Technion
Israel Institute of Technology

CRML
Control Robotics and Machine Learning Laboratory

Bioloid Robot Project

Final Report

Authors:
Michael Gouzenfeld
Alexey Serafimov

Supervisor:
Ido Cohen

Semester:
Winter 2012-2013

Department of Electrical Engineering

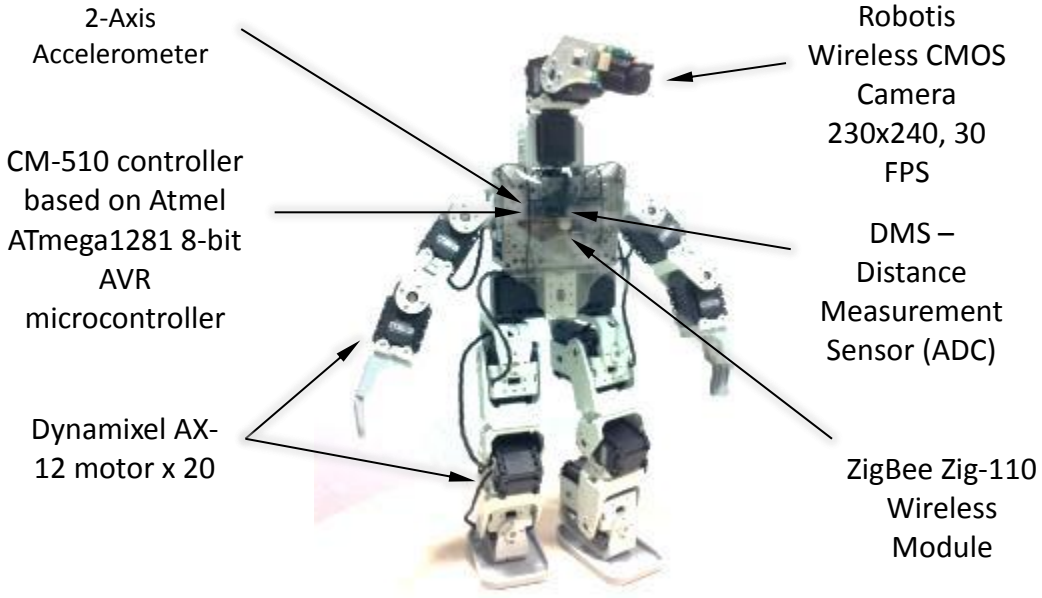
Contents

Introduction

The Bioloid robot is manufactured by ROBOTIS Company. The robot is marketed as a construction kit. There are several types of kits, such as: Beginner Kit, Comprehensive Kit, Expert Kit, Premium Kit etc. Each kit contains arrays of construction parts, sensors, peripheral devices, cables, screws, tools, SW etc. The difference between the kits is a variety of possible robots to be constructed from the parts and their features. For example, Beginner Kit allows building very simple robots, which consist of 1-4 Dynamixels (joint servo motors), such as a little car, a moving hand, a primitive snake, etc. Intermediate Kit's robots are little more advanced and may consist of 4-8 Dynamixels. The Expert Kit allows building relatively complex robots: Dog, Dinosaur, Spider and Humanoid. Those robots may consist of up to 20 Dynamixels. The project was initially launched on the Expert's Kit Humanoid robot, but later it was upgraded to the Premium Kit robot, because it has more features.

Additional information about the Bioloid robot can be found here:
http://www.robotis.com/

The Bioloid Robot structure



As mentioned earlier in the Introduction, the robot is built from the Premium Kit's parts, but with some add-ons and "add-offs". The main parts are:

1. CM-510 controller based on Atmel ATmega1281 8-bit AVR microcontroller. This is the main controller of the robot. All the dynamixels, sensors, indicators etc. are connected to it. Also, the CM-510 box contains some indicators (LEDs and buzzer) and buttons.
2. Dynamixel AX-12. This is a joint servo motor, which is connected to the dynamixel's bus. The Premium Kit robot consists of 18 joint body dynamixels. In this project 2 additional dynamixels were added to construct the robot's Head (total of 20 dynamixels).
3. Robot's Head consists of 2 dynamixels, which give 2 degrees of freedom: tilt and pan. The camera attached to the head.
4. Wireless CMOS Camera 230x240, 30 FPS. This camera is taken from the robot's Expert Kit. The camera is attached to the robot's head, but it's not connected to the robot's controller. The power to the camera is provided by the dynamixel bus. The image is transmitted to the camera's hub, which is connected to PC via USB.
5. ZigBee Zig-110 Wireless Module. This module is in charge of wireless communication to the remote control. Remote control can be performed by RC-100 (kind of joystick) or by another Zigbee device.
6. DMS – Distance Measurement Sensor (IR). This is an infra-red sensor indicating distance. In this project there is only a single IR sensor at the middle of the body, but additional sensors can be attached to the robot's feet.

7. 2-Axis Accelerometer. This device was marketed by the manufacturer as “Gyro” in the Premium Kit only, that’s why the kit was pretty attractive to purchase. But actually, the device is only an accelerometer. It means that it does not measure the absolute position of itself in the space, but only acceleration towards an axis. This limits the control of robot’s falling prevention, because if it starts fall slowly, the acceleration approaches 0 and the acceleration sensor is not sensitive enough.

The project’s goals

The project’s general goal is to make the Bioloid robot detect an object in the room, approach it and perform some actions on it.

To be more specific, the robot should detect a ball, approach it and kick it.

So, the project can be divided to the next sub-goals:

1. Robot’s motion controller - making the robot move autonomously.
2. Image processing - making a robot finding a ball in the room.
3. Wireless communication - sending the robot commands and getting statuses from it.
4. Main controller - in charge of synchronizing all the other parts.

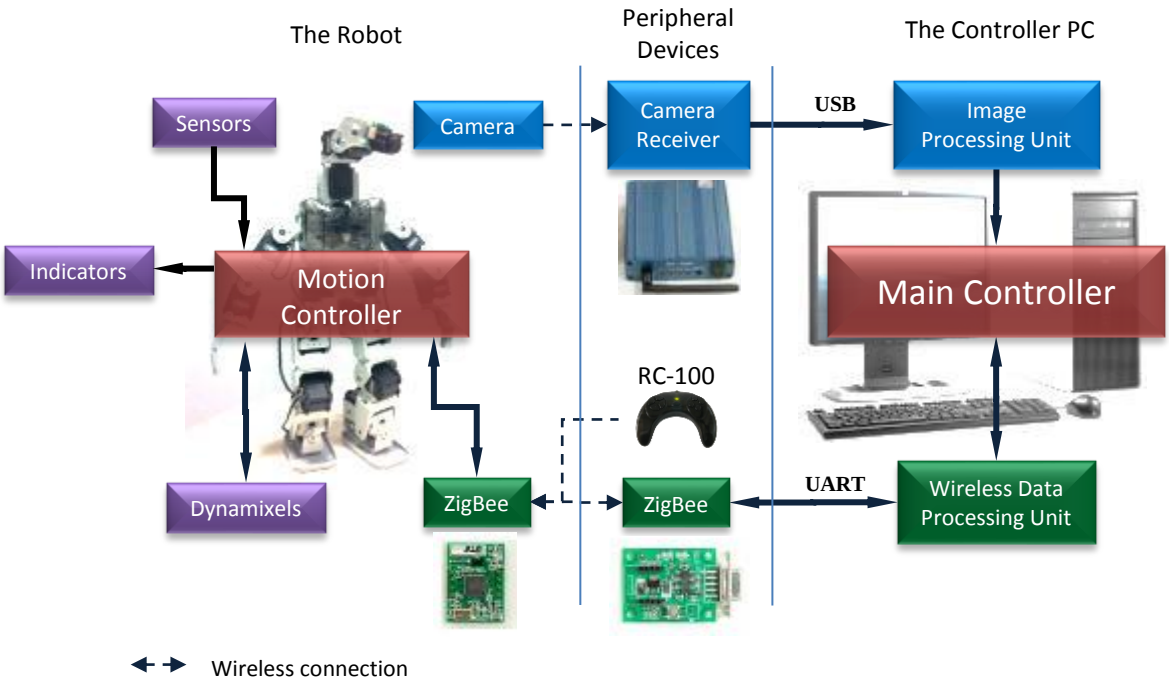
Requirements compliance

Here is the tracking table of the project’s requirements:

