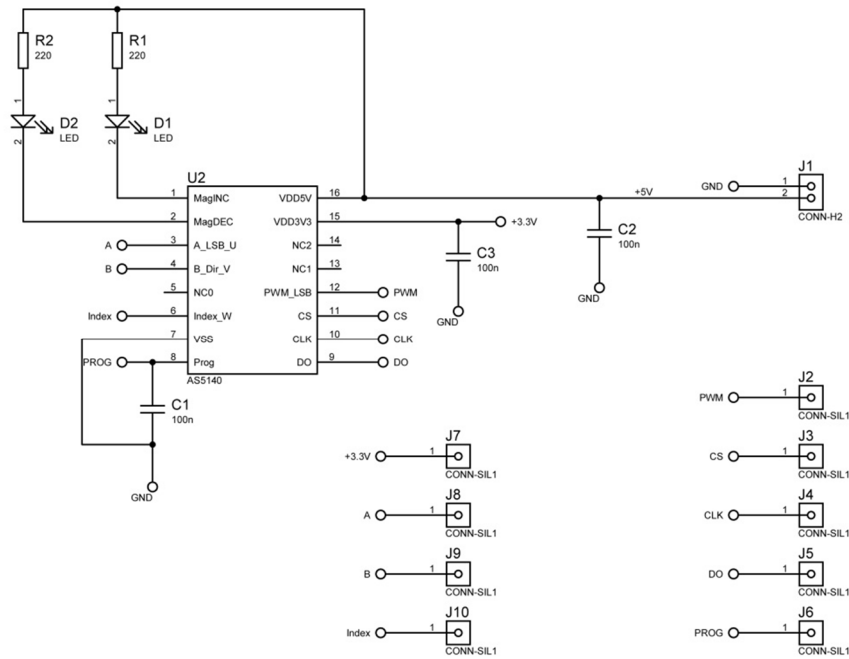


Figure 46. Encoder electric circuit

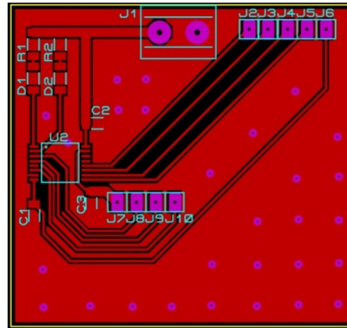


Figure 47. Encoder PCB

Appendix 2

List of used elements is shown in the table 2.

Table 2

Number	Component	Quantity
1	SOCKET 10 PIN	1
2	7805 VOLTAGE REGULATOR	1
3	AS5145 ENCODER	1
4	AVR MICROCONTROLLER ATMEGA8	1
5	CAPACITOR 1.0UF	4
6	CAPACITOR 100N	15
7	CAPACITOR 100UF	1
8	CAPACITOR 22PF	2
9	CAPACITOR 330NF	1
10	CAPACITOR 4.7UF	1
11	CAPACITOR 470UF	2
12	CONNECTOR 2.54MM	11
13	CRYSTAL 8MHZ	1
14	DIGITAL ISOLATOR ISO7221AD	1
15	SWITCHING DIODE	1
16	DIODE MUR8100EG	3
17	FT232R USB UART IC	1
18	HEAT SINK	3
19	INDUCTOR 1UH	2
20	IRLR3802PBF TRANSISTOR	4
21	LED, SMD, 1206 BLUE	9
22	LM358N	2
23	MINI USB B	1
24	POTENTIOMETER, 10K	1
25	RESISTOR 0R100	2
26	RESISTOR 10 KOHM	3
27	RESISTOR 1KOHM	2
28	RESISTOR 1MOHM	2
29	RESISTOR 20R0	2
30	RESISTOR 220 OHM	10
31	RESISTOR 3K74	2
32	RESISTOR 4R7	2
33	RESISTOR 5R1	2
34	SWITCH FSM4JRT	1
35	SWITCH SPDT-CO	1
36	TERMINAL BLOCK 2PIN	5
37	TPS2834D DRIVER	2
38	TRIMMER 1K	2

Appendix 3

This appendix contains the source codes of the microcontroller firmware.

