Ilia Kempi

Development of mobile crawling robot based on the pneumatic rubber actuators

Student exchange research

Helsinki Metropolia University of Applied Sciences

Akita National College of Technology

Bachelor of Engineering

Electronics department

Project

18.02.2014

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Ilia Kempi Development of mobile crawling robot based on the pneumatic rubber actuators 67 pages + 8 appendices 18 February 2014
Degree	Bachelor of Engineering
Degree Programme	Electronics
Specialisation option	Mechatronics
Instructor	Miyagawa Toyomi, Principal Lecturer

Elastic pneumatic actuators make imitation of a living organism motion possible. This research is focused on the elastic wave drive, implemented with rubber actuators. Similar type of gait, named Rectilinear locomotion, is found in nature and is used by snakes and caterpillars. Despite slow speed, this method of movement is most silent, and allows for a large horizontal load.

Based on that, a mobile crawling robot, powered by batteries and a gas tank, was developed. Actuators are interfaced with solenoid valves and microcontroller; robot is controlled by wireless remote. Motion speed can be precisely set with a joystick. Gas depletion is monitored and shown to the user.

Keywords	Mobile robot, Pneumatic Actuator, Rectilinear locomotion

Contents

1 Introduction

Pneumatic and hydraulic actuators, in contrast with conventional electromagnetic motors, provide an alternative approach to motion control in the field of mechatronics. Artificial muscles, driven by changes in the internal pressure, alter own shape to exert force on the surrounding world, thereby imitating motion of a living organism. This natural ability is important for certain appliances like animal and human robotics. Actuators require complex movement and force distribution pattern to mimic soft and gentle natural motion, not to mention compact size demands. Mechanical design of elastic actuators is quite challenging, as three-dimensional deformation during operation should be modelled.

Actuator control is often a challenging task as well. Precise movement of piezoelectric or pressure actuators is difficult to achieve, and many phenomena should be considered. Hysteresis, flow delay, and driving voltage/pressure drops are to be modelled and their effects included into the system output. Embedded digital electronics offers sufficient computing power to handle most of these processes. For actuator systems, whose physical outputs can be measured, feedback loops are common practice. Those allow for an immediate motion correction, and are utilizing analogue electronics.

In this project, a simple digital human-actuator interface was implemented in order to understand challenges of making such a system. During my research I have learned and understood a lot of new concepts, and did use my former knowledge as well. This report highlights project parts, whose required most of my attention.

2 Project definition

2.1 Allocation and department

Current research was conducted under student exchange agreement between Sendai National College of Technology, Japan and Metropolia University of Applied Sciences, Helsinki. I and other students were distributed into various technical schools among Tōhoku, north-eastern area of Japan, where Sendai is the largest city. I have been allocated to the Akita National College of Technology, Akita prefecture. By my application,

I was placed in the Mechanical Department, Mechatronics laboratory, under supervision of Miyagawa Toyomi, principal lecturer.

2.2 Setup

Professor Miyagawa laboratory and research facility specializes in mechatronics, robotics, mechanical system design, mechanical elements and welfare mechanical engineering. The laboratory is equipped for research and testing of basic mechanical, pneumatic and electrical systems.

I was provided own space in the laboratory, and given an office laptop for use in the project. Research facility has all necessary mechanical tools as well as digital multimeter, digital oscilloscope, air compressor, soldering iron, prototyping boards and DC power supply.

As a material for research, I was given a device named “Bubbler controller board” (Picture 1). It is the remains of year 1995 actuator research project, collaboration between Toshiba R&D and IBM Watson. The circuit board was used as a pneumatic valve controller to drive elastic actuators for testing.

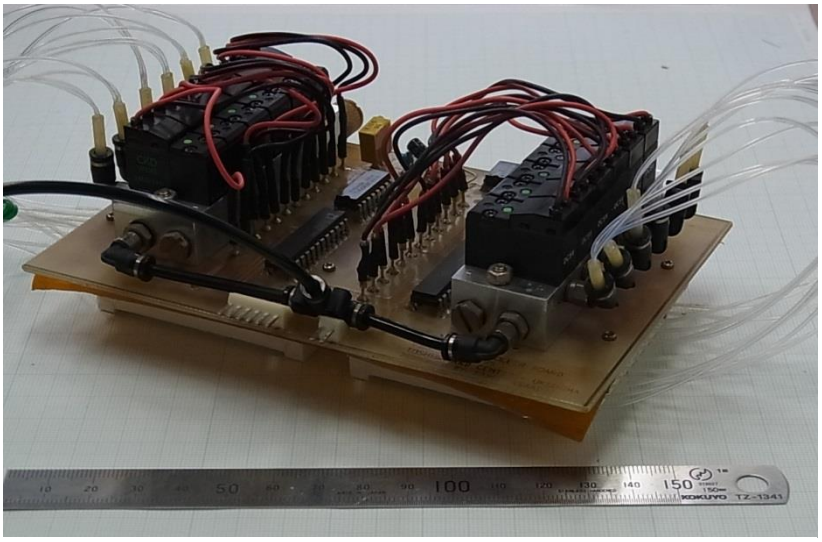


Figure 1. Bubbler controller board

It is still fully operational, its principles and structure are described further in the report, under the Research Section.

2.3 Goals and challenges

Purpose of the project is to design and make a mobile robot out of an actuator mechanism used in previous research. It should be powered by batteries and a small N_2 gas tank (Figure 2). If possible, its control should be wireless, meaning robot should have as less limitations of mobility as possible. Professor Miyagawa is to provide support with mechanical part. Project duration is five months.

Figure 2. NTG N_2 BLOW

Challenges of the project were:

- Work on the project alone
- All required information had to be self-studied
- All necessary components had to be estimated and ordered

2.4 Project timeline

Project had started in October 2013 and continued until February 2014. Table 1 provides chronological sequence of work on the project and highlights important milestones (green cells).

Working day starts at 9 AM and continues until about 3 PM, with lunch break of one hour. Daily workload was not the same during the project, and increased linearly, approaching closer to the deadline. Periods of higher workload are indicated by colour of

the table cells. Blue cells indicate work until about 5 PM, and red cells are for days over 5 PM.

Table 1. Project timeline

1 Oct 2013 - 14 Oct 2013	Project planning and actuator research
15 Oct 2013	Order of ATmega328P microcontrollers with the AVR Dragon programmer
16 Oct 2013	2-axis joystick is salvaged from the game controller
17 Oct 2013 - 22 Oct 2013	Prototyping the joystick with a temporary MSP430 Microcontroller
23 Oct 2013 - 29 Oct 2013	Connecting a second MSP430 through SPI serial connection
30 Oct 2013 - 2 Nov 2013	Hacking the bubbler controller board with dual MSP430 joystick-control system in order to drive actuators independently
3 Nov 2013	Prototype of robot steering controller works
4 Nov 2013 - 10 Nov 2013	Working on the preliminary PCB component list
11 Nov 2013 - 18 Nov 2013	Learning KiCAD software, first circuit design
19 Nov 2013 - 25 Nov 2013	Research on advanced robot control
26 Nov 2013 - 30 Nov 2013	First PCB Layout design
2 Dec 2013	AVR Dragon and ATmega328P have arrived
3 Dec 2013 - 12 Dec 2013	Learning AVR Dragon and Atmel studio. Programming ATmega328P
13 Dec 2013 - 18 Dec 2013	Design of a power supply, PCB and component list revision
19 Dec 2013 - 7 Jan 2014	Winter vacation
8 Jan 2014 - 13 Jan 2014	PCB chemical etching training and research. Final PCB component list is confirmed
14 Jan 2014	Raw PCB plates and all components have been ordered
15 Jan 2014 - 20 Jan 2014	Final PCB design. Programming
21 Jan 2014	Etching of controller and robot PCBs is successful
22 Jan 2014 - 30 Jan 2014	PCB drilling and assembly, firmware beta version
31 Jan 2014	Robot is complete and operational
1 Feb 2014 - 6 Feb 2014	Robot tests and corrections to the program
7 Feb 2014 - 18 Feb 2014	Report writing

In general, work was done every day, except weekends, national holidays and college-wise events. No work was done during winter vacation.

3 Research

3.1 Bubbler actuator

"Bubbler", a pneumatic rubber actuator (from now on simply actuator) consists of a silicone rubber slab, and its interior is divided into many chambers (Figures 3 and 4).

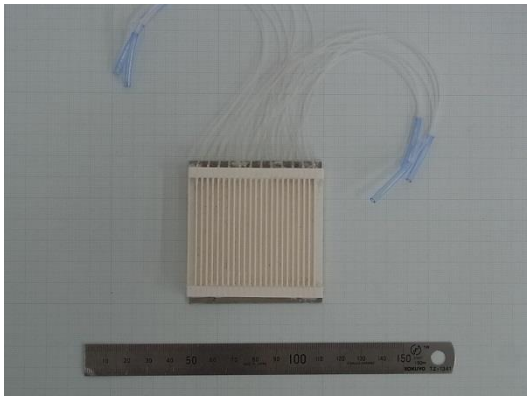


Figure 3. Actuator

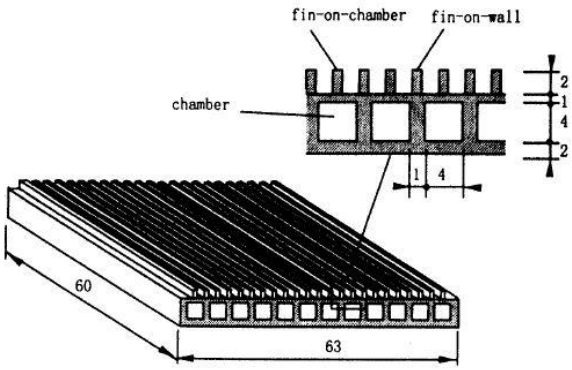


Figure 4. Structure of actuator (unit: mm)

When pressure is applied to the chamber, it changes shape, forming a bump on the actuator surface. The biggest allowed pressure is 0.3 MPa. If chambers are pressurized sequentially, elastic waves of various wavelength and frequency can be produced (Figure 5). Fins are formed on the face of the actuator, thereby amplifying surface deformation and wave motions as well. Each fin tip draws a polygon-like locus, altogether generating force to move the foreign objects in contact. Direction of movement is opposite to the wave propagation. If placed against another surface, actuator will move with certain speed, which is dependent on the elastic wave properties. Combined with a smart digital control, various types of waves can be generated, and this is the main principle of robot movement.

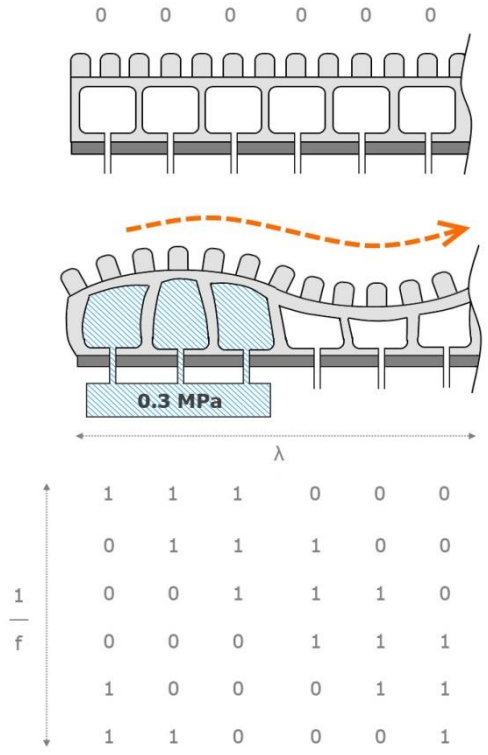


Figure 5. Actuator driving principle

Actuator deformation is closely studied in the “Pneumatic Rubber Actuator Driven by Elastic Travelling Waves” paper by Koichi Suzumori [1], where optimal driving characteristics of the actuator are experimentally found. Effective fin tip motion, in other words, its maximum displacement in parallel to the surface of the actuator, is named

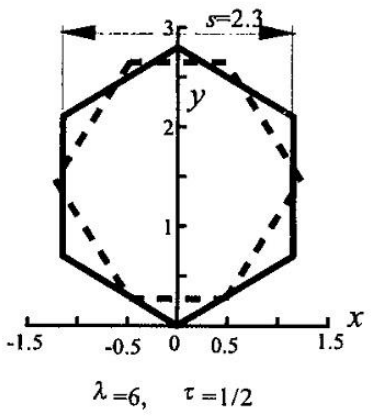


Figure 6. Fin displacement pattern (unit: mm)

step feed s (Figure 6). According to analysis, the elastic deformation occurs only within three empty chambers from pressurized chamber. Because of that, step feed becomes saturated at wavelength of 6 chambers. Optimal duty ratio of the driving pattern is found to be 0.5, as larger values are constricting the fin locus, reducing full swing of its tip. Taking into account chamber inflation pneumatic response, actuator travelling velocity is modelled by the first-order delay, and given as

$$v = \frac{\mu s f}{1 + (2\pi f T)^2}$$

where T and f represent the time delay constant and driving frequency respectively. μ represents the efficiency of travelling motion, which depends on the slipping between the actuator and travelling surface.

Experimental results show disagreement in actual velocity and predicted values. Speed of movement reaches its peak at the driving frequency of about 0.9 Hz, and decays

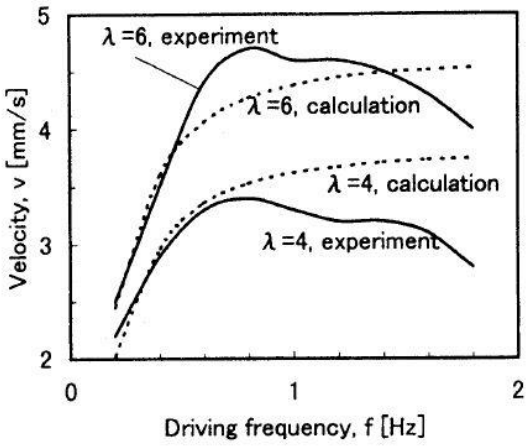


Figure 7. Velocity against driving frequency

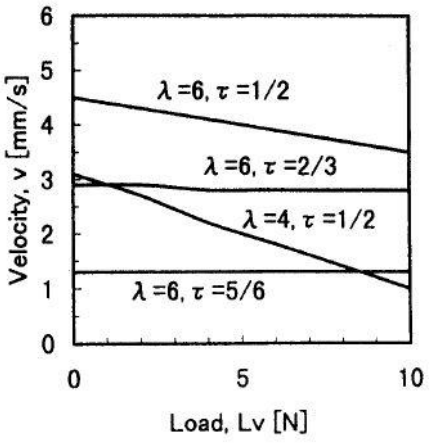


Figure 8. Velocity against vertical load

with further increase of frequency (Figure 7). If various types of loads are applied to the moving actuator, decrease of the speed is linear in every case. Most effective against vertical load (Figure 8), this 48.4 gram actuator has been reported to have load-to-weight ratio in the order of $40:1$.

Report demonstrates that optimal operation of the actuator is an even wave of six chambers' length with frequency of 0.9 Hz. With 12 chambers in total, selected wavelength pattern is periodical, and chamber pressurizing pipes are paired accordingly, i.e. 1st and 7th, 2nd and 8th and so on (see Section 6.2).

3.2 Locomotion

3.2.1 Rectilinear locomotion

Rectilinear locomotion is a common gait of caterpillars, which is also used by maggots, snakes and earthworms. Many studies about this type of locomotion in nature were conducted and published, earliest indicating in 1812 by Everard Home. Natural crawlers are travelling by producing elastic waves on the body. Waves are generated by means of rib muscle in the case of snakes, or "hydraulic skeleton" in the case of caterpillars (Figure 9).

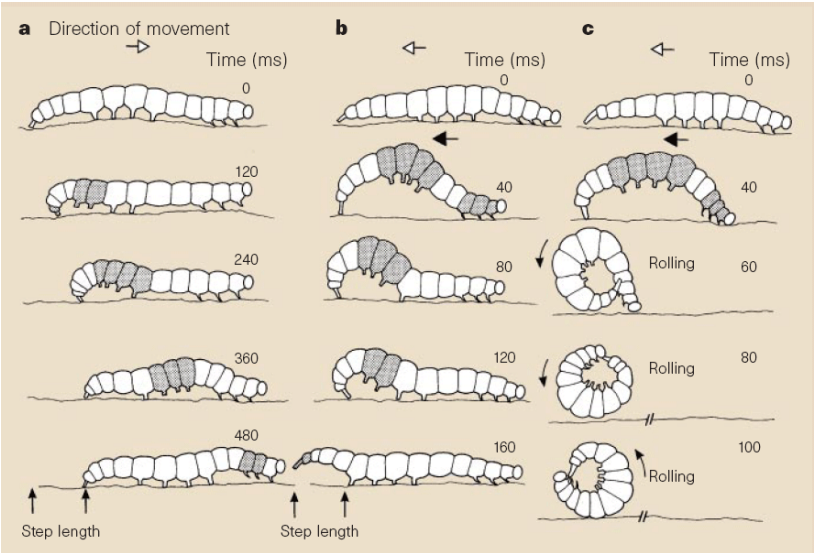


Figure 9. Caterpillar locomotion (source: Nature journal [2])

As can be deducted from the sequence, movement and wave directions are the same, while travelled surface is obviously pushed to the opposite direction. Contracted part is lifted above the surface, and extended part is pushed against, generating thrust motion. This mode of walking with many small legs is indeed similar to the fin movement of the given actuator. Snakes implement reversed pattern by extending the lifted parts and pushing the ground back with compressed part, resulting in the elastic waves in opposite direction to the path.

This snake gait is extensively described in the “Snakes mimic earthworms: propulsion using rectilinear travelling waves” paper, Interface journal 2013 [3]. In this study, some similar to the actuator travelling properties were experimentally found, for example, almost immediate transition from rest condition to the full speed, and same fast deceleration as well. It is also noted, that rectilinear locomotion benefits from climbing motion, which is indicated by increase of speed at inclined rough surfaces. No data was collected regarding unusual surfaces like sand, because snake use other types of locomotion for that case. For kinematic analysis, the reptile body was modelled by discretizing the body into nodes of equal mass. Model is then compared to the experimental data collected from different snakes. It is reported, that wavelength does not affect the travelling speed, while bigger amplitude yields bigger velocity by logarithmic dependence. Concerning optimality of rectilinear wave frequencies, emphasis is put on the Froude number, which is used to compare inertia to friction and gravity and defined as

$$\mathrm{Fr} = \frac{A}{\mu_b \tau^2 g}$$

where A is the wave amplitude, μ_b is the backward friction coefficient, τ stands for the wave period and g is the acceleration due to gravity.

3.2.2 Froude number

According to Wikipedia,

> The Froude number is the ratio of the centripetal force around the center of motion, the foot, and the weight of the animal walking:
>
> $$\mathrm{Fr} = \frac{centripetal\ force}{gravitational\ force} = \frac{mv^2/l}{mg} = \frac{v^2}{gl}$$
>
> where m is the mass, l is the characteristic length, g is the acceleration due to gravity and v is the velocity. The characteristic length, l, may be chosen to suit the study at hand. For instance, some studies have used the vertical distance of the hip joint from the ground, while others have used total leg length.
> The Froude number may also be calculated from the stride frequency f as follows:
>
> $$\mathrm{Fr} = \frac{v^2}{gl} = \frac{(lf)^2}{gl} = \frac{lf^2}{g}$$
>
> If total leg length is used as the characteristic length, then the theoretical maximum speed of walking has a Froude number of 1.0 since any higher value would result in 'take-off' and the foot missing the ground.

In the case of rectilinear locomotion, total leg length was replaced by the amplitude, with wave being similar to the infinite set of legs stepping on the surface, and friction coefficient was added to account for friction ratio.

Using the previous actuator experimental results [1] at optimal driving, namely, amplitude of 2.75 mm, and friction coefficient of 0.5, the maximum allowed period of wave propagation can be found by solving for τ:

$$\tau > \sqrt{\frac{0.00275\ m}{0.5 \cdot 9.81\ {}^{m}/_{s^2}}} \approx 0.0237\ s$$

which means, that maximum beneficial driving frequency of actuator is 42 Hz, and above that actuator fins will start to slip. It is clearly not the reason why factual velocity is decaying, because it does so already at average periods. However, according to the model, optimal frequencies of heavy snakes are at way lower than unity Froude numbers [3]. Reading the Figure 10, it is about 1 Hz for Dumeril's Boa specimen, when travelling speed peaks. For larger frequencies velocity is somewhat lower due to suboptimal Froude number.

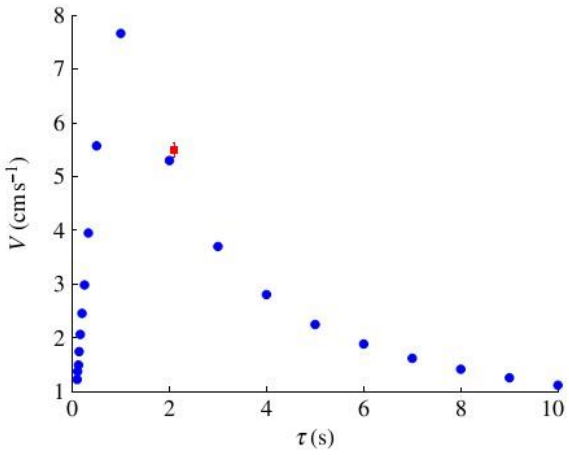


Figure 10. Velocity against wave period from Dumeril's Boa kinematic model

3.2.3 Pneumatic actuator system

Main reason for early slowing down in the pressure actuator system is the slow pneumatic response to the pressure input frequency. Chambers do not keep up with fast driver operation, restricting the step feed and dramatically decreasing speed. Second problem can be the pressure loss in the pneumatic system. Flow demand from the actuator has linear dependence on the actuator operating frequency, as the same number of chambers should be inflated and exhausted in shorter time. Rising pressure drop in the flow restricting valves and pipes affects chamber deformation, reducing the full swing. There is also hypothesis of harmonic oscillations in periodical motion of caterpillars, mentioned in "Kinematics of Soft-bodied, Legged Locomotion in Manduca sexta

Larvae” analysis, Biological Bulletin 2007 [4]. The critical damping, caused by actuator membrane during fast oscillations, and natural frequency of the periodical elastic inflation-exhaust system could explain peaking and falling velocity behavior.

More advanced and complex actuator characteristic model would definitely predict its kinematics more precisely. It can be done using higher order delay models, following the assumption in original research [1] or by nodal method, like used for snake analysis [3]. For now, regarding travelling, conclusions are:

- Velocity versus frequency is almost linear until 0.9 Hz, use of higher frequencies reduces velocity
- Amplitude will improve speed and can be slightly increased with applied pressure, but it is always a trade-off with air depletion
- Wavelengths other than 6 chambers are not optimal for normal surface
- Climbing up inclined plane is likely to benefit velocity
- There is no other way to improve locomotion profile without altering the actuator

3.3 Bubbler controller board

3.3.1 Overview

Bubbler controller board consists of 1.6 mm thick glass-epoxy base, with copper traces on the top side, and adhesive trace layer at the bottom. Two connectors are mounted to the edge; 6-pin connector is for control panel cord, and 2-pin connector for power.

Two CKD solenoid pneumatic valve arrays are fixed on the top side (Figure 11). They are also connected with two bubbler actuators mounted at 1 cm beyond the bottom side (Figure 12 and 13). Total weight of the device is 354 g. Control panel is presented by a metal box of about same size, with four directional buttons on the face (Figure 14).