No.	Requirement	Status	Notes
Functional requirements			
1	Robot’s motion controller implementation in C	✓	
2	Image processing implementation in C++	✓	
3	Wireless communication protocol implementation	✓	
4	Main controller implementation	×	Not implemented yet
Non-functional requirements			
1	User documentation	✓	
Extra features implementation			
1	Head tracking after the found object	+	
2	Obstacle avoiding using DMS sensor	+	
3	CM-510 sounds library API	+	
4	CM-510 LEDs functionality API	+	

Functionality description

The whole picture



The Motion Controller runs independently on the robot's controller. The Motion Controller is in charge of the whole robot's movements, including the body and the head. It controls dynamixels and indicators, and receives a data from the sensors. Also, it can receive and send commands via the ZigBee wireless device. The robot's action depends on the command received. The command can be received from the Wireless Data Processing Unit located on PC or from the RC-100 (remote controller) simultaneously. The data sent from the robot to the Wireless Data Unit is mostly the tilt and the pan positions of the robot's head.

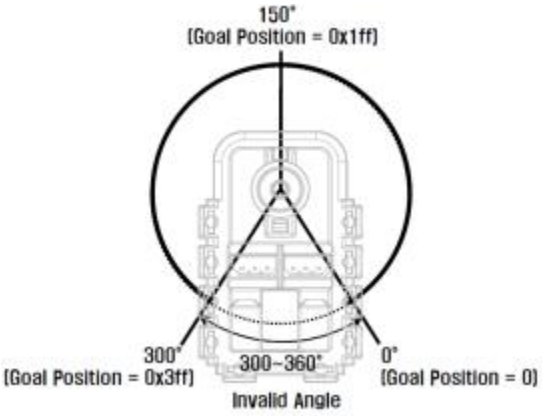
The camera located on the robot's head is not connected to the Motion Controller. It just sends image information to the Camera Receiver. The Camera Receiver is connected to the PC via USB and the video is processed by the Image Processing Unit.

Relying on the information provided by the Image Processing Unit and the Wireless Data Processing Unit the Main Controller decides which command is about to be sent to the robot to reach the goal.

Motion Controller

Terminology

Dynamixel: servo motor, which has a lot of configurable parameters (see appendix A).



Dynamixel's position: dynamixel's absolute angle in integer units in range of 0-1023.

The actual valid angle of dynamixel motor is 300˚. So the conversion formula of the angle in degrees into the integer units is:

$$Position = \frac{angle°}{300°} \cdot 1023$$

Dynamixel's speed: dynamixel's angular velocity of movement in integer units in range of 0-1023.

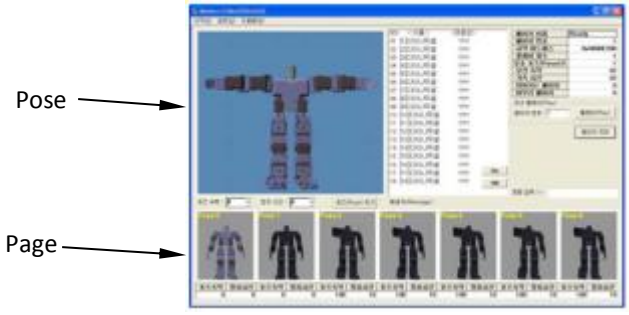
Regarding the AX-12 specification, the integer value of 1023 corresponds to 114 RPM. So the conversion formula of the angular velocity in degrees into the integer units is:

$$Speed = \frac{angular\ velocity\ [RPM]}{114\ [RPM]} \cdot 1023$$

Important note! The value of 0 is not the minimum speed, but is the maximum speed of the dynamixel it can produce, means - uncontrollable speed. The actual value of minimum speed is 1.

Pose: a vector of all dynamixel's positions, not including the head (18 dynamixels).

Page: a sequence of robot's poses to be performed.



Offsets vector: dynamixel's position offsets from a pose. Used to adjust the robot's stability.

Memory Management

The manufacturer provides only limited information about the controller's resources. The Expert Kit Manual states that in CM-5 unit there are 3 kinds of memory:

1. RAM: 4 KB. This is the operational memory, which is cleared on robot's power-off.
2. EEPROM: 4 KB. Here the boot manager is stored. Boot manager's goal is to run the program stored in the flash memory.
3. FLASH: 128 KB. Here all the data and the code are stored.

The CM-5 controller was used in the Expert Kit - the initial kit the project was launched on. As mentioned in the previous sections, the robot was upgraded to the Premium Kit, which works with the CM-510 controller.

The size of those memory spaces are not specified by the manufacturer in any Premium Kit manual, but some experiments show that the only change is in the FLASH memory, which grew up to 256 KB. Updating the controller with manufacturer's SW may affect the EEPROM and the FLASH regions. Updating the controller with user's SW affects only the FLASH region.

The manufacturer provides some SW tools for the robot's management. In addition, the manufacturer created some programs to be performed by the robot. Those programs are divided to two parts:

1. Motion - a set of poses/pages to be performed by the robot.
2. Task - the code (logic) that performs the motion pages above.

The manufacturer divided the whole Motion into pages of 7 poses in each page. For each pose in the page there are parameters of playing time and delay after the pose performing. Each page has parameters, such as: next page to play, exit page, page playing time, etc.

Some of SW tools provided are:

1. RoboPlus Motion. This tool allows editing the Motion part of robot's program.
2. RoboPlus Task. This tool allows editing the Task part of robot's program.
3. RoboPlus Terminal. This tool was intended to access the low level parts of the controller, such as memory, dynamixels parameters, etc. It looks like with the newer versions, the functionality becomes more limited. So with the recent versions the main usage of this tool is FLASH updating with compiled code.
4. RoboPlus Manager. Allows connecting to the robot's dynamixels and other devices to its status. Also, it allows updating the robot's controller with manufacturer's SW.

To avoid spending efforts on teaching the robot new poses, it was decided to use the Motion provided by the manufacturer within its Soccer Program. The Task from the Soccer Program, in contrast, is replaced with SW system written in C language. So, the robot's SW updating process is as following:

1. Updating the robot with manufacturer's Soccer Program.
2. Updating the robot with the user's code.

As a result of this process, the Motion section stays in robot's FLASH memory, but it accessed by the code provided by the user.