The "adc.c" source code is presented below:

```
#include "adc.h"
#include <delay.h>

void ADC_Init(void)
{
    ADMUX=ADC_VREF_TYPE & 0xff;
    ADCSRA=(1<<ADPS0)|(1<<ADPS1)|(1<<ADEN);
}

// Read the AD conversion result
unsigned int ADC_Read(unsigned char adc_input)
{
    ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
    // Delay needed for the stabilization of the ADC input voltage
    delay_us(10);
    // Start the AD conversion
    ADCSRA|=(1<<ADSC);
    // Wait for the AD conversion to complete
    while ((ADCSRA & 0x10)==0);
    ADCSRA|=(1<<ADIF);
    return ADCW;
}
```

Remote setting up the PID controller parameters based on visual observation of controlled motion

The "as5045.c" source code is presented below:

```
#include "as5045.h"
#include <delay.h>
#include <io.h>

void AS5045_Init()
{
    DDRB |= (1 << DDB0);          //prog
    PORTD &=~((1 << PIND7));      //do->low level
    DDRD |= (1 << DDD5)|(1 << DDD6); // cs,clk
    DDRD &=~((1 << DDD7));        //do->input
    PORTD.PORTD5 = 0; //cs=0
    PORTD.PORTD6 = 1; //clk=1
    PORTB.PORTB0 = 0; //prog=0
}

uint16_t AS5045_Ssi(void)
{
    uint16_t i, data;
    PORTD.PORTD5 = 0; //cs=0
    PORTD.PORTD6 = 0;
    data = 0;
    for (i = 0; i < 16; i++)
    {
        PORTD.PORTD6 = 1;
        delay_us(1);
        PORTD.PORTD6 = 0; // conversion after falling edge CS
        data = data << 1; // shift bits in target variable one to the left
        if ( PIND & (1<<PIND7) ) // read port bit
        {
            data |=0x01; // set new bit in 16 bit variable
        }
        delay_us(1);
    }
    PORTD.PORTD5 = 1; //cs=1
    return data;
}
```

