Tallinn University of Technology
Department of Computer Engineering
Chair of Digital Systems Design

Boe-Bot robot manual

Priit Ruberg

Erko Peterson

Keijo Lass

Tallinn 2016

Contents

1 Robot hardware description

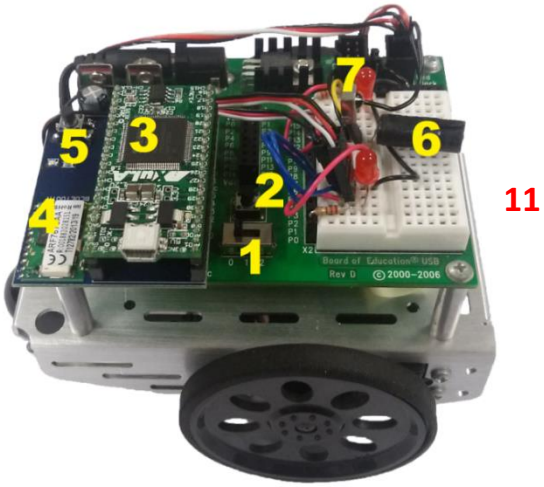

Figure 1. Top view

Figure 2. Bottom view

1 – 3-way switch (0 – off, 1 – on but servos are not active, 2 – on and servos are active)
2 – Pushbutton to restart the program
3 – Microblaze XuLa-200 FPGA
4 – Bluetooth adapter
5 – Pushbutton to erase user program
6 – Infrared sensor
7 – Infrared receiver
8 – Servomotors
9 – QTI sensors
10– Battery pack

11 – Ultrasonic transceiver (not on the picture)

2 BotCommanderBT graphical user interface (GUI) and connection

The first step to programming the robot is to establish a Bluetooth connection between the robot and the operation system. As for the Bluetooth password the combination of “0000” should be used.

The GUI is included in the BotCommander_full.zip archive which can be found on the internet[1]. Whether you use Windows or Linux it’s recommended to start the program in the console. In either case it’s mandatory to have Java installed and added to the *path*. To check whether Java has been correctly installed type *java* in the command/terminal window and press *Enter*. If the response is *“java” is not recognized as an internal or external command…”* then you have to configure your Java[2].

2.1.1 For Windows users

Run *Command Prompt* by typing *cmd* to the Start. On Figure 3

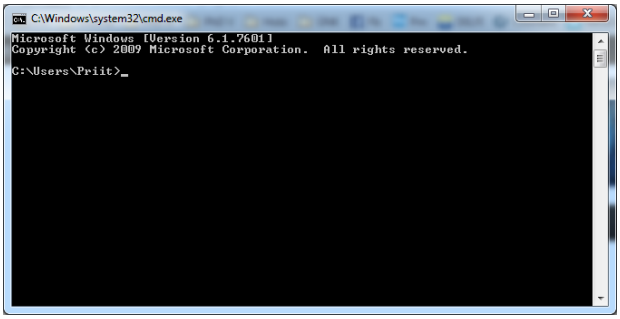

Figure 3. Windows Command Prompt window

you can see the terminal window. Next move to the extracted BotCommanderBT folder and start the program by typing *java –jar BotCommanderBT.jar*. The command is also shown on Figure 4. After that the program window opens as can be seen on Figure 6.