The CM-510 controller is based on Atmel ATmega1281 8-bit AVR microcontroller. This controller is built using Harvard architecture, which means that the program and the data spaces are separated: FLASH is used for the program and RAM is used for the data. It is a challenge to get constant data to be stored in the Program Space, and to retrieve that data to use it in the AVR application.The problem is exacerbated by the fact that the C Language was not designed for Harvard architectures, it was designed for Von Neumann architectures where code and data exist in the same address space. This means that any compiler for a Harvard architecture processor, like the AVR, has to use other means to operate with separate address spaces.

As mentioned before, all the SW is updated into the FLASH, means data space. On robot's boot, the code is loaded into the RAM, so in this state the program is in the RAM, but the DATA is in the FLUSH, i.e. the code and the data are on the separate memory spaces. To access the FLUSH, special AVR functions are used. In the current implementation the pages are copied from the FLUSH to the RAM and then performed.

Since the RAM is much smaller (4 KB) than FLUSH (256 KB), there is no space for all the pages in the RAM. This is resolved by implementing a caching mechanism using the Random Replacing Policy.

Pose playing

The main challenge in the pose playing mechanism is to make all relevant dynamixels start and stop moving simultaneously. This goal is reached by the following means:

1. Calculating the correct speed for each dynamixel to move with, according to the goal position.
2. Sending a broadcast message on the dynamixel's bus, so all the dynamixels will get the command at the same time.

To calculate the correct speed of each dynamixel, the following algorithm is used:

1. Assume the number of dynamixels is 18, so the arrays' size below is 18.
2. Assume the initial pose is: $initial\_pose[]$.
3. Assume the target pose is: $target\_pose[]$.
4. Assume the given speed for pose playing is: $speed$.
5. Need to calculate: $speed[i]$ for each $0 \leq i \leq 17$.
6. Set maximal position difference as:

$$\max\_pos\_diff = \max_{0 \leq i \leq 17} \{|target\_pose[i] - initial\_pose[i]|\}$$

7. The goal is to play the maximum moving dynamixel with the given speed and all the rest of dynamixels with relative speed. In this way we get:

$$speed[i] = \frac{|target\_pose[i] - initial\_pose[i]|}{\max\_pos\_diff} \cdot speed$$

Page playing

As mentioned before, each page has several parameters, such as:

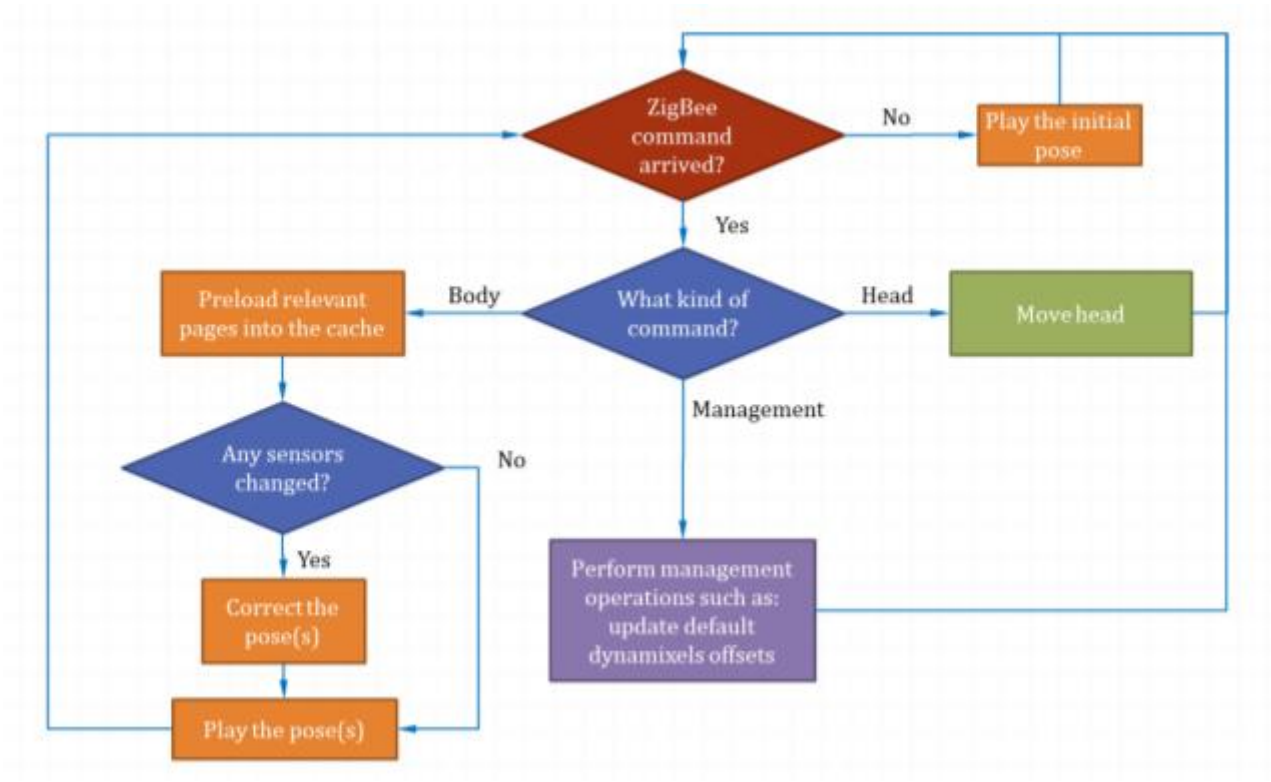
1. Play time/speed.
2. Play count.
3. Next page.
4. Exit page.
5. Each pose on the page has:
   a. Play time/speed.
   b. Delay.

The poses of the page are being played sequentially.

The speed for each pose is eventually defined by the pose's play time/speed and the page's play time/speed. I.e. the page's play speed is some kind of a factor to multiply each pose's speed on the page. After each pose there is a delay that might be applied, but in most pages the delay is 0. When reaching the end of the page, there is a jump to another page pointed by the Next Page parameter. If some event occurs during the page playing, the need of finishing the current page playing immediately may arise. In this case a jump to the Exit Page is performed.

Motion cycle

The general flow of the motion cycle can be described as following:



The controller is constantly waiting for the ZigBee command to arrive. The ZigBee transceivers can be configured for concurrent control of specific ZigBee device from several ZigBee devices. In this way the commands can be sent from RC-100 (remote controller) and from PC simultaneously. When no command arrives, the robot is in the idle mode, so it returns to its initial pose and no poses are played anymore.

There are 3 major command types:

1. Body commands. In charge of the robot's body movement. Here are the pages are played.
2. Head commands. In charge of the robot's head movement. Those are direct commands to the head's dynamixels: tilt or pan.
3. Management commands. Those commands are designed for making some adjustments, to avoid the need for repeated code compilation just to change some default values. The major use of these commands is to adjust dynamixels offsets. The offsets are differences to be added to all dynamixels positions in all poses/pages. With this ability it's possible to make some adjustments to the robot on-the-fly (without compiling and updating the SW). For example, the user can open the robot's knees a little bit, or to lean the robot forward or backward by adjusting its ankle or hip dynamixels.

