COLLEGE OF ENGINEERING

& BUILT ENVIRONMENT

SCHOOL OF ELECTRONIC AND

COMMUNICATIONS ENGINEERING

This Report is submitted in partial fulfilment of the requirements of the

B.Eng. Tech. in Electronic and Computer Systems

(DT079/DT079)

Security Robot

Final Project Report

Submitted by

Krzysztof Grobelak D09120590

30 May 2011

Supervisor: Lejla Rovcanin

1. Introduction:

The purpose of this report is to document the progress of work completed on the Spy Robot project. **Figure 1** illustrates main elements of the whole system. All elements mentioned in this report are fully functional, except where stated otherwise.

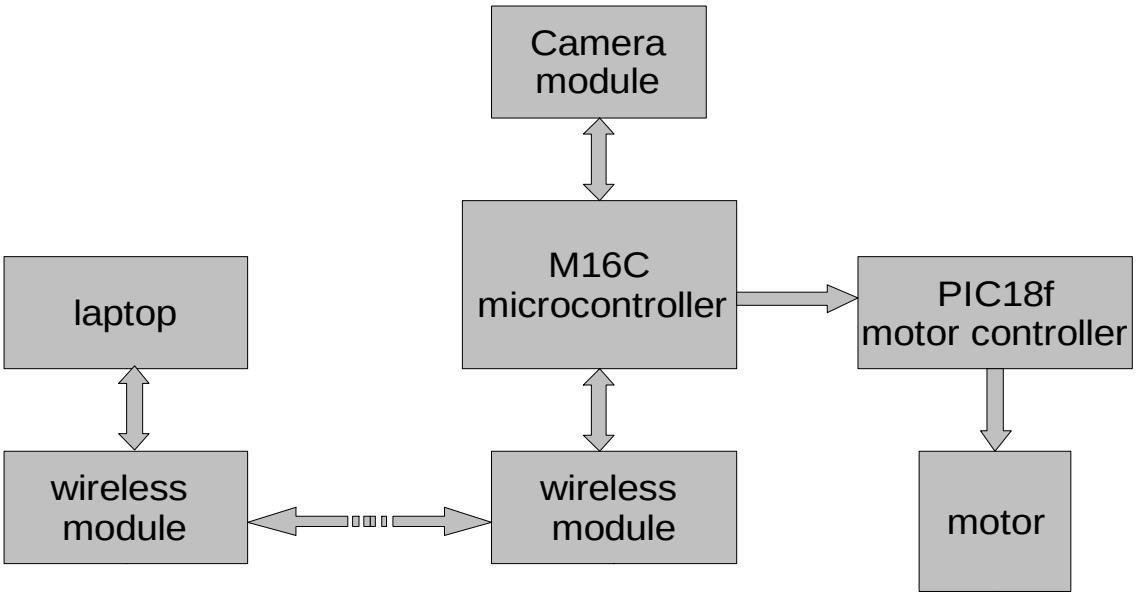

Figure 1 Block Diagram

2. System elements description

2.1 Renesas M16C microcontroller

Renesas **[1]** M16C M30626FJP **[2]** microcontroller is the brain of the system. It is a 16 bit microcontroller with 32KB of RAM and 512 KB of ROM, running at 24MHz frequency with PLL. It is responsible for sending commands to the motor controller using UART2. Additionally M16C is also responsible for communicating with the camera module. It uses two types of communication with the camera: I2C Protocol and parallel port. Commands are sent to the camera via I2C. The picture data

sent from the camera are captured by reading pins on PORT0. **Figure 2** shows how particular elements of the system are interfaced with the microcontroller. Renesas M16 is programmed using E8 programmer/debugger from Renesas. E8 uses M16C UART1 port for communication and debugging. It means that one UART module of the microcontroller cannot be used for any application as it would make debugging impossible.

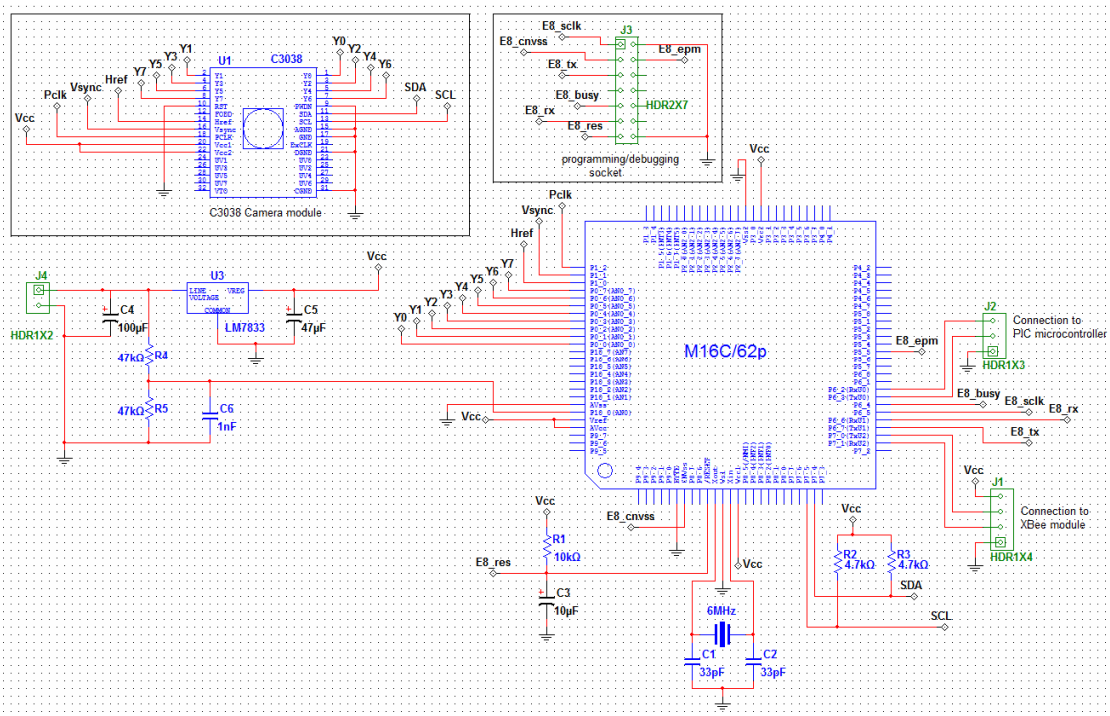

Figure 2 M16C circuit

The last task M16C performs is receiving commands from laptop and sending picture data extracted out of camera to laptop. The wireless connection between laptop and M16C is established by using two Xbee wireless modules. UART0 communication protocol is used for this purpose