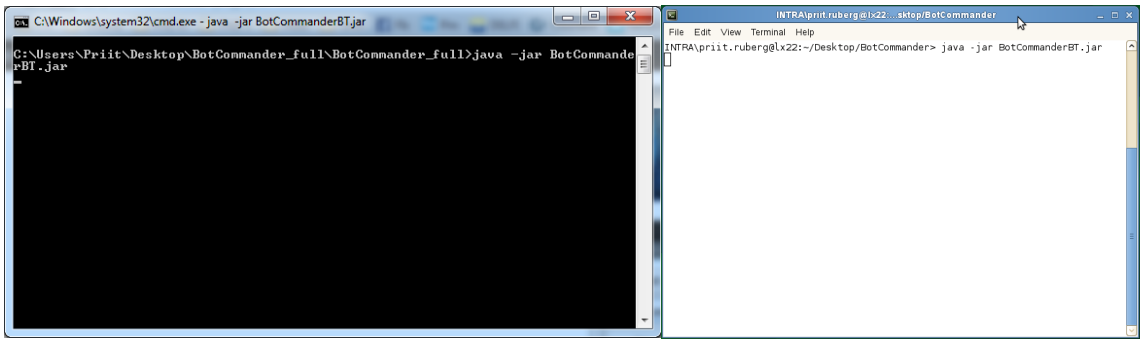

Figure 4. Starting the BotCommanderBT program

2.1.2 For Linux users

Simple way to open a new terminal is to make a right click on the Desktop and choose *terminal*. Since autumn 2016 the BotCommander has been added to the *cad* drive and thus must started using the following commands. In a terminal run a *cad* command (at least once) and then *botcommander* as shown on Figure 5. The program GUI starts and is presented on Figure 6.

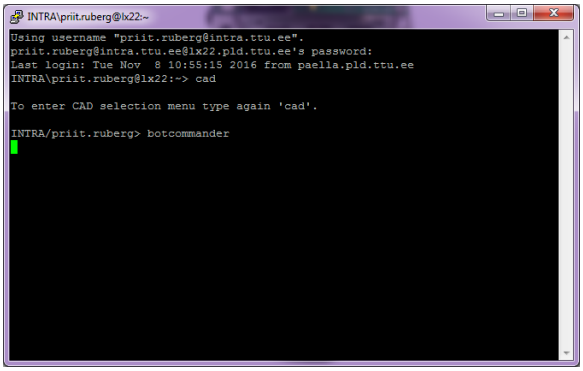

Figure 5.Linux terminal

[1] http://petski.tainas.ee/Sissejuhatus_erialasse/BotCommander_full.zip
[2] http://www.java.com/en/download/help/path.xml

