Hubo+ Manual

Prepared by Andy Park and Roy Chan

May 31st, 2012

ARTLAB
Assistive Robotics Technology

PURDUE
UNIVERSITY

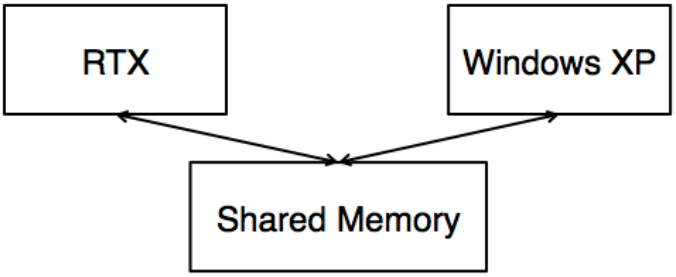
< Hubo+ Training at Drexel >

Updated on 05/31/2012 and written by Andy Park and Roy Chan.
Hubo+ wiki webpage: http://hubo.ece.drexel.edu/

1. Overview of Hubo+ system

- The weight without case is 37kg, and the weight with case is 42kg.
- The voltage that powers motors is 48V, and LiPo batteries are being used. Although the input voltage from the batteries is 48V, it is not directly being used but regulated at 30V and jumped to 48V so that stable input power is provided to motors. The average current is 8A. The voltage that powers up sensors is 12V.
- For hip, knee, shoulder joints, Brushless DC (BLDC) motors are being used and for each one the average power is 200W. For the joints that don't require large amount of torque, general DC motors are currently being used; JMC represents BLDC motors, EJMC represents general DC motors.
- maximum weight that can be grasped by each hand is in the range from 1.5 kg to 3.0kg. The real situation will be lower weight than the maximum one according to what DASL students have checked with their robots. They say that fingers are fragile and can be easily broken (many fingers of Hubo+ are already broken so they are shipped to Korea for repair).
- The total DOFs of the robot is 38 DOFs; 3DOF for neck, 1DOF for Waist, 5DOF for each hand, 6 DOF for each limb. (6 x 4 + 3 + 1 + 5 x 2 = 38)
- The robot is equipped with 5 sensors (1 3-axis Torque/Force sensor for each limb (wrist, ankle), 1 IMU at the waist).
- 2 PCs are used to control the robot – one is used to for body motion control and the other is for computation of other algorithms.
- For the operating system, windows-based RTX is being used. The RTX is very expensive so they are still using an older version (RTX running on windows XP). Visual studio 6 is used to compile the software running the demo program.
- The maximum walking speed is 3.6km/hr, and the normal walking speed is 1.8km/hr.

2. Software Architecture



- The core program provides basic controls (functions) of the 5 sensors such as sensors calibration and access of the status of the sensors. The status of the sensors will then be written in the shared memory.

- GUI has 2 functions. First, it executes several built-in motions (such as walking, dance) through the core program. Second, it displays the status of the sensors and motors from the shared memory.
- The frequency of each command in the core program is controlled by RTX. For the command of lower body, the frequency is 200Hz. For upper body, the frequency is 100Hz. The frequency of motor controller is 1kHz. The discrepancy of the frequency between the command and the motor controller is resolved using interpolation calculated on the motor controller board (PCI104). All the command from the core program to the motor controller is communicated using CAN (controller area network)

3. Calibration Procedure (Lower-body)

- The equipment for this procedure.
: Leveler, a metal plate, a ruler, a laser level, and Hubo-i

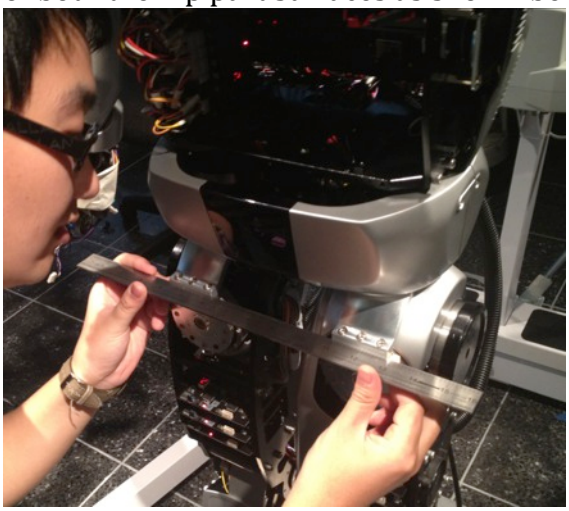
Laser Leveler


Leveler


Ruler

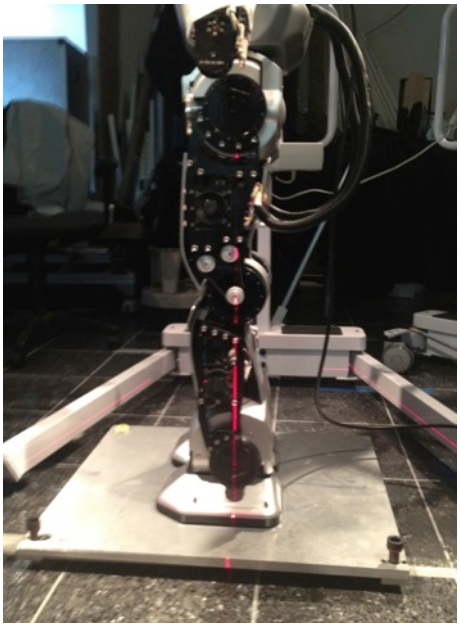
- Once the robot is started up and all the joints are initialized, we turn off the torques of ankle joints by pressing “Ankle off” button. For the convenience, we change the offset of LSP and RSP during the calibration of lower-body joints.
- We need to check whether the plate is level by using a leveler.

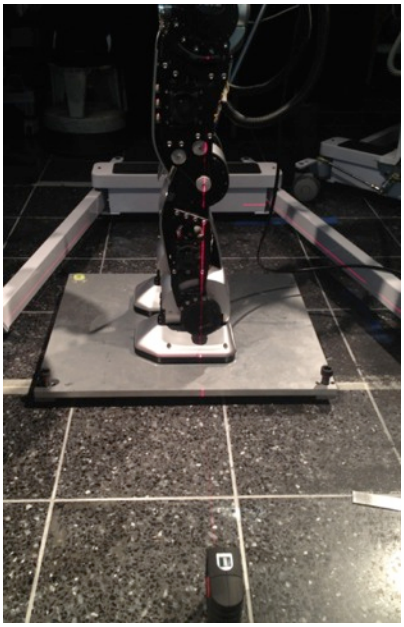
1) Hip Yaw
- Use a ruler to check there is no space between the metal ruler and the frontal part of both the hip part surfaces as shown below.



2) Knee Pitch

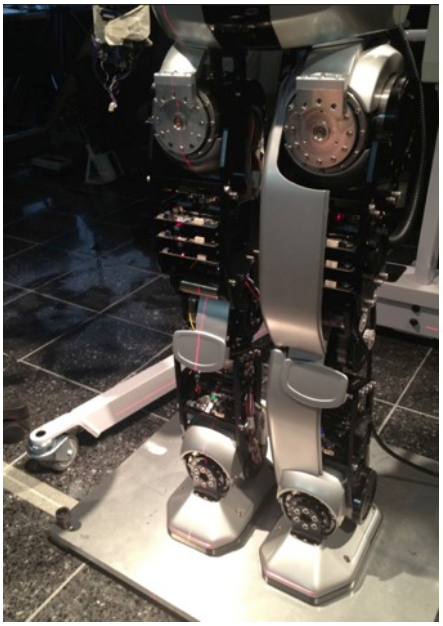
- Use a laser leveler to check the holes of three joints (hip, knee, ankle) from the side views.





3) Hip Roll

- Aim the laser at the hole in the hip roll joint, and check if the hole of ankle roll joint is aligned with the laser line.



4) Hip Pitch

- We also aim the laser beam at the hole of hip pitch joint and make sure the beam is aligned with the three holes (hip pitch, knee pitch, ankle pitch) and the vertical line on the upper-body case.

Note that if Hip Pitch has to be changed, Hip Roll should be changed again as we did in the previous step.

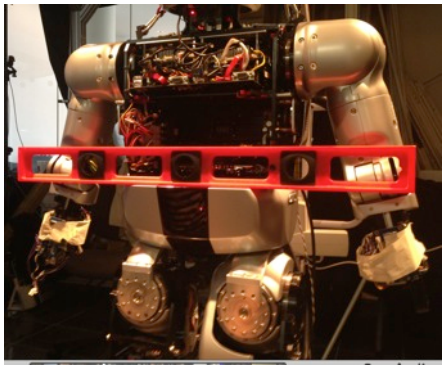
5) Save Offset for the changed values and encoder zero.
- Be sure not to save offset several times. Do it only once after getting all the offset values for hip and knee joints.
- Make sure to turn off sensor and motors and turn on again for the ankle joint calibration.

6) Ankle Joints Calibration
- For ankle joints, we first connect Hubo-i to the system as shown in the following picture. The connector for CAN and power (12V-DC) should be plugged into the hubo system. Note that the first column of the CAN sockets are for lower-body.

- We turn off the torques of the ankle joints first.

