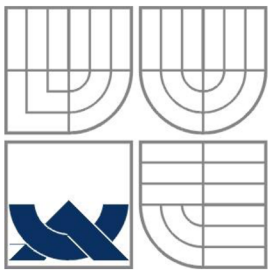
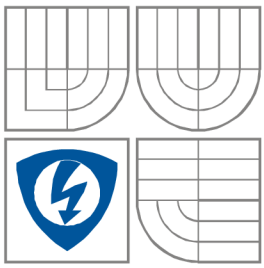


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

DÁLKOVĚ OVLÁDANÝ KOLOVÝ ROBOT

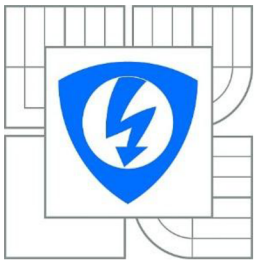
REMOTE CONTROLLED WHEEL ROBOT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
SÁNDOR RUHÁS

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. TOMÁŠ FLORIÁN

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Bakalářská práce

bakalářský studijní obor
Automatizační a měřicí technika

Student: Sándor Ruhás *ID:* 125302
Ročník: 3 *Akademický rok:* 2011/2012

NÁZEV TÉMATU:

Dálkově ovládaný kolový robot

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte dálkově ovládaný mobilní robot. Vyberte vhodný mikrokontrolér, kterým bude robot řízen. Jako pohony použijte servomotory od společnosti Dynamixel RX-64. Robot doplňte o vhodné snímače a bezdrátový modul. Ovládací software pro PC bude realizován v jazyce C#.

DOPORUČENÁ LITERATURA:

H.R. Everett, Sensors for Mobile Robots, A K Peters/CRC Press (July 15, 1995), ISBN-13: 978-1568810485

Termín zadání: 6.2.2012 *Termín odevzdání:* 28.5.2012

Vedoucí práce: Ing. Tomáš Florián
Konzultanti bakalářské práce:

doc. Ing. Václav Jirsík, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

Abstrakt:

Tato práce se zabývá návrhem a vytvořením kolového robotu na dálkové ovládání. První část práce obsahuje základní úvod do robotiky. Poté se čtenář seznámí se všemi kroky návrhu, které byly provedeny. Samotná konstrukce je rozdělena do dvou hlavních částí: návrh hardware a software. V kapitole návrhu hardware se čtenář dozví, jaké součástky byly vybrány a proč. Pro vysvětlení funkce součástek, budou uvedeny některé důležité parametry. Protože většina součástek může být nastavena různým způsobem, jsou hlavní nastavení detailně popsána.
V dokumentu je také uveden jednoduchý ilustrovaný popis hlavních funkcí obslužného software.
Na konci tohoto textu jsou diskutovány dosažené výsledky.

Klíčová slova:

Xbee, PIC24HJ64GP502, Dynamixel RX-64, robot, kompas, akcelerometr

Abstract:

This work deals with design and creation of a 4-wheel remote controlled robot.
The first part of the project includes some introduction to robotics. After that, the reader will get familiar with all the design steps which were made to achieve a functional robot. The design itself is divided into two major parts: hardware and software design. In the hardware design chapter the reader will learn what kind of parts where selected and why. Some main parameters of these parts will also be supplied to understand main functionality. Because most of these parts can be configured in some way, main configuration steps with detailed description can be found in the project for each configurable part.
Regarding the software, this document will supply a simple description about the main functions of the software illustrated with pictures.
In the end of this text the achieved results are discussed.

Keywords:

Xbee, PIC24HJ64GP502, Dynamixel RX-64, robot, compass, accelerometer

Bibliografická citace mé práce:

RUHÁS, S. *Dálkově ovládaný kolový robot.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2012. 58 s. Vedoucí bakalářské práce Ing. Tomáš Florián.

Prohlášení

Prohlašuji, že svoji bakalářskou práci na téma Dálkově ovládaný kolový robot jsem vypracoval samostatně pod vedením vedoucí bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne: **16. května 2012** .....

podpis autora

Poděkování

Děkuji vedoucí bakalářské práce Ing. Tomáš Florián za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne: **16. května 2012** .....

podpis autora

Contents

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Definition of robot

A robot is a reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialized devices trough variable programmed motions for the performance of a variety of tasks: Robot Institute of America, 1979

Types of robots

We can divide robots into a lot of categories; in the following list we find the main categories:

- Mobile Robots
- Stationary Robots
- Autonomous Robots
- Remote-control Robot

Laws of Robotics [Isaac Asimov]

0. A robot may not harm humanity, or, by inaction, allow humanity to come to harm.
1. A robot may not injure a human being or, trough inaction, allow a human being to come to harm.
2. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

Designing a robot

I always wanted to build a robot on my own, and to test different kind of sensors. This project offers the opportunity to do so.
Because the main task of this project gives me freedom, I had a chance to design the whole robot myself. This means that I can design the body, pick the sensors, design the whole controlling "principle" …etc.
This "freedom" also comes with a lot of questions:

- Should I make metal or plastic body?!
- How big the robot will be?!

- What kind of sensors will I use?!
- What will be the abilities of the robot?!
- PCB: use standard “trough-hole” parts or use SMD packages?!
- For what purposes will be this robot used?!

So I had to consider a lot of things before starting the design.
Because I love to try out new parts (and sensors too) I started to look around on the internet. In the meantime I decided that I don’t want to build a robot for a specific task, so the sensor selection will be almost random. From this comes out that my robot will serve demonstration purposes.
In the following chapters I will describe the different parts and sensors what I chose during the development

2 SELECTING THE MAIN PARTS OF THE ROBOT

2.1 Selecting an RF module for wireless communication

While looking for a suitable RF module, the following was considered:

- Maximum distance of transmission
- Transmission reliability
- Current consumption
- RF data rate
- Difficulty to use
- Price

Selected RF module: Xbee Pro S2 (International variant) from Digi International.

2.1.1 Main parameters & using the Xbee modules

Main parameters:

Supply voltage	3.0-3.4 V
RF data rate	250 000 bps
Range	1500m – outdoor, 60m – indoor
Operating current	170mA (max. output power)
Idle current	45mA

Figure 2-1: The Xbee PRO module [6]

Using Xbee modules:

Xbee's are easy-to-use modules for low cost wireless communication. They are interfacing to a host device using UART serial port. If the logic levels are the same, the module can be connected directly to a host MCU. An additional logic level converter is required if the logic levers are different (A hundreds of converters are available from different manufacturers). Hardware flow control also available, if the user wants to use this feature.

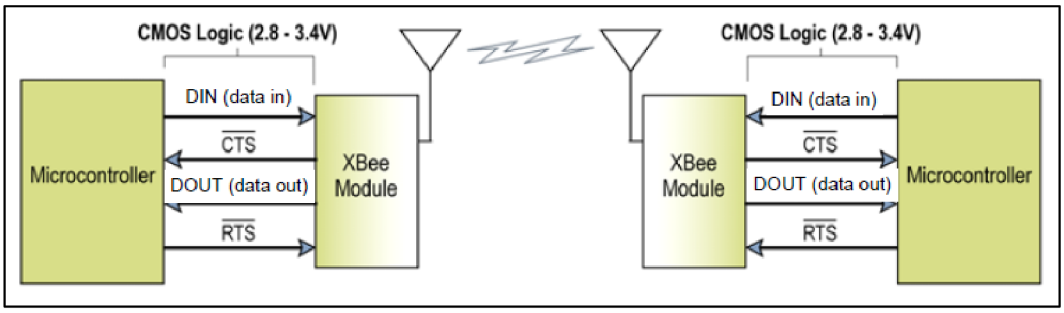


Figure 2-2: Connecting the modules to a host microcontroller [6]

We can use the module in 2 different modes: Transparent or API operation.

Transparent operation – the module acts as a serial line replacement. All UART data received through the DIN pin is queued up for RF transmission. When RF data is received, it is sent out through the DOUT pin. In transparent operation the destination is set at the main configuration (the user must reconfigure the module with entering to command –AT– mode)

API operation – This is a frame based operation. Every communication between the module and the host MCU is done by frames. The API operation is not that simple as the transparent mode, but offers a lot more. With using the API, we can send data to multiple destinations without entering to command mode, configure remote modules, sample IO lines on remote module and do a lot of other things.

In this project we will use API operation because of the flexibility.

Before using the modules, we have to configure them to meet our requirements. The manufacturer provides a very useful program for that, called the "X-CTU".

To successfully use the modules, we have to configure one of them as a coordinator, and the second one as a router (In this design 2 Xbee modules are required – one will be placed on the robot, the second will be connected to a PC). The coordinator creates a network, other devices (such as router, or end device) can join to the created network if the coordinator allows that. Every network has to contain minimally one coordinator.

With X-CTU we can upload the required firmware to the modules. After that, we can set all the necessary options. With using the API operation, we don't have to spend a lot of time with configuration – setting the speed of the serial communication and enabling API mode will be completely enough.

(Detailed information of the communication and packetizing will be discussed in the next chapters)

2.1.2 Xbee API operation

Like said in the previous chapter, in API operation mode all communication is done using frames.

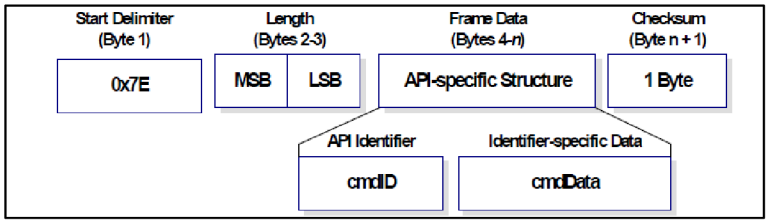


Figure 2-3: API frame structure [6]

- Every frame starts with the "start delimiter" – 0x7E
- A 2 byte value follows the start delimiter indicating the frame length (the number of bytes that will be contained in the frame data field excluding the checksum).
- Each API frame is identified with an ID, the available ID's are listed in the table below.
- After frame identification the module processes the data contained in the cmdData field. The meaning of data in this field depends on the frame identifier.
- The frame ends with a control checksum. To calculate checksum: excluding the frame delimiters and length we have to add all bytes together (with keeping the lowest 8 bits of the result) and subtract the result from 0xFF.

In the following situations the received frame will be discarded:
Invalid checksum, missing start delimiter, invalid length, invalid API identifier or the frame is received incorrectly.
In the following table we can find the available API frame types:

API Frame Names	API ID
AT Command	0x08
AT Command- Queue Parameter Value	0x09
ZigBee Transmit Request	0x10
Explicit Addressing ZigBee Command Frame	0x11
Remote Command Request	0x17
Create Source Route	0x21
AT Command Response	0x88
Modem Status	0x8A
ZigBee Transmit Status	0x8B
ZigBee Receive Packet	0x90
ZigBee Explicit Rx Indicator	0x91
ZigBee IO Data Sample Rx Indicator	0x92
XBee Sensor Read Indicator	0x94
Node Identification Indicator	0x95
Remote Command Response	0x97
Over-the-Air Firmware Update Status	0xA0
Route Record Indicator	0xA1
Many-to-One Route Request Indicator	0xA3

Table 2-1: API frame types [6]

In this project the following frame types will be used: ZigBee transmit request, ZigBee receive packet.

2.2 Selecting a suitable microcontroller

While looking for an appropriate MCU, the following was considered:

- Speed
- Amount of communication interfaces
- Number of general purpose IO-s
- Support availability

Selected MCU: PIC24HJ64GP502 from Microchip Technology Inc.

2.2.1 Main features of the selected MCU

This microcontroller was selected because of the 16-bit wide data path, the availability of the low-cost development kit: "Microstick" which supports source level debugging, and because I had some experiences with microcontrollers from Microchip Technology.

Main features:

- Modified Harvard architecture
- Up to 40 MIPS operation
- External crystal not necessary – contains an on-chip FRC oscillator
- C-compiler optimized instruction set
- Hardware support for SPI, I^2C and UART communication
- Up to five 16-bit timers
- PWM support
- 45 available interrupt sources, with programmable priority levels
- Remappable peripherals

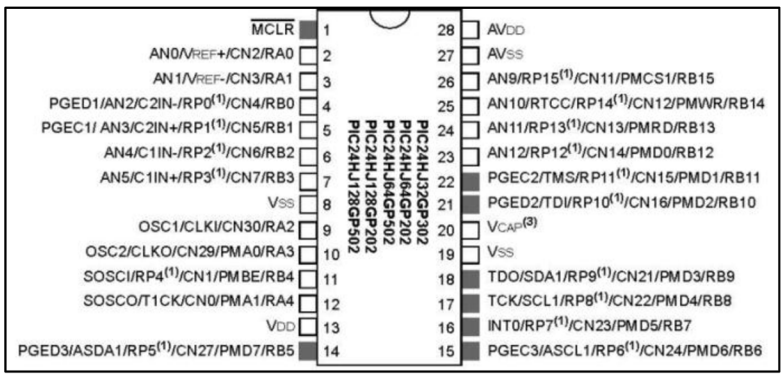


Figure 2-4: Microcontroller pin out [1]

2.2.2 Microcontroller configuration

Main configuration steps:

1. Configure and Tune the clock
2. Enable all peripherals which will be used
3. Disable unused peripherals for power saving
4. Disable A/D converter (not used in this project)
5. Map enabled peripherals to the best IO pin (considering the PCB design)
6. Configure interrupt(s)
7. Configure timer(s)
8. Test the configuration. (With testing the UART, FRC tuning can be confirmed. If UART fails sending or reading the appropriate character, the FRC tuning is wrong. The sent character won't appear on the receiver side correctly).

Actual configuration: FRC oscillator, maximum clock speed, UARTs enabled, SPI enabled, I2C enabled, Timer1-4 enabled, A/D disabled, Input Capture enabled, Output Compare enabled, interrupt enabled for UART and the timers.

2.3 Compass module

As a first sensor the HMC6352 Digital Compass Solution was selected from Honeywell.

The sensor includes 2-axis magneto-resistive sensors and additional circuits & algorithms for heading computation. The sensor calculates the heading and outputs it over I2C protocol. Data from the magneto-resistive sensors also available for reading.
The sensor can be easily interfaced to the PIC24HJ64GP502, the I2C protocol only requires 2 wires for establishing the connection. Two additional pull-up resistors are required for the SCL and SCA line.

In our configuration, the PIC will act as a master, while the compass will be act as a slave device.

Every transmission begins with the master device issuing a start sequence followed by the slave address byte. The slave address is 7 bit wide only (upper 7 bits), the remaining bit (least significant bit) is used to distinguish read/write operations. The default HMC6352 address is 0x42 for write operations, and 0x43 for read operations (factory default). After sending the address, the master waits for an acknowledge from the slave. After getting the acknowledge, the master sends the data bytes to the slave, or performs a read operation (depending on the sent address). The bus transactions are terminated with the master issuing a stop sequence.
In all times the master handles the clock signal!

We can use our compass in different modes of operation.
Supported modes:
Standby mode- Factory default, waiting for commands from master. No measurement until it's requested.
Query mode- The internal processor of the compass waits for an "A" command (command list can be found in appendix), performs a measurement, calculates the heading, and waits for a read command. After a read, the compass automatically performs another measurement and updates the data registers.
Continuous mode- The sensor performs periodic measurements with selectable rates of 1Hz, 5Hz, 10Hz or 20 Hz.

2.3.1 Compass configuration

The compass has two parameters which are controlling its operation. With configuring the *Operation Mode* register we can control the continuous measurement rate, set/reset function, and choose one of the three operation modes: Standby, Query, Continuous.
With configuring the *Output Mode* register we can set the compass to output heading data, or magnetometer data.
At first use I set the compass for continuous measurement, with a rate of 20 Hz. This configuration is saved to internal EEPROM, so further configuration not necessary.

Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)
0	M. Rate_H	M. Rate_L	Per. S/R	0	0	Op Mode_H	Op Mode_L

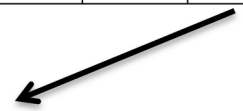
Bit 6	Bit 5	Description
0	0	1 Hz measurement rate
0	1	5 Hz measurement rate
1	0	10 Hz measurement rate
1	1	20 Hz measurement rate

Bit 1	Bit 0	Description
0	0	Standby mode
0	1	Query mode
1	0	Continuous mode
1	1	Not allowed

Bit 4 – Periodic Set/Reset – 0 = OFF, 1 = ON

Table 2-2: Operation mode register [5]

Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)
0	0	0	0	0	Mode	Mode	Mode



Bit 2	Bit 1	Bit 0	Description
0	0	0	Heading mode
0	0	1	Raw Magnetometer X Mode
0	1	0	Raw Magnetometer Y Mode
0	1	1	Magnetometer X Mode
1	0	0	Magnetometer Y Mode

Table 2-3: Output mode register [5]

2.4 Accelerometer

As the second sensor, the ADXL345 accelerometer from Analog Devices takes place on the robot.

The ADXL345 is a 3-axis accelerometer with selectable range of measurement. Selectable ranges are: ±2g, ±4g, ±8g, ±16g.

Its high resolution enables measurement of inclination changes less than 1.0°.

The accelerometer has other special functions such as activity/inactivity sensing, tap/double tap sensing, free fall detection, and interrupt capability. However we won't use all of these functions- so they will be disabled.

The accelerometer can communicate via SPI or I2C interface. We will use the 4-wire SPI configuration. Regarding the measurement range, for tilt measurement the lowest range is the best choice (Tilt is a static measurement where gravity is the acceleration being measured. Therefore, to achieve the highest degree resolution of a tilt measurement, a low-g, high sensitivity accelerometer is required).

Information about acceleration is stored to specified registers, which ones can be read by software. Acceleration information for each axis has 2 bytes length.

The accelerometer allows the user to read multiple bytes at once, so the 6 bytes (X-axis, Y-axis, Z-axis) can be read very quickly. The SPI protocol however requires always a write operation in same time as the read operation. We can't just read from, or just write to the device. So if we want to read from the target device, we have to write "dummy" data and during that, read the useful information. The software has to separate useful information from the information that has to be discarded.

2.4.1 Accelerometer configuration

After the voltage supply is applied to the chip, it enters to standby mode waiting for further commands.
To start measurement, device configuration is needed. Configuration has to be done in standby mode and when it's done, the user can enable measurement. Before changing any of the configuration settings, the measurements should be stopped.
In the following tables we can find the descriptions of all control registers that are used in this project.

D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	Low_Power	Rate			

Table 2-4: Register BW_RATE [4]

Low_Power bit: 0 = normal operation, 1 = reduced power operation (higher noise).
Rate bits: selecting device bandwidth and output data rate.

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	Auto_Sleep	Measure	Sleep	Wakeup	

Table 2-5: Register POWER_CTL [4]

Measure bit: 0 = standby mode, 1 = measurement mode
(Other bits aren't used, so description is not necessary)

D7	D6	D5	D4	D3	D2	D1	D0
Data_Ready	Single_Tap	Double_Tap	Activity	Inactivity	Free_Fall	Watermark	Overrun

Table 2-6: Register INT_ENABLE [4]

These bits are used to enable/disable interrupts. In the control software we will disable all interrupts.

2.5 Digital temperature sensor

The next sensor on the robot is the STCN75 digital temperature sensor from STMicroelectronics.
Main reasons why I have selected this sensor: I^2C interface, selectable bus address, low operating current (Typically 125 µA at 3.3V supply).
Because we have used the I^2C interface before, it's not a problem to attach another slave device to it. We just have to make sure that no address collision will occur. This criterion can be simply fulfilled because of the selectable bus address of the sensor.
The STCN75 provides 3 pins to select the interface address. All of the address selection pins can be simply grounded, with this step the temperature sensor address will be 0x90 for write operations and 0x91 for read operations –which is not in collision with the address of the compass (0x42 for write operation, 0x43 for read operation).
The sensor has also an open drain output which features a thermal alarm function. This feature is not used in this project.
The STCN75 stores the temperature as a 16-bit two's complement number, which can be read from the sensor at any time. The conversions are performed in the background so if the user reads the temperature during a conversion it won't affect the operation in progress.
Because none of the special features will be used in this project, we can skip the configuration step: studying the device datasheet [20] I have learned that after device startup a simple read operation will result as a reading from the temperature register.

Temperature data format
The left-most bit in the output data stream contains temperature polarity information for each conversion. If the sign bit is '0', the temperature is positive and if the sign bit is '1', the temperature is negative [20].

Temperature	Digital output	
	Binary	HEX
+125°C	0 1111 1010	0FA
+25°C	0 0011 0010	032
+0.5°C	0 0000 0001	001
0°C	0 0000 0000	000
-0.5°C	1 1111 1111	1FF
-25°C	1 1100 1110	1CE
-40°C	1 1011 0000	1B0
-55°C	1 1001 0010	192

Table 2-7: Relationship between temperature and digital output [20]

2.6 Proximity sensor

I have decided to equip the robot with an obstacle sensing device. For this purpose an infrared diode and an IR receiver module was chosen.
Selected parts: TSAL6200- infrared diode, TSOP4P38- IR receiver module for mid-range proximity sensors- both part from VISHAY semiconductors.

2.6.1 Working principle

The main working principle of this sensor is the following: with an infrared diode we are transmitting a light beam. This beam is reflected by obstacles located near to our robot. When an obstacle is close to the robot a strong reflection occurs (high percentage of emitted light is reflected), when an obstacle is far from the robot a weak reflection occurs. The amount of reflected light not only depends on obstacle distance, but also depends on the material of which the obstacle is made. Ambient light adds also an error to the results.

Our TSOP4P38 offers an easy way to assemble a simple proximity sensor. This sensor contains everything what we need: photo detector and conditioning circuits. The sensor outputs analog information about the reflection. On the sensor output we can measure pulses. The widths of these pulses are dependent on the absolute amount of reflected light from the infrared diode. When a strong reflection occurs the pulse width is longer, when a weak reflection occurs the pulse width is shorter (figure 2-5).
This sensor has one important requirement to work properly: the emitted light beam from the infrared diode must be modulated with 38 kHz carrier frequency.
Because the IR LED is placed next to the sensor, I had to separate them to avoid crosstalk.

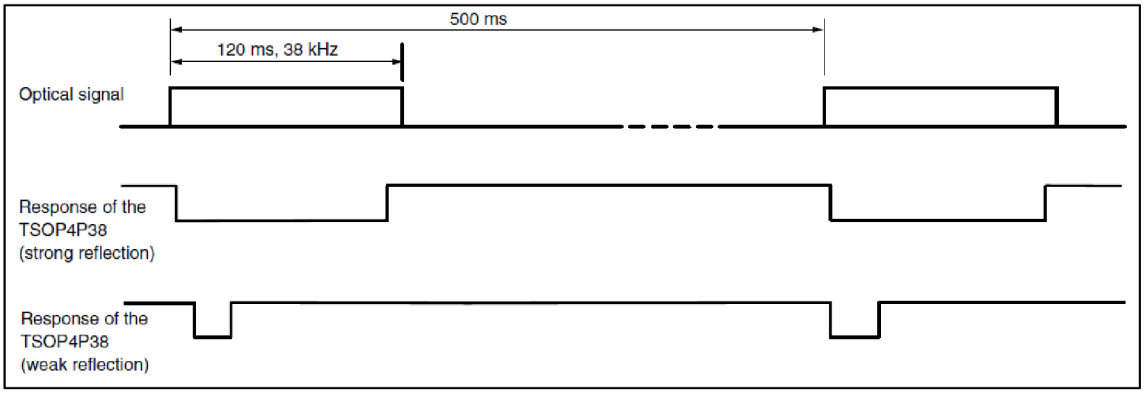


Figure 2-5: example of signal pattern [22]

2.6.2 Generating 38 kHz signal for the IR LED

There are many ways to generate the appropriate signal for the infrared diode. A have decided to generate this signal using the microcontroller what I selected before. This allows us to reduce the hardware costs. With the microcontroller the easiest way to generate this signal is using one of the integrated "output compare" modules. The output compare module can be configured to operate in PWM mode. Pulse width modulation (PWM) is a technique that allows us changing the duty cycle of a square wave while the period maintained constant (duty cycle = high pulse width divided by the period).

I will explain how the output compare module generates the needed signal. For the explanation I will use the following figure:

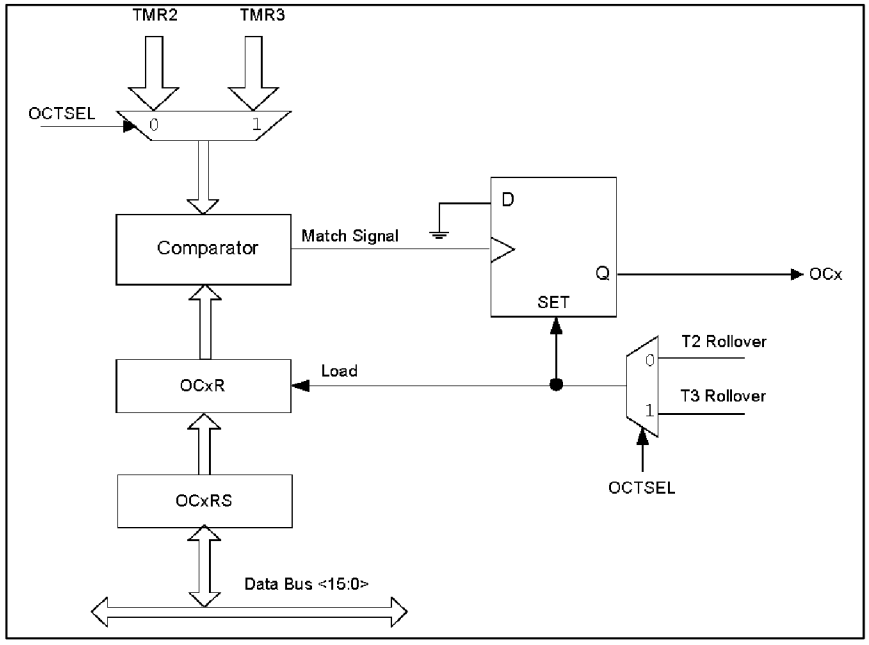


Figure 2-6: PWM mode block diagram[12]

As we see in the picture, we have to associate a timer with the output compare module before use. The output signal period will be equal with the timer module period. After configuring the timer, we just have to load the OCxRS register with a predefined value. The value in this register will control the high pulse width in the output signal.
The whole process can be described as follows:

- At the end of each timer period the output signal will go high, and in the same moment the OCxRS value is loaded into OCxR (The OCxR register can't be written directly in PWM mode)
- The comparator compares the timer value with the value in register OCxR
- When the timer value equals with the value in OCxR the output signal will go low.

It is obvious that the value in the OCxR register controls the output signal duty cycle. With adjusting the duty cycle we can control the power consumption of the infrared diode. Higher duty cycle means more forward current on the IR LED.

The following figure illustrates the signal generation in different modes of operation:

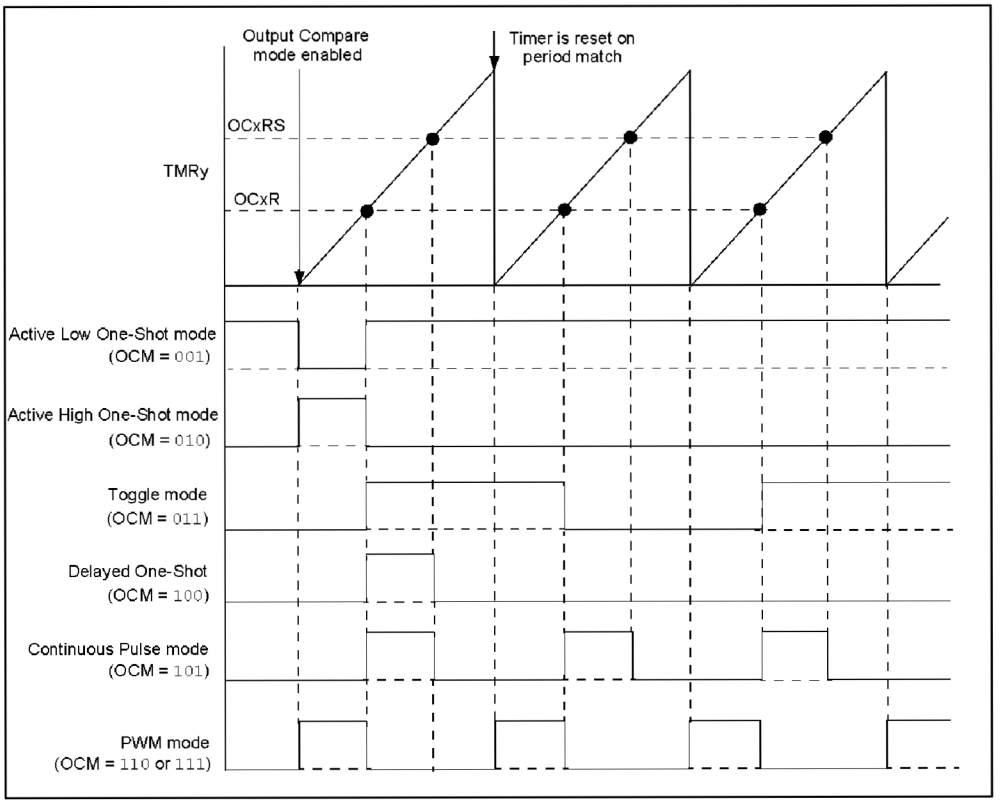


Figure 2-7: Output compare operation [12]

2.6.3 Measuring the pulse width

To obtain useful information from the sensor output we have to measure the pulse width of this signal. This task can be done in some different ways. The best way to measure the pulse width of a given signal is using one of the input capture modules in the microcontroller. The input capture allows us to measure precisely the pulse width. When this measurement is accomplished, we can calculate the distance between the robot and the obstacle (However, to get accurate results the sensor should be experimentally calibrated and tested- which is not the aim of this project).

The input capture module:

The input capture module function is to capture a timer value when an event occurs on a predefined input pin (event = change in logic level). Like in the case of the output compare module we can select one of the two offered timer modules as the time base. The input capture module can be configured in different modes (capture on rising edge, capture on falling edge, prescaler mode…), for the pulse width measurement the edge detect mode is the best. The input capture module has a four-level FIFO buffer; the user program can read the captured timer value from this buffer. A flag bit indicates when the buffer is empty or is containing captured values. The user program doesn't have to poll the input buffer because the input capture module will generate an interrupt whenever a capture event occurs. The next figure shows when the input capture module generates an interrupt (in different modes of operation):

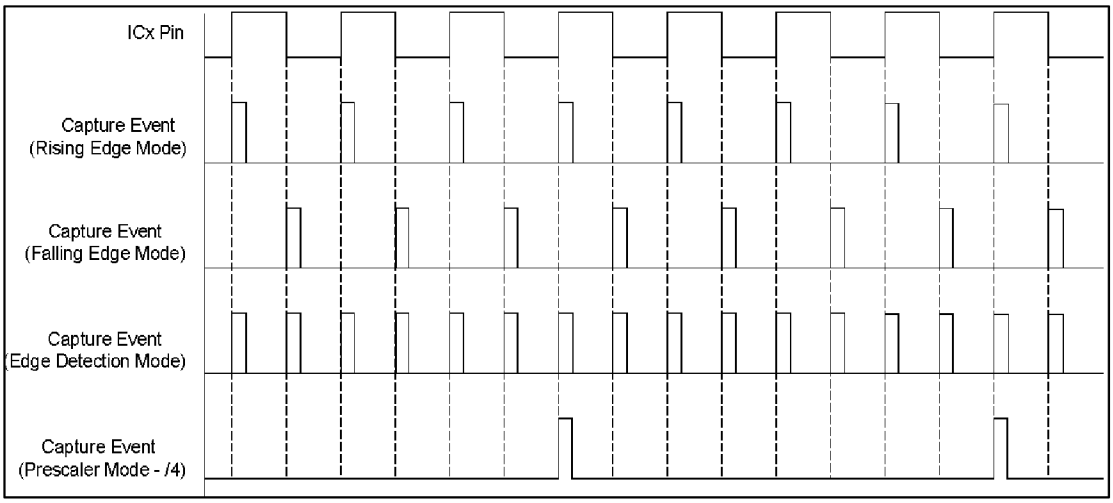


Figure 2-8: Input Capture Event Generation [11]

Since we don't know exactly how long will be the pulse width, the timer can overflow before the second capture event. Because this, we have to calculate with the timer overflow times. The exact equation for this calculus can be found in [8].

