Reference Guide on Assembling and Operating Lynxmotion's A-Pod Hexapod (August 2015)

Lisandra Jimenez, Lab Researcher, *Biomimetics & Cognitive Robotics Lab, Brooklyn College*

***Abstract*—This paper is a reference guide for assembling the robot's hardware and electronic components. It will also provide PC troubleshooting tips on connectivity and operation using the Lynxterm software. Other software are also mentioned for more advanced functionality of the 3DOF Hexapod. In addition, test codes for connectivity and gait setup via Arduino IDE are provided. Use this document as an instruction set. Note that we are not using the PS2 controller for this project.**

TABLE OF CONTENTS

KEYWORDS: Servo, Actuator, Molex, Inverse Kinematics, Jumper, Calibration

I. INTRODUCTION

THIS document contains instructions and troubleshooting tips for assembling and operating the Lynxmotion's A-Pod Hexapod.

II. HARDWARE COMPONENTS

The hardware contains 25 servos (mechanical joints), 1 SSC-32 microcontroller board, 1 BotBoarduino logic board, and the body. The Lynxmotion's A-Pod Hexapod Kit (including the assembly hardware) contains the following components: 1 body, 6 legs, 1 tail, 1 head/mandibles. The body's back has 6 exterior outlines for placement of the 6 horizontal servos (shoulder); each of these servos is connected to the vertical servo (elbow) found at the top of each leg. Each leg has another vertical servo (wrist) for a total of 18 servos connecting the body and the legs. In addition, the tail has 2 servos: horizontal and vertical; the head has 5 servos: the left and right mandibles have one servo each, the neck has 1 horizontal ("no" expression), 1 vertical ("yes" expression), and another vertical ("maybe" expression). The kit also comes with 6 leg switches and 6 10k ohm resistors to be soldered to each switch. When the tip of each leg is pressed, it clicks the switch to the true/false (0/1) state (see Fig. 1).

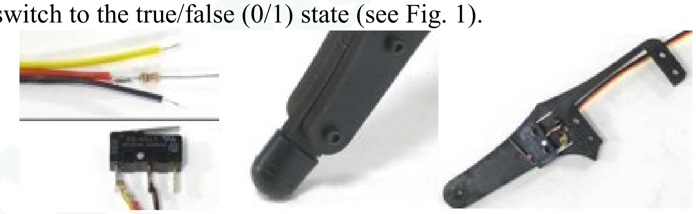


Fig. 1. Leg Switch on Tibia [1]

A. Servo

A servo is an actuator. It functions as a mechanical joint. The model used for this project is the HS645MG. Its horn is a round plate that can rotate 180° (see Fig. 2).

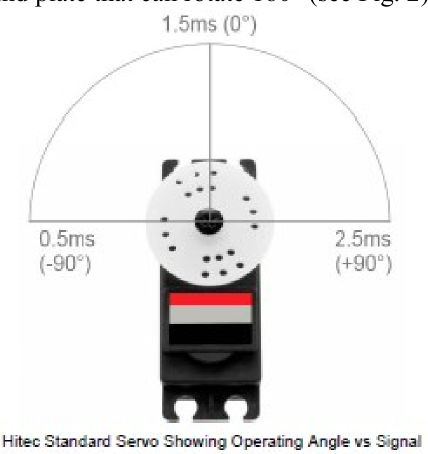
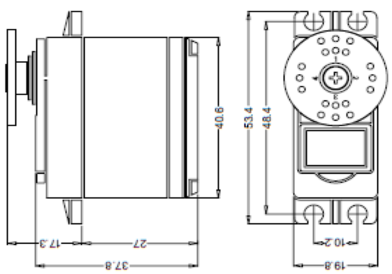


Fig. 2. Servo with plastic horn (Operating Angle vs. Pulse Signal) [2]

However, the code does not use degrees to determine the position of the servo. This servo can operate 180° when given a pulse signal ranging from 500usec (0.5ms) to 2500usec (2.5ms). In terms of angles, 500usec corresponds to -90°, 1500usec corresponds to the centered position of 0°, and 2500usec corresponds to 90°. It is shipped in its centered position as shown in Fig. 3. Hitec servo splines have 24 teeth, which means the servo horn can be moved around in 15° increments. During installation, if the servo rotates off-center

August 2015. This project is supported by the Biomimetic and Cognitive Robotics Lab of Brooklyn College under Prof. Grasso's leadership.
Lisandra Jimenez is a lab researcher for the Biomimetic and Cognitive Robotics Lab of Brooklyn College, Brooklyn, New York, and has just earned her Bachelors degree in Education and Computer Science (e-mail: lisandra.jimenez.NYC@gmail.com).
REV 9/6/2015

because the horn was accidentally moved out of position, it can be re-centerd manually by turning the horn to the extreme left and then to the extreme right. If necessary, make a notch above the hole on the horn where you have estimated to be the center. Carefully remove the horn while maintaining the spline's position and re-position the notch to face northward in its re-centered position. Fig. 3 is an illustration of a centered servo with its temporary plastic horn as it appears when it is shipped from the manufacturer. The servo has 3 twisted wires whose ends are attached together with a molex. The colors yellow-red-black represents signal-power-ground, respectively. The twisted wire is connected to one of the numbered channels (pins) on the SSC-32 board as shown in the schematics in Figures 5 and 6. The voltage for the servo can range from 4.8 to 6.0 volts depending on its model. The operating voltage for the Hitec HS-645 servo is 6.0V. More specs are shown in Fig.3.



CONTROL SYSTEM	:+PULSE WIDTH CONTROL 1500usec NEUTRAL	
OPERATING VOLTAGE RANGE	:4.8V TO 6.0V	
OPERATING TEMPERATURE RANGE	:-20 TO +60°C	
TEST VOLTAGE	:AT 4.8V	:AT 6.0V
OPERATING SPEED	:0.24sec/60° AT NO LOAD	:0.2sec/60° AT NO LOAD
STALL TORQUE	:7.7kg.cm(106.93oz.in)	:9.6kg.cm(133.31oz.in)
OPERATING ANGLE	:45°/ONE SIDE PULSE TRAVELING 400usec	
DIRECTION	:CLOCK WISE/PULSE TRAVELING 1500 TO 1900usec	
IDLE CURRENT	:8.8mA	:9.1mA
RUNNING CURRENT	:350mA	:450mA
DEAD BAND WIDTH	:8usec	
CONNECTOR WIRE LENGTH	:300mm(11.81in)	
DIMENSIONS	:40.6x19.8x37.8mm(1.59x0.77x1.48in)	
WEIGHT	:55.2g(1.94oz)	

Fig. 3. Servo Specifications [3]

Note: After assembly, each servo should be calibrated. Calibration will make its center position of 1500usec which corresponds to the middle position of 0° to be more precise. If the center is off by a lot, then re-assembly of the affected servo is required in ensure to ensure a graceful gait. A simple calibration software may be downloaded here: http://www.lynxmotion.com/p-854-free-download-lynx-hexapod-calibration-software.aspx and for instructions, scroll down to Steps 6 through 8 here: http://www.lynxmotion.com/images/html/build99f.htm. This program is capable of setting the SSC-32 register offsets for all servos. The DB9 serial cable will need to be connected to the SSC-32 board to perform the calibration.

III. Tutorials

Detailed assembly instructions may be found on Lynxmotion's website:

APOD-KT Information
NOTE: **We are NOT using PS2 Controls for our autonomous robot.**
1 Assembly Guide for A-Pod Leg
2 Assembly Guide for A-Pod Body
3 Assembly Guide for A-Pod Tail
4 Assembly Guide for A-Pod Mandibles
5 Tutorial for A-Pod PS2 Control (BotBoard2)
6 Tutorial for Servo Mid Position
7 Tutorial for Servo Horn Information
8 Tutorial for preparation of the laser-cut Lexan used in the kits

Assembly Guide 2

- http://www.robotshop.com/forum/lynxmotion-a-pod-buildlog-guide-t12491

BotBoarduino Manuals

- http://www.lynxmotion.com/images/html/build185.htm
- http://www.lynxmotion.com/images/html/build99f.htm

SSC-32 Manual

- http://www.lynxmotion.com/images/html/build136.htm

SSC-32 Binary Commands

- http://www.lynxmotion.com/images/html/build177.htm

SSC-32 General Purpose Sequencer

- http://www.lynxmotion.com/images/html/build137.htm

IV. Electrical Components

There are two circuit boards:

A. SSC-32 board

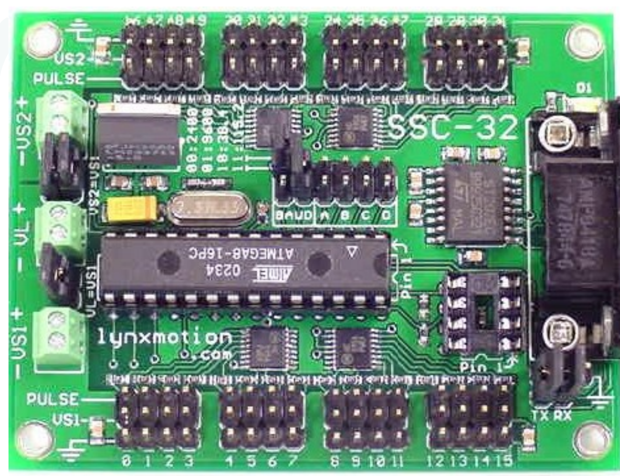


Fig. 4. SSC-32 Ver. 2.0 [4]

The SSC-32 Ver 2.0 used for our project is an older model which has a DB9 serial port connector attached to it. It does not store data. This board is used to control the servos via the Lynxterm application for the purposes of calibration and testing each servo for functionality. When the SSC-32 is first connected to the logic board, the green LED light remains lit. This is not a power indicator. If the board is connected to the logic board, a steady green light indicates that it is ready to receive data. The green LED for the SSC-32 will blink when it is receiving data. Or, if connected with the serial cable to the PC, the green LED will not be lit until it receives data again. The serial connection is used to transmit calibration data to the board via the Lynxterm application. The data received from Lynxterm is used to manipulate each servo. When calibrating, turn on the power source for the servos, i.e., the 6.0 Volt Ni-MH 2800mAh Servo Battery Pack. The SSC-32 board can support 32 servos. See Fig. 5 and Fig. 6 for the wiring diagram.

Servo Letter Definitions		
Left Right	Rear Middle Front	Knee Vertical Horizontal

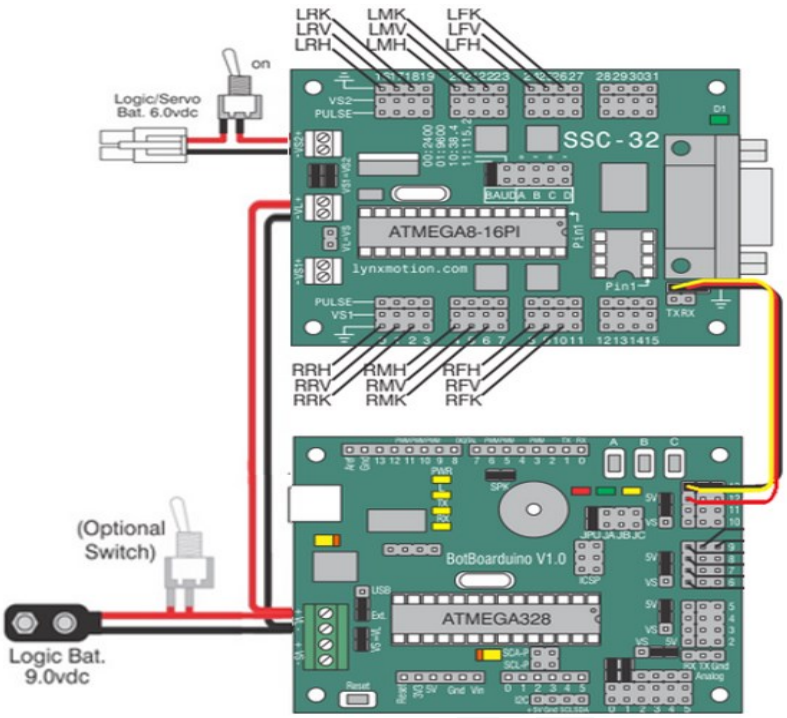


Figure 5. Wiring [3]

V. Power Source [6]

A. SSC-32 Ver 2.0

The SSC-32 controls the servos and the 6V Ni-MH 2800mAh battery pack is connected to a harness which is wired to the board's VS1 + and – terminals. The NiCad & Ni-MH Universal Smart Charger is recommended to recharge the battery. There are 2 banks of servo pins which are connected together by plugging a jumper on the VS1=VS2 pins so that both banks can share the same battery source (VS means voltage servos).

There is also the option of having the logic board share the same battery power source as the SSC-32 by plugging a jumper on the *VS=VL* pins (VL means voltage logic). However, this is not recommended due to battery drainage.

When testing the functionality of the servos, disconnect the boards from each other by removing the tx/rx/gnd wire and plugging in 2 jumpers in its place, vertically. Connect the DB9 serial cable to the SSC-32 board. On the SSC-32 board, set the baud rate to 115.2K by plugging 2 jumpers vertically over the pins marked *baud* (refer to the illustration on the SSC-32 user guide). On the PC side, go to device manager, and verify that the baud rate for the COM device is also set to 115.2K. Also make certain that the COM port number matches the number on the Lynxterm app under the field "COM Port." Since VS≠VL, both battery power sources must be on. The only purpose of the serial cable is to transmit data to the SSC-32 board. Open the Lynxterm application and begin begin testing each servo.

Servo Channels and Schematic Label Explanations					
RRH / 00	Right Rear Horizontal Hip	**LRH** / 16	Left Rear Horizontal Hip	**RM** / 28	Right Mandible
RRV / 01	Right Rear Vertical Hip	**LRV** / 17	Left Rear Vertical Hip	**LM** / 29	Left Mandible
RRK / 02	Right Rear Knee	**LRK** / 18	Left Rear Knee	**AP** / 30	Abdomen Pan
RMH / 04	Right Middle Horizontal Hip	**LMH** / 20	Left Middle Horizontal Hip	**AT** / 31	Abdomen Tilt
RMV / 05	Right Middle Vertical Hip	**LMV** / 21	Left Middle Vertical Hip	**HP** / 12	Mandible Pan
RMK / 06	Right Middle Knee	**LMK** / 22	Left Middle Knee	**HR** / 13	Mandible Rotate
RFH / 08	Right Front Horizontal Hip	**LFH** / 24	Left Front Horizontal Hip	**HT** / 14	Mandible Tilt
RFV / 09	Right Front Vertical Hip	**LFV** / 25	Left Front Vertical Hip		
RFK / 10	Right Front Knee	**LFK** / 26	Left Front Knee		

Figure 6. Servo Channels and Schematic Labels [5]

B. BotBoarduino (logic board)

The BotBoarduino logic board has an USB connector and it can store uploaded code from the Arduino IDE. It has a yellow power indicator LED that remains lit when it has an active power source, either via USB cable or the 9V battery. The wires for the 6 leg switches are connected to the logic board on channels 2-5 or 10-13. The manufacturer has set aside channels 6-9 for the PS2 controller which will not be used for this project.