2.2 PIC 18F1330 microcontroller

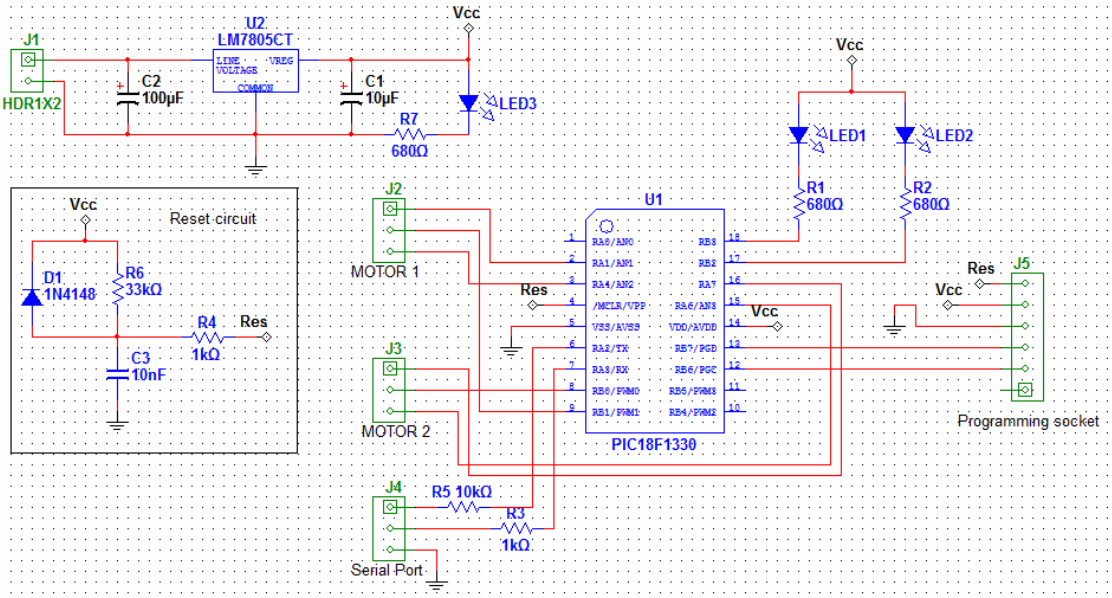

Figure 3 PIC circuit

PIC 18F1330 **[3]** is a medium range 18 pin microcontroller from Microchip **[4].** 18F1330 is an 8 bit microcontroller with 256B or SRAM and 8192B of ROM, 8MHz internal oscillator and 13 general purpose I/O pins. Total of 6 PWM outputs make this uC ideal for tasks where motor control is required. 18F1330 is responsible for controlling robots mobility. In this system its main purpose is to control two separate motor driver circuits. It operates as a slave device receiving instruction from the M16C trough its UART port. It uses very simple protocol. First byte it receives determines the command: go forward, go back and turn left or right. Second byte contains value of speed at which it has to perform the operation. Motor speed is regulated by applying Pulse Width Modulation (PWM) technique. Hardware PWM was used instead of software as this microcontroller has six dedicated pins for this purpose. Software PWM could be used but it would unnecessarily increase C code complexity.

2.3 Motor driver circuit

Circuit of the H – bridge motor driver is shown in **Figure 4**. This is a very popular circuit widely used in robotics for controlling DC motors. Depending on the signals applied to terminals A and B it provides: bi-directional motor operation, breaking and coasting. H-bridge operation in its simplest form can be reduced to selecting the route for the current to flow by turning on particular MOSFETS. If motor is to rotate clockwise then Q3 and Q4 must be switched on, while Q1 and Q2 stay off and vice versa for anticlockwise operation. In order to break, Q1 and Q4 need to be turned on with Q3 and Q2 off. If Q3 and Q2 are on, while Q1 and Q4 are off, motor will be in a state called coasting, which means that it can rotate freely if external force is applied. Diodes D1-D4 are called kickback diodes and their function is to protect the transistors from back EMF, created when current flowing trough inductive devices, like motors, changes. They are not essential in this kind of circuit, where voltages are relatively small, but they provide additional protection for the circuit making it more robust and it is a good practice to do so. The primary purpose of transistors Q5 – Q8 is to prevent condition called straight trough when Q1 and Q3 or Q2 and Q4 are turned on at the same time. This condition is very dangerous for the circuit as those pairs of transistors, when turned on, create essentially a short between Vcc and GND which will result in very high current, high amounts of heat and destruction of circuit. Secondary role of Q5 – Q8 is reducing number of pins required for control.