2.7 LED driver

When I finished adding the sensors to the robot, i had in mind the following question: Why not use the robot at night, or in poor light conditions?!
So I decided to equip the robot with some light emitting diodes. This decision popped up some further problems/questions: how many LEDs to be used, low-power or high-power LEDs- and if high power LEDs how they will be controlled (LED dimming nowadays can be considered as a standard). Because the robot will be supplied from battery I had to choose an efficient method to control these LEDs.

Chosen LEDs: Luxeon Star LED from Philips with green color and 1W of power. Quantity: 4 (two for the front side, two for the back side).

Because of the battery supply I decided to use a buck converter to supply the LEDs.
Chosen buck converter: LM3407 (350mA constant current output floating buck switching converter for high power LEDs).

Advantages of the selected buck converter:

- Ensures constant current to the LEDs (voltage drop in the supply don't affects the output current)
- The number of LEDs can be increased without a problem (max. 7 LEDs)
- The actual current to the LEDs can be set with a single resistor
- PWM dimming available

At first I wanted 1 buck converter / diode, but there was not enough remappable I/O pin in the microcontroller for PWM dimming- so I decided to assemble one control circuit with buck converter for the front LEDs and one control circuit with buck converter for the rear LEDs. This configuration allows us to separately switch ON/OFF the two set of LEDs. The dimming is done by only one PWM signal, so the front and the rear LEDs will be at same power.

2.8 Dynamixel RX-64

The RX-64 actuator is a compact and smart actuator which contains a gear reducer, a precision DC motor (MAXON motor) and a control circuitry. The RX-64 is using serial link for communication realizing the RS-485 standard. The standard allows operating multiple actuators in a single link, using "daisy chain" connection.
The actuator can be operated with a wide range of supply voltage (12-21 V DC), but in accordance with datasheet [3] it should be 18V DC.
Because our microcontroller is not supporting the RS485 standard, we have to attach between the actuator and the microcontroller an UART/RS485 converter:

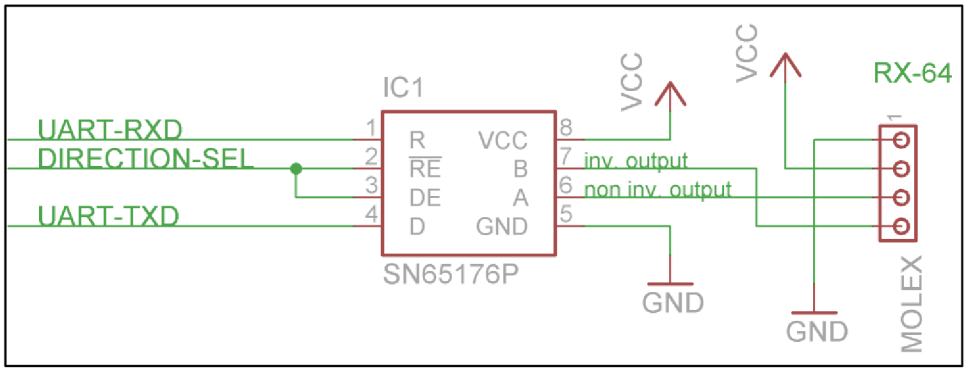


Figure 2-9: UART/RS485 converter

The direction of data signals are specified by the state of RE, DE pins (connected together, from now will be referred as "DIRECTION-SEL").
When the DIRECTION-SEL is at logic high: TxD -> B,A
When the DIRECTION-SEL is at logic low: B,A -> RxD

2.8.1 Communication with the RX-64

Communication between the actuator and the MCU is realized as follows:
The MCU sends an instruction packet addressed to the desired actuator, and then pulls the DIRECTION-SEL to logic low – allowing the actuator to respond with a status packet. The actuator performs the required instruction, but only if the following conditions are met:

- It is a valid instruction
- The packet contains a valid actuator ID
- There is a valid checksum at the end of the packet

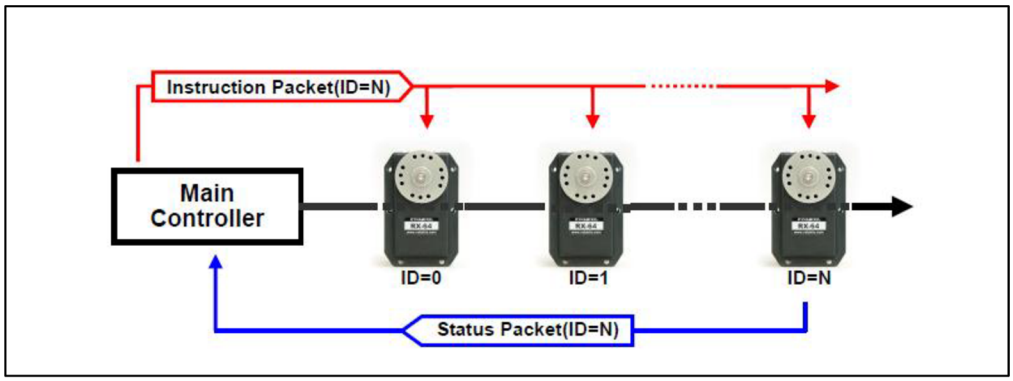


Figure 2-10: MCU<-> RX-64 communication [3]

Before using any of the actuators, the ID must be set. Multiple actuators with the same ID will cause communication problems (packet collision caused by the returned status packet).

ID's are ranging from 0x00 to 0xFD (max. 254 actuator). ID 0xFE is reserved for broadcasting. A packet sent with the broadcast address will not return any status packets.

2.8.2 Using the actuators

Instruction packet format

0xFF	0xFF	ID	Length	Instruction	Param.1	…	Param. N	Checksum

Table 2-8: Dynamixel instruction packet[3]

The meaning of each packet byte definition is as the following:

0xFF, 0xFF	-indicates the start of an incoming packet
ID	-the unique ID of a Dynamixel unit (0xFE = broadcast ID)
Length	-compute as: number of parameters + 2
Instruction	-instruction to perform (possible instructions are discussed later)
Parameter N	-additional information to the instruction
Checksum	-compute as: ~(ID + Length + Instruction + Param.1 + Param. N) (~ represents logical NOT)

The following instructions are available:
Ping, Read Data, Write Data, Register Write, Action, Reset, Sync Write.

In this project only the "write data" instruction is used, so we can discuss it in details:
-The main function of this instruction is to write data into the control table.
-Number of parameters: 1 or 2, depending on the address location in the control table.
-Instruction: 0x03
-First parameter: starting address of the location where the data is to be written.

The control table:
Each Dynamixel unit contains a control table. The control table controls the actuator operation. We can set lots of parameters in the control table, but in this project the following parameters will do the work: CW angle limit, CCW angle limit, Moving speed. We have to set the CW and CCW angle limits to zero, to achieve endless turn.
Then we only need to set the moving speed in the control table to get the robot moving.
(The whole control table can be found in [3]).

The first step what we have to do with the dynamixel actuators is to change their identification number (ID). The factory default ID is: 0x01. With precaution, I decided to reset all the actuators to their factory settings. This step ensures that all actuators will have the same settings.
For the broadcast reset I had used the following instruction packet:

0xFF	0xFF	0xFE	0x02	0x06	0xF9

Table 2-9: Dynamixel instruction packet- RESET

After resetting I have assigned an ID to all dynamixel actuators. The next step is to set the CW and CCW angle limits (as mentioned before) to zero.
The last step is to set the moving speed. I will describe this step in more detail, because we are not just setting the moving speed, but the direction as well.

Setting the dynamixel actuator moving speed and the direction of the movement:

BIT	15~11	10	9	8	7	6	5	4	3	2	1	0
Value	0	Turn direction	Speed value									

Table 2-10: Part of the dynamixel control table- Moving speed setting [3]

In the previous table we can see that for controlling the speed we have only 10 bits. This means that we can achieve 1024 different speed settings. In the other hand, this amount of available speed setting is unnecessary. In the practice small changes in the moving speed are almost unnoticeable.
We can set the moving direction of the actuator by setting/clearing the tenth bit at the proper memory address of the control table. To increase the actuator lifetime, the turn direction should be changed at low speeds.

2.9 Putting the pieces together

2.9.1 Mechanical solutions

During the development of electronics I decided that the robot will have a plastic body. This will ensure the minimal weight of the body. I wanted a very simple solution so I bought a simple plastic box. To this I mounted the 4 dynamixel actuator units from outside.

2.9.2 Electronics

Nowadays a lot of part is available only in SMD package. This was the primary reason for me to pick almost all of the parts with SMD package. The secondary reason to do so is that I wanted to produce a small PCB what will easily fit to the plastic box.
This decision what I made according the SMD packages raised more problem. For development I always used a solderless breadboard. Of course the SMD packages are not compatible with that. Because of this I had 2 choices: to make an adapter to the breadboard for all SMD components or to design & realize the whole circuit without previous testing.
In the end I made some adapters of course (Xbee modules for example), but in the case of the LED driver, RS485 converter, thermometer I tried out the functionality only when the boards where finished.
I was lucky because all of the parts worked fine from the first “power-up”.

2.9.3 Schematic diagrams and PCB designs

In the beginning of the schematic and PCB drawing I experienced some problems, because just in that time I changed to new- unfamiliar- design software. As the time passed this initial problems disappeared. During the design I had to draw some of the schematic symbols and PCB footprints myself (some of the parts where too new, that the design software does not included the specific symbol & footprint yet).
I made all of the PCBs to one-sided. This allowed me to manufacture the boards at home.
The word “SMD” changes a lot of things both in the PCB design and PCB assembly process. The final board of course will be smaller, but the manufacturing process can be challenging. Not to mention the repairing difficulties. Anyway, I didn’t mind that I chose SMD parts- the technology evolves fast; we have to do the same.

3 SOFTWARE

3.1 The software in the microcontroller

Because this was my first time to work with a 16-bit microcontroller I had to read a lot of datasheets and user manuals before starting the programming itself. The 16-bit architecture allowed me to write the program in the C language and not in the assembler language. This has some advantages and disadvantages. The main advantage in using the C language is the code clarity; the main disadvantage is the compiled code size.

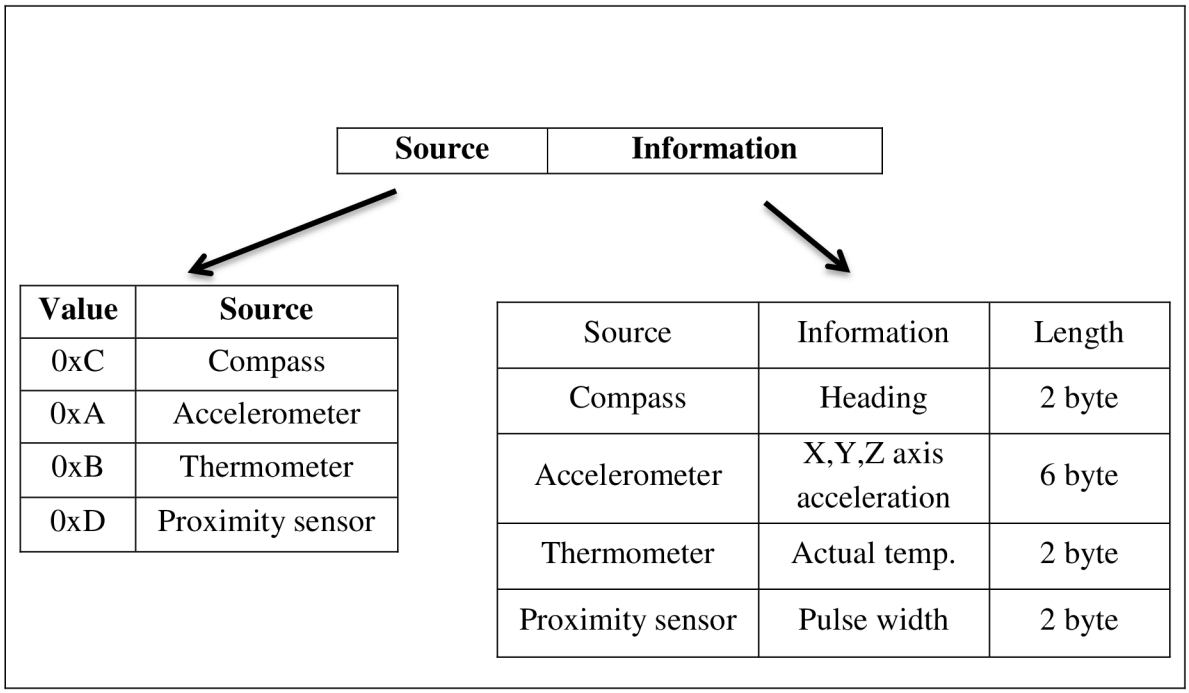
The program in the microcontroller has the following responsibilities:

- Self-configuration: enabling all the necessary peripheral modules
- Communication with the control program (PC) over the Xbee modules. This includes packet assembly for transmitting, and packet processing (received packet).
- PWM signal generation for the IR LED (38 kHz carrier)
- PWM signal generation for the LED dimming (power LEDs)
- Compass configuration, reading the information about the heading
- Accelerometer configuration, acquiring the acceleration of each axis from the sensor.
- Reading the actual temperature from the thermometer
- Measuring the pulse widths on the TSOP4P38 output
- Controlling the dynamixel actuators- speed and direction, all actuators independently.

In *Figure 3-1* we can see how the program works. The flowchart is simplified, some details are not included.
We can see from the flowchart that the main program configures the oscillator, and then maps all the peripherals to the predefined IO pins. The mapping process has to be done before using these peripherals. The peripherals can be remapped during program flow, but before remapping the program should disable the actual module to prevent unexpected operation.
After these steps the program configures every required module (operation mode, speed, interrupts …etc.). When that is finished, the program toggles ON/OFF the power LEDs- this indicates to the user that the microcontroller is finished with the initialization steps. When finished with this startup "animation", the program enters to an endless loop. In this endless loop the program acquires information from all sensors, and then sends the acquired data to the Xbee module. Because the Xbee is configured for API operation,

the microcontroller has to assemble the “transmit request packet” (described in previous chapters). If the packet is assembled correctly, the Xbee will transmit it to the destination. The destination device (second Xbee) is connected to a PC.

Because each sensor formats its output data in a different way, the program has to process these differences. It’s obvious that the transmit request packet will carry different information for each sensor. Because of this, the program in the microcontroller has to mark these different packets – allowing the software on the PC to recognize from which sensor the information is.
So I had to choose a method, how to mark these packets. The following table shows how the packets are marked:



Value	Source
0xC	Compass
0xA	Accelerometer
0xB	Thermometer
0xD	Proximity sensor

Source	Information	Length
Compass	Heading	2 byte
Accelerometer	X,Y,Z axis acceleration	6 byte
Thermometer	Actual temp.	2 byte
Proximity sensor	Pulse width	2 byte

Table 3-1: Detailed information of the transmitted packet

Of course, this is not the whole “transmit request” packet, just a part of it- the useful information.

While the program runs in the endless cycle, the flow can be interrupted by the hardware. More precisely the following modules can interrupt the main program flow:
UART modules, some of the TIMER modules, the Output Compare module and the Input capture module. When one of the two UART modules interrupts the main program flow means, that there is information in the input buffer that needs to be read. The first UART is connected directly to the Xbee module, the second to the dynamixel actuators trough an RS485 driver. When an interrupt comes from the TIMER1 module, the program controls if there is a whole packet received by the UARTs or not. If there is a whole packet available, the program processes it.