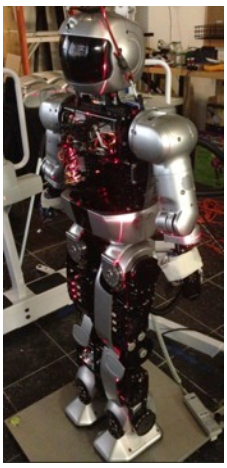
6-1) Ankle Roll

- We check whether the metal plate is level and the waist body is level by using a leveler as shown below.



6-2) Ankle Pitch

- Use the laser leveler to make sure the beam lies on the edge of the upper body to calibrate one ankle pitch (either left or right ankle pitch). Then, to calibrate the other ankle pitch, use the metal ruler to make sure there is no space between the metal ruler and the front parts of both ankles.

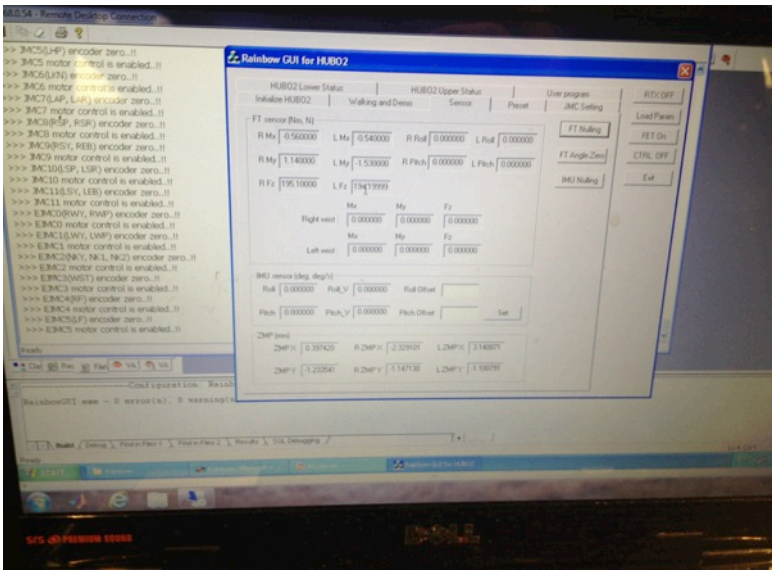


- Using Hubo-i we check the encoder value and add it to the offset in the home set1 of settings menu. Then we turn off sensor and motors and turn them on again.

- Whenever any part is replaced in lower-body, we should repeat the procedure above to make sure all joints are calibrated.

- We should suspect ankle if even after hip and knee calibration the sensor values do not look right because for ankles we don't have a good measure.

- Check the force sensor value along z-axis and IMU sensor values. For setting of offset of IMU sensors, we should subtract out the amount of the current IMU sensor values as shown below.



- Note that while the robot is in “walk ready pose”, do not use “init all pos” button to initialize all joints. Instead, use “go home” to initialize all joints.

4. Disassembling/Assembling parts

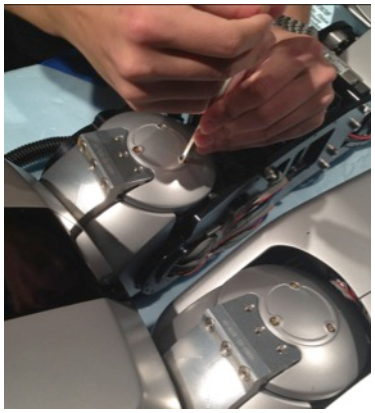
1) Hip Roll joint

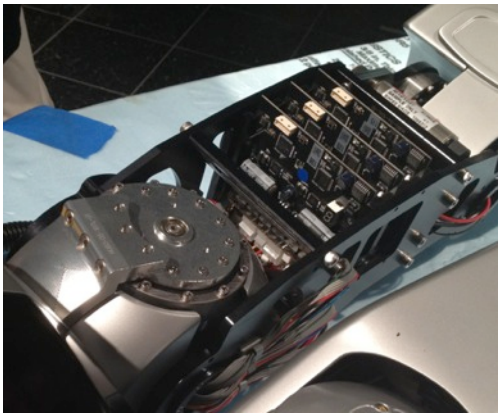
- Tools

: Wrenches, Screw Drivers, Lubricant, box to put screws and gears

1-1) Disassembly

- Remove the shells of upper leg and roll joint

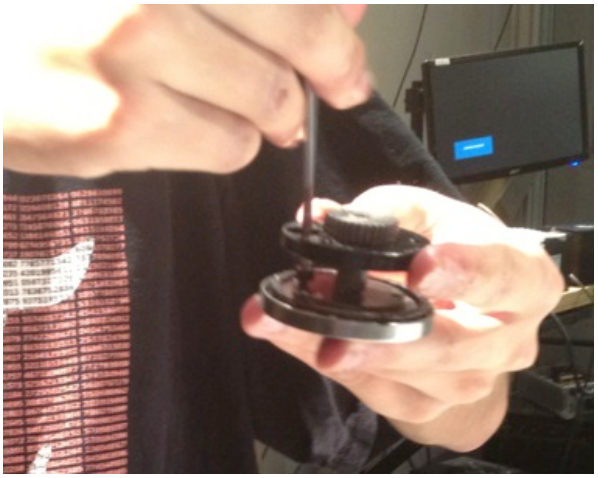




- Remove harmonic drive cover and bearings and gears

- Fix the bearing and remove screws from the bearing through the holes as shown below.

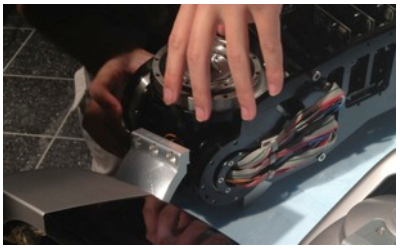
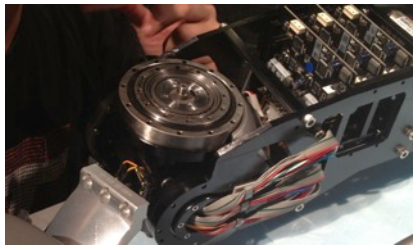
1-2) Disassembly
- Attach the bearing to the gear using screws as shown below.



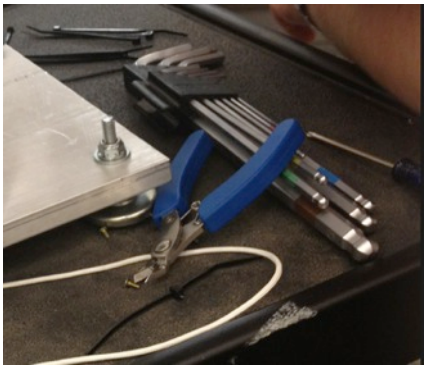
- Insert the wave generator into the bearing like shown below.

- Apply proper amount of lubricant on the gears and shaft and insert the gear into the shaft. Attach the black harmonic mount to the bearing using the screws as shown below.


- Put the harmonic drive back into the hip roll joint and make the gears meet correctly as shown below. Make them tight again using screws. When screwing, the wrench should be grasped properly like in the picture to prevent the screw head from getting damaged. Make sure not to make it too tight. It is advised to twist with good grasp and proper amount of force three times, calling “one, two, three.”

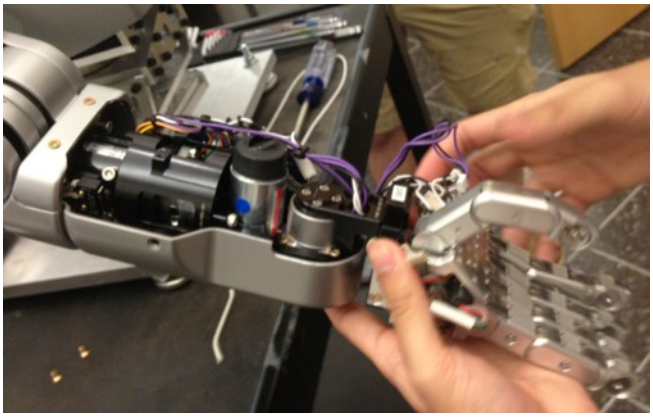




2) Hand and Fingers
- Tools: wrenches, cable ties, screw driver, nipper

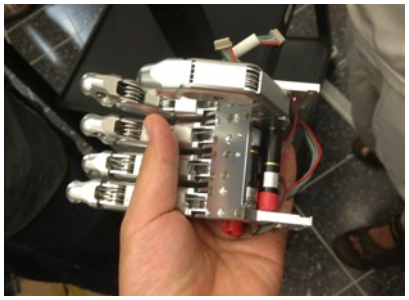
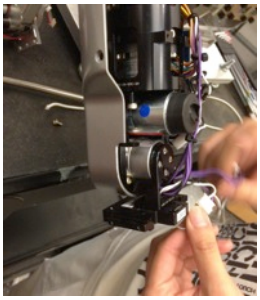


2-1) Disassembly

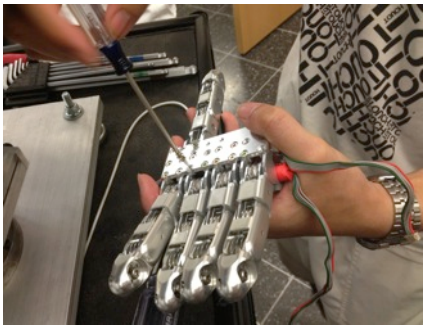
- Remove the shell for the wrist.

- Separate hand from the wrist. Make sure to disconnect the cables from the board of finger joint controllers before detaching the hand.