B. *BotBoarduino*

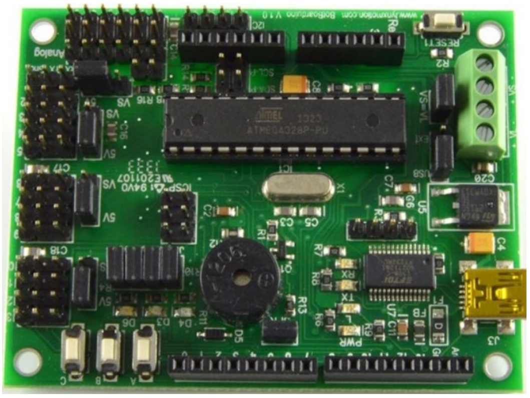


Fig. 7. BotBoarduino [7]

When using the USB cable to upload code to the logic board, if the power indicator LED is not lit on the BotBoarduino, then do the following: 1) remove the triple wires connected to its tx/rx/gnd pins, and 2) move the jumper (a very small, rectangular black cap) from the *ext* pin which stands for external power source such as a battery, to the USB pin. The power indicator should light.

After uploading code to the logic board, disconnect the USB cable, turn on the logic board's 9V battery, re-connect the wire to the tx/rx/grd pins on both boards, change the jumpers back to the *ext* pin on the logic board, clear the space around the robot, turn on the servo's 6V battery and the robot should spring into action.

C. *Connecting both boards*

The diagram for the power setup is shown in Fig.5. Each board has a tx/rx/grd connection which consists of 3 pins: 1 pin is for transmitting data, 1 pin is for receiving data, and 1 pin is for ground. After uploading the code into the logic board, and after removing the USB cable and powering the batteries, etc., connect both boards by connecting the triple wire to the tx/rx/gnd pins on both boards. Make sure the jumper is set to *ext* on the BotBoarduino.

VI. SOFTWARE

A. *Lynxterm*

Lynxterm makes it easy to quickly test all functionality of the SSC-32 Servo Controller. It has more features than the simple Hex-Calibration app. Tip: after adjusting each servo, take a screen snapshot of the offsets just in case the app defaults to the basic settings (note that the position offset is restricted to -100us to 100us which is around 15°). The snapshot will give serve as a reference to quickly enter the offset calculations in the field next to each channel instead of having to play with the control bar to readjust each servo again.

Connect the DB9 serial cable, download the driver and save it to the desktop for easy retrieval since it may be required every time the app is opened. The driver is found here under Step 3: http://www.lynxmotion.com/images/html/build184.htm. Then download the app here: http://www.lynxmotion.com/p-567-free-download-lynxterm.aspx. When the app is opened, click on "Firmware," browse to the location of the driver file and click "Begin Update." Check the com port to make sure it is set correctly (check the PC's device manager too and the baud should be set to 115.2k) then follow the instructions here: http://www.lynxmotion.com/images/html/build184.htm.

B. *Arduino IDE*

The Arduino Integrated Development Environment may be downloaded here: https://www.arduino.cc/en/Main/Software. Programming and uploading the code to the logic board is done through this IDE. The relevant libraries will need to be downloaded too.

Click here and scroll down to Step 5: http://www.lynxmotion.com/images/html/build99f.htm to find the code that makes use of the **inverse kinematics** which provides the most optimal configuration for movement of the legs. It will need to be modified for an autonomous project. The zip file may be accessed directly from here: https://github.com/Lynxmotion/3DOF-4DOF-Hex/zipball/Botboarduino.

A very useful tutorial and collection of basic codes to test connectivity and functionality are found here: https://www.arduino.cc/en/Tutorial/HomePage.

C. *Other Controller Software*

As of this writing, I have made another discovery of an interesting controller using VisualBasic, and it includes programming that also uses inverse kinematics.

- VB Windows Hexapod Control Program based on Xan's Code: http://www.robotshop.com/forum/vb-windows-hexapod-control-program-based-on-xan-s-code-t4397
- VB zip file https://app.box.com/shared/eq98sfq2ir
- Setup and operations guide for the VB.net control program V1.0 https://app.box.com/shared/snrj6ccm6d

VII. CODE

A. *Test for Connectivity*

Blink: A simple Arduino code to test the upload function to the logic board. Temporarily remove any wires connected to Pin 13. If the upload is successful, the LED on the board will blink continuously. The example below is from the public domain:

```
/*
  Blink
  Turns on an LED on for one second, then off for one second, repeatedly.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;
// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}
// the loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

B. Walk Forward Code

This code was modified to add the 3rd degree of motion by adding parameters to the function *servoMove()* to include the 3 tibias (lower leg/wrist) for each tripod gait. Tripods A and B each control 3 legs. Tripod A moves the right middle leg and the left front and rear legs. Tripod B moves the left middle leg and the right front and rear legs. The middle legs will have the greatest range of motion so that it can move north and south from its center. Its initial position will face north. Each front leg whose initial position will be aligned with its front shoulders will lift towards the north (head), press down on the ground and pull its body forward until the front leg returns to its initial position. Each hind leg whose initial southernmost position is set closest towards the tail, will lift and move northward up to but slightly behind its rear shoulders, press down on the ground and push its body forward until the hind leg returns to its initial southernmost position towards the tail (see Fig. 8).

Fig. 8. Position of Legs While in Motion

Source Code: The code is found in **Appendix A**.

VIII. Results

The modified *walk forward* gait raises the femur (upper leg) closer to the body and drops the tibia so that the body doesn't drag on the floor.

IX. Conclusion and Future Work

A. Sensors, Switches and Algorithm Suitable for Autonomy

The following website has useful information on sensors: http://roborobotics.com/Robotics/Sensors/robot-sensors.html.

A test code can be written for the existing leg switches that will perform the following: When the hexapod is lifted from the ground, all switches will be in a state of 0, the hexapod will pause its current tripod gait sequence and while it is in midair, it will switch to another tripod sequence or it can just drop its legs and wiggle its tail. When it is returned to the ground, it will resume its original tripod sequence.

Infrared range-finder sensors that can read the forward distance of an object can be installed, and code written to protect the Hexapod from crashing into walls.

A mini photocell light sensor may be used to have the Hexapod search for a dark area.

A new analog CO_2 sensor just came out on the market (see Fig. 9). It detects CO_2 and is compatible with Arduino. The Hexapod can sound off a beep whenever it perceives the gas.

Fig. 9 CO2 Sensor Compatible with Arduino [8]

The algorithm suitable for building an autonomous robot is discussed below.

B. The Ripple Gait and Inverse Kinematics

The ripple gait is where the tripod group move each leg in a cascading form. For example, the sequence for Tripod A's legs in a ripple gait would be: left front -short time delay(std)- right center -std- left rear -std. Instead of all 3 legs moving at the same time, leg 1 is followed by leg 2 which is then followed by leg 3, with less than a second time delay intervals between each step so the ripple effect can barely be discerned with the naked eye.

For autonomy, it is crucial to use an inverse kinematic algorithm in your code.

"Robot kinematics applies geometry to the study of the movement of multi-degree of freedom kinematic chains that form the structure of robotic systems. The emphasis on geometry means that the links of the robot are modeled as rigid bodies and its joints are assumed to provide pure rotation or translation.

Robot kinematics studies the relationship between the dimensions and connectivity of kinematic chains and the position, velocity and acceleration of each of the links in the robotic system, in order to plan and control movement and to compute actuator forces and torques. Inverse kinematics specifies the end-effector location and computes the associated joint angles."[9]

This requires very advanced programming skills, a lot of patience and undivided time. Just to give you an idea of its complexity, I have pasted the source code which is publicly known to have bugs. It was written by Jeroen Janssen (aka Xan) in visual basic for a PS2 controller. The code was later converted to C++ and Arduino by Kurt Eckhardt. I believe the original version in VB may have little to no bugs. The entire Arduino code is found in **Appendix B**. For C++ use the following link: https://github.com/Lynxmotion/3DOF-4DOF-Hex/zipball/Botboarduino.

I have also pasted a simpler Arduino code that uses inverse kinematics, but note that it's for a robotic arm and it uses the PS2 controller. The source code is found in **Appendix C** and here: https://raw.githubusercontent.com/EricGoldsmith/AL5D-BotBoarduino-PS2/master/PS2_IK_Control/PS2_IK_Control.ino.

REFERENCES

[1] Lynxmotion (acquired by RobotShop Inc. in 2012, Quebec Canada) [Online]. Available: http://www.lynxmotion.com/images/html/build180.htm

[2] Lynxmotion (acquired by RobotShop Inc. in 2012, Quebec Canada) [Online]. Available: http://www.lynxmotion.com/images/data/lynxmotion_ssc-32u_USB_user_guide.pdf

[3] Phidgets [Online]. Available: http://www.phidgets.com/documentation/Phidgets/3204_0_hs645.pdf

[4] Roborobotics [Online]. Available: http://roborobotics.com/Animatronics/show-control-hardware/Servo-controller.html

[5] Lynxmotion (acquired by RobotShop Inc. in 2012, Quebec, Canada) [Online]. Available: http://www.lynxmotion.com/images/html/build99f.htm

[6] J. Arcand (September 2012). RobotShop Inc., Quebec, Canada. [Online] Available: http://www.robotshop.com/blog/en/guide-power-supply-options-lynxmotion-controllers-12178

[7] RobotShop Inc., Quebec, Canada. [Online] Available: http://www.robotshop.com/en/lynxmotion-botboarduino-robot-controller.html

[8] Lynxmotion (acquired by RobotShop Inc. in 2012, Quebec, Canada) [Online]. Available: http://www.robotshop.com/en/co2-sensor-arduino-compatible.html

[9] Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Robot_kinematics#Inverse_kinematics

APPENDIX A

```
/*
Author: Lisandra Jimenez
This program lets BotBoarduino send commands to LynxMotion SSC-32 board to make hexapod robot move forward, which is an eight step (stage) process and each step take 1 second to complete. Adapted from Amrik Singh's code.

Added additional parameters to include movement for each tibia (lower arm/wrist) for more flexibility in movement.
*/
void setup() {
  Serial.begin(115200);     // begin communication using 115.2K baud rate
  while(!Serial){
    ;// wait while connecting
  }
  #define MOVETIME 500  //a constant to change the stride duration of each step
}
/*
tripodA: right front leg(RFL), right rear leg(RRL) and left middle leg(LML)
tripodB: left front leg(LFL), left rear leg(LRL) and right middle leg(RML)
*/

void triPodA_pickUp(){
   servoMove(1,1700,9,1700,21,1200,2,1900,10,1900,22,1000,MOVETIME);
  }
void triPodA_forward(){
   servoMove(0,1700,8,1650,20,1300,2,1900,10,1900,22,1100,MOVETIME);
}
void triPodA_onGround(){
   servoMove(1,1500,9,1500,21,1400,2,1900,10,1900,22,1100,MOVETIME);   // #4 raised 21 to 1400, raise 1 and 9 by 100
  }
void triPodA_push(){
   servoMove(0,1500,8,1500,20,1500,2,1900,10,1900,22,1100,MOVETIME);
}

void triPodB_pickUp(){   servoMove(17,1300,25,1300,5,1550,18,1100,26,1100,6,1900,MOVETIME);
  }
void triPodB_forward(){   servoMove(16,1300,24,1350,4,1700,18,1100,26,1100,6,1900,MOVETIME);
}
void triPodB_onGround(){   servoMove(17,1500,25,1500,5,1350,18,1100,26,1100,6,1900,MOVETIME);   // #4 lowered 5 to 1350
  }
void triPodB_push(){   servoMove(16,1500,24,1500,4,1500,18,1100,26,1100,6,1900,MOVETIME);
}

/* Comment Section Follows:
  void servoMove()
Input:
     byte servo1 - the pin number of a servo to be moved
     int position1 – the end position for servo1 (values are 500-2500)
     byte servo2 rhw pin number of the second servo to move
     int position2 the end position for servo2
     byte servo3 - the pin number of the third servo to move
     int position - the end position for servo3
    byte servo4 – the pin # of the fourth servo to move (tibia’s initial setting)
    int tibpos1 – the end position for the first tibia (repeated for tibia 2 and 3)
     int time - the time, in mS to complete the 3 servo moves (MOVETIME constant is set to 500)
Process:
     This function allows 6 servos (i.e. the elbows and wrists for one tripod) to be moved simultaneously to the desired positions and includes a stride duration parameter in which we request that the hexapod complete the move in the requested time.  The
```

```
function converts the commands to a format which the SSC-32 board will be able to understand.  Please refer to the SSC-32 manual (http://www.lynxmotion.com/images/html/build136.htm) for this and more commands.

Output:
      This function returns a void value, but the output becomes the actual movement of the hexapod
End of Comment
*/

void servoMove(byte servo1, int position1, byte servo2, int position2, byte servo3, int position3, byte servo4, int tibpos1, byte servo5, int tibpos2, byte servo6, int tibpos3, int time) {

   Serial.print("#"); //i.e. the # of the pin
   Serial.print(servo1);
   Serial.print(" P");
   Serial.print(position1); //i.e. the position we want the servo1 to move to
   Serial.print("#");
   Serial.print(servo2);
   Serial.print(" P");
   Serial.print(position2);
   Serial.print("#");
   Serial.print(servo3);
   Serial.print(" P");
   Serial.print(position3);
   Serial.print("#");          // 1st tibia pin number
   Serial.print(servo4);
   Serial.print(" P");
   Serial.print(tibpos1);      //  first tibia position
   Serial.print("#");          // 2nd tibia pin number
   Serial.print(servo5);
   Serial.print(" P");
   Serial.print(tibpos2);         // second tibia position
   Serial.print("#");          // 3rd tibia pin number
   Serial.print(servo6);
   Serial.print(" P");
   Serial.print(tibpos3);      // third tibia position
   Serial.print(" T");
   Serial.print(time);
   Serial.println();         // carriage return sends the Serial command
   delay(time);              // force Arduino to wait for the move to complete
}

bool initialStartDelay = true;  // this variable allows the one time delay before
                                // the hexapod starts to walk

// the loop routine runs over and over forever

void loop() {

/*
The hexapod picks up tripodA, brings it forward, brings tripodA on ground to hold the weight and then picks up tripodB, brings it forward, and the "tripodA" push robot forward and next tripodB makes contact with ground, and also pushes the hexapod forward
*/

//enter this loop only while initialStartDelay is true for one-time pause
  if (initialStartDelay){
   delay(6000);    // delay(do nothing) for 6 seconds
   initialStartDelay = false; // set the variable to false so this loop is never entered again
  }
```

```
triPodA_pickUp();
triPodA_forward();
triPodA_onGround();
triPodB_pickUp();
triPodB_forward();
triPodA_push();      // this is what propels the hexapod forward
triPodB_onGround();
triPodB_push();      // this also propels the hexapod forward
}
```

APPENDIX B

```
//==================================================================================
//Project Lynxmotion Phoenix Hexapod
//Description: Phoenix software
//Software version: V2.0
//Date: 29-10-2009
//Programmer: Jeroen Janssen [aka Xan]
//        Kurt Eckhardt(KurtE) converted to C and Arduino
//
// This version of the Phoenix code was ported over to the Arduino Environement
// and is specifically configured for the Lynxmotion BotBoarduino
//
//NEW IN V2.1 (2013-05-17)
//   - setup() made more generic by replacing exact pin-n° by PS2_CMD
//NEW IN V2.X
//==================================================================================
//
//KNOWN BUGS:
//    - Lots ;)
//
//==================================================================================
// Header Files
//==================================================================================

