Light seeking robot swarm

TTOW0310 Embedded Systems Design and Development
Final report

Lecturer: Jouko Kotkansalo

Team: Esa Jääskeläinen

Philipp Brogli

Table of Contents

Preface

The original start report of the course described our goal as follows:

For fishes in the shallow water a quick detection of a shadow can decide between life and death. With that in mind we build five identical robots which simulate similar behaviour.

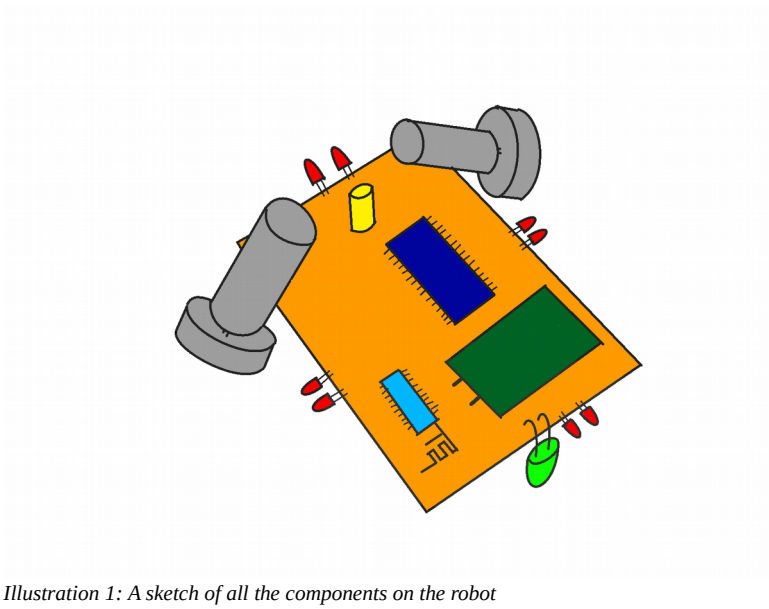

Illustration 1: A sketch of all the components on the robot

The robots all get placed onto a flat surface which is at maximum 2 by 2 meters wide. With an LDR (Figure 1: yellow) each robot measures the brightness above it. For that to work best there has to be a ceiling with a homogenic brightness. With two motors (Figure 1: grey) they can move on top of the surface. An RGB LED (Figure 1: light green) can be used to show the status of the robot. It is pointed downwards and is also used by the robot to glide on. That reduces the friction and simplifies the mechanical parts of the robot. With IR LEDs (Figure 1: red) the Robot assumes the direction and distance of the other robots. The robots communicate with each other with the use of a radio (Figure 1: light blue). The Microcontroller (Figure 1: dark blue) controls every move of the robot. And the robot is powered by a battery (Figure 1: dark green).

As a final result, the robots compare the brightness of themselves with the one of the others. If theirs is noticeably darker than one of the others, they move to the brighter one. So if you light with a flashlight onto one of the robots all of them should move towards that spot. If the brightness of only one robot is darker than the others it moves to the closest one with a brighter value. Therefore if you cover one or more robots with a black object they move out of the dark spot to another robot. If all the brightness's are the same they stay where they are.

In the beginning it was decided that I (Esa) would mostly work with the hardware, as I had the needed tools and materials to design and build the hardware-side, and Philipp would concentrate more on the software, especially the radio communication, and help with the hardware and general design on the side.

On top of the practical reasons for dividing the hardware and software like this (Philipp, being an exchange student here just for the semester, of course didn't have his own electronics "workshop" to do things in), I've worked about a decade as a software designer, and programmed a couple of decades as a hobbyist, while electronics is a fairly recent hobby of mine (couple of years, although I do have some background in basics of electronics from earlier 2nd level studies from around the turn of the millennia and later in JAMK around 2005-2006). Philipp on the other hand is majoring on

electrical engineering, with computer science/software as a minor (or at least that's how I understood it), so that both of us would work on our "weaker" fields.

We knew from the start that this would be difficult to complete with the limited time available. The final robots made have all the necessary hardware and most of the software components, yet the software wasn't completely finished, due to delays with getting the hardware done, issues with the radio modules and lack of time.

This report describes the design, development and testing of the hardware, the software design and components that were made and the conclusions. The hardware text is mine, Philipp wrote almost all of the Software-parts.

System parts

A lot of the components used in the project were picked based on what was already available ”on the shelf”, like common passives, 74-series logic ICs, radio modules, microcontroller-boards, RGB-LEDs and such. Many more special-purpose components still needed to be ordered specifically for the project, such as the DC/DC -switching mode converter (buck converter) components, all IR-components and motor-drive chips, and some extra motors and radio-modules, as not enough was available from the start for all the robots.

Microcontroller

The MCU (Microcontroller unit) chosen for the project was Atmel ATMega328, or more precisely, an Arduino Nano, which uses ATMega328P. Other options would have been other Arduinos based on different ATMegas, Espressif ESP8266s, available in various boards, which run at a higher clock rate, have more program- and data memory, and also have WiFi (WLAN) -connection capability, and ”Blue pill” ARM-boards based on ST Microelectronix’s STM32F103, an ARM Cortex M3, which again would have higher clock rate than the ATMega and has more memory.

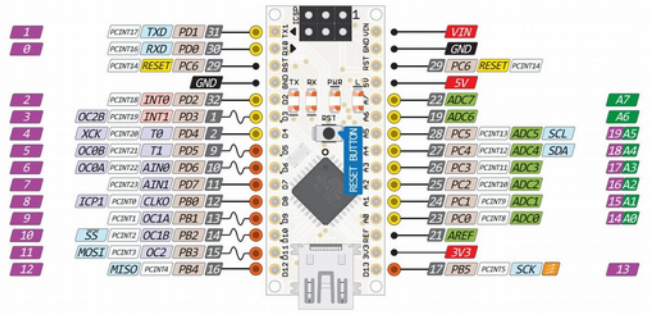

Illustration 2: Arduino Nano pinout

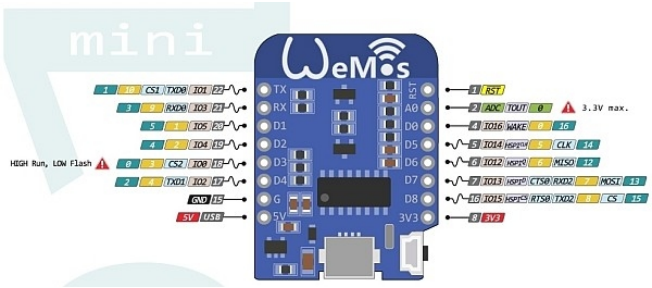

Illustration 3: Wemos D1 Mini pinout

Wemos D1 Mini, a board based on Espressif ESP8266, was discarded due to having a low amount of GPIO -pins available. Also the WiFi-capability and TCP/IP -stack wouldn’t be useful in this project.

The STM32F103 -based ”Blue pill” -boards would have lots of GPIO-pins, memory and performance, but programming an ARM-based MCU would be more complex. Also, the program needed for the robots wouldn’t be that big or use large amounts of memory, so most of the extra program- and data-memory would just be wasted.

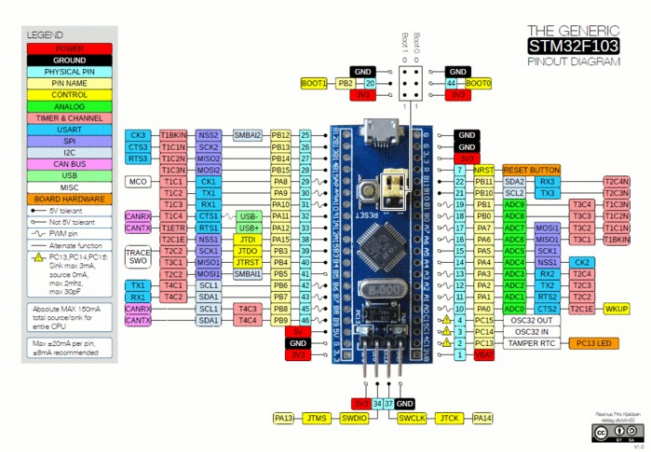

Illustration 4: ”Blue pill”, a generic STM32F103 -board pinout

Arduino Nano / ATMega328 was chosen based on having a large enough number of GPIO -pins, likely a sufficient amount of memory and being

relatively simple to program on, because a lot of the platform details are hidden behind the Arduino core libraries. If need be, the underlying registers and AVR library -functions could still be used.

Power supply

Since the robots should be mobile and autonomous, it was decided early on that they would have to carry their own power supply with them, which should be some type of battery. To prevent waste and needing to buy more, the battery should be rechargeable, relatively light and small for the robot to carry, and ideally of a common form-factor, to make it reusable in other future projects.

Battery selection

Lithium-ion cells were considered due to their higher energy density and voltage, which is nominally 3.6...3.7V per cell and 4.2V per cell for full charge for most chemistries (such as LCO, NMC and NCA, but excluding Li-titanate and LiFePo) [1]. Unfortunately, a good quality, high capacity Lithium-Nickel-Cobalt-Aluminum (NCA) or Lithium-Nickel-Manganese-Cobalt (NMC) 18650-cell from a well-known manufacturer (such as Sony, Panasonic, LG, Samsung...) costs about 5-10€/piece [2], and depending on whether the voltage would be stepped up (boost-converter) or down (linear regulator or buck converter), 5 or 10 pieces would be needed. While I did have a good number LG MH1 (NCA, 3100mAh) -cells available, they would have to be dismantled from 16S -packs (16 cells in series, 67.2V maximum voltage), and ”naked”, unprotected lithium-ion cells can also be very dangerous, as a shorted or overcharged cell can catch fire or explode violently, so the idea was abandoned.

Separate single alkaline or rechargeable AA or AAA NiMH-cells were also considered, but they would require more space for multiple cells in series, or a step-up (boost) converter to bring up the voltage high enough for a lower amount of cells. Also, stepping up the voltage draws more current from the cell than the actual circuitry uses, so the internal resistance could become a problem.

At this point we knew we’d likely be regulating down to needed voltages (5V for the motors and MCU, 3.3V for the radio-module) instead of stepping up. The final decision to use 9V rechargeable NiMH-batteries came out later, so most of the calculations were done using a wider possible input voltage range, using 6V as minimum and 10V as maximum, to still account for the possibility of using lithium. Also the amount of current needed was an unknown, since the final motors to use were also still undecided, so 1-2A was used as an estimate for maximum current for the most parts.

It should also be noted that most ”9V” NiMHs aren’t actually 9V (nominal), as the voltage depends on the charge state, and internally they may contain 6, 7 or 8 NiMH-cells in series. With 1.2V nominal voltage per cell, they have a nominal voltage of 7.2V, 8.4V or 9.6V for 6, 7 and 8 cells, respectively. Discharged, the cell voltage drops to around 1V per cell, and fully charged, goes to about 1.4-1.5V per cell, although it is said that the open circuit voltage drops fast back towards the nominal voltage when starting to discharge. Also series of cells without any way to balance the cells

means that some of the cells might get over- or undercharged and possibly the entire battery dies because of a single cell. Unlike lithium-ion cells, NiMH suffers from the ”memory effect”[3], where partial discharge can lead to loss of capacity, although it is at least partially reversable through charge/discharge -cycling.

The batteries used in the robots were Clas Ohlson 200mAh / 7-cell rechargeable NiMH-batteries.

Voltage regulation

Linear regulation would be easy to implement, but would waste power through the voltage drop of the regulator. Although the motors shouldn’t run for very long at a time, depending on the amount of current the motors would draw, power dissipation in the regulator could become a problem. Also it was suspected that the runtime of a robot per charge would be reduced, as it would discharge the battery faster through wasted power. At a nominal voltage of 8.4V, regulating it down to 5V with a linear regulator would mean that 3.4V would have to be dropped over the regulator. In reality, the output voltage of the battery would also drop along with the current draw, due to the internal resistance of battery itself. Still, assuming that the voltage drop of the battery itself would be negligible, drawing 1A from the battery would mean 3.4W of power dissipation on the regulator.

Something like an AMS1117-5.0, a 5V linear regulator, in a SOT-223 -package has junction-to-ambient thermal resistance of about 90°C/W, although it may be lower depending how much heat can be shed through the tab, and a maximum junction temperature of 125°C [4]. When dissipating entire watts, while internal protections may shutdown the device to protect itself during overheating or overload, it would still mean that the robots could reboot suddenly during operation when the voltage momentarily drops too low. While the needed current might be over-estimated, it was decided that that a switching regulator would be a better option.

Step-down (buck) converter

Neither of us had experience with designing switching regulators, so research needed to be conducted into how such devices are designed. There are multiple sources available in the internet, which were studied to gain general knowledge, as well as sub-chapter 9.6 of the Voltage Regulation and Power Conversion -chapter dedicated to switching regulators in The Art of Electronics[5].

While learning the basics wasn’t that hard, there are very many intricate details that go into switching mode power supply design, like different topologies and continuous vs. discontinuous conduction modes. Luckily, a wide range of controller ICs for switching-mode power supplies (SMPS) and in this case, specifically switching DC/DC -converters, are readily available from multiple manufacturers, with typical application schematics and equations for calculating needed component values, easing the work a lot.

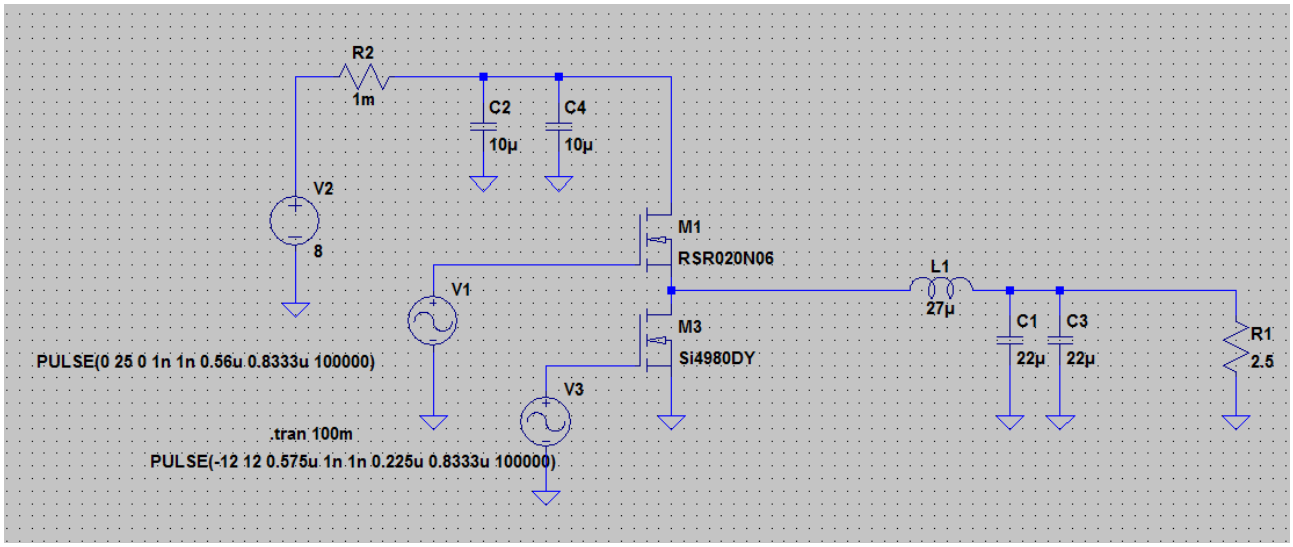

Illustration 5: A crude simulation of a switching regulator in LTSpice. At this point, the values for the capacitors are already the same as the ones picked for the actual circuit. The duty-cycle is "hand-tuned" to give out 5V over the load R1 at roughly 1.2MHz, this was mostly to see if I was even in the right ballpark with the ripple-calculations.

Converter IC selection

I spent a good time searching ICs from Farnell's catalog, and after reading more on the subject, went for the converter ICs that used much higher frequencies than most of the older designs, which usually seemed to be around 50-150kHz. The final pick was Richtek's RT7297B, a 3A max, 18V max input, 1.2MHz synchronous step-down converter [6]. The higher frequency not only requires a much smaller inductor than the lower frequency ones, but it also doesn't require that much capacitance to filter the ripple, as is shown in the following equations, making it possible to use cheap, small and very low ESR MLCCs (multi-layer ceramic capacitors) in SMD-packages instead of more bulky low ESR polymer-capacitors or such.

The only downside I saw was that it used an SOP-8 (Small Outline Package, 8 pins) -package with exposed pad, something I'd never seen before and to which the standard libraries of KiCad, the EDA-package I use, didn't have a footprint for. Creating a custom footprint is fairly easy though, and the datasheets contained the dimensions, so I went with the chip.

Inductor selection

While the datasheet of RT7297B does not give out an equation for an inductor-value (it does list out suggested values for different output voltages, though), there are multiple other sources describing switching regulator design for different topologies based on known, estimated or wanted values (such as [7] [8] [9] [10], among others), and give out equations for calculating values for the inductor, ripple voltages & currents and capacitor values. I must admit that I didn't read all the sources word-for-word, nor did I write down all the sites, in general, the field of SMPS-design seems to be so vast that it would have taken more time than the entire project to get a good understanding of all the details.

In the end, I used an equation for picking a suitable inductor-value for a switching regulator, in the form of

$$L_{min} = \frac{(V_{in} - V_{out}) \cdot V_{out}}{\Delta I_L \cdot f \cdot V_{in}}$$, although many sources write it in form

$$L_{min} = [\frac{V_{out}}{f \cdot \Delta I_L}] \cdot [1 - \frac{V_{out}}{V_{in}}]$$,

where L_{min} is the minimum inductance, V_{in} is the input voltage (maximum should be used), V_{out} is the output of the regulator, f is the frequency of the switcher and ΔI_L is the ripple current going through the inductor.

Initially I used far too low ripple current values based on some earlier calculations (which aren't reproduced in this report, as they're a poorly noted scribbles on multiple pages of my pad), likely for some other topology, and ended up with far too high values for the inductance, above 25µH. The datasheet suggested that 6.8µH would be enough for 5V output, but after the teacher confirmed that using a higher inductance likely won't hurt and gives less voltage ripple (but also slower transient response), I ordered both 8.2µH and 27µH power inductors with RMS currents above 2A, and made test boards for both with similarly valued other components to compare them.

Inductors used:

BOURNS SDE1006A-8R2M SMD power inductor, 8.2 µH, 4.1A I_{rms}, 4.3A I_{sat}, unshielded, 48mΩ

BOURNS SDE1006A-270M SMD power inductor, 27µH, 2.3A I_{rms}, 2.5A I_{sat}, unshielded, 110mΩ

Since I now knew (at least roughly) the amount of inductance to use, I could calculate the correct ripple current values:

$$\Delta I_L = [\frac{V_{out}}{f \cdot L}] \cdot [1 - \frac{V_{out}}{V_{in}}]$$,

where f = 1.2MHz, V_{out} = 5V

V_{in} = 10V L=8.2µH : 0.254065A L=27µH : 0.07716A

V_{in} = 6V L=8.2µH : 0.084688A L=27µH : 0.002178A

Having at least ballpark -figures for current ripples with the chosen inductor-values, I could then move on to check the capacitors the datasheets suggested. Not that I have any reason to doubt the manufacturer figures, but it couldn't hurt.

C_{in} and C_{out} selection

The capacitors have a very important role in switching mode power supplies, as they smooth the output and input voltage while the inductor is switched to alter the output voltage. Ripple currents will flow through the capacitors, and low ESR (equivalent series resistance) is a requirement to keep the power dissipation and output voltage ripple down.

RMS current at input capacitor can be calculated with the equation

$$I_{RMS} = \frac{I_{out(MAX)} \cdot V_{out}}{V_{in}} \cdot \sqrt{\frac{V_{in}}{V_{out}} - 1} \quad ,$$

where I_{RMS} is the RMS (root mean square) -value of the current ripple, $I_{out(MAX)}$ is the maximum output current from the converter, and V_{in} and V_{out} are the input- and output voltages, respectively.

As the highest ripple current occurs at $V_{in} = 2 * V_{out}$, which also happens at the values chosen for the input and output (10V and 5V), the peak ripple current is half of the output current

$$I_{RMS(MAX)} = I_{out(MAX)} / 2$$

At 2A peak, this means 1A ripple current. Taiyo Yuden datasheet graph shows that with 1A/1MHz ripple current, the capacitor should heat up about 2 degrees Celcius above ambient.