The "main.c" source code is presented below:

```
/**
 *
 * Author(s)...: Dmitry Kochubey
 *
 * Target(s)...: Atmega8
 *
 * Compiler....: CodeVision 2.05
 *
 * Description.: Propeler project v0.1
 *
 * Data.....: 14.04.14
 */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "compilers.h"
#include "config.h"
#include "parser.h"
#include "usart.h"
#include "pwma.h"
#include "adc.h"
#include "as5045.h"

/*Comands*/
__flash char comStat[] = "state"; //Status
__flash char comPwma[] = "pwma"; //Set pwma
__flash char comPwmb[] = "pwmb"; //Set pwmb
__flash char comAngl[] = "angle"; //Set angle
__flash char comPreg[] = "p"; //Set proportional coefficient of PID controller
__flash char comIreg[] = "i"; //Set integral coefficient of PID controller
__flash char comDreg[] = "d"; //Set differential coefficient of PID controller
__flash char comStart[] = "start"; //Start
__flash char comStop[] = "stop"; //Stop

/*Answers*/
__flash char error[] = "error\r";
__flash char ok[] = "ok\r";
```



```
__flash char start[] = "ready\r";

/* declare variables */
int vla = 0;
int vlb = 0;
float vlm = 120.0;

short int stp, enc_zero, psn, old_psn;

int turn=0;
int sflag=0;
int pwmf = 0;
unsigned char infoc[10];
unsigned char infog[10];

/*Variables of PID*/
float CMNDVX;
float INTX;
float ADRCX;
float ERRX;
float ERRM1X;
float NEWDTY;
float MAXDTY = 0.9;
float SMTIME = 0.01;
float KP;
float KI;
float KD;

/*Variables of PI*/
float KP1;
float KI1;

float KP2;
float KI2;

float INTX1;
float INTX2;

float NEWDTY1;
```

```
float NEWDTY2;

float ADRCX1;
float ADRCX2;

/*PI*/
float pi_calc(float pi_kp, float pi_ki, float pi_sp, float pi_ig, float time, float pi_fb, float pi_lm)
{
float pi_er, pi_rg;
pi_er = (pi_sp - pi_fb);
pi_ig+=pi_ki*pi_er*time;
if(pi_ig>pi_lm)
{
pi_ig=pi_lm;
}
else if(pi_ig<-pi_lm)
{
pi_ig=-pi_lm;
}
pi_rg = pi_kp*(pi_er)
+ pi_ig;
if(pi_rg > pi_lm)
{
pi_rg = pi_lm;
}
else if(pi_rg < -pi_lm)
{
pi_rg = -pi_lm;
}
return pi_rg;
}

/*PID*/
void pid_calc(void)
{
ERRX = (CMNDVX - ADRCX);
INTX+=KI*ERRX*SMTIME;
if(INTX>MAXDTY)
{
```



```
INTX=MAXDTY;
}
else if(INTX<-MAXDTY)
{
INTX=-MAXDTY;
}

NEWDTY = KP*(ERRX)
+ INTX
+ KD*((ERRX - ERRM1X)/SMTIME);

if(NEWDTY > MAXDTY)
{
    NEWDTY = MAXDTY;
}
else if(NEWDTY < -MAXDTY)
{
    NEWDTY = -MAXDTY;
}
ERRM1X = ERRX;
}

interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
if (pwmf==0)
{
    TCNT0=0x9E;

    PWM_A(vla);
    PWM_B(vlb);

    psn=(AS5045_Ssi() & 0b111111111110000) >> 4;
    psn-=enc_zero;

    if (psn<0)
    {
        psn=4095+psn;
    }
    if (psn<45&&old_psn>4050)
```

```
{
turn++;
}
else if(psn>4050&&old_psn<45)
{
turn--;
}

old_psn=psn;

if (turn!=0)
{
psn=psn+(turn*4095);
}

ADRCX = ((float)psn)/4095;

pid_calc(); /*PID LOOP/PWM routine */

if (NEWDTY < 0){
vla = (int)((MAXDTY+NEWDTY) * vlm);
vlb = (int)(MAXDTY * vlm);
}
else if(NEWDTY > 0)
{
vla = (int)(MAXDTY * vlm);
vlb = (int)((MAXDTY-NEWDTY) * vlm);
}
else if (NEWDTY == 0)
{
vla = (MAXDTY * vlm);
vlb = (MAXDTY * vlm);
}
vla = pi_calc(KP1, KI1, vla, INTEX1, ADRCX1, MAXDTY);
vlb = pi_calc(KP2, KI2, vlb, INTEX2, ADRCX2, MAXDTY);
}
}
```

```
/*Comand processor*/
void PARS_Handler(uint8_t argc, char *argv[])
{
    uint8_t value = 0;
    char __flash *resp = error;

    if (PARS_EqualStrFl(argv[0], comPwmb)){
        if (argc > 1){
            value = PARS_StrToUchar(argv[1]);
            if (value <= 250){
                PWM_B(value);
                pwmf = 1;
                resp = ok;
            }
            else{
                resp = error;
            }
        }
    }
    else if (PARS_EqualStrFl(argv[0], comPwma)){
        if (argc > 1){
            value = PARS_StrToUchar(argv[1]);
            if (value <= 250){
                PWM_A(value);
                pwmf = 1;
                resp = ok;
            }
            else{
                resp = error;
            }
        }
    }
    else if (PARS_EqualStrFl(argv[0], comStart)){
        sflag=1;
        PORTC.3=1;
        PORTC.4=1;
        resp = ok;
    }
}
```

```
else if (PARS_EqualStrFl(argv[0], comStop)){
    sflag=0;
    PORTC.3=0;
    PORTC.4=0;
    pwmf = 0;
    resp = ok;
}
else if (PARS_EqualStrFl(argv[0], comPreg)){
    if (argc > 1){
        KP = atof(argv[1]);
        resp = ok;
    }
}
else if (PARS_EqualStrFl(argv[0], comIreg)){
    if (argc > 1){
        KI = atof(argv[1]);
        resp = ok;
    }
}
else if (PARS_EqualStrFl(argv[0], comDreg)){
    if (argc > 1){
        KD = atof(argv[1]);
        resp = ok;
    }
}
else if (PARS_EqualStrFl(argv[0], comAngl)){
    if (argc > 1){
        CMNDVX=(atof(argv[1]))/360;
        resp = ok;
    }
}
else if (PARS_EqualStrFl(argv[0], comStat)){
    resp = start;
}

USART_SendStrFl(resp);
}
```

```
void main( void )
{
/*Variables*/
char sym;

/*Libraries initialization*/
USART_Init(USART_NORMAL, 57600);
PARS_Init();
PWMAB_Init();
ADC_Init();
AS5045_Init();

DDRC|=0b00111000;
DDRDI|=0b00010000;

TCCR0=0x05;
TCNT0=0x9E;
TIMSK=0x01;

#asm("sei");

delay_ms(50);
enc_zero=(AS5045_Ssi() & 0b111111111110000)>> 4;

KP=50.0;
KI=10.0;
KD=20.0;

KP1=50.0;
KI1=10.0;

KP2=50.0;
KI2=10.0;

while(1){
delay_ms(50);

if (PIND.2)
```



```
{  
PORTC.3=1;  
PORTC.4=1;  
stp=ADC_Read(ADC2)*4;  
CMNDVX=((float)stp)/4095;  
}  
else if (sflag==0)  
{  
PORTC.3=0;  
PORTC.4=0;  
}  
  
if (USART_GetRxCount()){  
sym = USART_GetChar();  
PARS_Parser(sym);  
}  
delay_ms(100);  
}  
}
```