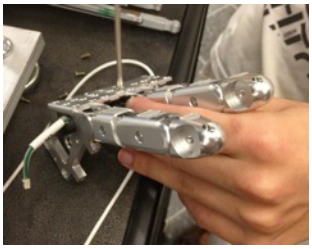




- Each finger has a number so be sure to follow the order (0 2 4 3 1) when removing fingers from the hand body. Use thumb and index finger to slide out each finger.



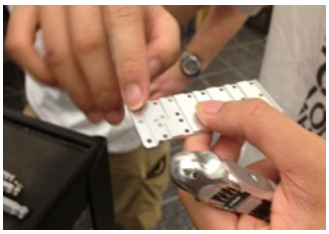
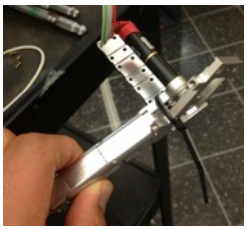
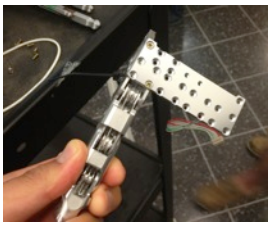

- Once each finger is separated, use a cable tie to hold both sides of frames of each finger.

- When unscrewing, use thumb and index finger to hold each finger. Otherwise, the finger may get broken (there is no part holding both sides of each finger).

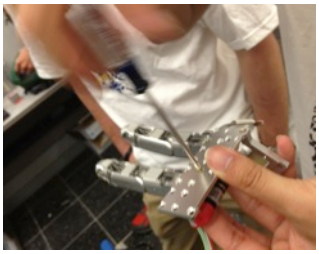
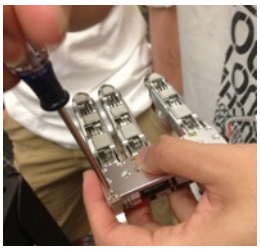


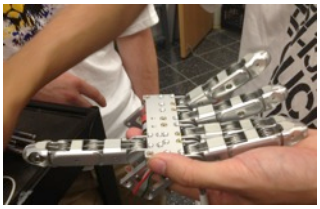
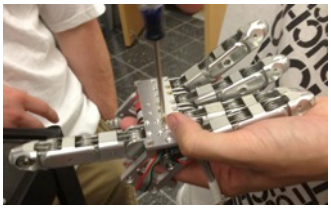
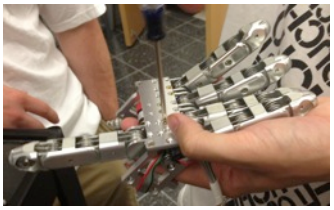
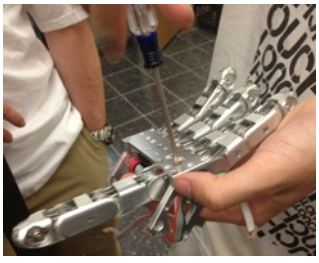
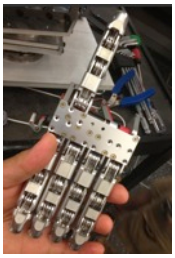
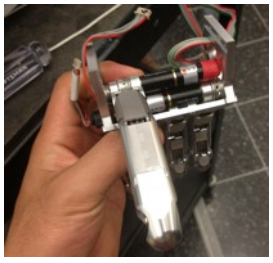
2-2) Assembly

- The finger mount has an indentation to fix each finger. Slide each finger in the indentation following the reversed order of assembly (1 3 4 2 0). Screw opposite side first to make sure the fingers are placed correctly.

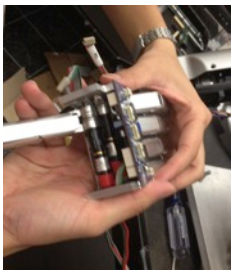
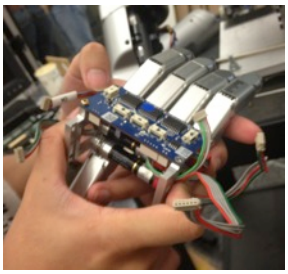




- Continue to mount the rest of fingers on the finger mount following the order.

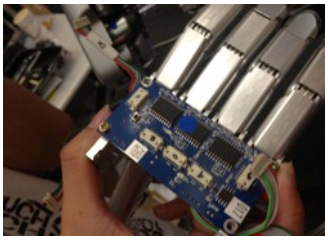
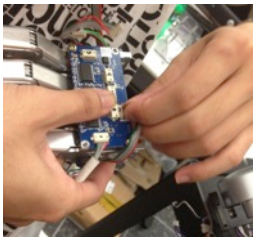
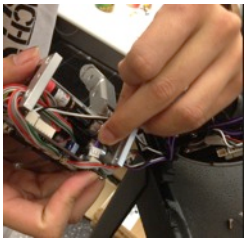



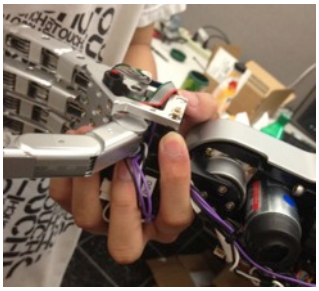
- Mount the controller board after all fingers are mounted.

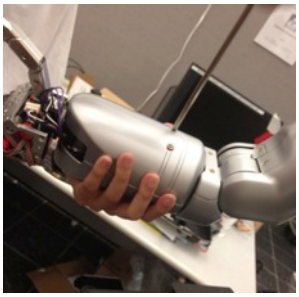
- There is a socket having a number for each finger on the controller board for right hand. And numbers on the socket are for left hand. Insert the connector from each finger to the corresponding socket on the board.

- Once the controller board and all five fingers are connected, mount hand on the wrist by carefully sliding.



- Put the shell back on the wrist.

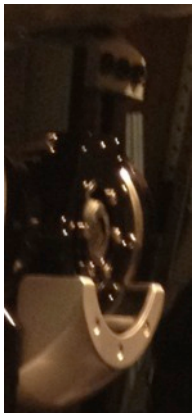


5. Troubleshooting

1) Check Home Position

- When you click “go to home” button, check if each joint is at its home position. If not, that joint is not working properly.

2) Check the six screws for each leg are tight. They get easily loose. Note that the three screws are on the rear side are under the cap.



3) Make sure the shaft and the gears are tight. If they are loose, the joint doesn’t work even though you can hear the motor rotating.

4) Check the sensor reading (F/T sensor)

- Check if the sensor values go to zero after F/T nulling
- Check each F/T sensor value in single support phase for each foot. See if the measured weight is the same from both sensors.

5) Check the sensor reading (IMU sensor)

- check the values (angles) by tilting the robot manually
- check the position and velocity
- check the converging speed

6) ZMP

- Be sure to initialize ZMP and save the new data.
- Note that “ZMP initialization” can be only used in walk-ready pose.

6. Walking Parameter Tuning

1) Swaying parameter (lateral direction)

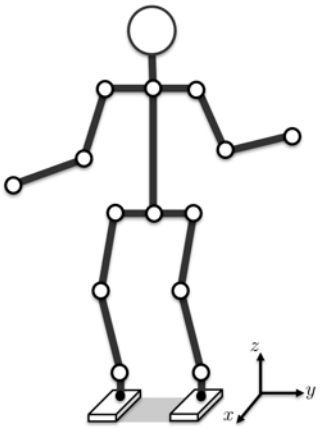
- Usually falling direction tells whether to increase or decrease the sway.
- How to tune sway parameter differs in start walking phase and walking phase

: In start walking phase, if the robot is lifting up its right foot and falling to the left, it means the sway is small. So, increase the sway until the robot is balanced. However, in walking phases, if the right foot is in swing phase and the robot is falling to the left, it means that the sway is too big. Then, decrease the sway until the robot is balanced.

2) Check the floor is level using leveler otherwise walking may not be stable. See Fz values while the robot standing on the floor. All Fz values should be very close.

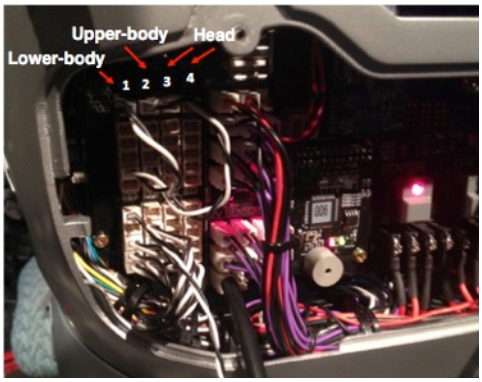
6. Joint Coordinate Systems

: Every joint in Hubo+ follows one common coordinate system shown in the figure below. The sign of each joint angle can be determined from this coordinate frame.