The equation for minimum capacitance *C* is

$$C = I_{peak} \cdot \left(\frac{T_{on(max)}}{2 \cdot \Delta V_{out}}\right) \quad ,$$

where I_{peak} is the peak current, $T_{on(max)}$ is the time the output inductor is connected to power, calculated from the frequency and maximum duty cycle of the switching controller, and ΔV_{out} is the wanted (or tolerated) ripple voltage on the output.

The $T_{on(max)}$ -value was calculated from the minimum (1MHz) switching frequency and maximum duty cycle of 78% of the Richtek chip, the datasheet lists the frequency as 1.2MHz typical, 1MHz min and 1.4MHz max. 2A peak current was best guess at this point, as we didn't know how much current the motors would draw. 50 mV maximum ripple seemed more or less "reasonable" at 1% of output voltage, which I later on learned that for example common ATX-standard power supplies for computers allow[11]. There are usually no specific maximum voltage ripple specifications for the digital parts, but up to 3% ripple has been suggested as "tolerable" for ATMegas, for example.

The values used were $I_{peak} = 2A$ (worst case), $T_{on(max)} =$ 0.78uS, and ΔV_{out} = 50mV. Thanks to the high switching frequency, the output minimum capacitance needed for 50mV ripple at 2A is only around 16µF. The Richtek datasheet suggests 2 x 22µF MLCCs at the output though, going as far as to suggest specific manufacturers and models for the capacitors.

The final models were picked from the suggestions, using two paralleled Murata 22uF / 10V +-20% X5R (GRM31CR61A226ME19) for output, and two paralleled Taiyo Yuden 10uF / 25V +-20% X5R (TMK316BJ106ML-T) for input, both in 1206 SMD packages. At later stage of the design, a 470uF electrolytic (unknown Chinese brand) was also added near the input rail.

The capacitors have low ESR (judging from the datasheet graph, the ESR of both caps is roughly 10mΩ at 1..1.4MHz) and don't heat up much with ripple currents at or below 2A.

The specifications for the capacitors are

C_{out} = 2 x Murata GRM31CR61A226ME19 22uF / 10V +-20% X5R
http://www.farnell.com/datasheets/2047895.pdf?_ga=2.225754714.1714307451.1511982804-1104929158.1511982804

ESR around 10 milliohms @ 1.2MHz, temp rises around 1.5°C @ 1A, 4°C @ 2A, for 1MHz ripple

C_{in} = 2 x Taiyo Yuden TMK316BJ106ML-T 1206 MLCC, 10uF / 25V +-20% X5R

http://www.farnell.com/datasheets/2081945.pdf?_ga=2.225754714.1714307451.1511982804-1104929158.1511982804

ESR around 10 milliohms @ 1.2MHz, temp rises around 2°C @ 1A, 9°C @ 2A, for 1MHz ripple

Output ripple voltage

The maximum output voltage ripple can be estimated by the equation

$$\Delta V_{out} \leq \Delta I_L \cdot [ESR + \frac{1}{8 \cdot f \cdot C_{out}}] \quad ,$$

where ΔV_{out} is the voltage ripple, ΔI_L is the current ripple, ESR is the capacitor Equivalent Series Resistance, f is the switching frequency and C_{out} is the output capacitance.

According to this, the ripples should be at or below

8.2µH ΔI_L = 0.254065A => ΔV_{out} = 0.00314V = 1.872mV

27µH ΔI_L = 0.07716A => ΔV_{out} = 0.000954V = 0.568mV

However, the final measured voltage ripples for the test boards were larger than this, especially for the 27µH test board.

Power dissipation

Although I'm not completely sure how the power dissipation for a switching controller should be calculated, at least a rough estimation can be done based on the on-resistances ($R_{ds(ON)1}$ and $R_{ds(ON)2}$) of the internal MOSFET-switches and the maximum currents.

The estimated average power dissipation of a single switch (P_{sw}) should be

$$P_{sw} = I_{avg}^{2} * R_{ds(ON)} \quad ,$$

where I_{avg} is the average current passing through the switch and $R_{ds(ON)}$ is the on-resistance of the drain-source -channel, given in the datasheet, 110mΩ for the high-side switch and 90mΩ for the low-side switch. The function block diagram of the datasheet also shows a current sense resistor R_{sense} above the high-side switch, but no value is given for the resistance. It is unclear whether the sense resistor value is included in the high-side switch on-resistance.

To give even more headroom (pessimistic approach), it could be said that the average current is as high as the output current of the switcher, and that the high side is being kept constantly open at full 100% duty cycle. Of course this is not true, as the operation of the switcher is based on turning the switches on and off to pulse the inductor, but it gives a maximum upper limit to the power dissipation. Even with 2A continuous output current and 110mΩ on-resistance, the power dissipation is still only 0.44W. Of course this doesn't take into account the switching losses of the internal MOSFETs, as they pass through the linear-region before turning fully on or off.

The thermal junction-to-ambient resistance (θJA) is given as 75°C/W and junction-to-case (θJC) is given as 15°C/W for the exposed pad, although the numbers are based on 4-layer board, which has more thermal mass than a single- or double-sided board:

> *Note 2. θJA is measured at TA = 25°C on a high effective thermal conductivity four-layer test board per JEDEC 51-7. θJC is measured at the exposed pad of the package*

The absolute maximum junction temperature is listed as 150°C.

To continue on with pessimistic approach, the θJA (junction-to-ambient thermal resistance) could be estimated to be much, much higher, for example at 125°C/W, although in reality it's probably lower than that even with single-layer board, of course depending on the thermal connection between the exposed pad and copper layer, and copper layer size and thermal resistance to ambient air. Even with this number, in 25°C ambient temperature, the chip should be able to function up to 1W power dissipation. This estimation should leave enough headroom, and the power dissipation of the chip should not become a problem under normal circumstances, unless the thermal resistance could be even higher, or the internal R_{sense} -resistance is not included in the high-side switch value and is relatively high.

In the actual boards, the exposed pad is soldered directly to exposed copper underneath the chip with solder paste and hot air, and the chip is in free air, not encased and with no other components running hot around it. To make sure the thermal connection is well made, the pad was soldered first, checked that the chip is firmly connected to board, and then the legs were separately soldered. In

the final fabricated 2-layer boards, four through-plated vias were placed under the pad to help dissipating the heat through copper ground-planes on both sides.

Test boards

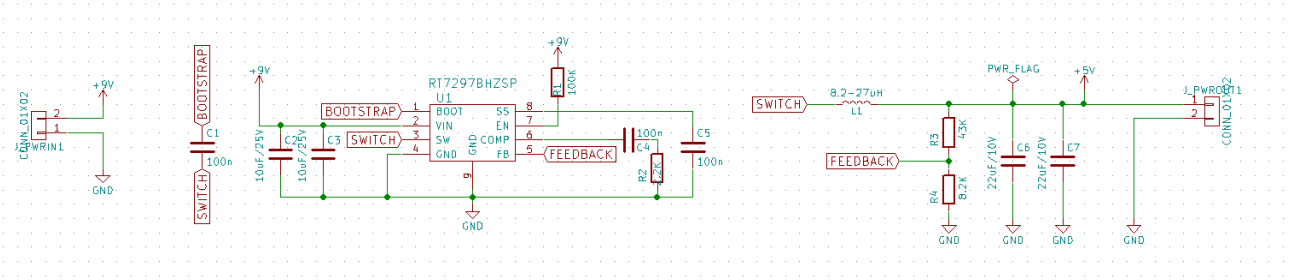

Illustration 6: Schematic for the SMPS test-boards using RT7297B

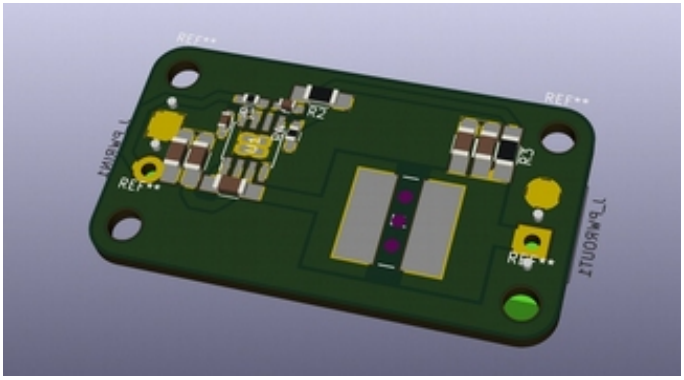

Illustration 7: 3D-view of the layout design. The landing pads for the inductor are larger in case at a later point physically larger inductors would be used.

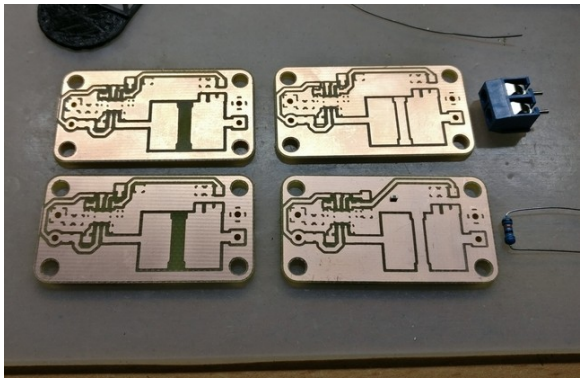

Illustration 8: Milled testing boards, with slightly different copper removals around the inductor landing pads.

The testing board schematic was pretty much straight out of the datasheet for the chip. The passives in addition to the input- and output-capacitors are for setting the feedback voltage, so that the output is 5V and for soft-start and compensation of the regulation loop.

For 5V output, the datasheet suggests values of 62kΩ for R3 and 11.8kΩ for R4 in the schematic for feedback voltage division. As my resistor-selection for 0603 through 1206 SMD-packages is based on E24-series, I didn't have such values available. The original values give a ratio of 0,1903… for the resistors (R4/R3), so quickly trying against E24-values gave a pretty close match with 43kΩ and 8.2kΩ (8.2kΩ / 43kΩ = 0.1906…). More formally, the controller adjusts the duty cycle to cause an output voltage so that the feedback voltage is at the reference voltage of 0.8V (±1.5%):

$$5\,V \cdot \frac{8.2\,k\Omega}{8.2\,k\Omega + 43\,k\Omega} = 0.800781\,V$$

The measured board output voltage with these values was around 5.00-5.01V in otherwise steady conditions (constant current draw).

Measurements

To get more accurate results from the ripple measurement, a ground-spring (”tip-and-barrel”) was used instead of a ground-clip, as instructed in ”Measuring Output Ripple and Switching Transients in Switching Regulators”[12], and the measurement was done over an output capacitor, together with 1X probe attenuation, AC-coupling and bandwidth-limit (at 20MHz).

AN-1144 Application Note

Probing on the output capacitor using a grounded coil wire (see Figure 10) produces nearly optimal ripple detail. Trace inductance on the board is significantly reduced and switching transient amplitude is decreased. However, a low signal silhouette is still evidently superimposed on the ripple as shown in Figure 11.

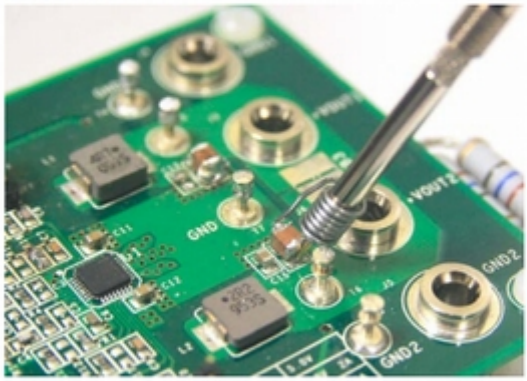

Figure 10. Tip and Barrel Method Probed on Output Capacitor Using Coil Wire

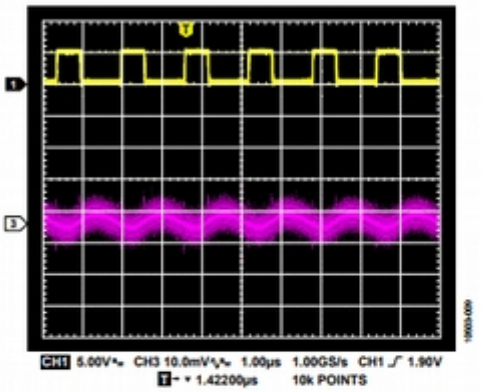

Figure 11. Switch Node and AC-Coupled Output Waveform from Figure 10

BEST METHOD

The best method uses a 50 Ω coaxial cable maintained in a 50 Ω environment and terminated by the selectable 50 Ω input impedance of the oscilloscope. Place a dc blocking capacitor between the output capacitor of the switching regulator and the oscilloscope input. The other end of the cable can be soldered directly to the output capacitor using very short flying leads (see Figure 12 and Figure 13). This preserves signal integrity, specifically when measuring very low level signals over a wide bandwidth.

Illustration 9: Tip-and-barrel -method described in the Analog Devices application note. I couldn't use the best method (50Ω coaxial), as I don't have such cables.

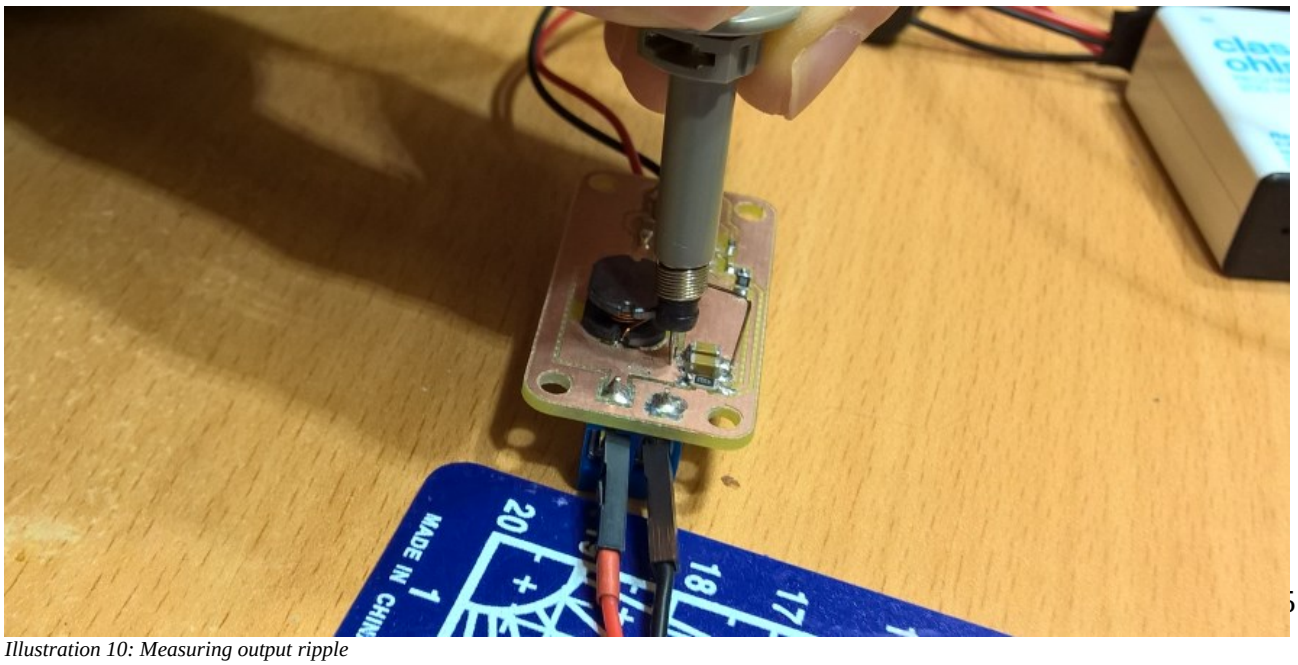

Illustration 10: Measuring output ripple

It proved to be pretty difficult to hold the probe in place while using other hand to stop the oscilloscope measument to get steadier screenshots. Even a very slight movement of the tip or the spring causes a lot of transient noise spikes in the capture or exaggerated ripple.

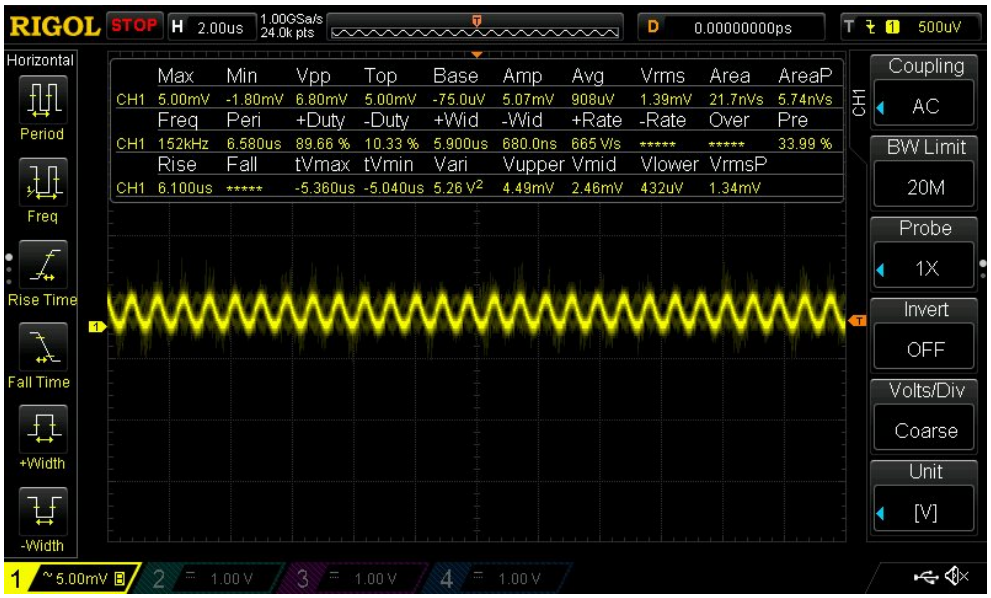

Illustration 11: 8.2µH ripple

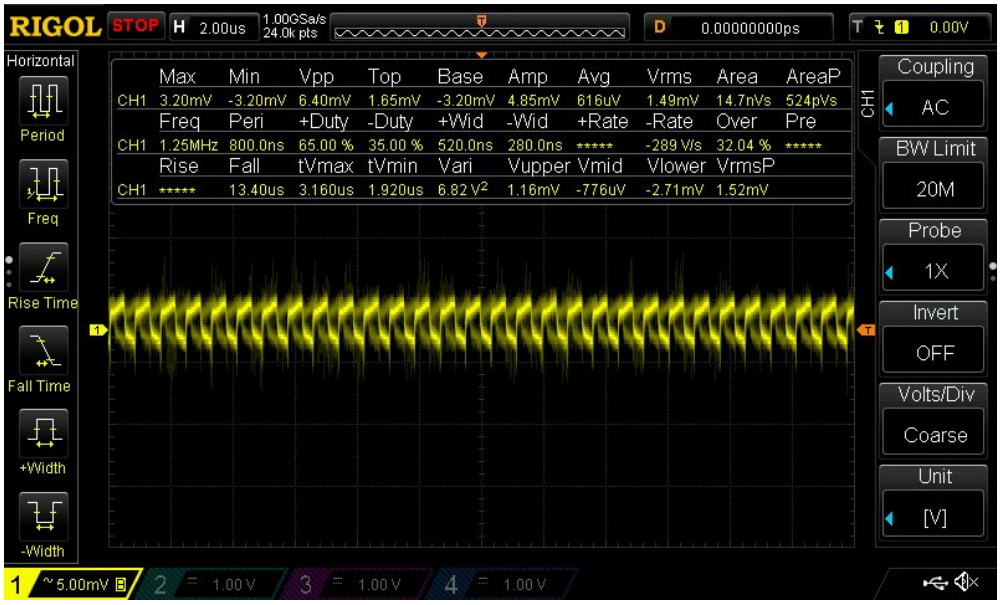

Illustration 12: 27µH ripple

From the measurements it would seem that in the 27µH -version, the wave does have a different shape versus the 8.2µH version, the spikes might be ringing going on in the circuit. The datasheet shows same value compensation capacitor but different resistor for the compensation -pin for different voltages using larger inductances, unfortunately, no equation for calculating correct values for the compensation was given, only a table for different output voltages. Also, in hindsight, the measurements should have been done with higher loads, the output current was around 20mA,

forcing the switching converter to work at very low duty cycle or even in discontinuous mode (the inductor current dropping to zero momentarily), and the battery (input) voltage may have been at least slightly different between the boards, which might affect the current ripple and thus the voltage ripple. Either way, the voltage ripple appeared not to be an issue, as it's still far below the target of 1%.

