Spring 3-2009

“HUMANOID ROBOT” “PROGRAMMING OF MOVEMENT PATTERNS” “HUMANOID - BIOLOID”

Faton Bekteshi

Follow this and additional works at: https://knowledgecenter.ubt-uni.net/etd

Part of the Computer Sciences Commons
“HUMANOID ROBOT”
“PROGRAMMING OF MOVEMENT PATTERNS”
“HUMANOID - BIOLOID”

A Bachelor Thesis submitted for the degree of
“Bachelor of Science in Computer Science and Technology”
at the University for Business and Technology

UBT

Supervised by:
Univ.Prof.Dipl.-Ing.Dr.Dr.h.c.mult. Peter Kopacek

Faton Bektashi
Id no: 2006070043 / B

March – 2009
# Contents

**Abstract** ................................................................................................................................................. 5

1. **Introduction** ........................................................................................................................................ 6

1.1 Bioloid History ...................................................................................................................................... 6

1.2 Purposes of this thesis ............................................................................................................................ 7

1.3 Control solutions for stabilizing under actuated robots ........................................................................ 9

2. **Software Utilities** ................................................................................................................................. Error! Bookmark not defined.

2.1 Introduction to Motion Editor .................................................................................................................. Error! Bookmark not defined.

2.2 Introduction to Behaviour Control Program ............................................................................................ Error! Bookmark not defined.

2.3 Introduction to Programming in C .......................................................................................................... 15

3. **Software Description** ............................................................................................................................ Error! Bookmark not defined.

3.1 Motion Editor .......................................................................................................................................... 16

3.2 Behaviour Control Program .................................................................................................................... 23

3.3 Programming in C .................................................................................................................................. 26

3.3.1 Communication ................................................................................................................................. 267

3.3.2 Recharge ........................................................................................................................................... 264

4. **Future Work** ......................................................................................................................................... 58

5. Bibliography................................................................................................................................................. 59
Preface

This thesis would never be done without the fully support, orientation, dedication and availability of our supervisors, Prof. Dr. Peter Kopacek and Msc. Artan Dermaku.

We must thank all our colleagues from “UBT-Skanderbeg Team”, for the friendly work environment they provided (and their still doing), for the help they gave to us in many different occasion and problems we had during programming robot in our Robotics laboratory.

We would like also to add a special thanks to our Commanders and Commanding chain from Military Academy, for fully support with “punishments and isolations” they used to give us, so we had the time of the world working in this thesis direction, otherwise we wouldn’t make it in time.

And least but not last, we shall express, from the bottom of our hearts, our deepest appreciation to our parents and family for the support, dedication, understanding and love.
Abstract

This thesis was developed in collaboration with Univ.Prof.Dipl.-Ing.Dr.Dr.h.c.mult. Peter Kopacek and Msc. Artan Dermaku and boosted the creation of the Humanoid Robotics Laboratory of UBT, at University for Business and Technology in Prishtina. ([http://www.ubt-uni.net/](http://www.ubt-uni.net/)).

The main objective of the Humanoid Robot, for the time being, is to develop a humanoid robot able to make complex motions like walking, running and jumping through real-time feedback control techniques.

This exposition presents a Linear Quadratic Regulator (LQR) controller for the simulation and control of the humanoid robot. Also our objective during programming the robot was to make him able improving military skills into it like, making pushups, saluting, walking through line without having problems, make turns in case of danger zones and unlike positions which would make him think and feel the danger.

Another important improvement during programming was idea to make him defend himself in case of touching him, make sensors feel the hand and react in those cases.

This thesis is divided in co-work and it's divided into two places:
1. Movement Patterns of Bioloid by Jeton Ahmeti (chapter 2, chapter 3)
2. Programming of Movement patterns of Bioloid by Faton Bektishi

Keywords: Humanoid Robots, Humanoid Identification, Humanoid Simulation, Linear Quadratic, Regulator (LQR).

---

1 Refer Jeton Ahmeti thesis “Movement patterns of Bioloid” chapter 2 and 3
1. Introduction
We are all, human and humanoid alike, whether made of flesh or of metal, basically just sociable machines.

Robin Marantz Henig

1.1. Bioloid History

A long standing desire that human-like robots could co-exist with human beings has made the researchers think that the humanoid robotics industry will be a leading industry in the twentieth-first century (Kim et al., 2007). This thought comes from the fact that technology is finally getting ready for this purpose. Fastest microprocessors, super computers, high-torque servo-actuators, precise sensors along with new advances in control techniques, artificial intelligent and artificial sound/vision recognition, all embedded in better and better mechanical design machines made the believe that this dream might became true in a nearly future. But, humanoid robots will not only be able to socialize with the human-being but will also be able to replace him even in the tedious and dangerous tasks, ranging from rescuing situations to interplanetary exploration.

Robot is a virtual or mechanical artificial agent. In practice, it is usually an electro-mechanical system which, by its appearance or movements, conveys a sense that it has intent or agency of its own. The word robot can refer to both physical robots and virtual software agents, but the latter are usually referred to as bots. There is no consensus on which machines qualify as robots, but there is general agreement among experts and the public that robots tend to do some or all of the following: move around, operate a mechanical limb, sense and manipulate their environment, and exhibit intelligent behavior, especially behavior which mimics humans or other animals. Stories of artificial helpers and companions and attempts to create them have a long history but fully autonomous machines only appeared in the 20th century. The first digitally operated and programmable robot, the Unimate, was installed in 1961 to lift hot pieces of metal from a die casting machine and stack them. Today, commercial and industrial
robots are in widespread use performing jobs more cheaply or with greater accuracy and reliability than humans. They are also employed for jobs which are too dirty, dangerous or dull to be suitable for humans. Robots are widely used in manufacturing, assembly and packing, transport, earth and space exploration, surgery, weaponry, laboratory research, and mass production of consumer and industrial goods.

1.2 Purposes of this thesis

And only after achieving the natural walking and locomotion of a humanoid in the human environment, are humanoid robots able to learn how to interact with it and socialize with humans, making use of all of its artificial intelligent. Walking locomotion is not a trivial concept for the human understanding. In fact, only recently studies (Sockol et al., 2007) demonstrated, by comparing the oxygen consumption of humans and chimpanzees while walking on a treadmill, that the human-being evolved to walk upright in two legs (bipedalism) since it makes the walking far more efficient in terms of wasting energy.

Understanding the natural and smooth walking of a human is also another challenge since generating a stable walking motion for this multi-body system, which is highly nonlinear, is a very complex one.

The most common strategy, nowadays, and based on dynamic walking, are the zero moment point (ZMP) (Kim et al., 2007) and the contact wrench sum (CWS) (Hirukawa et al., 2007). These techniques, which main principle is to cancel the total inertial forces actuating on the humanoid with the floor reaction force, are implemented in some of the most famous researcher humanoid robots like QRIO from Sony, ASIMO from Honda or HRP-2/HRP-3 from Kawada, allowing them to walk on uneven terrain and inclined plans, to run and to climb stairs. Another control strategy is based on biologically realistic walking (Popovic and Herr, 2004) and on the principle of spin angular momentum regulation.
Recently, a dynamic balancing strategy control has also been successfully applied to the Dexter humanoid robot from Anybots. In this case and as opposite of ZMP strategy, it does not need preprogrammed footprints, being able to walk like a human and even to jump. Many other strategies have been studied, one last for instance is the passive-dynamic walking (PDW) that requires no external control or energy input, being the movement governed by the natural swinging of the legs (Collins et al., 2005; Asano and Luo, 2007).

In terms of current commercially available humanoid robots, they are still designed to perform motions using open-loop control providing the users a simple paradigm to create pre orchestrated multi-DOF walking gaits. These robots are usually not able to move on uneven terrain and it is difficult or impossible to get them to perform movements that require instantaneous reaction to momentary instability.