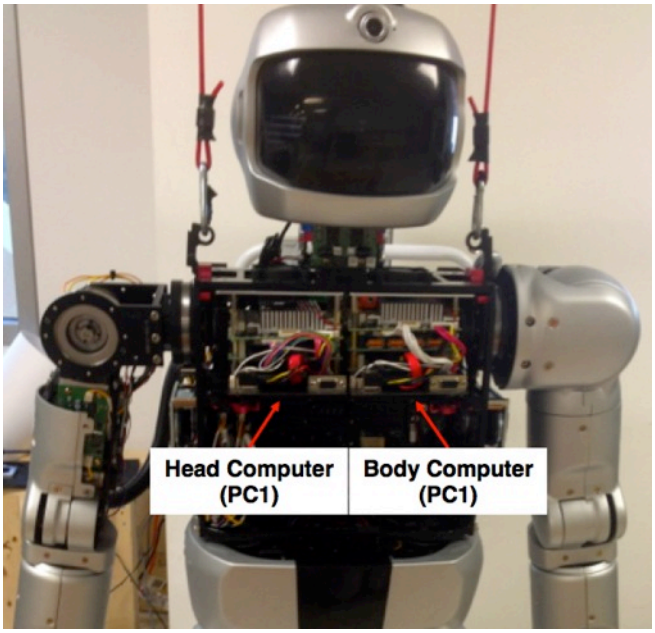
8. Plugging in CAN communication connectors.

: There are four columns of sockets for CAN communication. The first column is for lower-body and the second column is for upper-body.



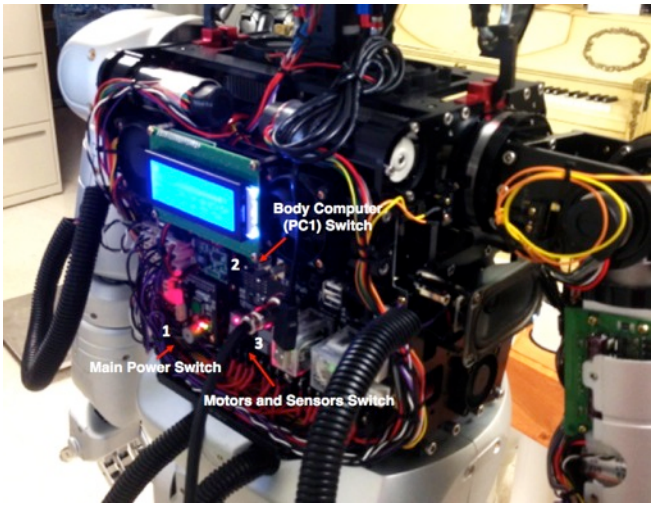
9. Computers in the Robot

: There are two PCs inside the robot. As shown in the figure below, the computer placed in the right is PC1 (body computer). Most of the programs (RTX, Rainbow) are running on this pc. The computer placed in the left is PC2 (head computer) and it can be used for vision processing and some other computations. Both computers can be connected via serial communication or UDP.



10. Power On Switches

: When starting up the robot, power on switches in the back following this order.



10. Codes for testing written during the training

Task**:** Move both elbow joints up and down with different speeds.

1) In the main function of “Core.cpp” file

```
case ARM_MOVE_TEST: // by Andy and Roy

                        //RtWprintf(L"\n Test of Arm Movement!\n");
                        //pSharedMemory->CommandFlag = NO_ACT;
                        if(Joint[REB].MoveFlag == false)
                        {
```

```
                                SetMoveJointAngle(REB,-60.0f,2000.0f,0x01);
                                SetMoveJointAngle(LEB,-60.0f,1000.0f,0x01);
                                RtWprintf(L"\n Test of Arm Movement!\n");
                                pSharedMemory->MotorControlMode =
CTRLMODE_POSITION_CONTROL_WIN;
                                pSharedMemory->CommandFlag = NO_ACT;
                        }
                        break;

case UNDO_ARM_MOVE_TEST: // by Andy and Roy

                        //RtWprintf(L"\n Test of Arm Movement!\n");
                        //pSharedMemory->CommandFlag = NO_ACT;
                        if(Joint[REB].MoveFlag == false)
                        {
                                SetMoveJointAngle(REB,-40.0f,1000.0f,0x01);
                                SetMoveJointAngle(LEB,-40.0f,1000.0f,0x01);
                                RtWprintf(L"\n Initial pose!\n");
                                pSharedMemory->MotorControlMode =
CTRLMODE_POSITION_CONTROL_WIN;
                                pSharedMemory->CommandFlag = NO_ACT;
                        }
                        break;
```

2) In “CommonDefinition.h” file

```
// commands from win32 to RTX ------------------------------------------------------------//
typedef enum
{
        NO_ACT,
        EXIT_PROGRAM,
                :
                :
                :
        ARM_MOVE_TEST,          // by Andy and Roy
        UNDO_ARM_MOVE_TEST      // by Andy and Roy
} _COMMAND_FLAG;
```

3) In “UserDlg.cpp” file

```
void CUserDlg::OnArmMoveTest()
{
        /* OnArmMoveTest()
        **
        ** We basically test robot arm motion here
        **
        ** Author:   Andy and Roy
        ** Created:  5/20/2012
        **
        */
```

```
        //prinf("11111111");
        CString strTemp;
        if(pSharedMemory->CommandFlag == NO_ACT)                    // Make sure
no other command is running
        {
                // Read the text off of the button to determine whether to go to or return
from the dance pose
                GetDlgItem(IDC_ARM_MOVE_TEST)->GetWindowText(strTemp);
                if(strTemp == "ArmMoveTest")
                {
                        GetDlgItem(IDC_ARM_MOVE_TEST)->SetWindowText("Init"); //
Change the command to end the dance pose
                        pSharedMemory->CommandFlag = ARM_MOVE_TEST;          // Run
the code in the case for DANCE_POSE_BASE command
                }
                else
                {
                        GetDlgItem(IDC_ARM_MOVE_TEST)-
>SetWindowText("ArmMoveTest"); // Change the command to go to the dance pose
                        pSharedMemory->CommandFlag = UNDO_ARM_MOVE_TEST;//
Return to walk ready after dance
                }
        }

}
```

11. Some useful codes from Kiwon

1) Smooth Movement

```
Joint[WST].MoveFlag = true; Joint[WST].RefAngleToGo = pSharedMemory->
npos[motion_row_num][0];
 Joint[LHY].MoveFlag = true; Joint[LHY].RefAngleToGo = pSharedMemory->
npos[motion_row_num][1];
 Joint[LHR].MoveFlag = true; Joint[LHR].RefAngleToGo = pSharedMemory->
npos[motion_row_num][2];
 Joint[LHP].MoveFlag = true; Joint[LHP].RefAngleToGo = pSharedMemory->
npos[motion_row_num][3];
....
pSharedMemory->MotorControlMode = CTRLMODE_POSITION_CONTROL_WIN;
```

2) Get FT sensor value

```
if(Joint[LSP].MoveFlag == false){
   filter_i = filter_i+1;
 if (filter_i > (filterlength-1))
 {filter_i = 0;}
  meanfilter[filter_i] = FTSensor[RWFT].Fz;
```

```
  }
```

3) Check Movement is over

```
if(DrexelSequence == 0x01)
 {
  if(Joint[LSP].MoveFlag == false) DrexelSequence = 0x02;
 }
```

4) Change Walking Parameter

```
else if(DrexelSequence == 0x05)
   {
      DrexelSequence = 0x06;
  pSharedMemory->JW_temp[0] = 0.0605f; // sway
  pSharedMemory->JW_temp[3] = 60.0f; // hold time
  pSharedMemory->JW_temp[6] = 0.05f/2.0f; // forward step length
  pSharedMemory->JW_temp[9] = 0.0f/2.0f; // side step length
  pSharedMemory->JW_temp[12] = 0.0f/2.0f; // rotation angle
  pSharedMemory->JW_temp[15] = 40.0f/2.0f; // DSP time
  pSharedMemory->JW_temp[16] = 800.0f;  // step period
  pSharedMemory->JW_temp[17] = -1; // step count
  pSharedMemory->JW_temp[18] = 1.25f; // start sway
  pSharedMemory->JW_temp[19] = 1.2f; // stop sway
  pSharedMemory->JW_temp[1] = 1.0f;
  pSharedMemory->JW_temp[2] = 1.0f;
  pSharedMemory->JW_temp[4] = 1.0f;
  pSharedMemory->JW_temp[5] = 1.0f;
  pSharedMemory->JW_temp[7] = 0.0f;
  pSharedMemory->JW_temp[8] = 0.0f;
  pSharedMemory->JW_temp[10] = 1.0f;
  pSharedMemory->JW_temp[11] = 1.0f;
  pSharedMemory->JW_temp[13] = 1.0f;
  pSharedMemory->JW_temp[14] = 1.0f;
  pSharedMemory->JW_temp[20] = 60.0f; // 60.0f;
  pSharedMemory->JW_temp[21] = 1.0f;
  pSharedMemory->JW_temp[22] = 1.0f;
  pSharedMemory->JW_temp[23] = 0.0f;
  pSharedMemory->JW_temp[24] = 0.0f;
  pSharedMemory->JW_temp[25] = 0.0f;
  pSharedMemory->JW_temp[26] = 0.0f;
  pSharedMemory->CommandFlag = GOTO_FORWARD;
 }
```

5) STOP

```
else if(DrexelSequence == 0x07)
   {
  DrexelSequence = 0x08;
  pSharedMemory->CommandFlag = STOP_WALKING;
   }
```

6) Check if walking is stopped

```
else if(DrexelSequence == 0x08)
   {
  if(WalkingPhaseNext[Z] == WALK_READY)
 {
  if(WalkingInfo[RIGHT][Z].MoveFlag == false)
  {
   if(DrexelOffEnabled == 0x01)
   {
    DrexelSequence = 0x09;
    DrexelOffEnabled = 0x00;
   }
   else DrexelSequence = 0x02;
   gogo = 0; stopstop=0;
   SetMoveJointAngle(RSP, 0.0f, 1000.0f, 0x01);
   SetMoveJointAngle(LSP, 0.0f, 1000.0f, 0x01);
   SetMoveJointAngle(REB, -70.0f, 1000.0f, 0x01);
        SetMoveJointAngle(LEB, -70.0f, 1000.0f, 0x01);
   /*SetMoveJointAngle(RSP, 30.6510f, 1000.0f, 0x00);
   SetMoveJointAngle(LSP, 30.6510f, 1000.0f, 0x00);
   SetMoveJointAngle(REB, -38.6822f, 1000.0f, 0x00);
        SetMoveJointAngle(LEB, -38.6822f, 1000.0f, 0x00);*/
            /*SetMoveJointAngle(RSP, 0.0f, 1000.0f, 0x01);
   SetMoveJointAngle(LSP, 0.0f, 1000.0f, 0x01);
   SetMoveJointAngle(REB, -70.0f, 1000.0f, 0x01);
        SetMoveJointAngle(LEB, -70.0f, 1000.0f, 0x01);*/
   pSharedMemory->MotorControlMode = CTRLMODE_POSITION_CONTROL_WIN;
  }
 }
   }
```

The following documents are prepared by Drexel University.