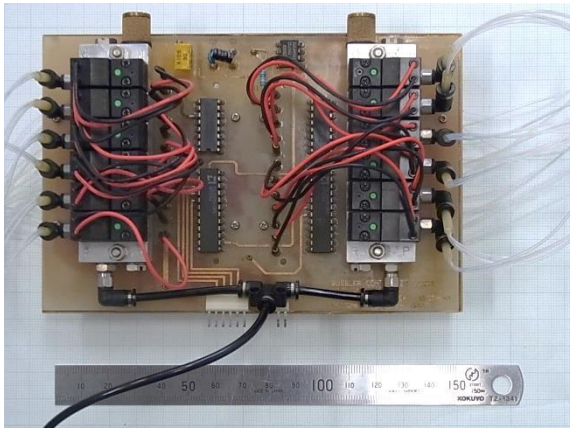


Figure 11. Bubbler controller board top

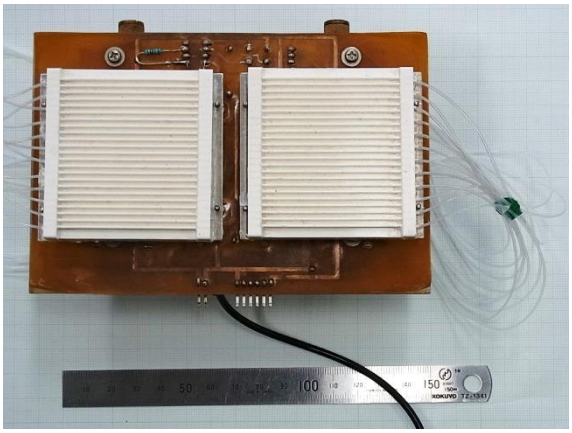


Figure 12. Bubbler controller board bottom

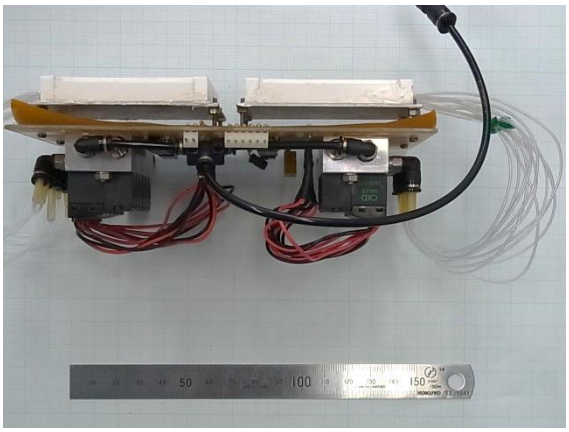


Figure 13. Bubbler controller board front

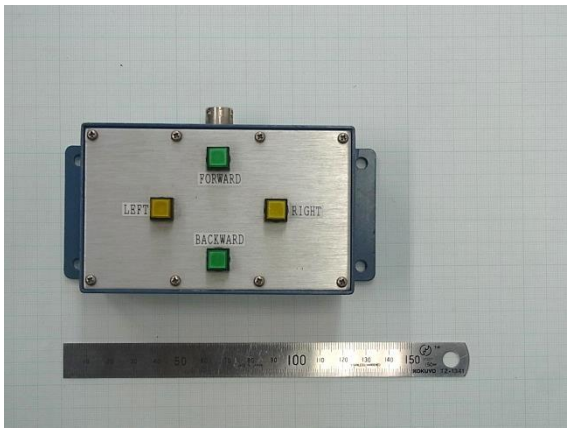


Figure 14. Control panel

Board is powered with 5 Volt DC and a pneumatic pressure of range 0.15 - 0.3 MPa. When FORWARD or BACKWARD buttons are pressed on the control panel, backward or forward travelling waves are simultaneously generated on the actuators. When LEFT or RIGHT buttons are pressed, waves are generated in the reciprocal way such that board rotates counterclockwise or clockwise respectively. When two buttons are pressed at the same time or no buttons are pressed, actuators are exhausted immediately and operation is halted. Board draws current of 0.3 mA in standby mode and 1.35 A during valve operation.

3.3.2 Controller logic

Control logic is presented by two Programmable Array Logic (PAL) integrated circuits (Figure 15). Such types of chips consist of input, programmable logic plane and output

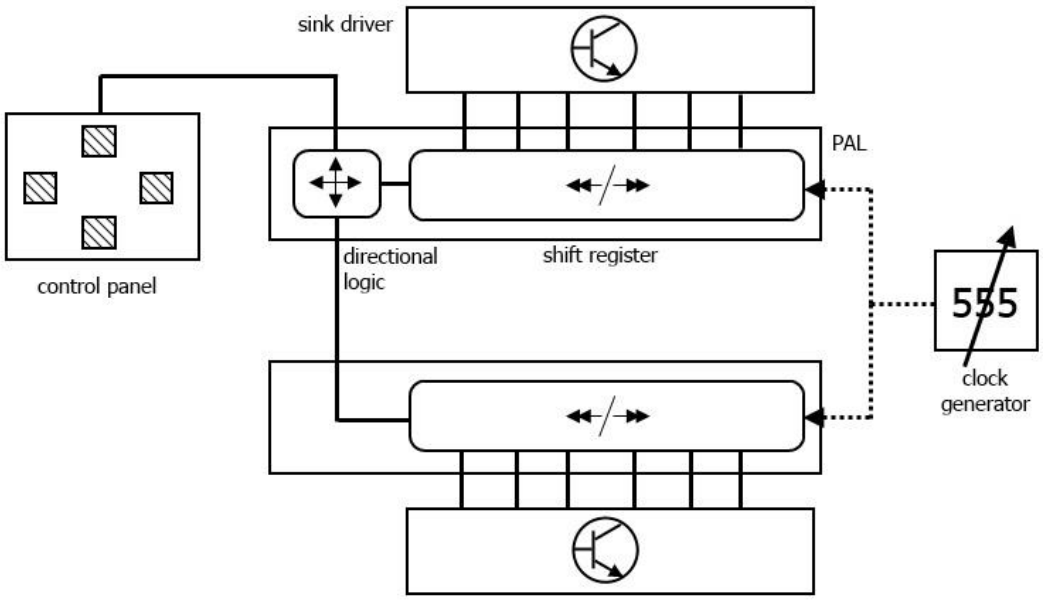


Figure 15. Bubbler controller board block diagram

macro cells. With operation principle similar to FPGA, PAL was programmed by Hardware Description Language (HDL) Verilog. Operation of the logic is synchronous, and clocked by 555 timer. Timing period RC circuit includes potentiometer, which can be used to set various driving frequencies.

Four of the six control panel traces are connected to one of the chips. Each directional button, when pressed, puts active low on the corresponding PAL input. Directional logic, most probably implemented as a function by truth table, decodes the direction into logic instruction for both arrays, and inhibits action when more than one button is pressed. The driving pattern is implemented with bidirectional shift register, which by definition needs only two bit instruction to operate with observed functionality, one bit for direction and one bit for enable/disable shifting. Outputs energize solenoid valves by means of Darlington sink drivers.

Travelling experiments are consistent with data presented in the Section 3.1 Bubbler actuator. It is also observed, that pressurizing pattern resets every time button is released, and is the same for both PALs. Massive board of superior size, elevated above actuator, is affecting overall device balance. Instant, when both actuators have zero amplitude from one end, results in slight board swinging during start pressurizing pattern 111000 and 000111 as well. Yet, when with maximum operating amplitude of 4mm

at 0.3 MPa, there is no threat to the stability of the moving system, but only possibility of small unwanted oscillations.

With this actuator layout similar to the caterpillars of tank, Bubbler controller board has 3 degrees of freedom. Having the optimal wavelength shorter than actuator more than two times, there is no necessity to place two actuators in series. Mobile robot with similar valve and actuator layout will benefit from symmetrical mass distribution and individual driving of caterpillars.

4 Firmware design

4.1 Microcontroller

4.1.1 Atmel ATmega328P

Robot remote control system should consist of two smart elements, human interface controller, and an actuator driver. The latter in our case deals with 12 solenoid valves at relatively low frequencies, and should connect with the controller through wireless module. It is most likely for radio module to be interfaced with widely used high-speed Serial Peripheral Interface (SPI), with a number of pins greater than 3. Including various power and reset connections, pin count of the driver should be more than 20, while other parameters' range is wide. Considering limited PCB manufacturing capabilities, and for the ease of prototyping, microcontroller in DIP through-hole package is the best choice, and can be drop-in replaced in case of failure.

ATmega328P 8-bit microcontroller by Atmel was selected. While offering standard set of useful features, it has 28-PDIP version (Figure 26) and is relatively cheap. It is also quite popular, and has a lot of related information and discussions on the Internet. Both interface and driver can be implemented using similar chips, ensuring high code compatibility.

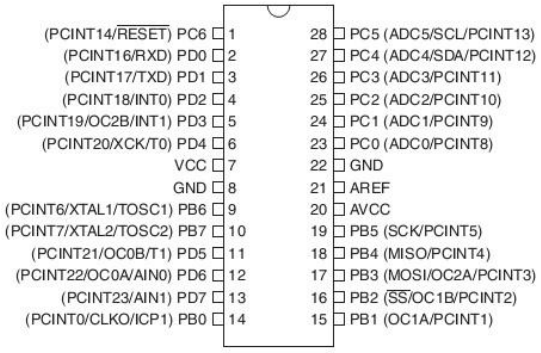


Figure 16. ATmega328P DIP configuration

Features of interest (ATmega datasheet [5]):

- 8 MHz operation at 3 V
- 32KBytes of In-System Self-Programmable Flash program memory
- 1KByte EEPROM
- On-chip 2-cycle Multiplier
- 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
- 6-channel 10-bit ADC
- Master/Slave SPI Serial Interface
- Byte-oriented 2-wire Serial Interface
- Programmable Brown-out Detection
- Operating voltage range of 1.8 - 5.5 V

4.1.2 Developer tools

Atmel Studio 6 was used as a programming and debugging environment (Figure 17). It has a lot of useful features, and compatible with all Atmel microcontrollers. It is possible to use C, C++, or assembler to create firmware. In this project, capability of C programming language is sufficient to create all necessary robot control algorithms.

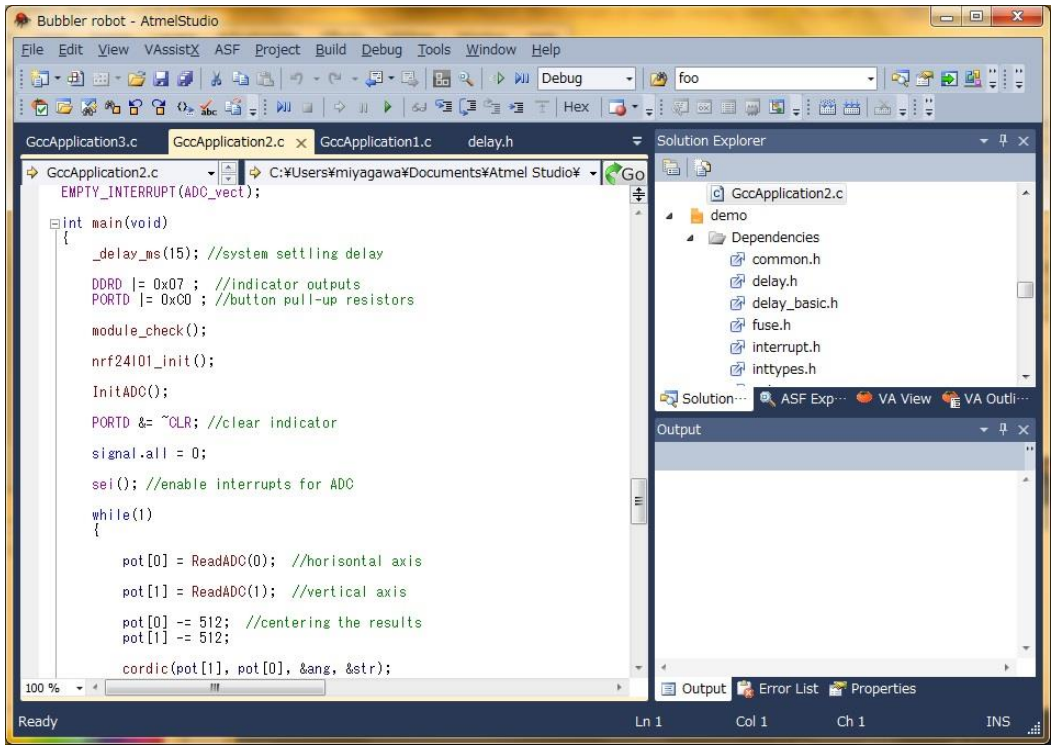


Figure 17. Atmel Studio 6

Most useful features at this level of programming:

- Fast navigation to the function or variable origin
- Type suggestions
- “Start without debugging” button, fast build and upload with single click

Atmel Studio uses avr-gcc compiler chain to build the C code, which can be then programmed into chip memory with a programmer device. Cheap and simple AVR Dragon is chosen as programmer/debugger (Figure 18). It is connected to the PC using an USB cable and is capable to program ATmega328P with several distinct programming modes. DebugWire programming was used, as a most convenient mode for prototyping.

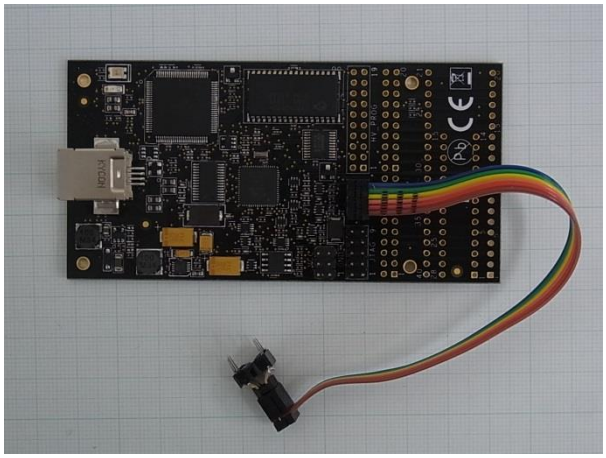


Figure 18. AVR Dragon with improvised prototype board connector

4.1.3 ISP and DebugWire

In-System Programming (ISP) is one of programming modes supported by Atmel microcontrollers. With this mode, all chip memory and registers can be accessed without ejecting integrated circuit from the PCB. Similar types of programming are widely used by other manufacturers (Texas Instruments, Cypress, etc.). ISP is based on the SPI connection protocol and consists of six signal lines. Smallest ISP connector is 3x2 male, 2.54 mm pitch, with pin layout described in the Figure 19. Same standard connector is present on the AVR Dragon, allowing to couple programmer to the board with one cable.

MOSI, MISO, SCLK pins are connected to the SPI lines of ATmega328P. VTG and GND pins are used to detect in-system operating voltage for correct programming operation. RESET pin becomes bidirectional DebugWire data line during programming and debugging process, halting execution of program in Flash and giving instructions to internal programming logic of ATmega.

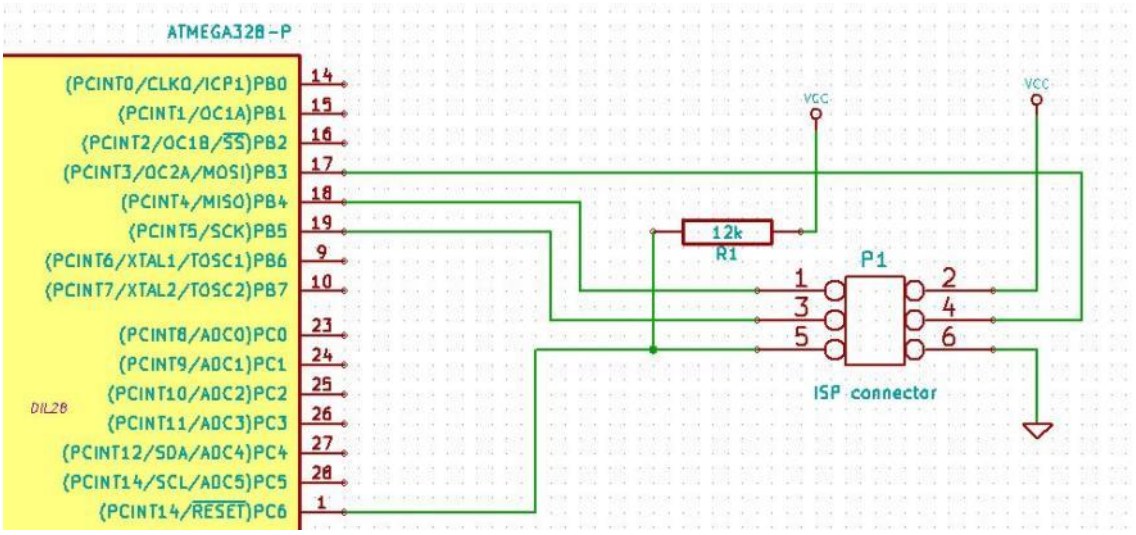


Figure 19. In-System Programming interface

The actual problem I have encountered during debug process is that In-System Programming circuitry interferes with the SPI communication of ATmega and other modules. In the same way, modules could interfere with programming operation, if accidentally activated. For this type of complex situations, official proposed solution is to isolate ATmega ISP connection from main lines with resistors [6]. This allows the difference of logic voltages on different ends of the line, leaving voltage drop on the resistor.

To avoid complicated design, various operation patterns of AVR Dragon were studied. In fact, during debugging process, only DebugWire data line is active, and VTG and GND connections are necessary. This workaround removed mentioned problems, and greatly helped in troubleshooting of wireless modules. Moreover, prior to debug sequence, code is compiled and uploaded to the chip using only RESET line. It is also possible to execute only compile-upload sequence, like said in the Chapter 4.1.2. Only one drawback of such light DebugWire programming is found. Fuse registers, used to store important settings, cannot be accessed or programmed in this way, and should be changed in ISP.

4.1.4 MSP430 Launchpad

Prior to the ATmega, part of the robot algorithms was designed and tested using LaunchPad development board by Texas Instruments (Figure 20). It features 16-bit MSP430g2553 microcontroller, which has list of features similar to that of ATmega328P. Programming environment used was TI Code Composer Studio. Compiled code is uploaded on the chip using TI proprietary Spy-Bi-Wire interface, which uses RESET line in a way similar to DebugWire.

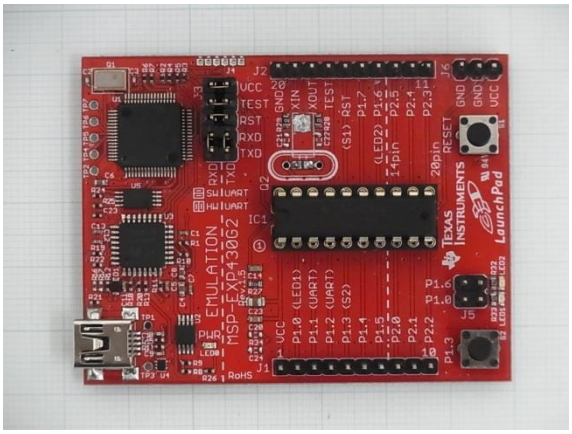


Figure 20. Texas instruments Launchpad

4.2 Wireless module

4.2.1 NRF24L01

The nRF24L01+ wireless module is a small breakout PCB for the one-chip transceiver of the same name by Nordic Semiconductor. Module has meander style PCB trace antenna and all components necessary for the radio operation (Figure 21). Module is interfaced with 4x2 2.54mm pitch male connector, and addressed through SPI. This makes module easy for prototype and replace in case of failure. It also has reasonable price among other wireless solutions on the market.

Features of interest (nRF24L01 datasheet [7]):

- Worldwide 2.4GHz ISM band
- Ultra low power operation
- 1.9 to 3.6 V supply range
- Auto packet transaction handling
- Auto acknowledgment feature

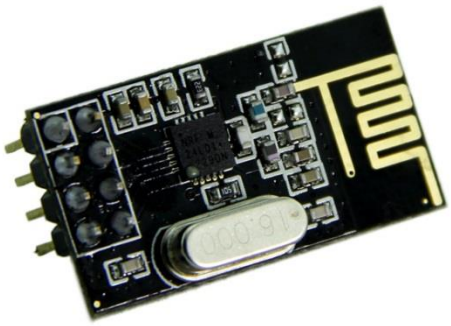


Figure 21. nRF24L01+ module

In addition to the V_{CC} and Ground lines, breakout board connector is tied to the data and control interface pins of the radio chip [7]:

> The data and control interface gives you access to all the features in the nRF24L01. The data and control interface consists of the following six 5Volt tolerant digital signals:
> • **IRQ** (this signal is active low and is controlled by three maskable interrupt sources)
> • **CE** (this signal is active high and is used to activate the chip in RX or TX mode)
> • **CSN** (SPI signal)
> • **SCK** (SPI signal)
> • **MOSI** (SPI signal)
> • **MISO** (SPI signal)

Communication through SPI is performed in conventional way (Figure 22). After initiating transmission by putting CSN line low, microcontroller clocks the SCK line and shifts

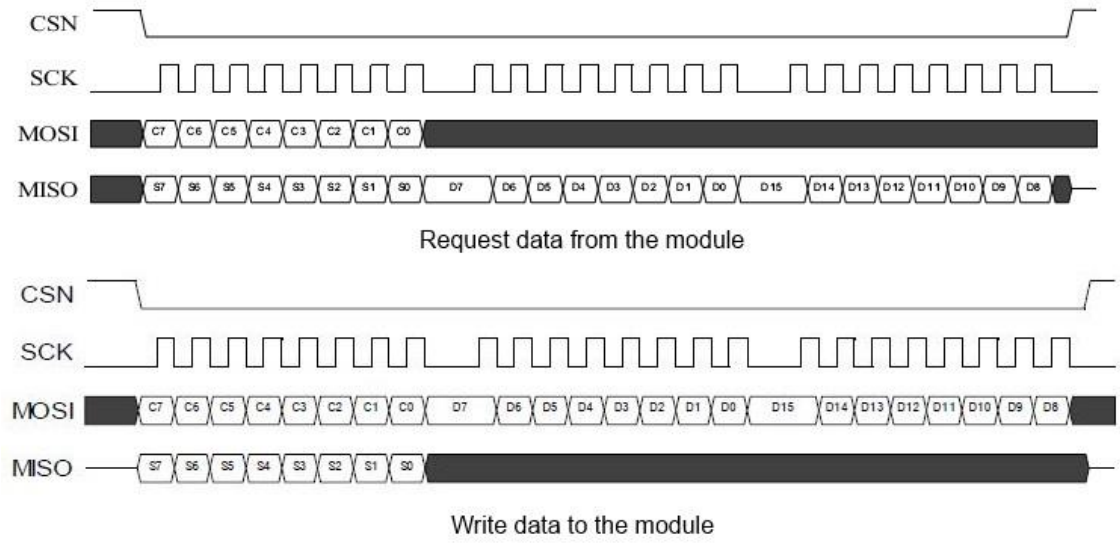


Figure 22. SPI communication with nRF24L01+ module

command bits C_n on the MOSI line. In the same instant, STATUS register bits S_n are shifted onto the MISO line, followed by data bits D_n [7].