Other effects causing deviation from the calculated values might be things like measurement artifacts, higher ripple currents than the equations suggested, stray capacitances and inductances on the boards and higher ESRs in the capacitors than the datasheets suggest. The equations might also be too far simplified for switching occurring at a relatively high frequency, as most switching converters seem to work with frequencies from 50kHz to 150kHz.

Transient response

The transient response, that is the change in output voltage caused by change in load current (such as motors turning on) can become a problem if the voltage drops low enough for a long enough time to brown out the MCU, which could then either get stuck or reset. The current draw of an Arduino Nano / ATMega328P isn't that much, around 20mA, so bypass capacitors should be able to keep things running during short enough transient voltage drops in the actual switcher output.

The transient response was tested only by using a HP6632A as a load. The HP6632A is actually a programmable linear DC-power supply, not an actual electronic load. However, although not exactly designed for it, it has the ability to work as a load, up to a point. Current sinking can be handled by setting the supply output voltage to below the actual output of the switching power supply, at which point the power supply will sink current instead of sourcing it, within 250mA above the current limit set, but care should be taken not to exceed the maximum voltage or the current sinking capabilities. For currents below around 250mA, the output voltage can be adjusted to near what the switching supply is outputting to get a small enough voltage difference that produces lower current.

"*The HP 6632A is designed to be a two quadrant power supply. It has a positive output voltage, but it has the ability to source current (normal power supply behaviour) and sink current (unusual).*

The negative current limit is 0.25A higher than the positive current limit. So if you program the output current to 1A maximum and connect a power supply that is higher voltage than the HP 6632A it will sink up to 1.25A. That is it behaves like an electronic load.

The HP 6632A (and B) are unusual in the sink / source capability is more or less symmetrical. Other HP power supplies can sink current, but the current is typically limited to around 1/8th of the sourcing capability. They call this a down programmer.

Why did they do this?

To speed up test time when adjusting to a lower voltage. A shorter test time means more test through put.
It is not meant to be used as an electronic load.

You do not want the OVP circuit to trip if you are connected to a battery. The fuse will blow. " [13]

The battery mentioned in the above quote doesn't mean anything like a small 9V battery, but something that can output much higher currents, such as car battery or a lithium cell. If the OVP would trigger and short circuit (crowbar) the output, by for example accidentally setting the OVP too low, the results would be far less dangerous with a NiMH that has relatively high internal resistance, versus something like a lithium cell or car battery, which are capable of spewing out tens or a hundreds of amperes.

First the good news. When going from 300mA to 680mA, the output voltage transient is fairly small:

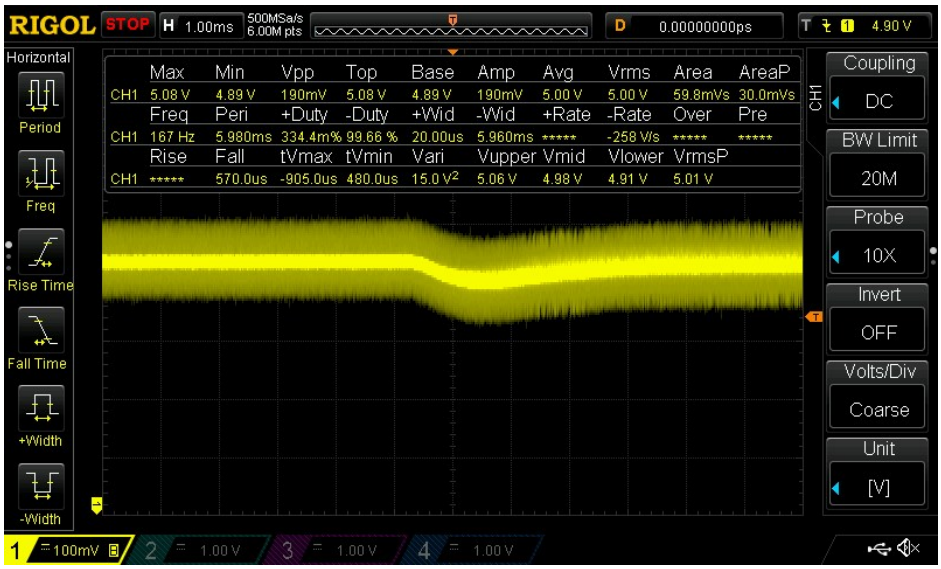

Illustration 13: 8.2µH transient response going from 300mA to 680mA

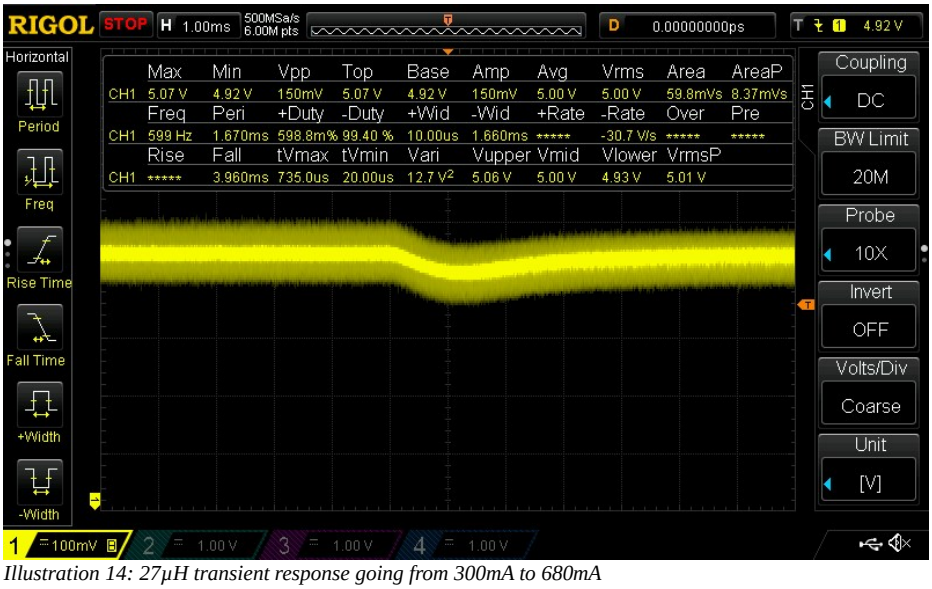

Illustration 14: 27µH transient response going from 300mA to 680mA

There doesn't seem to be that much difference in the response despite different inductors. The voltage doesn't really drop that much, and shouldn't be an issue with other parts of the circuit. 27µH doesn't seem much slower than the 8.2µH. The battery voltage dips below 7V from about 8.5V open circuit voltage with 680mA output, which indicates that the battery has at least somewhat significant internal resistance (couple of ohms), and there are difference between batteries, as some of them didn't regain as high voltage (might be charge-state related, though).

The trouble actually starts when the current draw of the load goes from a low value (about 20mA) up to 300mA. It appears that the voltage drops too much and the undervoltage protection of the switching controller chip triggers a "hiccup-mode".

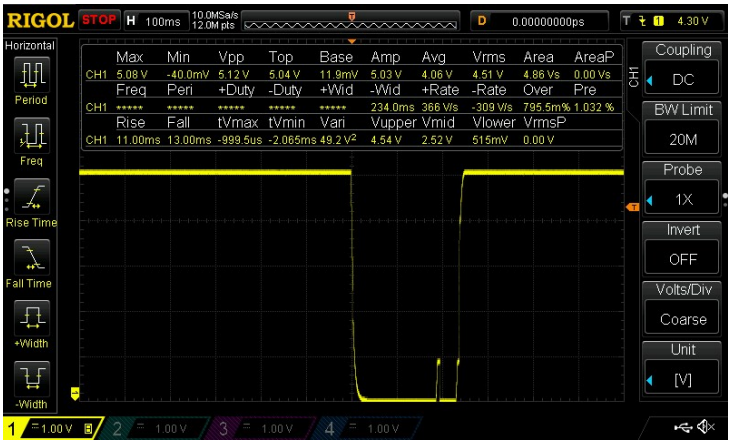

Illustration 15: 8.2µH enters undervoltage protection hiccup-mode for a good 200ms when the output current goes from about 20mA up to 300mA

Illustration 17: 27µH entering hiccup-mode

Under Voltage Protection

Hiccup Mode

For the RT7297BH, it provides Hiccup Mode Under Voltage Protection (UVP). When the VFB voltage drops below 0.4V, the UVP function will be triggered to shut down switching operation. If the UVP condition remains for a period, the RT7297BH will retry automatically. When the UVP condition is removed, the converter will resume operation. The UVP is disabled during Soft-Start period.

Hiccup Mode

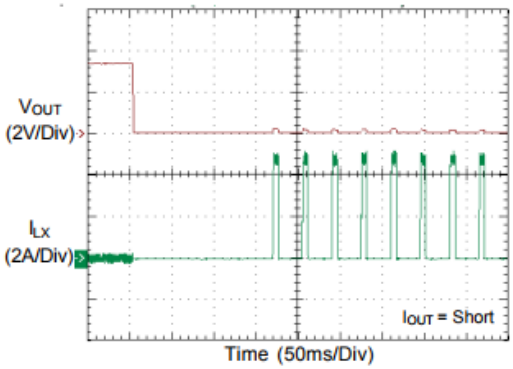

Figure 5. Hiccup Mode Under Voltage Protection

Illustration 16: Explanation of the "hiccup mode" in RT7297B datasheet

I was worried if this was caused by the internal resistance of the NiMH-batteries and if I should then switch to lithium-cells for power source, but testing with a Nano and the motors connected to a breadboard with the motor-drive chip, the MCU did not reset when the motors turn on, so more likely it's an issue combining the voltage drop caused by the internal resistance of the battery with the HP6632A current limit not triggering fast enough when turning the output off (probably not a good way to change the load, but it was easier to do while using one hand to hold the probe in

place, and it would cause a current draw of 300mA), and the output actually gets short circuited over a low impedance for a brief while there, dropping the battery voltage while completely draining the capacitors of the converter circuit and causing the chip to enter the undervoltage protection -mode.

As everything worked fine with the Nano and motor drive -test circuit, no more time was spent trying to improve the situation at this point. In the end, the actual boards used the switching circuits with 8.2µH inductors, as the 27µH-version had something that looked like ringing in the output, and otherwise the differences weren't that big. Measuring the actual efficiency of the switching circuitry was skipped due to time constraints and not having good equipment for that (I'd prefer two 6632A's, one as supply and one as load).

When designing the actual circuit for the robots, 470µF low ESR aluminum electrolytic capacitors were added near the input-side capacitors, to help out with sudden current spikes, and smaller bypass capacitors were added near the other parts.

IR-modules

The IR-modules are used to turn on the IR-leds around the perimeter of the robot, to give a signal for the other robots to measure with the IR-photodiodes on the same modules, the result of which is used to roughly measure distance and direction between the robots.

IR-LED

Couple of different IR-LEDs were tried, the final choice being LL-S150IRC-2A [14], an 1206 SMD-packaged wide-angle (140°) / 100mW IR-led from Lucky Light. The other tested ones included no-named Chinese 5mm through-hole IR-LEDs and Optosupply OSI5LA56A1A, a 5mm (or 4.8mm) through-hole wide-angle (100°) / 40mW IR-LED. The LLL-S150IRC-2A was selected based on being relatively cheap and an SMD-device (less holes needed to drill in the module-boards), and having a wide half-power angle (140° / ±70°), so it would better cover the entire surroundings of the robot, leaving very few blind spots.

IR-LED as input

For reading the IR-radiation, the first try was simply to use another LED, as an LED produces small voltage across itself when exposed to light with the same spectrum as it emits. However, the voltage produced was too small and imprecise to be used for measurement, although it could have been amplified.

IR-phototransistor

IR-phototransistors were tried next, and proved much more sensitive. Lucky Lights' LL-S150PTC-1A (120°) transistors were tried, but they were actually too sensitive, picking up too much IR-noise from the ambient environment to be really useful. Likely they'd work better as a switch.

IR-photodiode

Finally, photodiodes seemed more suitable for the measurement, although I hadn't ever used such diodes before and didn't know much anything about them. A reversed diode will allow current to pass through it when exposed to IR-light of suitable wavelength (of course, all the LEDs, phototransistors and diodes were sensitive to the same wavelength, 940nm). The final photodiode used was Vishay's VEMD10940F [15], a 150° SMD-packaged silicon PIN [16] photodiode. Vishays' BP104 (130°) was tried also, but it had unusual through-hole footprint and was more expensive.

The initial try was to simply put the diode in series with a pull-up resistor to the +5V -line. While it "sort of worked", it wasn't a very good way to go, as the small current the diode passes (a few microamps or less) meant that the resistor needed to be very large (hundreds of kilos or megaohms), and the voltage-signal was then transferred over a wire and through the analog-multiplexer to a very high input impedance amplifier, before being fed into the ADC-input. The low energy signal was very prone to noise being picked up by the wire acting as an antenna of sorts.

Transimpedance amplifier

Reading up further on the photodiode-usage, the commonly referred method was to use what is known as a transimpedance amplifier or ”current-to-voltage -converter”[17]. The small current passing through the diode is converted into a voltage with an operational amplifier (op-amp).

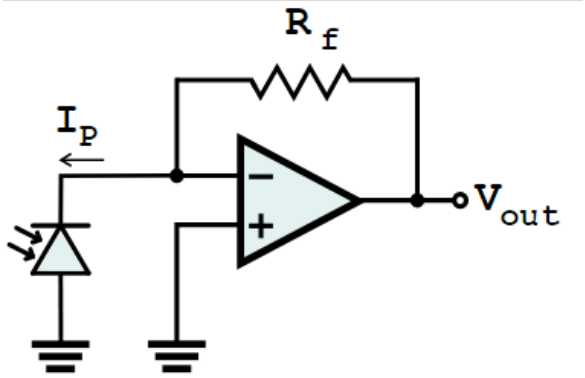

Illustration 18: Transimpedance amplifier in short circuit-mode.

Picture courtesy of Zen-in / Wikipedia, https://en.wikipedia.org/wiki/Transimpedance_amplifier#/media/File:TIA_simple.svg CC-BY-SA 3.0 license: https://creativecommons.org/licenses/by-sa/3.0/

The final circuit used was the configuration shown in the above illustration, known as ”transimpedance amplifier in short-circuit mode”, where the diode is reverse-biased against ground, and the cathode is connected to the inverting input of an op-amp, and fed current by the feedback-resistor coming from the output of the op-amp (see for example [18] [19]).

Some sources suggest that when using single input supply (no negative rail, just +5V and GND feeding the op-amp), the non-inverting input should be biased above GND, but it seems to work just fine even with the input connected directly to ground.

In an ideal situation, the output voltage follows the current in a linear fashion:

$V_{out} = -I_p \cdot R_f$, where V_{out} is the output voltage of the op-amp, $-I_p$ is the current going through the photodiode (negative, because the diode is reversed) and R_f is the feedback-resistor.

In reality, the bias-current and open-loop gain of the op-amp will have an effect, as well as the thermal noise in the feedback resistor, diode capacitance any stray-capacitance in the circuit, lowering the total gain and bandwidth.

As for the frequency response, the stray capacitance from board traces, assumed to be in the order of couple of picofarads, in parallel with the feedback resistor R_f alone limits the bandwidth to less than 10kHz (this is looked into in more detail in a latter subchapter). The selection of the op-amp was done more based on very low input bias- and offset-currents (0.5 picoamperes and 0.25pA, respectively) and rail-to-rail capability, so it wouldn’t skew the readings from the very low currents the diode draws.

Luckily, in this case, a low bandwidth won't matter, in fact it's better that the gain drops with higher frequencies, as that just filters out noise with fast edges. The actual measurement happens as "single-shot" and relatively infrequently (up to 10 times per second). If the signal takes a while to rise and the robots need to wait a few milliseconds to measure the IR-values after another one has turned on its IR-LEDs, it won't matter much, and real world -testing with prototype boards showed that the output gives steady values (not much internal noise, ambient noise sources, such as lighting, are another matter) and has a useful range for our purposes.

IR range and coverage

The wide angle of the IR-LEDs and the photodiodes causes the radiation to spread out over a large area at distances, and the power per unit-area becomes increasingly small, so a high gain is still needed. Most remote controls and such use much narrower "beam", and thus can reliably transmit signals over much higher range, but require more precise aiming. We still needed to use wider angle, as it wasn't really an option to use more IR-modules due to cost and assembly time (more modules, more shift-registers, more other components).

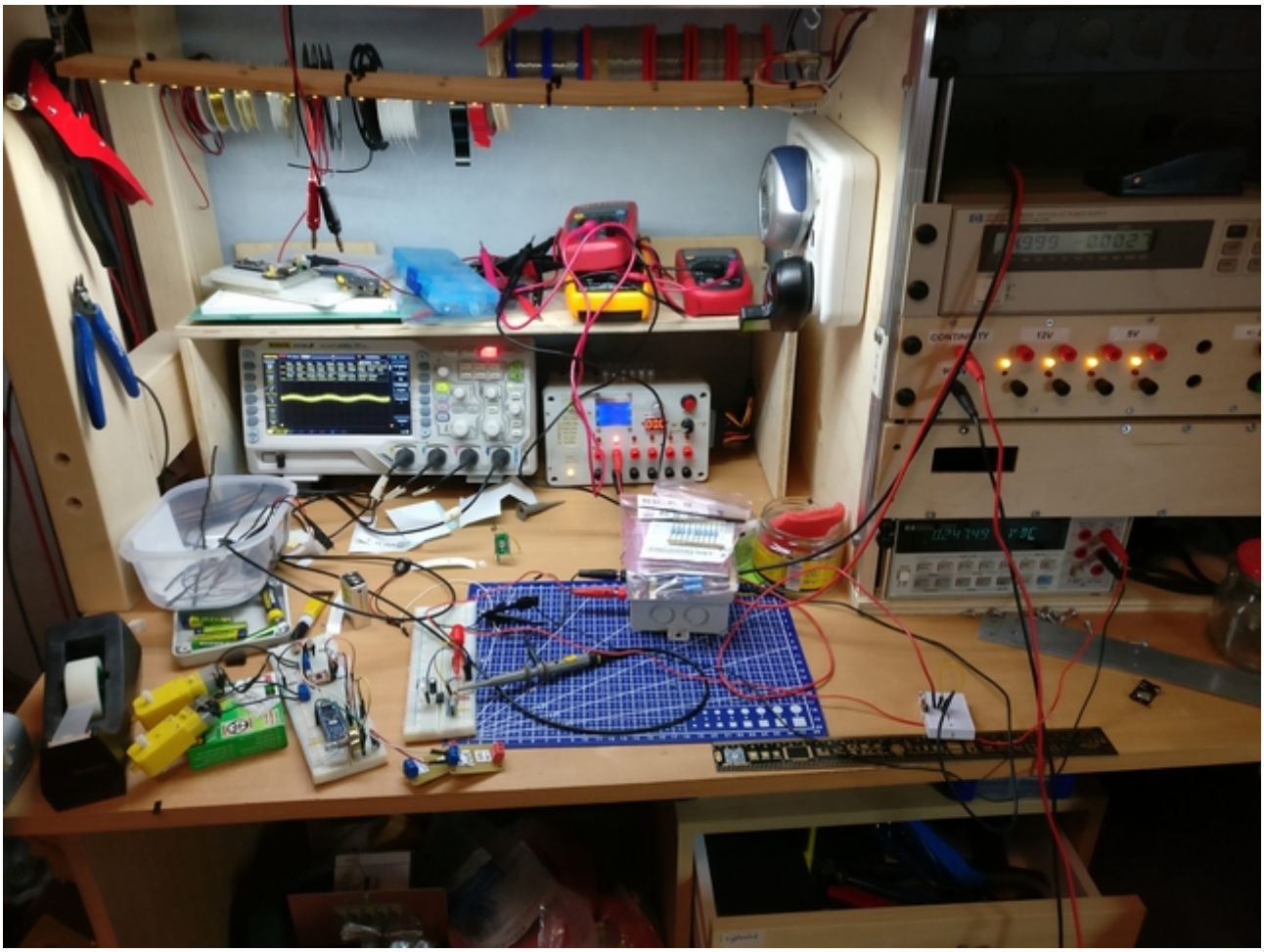

Illustration 19: Set up for measuring IR-photodiode/transimpedance amplifier output. Only LED-lighting is used, as it gives out less IR-interference than fluorescent-, incandescent- or halogen-lighting. The breadboard-test circuit for testing that the MCU doesn't brown out when motors turn on can be seen on the left side.