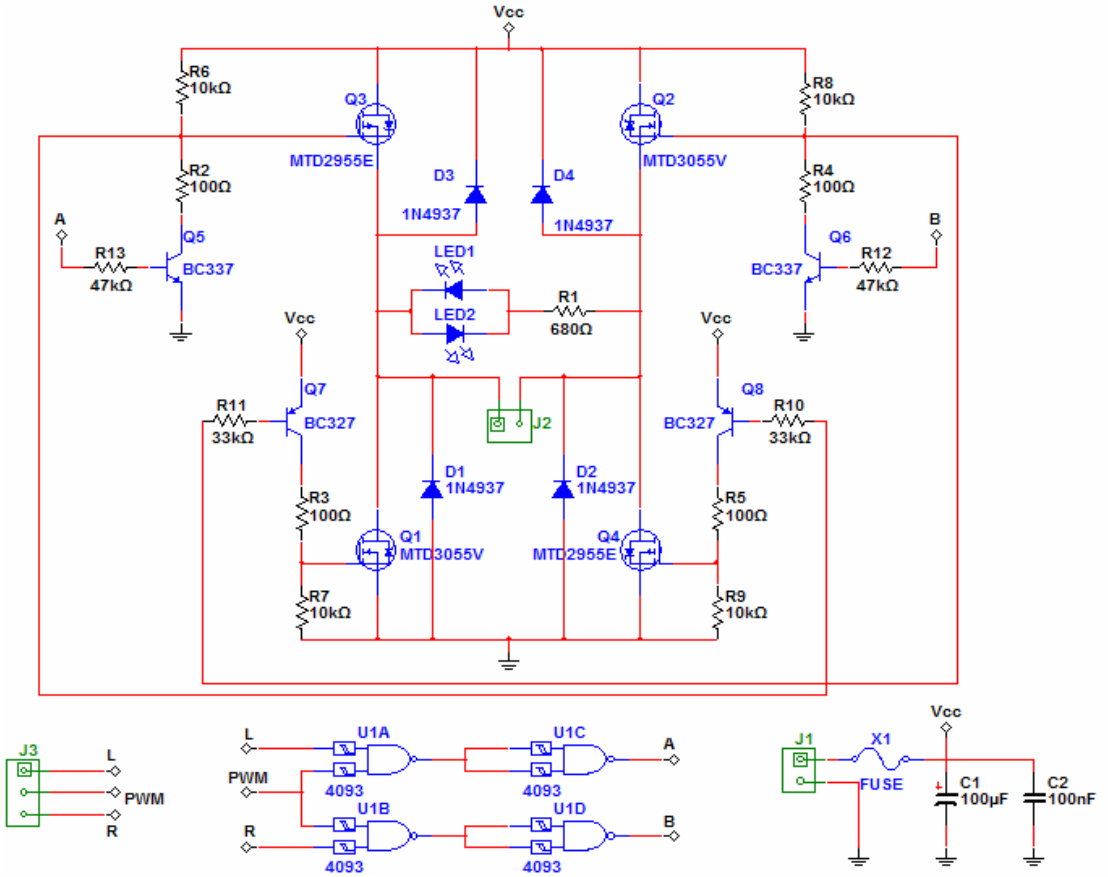

Figure 4 H - bridge motor driver circuit

4093 CMOS **[5]** is a logic gate used to further enhance motor control.

L	R	PWM	Motor operation
x	x	L	stop
H	L	H	left
L	H	H	right
H	H	H	coast
L	L	H	break

Table 1 H - bridge motor driver truth table

Table 1 illustrates the signals used for controlling the circuit. “x” represent “don’t care” state, “H” stands for logic high and “L” stands for logic low. A whole range of movements is encoded into particular combination of those three signals. Adding U1 is not necessary, but was added to this circuit to save PWM outputs on the microcontroller, so that only one PWM output is needed per driver rather than two.

2.4 Battery and Current Consumption

A high capacity Ni – MH (Nickel – Metal Hydride) battery was chosen to provide power for the robot. It has a working voltage of 6V and a capacity of 1600 mA.

Electrical characteristics of the robot are as follows:

- Quiescent current of robot 95 – 100mA
- While moving 550 – 650mA

Therefore the battery allows for a constant operation of the device for about 2 hours. Increase of current during image capture is negligible as the operating current of the camera module is less than 10mA and quiescent current les than 1mA. Therefore the motors and motor driver circuit is the most current consuming part of the circuit.

2.5 Camera module

The camera used in this project is a C3038 **[6]** camera module. It is based on the OV6630 **[7]** chip from OmniVision. This camera uses an I2C **[8]** Protocol for setting up the camera parameters such as: output picture format, picture size, speed, brightness, etc. The camera has a digital parallel output for sending out the picture data Y[0:7] and UV[0:7]. Two different picture formats can be selected: GRB 4:2:2 and YCrCb 4:2:2. In this project the latter was used for the sake of simplicity and it should produce a picture in greyscale format (shades of black and white). YCrCb 4:2:2 format means that the camera outputs 4 pixels of Y, 2 pixels of Cr and 2 pixels of Cb. In this project only Y channel is captured and used to create picture. Y stands for luma or luminance and it contains information on the brightness of the picture. Cr and Cb components are called chrominance and contain color information. The module also supports 2 picture resolution types: QCIF and CIF.