4.2.2 Radio control functions

To handle commands, radio configuration and control routines I used nRF24L01 control library adapted for Atmega microcontrollers [8]. Such hardware abstraction allows sending and receiving data using only few functions, included in the library:

```
void nrf24l01_init();  //preliminary configuration of radio module
uint8_t nrf24l01_readready(); //check for received data available
void nrf24l01_read(uint8_t *data); //read received packets from the module
uint8_t nrf24l01_write(uint8_t *data); //write data to the module and send by radio
```

With this library, it is also possible to use Auto acknowledge feature, embedded in nRF24L01 firmware. When both radio modules are configured for acknowledge, after successful transmission receiver automatically sends acknowledge packet back. In other case, transmitter waits certain amount of time before resending data or giving transmission failure flag.

NRF24L01 library functions use ATmega SPI peripheral to communicate with the radio module, with all serial operations wrapped in a small function:

```
uint8_t spi_writereadbyte(uint8_t data) {
    SPDR = data; // load serial shift register with data
    while((SPSR & (1<<SPIF)) == 0); //poll CPU until transmission finished
    return SPDR; // return received data from the shift register
}
```

This is a simple one-byte duplex function, based on the ability of SPDR register to shift MOSI data out from one side and load MISO data from another end at the same time.

Based on the SPI function, troubleshooting feature has been implemented on both robot and remote control systems. Function *module_check* sends dummy data to the module to confirm adequate response, and is used before initialization of the radio settings:

```
void module_check(){
            uint8_t test;

            spi_init(); //initialization of SPI peripheral

            nrf24l01_CSNlo;  //put CSN line low

            _delay_us(10);  //wait for the nrf24l01 MCU to react on boot

            test = spi_writereadbyte(0x00);  //send dummy data

            if ((test == 0x00)||(test == 0xFF)) {

                        PORTD |= 0x03;  // error message
                        while(1);      // trap CPU
            }
            nrf24l01_CSNhi;  //put CSN line high
}
```

Shifting zero byte to the MOSI line will generate clock signal on the SCK without giving any command to the transmitter MCU. Like it was explained earlier in Chapter 4.2.1, STATUS register bits are shifted automatically with the first 8 clock cycles. According to the datasheet [8], STATUS register default value is 0001110. There is possibility of bit slip during first transmission between ATmega and radio module. Because of that, first value of STATUS register is compared to the all zero and all one. If device sends something other than that, it is considered operational. If not, CPU is captured in infinite loop and error message is displayed. Error also indicated if module is not present in in the system (all zero response).

4.3 Remote control

4.3.1 Architecture

Purpose of the remote is to gather data and send it to the driver system. Architecture of the remote is presented in Figure 23. Human interface of robot steering is implemented

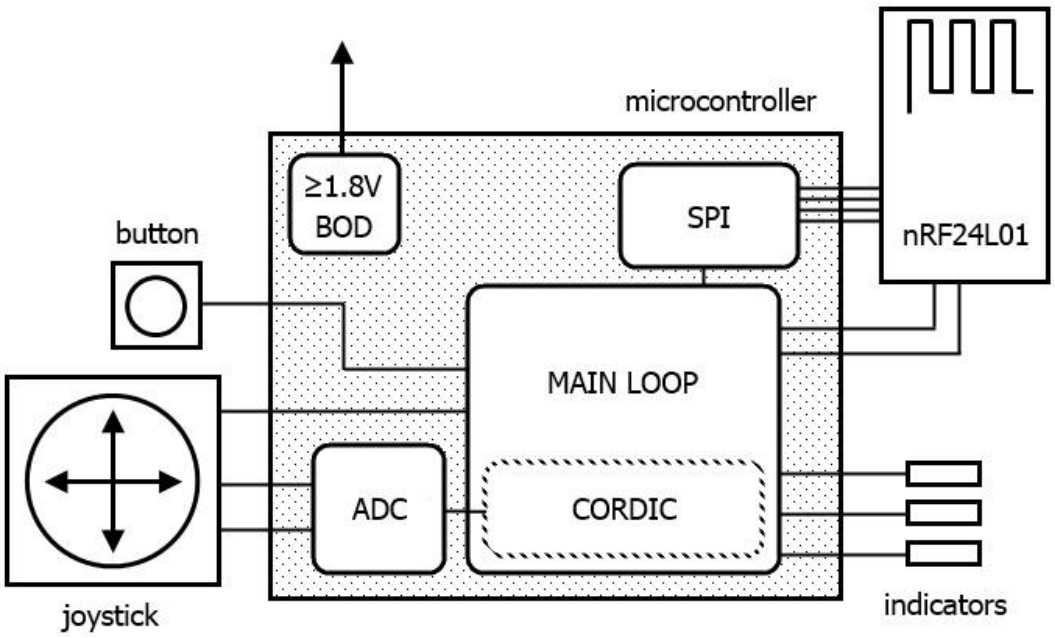


Figure 23. Remote control architecture

using thumb joystick with two degrees of freedom. Joystick also has a button which is activated if shaft is pressed down. It is used to shuffle the dynamic range of actuator driving frequency. Current frequency mode is displayed at the indicator (Figure 24). Additional button is used to reset the gas tank gauge, which is placed on robot.

Joystick data is sampled with Analog-to-Digital Converter (ADC) and processed with the CORDIC algorithm. Depending on the frequency setting, information is modified and then fed to the RF transmitter through SPI.

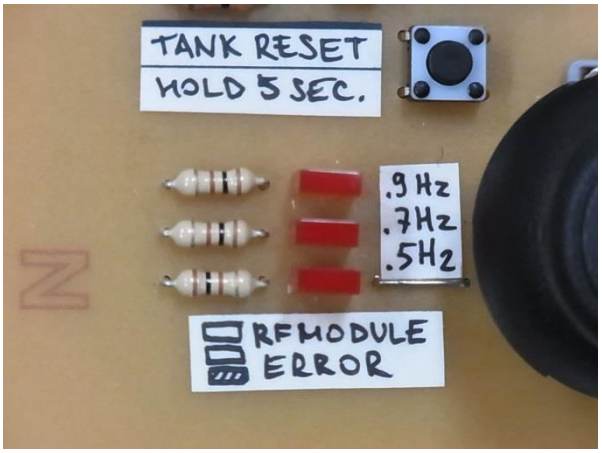


Figure 24. Remote control indicator

Internal Brown-Out Detector (BOD) is used to hold the microcontroller in reset state during periods of unacceptably low voltage (brownouts). Generally, if voltage is less than specified operational minimum of 1.8 V, ATmega328P CPU might act unpredictably [5]. Because remote control is powered by pair of alkaline batteries, BOD is set to 1.8 V to inhibit probable low-voltage operations. The BOD settings are defined in fuse registers, separately from Flash program memory, and cannot be changed during runtime.

4.3.2 Main loop

Remote control program is an infinite repeating sequence implemented with while-loop. Manual user input is sampled and sent to the driver 430 times a second, allowing fast response of the system. Periods of while-loop execution in active and standby mode are equal to 2.315 ms and 14.50 ms respectively. Loop period is used as a time increment for events longer than maximum loop duration. There is a small decrease in the ATmega clock frequency when batteries wear out, so loop period is a subject to change.

Program start is followed by startup delay of 15 milliseconds, allowing the system voltage and CPU clock settle. NRF24L01 module has startup time of 10.3 ms, stated in the datasheet [7]. 15 ms should be enough before executing function *module_check*, described in Chapter 4.2.2. If module does not respond accordingly, error message is displayed in the indicator, and program execution is stopped until system is reset. This is done before initialization of RF module, preventing the system from hanging during runtime. Together with nRF24L01 module, ADC is set up, and system interrupts are enabled (see Chapter 4.3.3 for more information). Flowchart of the main loop is presented in Figure 25.

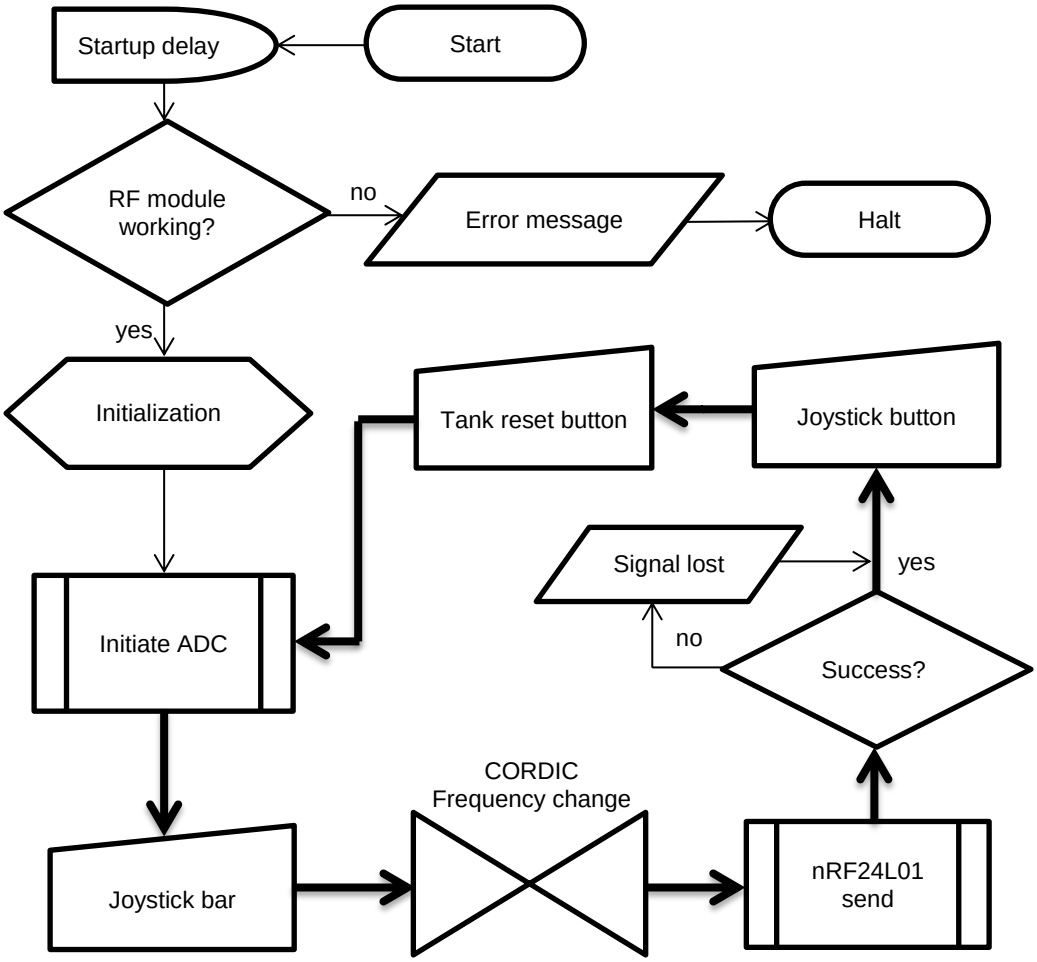


Figure 25. Remote control program flowchart

Transitions drawn with strong lines do happen most often; thin arrows are indicating rare operations. Joystick reading and processing algorithms are closely described in the following Chapters 4.3.3 – 4.3.5.

Function *nrf24l01_write* from the radio module library is used to send *buffer* data to the driver. When output of this function equals one, it means Auto acknowledge packet is received, and transmission was successful (Chapter 4.2.2). *Ack* flag, which stores this information, is checked with if-statement. In case of successful transmission, port D indicator of corresponding state is lit up.

```
ack = nrf24l01_write(buffer);

if(ack){
              PORTD |= state;  //set output bit high
              delay.lost = 0;
              }
else {
              delay.lost += 1;
              if (delay.lost > 60){
                            PORTD ^= state;  //invert output bit
                            delay.lost = 0;
              }
}
```

When transmission is not acknowledged, *lost* counter is incremented with every loop rotation. If *ack* equals zero for more than 60 loop iterations, state indicator will blink with periods of $60 * 14.50\,ms = 870\,ms$. Extended loop period means that radio module is in standby state waiting for receiver.

Tank reset signal is activated in the same fashion to protect air gauge data from accidental erasure.

```
if(!(PIND & 0x40)){

              delay.reset += 1;
              if (delay.reset > 1000){signal.reset = 11;}
}
else {
              signal.reset = 0;
              delay.reset = 0;
}
```

If tank reset button (Figure 24) is active low for over thousand cycles, reset instruction is generated for next transmission. Button should be pressed somehow longer, than calculated value of $1000 * 2.315\,ms = 2.3\,s$. To avoid user confusion, 5 second time is recommended.

Similarly to the counters, additional instructions to the driver module are combined into one byte using bit field notation.

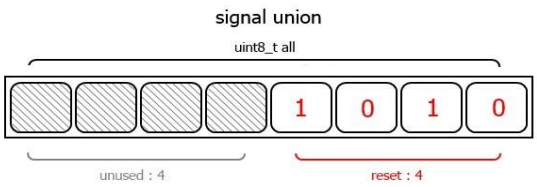


Figure 26. Instruction byte

```
union control
{
		uint8_t all;
		struct
		{
				uint8_t reset:4;
				uint8_t unused:4;
		};
};
union control signal;
```

It is more difficult for hardware to generate long bit pattern accidentally. Because of that, reset instruction is four bits long and is addressed under the *signal* union (Figure 26). Unused bits are reserved for future use. Whole instruction byte is named *all*, and is used to address all instruction bits at the same time.

Pressing the joystick button shuffles *state* variable. To make program react to one button press at a time, functionality similar to the flip-flop is carried out:

```
if(!(PIND & 0x80)){ press = 1; }
if((press)&&(PIND & 0x80)){

		press = 0;
		PORTD &= ~CLR;

		switch (state){
				case DOTFIVE:{
						state = DOTSEVEN;
						break;
				}
				case DOTSEVEN:{
						state = DOTNINE;
						break;
				}
				case DOTNINE:{
						state = DOTFIVE;
						break;
				}
		}
}
```

When button is pressed, appropriate *press* flag is set. Starting from next loop rotation, button pin will be checked for a release. Minimum observed button press duration is 30 ms. This means, that there practically no possibility of skipping a button press even during the standby mode with period of 14.50 ms.

4.3.3 Joystick and ADC

By tilting joystick to different directions, information about direction and speed can be collected by device. Simple joystick (Figure 22) was taken out of game controller for this purpose. Joystick shaft is connected with two rotating rings to the potentiometers. Coordinates of the tip can be estimated by reading voltage on the middle pin of each potentiometer. ATmega328P has an ADC with 10-bit resolution, which means that full range of one axis is 1024 logic levels. Horizontal and vertical axis potentiometers are connected to the ADC channel 0 and channel 1 respectively. ADC peripheral is configured with *InitADC* function.

Figure 22. Potentiometer joystick

```
void InitADC()
{
		ADMUX |= (1<<REFS0); // AVcc is selected as reference
		ADCSRA |= (1<<ADPS2)|(1<<ADPS1)|(1<<ADIE)|(1<<ADEN);  //ADC clock prescaler
		factor of 64, ADC interrupt enable, ADC operation enable

		DIDR0 = 0x03; // disable digital input on first 2 ADC channels
		set_sleep_mode(SLEEP_MODE_ADC); //selecting and enabling ADC noise reduc-
		tion mode
		sleep_enable();
}
```

ADC Noise reduction mode is used during conversion. It means that processor is put to sleep mode to inhibit high-frequency noise in the analogue circuitry. Empty ADC interrupt vector is automatically accessed on the end of conversion to exit sleep mode, and ADSC flag is cleared. *ReadADC* function is used to read voltage level on the selected ADC channel.

```
EMPTY_INTERRUPT(ADC_vect);

uint16_t ReadADC(uint8_t ADCchannel)
{
		ADMUX = (ADMUX & 0xF0) | (ADCchannel & 0x0F); //Channel is applied to the
		control register through safety mask and operation is started

		do {sleep_cpu();} while( ADCSRA & (1<<ADSC) ); //Processor will return to
		sleep for any interrupt except ADC

		return ADC;} //Processor woke up, return conversion value
```

Raw ADC readings of joystick at rest will always be around 512, because potentiometers in such case are set in the middle, dividing the operating voltage by half. This point is set to be an origin of the joystick coordinate system by subtracting 512 from every reading (Figure 23). To simplify steering algorithm, direction and magnitude of the joystick tilt is encoded with Polar system. In order to convert joystick coordinates, CORDIC algorithm is utilized and described in the Chapter 4.3.4. CORDIC function collects information about signs and magnitude of x and y. Angle in a range of [-180° ; 180°] and hypotenuse are addressed to the function as pointers.

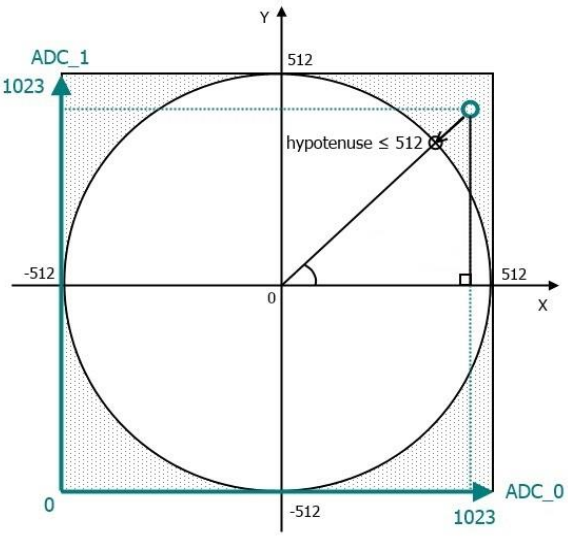


Figure 23. Joystick coordinate system

For symmetrical encoding of steering, new angle convention was implemented. It means that zero angle reference corresponds to the positive Y axis, and interpreted as forward motion, while negative Y direction is backward motion. Then, positive and negative angles are for rightward and leftward directions accordingly. To get such angle from Cartesian coordinates, values are fed into CORDIC function in reverse order, transposing the coordinate frame. Variables *ang* and *str* are used for direction angle and incline strength respectively. To take two output values out simultaneously, pointers to mentioned variables' addresses are passed to the function as arguments:

```
pot[0] = ReadADC(0);
pot[1] = ReadADC(1);

pot[0] -= 512;
pot[1] -= 512;

cordic(pot[1], pot[0], &ang, &str);

str = str>512 ? 512 : str;
```

In order to keep the vector magnitude the same for every joystick position, distance larger than 512 is cropped to the 512 by subtraction, forming fine polar circle of allowed values, as shown on Figure 23.

4.3.4 CORDIC

CORDIC algorithm is used to approximate coordinates and vectors using iterative search, and in some way resembles bisection method of finding function roots. At first, CORDIC was utilized during the preliminary robot control prototyping. MSP430g2553 does not have hardware multiplier, which makes it difficult to perform trigonometric operations without additional research. Mentioned algorithm uses bit shift, subtract and add operation to rotate vectors, and is widely used in hardware-constrained applications. Code implementation for MSP 430 was found at TI support forum [9] and used for joystick direction calculation as arctangent function.

CORDIC algorithm was migrated to the ATmega robot control program for research purposes, and has shown satisfactory results. Function Atan2, included in the Atmel Studio math library, computes angles with great amount of precision. Experiments show, that it takes from 362 μs to 564 μs to calculate arctangent and hypotenuse from joystick readings, using math library. CORDIC operates in a fixed point format and always takes equal time of 234 μs to process same values with precision of two decimal places. This is beneficial in both speed and stability; also, further accelerating of algorithm is possible by allowing bigger error.

At start of CORDIC operation, resulting angle equals zero. Vector p_0, defined by input, is repeatedly rotated towards x axis with steps of decreasing magnitude α_n (Figure 24, Coranac.com [10]). To rotate vector, endpoint coordinates are reciprocally scaled using bit shift (p_1, p_2, p_3). Arctangent result is accumulated every iteration by values of α, defined in the lookup table ($\alpha_0 + \alpha_1 - \alpha_2$ ... etc). Eventually, desired precision is achieved.

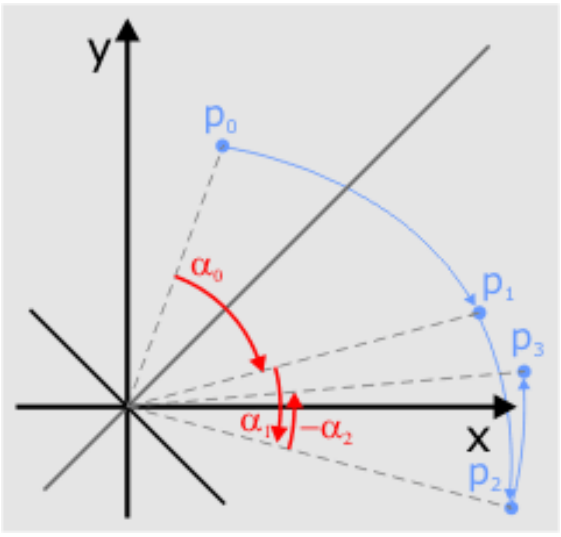


Figure 24. Angle approximation with CORDIC

Angles in the approximation look up table are pre-calculated arctangent values. For fixed-point bit operation, they are presented in 16-bit offset binary. Here, this notation splits range of [90; -90] degrees into 2^{16} val-

ues of the same size [11]. To convert angle in degrees to the offset binary, we can use the following function: $B(\alpha) = {}^{\alpha \cdot 2^{16}}/_{180} = \alpha \cdot 182$.

```
const signed int atanAngles[14] = {
                0x12E3, //atan(2^-1) = 26.5651
                0x09FB, //atan(2^-2) = 14.0362
                0x0511, //7.12502
                0x028B, //3.57633
                0x0146, //1.78981
                0x00A3, //0.895174
                0x0051, //0.447614
                0x0029, //0.223808
                0x0014, //0.111904
                0x000A, //0.05595
                0x0005, //0.0279765
                0x0003, //0.0139882
                0x0002, //0.0069941
                0x0001 //0.0035013
};
```

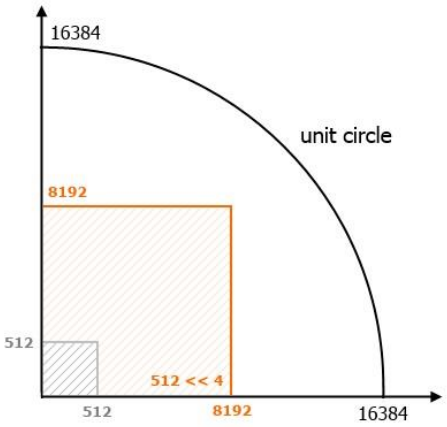


Figure 25. CORDIC unit circle

Input values are presented in the offset binary code as well. Algorithm works for the relative coordinates from the unit circle, with 1 equal to the 16384, or 2^{14}. Because of this, joystick readings are scaled appropriately. Initial area of 512^2 is magnified with a bit shift, as shown in the Figure 25. This basic operation greatly improves angle approximation, and does not slow down the algorithm.

```
#define ABS(x) x>0 ? x : -x //this macro returns absolute value of x
#define ROT180 32760 //180 degree rotation in offset binary
#define ROT90 16380 //180 degree rotation in offset binary

void cordic(int16_t x, int16_t y, int16_t *angle, int16_t *hypotenuse){

                int16_t theta = 0, x1, y1, reg;
                uint8_t i;
                uint8_t shift = 0;
                const int16_t *atanptr = atanAngles;  //pointer to the lookup table

                x1 = ABS(x << 4); // expanding joystick readings for better approximation
                y1 = ABS(y << 4);

                if(x1 < y1) { // if over 45 degrees, values are transposed in respect to
                the x = y line
                                reg = x1;
                                x1 = y1;
                                y1 = reg;
                                shift = 1;
                }

                for( i=0; i<=12; i++ ) { // iterative rotation of vector to the x axis
```

```
                    if(y1 < 0){
                                theta -= *atanptr++;  // theta is used to accu-
                                mulate look up table values

                                reg = x1 - (y1 >> i);
                                y1 = (x1 >> i) + y1;
                                x1 = reg;
                    }
                    else {
                                theta += *atanptr++;
                                reg = x1 + (y1 >> i);
                                y1 = -(x1 >> i) + y1;
                                x1 = reg;
                    }
          }

          if(shift) theta = ROT90 - theta; //fix for angles that should be greater
          than 45

          if(x < 0){  //second and third quadrant check
                    if(y < 0) theta -= ROT180;     //Y check
                    else theta = ROT180 - theta;
          }
          else{ //fourth quadrant
                    if(y < 0) theta = -theta;
          }

          x1 = x1 >> 4; // constrict the value of vector back

          *angle = theta / 182; // conversion from offset binary notation
          *hypotenuse = x1 * 0.60725293; // get original scale of hypotenuse
}
```

By definition of CORDIC, original MSP430 algorithm was improved using ATmega328P hardware multiplier. After last iteration, rotated input vector is scaled back to the original value by bit shift and multiplication with constant, with hypotenuse being equal to the result x coordinate. To get rotation in degrees, inverse offset binary angle function is used: $B^{-1}(\alpha) = {}^{\alpha}/_{182}$. Output values are written to the addresses given by input pointers.