#if ARDUINO>99
#include <Arduino.h>
#else
#endif
#include <PS2X_lib.h>
#include <pins_arduino.h>
#include <SoftwareSerial.h>
#include "Hex_globals.h"
#define BalanceDivFactor 6    //;Other values than 6 can be used, testing...CAUTION!! At your own risk ;)

//--------------------------------------------------------------------
//[TABLES]
//ArcCosinus Table
//Table build in to 3 part to get higher accuracy near cos = 1.
//The biggest error is near cos = 1 and has a biggest value of 3*0.012098rad = 0.521 deg.
//-    Cos 0 to 0.9 is done by steps of 0.0079 rad. [1/127]
//-    Cos 0.9 to 0.99 is done by steps of 0.0008 rad [0.1/127]
//-    Cos 0.99 to 1 is done by step of 0.0002 rad [0.01/64]
//Since the tables are overlapping the full range of 127+127+64 is not necessary. Total bytes: 277

static const byte GetACos[] PROGMEM = {
        255,254,252,251,250,249,247,246,245,243,242,241,240,238,237,236,234,233,232,231,229,228,227,225,
        224,223,221,220,219,217,216,215,214,212,211,210,208,207,206,204,203,201,200,199,197,196,195,193,
        192,190,189,188,186,185,183,182,181,179,178,176,175,173,172,170,169,167,166,164,163,161,160,158,
        157,155,154,152,150,149,147,146,144,142,141,139,137,135,134,132,130,128,127,125,123,121,119,117,
        115,113,111,109,107,105,103,101,98,96,94,92,89,87,84,81,79,76,73,73,73,72,72,72,71,71,71,70,70,
        70,70,69,69,69,68,68,68,67,67,67,66,66,66,65,65,65,64,64,64,63,63,63,62,62,62,61,61,61,60,60,59,
        59,59,58,58,58,57,57,57,56,56,55,55,55,54,54,53,53,53,52,52,51,51,51,50,50,49,49,48,48,47,47,47,
        46,46,45,45,44,44,43,43,42,42,41,41,40,40,39,39,38,37,37,36,36,35,34,34,33,33,32,31,31,30,29,28,
        28,27,26,25,24,23,23,23,23,22,22,22,22,21,21,21,21,20,20,20,19,19,19,19,18,18,18,17,17,17,17,16,
        16,16,15,15,15,14,14,13,13,13,12,12,11,11,10,10,9,9,8,7,6,6,5,3,0 };//
```

```
//Sin table 90 deg, persision 0.5 deg [180 values]
static const word GetSin[] PROGMEM = {0, 87, 174, 261, 348, 436, 523, 610, 697, 784, 871, 958, 1045, 1132, 1218, 1305, 1391,
1478, 1564,
        1650, 1736, 1822, 1908, 1993, 2079, 2164, 2249, 2334, 2419, 2503, 2588, 2672, 2756, 2840, 2923, 3007,
        3090, 3173, 3255, 3338, 3420, 3502, 3583, 3665, 3746, 3826, 3907, 3987, 4067, 4146, 4226, 4305, 4383,
        4461, 4539, 4617, 4694, 4771, 4848, 4924, 4999, 5075, 5150, 5224, 5299, 5372, 5446, 5519, 5591, 5664,
        5735, 5807, 5877, 5948, 6018, 6087, 6156, 6225, 6293, 6360, 6427, 6494, 6560, 6626, 6691, 6755, 6819,
        6883, 6946, 7009, 7071, 7132, 7193, 7253, 7313, 7372, 7431, 7489, 7547, 7604, 7660, 7716, 7771, 7826,
        7880, 7933, 7986, 8038, 8090, 8141, 8191, 8241, 8290, 8338, 8386, 8433, 8480, 8526, 8571, 8616, 8660,
        8703, 8746, 8788, 8829, 8870, 8910, 8949, 8987, 9025, 9063, 9099, 9135, 9170, 9205, 9238, 9271, 9304,
        9335, 9366, 9396, 9426, 9455, 9483, 9510, 9537, 9563, 9588, 9612, 9636, 9659, 9681, 9702, 9723, 9743,
        9762, 9781, 9799, 9816, 9832, 9848, 9862, 9876, 9890, 9902, 9914, 9925, 9935, 9945, 9953, 9961, 9969,
        9975, 9981, 9986, 9990, 9993, 9996, 9998, 9999, 10000 };//


//Build tables for Leg configuration like I/O and MIN/imax values to easy access values using a FOR loop
//Constants are still defined as single values in the cfg file to make it easy to read/configure


// Servo Horn offsets
#ifdef cRRFemurHornOffset1   // per leg configuration
static const short cFemurHornOffset1[] PROGMEM = {cRRFemurHornOffset1,  cRMFemurHornOffset1,  cRFFemurHornOffset1,
cLRFemurHornOffset1,  cLMFemurHornOffset1,  cLFFemurHornOffset1};
#define CFEMURHORNOFFSET1(LEGI) ((short)pgm_read_word(&cFemurHornOffset1[LEGI]))
#else   // Fixed per leg, if not defined 0
#ifndef cFemurHornOffset1
#define cFemurHornOffset1  0
#endif
#define CFEMURHORNOFFSET1(LEGI)  (cFemurHornOffset1)
#endif

#ifdef c4DOF
#ifdef cRRTarsHornOffset1   // per leg configuration
static const short cTarsHornOffset1[] PROGMEM = {cRRTarsHornOffset1,  cRMTarsHornOffset1,  cRFTarsHornOffset1,
cLRTarsHornOffset1,  cLMTarsHornOffset1,  cLFTarsHornOffset1};
#define CTARSHORNOFFSET1(LEGI) ((short)pgm_read_word(&cTarsHornOffset1[LEGI]))
#else   // Fixed per leg, if not defined 0
#ifndef cTarsHornOffset1
#define cTarsHornOffset1  0
#endif
#define CTARSHORNOFFSET1(LEGI)  cTarsHornOffset1
#endif
#endif

//Min / imax values
const short cCoxaMin1[] PROGMEM = {cRRCoxaMin1,  cRMCoxaMin1,  cRFCoxaMin1,  cLRCoxaMin1,  cLMCoxaMin1,
cLFCoxaMin1};
const short cCoxaMax1[] PROGMEM = {cRRCoxaMax1,  cRMCoxaMax1,  cRFCoxaMax1,  cLRCoxaMax1,  cLMCoxaMax1,
cLFCoxaMax1};
const short cFemurMin1[] PROGMEM ={cRRFemurMin1, cRMFemurMin1, cRFFemurMin1, cLRFemurMin1, cLMFemurMin1,
cLFFemurMin1};
const short cFemurMax1[] PROGMEM ={cRRFemurMax1, cRMFemurMax1, cRFFemurMax1, cLRFemurMax1, cLMFemurMax1,
cLFFemurMax1};
const short cTibiaMin1[] PROGMEM ={cRRTibiaMin1, cRMTibiaMin1, cRFTibiaMin1, cLRTibiaMin1, cLMTibiaMin1,
cLFTibiaMin1};
```

```
const short cTibiaMax1[] PROGMEM = {cRRTibiaMax1, cRMTibiaMax1, cRFTibiaMax1, cLRTibiaMax1, cLMTibiaMax1,
cLFTibiaMax1};

#ifdef c4DOF
const short cTarsMin1[] PROGMEM = {cRRTarsMin1, cRMTarsMin1, cRFTarsMin1, cLRTarsMin1, cLMTarsMin1, cLFTarsMin1};
const short cTarsMax1[] PROGMEM = {cRRTarsMax1, cRMTarsMax1, cRFTarsMax1, cLRTarsMax1, cLMTarsMax1, cLFTarsMax1};
#endif


//Leg Lengths
const byte cCoxaLength[] PROGMEM = {cRRCoxaLength,  cRMCoxaLength,  cRFCoxaLength,  cLRCoxaLength,  cLMCoxaLength,
cLFCoxaLength};
const byte cFemurLength[] PROGMEM = {cRRFemurLength, cRMFemurLength, cRFFemurLength, cLRFemurLength,
cLMFemurLength, cLFFemurLength};
const byte cTibiaLength[] PROGMEM = {cRRTibiaLength, cRMTibiaLength, cRFTibiaLength, cLRTibiaLength, cLMTibiaLength,
cLFTibiaLength};
#ifdef c4DOF
const byte cTarsLength[] PROGMEM = {cRRTarsLength, cRMTarsLength, cRFTarsLength, cLRTarsLength, cLMTarsLength,
cLFTarsLength};
#endif


//Body Offsets [distance between the center of the body and the center of the coxa]
const short cOffsetX[] PROGMEM = {cRROffsetX, cRMOffsetX, cRFOffsetX, cLROffsetX, cLMOffsetX, cLFOffsetX};
const short cOffsetZ[] PROGMEM = {cRROffsetZ, cRMOffsetZ, cRFOffsetZ, cLROffsetZ, cLMOffsetZ, cLFOffsetZ};

//Default leg angle
const short cCoxaAngle1[] PROGMEM = {cRRCoxaAngle1, cRMCoxaAngle1, cRFCoxaAngle1, cLRCoxaAngle1, cLMCoxaAngle1,
cLFCoxaAngle1};

//Start positions for the leg
const short cInitPosX[] PROGMEM = {cRRInitPosX, cRMInitPosX, cRFInitPosX, cLRInitPosX, cLMInitPosX, cLFInitPosX};
const short cInitPosY[] PROGMEM = {cRRInitPosY, cRMInitPosY, cRFInitPosY, cLRInitPosY, cLMInitPosY, cLFInitPosY};
const short cInitPosZ[] PROGMEM = {cRRInitPosZ, cRMInitPosZ, cRFInitPosZ, cLRInitPosZ, cLMInitPosZ, cLFInitPosZ};


// Define some globals for debug information
boolean g_fShowDebugPrompt;
boolean g_fDebugOutput = true;


//--------------------------------------------------------------------
//[REMOTE]
#define cTravelDeadZone        4    //The deadzone for the analog input from the remote
//====================================================================
//[ANGLES]
short         CoxaAngle1[6];    //Actual Angle of the horizontal hip, decimals = 1
short         FemurAngle1[6];   //Actual Angle of the vertical hip, decimals = 1
short         TibiaAngle1[6];   //Actual Angle of the knee, decimals = 1
#ifdef c4DOF
short         TarsAngle1[6];     //Actual Angle of the knee, decimals = 1
#endif

//--------------------------------------------------------------------
//[POSITIONS SINGLE LEG CONTROL]
```

```
short       LegPosX[6];   //Actual X Posion of the Leg
short       LegPosY[6];   //Actual Y Posion of the Leg
short       LegPosZ[6];   //Actual Z Posion of the Leg
//--------------------------------------------------------------------
//[INPUTS]

//--------------------------------------------------------------------
//[GP PLAYER]
//--------------------------------------------------------------------
//[OUTPUTS]
boolean     LedA;   //Red
boolean     LedB;   //Green
boolean     LedC;   //Orange
boolean     Eyes;   //Eyes output
//--------------------------------------------------------------------
//[VARIABLES]
byte        Index;              //Index universal used
byte        LegIndex;             //Index used for leg Index Number

//GetSinCos / ArcCos
short       AngleDeg1;      //Input Angle in degrees, decimals = 1
short       sin4;         //Output Sinus of the given Angle, decimals = 4
short       cos4;         //Output Cosinus of the given Angle, decimals = 4
short       AngleRad4;      //Output Angle in radials, decimals = 4

//GetAtan2
short       AtanX;         //Input X
short       AtanY;         //Input Y
short       Atan4;         //ArcTan2 output
short       XYhyp2;         //Output presenting Hypotenuse of X and Y

//Body Inverse Kinematics
short       PosX;         //Input position of the feet X
short       PosZ;         //Input position of the feet Z
short       PosY;         //Input position of the feet Y
//long       TotalX;         //Total X distance between the center of the body and the feet
//long       TotalZ;         //Total Z distance between the center of the body and the feet
long        BodyFKPosX;      //Output Position X of feet with Rotation
long        BodyFKPosY;      //Output Position Y of feet with Rotation
long        BodyFKPosZ;      //Output Position Z of feet with Rotation
// New with zentas stuff
short       BodyRotOffsetX;   //Input X offset value to adjust centerpoint of rotation
short       BodyRotOffsetY;   //Input Y offset value to adjust centerpoint of rotation
short       BodyRotOffsetZ;   //Input Z offset value to adjust centerpoint of rotation


//Leg Inverse Kinematics
long        IKFeetPosX;      //Input position of the Feet X
long        IKFeetPosY;      //Input position of the Feet Y
long        IKFeetPosZ;      //Input Position of the Feet Z
boolean     IKSolution;      //Output true if the solution is possible
boolean     IKSolutionWarning;   //Output true if the solution is NEARLY possible
boolean     IKSolutionError;   //Output true if the solution is NOT possible
//--------------------------------------------------------------------
//[TIMING]
unsigned long  lTimerStart;   //Start time of the calculation cycles
```

```
unsigned long   lTimerEnd;       //End time of the calculation cycles
byte            CycleTime;       //Total Cycle time

word            ServoMoveTime;       //Time for servo updates
word            PrevServoMoveTime;    //Previous time for the servo updates

//--------------------------------------------------------------------
//[GLOABAL]
//--------------------------------------------------------------------

// Define our global Input Control State object
INCONTROLSTATE   g_InControlState;      // This is our global Input control state object...

// Define our ServoWriter class
ServoDriver  g_ServoDriver;      // our global servo driver class

boolean        g_fLowVoltageShutdown;    // If set the bot shuts down because the input voltage is to low
word           Voltage;


//--boolean        g_InControlState.fHexOn;          //Switch to turn on Phoenix
//--boolean        g_InControlState.fPrev_HexOn;       //Previous loop state
//--------------------------------------------------------------------
//[Balance]
long          TotalTransX;
long          TotalTransZ;
long          TotalTransY;
long          TotalYBal1;
long          TotalXBal1;
long          TotalZBal1;

//[Single Leg Control]
byte          PrevSelectedLeg;
boolean        AllDown;

//[gait]

short        NomGaitSpeed;       //Nominal speed of the gait
short         TLDivFactor;        //Number of steps that a leg is on the floor while walking
short         NrLiftedPos;        //Number of positions that a single leg is lifted [1-3]
byte          LiftDivFactor;      //Normaly: 2, when NrLiftedPos=5: 4

boolean        HalfLiftHeigth;      //If TRUE the outer positions of the ligted legs will be half height

boolean        TravelRequest;       //Temp to check if the gait is in motion
byte          StepsInGait;        //Number of steps in gait

boolean        LastLeg;            //TRUE when the current leg is the last leg of the sequence
byte          GaitStep;          //Actual Gait step

byte          GaitLegNr[6];       //Init position of the leg

byte          GaitLegNrIn;        //Input Number of the leg

long          GaitPosX[6];        //Array containing Relative X position corresponding to the Gait
long          GaitPosY[6];        //Array containing Relative Y position corresponding to the Gait
```

```
long        GaitPosZ[6];       //Array containing Relative Z position corresponding to the Gait
long        GaitRotY[6];       //Array containing Relative Y rotation corresponding to the Gait


boolean       fWalking;          //  True if the robot are walking
boolean       fContinueWalking;    // should we continue to walk?