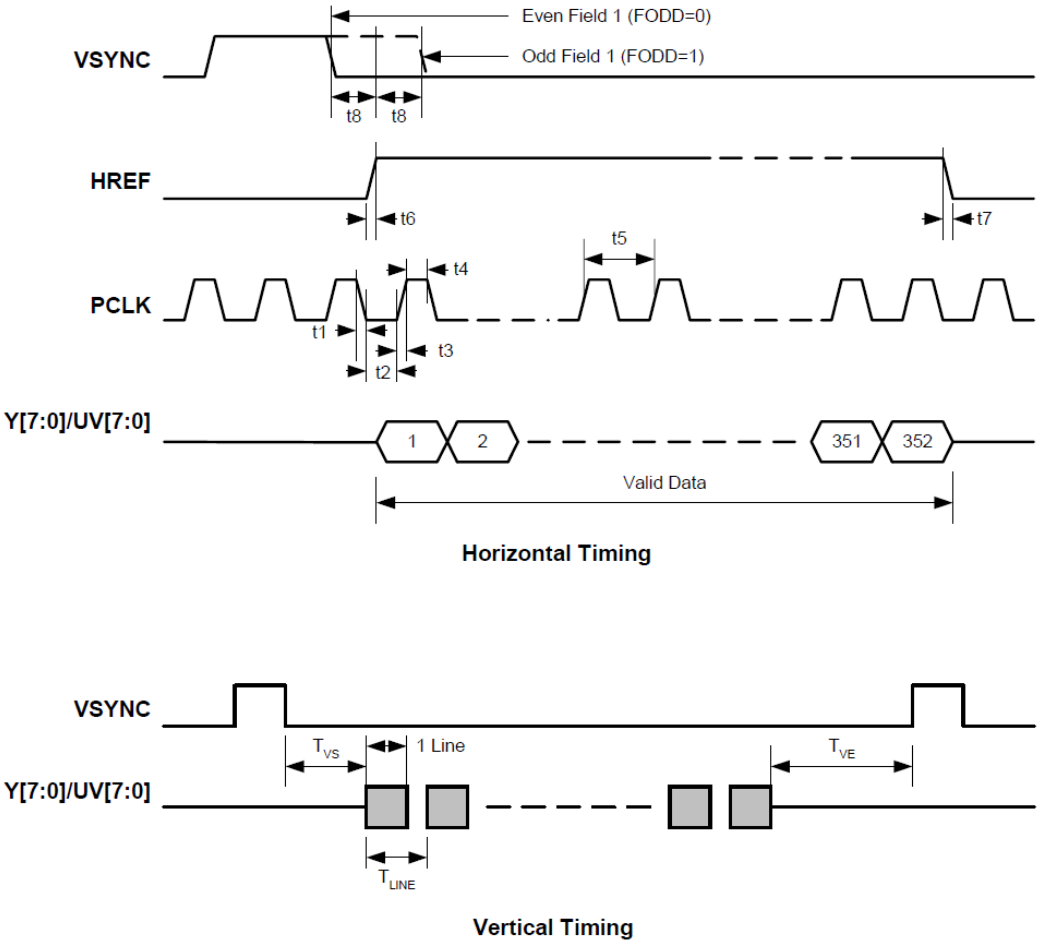

Figure 5 Timing diagram

CIF stands for: Common Intermediate Formats and it is a format used to standardise horizontal and vertical resolutions of pixels in YCrCb video formats. CIF defines pixel resolution of 352x288. QCIF stands for Quarter CIF which makes the picture area a quarter of a CIF format and its pixel resolution is 176x144. Camera uses several signals to synchronise its operation. The signals are as follows: Vsync, Href and PCLK. **Figure 5** illustrates the timing diagram of the device. As shown on the diagram above, a pulse on the Vsync pin indicates the start of a frame, transition from low to high on the Href pin indicates start of a line and the rising edge on the PCLK signal indicates valid data on the Y output. Camera operates at a 17MHz frequency, which normally would be to fast for the M16C, but it is possible to reduce the camera speed by writing the required value to one of its registers using the I2C protocol and reducing the frequency significantly to approximately 100 kHz.

2.6 I^2C Protocol

I^2C communication protocol is used to send settings commands to the camera.

I^2C stands for Inter Integrated Circuit and was developed by Phillips. M16C contains dedicated pins for I^2C communication, but unfortunately they are shared with the UART ports and since all UART modules are used the software I2C had to be implemented. Creating libraries to implement software I2C required a good understanding of the protocol. The benefit of the software I^2C is that it allows any two digital I/O pins to be used for communication and the libraries are platform independent therefore they can be easily ported to any other microcontroller.

I^2C is a master-slave, serial communication. It uses two wires usually called SDA and SCL (data and clock). In this system M16C microcontroller is the master and camera

is the slave. All transmissions must be initialized by a master device. In order to send any data, the following steps must be taken:

- Master generates “start condition”
- Master sends an address of the slave device
- Slave responds with “ACK”
- Master sends the number of register it wishes to access
- Slave responds with “ACK”
- Master sends data to slave
- Master generates “stop condition”

Data is serially sent onto the data line SDA and this process is synchronised with clock signal on the SCL line. Example timing diagram is shown on **Figure 6**.

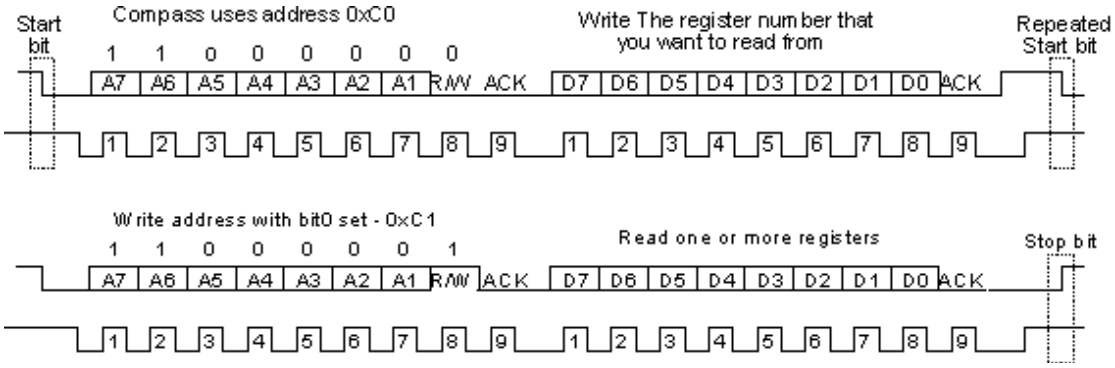

Figure 6 Example I2C timing diagram

2.7 Wireless module

Wireless communication between robot and the PC was established using XBee Series 2 **[9]** wireless module manufactured by Digi **[10].** It is an OEM RF module for wireless communication. It operates within 2.4 GHz frequency band with range up to 40m. **Figure 7** shows Xbee Series 2 module used in this project. In order to interface

the module to microcontroller and PC an adapter circuit had to be made. This circuit can be seen on **Figure 8**. The adapter circuit allows the device to be connected to many different devices like: microcontrollers, PC, laptops, GSM modules, AT modems and works on voltage range from 3.3 to 5V.