A popular way to compensate for these predicaments is to over-capacitate servo torques and to incorporate large foot soles, low center-of-mass and better shock absorption, resulting in humanoid robots with little resemblance to the human physique, just as RoboSapiens from Wow Wee Toys (Figure 1.1b).

Fig. 1.1: The humanoid robot from Honda and a commercial humanoid toy from Wow Wee Toys (http://www.electronickits.com/robot/Bioloid)
The long term objectives of this thesis are to allow affordable commercial humanoid robots to walk, run, skateboard, and jump and in general to react in a human-like physical way in dynamically unstable situations and uneven terrain. These goals can be achieved by applying closed-loop control techniques to the humanoid robot servos. The input data stream should consist of a multitude of sensors including servo position and torque, acceleration and inertial moment. The closed loop control cycle should actuate the servos at rates of, at least 50Hz, which would give good responsiveness in a dynamic environment.

This thesis, however, has a more humble goal since it initiates the study of humanoids. Therefore, it is necessary to prepare the necessary conditions before achieving the desired long term goals.

1.3 Control solutions for stabilizing under actuated robots

Many studies were done with under actuated robots (Lee and Cover stone Carroll, 1998; Aurelie et al., 2006). These robots are generally composed of two or three links
in which the first joint is not actuated, or passive, whereas the others are actuated or active (Figure 1.3). The objective is to swing up the robot from the vertical stable position to the upside-down position and then maintaining its unstable pose.

**Fig 1.3:** A 3-link under actuated robot in the upside-down region (http://www.electronickits.com/robot/Bioloid)

Stabilizing the humanoid in his upright position is a challenging task and requires non-cancelation control techniques since the system is non-minimum phase. The most common controller used as a first attempt is the Linear Quadratic Regulator (LQR) (Spong, 1994; Lee and Coverstone Carroll, 1998). In this type of controller, a vector of constant gains (LQR gains or Kalman gains) are applied to the vector state of the system, obtaining new torque inputs for the system in order to stabilized it.

Another popular technique uses Partial Feedback Linearization to swing up a two-link under-actuated robot (Spong and Block, 1995) and to stabilize it (Lee and Coverstone Carroll, 1998). This control strategy derives from fully actuated robots, with no passive joints. For this type of under actuated robots, the model can be fully feedback liberalized by a nonlinear feedback law (Spong and Vidyasagar, 1989). However, for under actuated robots, that is only true for the actuated joints (Partial Feedback Linearization).

The rest of the dynamics of the system would still remain nonlinear. The solution found by (Spong, 1994) was to introduce a new condition called, Strong Inertial
Coupling, to linearism also the dynamics corresponding to the passive joints. In this way, it was possible to feedback linearism an under actuated planar robot. Partial Feedback Linearization demands a full knowledge of the model. Aiming a more robust stabilization technique that could handle model uncertainty, other techniques, like for instance, the nonlinear Sliding Mode Control (Utkin, 1992) technique was took in consideration for stabilizing a two-link under actuated robot in its upright unstable position (Lee and Cover stone-Carroll, 1998) and for the both phases, swing-up and stabilization (Qian et al., 2007).

In this strategy, an additional term responsible for handling the uncertainties of the model is summed to a feedback liberalizing controller improving the overall robustness of the controller. Intelligent control has also been used to control under actuated robots. Among other strategies, an intelligent adaptive fuzzy radial Gaussian neural networks system for stabilizing a two-link under actuated robot in vertical unstable position (Qian et al., 2006) demonstrated to be globally stable, while an adaptive GA-tuning fuzzy PID control scheme, for the swing-up and stabilization of the same under actuated robot, has been implemented with some successfully results (Wu et al., 2007).

Using the included motion editor software and visual programming environment, you can make the Bioloid interact with its surroundings and perform complex movements. Motions are built up frame-by-frame like a story board in an animation sequence. This allows quite complicated movements to be programmed. Once a motion has been defined it can then be downloaded into the Bioloid's memory and called from the Behavior Control Program. The Bioloid comes with several example programs to make it walk, avoid obstacles and interact with sound. This is an impressive platform for robotics allowing for many configurations, many ways to sense the environment, and a well-developed system for programming actions.
2. Software Utilities

There are two software utilities that come with the Bioloid kit, these are:

1. **Motion Editor** - a GUI that allows you to create motion sequences for your robot, &
2. **Behavior Control Program** - a GUI that allows you to program sequences of events/motions.

**Motion Editor** in conjunction with the **Behavior Control Program** allows the programmer to use the **Motion Editor** to generate the robots motion sequences and then use the **Behavior Control Program** to implement the logic (or intelligence) of the robot.

Both of these utilities are Freeware

2.1 Introduction to Motion Editor

![Interface of Motion Editor](http://www.super-science-fair-projects.com)

**Fig.2.1.1 Interface of Motion Editor (http://www.super-science-fair-projects.com)**

The motion editor has a graphical user interface that allows the user to edit a multi-jointed robot made up of Robotis servos. A user can create and edit motions by moving
the joints by hand and saving each pose using the motion editor. The user can also
connect or repeat edited motions.

The image to the left shows how motions are built up frame-by-frame - very similar to a
story-board in an animation sequence. This allows quite complicated "animations" to be
quickly programmed and tested.

Once a motion has been defined it can then be downloaded into the CM-5's FLASH and
called from the Behavior Control Program.

2.2 Introduction to Behavior Control Program

![Fig.2.2.1 Interface of Behavior Control Program](http://www.super-science-fair-projects.com)

A robot is a machine that can behave in various ways. However, it can do so only when
there is a program that tells how the robot should act for a certain situation. This
program is called the “Behavior control program." A Behavior control program is a series
of rules that define the action a robot should take for the given state.

A series of commands are entered that allow the programmer to interrogate ALL the
functionality of the AX-12, AX-S1 and the CM-5 mcu module. The functionality
accessible within the CM-5 includes playing motion sequences, external pushbuttons and the CM-5’s timer.

The commands provided with the Behavior Control Program include:

- program control commands (START, END),
- conditional branching commands (IF, ELSE IF, ELSE, CONT IF) with conditional operations ($=$, $>$, and $>=$, $<$, and $<=$),
- program sequencing commands (JUMP & CALL/RETURN),
- numeric commands (COMPUTE), and
- assignment commands (LOAD).

NB. Command lines can be given meaningful names of LABELS.

Another feature of the Behavior Control Program is the debugging function which allows variables to be displayed on the PC’s screen whilst the program is executing - making it very easy to calibrate AX-S1 sensors.

![Behavior control program process](http://www.super-science-fair-projects.com)
2.3 Introduction to Programming in C

The CM-5 controller board is based on the popular Atmel128 MCU. It is also possible to program the CM-5 module using freeware GNU GCC compiler WinAVR available by AVR Freaks. This gcc compiler is a great freeware tool, coupled with the editor Programmers Notepad.

This is an option for experienced programmers only though. You will need to start from the ground up - write all the communication protocol's from scratch. But the power that it gives you over the robot is "unlimited". For example, it is possible to create Closed-Loop control algorithms and distributed control by relaying sensory information to an external computer such as a Gumstics, or a remote PC via Bluetooth or ZigBee. If you are an experienced programmer, this robot kit is stuff for dreams...
3. Software Description

3.1 Motion Editor

Here we will see some examples of Humanoid Bioloid Robot, using Motion Editor.

In this case we wanted to make robot make Pushups, and we decided to use Motion Editor, and make him move up & down with the speed we waned him to make movements. It’s been quite easy implementing this kind of movements as you can see in the samples below how we store (drag & drop) movements.

Motion editor heled and makes our job quite easy implementing movements because you can store a move, increase & decrease the speed of the robot movement, move indicators of the robot in the positions you want to, which maked us store and finish the samples in quite short time.

Fig. 3.1.1 Interface of Motion Editor
As you all can see, when you firstly open the user interface of Motion editor everything is blank, no Robot detected, no space to do anything.