The needed feedback-resistor was tested by starting from a few hundred kilo-ohms, and going up, until noticing that a very large resistor is needed to be able to get high enough voltage from the op-amp up to about half a meter, but I didn't have larger resistors than 10MΩ available, and didn't want to start putting more in series, thus the selection of 10 megaohms.

The following table lists output voltage of the op-amp for 1MΩ and 10MΩ feedback-resistors at different distances, although the op-amp at this point was a DIP-packaged MCP602[20]. Measured with a HP34401A -tabletop multimeter (calibration unknown, but measures 0.01% resistors and 0.05% voltage references within tolerance, so at least not far off):

Distance	**1MΩ R_f**	**10MΩ R_f**
2,5cm	1130mV	4980mV (limited by rail)
5cm	286mV	3140mV
7,5cm	125mV	1390
10cm	71.9mV	850mV
15cm	37.2mV	397mV
20cm	23.8mV	242mV
25cm	18.4mV	180mV
30cm	15.4mV	147mV
35cm	13.9mV	128mV
40cm	13.3mV	114mV
45cm	12.6mV	105mV
”Infinite” (IR-light off)	9.4mV	65mV

As can be seen from the results, a little more gain wouldn't hurt. The ATMega328P has 10-bit ADC-inputs and, in this case, 5V reference, giving a resolution of 5V / 1024 = 0,00488...V or about 5mV per LSB (least-significant bit). There's likely noise of a few LSBs in the measurement, so differentiating between 40 and 45cm might be difficult with the 10MΩ feedback, but it hardly matters as we're not interested in absolute distances. With the 1MΩ feedback, the voltage differences are too low to measure with ADC, unless using a much lower reference voltage than 5V.

Even with higher gain, the useful range is still limited to the point where the measurement starts to get close to ambient noise, higher gain would mean getting higher readings at the farther ranges, which could help differentiating the actual signal from noise, but the output would hit the rail sooner when getting closer and the range of values would be smaller, as also the ambient noise

would be amplified more. Although the amplifier output should be at least roughly linear in regards to the current going through the photodiode, and the photodiode should conduct linearly versus the received IR-radiation, the final output is not linear, as the IR-radiation from the LEDs is spread over a larger area the farther away the source is and diminishes non-linearly.

Another source of non-linearity comes from the overlapping light-cones of the IR-emitters when all LEDs are turned on at the same time. Since the lenses are as wide as 140 degrees, there's bound to be areas between the modules where the intensity is higher due to two LEDs radiating in the same envelope. The effect likely isn't that strong further away though, and allows larger range as more power output is available. There are smallish blind-spots with no IR-cover around the robot, but they shouldn't become a problem, as they're so near to the robot.

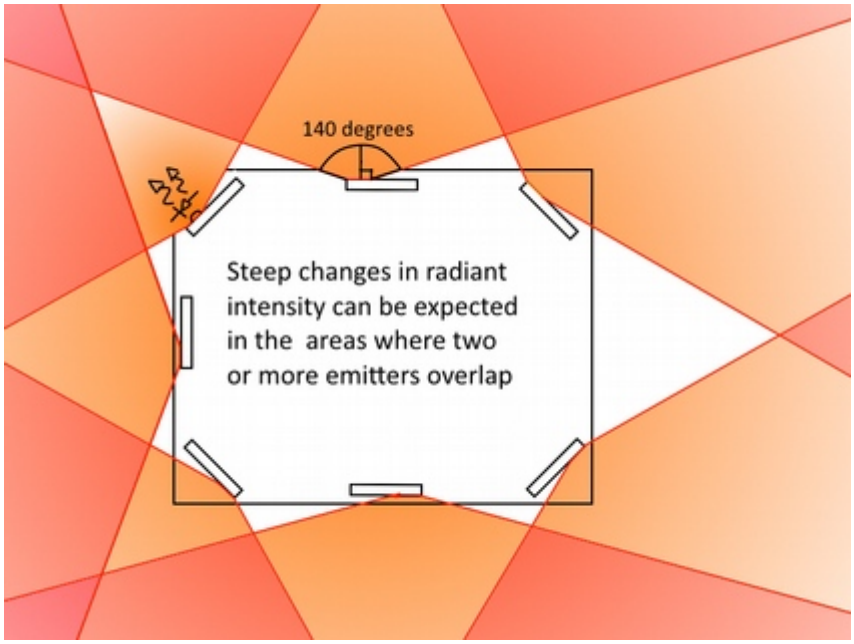

Illustration 20: Overlapping areas of the IR-LED radiation causes "hot zones", where the intensity of the emitted IR is higher.

Finally, the LED and photodiode radiant intensity/sensitivity -patterns aren't uniform, meaning that the emission isn't as strong for the LED or as sensitive for the photodiode across the entire angle of output/input. As the LED output diminishes towards the edges, it may actually even smooth out the sharp transitions between two adjacent LEDs overlapping their radiation.

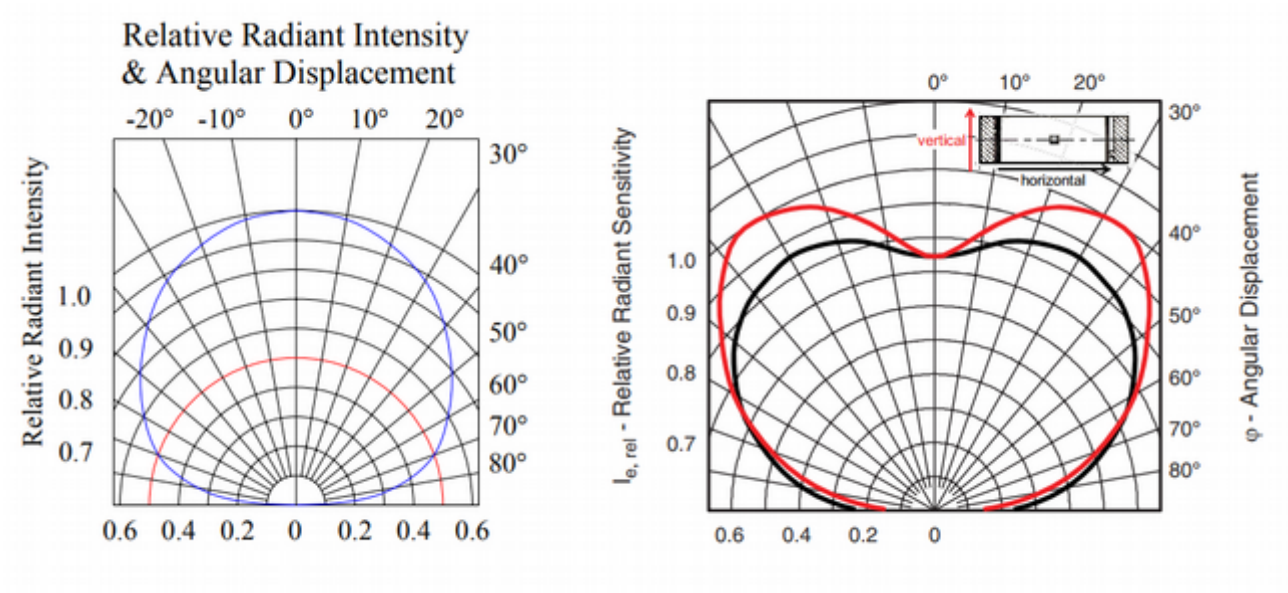

Illustration 21: Relative values of radiant intensity for the LEDs on the left, relative radiant sensitivity for the photodiode on the right

With all the sources of non-linearity, at best, a rough guess at distance can be made based on this, but the robots are not dependent on knowing a ”real” distance. As long as they can measure sort of a ”fuzzy” or relative distance and direction based on known directions of the modules on the board, it is enough.

While the photodiodes aren’t as sensitive as the phototransistors for ambient noise, they still do pickup stray IR from the surroundings. My study has only LED -lighting (strips on desk-shelves, soldering station desk and LED-bulbs in the ceiling), and the readings were less than 10 (with 10-bit ADC, so the value can be between 0 and 1023), typically something like 0-5 with the 10MΩ feedback resistors. However, if the robots are taken near incandescent, fluorescent or especially halogen lighting, they start picking up the IR from the light sources. Halogen bulbs seem to emit enough IR in suitable wavelenght that they completely drown out the actual signals, even several meters away, so the robots are pretty much unusable in such environments.

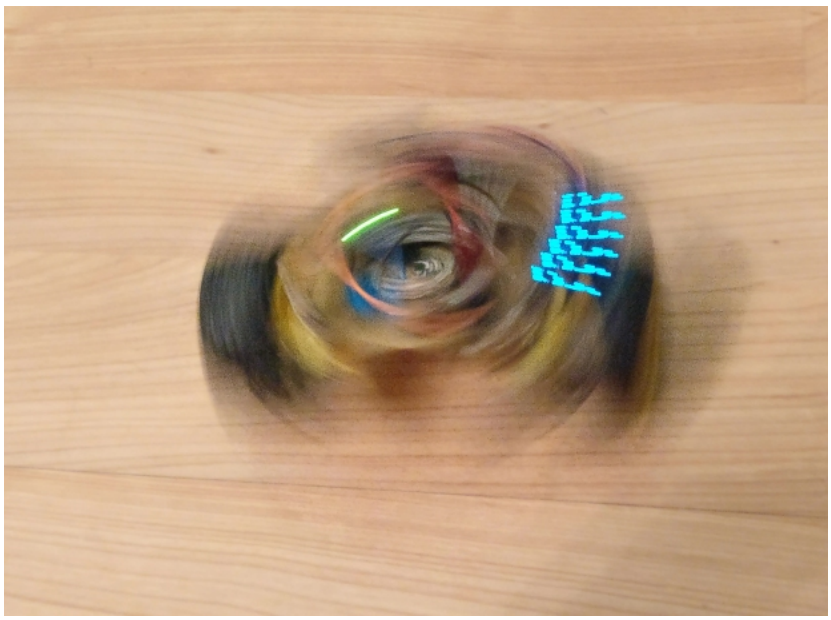

Illustration 22: Prototype model going ”crazy” and spinning fast as it picks up strong false signals from 3 x 42W halogen bulbs several meters away.

Illustration 23: Having only LED-lighting in the study, including the ceiling lights, the IR-photodiodes didn't pick up but minimal noise.

To cope with noise, up to a point, a simple averaging was also used in software, to get a sort of a ”baseline” for the ambient noise, and only reacting to significant enough outliers, that could be assumed to be the actual wanted signal. This method was not perfect, but makes the robots less ”twitchy”, ie. not as likely to react to stray signals from the environment. Still, with enough ambient IR-noise, it becomes impossible to pick the actual robot IR-emitters from the surrounding noise, or at least it drops the usable range, as the wanted IR-signal must be stronger than the noise.

IR Module schematic and layout

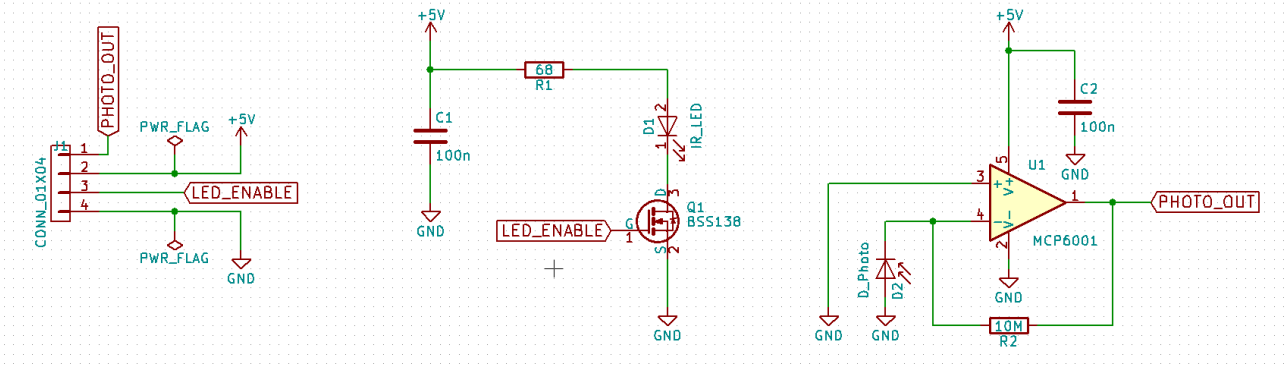

Illustration 24: Final IR-module schematic

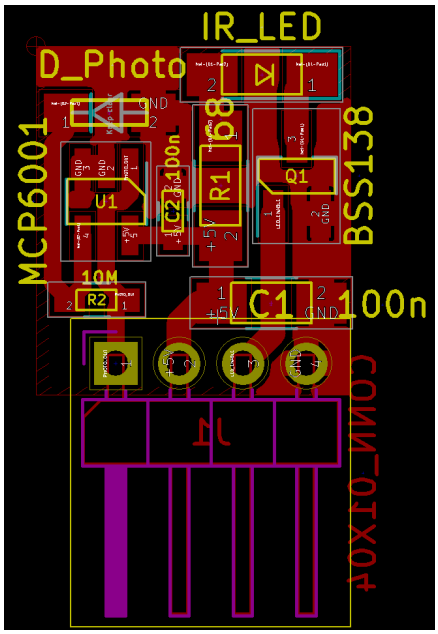

Illustration 25: IR-module layout

The above schematic is the final version of the IR-modules. It's relatively simple, consisting of a MOSFET-controlled IR-LED with a forward resistor and the IR-photodiode with the transimpedance amplifier explained in the prior chapter. Again, the footprint of the photodiode was a bit unusual, so I had to design it separately in KiCAD, although normal 1206 -footprint could have worked also.

Unlike the schematic states, the op-amp used is MIC7300[21] and not MCP6001,but the picked models had the exact same pinout, so they're interchangeable. Each robot has seven modules going around it at 45 degree angles, pointing to each direction except straight backwards. The backwards facing module was left off due to routing issues in the single-layer mainboard of the robot.

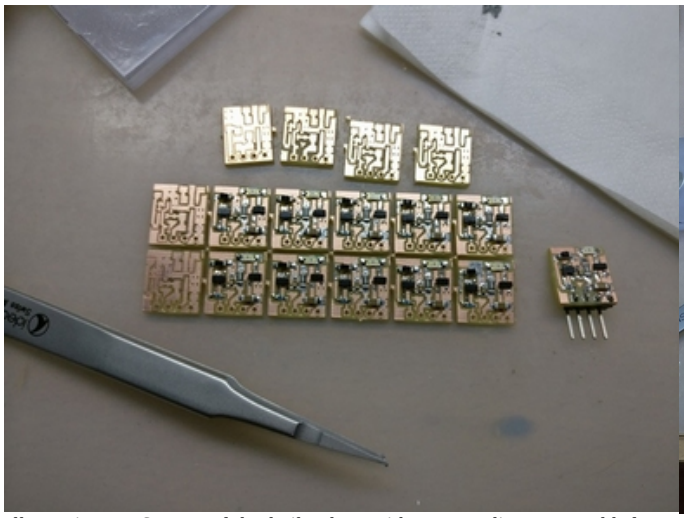

Illustration 26: Some modules built, along with some earlier unassembled prototype-board above

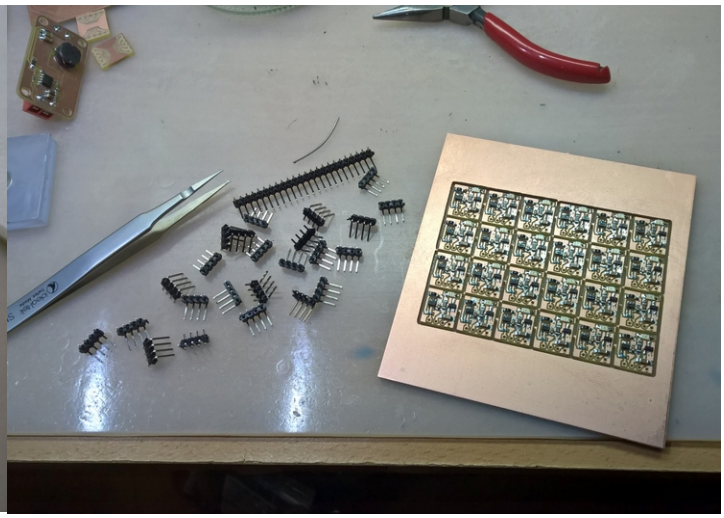

Illustration 27: Last modules being assembled much later. For 5 robots, 35 modules were needed, but the total amount made was over 40 pieces, as things needed to be tested out first.

The modules were made separate to the actual boards of the robots to make it easy to try them out, have them at 90 degree angle in comparison to the level board and to make it possible to reuse the robot boards as a more general sensor-platforms later on, if wanted.

The small modules were relatively fast to mill and assemble, care just needed to be taken to place the LEDs or photodiodes the right way around. Although the op-amps used a small SOT-23-5 -case with 0.95mm pin-pitch, soldering the chip was surprisingly easy.

Power dissipation

The IR-LEDs had a maximum power dissipation up to 100mW (continuous), and it was considered at one point whether they should be pulsed with higher power to get more range, but the idea was abandoned. The turn-on/off is handled straightforwardly through the MOSFET Q1, a BSS138[22], which is a logic-level N-channel enhancement mode MOSFET in an SMD-package (SOT-23). Gate resistor was omitted to save on space and assembly time, and the device itself has low gate charge (around 1.5nC at 5V), so the current spikes will be very short-lived. The turn-on voltage is provided directly from an output of the shift-registers, which are capable of continuous currents of only +-35mA sourcing/sinking, but so far none of them have failed. It would seem that the short-lived current spike isn't high enough to damage the device, especially as the MOSFETs are turned on and off relatively rarely and not being driven by PWM or such.

The size of the LED forward resistor R1 was based on measured voltage drop of the IR-LEDs. We wanted as much power as possible from the LEDs, so that they would be visible to the receivers as far as possible. The IR-LEDs have forward voltage of around 1.275V at 50mA, which is also listed as the absolute maximum current. Power dissipation-wise, this should still be below the 100mW limit. With 5V input and over a 68Ω resistor, the current becomes

$$I = \frac{U_{\text{in}} - U_{led}}{R} = \frac{5\,V - 1.275\,V}{68\,\Omega} = 54.8\,mA$$

The power dissipation of the LED is about 70mW at this point, whereas the resistor has to dissipate about 204mW. The used resistor is 1206-cased and has maximum P_D of 250mW. The IR-LED used in testing the ranges and measurements was continuously used at these currents, and has not failed, so it seems going over the maximum current of the LED isn't certain to destroy the device, at least as long as the maximum power dissipation is kept in check, although usually crossing the absolute maximum ratings of the manufacturers isn't that wise. In hindsight, 75Ω might have been a better choice, as it would limit the current to pretty much 50mA, with LED power at 63.3mW.

The peak forward current of the LED is given as 1A at 10% duty cycle and 100µs pulse width, but no values in-between are given, so it's entirely possible that the device gets damaged over time when the maximum current is exceeded continuously. Also, as the device heats up during use, the forward voltage of the LED itself drops, leading to more current flowing as the voltage over the resistor rises. The trouble with identifying possibly broken IR-LEDs is that naked eye cannot see the IR-radiation, so inspection would have to be done with an IR-receiver or trying to measure the IR-LED voltage and/or current to see any abnormal values indicating damaged or destroyed device.

To ease the dissipation on the resistor, 3.3V voltage could have been used with a smaller resistor, as the board regulates 3.3V with a linear regulator from the 5V output of the switching converter for the radio module, but this would have led to a smaller output voltage range for the op-amp, as it's powered from the same rail.

Layout effect in transimpedance amplifier bandwidth

The layout of the module had one thing that wasn't really considered during the design, but actually turned out (likely) to be beneficial. Later on when thinking if I should add a capacitor in parallel with the feedback resistor, I thought I'd check if I could figure out how much capacitance there is in the traces of the board itself, and if it was even necessary to add any separate filter capacitor.