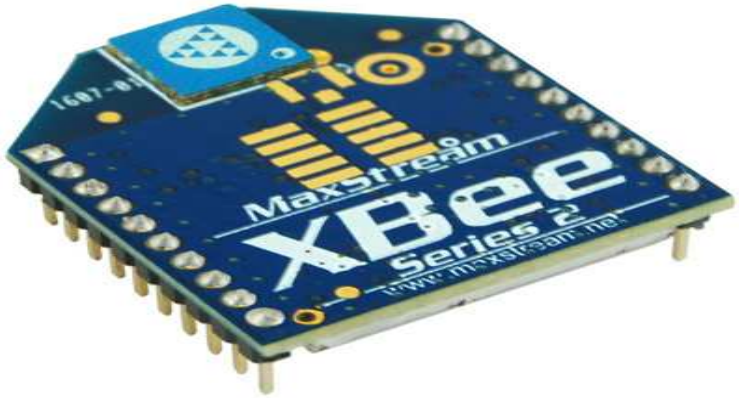

Figure 7 XBee series 2 RF module

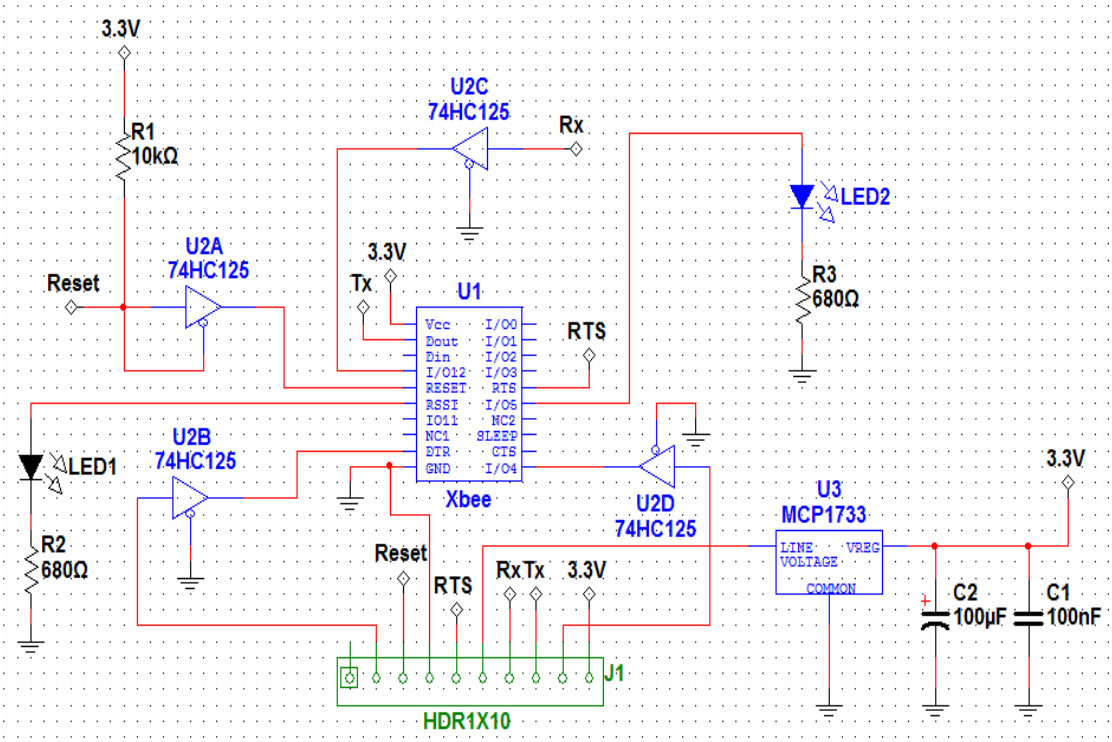

Figure 8 XBee adapter circuit

An Xbee module must be specifically configured before it can be used. Proper configuration is essential for correct device operation. Following features must be configured: baud rate, device ID, destination address, network address. One Xbee module must be set as a coordinator while other as an end device. Setup of these features should allow for establishment of connection between the modules. Configuration can be performed in two ways, by using AT command mode or using X-CTU – software provided by the manufacturer of Xbee modules for performing configuration from PC. X-CTU can be downloaded from the manufacturer's website.
The transmission is performed at the maximum supported baud rate: 115200 bps = 115.2 kbps (kilo bits per second).

3. Software

Since two microcontrollers were used in this project, each of them required its own software. A dedicated application was created for the PC to provide wireless control over the device. Microcontroller firmware was developed using C language and the PC application was written in Java.

3.1 M16C firmware

Code was developed using RENESAS High-Performance Embedded Workshop (HEW) IDE. It is a toolchain developed specifically for RENESAS microcontrollers (M16C, R8C, H8, and M32C). The compiler used was a M16C Series, R8C Family Compiler V.5.45 Release 01.

The C code is responsible for several tasks. Primarily it awaits commands from PC. Communication with PC is established using UART 2 port and is interrupt driven. It

means that it can receive data from PC while performing other tasks. The only case when interrupts are disabled is when picture data are transmitted to PC. It is done so in order to prevent any disturbance of the transmitted data stream. Some of the commands are destined for the PIC microcontroller. If the first received byte is equal to ASCII value of ‘m’, the following bytes will be forwarded to the PIC microcontroller trough the UART 2 port. Receiving ‘q’ or ‘c’ indicates a picture request, where ‘q’ stands for QCIF format and ‘c’ stands for CIF format. When one of those commands is received, an attempt is made to capture the picture. In order to take a picture a command is sent on the I2C line to the camera module indicating required resolution (CIF or QCIF) followed up by command to reduce a PCLK frequency. Also a notification is sent to PC that contains data about image type and number of bytes that are going to be sent. **Listing 1** shows C routine used to do read actual pixel data from the camera. The routine checks which resolution was requested and sets variables accordingly. Next it waits for synchronising signals from camera module. Routine reads a pixel in the first row of the first column then pixel in second row of first column and so on until all rows are read. Then it proceeds to second column. Pixel data are sent to PC as soon as they are read. Another task of M16C is to monitor a voltage on the battery. An ADC (Analog to Digital Converter) on board of M16C was used for this purpose. Battery voltage is measured every 30s. TimerA0 interrupt is used for this purpose. Every 30s an ADC value is read from pin AN0 and raw ADC value is sent to PC where it is converted to volts and displayed in GUI. All data sent from M16C to PC adhere to simple communication protocol called TLV (Type, Length and Value).

```
void grabPic(UBYTE type){
     UWORD y, r, h;
     UWORD horizontal, vertical;

     if(type == 'c'){
          horizontal = 352;
          vertical = 244;
     }else{
          horizontal = 176;
          vertical = 144;
     }
     for(y = 0; y < horizontal; y++){

          while(Vsync == 1);
          while(Vsync == 0);
          for(r = 0; r < vertical; r++){

               while(Href == 0);
               for(h = 0; h < y; h++){

                    while(Dclk == 1);
                    while(Dclk == 0);
               }
               UART0putc(DATA);
               while(Href == 1);
          }
     }
}
```