When you firstly open the page you have to be sure that you plugged in the cable which makes the communication with Robot & PC, otherwise there would be no signal which transmits data through robot.

After you plug in the cable, open File and then Connect (CM-5) which is going to connect Robot & Motion Editor with Robot. Fig 3.1.2 below:

After you click on Connect (CM-5) you’ll find another window Fig 3.1.3 which is going to ask you which port are you going to connect.

**Fig 3.1.2 Connecting robot with Motion Editor**
After we connected serial Port COM 1 it will show us a Robot which communicates with user Interface of Motion Editor Fig 3.1.4 and shows Robot in First Position.

In this Fig 3.1.4, we also can see frames “Pose 0, Pose 1, ...” which tells us the postions and movements of robot. In right side we also can see dark side of Frames.
“Pose 0, Pose 1, ...” because we didn’t store any of the movements still.

Fig 3.1.5 Robot in Second Position

Fig 3.1.5 shows us second movement of robot (we have to add here that movements of robot are made manually, so when we add a position we store it by ADD POSE which is down in motion editor below robot) after we add pose we get this movement (pose 1)

Fig 3.1.6 Robot Position
Fig 3.1.6 shows us third move of robot, stored in Pose Editor. It’s very important when we store a position to make ON & OFF in the middle of editor, because there are the coordinates of Bioloid which he has to play after we finish examples.

![Pose Editor](image)

**Fig 3.1.7 Robot Position**

Fig 3.1.7 (Pose 3) We stored next move, and the indicators in the middle of editor has changed because of values which robot takes automatically, which position he takes, the vectors of bioloid changes and values too.

Fig 3.1.8 (Pose 5) so on we stored the positions of robot and we completed one circle of robot movements as it has been showed in figure below.
In Fig. 3.1.9 we showed all necessary movements which are needed for robot making pushups, so we press Play Now and he would automatically start moving.

It’s important the speed of robot movement to be stored on Motion Editor interface because it will help us make him move the way we want to.
We also can Open or Save a robot movements from the Disk or Media we stored or we save it. It helps us save movements and start over new programming details and run the program in different cases. Fig 3.1.10

![Motion Editor](image)

**Fig 3.1.10** *Save Robot Position*

**Motion Editor** is a 3D interface for setting servo positions and motion sequences. Motions created in this program can be called and used by the Behavior Control Programmer.
3.2 Behavior Control Program

Another user Interface which helps us makes robot movements without being professionals. With B.C.P we can “Program” a robot movements using pretty easy programming language and save movements.

Behavior Control Program helps us store moves without writing them in any different programming languages or different user interfaces. As we can see below in Fig. 3.2.1 we start similar as in Motion editor, we have to Start it, Call a move, give a condition what kind of movement, speed, motion, and robot has to do.

![Fig. 3.2.1 Behavior Control Program Interface](image)
With Behavior Controller Program we can import, export, save, movements of robot. As it’s shown in Fig below, we can update new movement of robot; download it from the robot and save it in disk.

It’s important having robot connected and not having problem with connection, otherwise robot would not respond to pc and robot himself.

**Fig. 3.2.2 Update new program**

Here we see if we double click on a row, it immediate shows a meny downstairs with the cases which we want to implement for the next move of robot.

It also uses (have) GUI (photo) of devices that we want to implement, which makes us easy to use and prepare it for future work.
**Fig 3.3.3 Cases of Behavior Control Program**

**Behavior Control Programmer** is an application that sets rules for the robot’s behavior. You can string together series of motions created in Motion Editor, or create looping and branching behavior patterns based on sensor values. All of this is achieved via a graphical sequencer interface.
3.3 Programming in C

3.3.1 Communication:

/*
 * Based On "Example of Dynamixel Evaluation with Atmega128 by BS KIM"
 */

#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

#include "../include/communication.h"
#include "../include/uart.h"
#include "../include/interpolation.h"

// --- Global Variable Number ---
volatile byte gbpRxInterruptBuffer[256];
byte gbpParameter[128];
byte gbRxBufferReadPointer;
byte gbpRxBuffer[128];
byte gbpTxBuffer[128];
volatile byte gbRxBufferWritePointer;

// read len bytes from serial
void readData(byte *bData, int len)
{
    while (len-- > 0)
    {
        *bData = uart1_getc();
        bData++;
    }
}
// write len bytes to serial
void writeData(byte *bData, int len)
{
    while (len-- > 0)
    {
        uart1putc(bData++);
    }
}

void initMessageData(struct MessageData &msgData)
{
    msgData.msgIndex = 0;
    msgData.msgProcessed = false;
}

void sendStatusInformation(byte* sendBuffer, struct RobotData &robotData, struct InterpolationData &ipoData, struct SCurveParameters &params)
{
    sendBuffer[0] = (robotData.writeTimeDiffAverage & 0xFF);
    sendBuffer[1] = (robotData.writeTimeDiffAverage & 0xFF00) >> 8;
    sendBuffer[2] = (robotData.readTimeDiff & 0xFF);
    sendBuffer[3] = (robotData.readTimeDiff & 0xFF00) >> 8;
    sendBuffer[4] = (ipoData.ipoPause & 0xFF);
    sendBuffer[5] = (ipoData.ipoPause & 0xFF00) >> 8;
    sendBuffer[6] = robotData.writeBufferLength & 0xFF;
    writeData((byte*)sendBuffer, 7);
}

// init serial comm
void PortInitialize(void)
{
    DDRA = DDRB = DDRC = DDRD = DDRF = DDRG = 0; //Set all port to input direction first.
    PORTB = PORTC = PORTD = PORTE = PORTF = PORTG = 0x00; //PortData initialize to 0
cbi(SFIOR,2); // All Port Pull Up ready
DDRE |= (BIT_RS485_DIRECTION0|BIT_RS485_DIRECTION1); // set output the bit RS485 direction

DDRD |= (BIT_ZIGBEE_RESET|BIT_ENABLE_RXD_LINK_PC|BIT_ENABLE_RXD_LINK_ZIGBEE);

PORTD &= ~_BV(BIT_LINK_PLUGIN); // no pull up
PORTD |= _BV(BIT_ZIGBEE_RESET);
PORTD |= _BV(BIT_ENABLE_RXD_LINK_PC);
PORTD |= _BV(BIT_ENABLE_RXD_LINK_ZIGBEE);

} /*
TxPacket() send data to RS485.
 TxPacket() needs 3 parameter; ID of Dynamixel, Instruction byte, Length of parameters.
 TxPacket() return length of Return packet from Dynamixel.
 */

byte TxPacket(byte bID, byte bInstruction, byte bParameterLength)
{
    byte bCount,bCheckSum,bPacketLength;

    gbpTxBuffer[0] = 0xff;
    gbpTxBuffer[1] = 0xff;
    gbpTxBuffer[2] = bID;
    gbpTxBuffer[3] = bParameterLength+2; // Length(Paramter,Instruction,Checksum)
    gbpTxBuffer[4] = bInstruction;

    for(bCount = 0; bCount < bParameterLength; bCount++)
    {
        gbpTxBuffer[bCount+5] = gbpParameter[bCount];
    }

    bCheckSum = 0;
    bPacketLength = bParameterLength+4+2;

    for(bCount = 2; bCount < bPacketLength-1; bCount++) // except 0xff, checksum
    {
        bCheckSum += gbpTxBuffer[bCount];
    }
GBP TX Buffer[bCount] = ~bCheckSum; //Writing Checksum with Bit Inversion