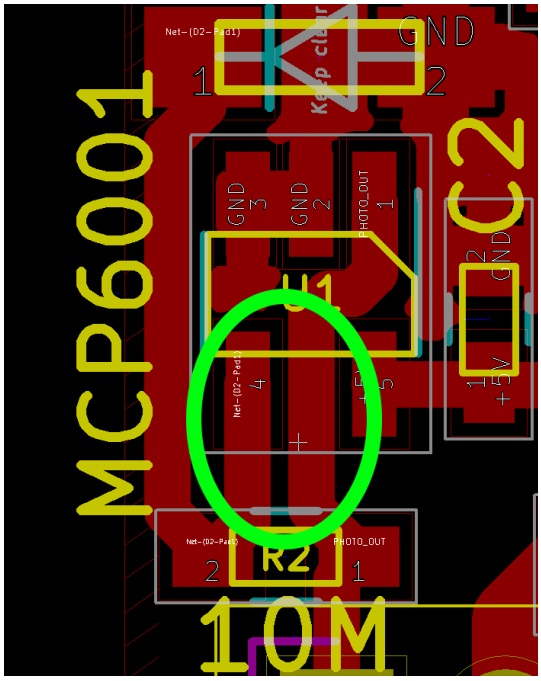

Illustration 28: Stray capacitance in the op-amp feedback-loop

The traces circled with the green ellipse run in parallel, the left-side one between the inverting input of the op-amp, the 10MΩ feedback resistor and the cathode of the photodiode. The right-side one is the output of the op-amp. The traces running in parallel form a small capacitor, the value of which was estimated using the equation found from Electrical Engineering Stackexchange[23]:

$$\frac{C_s}{l} = 0.12 \cdot \frac{t}{w} + 0.09 \cdot (1 + d) \cdot \log_{10}\left(l + \frac{2w}{g} + \frac{w^2}{g^2}\right) \quad ,$$

where C_s / l is the stray capacitance across length in picofarads per centimeter (pf/cm), t is the thickness of the trace (in µm), d is the dielectric constant (around 4.2 for FR4, 1 for air), g is the gap size (in mm), and l is the length of the parallel traces (in mm).

Measuring the values from the layout-drawing, I got the following:

w = 0.7mm

t = 35µm

d = 1 (as the material between the traces is just air)

g = 0.15mm (single pass with 30-degree / 0.1mm V-bit at 0.095mm depth)

l = 2.8mm

Inserting these into the equation gives out 6.27546...pF/cm, or about 1.75pF of capacitance for the entire parallel length of the traces, rounded to 2pF take errors and omissions into account. What this means is that there's a (roughly) 2pF capacitor in parallel with the feedback resistor, and thus it forms a low-pass filter. The filter cutoff frequency f_c is around

$$f_c = \frac{1}{2\pi \cdot R_f \cdot C_p} \quad ,$$

where R_f is the feedback resistance and C_p is the parallel stray capacitance from the traces. Around 2pF isn't much capacitance, but coupled with the high feedback resistance (10MΩ), the cutoff frequency becomes about 8kHz. If the capacitance would be only half of estimation (1pF), the cutoff would still happen around 16kHz. Like mentioned in an earlier chapter, this isn't actually a problem, as the photodiodes are meant to be measuring fairly steady signals (ideally 0Hz frequency / DC), so the low-pass filter mostly just attenuates extra noise, albeit it might have some effect in the signal rise time too, and there are other stray capacitances in the board also, which may affect things.

I can't claim to be sure that the above is correct, and the manufacturing tolerances in the milled boards are likely to cause differences from one board to another. Nevertheless, there is no need for a separate filtering capacitor, as the smallest capacitors are usually in the single picofarad -range, and likely the board carries at least closer to one picofarad or more on its own, already pushing the bandwidth down to single or tens of kilohertz.

Radio module

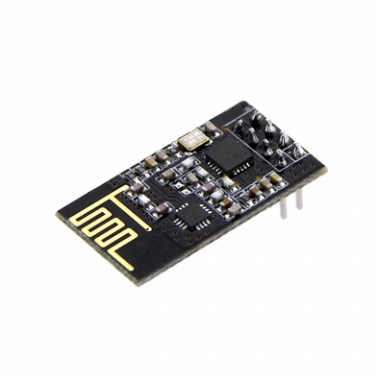

Illustration 29: 2.4GHz radio-module

For communication, each robot has a factory-built radio-module based on nRF24L01+ (also known as nRF24L01P) from Nordic Semiconductor[24]:

> *The Nordic nRF24L01+ is a highly integrated, ultra low power (ULP) 2Mbps RF transceiver IC for the 2.4GHz ISM (Industrial, Scientific and Medical) band. With peak RX/TX currents lower than 14mA, a sub µA power down mode, advanced power management, and a 1.9 to 3.6V supply range, the nRF24L01+ provides a true ULP solution enabling months to years of battery life from coin cell or AA/AAA batteries. The Enhanced ShockBurst™ hardware protocol accelerator offloads time critical protocol functions from the application microcontroller enabling the implementation of advanced and robust wireless connectivity with low cost 3rd-party microcontrollers.*
>
> *The Nordic nRF24L01+ integrates a complete 2.4GHz RF transceiver, RF synthesizer, and baseband logic including the Enhanced ShockBurst™ hardware protocol accelerator supporting a high-speed SPI interface for the application controller. No external loop filter, resonators, or VCO varactor diodes are required, only a low cost ±60ppm crystal, matching circuitry, and antenna.*

Fake chips?

To be exact, our radio-modules with the chips, needed supporting components and antennas were bought from China, so the actual chips might not be originals. I've never before had any problems with Chinese Arduino-clones or modules like TFT- or OLED-displays, gyro/accelerometers and such, but RF-stuff is of course much more sensitive.

Hackaday had a short article[25] warning of some issues with the cheap clones:

> *The nRF24L01+ is a highly integrated, ultra low power (ULP) 2Mbps RF transceiver IC for the 2.4GHz ISM (Industrial, Scientific and Medical) band. Popular, widely used and inexpensive – and the counterfeit foundries are drawn to it like honey bees to nectar. But to replicate and make it cheaper than the original, one needs to cut several corners. In this case, the fakes use 350nm technology, compared to 250nm in the original and have a larger die size too.*
>
> ***These differences mean the fakes likely have higher power usage and lower sensitivities, even though they are functionally identical.*** *The foundry could have marked these devices as Si24R1, which is compatible with the nRF24L01 and no one would have been wiser. But the lure of higher profits was obviously too tempting.*

Like people in the comments of the article, Philipp had a number of problems and weirdness with getting the modules to work and messages to transmit reliably even over very short distances, although some part of the problems might have been also caused or added to by poor connections, interference and 3.3V output from the Arduino Nanos, as most of the development was made on breadboards with jumper wires and powered from a computer USB-port, which might add noise in the circuit. Still, a lot of it could also come down to poor quality clones. While the modules with cloned chips should be compatible with the ones having original chips, there are some that have slightly different behavior of registers, and work differently with the settings of the library versus the originals [26] :

> *Let me start by saying that I do have a dog in this fight as I work for Nordic Semiconductor.*
>
> *The re-marked nRF24L01P (+) clones are not 100% register compatible. The issue with the counterfeit devices is that when they enabled "Dynamic Payload Length" (EN_DPL) in the "FEATURE" register, one bit get's activated in the on-air payload (the NO_ACK bit) This bit should be active high (according to the Nordic datasheet), but it's actually implemented the other way around. When EN_DPL is activated, the NO_ACK bit get reversed in the real nRF-devices. They did such a good job of cloning they cloned the datasheet error into the device!!!*
>
> *If someone is using the fake parts on both ends it will work to some degree. In a mixed environment where you may have both real L01P and the fake ones you will have issues. But don't look to be fully "compatible" with the real nRF24L01P firmware wise. And on a side note, As the nRF24LE1 and nRF24LU1P both use the nRF24L01P radio this makes the counterfeit parts incompatible with those as well.*

I had a little bit better luck in reliability doing some simple tests that came with the library and the actual final boards over short ranges while fiddling with the library settings, but that was already late into the project. It likely helps that the final boards have their own 3.3V regulator on the robot board and no jumper wires to pick up interference. In hindsight, it would have probably been better to include a filter in the 3.3V line before the modules, as they're said to be sensitive to noise and fluctuations in the power input. Such could be later on created as separate boards that sit between the radio-modules and the pinheader to which they go in the robot boards, if need be.

The actual designed protocol and behavior of the robots is explained by Philipp in the Software-section, although we didn't have enough time to implement it.

Moving the robot

Motors

We considered continuous servos, steppers, ”plain” ungeared DC and geared DC motors as options when looking for motors. I had some motors on my shelves, and the final pick was a cheap geared 1-phase DC-motor, as they were simple to control, and had a 6V maximum voltage, so they could be run at almost maximum speed just with the 5V output from the switching power supply.

Illustration 30: Geared DC ”toy” motor, 1:48 gearing, 6V max voltage

The reasons for choosing them was that the robots do not need to move very fast and gearing gives more torque, so the weight of the robot shouldn’t become an issue. I didn’t have 5 pairs (10 total) of the motors, so I ordered more, luckily they arrived quite fast, and in the mean time, I could already start testing and development with the ones I had.

Motor drive

Initially, I thought to use some common quad half-bridge/dual H-bridge IC that many hobbyist projects use for small-powered motors, like L293[27] or L298[28].

The problem with those chips is that they loose a lot of power internally. I learned this the hard way, after making a self-balancing 2-wheeled robot earlier, and wondering why it stopped driving the motors after a minute or two. Initially I thought the problem was with my code, crashing or getting stuck in an infinite loop somewhere, until I burned my finger on the large heatsink of the L298 -module. Infrared thermometer revealed that the heatsink was around 95...100° C when the motors stopped. Likely the driver went into a thermal shutdown at that point. Investigating the issue further, I found this[29] :

> *The L298 “steals” voltage from the power supply. More formally, there is a voltage drop between the input voltage and the L298’s motor outputs. The worst-case drop in voltage is 3.2V at 1A of motor phase current and 4.9V at 2A of motor phase current.*
>
> *This voltage drop leads to power losses in the L298 according to the equation:*
>
> $$P_{loss} = V_{drop} \cdot I_{motor}$$
>
> *According to the datasheet, typical voltage drop (the center column on the right side of the datasheet) in each L298 channel is 2V+1.7V=3.7V (worst-case is 4.9V). If the motor current is 2A, the typical power lost in the L298 would be 7.4W (worst-case is 9.8W). Not only is this power that is wasted because it never makes it to your motor, it is a power loss that heats up the L298.*

Hardly ideal, and not wanting to run into similar issues in this project, I looked for alternatives. I briefly considered building the half bridges from discrete power MOSFETs and gate driver ICs or even charge pumps (for the high-side MOSFETs), but that would take a lot of time and board space.

Luckily, someone in the PICAXE -forums had already done the work[30], and found some alternatives. The final selection was LV8548MC[31], a 2-channel low saturation voltage forward/reverse driver IC. It handles up to 20V (16V recommended max), can drive two separate 1-phase DC-motors backwards and forwards (or a single stepper, or one 1-phase motor with double the power by paralleling both channels), handle 1A per channel and comes in a SOIC-10 -package. Unfortunately, the package uses 1mm pin-pitch instead of 1.27mm (0.05”), so once again I had to make a custom footprint for Kicad to mill out breakouts for breadboards and for the final boards.

Internally, the half-bridges are made with MOSFETs instead of BJTs (Bipolar Junction Transistor), and the $R_{DS(ON)}$ -resistance of both high- and low-side MOSFETs combined is 1Ω at 1A typically. Using both channels at full current would exceed the maximum power dissipation of 1W, but I trusted that it wouldn’t blow up immediately, and the motors aren’t being driven constantly at full power. Later on, testing also revealed that the motors didn’t need as much current as was estimated, the short-circuit current of a single motor was around 350mA per motor at 5V, if the motor is blocked from turning. That meant that the power dissipation would stay below the maximum, even if trying to drive both motors continuosly while they cannot turn. Once the motor is turning, the current is even lower, as the back-EMF (ElectroMotive Force) the turning motor generates makes the voltage difference smaller, and less current flows.

Driving the motors with the chip is straightforward, as it only has 2 input pins for each motor, although the PWM-duty cycle needed to be reversed for the other direction, due to the way the internal chip logic worked. 100% duty-cycle in reverse means that the motor is stopped, as the motor is braking when both IN1 and IN2 (or IN3 and IN4 for the other motor) are high.

Operation explanation

1. DCM output control logic

Input				Output				Remarks	
IN1	IN2	IN3	IN4	OUT1	OUT2	OUT3	OUT4		
L	L	L	L	OFF	OFF	OFF	OFF	Stand-by	
L	L			OFF	OFF			1CH	Stand-by
H	L			H	L				Forward
L	H			L	H				Reverse
H	H			L	L				Brake
		L	L			OFF	OFF	2CH	Stand-by
		H	L			H	L		Forward
		L	H			L	H		Reverse
		H	H			L	L		Brake

Illustration 31: Control logic of the LV8548MC

The datasheet also states that either pin can be driven by PWM (Pulse Width Modulation) to control the motor speed, of course taking into account the aforementioned need to ”reverse” the duty cycle when the motor is being driven backwards.

The chips were tested separately by milling a board with the 1mm -pitched SOIC-10 footprint and connecting it with an Arduino Nano and the motors on a breadboard. Powering with the testing switching converter-boards from the 9V batteries with 100µF capacitor near the motor driver IC, it could also be tested that the voltage didn’t drop enough to turn off the MCU.

The chip doesn't really need much passives around it, mostly just a capacitor at the power input to handle spikes. Initially, I planned to add a "pi-filter" -kind of setup at the power-input with a ferrite bead to filter out any noise and spikes from the motor and prevent them from entering the main 5V -line, but after testing it with the ferrite replaced simply by a 0-ohm resistor, no ill effects were noticed, and the ferrite was left out also in the final boards.

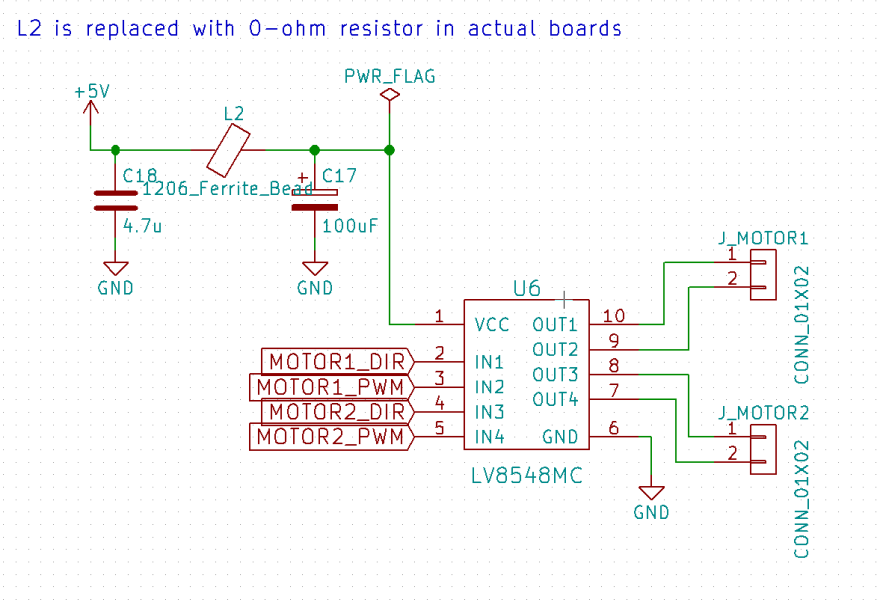

Illustration 32: Motor-drive -part of the robot circuit

Likely the initial values picked for the filter would have been way off anyway, I'd at least suspect that C18 should be (much) larger. Simulating a ferrite bead isn't very easy, as it doesn't really act like a "pure" inductor and typically has the highest reactance in the tens or hundreds of megahertz-range, although LTSpice does have some ferrite bead-models. If the noise would become an issue, likely a "real" filter with an inductor and suitably valued capacitors should have been designed.

The 100µF capacitor near the chip is a low ESR polymer -type (Elite UPE1C101MNN6305[32], 24mΩ max ESR at 100-300kHz), as I figured it could better handle possible ripple current feeding back through the MOSFET-body diodes from the motors. Due to time limits, not that much consideration of all the details was put into this after knowing that the noise and transients wouldn't seem to be an issue.

Shift registers and analog demultiplexing

As there weren't enough ADC-inputs and digital outputs to control and read everything, a multiplexing/demultiplexing -scheme was used to control more outputs and read more inputs with fewer pins.

74HC595D [33] is an 8-bit serial-in, parallel-out -shift register with output latches. Two or more registers can be cascaded together to form a shift-register with more output bits. Two chips were chained to gain 16-bit output from only 3 output pins taken to the shift clock-input, data-input and latch clock-input -pins of the shift register. Same clocks go in both, while the serial data output of the first register is fed to the second to cascade them.

Normally more pins would be used to control the output enable and master reset of the chips, but they were hardwired on the board due to routing issues, as initial boards were single-layered. This requires the software to write the shift registers at start up, as the outputs are always active and are in an unknown state during powering on, so 16 bits must be clocked into the chips and latched to outputs. Also due to routing issues, one output pin of each chip was left unused.

The shift-registers control the IR-leds in the IR-modules (7 bits), the RGB status-led (3 bits), and the analog demultiplexer (4 bits). The RGB-led is powered directly through the shift-register chip, but the current is below the maximum output of a single pin, and the total current output is below the maximum of the chip, even when all three colors (red, green, blue) of the LED are turned on. To get more even brightness for all the colors, the forward voltages for each color of the LED were measured with constant current, and resistors were sized accordingly.

74HC4051[34] is an 8-channel analog multiplexer / demultiplexer, in a single-pole/octal-throw (1-to-8) -configuration. It is used to demultiplex all the IR-module transimpedance amplifier outputs to a single ADC-input pin, so the robots can measure the IR-readings one-by-one from all the modules around the board. The inputs of the chip are controlled through the shift-registers, not requiring any more output-pins from the MCU, and only a single ADC-input pin.

The chip does add some resistance in the path that actually changes slightly with the passed through and supply voltage (roughly 80-100Ω for input signal within 0-5V signal and supply voltage of 5V), but it didn't matter that much here, as the output of the amplifier in the IR-module has low impedance and the other end was initially planned to be connected to the input of an op-amp (very high input impedance, 10^{13}Ω for the MCP602). As the circuit for the IR-modules was modified, the op-amp on the mainboard was bypassed in the final design, and the output of the 74HC4051 goes directly to the MCU ADC-pin. The ADC-inputs have lower impedance, than the op-amp but still high enough for the added resistance from the 74HC4051D to be negligible [35]:

> *The ADC is optimized for analog signals with an output impedance of approximately 10 kΩ or less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long time the source needs to charge the S/H capacitor, with can vary widely. The user is recommended to only use low impedance sources with slowly varying signals, since this minimizes the required charge transfer to the S/H capacitor.*

Pinout of the MCU for the robot

Even with two cascaded shift-registers, we didn't have enough "plain" digital output pins for all the functions, but luckily ATMega328's have the ability to use the ADC (Analog-to-digital -converter) -pins also as GPIO (General purpose I/O, digital pins), so some of those were used as digital outputs. The exception are the ADC-pins 6 and 7 (A6 and A7 in Arduino definitions), only available in the 32-pin TQFP-packaged ATMega328P (which Arduino Nanos use), that can be only used as ADC-pins.

Of course, if really needed, even more shift-registers could have been added, but that again would have caused more trouble with board space and traces.

Pin (Arduino / ATMega)	**Description**
D2 / PORTD.2	IRQ for NRF24L01
D3 / PORTD.3	Direction for motor 1
D4 / PORTD.4	Direction for motor 2
D5 / PORTD.5	PWM for motor 1
D6 / PORTD.6	PWM for motor 2
D7 / PORTD.7	CE for NRF24L01
D8 / PORTB.0	CSN for NRF24L01
D9 / PORTB.1	**FREE**, PWM-capable
D10, PORTB.2	**FREE?**, PWM-capable, SPI Slave Select Might need to be left alone, in case the radio-library / ATMega hardware SPI uses the SS-pin, even though it isn't connected.
D11 / PORTB.3	MOSI for NRF24L01
D12 / PORTB.4	MISO for NRF24L01
D13 / PORTB.5	SCK for NRF24L01
A0 / PORTC.0	ADC for IR-value reading through 74HC4051
A1 / PORTC.1	DS (Data) for 74HC595 serial-to-parallel (digital input)
A2 / PORTC.2	SH_CP (Shift clock pulse) for 74HC595 serial-to-parallel (digital input)
A3 / PORTC.3	ST_CP (Store clock pulse) for 74HC595 serial-to-parallel (digital input)
A4 / PORTC.4	**RESERVED** (I^2C, SDA)

A5 / PORTC.5	**RESERVED** (I^2C, SCL)
A6 / ADC6 (only on 32-pin TQFP)	**FREE**, ADC, *cannot* be used as GPIO
A7 / ADC7 (only on 32-pin TQFP)	LDR reading

Robot mainboard and mechanical design & testing

The board schematic and layout were designed in KiCAD. Initially, the idea was to mill out all the robot boards with a small tabletop-CNC, but due to time restraints, the final boards were ordered from a Chinese PCB fabrication service, EasyEDA.