1. **Hubo+ Operating Instructions**
2. **Hubo II CAN Message Protocols**
3. **Programming with Windows Visual C++**
4. **Hubo-i: Hubo Protocol commander**

Hubo+ Operating Instructions

May 1, 2012

1 Connection

To remote desktop into Hubo, use the following information to connect via metHubo:

Robot	IP	username	password
Hubo+ 1 PC1	192.168.0.51	hubo	1234
Hubo+ 2 PC1	192.168.0.52	hubo2	1234
Hubo+ 3 PC1	192.168.0.53	hubo2	1234
Hubo+ 4 PC1	192.168.0.54	hubo	1234
Hubo+ 5 PC1	192.168.0.55	hubo	1234
Hubo+ 6 PC1	192.168.0.56	hubo	1234

2 Robot Setup

2.1 Power On

1. Put a battery in the robot (calibration was done with the weight of the battery present)
2. Plug in and turn on power supply
3. Turn off the E-stop (allow power)
4. Turn on Hubo via toggle switch on back. This switch is under a small PCB next to the switch labeled SENSOR.
5. Turn on body computer via toggle switch labeled PC1
6. Remote desktop into PC1

2.2 Run Code and Get to Home Position

1. Open My Computer → C: → Rainbow → Rainbow.dsw
   - **Rainbow** and **Rainbow GUI** should be visible in the left panel. If not, make sure you are using the file view tab.
2. Build Rainbow (the RTX side of the program)
   - Right click on **Rainbow** in the left menu and choose Build (Section Only).
   - Make sure the output has no errors.
   - If the output says Debug instead of Release, select Rainbow-RTSS win32 Release from Build → Set Active Configuration, then build again.
3. Build Rainbow GUI (the Windows and GUI side of the program)
   - Right click on **Rainbow GUI** in the left menu and choose Build.
   - Make sure the output has no errors.

- If the output says Debug instead of Release, or if the Rainbow-GUI code is not the active project, select Rainbow-GUI win32 Release from Build → Set Active Configuration, then build again.

4. Turn on sensors and motors via remote (button 2) or switches in back (labeled SENSOR and ON).
5. Click ! to start the program
   - This starts the windows side of the program
6. Turn on RTX (button RTX on)
   - The RTX message window will pop up and indicate that RTX has successfully turned on
   - Wait 3-4 seconds before continuing
7. Load parameters (button Load Param)
   - Make sure they all say ok. You may get errors if you load parameters too soon after starting RTX
8. Make sure the robot is close to its home position
   - arms and legs should be straight and perpendicular to the floor.
   - Feet should be parallel to the floor.
9. Turn on the FETS (FET on).
   - Motors will now be "sticky" or hard to move
10. Turn on controllers (CTRL ON)
    - The main portion of the GUI will now have more tabs and buttons to use
    - Check that the motors are locked into position.
11. Initialize the robot to home position (All Init Pos)
    - Robot will find limits by moving motors and go into it's totally straight home position.
    - Make sure the status is ok: home search results under the Hubo2 Upper Status and Hubo2 Lower Status tabs are all 6.
    - Controller boards will flash the status (hopefully 6) and then resume blinking. If they don't start blinking, shut off the motors and restart this process.
    - Only do this if the home position is already set up correctly. This usually needs to be redone if the robot has an accident or appears to be misaligned.

2.3 Go to Walk Ready Position

1. Go to the Walking and Demo Tab
2. Click Range on button
   - This enables joint limitations to (help) prevent the robot from damaging itself
3. Click Walk Ready
   - Robot will go into a crouch

2.4 Go to stored ZMP Position

1. Click ZMP Init Pos.
   - The robot will load stored ZMP data and move to the saved position

2.5 Sensor Setup

1. While the robot is in the air, press the FT nulling button under the sensor tab.
   - The robot MUST be in the air.
   - Values for L and R My, Mx, and Fz should be close to 0.
   - This initializes the Force Torque sensors to zeros (no force when in the air).
2. Lower robot onto the floor or level surface you will be using.
3. Click on FT Angle Zero
   - This sets the accelerometer angle in ankle to 0
   - R and L Roll and Pitch should be almost 0
4. Click on IMU Nulling
   - This zeros the inertial measurement unit
   - IMU roll should be about 0, if it's not you need to recheck the home position (Section 5)

3 Robot Shutdown

1. Raise the robot back into the air
2. Put the robot back in Home position (click Goto Home)
3. Turn motor controllers off (Ctrl Off)
4. Turn off RTX (RTX Off)
5. Exit the program
6. Power off the sensors and motors (remote button 1 or the OFF switch on the robot's back)
7. Shut down the computer (run shutdown -t 0)
8. End the remote desktop session
9. Power down the robot

4 Demonstrations and Running Code

4.1 Handshake

Don't use this right now. If things aren't exactly right, or the person tries to shake the robot's hand too soon, it could start to spasm.

1. Initial Setup: You have to run C - Grasping first. For some reason this allows the hands to grasp afterwards.
   (a) Select C-Grasping from the third drop-down menu under the Demos section
   (b) Press the Grasp Pos button. Robot will extend its right arm.
   (c) Press Grasp On. Robot will close its hand.
   (d) Press Grasp Off. Robot will open its hand fully.
   (e) Press Init Pos (formerly the Grasp Pos button). Robot will lower its arm.
2. Handshaking.

(a) Select Handshaking from the third drop-down menu under the Demos section

(b) Press the Grasp Pos button. Robot will extend its right arm.

(c) Have someone hold the robot's hand to shake it.

(d) Press Grasp On. Robot will close its hand.

(e) Press Start. Robot will move its arm when the person shakes its hand.

(f) Press Grasp Off. Robot will open its hand fully.

(g) Press Init Pos (formerly the Grasp Pos button). Robot will lower its arm.

Notes: Do NOT press buttons out of order. Just don't do it, the robot could begin to shake or a motor could fail and make an awful noise.

4.2 Walking: random thoughts

When lowering the robot onto an unlevel surface (not the balance plate), lower it quickly to the ground. Lowering it slowly can allow the robot to hit the floor multiple times, causing sensor values to change several times.

To walk, set the step count (eg 3 steps). The button walking set will save changes made to parameters (must click this every time, even if no changes were made) and GO! will have the robot walk. Hubo will not use ZMP balancing while walking, only for 3 seconds to "adjust" after it has finished walking.

Set step count to -1 to keep walking until stop is pressed. For walking forward, start with small step sizes (.05m) but .1m, maybe .15m should be safe.

Sway is important for walking, a too large or too small value will make the robot walk strangely. Smaller sway to walk fast, but changes with step size. The start sway value multiplies the sway for the first step and stop sway value multiplies the sway for the last step.

5 Set the Home Position

After an accident, shipping, or if the robot doesn't seem to be walking properly you need to check the home position. Change parameters on the GUI (Initialize tab) to move joints to correct home position.

1. Follow all steps in section 2.1 and section 2.2
   - If there is no previous set home position (or it is dangerous to the robot) stop after step 10 in section 2.2.
2. Fix the hip yaw.
   - The center of the hip roll motors (front of robot) should line up horizontally. Use a ruler to make sure the legs aren't turned out or in.
3. Fix the knees (both sides)
   - Turn off the ankle and set the robot on the plate.
   - Use the laser leveler to make sure the center of the ankle, knee, and hip joints line up (side view of robot).
4. Fix the hip roll
   - From the front of the robot, use the laser leveler to make sure the ankle motor and hip roll motor are directly in line with each other.
5. Fix the hip pitch
   - Check that the legs are directly beside each other. Use a straightedge, it should make contact with both sides of both legs when lined up horizontally across the front of the legs. (eg. there should be no gap due to one leg being further back).

6. Check Hip roll (again)
    - If the hip pitch has been changed, recheck the hip roll.
7. Set the Ankles
    - Hold the robot so that the center of the ankle, knee, and hip joints line up (side view of robot).
    - Use the Hubo-i to find the encoder values for this position and add it to the stored home position
    - *** We need more detailed instructions for this ***

6 Set the ZMP Values

Find new ZMP values whenever you change the home position or if the robot seems unbalanced.