4.3.5 Driver instruction

From actuator profile, expressed in the Figure 7 at the Section 3.1, we know that in the range from 0.2 Hz to 0.9 Hz travelling velocity is proportional to frequency. These limits are joined and fit together with dynamic range of the joystick incline. Thus, speed of the actuator locomotion can be finely controlled with slight tilt of the joystick. However, it was observed that most controller users usually push the joystick tip to the perimeter, reaching the maximum incline value.

To make motion speed control even more flexible, three limits on the top speed during maximum incline are introduced on Figure 26. The frequency mode is 0.7 Hz by default,

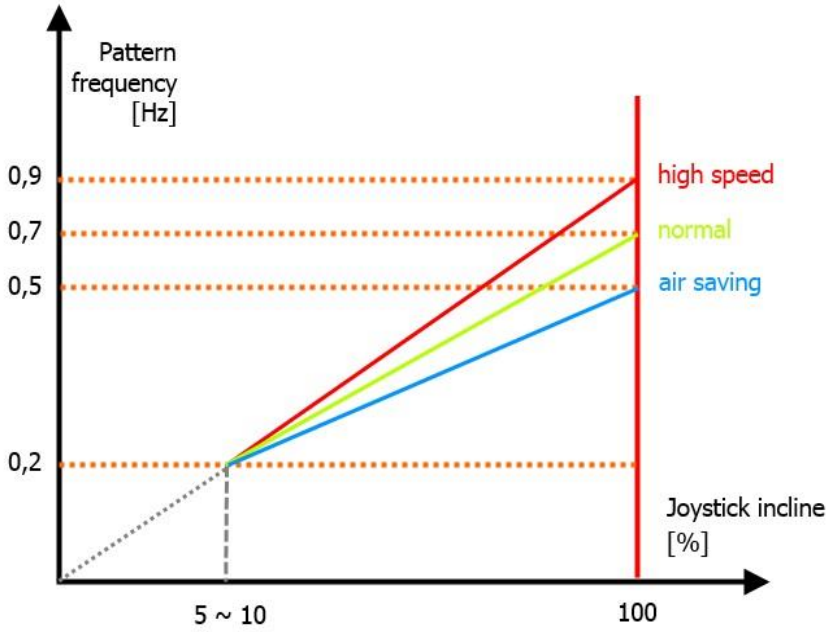


Figure 26. Actuator driving frequency modes

and it can be shifted by pressing the joystick bar. Depending on the given frequency mode *factor*, function *freq_change* scales joystick input *tilt* of range [0 ; 512] units to one of following ranges: [0; 329] for 0.9 Hz, [0 ; 302] for 0.7 Hz, and [0; 253] for 0.5 Hz.

```
int16_t freq_change(uint8_t factor, int16_t tilt){

            switch (factor) {
                        case DOTNINE:{
                                    tilt *= 0.642578125;
                                    tilt = tilt<18 ? 0 : (tilt - 17);
                                    break;
                        }
                        case DOTSEVEN:{
                                    tilt *= 0.58984375;
                                    tilt = tilt<16 ? 0 : (tilt - 15);
                                    break;
                        }
                        case DOTFIVE:{
                                    tilt *= 0.494140625;
                                    tilt = tilt<13 ? 0 : (tilt - 12);
                                    break;
                        }
                        default: break;
                        }
            return tilt;
}
```

To leave blind spot of no reaction in the middle of joystick (marked by grey dotted line), output values less than 10% of maximum are neutralized to zero. Values, greater than blind spot threshold, have 10% subtracted. This ensures that all frequency instructions to the driver are starting at the same logical level, as marked on the Figure 26.

After joystick hypotenuse is scaled, all instruction values are given to the transmission buffer. Because radio module receives information by SPI one byte at a time, buffer is defined as 8-bit variable array. The radio message is composed from 5 bytes. 16-bit values of the direction angle and joystick incline are put in series; 8-bit signal union is added at the end (Figure 27).

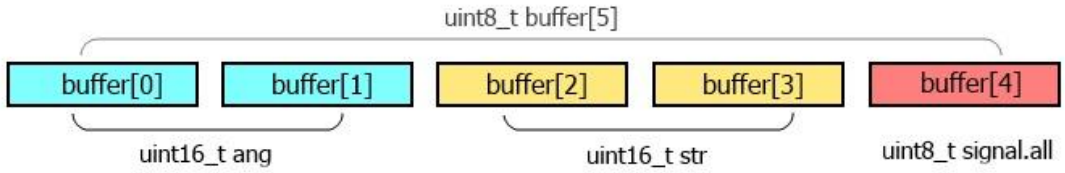


Figure 27. Radio message buffer

```
buffer[0] = ang >> 8;
buffer[1] = ang;
buffer[2] = str >> 8;
buffer[3] = str;
buffer[4] = signal.all;
```

The *str* and *ang* variables are truncated with bit shift, and assigned to the buffer just before *nrf24l01_write* function call.

4.4 Robot driver

4.4.1 Architecture

Actuator driving system is designed to transform the human interface input into pressurizing pattern. Architecture of the robot driver is presented on the Figure 28. The information is received from the wireless module with SPI peripheral. During program operation, timer module is continuously updated with pattern period value, which is calculated from the received data. Timing of the actuators is performed in two separate compare registers, providing independent intervals for each sink driver.

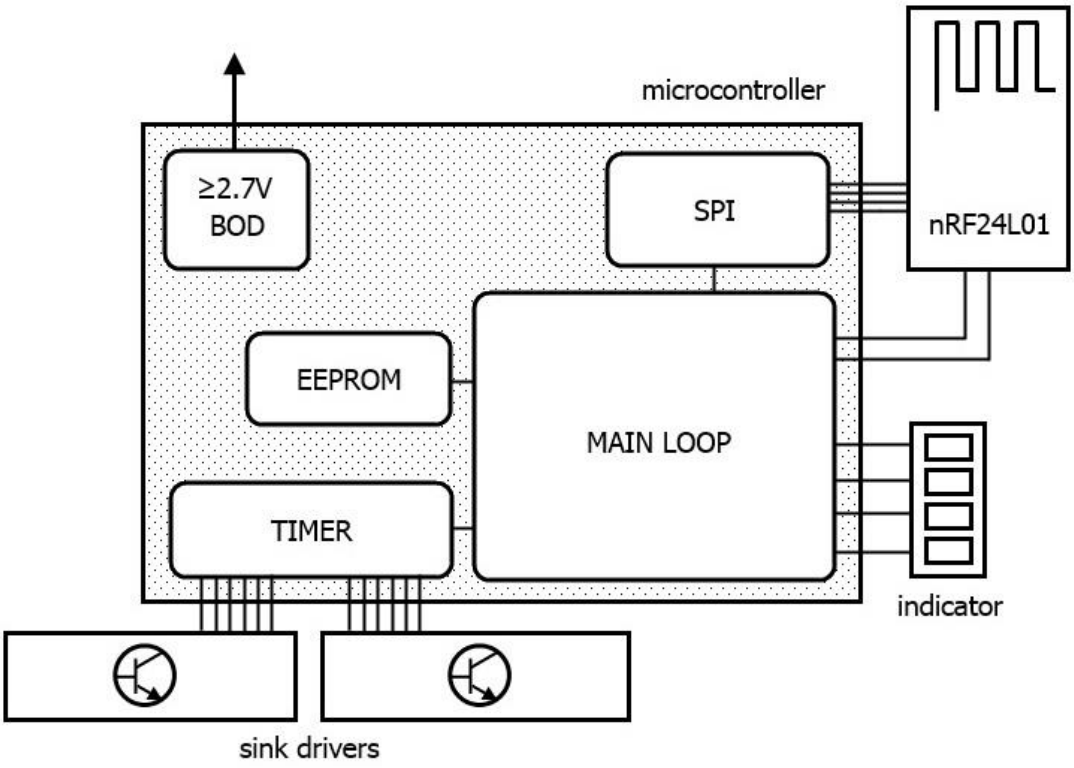


Figure 28. Robot driver architecture

During robot movement, exhausted air is counted, and remaining air value is displayed at the indicator. From time to time, air gauge information is saved to the internal EEPROM memory. When system boots up again, this information is recalled and updated.

Similarly to the remote control, brownout detector is set to prevent low voltage operation. Robot has switching digital power supply of 3.3 V; any input voltage other than that is considered to be abnormal, meaning the switching regulator does not operate properly. BOD level is placed to the closest setting of 2.7 V.

4.4.2 Main loop

Robot driver program program is a state machine implemented with while-loop. Instructions from the remote control are monitored with every loop rotation, and appropriate actions are carried out. Compared to the 2.315 ms while-loop execution period of the remote control, robot driver program rotates in 535 µs. It means that instructions are executed faster than they are incoming, and control lag is really small.

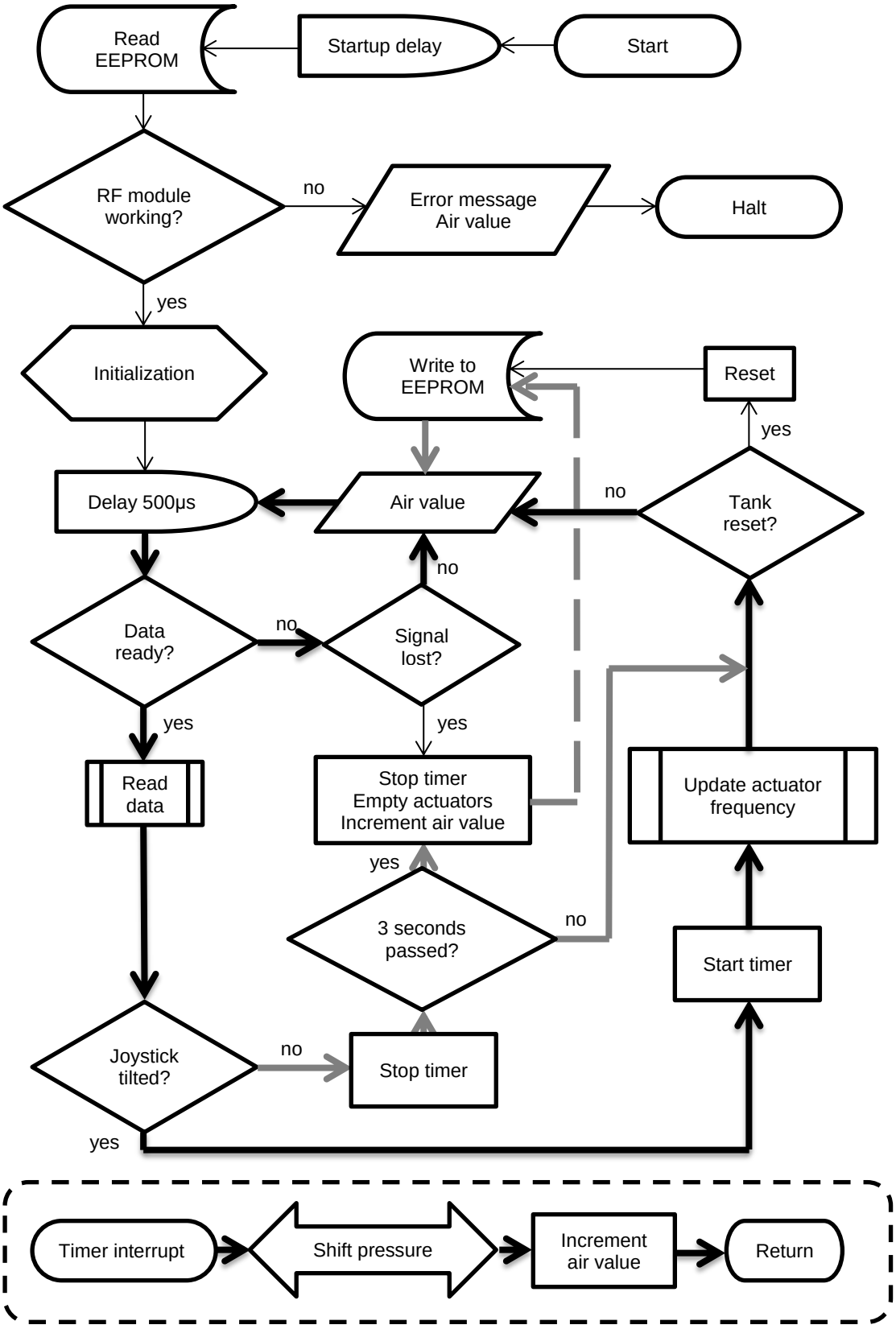


Figure 29. Robot driver program flowchart

Such kind of performance is achieved by moving most of the calculations to the remote microcontroller. Flowchart of the program is presented on the Figure 29. Main loop sequence during robot locomotion is marked with thick black lines; operation during standby is indicated with thick grey lines. Thin lines are expressing rare program operations.

System start-up is followed by a long delay of 150 ms. Like it was described in the Chapter 4.3.2, this is necessary for the RF module to boot first, and for system voltage level to settle. Because robot power supply is heavily filtered from the valve overshoots, it takes a while until big capacitors will charge (more information in Chapter 5.3.2).

EEPROM memory is accessed to recall previously saved tank depletion value (Chapter 4.4.3). Function *module_check*, explained in the Chapter 4.2.2, verifies nRF24L01 module response, and gives an error message in case of status register mismatch. Error message is displayed at the indicator bar for 3 seconds, and then CPU is trapped in the infinite loop, continuously displaying the air value. If radio module passes the

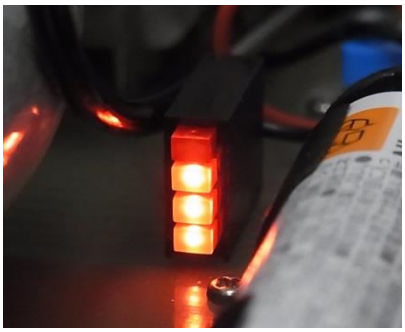


Figure 30. Air indicator

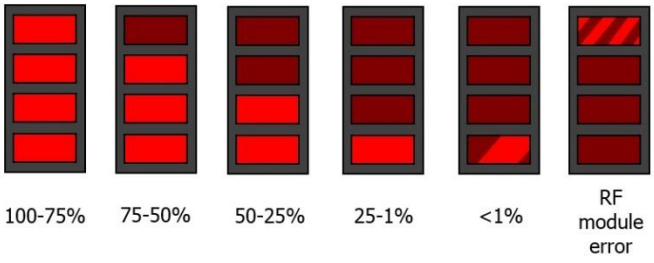


Figure 31. Air indicator states

test, it is prepared for communication, and program advances into the main loop. Function *show_air* is called at the end of any loop sequence to show the current condition of the air tank on the indicator (Figure 30). Air gauge is able to display the remaining air with precision of 25% (Figure 31). When the air counter reaches the limit, the last LED is blinking with a period of one second. In error message, top LED blinks 3 times a second.

```
void show_air(uint16_t value)
```

Air counter, used with this function, is an unsigned 16-bit variable, which is used to store the amount of chamber exhaustions. It is initialized as zero, and then overwritten from EEPROM register.

```
uint16_t air = 0;
```

Every time there is a command to empty a number of chambers, this counter is incremented by the same number. There are always 12 chambers pressurized at the same time, so *air* is increased by 12, when actuator is fully exhausted by the joystick or connection timeout. After each pressurizing pattern shift, during the timer interrupt, air counter is increased by two (see Chapter 4.4.5). Maximum capacity of the gas tank is experimentally found to be 3000 chambers. This value is split to the ranges of 750 and compared with air counter in *show_air* command.

Before reading the transmission status, there is a 500 µs delay. The radio frequency module could be read more often, but then the process of SPI communication would be interfered by the timer interrupt more frequently, which is not wanted. Arrival of the new radio packet is confirmed with *nrf24l01_readready* command:

```
_delay_us(500);

if(nrf24l01_readready(0)){ //check for new data from the pipe 0

                nrf24l01_read(buffer);

                heading = buffer[0] << 8 | buffer[1];
                incline = buffer[2] << 8 | buffer[3];
                signal.all = buffer[4];

                disconnect = 0; //reset the disconnect counter
                flag.presave = 1;
                }
```

The fresh data from the remote control is collected from the buffer, and assigned to the appropriate variables (Chapter 4.3.5 Driver instructions). Usually, new packet is coming in four or five loop durations. In order to detect that remote is disconnected, *disconnect* counter is incremented every time receiver is empty. When *disconnect* is more than 10, the joystick parameters are reset.

```
else {
                disconnect += 1;
                if (disconnect > 10){ //no answer for too long time
                                heading = 0;
                                incline = 0;
                                disconnect = 0;
                                if (flag.presave){ //check if there was disconnect backup

                                                flag.presave = 0;
                                                memory(air, 1); //save air
                                }
                }
}
```

Putting the joystick's *incline* variable to zero stops the robot locomotion immediately. Because it is more likely for signal to be lost when remote is switched off and work with the robot is finished, air counter backup is made. *Presave* flag is used to make sure that there is only one backup per each disconnect.

When joystick is outside of "blind spot", incoming *incline* variable becomes more than zero, and robot locomotion is enabled by starting the timer. During movement, timer registers are updated using function *timer_period,* explained in the Chapter 4.4.4. New register values are calculated from *heading* and *incline* variables. Flag *eco* is kept as long as actuator is pressurized.

```
if (incline > 0){  //operating incline is received

                exhaust = 0;
                flag.eco = 1;   //actuators are not empty
                timer_period(heading, incline);  //update timer values
                TCCR1B |= (1<<CS12);     //start timer with prescaler of 256

                if(flag.start){      // in case of first pressurizing
                                OCR1B = 5787;
                                OCR1A = 5787;
                                flag.start = 0;
                }
}
```

First shift periods are calculated at the very moment joystick leaves the "blind spot", which means the smallest incline and therefore lowest frequency. This happens because joystick bar is not tilted immediately to the desired speed value, but in about third of a second time. Longest wave pattern shift (0.83 s) would take place every time robot starts to move. To prevent this effect, *start* flag is introduced. First values of the registers are set to 5787, which corresponds to the fastest pattern shift (0.18 s), giving time

for joystick bar positioning. Described functionality provides impression of immediate robot reaction indeed.

When *incline* equals to zero, it means that signal is lost, or joystick is released. In this case, timer is halted and robot locomotion is ceased. However, there is a high possibility for the robot operation to continue, and program waits about three seconds, before actuators are emptied. This approach helps to save gas, and prevents excess jumping of the robot during momentary joystick release.

```
else{
                TCCR1B = 0;    //halt timer
                if (flag.eco){exhaust += 1;} //if chambers are full, count loops for about
                3 seconds before exhaust

                if (exhaust > 5000){

                                TCNT1 = 0;    //timer reset
                                flag.start = 1; //next pressurizing should happen fast
                                flag.eco = 0; //actuators are empty
                                exhaust = 0;

                                PORTC &= ~0x3F; //all valves are shut
                                PORTB &= ~0xC0;
                                PORTD &= ~0xF0;
                                air += 12;

                                rpos = 1; lpos = 4; //reset the pattern to the most stable one
                                memory(air, 1); //backup the air counter
                                }
}
```

It can be noted, that variables *lpos* and *rpos,* standing for the left and right actuator pattern position respectively, are set to the various values. This is done before every travelling session in attempt to increase balance of the robot frame. Figure 32 compares two different cases of the chamber pressurizing:

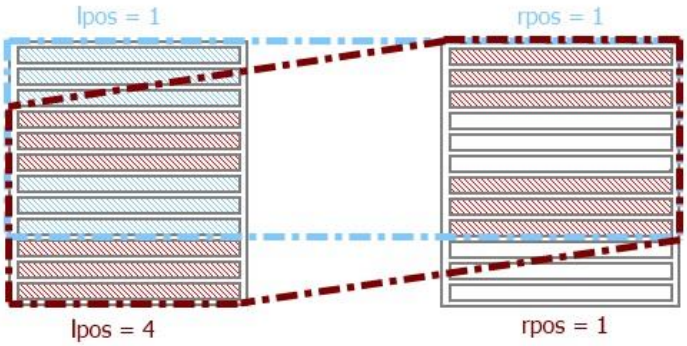


Figure 32. Actuator area comparison

It is possible to improve locomotion balance of the Bubbler controller board, observed in the Chapter 3.3.2. Compared to the previous initial pattern (marked blue on the picture), total actuator area is maximized, when chambers are swollen in the reciprocal manner (red dashed line). Obviously, during actual motion with individual actuator speeds, this situation is rare. Setting complementary initial pattern improves occurrence of this condition, and removes unwanted frame swinging in the beginning of locomotion. More about robot balance can be found in the Section 6.1 of Mechanical considerations.

At the end of the loop, air counter reset signal is monitored:

```
if (air > 60000) {air = 5000;} //preventing 16bit value from overflow

if ((signal.reset == 11)&&!(flag.reset)){ //reset signal just happened

              flag.reset = 1; //reset has been executed
              air = 0;
              memory(0, 1); //write zero to the EEPROM
              }

if (!signal.reset){flag.reset = 0;} //reset button has been released
```

When tank reset button at the remote is pressed for more than 3 seconds, reset signal is detected, and all information about tank depletion is nullified. *Reset* flag is enabled to prevent redundant overwriting of the EEPROM.

4.4.3 EEPROM memory

ATmega328P has 1 Kbyte of EEPROM memory, which persists during power off and has an endurance of at least 100,000 write/erase cycles. To access the EEPROM registers, two specific functions were taken directly from the ATmega datasheet [5]:

```
uint8_t EEPROM_read(uint16_t uiAddress)
void EEPROM_write(uint16_t uiAddress, uint8_t ucData)
```

EEPROM_read function is used to read one byte from the 16-bit memory address in the range from 0 to 1023. Function *EEPROM_write* erases the register, and writes the byte into it. In contrast to reading function, which happens immediately, EEPROM

overwrite routine lasts 3.3 ms [5]. Interrupts should be disabled during memory programming.