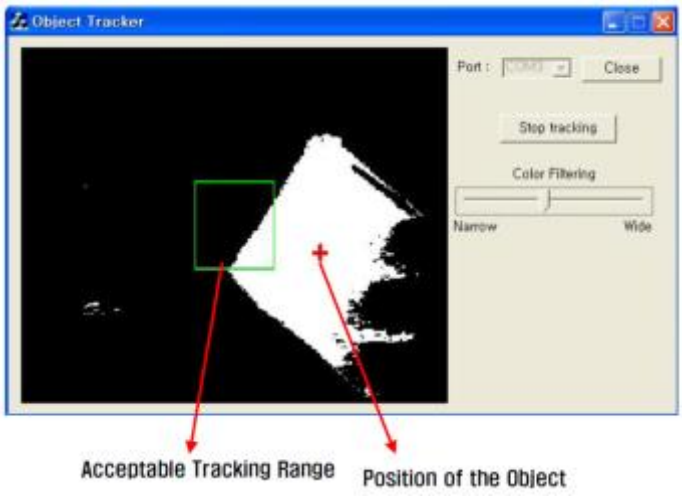
As mentioned above, the motion cycle includes constant check for sensors change. The change in sensors value may result in body movement change. For example:

1. Accelerometer detects falling in X-axis or Y-axis direction. As a result the appropriate offsets will be added to robot's ankle dynamixels to prevent falling.
2. DMS detects an object coming close at some distance during the "Step Forward" commands being constantly sent. The robot will try to avoid the obstacle by jumping to "turn right" pages, instead of continuing playing "walking forward" pages. When the way is clear, the robot will continue performing "walking forward pages".
3. Start button pressed during the robot performance. Robot will reboot and will perform its program from the start.

Image Processing Unit

Image Processing Unit is responsible for video processing and providing the Main Controller (not implemented yet) with the data regarding the object location on the screen. This unit implemented in C++ language, using the SW libraries provided by the manufacturer. It runs on the PC and the main reason for this is that AVR controller is too weak to deal with video data.
Another feature implemented in this unit is Object Tracking. It means, when the object is detected by the camera and it starts moving relatively to the camera (maybe it's the robot who actually moves) the commands to the robot's head are sent via ZigBee interface to keep the camera on the object.
The object detection is done by the following method:

1. Defining the object's color.
2. Keeping the robot's environment color (background) different from the object's color.
3. Averaging the object's color geometrical locations on the screen - getting the average coordinates.



Wireless Data Processing Unit

This unit is responsible for ZigBee commands transmitting and receiving.

It uses the manufacturer's library to activate the ZigBee device. The manufacturer implementation of ZigBee communication uses 2-bytes commands. In order to enable the simultaneous control of the robot by RC-100 and PC and to expand the given set of commands by the manufacturer's robot's Soccer Program, the need to expand ZigBee frames arose. So the manufacturer's library was changed to support alternatively 4-bytes commands. In this way, the RC-100 continues sending 2 bytes frames, but the PC sends 4 bytes frames. Actually, that's how the robot's ZigBee interface distinguishes between the 2 kinds of commands and decodes it appropriately. Due to the frame enlargement, the robot's head movement ability was expanded. Instead of sending the directions of movement (left, right, up, down) at maximum speed until the dynamixel reaches its maximum position, now it's possible to send the exact goal position and the speed to the robot's head dynamixels in the same frame.

The Wireless Data Processing Unit runs on the PC and is currently merged with the Image Processing Unit (due to non-implemented Main Controller).

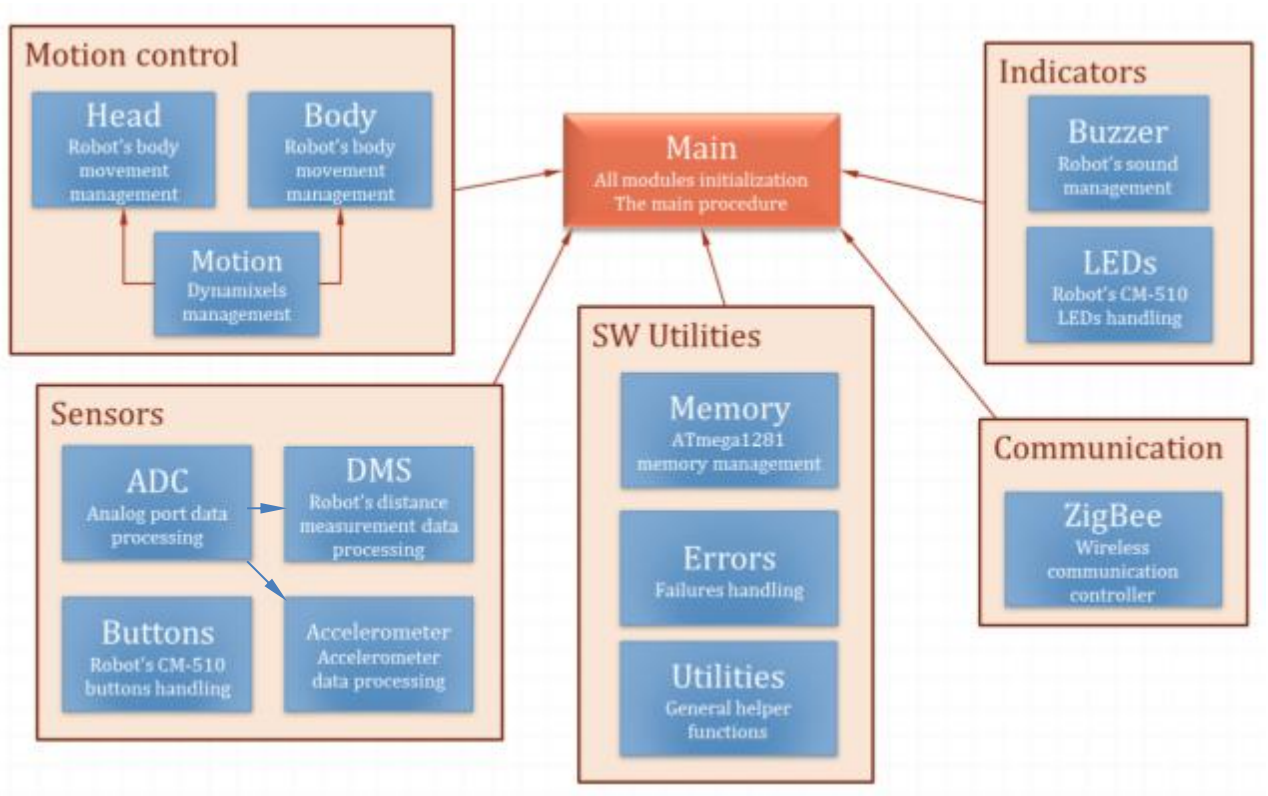
Implementation

Motion Controller

The Motion Controller implemented in C using AVR library and manufacturer's provided libraries.