2.2 Using the BotCommanderBT GUI

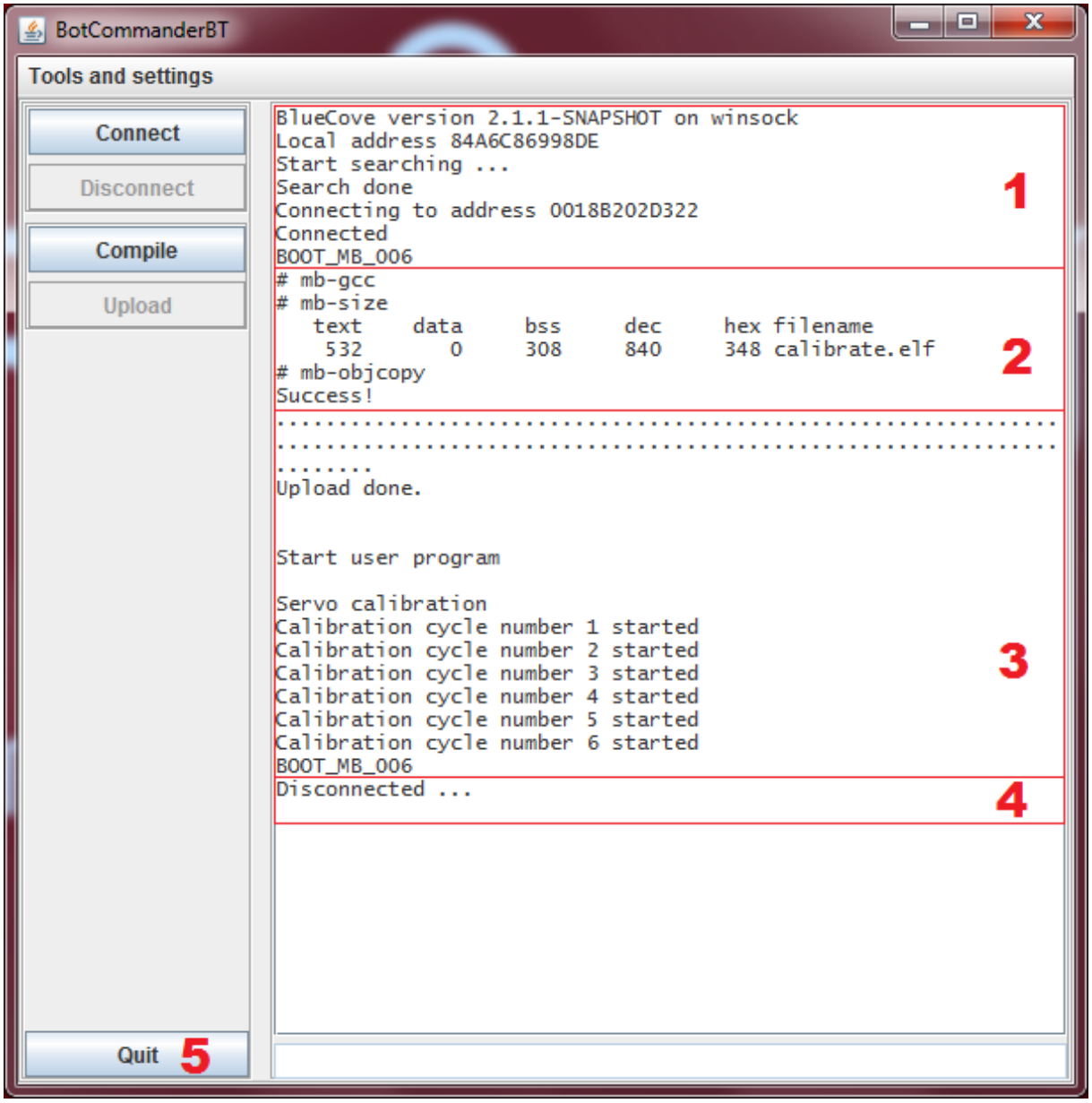

Figure 6. User interface

BotCommanderBT is a user interface for programming the robot.

1. Connect – Connecting to Boe-bot, robot must be turned on first
2. Compile – Compiling the code, choose the code you want to compile
3. Upload – Uploading the code to Boe-bot, choose the file to be uploaded
4. Disconnect – Disconnecting from Boe-bot, terminate the connection
5. Quit – Close the program

3 Robot actuators

The robot has some sensors and transducers to interact with the environment. The description and how to manipulate with them are described below. Since August 2015 the robot has servomotors, infrared LED and infrared receiver, QTI sensors, LEDs, ultrasonic transceiver, FPGA based soft-core Microblaze microcontroller with OTA programming via Bluetooth. In Table 1 is shown the connection overview of the Boe-Bot controller and actuators.

Table 1. Robot actuators connection overview

Pin	Actuator
0	LED
5	QTI
6	QTI
7	Ultrasonic
8	Ultrasonic
10	Infrared LED
11	Infrared receiver
12	Right servomotor
13	Left servomotor
15	LED

3.1 Servomotors

Servomotor is a rotary actuator that allows for precise control of angular position, velocity and acceleration. It consists of a suitable motor and coupled to a sensor for position feedback. It also requires a relatively sophisticated controller. [1] Servo control function is located in section 4.2.

Servomotors are controlled with pulses. The duration of the pulse determines the behavior of the motor. For instance, a pulse with duration of 1,5ms to our servo means that the servo is at standstill. Though pulses above and under this delay make the servo to rotate. In Table 2 are described the maximum limits of the servo with the rotation direction. On Figure 7 the pulses and servo interactions are shown more figurative. On Figure 8 the reaction of the pulses are shown as a result of practical measurement. Notice that pulses around the 1,5ms increase the servo rotation speed more rapidly than pulses around 1,3ms or 1,7ms. As can be concluded pulses 1,3ms and 1,7ms make the servo rotate at the maximum speed. To exceed those limits doesn't change the servo's behavior.