RS485_TXD;
for(bCount = 0; bCount < bPacketLength; bCount++)
{
    sbi(UCSR0A,6); //SET_TXD0_FINISH;
    TxD80(gbpTxBuffer[bCount]);
}

while(!CHECK_TXD0_FINISH) //Wait until TXD Shift register empty
    ;// nothing

RS485_RXD;

return(bPacketLength);
}

/*
RxPacket() read data from buffer.
RxPacket() need a Parameter; Total length of Return Packet.
RxPacket() return Length of Return Packet.
*/

byte RxPacket(byte bRxPacketLength)
{
    #define RX_TIMEOUT_COUNT2 3000L
    #define RX_TIMEOUT_COUNT1 (RX_TIMEOUT_COUNT2*10L)
    
    unsigned long ulCounter;
    byte bCount, bLength, bChecksum;
    byte bTimeout;

    bTimeout = 0;
    for(bCount = 0; bCount < bRxPacketLength; bCount++)
    {
        ulCounter = 0;
while(gbRxBufferReadPointer == gbRxBufferWritePointer)
{
    if(ulCounter++ > RX_TIMEOUT_COUNT1)
    {
        bTimeout = 1;
        break;
    }
}

if(bTimeout)
    break;

    gbpRxBuffer[bCount] = gbpRxInterruptBuffer[gbRxBufferReadPointer++];
}
bLength = bCount;
bChecksum = 0;

if(gbpTxBuffer[2] != BROADCASTING_ID)
{
    if(bTimeout && bRxPacketLength != 255)
    {
        //TxDString("\n [Error:RxD Timeout]");
        CLEAR_BUFFER;
    }
}

if(bLength > 3) //checking is available.
{
    if(gbpRxBuffer[0] != 0xff || gbpRxBuffer[1] != 0xff )
    {
        //TxDString("\n [Error:Wrong Header]");
        CLEAR_BUFFER;
        return 0;
    }
}

{
    //TxDString("\n [Error:TxD != RxID]");
}
CLEAR_BUFFER;
return 0;
}

if(gbpRxBuffer[3] != bLength-4)
{
    //TxDString("\n\n [Error:Wrong Length]");
    CLEAR_BUFFER;
    return 0;
}

for(bCount = 2; bCount < bLength; bCount++)
    bChecksum += gbpRxBuffer[bCount];

if(bChecksum != 0xff)
{
    //TxDString("\n\n [Error:Wrong CheckSum]");
    CLEAR_BUFFER;
    return 0;
}
}
return bLength;

/*Hardware Dependent Item*/
#define TXD1_READY bit_is_set(UCSR1A,5) //UCSR1A_Bit5
#define TXD1_DATA (UDR1)
#define RXD1_READY bit_is_set(UCSR1A,7)
#define RXD1_DATA (UDR1)

#define TXD0_READY bit_is_set(UCSR0A,5)
#define TXD0_DATA (UDR0)
#define RXD0_READY bit_is_set(UCSR0A,7)
#define RXD0_DATA (UDR0)

/*
SerialInitialize() set Serial Port to initial state.
SerialInitialize() needs port, Baud rate, Interrupt value.
*/

void SerialInitialize(byte bPort, byte bBaudrate, byte bInterrupt)
{
    if(bPort == SERIAL_PORT0)
    {
        UBRR0H = 0; UBRR0L = bBaudrate;
        UCSR0A = 0x02;  UCSR0B = 0x18;
        if(bInterrupt & RX_INTERRUPT)
            sbi(UCSR0B,7); // RxD in
            ubrr(interrupt & RX_INTERRUPT)
        sbi(UCSR0B,7); // Rx interrupt enable
        UCSR0C = 0x06; UDR0 = 0xFF;
        sbi(UCSR0A,6);//SET_TXD0_FINISH; // Note. set 1, then 0 is read
    }
    else if(bPort == SERIAL_PORT1)
    {
        UBRR1H = 0; UBRR1L = bBaudrate;
        UCSR1A = 0x02;  UCSR1B = 0x18;
        if(bInterrupt & RX_INTERRUPT)
            sbi(UCSR1B,7); // RxD interrupt enable
        if(bInterrupt & TX_INTERRUPT)
            sbi(UCSR1B,6); // RxD interrupt enable
        UCSR1C = 0x06; UDR1 = 0xFF;
        sbi(UCSR1A,6);//SET_TXD1_FINISH; // Note. set 1, then 0 is read
    }
}

/*
TxD80() send data to USART 0.
*/

void TxD80(byte bTxdData)
while(!TXD0_READY);
TXD0_DATA = bTxdData;

/*
TXD81() send data to USART 1.
*/
void TxD81(byte bTxdData)
{
    while(!TXD1_READY);
    TXD1_DATA = bTxdData;
}

/*
RxD81() read data from UART1.
RxD81() return Read data.
*/
byte RxD81(void)
{
    while(!RXD1_READY);
    return(RXD1_DATA);
}

byte RxD81Async(void)
{
    return(RXD1_DATA);
}

/*
SIGNAL() UART0 Rx Interrupt - write data to buffer
*/
SIGNAL(SIG_UART0_RECV)
{
    gbpRxInterruptBuffer[gbRxBufferWritePointer++] = RXD0_DATA;
}
3.1.2 Recharge:

// system
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>
#include <util/delay.h>
#include <math.h>

// User
#include "./include/constants.h"
#include "./include/types.h"
#include "./include/communication.h"
#include "./include/timer.h"
#include "./include/crc.h"
#include "./include/uart.h"
#include "./include/interpolation.h"
#include "./include/recharge.h"

// recharge batteries functions
// thanks to kess and pieddemamouth (robosavvy-forums)
unsigned int readADC(byte channel)
{
    ADMUX = 0x40 | channel;
    ADCSRA = 0xC6;
    while(bit_is_set(ADCSRA, ADSC));
    return (ADCL | (ADCH<<8));
}

float getTrimmedAverage(unsigned int *voltage, int len, int trim)
{
    unsigned int min;
    int minId, i, j;
    for (i=0; i<len; i++)
    {
        min = voltage[i];
        }
minId = i;
for (j=i+1; j<len; j++)
    if (voltage[j] < min)
        {
            minId = j;
            min = voltage[j];
        }

min = voltage[i];
voltage[i] = voltage[minId];
voltage[minId] = min;
}

// calc average (without REMOVECYCLES largest and smallest values)
unsigned long int tmp = voltage[trim];
for (i=trim+1; i<len-2*trim; i++)
    {
        tmp += voltage[i];
    }

return (float) tmp / ((float)(len-2*trim));
}

void rechargeBatteries(unsigned int maxCycles)
{
    // configuration
    cli();

    PORTE |= _BV(PE4);  //enable pull-up resistor 4
    EIMSK |= _BV(INT4); //enable interrupt 4
    DDRC = 0xFF;
    PORTC = 0xFF;
    DDRB |= _BV(DDB5); //output for bit PB5
    //ADC configuration
    ADMUX = 0x10;
    ADCSRA = 0x80;
sei();

unsigned int voltage[RECHARGE CYCLES], voltageCounter, counter, average, oldAverage;
byte buffer[8];

// enable charging
PORTB &= ~PB5;

counter = 0;
voltageCounter = 0;
average = 0;
oldAverage = 10000;

while (true)
{
  voltage[voltageCounter] = readADC(1);
  voltage[voltageCounter] *= 2;
  voltageCounter++;

  if (voltageCounter >= RECHARGE CYCLES)
  {
    voltageCounter = 0;

    average = (unsigned int) (getTrimmedAverage(voltage, RECHARGE CYCLES,
    RECHARGE REMOVE CYCLES) + 0.5);

    if (average >= oldAverage) // unchanged
      counter++;
    else
    {
      counter = 0; // reset counter if changed
      oldAverage = average;
    }
  }
}

if (counter >= maxCycles)
buffer[0] = 0xFF;
buffer[1] = 0xFF;
buffer[2] = 0xFF;
buffer[3] = 0xFF;
buffer[4] = 0xFF;
buffer[5] = 0xFF;
buffer[6] = 0xFF;
buffer[7] = 0xFF;
writeData(buffer, 8);
break;

buffer[0] = voltage[(voltageCounter - 1) % RECHARGECYCLES] & 0xFF;
buffer[1] = (voltage[(voltageCounter - 1) % RECHARGECYCLES]) >> 8;
buffer[2] = average & 0xFF;
buffer[3] = (average) >> 8;
buffer[4] = counter & 0xFF;
buffer[5] = (counter & 0xFF00) >> 8;
buffer[6] = maxCycles & 0xFF;
buffer[7] = (maxCycles & 0xFF00) >> 8;
writeData(buffer, 8);

sleep(RECHARGEPAUSE); //100ms

// disable charging
PORTB |= PB5;