//==============================================================================
// Function prototypes
//==============================================================================
extern void GaitSelect(void);
extern void  WriteOutputs(void);
extern void SingleLegControl(void);
extern void GaitSeq(void);
extern void BalanceBody(void);
extern void CheckAngles();

extern void    PrintSystemStuff(void);          // Try to see why we fault...


extern void BalCalcOneLeg (short PosX, short PosZ, short PosY, byte BalLegNr);
extern void BodyFK (short PosX, short PosZ, short PosY, short RotationY, byte BodyIKLeg) ;
extern void LegIK (short IKFeetPosX, short IKFeetPosY, short IKFeetPosZ, byte LegIKLegNr);
extern void Gait (byte GaitCurrentLegNr);
extern short GetATan2 (short AtanX, short AtanY);


//--------------------------------------------------------------------------
// SETUP: the main arduino setup function.
//--------------------------------------------------------------------------
void setup(){

  int error;

  g_fShowDebugPrompt = true;
  g_fDebugOutput = false;
#ifdef DBGSerial
  DBGSerial.begin(57600);
#endif
  // Init our ServoDriver
  g_ServoDriver.Init();

  pinMode(PS2_CMD, INPUT);
  if(!digitalRead(PS2_CMD))
    g_ServoDriver.SSCForwarder();

  //Checks to see if our Servo Driver support a GP Player
//    DBGSerial.write("Program Start\n\r");
  // debug stuff
  delay(10);


  //Turning off all the leds
  LedA = 0;
```

```
  LedB = 0;
  LedC = 0;
  Eyes = 0;

  //Tars Init Positions
  for (LegIndex= 0; LegIndex <= 5; LegIndex++ )
  {
    LegPosX[LegIndex] = (short)pgm_read_word(&cInitPosX[LegIndex]);    //Set start positions for each leg
    LegPosY[LegIndex] = (short)pgm_read_word(&cInitPosY[LegIndex]);
    LegPosZ[LegIndex] = (short)pgm_read_word(&cInitPosZ[LegIndex]);
  }

  //Single leg control. Make sure no leg is selected
  g_InControlState.SelectedLeg = 255; // No Leg selected
  PrevSelectedLeg = 255;

  //Body Positions
  g_InControlState.BodyPos.x = 0;
  g_InControlState.BodyPos.y = 0;
  g_InControlState.BodyPos.z = 0;

  //Body Rotations
  g_InControlState.BodyRot1.x = 0;
  g_InControlState.BodyRot1.y = 0;
  g_InControlState.BodyRot1.z = 0;
  BodyRotOffsetX = 0;
  BodyRotOffsetY = 0;        //Input Y offset value to adjust centerpoint of rotation
  BodyRotOffsetZ = 0;


  //Gait
  g_InControlState.GaitType = 1;  // 0; Devon wanted
  g_InControlState.BalanceMode = 0;
  g_InControlState.LegLiftHeight = 50;
  GaitStep = 1;
  GaitSelect();

  g_InputController.Init();

  // Servo Driver
  ServoMoveTime = 150;
  g_InControlState.fHexOn = 0;
  g_fLowVoltageShutdown = false;
}

//==============================================================================
// Loop: the main arduino main Loop function
//==============================================================================


void loop(void)
{
  //Start time
  lTimerStart = millis();
  //Read input
```

```
  CheckVoltage();      // check our voltages...
  if (!g_fLowVoltageShutdown)
    g_InputController.ControlInput();

  WriteOutputs();      // Write Outputs


#ifdef OPT_GPPLAYER
  //GP Player
  g_ServoDriver.GPPlayer();
#endif

  //Single leg control
  SingleLegControl ();

  //Gait
  GaitSeq();

  //Balance calculations
  TotalTransX = 0;    //reset values used for calculation of balance
  TotalTransZ = 0;
  TotalTransY = 0;
  TotalXBal1 = 0;
  TotalYBal1 = 0;
  TotalZBal1 = 0;
  if (g_InControlState.BalanceMode) {
    for (LegIndex = 0; LegIndex <= 2; LegIndex++) {   // balance calculations for all Right legs

      BalCalcOneLeg (-LegPosX[LegIndex]+GaitPosX[LegIndex],
            LegPosZ[LegIndex]+GaitPosZ[LegIndex],
            (LegPosY[LegIndex]-(short)pgm_read_word(&cInitPosY[LegIndex]))+GaitPosY[LegIndex], LegIndex);
    }

    for (LegIndex = 3; LegIndex <= 5; LegIndex++) {   // balance calculations for all Right legs
      BalCalcOneLeg(LegPosX[LegIndex]+GaitPosX[LegIndex],
            LegPosZ[LegIndex]+GaitPosZ[LegIndex],
            (LegPosY[LegIndex]-(short)pgm_read_word(&cInitPosY[LegIndex]))+GaitPosY[LegIndex], LegIndex);
    }
    BalanceBody();
  }


 //Reset IKsolution indicators
   IKSolution = 0 ;
   IKSolutionWarning = 0;
   IKSolutionError = 0 ;

   //Do IK for all Right legs
   for (LegIndex = 0; LegIndex <=2; LegIndex++) {
    BodyFK(-LegPosX[LegIndex]+g_InControlState.BodyPos.x+GaitPosX[LegIndex] - TotalTransX,
        LegPosZ[LegIndex]+g_InControlState.BodyPos.z+GaitPosZ[LegIndex] - TotalTransZ,
        LegPosY[LegIndex]+g_InControlState.BodyPos.y+GaitPosY[LegIndex] - TotalTransY,
        GaitRotY[LegIndex], LegIndex);

    LegIK (LegPosX[LegIndex]-g_InControlState.BodyPos.x+BodyFKPosX-(GaitPosX[LegIndex] - TotalTransX),
        LegPosY[LegIndex]+g_InControlState.BodyPos.y-BodyFKPosY+GaitPosY[LegIndex] - TotalTransY,
```

```
        LegPosZ[LegIndex]+g_InControlState.BodyPos.z-BodyFKPosZ+GaitPosZ[LegIndex] - TotalTransZ, LegIndex);
  }

  //Do IK for all Left legs
  for (LegIndex = 3; LegIndex <=5; LegIndex++) {
    BodyFK(LegPosX[LegIndex]-g_InControlState.BodyPos.x+GaitPosX[LegIndex] - TotalTransX,
        LegPosZ[LegIndex]+g_InControlState.BodyPos.z+GaitPosZ[LegIndex] - TotalTransZ,
        LegPosY[LegIndex]+g_InControlState.BodyPos.y+GaitPosY[LegIndex] - TotalTransY,
        GaitRotY[LegIndex], LegIndex);
    LegIK (LegPosX[LegIndex]+g_InControlState.BodyPos.x-BodyFKPosX+GaitPosX[LegIndex] - TotalTransX,
        LegPosY[LegIndex]+g_InControlState.BodyPos.y-BodyFKPosY+GaitPosY[LegIndex] - TotalTransY,
        LegPosZ[LegIndex]+g_InControlState.BodyPos.z-BodyFKPosZ+GaitPosZ[LegIndex] - TotalTransZ, LegIndex);
  }

  //Check mechanical limits
  CheckAngles();

  //Write IK errors to leds
  LedC = IKSolutionWarning;
  LedA = IKSolutionError;

  //Drive Servos
  if (g_InControlState.fHexOn) {
    if (g_InControlState.fHexOn && !g_InControlState.fPrev_HexOn) {
      MSound(SOUND_PIN, 3, 60, 2000, 80, 2250, 100, 2500);
#ifdef USEXBEE
      XBeePlaySounds(3, 60, 2000, 80, 2250, 100, 2500);
#endif

      Eyes = 1;
    }

    //Calculate Servo Move time
    if ((abs(g_InControlState.TravelLength.x)>cTravelDeadZone) || (abs(g_InControlState.TravelLength.z)>cTravelDeadZone)
||
        (abs(g_InControlState.TravelLength.y*2)>cTravelDeadZone)) {
      ServoMoveTime = NomGaitSpeed + (g_InControlState.InputTimeDelay*2) + g_InControlState.SpeedControl;

      //Add aditional delay when Balance mode is on
      if (g_InControlState.BalanceMode)
        ServoMoveTime = ServoMoveTime + 100;
    } else //Movement speed excl. Walking
      ServoMoveTime = 200 + g_InControlState.SpeedControl;

    // note we broke up the servo driver into start/commit that way we can output all of the servo information
    // before we wait and only have the termination information to output after the wait.  That way we hopefully
    // be more accurate with our timings...
    StartUpdateServos();

    // See if we need to sync our processor with the servo driver while walking to ensure the prev is completed before sending
the next one

    fContinueWalking = false;

    // Finding any incident of GaitPos/Rot <>0:
    for (LegIndex = 0; LegIndex <= 5; LegIndex++) {
```

```
        if ( (GaitPosX[LegIndex] > 2) || (GaitPosX[LegIndex] < -2)
             || (GaitPosY[LegIndex] > 2) || (GaitPosY[LegIndex] < -2)
             || (GaitPosZ[LegIndex] > 2) || (GaitPosZ[LegIndex] < -2)
             || (GaitRotY[LegIndex] > 2) || (GaitRotY[LegIndex] < -2) )   {
          fContinueWalking = true;
          break;
        }
      }
      if (fWalking || fContinueWalking) {
        word  wDelayTime;
        fWalking = fContinueWalking;

        //Get endtime and calculate wait time
        lTimerEnd = millis();
        if (lTimerEnd > lTimerStart)
          CycleTime = lTimerEnd-lTimerStart;
        else
          CycleTime = 0xffffffffL - lTimerEnd + lTimerStart + 1;

        // if it is less, use the last cycle time...
        //Wait for previous commands to be completed while walking
        wDelayTime = (min(max ((PrevServoMoveTime - CycleTime), 1), NomGaitSpeed));
        delay (wDelayTime);
      }

    } else {
      //Turn the bot off
      if (g_InControlState.fPrev_HexOn || (AllDown= 0)) {
        ServoMoveTime = 600;
        StartUpdateServos();
        g_ServoDriver.CommitServoDriver(ServoMoveTime);
        MSound(SOUND_PIN, 3, 100, 2500, 80, 2250, 60, 2000);
#ifdef USEXBEE
        XBeePlaySounds(3, 100, 2500, 80, 2250, 60, 2000);
#endif
        delay(600);
      } else {
        g_ServoDriver.FreeServos();
        Eyes = 0;
      }
        // We also have a simple debug monitor that allows us to
      // check things. call it here..
#ifdef OPT_TERMINAL_MONITOR
      if (TerminalMonitor())
        return;
#endif
      delay(20);  // give a pause between times we call if nothing is happening
    }

    // Xan said Needed to be here...
    g_ServoDriver.CommitServoDriver(ServoMoveTime);
    PrevServoMoveTime = ServoMoveTime;

    //Store previous g_InControlState.fHexOn State
    if (g_InControlState.fHexOn)
      g_InControlState.fPrev_HexOn = 1;
```

```
   else
      g_InControlState.fPrev_HexOn = 0;
}


void StartUpdateServos()
{
   byte    LegIndex;

   // First call off to the init...
   g_ServoDriver.BeginServoUpdate();    // Start the update

   for (LegIndex = 0; LegIndex <= 5; LegIndex++) {
#ifdef c4DOF
      g_ServoDriver.OutputServoInfoForLeg(LegIndex, CoxaAngle1[LegIndex], FemurAngle1[LegIndex], TibiaAngle1[LegIndex],
sTarsAngle1[LegIndex]);
#else
      g_ServoDriver.OutputServoInfoForLeg(LegIndex, CoxaAngle1[LegIndex], FemurAngle1[LegIndex], TibiaAngle1[LegIndex]);
#endif
   }
}




//--------------------------------------------------------------------
//[WriteOutputs] Updates the state of the leds
//--------------------------------------------------------------------
void WriteOutputs(void)
{
 #ifdef cEyesPin
   digitalWrite(cEyesPin, Eyes);
#endif
}
//--------------------------------------------------------------------
//[CHECK VOLTAGE]
//Reads the input voltage and shuts down the bot when the power drops
byte s_bLVBeepCnt;
bool CheckVoltage() {
#ifdef  cVoltagePin
#ifdef cTurnOffVol
   Voltage = analogRead(cVoltagePin); // Battery voltage
   Voltage = ((long)Voltage*1955)/1000;

   if (!g_fLowVoltageShutdown) {
      if ((Voltage < cTurnOffVol) || (Voltage >= 1999)) {
#ifdef DBGSerial
         DBGSerial.println("Voltage went low, turn off robot");
#endif
       //Turn off
      g_InControlState.BodyPos.x = 0;
      g_InControlState.BodyPos.y = 0;
      g_InControlState.BodyPos.z = 0;
      g_InControlState.BodyRot1.x = 0;
      g_InControlState.BodyRot1.y = 0;
      g_InControlState.BodyRot1.z = 0;
```

```
      g_InControlState.TravelLength.x = 0;
      g_InControlState.TravelLength.z = 0;
      g_InControlState.TravelLength.y = 0;
      g_InControlState.SelectedLeg = 255;
      g_fLowVoltageShutdown = 1;
        s_bLVBeepCnt = 0;    // how many times we beeped...
      g_InControlState.fHexOn = false;
    }
#ifdef cTurnOnVol
  } else if ((Voltage > cTurnOnVol) && (Voltage < 1999)) {
#ifdef DBGSerial
        DBGSerial.println("Voltage restored");
#endif
        g_fLowVoltageShutdown = 0;

#endif
  } else {
    if (s_bLVBeepCnt < 5) {
      s_bLVBeepCnt++;
      MSound(SOUND_PIN, 1, 45, 2000);
    }
    delay(2000);
  }
#endif
#endif
  return g_fLowVoltageShutdown;
}

//--------------------------------------------------------------------
//[SINGLE LEG CONTROL]
void SingleLegControl(void)
{

  //Check if all legs are down
    AllDown = (LegPosY[cRF]==(short)pgm_read_word(&cInitPosY[cRF])) &&
          (LegPosY[cRM]==(short)pgm_read_word(&cInitPosY[cRM])) &&
          (LegPosY[cRR]==(short)pgm_read_word(&cInitPosY[cRR])) &&
          (LegPosY[cLR]==(short)pgm_read_word(&cInitPosY[cLR])) &&
          (LegPosY[cLM]==(short)pgm_read_word(&cInitPosY[cLM])) &&
          (LegPosY[cLF]==(short)pgm_read_word(&cInitPosY[cLF]));

  if (g_InControlState.SelectedLeg<=5) {
    if (g_InControlState.SelectedLeg!=PrevSelectedLeg) {
      if (AllDown) { //Lift leg a bit when it got selected
        LegPosY[g_InControlState.SelectedLeg] = (short)pgm_read_word(&cInitPosY[g_InControlState.SelectedLeg])-20;

        //Store current status
         PrevSelectedLeg = g_InControlState.SelectedLeg;
      } else {//Return prev leg back to the init position
        LegPosX[PrevSelectedLeg] = (short)pgm_read_word(&cInitPosX[PrevSelectedLeg]);
        LegPosY[PrevSelectedLeg] = (short)pgm_read_word(&cInitPosY[PrevSelectedLeg]);
        LegPosZ[PrevSelectedLeg] = (short)pgm_read_word(&cInitPosZ[PrevSelectedLeg]);
      }
    } else if (!g_InControlState.fSLHold) {
      LegPosY[g_InControlState.SelectedLeg] = LegPosY[g_InControlState.SelectedLeg]+g_InControlState.SLLeg.y;
```