The "parser.c" source code is presented below:

```
#include "parser.h"

char buf[SIZE_RECEIVE_BUF];
char *argv[AMOUNT_PAR];
uint8_t argc;

uint8_t i = 0;
uint8_t flag = 0;

void PARS_Init(void)
{
    argc = 0;
    argv[0] = buf;
    flag = FALSE;
    i = 0;
}

void PARS_Parser(char sym)
{
    if (sym != '\r'){
        if (i < SIZE_RECEIVE_BUF - 1){
            if (sym != ' '){
                if (!argc){
                    argv[0] = buf;
                    argc++;
                }

                if (flag){
                    if (argc < AMOUNT_PAR){
                        argv[argc] = &buf[i];
                        argc++;
                    }
                }
                flag = FALSE;
            }

            buf[i] = sym;
            i++;
        }
    }
}
```

```
else{
    if (!flag){
        buf[i] = 0;
        i++;
        flag = TRUE;
    }
}
buf[i] = 0;
return;
}
else{
    buf[i] = 0;

    if (argc){
        PARS_Handler(argc, argv);
    }
    argc = 0;
    flag = FALSE;
    i = 0;
}
}

#ifdef __GNUC__

uint8_t PARS_EqualStrFl(char *s1, char const *s2)
{
    uint8_t i = 0;

    while(s1[i] == pgm_read_byte(&s2[i]) && s1[i] != '\0' && pgm_read_byte(&s2[i]) != '\0'){
        i++;
    }

    if (s1[i] == '\0' && pgm_read_byte(&s2[i]) == '\0'){
        return TRUE;
    }
    else{
        return FALSE;
    }
}
```

```
}  
  
#else  
  
uint8_t PARS_EqualStrFl(char *s1, char __flash *s2)  
{  
    uint8_t i = 0;  
  
    while(s1[i] == s2[i] && s1[i] != '\0' && s2[i] != '\0'){  
        i++;  
    }  
  
    if (s1[i] == '\0' && s2[i] == '\0'){  
        return TRUE;  
    }  
    else{  
        return FALSE;  
    }  
}  
  
#endif  
  
uint8_t PARS_StrToUchar(char *s)  
{  
    uint8_t value = 0;  
  
    while(*s == '0'){  
        s++;  
    }  
  
    while(*s){  
        value += (*s - 0x30);  
        s++;  
        if (*s){  
            value *= 10;  
        }  
    };  
    return value;  
}
```