In the robot driver program, EEPROM access is done as rarely as possible and preferably during standby, to avoid interference with timer interrupts and prevent excess erasing of memory registers. All routines are gathered into one universal *memory* function. It is called to write and to read 16-bit air counter data by truncating and assembling value with bit shift. Argument *io* is used to select between read and write operation.

```
uint16_t memory(uint16_t bytes, uint8_t io){

              uint8_t pack[2];

              pack[0] = bytes >> 8;
              pack[1] = bytes;

              cli();  // disable global interrupts

              if (io){ // 1 write, 0 read
                            EEPROM_write(500, pack[0]);
                            EEPROM_write(501, pack[1]);
                            while(EECR & (1<<EEPE)); //wait until write is finished
                            }
              else{
                            pack[0] = EEPROM_read(500);
                            pack[1] = EEPROM_read(501);
                            bytes = pack[0] << 8 | pack[1];
                            }

              sei(); // enable global interrupts

              return bytes;
}
```

Output of this function can be ignored when using write feature, as well as input does not matter when read feature is selected. Because EEPROM module operates independently of the processor, write sequence is finished by polling the EEPE flag to hold back CPU operation until memory is programmed.

4.4.4 Steering system

The main advantage of independent actuator driving is the ability to move the robot frame in many directions. Slight turning motion can be performed by slowing down the inner actuator while keeping outer actuator frequency intact. To make this happen,

program functionality, with output similar to the car differential, was implemented. Figure 33 demonstrates various modes used to drive actuator waves. Notice that it uses transposed angle convention, explained in the Chapter 4.3.3. The area of the joystick circle is divided into 8 zones of different sizes. Forward, backward and rotary motions are performed by switching the actuator motion direction while varying speed symmetrically, according to the joystick bar incline.

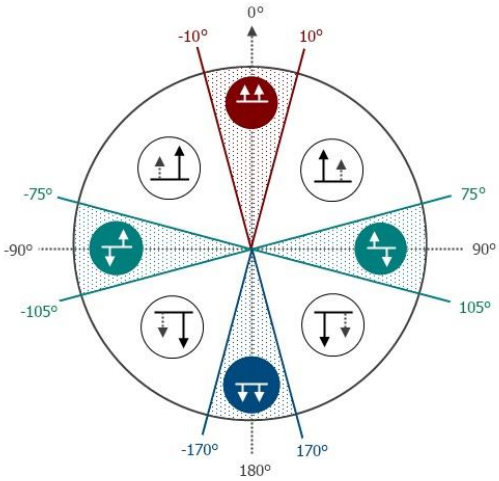


Figure 33. Joystick steering modes

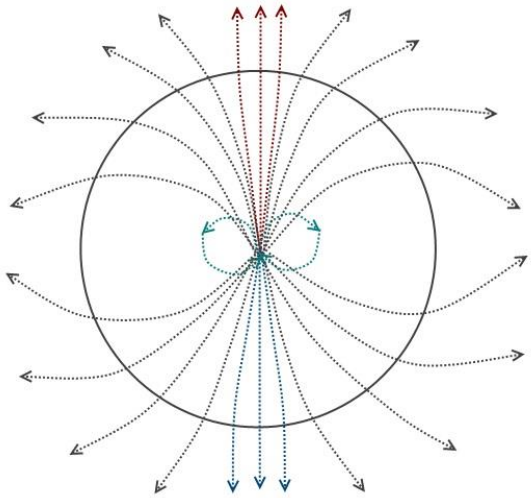


Figure 34. Robot movement patterns

Large white areas correspond to the differential actuator motion resulting in an arc trajectory. When joystick is inclined into one of these zones, actuator marked with black arrow is driven proportionally to the incline, while another one's speed is additionally scaled smaller with a greater angle from the vertical axis. This results in the directional patterns shown at the Figure 34. Radius of turning movement decreases when joystick is inclined farther to the side, eventually it becomes zero, and robot rotates around its own centre.

Asynchronous motion is reliant on the pressurizing pattern period, calculated individually for each actuator. Function *timer_period* is used to calculate both periods from polar coordinates of the joystick tip. Operating direction is narrowed down by comparing input angle with ranges displayed at the Figure 33. Then global flags *rdir* and *ldir* are written, to specify right and left actuator moving directions. Actuator interval is found by multiplication of joystick's *incline* value and a scaling factor.

When received from the remote controller, *incline* data is already fit into one of speed modes, indicated on a Figure 26 (Chapter 4.3.5). Dynamic ranges of *freq_change* function were selected in such way, that *timer_period* output will be limited by the chamber periods of 0.183, 0.238 or 0.333 seconds. Total frequency of 6-chamber wave will be:

$$f = \frac{1}{6 \cdot \tau_{chamber}}$$

The scaling factor has a maximum value of 65. When joystick is in one of white areas shown at Figure 33, scaling factor is equal to the angle from the closest rotary motion zone. For example, at angle of 25 degrees scaling factor is equal 50, and for -135 degrees it equals 30. At zones of synchronous actuator motion, scaling is not necessary, and factor is fixed at 65. Product of *incline* and scaling factor is subtracted from the number 26067, which stands for the maximum chamber period (Figure 35).

Figure 35. Timer period calculation

The driving period calculation can be generalized as: “For lower speeds, joystick incline value is further reduced, resulting in longer periods and lower driving frequency”.

```
void timer_period(int16_t ang, uint16_t str){

            if ((ang > -75) && (ang < 75)){  // forward motion
                        flag.rdir = 1; // both actuators are set forward
                        flag.ldir = 1;
                        if ((ang >= -10) && (ang <= 10)){ //synchronous forward
                                    temp.right = 65 * str;
                                    temp.left = temp.right; //periods are equal
                        }
                        else{
                                    if (ang > 0){ //northeast differential
                                                temp.angle = 75 - ang;
                                                temp.right = temp.angle * str;
                                                temp.left = 65 * str;
                                    }
                                    else{ //northwest differential
                                                temp.angle = 75 + ang;
```

```
                                        temp.left = temp.angle * str;
                                        temp.right = 65 * str;
                                }
                        }
                }
                if ((ang > 105) || (ang < -105)){ // back motion
                        flag.rdir = 0; // both actuators are set backward
                        flag.ldir = 0;
                        if ((ang >= 170) || (ang <= -170)){  //synchronous backward
                                temp.right = 65 * str;
                                temp.left = temp.right; //periods are equal
                        }
                        else{
                                if (ang > 0){ //southeast differential
                                        temp.angle = ang - 105;
                                        temp.right = temp.angle * str;
                                        temp.left = 65 * str;
                                }
                                else{ //southwest differential
                                        temp.angle = ang + 105;
                                        temp.angle = -temp.angle;
                                        temp.left = temp.angle * str;
                                        temp.right = 65 * str;
                                }
                        }
                }
                if ((ang >= 75) && (ang <= 105)){  // rotation clockwise
                        flag.rdir = 0; //actuators  are driven in reverse
                        flag.ldir = 1;
                        temp.right = 65 * str;
                        temp.left = temp.right; //periods are equal
                }
                if ((ang >= -105) && (ang <= -75)){  // rotation counterclockwise
                        flag.rdir = 1; //actuators  are driven in reverse
                        flag.ldir = 0;
                        temp.right = 65 * str;
                        temp.left = temp.right; //periods are equal
                }

                clean.right = TMAX - temp.right; //subtract from maximum interval
                clean.left = TMAX - temp.left;
}
```

Structure, named *clean*, is used to gather final period results. To be safely accessed from the timer interrupt, these global variables are updated once, in the end of calculation. This prevents left and right period mismatch in case interrupt happens in the middle of the calculation.

4.4.5 Actuator timing

Timer peripheral of ATmega328P is utilized to clock the actuator periods. Like in MSP430 and many other microcontrollers, timer can be configured to generate independent time intervals. ATmega's 16-bit Timer1 module is clocked from the main RC oscillator. Frequency of the timer ticks can be modified with clock prescaler. In this application, prescaler is set to divide the processor clock by factor of 256, resulting in the 32 µs tick at 8 MHz CPU.

Two capture registers, OCR1A and OCR1B are used to generate pattern periods for left and right actuator respectively. During normal operation, Timer1 module compares its counter value with both capture registers every tick. Corresponding interrupt request is generated at register value match. When interrupt happens, current CPU operation is postponed, and instruction situated at interrupt vector address is executed. During this subroutine, pressurizing bit pattern is shifted to one position, in the direction defined by actuator *dir* flag. Each interrupt, position counter *pos* is incremented by one.

```
ISR(TIMER1_COMPA_vect){ // left actuator Interrupt Service Routine

            if (flag.ldir) {lpos -= 1;} // wave direction is opposite to motion
            else {lpos += 1;}
            if (lpos > 6) {lpos = 1;} // position is changed periodically
            if (lpos < 1) {lpos = 6;}
            switch (lpos){

                        case 1:{ PORTC &= ~0x38;     //00 000111
                                    PORTC |= 0x07;
                                    break;}
                        case 2:{ PORTC &= ~0x31;     //00 001110
                                    PORTC |= 0x0E;
                                    break;}
                        case 3:{ PORTC &= ~0x23;    //00 011100
                                    PORTC |= 0x1C;
                                    break;}
                        case 4:{ PORTC &= ~0x07;    //00 111000
                                    PORTC |= 0x38;
                                    break;}
                        case 5:{ PORTC &= ~0x0E;     //00 110001
                                    PORTC |= 0x31;
                                    break;}
                        case 6:{ PORTC &= ~0x1C;    //00 100011
                                    PORTC |= 0x23;
                                    break;}
                        default: break;
            }
            air += 2;
```

```
            OCR1A += clean.left; // pick up next time period
}
```

It is important to drain supply current as little as possible. When bit pattern is changed, unneeded valves are shut first, and new ones are energised after. In one interrupt routine two chambers are pressurized, two chambers are exhausted, and air counter value is incremented by 2.

New period value is added to the OCR register, so that next interrupt will be requested after fresh amount of ticks. The timing operation is illustrated at the Figure 36:

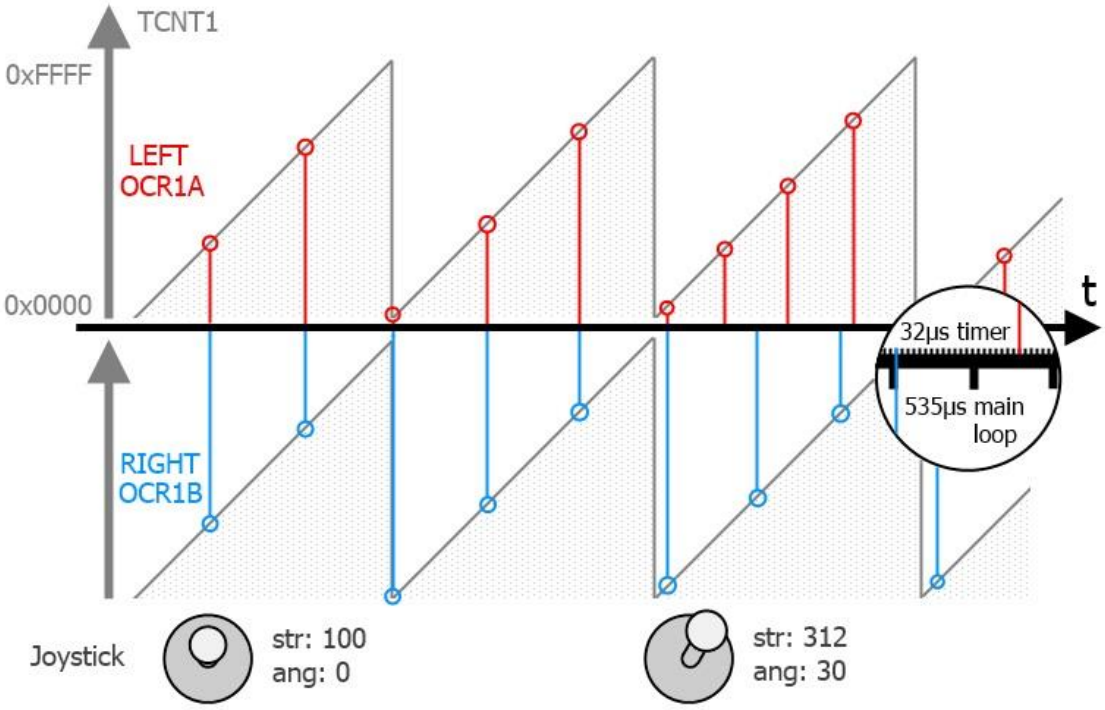


Figure 36. Actuator timing operation

When sum of OCR register value and new period is bigger than 65535 (2^{16}), register overflows, keeping the overhead. Timer counter TCNT1 register also overflows when the maximum 16-bit value is reached. Combined with the remote control sampling rate, the *timer_period* function provides new values at least 370 times a second.

Example of the timing from the joystick readings is demonstrated in the picture: when joystick is slightly tilted forward at angle of zero degrees, actuators are driven synchronously; if joystick is inclined to the right, left actuator is going at full speed and right actuator is slowed down, resulting in the turning motion.

5 Electrical design

5.1 KiCAD

PCB layouts of remote control and robot frame were designed using KiCAD electronic design automation software. KiCAD is an open source software suite, which includes schematic editor, printed circuit board editor, and large component footprint database. It can be used to create complex designs; however, it was selected by me because of its simplicity and availability. Like most design automation tools, KiCAD is user friendly

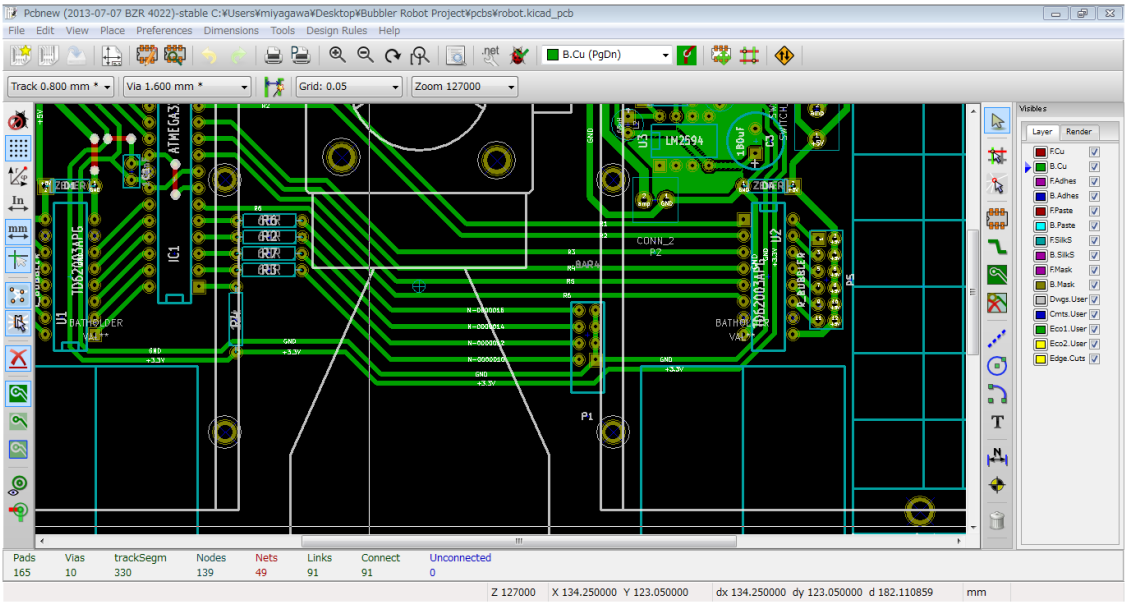


Figure 37. KiCAD Pcbnew PCB Layout editor

and easy to master. Most helpful features are as follows:

- Flexible footprint system, where any schematic component can be associated to any footprint
- Precise plotting algorithm allows easily print fine 1:1 scale copper layouts for etching
- Drill map generation (see chapter 7.1)

5.2 Remote control

Power source for the remote control is selected to be a pair of AAA batteries. Judging by the list of the components presented in the Table 2, supply voltage range is wide, and allows for 3 V operation. It is also safe to use alkaline batteries, because their characteristic depleting of voltage with time is acceptable in this case.

Table 2. Remote control voltage demand

Element	Operating voltage, V
ATmega328P	1.8 – 5.5
nRF24L01+	1.9 – 3.6
LED	≥2.2
Joystick	Any

ATmega supply current is drawn in very short spikes on the clock edges [6]. In order to compensate high frequency current spikes, decoupling capacitor of value 100nF is placed as close as possible to the supply pins of the microcontroller. NRF24L01+ module has decoupling capacitors mounted on-board, so the transceiver digital circuitry is safe as well. To provide additional pool of immediate charge for the circuit, one polarized 2.2 µF capacitor is connected in parallel to the supply pins. Analogue reference pin Aref is connected to the ground through 100 nF capacitor, and traced separately from microcontroller digital ground, according to the guidelines given in the datasheet [5]. AVcc pin, used as voltage source for ADC, is connected to the Vcc pin directly without low pass filter. During conversion, ADC module relies to decoupling capacitor on the digital supply, while CPU is in sleep mode (Chapter 4.3.3). Described approach does not require additional components, and joystick readings are kept precise enough.

Joystick potentiometers are connected to the ground and supply lines in such manner, that ADC readings will be increasing from right to left and from bottom to top, resembling Cartesian coordinate plane (Figure 38). Buttons are read active low by the microcontroller. This functionality is implemented with ATmega328P internal pull-up resistors, reducing the need of mounted components. When button is released, pin is connected to the supply through the high-value resistor in series, resulting in the logical high on the input. When button is pressed, microcontroller pin is shorted to the ground, putting active low in the PIN register, and excess

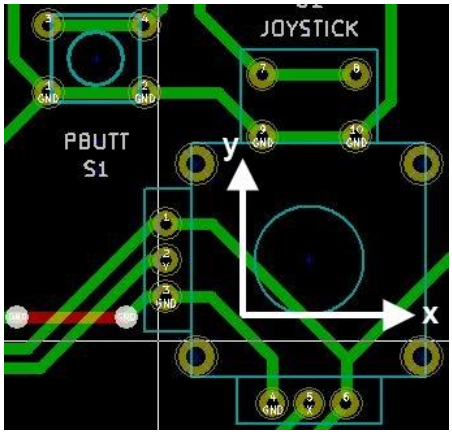


Figure 38. Joystick and buttons

voltage is left across the pull-up resistor. The 30 kΩ pull-up resistor is connected to reset pin of the ATmega, providing constant logical high state.

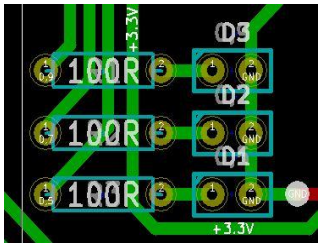


Figure 39. LED indicator

Three light-emitting diodes are forming the indicator. Efficient 15 mA diodes were selected for this purpose. Current limiting resistors of 100 Ω are placed in series (Figure 39).

Main SPI signal lines do not intersect and are kept as short as possible. For this purpose, radio module was placed as close as possible to the ATmega SPI I/O. Interrupt request (IRQ) and receive-transmit chip enable (RXTXCE) are not switching on high frequency during operation, thereby they are low-priority and connected above the copper layer (Figure 40). NRF24L01 connector is placed as close to the border as possible. Mounted module is reaching out the PCB area to avoid signal attenuation and reflections off copper layer.

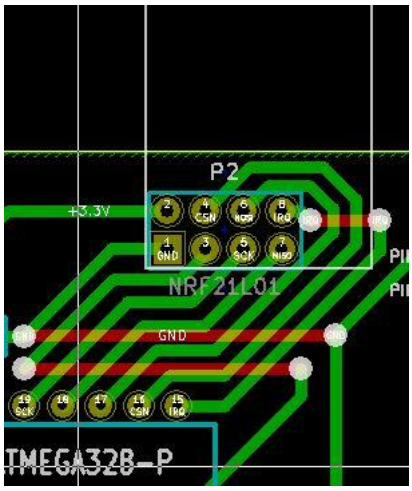


Figure 40. nRF24L01 connection

5.3 Robot

5.3.1 Solenoid valve drive

Robot actuator control is based on CKD 3MB0 pneumatic solenoid valve. To pressurize actuator chambers in desired pattern, two arrays of 6 valves each are used. During robot movement, 6 solenoids are energized at the same time. Driver firmware ensures that not more than 6 valves are active at the same time, preventing excess current draw (Chapter 4.4.5). Each 3MB0 valve is powered with 5 V DC, using energy of 1 W. By the electric specifications provided in the catalogue [12], applicable voltage can vary ±10%, defining wide operating range of 5.5 – 4.5 V. Maximum electric current, drawn by the solenoid system is found as:

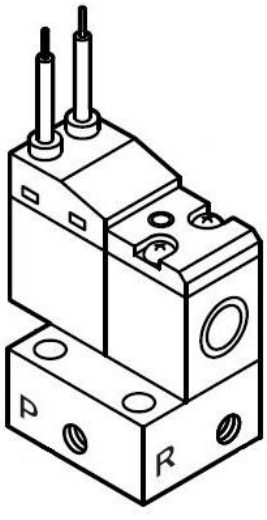


Figure 41. CKD 3MB0 pneumatic valve

$$\mathrm{I}_{max} = 6 * \frac{1\,W}{5\,V} = 1.333\,A$$

To provide portable power source for such load, research on the battery elements was conducted.

Voltage delivered by battery elements is reduced with discharge. Because of that, duration and range of useful supplied voltage should be considered individually for each application. There is clear tendency of batteries to deliver less charge if load current exceeds Ampere-hour rating of one hour. It means, that useful capacity of the battery decreases, if charge depletes too fast. Internal resistance of the battery cell also should be recognized, as large loads cause bigger internal voltage drop.

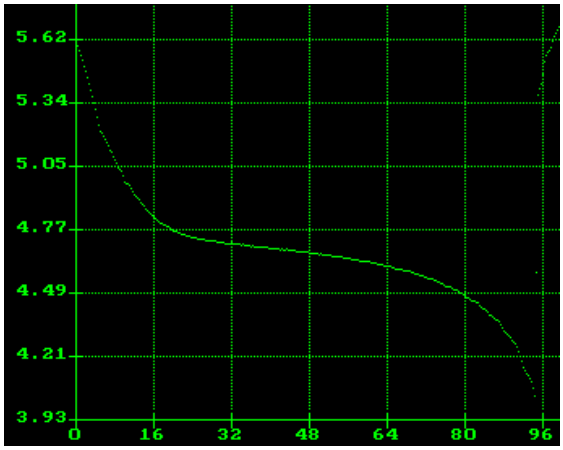


Figure 42. Sanyo's Eneloop discharge test. Horisontal axis – time in minutes, vertical axis – voltage in volts

Despite tiny internal resistance, alkaline batteries are the worst choice for this application, showing really steep voltage decrease if current demand is close to the capacity rating. Batteries with larger capacity are heavier and require more space. Nickel metal hydride (NiMH) batteries have demonstrated good longevity and insignificant voltage drop, even when supplying large currents. Figure 42, borrowed from stefanv.com [13], presents 1.2 A discharge curve of 4 NiMH Eneloop AA batteries by Sanyo. As we can see from the graph, 4-battery pack is theoretically capable of driving the 6 valves for 80 minutes, until lowest applicable voltage of 4.5 V is reached. Region of slightly excess voltage during first two minutes is neglected. Total internal resistance of four Eneloop batteries is estimated 0.384 Ω [13]. Eneloop, being the good solution for a portable supply, was purchased and utilized in the system.