```
        LegPosX[g_InControlState.SelectedLeg] =
(short)pgm_read_word(&cInitPosX[g_InControlState.SelectedLeg])+g_InControlState.SLLeg.x;
        LegPosZ[g_InControlState.SelectedLeg] =
(short)pgm_read_word(&cInitPosZ[g_InControlState.SelectedLeg])+g_InControlState.SLLeg.z;
      }
    } else {//All legs to init position
      if (!AllDown) {
        for(LegIndex = 0; LegIndex <= 5;LegIndex++) {
          LegPosX[LegIndex] = (short)pgm_read_word(&cInitPosX[LegIndex]);
          LegPosY[LegIndex] = (short)pgm_read_word(&cInitPosY[LegIndex]);
          LegPosZ[LegIndex] = (short)pgm_read_word(&cInitPosZ[LegIndex]);
        }
      }
      if (PrevSelectedLeg!=255)
        PrevSelectedLeg = 255;
    }
}


//--------------------------------------------------------------------
void GaitSelect(void)
{
  //Gait selector
  switch (g_InControlState.GaitType)  {
    case 0:
      //Ripple Gait 12 steps
      GaitLegNr[cLR] = 1;
      GaitLegNr[cRF] = 3;
      GaitLegNr[cLM] = 5;
      GaitLegNr[cRR] = 7;
      GaitLegNr[cLF] = 9;
      GaitLegNr[cRM] = 11;

      NrLiftedPos = 3;
      HalfLiftHeigth = 3;
      TLDivFactor = 8;
      StepsInGait = 12;
      NomGaitSpeed = 70;
      break;
    case 1:
      //Tripod 8 steps
      GaitLegNr[cLR] = 5;
      GaitLegNr[cRF] = 1;
      GaitLegNr[cLM] = 1;
      GaitLegNr[cRR] = 1;
      GaitLegNr[cLF] = 5;
      GaitLegNr[cRM] = 5;

      NrLiftedPos = 3;
      HalfLiftHeigth = 3;
      TLDivFactor = 4;
      StepsInGait = 8;
      NomGaitSpeed = 70;
      break;
    case 2:
      //Triple Tripod 12 step
```

```
        GaitLegNr[cRF] = 3;
        GaitLegNr[cLM] = 4;
        GaitLegNr[cRR] = 5;
        GaitLegNr[cLF] = 9;
        GaitLegNr[cRM] = 10;
        GaitLegNr[cLR] = 11;

        NrLiftedPos = 3;
        HalfLiftHeigth = 3;
        TLDivFactor = 8;
        StepsInGait = 12;
        NomGaitSpeed = 60;
        break;
      case 3:
        // Triple Tripod 16 steps, use 5 lifted positions
        GaitLegNr[cRF] = 4;
        GaitLegNr[cLM] = 5;
        GaitLegNr[cRR] = 6;
        GaitLegNr[cLF] = 12;
        GaitLegNr[cRM] = 13;
        GaitLegNr[cLR] = 14;

        NrLiftedPos = 5;
        HalfLiftHeigth = 1;
        TLDivFactor = 10;
        StepsInGait = 16;
        NomGaitSpeed = 60;
        break;
      case 4:
        //Wave 24 steps
        GaitLegNr[cLR] = 1;
        GaitLegNr[cRF] = 21;
        GaitLegNr[cLM] = 5;

        GaitLegNr[cRR] = 13;
        GaitLegNr[cLF] = 9;
        GaitLegNr[cRM] = 17;

        NrLiftedPos = 3;
        HalfLiftHeigth = 3;
        TLDivFactor = 20;
        StepsInGait = 24;
        NomGaitSpeed = 70;
        break;
  }
}

//--------------------------------------------------------------------
//[GAIT Sequence]
void GaitSeq(void)
{
  //Check if the Gait is in motion
  TravelRequest = ((abs(g_InControlState.TravelLength.x)>cTravelDeadZone) ||
(abs(g_InControlState.TravelLength.z)>cTravelDeadZone) || (abs(g_InControlState.TravelLength.y)>cTravelDeadZone));
  if (NrLiftedPos == 5)
   LiftDivFactor = 4;
```

```
    else
     LiftDivFactor = 2;

   //Calculate Gait sequence
    LastLeg = 0;
    for (LegIndex = 0; LegIndex <= 5; LegIndex++) { // for all legs
       if (LegIndex == 5) // last leg
          LastLeg = 1 ;

       Gait(LegIndex);
    }   // next leg
}


//--------------------------------------------------------------------
//[GAIT]
void Gait (byte GaitCurrentLegNr)
{


    //Clear values under the cTravelDeadZone
    if (!TravelRequest) {
       g_InControlState.TravelLength.x=0;
       g_InControlState.TravelLength.z=0;
       g_InControlState.TravelLength.y=0;
    }
    //Leg middle up position
    //Gait in motion                                                    Gait NOT in motion, return to home position
    if ((TravelRequest && (NrLiftedPos==1 || NrLiftedPos==3 || NrLiftedPos==5) &&
          GaitStep==GaitLegNr[GaitCurrentLegNr]) || (!TravelRequest && GaitStep==GaitLegNr[GaitCurrentLegNr] &&
((abs(GaitPosX[GaitCurrentLegNr])>2) ||
            (abs(GaitPosZ[GaitCurrentLegNr])>2) || (abs(GaitRotY[GaitCurrentLegNr])>2)))) { //Up
       GaitPosX[GaitCurrentLegNr] = 0;
       GaitPosY[GaitCurrentLegNr] = -g_InControlState.LegLiftHeight;
       GaitPosZ[GaitCurrentLegNr] = 0;
       GaitRotY[GaitCurrentLegNr] = 0;
    }
    //Optional Half heigth Rear (2, 3, 5 lifted positions)
    else if (((NrLiftedPos==2 && GaitStep==GaitLegNr[GaitCurrentLegNr]) || (NrLiftedPos>=3 &&
          (GaitStep==GaitLegNr[GaitCurrentLegNr]-1 || GaitStep==GaitLegNr[GaitCurrentLegNr]+(StepsInGait-1))))
          && TravelRequest) {
       GaitPosX[GaitCurrentLegNr] = -g_InControlState.TravelLength.x/LiftDivFactor;
       GaitPosY[GaitCurrentLegNr] = -3*g_InControlState.LegLiftHeight/(3+HalfLiftHeigth);     //Easier to shift between div factor:
/1 (3/3), /2 (3/6) and 3/4
       GaitPosZ[GaitCurrentLegNr] = -g_InControlState.TravelLength.z/LiftDivFactor;
       GaitRotY[GaitCurrentLegNr] = -g_InControlState.TravelLength.y/LiftDivFactor;
    }

    // Optional Half heigth front (2, 3, 5 lifted positions)
    else if ((NrLiftedPos>=2) && (GaitStep==GaitLegNr[GaitCurrentLegNr]+1 || GaitStep==GaitLegNr[GaitCurrentLegNr]-
(StepsInGait-1)) && TravelRequest) {
       GaitPosX[GaitCurrentLegNr] = g_InControlState.TravelLength.x/LiftDivFactor;
       GaitPosY[GaitCurrentLegNr] = -3*g_InControlState.LegLiftHeight/(3+HalfLiftHeigth); // Easier to shift between div factor:
/1 (3/3), /2 (3/6) and 3/4
       GaitPosZ[GaitCurrentLegNr] = g_InControlState.TravelLength.z/LiftDivFactor;
       GaitRotY[GaitCurrentLegNr] = g_InControlState.TravelLength.y/LiftDivFactor;
```

```
   }

   //Optional Half heigth Rear 5 LiftedPos (5 lifted positions)
   else if (((NrLiftedPos==5 && (GaitStep==GaitLegNr[GaitCurrentLegNr]-2 ))) && TravelRequest) {
    GaitPosX[GaitCurrentLegNr] = -g_InControlState.TravelLength.x/2;
      GaitPosY[GaitCurrentLegNr] = -g_InControlState.LegLiftHeight/2;
      GaitPosZ[GaitCurrentLegNr] = -g_InControlState.TravelLength.z/2;
      GaitRotY[GaitCurrentLegNr] = -g_InControlState.TravelLength.y/2;
    }

   //Optional Half heigth Front 5 LiftedPos (5 lifted positions)
   else if ((NrLiftedPos==5) && (GaitStep==GaitLegNr[GaitCurrentLegNr]+2 || GaitStep==GaitLegNr[GaitCurrentLegNr]-
(StepsInGait-2)) && TravelRequest) {
      GaitPosX[GaitCurrentLegNr] = g_InControlState.TravelLength.x/2;
      GaitPosY[GaitCurrentLegNr] = -g_InControlState.LegLiftHeight/2;
      GaitPosZ[GaitCurrentLegNr] = g_InControlState.TravelLength.z/2;
      GaitRotY[GaitCurrentLegNr] = g_InControlState.TravelLength.y/2;
   }

  //Leg front down position
  else if ((GaitStep==GaitLegNr[GaitCurrentLegNr]+NrLiftedPos || GaitStep==GaitLegNr[GaitCurrentLegNr]-(StepsInGait-
NrLiftedPos))
        && GaitPosY[GaitCurrentLegNr]<0) {
      GaitPosX[GaitCurrentLegNr] = g_InControlState.TravelLength.x/2;
      GaitPosZ[GaitCurrentLegNr] = g_InControlState.TravelLength.z/2;
      GaitRotY[GaitCurrentLegNr] = g_InControlState.TravelLength.y/2;
      GaitPosY[GaitCurrentLegNr] = 0; //Only move leg down at once if terrain adaption is turned off
   }

   //Move body forward
   else {
      GaitPosX[GaitCurrentLegNr] = GaitPosX[GaitCurrentLegNr] - (g_InControlState.TravelLength.x/TLDivFactor);
      GaitPosY[GaitCurrentLegNr] = 0;
      GaitPosZ[GaitCurrentLegNr] = GaitPosZ[GaitCurrentLegNr] - (g_InControlState.TravelLength.z/TLDivFactor);
      GaitRotY[GaitCurrentLegNr] = GaitRotY[GaitCurrentLegNr] - (g_InControlState.TravelLength.y/TLDivFactor);
   }


   //Advance to the next step
   if (LastLeg)  {  //The last leg in this step
      GaitStep = GaitStep+1;
      if (GaitStep>StepsInGait)
           GaitStep = 1;
   }
}


//--------------------------------------------------------------------
//[BalCalcOneLeg]
void BalCalcOneLeg (short PosX, short PosZ, short PosY, byte BalLegNr)
{
   short        CPR_X;        //Final X value for centerpoint of rotation
   short        CPR_Y;        //Final Y value for centerpoint of rotation
   short        CPR_Z;        //Final Z value for centerpoint of rotation
   long         lAtan;
```

```
    //Calculating totals from center of the body to the feet
    CPR_Z = (short)pgm_read_word(&cOffsetZ[BalLegNr]) + PosZ;
    CPR_X = (short)pgm_read_word(&cOffsetX[BalLegNr]) + PosX;
    CPR_Y = 150 + PosY;      // using the value 150 to lower the centerpoint of rotation 'g_InControlState.BodyPos.y +

    TotalTransY += (long)PosY;
    TotalTransZ += (long)CPR_Z;
    TotalTransX += (long)CPR_X;

    lAtan = GetATan2(CPR_X, CPR_Z);
    TotalYBal1 += (lAtan*1800) / 31415;

    lAtan = GetATan2 (CPR_X, CPR_Y);
    TotalZBal1 += ((lAtan*1800) / 31415) -900; //Rotate balance circle 90 deg

    lAtan = GetATan2 (CPR_Z, CPR_Y);
    TotalXBal1 += ((lAtan*1800) / 31415) - 900; //Rotate balance circle 90 deg

}
//--------------------------------------------------------------------
//[BalanceBody]
void BalanceBody(void)
{
    TotalTransZ = TotalTransZ/BalanceDivFactor ;
    TotalTransX = TotalTransX/BalanceDivFactor;
    TotalTransY = TotalTransY/BalanceDivFactor;

    if (TotalYBal1 > 0)      //Rotate balance circle by +/- 180 deg
        TotalYBal1 -=  1800;
    else
        TotalYBal1 += 1800;

    if (TotalZBal1 < -1800)   //Compensate for extreme balance positions that causes owerflow
        TotalZBal1 += 3600;

    if (TotalXBal1 < -1800)   //Compensate for extreme balance positions that causes owerflow
        TotalXBal1 += 3600;

    //Balance rotation
    TotalYBal1 = -TotalYBal1/BalanceDivFactor;
    TotalXBal1 = -TotalXBal1/BalanceDivFactor;
    TotalZBal1 = TotalZBal1/BalanceDivFactor;
}


//--------------------------------------------------------------------
//[GETSINCOS] Get the sinus and cosinus from the angle +/- multiple circles
//AngleDeg1     - Input Angle in degrees
//sin4       - Output Sinus of AngleDeg
//cos4         - Output Cosinus of AngleDeg
void GetSinCos(short AngleDeg1)
{
    short       ABSAngleDeg1;   //Absolute value of the Angle in Degrees, decimals = 1
    //Get the absolute value of AngleDeg
    if (AngleDeg1 < 0)
        ABSAngleDeg1 = AngleDeg1 *-1;
```

```
    else
        ABSAngleDeg1 = AngleDeg1;

    //Shift rotation to a full circle of 360 deg -> AngleDeg // 360
    if (AngleDeg1 < 0)    //Negative values
        AngleDeg1 = 3600-(ABSAngleDeg1-(3600*(ABSAngleDeg1/3600)));
    else              //Positive values
        AngleDeg1 = ABSAngleDeg1-(3600*(ABSAngleDeg1/3600));

    if (AngleDeg1>=0 && AngleDeg1<=900)     // 0 to 90 deg
    {
        sin4 = pgm_read_word(&GetSin[AngleDeg1/5]);            // 5 is the presision (0.5) of the table
        cos4 = pgm_read_word(&GetSin[(900-(AngleDeg1))/5]);
    }

    else if (AngleDeg1>900 && AngleDeg1<=1800)     // 90 to 180 deg
    {
        sin4 = pgm_read_word(&GetSin[(900-(AngleDeg1-900))/5]); // 5 is the presision (0.5) of the table
        cos4 = -pgm_read_word(&GetSin[(AngleDeg1-900)/5]);
    }
    else if (AngleDeg1>1800 && AngleDeg1<=2700) // 180 to 270 deg
    {
        sin4 = -pgm_read_word(&GetSin[(AngleDeg1-1800)/5]);     // 5 is the presision (0.5) of the table
        cos4 = -pgm_read_word(&GetSin[(2700-AngleDeg1)/5]);
    }

    else if(AngleDeg1>2700 && AngleDeg1<=3600) // 270 to 360 deg
    {
        sin4 = -pgm_read_word(&GetSin[(3600-AngleDeg1)/5]); // 5 is the presision (0.5) of the table
        cos4 = pgm_read_word(&GetSin[(AngleDeg1-2700)/5]);
    }
}


//--------------------------------------------------------------------
//(GETARCCOS) Get the sinus and cosinus from the angle +/- multiple circles
//cos4        - Input Cosinus
//AngleRad4     - Output Angle in AngleRad4
long GetArcCos(short cos4)
{
    boolean NegativeValue/*:1*/;    //If the the value is Negative
    //Check for negative value
    if (cos4<0)
    {
        cos4 = -cos4;
        NegativeValue = 1;
    }
    else
        NegativeValue = 0;