CRC:

#include "..\include\types.h"

/* crc-ccitt mask */
#define poly 0x1021
static word crc_table[256] = {
0x0000, 0x1189, 0x2321, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
0x8c48, 0x99dc, 0xa0f5a, 0xb0bed3, 0xc1a6c, 0xd1dbe5, 0xe197e, 0xf1f8f7,
0x10b1, 0x20108, 0x303393, 0x40221a, 0x5056a5, 0x60472c, 0x7075b7, 0x80643e,
0x909cc9, 0xa08d40, 0xb0xbfd, 0xc0xae52, 0xd0xdcb, 0xe0x0f9f, 0xf0xe876,
0x80x2102, 0x90x308b, 0xa0x0210, 0xb0x1399, 0xc0x6726, 0xd0x76af, 0xe0x4434, 0xf0x55bd,
0x10xad4a, 0x20xbcc3, 0xb0x9fd1, 0xc0xeb6e, 0xd0x0fae7, 0xe0xcc7c, 0xf0x0d9f5,
0x20x3183, 0x30x200a, 0x10x1291, 0x00x0318, 0x70x77a7, 0x60x662e, 0x50x54b5, 0x40x453c,
0x00xbdcb, 0x10xac42, 0x20x9ed9, 0x30x8f50, 0x40xfbef, 0x50x0e6a66, 0x60x8d8f, 0x70xc974,
0x80x4204, 0x90x538d, 0xa0x6116, 0xb0x709f, 0xc0x0420, 0xd0x15a9, 0xe0x2732, 0xf0x36bb,
0x10xce4c, 0x20xdfc5, 0x30xed5e, 0x40xf7d7, 0x50x8868, 0x60x99e1, 0x70x0ab7a, 0x80x0baf3,
0xb0x5285, 0xc0x430c, 0xd0x7197, 0xe0x601e, 0xf0x141a, 0x00x0528, 0x10x37b3, 0x20x263a,
0x30xdedc, 0x40xcf44, 0x50xfd9f, 0x60xec56, 0x70x99e9, 0x80x8960, 0x90xbbbf, 0xa0x0aa72,
0xb0x6306, 0xc0x728f, 0xd0x4014, 0xe0x519d, 0xf0x0252, 0x00x34ab, 0x10x0630, 0x20x17b9,
0x30x4ef4e, 0x40xfec7, 0x50xc465c, 0x60xddd5, 0x70x96a6, 0x80xb8e3, 0x90x8a78, 0xb0x9bf1,
0x10x7387, 0x20x620e, 0x30x5095, 0x40x0411c, 0x50x35a3, 0x60x242a, 0x70x16b1, 0x80x0738,
0x90x0f7d, 0xa0x46e46, 0xb0xcdcd, 0xc0xcd54, 0xd0xb9eb, 0xe0x8a62, 0xf0x99a9, 0x00x8b70,
0x10x8408, 0x20x9581, 0x30xa71a, 0x40xb693, 0x50xc22c, 0x60xd3a5, 0x70xe13e, 0x80xf0b7,
0x90x0840, 0xa0x19c9, 0xb0x2b52, 0xc0x3adb, 0xd0x4e64, 0xe0x05fed, 0xf0x6d76, 0x00x07cf,
0xb0x9489, 0xc0x8500, 0xd0xb79b, 0xe0xa612, 0xf0xd2ad, 0x00xc324, 0x10xf1bf, 0x20xe036,
0x30x18c1, 0x40x0948, 0x50x3bd3, 0x60x2a5a, 0x70x5e5e, 0x80x4f6c, 0x90x7d7f, 0xa0x6c7e,
0xb0xa5a, 0xc0xb483, 0xd0x8618, 0xe0x9791, 0xf0xe32e, 0x00xf2a7, 0x10xc03c, 0x20x0d1b5,
0x30x2942, 0x40x38cb, 0x50x0a50, 0x60x1bd9, 0x70x6f66, 0x80x7eef, 0x90x4c74, 0xa0x55dfd,
0xb0xb58b, 0xc0xa402, 0xd0x9699, 0xe0x8710, 0xf0x03af, 0x00xe22e, 0x10xd0bd, 0x20xc134,
0x30x39c3, 0x40xb84a, 0x50x1ad1, 0x60x0b58, 0x70x7f7e, 0x80x6e6e, 0x90x5cf5, 0xa0x4d7c,
0xb0xc60c, 0xc0xd7b5, 0xd0x0e51, 0xe0xf497, 0xf0x8028, 0x00x91a1, 0x10xa33a, 0x20xb2b3,
0x30x4a44, 0x40x5bcd, 0x50x6956, 0x60x78df, 0x70x0c60, 0x80x1de9, 0x90x2f72, 0xa0x3efb,
0xb0x6d8d, 0xc0x704f, 0xd0xf59f, 0xe0x4e16, 0xf0x90a9, 0x00x8120, 0x10xb3bb, 0x20xa232,
0x30x5ac5, 0x40x4b4c, 0x50x7d97, 0x60x685e, 0x70x1ce1, 0x80x0d68, 0x90x3ff3, 0xa0x2e7a,
0xb0xe70e, 0xc0x6f87, 0xd0xc41c, 0xe0xd595, 0xf0xa12a, 0x00xb0a3, 0x10x8238, 0x20x93b1,
0x30x6b46, 0x40x7acf, 0x50x4854, 0x60x59dd, 0x70x2d62, 0x80x3ce4, 0x90x0e70, 0xa0x1ff9,
0xb0xf78f, 0xc0xe606, 0xd0x49d, 0xe0xc514, 0xadxb1, 0xa0xa22, 0xb0x92b9, 0xc0x8330,
0xd0x7bc7, 0xe0x6a4e, 0xf0x58d5, 0x00x495c, 0x10x3de3, 0x20xc26a, 0x30x1ef1, 0x40x7f8 }

word calcTable(byte *buffer, int len) {
}
word crc = 0;

while (len-- != 0)
{
    crc = (crc >>= 8) ^ crc_table[(crc ^ *buffer) & 0xFF];

    buffer++;
}

return crc;
}

void initCrc(word &crc)
{
    crc = 0xFFFF;
}

void updateCrc(word &crc, byte data)
{
    unsigned short i, v, xor_flag;

    v = 0x80;

    for (i=0; i<8; i++)
    {
        if (crc & 0x8000)
        {
            xor_flag= 1;
        }
        else
        {
            xor_flag= 0;
        }
        crc = crc << 1;

        if (data & v)
        {
        
            if (crc & 0x8000)
            {
                xor_flag= 1;
            }
            else
            {
                xor_flag= 0;
            }

            crc = crc << 1;

            if (data & v)
crc = crc + 1;
}

if (xor_flag)
{
    crc = crc ^ poly;
}

v = v >> 1;
}
}

void finishCrc(word &crc)
{
    unsigned short i, xor_flag;

    for (i=0; i<16; i++)
    {
        if (crc & 0x8000)
        {
            xor_flag= 1;
        }
        else
        {
            xor_flag= 0;
        }
        crc = crc << 1;
        if (xor_flag)
        {
            crc = crc ^ poly;
        }
    }
}

word calcCrc(byte *buffer, int len)
{
//return calcTable(buffer, len);

    word crc;
initCrc(crc);
while (len-- > 0)
{
    updateCrc(crc, *buffer);
    buffer++;
}
finishCrc(crc);
return crc;
}

UART:

/*
 * constants and macros
 */

/* size of RX/TX buffers */
define UART_RX_BUFFER_MASK ( UART_RX_BUFFER_SIZE - 1)
define UART_TX_BUFFER_MASK ( UART_TX_BUFFER_SIZE - 1)