Remote setting up the PID controller parameters based on visual observation of controlled motion

The "pwmab.c" source code is presented below:

```
#include "pwmab.h"

void PWMAB_Init(void)
{
    DDRB|=(1<<DDB1)|(1<<DDB2);
    TCCR1A=(1<<WGM10)|(1<<COM1B1)|(1<<COM1A1);
    TCCR1B=(1<<CS12)|(1<<WGM12);
}

void PWM_A(int pls)
{
    if (pls>=lim) {
        pls=lim;
    };
    OCR1A=pls;
}

void PWM_B(int pls)
{
    if (pls>=lim) {
        pls=lim;
    };
    OCR1B=pls;
}
```

Remote setting up the PID controller parameters based on visual observation of controlled motion

The "usart.c" source code is presented below:

```
#include "usart.h"

static volatile char usartTxBuf[SIZE_BUF_TX];
static volatile uint8_t txBufTail = 0;
static volatile uint8_t txBufHead = 0;
static volatile uint8_t txCount = 0;

static volatile char usartRxBuf[SIZE_BUF_RX];
static volatile uint8_t rxBufTail = 0;
static volatile uint8_t rxBufHead = 0;
static volatile uint8_t rxCount = 0;

#ifndef F_CPU
#error "F_CPU is not defined"
#endif

void USART_Init(uint8_t regime, uint16_t baudRate)
{
    uint16_t ubrrValue;

    uint8_t save = SREG;
    #asm("cli");

    txBufTail = 0;
    txBufHead = 0;
    txCount = 0;

    rxBufTail = 0;
    rxBufHead = 0;
    rxCount = 0;

    UCSRB = 0;
    UCSRC = 0;

    if (regime == USART_NORMAL){
        ubrrValue = F_CPU/(16UL*baudRate) - 1;
    }
    else{
```

```
    ubrrValue = F_CPU/(8UL*baudRate) - 1;
}

UBRRH = (uint8_t)(ubrrValue >> 8);
UBRRL = (uint8_t)ubrrValue;

UCSRA = (1<<(1 & U2X));
UCSRB = (1<<RXCIE)|(1<<RXEN)|(1<<TXEN);
UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0);

SREG = save;
}

uint8_t USART_GetTxCount(void)
{
    return txCount;
}

void USART_PutChar(char sym)
{
    uint8_t save;

    while(txCount == SIZE_BUF_TX);

    save = SREG;
    #asm("cli");
    if (!txCount){
        UCSRB |= (1<<UDRIE);
    }
    if (txCount < SIZE_BUF_TX){
        usartTxBuf[txBufTail] = sym;
        txCount++;
        txBufTail++;
        if (txBufTail == SIZE_BUF_TX) txBufTail = 0;
    }
    SREG = save;
}

void USART_SendStr(char * data)
```

```
{
char sym;
while(*data){
    sym = *data++;
    USART_PutChar(sym);
}
}

void USART_SendStrFl(char __flash * data)
{
char sym;
while(*data){
    sym = *data++;
    USART_PutChar(sym);
}
}

interrupt [USART_DRE] void usart_udre(void)
{
if (txCount > 0){
    UDR = usartTxBuf[txBufHead];
    txCount--;
    txBufHead++;
    if (txBufHead == SIZE_BUF_TX) txBufHead = 0;
}
else{
    UCSRB &= ~(1<<UDRIE);
}
}

uint8_t USART_GetRxCount(void)
{
return rxCount;
}

char USART_GetChar(void)
{
char sym;
uint8_t save;
```

```
if (rxCount > 0){
    sym = usartRxBuf[rxBufHead];
    rxBufHead++;
    if (rxBufHead == SIZE_BUF_RX) rxBufHead = 0;
    save = SREG;
    #asm("cli");
    rxCount--;
    SREG = save;
    return sym;
}
return 0;
}

interrupt [USART_RXC] void usart_rxc(void)
{
    char data = UDR;
    if (rxCount < SIZE_BUF_RX){
        usartRxBuf[rxBufTail] = data;
        rxBufTail++;
        if (rxBufTail == SIZE_BUF_RX) rxBufTail = 0;
        rxCount++;
    }
}
```