    //Limit cos4 to his maximal value
    cos4 = min(cos4,c4DEC);

    if ((cos4>=0) && (cos4<9000))
    {
        AngleRad4 = (byte)pgm_read_byte(&GetACos[cos4/79]);
```

```
      AngleRad4 = ((long)AngleRad4*616)/c1DEC;          //616=acos resolution (pi/2/255) ;
   }
   else if ((cos4>=9000) && (cos4<9900))
   {
      AngleRad4 = (byte)pgm_read_byte(&GetACos[(cos4-9000)/8+114]);
      AngleRad4 = (long)((long)AngleRad4*616)/c1DEC;          //616=acos resolution (pi/2/255)
   }
   else if ((cos4>=9900) && (cos4<=10000))
   {
      AngleRad4 = (byte)pgm_read_byte(&GetACos[(cos4-9900)/2+227]);
      AngleRad4 = (long)((long)AngleRad4*616)/c1DEC;          //616=acos resolution (pi/2/255)
   }

   //Add negative sign
   if (NegativeValue)
      AngleRad4 = 31416 - AngleRad4;

   return AngleRad4;
}

unsigned long isqrt32 (unsigned long n) //
{
      unsigned long root;
      unsigned long remainder;
      unsigned long  place;

      root = 0;
      remainder = n;
      place = 0x40000000; // OR place = 0x4000; OR place = 0x40; - respectively

      while (place > remainder)
      place = place >> 2;
      while (place)
      {
            if (remainder >= root + place)
            {
                  remainder = remainder - root - place;
                  root = root + (place << 1);
            }
            root = root >> 1;
            place = place >> 2;
      }
      return root;
}

//-------------------------------------------------------------------
//(GETATAN2) Simplyfied ArcTan2 function based on fixed point ArcCos
//ArcTanX        - Input X
//ArcTanY        - Input Y
//ArcTan4         - Output ARCTAN2(X/Y)
//XYhyp2          - Output presenting Hypotenuse of X and Y
short GetATan2 (short AtanX, short AtanY)
{
   XYhyp2 = isqrt32(((long)AtanX*AtanX*c4DEC) + ((long)AtanY*AtanY*c4DEC));
   GetArcCos (((long)AtanX*(long)c6DEC) /(long) XYhyp2);
```

```
  if (AtanY < 0)            // removed overhead... Atan4 = AngleRad4 * (AtanY/abs(AtanY));
    Atan4 = -AngleRad4;
  else
    Atan4 = AngleRad4;
  return Atan4;
}

//------------------------------------------------------------------
//(BODY INVERSE KINEMATICS)
//BodyRotX       - Global Input pitch of the body
//BodyRotY       - Global Input rotation of the body
//BodyRotZ       - Global Input roll of the body
//RotationY      - Input Rotation for the gait
//PosX           - Input position of the feet X
//PosZ           - Input position of the feet Z
//SinB           - Sin buffer for BodyRotX
//CosB           - Cos buffer for BodyRotX
//SinG           - Sin buffer for BodyRotZ
//CosG           - Cos buffer for BodyRotZ
//BodyFKPosX     - Output Position X of feet with Rotation
//BodyFKPosY     - Output Position Y of feet with Rotation
//BodyFKPosZ     - Output Position Z of feet with Rotation
void BodyFK (short PosX, short PosZ, short PosY, short RotationY, byte BodyIKLeg)
{
  short     SinA4;     //Sin buffer for BodyRotX calculations
  short     CosA4;     //Cos buffer for BodyRotX calculations
  short     SinB4;     //Sin buffer for BodyRotX calculations
  short     CosB4;     //Cos buffer for BodyRotX calculations
  short     SinG4;     //Sin buffer for BodyRotZ calculations
  short     CosG4;     //Cos buffer for BodyRotZ calculations
  short     CPR_X;     //Final X value for centerpoint of rotation
  short     CPR_Y;     //Final Y value for centerpoint of rotation
  short     CPR_Z;     //Final Z value for centerpoint of rotation

  //Calculating totals from center of the body to the feet
  CPR_X = (short)pgm_read_word(&cOffsetX[BodyIKLeg])+PosX + BodyRotOffsetX;
  CPR_Y = PosY + BodyRotOffsetY;     //Define centerpoint for rotation along the Y-axis
  CPR_Z = (short)pgm_read_word(&cOffsetZ[BodyIKLeg]) + PosZ + BodyRotOffsetZ;

  //Successive global rotation matrix:
  //Math shorts for rotation: Alfa [A] = Xrotate, Beta [B] = Zrotate, Gamma [G] = Yrotate
  //Sinus Alfa = SinA, cosinus Alfa = cosA. and so on...

  //First calculate sinus and cosinus for each rotation:
  GetSinCos (g_InControlState.BodyRot1.x+TotalXBal1);
  SinG4 = sin4;
  CosG4 = cos4;

  GetSinCos (g_InControlState.BodyRot1.z+TotalZBal1);
  SinB4 = sin4;
  CosB4 = cos4;

  GetSinCos (g_InControlState.BodyRot1.y+(RotationY*c1DEC)+TotalYBal1) ;
  SinA4 = sin4;
  CosA4 = cos4;
```

```
   //Calcualtion of rotation matrix:
     BodyFKPosX = ((long)CPR_X*c2DEC - ((long)CPR_X*c2DEC*CosA4/c4DEC*CosB4/c4DEC -
(long)CPR_Z*c2DEC*CosB4/c4DEC*SinA4/c4DEC
          + (long)CPR_Y*c2DEC*SinB4/c4DEC ))/c2DEC;
     BodyFKPosZ = ((long)CPR_Z*c2DEC - ( (long)CPR_X*c2DEC*CosG4/c4DEC*SinA4/c4DEC +
(long)CPR_X*c2DEC*CosA4/c4DEC*SinB4/c4DEC*SinG4/c4DEC
          + (long)CPR_Z*c2DEC*CosA4/c4DEC*CosG4/c4DEC - (long)CPR_Z*c2DEC*SinA4/c4DEC*SinB4/c4DEC*SinG4/c4DEC
          - (long)CPR_Y*c2DEC*CosB4/c4DEC*SinG4/c4DEC ))/c2DEC;
     BodyFKPosY = ((long)CPR_Y  *c2DEC - ( (long)CPR_X*c2DEC*SinA4/c4DEC*SinG4/c4DEC -
(long)CPR_X*c2DEC*CosA4/c4DEC*CosG4/c4DEC*SinB4/c4DEC
          + (long)CPR_Z*c2DEC*CosA4/c4DEC*SinG4/c4DEC + (long)CPR_Z*c2DEC*CosG4/c4DEC*SinA4/c4DEC*SinB4/c4DEC
          + (long)CPR_Y*c2DEC*CosB4/c4DEC*CosG4/c4DEC ))/c2DEC;
}



//--------------------------------------------------------------------
//[LEG INVERSE KINEMATICS] Calculates the angles of the coxa, femur and tibia for the given position of the feet
//IKFeetPosX          - Input position of the Feet X
//IKFeetPosY          - Input position of the Feet Y
//IKFeetPosZ          - Input Position of the Feet Z
//IKSolution          - Output true if the solution is possible
//IKSolutionWarning     - Output true if the solution is NEARLY possible
//IKSolutionError    - Output true if the solution is NOT possible
//FemurAngle1          - Output Angle of Femur in degrees
//TibiaAngle1          - Output Angle of Tibia in degrees
//CoxaAngle1           - Output Angle of Coxa in degrees
//--------------------------------------------------------------------
void LegIK (short IKFeetPosX, short IKFeetPosY, short IKFeetPosZ, byte LegIKLegNr)
{
   unsigned long    IKSW2;           //Length between Shoulder and Wrist, decimals = 2
   unsigned long    IKA14;           //Angle of the line S>W with respect to the ground in radians, decimals = 4
   unsigned long    IKA24;           //Angle of the line S>W with respect to the femur in radians, decimals = 4
   short           IKFeetPosXZ;    //Diagonal direction from Input X and Z
#ifdef c4DOF
// these were shorts...
   long          TarsOffsetXZ;    //Vector value \ ;
   long          TarsOffsetY;     //Vector value / The 2 DOF IK calcs (femur and tibia) are based upon these vectors
   long          TarsToGroundAngle1;    //Angle between tars and ground. Note: the angle are 0 when the tars are perpendicular
to the ground
   long          TGA_A_H4;
   long          TGA_B_H3;
#else
#define TarsOffsetXZ 0        // Vector value
#define TarsOffsetY  0     //Vector value / The 2 DOF IK calcs (femur and tibia) are based upon these vectors
#endif


   long          Temp1;
   long          Temp2;
   long          T3;

   //Calculate IKCoxaAngle and IKFeetPosXZ
   GetATan2 (IKFeetPosX, IKFeetPosZ);
   CoxaAngle1[LegIKLegNr] = (((long)Atan4*180) / 3141) + (short)pgm_read_word(&cCoxaAngle1[LegIKLegNr]);
```

```
   //Length between the Coxa and tars [foot]
   IKFeetPosXZ = XYhyp2/c2DEC;
#ifdef c4DOF
   // Some legs may have the 4th DOF and some may not, so handle this here...
   //Calc the TarsToGroundAngle1:
   if ((byte)pgm_read_byte(&cTarsLength[LegIKLegNr])) {    // We allow mix of 3 and 4 DOF legs...
      TarsToGroundAngle1 = -cTarsConst + cTarsMulti*IKFeetPosY + ((long)(IKFeetPosXZ*cTarsFactorA))/c1DEC - 
((long)(IKFeetPosXZ*IKFeetPosY)/(cTarsFactorB));
      if (IKFeetPosY < 0)     //Always compensate TarsToGroundAngle1 when IKFeetPosY it goes below zero
         TarsToGroundAngle1 = TarsToGroundAngle1 - ((long)(IKFeetPosY*cTarsFactorC)/c1DEC);     //TGA base, overall rule
      if (TarsToGroundAngle1 > 400)
         TGA_B_H3 = 200 + (TarsToGroundAngle1/2);
      else
         TGA_B_H3 = TarsToGroundAngle1;

      if (TarsToGroundAngle1 > 300)
         TGA_A_H4 = 240 + (TarsToGroundAngle1/5);
      else
         TGA_A_H4 = TarsToGroundAngle1;

      if (IKFeetPosY > 0)    //Only compensate the TarsToGroundAngle1 when it exceed 30 deg (A, H4 PEP note)
         TarsToGroundAngle1 = TGA_A_H4;
      else if (((IKFeetPosY <= 0) & (IKFeetPosY > -10))) // linear transition between case H3 and H4 (from PEP: H4-K5*(H3-H4))
         TarsToGroundAngle1 = (TGA_A_H4 -(((long)IKFeetPosY*(TGA_B_H3-TGA_A_H4))/c1DEC));
      else              //IKFeetPosY <= -10, Only compensate TGA1 when it exceed 40 deg
         TarsToGroundAngle1 = TGA_B_H3;

      //Calc Tars Offsets:
      GetSinCos(TarsToGroundAngle1);
      TarsOffsetXZ = ((long)sin4*(byte)pgm_read_byte(&cTarsLength[LegIKLegNr]))/c4DEC;
      TarsOffsetY = ((long)cos4*(byte)pgm_read_byte(&cTarsLength[LegIKLegNr]))/c4DEC;
   } else {
      TarsOffsetXZ = 0;
      TarsOffsetY = 0;
   }
#endif

   //Using GetAtan2 for solving IKA1 and IKSW
   //IKA14 - Angle between SW line and the ground in radians
   IKA14 = GetATan2 (IKFeetPosY-TarsOffsetY, IKFeetPosXZ-(byte)pgm_read_byte(&cCoxaLength[LegIKLegNr])-TarsOffsetXZ);

   //IKSW2 - Length between femur axis and tars
   IKSW2 = XYhyp2;

   //IKA2 - Angle of the line S>W with respect to the femur in radians
   Temp1 = ((((long)(byte)pgm_read_byte(&cFemurLength[LegIKLegNr])*(byte)pgm_read_byte(&cFemurLength[LegIKLegNr])) - 
((long)(byte)pgm_read_byte(&cTibiaLength[LegIKLegNr])*(byte)pgm_read_byte(&cTibiaLength[LegIKLegNr])))*c4DEC + 
((long)IKSW2*IKSW2));
   Temp2 = (long)(2*(byte)pgm_read_byte(&cFemurLength[LegIKLegNr]))*c2DEC * (unsigned long)IKSW2;
   T3 = Temp1 / (Temp2/c4DEC);
   IKA24 = GetArcCos (T3 );
   //IKFemurAngle
   FemurAngle1[LegIKLegNr] = -(long)(IKA14 + IKA24) * 180 / 3141 + 900 + CFEMURHORNOFFSET1(LegIKLegNr);

   //IKTibiaAngle
```

```
   Temp1 = ((((long)(byte)pgm_read_byte(&cFemurLength[LegIKLegNr])*(byte)pgm_read_byte(&cFemurLength[LegIKLegNr])) +
((long)(byte)pgm_read_byte(&cTibiaLength[LegIKLegNr])*(byte)pgm_read_byte(&cTibiaLength[LegIKLegNr])))*c4DEC -
((long)IKSW2*IKSW2));
   Temp2 = (2*(byte)pgm_read_byte(&cFemurLength[LegIKLegNr])*(byte)pgm_read_byte(&cTibiaLength[LegIKLegNr]));
   GetArcCos (Temp1 / Temp2);
   TibiaAngle1[LegIKLegNr] = -(900-(long)AngleRad4*180/3141);

#ifdef c4DOF
   //Tars angle
   if ((byte)pgm_read_byte(&cTarsLength[LegIKLegNr])) {    // We allow mix of 3 and 4 DOF legs...
      TarsAngle1[LegIKLegNr] = (TarsToGroundAngle1 + FemurAngle1[LegIKLegNr] - TibiaAngle1[LegIKLegNr])
          + CTARSHORNOFFSET1(LegIKLegNr);
   }
#endif

   //Set the Solution quality
   if(IKSW2 < ((byte)pgm_read_byte(&cFemurLength[LegIKLegNr])+(byte)pgm_read_byte(&cTibiaLength[LegIKLegNr])-
30)*c2DEC)
      IKSolution = 1;
   else
   {
      if(IKSW2 <
((byte)pgm_read_byte(&cFemurLength[LegIKLegNr])+(byte)pgm_read_byte(&cTibiaLength[LegIKLegNr]))*c2DEC)
         IKSolutionWarning = 1;
      else
         IKSolutionError = 1    ;
   }
}


//--------------------------------------------------------------------
//[CHECK ANGLES] Checks the mechanical limits of the servos
//--------------------------------------------------------------------
void CheckAngles(void)
{

   for (LegIndex = 0; LegIndex <=5; LegIndex++)
   {
      CoxaAngle1[LegIndex]  = min(max(CoxaAngle1[LegIndex], (short)pgm_read_word(&cCoxaMin1[LegIndex])),
            (short)pgm_read_word(&cCoxaMax1[LegIndex]));
      FemurAngle1[LegIndex] = min(max(FemurAngle1[LegIndex], (short)pgm_read_word(&cFemurMin1[LegIndex])),
            (short)pgm_read_word(&cFemurMax1[LegIndex]));
      TibiaAngle1[LegIndex] = min(max(TibiaAngle1[LegIndex], (short)pgm_read_word(&cTibiaMin1[LegIndex])),
            (short)pgm_read_word(&cTibiaMax1[LegIndex]));
#ifdef c4DOF
      if ((byte)pgm_read_byte(&cTarsLength[LegIndex])) {    // We allow mix of 3 and 4 DOF legs...
         TarsAngle1[LegIndex] = min(max(TarsAngle1[LegIndex], (short)pgm_read_word(&cTarsMin1[LegIndex])),
            (short)pgm_read_word(&cTarsMax1[LegIndex]));
      }
#endif
   }
}