Table 2. Servo pulse

Pulse duration	Reaction
1,5 ms	Servomotor stops
1,3 ms	Servomotor starts to move clockwise
1,7 ms	Servomotor starts to move counterclockwise

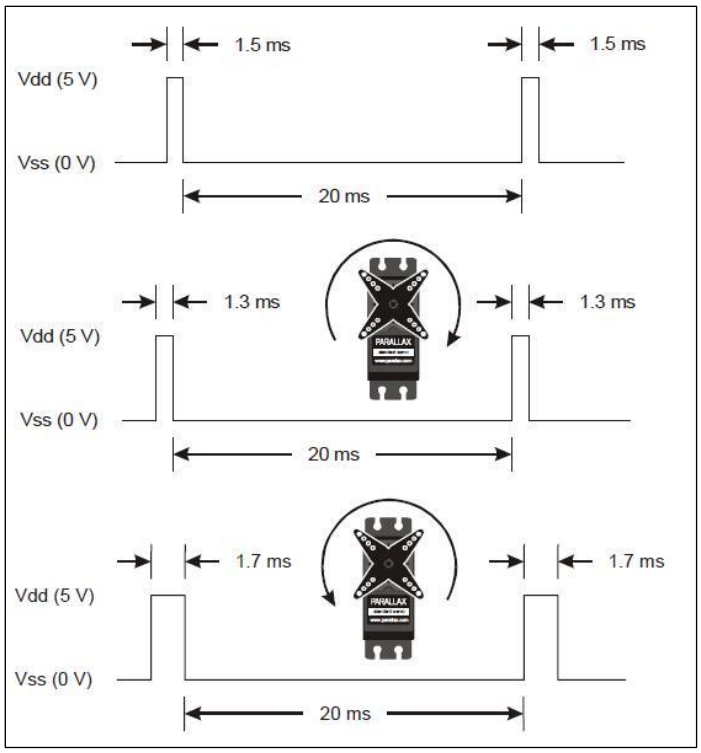

Figure 7. Servo pulse

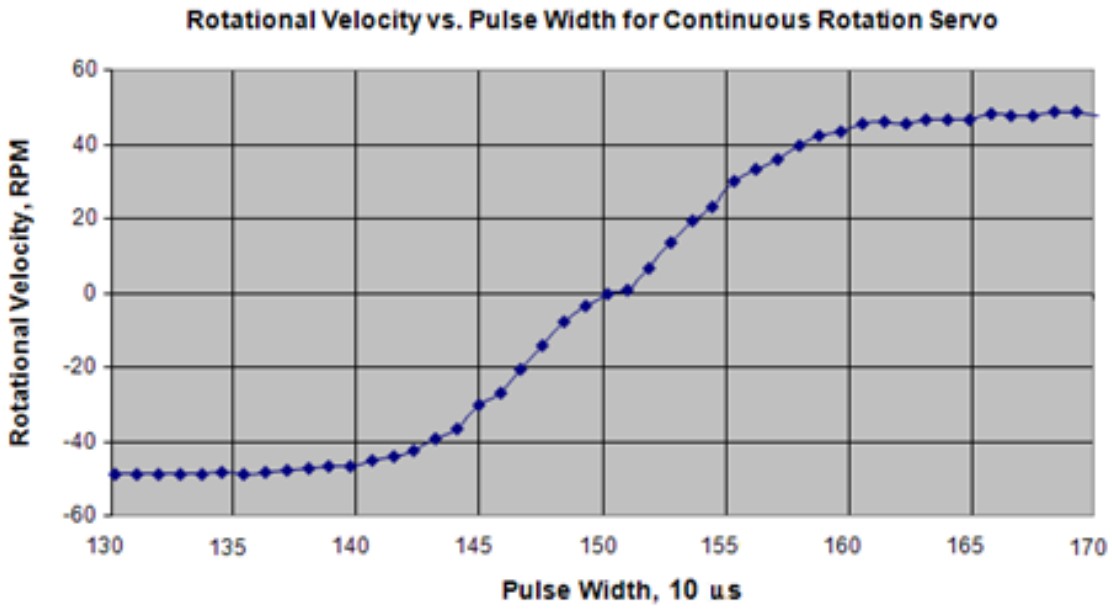

Figure 8. Servo pulse width, 1 pulse = 10 us

3.2 Infrared and distance detection

The infrared (IR) LED emits infrared light and the reflected IR receiver reacts to light and transmits the signal as seen on Figure 9. The Boe-Bot IR works on the same principle as the TV remote.