Due to the limited capability of ATmega ports, in order to operate pneumatic valves, sink drivers of the same model as in Bubbler control board were used. Structure of the Toshiba TD62003 driver is presented at Figure 43. Driver is based on a Darlington pair, and has clamp diodes between each output and Vcc for switching inductive loads [14].

There is a certain negative effect related to the operation of solenoid. Current surge happens, when large inductive load of valve is de-energized with a switch, cutting the high DC current path. To prevent overvoltage across the switch and provide way for the current, clamp diodes are present in the system. However, returning current is bounced back at the Vcc node, momentarily increasing the supply voltage (Figure 44).

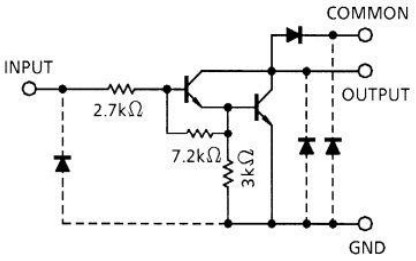


Figure 43. Sink driver structure

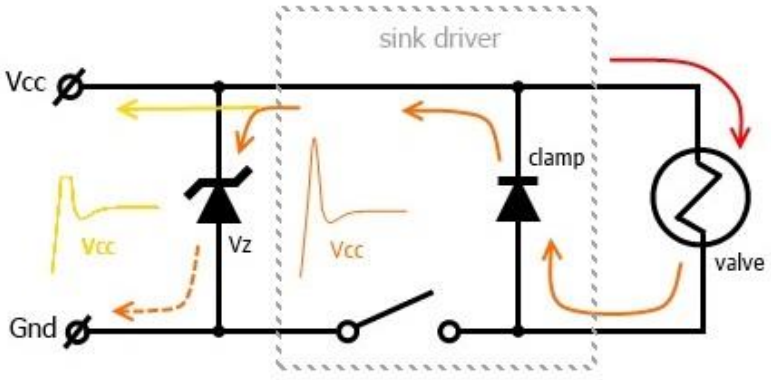


Figure 44. Surge suppressor operation

Zener diodes are utilized as surge suppressor, by definition conducting reverse current starting from Zener voltage V_z. Four NiMH batteries' maximum possible voltage is assumed to be less than 6 V at full charge. Surge suppressor diode is selected in such way that threshold is not too high for nominal Eneloop voltages of 4.8 V and not too low for the fresh charge, because in that case suppressor will start to discharge batteries. In this application, two 6 V Zener diodes with power rating of 5 W were chosen. Each suppressor can withstand short current spikes up to 12.7 A.

On the PCB, Zener diodes are placed very close to the drivers, ensuring that big spike does not propagate to the other part of the system (Figure 45).

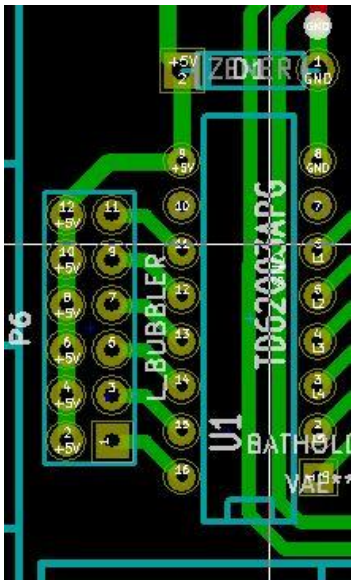


Figure 45. Solenoid driver

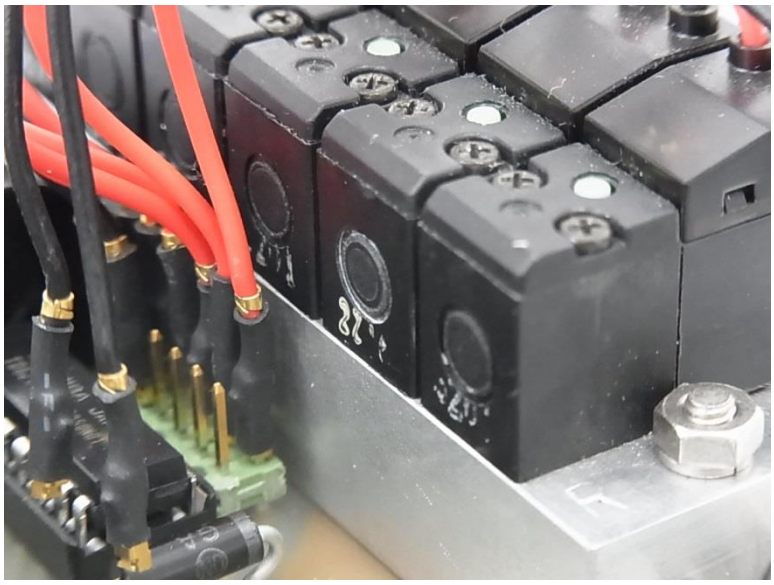


Figure 46. CKD 3MB0 valve connection

The solenoid valves are connected individually to the male DIP port, located next to the Toshiba TD62003 driver (Figure 46).

5.3.2 Digital power supply

Control circuitry voltage demand is of the same order as presented on Table 2. To set up fail-proof voltage supply in the system powered with a battery pack, TI LM2594 switching voltage regulator is used. This integrated circuit provides active functions for a buck regulator, operating on a fixed frequency of 150 kHz. Texas Instruments LM2594 datasheet [15] gives extensive guidelines on its application, thereby simplifying the electrical design procedure. Figure 47, borrowed from the datasheet, pictures the typical circuit of the step-down regulator:

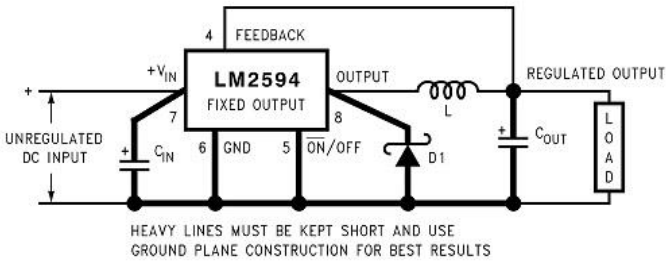


Figure 47. LM2594 circuit

Values of inductor L and capacitors C_{in}, C_{out} selected from the datasheet tables [15]. Recommended value of L is 68 nH, C_{out} and C_{in} are 180 nF and 68 nF respectively. Assuming maximum input voltage to be 5.5 V, lowest duty cycle for 3.3 V output should be 0.6. Predicted output voltage ripple is calculated with buck regulator formula:

$$\Delta Vf = \frac{\Delta i_c}{8 \cdot f \cdot C} = \frac{V_o}{8 \cdot f^2 \cdot L \cdot C}(1 - D)$$

$$\Delta Vf = \frac{3.3V}{8 \cdot 150000\ Hz^2 \cdot 0{,}000068\ H \cdot 0{,}00018\ F}(1 - 0{,}6) = 0{,}000599\ V$$

Such voltage ripple is acceptable, so selected components were included in design.

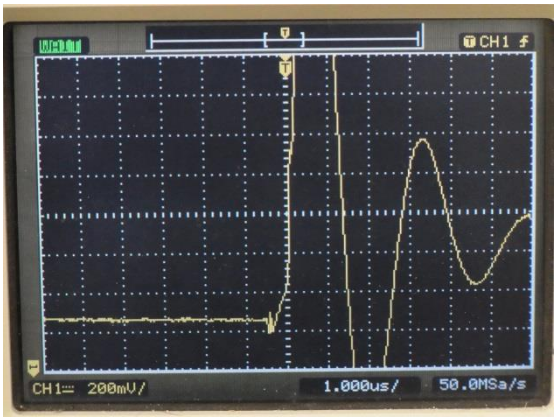


Figure 48. Solenoid voltage surge

Solenoid voltage surge, explained in the previous chapter, was analysed. Figure 48 displays oscilloscope readings from the Bubbler controller board supply. In 6 valves' shutdown transition, overshoot magnitude is bigger than 1 V, and ringing period is about 3 µs is present. Using only surge suppressor does not eliminate most of these problems, but only inhibits the spikes to exceed the 6 V level. By so-called "belt and braces" principle, digital voltage supply is additionally isolated from the outer 5 V circuit with a low-pass filter. Estimated ringing frequency is $^1/_{2{,}8\mu s} \approx 357\ kHz$. For good attenuation, filter cut-off frequency was selected to be in order of two decades below the transition effect, say, 2 kHz. Utilizing the 68 nF C_{in} capacitor of the voltage supply as the component of LC-filter, closest value of the filter inductor will be 100 nH, providing us with cut-off frequency f_c:

$$f = \frac{1}{2\pi\sqrt{LC}} = \frac{1}{2\pi\sqrt{0{,}00068\ F \cdot 0{,}0001\ H}} = 1930\ Hz$$

To prevent possible oscillations on the filter, small critical damping resistor was added in series between inductor and power supply node.

Value of damping resistor is determined as:

$$R_{crit} = 2\sqrt{L/C} = 2\sqrt{\frac{0{,}0001\ H}{0{,}000068\ F}} \approx 2.425\ \Omega$$

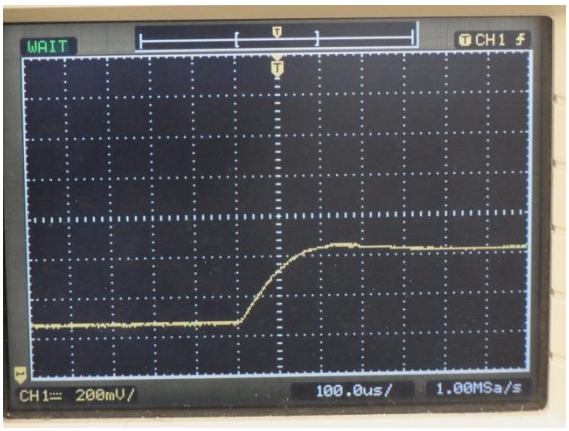


Figure 49. Solenoid surge after filtering

Filtered transition shape is presented in the Figure 49. The transient from 4.7 V to 5 V seems to be over damped, but is more acceptable than previous situation. More efficient way of removing voltage surge could be implemented; yet, given filter is good enough to protect the radio module and microcontroller.

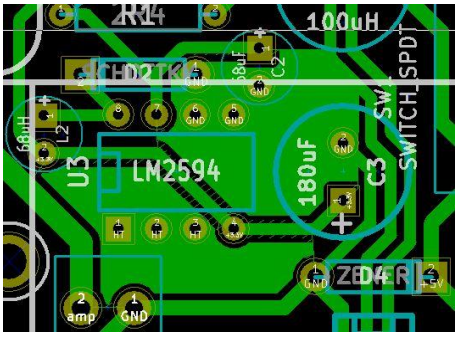


Figure 50. Heat and ground planes

Following the layout guidelines from the LM2594 datasheet [15], ground and heat sink planes were included into PCB design, as shown on the Figure 50. NC pins 1, 2 and 3 of regulator are put together to the unconnected copper polygon for heat dissipation. Ground pins 5 and 6, as well as Schottky diode and capacitors cathodes, are interconnected into one ground copper area for better conductivity.

To minimize magnetic noise caused by high current in the pneumatic valve wires, power line loop areas are minimized, as shown on the Figure 51. Two upper traces are supplying the left solenoid array, and two lowest are powering the microcontroller from the digital supply. Thickness of power traces was determined by rule of thumb: one millimeter width for current of one Ampere. Total system current is coming by 1.5 mm width track from the screw terminal to the power switch pin. Each solenoid array is powered with 1 mm tracks; every other trace is of default 0.8 mm width. Solenoid ground and digital ground are separated from each other, and connected only through the ground plane of the digital power supply.

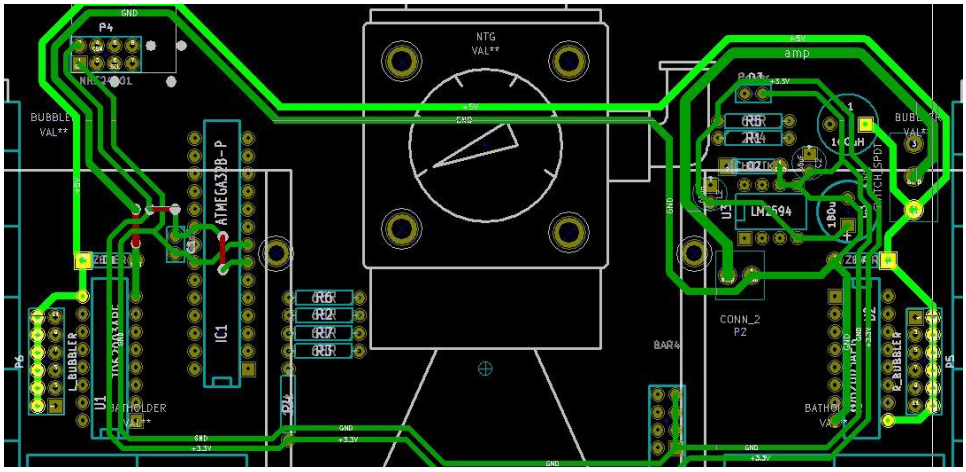


Figure 51. Power lines

Green LED, connected directly to the digital power supply through 100 Ohm serial resistor, indicates the power-on state of the device. Microcontroller, LED gauge and radio module are treated identically as in the remote controller (see Section 5.2).

6 Mechanical considerations

6.1 Robot balance

Balance of the robot frame was estimated roughly and managed by positioning the component shapes in the KiCAD.

The main problem of robot balance is the length of the gas supply, being the largest element in the system. It is also the heaviest, 551 g when empty and 569 g when full. Gas supply center of mass position was approximated experimentally, being right in between flow controller and gas tank (red dashed line in the Figure 52).

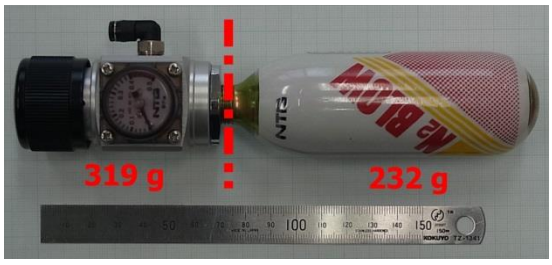


Figure 52. Gas supply mass distribution

Because pressure valves and actuators are mounted to the frame symmetrically, gas supply was attached with the center of mass line in the middle of the PCB. Four Eneloop batteries add additional 108 g at the rear. Compared to the total weight of the robot excluding actuators being 1043 g,

batteries are about 10% of the mass. Due to this small misbalance, line of equilibrium is shifted towards battery holders together with actuator midpoint, marked with a blue dashed line on the Figure 53:

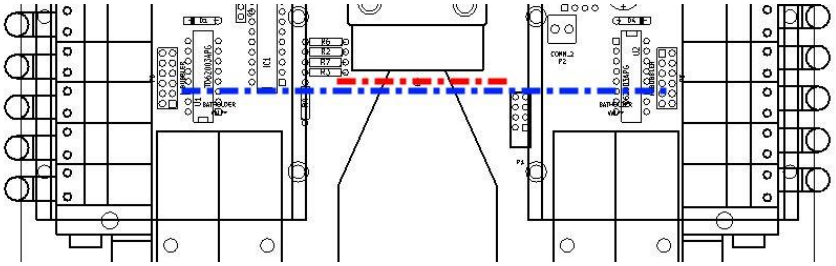


Figure 53. Robot frame equilibrium

Midpoint of gas supply and valve arrays is noted by the red dashed line.

Printed circuit board is elevated above actuators with 8 mm spacers (Figure 54). This prevents the trough-hole component legs from the short circuit, but does not improve situation with high center of mass, same of Bubbler controller board, described in the Chapter 3.3.1.

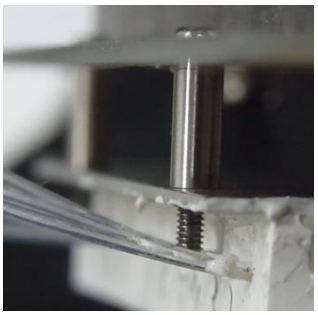


Figure 54. 8-mm spacers

Balance can be upgraded further by using only surface mounted electronic components and fixing the actuators tight to the board.

6.2 Pneumatic circuit

The crawling robot utilizes exactly the same pneumatic circuit used in the Bubbler controller board. Its schematic diagram is presented in the Figure 55. The connection is quite straightforward, actuator chambers are connected to the valves in recurring manner, simplifying periodic pattern generation.

Gas from the tank is delivered to the system through the flow control valve. Operating pressure of the actuators is in range from 0.15 – 0.3 MPa. Solenoid valves do not restrict flow, and deliver all pressure to the chambers, when energized. When switched off, valve exhausts the actuator port to atmosphere, setting chamber to zero MPa.

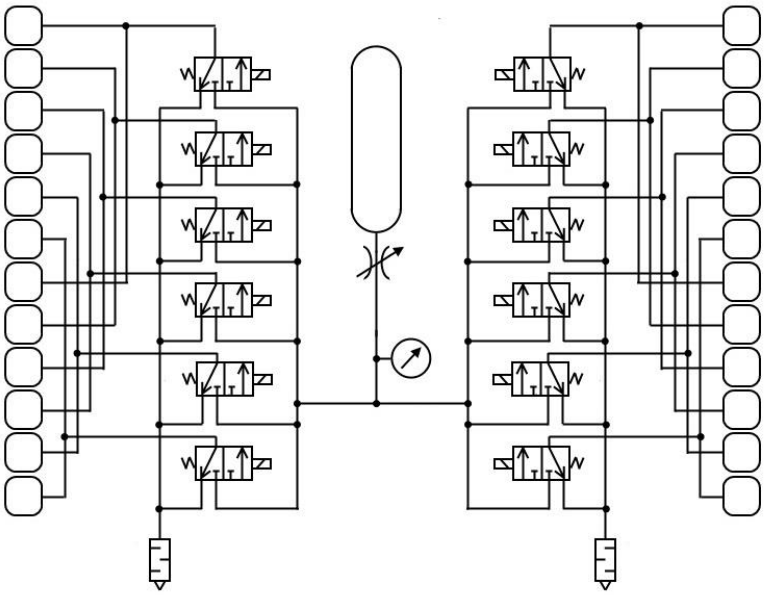


Figure 55. Pneumatic circuit

Volume of the tank chamber is 95 ml. Initially, N_2 tank has 15.2 liters of gas, resulting in absolute pressure 16 MPa, or 15.9 MPa gauge pressure (relative to the atmosphere). Gas, drained from the tank is monitored in robot firmware by counting the number of exhausted chambers.

Using chamber dimensions provided in the Section 3.1 Figure 4, volume of one chamber can be found as $\mathrm{V}_0 = 4\ \mathrm{mm} \times 4\ \mathrm{mm} \times 60\ \mathrm{mm} = 96\ mm^3 = 0{,}96\ ml$. When actuator chamber is filled with compressed air, it increases in size and expands the volume. To detect, how much gas is used for one chamber, depleting experiment was carried out. Air balloon of 24 l volume was emptied by continuous pressurizing of actuator chambers, until internal air pressure dropped from 0.725 MPa to 0.425 MPa, totaling in 0.3 MPa pressure drop. Overall amount of pressurized chambers, counted by robot firmware, was 14466. The volume of inflated chamber can be found by division:

$$\mathrm{V}_{0.3} = \frac{24\ l}{14466} = 0{,}001659\ l = 1{,}659\ ml$$

This means that inflated chamber expands for 173% of initial volume. Working capacity of the portable N_2 gas tank at 0.3 MPa in terms of inflated chambers can be calculated as:

$$\mathrm{C} = \frac{0.1\ \mathrm{MPa} \cdot 15{,}2\ l}{0.3\ \mathrm{MPa} \cdot 0{,}001659\ l} = 3054$$

This value is rounded down to 3 thousand for use with the air gauge, explained in Chapter 4.4.2. If actuators are driven with constant frequency of, for example, 0.7 Hz, the gas flow rate will be

$$Q = 0{,}7\frac{1}{s} \cdot 6 \cdot 2 \cdot 2 \cdot 0{,}001659\,\mathrm{l} \approx 27{,}9\ {}^{ml}/_{s}$$

and tank would be capable to supply the gas for

$$\mathrm{t}_{0{,}7} = \frac{0.1\ \mathrm{MPa} \cdot 15{,}2\ l}{0.3\ \mathrm{MPa} \cdot 0{,}0279\ l} = 181\ s \approx\ 3\ min$$

Gas tank life for the frequencies of 0,9 Hz and 0,5 Hz is 141 s and 254 s respectively. Obviously, lower driving frequencies drain less gas at a time.

7 Device assembly

7.1 PCB manufacturing

Printed circuit boards for the robot and remote were manufactured by chemical etching. Sunhayato one-sided photosensitive boards were available for selection, and appropriate ones were picked. Prior to the manufacture, PCB layout was optimized for the decided surface area, and pads sizes were reconfigured to resist chemical undercut.

For the remote control, smallest 1.6 x 75 x 100 mm board from phenolic paper (FR-1) was selected (Figure 56, on the left). Its small system footprint and trivial operation are implemented beneficially, using cheapest available board.

Figure 56. Photosensitive boards

Basic robot footprint is at least 170 mm wide and 100 mm long. Conventional size of 150 x 200 mm was selected, allowing wider component distribution (Figure 56, on the right). Due to the heavy components such as gas tank and solenoid valves, strong glass-reinforced epoxy laminate (FR-4) of

1.6 mm thickness was used. Its flexural strength, announced by manufacturer, ranges from 440 to 540 MPa, which is four times stronger than that of FR-1 (110 MPa).
Copper layer stencil is printed on special OHP paper (Figure 56) to transfer the layer pattern to the PCB. This was done using special light box, displayed at the Figure 57

Figure 57. Light box

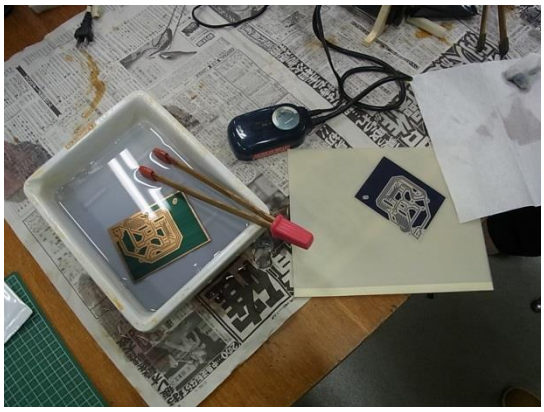


Figure 58. Pattern development

Photosensitive board is exposed in UV light for period defined with digital timer, usually 90 seconds. Once the pattern is transferred, photosensitive layer is chemically removed with Natrium Hydroxide (Figure 58). After this, board is submerged into etching tank with Ferric Chloride solution, where etchant is warmed up to 40 degrees Celsius, and agitated with bubbles to speed up the etching process (Figure 59).