Appendix 4

This appendix contains the source codes of the ProJet software.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading;
using System.Windows.Forms;
using System.IO;
using SKYPE4COMLib;

namespace Propeller
{
    public partial class Form1 : Form
    {
        private Skype skype;
        private const string trigger = "+";
        private const string name = "Drive";
        private string message;
        private string client="";

        public Form1()
        {
            InitializeComponent();
        }

        #region Form Handlers
        private void Form1_Load(object sender, EventArgs e)
        {
            string[] myPort;

            myPort = System.IO.Ports.SerialPort.GetPortNames();

            comboBox1.Items.AddRange(myPort);
            string[] parameter = new string[7];
            using (var load = new System.IO.StreamReader(@"parameters.ini"))
            {
                int i=0;
                while ((parameter[i]=load.ReadLine())!=null)
                {
                    i+=1;
                }
                textBox6.Text = parameter[0];
                textBox7.Text = parameter[1];
                textBox5.Text = parameter[2];
                textBox4.Text = parameter[3];
                textBox2.Text = parameter[4];
                textBox3.Text = parameter[5];
                int.TryParse(parameter[4], out i);
                trackBar1.Value = i;
                int.TryParse(parameter[5], out i);
                trackBar2.Value = i;
            }
        }
    }
}
```

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    if (serialPort1.IsOpen == true)
    { serialPort1.Close(); }
    timer1.Stop();
    using (var save = new System.IO.StreamWriter(@"parameters.ini"))
    {
        save.WriteLine(textBox6.Text);
        save.WriteLine(textBox7.Text);
        save.WriteLine(textBox5.Text);
        save.WriteLine(textBox4.Text);
        save.WriteLine(textBox2.Text);
        save.WriteLine(textBox3.Text);
    }
}
#endregion

private void button2_Click(object sender, EventArgs e)
{
    serialPort1.Write("pwma " + textBox2.Text + "\r");
}

private void button3_Click(object sender, EventArgs e)
{
    if (button3.Text.Equals("Connect"))
    {
        if (comboBox1.Text.Equals(""))
        {
            MessageBox.Show("WARNING: SELECT PORT");
        }
        else
        {
            serialPort1.BaudRate = 57600;
            serialPort1.PortName = comboBox1.Text.ToString();
            if (serialPort1.IsOpen == false)
            {
                serialPort1.Open();
                timer1.Start();
            }
            checkBox1.Enabled = true;
            if (checkBox1.Checked == false)
            {
                en_buttons();
            }
            serialPort1.Write("state\r");
            button3.Text = "Disconnect";
        }
    }
    else
    {
        if (serialPort1.IsOpen == true)
        { serialPort1.Close(); }
        timer1.Stop();
        checkBox1.Enabled = false;
        textBox1.Text="";
        di_buttons();
        button3.Text = "Connect";
    }
}

private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
```

```
{
    serialPort1.PortName = comboBox1.Text.ToString();
}

private void button4_Click(object sender, EventArgs e)
{
    serialPort1.Write("p " +textBox7.Text + "\r");
}

private void trackBar1_Scroll(object sender, EventArgs e)
{
    textBox2.Text = trackBar1.Value.ToString();
}

private void timer1_Tick(object sender, EventArgs e)
{
    if (serialPort1.BytesToRead > 0)
    {
        message = serialPort1.ReadExisting();
        textBox1.Text = textBox1.Text + Environment.NewLine + message;
        textBox1.SelectionStart = textBox1.Text.Length;
        textBox1.ScrollToCaret();
        textBox1.Refresh();
        if (checkBox1.Checked == true)
        {
            skype.SendMessage(client, name + " answer: " + message);
        }
    }
}

private void trackBar2_Scroll(object sender, EventArgs e)
{
    textBox3.Text = trackBar2.Value.ToString();
}

private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    if (checkBox1.Checked == true)
    {
        skype = new Skype();
        if (!skype.Client.IsRunning)
        {
            skype.Client.Start(true, true);
        }
        skype.Attach(7, false);
        skype.MessageStatus += new
        _ISkypeEvents_MessageStatusEventHandler(skype_MessageStatus);

        di_buttons();
    }
    else
    {
        en_buttons();
    }
}

private void en_buttons()
{
    button1.Enabled = true;
    button2.Enabled = true;
}
```

```

        button4.Enabled = true;
        button5.Enabled = true;
        button6.Enabled = true;
        button7.Enabled = true;
        button8.Enabled = true;
    }

    private void di_buttons()
    {
        button1.Enabled = false;
        button2.Enabled = false;
        button4.Enabled = false;
        button5.Enabled = false;
        button6.Enabled = false;
        button7.Enabled = false;
        button8.Enabled = false;
    }

    private void skype_MessageStatus(ChatMessage msg, TChatMessageStatus status)
    {
        if (msg.Body.IndexOf(trigger) >= 0 && checkBox1.Checked == true)
        {
            string command = msg.Body.Remove(0, trigger.Length).ToLower();
            client = msg.Sender.Handle;
            if (command.IndexOf(' ') == -1)
            {
                skype.SendMessage(client, name + " answer: " +
ProcessCommand(command, "", client));
                command = "";
            }
            else
            {
                string[] comind = command.Split((new Char[] { ' ' }));
                if (comind.Length > 2)
                {
                    skype.SendMessage(client, name + "WRONG COMMAND");
                }
                else
                {
                    skype.SendMessage(client, name + " answer: " +
ProcessCommand(comind[0], comind[1], client));
                }
            }
        }
    }

    private void skype_call(string user)
    {
        Call call = skype.PlaceCall(user);
        do
        {
            System.Threading.Thread.Sleep(1);
        } while (call.Status != TCallStatus.clsInProgress);
        call.StartVideoSend();
    }

    private string ProcessCommand(string com, string num, string nick)
    {
        string result="Error";
        switch (com)
        {