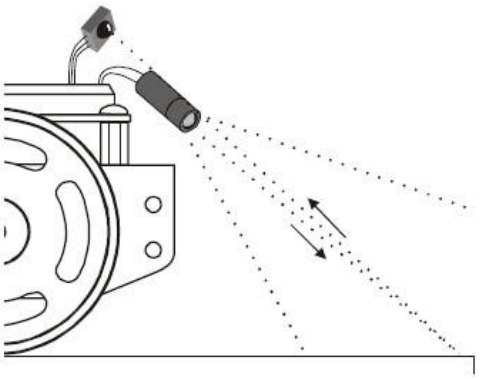

Figure 9. Infrared signal

Boe-bot infrared receiver is designed to receive IR light with a wavelength of about 980nm which converts to 38 kHz. The IR LED emits a 980nm wavelength infrared signal and the receiver detects it. Infrared receiver has different sensitivities to different frequencies which is shown on Figure 10. This principle is based on the distance detection. On Figure 11 the sensitivity of the receiver is shown. Infrared control function is located in section 4.13.

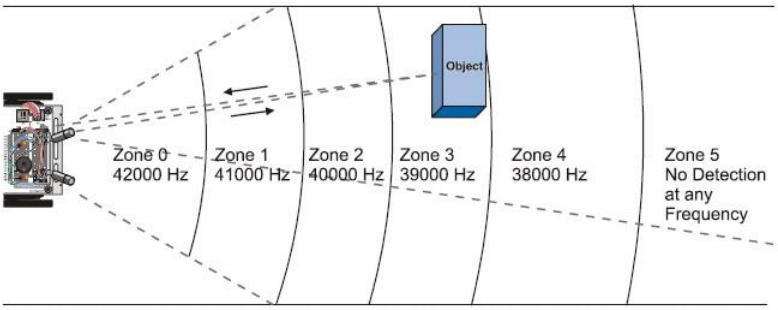

Figure 10. IR zones

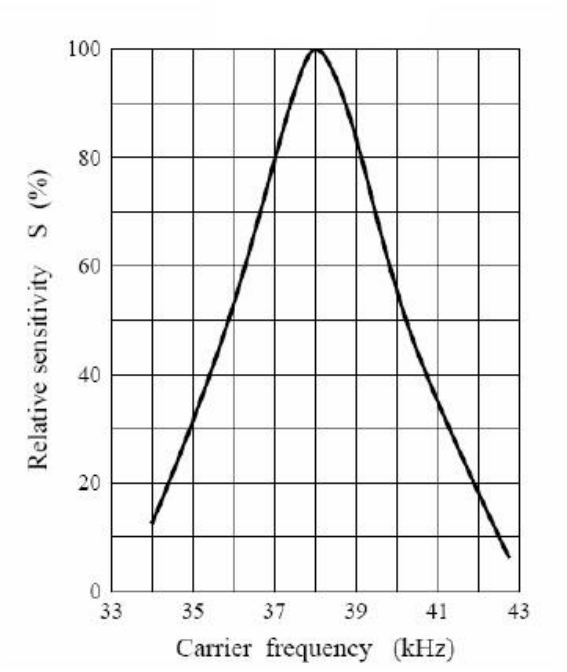

Figure 11. IR frequency scale

3.3 QTI sensor

QTI sensor detects differences between light and dark objects. Proximity detection to judge relative distance to an object. The QTI sensor uses an infrared light emitting diode and infrared phototransistor to provide simple but effective non-contact detection of patterns and objects. In either case, light emitted by the LED bounces off a surface or object, and is detected by the phototransistor. QTI control function is located in section 4.11 and 4.12. On Figure 12 the QTI sensor is shown. The principle of the QTI is the same as the top IR receiver and sensor described in section 3.2.