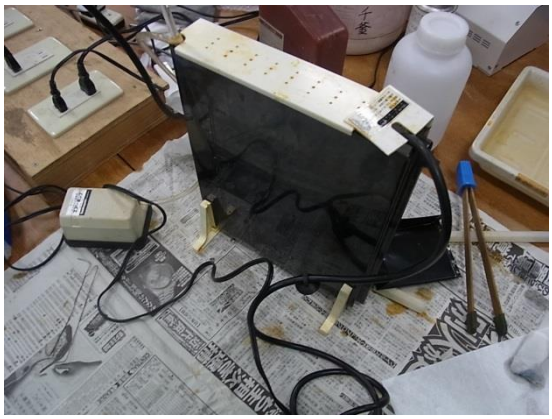


Figure 59. Etching tank

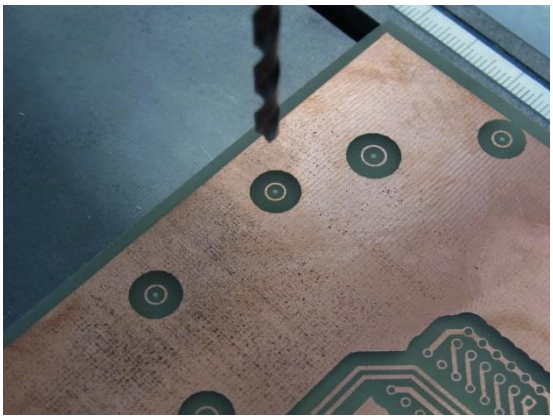


Figure 60. Drilling halos

Dark areas of the board are not affected by etchant, and are removed by full board exposure after etching is done. Surplus area of the robot PCB was cut out, leaving frame of 119 x 200 mm in size.

Drilling of the holes was done manually using desk drill press. Figure 60 illustrates copper halos and manual puncture made to assist the drill positioning.

Drill map, generated by KiCAD software, has proven itself to be helpful as a lookup table during manual drilling of hundreds of holes (Figure 61).

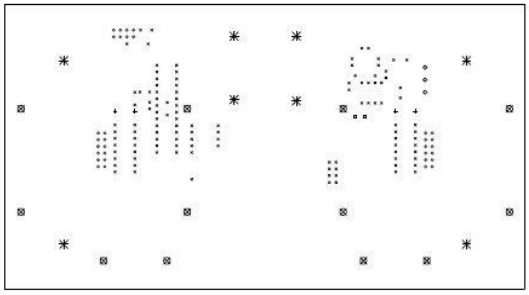


Figure 61. Robot drill map

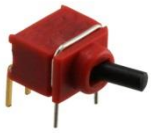
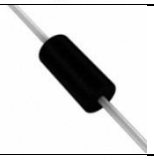
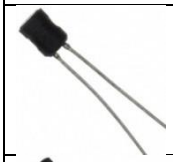
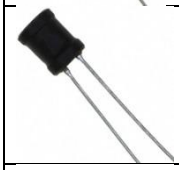
Components have been soldered to the PCB by hand. Undercut traces, whose were reduced in size during etching process, were made thicker manually with solder. In general, only trough hole components have been purposely selected for the ease of hand assembly.

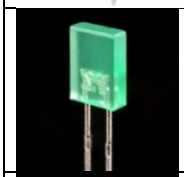
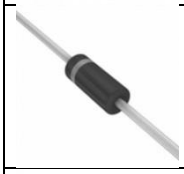
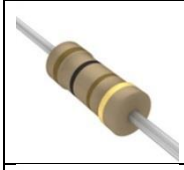
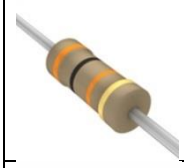
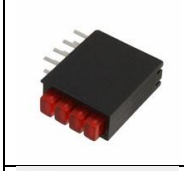
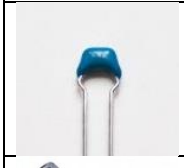
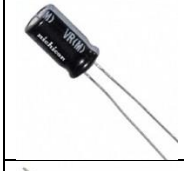
7.2 Bill of materials

Total cost of electronics components, used in the robot system is estimated and presented in the Table 3:

Table 3. Bill of materials

Picture	Qty.	Product name	Price, yen	
			per pcs.	total
	2	8-connector female 2.54 mm pitch	130	260
	2	12- connector male 2.54 mm pitch	209	418
	2	Toshiba TD62003 driver 7 ch DIP-16	107	214
	2	DIP-28 socket	81	162
	2	DIP-16 socket	44	88

	1	Holder for 2 AAA batteries	119	119
	1	Tactile switch SPDT 0.05A 12V	12	12
	1	Toggle swtich SPDT 0.4VA 20V	347	347
	1	Toggle switch SPDT 5A 120V	335	335
	1	Diode Shottky 1N5817 20V 1A	51	51
	1	TI LM2594N-3.3 regulator 0.5A 8-DIP	350	350
	1	Inductor 68uH 0.40A	34	34
	1	Inductor 100uH 1A	68	136
	2	Holder for 2 AA batteries	216	432
	1	Aluminium capacitor 68uF 10V 20%	56	56
	1	Aluminium capacitor 180UF 25V 20%	56	56

	1	DIP-8 socket	22	22
	1	LED diffused green 568nm	17	17
	3	LED diffused red 625nm	17	51
	2	Screw terminal 2 3.5mm	89	178
	1	Zener diode 1N5340B 6V 5W	63	126
	8	Resistor 100 Ohm	9	72
	2	Resistor 30 kOhm	9	18
	1	Quad LED indicator red diffused	145	145
	3	Film capacitor 0.1uF	33	99
	1	Aluminium capacitor 2.2 uF 50V	26	26
	1	Resistor 2.4 Ohm 1W 5%	39	78

	1	2 axis joystick potentiometer w/ button	561	561
	2	AAA Battery Alkaline Panasonic 1.5V	39	78
	1	AA Battery Rechargeable NiMH 1.2V Eneloop 4-pack	1925	1925
	1	Sunhayato Photo sensitive copper board FR-1 1.6t 75x100 mm	494	494
	1	Sunhayato Photo sensitive copper board FR-4 1.6t 150x200 mm	2058	2058
	2	Atmel ATmega328P 28DIP 8-bit MCU	340	680
	2	nRF24L01+ 2.4GHz module	1090	2180
			Total	11771

It should be considered that bill of materials does not include prototyping and manufacturing cost. Some of listed materials are provided by the Akita National College of Technology.

8 Results

Construction according to design has been successful, resulting in finely running actuator control system. Robot and joystick have been tested and evaluated. Operating specifications of both devices are listed in the Table 4:

Table 4. Operating specifications of the mobile robot system

	Remote control	Robot
Dimensions (W x L x H), mm	96 x 114 x 47	300 x 197 x 75
Weight, g	67	1137
Operating voltage, V	1.9 - 3.6	4.6 - 5.5
Supply current, mA	19 @ 3.0V	82 (824) @ 4.8V
Battery element	2 x AAA 1.5V	4 x AA NiMH 1.2V

Remote control is powered by alkaline batteries, but any kind of AAA battery will do, as long as voltage requirements are met. Robot width is provided including the actuator tubing which extends to the sides. Photographs of the final assembly are presented in the Figures 62 – 67:

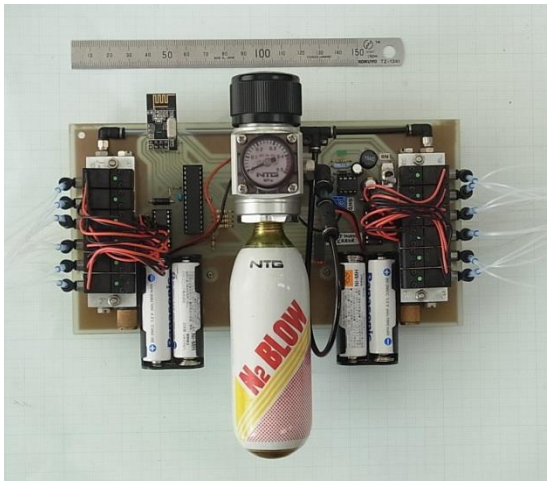


Figure 62. Robot, top view

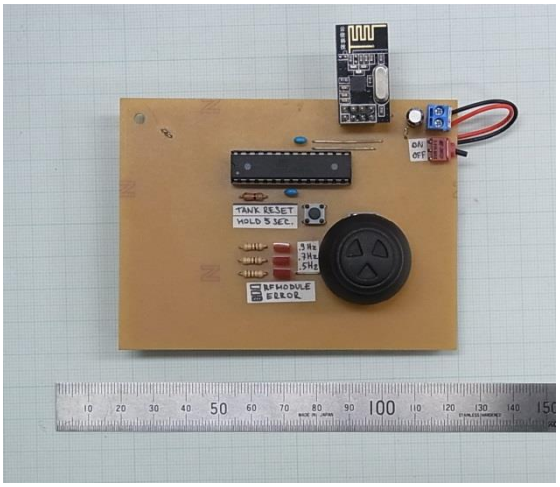


Figure 63. Remote control, top view

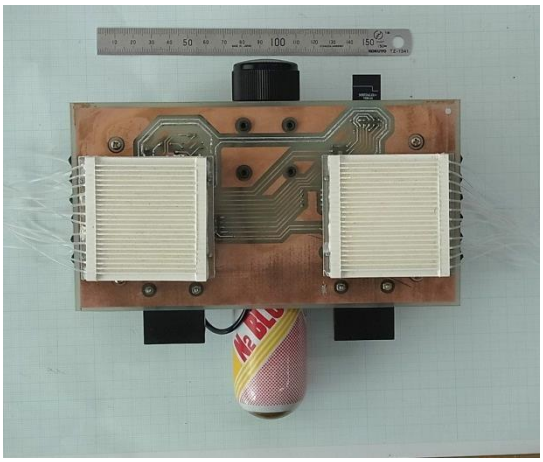


Figure 64. Robot, bottom view

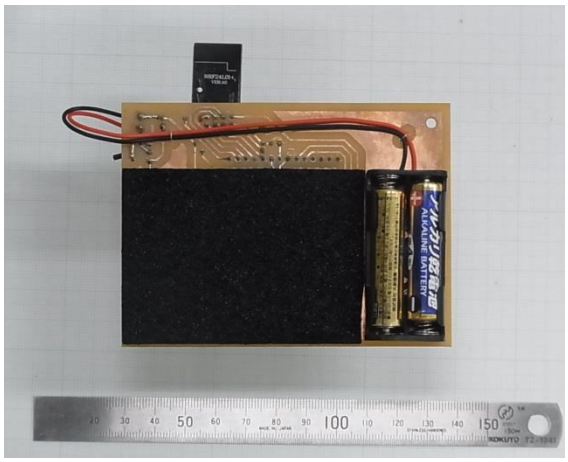


Figure 65. Remote control, bottom view

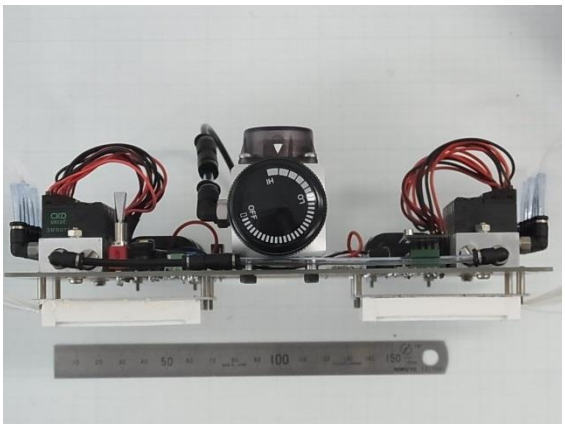


Figure 66. Robot front view

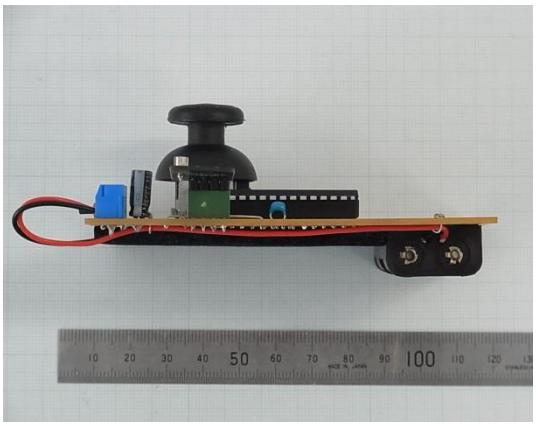


Figure 67. Remote control front view

Components of the remote are located in such way, that it is easy to hold and operate even with one hand. However, tank reset button is purposely placed behind the joystick bar, so it is difficult to reach it accidentally. Wire terminals, power switch positions and indicators of both boards are marked with adhesive labels.

The operating range of the radio link was tested in the college corridor and laboratory. Direct open range in the corridor is 74 meters; range through the concrete wall with steel security door of thicknesses 15 mm and 6 mm respectively is found to be 17 meters. Results show, that robot control operation is reliable and practically can be limited only by user's line of sight.

N_2 gas tank longevity is proven to be estimated correctly. Approximately after three minutes of normal speed motion, the air gauge gives a depletion signal by blinking the light, and before long, tank capacity runs out.

Robot travelling tests were conducted on the flat surface covered by paper. The friction ratio between actuator silicon and paper is exactly 0.5. Operating pressure of 0.2 MPa was applied to the actuator during the experiment. Speed versus frequency plot is presented in Figure 68.

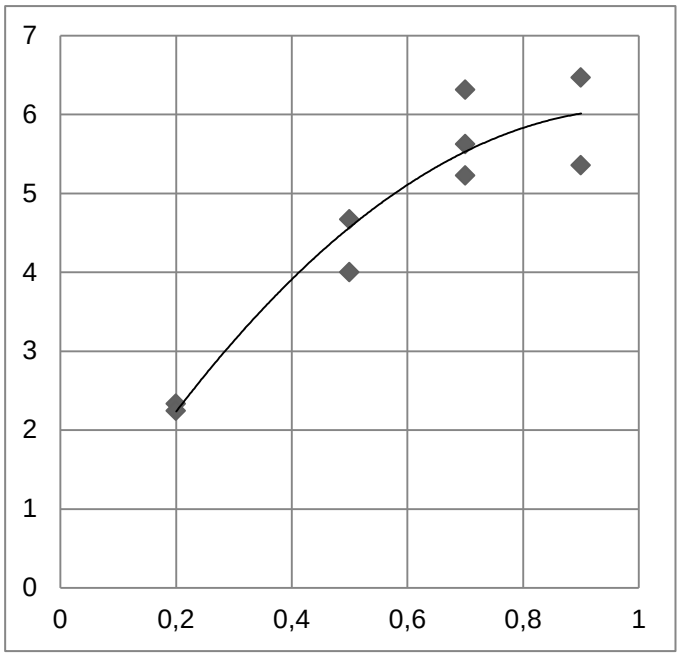


Figure 68. Speed (mm/s) versus driving frequency (Hz)

Compared to the previous travelling experiment data, given at Figure 7 in Section 3.1, these results show significant improvement in the travelling speed. Having same friction coefficient, this time actuators are carrying larger mass and have bigger amplitude due to higher operating pressure (previous tests are made at 0.15 MPa)

These results prove the assumption of increased step feed due to amplitude, and generally comply with theory of rectilinear locomotion, presented in the Section 3.2.

9 Conclusion

Harmonious operation of the controller system and controlled system involves fail-proof algorithms, with handling of many details and situations. This applies to the relationship between programs of different hierarchy, as well as to human – device interaction.

Interface of actuator is heavily dependent on the physical model, which is used to adapt output action to the input of the control system.

The bubbler actuator has demonstrated unusual way of transforming energy into motion. While it is not so efficient for straightforward portable application, it successfully imitates the nature. Observed details are summarized as follows:

Pros:

- Operation, by itself, is silent
- Waterproof and dustproof
- Compliant body, does not damage surface and has shape-adaptability
- Thin
- High load-to-weight ratio

Cons:

- Drains compressed air relatively fast
- Quite slow for most travelling appliances

Possible applications:

- Amphibian, cleaning of wet places
- Surgical remote and invasive diagnostics tool

10 Self-assessment

In this project I have introduced myself with the nature of robotics. Making of an autonomous system from analogue input to analogue output, with digital processing in between, have not been an easy task. During the research, many things have come in handy, and even more information was barely useful.

Actuators and driver algorithms were completely new for me, same for the portable systems. Now I am more fluent in embedded systems, battery operation, and design of various power supplies is not a problem anymore. A lot of creativity was put into actuator application, and for most of ideas, debugging and correction took place, revealing my own mistakes. A lot of programming-related tricks have been learned.

Most of all I appreciate to get in touch with pneumatics. The similarities between electronic and pneumatic operations have provided me with new insights.

Working alone in another country has improved my “project survivability”. I have revised own self-management, and learned to set priorities. Selecting only necessary

materials and make something with that little I have at hand, gave me more independence in whatever I was doing.

I thank Professor Tanaka and Professor Yamazaki for giving electrical hints; head of the Robot club Mr. Kobayashi for help with manufacturing circuit boards; and especially, Professor Miyagawa for exceptional supervision and inspiration.

References

1 Koichi SUZUMORI. Pneumatic Rubber Actuator Driven by Elastic Travelling Waves, JSME International Journal Series C, Vol. 42, No. 2, 1999

2 Caterpillar kinematics. John Brackenbury, Nature 390, 453 (4 December 1997) doi:10.1038/37253

3 Marvi H, Bridges J, Hu DL. 2013 Snakes mimic earthworms: propulsion using rectilinear travelling waves. J R Soc Interface 10: 20130188. dx.doi.org/10.1098/rsif.2013.0188

4 Kinematics of Soft-bodied, Legged Locomotion in Manduca sexta Larvae Biol. Bull.212:130 –142. (April 2007)

5 Atmel 8-bit Microcontroller with 4/8/16/32KBytes In-System Programmable Flash [DATASHEET] 8271G–AVR–02/2013

6 Atmel AVR042: AVR Hardware Design Considerations APPLICATION NOTE 2521L−AVR−07/2013

7 nRF24L01 Single Chip 2.4GHz Transceiver datasheet version 2.0, July 2007 www.nordicsemi.com/eng/nordic/download_resource/8041/1/42116424

8 avr nRF24L01 library running on atmega - v.02 davidegironi.blogspot.jp/2012/09/avr-nrf24l01-library-running-on-atmega.html

9 Math resources for MSP430G2553 – TI E2E Community forum e2e.ti.com/support/microcontrollers/msp430/f/166/p/248760/916614.aspx

10 Off on a tangent : a look at arctangent implementations, Jasper Vijn www.coranac.com/documents/arctangent/

11 Titi Trandafir. Fixed Point Two’s Complement CORDIC Arithmetic on MSP430 www.microtrendsys.com/ATC2004.pdf

12 CKD 3MA0/3MB0 3 port direct acting valve catalogue ckd.co-site.jp/ad/biochemical/Catalog/3M-Series.pdf

13 Review: Testing Sanyo’s Eneloop Low Self-Discharge Rechargeable Battery www.stefanv.com/electronics/sanyo_eneloop.html

14 Toshiba TD62003APG 7-channel Darlington Sink Driver www.toshiba.com/taec/components2/Datasheet_Sync/200910/DST_TD62003APG-TDE_EN_11444.pdf

15 Texas instruments LM2594/LM2594HV Datasheet, SNVS118C – DECEMBER 1999

Remote control program code

```
#define F_CPU 8000000UL
#define CLR 0x07
#define DOTNINE 0x01
#define DOTSEVEN 0x02
#define DOTFIVE 0x04
#define ABS(x) x>0 ? x : -x
#define ROT180 32760
#define ROT90 16380
#include <avr/io.h>
#include <avr/interrupt.h>
#include <math.h>
#include <util/delay.h>
#include <avr/sleep.h>
#include <nrf24l01.h>
#include <spi.h>

int16_t pot[2], str, ang;
uint8_t buffer[5], ack, press = 0, state = DOTSEVEN, lost;

const signed int atanAngles[14] = {
              0x12E3,
              0x09FB,
              0x0511,
              0x028B,
              0x0146,
              0x00A3,
              0x0051,
              0x0029,
              0x0014,
              0x000A,
              0x0005,
              0x0003,
              0x0002,
              0x0001
};

struct time{
              uint8_t lost;
              uint16_t reset;
              } delay;

union control
{
              uint8_t all;
              struct
              {
                            uint8_t reset:4;
                            uint8_t unused:4;
              };
};
```

```
union control signal;

void module_check();
void InitADC();
uint16_t ReadADC(uint8_t ADCchannel);
void cordic(int16_t x, int16_t y, int16_t *angle, int16_t *hypotenuse);
int16_t freq_change(uint8_t factor, int16_t tilt);
EMPTY_INTERRUPT(ADC_vect);

int main(void)
{
            _delay_ms(15);
            DDRD |= 0x07 ;
            PORTD |= 0xC0 ;
            module_check();
            nrf24l01_init();
            InitADC();
            PORTD &= ~CLR
            signal.all = 0;
            sei();
            while(1)
            {
                        pot[0] = ReadADC(0);
                        pot[1] = ReadADC(1);
                        pot[0] -= 512;
                        pot[1] -= 512;
                        cordic(pot[1], pot[0], &ang, &str)
                        str = str>512 ? 512 : str;
                        str = freq_change(state, str);
                        buffer[0] = ang >> 8;
                        buffer[1] = ang;
                        buffer[2] = str >> 8;
                        buffer[3] = str;
                        buffer[4] = signal.all;
                        ack = nrf24l01_write(buffer);
                        if(ack){
                                    PORTD |= state;
                                    delay.lost = 0;
                                    }
                                    else {
                                                delay.lost += 1;
                                                if (delay.lost > 60){
                                                            PORTD ^= state;
                                                            delay.lost = 0;
                                                }
                                                }
                        if(!(PIND & 0x80)){ press = 1; }
                        if((press)&&(PIND & 0x80)){
                                    press = 0;
                                    PORTD &= ~CLR;
                                    switch (state){
                                                case DOTFIVE:{
                                                            state = DOTSEVEN;
                                                            break;
```