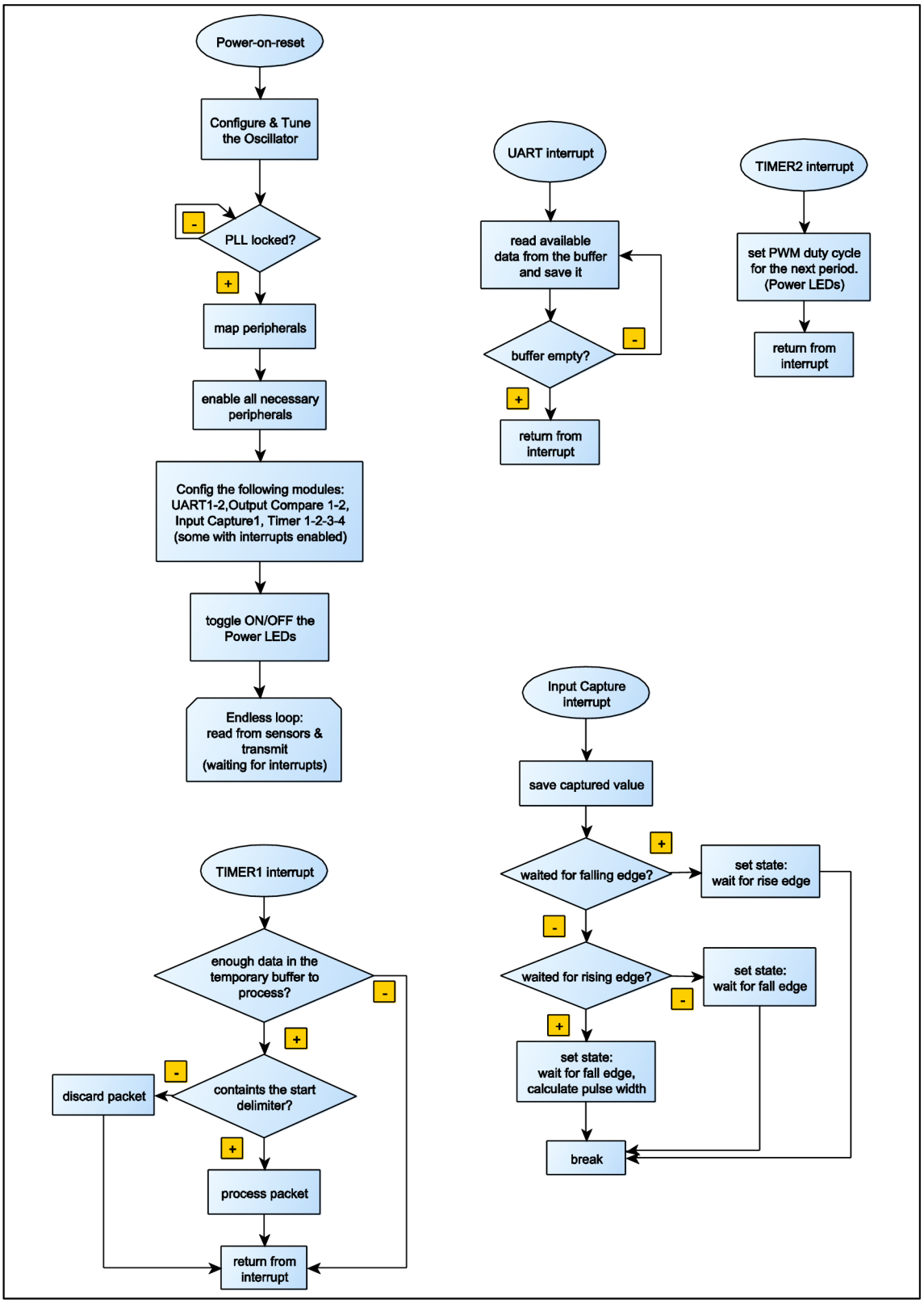


Figure 3-1: MCU program- simplified flow chart

3.2 The software for the PC

The user interface to the robot is written in the C# language. In the beginning, I decided to create the software as an MDI application, so in the main window (parent) the user will be able to open more than one secondary window (child). This allows the user to use more than one feature at a time.

Let's discuss the user interface in details:

3.2.1 The main –parent– window

In the following figure we can see the important parts of the parent window:

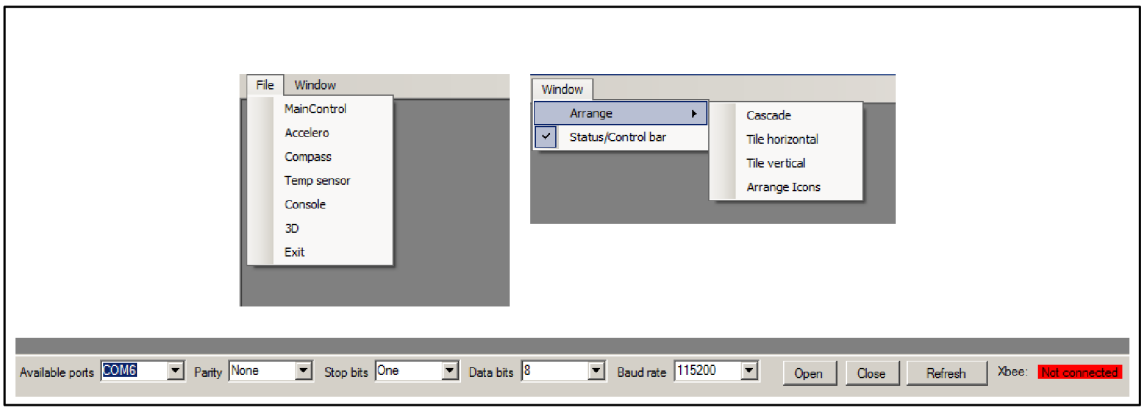


Figure 3-2: Important parts of the parent window

Under the File menu the user can open any of the child windows. The Window menu allows some simple arrangements of these opened windows. The Window menu also allows the user to hide the "control tab" which is located at the bottom side of the main window.
In the control tab the user can select one of the offered serial ports, and select desired parameters before opening. When the serial port opening was successful the program will disable these controls to avoid further modification. When the port is closed, the controls will be enabled again. Because another program can use our selected serial port, we have to make sure the program won't crash during the opening process.

3.2.2 The „Compass“ child window

When the user opens the Compass window from the File menu the following window will appear:

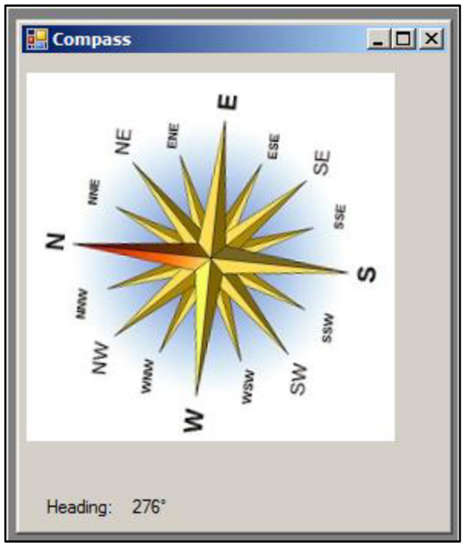


Figure 3-3: The Compass child window

When this window is opened, the program will draw and rotate a picture according to heading information. The concrete heading is also displayed (in degrees) at the bottom of the window. The picture is downloaded from:
http://projectgroundswell.com/2010/04/23/a-man-and-his-bicycle/600px-compass_rose_english_north-svg/

3.2.3 The "Accelerometer" child window

In the following figure (Figure 3-4) we can see how the data from the accelerometer is displayed.

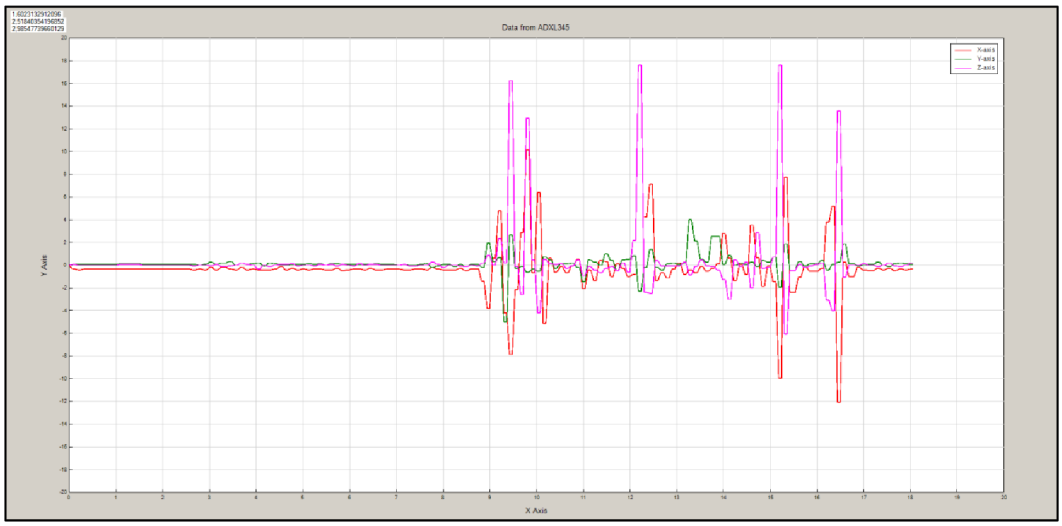


Figure 3-4: The Accelerometer child window

The ADXL345 is a 3-axis accelerometer, so we are displaying here 3 curves in the graph- each with a different color. In this graph the acceleration is displayed (with informative character), however we can calculate the inclination from the acceleration. For this calculus we can use the following formulas[1]:

X-axis inclination:

$$\theta = \tan^{-1}\left(\frac{A_{x,out}}{\sqrt{A_{y,out}^2+A_{z,out}^2}}\right) \quad (1.)$$

Y-axis inclination:

$$\psi = \tan^{-1}\left(\frac{A_{y,out}}{\sqrt{A_{x,out}^2+A_{z,out}^2}}\right) \quad (2.)$$

Z-axis inclination:

$$\phi = \tan^{-1}\left(\frac{\sqrt{A_{y,out}^2+A_{z,out}^2}}{A_{z,out}}\right) \quad (3.)$$

Inclination values are displayed at the top-left corner of the window; however the practical use of these values will be discussed later.

[1] Formulas where copied from [25]

3.2.4 The "Console" child window

In the Console window (Figure 3-5) we can control the incoming transmission. At first sight this is not an important thing to do, but when we are developing a program which processes some incoming data, that's a basic thing that in some way we're displaying the original incoming transmission. When it's done, we can start to develop the processing algorithms. In *Figure 3-5* we can clearly see all the incoming data from the robot: we can see at the beginning of each packet there is the "start delimiter" which is followed by the packet length (2 bytes), packet type, 64-bit source address, 16-bit source network address and the receive options. Just after these information comes the most important part of the message: information from the sensors. As discussed before, the information from each sensor is marked with a character before the actual information.

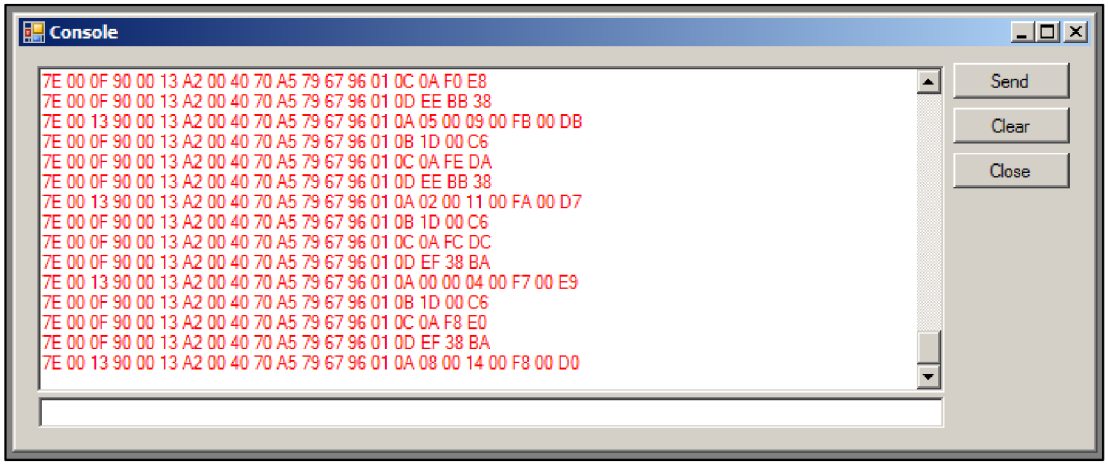


Figure 3-5: The Compass child window

The reader may have noticed that there is a send button in the top-right corner of this window. This button is not yet used; it was placed there for future use together with the textbox (bottom part of the window) - to allow the user sending specific messages to the robot. This function will be implemented when someone will develop further this program/project.

3.2.5 The “3D” child window

During the development I decided to display the information from the accelerometer not just in a simple graph, but also in a 3-Dimensional form. Because we are calculated the inclination before, this is the best moment to use this data.
Figure 3-6 shows the practical implementation of the calculated inclination.

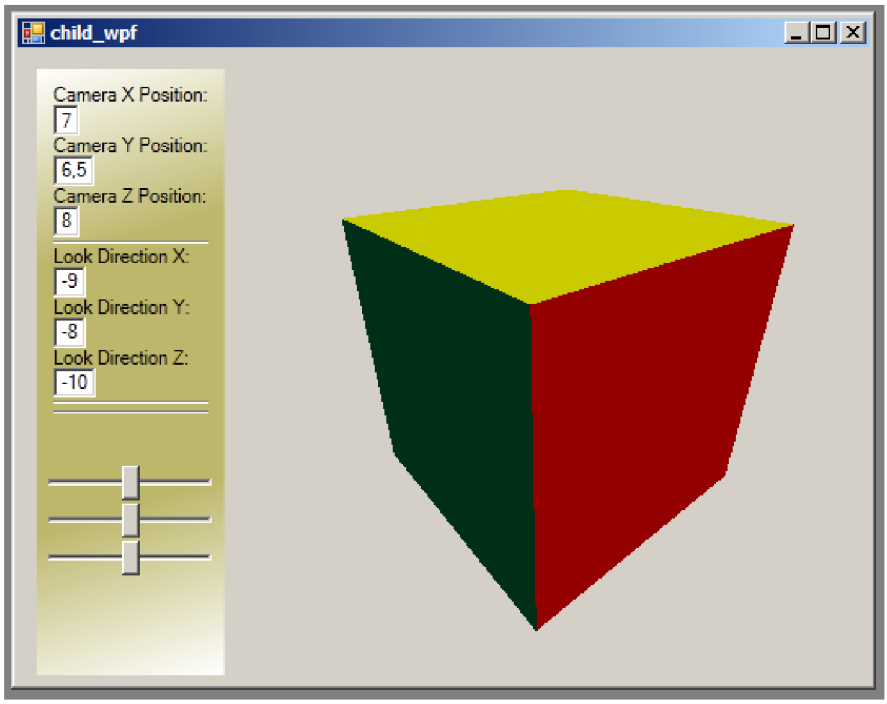


Figure 3-6:The 3D child window

With this part of the software I had some problems, because in the beginning a created this program as a “WindowsForm” application. This meant that drawing and rotating 3D objects will be a big problem. I had 2 choices: re-create the whole program as a WPF application (which offers everything what we need for 3D drawing & rotating), or to find some other way to do the drawing. A did a lot of research on the internet, when I found out that a WPF application can be integrated to an existing WindowsForm application.
The tutorial what I used to create this 3D animation is available at [26]. The source code of this tutorial was modified by me. The original source code of this tutorial contained only the rendering of a 3D-cube (Figure 3-6), so I had to modify it to add rotation availability. The rotation is done by object transformations.
This 3D animation is using the acquired data from the compass and the accelerometer. Because the compass is not tilt compensated, the heading information won’t be valid if the robot is tilted. Therefore if the inclination exceeds a predefined value, the program

ignores the heading information from the compass, and leaves the last value of rotation displayed. When the inclination drops below the predefined limit, the program will again use the heading value from the compass for the animation.

3.2.6 The “main control” child window

There is only one thing what we haven’t discussed yet, and that’s the actual control of the robot.
I had in mind that I should add interactive control ability to the user program. So I decided that the robot will be “guided” by the help of the mouse. To allow this functionality I added a “picture –box” control to the Main Control window (Figure 3-7), in this picture box will be the user able to control the robot movement. When the user clicks with the mouse (and holds the button) inside the picture box, a line will appear inside the picture box connecting the cursor and the bottom-center side of the picture box together. From the length of this line and its position (compared to the picture box boarders) the program calculates the necessary speed for each dynamixel actuator. The length of the line gives the total speed which is then divided between the left-side and right-side servos. When the left-side servos are turning faster the robot will turn right, in opposite case the robot will turn left. When the user releases the button, the robot stops.
The program distinguishes the left-button click from the right-button click. This allows moving direction change.