Listing 1 Routine used for picture capturing

First byte indicates, as the name suggests, the type of information. Three types of data are available in this project: QCIF image, CIF image or Motor Info. The following 4 bytes contain value indicating number of data bytes that will be sent. I.e. if sending QCIF picture data frame will be:

0x01, 0x00, 0x00, 0x63, 0x00, (data stream)…..

3.2 PIC 18F1330

Code was developed in Microchips MPLAB IDE. It is a free toolchain for development of embedded application for PIC series microcontrollers and compiled with MPLAB C18 LITE compiler.

The microcontroller's only task is to provide control over motors. PIC's only UART port is interrupt driven. Each received byte is added to the data buffer. Each command

consists of two bytes: first – command (ASCII **[11]** sign > 127), second – speed (ASCII sign < 128). Timer0 interrupt fires every 10ms and checks if a new command came in. In case of receiving command to go forward the algorithm slowly accelerates until it reaches the required speed. In case of stop command speed is slowly decreased until the vehicle stops. Timer interrupt also provides a degree of safety measures in case of loosing communication with the vehicle. If no new command is received within 300ms the vehicle will stop.

3.3 Robot Control Java Application

The Java application running on the PC provides a way for the user to control the robot. It allows controlling movement and allows taking pictures. It also provides updates about the state of the robot.

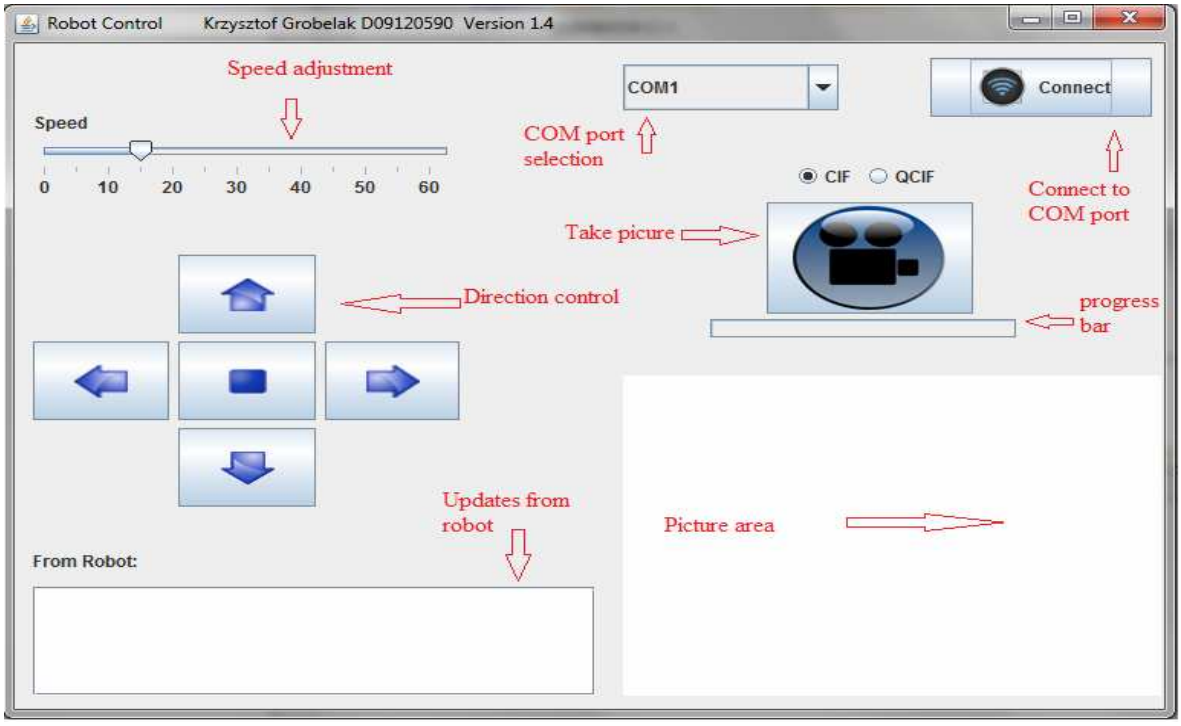

Figure 9 Robot Control GUI

Figure 9 shows user interface used for controlling the device. Five directional buttons provide means to move vehicle between places, slider bar for adjusting the speed, a button to take the picture, box for selecting COM port, “Connect” button for opening a communication channel with the device, bar indicating progress of image download, area for picture to be displayed and small area for displaying update from the robot. Direction control can also be performed using keyboard, as clicking buttons with a mouse is not as intuitive as using a keyboard. Development of the Java application was one of the most difficult parts of the whole project. The program consists of a GUI which reacts to user inputs and results with different commands being sent to the robot.