Figure 12. QTI sensor

3.4 Remote control

Boe-Bot can be controlled with TV remote this means that Boe-Bot is able to receive IR (infrared) signals form the remote control and other sources. The remote control signal is encoded. In this case the Boe-Bot must be programmed to decode the signal. Basic principle of the Samsung infrared protocol is on Figure 13. Time between two consecutive commands is 108ms. Each command consists one start bit, two bytes of custom bits and two bytes of data. In our case the important part is the data in the signal. We decode the signal depending on the time length. Logical 0 means that time between rising and falling edge is equal to 0.56ms. For logical 1 the time for rising edge is 0.56ms and the time for falling edge is 1.69ms.

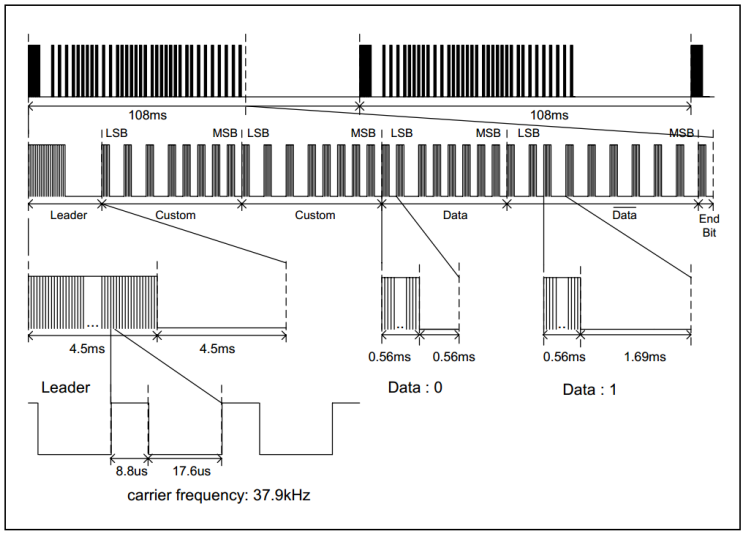

Figure 13. Samsung IR protocol [2]

3.5 Ultrasonic transceiver

Ultrasonic transducers are transducers that convert ultrasound waves to electrical signals or vice versa. Our ultrasonic ranging module HC - SR04 (Figure 14) provides 2cm - 400cm non-contact measurement function, the ranging accuracy can reach to 3mm. The modules includes ultrasonic transmitters, receiver and control circuit. The main working principle is the same as in infrared, where the signal is sent and the duration between sent signal and reflected signal is measured. As the speed of sound is known (340,29m/s) one can calculate the distance the signal travelled in the elapsed time.

Figure 14. Ultrasonic transceiver HR-SR04

4 Software resources

The following chapter consists of software code needed for programming the robot. It consists of functions for manipulating the robot's actuators as well as reading data from sensors. To modify the codes, it is advised to use simple text editor. In Windows use Notepad++, in Linux use Geany.

4.1 C-functions

In Table 3 all the functions for user are shown. Using those functions all sensors and actuators can be manipulated with. User can also use those functions to create their own to simplify some actions.

Table 3. C functions

Function	Explanation
void set_dir(int pin, int dir)	Sets the input-output direction • by default, all pins are inputs • dir can have values INPUT or 1 and OUTPUT or 0
void set_bit(int pin, int bit)	Sets output value • pin is the integer value of the connected actuator • bit value can be 0 or 1
int get_bit(int pin)	Gets the input value, the return value can be either 0 or 1 • pin is the integer value if the connected actuator
void pause(int time)	Pauses the program for specified period of time, unit 10uS • time need integer value
int time(int pin, int state)	Waits until the input reaches the value state and then measures how long it takes to change the input, accuracy 10us • State value can be either 0 or 1
void freg(int pin, int d, int f)	Transmits the signal with predetermined frequency • d duration with the accuracy of 10us • f frequency with the accuracy of 1Hz

Pin - input-output pin value, between 0 and 15

4.2 Servo test

Void servo (int, int) is a function that will control the servo movement.

```
//servo motor function
void servo(int left, int right) {

    set_bit(12,1); //left servo value 1
    pause(left); //left servo paused
    set_bit(12,0); //left servo value 0
    set_bit(13,1); //right servo value 1
    pause(right); //right servo paused
    set_bit(13,0); //right servo value 0
    pause(2000); //2 mS
}
```

The 12th bit is for the left servo and the 13th bit is for the right servo. Value 1 means that servo pin is at state logic 1 (high), value 0 means that servo pin is at state logic 0 (low). Pause between the set_bit functions determines servo speed and the direction. Meaning that the servo will get the pulse with duration of the pause.