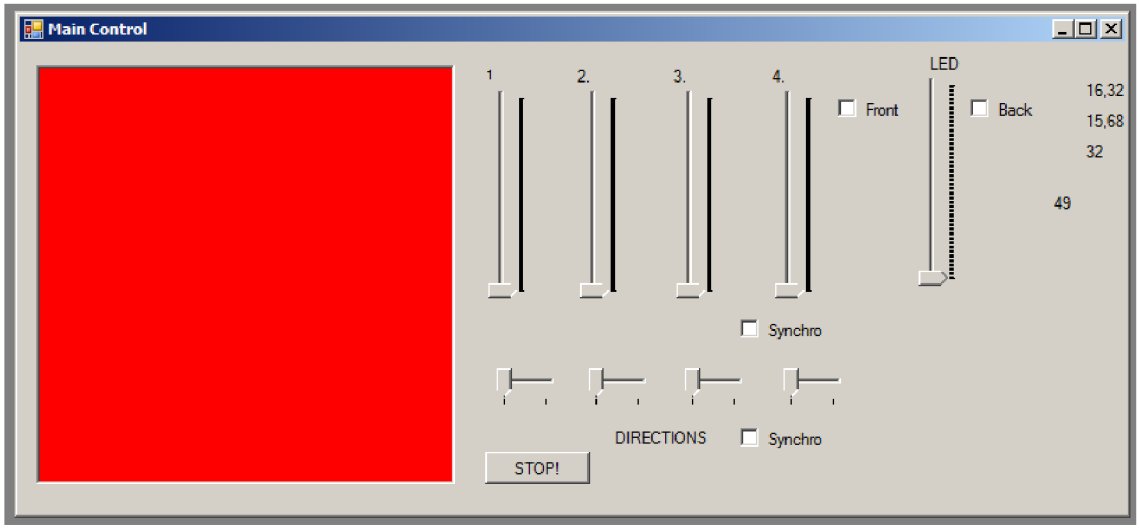


Figure 3-7: Main Control window

In some cases the situation requires to operate the servos separately. The program allows that too, for that reason I added “slider” controls. The moving speed and direction can be controlled with these sliders for each servo. The last slider serves for LED dimming control. With that we can control the brightness of the power LEDs.

4 SUMMARY AND CONCLUSIONS

I have successfully designed and realized a remote controlled robot. If we take a close look at the sensors, the robot contains the following: a compass, a 3-axis digital accelerometer, a thermometer and a mid-range proximity sensor. Almost all of them are working without a single problem. I experienced some problems with the proximity sensor. During the development I corrected one mistake in the software, but the problem with this sensor still remained. In my opinion there is not enough isolation between the IR-diode and the sensor to avoid crosstalk. Because the lack of time I haven't corrected this error, so the control program just displays a meaningless value from the sensor – which should be of course the pulse width at the sensor output.

As the heart of my robot I used a 16-bit microcontroller from Microchip Technology. It handles all the attached sensors and the communication trough the Xbee module too. Communication over these Xbee modules can be an interesting challenge. In the beginning I tried the "transparent" operation, which means that the Xbee module simply replaces a serial link. This method proved to be an ineffective one in this project. After that I decided to use the second option- the API mode. This meant that for success communication I had program the microcontroller to assemble the corresponding packet (in API mode the Xbee module accepts only packets-in a predefined format). At this point of development I experienced the following problem: the wireless communication speed was very low. After 2 days of debugging I found the problem. Every packet which has to be transmitted to another device (router, end device…) contains the destination address. But for the coordinator there is a reserved address. So I decided to use this reserved address instead of the actual – as the packet destination. However I was not aware of the following: finding the coordinator on a network takes some time (The Xbee automatically searches for the coordinator). After changing this reserved address to the actual, the problem was solved.

The main control software for the PC is realized in the C# language. This software provides a simple user interface to control the robot. An MDI application was created to allow more comfort to the user. This software processes all information coming from the robot. After processing the program displays the useful information to the user. The interface allows the user controlling the robot interactively with the mouse. All dynamixel actuators can be controlled separately if desired.

My robot isn't perfect; there are some bugs in the software which needs to be corrected in the future. I hope that I will have an opportunity to continue the development of this robot, because there are some features that I wanted to try- both in software and hardware.

5 GLOSSARY

PIC	Peripheral Interface Controller
MCU	Microcontroller Unit
I^2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver Transmitter
PCB	Printed Circuit Board
EEPROM	Electrically Erasable Programmable Read-only Memory
MDI	Multiple Document Interface
API	Application Programming Interface
PLL	Phase Locked Loop
WPF	Windows Presentation Foundation
SMD	Surface Mount Device

6 REFERENCES AND BIBLIOGRAPHY

[1] Microchip [online]. Datasheet *PIC24HJ32GP302/304, PIC24HJ64GPX02/X04 and PIC24HJ128GPX02/X04*. Microchip Technology Inc. [cit. 21.05.2012]. Available at www: http://ww1.microchip.com/downloads/en/DeviceDoc/70293F.pdf

[2] Microchip [online]. User's guide *MPLAB® C30 C COMPILER*. Microchip Technology, Inc. [cit. 21.05.2012]. Available at www: http://ww1.microchip.com/downloads/en/DeviceDoc/MPLAB%20C30%20UG_DS-51284f.pdf

[3] Megarobot [online]. User's manual *Dynamixel RX-64*. ROBOTIS. [cit. 21.05.2012]. Available at www: http://www.megarobot.net/cj/manualy/robotis/RX64_aj.pdf

[4] Analog Devices [online]. Datasheet *ADXL345 Digital 3-Axis Accelerometer*. Analog Devices, Inc. [cit. 21.05.2012]. Available at www: http://www.analog.com/static/imported-files/data_sheets/ADXL345.pdf

[5] Honeywell [online]. Datasheet *2-Axis Compass with Algorithms HMC6352*. Honeywell International, Inc. [cit. 21.05.2012]. Available at www: http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Missiles-Munitions/HMC6352.pdf

[6] Digi International [online]. Datasheet *Xbee®/Xbee-PRO® ZB RF Modules*. Digi International, Inc. [cit. 21.05.2012]. Available at www: ftp://ftp1.digi.com/support/documentation/90000976_H.pdf

[7] SHARP, J. *Microsoft Visual C# 2010 Step by Step*, Microsoft press, 2010

[8] Robert Reese, J.W. Bruce, Bryan A. Jones Microcontrollers- From Assembly Language to C Using the PIC24 Family, Course Technology, 2009

[9] Microchip [online]. 16-bit MCU and DSC Programmer's Reference Manual, Microchip Technology, Inc. [cit. 21.05.2012]. Available at www: http://ww1.microchip.com/downloads/en/devicedoc/prog_ref_manual.pdf

[10] Microchip [online]. 16-bit Language Tools Libraries, Microchip Technology, Inc. [cit. 21.05.2012]. Available at www: http://ww1.microchip.com/downloads/en/DeviceDoc/51456G.pdf

[11] Microchip [online]. Reference manual *Section 12. Input Capture*, Microchip Technology, Inc. [cit. 21.05.2012]. Available at www: http://ww1.microchip.com/downloads/en/DeviceDoc/70198D.pdf

[12] Microchip [online]. Reference manual *Section 13. Output Compare*, Microchip Technology, Inc. [cit. 21.05.2012]. Available at www: http://ww1.microchip.com/downloads/en/DeviceDoc/70209A.pdf

[13] Microchip [online]. Reference manual *Section 18. Serial Peripheral Interface*, Microchip Technology, Inc. [cit. 21.05.2012]. Available at www: http://ww1.microchip.com/downloads/en/DeviceDoc/70206b.pdf

[14] Microchip [online]. Reference manual *Section 19. Inter-Integrated Circuit,* Microchip Technology, Inc. [cit. 21.05.2012]. Available at www: http://ww1.microchip.com/downloads/en/DeviceDoc/70195B.pdf

[15] Microchip [online]. Reference manual *Section 17. UART,* Microchip Technology, Inc. [cit. 21.05.2012]. Available at www: http://ww1.microchip.com/downloads/en/DeviceDoc/70188C.pdf

[16] Microchip [online]. Reference manual *Section 11. Timers*, Microchip Technology, Inc. [cit. 21.05.2012]. Available at www: http://ww1.microchip.com/downloads/en/DeviceDoc/70205B.pdf

[17] Microchip [online]. Reference manual *Section 6. Interrupts*, Microchip Technology, Inc. [cit. 21.05.2012]. Available at www: http://ww1.microchip.com/downloads/en/DeviceDoc/70184b.pdf

[18] Microchip [online]. Reference manual *Section 7. Oscillator*, Microchip Technology, Inc. [cit. 21.05.2012]. Available at www: http://ww1.microchip.com/downloads/en/DeviceDoc/70186E.pdf

[19] Jack Xu. Practical WPF Charts and Graphics, Apress, 2009

[20] STMicroelectronics [online]. Datasheet *STCN75- Digital temperature sensor and thermal watchdog*. STMicroelectronics. [cit. 21.05.2012]. Available at www: http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00153589.pdf

[21] Vishay Intertechnology [online]. Datasheet *TSAL6200 High Power Infrared Emitting Diode*. Vishay Intertechnology, Inc. [cit. 21.05.2012]. Available at www: http://www.vishay.com/docs/81010/tsal6200.pdf

[22] Vishay Intertechnology [online]. Datasheet TSOP4P38 IR Receiver Modules for Mid Range Proximity Sensors. Vishay Intertechnology, Inc. [cit. 21.05.2012]. Available at www: http://www.vishay.com/docs/83305/tsop4p.pdf

[23] Texas Instruments [online]. Datasheet *LM3407 350mA, Constant Current Output Floating Buck Switching Converter for High Power LEDs*. National Semiconductor [cit. 21.05.2012]. Available at www: http://www.ti.com/lit/ds/symlink/lm3407.pdf

[24] H.R. Everett, Sensors for Mobile Robots, A K Peters/CRC Press (July 15, 1995)

[25] Analog Devices [online]. Application Note *Using an Accelerometer for Inclination Sensing*. Analog Devices, Inc. [cit. 21.05.2012]. Available at www: http://www.analog.com/static/imported-files/application_notes/AN-1057.pdf

[26] Window Presentation Foundation (WPF) 3D Tutorial [online]. [cit. 21.05.2012]. Available at www: http://kindohm.com/technical/WPF3DTutorial.htm

LIST OF ATTACHMENTS:

Attachment 1: Schematic diagrams & PCB designs
Attachment 2: Bill of materials
Attachment 3: Pictures of the finished robot

CONTENT OF THE SUPPLIED CD:

- Microcontroller source code
- Main control program- Visual Studio Project
- Bachelor thesis Remote Controlled 4-wheel robot
- Schematic diagrams & PCB designs (in PDF format)