#if ( UART_RX_BUFFER_SIZE & UART_RX_BUFFER_MASK )
#error RX buffer size is not a power of 2
#endif
#endif
#error TX buffer size is not a power of 2
#endif

#if defined(__AVR_AT90S2313__) \\
|| defined(__AVR_AT90S4414__) || defined(__AVR_AT90S4434__) \\
|| defined(__AVR_AT90S8515__) || defined(__AVR_AT90S8535__) \\
|| defined(__AVR_ATmega103__) 
/* old AVR classic or ATmega103 with one UART */
#define AT90_UART
#define UART0_RECEIVE_INTERRUPT   SIG_UART_RECV
#define UART0_TRANSMIT_INTERRUPT  SIG_UART_DATA
#define UART0_STATUS   USR
#define UART0_CONTROL  UCR
#define UART0_DATA     UDR
#define UART0_UDRIE    UDRIE
#endif

#elif defined(__AVR_AT90S2333__) || defined(__AVR_AT90S4433__) 
/* old AVR classic with one UART */
#define AT90_UART
#define UART0_RECEIVE_INTERRUPT   SIG_UART_RECV
#define UART0_TRANSMIT_INTERRUPT  SIG_UART_DATA
#define UART0_STATUS   UCSRA
#define UART0_CONTROL  UCSRB
#define UART0_DATA     UDR
#define UART0_UDRIE    UDRIE
#endif

#elif defined(__AVR_ATmega8__) || defined(__AVR_ATmega16__) || defined(__AVR_ATmega32__) \\
|| defined(__AVR_ATmega8515__) || defined(__AVR_ATmega8535__) \\
|| defined(__AVR_ATmega323__) 
/* ATmega with one USART */
#define ATMega_USART
#define UART0_RECEIVE_INTERRUPT   SIG_UART_RECV
#define UART0_TRANSMIT_INTERRUPT  SIG_UART_DATA
#define UART0_STATUS   UCSRA
#define UART0_CONTROL  UCSRB
#define UART0_DATA     UDR
#define UART0_UDRIE    UDRIE
#endif

#elif defined(__AVR_ATmega163__) 
/* ATmega163 with one UART */
#define ATMega163_USART
#define UART0_RECEIVE_INTERRUPT   SIG_UART_RECV
#define UART0_TRANSMIT_INTERRUPT  SIG_UART_DATA
#define UART0_STATUS   UCSRA
#define UART0_CONTROL  UCSRB
#define UART0_DATA     UDR
#define UART0_UDRIE    UDRIE
#endif
#define ATMEGA_UART
#define UART0_RECEIVE_INTERRUPT SIG_UART_RECV
#define UART0_TRANSMIT_INTERRUPT SIG_UART_DATA
#define UART0_STATUS UCSRA
#define UART0_CONTROL UCSRB
#define UART0_DATA UDR
#define UART0_UDRIE UDRIE

#else defined(__AVR_ATmega162__) 

/* ATmega with two USART */
#define ATMEGA_USART0
#define ATMEGA_USART1
#define UART0_RECEIVE_INTERRUPT SIG_USART0_RECV
#define UART1_RECEIVE_INTERRUPT SIG_USART1_RECV
#define UART0_TRANSMIT_INTERRUPT SIG_USART0_DATA
#define UART1_TRANSMIT_INTERRUPT SIG_USART1_DATA
#define UART0_STATUS UCSR0A
#define UART0_CONTROL UCSR0B
#define UART0_DATA UDR0
#define UART0_UDRIE UDRIE0
#define UART1_STATUS UCSR1A
#define UART1_CONTROL UCSR1B
#define UART1_DATA UDR1
#define UART1_UDRIE UDRIE1

#else defined(__AVR_ATmega64__) || defined(__AVR_ATmega128__) 

/* ATmega with two USART */
//#define ATMEGA_USART0

#define ATMEGA_USART1
#define UART0_RECEIVE_INTERRUPT SIG_UART0_RECV
#define UART1_RECEIVE_INTERRUPT SIG_UART1_RECV
#define UART0_TRANSMIT_INTERRUPT SIG_UART0_DATA
#define UART1_TRANSMIT_INTERRUPT SIG_UART1_DATA
#define UART0_STATUS UCSR0A
#define UART0_CONTROL UCSR0B
#define UART0_DATA UDR0
#define UART0_UDRIE UDRIE0
#define UART1_STATUS UCSR1A
#define UART1_CONTROL UCSR1B
#define UART1_DATA UDR1
#define UART1_UDRIE UDRIE1

#if defined(__AVR_ATmega161__)
/* ATmega with UART */
#error "AVR ATmega161 currently not supported by this library!"
#elif defined(__AVR_ATmega169__)
/* ATmega with one USART */
#define ATMEGA_USART
#define UART0_RECEIVE_INTERRUPT SIG_USART_RECV
#define UART0_TRANSMIT_INTERRUPT SIG_USART_DATA
#define UART0_STATUS UCSRA
#define UART0_CONTROL UCSRB
#define UART0_DATA UDR
#define UART0_UDRIE UDRIE
#elif defined(__AVR_ATmega48__) || defined(__AVR_ATmega88__) || defined(__AVR_ATmega168__)
/* ATmega with one USART */
#define ATMEGA_USART0
#define UART0_RECEIVE_INTERRUPT SIG_USART_RECV
#define UART0_TRANSMIT_INTERRUPT SIG_USART_DATA
#define UART0_STATUS UCSR0A
#define UART0_CONTROL UCSR0B
#define UART0_DATA UDR0
#define UART0_UDRIE UDRIE0
#elif defined(__AVR_ATtiny2313__)
#define ATMEGA_USART
#define UART0_RECEIVE_INTERRUPT SIG_USART0_RX
#define UART0_TRANSMIT_INTERRUPT SIG_USART0_UDRE
#define UART0_STATUS UCSRA
#define UART0_CONTROL UCSRB
#define UART0_DATA UDR
#define UART0_UDRIE UDRIE
#elif defined(__AVR_ATmega329__) || defined(__AVR_ATmega3290__) ||
    defined(__AVR_ATmega649__) || defined(__AVR_ATmega6490__) ||
    defined(__AVR_ATmega325__) || defined(__AVR_ATmega3250__) ||
    defined(__AVR_ATmega645__) || defined(__AVR_ATmega6450__)
/* ATmega with one USART */
#define ATMEGA_USART0