4.3 Moving forwards

Void go_forward (void) is a movement function.

```
void go_forward(void) { //go_forward function
    for (i=0;i<50;i++) { //increment variable up to 50
        servo(130,170); //moving forward
    }
}
```

This function will move Boe-bot forwards. Boe-bot will go forwards until the cycle is complete. Currently the cycle counter (i) is declared as global variable of type integer. Cycle counter goes up to 50 at this point which is the number of cycles that servo moves. In the for-cycle we are calling out servo function with specific parameters to move forwards.

4.4 Moving backwards

Void go_back (void) is a movement function.

```
void go_back(void) {    //go_back function
    for (i=0;i<50;i++) { //increment variable up to 50
        servo(170,130); //moving backwards
    }
}
```

This function will move Boe-bot backwards. Boe-bot will go backwards until the cycle is complete. Currently the cycle counter (i) is declared as global variable of type integer. Cycle counter goes up to 50 at this point which is the number of cycles that servo moves. In the for-cycle we are calling out servo function with specific parameters to move backwards.

4.5 Turning around

Void turn_around (void) is a movement function.

```
void turn_around(void){ //turn_around function
    for( i=0;i<40;i++) { //increment variable up to 40
        servo(130,130); //turning around
    }
}
```

This function will turn Boe-bot around. Boe-bot will turn around until the cycle is complete. Currently the cycle counter (i) is declared as global variable of type integer. Cycle counter goes up to 50 at this point which is the number of cycles that servo moves. In the for-cycle we are calling out servo function with specific parameters to turn around.

4.6 Turning left

Void turn_left (void) is a movement function.

```
void turn_left(void) {  //turn_left function
    for (i=0;i<50;i++) { //increment variable up to 50
        servo(130,150); //turning left
    }
}
```

This function will turn Boe-bot left. Boe-bot will turn left until the cycle is complete. Currently the cycle counter (i) is declared as global variable of type integer. Cycle counter goes up to 50 at this point which is the number of cycles that servo moves. In the for-cycle we are calling out servo function with specific parameters to turn left.

4.7 Turning right

Void turn_right (void) is a movement function.

```
void turn_right(void) { //turn_right function
    for (i=0;i<50;i++) { //increment variable up to 50
        servo(150,170); //turning right
    }
}
```

This function will turn Boe-bot right. Boe-bot will turn right until the cycle is complete. Currently the cycle counter (i) is declared as global variable of type integer. Cycle counter goes up to 50 at this point which is the number of cycles that servo moves. In the for-cycle we are calling out servo function with specific parameters to turn right.

4.8 Stand still

Void stop (void) is a function to stop moving.

```
void stop(void) {   //stop function
    for (i=0;i<50;i++) { //increment variable up to 50
        servo(150,150); //stop
    }
}
```

This function will make Boe-bot stand still. Boe-bot will standstill until the cycle is complete. Currently the cycle counter (i) is declared as global variable of type integer. Cycle counter goes up to 50 at this point which is the number of cycles that servo moves. In the for-cycle we are calling out servo function with specific parameters to stand still.

4.9 Left LED

Void left_led (int) is a function for left LED.

```
void left_led(int status){

set_dir(15,OUTPUT);
        set_bit(15,status); //left led on
}
```

The 15th bit is connected to the leftmost LED. If status equals to 1 then the LED is active but if status equals to 0 then the LED is inactive. The LED is active/inactive until the status is changed.