The first layout was done with single-layer boards in mind, and relied on jumper-wires on a few places to go over other traces and to connect ground planes. The IR-module measurements were transferred over wires to a connector next to the analog demultiplexer-chip, and then fed to an op-amp for further amplification. Due to the high-impedance input of the op-amp, the wires could pick up noise easily.

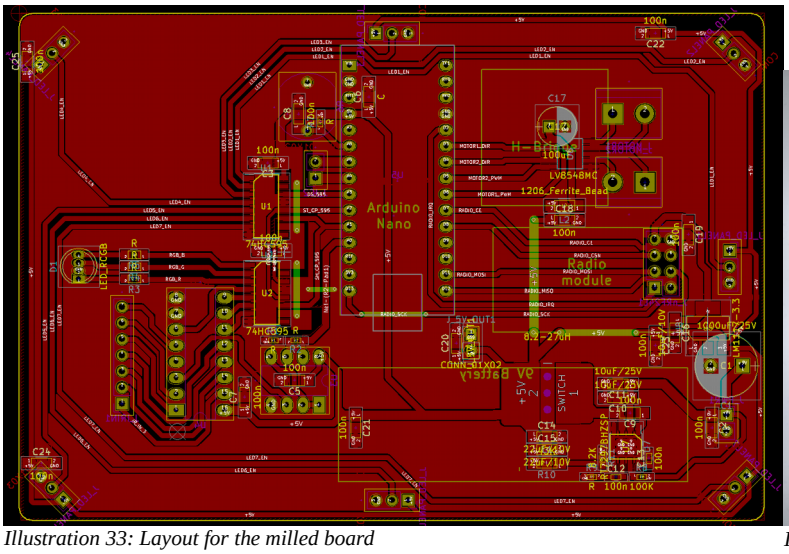

Illustration 33: Layout for the milled board

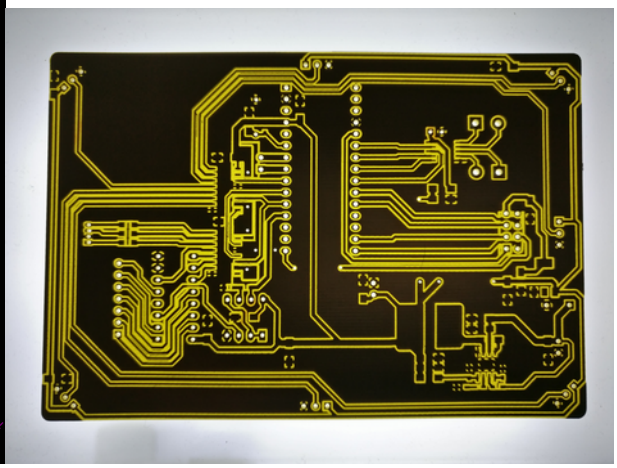

Illustration 34: Milled board being inspected on a light-table

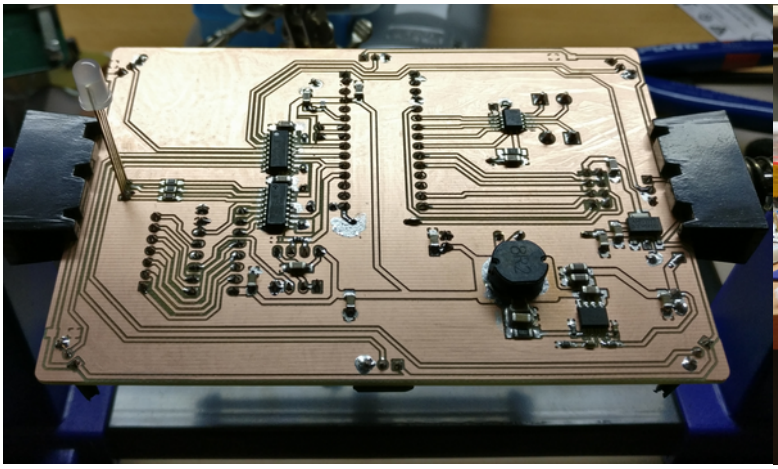

Illustration 35: First robot board assembled.

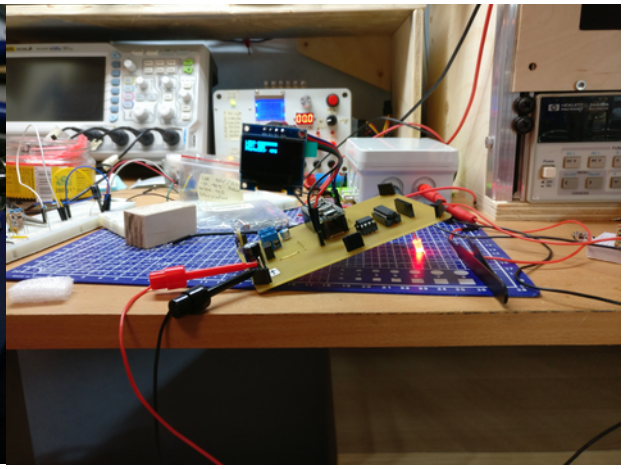

Illustration 36: First board basic software tests with a display controlled over I²C.

The first prototype had the motors hot-glued directly to the board, partially covering the switching converter IC, but no heat problems seemed to occur at any point. The RGB-LED that the robot leaned on in the “non-motorized” end broke off eventually, and was replaced with a nylon spacer,

again hot glued to the board. All the IR-modules were connected to the board and a testing software displaying the measured ADC-values on an OLED-display was written to see that everything works as expected. Simple motor control functions were written so that the robot could turn to face the IR-emitter and drive to it.

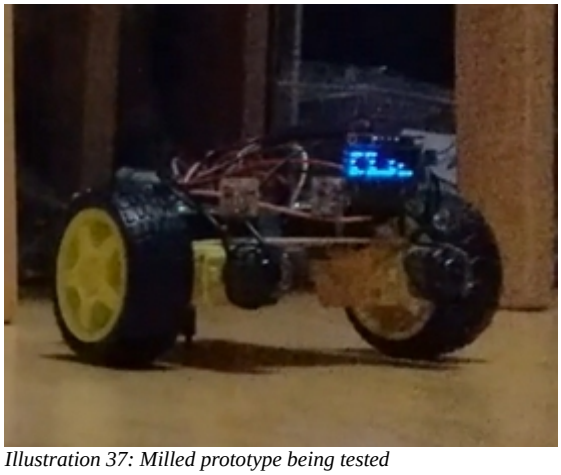

Illustration 37: Milled prototype being tested

The schematic was changed on a couple of occasions since the first board was made. Since the later IR-modules had their own amplifier on the modules themselves, the robot-board didn't need to amplify the inputs anymore. When modifying the schematic and the board, the op-amp footprint was still left in place in the board, but a 0-ohm resistor was added to make it possible to bypass the op-amp. Also more capacitors were added, specifically each IR-module had its own 100nF bypass on the board also, although this might have been unnecessary.

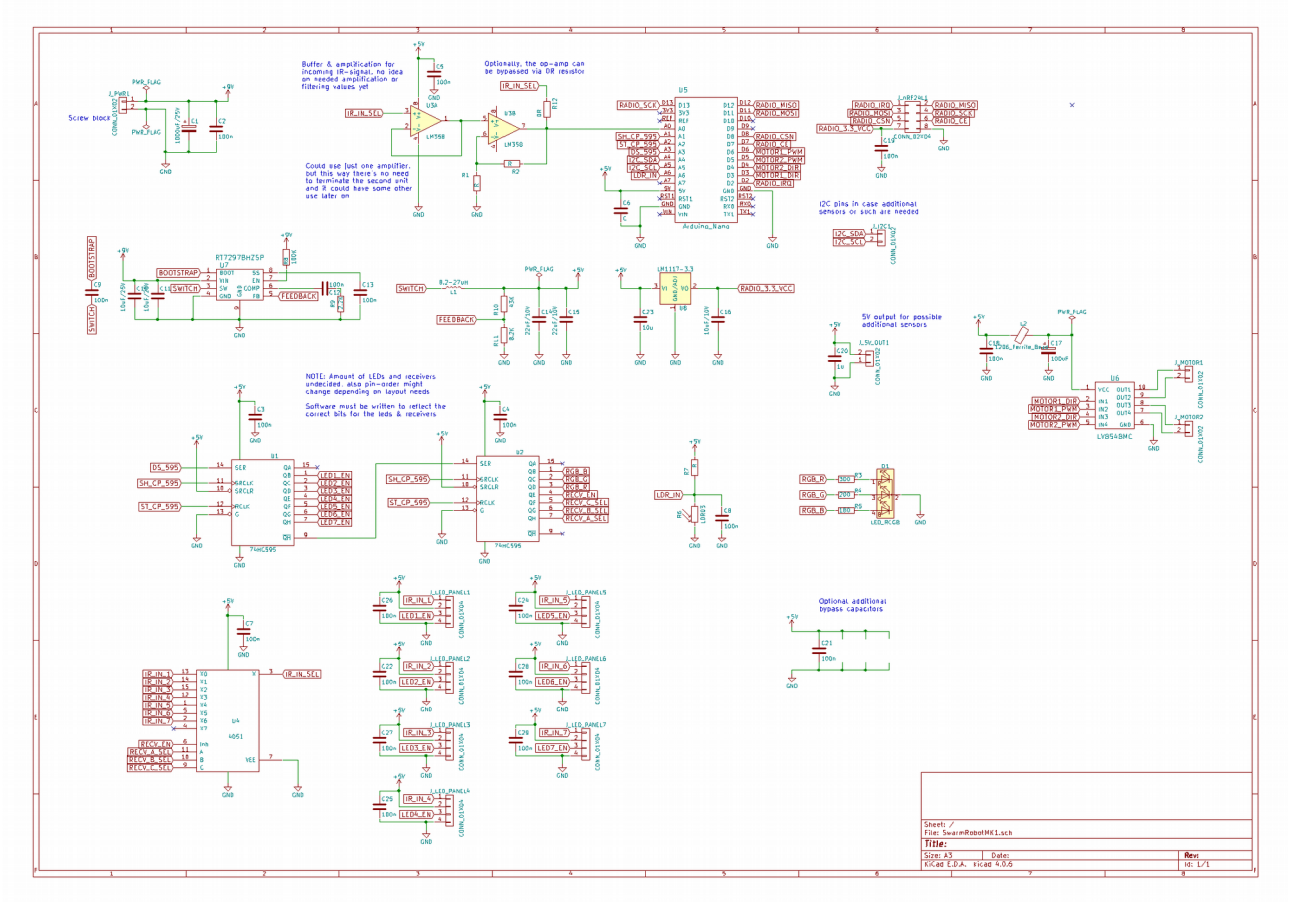

Illustration 38: Later-stage schematic with op-amp bypass-resistor and more capacitors added. The image compression used in the document makes it hard to read, the final schematic used in the fabricated boards should be added as a separate PDF along with the report.

Since the hardware was tested to work with the milled board, risk of design failures was small, so once the decision to use fabrication service instead of milling all the boards was done, the layout

was modified only slightly, so that no jumpers were needed anymore, and all the traces could be ran on the board, with no need to add wires for the modules. Some vias were added to connect the ground-planes on both sides as well as to help with the heat dissipation of the switching converter by placing multiple vias under the exposed pad of the chip. Mounting holes were added to more easily attach the nylon-spacers and motors, although later on the motors were connected by 3D-printed brackets.

As the order amount was for 10 boards (for the same price as 5), additional connectors for 5V and 3.3V outputs were addes, as well as connectors for unused pins. This makes it possible to use the same boards for some other project, of course within the limits of what the board was originally designed to do. Redesigning the entire layout more carefully, the size could have been made much smaller, but the order needed to be made quickly, as it would take over a week for the boards to arrive (9 days in total, with DHL courier service), so a large part of the layout was exactly the same as before.

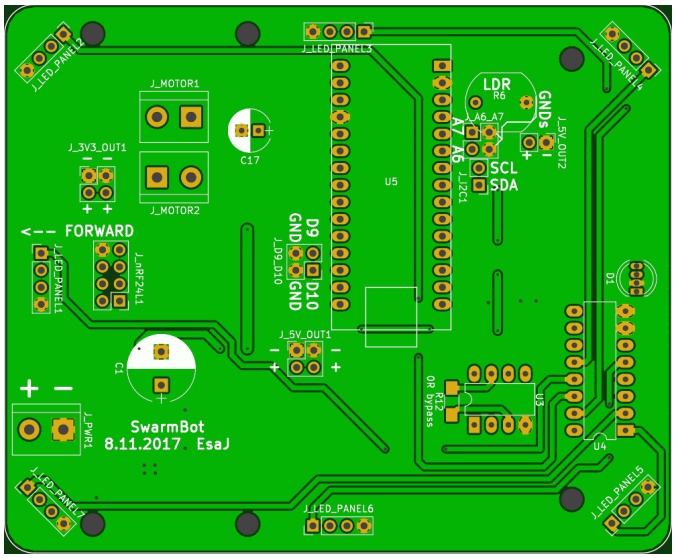

Illustration 39: Top-side of the board

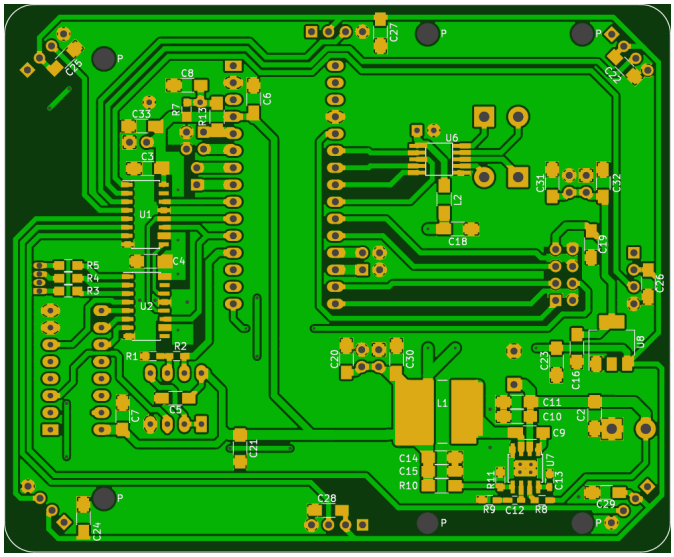

Illustration 40: Bottom (SMD)-side of the board

The boards arrived on Friday 17th of November. Over the weekend, I built the rest of the IR-modules and one of the robots, but ran into problems on attaching the motors. I quickly designed a simple motor-bracket for holding the motors, and 3d-printed out 10 pieces of them. The size was off by about 0.25mm, so the motors still needed to be hot-glued to the brackets so that they wouldn't come off in collisions. The rest of the robots were completed over the following week, two weeks before the deadline of 8th of December.

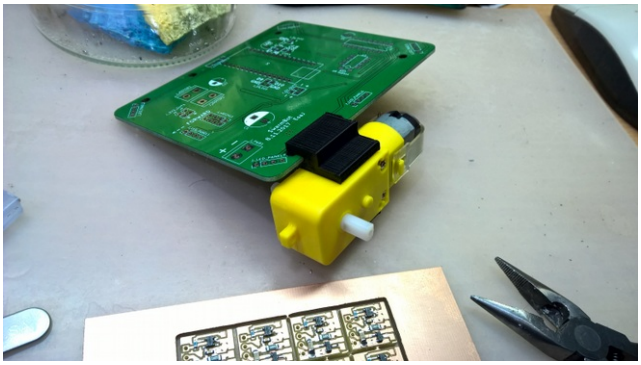

Illustration 41: 3D-printed motor bracket

Illustration 42: One robot board completed (without motors), rest under work

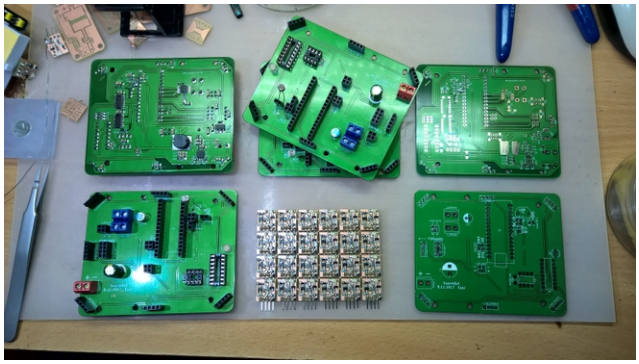

Illustration 43: All boards and rest of the IR-modules completed

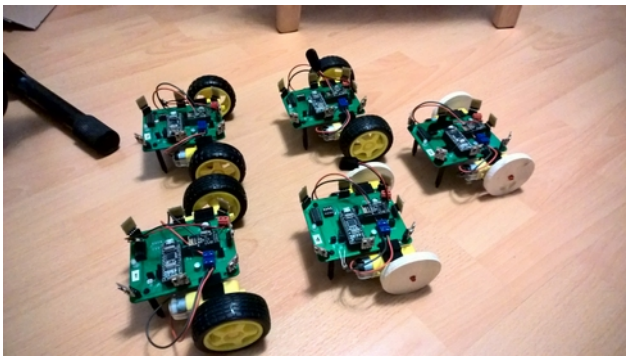

Illustration 44: Robots complete. Didn't have enough of those plastic wheels, so some had to be made out of plywood with a holesaw.

Once the robots were complete, the testing software was modified so that if the LDR-reading of a robot was large enough (or actually small enough, as the voltage in the ADC-input lowers the less resistance the LDR has), it would stop its own measurements and turn on all the IR-LEDs. The other robots around it would then drive to that robot (assuming they were near enough to “see” the IR-signal). Testing that all the robots can both turn the IR-LEDs and see the other ones, I then gave Phillip three of the robots to finish working on the radio-code. Unfortunately, at this point he had only a week to work on those and the final report about the software, as the exchange students left for a Lapland-trip on the Saturday 2nd of December, and wouldn’t return until after the deadline. He still managed to get the radio-communication working, but didn’t have enough time to implement the state-machine and finer details to make the robots behave as planned, as he needed to use some of the time to write the software-portion for this document.