1. Follow all steps in sections 2.1- 2.3 and 2.5 (skipping section 2.4) using a balance plate whenever the robot is on the ground.
2. Go to the Walking and Demo Tab
3. Click on ZMP init start
    - This will find the center of mass.
    - It uses IMU and FT, so these need to already be set up.
    - Check that the robot's body is actually aligned and straight.
    - Robot will move back and forth slightly, should not move side to side much at all.
4. Check the values in the sensor tab
    - ZMP x and y should be almsot 0
    - R and L My and Mx should be about 0
    - R and L Fz should be equal
    - IMU roll and pitch should be 0
5. Go back to Walking and Demo tab and click ZMP init stop
6. To save the position so it can be used later, click the ZMP init save button. Now this new position will be loaded when ZMP Init Pos. is pressed.

7 Misc. Notes

- The battery doesn't charge when Hubo is plugged in, it needs to be removed to charge.
- Remote: button 1 turns off the sensors and motors, button 2 turns them on.
- Robot needs at about 52 (or more) volts to run properly. The minimum is 48V.

Hubo II CAN Message Protocols

TX Message ID

Motor Command Message ID 0x01
Sensor Command Message ID 0x02
Reference Message ID 0x10 + BNO

RX Message ID

FT Sensor data Message ID 0x40 + SBNO
Tilt Sensor/IMU data Message ID 0x50 + SBNO
Encoder value Message ID 0x60 + BNO
Status Message ID 0x150 + BNO
Board Information Message ID 0x190 + BNO + BOFF
Board Para & Current Message ID 0x1C0 + BNO
(NOTE)

BNO: Board Number
SBNO: Board Number for Sensor Boards. SBNO=BNO-0x2F
BOFF=0 for BNO < 0x30
BOFF=0x80 for BNO >= 0x30

CAN packet coding example

Mail Box ID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8
MesgID	DL	4-Byte-INT				2-Byte-INT		1-BYTE-INT	-

DL: Data length
4_Byte_INT = Byte0 | (Byte1<<8) | (Byte1<<16) | (Byte1<<24)
2_Byte_INT = Byte4 | (Byte5<<8)
1_Byte_INT = Byte6

Board Description Table

Board Name	Board No (BNO)	SBNO	Description	No of Axis	Board Type	Firmware No
JMC0	0x00(0)	-	Right Hip Yaw/Right Hip Roll	2	2	1
JMC1	0x01(1)	-	Right Hip Pitch	1	1	1
JMC2	0x02(2)	-	Right Knee	1	1	1
JMC3	0x03(3)	-	Right Ankle Pitch/Right Ankle Roll	2	2	1
JMC4	0x04(4)	-	Left Hip Yaw/Left Hip Roll	2	2	1
JMC5	0x05(5)	-	Left Hip Pitch	1	1	1
JMC6	0x06(6)	-	Left Knee	1	1	1
JMC7	0x07(7)	-	Left Ankle Pitch/Left Ankle Roll	2	2	1
JMC8	0x08(8)	-	Right Shoulder Pitch/Right Shoulder Roll	2	2	1
JMC9	0x09(9)	-	Right Shoulder Yaw/Right Elbow	2	2	1
JMC10	0x0A(10)	-	Left Shoulder Pitch/Left Shoulder Roll	2	2	1
JMC11	0x0B(11)	-	Left Shoulder Yaw/Left Elbow	2	2	1
JMC12	0x0C(12)	-	extra	-	-	-
JMC13	0x0D(13)	-	extra	-	-	-
JMC14	0x0E(14)	-	Smart Power Controller	-	9	4
JMC15	0x0F(15)	-	extra	-	-	-
						-
EJMC0	0x20(32)	-	Right Wrist Yaw/Right Pitch	2	3	1
EJMC1	0x21(33)	-	Left Wrist Yaw/Left Wrist Pitch	2	3	1
EJMC2	0x22(34)	-	Nect Yaw/Neck 1/Neck 2	3	3	1
EJMC3	0x23(35)	-	Waist	1	1	1
EJMC4	0x24(36)	-	Right Finger0/Finger1/Finger2/Finger3/Finger4	5	5	1
EJMC5	0x25(37)	-	Left Finger0/Finger1/Finger2/Finger3/Finger4	5	5	1
EJMC6	0x26(38)	-	Extra	-	-	-
				-	-	-
FT0	0x30(48)	1	Right Foot F/T sensor	-	6	2
FT1	0x31(49)	2	Left Foot F/T Sensor	-	6	2
IMU0	0x32(50)	3	IMU sensor 0	-	7	3
IMU1	0x33(51)	4	IMU sensor 1	-	7	3
IMU2	0x34(52)	5	IMU sensor 2	-	7	3
FT3	0x35(53)	6	Right Wrist F/T Sensor	-	8	5
FT4	0x36(54)	7	Left Wrist F/T Sensor	-	8	5

1. Motor Control Board

Command Message for Motor Boards (Message ID=0x01)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Set & Req. Board Info.[1)]	BNO	0x01	CANR	-	-	-	-	-
Req. Board Status[2)]	BNO	0x02	-	-	-	-	-	-
Req. Encoder Position[3)]	BNO	0x03	FES	-	-	-	-	-
Req. Current[4)]	BNO	0x04	-	-	-	-	-	-
Reset Encoder to zero	BNO	0x06	CH	-	-	-	-	-
Set position gain0	BNO	0x07	Kp0		Ki0		Kd0	
Set position gain1	BNO	0x08	Kp1		Ki1		Kd1	
Set current gain0	BNO	0x09	KPt0		KDt0		KF0	
Set current gain1	BNO	0x0A	KPt1		KDt1		KF1	
Turn ON/OFF Driver(HIP)	BNO	0x0B	HIP_EN	-	-	-	-	-
Open loop PWM(1, 2CH)	BNO	0x0D	PUL_ON	DIR0	DUTY0	DIR1	DUTY1	-
for Finger(5CH)	BNO	0x0D	PUL_ON	D_DT0	D_DT1	D_DT2	D_DT3	D_DT4
for Neck(3CH)	BNO	0x0D	PUL_ON	D_DT0	D_DT1	D_DT2	-	-
Turn ON Controller	BNO	0x0E	-	-	-	-	-	-
Turn OFF Controller	BNO	0x0F	-	-	-	-	-	-
Set Control mode	BNO	0x10	FBC	-	-	-	-	-
Go to Home Offset	BNO	0x11	CH&D	SDR	H_OFFSET(확인)			
Set Dead Zone	BNO	0x20+CH	DZone	-	-	-	-	-
Req. Board Parameters[5)]	BNO	0x24	PARM	-	-	-	-	-
Set Home Search Para.	BNO	0x30+CH	SRL	SDR	OFFSET			
Set Encoder Resolution	BNO	0x38+CH	ENC_RE		-	-	-	-
Set Max. Acc.& Vel.	BNO	0x40+CH	MACC		MVEL		-	-
Set Lower Position Limit	BNO	0x50+CH	MPS	MPOS1				-
Set Upper Position Limit	BNO	0x56+CH	MPS	MPOS2				-
Set Home Vel. & Acc.	BNO	0x60+CH	HMA	HMV1	HMV2	SRM	LIMD	-
Set Gain Override	BNO	0x6F	GOVW0	GOVW1	GDUR		-	-
Set New Board Number	BNO	0xF0	NEW_BNO	CANR	-	-	-	-
Set Jam & PWM Sat. lim.	BNO	0xF2	JAM_LIM		PWM_LIM		LIMD	JAMD
Set Error Bound	BNO	0xF3	I_ERR		B_ERR		E_ERR	
Initialize Board	BNO	0xFA	-	-	-	-	-	-

Reference Message for Motor Boards (Message ID=0x10 + BNO)

Description	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
Send Position Reference	REF0				REF1			
for Finger(5CH)	REFR0	REFR1	REFR2	REFR3	REFR4			
for Neck(3CH)	REFR0	REFR1	REFR2					

(NOTE)

1. REF0=Byte0 | (Byte1<<8) | (Byte2<<16) | (Byte3<<24)
   REF1=Byte4 | (Byte5<<8) | (Byte6<<16) | (Byte7<<24)
2. Reference for Finger and Neck is given by differential value of position, REFRx.

Single byte REFRx is coded by 2's Complementary binary. To decode a negative number, (int)VAL = REFRx – 0x100 .

3. If the control mode is "position" the reference value is POSITION in encoder unit.
4. If the control mode is "current", the reference value is Current in 0.01A unit for finger board or 0.05A unit for the other boards.