//--------------------------------------------------------------------
```

```
// Why are we faulting?
//-------------------------------------------------------------------


void PrintSystemStuff(void)          // Try to see why we fault...
{

}

// BUGBUG:: Move to some library...
//==============================================================================
//    SoundNoTimer - Quick and dirty tone function to try to output a frequency
//          to a speaker for some simple sounds.
//==============================================================================
void SoundNoTimer(uint8_t _pin, unsigned long duration,  unsigned int frequency)
{
#ifdef __AVR__
  volatile uint8_t *pin_port;
  volatile uint8_t pin_mask;
#else
  volatile uint32_t *pin_port;
  volatile uint16_t pin_mask;
#endif
  long toggle_count = 0;
  long lusDelayPerHalfCycle;

  // Set the pinMode as OUTPUT
  pinMode(_pin, OUTPUT);

  pin_port = portOutputRegister(digitalPinToPort(_pin));
  pin_mask = digitalPinToBitMask(_pin);

  toggle_count = 2 * frequency * duration / 1000;
  lusDelayPerHalfCycle = 1000000L/(frequency * 2);

  // if we are using an 8 bit timer, scan through prescalars to find the best fit
  while (toggle_count--) {
    // toggle the pin
    *pin_port ^= pin_mask;

    // delay a half cycle
    delayMicroseconds(lusDelayPerHalfCycle);
  }
  *pin_port &= ~(pin_mask);  // keep pin low after stop

}

void MSound(uint8_t _pin, byte cNotes, ...)
{
  va_list ap;
  unsigned int uDur;
  unsigned int uFreq;
  va_start(ap, cNotes);

  while (cNotes > 0) {
    uDur = va_arg(ap, unsigned int);
```

```
    uFreq = va_arg(ap, unsigned int);
    SoundNoTimer(_pin, uDur, uFreq);
    cNotes--;
  }
  va_end(ap);
}

#ifdef OPT_TERMINAL_MONITOR
//==============================================================================
// TerminalMonitor - Simple background task checks to see if the user is asking
//    us to do anything, like update debug levels ore the like.
//==============================================================================
boolean TerminalMonitor(void)
{
  byte szCmdLine[5];  // currently pretty simple command lines...
  int ich;
  int ch;
  // See if we need to output a prompt.
  if (g_fShowDebugPrompt) {
    DBGSerial.println("Arduino Phoenix Monitor");
    DBGSerial.println("D - Toggle debug on or off");
#ifdef OPT_FIND_SERVO_OFFSETS
    DBGSerial.println("O - Enter Servo offset mode");
#endif
#ifdef OPT_SSC_FORWARDER
    DBGSerial.println("S - SSC Forwarder");
#endif
    g_fShowDebugPrompt = false;
  }

  // First check to see if there is any characters to process.
  if (ich = DBGSerial.available()) {
    ich = 0;
    // For now assume we receive a packet of data from serial monitor, as the user has
    // to click the send button...
    for (ich=0; ich < sizeof(szCmdLine); ich++) {
      ch = DBGSerial.read();       // get the next character
      if ((ch == -1) || ((ch >= 10) && (ch <= 15)))
        break;
       szCmdLine[ich] = ch;
    }
    szCmdLine[ich] = '\0';    // go ahead and null terminate it...
    DBGSerial.print("Serial Cmd Line:");
    DBGSerial.write(szCmdLine, ich);
    DBGSerial.println("!!!");

    // So see what are command is.
    if (ich == 0) {
      g_fShowDebugPrompt = true;
    } else if ((ich == 1) && ((szCmdLine[0] == 'd') || (szCmdLine[0] == 'D'))) {
      g_fDebugOutput = !g_fDebugOutput;
      if (g_fDebugOutput)
        DBGSerial.println("Debug is on");
      else
        DBGSerial.println("Debug is off");
#ifdef OPT_FIND_SERVO_OFFSETS
```

```
      } else if ((ich == 1) && ((szCmdLine[0] == 'o') || (szCmdLine[0] == 'O'))) {
        g_ServoDriver.FindServoOffsets();
#endif
#ifdef OPT_SSC_FORWARDER
      } else if ((ich == 1) && ((szCmdLine[0] == 's') || (szCmdLine[0] == 'S'))) {
        g_ServoDriver.SSCForwarder();
#endif
      }

      return true;
   }
   return false;
}
#endif

//--------------------------------------------------------------------
// SmoothControl (From Zenta) -  This function makes the body
//          rotation and translation much smoother while walking
//--------------------------------------------------------------------
short SmoothControl (short CtrlMoveInp, short CtrlMoveOut, byte CtrlDivider)
{
   if (fWalking)
   {
      if (CtrlMoveOut < (CtrlMoveInp - 4))
         return CtrlMoveOut + abs((CtrlMoveOut - CtrlMoveInp)/CtrlDivider);
      else if (CtrlMoveOut > (CtrlMoveInp + 4))
         return CtrlMoveOut - abs((CtrlMoveOut - CtrlMoveInp)/CtrlDivider);
   }

   return CtrlMoveInp;
}
```

APPENDIX C

```
/**************************************************************
*   Inverse Kinematics code to control a (modified)
*   LynxMotion AL5D robot arm using a PS2 controller.
*
*   Original IK code by Oleg Mazurov:
*       www.circuitsathome.com/mcu/robotic-arm-inverse-kinematics-on-arduino
*
*   Great intro to IK, with illustrations:
*       github.com/EricGoldsmith/AL5D-BotBoarduino-PS2/blob/master/Robot_Arm_IK.pdf
*
*   Revamped to use BotBoarduino microcontroller:
*       www.lynxmotion.com/c-153-botboarduino.aspx
*   Arduino Servo library:
*       arduino.cc/en/Reference/Servo
*   and PS2X controller library:
*       github.com/madsci1016/Arduino-PS2X
*
*   PS2 Controls
*       Right Joystick L/R: Gripper tip X position (side to side)
*       Right Joystick U/D: Gripper tip Y position (distance out from base center)
*       R1/R2 Buttons:      Gripper tip Z position (height from surface)
*       Left  Joystick L/R: Wrist rotate (if installed)
*       Left  Joystick U/D: Wrist angle
*       L1/L2 Buttons:      Gripper close/open
*       X Button:           Gripper fully open
*       Digital Pad U/D:    Speed increase/decrease
*
*   Eric Goldsmith
*   www.ericgoldsmith.com
*
*   Current Version:
*       https://github.com/EricGoldsmith/AL5D-BotBoarduino-PS2
*   Version history
*       0.1 Initial port of code to use Arduino Server Library
*       0.2 Added PS2 controls
*       0.3 Added constraint logic & 2D kinematics
*       0.4 Added control to modify speed of movement during program run
*       0.5 Write to servos directly in microseconds to improve resolution
*           Should be accurate to ~1/2 a degree
*
*    To Do
*    - Improve arm parking logic to gently move to park position
*
* This program is free software: you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation, either version 3 of the License, or
* (at your option) any later version.
* This program is distributed in the hope that it will be useful,
* but WITHOUT ANY WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
* GNU General Public License for more details.
* <http://www.gnu.org/licenses/>
*
**************************************************************/
```

```
#include <Servo.h>
#include <PS2X_lib.h>

int dummy;                  // Defining this dummy variable to work around a bug in
the
                            // IDE (1.0.3) pre-processor that messes up #ifdefs
                            // More info:
http://code.google.com/p/arduino/issues/detail?id=906
                            //
http://code.google.com/p/arduino/issues/detail?id=987
                            //
http://arduino.cc/forum/index.php/topic,125769.0.html

//#define DEBUG             // Uncomment to turn on debugging output

#define CYL_IK              // Apply only 2D, or cylindrical, kinematics. The X-axis
component is
                            // removed from the equations by fixing it at 0. The arm
position is
                            // calculated in the Y and Z planes, and simply rotates
around the base.

//#define WRIST_ROTATE      // Uncomment if wrist rotate hardware is installed

// Arm dimensions (mm). Standard AL5D arm, but with longer arm segments
#define BASE_HGT 80.9625    // Base height to X/Y plane 3.1875"
#define HUMERUS 263.525     // Shoulder-to-elbow "bone" 10.375"
#define ULNA 325.4375       // Elbow-to-wrist "bone" 12.8125"
#define GRIPPER 73.025      // Gripper length, to middle of grip surface 2.875"
(3.375" - 0.5")

// Arduino pin numbers for servo connections
#define BAS_SERVO_PIN 2     // Base servo HS-485HB
#define SHL_SERVO_PIN 3     // Shoulder Servo HS-805BB
#define ELB_SERVO_PIN 4     // Elbow Servo HS-755HB
#define WRI_SERVO_PIN 10    // Wrist servo HS-645MG
#define GRI_SERVO_PIN 11    // Gripper servo HS-422
#ifdef WRIST_ROTATE
 #define WRO_SERVO_PIN 12   // Wrist rotate servo HS-485HB
#endif

// Arduino pin numbers for PS2 controller connections
#define PS2_CLK_PIN 9       // Clock
#define PS2_CMD_PIN 7       // Command
#define PS2_ATT_PIN 8       // Attention
#define PS2_DAT_PIN 6       // Data

// Arduino pin number of on-board speaker
#define SPK_PIN 5

// Define generic range limits for servos, in microseconds (us) and degrees (deg)
// Used to map range of 180 deg to 1800 us (native servo units).
// Specific per-servo/joint limits are defined below
#define SERVO_MIN_US 600
#define SERVO_MID_US 1500
#define SERVO_MAX_US 2400
#define SERVO_MIN_DEG 0.0
#define SERVO_MID_DEG 90.0
#define SERVO_MAX_DEG 180.0

// Set physical limits (in degrees) per servo/joint.
```

```
// Will vary for each servo/joint, depending on mechanical range of motion.
// The MID setting is the required servo input needed to achieve a
// 90 degree joint angle, to allow compensation for horn misalignment
#define BAS_MIN 0.0         // Fully CCW
#define BAS_MID 90.0
#define BAS_MAX 180.0       // Fully CW

#define SHL_MIN 20.0        // Max forward motion
#define SHL_MID 81.0
#define SHL_MAX 140.0       // Max rearward motion

#define ELB_MIN 20.0        // Max upward motion
#define ELB_MID 88.0
#define ELB_MAX 165.0       // Max downward motion

#define WRI_MIN 0.0         // Max downward motion
#define WRI_MID 93.0
#define WRI_MAX 180.0       // Max upward motion

#define GRI_MIN 25.0        // Fully open
#define GRI_MID 90.0
#define GRI_MAX 165.0       // Fully closed

#ifdef WRIST_ROTATE
 #define WRO_MIN 0.0
 #define WRO_MID 90.0
 #define WRO_MAX 180.0
#endif

// Speed adjustment parameters
// Percentages (1.0 = 100%) - applied to all arm movements
#define SPEED_MIN 0.5
#define SPEED_MAX 1.5
#define SPEED_DEFAULT 1.0
#define SPEED_INCREMENT 0.25

// Practical navigation limit.
// Enforced on controller input, and used for CLV calculation
// for base rotation in 2D mode.
#define Y_MIN 100.0         // mm

// PS2 controller characteristics
#define JS_MIDPOINT 128     // Numeric value for joystick midpoint
#define JS_DEADBAND 4       // Ignore movement this close to the center position
#define JS_IK_SCALE 50.0    // Divisor for scaling JS output for IK control
#define JS_SCALE 100.0      // Divisor for scaling JS output for raw servo control
#define Z_INCREMENT 2.0     // Change in Z axis (mm) per button press
#define G_INCREMENT 2.0     // Change in Gripper jaw opening (servo angle) per button
press

// Audible feedback sounds
#define TONE_READY 1000     // Hz
#define TONE_IK_ERROR 200   // Hz
#define TONE_DURATION 100   // ms

// IK function return values
#define IK_SUCCESS 0
#define IK_ERROR 1          // Desired position not possible

// Arm parking positions
#define PARK_MIDPOINT 1     // Servos at midpoints
```

```
#define PARK_READY 2        // Arm at Ready-To-Run position

// Ready-To-Run arm position. See descriptions below
// NOTE: Have the arm near this position before turning on the
//       servo power to prevent whiplash
#ifdef CYL_IK   // 2D kinematics
 #define READY_BA (BAS_MID - 45.0)
#else           // 3D kinematics
 #define READY_X 0.0
#endif
#define READY_Y 170.0
#define READY_Z 45.0
#define READY_GA 0.0
#define READY_G GRI_MID
#ifdef  WRIST_ROTATE
 #define READY_WR WRO_MID
#endif

// Global variables for arm position, and initial settings
#ifdef CYL_IK   // 2D kinematics
 float BA = READY_BA;       // Base angle. Servo degrees - 0 is fully CCW
#else           // 3D kinematics
 float X = READY_X;         // Left/right distance (mm) from base centerline - 0 is
straight
#endif
float Y = READY_Y;          // Distance (mm) out from base center
float Z = READY_Z;          // Height (mm) from surface (i.e. X/Y plane)
float GA = READY_GA;        // Gripper angle. Servo degrees, relative to X/Y plane - 0
is horizontal
float G = READY_G;          // Gripper jaw opening. Servo degrees - midpoint is
halfway open
#ifdef  WRIST_ROTATE
 float WR = READY_WR;       // Wrist Rotate. Servo degrees - midpoint is horizontal
#endif
float Speed = SPEED_DEFAULT;

// Pre-calculations
float hum_sq = HUMERUS*HUMERUS;
float uln_sq = ULNA*ULNA;

// PS2 Controller object
PS2X    Ps2x;

// Servo objects
Servo   Bas_Servo;
Servo   Shl_Servo;
Servo   Elb_Servo;
Servo   Wri_Servo;
Servo   Gri_Servo;
#ifdef WRIST_ROTATE
 Servo   Wro_Servo;
#endif

void setup()
{
#ifdef DEBUG
    Serial.begin(115200);
#endif

    // Attach to the servos and specify range limits
    Bas_Servo.attach(BAS_SERVO_PIN, SERVO_MIN_US, SERVO_MAX_US);
```

```
    Shl_Servo.attach(SHL_SERVO_PIN, SERVO_MIN_US, SERVO_MAX_US);
    Elb_Servo.attach(ELB_SERVO_PIN, SERVO_MIN_US, SERVO_MAX_US);
    Wri_Servo.attach(WRI_SERVO_PIN, SERVO_MIN_US, SERVO_MAX_US);
    Gri_Servo.attach(GRI_SERVO_PIN, SERVO_MIN_US, SERVO_MAX_US);
#ifdef WRIST_ROTATE
    Wro_Servo.attach(WRO_SERVO_PIN, SERVO_MIN_US, SERVO_MAX_US);
#endif

    // Setup PS2 controller. Loop until ready.
    byte    ps2_stat;
    do {
        ps2_stat = Ps2x.config_gamepad(PS2_CLK_PIN, PS2_CMD_PIN, PS2_ATT_PIN,
PS2_DAT_PIN);
#ifdef DEBUG
        if (ps2_stat == 1)
            Serial.println("No controller found. Re-trying ...");
#endif
    } while (ps2_stat == 1);