Attachment 1: Schematic diagrams & Printed circuit board designs

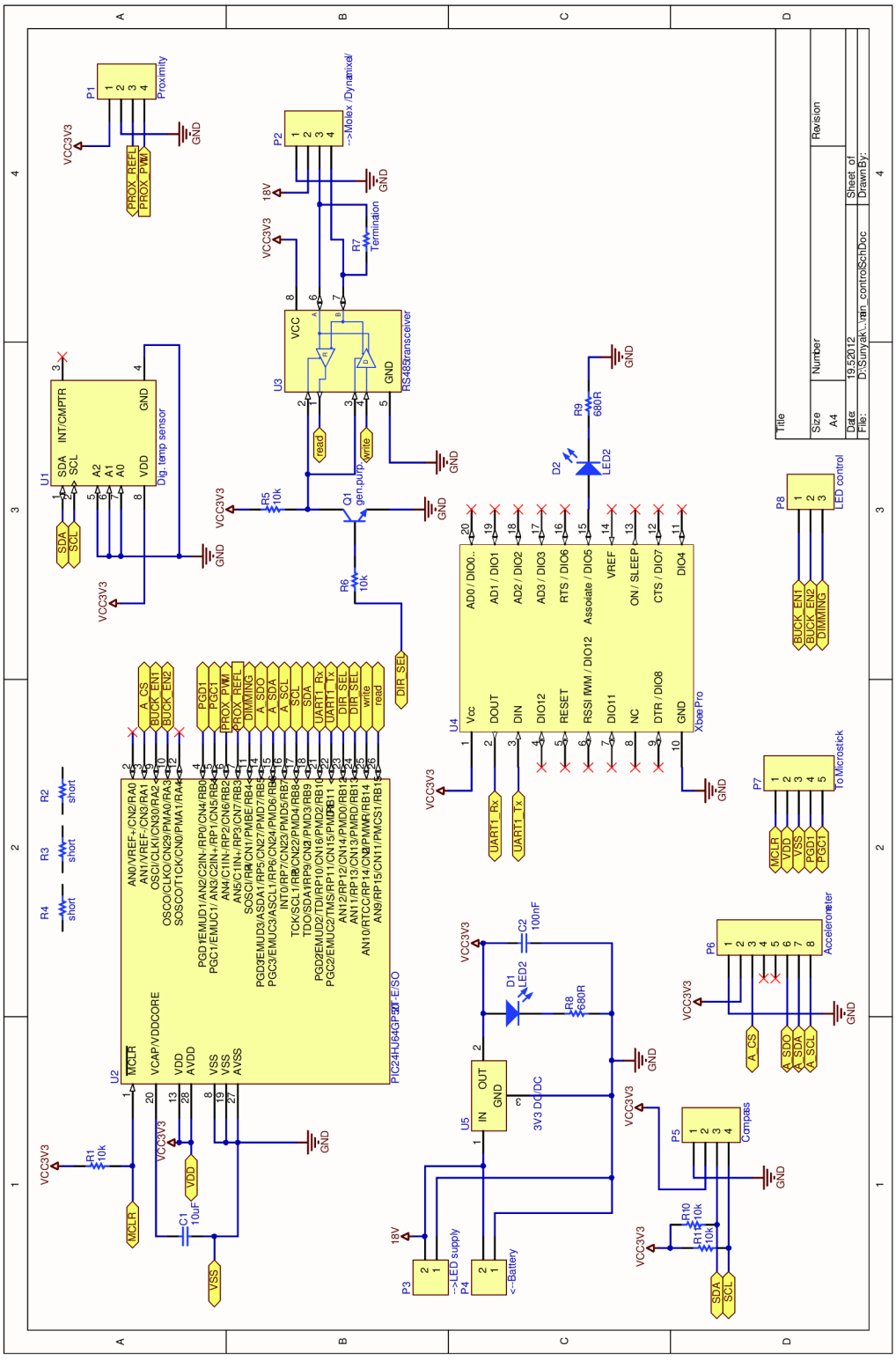


Figure 6-1: Schematic diagram- Main Control Circuit

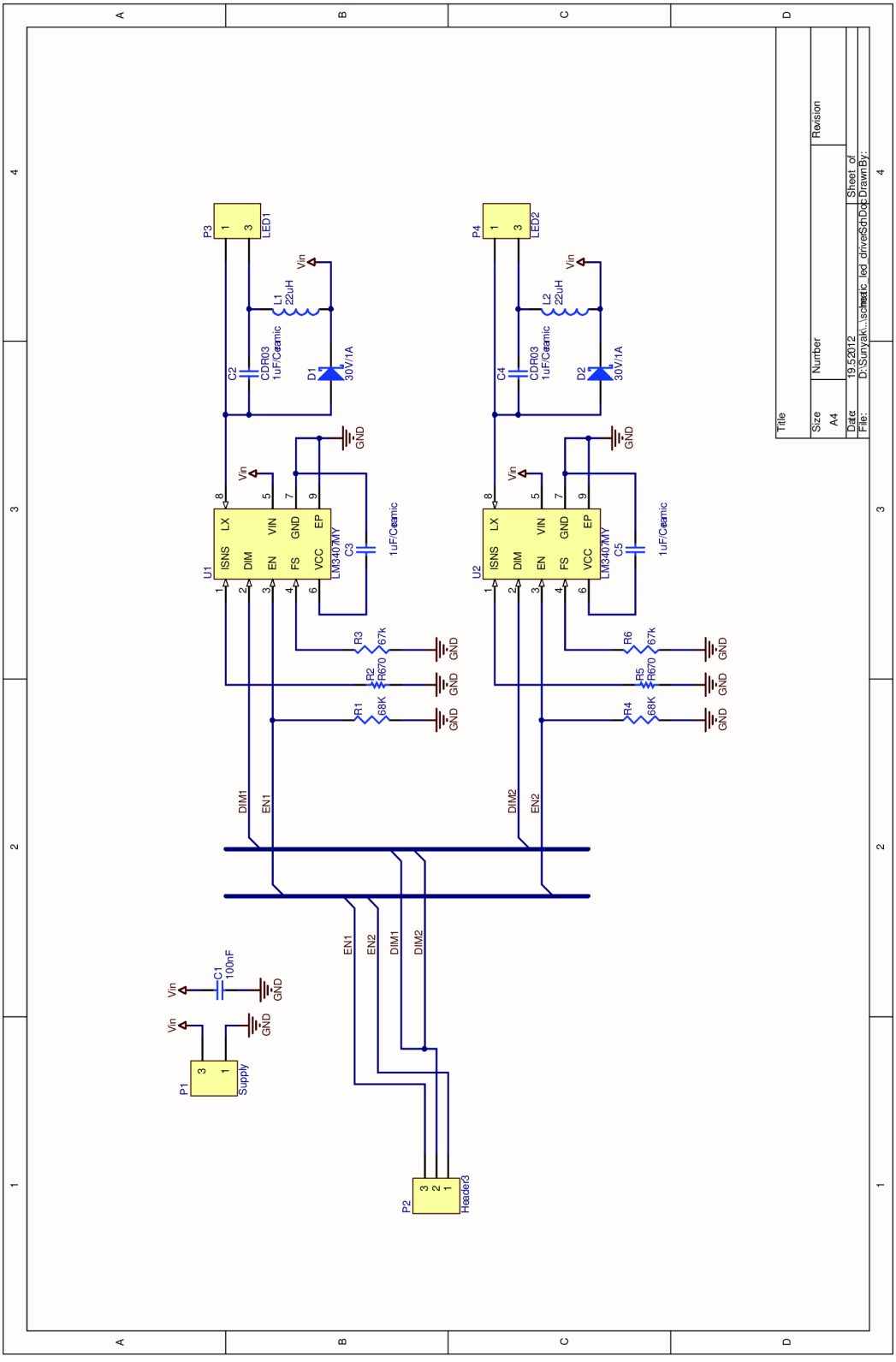


Figure 6-2: Schematic diagram- LED driver

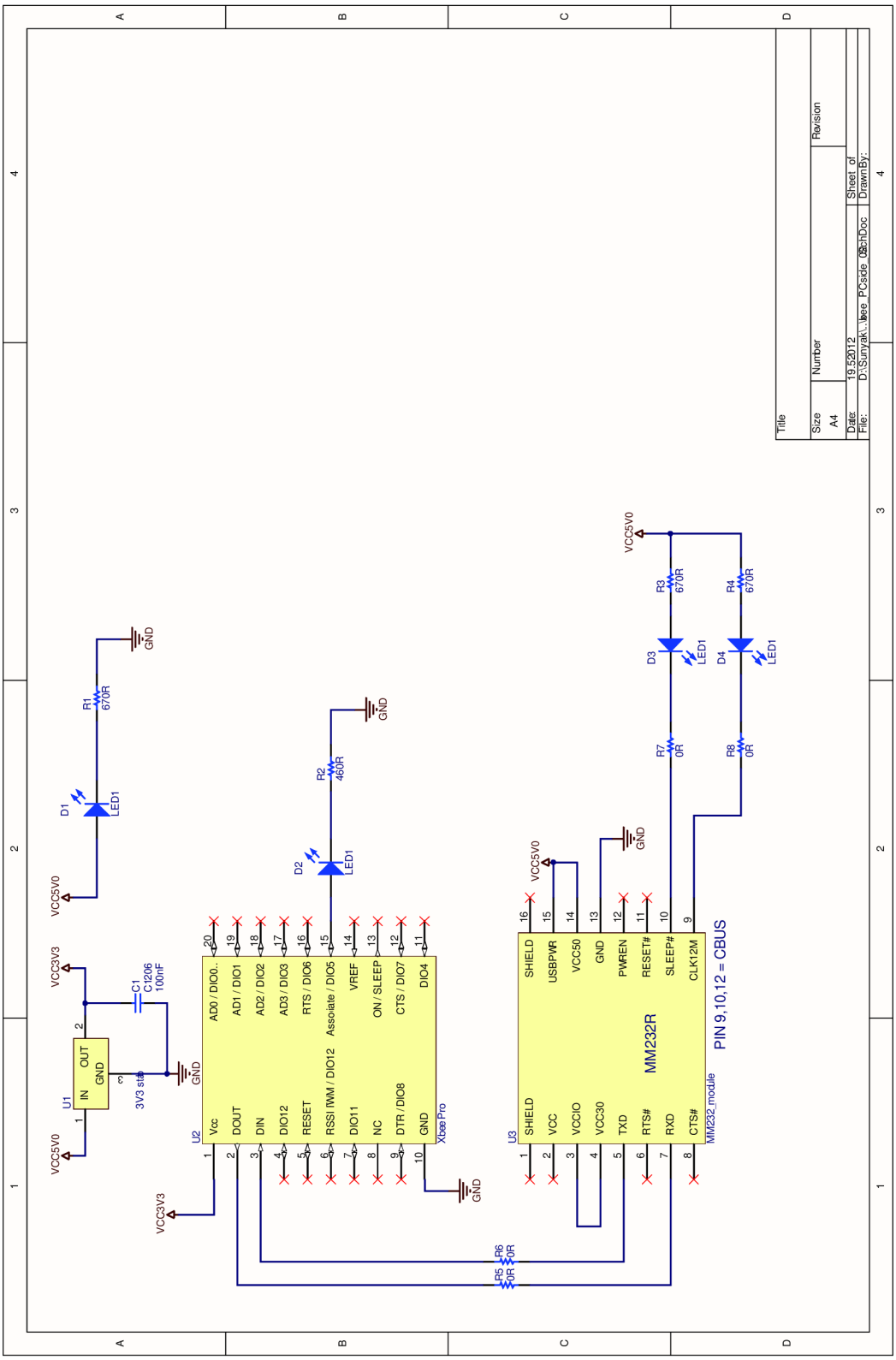


Figure 6-3: Schematic diagram- Xbee <-> PC

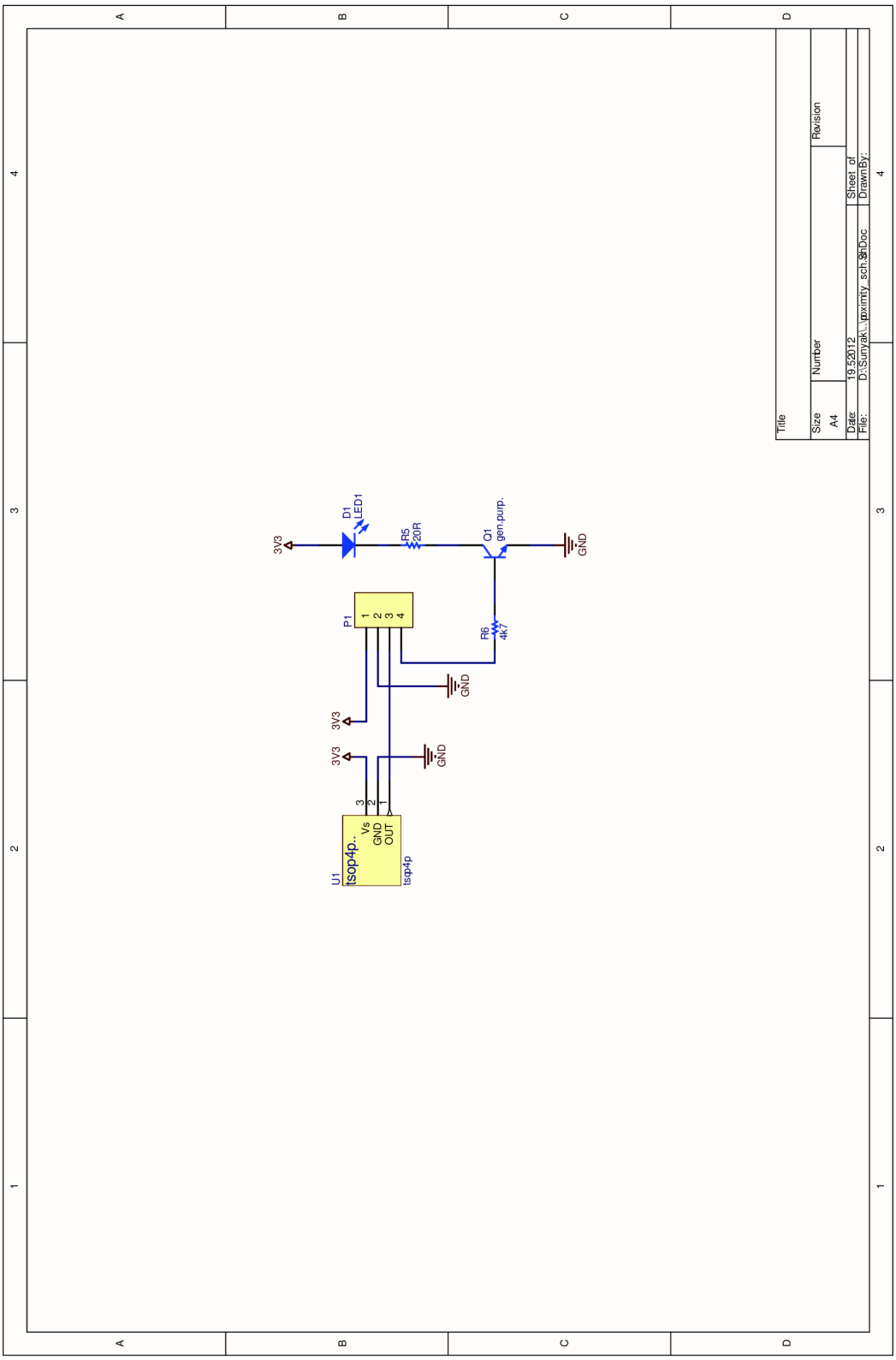


Figure 6-4: Schematic diagram- IR sensor

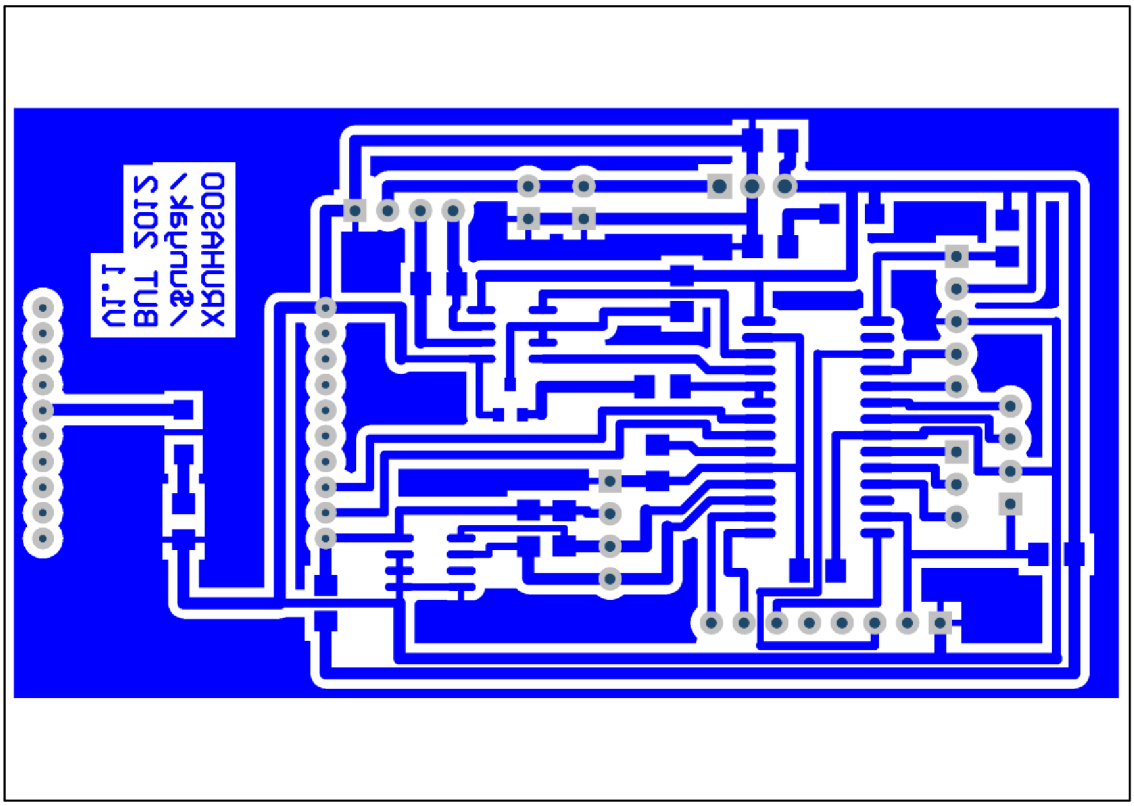


Figure 6-5: PCB- Main Control Circuit

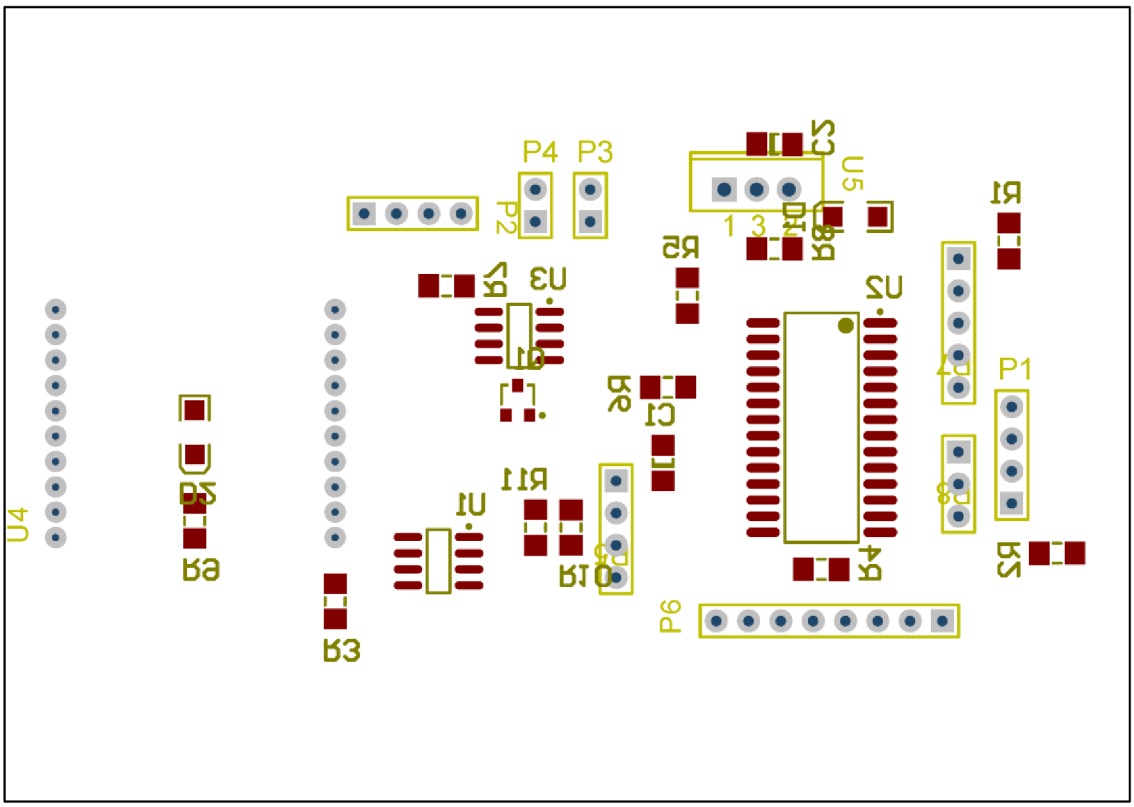


Figure 6-6: PCB component placement- Main Control Circuit

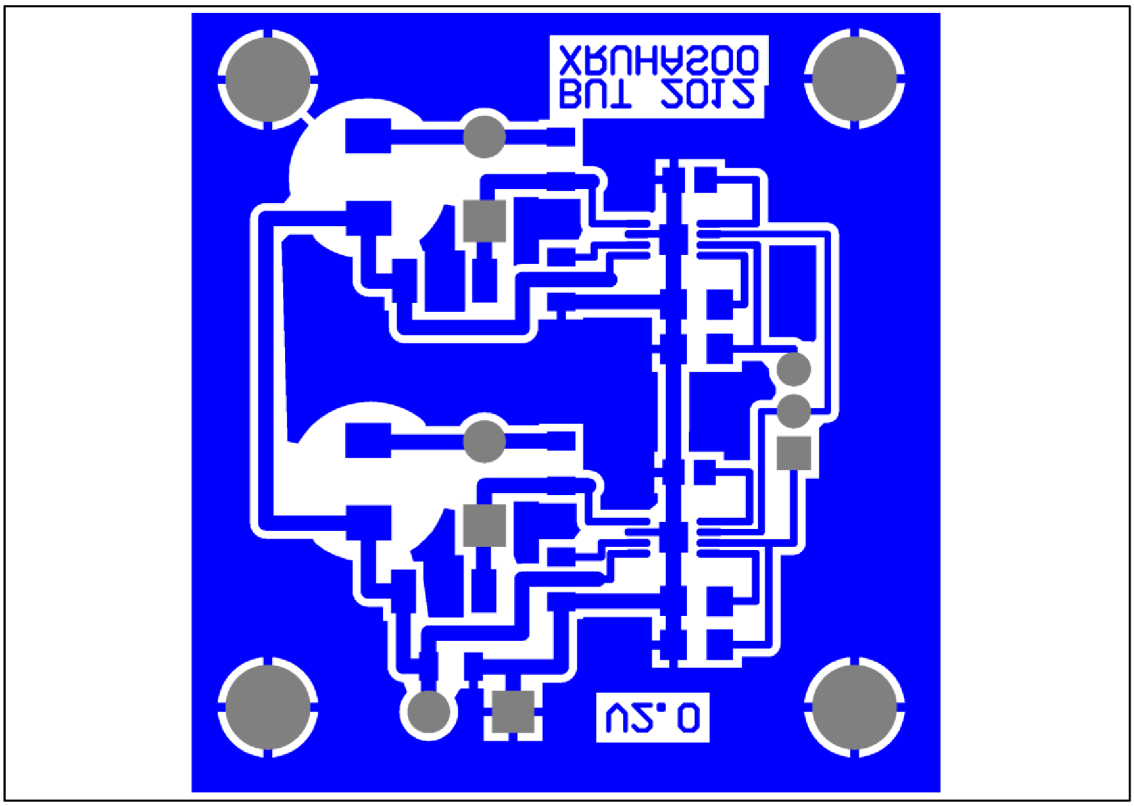


Figure 6-7: PCB- LED driver

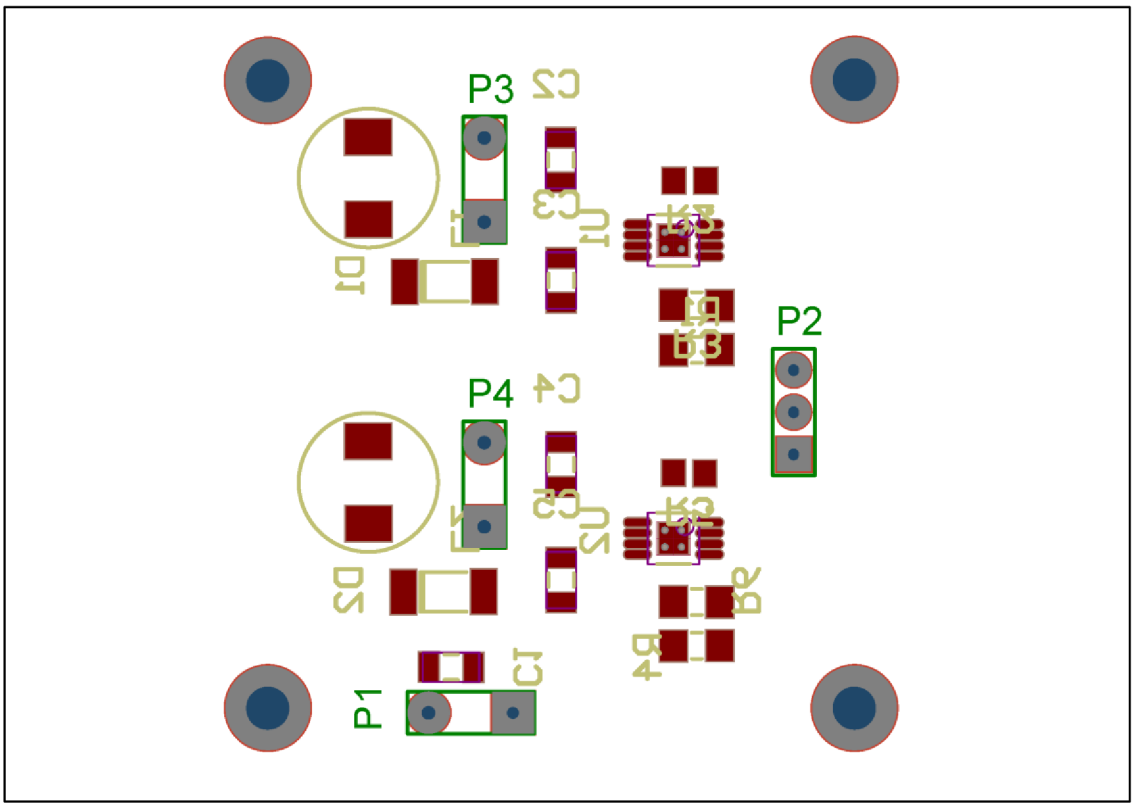


Figure 6-8: PCB component placement- LED driver

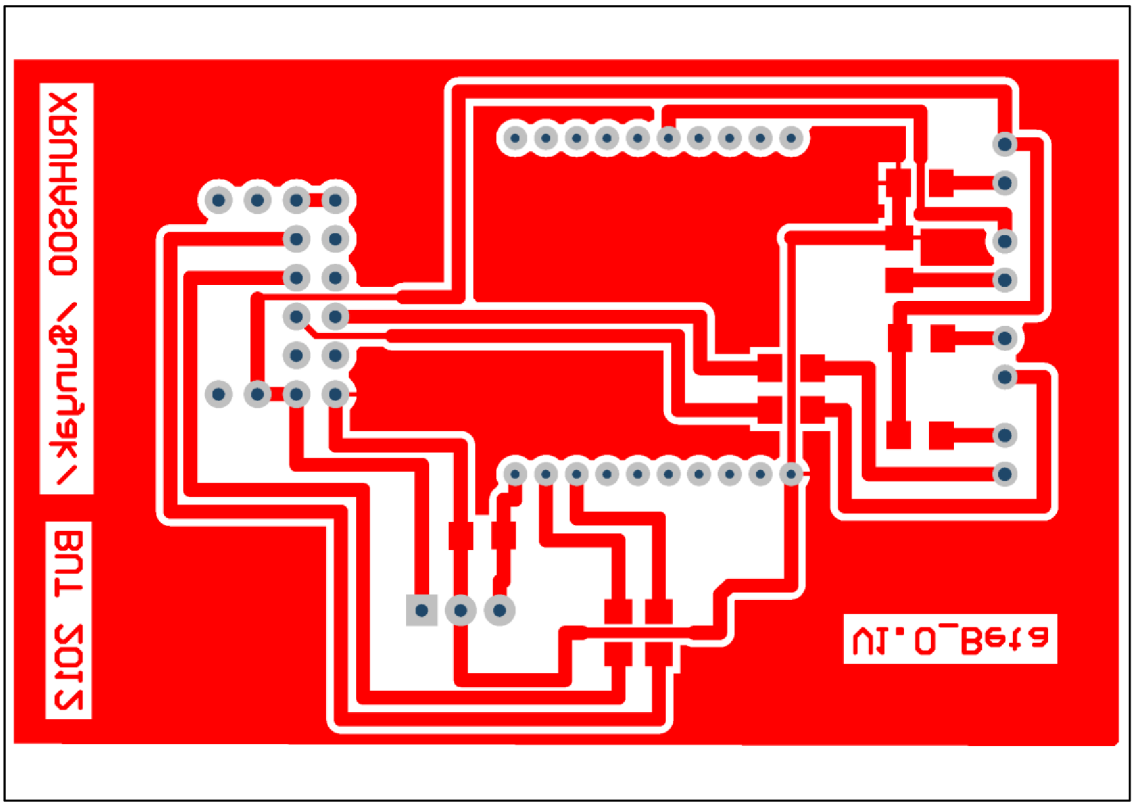


Figure 6-9: PCB- Xbee <->PC

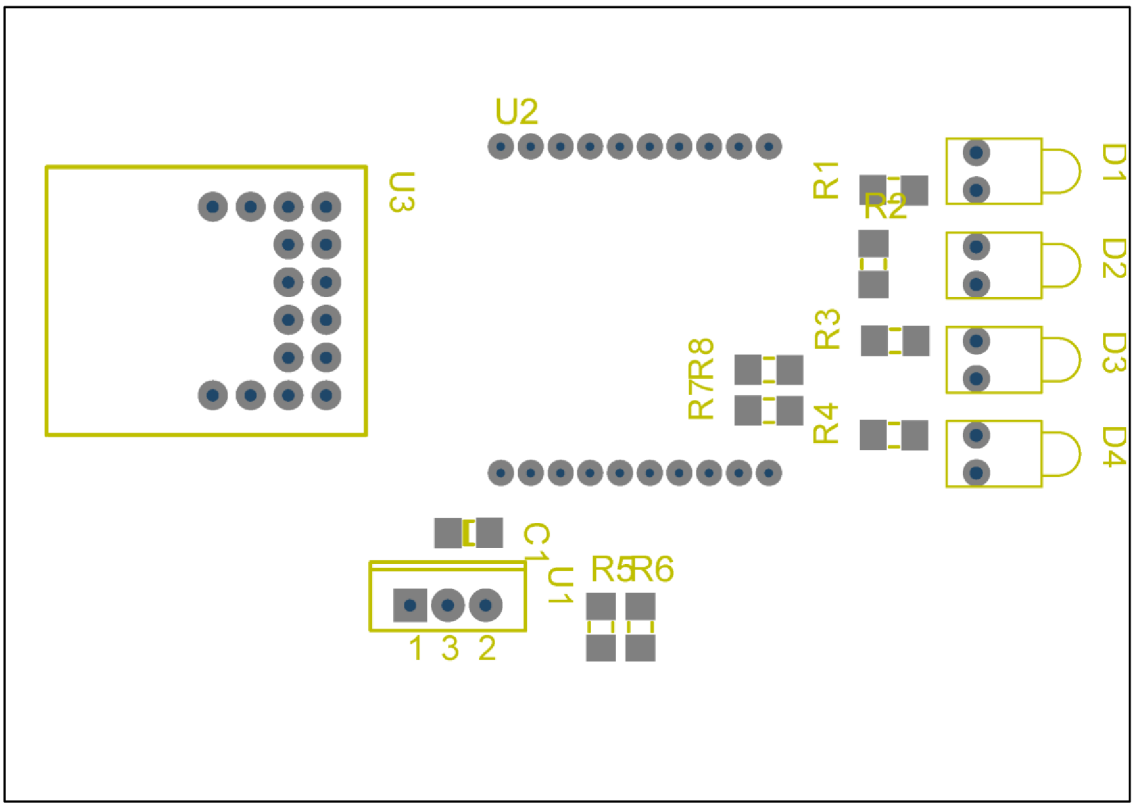


Figure 6-10: PCB component placement- Xbee<->PC

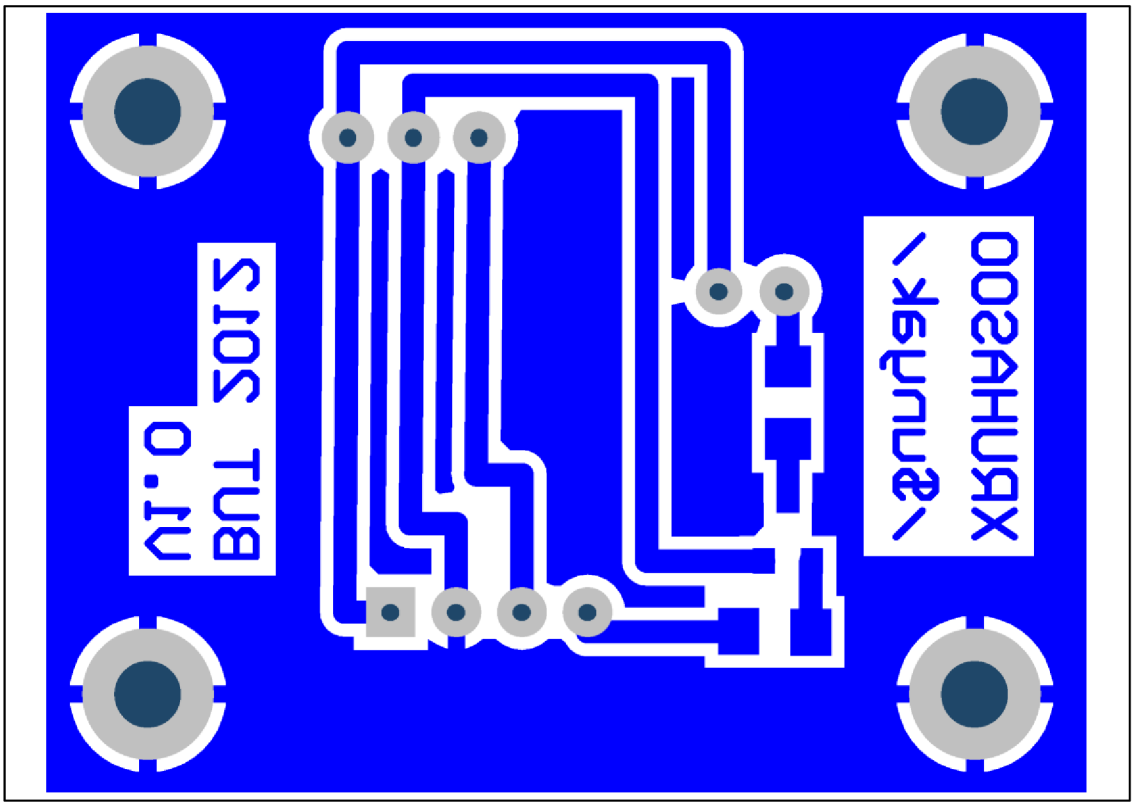