General Structure

The general structure of the Motion Controller can be described as following:



Files overview

Generally, each module owns 2 files: <bics_modulename>.h with declarations and <bics_modulename>.c with its implementation. But there are some exceptions, as detailed below.
BICS is an abbreviation of Bioloid Internal Controller Software.

main.c

This is the main file, which holds the general infinite loop of robot's activity. It is the entry point for the entire Motion Controller project. Naturally, the main uses the APIs of all the rest modules.
The major internal functions implemented in the main are:

1. Motion cycle implementation - loading and playing pages according to the received ZigBee command.
2. Acknowledge ZigBee commands transmitting.
3. Page cache management (using the Random Replacing Policy).
4. Module self-testing procedures, such as:

```
ErrorCode testLeds();
ErrorCode testBuzzer();
ErrorCode testDMS();
ErrorCode testButtons();
ErrorCode testZigbee();
ErrorCode testBody();
ErrorCode testHead();
```

motion.c / motion.h

Those files implement the dynamixels access API.
The functions implemented here are:

```
void initMotion();
void setDynamixelPosition( int ID, int position, int speed );
int  getDynamixelPosition( int ID );
void setDynamixelPositionOffset( int ID, int offset, int speed );
void stopDynamixel( int ID );
void syncSetDynamixelPosition( int numberOfDynamixels, int dynamixelID[],
                               uint16_t positionVector[],
                               uint16_t speedVector[] );
void syncSetDynamixelSoftness( int numberOfDynamixels, int dynamixelID[],
                               uint8_t softnessVector[] );
void syncSetDynamixelTorque( int numberOfDynamixels, int dynamixelID[],
                             uint16_t torqueVector[] );
```

The functions, which begin with "sync", are the functions for concurrent writing to all the dynamixels on the bus.

memory.c / memory.h

Here implemented the access to the program space (FLASH) and copying of the pages to the data space.

The main functions are:

```
void copyPoseToDataSpace( Pose destPose, uint32_t sourcePose );
void copyPageToDataSpace( MotionPage destPage, uint32_t sourcePage );
```

body.c / body.h

Those files implement the robot's body motion, using the motion.h and the memory.h API.

The main functions here are:

```
void getCurrentPose( Pose pose );
void setPose( Pose newPose, int maxDynamixelSpeed, int poseHoldTime );
void getPoseFromPage( int pageNumber, int poseNumber, Pose destPose );

void getMotionPage( int pageNumber, MotionPage *destPage );
void playMotionPage( MotionPage *page, PoseOffsetVector poseOffsetVector );
void playPage( int pageNumber, MotionPage *page,
               PoseOffsetVector poseOffsetVector,
               int *nextPageNumber, int *exitPageNumber );

void initPoseOffsetVector( PoseOffsetVector poseOffsetVector );
void copyPoseOffsetVector( PoseOffsetVector sourcePoseOffsetVector,
                           PoseOffsetVector targetPoseOffsetVector );
void presetPoseOffsetVector( PoseOffsetVector poseOffsetVector );
void changeDynamixelOffset( int dynamixelID, int offset,
                            PoseOffsetVector poseOffsetVector );
void setDynamixelOffset( int dynamixelID, int offset,
                         PoseOffsetVector poseOffsetVector );
void initMotionPage( MotionPage *page );
```

head.c / head.h

Those files implement the robot's head motion, using the motion.h API.

The main functions here are:

```
void initHead();

void panHead(  int position, int speed );
void tiltHead( int position, int speed );

void moveHead( int panPosition, int panSpeed, uint16_t tiltPosition,
               uint16_t tiltSpeed );

void panHeadByOffset(  int offset, int speed );
void tiltHeadByOffset( int offset, int speed );

int getHeadPanPosition();
int getHeadTiltPosition();

void stopHeadPan();
void stopHeadTilt();
void stopHead();
```

zigbee.c / zigbee.h

Those files implement wireless communication API.

The main functions here are:

```
void    initZigbee();
Boolean isRCButtonPressed( int RCButton );
void    parseRxData( unsigned long int receiveData, ZigbeeRxPacket* packet );
```

ADC.c / ADC.h

Those files implement analog-to-digital port access. There are 6 such ports supported by the controller.

The port distribution in this project defined in ADC.h as:

```
typedef enum
{
    ADC_PORT_1 = 1,
    ADC_PORT_2 = 2,
    ADC_PORT_3 = 3,  // Accelerometer X
    ADC_PORT_4 = 4,  // Accelerometer Y
    ADC_PORT_5 = 5,  // DMS
    ADC_PORT_6 = 6,
} ADCPort;
```

The only function implemented here is:

```
int readValueFromADCPort( int portNum );
```

accelerometer.c / accelerometer.h

Those files implement accelerometer API, using ADC.h API.

The main functions here are:

```
int initAccelerometers( int *balancedX, int *balancedY );

int getAccelerometerX();
int getAccelerometerY();

void getBalancedOffsetVector( int balancedX, int balancedY,
                              PoseOffsetVectorPointer balancedOffsetVector,
                              PoseOffsetVectorPointer initialOffsetVector );
```

DMS.c / DMS.h

Those files implement Distance Measurement Sensor API, using ADC.h API.

The only function here is:

```
int getDMSValue();
```

buttons.c / buttons.h

Those files implement CM-510 Buttons functionality API (see Appendix F).

The only function here is:

```
    Button getButton ();
```

leds.c / leds.h

Those files implement CM-510 LEDs functionality API (see Appendix F).

The functions implemented here are:

```
void turnLedOn ( Led led );
void turnLedOff( Led led );
void toggleLed ( Led led );
```

buzzer.c / buzzer.h

Those files implement CM-510 buzzer functionality API (see Appendix G).

The functions implemented here are:

```
void playSound( double frequency, double delay );
void playNote( int octave, Note note, double delay );
```

utilities.c / utilities.h

Those files implement some helper functions, such as:

```
double getPower( int base, int power );
```

errors.c / errors.h

Those files implement errors handling function:

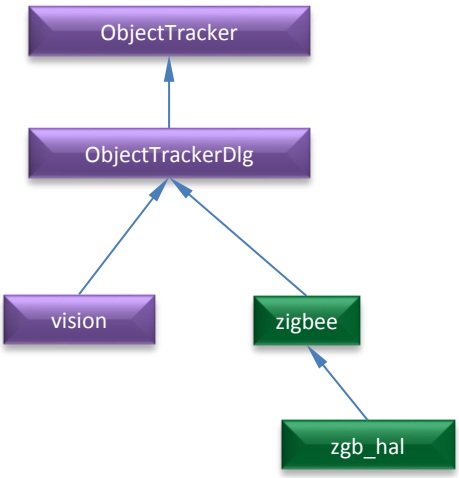
```
void handleError ( ErrorCode errorCode );
```

The error codes are defined in the Appendix G.

Image Processing Unit

The Image Processing unit is implemented in C++ using manufacturer's provided libraries.

As mentioned before this unit is merged with the Wireless Data Processing Unit. The general structure of those modules is following:



Files overview

ObjectTrackerDlg.cpp / ObjectTrackerDlg.h

Those files are based on manufacturer's SW example for object tracking. The manufacturer's example is very primitive, such that the result of its performance is Parkinson-style moving of a single dynamixel.

Those files were changed and improved to create smooth object tracking without unwanted movements.

Also, in those files the dialog window is implemented, in which besides the image provided by the robot's camera, but also useful parameters are displayed.

Here defined and implemented the CObjectTrackerDlg class, whose methods provide the full control of the dialog window, the image data received from the robot's camera and also the control of the robot's head by using ZigBee communication.

Most of methods here are handlers of events related to the dialog window, such as reaction to pressed button, closing/opening a window, paint and timer events, etc.

The main method here is:

```
void CObjectTrackerDlg::OnTimer( UINT nIDEvent )
```

This method handles the timer event, which means it works constantly after the window is initialized.