```

```

        case "drive":
            serialPort1.Write("state\r");
            ThreadStart connect = new ThreadStart(delegate { skype_call(nick);
});
            Thread scc = new Thread(connect);
            scc.Start();
            result = "Connecting";
            break;
        case "help":
            result = "Not presented";
            break;
        case "start":
            serialPort1.Write("start\r");
            result = "started";
            break;
        case "stop":
            serialPort1.Write("stop\r");
            result = "stopped";
            break;
        case "angle":
            serialPort1.Write("angle " + num + "\r");
            result = "angle is set";
            break;
        case "p":
            serialPort1.Write("p " + num + "\r");
            result = "P coefficient is set";
            break;
        case "i":
            serialPort1.Write("i " + num + "\r");
            result = "I coefficient is set";
            break;
        case "d":
            serialPort1.Write("d " + num + "\r");
            result = "D coefficient is set";
            break;
        case "pwma":
            serialPort1.Write("pwma " + num + "\r");
            result = "PWMA is set";
            break;
        case "pwmb":
            serialPort1.Write("pwmb " + num + "\r");
            result = "PWMB is set";
            break;
        default:
            result = "Unknown command";
            break;
    }
    return result;
}

private void button1_Click(object sender, EventArgs e)
{
    serialPort1.Write("angle " + textBox6.Text + "\r");
}

private void button5_Click(object sender, EventArgs e)
{
    serialPort1.Write("i " + textBox5.Text + "\r");
}

private void button6_Click(object sender, EventArgs e)

```

```
{
    serialPort1.Write("d " + textBox4.Text + "\r");
}

private void button7_Click(object sender, EventArgs e)
{
    serialPort1.Write("pwm " + textBox3.Text + "\r");
}

private void button8_Click(object sender, EventArgs e)
{
    if (button8.Text.Equals("Start"))
    {
        serialPort1.Write("start\r");
        button8.Text = "Stop";
    }
    else
    {
        serialPort1.Write("stop\r");
        button8.Text = "Start";
    }
}
}
}
```