Figure 6-11: PCB- IR sensor

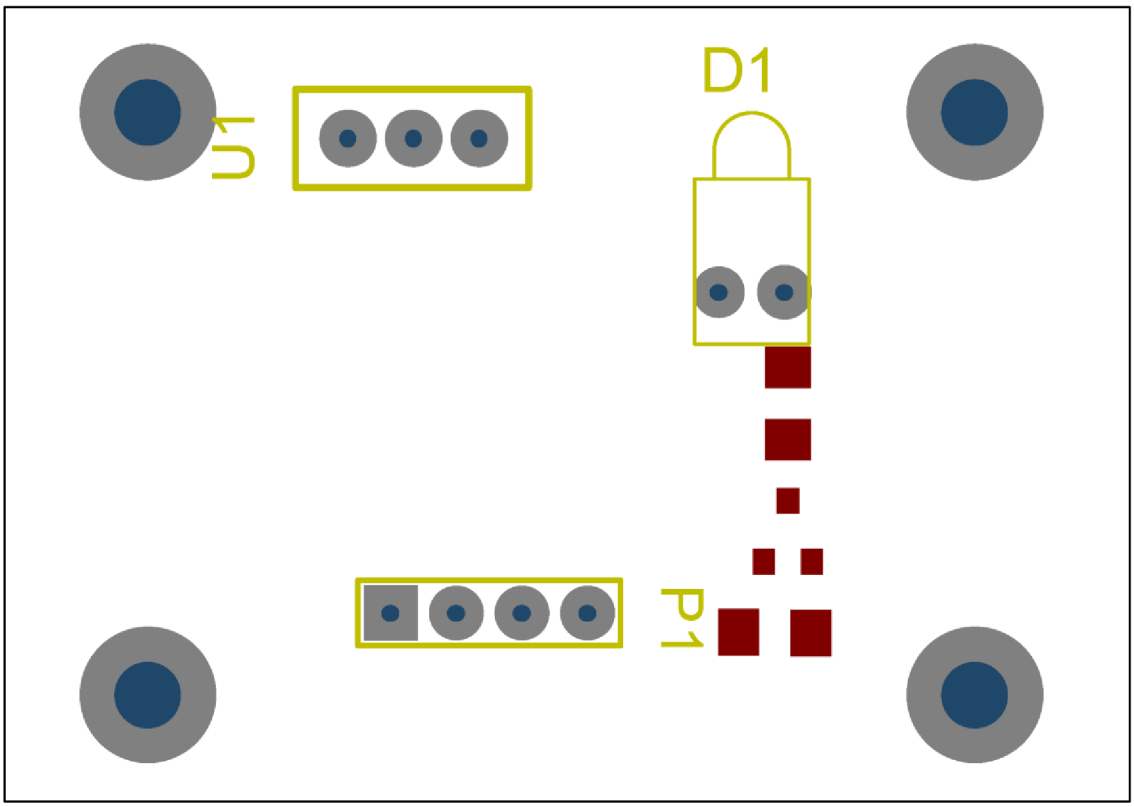


Figure 6-12: PCB component placement- IR sensor

Attachment 2: Bill of materials

Main Control Circuit

STCN75	U1
PIC24HJ64GP502	U2
RS485 transceiver	U3
Xbee Pro S2	U4
DC/DC converter 1A	U5
10uF /1206	C1
100nF /1206	C2
LED /1206	D1, D2
Header, 4-pin	P1, P2, P5
Terminal, 2-pin	P3, P4
Header, 8-pin	P6
Header, 5-pin	P7
Header, 3-pin	P8
NPN general purpose transistor /SMD	Q1
10kΩ /1206	R1, R5, R6, R10, R11
0Ω /1206	R2, R3, R4
RS485 termination resistor, not included	R7
680Ω /1206	R8, R9

Board with Xbee module for PC

100nF /1206	C1
LED 5mm /green	D1, D2, D3, D4
670Ω /1206	R1, R3, R4
460Ω /1206	R2
0Ω /1206	R5, R6, R7, R8
LF33CV	U1
Xbee Pro S2	U2
MM232 USB/UART dev. module	U3

IR proximity sensor

TSAL6200	D1
20Ω /1206	R5
4k7 /1206	R6
NPN general purpose transistor /SMD	Q1
TSOP4P38 – IR receiver	U1
Header, 4-pin	P1

LED driver circuit

100nF, ceramic /1206	C1