#ifdef DEBUG
    switch (ps2_stat) {
        case 0:
            Serial.println("Found Controller, configured successfully.");
            break;
        case 2:
            Serial.println("Controller found but not accepting commands.");
            break;
        case 3:
            Serial.println("Controller refusing to enter 'Pressures' mode, may not
support it. ");
            break;
    }
#endif

    // NOTE: Ensure arm is close to the desired park position before turning on servo
power!
    servo_park(PARK_READY);

#ifdef DEBUG
    Serial.println("Start");
#endif

    delay(500);
    // Sound tone to indicate it's safe to turn on servo power
    tone(SPK_PIN, TONE_READY, TONE_DURATION);
    delay(TONE_DURATION * 2);
    tone(SPK_PIN, TONE_READY, TONE_DURATION);

}

void loop()
{
    // Store desired position in tmp variables until confirmed by set_arm() logic
#ifdef CYL_IK   // 2D kinematics
    // not used
#else           // 3D kinematics
    float x_tmp = X;
#endif
    float y_tmp = Y;
    float z_tmp = Z;
    float ga_tmp = GA;
```

```
    // Used to indidate whether an input occurred that can move the arm
    boolean arm_move = false;

    Ps2x.read_gamepad();        //read controller

    // Read the left and right joysticks and translate the
    // normal range of values (0-255) to zero-centered values (-128 - 128)
    int ly_trans = JS_MIDPOINT - Ps2x.Analog(PSS_LY);
    int lx_trans = Ps2x.Analog(PSS_LX) - JS_MIDPOINT;
    int ry_trans = JS_MIDPOINT - Ps2x.Analog(PSS_RY);
    int rx_trans = Ps2x.Analog(PSS_RX) - JS_MIDPOINT;

#ifdef CYL_IK   // 2D kinematics
    // Base Position (in degrees)
    // Restrict to MIN/MAX range of servo
    if (abs(rx_trans) > JS_DEADBAND) {
        // Muliplyting by the ratio (Y_MIN/Y) is to ensure constant linear velocity
        // of the gripper as its distance from the base increases
        BA += ((float)rx_trans / JS_SCALE * Speed * (Y_MIN/Y));
        BA = constrain(BA, BAS_MIN, BAS_MAX);
        Bas_Servo.writeMicroseconds(deg_to_us(BA));

        if (BA == BAS_MIN || BA == BAS_MAX) {
            // Provide audible feedback of reaching limit
            tone(SPK_PIN, TONE_IK_ERROR, TONE_DURATION);
        }
    }
#else           // 3D kinematics
    // X Position (in mm)
    // Can be positive or negative. Servo range checking in IK code
    if (abs(rx_trans) > JS_DEADBAND) {
        x_tmp += ((float)rx_trans / JS_IK_SCALE * Speed);
        arm_move = true;
    }
#endif

    // Y Position (in mm)
    // Must be > Y_MIN. Servo range checking in IK code
    if (abs(ry_trans) > JS_DEADBAND) {
        y_tmp += ((float)ry_trans / JS_IK_SCALE * Speed);
        y_tmp = max(y_tmp, Y_MIN);
        arm_move = true;

        if (y_tmp == Y_MIN) {
            // Provide audible feedback of reaching limit
            tone(SPK_PIN, TONE_IK_ERROR, TONE_DURATION);
        }
    }

    // Z Position (in mm)
    // Must be positive. Servo range checking in IK code
    if (Ps2x.Button(PSB_R1) || Ps2x.Button(PSB_R2)) {
        if (Ps2x.Button(PSB_R1)) {
            z_tmp += Z_INCREMENT * Speed;   // up
        } else {
            z_tmp -= Z_INCREMENT * Speed;   // down
        }
        z_tmp = max(z_tmp, 0);
        arm_move = true;
    }
```

```
    // Gripper angle (in degrees) relative to horizontal
    // Can be positive or negative. Servo range checking in IK code
    if (abs(ly_trans) > JS_DEADBAND) {
        ga_tmp -= ((float)ly_trans / JS_SCALE * Speed);
        arm_move = true;
    }

    // Gripper jaw position (in degrees - determines width of jaw opening)
    // Restrict to MIN/MAX range of servo
    if (Ps2x.Button(PSB_L1) || Ps2x.Button(PSB_L2)) {
        if (Ps2x.Button(PSB_L1)) {
            G += G_INCREMENT;   // close
        } else {
            G -= G_INCREMENT;   // open
        }
        G = constrain(G, GRI_MIN, GRI_MAX);
        Gri_Servo.writeMicroseconds(deg_to_us(G));

        if (G == GRI_MIN || G == GRI_MAX) {
            // Provide audible feedback of reaching limit
            tone(SPK_PIN, TONE_IK_ERROR, TONE_DURATION);
        }
    }

    // Fully open gripper
    if (Ps2x.ButtonPressed(PSB_BLUE)) {
        G = GRI_MIN;
        Gri_Servo.writeMicroseconds(deg_to_us(G));
    }

    // Speed increase/decrease
    if (Ps2x.ButtonPressed(PSB_PAD_UP) || Ps2x.ButtonPressed(PSB_PAD_DOWN)) {
        if (Ps2x.ButtonPressed(PSB_PAD_UP)) {
            Speed += SPEED_INCREMENT;   // increase speed
        } else {
            Speed -= SPEED_INCREMENT;   // decrease speed
        }
        // Constrain to limits
        Speed = constrain(Speed, SPEED_MIN, SPEED_MAX);

        // Audible feedback
        tone(SPK_PIN, (TONE_READY * Speed), TONE_DURATION);
    }

#ifdef WRIST_ROTATE
    // Wrist rotate (in degrees)
    // Restrict to MIN/MAX range of servo
    if (abs(lx_trans) > JS_DEADBAND) {
        WR += ((float)lx_trans / JS_SCALE * Speed);
        WR = constrain(WR, WRO_MIN, WRO_MAX);
        Wro_Servo.writeMicroseconds(deg_to_us(WR));

        if (WR == WRO_MIN || WR == WRO_MAX) {
            // Provide audible feedback of reaching limit
            tone(SPK_PIN, TONE_IK_ERROR, TONE_DURATION);
        }
    }
#endif

    // Only perform IK calculations if arm motion is needed.
```

```
    if (arm_move) {
#ifdef CYL_IK   // 2D kinematics
        if (set_arm(0, y_tmp, z_tmp, ga_tmp) == IK_SUCCESS) {
            // If the arm was positioned successfully, record
            // the new vales. Otherwise, ignore them.
            Y = y_tmp;
            Z = z_tmp;
            GA = ga_tmp;
        } else {
            // Sound tone for audible feedback of error
            tone(SPK_PIN, TONE_IK_ERROR, TONE_DURATION);
        }
#else           // 3D kinematics
        if (set_arm(x_tmp, y_tmp, z_tmp, ga_tmp) == IK_SUCCESS) {
            // If the arm was positioned successfully, record
            // the new vales. Otherwise, ignore them.
            X = x_tmp;
            Y = y_tmp;
            Z = z_tmp;
            GA = ga_tmp;
        } else {
            // Sound tone for audible feedback of error
            tone(SPK_PIN, TONE_IK_ERROR, TONE_DURATION);
        }
#endif

        // Reset the flag
        arm_move = false;
    }

    delay(10);
 }

// Arm positioning routine utilizing Inverse Kinematics.
// Z is height, Y is distance from base center out, X is side to side. Y, Z can only
be positive.
// Input dimensions are for the gripper, just short of its tip, where it grabs things.
// If resulting arm position is physically unreachable, return error code.
int set_arm(float x, float y, float z, float grip_angle_d)
{
    //grip angle in radians for use in calculations
    float grip_angle_r = radians(grip_angle_d);

    // Base angle and radial distance from x,y coordinates
    float bas_angle_r = atan2(x, y);
    float rdist = sqrt((x * x) + (y * y));

    // rdist is y coordinate for the arm
    y = rdist;

    // Grip offsets calculated based on grip angle
    float grip_off_z = (sin(grip_angle_r)) * GRIPPER;
    float grip_off_y = (cos(grip_angle_r)) * GRIPPER;

    // Wrist position
    float wrist_z = (z - grip_off_z) - BASE_HGT;
    float wrist_y = y - grip_off_y;

    // Shoulder to wrist distance (AKA sw)
    float s_w = (wrist_z * wrist_z) + (wrist_y * wrist_y);
    float s_w_sqrt = sqrt(s_w);
```

```
    // s_w angle to ground
    float a1 = atan2(wrist_z, wrist_y);

    // s_w angle to humerus
    float a2 = acos(((hum_sq - uln_sq) + s_w) / (2 * HUMERUS * s_w_sqrt));

    // Shoulder angle
    float shl_angle_r = a1 + a2;
    // If result is NAN or Infinity, the desired arm position is not possible
    if (isnan(shl_angle_r) || isinf(shl_angle_r))
        return IK_ERROR;
    float shl_angle_d = degrees(shl_angle_r);

    // Elbow angle
    float elb_angle_r = acos((hum_sq + uln_sq - s_w) / (2 * HUMERUS * ULNA));
    // If result is NAN or Infinity, the desired arm position is not possible
    if (isnan(elb_angle_r) || isinf(elb_angle_r))
        return IK_ERROR;
    float elb_angle_d = degrees(elb_angle_r);
    float elb_angle_dn = -(180.0 - elb_angle_d);

    // Wrist angle
    float wri_angle_d = (grip_angle_d - elb_angle_dn) - shl_angle_d;

    // Calculate servo angles
    // Calc relative to servo midpoint to allow compensation for servo alignment
    float bas_pos = BAS_MID + degrees(bas_angle_r);
    float shl_pos = SHL_MID + (shl_angle_d - 90.0);
    float elb_pos = ELB_MID - (elb_angle_d - 90.0);
    float wri_pos = WRI_MID + wri_angle_d;

    // If any servo ranges are exceeded, return an error
    if (bas_pos < BAS_MIN || bas_pos > BAS_MAX || shl_pos < SHL_MIN || shl_pos >
SHL_MAX || elb_pos < ELB_MIN || elb_pos > ELB_MAX || wri_pos < WRI_MIN || wri_pos >
WRI_MAX)
        return IK_ERROR;

    // Position the servos
#ifdef CYL_IK   // 2D kinematics
    // Do not control base servo
#else           // 3D kinematics
    Bas_Servo.writeMicroseconds(deg_to_us(bas_pos));
#endif
    Shl_Servo.writeMicroseconds(deg_to_us(shl_pos));
    Elb_Servo.writeMicroseconds(deg_to_us(elb_pos));
    Wri_Servo.writeMicroseconds(deg_to_us(wri_pos));

#ifdef DEBUG
    Serial.print("X: ");
    Serial.print(x);
    Serial.print("  Y: ");
    Serial.print(y);
    Serial.print("  Z: ");
    Serial.print(z);
    Serial.print("  GA: ");
    Serial.print(grip_angle_d);
    Serial.println();
    Serial.print("Base Pos: ");
    Serial.print(bas_pos);
    Serial.print("  Shld Pos: ");
```

```
    Serial.print(shl_pos);
    Serial.print("  Elbw Pos: ");
    Serial.print(elb_pos);
    Serial.print("  Wrst Pos: ");
    Serial.println(wri_pos);
    Serial.print("bas_angle_d: ");
    Serial.print(degrees(bas_angle_r));
    Serial.print("  shl_angle_d: ");
    Serial.print(shl_angle_d);
    Serial.print("  elb_angle_d: ");
    Serial.print(elb_angle_d);
    Serial.print("  wri_angle_d: ");
    Serial.println(wri_angle_d);
    Serial.println();
#endif

    return IK_SUCCESS;
}

// Move servos to parking position
void servo_park(int park_type)
{
    switch (park_type) {
        // All servos at midpoint
        case PARK_MIDPOINT:
            Bas_Servo.writeMicroseconds(deg_to_us(BAS_MID));
            Shl_Servo.writeMicroseconds(deg_to_us(SHL_MID));
            Elb_Servo.writeMicroseconds(deg_to_us(ELB_MID));
            Wri_Servo.writeMicroseconds(deg_to_us(WRI_MID));
            Gri_Servo.writeMicroseconds(deg_to_us(GRI_MID));
#ifdef WRIST_ROTATE
            Wro_Servo.writeMicroseconds(deg_to_us(WRO_MID));
#endif
            break;

        // Ready-To-Run position
        case PARK_READY:
#ifdef CYL_IK   // 2D kinematics
            set_arm(0.0, READY_Y, READY_Z, READY_GA);
            Bas_Servo.writeMicroseconds(deg_to_us(READY_BA));
#else           // 3D kinematics
            set_arm(READY_X, READY_Y, READY_Z, READY_GA);
#endif
            Gri_Servo.writeMicroseconds(deg_to_us(READY_G));
#ifdef WRIST_ROTATE
            Wro_Servo.writeMicroseconds(deg_to_us(READY_WR));
#endif
            break;
    }

    return;
}

// The Arduino Servo library .write() function accepts 'int' degrees, meaning
// maximum servo positioning resolution is whole degrees. Servos are capable
// of roughly 2x that resolution via direct microsecond control.
//
// This function converts 'float' (i.e. decimal) degrees to corresponding
// servo microseconds to take advantage of this extra resolution.
int deg_to_us(float value)
{
```

```
    // Apply basic constraints
    if (value < SERVO_MIN_DEG) value = SERVO_MIN_DEG;
    if (value > SERVO_MAX_DEG) value = SERVO_MAX_DEG;

    // Map degrees to microseconds, and round the result to a whole number
    return(round(map_float(value, SERVO_MIN_DEG, SERVO_MAX_DEG, (float)SERVO_MIN_US,
(float)SERVO_MAX_US)));
}

// Same logic as native map() function, just operates on float instead of long
float map_float(float x, float in_min, float in_max, float out_min, float out_max)
{
  return ((x - in_min) * (out_max - out_min) / (in_max - in_min)) + out_min;
}
```

REFERENCES

[1] Lynxmotion (acquired by RobotShop Inc. in 2012, Quebec Canada) [Online]. Available: http://www.lynxmotion.com/images/html/build180.htm

[2] Lynxmotion (acquired by RobotShop Inc. in 2012, Quebec Canada) [Online]. Available: http://www.lynxmotion.com/images/data/lynxmotion_ssc-32u_USB_user_guide.pdf

[3] Phidgets [Online]. Available: http://www.phidgets.com/documentation/Phidgets/3204_0_hs645.pdf

[4] Roborobotics [Online]. Available: http://roborobotics.com/Animatronics/show-control-hardware/Servo-controller.html

[5] Lynxmotion (acquired by RobotShop Inc. in 2012, Quebec, Canada) [Online]. Available: http://www.lynxmotion.com/images/html/build99f.htm

[6] J. Arcand (September 2012). RobotShop Inc., Quebec, Canada. [Online] Available: http://www.robotshop.com/blog/en/guide-power-supply-options-lynxmotion-controllers-12178

[7] RobotShop Inc., Quebec, Canada. [Online] Available: http://www.robotshop.com/en/lynxmotion-botboarduino-robot-controller.html

[8] Lynxmotion (acquired by RobotShop Inc. in 2012, Quebec, Canada) [Online]. Available: http://www.robotshop.com/en/co2-sensor-arduino-compatible.html

[9] Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Robot_kinematics#Inverse_kinematics