Here the data received from ZigBee is analyzed and commands are transmitted to the robot.

ObjectTracker.cpp / ObjectTracker.h

Those are Object Tracker application files. They include an application wrapper functions for interface with OS (Windows).

Here the instance of the CObjectTrackerDlg class mentioned above is created.

vision.c / vision.h

Those are manufacturer's library files for working with a camera. No changes were made here.

Wireless Data Processing Unit

The purpose of this unit is to send/receive ZigBee commands from/to PC to/from the robot and from the remote control (RC-100).

The unit is implemented in C language, using and changing the manufacturer's ZigBee API.

This unit is merged with the Image Processing unit, see the previous section for the structure diagram.

Files overview

zgb_hal.c / zgb_hal.h

Those files implement the Hardware Abstraction Layer of ZigBee. No changes were made in those files.

zigbee.c/ zigbee.h

Here the ZigBee communication functions are implemented, which provide interface to the Image Processing Unit for data transferring with the robot.

The methods implemented here are:

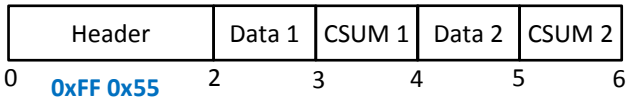
```
////////////// device control methods ////////////////////////
int  __stdcall zgb_initialize( int devIndex );
void __stdcall zgb_terminate();

////////// communication methods //////////////////////
int __stdcall zgb_tx_data(int data);
int __stdcall zgb_rx_check();
int __stdcall zgb_rx_data();
```

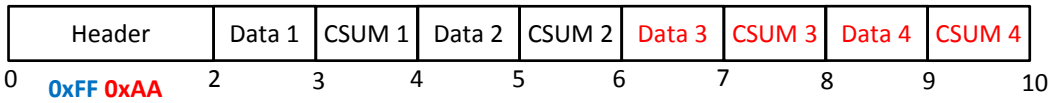
Also, the RC-100 buttons key values defined here:

```
////////// define RC-100 button key value ////////////////
#define RC100_BTN_U         (1)
#define RC100_BTN_D         (2)
#define RC100_BTN_L         (4)
#define RC100_BTN_R         (8)
#define RC100_BTN_1         (16)
#define RC100_BTN_2         (32)
#define RC100_BTN_3         (64)
#define RC100_BTN_4         (128)
#define RC100_BTN_5         (256)
#define RC100_BTN_6         (512)
```

As mentioned in General Description of this unit, the change to the manufacturer's ZigBee library was made. The manufacturer's frame transferred by the ZigBee devices has the following format (offsets in bytes):



It is not possible (means, too much effort ☺) to change the frame format sent from RC-100 controller. So, only the format of the frame, which is sent from the PC, was changed. The format is following:



Now, it is possible to transfer 4 bytes of data from PC instead of 2.

The Header of the frame was changed to be able distinguishing the packet received from RC-100 and from PC.

The CSUM (checksum) is simple the bitwise "NOT" of the data (CSUM i = ~ Data i).

Future possible development directions of the project

- Implementation of the Main Controller
- Placing the main controller on the robot (some lite weight, but strong computing device), to make the robot completely independent
- Upgrading the accelerometers (called “gyro” by the manufacturer) with the real gyros, to improve robot’s stability (falling prevention)
- Improving the motion algorithms and flows by the principle:
    - Efficiency in cost of modularity
- Upgrading the camera for better performance
- Improving image processing for new features to be possible, such as:
    - Easy obstacle overcoming instead of avoiding
- Adding some device to the robot, which will allow:
    - Navigation
    - Shortest path finding

Bibliography

1. ROBOTIS site: http://www.robotis.com/
2. Bioloid User's Guide, ROBOTIS (pdf)
3. Expert manual, ROBOTIS (pdf)
4. ZigBee Module ZIG-100, ROBOTIS (pdf)
5. Dynamixel AX-12, User's manual, ROBOTIS (pdf)
6. AVR LIBC site: http://www.nongnu.org/avr-libc/user-manual/pages.html
7. 8-bit Microcontroller with 64K/128K/256K Bytes In-System Programmable Flash, Atmel.
8. Bioloid based Humanoid Soccer Robot Design, Joerg Christian Wolf, Phil Hall, Paul Robinson, Phil Culverhouse, Centre for Robotics and Intelligent Systems, University of Plymouth, Drake Circus, Plymouth, PL4 8AA, United Kingdom (pdf)
9. The C++ Programming Language, Bjarne Stroustrup, 3rd. edition, AT&T Labs Murray Hill, New Jersey, ISBN 0-201-88954-4
10. Google

Appendix A - Dynamixel AX-12 parameters

Taken from bics_motion.h:

```
//-------------------------- [EEPROM] area ------------------------------------
#define P_MODEL_NUMBER_L            0x00    // Model number lower byte address
#define P_MODEL_NUMBER_H            0x01    // Model number higher byte address
#define P_VERSION                   0x02    // Firmware version address
#define P_ID                        0x03    // Dynamixel ID address
#define P_BAUD_RATE                 0x04    // Dynamixel baudrate address
#define P_RETURN_DELAY_TIME         0x05    // Return delay time address
#define P_LIMIT_TEMPERATURE         0x0b    // Limited temperature address
#define P_DOWN_LIMIT_VOLTAGE        0x0c    // Down limited voltage address
#define P_UP_LIMIT_VOLTAGE          0x0d    // Up limited voltage address
#define P_RETURN_LEVEL              0x10    // Status return level address

// AX-12 only parameters
#define P_CW_ANGLE_LIMIT_L          0x06    // CW limited angle lower byte address
#define P_CW_ANGLE_LIMIT_H          0x07    // CW limited angle higher byte address
#define P_CCW_ANGLE_LIMIT_L         0x08    // CCW limited angle lower byte address
#define P_CCW_ANGLE_LIMIT_H         0x09    // CCW limited angle higher byte address
#define P_MAX_TORQUE_L              0x0e    // Max torque lower byte address
#define P_MAX_TORQUE_H              0x0f    // Max torque higher byte address
#define P_ALARM_LED                 0x11    // Alarm LED address
#define P_ALARM_SHUTDOWN            0x12    // Alarm shutdown address
#define P_TORQUE_ENABLE             0x18    // Torque enable flag address
#define P_LED                       0x19    // LED on/off flag address
#define P_CW_COMPLIANCE_MARGIN      0x1a    // CW compliance margin address
#define P_CCW_COMPLIANCE_MARGIN     0x1b    // CCW compliance margin address
#define P_CW_COMPLIANCE_SLOPE       0x1c    // CW compliance slope address
#define P_CCW_COMPLIANCE_SLOPE      0x1d    // CCW compliance slope address
#define P_GOAL_POSITION_L           0x1e    // Goal position lower byte address
#define P_GOAL_POSITION_H           0x1f    // Goal position higher byte address
#define P_GOAL_SPEED_L              0x20    // Goal speed lower byte address
#define P_GOAL_SPEED_H              0x21    // Goal speed higher byte address
#define P_TORQUE_LIMIT_L            0x22    // Limited torque lower byte address
#define P_TORQUE_LIMIT_H            0x23    // Limited torque higher byte address
#define P_PRESENT_POSITION_L        0x24    // Present position lower byte address
#define P_PRESENT_POSITION_H        0x25    // Present position higher byte address
#define P_PRESENT_SPEED_L           0x26    // Present speed lower byte address
#define P_PRESENT_SPEED_H           0x27    // Present speed higher byte address
#define P_PRESENT_LOAD_L            0x28    // Present load lower byte address
#define P_PRESENT_LOAD_H            0x29    // Present load higher byte address
#define P_MOVING                    0x2e    // Moving state flag address
#define P_PUNCH_L                   0x30    // Punch lower byte address
#define P_PUNCH_H                   0x31    // Punch higher byte address

//------------------------------ [RAM] area ------------------------------------
#define P_PRESENT_VOLTAGE           0x2a    // Present voltage address
#define P_PRESENT_TEMPERATURE       0x2b    // Present temperature address
#define P_REGISTERED_INSTRUCTION    0x2c    // Registered instruction address
#define P_LOCK                      0x2f    // EEPROM lock flag address
```