Whenever robot sends data to PC a serial event is generated by the serial port controller, which results in creating a thread SerialPortReader that handles all received data. As mentioned earlier a TLV protocol was used for that purpose. If the image is sent, then a BMP file is created on hard disc drive and the data (file header, info header, palette table and pixel data) are written into it. Then the file is saved with the time stamp as name and displayed in user interface. If there are robot updates sent then the handler simply displays them in user interface. Using threads was not necessary for development of this application but they were implemented for educational purposes. The difficult task was to synchronise the SerialPortReader thread with the GUI so that updates could be displayed to the user as soon as they were received, like in the case of progress bar or battery updates. This was achieved using a design pattern called MVC **[12]** (Model View Controller). According to this pattern the SerialPortReader is the Controller, user interface is the View and the Model is the interface class RobotListener, which provides a bridge between SerialPortReader and the user interface. The two remaining classes are

RobotController, which contains methods needed for opening and closing serial port as well as methods necessary for sending and reading bytes from serial port. The remaining class is BMP class that contains all the methods required for creating the bmp file and saving it on disk. Classes required for serial port control were provided by java.comms package. This package is not part of the standard java library and needs to be installed separately. This package can be downloaded from the Oracle website.

The Java application was developed using NetBeans **[13]** Integrated Development Environment and the GUI was created using the JBuilder plugin for NetBeans.

3.4 BMP file format

BMP **[14]** File Format also known as Bitmap Image File was used for the purpose of saving the captured image. The BMP format was chosen for storing image due to its simplicity and the fact that they are supported by all available operating systems and software. BMP file consists of:

- Bitmap file header
- Bitmap information header
- Color palette
- Bitmap pixel data

Figure 10 represents headers within the BMP file. Bitmap File header contains information require for identifying the file. First 2 bytes are: ‘B’ and ‘M’ followed by 4 bytes containing file size. Equation to calculate file size is as follows:

FileSize = FileHeaderSize + InfoHeaderSize + ColorTableSize + ImageWidth * ImageHeight

QCIF image size can be calculated as follows:

FileSize = 14 + 40 + 1024 + (176 * 144) = 26422 = 25.8KB

The next 4 bytes are reserved and followed by the last 4 bytes of Info Header containing address of the beginning of pixel data.

After the headers are written into a file, a color (or palette) table must be provided. Color table is a block of bytes containing all the colors used in the picture. For purposes of this project a 256 color greyscale color table was used. Each entry of the color table consists of four bytes: red, green, blue, reserved (0x00).

Name	Size	Description
Header	14 bytes	Windows Structure: BITMAPFILEHEADER
Signature	2 bytes	'BM'
FileSize	4 bytes	File size in bytes
reserved	4 bytes	unused (=0)
DataOffset	4 bytes	File offset to Raster Data
InfoHeader	40 bytes	Windows Structure: BITMAPINFOHEADER
Size	4 bytes	Size of InfoHeader =40
Width	4 bytes	Bitmap Width
Height	4 bytes	Bitmap Height
Planes	2 bytes	Number of Planes (=1)
BitCount	2 bytes	Bits per Pixel 1 = monochrome palette. NumColors = 1 4 = 4bit palletized. NumColors = 16 8 = 8bit palletized. NumColors = 256 16 = 16bit RGB. NumColors = 65536 24 = 24bit RGB. NumColors = 16M
Compression	4 bytes	Type of Compression 0 = BI_RGB no compression 1 = BI_RLE8 8bit RLE encoding 2 = BI_RLE4 4bit RLE encoding
ImageSize	4 bytes	(compressed) Size of Image
It is valid to set this =0 if Compression = 0		
XpixelsPerM	4 bytes	horizontal resolution: Pixels/meter
YpixelsPerM	4 bytes	vertical resolution: Pixels/meter
ColorsUsed	4 bytes	Number of actually used colors
ColorsImportant	4 bytes	Number of important colors 0 = all

Figure 10 BMP data headers

To create a greyscale each color was assigned the same value in range from 0 to 255.Thus the entries would be as follows:

1. red = 0x00, green = 0x00, blue = 0x00, reserved
2. red = 0x01, green = 0x01, blue = 0x01, reserved
3. red = 0x02, green = 0x02, blue = 0x02, reserved

……..

254. red = 0xfe, green = 0xfe, blue = 0xfe, reserved

255. red = 0xff, green = 0xff, blue = 0xff, reserved

Last segment of the BMP file are the actual image data bytes. Those are written directly into a file without any modification as soon as they are all received.

4. Comments

There are several elements in the project that could be improved. The most important one is to reduce the time that it takes to transmit the image from M16C to PC. The limiting factor is the speed of the microcontroller, which operates at 24MHz. It is significantly too slow a speed to capture the image at real time. A high end microcontroller like ARM would have to be used for that purpose. Other ways to improve the capture speed would b to use DMA channels to relay data from the parallel port to the serial port.

Pictures shown on **Figures 11 – 14** in appendices contain some colors in them. This is a result of errors in generation of the palette tables. Values of red, green and blue were not equal for a particular shade. Properly generated greyscale should not contain any colors in them. **Figures 12** and **13** also show problem with writing images to file. The image seems to be cut and the top par of image is placed at the bottom. This was result of the pixel data being written at the wrong address in the file.

Another way of improving the vehicle would be reducing the current consumption of the circuits. It could be achieved by disconnecting unused parts of the circuitry, i.e. disconnecting the motors and motor drivers entirely during image capturing.

5. Summary

Development of this project was a great opportunity to both test my current knowledge of electronic engineering and to obtain new skills and deepen my understanding of electronic concepts in areas of wired and wireless communication, programming, digital signal processing, mechatronics, etc. I was able to learn and put into practice concepts that go beyond the scope of college curriculum like I2C protocol, ZigBee protocol, video signal processing, Pulse Width Modulation, threads and design patterns. Almost all major proposals of the project were fulfilled, except one, which was a sound capture and transmission. Due to time restrictions I was unable to implement that element into my project. The amount of time required to achieve this task would be significant and comparable to time dedicated to making the camera work properly.