#define ATMEGA_USART
#define UART0_RECEIVE_INTERRUPT SIG_USART0_RX
#define UART0_TRANSMIT_INTERRUPT SIG_USART0_UDRE
#define UART0_STATUS UCSRA
#define UART0_CONTROL UCSRB
#define UART0_DATA UDR
#define UART0_UDRIE UDRIE
```c
#define UART0_RECEIVE_INTERRUPT SIG_UART_RECV
#define UART0_TRANSMIT_INTERRUPT SIG_UART_DATA
#define UART0_STATUS UCSR0A
#define UART0_CONTROL UCSR0B
#define UART0_DATA UDR0
#define UART0_UDRIE UDRIE0
#elif defined(__AVR_ATmega2560__) || defined(__AVR_ATmega1280__) || defined(__AVR_ATmega640__)
/* ATmega with two USART */
#define ATMEGA_USART0
#define ATMEGA_USART1
#define UART0_RECEIVE_INTERRUPT SIG_USART0_RECV
#define UART1_RECEIVE_INTERRUPT SIG_USART1_RECV
#define UART0_TRANSMIT_INTERRUPT SIG_USART0_DATA
#define UART1_TRANSMIT_INTERRUPT SIG_USART1_DATA
#define UART0_STATUS UCSR0A
#define UART0_CONTROL UCSR0B
#define UART0_DATA UDR0
#define UART0_UDRIE UDRIE0
#define UART1_STATUS UCSR1A
#define UART1_CONTROL UCSR1B
#define UART1_DATA UDR1
#define UART1_UDRIE UDRIE1
#elif defined(__AVR_ATmega644__)  /* ATmega with one USART */
#define ATMEGA_USART0
#define UART0_RECEIVE_INTERRUPT SIG_USART_RECV
#define UART0_TRANSMIT_INTERRUPT SIG_USART_DATA
#define UART0_STATUS UCSR0A
#define UART0_CONTROL UCSR0B
#define UART0_DATA UDR0
#define UART0_UDRIE UDRIE0
#elif defined(__AVR_ATmega164P__) || defined(__AVR_ATmega324P__) || defined(__AVR_ATmega644P__)  /* ATmega with two USART */
#define ATMEGA_USART0
#define ATMEGA_USART1
#define UART0_RECEIVE_INTERRUPT SIG_USART_RECV
#define UART1_RECEIVE_INTERRUPT SIG_USART1_RECV
#define UART0_TRANSMIT_INTERRUPT SIG_USART0_DATA
#define UART1_TRANSMIT_INTERRUPT SIG_USART1_DATA
#define UART0_STATUS UCSR0A
#define UART0_CONTROL UCSR0B
#define UART0_DATA UDR0
#define UART0_UDRIE UDRIE0
#define UART1_STATUS UCSR1A
#define UART1_CONTROL UCSR1B
#define UART1_DATA UDR1
#define UART1_UDRIE UDRIE1
#elif defined(__AVR_ATmega164P__) || defined(__AVR_ATmega324P__) || defined(__AVR_ATmega644P__)  /* ATmega with two USART */
#define ATMEGA_USART0
#define ATMEGA_USART1
```
#define UART0_RECEIVE_INTERRUPT   SIG_USART_RECV
#define UART1_RECEIVE_INTERRUPT   SIG_USART1_RECV
#define UART0_TRANSMIT_INTERRUPT  SIG_USART_DATA
#define UART1_TRANSMIT_INTERRUPT  SIG_USART1_DATA
#define UART0_STATUS   UCSR0A
#define UART0_CONTROL  UCSR0B
#define UART0_DATA     UDR0
#define UART0_UDRIE    UDRIE0
#define UART1_STATUS   UCSR1A
#define UART1_CONTROL  UCSR1B
#define UART1_DATA     UDR1
#define UART1_UDRIE    UDRIE1
#else
#error "no UART definition for MCU available"
#endif

/*
 *  module global variables
 */
#if defined( ATMEGA_USART0 )
static volatile unsigned char UART_TxBuf[UART_TX_BUFFER_SIZE];
static volatile unsigned char UART_RxBuf[UART_RX_BUFFER_SIZE];
static volatile unsigned char UART_TxHead;
static volatile unsigned char UART_TxTail;
static volatile unsigned char UART_RxHead;
static volatile unsigned char UART_RxTail;
static volatile unsigned char UART_LastRxError;
#endif

#if defined( ATMEGA_USART1 )
static volatile unsigned char UART1_TxBuf[UART_TX_BUFFER_SIZE];
static volatile unsigned char UART1_RxBuf[UART_RX_BUFFER_SIZE];
static volatile unsigned char UART1_TxHead;
static volatile unsigned char UART1_TxTail;
static volatile unsigned char UART1_RxHead;
static volatile unsigned char UART1_RxTail;
*/
static volatile unsigned char UART1_LastRxError;
#endif

#if defined( ATMEGA_USART0 )
SIGNAL(UART0_RECEIVE_INTERRUPT)

@GetMappingArea

/***********************************************************/
Function: UART Receive Complete interrupts
Purpose: called when the UART has received a character
/**************************************************************/
{
    unsigned char tmphead;
    unsigned char data;
    unsigned char usr;
    unsigned char lastRxError;

    /* read UART status register and UART data register */
    usr  = UART0_STATUS;
    data = UART0_DATA;

    /* */
#if defined( AT90_UART )
    lastRxError = (usr & (_BV(FE)|_BV(DOR)) );
#elif defined( ATMEGA_USART )
    lastRxError = (usr & (_BV(FE)|_BV(DOR)) );
#elif defined( ATMEGA_USART0 )
    lastRxError = (usr & (_BV(FE0)|_BV(DOR0)) );
#elif defined( ATMEGA_UART )
    lastRxError = (usr & (_BV(FE)|_BV(DOR)) );
#endif

    /* calculate buffer index */
    tmphead = ( UART_RxHead + 1) & UART_RX_BUFFER_MASK;
if ( tmphead == UART_RxTail ) {
    /* error: receive buffer overflow */
    lastRxError = UART_BUFFER_OVERFLOW >> 8;
} else{
    /* store new index */
    UART_RxHead = tmphead;
    /* store received data in buffer */
    UART_RxBuf[tmphead] = data;
}
UART_LastRxError = lastRxError;

SIGNAL(UART0_TRANSMIT_INTERRUPT)
/*************************************************************************/
Function: UART Data Register Empty interrupts
Purpose: called when the UART is ready to transmit the next byte
/*************************************************************************/
{
    unsigned char tmptail;
    if ( UART_TxHead != UART_TxTail) {
        /* calculate and store new buffer index */
        tmptail = (UART_TxTail + 1) & UART_TX_BUFFER_MASK;
        UART_TxTail = tmptail;
        /* get one byte from buffer and write it to UART */
        UART0_DATA = UART_TxBuf[tmptail]; /* start transmission */
    } else{
        /* tx buffer empty, disable UDRE interrupt */
        UART0_CONTROL &= ~_BV(UART0_UDRIE);
    }
}
Function: uart_init()
Purpose: initialize UART and set baudrate
Input: baudrate using macro UART_BAUD_SELECT()
Returns: none
***************************************************************************/

void uart_init(unsigned int baudrate)
{
    UART_TxHead = 0;
    UART_TxTail = 0;
    UART_RxHead = 0;
    UART_RxTail = 0;

#if defined( AT90_UART )
    /* set baud rate */
    UBRR = (unsigned char)baudrate;

    /* enable UART receiver and transmitter and receive complete interrupt */
    UART0_CONTROL = _BV(RXCIE)|_BV(RXEN)|_BV(TXEN);
#endif

#if defined( ATMEGA_USART )
    /* Set baud rate */
    if ( baudrate & 0x8000 )
    {
        UART0_STATUS = (1<<U2X); //Enable 2x speed
        baudrate &= ~0x8000;
    }

    UBRRH = (unsigned char)(baudrate>>8);
    UBRRL = (unsigned char) baudrate;

    /* Enable USART receiver and transmitter and receive complete interrupt */
    UART0_CONTROL = _BV(RXCIE)|(_<<RXEN)|(1<<TXEN);
#endif

    /* Set frame format: asynchronous, 8data, no parity, 1stop bit */
#ifdef URSEL
    UCSRC = (1<<URSEL)|(3<<UCSZ0);
#else
    UCSRC = (3<<UCSZ0);
#endif
#endif

#elif defined (ATMEGA_USART0)

/* Set baud rate */
if (baudrate & 0x8000)
{
    UART0_STATUS = (1<<U2X0); //Enable 2x speed
    baudrate &= ~0x8000;
}
UBRR0H = (unsigned char)(baudrate>>8);
UBRR0L = (unsigned char) baudrate;

/* Enable USART receiver and transmitter and receive complete interrupt */
UART0_CONTROL = _BV(RXCIE0)|(_BV(RXEN0)|(_BV(TXEN0);

/* Set frame format: asynchronous, 8data, no parity, 1stop bit */
#ifdef URSEL0
    UCSR0C = (1<<URSEL0)|(_BV(UCSZ00);
#else
    UCSR0C = (_BV(UCSZ00);
#endif

#elif defined (ATMEGA_UART)