Software

General function

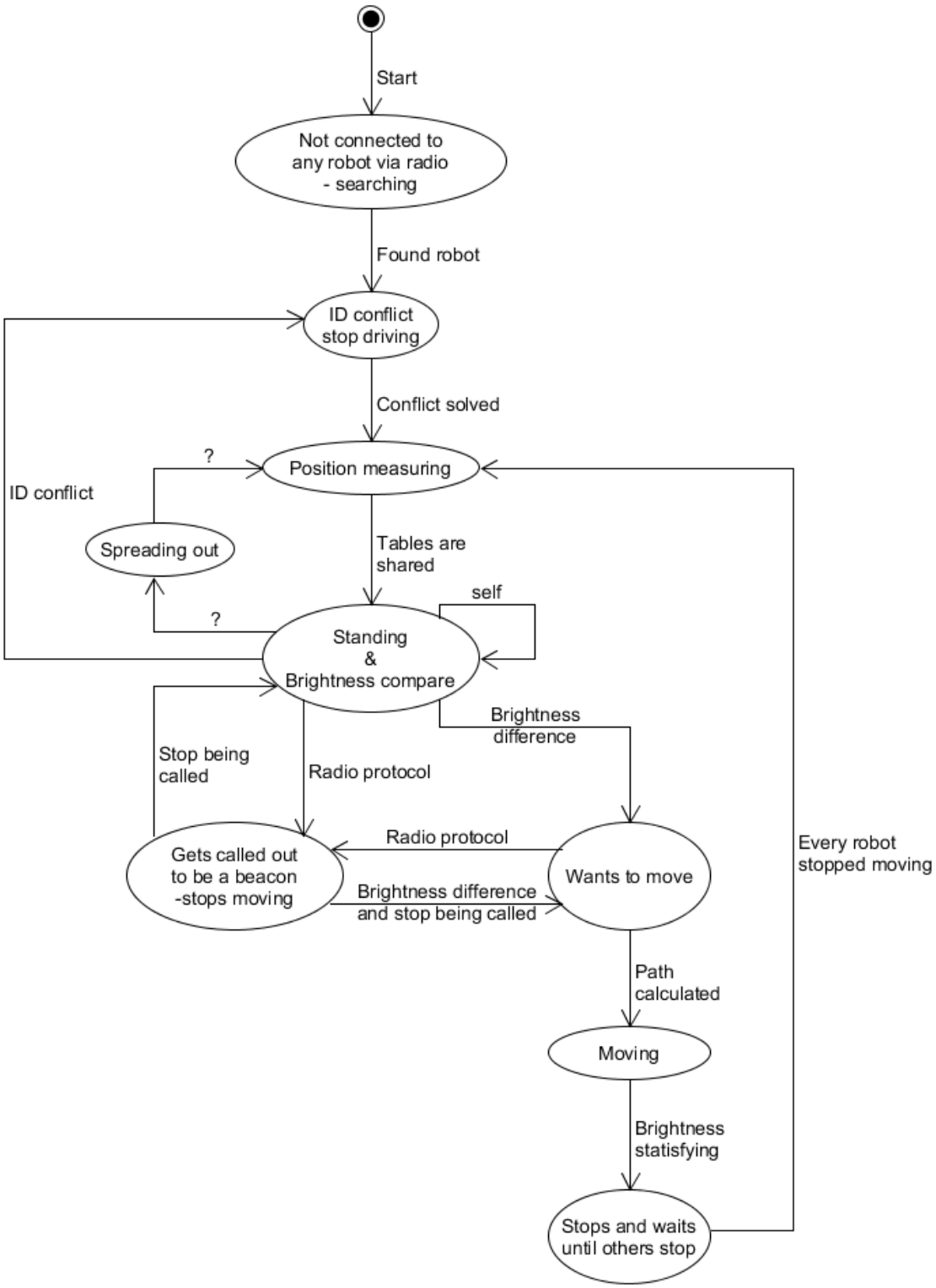

Illustration 45: Robot state-diagram

The current status of the software is that all the basic functions are implemented. The robots can drive, control the LEDs, measure the brightness and Infrared and they can communicate with the radio. The only thing that is left is to implement the way everything is controlled. That is basically what this chapter is about.

Not connected to any robot via radio:

Upon start up the robot first tries to find other robots to communicate with. It sends in random intervals a message into the world. If there is no group around but only a single robot they react to each other and form a group. Groups are a lot more active with the radio than a single robot. So it's highly likely for the robot to spot a group before there is even a need to send the first message.

ID conflict:

After a robot joined or left the group the robots reorder the ID. Each robot has an own ID number whereas the first robot has the ID number 1. The next one has an ID of 2 and then 3 until every robot has an ID.

There is no plan jet on how to implement that feature.

Position measuring:

After the group composition changed or after some of the robots moved they have to evaluate which robots are in visible range and where they are relative to the each other. After every robot has finished measuring they share the data and store it into a list on each robot (see chapter List of robots).

Standing & Brightness compare:

After the position is known the robots go into some sort of waiting modus. The only thing they do is share some data with each other. In the end the robots want to drive to a bright spot. The brightness of each robot, which is constantly shared, is constantly compared to the own brightness. If the own brightness is low compared to the others they try to go to that one. If they are in close proximity they can just drive to it directly. That is what the current program does. But if the group is spread out more widely they can't see each other and therefor they can't drive to the intended place. That is why they store the data on which robot sees which one. With that information they can calculate if there is a path between the goal and the current position. The following picture illustrates a possible scenario.

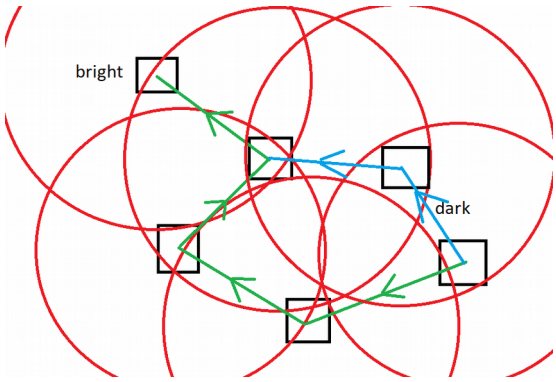

Illustration 46: IR-ranges and pathfinding

In that picture the robot in the dark place doesn't see the goal directly but with the knowledge about the IR measurement of the others there is a way to calculate it.

Wants to move:

With the brightness difference the robot decides to move. But before it can start driving the path has to be calculated first. For that the pathfinding algorithm A* is used. With it there can be calculated if there is a path and which one is the shortest. A* basically tries every possibility until it find the goal. It can each step of the path can have its weight. The more weight it has the longer the path is in the end. In this case the robots can measure a rough distance and a rough angle. The longer the distance and the sharper the angle is the higher the weight.

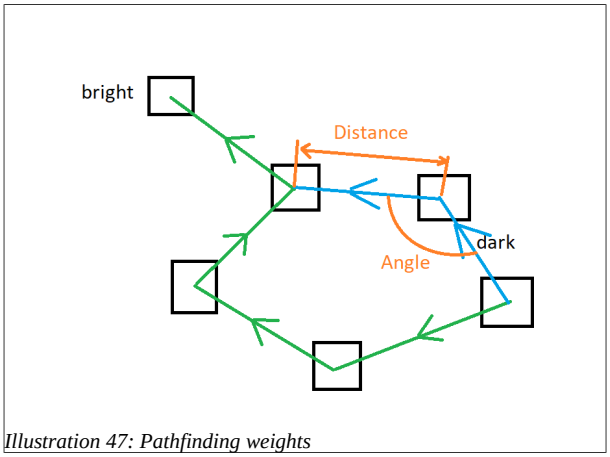

Illustration 47: Pathfinding weights

Gets called out to be a beacon:

As long as the robots stay still every position is known. As soon as one moves the list of each robot is faulty and the same path might be impossible. If robot A want's to go to robot C via robot A both robot B and C have to wait until robot A has reached its goal. In the protocol (see chapter Protocol for the radio) robot A mentions B and C. As soon as the end goal C is visible B is no longer on the list in the protocol. Therefor B can move as well if it wants to.

Moving:

Inside the list there is a bit which sates on which robot has moved since the last measurement.

Stops and waits until others stop:

If the robot reached its goal and other robots are still moving it has to wait. As soon as every robot stopped moving they measure the distance between each other and everything starts all over again.

Spreading out:

Because the robots only move closer to each other there is a need for something to drive them apart. But they shouldn't move too far because then they couldn't see each other again. If some of the other robots get cut off from the group this step could also help them get them back. But there is no plan jet on how to solve that.

List of robots

```
struct List{
  char idOwn;
  char brightness;
  char idCurrent;
  char groupSize;
  char singleBits;//bit0: ownBeaconMode, bit1: ownMoved, bit2: rob1Moved,
                  //bit3: rob2Moved, bit4: rob3Moved, bit5: rob4Moved
  char visibleRobots[3][4];
  char otherRobots[14][4];
};
extern struct List listRobots;
```

This list contains the most important knowledge the robot. The data in there is needed for the protocol and the calcullation of the path with the A* algorithm.

Visible robot:

In total there can be 4 robots visible with the infrared photodiodes. With the array visibleRobots[3] [4] there is space to store the ID, the distance (abs) and the direction (ang) of all of them.

On IR visible robot with lowest ID			**On IR visible robot with second lowest ID**			**On IR visible robot with third lowest ID**			**On IR visible robot with forth lowest ID**		
ID	abs	ang	ID	abs	ang	ID	abs	Ang	ID	abs	Ang

Other robot:

Basically the same information as you can find in visibleRobots is useful to know about the other robots.

ID of robot	**Brightness**	**On IR visible robot with lowest ID**			**On IR visible robot with second lowest ID**			**On IR visible robot with third lowest ID**			**On IR visible robot with forth lowest ID**		
ID	Brightness	ID	abs	ang	ID	abs	ang	ID	abs	ang	ID	abs	ang
ID	Brightness	ID	abs	ang	ID	abs	ang	ID	abs	ang	ID	abs	ang
ID	Brightness	ID	abs	ang	ID	abs	ang	ID	abs	ang	ID	abs	ang
ID	Brightness	ID	abs	ang	ID	abs	ang	ID	abs	ang	ID	abs	ang

Protocol for the radio

The protocol helps the robots communicate between each other. It is build up in Starting info, Message and End info. The Starting info of every message are these four header bytes.

ID_{own}	Group size	Message Size	Task

The ID_{own} and groupsize is there to identify the robot and help solving ID conflict. The message size and the task is there to tell the receivers what the message contains. Whenever the robot has sent all data which it wanted to send it ends the message with the bytes

Brightness	ID_{next}

ID_{next} has normally the value ID_{own}+1. If ID_{own} has the same value as group size then ID_{next} is 1. Of course you can turn of a robot while the others are still running. If that is the case this robot won't respond to the previous robot and the Robot sends an ID conflict.

Not connected:

If the robot is not connected the message contains only the starting information. The robot always switches between a longer period of random length of reading and sending a short message. As soon as it receives a message from another robot it goes into the ID conflict state.

Starting Info

Task: 0001'0000 (16)

ID conflict:

There is nothing planned jet on how to solve that Task.

Task: 0010'0000 (32)

Measuring

Here the goal is for every robot to know where which robot is. So every Robot lights up its IR LEDs one at a time. That way the surrounding robots know its position and the power consumption is low enough that the battery can support it.

When it is the robot turn, it first sends just the starting info.

Starting Info

Task: 0010'1000 (40)

After that it lights up all the LEDs one after the other. When it has finished with that it sends the Starting info again and its brightness and the ID of the next robot.

Starting Info	Ending Info

Task: 0010’1001 (41)

This step of the flow chart gets executed by every robot of the same group. When all the robots finished this task they make another round to share their measurement with the following message.

Starting Info	ID_1	abs_1	ang_1	ID_2	abs_2	ang_2	…	Ending Info

Task: 0010’1010 (42)

ID_n, abs_n and ang_n: transmit the data of the visible robots. The content of the data can be found in chapter Data of the own robot.

Standing still:

In this step the robots basically only share their brightness and show that they are still active. That is done with the following message.

Starting Info	Ending Info

Task: 0011’0000 (48)

Wants to move:

As soon as there is another robot at a much brighter spot this robot wants to move to the brighter one. The A* Algorithm calculates the shortest possible path to the other robot.

Starting Info	Ending Info

Task: 0100’0000 (64)

Moving:

Shortly after the path to a brighter spot is calculated the robot starts to move. It also tells the other robots where it wants to move to and which robot it has to follow to get to that destination. The robot always moves to the left most visible robot on the message. All the robots right to that on the message get removed from that list. All the robots in the List aren’t allowed to move because then the position after moving isn’t known anymore.

Starting Info	$ID_{destination}$	$ID_{second\ last}$	$ID_{third\ last}$	…	Ending Info

Task: 0100’0001 (65)

Beacon Mode:

When in beacon mode the robot is not moving. This step has priority over the step robot wants to move and standing still. As long as a robot that wants to move or is moving has this robot on its message, this step is active for this robot. The other robots must know where this one is and therefor this robot lights up its IR LEDs one at a time. As soon as it isn't on the message anymore it stops the beacon mode.

Before the LEDs light up this message gets send via the radio.

Starting Info

Task: 0100'0010 (66)

After that this message gets send.

Starting Info	Ending Info

Task: 0100'0011 (67)

Stopped moving but waiting:

When a robot moves the position has to be measured again afterwards. When several robots are moving the position can only be measured again when all robots stopped moving again. The robots which have finished moving and are waiting for the others are in this step. When all robots have finished they change into the step position measuring.

Starting Info	Ending Info

Task: 0100'0100 (68)

Spreading out:

There is nothing planned jet on how to solve that Task.

0101'0000 (80)

About the code

Although the avr-gcc handles the code as C++, which was necessary due to the radio-library being C++ -code, the software was written in more of a C -like fashion. For the sake of making it easier to read it was split into several header and .cpp (although that's the extension for C++) files. The header files contain the declarations of the functions and the variables. In the .cpp files the actual functions of the program are implemented.

In general, most of the code is written on top of the Arduino core functionality, with the exception of the shift-register code directly using the PORTC-register of AVR core to speed up writing the shift-register states.

Protothreads

To further improve the readability the library protothreads was used for handling simple timing. The following image shows an example on how it's used.

```
#include "proto_thread.h" //Simple co-operative threading
pt sensorPt;

void setup()
{
  PT_INIT(&sensorPt);
}

void loop()
{
  updateSensors(&sensorPt);
}


PT_THREAD(updateSensors(struct pt *pt))
{
  PT_BEGIN(pt);
  while(1)
  {
    PT_WAIT_MS(pt, 100);

    measureLDR();
    measureAllPhotoDiodes();
  }
  PT_END(pt);
}
```

During the setup -part, the protothread-structs, like sensorPt, are initialized. While the program is running, when the thread is called, the macros defined in the protothread-header handle timing and returns from the function. For example, to set the time on how often the code inside the thread is executed, PT_WAIT_MS -macro is called. If less than the given amount of milliseconds has been passed since last call, the execution will return immediately, and the lines following the PT_WAIT_MS -line won't be executed.

defines.h: common definitions

For testing purposes some of the code can be altered or disabled through #define -macros. For example when wanting to work on a robot without it starting to run around due to IR-noise or such, the motor control code could be disabled.

All the disable functions were placed in the defines.h file. Commenting and uncommenting certain defines was used to control the behaviour of parts of the software.

```
#define DISABLE_MOTORS      //Completely disables (removes) all motor control code. Meant for testing s
                            //want the robot to start moving around
```

Inside the .cpp files the preprocessor looks if the motors are enabled or disabled with the following lines.

```
#ifndef DISABLE_MOTORS
  #include "Motor.h"
#endif
```

Furthermore, the defines -file contains all the definitions for the board pins, helper -macros, limits, bit masks and such.

```
#ifndef PROJECT_DEFINES_H
#define PROJECT_DEFINES_H

//Feature set
//#define DISPLAY_ENABLED   //Enables display, note that the robot won't do anything if this is enabled and
there's no display

#define DISABLE_MOTORS      //Completely disables (removes) all motor control code. Meant for testing stuff
when you don't
                            //want the robot to start moving around
#define REVERSE_MOTORSPEED  //Reverses the PWM-control for motors, needed for robots #2-5 (as robot #1 had
different motors
                            //with reversed direction vs. the others, and of course I wired the motors
according to #1 directions)


//Bit manipulation macros
#define bit_get(p,m) ((p) & (m))
#define bit_set(p,m) ((p) |= (m))
#define bit_clear(p,m) ((p) &= ~(m))
#define bit_flip(p,m) ((p) ^= (m))
#define bit_write(c,p,m) (c ? bit_set(p,m) : bit_clear(p,m))
#define BIT(x) (0x01 << (x))


//**** Pin definitions
//Photodiode signal input
#define PHOTO_IN A0

//LDR
#define LDR_IN A7

//Shift register (74HC595)
#define SHIFT_CLOCK A1
#define SHIFT_STORE A2
#define SHIFT_DATA A3

//Direct port manipulation for faster writing
#define PORTC_SHIFT_MASK B00001110
#define PORTC_SHIFT_CLOCK BIT(1)
#define PORTC_SHIFT_STORE BIT(2)
#define PORTC_SHIFT_DATA BIT(3)

//Radio pins
#define RADIO_IRQ 2
#define RADIO_CE 7
#define RADIO_CSN 8
#define RADIO_MOSI 11
#define RADIO_MISO 12
#define RADIO_SCK 13
```

```
//Motor outputs
#define MOTOR1_PWM 6
#define MOTOR1_DIR 3
#define MOTOR2_PWM 5
#define MOTOR2_DIR 4

//**** Shift registers
//Leds/photodiodes are positioned as such:
/*
 +--------------------+
 |LED4   LED3     LED2|
 |                LED1|   -- Front --> (motors are on the front side)
 |LED5   LED6     LED7|
 +--------------------+
*/
//1st byte LSB is unused (unconnected in chip)
#define LED1 BIT(1)
#define LED2 BIT(2)
#define LED3 BIT(3)
#define LED4 BIT(4)
#define LED5 BIT(5)
#define LED6 BIT(6)
#define LED7 BIT(7)
#define ALL_LEDS 0xFE

//2nd byte LSB is unused (unconnected in chip)
#define RGB_B BIT(9)
#define RGB_G BIT(10)
#define RGB_R BIT(11)
#define PHOTOREAD_DISABLE BIT(12)  //NOTE: Active low
#define PHOTOREAD_C BIT(13)
#define PHOTOREAD_B BIT(14)
#define PHOTOREAD_A BIT(15)
#define PHOTOREAD_CHANNEL_MASK 0xE000


//** Robot status data
#define LED_COUNT 7
#define FILTER_ALPHA 0.3f

//Indices for directions of LEDs/readings
#define FRONT_INDEX 0
#define FRONT_LEFT_INDEX 1
#define LEFT_INDEX 2
#define BACK_LEFT_INDEX 3
#define BACK_RIGHT_INDEX 4
#define RIGHT_INDEX 5
#define FRONT_RIGHT_INDEX 6
#define NONE_INDEX 127

//Minimum difference for strongest reading vs. average required for the robot to react
#define MINREADING_DIFF_TO_AVERAGE 20

#endif //#ifndef PROJECT_DEFINES_H
```

Motor

The motors are controlled by two values, direction and speed. The speed controls the pwm signal on an H bridge. Since the robot has two Motors which are driven individually they can make the robot drive forwards, backwards, in a curve or make it turn on spot.

PWM-frequency could be made higher by modifying the timer division, but that would require also changing all the timing code to match the changed division, as the timer used for PWM of the motor control pins is the same one that Arduino core uses for other timing.

Shift registers

Shift registers are used to multiplex output pins. The state to write into the registers is stored in shiftState -variable. Last written state is kept in oldState-variable, so call to writeShiftRegisterState() does not write the shift registers unless something has changed.

```
#include "ShiftRegister.h"

uint16_t shiftState = 0;
uint16_t oldState = 0xFFFF;

void writeShiftRegisterState()
{
  /*
  //The simple version, digitalWrite is slow
  digitalWrite(SHIFT_STORE, LOW);
  shiftOut(SHIFT_DATA, SHIFT_CLOCK, MSBFIRST, shiftState);
  shiftOut(SHIFT_DATA, SHIFT_CLOCK, MSBFIRST, shiftState >> 8);
  digitalWrite(SHIFT_STORE, HIGH);
  */

  //Bail out if the state is the same, no need to rewrite it
  if(oldState == shiftState)
  {
    return;
  }

  //Set shift clock & store clock low (although they already should be, but just in case there's some "funny"
state at start up or such)
  bit_clear(PORTC, PORTC_SHIFT_CLOCK | PORTC_SHIFT_STORE);

  //Value is shifted on rising edge of SHIFT_CLOCK, go through the shiftState bit-by-bit
  for(int i = 15; i >= 0; i--)
  {
    //Note that the data is written MSB first (as the first bit entered will be the value of the last output
of the shift registers)
    uint8_t nextBit = (shiftState >> i) & 1;
    bit_write(nextBit, PORTC, PORTC_SHIFT_DATA);
    bit_set(PORTC, PORTC_SHIFT_CLOCK);
    bit_clear(PORTC, PORTC_SHIFT_CLOCK);
  }

  //Store value (ie. activate shift-register outputs to current values)
  bit_set(PORTC, PORTC_SHIFT_STORE);
```

```
  oldState = shiftState;
}

void turnOn(uint16_t bitVal)
{
  bit_set(shiftState, bitVal);
}

void turnOff(uint16_t bitVal)
{
  bit_clear(shiftState, bitVal);
}

uint8_t isEnabled(uint16_t bitVal)
{
  return (shiftState & bitVal) > 0 ? 1 : 0;
}

void initShiftRegister()
{
  //Write out the shift register, the reset- and output enable -pins are not available so the state can be
pretty much anything at start up
  shiftState = RGB_R | PHOTOREAD_DISABLE; //Just turn on the red led and disable analog multiplexer
  writeShiftRegisterState();
}
```

Photodiodes and brightness reading

Infrared LED and Infrared photodiode

The IR_{LED} are positioned around the robot to show the other robots its position. At the moment the IR_{LED} are turned off when the brightness level is low. If it gets brighter they get turned on. The robots measure the IR reading every 100ms. As soon as they see a robot turning on their IR they start driving to it.