5. References:

[1] Renesas Electronics - http://am.renesas.com/

[2] M16C M30626FJP

http://documentation.renesas.com/eng/products/mpumcu/rej03b0001_16c62pds.pdf

[3] PIC 18F1330

http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en022957

[4] Microchip Technology Inc. - http://www.microchip.com/

[5] 4093 QUAD NAND GATE

http://www.ece.ucsb.edu/courses/ECE002/2B_W11Shynk/CD4093.pdf

[6] C3038 datasheet - http://www.datasheetarchive.com/C3038-datasheet.html

[7] OV6630 datasheet - http://www.datasheetarchive.com/OV6630-datasheet.html

[8] I2C - http://www.nxp.com/documents/user_manual/UM10204.pdf

[9] XBee Series 2 http://ftp1.digi.com/support/documentation/90000866_A.pdf

[10] Digi Internationl Inc. - http://www.digi.com/

[11] ASCII format http://www.asciitable.com/

[12] Head First Design Patterns By Eric T Freeman, Elisabeth Robson, Bert Bates, Kathy Sierra

[13] NetBeans http://netbeans.org/

[14] BMP file format - http://www.daubnet.com/en/file-format-bmp

Appendices:

Some example pictures captured so far:

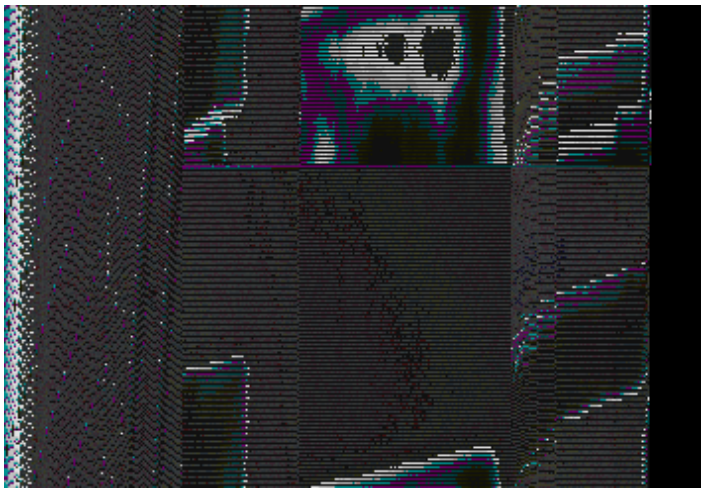

Figure 11 A room – very first picture taken

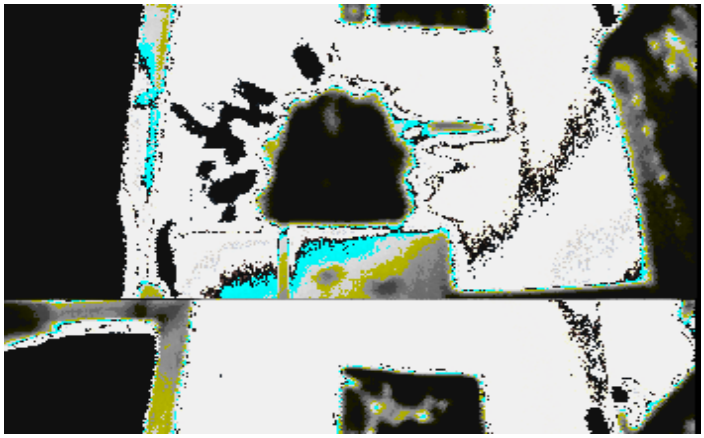

Figure 12 Cactus – the first picture with recognizable features

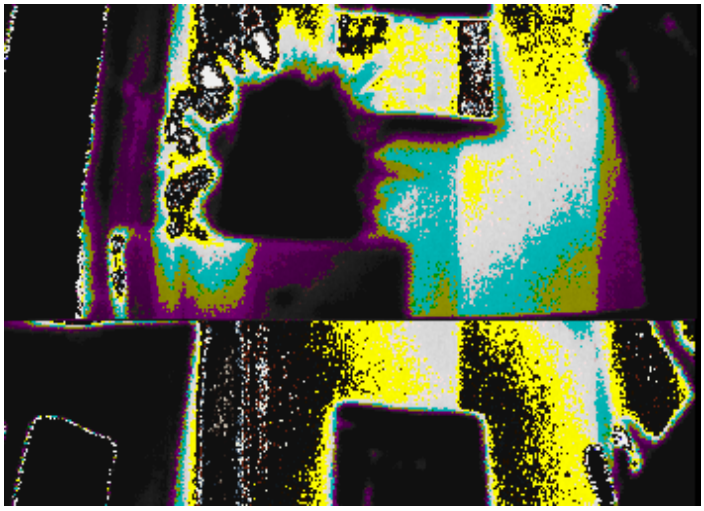

Figure 13 Cactus

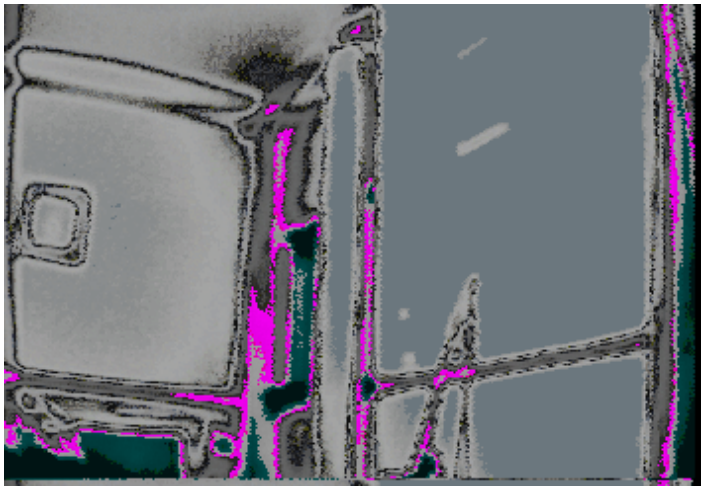

Figure 14 Room

Figure 15 Living room

Figure 16 Kitchen door