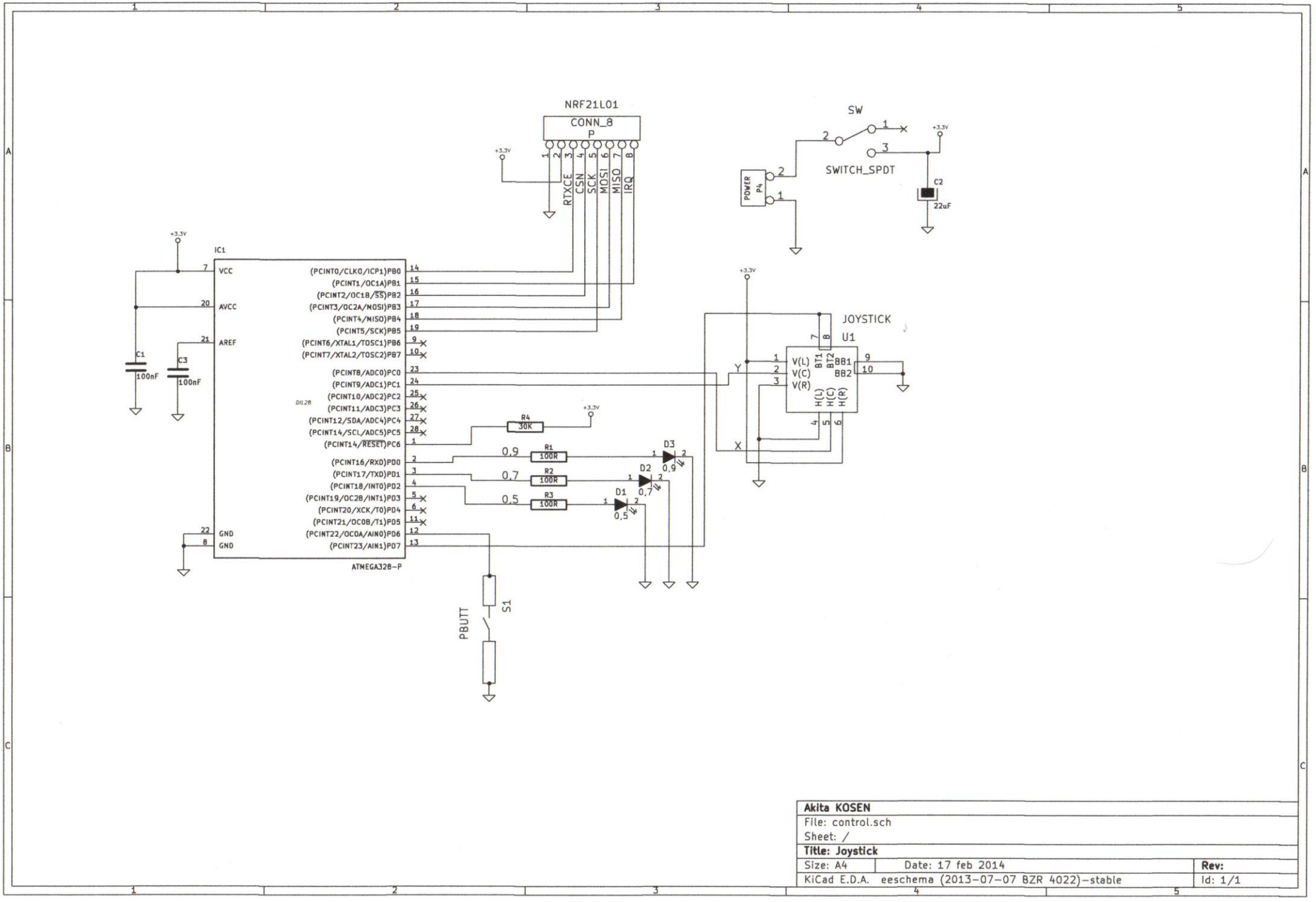
```
                                                }
                                                case DOTSEVEN:{
                                                        state = DOTNINE;
                                                        break;
                                                }
                                                case DOTNINE:{
                                                        state = DOTFIVE;
                                                        break;
                                                }
                                        }
                                }
                                if(!(PIND & 0x40)){
                                        delay.reset += 1;
                                        if (delay.reset > 1000){signal.reset = 11;}
                                }
                                else {
                                        signal.reset = 0;
                                        delay.reset = 0;
                                }
                        }
                }
```

Robot driver program code

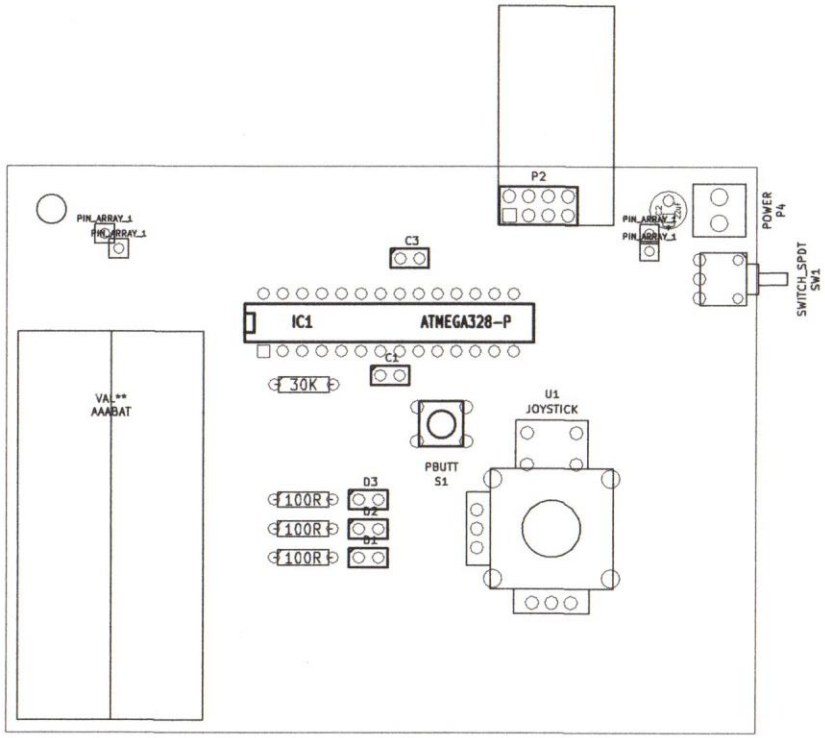
```
#define F_CPU 8000000UL
#define TMAX 26067
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <util/delay.h>
#include <nrf24l01.h>
#include <spi.h>

union control
{
		uint8_t all;
		struct
		{
			uint8_t reset:4;
			uint8_t unused:4;
		};
};

union control signal;

struct slots {
		uint8_t rdir:1;
		uint8_t ldir:1;
		uint8_t eco:1;
		uint8_t start:1;
		uint8_t reset:1;
		uint8_t presave:1;
		uint8_t unused:2;
		} flag;

uint8_t buffer[5];
volatile uint8_t rpos = 1, lpos = 4, disconnect = 0;
int16_t heading;
uint16_t air = 0, incline, exhaust = 0;

struct timing {
		uint16_t left;
		uint16_t right;
		int16_t angle;
		};

struct timing temp;
struct timing clean;

void show_air(uint16_t value);
void EEPROM_write(uint16_t uiAddress, uint8_t ucData);
uint8_t EEPROM_read(uint16_t uiAddress);
uint16_t memory(uint16_t bytes, uint8_t io);
void timer_period(int16_t ang, uint16_t str);
void module_check(uint16_t post)
```

```
ISR(TIMER1_COMPA_vect);
ISR(TIMER1_COMPB_vect);

int main(void)
{
		_delay_ms(150);
		DDRD |= 0xFF;
		DDRB |= 0xC0;
		DDRC |= 0x3F;
		air = memory(0, 0);
		module_check(air);
		TIMSK1 = (1 << OCIE1A)|(1 << OCIE1B);
		nrf24l01_init();
		flag.start = 1;
		flag.reset = 0;
		while(1)
		{
			_delay_us(500);
			if(nrf24l01_readready(0)){
				nrf24l01_read(buffer);
				heading = buffer[0] << 8 | buffer[1];
				incline = buffer[2] << 8 | buffer[3];
				signal.all = buffer[4];
				disconnect = 0;
				flag.presave = 1;
			}
			else {

				disconnect += 1;
				if (disconnect > 10){
					heading = 0;
					incline = 0;
					disconnect = 0;
					if (flag.presave){
						flag.presave = 0;
						memory(air, 1);
					}
				}
				}
			if (incline > 0){
				exhaust = 0;
				flag.eco = 1;
				timer_period(heading, incline);
				TCCR1B |= (1<<CS12);
				if(flag.start){
					OCR1B = 5787;
					OCR1A = 5787;
					flag.start = 0;
				}
			}
			else{

				TCCR1B = 0;
				if (flag.eco){exhaust += 1;}
```

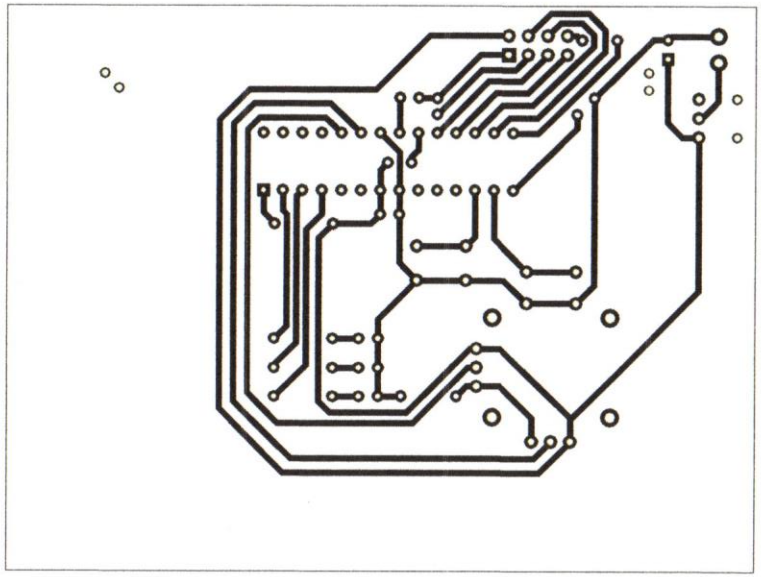
```
                        if (exhaust > 5000){
                                TCNT1 = 0;
                                flag.start = 1;
                                flag.eco = 0;
                                exhaust = 0;
                                PORTC &= ~0x3F;
                                PORTB &= ~0xC0;
                                PORTD &= ~0xF0;
                                air += 12;
                                rpos = 1; lpos = 4;
                                memory(air, 1);
                                }
                }
                if (air > 60000) {air = 5000;}
                if ((signal.reset == 11)&&!(flag.reset)){
                        flag.reset = 1;
                        air = 0;
                        memory(0, 1);
                }
                if (!signal.reset){flag.reset = 0;}
                }
}
```



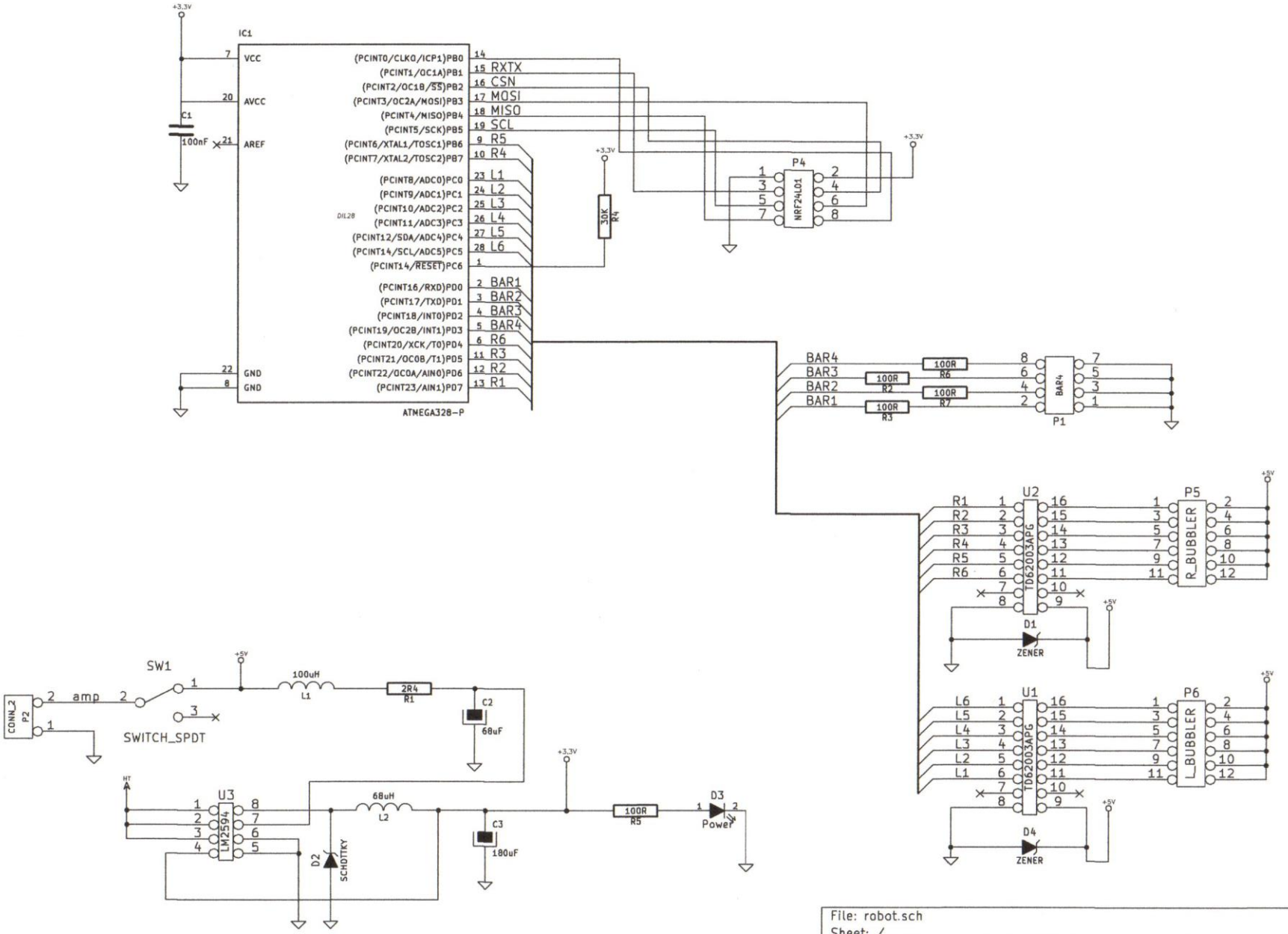
Akita KOSEN		
File: control.sch		
Sheet: /		
Title: Joystick		
Size: A4	Date: 17 feb 2014	**Rev:**
KiCad E.D.A. eeschema (2013-07-07 BZR 4022)-stable		Id: 1/1

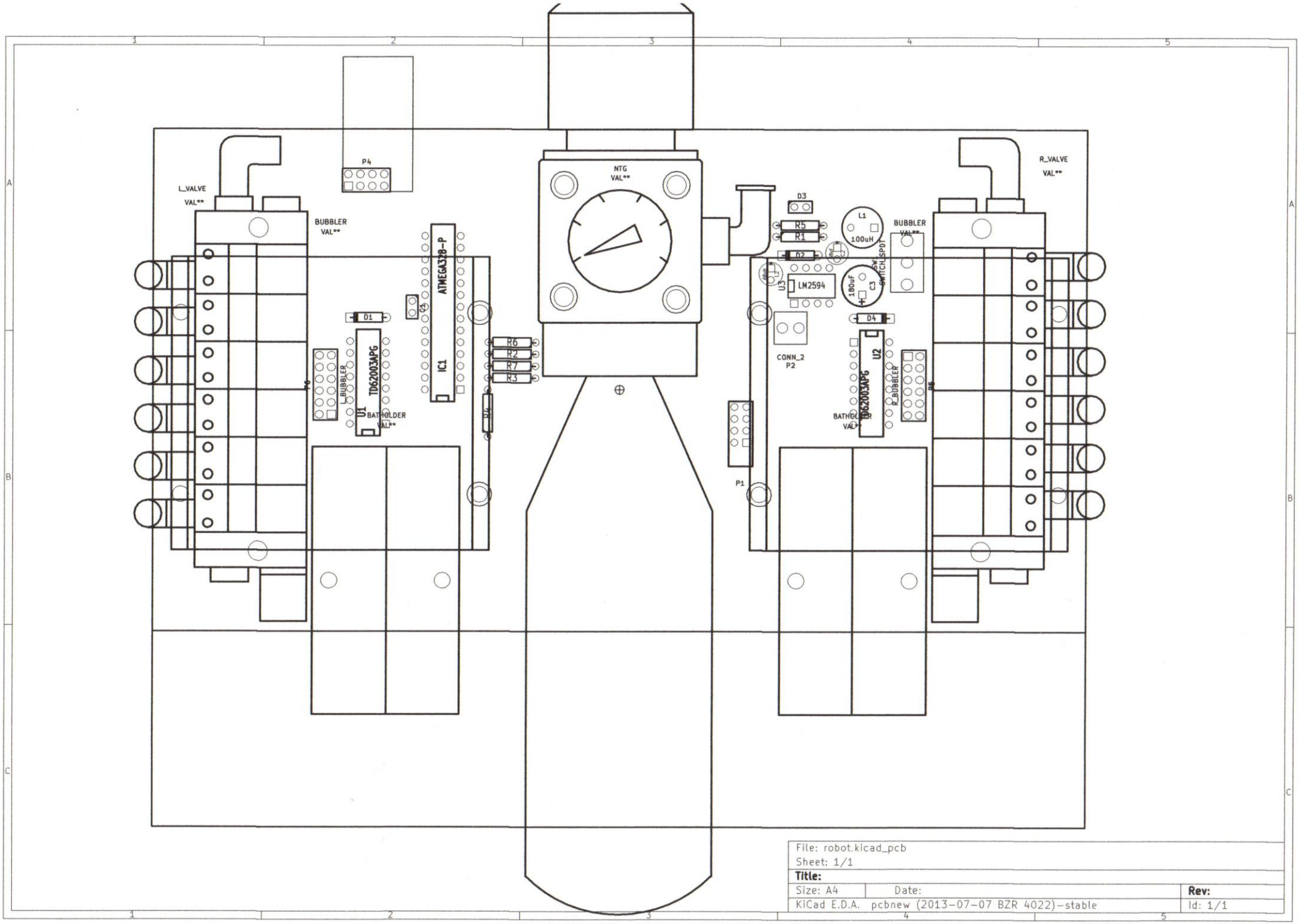


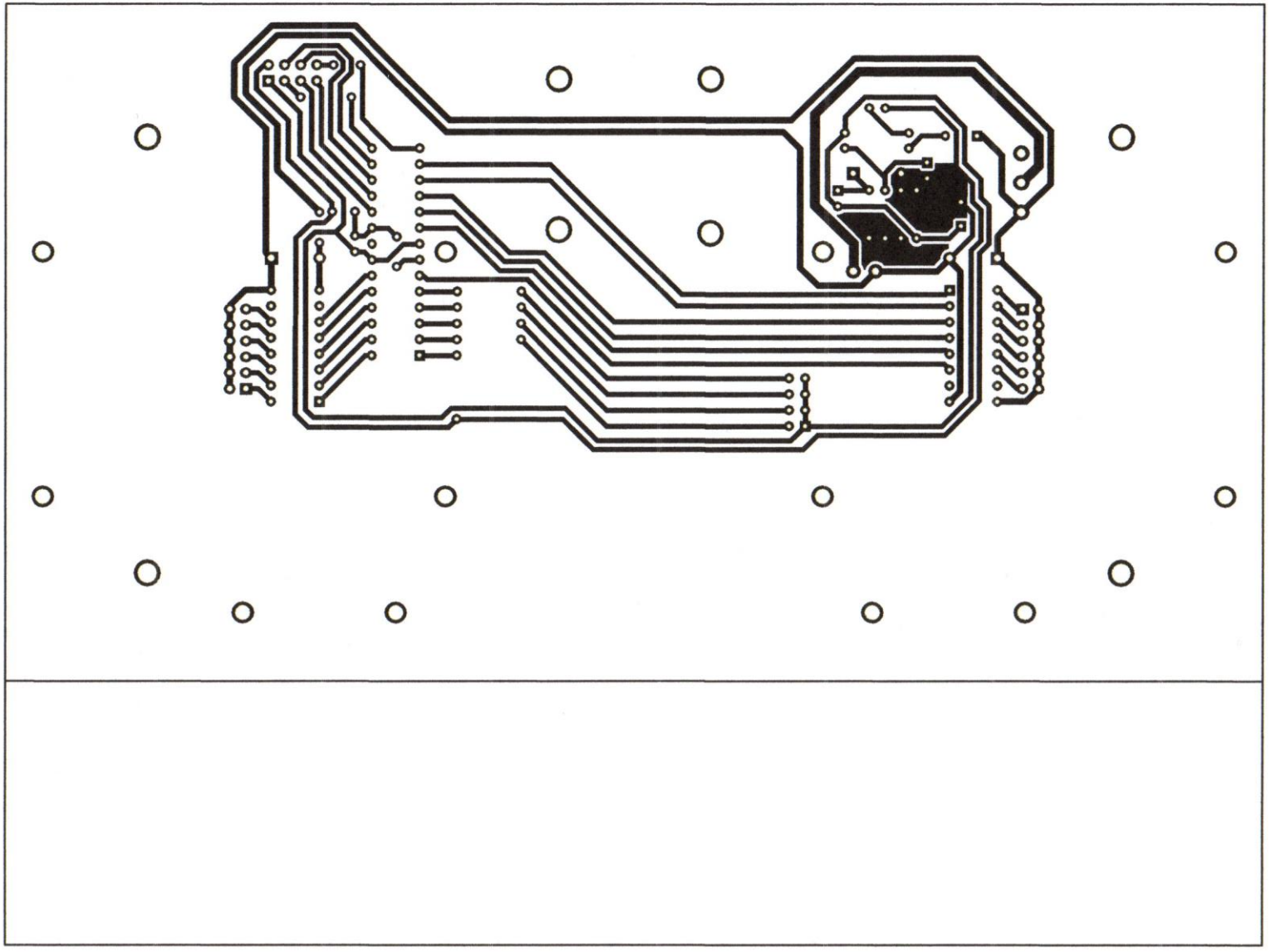
File: control.kicad_pcb		
Sheet: 1/1		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A.	pcbnew (2013-07-07 BZR 4022)-stable	Id: 1/1



File: control.kicad_pcb		
Sheet: 1/1		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. pcbnew (2013-07-07 BZR 4022)-stable		Id: 1/1


IC1
ATMEGA328-P
DIL28
VCC
AVCC
AREF
GND
GND
C1
100nF
(PCINT0/CLKO/ICP1)PB0
(PCINT1/OC1A)PB1
(PCINT2/OC1B/SS)PB2
(PCINT3/OC2A/MOSI)PB3
(PCINT4/MISO)PB4
(PCINT5/SCK)PB5
(PCINT6/XTAL1/TOSC1)PB6
(PCINT7/XTAL2/TOSC2)PB7
(PCINT8/ADC0)PC0
(PCINT9/ADC1)PC1
(PCINT10/ADC2)PC2
(PCINT11/ADC3)PC3
(PCINT12/SDA/ADC4)PC4
(PCINT14/SCL/ADC5)PC5
(PCINT14/RESET)PC6
(PCINT16/RXD)PD0
(PCINT17/TXD)PD1
(PCINT18/INT0)PD2
(PCINT19/OC2B/INT1)PD3
(PCINT20/XCK/T0)PD4
(PCINT21/OC0B/T1)PD5
(PCINT22/OC0A/AIN0)PD6
(PCINT23/AIN1)PD7
RXTX
CSN
MOSI
MISO
SCL
R5
R4
L1
L2
L3
L4
L5
L6
BAR1
BAR2
BAR3
BAR4
R6
R3
R2
R1
+3.3V
30K
R4
P4
NRF24L01
P1
BAR4
100R
R6
R2
R7
R3
U2
TD62003APG
P5
R_BUBBLER
D1
ZENER
+5V
U1
P6
L_BUBBLER
D4
SW1
SWITCH_SPDT
amp
CONN_2
P2
100uH
L1
2R4
R1
C2
68uF
U3
LM2594
68uH
L2
C3
180uF
D2
SCHOTTKY
100R
R5
D3
Power
File: robot.sch
Sheet: /
Title:
Size: A4
Date: 17 feb 2014
Rev:
KiCad E.D.A. eeschema (2013-07-07 BZR 4022)-stable
Id: 1/1


L_VALVE
VAL**
BUBBLER
VAL**
P4
NTG
VAL**
R_VALVE
VAL**
BUBBLER
VAL**
D3
R5
R1
D2
L1
100uH
U3
LM2594
180uF
C3
SWITCHSPDT
D4
CONN_2
P2
U2
TD62003APG
R_BUBBLER
R6
R2
R7
R3
R4
D1
C1
ATMEGA328-P
IC1
L_BUBBLER
U1
TD62003APG
BATHOLDER
VAL**
P1
File: robot.kicad_pcb
Sheet: 1/1
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. pcbnew (2013-07-07 BZR 4022)-stable
Id: 1/1



File: robot.kicad_pcb		
Sheet: 1/1		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A.	pcbnew (2013-07-07 BZR 4022)-stable	Id: 1/1