Return Message From Motor Boards

Description	MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
1) Send Board Info	0x190+BNO + BOFF	CANR	1-	BTY	VER0	VER1	VER2	VER3	-
2) Send Board Status	0x150+BNO	STAT00	STAT01	STAT02	STAT03	STAT10	STAT11	STAT12	STAT13
for Neck & Finger	""	STAT0	STAT1	STAT2	STAT3	STAT4	-	-	-
3) Send Encoder Posi.	0x60+BNO	M0_POS				M1_POS			
for Neck	""	M0_POS		M1_POS		M2_POS			
for Finger, FES=0	""	M0_POR		M2_POR		M2_POS			
for Finger, FES=1	""	M3_POS		M4_POS					
4) Send Current	0x1C0+BNO	B0	B1	B2	B3	B4	B5	B6	B7
5) Send Board Para.	0x1C0+BNO	B0	B1	B2	B3	B4	B5	B6	B7

Default values

BNO	Joint	Limit		PID Gain				Homeset1			Homeset2				Encoder		
		MPOS1	MPOS2	Kp	Kd	Ki	DZ	OFF	HLIM	HLD	HV1	HV2	HMA	SM	ERS	AS	MD
0	RHY	−90000	90000	200	500	0	44	0	20	1	40	50	0.1	0	4000	1	1
	RHR	−150000	120000	200	500	0	44	4500	20	1	40	50	0.1	0	4000	1	0
1	RHP1	−165000	130000	200	500	0	44	−4000	20	0	40	50	0.1	0	4000	1	1
	RHP2																
2	RKN1	−30000	220000	200	500	0	44	−13000	20	1	40	50	0.1	0	4000	1	0
	RKN2																
3	RAP	−170000	170000	200	500	0	44	0	20	1	40	50	0.1	0	4000	1	0
	RAR	−78000	78000	200	500	0	44	12000	20	1	40	50	0.1	0	4000	1	1
4	LHY	−90000	90000	200	500	0	44	0	20	0	40	50	0.1	0	4000	1	1
	LHR	−120000	150000	200	500	0	44	−4500	20	0	40	50	0.1	0	4000	1	0
5	LHP1	−165000	130000	200	500	0	44	−4000	20	0	40	50	0.1	0	4000	1	0
	LHP2																
6	LKN1	−30000	220000	200	500	0	44	−13000	20	1	40	50	0.1	0	4000	1	1
	LKN2																
7	LAP	−170000	170000	200	500	0	44	0	20	1	40	50	0.1	0	4000	1	1
	LAR	−78000	78000	200	500	0	44	−8000	20	0	40	50	0.1	0	4000	1	1
8	RSP	−310000	280000	100	250	0	28	0	20	1	40	50	0.1	0	4000	1	1
	RSR	−180000	10000	100	250	0	28	13333	20	1	40	50	0.1	0	4000	1	1
9	RSY	−150000	100000	100	250	0	28	0	20	1	40	50	0.1	0	4000	1	1
	REB	−190000	0	100	250	0	28	0	20	1	40	50	0.1	0	4000	1	1
10	LSP	−310000	280000	100	250	0	28	0	20	1	40	50	0.1	0	4000	1	0
	LSR	−10000	180000	100	250	0	28	−11666	20	0	40	50	0.1	0	4000	1	1
11	LSY	−100000	150000	100	250	0	28	0	20	0	40	50	0.1	0	4000	1	1
	LEB	−190000	0	100	250	0	28	10000	20	1	40	50	0.1	0	4000	1	0
32	RWY	−100000	180000	2000	5000	0	140	0	100	0	100	100	1	1	4000	1	0
	RWP	−70000	90000	2000	5000	0	140	100000	100	0	100	100	1	2	4000	1	0
33	LWY	−180000	100000	2000	5000	0	140	0	100	1	100	100	1	1	4000	1	0
	LWP	−70000	90000	2000	5000	0	140	100000	100	0	100	100	1	2	4000	1	1
34	NKY	−4000	4000	40	100	0	90	0	30	1	2	2	1	1	128	1	1
	NK1	−1300	1050	40	100	0	90	−1470	30	1	2	2	1	2	128	1	0
	NK2	−1300	1050	40	100	0	90	−1470	30	1	2	2	1	2	128	1	0
35	WST	−170000	170000	200	500	0	44	0	20	0	40	50	0.1	0	4000	1	0
36	RH0	−4800	4400	50	100	0	0	−5000	100	1	20	20	1	2	64	1	0
	RH1	−4800	4400	50	100	0	0	−5000	100	1	20	20	1	2	64	1	1
	RH2	−4800	4400	50	100	0	0	−5000	100	1	20	20	1	2	64	1	1
	RH3	−4800	4400	50	100	0	0	−5000	100	1	20	20	1	2	64	1	0
	RH4	−4800	4400	50	100	0	0	−5000	100	1	20	20	1	2	64	1	0
37	LH0	−4800	4400	50	100	0	0	−5000	100	1	20	20	1	2	64	1	1
	LH1	−4800	4400	50	100	0	0	−5000	100	1	20	20	1	2	64	1	0
	LH2	−4800	4400	50	100	0	0	−5000	100	1	20	20	1	2	64	1	0
	LH3	−4800	4400	50	100	0	0	−5000	100	1	20	20	1	2	64	1	1
	LH4	−4800	4400	50	100	0	0	−5000	100	1	20	20	1	2	64	1	1

MPOS1: MAX_POS1, MPOS2: MAX_POS2, DZ: D_ZONE, OFF: OFFSET, HLIM: Home_Lim, HLD: Lim_dir, HV1: V_max1, HV2: V_max2, HMA: A_max,
SM: Search Mode, ERS: Encoder resolution, AS: Auto scale, MD: Motor Dirction

BNO	Joint	Speed Limits		Jam and Power Saturation Limit			Error Limit	
		Vmax	Amax	JAM_LIM(sec10)	JAMD(%)	PWM_LIM(sec10)	I_ERR	B_ERR
0	RHY	1500 (22500rpm)	20 (	3	50	5	20000	10000
	RHR	1500	20	3	50	5	20000	10000
1	RHP1	1500	20	3	50	5	20000	10000
	RHP2							
2	RKN1	1500	20	3	50	5	20000	10000
	RKN2							
3	RAP	1500	20	3	50	5	20000	10000
	RAR	1500	20	3	50	5	20000	10000
4	LHY	1500	20	3	50	5	20000	10000
	LHR	1500	20	3	50	5	20000	10000
5	LHP1	1500	20	3	50	5	20000	10000
	LHP2							
6	LKN1	1500	20	3	50	5	20000	10000
	LKN2							
7	LAP	1500	20	3	50	5	20000	10000
	LAR	1500	20	3	50	5	20000	10000
8	RSP	1500	20	3	50	5	20000	10000
	RSR	1500	20	3	50	5	20000	10000
9	RSY	1500	20	3	50	5	20000	10000
	REB	1500	20	3	50	5	20000	10000
10	LSP	1500	20	3	50	5	20000	10000
	LSR	1500	20	3	50	5	20000	10000
11	LSY	1500	20	3	20	5	20000	10000
	LEB	1500	20	3	50	5	20000	10000
32	RWY	-100000	180000	2000	5000	0	0	10000
	RWP	-70000	90000	2000	5000	0	100000	10000
33	LWY	-180000	100000	2000	5000	0	0	10000
	LWP	-70000	90000	2000	5000	0	100000	10000
34	NKY	-4000	4000	40	100	0	0	30
	NK1	-1300	1050	40	100	0	-1470	30
	NK2	-1300	1050	40	100	0	-1470	30
35	WST	-170000	170000	200	500	0	0	20
36	RH0	-4800	4400	50	100	0	-5000	100
	RH1	-4800	4400	50	100	0	-5000	100
	RH2	-4800	4400	50	100	0	-5000	100
	RH3	-4800	4400	50	100	0	-5000	100
	RH4	-4800	4400	50	100	0	-5000	100
37	LH0	-4800	4400	50	100	0	-5000	100
	LH1	-4800	4400	50	100	0	-5000	100
	LH2	-4800	4400	50	100	0	-5000	100
	LH3	-4800	4400	50	100	0	-5000	100
	LH4	-4800	4400	50	100	0	-5000	100

Detail Description of Command Message

1. Set and Request Board Information(RBI: 0x01)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x01	CANR	-	-	-	-	-

CANR: CAN rate in msec. Default: 5(ms)

Return:

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x190+BNO + BOFF	CANR	1	BTY	VERSION				-

CANR: Echo CANR to confirm CAN rate.

BTY: Board Type:

1: 1CH – 2 Motor BLDC Board

2: 2CH – 2 Motor BLDC Board

3: 3CH DC motor Board for Neck joint or 2CH DC Board for Wrist

4: 1CH – 1 Motor BLDC Board for Waist

5: 5CH -5 Motor Board for Hand(Finger)

6: F/T Sensor for Foot

7: Firmware for IMU Board

8: F/T Sensor for Wrist

9: Smart Power Control Board

VERSION is consisted of 7 digit decimal number as below:

D6	D5	D4	D3	D2	D1	D0
Firmware No	Version code					

Firmware No:

1: Firmware for Motor Board

2: Firmware for F/T Sensor for Foot

3: Firmware for IMU Board

4: Firmware for Smart Power Board

5: Firmware for F/T Sensor for Wrist

Version code:

D5D4: yy

D3D2: mm

D1D0: dd

2. Request Board Status and Error Flags (RBS: 0x02)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x02	-	-	-	-	-	-

Action: Request the Board to send the Status and Error flags. The Board automatically sends Status and Error flags without request if any change of the status is detected.

Return:

For 1CH and 2CH Board:

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x150+BNO	STAT00	STAT01	STAT02	STAT03	STAT10	STAT11	STAT12	STAT13

STATx0

b7	b6	b5	b4	b3	b2	b1	b0
HMEx				LIMx	MODx	RUNx	HIPx

HIPx: 1: Motor x driver ON, 0: OFF

RUNx: 1: Motor x Controller ON, 0:OFF

MODx: 1: Motor x Current Feedback mode, 0: Position Feedback mode

MIMx: 1: Motor x Limit switch ON, 0: Limit switch OFF

HME: See 16.