Dinamyxel's parameters MIN/MAX values list (from AX-12.pdf):

Write Address	Writing Item	Length (bytes)	Min	Max
3(0X03)	ID	1	0	253(0xfd)
4(0X04)	Baud Rate	1	0	254(0xfe)
5(0X05)	Return Delay Time	1	0	254(0xfe)
6(0X06)	CW Angle Limit	2	0	1023(0x3ff)
8(0X08)	CCW Angle Limit	2	0	1023(0x3ff)
11(0X0B)	the Highest Limit Temperature	1	0	150(0x96)
12(0X0C)	the Lowest Limit Voltage	1	50(0x32)	250(0xfa)
13(0X0D)	the Highest Limit Voltage	1	50(0x32)	250(0xfa)
14(0X0E)	Max Torque	2	0	1023(0x3ff)
16(0X10)	Status Return Level	1	0	2
17(0X11)	Alarm LED	1	0	127(0x7f)
18(0X12)	Alarm Shutdown	1	0	127(0x7f)
19(0X13)	(Reserved)	1	0	1
24(0X18)	Torque Enable	1	0	1
25(0X19)	LED	1	0	1
26(0X1A)	CW Compliance Margin	1	0	254(0xfe)
27(0X1B)	CCW Compliance Margin	1	0	254(0xfe)
28(0X1C)	CW Compliance Slope	1	1	254(0xfe)
29(0X1D)	CCW Compliance Slope	1	1	254(0xfe)
30(0X1E)	Goal Position	2	0	1023(0x3ff)
32(0X20)	Moving Speed	2	0	1023(0x3ff)
34(0X22)	Torque Limit	2	0	1023(0x3ff)
44(0X2C)	Registered Instruction	1	0	1
47(0X2F)	Lock	1	1	1
48(0X30)	Punch	2	0	1023(0x3ff)

[Control Table Data Range and Length for Writing]

Appendix B - Motion page structure

As mentioned in the Memory Management section above, the motion pages reside in the FLASH memory region. The base address is 0x1e000. The page size is 512 bytes. Those and other constants are defined in bics_memory.h:

```
#define PAGE_0_ADDRESS              0x1e000  // Page no.0 address in flash memory
#define PAGE_SIZE                       512  // One page size (0x00200)
#define PAGE_POSE_DATA_LENGTH            64  // One pose size (0x00010) (next
                                             // pose's 1st dynamixel offset)

#define PAGE_NAME_MAX_LENGTH             13
#define PAGE_MAX_STEPS                    7  // Max number of poses on the page

// Page data offsets
#define PAGE_NAME_OFFSET                  0
#define PAGE_PLAY_COUNT_OFFSET           15
#define PAGE_PLAY_SPEED_OFFSET           16
#define PAGE_STEPS_NUMBER_OFFSET         20
#define PAGE_CONTROL_CODE_OFFSET         21
#define PAGE_SPEED_OFFSET                22
#define PAGE_DXL_SETUP_OFFSET            23
#define PAGE_ACCELERATION_TIME_OFFSET    24
#define PAGE_NEXT_OFFSET                 25
#define PAGE_EXIT_OFFSET                 26
#define PAGE_JOINT_SOFTNESS_OFFSET       32
#define PAGE_POSE_0_OFFSET               66  // The address of the 1st dynamixel
                                             // position on the page (byte 0x42)
#define PAGE_POSE_0_PAUSE_OFFSET        126
#define PAGE_POSE_0_SPEED_OFFSET        127
```

Appendix C - Joint names

Taken from bics_body.h:

```
typedef enum
{
   RIGHT_SHOULDER_PITCH =  1,  LEFT_SHOULDER_PITCH =  2,
   RIGHT_SHOULDER_ROLL  =  3,  LEFT_SHOULDER_ROLL  =  4,
   RIGHT_ELBOW_ROLL     =  5,  LEFT_ELBOW_ROLL     =  6,
   RIGHT_HIP_YAW        =  7,  LEFT_HIP_YAW        =  8,
   RIGHT_HIP_ROLL       =  9,  LEFT_HIP_ROLL       = 10,
   RIGHT_HIP_PITCH      = 11,  LEFT_HIP_PITCH      = 12,
   RIGHT_KNEE_PITCH     = 13,  LEFT_KNEE_PITCH     = 14,
   RIGHT_ANKLE_PITCH    = 15,  LEFT_ANKLE_PITCH    = 16,
   RIGHT_ANKLE_ROLL     = 17,  LEFT_ANKLE_ROLL     = 18
} JointName; // Dynamixel ID
```

Appendix D - Motion pages list

Taken from bics_motion.h:

```
typedef enum
{
   PAGE_INIT                        =  1, PAGE_BALANCE                    =  2,
   PAGE_FORWARD_RIGHT_FOOT_START    =  3, PAGE_FORWARD_LEFT_FOOT_START    =  4,
   PAGE_FORWARD_RIGHT_LEFT_FOOT     =  5, PAGE_FORWARD_LEFT_RIGHT_FOOT    =  6,
   PAGE_FORWARD_RIGHT_FOOT_EXIT     =  7, PAGE_FORWARD_LEFT_FOOT_EXIT     =  8,
   PAGE_FORWARD_RIGHT_LEFT_FOOT2    =  9, PAGE_FORWARD_LEFT_RIGHT_FOOT2   = 10,//?
   PAGE_GO_FORWARD_AND_TURN_RIGHT   = 11, PAGE_GO_FORWARD_AND_TURN_LEFT   = 12,
   PAGE_BACKWARD_RIGHT_FOOT_START   = 13, PAGE_BACKWARD_LEFT_FOOT_START   = 14,
   PAGE_BACKWARD_RIGHT_LEFT_FOOT    = 15, PAGE_BACKWARD_LEFT_RIGHT_FOOT   = 16,
   PAGE_BACKWARD_RIGHT_FOOT_EXIT    = 17, PAGE_BACKWARD_LEFT_FOOT_EXIT    = 18,
   PAGE_STEP_RIGHT_FRONT_PIVOT      = 19, PAGE_STEP_LEFT_FRONT_PIVOT      = 20,
   PAGE_STEP_RIGHT_BACK_PIVOT       = 21, PAGE_STEP_LEFT_BACK_PIVOT       = 22,
   PAGE_STEP_RIGHT                  = 23, PAGE_STEP_LEFT                  = 24,
   PAGE_STEP_RIGHT_FAST             = 25, PAGE_STEP_LEFT_FAST             = 26,
   PAGE_TURN_RIGHT                  = 27, PAGE_TURN_LEFT                  = 28,
   PAGE_GET_UP_FROM_CHEST           = 29, PAGE_GET_UP_FROM_BACK           = 30,

   PAGE_RESERVED_31                 = 31,

   PAGE_RIGHT_POWER_SHOT_PREP       = 32, PAGE_RIGHT_POWER_SHOT           = 33,
   PAGE_LEFT_POWER_SHOT_PREP        = 34, PAGE_LEFT_POWER_SHOT            = 35,
   PAGE_FORWARD_RIGHT_SHOT          = 36, PAGE_FORWARD_LEFT_SHOT          = 37,
   PAGE_RIGHT_SHOT                  = 38, PAGE_LEFT_SHOT                  = 39,
   PAGE_RIGHT_PASS                  = 40, PAGE_LEFT_PASS                  = 41,
   PAGE_BACKWARD_RIGHT_SHOT         = 42, PAGE_BACKWARD_LEFT_SHOT         = 43,

   PAGE_RESERVED_44                 = 44,

   PAGE_PREPARE_BLOCKING            = 45,
   PAGE_BLOCK_RIGHT                 = 46,
   PAGE_BLOCK_FRONT                 = 47,
   PAGE_BLOCK_LEFT                  = 48,

   PAGE_RESERVED_49                 = 49,

   PAGE_CHEER_1                     = 50,
   PAGE_CHEER_2                     = 51,
   PAGE_CHEER_3                     = 52,

   PAGE_BEAT_CHEST_1                = 53,
   PAGE_BEAT_CHEST_2                = 54,
} PageName;
```