/* set baud rate */
if (baudrate & 0x8000)
{
    UART0_STATUS = (1<<U2X); //Enable 2x speed
    baudrate &= ~0x8000;
}
UBRRHI = (unsigned char)(baudrate>>8);
UBRR  = (unsigned char) baudrate;

/* Enable UART receiver and transmitter and receive complete interrupt */
UART0_CONTROL = _BV(RXCIE0)|(_BV(RXEN0)|(_BV(TXEN);

#endif
Function: uart_getc()
Purpose: return byte from ringbuffer
Returns: lower byte: received byte from ringbuffer
         higher byte: last receive error
unsigned int uart_getc(void)
{
    unsigned char tmptail;
    unsigned char data;

    if ( UART_RxHead == UART_RxTail ) {
        return UART_NO_DATA; /* no data available */
    }

    /* calculate /store buffer index */
    tmptail = (UART_RxTail + 1) & UART_RX_BUFFER_MASK;
    UART_RxTail = tmptail;

    /* get data from receive buffer */
    data = UART_RxBuf[tmptail];

    return (UART_LastRxError << 8) + data;
}

Function: uart_putchar()
Purpose: write byte to ringbuffer for transmitting via UART
Input:   byte to be transmitted
Returns: none
void uart_putchar(unsigned char data, bool start)
{
    unsigned char tmphead;
tmphead = (UART_TxHead + 1) & UART_TX_BUFFER_MASK;

while ( tmphead == UART_TxTail ){
    /* wait for free space in buffer */
}

UART_TxBuf[tmphead] = data;
UART_TxHead = tmphead;

/* enable UDRE interrupt */
if (start)
    UART0_CONTROL |= _BV(UART0_UDRIE);

}/* uart_putchar */

/**************************************************************************
Function: uart_puts()
Purpose:  transmit string to UART
Input:    string to be transmitted
Returns:  none
**************************************************************************/

void uart_puts(const char *s )
{

    while (*s)
    {
        uart_putchar(*s++);
    }

}/* uart_puts */

/**************************************************************************
Function: uart_puts_p()
Purpose:  transmit string from program memory to UART
Input:    program memory string to be transmitted
Returns:  none
**************************************************************************/

void uart_puts_p(const char *progmem_s )
{
    register char c;

while ( (c = pgm_read_byte(progmem_s++)) )
    uart_putchar(c);

}/* uart_puts_p */
#endif

#ifdef( ATMEGA_USART1 )

SIGNAL(UART1_RECEIVE_INTERRUPT)
/* read UART status register and UART data register */
usr = UART1_STATUS;
data = UART1_DATA;

/* calculate buffer index */
tmphead = ( UART1_RxHead + 1) & UART_RX_BUFFER_MASK;

if ( tmphead == UART1_RxTail ) {
    /* error: receive buffer overflow */
    lastRxError = UART_BUFFER_OVERFLOW >> 8;
} else {
/* store new index */
UART1_RxHead = tmphead;
/* store received data in buffer */
UART1_RxBuf[tmphead] = data;
}
UART1_LastRxError = lastRxError;

SIGNAL(UART1_TRANSMIT_INTERRUPT)
/*---------------------------------------------*/
Function: UART1 Data Register Empty interrupt
Purpose: called when the UART1 is ready to transmit the next byte
---------------------------------------------*/
{
    unsigned char tmptail;

    if ( UART1_TxHead != UART1_TxTail) {
        /* calculate and store new buffer index */
        tmptail = (UART1_TxTail + 1) & UART_TX_BUFFER_MASK;
        UART1_TxTail = tmptail;
        /* get one byte from buffer and write it to UART */
        UART1_DATA = UART1_TxBuf[tmptail]; /* start transmission */
    }else{
        /* tx buffer empty, disable UDRE interrupt */
        UART1_CONTROL &= ~_BV(UART1_UDRIE);
    }
}

/*---------------------------------------------*/
Function: uart1_init()
Purpose: initialize UART1 and set baudrate
Input: baudrate using macro UART_BAUD_SELECT()
Returns: none
---------------------------------------------*/
```c
void uart1_init(unsigned int baudrate)
{
    UART1_TxHead = 0;
    UART1_TxTail = 0;
    UART1_RxHead = 0;
    UART1_RxTail = 0;

    /* Set baud rate */
    if ( baudrate & 0x8000 )
    {
        UART1_STATUS = (1<<U2X1);  //Enable 2x speed
        baudrate &= ~0x8000;
    }
    UBRR1H = (unsigned char)(baudrate>>8);
    UBRR1L = (unsigned char) baudrate;

    /* Enable USART receiver and transmitter and receive complete interrupt */
    UART1_CONTROL = _BV(RXCIE1)|(1<<RXEN1)|(1<<TXEN1);

    /* Set frame format: asynchronous, 8data, no parity, 1stop bit */
    #ifdef URSEL1
    UCSR1C = (1<<URSEL1)|(3<<UCSZ10);
    #else
    UCSR1C = (3<<UCSZ10);
    #endif

    /* uart_init */
}
```

---

### Function: uart1_getc()
**Purpose:** return byte from ringbuffer
**Returns:** lower byte: received byte from ringbuffer
higher byte: last receive error

---

```c
unsigned int uart1_getc(void)
{
```
unsigned char tmptail;
unsigned char data;

if ( UART1_RxHead == UART1_RxTail ) {
    return UART_NO_DATA; /* no data available */
}

/* calculate /store buffer index */
tmptail = (UART1_RxTail + 1) & UART_RX_BUFFER_MASK;
UART1_RxTail = tmptail;

/* get data from receive buffer */
data = UART1_RxBuf[tmptail];

return (UART1_LastRxError << 8) + data;

}/* uart1_getc */

/**************************************************************************
 Function: uart1_putc()
 Purpose: write byte to ringbuffer for transmitting via UART
 Input:    byte to be transmitted
 Returns:  none
**************************************************************************/

void uart1_putc(unsigned char data)
{
    unsigned char tmphead;

tmphead = (UART1_TxHead + 1) & UART_TX_BUFFER_MASK;

while ( tmphead == UART1_TxTail ){
    ;/* wait for free space in buffer */
}
UART1_TxBuf[tmphead] = data;
UART1_TxHead = tmphead;

/* enable UDRE interrupt */
UART1_CONTROL |= _BV(UART1_UDRIE);

}/* uart1_putchar */

.Tx
Function: uart1_puts()
Purpose: transmit string to UART1
Input: string to be transmitted
Returns: none
**************************************************************************/void uart1_puts(const char *s )
{
    while (*s)
        uart1_putchar(*s++);
}

}/* uart1_puts */

Function: uart1_puts_p()
Purpose: transmit string from program memory to UART1
Input: program memory string to be transmitted
Returns: none
**************************************************************************/

void uart1_puts_p(const char *progmem_s )
{
    register char c;

    while ( (c = pgm_read_byte(progmem_s++)) )
        uart1_putchar(c);

}/* uart1_puts_p */

#endif
4. Future Work

With a single Bioloid kit, we can reconfigure it to be a hexapod, quadrapod, wheeled boot, various bipeds, a 6+ axis arm, etc and our plan for future is to program the software for each of all. It also comes with a range/sound sensor unit, and a GUI program interface for autonomous programs and 'pose and capture' sequencing software. Also we want to buy another Bioloid in the future to make the software in basketball and football with Bioloid, to be the participants in the European and World Cup.

Another goal is improving the camera of Bioloid. In the future we have to change it and connect with bioloid, like mobile phone or something similar, because we want to do the exercises which are based with new camera and to do the new application like: listening, seeing, detecting the object which around him, listen the command and execute them, programming to answer the easy questions, etc. In other words gives the Bioloid a new brain, Wi-Fi communication, emotional expressions, voice etc.

All the source code and diagrams are available. Behavior program reads commands sent by the robot head and runs respective motion tables. It also passes information about robot's status and robot-head so that robot head can take robot status into account in its program.
5. Bibliography


http://www.electronickits.com/robot/Bioloid

http://www.super-science-fair-projects.com