1uF, ceramic /1206	C2, C3, C4, C5
Schottky, 30V/1A	D1, D2
22uH/SMD	L1, L2
Terminal, 2-pin	P1, P3, P4
Header, 3-pin	P2
68kΩ /1206	R1, R4
67kΩ /1206	R3, R6
670Ω /1206	R2, R5
LM3407 /eMSOP-8	U1, U2

Miscellaneous

4x Green LED -1W
4x Dynamixel RX-64
1x Plastic enclosure
1x 3A fuse
HMC6352 digital 2-axis compass
ADXL345 3-axis digital accelerometer

Attachment 3: Pictures of the finished robot

Figure 6-13: The finished robot

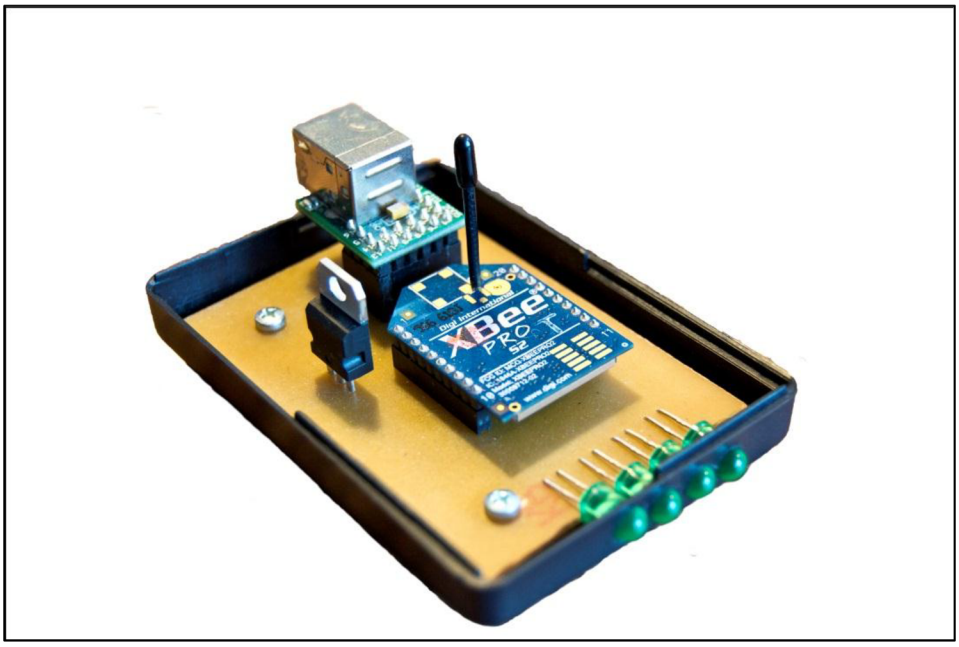


Figure 6-14: Board with Xbee module- PC side

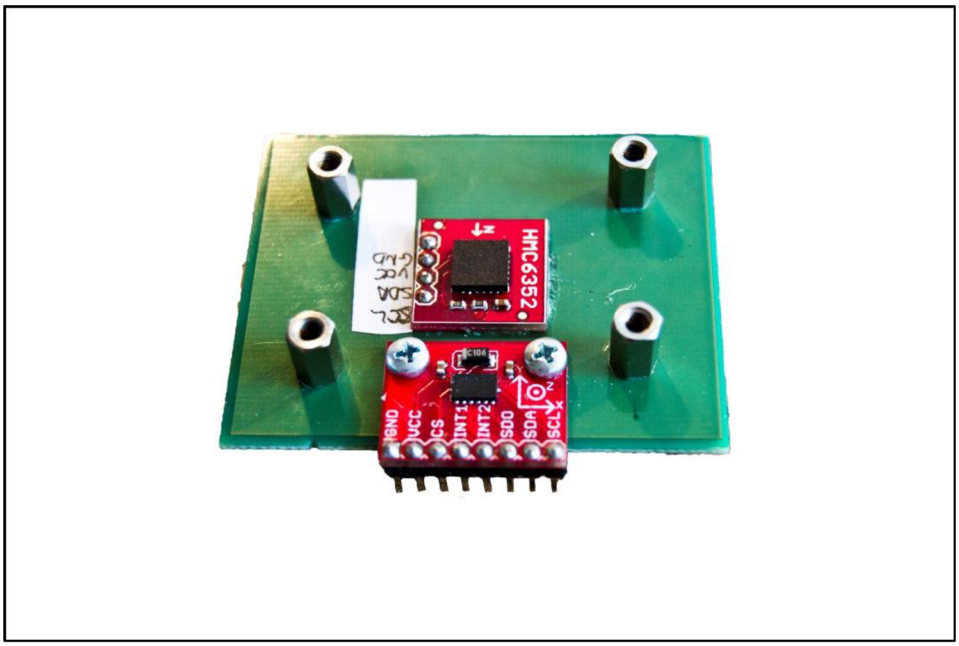


Figure 6-15 Compass and the Accelerometer mounted to a cuprextit board

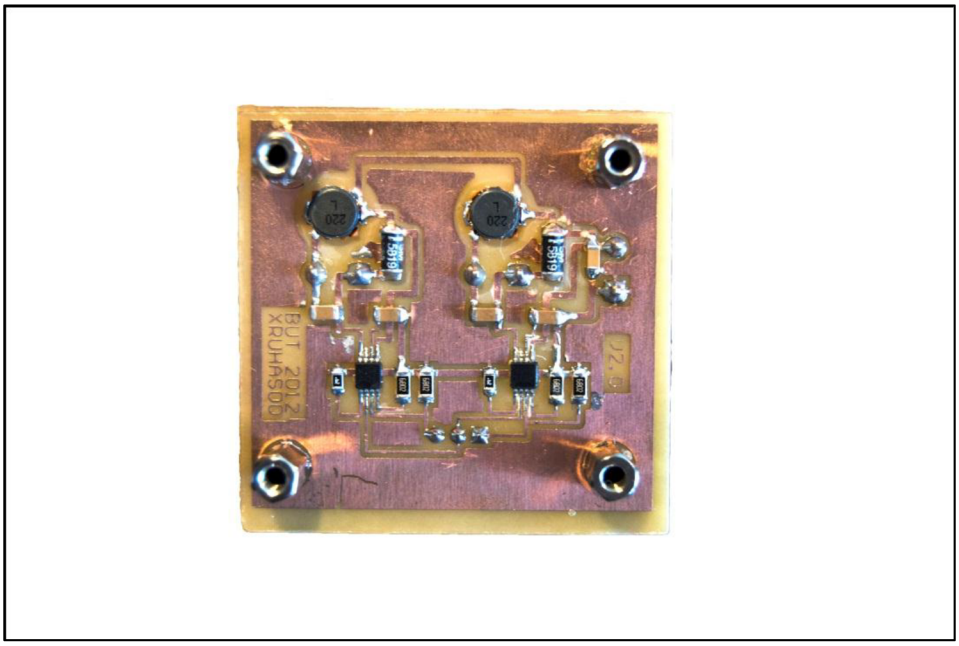


Figure 6-16: The finished LED driver

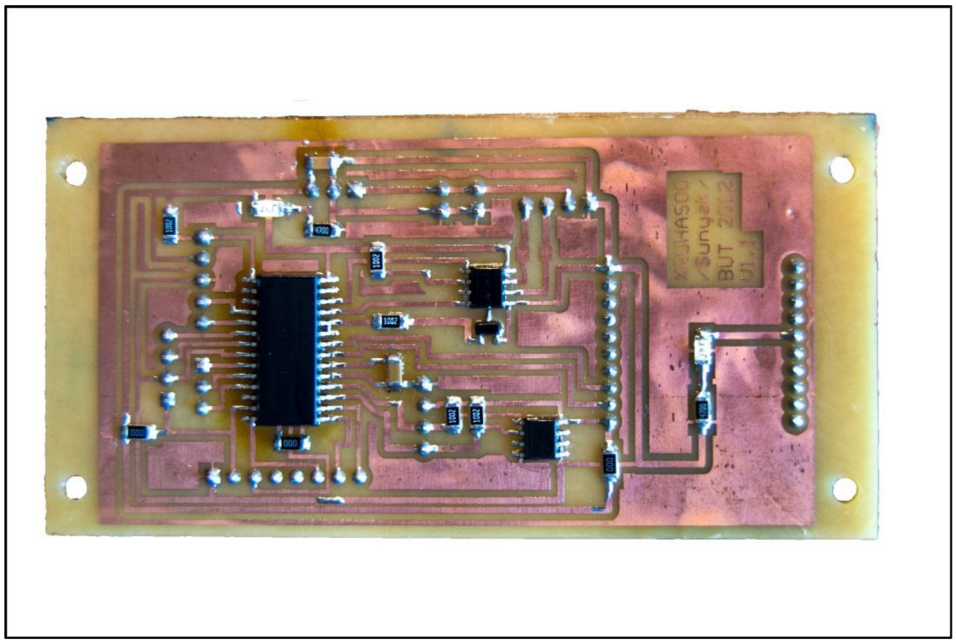


Figure 6-17: Finished main circuit- bottom side

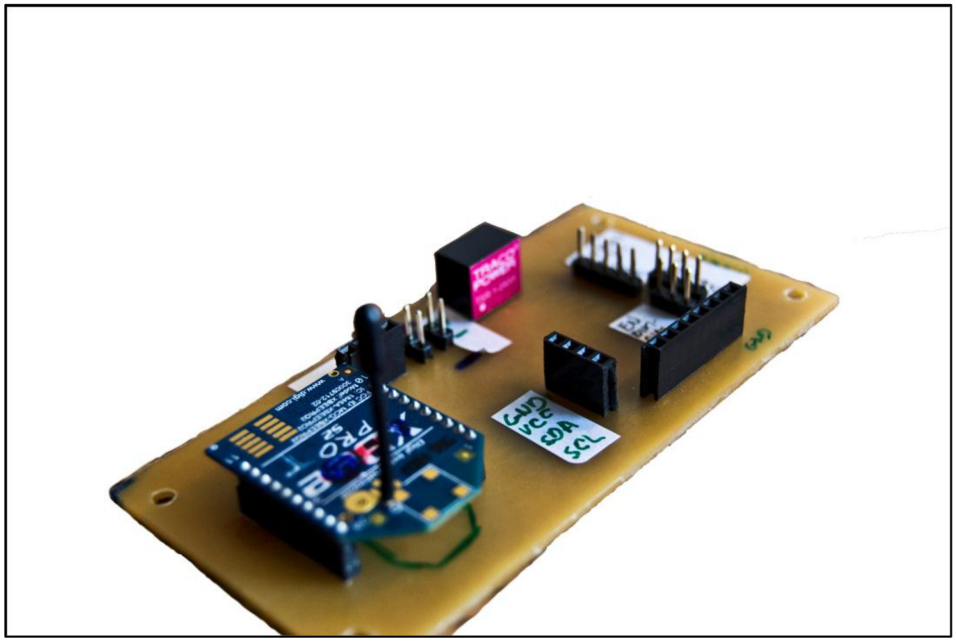


Figure 6-18: Finished main circuit- top side

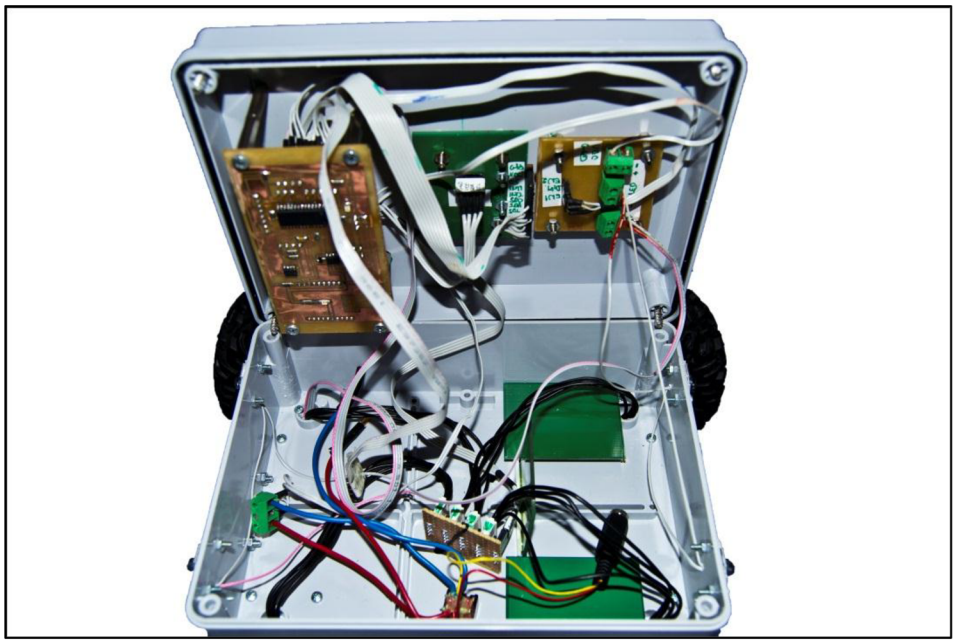


Figure 6-19: Finished robot- inside view