Appendix E - Remote control commands (RC-100/PC)

Command	Buttons									
BODY										
Move forward	UP									
Move backward		DOWN								
Turn left			LEFT							
Turn right				RIGHT						
Move forward with left turn	UP		LEFT							
Move forward with right turn	UP			RIGHT						
Move forward (differently?)	UP									6
Step right				RIGHT					5	
Step left			LEFT						5	
Step right fast				RIGHT					5	6
Step left fast			LEFT						5	6
Step right with front pivot	UP			RIGHT					5	
Step left with front pivot	UP		LEFT						5	
Step right with back pivot		DOWN		RIGHT					5	
Step left with back pivot		DOWN	LEFT						5	
Torque off		DOWN					3		5	6
Torque on	UP				1				5	6
Stand up from chest	UP				1					
Stand up from back		DOWN			1					
Beat the chest			LEFT		1					
Cheer				RIGHT	1					
Kick forward with left foot	UP					2				
Power kick forward with left foot	UP					2				6
Kick forward with right foot	UP							4		
Power kick forward with right foot	UP							4		6
Kick backward with left foot		DOWN				2				
Kick left			LEFT			2				
Pass left				RIGHT		2				
Kick backward with right foot				RIGHT				4		
Kick right				RIGHT				4		
Pass right			LEFT					4		
Prepare for blocking							3			
Block front side	UP						3			
Block left side			LEFT				3			
Block right side				RIGHT			3			
HEAD										
Move head up	UP				1	2				
Move head down		DOWN			1	2				
Move head left			LEFT		1	2				
Move head right				RIGHT	1	2				
Move head up and left	UP		LEFT		1	2				
Move head up and right	UP			RIGHT	1	2				
Move head down and left		DOWN	LEFT		1	2				
Move head down and right		DOWN		RIGHT	1	2				
Stop head tilting	UP	DOWN			1	2				
Stop head panning			LEFT	RIGHT	1	2				
Stop head motion	UP	DOWN	LEFT	RIGHT	1	2				
Move head up and stop panning	UP		LEFT	RIGHT	1	2				
Move head down and stop panning		DOWN	LEFT	RIGHT	1	2				
Move head left and stop tilting	UP	DOWN	LEFT		1	2				
Move head right and stop tilting	UP	DOWN		RIGHT	1	2				
MANAGEMENT										
Ankle pitch tuning – forward	UP				1	2	3			
Ankle pitch tuning – backward		DOWN			1	2	3			
Knee pitch tuning – forward	UP				1	2	3	4		
Knee pitch tuning – backward		DOWN			1	2	3	4		
Hip roll tuning – inside			LEFT		1	2	3			
Hip roll tuning – outside				RIGHT	1	2	3			
Ankle roll tuning – inside			LEFT		1	2	3	4		
Ankle roll tuning – outside				RIGHT	1	2	3	4		

Appendix F - CM-510 LEDs, buttons and buzzer codes

Taken from bics_leds.h:

```
typedef enum
{
	LED_POWER	= 0x01,
	LED_TX		= 0x02,
	LED_RX		= 0x04,
	LED_AUX		= 0x08,
	LED_MANAGE	= 0x10,
	LED_PROGRAM	= 0x20,
	LED_PLAY	= 0x40,
	LED_ALL		= 0x7F,
} Led;
```

Taken from bics_buttons.h:

```
typedef enum
{
	BUTTON_NONE	= 0x00,
	BUTTON_START	= 0x01,
	BUTTON_UP	= 0x10,
	BUTTON_DOWN	= 0x20,
	BUTTON_LEFT	= 0x40,
	BUTTON_RIGHT	= 0x80,
} Button;
```

Taken from bics_buzzer.h:

```
typedef enum
{
	NOTE_C		= 1,	NOTE_B_SHARP	= 1,
	NOTE_C_SHARP	= 2,	NOTE_D_FLAT	= 2,
	NOTE_D		= 3,
	NOTE_D_SHARP	= 4,	NOTE_E_FLAT	= 4,
	NOTE_E		= 5,	NOTE_F_FLAT	= 5,
	NOTE_F		= 6,	NOTE_E_SHARP	= 6,
	NOTE_F_SHARP	= 7,	NOTE_G_FLAT	= 7,
	NOTE_G		= 8,
	NOTE_G_SHARP	= 9,	NOTE_A_FLAT	= 9,
	NOTE_A		= 10,
	NOTE_A_SHARP	= 11,	NOTE_B_FLAT	= 11,
	NOTE_B		= 12,	NOTE_C_FLAT	= 12,
} Note;
```

Appendix G - Motion Controller's error codes

Taken from bics_errors.h:

```
typedef enum // Defines the errors codes
{
	SUCCESS                          =  0,
	GENERAL_FAILURE                  = -1,
	INVALID_DYNAMIXEL_ID             = -2,
	INVALID_DYNAMIXEL_POSITION       = -3,
	INVALID_DYNAMIXEL_SPEED          = -4,
	INVALID_HEAD_TILT_POSITION       = -5,
	INVALID_HEAD_PAN_POSITION        = -6,
	DYNAMIXEL_COMMUNICATION_FAILURE  = -7,
	INVALID_PAGE_NUMBER              = -8,
	INVALID_POSE_NUMBER              = -9,
	INVALID_NUMBER_OF_DYNAMIXELS     = -10,
	INVALID_PAGE_NUMBER_TO_PLAY      = -11,
	ACCELEROMETER_ERROR              = -12,
	DIVISION_BY_ZERO                 = -13,
	UNDEFINED_VALUE_RESULT           = -14,
} ErrorCode;
```