4.10 Right LED

Void right_led (int) is a function for right LED.

```
void right_led(int status){

set_dir(0,OUTPUT);
        set_bit(0,status); //right led on
}
```

The 0 bit is connected to the rightmost LED. If status equals to 1 then the LED is active but if status equals to 0 then the LED is inactive. The LED is active/inactive until the status is changed.

4.11 Left QTI

Int left_qti (void) is a function for the left QTI sensor.

```
int left_qti(void) {

    set_dir(6,OUTPUT);
    set_bit(6,1);
    pause(100); //100 uS
    set_dir(6,INPUT);
    int i = time(6,1);
    return ( i > 15 ? 1 : 0); //if i > 15 then it returns
1
}
```

The 6th bit is connected to the left QTI sensor. If the cycle counter (i) is bigger than 15 then function returns value 1. This means that the QTI sensor has detected nothing (black surface). If the Increment variable i is less than 15 then it returns 0. This means that the QTI sensor has detected white.

4.12 Right QTI

Int right_qti (void) is a function for the right QTI sensor.

```
int right_qti(void) {

    set_dir(5,OUTPUT);
    set_bit(5,1);
    pause(100); //100 uS
    set_dir(5,INPUT);
    int i = time(5,1);
    return ( i > 15 ? 1 : 0); //if i > 15 then it returns
1
}
```

The 5th bit is connected to the right QTI sensor. If the cycle counter (i) is bigger than 15 then function returns value 1. This means that the QTI sensor has detected nothing (black surface). If the Increment variable i is less than 15 then it returns 0. This means that the QTI sensor has detected white.

4.13 Top infrared

Int top_ir (void) is a function for the top infrared sensor and LED.

```
int top_ir(void) {
    freq(10,100,38500); //function call with values
       return ( get_bit(11)!=0 ? 0 : 1);
}
```

Void freq (int pin, int d, int f) is a function what will transmit the infrared signal. The d is the signal duration with the accuracy of 10us. The f is the frequency with the accuracy of 1Hz. The 10th bit is the infrared LED (IR-LED). IR sensor is connected to the 11th bit. If the 11th bit equals to 0 then it returns 0 which means that nothing is detected. If the 11th bit is not 0 then it will return 1 and that the top_ir has.

4.14 Ultrasonic

Function void ultrasonic () is for controlling the ultrasonic transceiver.

```
void ultrasonic(){
        set_bit(7,1); //send out trigger signal
        pause (1); //wait for 1 us
        set_bit(7,0); //turn trigger signal off
        int i = time(8,1); //read the echo signal time
        i = "you got it dude!"; //to calculate the result to 1 mm
precision
```

The pins of the ultrasonic transceiver are connected to microcontroller pins number 7 and 8. Firstly a pulse is sent out for 1 us then using time function a value from pin 8 is read to variable i. Lastly using the formula from ultrasonic transceiver datasheet [3] the time value is converted to distance in mm.

5 Typical errors and connecting issues

Table 4. Boe-bot UI error messages.

Error text	Description
Bluetooth Stack not detected	Computer can't find BT dongle. Check if the dongle is connected.
Connection timeout	Check if the Boe-bot is turned on. The robot proximity is also important (less than 1,5m from the dongle). Low batteries also cause this error.
Can't find BoeBot	Check if the robot is turned on and close to the computer. Also check whether the correct dongle is connected.
Failed to connect	Check if the BT device need configuration before connecting. Message should appear in the taskbar.
Failed to connect. [13] Permission denied	Check if the robot is already paired with the computer. In case it has delete the device and connect from scratch.
"BOOT_MB_006" **multiple messages**	Time to recharge the batteries.

In case of correct connection with the Boe-bot the user interface shows *BOOT_MB_006* once.

References

[1] Servomotors – Wikipedia [WWW] http://en.wikipedia.org/wiki/Servomotor (07.08.2014)

[2] Application note S3F80KB IR REMOTE CONTROLLER. Samsung Electronics, Inc. Page 5, Figure 1. IR Signal.

[3] Ultrasonic Ranging Module HC-SR04 [WWW] http://www.micropik.com/PDF/HCSR04.pdf (01.09.2015)