There are seven IR photodiode one in the front, three to the left and three to the right. Because of that the robot knows where the IR light source is. With that knowledge it can turn either left or right to adjust itself correctly and afterwards it drives to this IR source. One problem that exist at the moment is that the normal light sources have enough infrared in them to make the robot drive into a random direction. If the room lighting is done with LED lamps the background noise is not big enough for that to occur. Instead of replacing the room lighting there is also the possibility to reduce the sensitivity of the IR photodiodes.

LDR

The LDR-value is simply read through the analog pin it is connected to.

Analog demultiplexing for the IR-modules

The demultiplexing of the ADC-input is handled through an analog demultiplexer. The 74HC4051 is controlled through the shift registers.

```
#include "PhotoDiode.h"
```

```
uint16_t lastPhotoReadings[LED_COUNT];
uint8_t strongestReadingIndex = NONE_INDEX;
uint16_t strongestReading = 0;
uint16_t averageReading = 0;

void initPhotoDiodes()
{
  pinMode(PHOTO_IN, INPUT);
  memset(lastPhotoReadings, 0, LED_COUNT * sizeof(uint16_t));
}

void measurePhotoDiode(uint8_t diodeNumber)
{
  //Select the correct channel
  ((diodeNumber & 1) > 0) ? turnOn(PHOTOREAD_A) : turnOff(PHOTOREAD_A);
  ((diodeNumber & 2) > 0) ? turnOn(PHOTOREAD_B) : turnOff(PHOTOREAD_B);
  ((diodeNumber & 4) > 0) ? turnOn(PHOTOREAD_C) : turnOff(PHOTOREAD_C);

  turnOff(PHOTOREAD_DISABLE);
  writeShiftRegisterState();

  //Let the value settle a bit
  //TODO: Is this needed really?
  delay(1);

  //Filter the value a bit
  lastPhotoReadings[diodeNumber] = (uint16_t)((1.0f - FILTER_ALPHA) * analogRead(PHOTO_IN) + FILTER_ALPHA *
lastPhotoReadings[diodeNumber]);
  //lastPhotoReadings[diodeNumber] = (uint16_t)(((uint32_t)1023*lastPhotoReadings[diodeNumber] +
(uint32_t)analogRead(PHOTO_IN)) / 1024);

  turnOn(PHOTOREAD_DISABLE);
  writeShiftRegisterState();
}

void measureAllPhotoDiodes()
{
  for(int i = 0; i < LED_COUNT; i++)
  {
    measurePhotoDiode(i);
  }

  averageReading = 0;
  uint32_t tempAvg;
  strongestReading = 0;
  for(int i = 0; i < LED_COUNT; i++)
  {
    uint16_t r = lastPhotoReadings[i];
    tempAvg += r;
    if(r > strongestReading)
    {
      strongestReading = r;
      strongestReadingIndex = i;
    }
  }

  averageReading = (uint16_t)(tempAvg/LED_COUNT);

  if((strongestReading - averageReading) < MINREADING_DIFF_TO_AVERAGE)
  {
    strongestReadingIndex = NONE_INDEX;
```

```
    strongestReading = 0;
  }
}
```

Status LED

The status LED is a RGB LED and its goal is to show the user what the robot is currently doing.

At the moment the LED is blue if the brightness reading is very low. When the brightness reading gets higher than a certain level the LED turns red. The different colors of the LED are turned on and off through the shift registers.

When the program is more finished and has a more complex function the status LED has to signal the user with more information.

Radio

The radio library used here is called RF24[36]. The radio has six different channels. This Software only uses one of them. Most of the time the robots are listening for incoming messages. The robot only switches to the sending modus when it send the protocol. Afterwards it immediately switches back. While in a sending modus the radio can't listen for incoming messages. But if the radio is in a sending modus it stores the message until it is overwritten by the next incoming message. This message can get called at any time.

In this project there are several participant in the communication via radio. The sending one is the master and the rest are Slaves. Normally the radio expects a response from the receiver because messages can get lost. Since there are an undefined number of listeners around that doesn't work. The only place where a check is crucial is at the handover of the master position to the next robot. If the previous master receives the message of the next master then the handover was successful. If there is no reply the previous master retries again.

```
#ifndef RADIO_H
#define RADIO_H

#include <Arduino.h>
#include <stdint.h>
#include "defines.h"

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

struct List{
  char idOwn;
  char brightness;
  char idCurrent;
  char groupSize;
  char singleBits;//bit0: ownBeaconMode, bit1: ownMoved, bit2: rob1Moved,
                  //bit3: rob2Moved, bit4: rob3Moved, bit5: rob4Moved
  char visibleRobots[3][4];
```

```
  char otherRobots[14][4];
};
extern struct List listRobots;

#define UNCONNECTED 16
#define IDCONFLICT 32
#define MEASURING1 40
#define MEASURING2 41
#define MEASURING3 42
#define STANDING 48
#define WANTTOMOVE 64
#define MOVING 65
#define BEACON1 66
#define BEACON2 67
#define WAITING 68
#define SPREADINGOUT 80

void initRadio();
void checkRadio();
char write(char flag);
char callculateNextID();
int createProtocol(char *msg, char maxSizeMsg);
char read(char flag);
void readProtocol(char *message);

#endif // #ifndef RADIO_H
#include "Radio.h"

struct List listRobots;
char msgBuffer[32];
RF24 radio(7, 8); // CNS, CE
const byte address[6] = "00001";
static char flag;

void initRadio() {
  radio.begin();
  radio.setPALevel(RF24_PA_MAX);
  radio.setDataRate(RF24_250KBPS);
  radio.enableAckPayload();
  flag=read(flag);

// hardcoded stuff, gets removed later on
  listRobots.idOwn=2;
  listRobots.idCurrent=1;
  listRobots.groupSize=2;
  listRobots.brightness=86;
}

void checkRadio(){
  static char sendMsg=0;

  if(listRobots.idCurrent==listRobots.idOwn){
    sendMsg=1;
  }else{
    if (radio.available()) {
      char messageBack[35] = "";
      radio.read(&messageBack, sizeof(messageBack));
      readProtocol(messageBack);
      sendMsg=2;
    }else{
      if(sendMsg!=2){
```

```
        sendMsg=1;
      }
    }
  }
    Serial.println((int)sendMsg);
  if(sendMsg==1){
    flag=write(flag);
    int Size=createProtocol(msgBuffer, 32);
    radio.write(msgBuffer, Size);
    Serial.println(Size);
    sendMsg=0;
    flag=read(flag);
  }
}


//=============================================================================================
// Methods write
//=============================================================================================


char write(char flag){
  if(flag!=0){
    radio.stopListening();
    radio.closeReadingPipe(address);
    radio.openWritingPipe(address);
    radio.flush_tx();
    flag=0;
  }
  return flag;
}

char callculateNextID(){
  char idNext;
  if(listRobots.idOwn<listRobots.groupSize){
    idNext=listRobots.idOwn+1;
  }else{
    idNext=1;
  }
  return idNext;
}

int createProtocol(char *msg, char maxSizeMsg){
  static char task = STANDING;
  char endInformation=0;
  *msg=listRobots.idOwn;
  *(msg+1)=listRobots.groupSize;
  *(msg+3)=task;
  int sizeMsg=4;


  Serial.println("======================");
  switch(task) {

   case UNCONNECTED :
      sizeMsg=3;
      Serial.println("Not connected");
      break;

    case IDCONFLICT :
      sizeMsg=3;
```

```
      Serial.println("ID conflict");
      break;

    case MEASURING1 :
      sizeMsg=3;
      Serial.println("Measuring");
      break;

    case STANDING :
      endInformation=1;
      Serial.println("Standing still");
      break;

    case WANTTOMOVE :
      sizeMsg=5;
      Serial.println("Beacon Mode");
      break;

    case MOVING :
      sizeMsg=9;
      Serial.println("Wants to move");
      break;

    case BEACON1 :
      sizeMsg=3;
      Serial.println("Moving");
      break;

    case WAITING :
      sizeMsg=5;
      Serial.println("Stopped Moving but waiting");
      break;

    case SPREADINGOUT :
      sizeMsg=3;
      Serial.println("Spreading out");


      break;

   default :
    Serial.println("task invalid");
  }
  if(endInformation==1){
    *(msg+sizeMsg)=listRobots.brightness;
    sizeMsg++;

    char idNext=callculateNextID();
    listRobots.idCurrent=idNext;
    *(msg+sizeMsg)=idNext;
  }
  sizeMsg++;
  *(msg+2)=sizeMsg;
  return sizeMsg;
}


//=============================================================================================
// Methods read
//=============================================================================================
```

```
char read(char flag){
  if(flag!=1){
    radio.openReadingPipe(0, address);
    radio.startListening();
    flag=1;
  }
  return flag;
}

void readProtocol(char *message){
  char endInformation=0;

  Serial.println("======================");
  char id=*message;
  char groupsize=*(message+1);
  char sizeMsg=*(message+2);
  char task=*(message+3);
  Serial.println((int)id);
  Serial.println((int)groupsize);
  Serial.println((int)sizeMsg);
  Serial.println((int)task);
  if(task==STANDING){
    endInformation=4;
  }

  if(endInformation!=0){
    listRobots.otherRobots[1][id]=*(message+endInformation);
    listRobots.idCurrent=*(message+endInformation+1);
    Serial.println((int)listRobots.otherRobots[1][id]);
    Serial.println((int)listRobots.idCurrent);
  }

}
```

Debugging with OLED-displays

To ease with debugging, a display could be used through the I^2C -bus. The display was controlled using the SSD1306Ascii -library[37], as no images needed to be drawn.

Conclusions

While the project wasn't finished as planned, the hardware is working as intended, and the software-side basically has the needed building blocks for making the final logic where the robots actually communicate and work as a swarm. I think both of us learned new things along the way, and at least I personally don't feel that the project was any sort of failure either, despite missing deadlines and not getting everything done.

What failed

The delays with getting the hardware done caused the software-side to fall behind, as for example working on the moving logic was hard without the actual robots, and for actual "swarm"-logic, a minimum of 3 robots is needed. The issues were mostly with lack of time, as I changed jobs in the middle of the course, which led to long work weeks finishing and documenting stuff at previous employer, already attending meetings and such at my new employer during my notice-period and then launching directly into a new, tightly scheduled work-project, not leaving that much time and energy to work on the robot-project during weeks, throwing the original schedule out the window.

While the original schedule did have some leeway and delays were expected, multiple separate test-boards needed to be designed and milled, for the switching converters, IR-modules and also for the motor-chips, as SMD-chips cannot be breadboarded directly and the footprints weren't "standard" SOPs or such, to which I would have had adapter boards on the shelf. Datasheets needed to be read and compared, and more IR-components and op-amps needed to be ordered after testing the different IR-LEDs, phototransistors and photodiodes. Even though the final robot boards were fabricated elsewhere, the milling, assembly and testing of the over 40 IR-module boards, including the prototypes and the single prototype robot mainboard took quite a lot of time. I haven't kept a record, but I'd estimate the hardware design and implementation took something between 150 and 200 hours in total. Falling behind on one thing meant that all the other things were pushed further forwards, and the compounded delays then reflected on the software-side. The last week of the project was dedicated to writing this document.

I would have liked to make more measurements of the boards, for example the higher than expected ripple-issue in the switching converters. The test-setup wasn't too good, as I don't have an actual adjustable DC-load, probably should make one, and/or maybe get another second-hand HP6632 so I could use one as power supply and the other one as load to have more controlled testing environment. It could be simply that the load was too small (about 20mA), and the ripple would be within the expected values, if measured with higher load, but I never had the time to redo the measurements, and the small current reflects the situation when only the MCU and the ICs are drawing current. The motor noise was never investigated further, and the inductor/ferrite bead in the filter was simply replaced with a 0Ω-resistor, but it never seemed to be an issue.

The radio-code was very problematic due to not knowing for sure where the issues were (code, library, faulty/misbehaving clone-chips, wiring, timing, noise, brown outs...), and the transmission

was constantly unreliable, making it frustrating to work with. I couldn't give Philipp as much support as I had wanted to, and he had to wrestle with the issues mostly by himself, as in the beginning we had only three radio-modules, two of which were with him, so I couldn't try out things at home not having at least two modules to work with. The extra radio-modules didn't arrive until late October, at which point I was already too busy with everything else to try much with them myself. With the final hardware, the reliability seemed better, so it could be that at least some of the issues were in the connections or the power supply of the breadboarded radio modules, but there wasn't enough time left to finish both the software and this final report.

What succeeded

Despite the issues, the robots work, and the actual functionality can be implemented later on. The final hardware has worked as expected, we could have had much more issues with that, while it isn't really that complicated (at least the parts that were designed for this project, not the ready-made display- and especially the radio-modules, RF design is pure black magic to me). I had never before worked with switching converters or transimpedance amplifiers, which also explains why there's so much written about them in this document, so that was something completely new to me. I can hardly say that I "know" switching converter design now, but at least the basics. I'm happy with the end results, although of course there's always things that could be improved.

The combination of producing prototypes with milled boards (of course they could also be etched) and then testing that things work before ordering fabricated boards worked well. I've had a friend order boards directly from his designs, only to notice that they're missing a trace or have a wrong footprint for a component after receiving 10 or more boards, leading to waste of money, time and resources. Most of my own projects only ever need one or two boards of the same design, but I will definitely be using fabrication services in the future when larger amounts of the same board is needed.

Philipp did get the robot state-logic and radio-protocol designed and the modules working despite all the issues, but there wasn't enough time to implement them with the actual robots.

In the end, I still have 5 leftover boards from the batch, but they could be used for other projects, such as some kind of sensor-platforms or such. The I^2C -pins were left available, and each of the IR-module connectors has 5V supply, one digital input coming from the shift-register and one analog (or could be as well used as digital) output, so they can be used for other stuff too. The mainboards have extra 5V- and 3.3V-supply output pins, connectors for the MCU pins that were unused for the robots, and possibility to use amplification in the demultiplexed analog input, so other devices could be attached too.

I do have an honest intent to finish the software side too, and to explore possible further use cases from there on out, but I might need a short vacation from the project first.

Esa Jääskeläinen

7.12.2017

References

1. Battery University. BU-205: Types of Lithium-ion. http://batteryuniversity.com/learn/article/types_of_lithium_ion

2. Nkon.nl. Selection of available lithium-ion 18650-cells on an European reseller website https://eu.nkon.nl/rechargeable/18650-size.html

3. Wikipedia. Memory Effect -article. https://en.wikipedia.org/wiki/Memory_effect

4. Advanced Monolithic Systems Inc. Datasheet for AMS1117, 1A Low Dropout Voltage Regulator http://www.advanced-monolithic.com/pdf/ds1117.pdf

5. Horowitz, P. & Hill, W. The Art of Electronics, third edition. Pages 636-672.

6. Richtek Technology Corp. Datasheet for Richtek RT7297B, a synchronous step-down converter http://www.farnell.com/datasheets/1756307.pdf

7. Mosaic Industries. Designing Step-Down (Buck) Switching Regulators http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/electronic-circuits/step-down-switching-regulator

8. ON Semiconductor. Switch-Mode Power Supply Reference Manual https://www.onsemi.com/pub/Collateral/SMPSRM-D.PDF

9. Daycounter Inc. Buck Switching Converter Design Equations http://www.daycounter.com/LabBook/BuckConverter/Buck-Converter-Equations.phtml

10. Texas Instruments Inc. Output Ripple Voltage for Buck Switching Regulator. http://www.ti.com/lit/an/slva630a/slva630a.pdf

11. Intel Corporation. ATX12V Power Supply Design Guide http://www.formfactors.org/developer/specs/ATX12V%20PSDG2.01.pdf

12. Analog Devices, Inc. Analog Devices Application Note AN-1144: Measuring Output Ripple and Switching Transients in Switching Regulators http://www.analog.com/media/en/technical-documentation/application-notes/AN-1144.pdf

13. EEVBlog forum. A forum post detailing HP6632A usage as a load. https://www.eevblog.com/forum/testgear/hp-6632a-negative-constant-current/

14. LuckyLight Electronics Co. Datasheet for LL-S150IRC-2A, an Infrared Chip LED http://www.luckylight.cn/UploadFiles/LL-S150IRC-2A.pdf

15. Vishay Intertechnology, Inc. Datasheet of VEMD10940F silicon PIN Photodiode https://www.vishay.com/docs/84171/vemd10940f.pdf

16. Wikipedia. PIN diode -article. https://en.wikipedia.org/wiki/PIN_diode

17. Wikipedia. Transimpedance Amplifier -article. https://en.wikipedia.org/wiki/Transimpedance_amplifier

18. Texas Instruments, Inc. Mesaurement of Photodiode Currents. http://www.ti.com/europe/downloads/light_measurement_photodiode_currents.pdf

19. Vishay Intertechnology, Inc. Measurement Techniques. Page 7, Fig 21. https://www.vishay.com/docs/80085/measurem.pdf

20. Microchip Technology, Inc. Datasheet for MCP601/1R/2/3/4. 2.7V to 6.0V Single Supply CMOS Op Amps. http://ww1.microchip.com/downloads/en/DeviceDoc/21314g.pdf

21. Micrel, Inc. Datasheet of MIC7300 High output drive rail-to-rail op-amp. http://ww1.microchip.com/downloads/en/DeviceDoc/mic7300.pdf

22. Diodes Incorporated. Datasheet for BSS138, N-channel enhancement modfe MOSFET, https://www.diodes.com/assets/Datasheets/ds30144.pdf

23. Electrical Engineering Stackexchange. Answer on calculating Adjacent trace capacitance (horizontal). https://electronics.stackexchange.com/questions/264229/adjacent-trace-capacitance-horizontal/264235#264235

24. Nordic Semiconductor. nRF24L01+ Ultra low power 2.4GHz RF Transceiver IC. https://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01P

25. Hackaday.com. Nordic nRF24L01+ - Real vs. Fake -article. https://hackaday.com/2015/02/23/nordic-nrf24l01-real-vs-fake/

26. Hackaday.com. Nordic nRF24L01+ - Real vs. Fake -article, user comment from Nordic Semi employee. https://hackaday.com/2015/02/23/nordic-nrf24l01-real-vs-fake/#comment-2474764

27. Texas Instruments, inc. Datasheet of L293x Quadruple Half-H Drivers. http://www.ti.com/lit/ds/symlink/l293.pdf

28. ST Microelectronics. Datasheet of L298 Dual Full-bridge Driver. http://www.st.com/stonline/books/pdf/docs/1773.pdf

29. rugged-circuits.com. The Motor Driver Myth -article. https://www.rugged-circuits.com/the-motor-driver-myth/

30. PICAXE forums. An alternative to the L293D dual H Bridge IC -message thread. http://www.picaxeforum.co.uk/archive/index.php/t-28938.html

31. ON Semiconductor, inc. Datasheet of LV8548MC, Low saturation motor driver. http://www.onsemi.com/pub/Collateral/LV8548MC-D.PDF

32. Elite/Chinsan Electronic. Datasheet of UPE series, Conductictive Polymer Aluminum Solid Capacitors. http://www.chinsan.com/wp-content/uploads/datasheet/cs-cap/UPE.pdf

33. Nexperia, Inc. Datasheet of 74HC595, an 8-bit serial-in, parallel-out -shift register. https://assets.nexperia.com/documents/data-sheet/74HC_HCT595.pdf

34. Nexperia, Inc. Datasheet of 74HC4051, an 8-channel analog multiplexer/demultiplexer. https://assets.nexperia.com/documents/data-sheet/74HC_HCT4051.pdf

35. Atmel, Inc. Datasheet for ATMega328/328P, ADC input impedances. Page 312. http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

36. Charl Thiem / trojanc. Optimized fork of nRF24L01 for Arduino & Raspberry Pi/Linux Devices. https://github.com/nRF24/RF24

37. Bill Greiman / greiman. SSD1306Ascii: Character drawing library for small OLED displays that use SSD1306 controller. https://github.com/greiman/SSD1306Ascii