STATx1

b7	b6	b5	b4	b3	b2	b1	b0
-	MO1x	MO0x	FLTx	ENCx	BIGx	PWMx	JAMx

JAMx: 1: Motor x JAM detected, 0: Normal

PWMx: 1: Motor x PWM saturation detected, 0: Normal

BIGx: 1: Motor x Position error is bigger than BERR, 0: Normal

ENCx: 1: Motor x Encoder failure detected, 0: Normal

FLTx: 1: Motor x Fault signal from motor driver x detected, 0: Normal

MO0: 1: Motor0 fail for type 1 board. 0: Normal

MO1: 1: Motor1 fail for type 1 board. 0: Normal

STATx2

b7	b6	b5	b4	b3	b2	b1	b0
TPx		ABSx	TMPx	ACCx	VELx	PS2x	PS1x

PS1x: 1: Motor x Lower limit error, 0: Normal

PS2x: 1: Motor x Upper limit error, 0: Normal

VELx: 1: Motor x Over velocity error, 0: Normal

ACCx: 1: Motor x Over acceleration error, 0: Normal

TMPx: 1: Motor x Over temperature error, 0: Normal

TPx: Reserved for Motor x

STATx3: Reserved for Motor x

For 5CH (Finger) and 3CH (Neck) Boards:

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x150+BNO	STAT0	STAT1	STAT2	STAT3	STAT4	STATB	-	-

STATx

b7	b6	b5	b4	b3	b2	b1	b0
ENCx	BIGx	PWMx	JAMx	LIMx	MODx	RUNx	HIPx

HIPx: 1: Motor x driver ON, 0: OFF

RUNx: 1: Motor x Controller ON, 0:OFF

MODx: 1: Motor x Current Feedback mode, 0: Position Feedback mode

LIMx: 1: Motor x Limit switch ON, 0: Limit switch OFF

JAMx: 1: Motor x JAM detected, 0: Normal

PWMx: 1: Motor x PWM saturation detected, 0: Normal

BIGx: 1: Motor x Position error is bigger than BERR, 0: Normal

ENCx: 1: Motor x Encoder failure detected, 0: Normal

STATB

b7	b6	b5	b4	b3	b2	b1	b0
SPA	SPA	SPA	SPA	SPA	SPA	SPA	-

3. Request Encoder Position (REP: 0x03)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x03	FES	-	-	-	-	-

FES: Finger Position Send flag. FES is ignored for other case than finger.

0: Send M0_POS, M1_POS, M2_POS for finger position

1: Send M3_POS, M4_POS for finger position

Return:

For 1CH and 1CH Board:

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x60+BNO	M0_POS				M1_POS			

For 3CH neck board:

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x60+BNO	M0_POS		M1_POS		M2_POS		-	-

For 5CH finger board:

FES=0:

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x60+BNO	M0_POS		M1_POS		M2_POS		-	-

FES=1:

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x60+BNO	M3_POS		M4_POS		-	-	-	-

4. Request Current value (RCU: 0x04)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x04	-	-	-	-	-	-

Retrun:

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x1C0+BNO	B0	B1	B2	B3	B4	B5	B6	B7

For Board Type 1(CH0 only), 2, 4, 3(Wrist):

B0 = Motor0_CUR>>2

B1 = Motor1_CUR>>2

B2 = TEMP>>2

B3 = (Motor0_CUR&0x03) | (Motor0_CUR&0x03)<<2 | (TEMP&0x03)<<4

B4 = Motor0_JUL>>2

B5 = Motor0_JUL>>2

B6 = SPARE>>2

B7 = (Motor0_JUL&0x03) | (Motor0_JUL&0x03)<<2 | (SPARE&0x03)<<4

(NOTE: Motorx_CUR: Motorx Current in 10mA unit. Actual current is calculated by Ampare = Motorx_CUR/100.

Motorx_JUL: Heat generated by Motor x in Joule.

For Board Type 3(Neck)

B0 ~ B3: same as above

B4 = Motor2_CUR>>2

B5 = Motor2_CUR*0x03

(NOTE: Motorx_CUR: Motorx Current in 4mA unit. Actual current is calculated by Ampare = Motorx_CUR/250.

For Board Type 5(Finger)

B0 = Motor0_CUR>>2

B1 = Motor1_CUR>>2

B2 = Motor2_CUR>>2

B3 = (Motor0_CUR&0x03) | (Motor0_CUR&0x03)<<2 | (Motor2_CUR&0x03)<<4

B4 = Motor3_CUR>>2

B5 = Motor4_CUR>>2

B6 = SPARE>>2

B7 = (Motor3_CUR&0x03) | (Motor4_CUR&0x03)<<2 | (SPARE&0x03)<<4

(NOTE: Motorx_CUR: Motorx Current in 1mA unit. Actual current is calculated by Ampare = Motorx_CUR/1000.

5. Reset Encoder to Zero (REZ: 0x06)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x06	CH	-	-	-	-	-

CH: Channel No.

CH= 0 ~ 4 according to the board.

CH= 0xF selects ALL Channel

Action:

1. Set encoder(s) to Zero.
2. Initialize internal parameters.
3. Reset Fault and Error Flags.

6. Set Motor Position Gain 0 (SMG0: 0x07)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x07	Kp0		Ki0		Kd0	

Sets Channel 0's PID gains.

For 3CH-Neck-Board and 5CH-Finger-Board this command sets ALL Channels' PID gains.

7. Set Motor Position Gain 1 (SMG1: 0x08)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x08	Kp1		Ki1		Kd1	

Sets Channel 1's PID gains.

8. Set Motor Current Gain 0 (SMC0: 0x09)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x09	Kpt0		Kdt0		Kf0	

Sets Channel 0's PI and filter gains.

For 3CH-Neck-Board and 5CH-Finger-Board this command sets ALL Channels' PID gains.

9. Set Motor Current Gain 1 (SMC1: 0x0A)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x0A	Kpt1		Kdt1		Kf1	

Sets Channel 1's PI and filter gains.

10. Motor Driver Enable/Disable (MDE: 0x0B)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x0B	HIP_EN	-	-	-	-	-

HIP_EN

1: Set HIP_EN bit enables Motor driver for all channels. Also DISABLES position feedback.

0: Clear HIP_EN bit disables Motor driver for all channels.

11. Reserved (0x0C)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x0C		-	-	-	-	-

12. Open loop PWM duty command in Percent(%) (PDU: 0x0D)

For 1CH and 2CH boards

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x0D	PUL_ON	DIR0	DUTY0	DIR1	DUTY1	B4

For 5CH boards

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x0D	PUL_ON	D_DT0	D_DT1	D_DT2	D_DT3	D_DT4

For 3CH boards

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x0D	PUL_ON	D_DT0	D_DT1	D_DT2	-	-

DIRx: 1: CW, 0:CCW

DUTYx: Percent PWM Duty. -100 < DUTYx < 100

D_DTx:

b7	b6	b5	b4	v3	b2	b1	b0
DIRx	DUTYx						

Action:

1. Disable Motor position feedback.
2. PUL_ON:

   1: Pulse out to run motor in specified PWM duty and direction

   0: Enforce Zero duty to stop motor. Back EMF will break the motor.

13. Turn on the Feedback Controller (CON: 0x0E)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x0E	-	-	-	-	-	-

Action: Activate Feedback Controller. There are two control modes: Position feedback and Current feedback. This command also enables motor driver.

14..Turn off the Feedback Controller (COF: 0x0F)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x0F	-	-	-	-	-	-

Action: De-activate Feedback Controller. This command also disables motor divers.

15..Set Control mode (SCM: 0x10)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x10	FBC	-	-	-	-	-

Action: Select one of two feedback controllers: Position and Current.

FBC:

1: Current Control

0: Position Control

16.FIND the Limit and GO to the Offset (LGO: 0x11)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x11	CH&D	HO0H	HO0L	HO1H	HO1L	HH01

CH&D

b7	b6	b5	b4	b3	b2	b1	b0
CH				x	x	/DT	SDR

CH: Channel number, CH=0xF selects ALL channel

/DT:

0: Get Offset value from Byte 3 ~ Byte 7. (See 18)

1: Ignore data in Byte 3 ~ Byte 7.

SDR: Search direction. (See 18). If /DT is set, SDR is ignored.

HH01

b7	b6	b5	b4	b3	b2	b1	b0
S0	HH0			S1	HH1		

OFFSET0= Sign(S0)*(HO0L | HO0H<<8 | HH0<<16)

OFFSET1= Sign(S1)*(HO1L | HO1H<<8 | HH1<<16)

Sign(Sx) = 1 if Sx=0

Sign(Sx) = 0 if Sx=1

Action:

1. Find the Limit switch and Index signal to get absolute position.
2. Go to the Offset position from the Index.
3. Set Encoder position value to Zero.
4. Activate position feedback controller.
5. Either Limit switch or Index signal is not found it sets fault bit and deactivates position controller.
6. LED displays the search and go status as HME.
   NOTE: There are three Find-the-Limit modes. See 2.

Return: Returns HME with other status flags. See 2.

HME:

0x1: Start to find the Limit switch.
0x2: Limit switch-ON is found.
0x3: Limit switch-OFF is found and go backward to get ON.
0x4: Index signal detected.
0x5: Start to move to Offset position.
0x6: Arrived at Offset position. -> Success!

0xD: Fail to find backward limit switch signal.
0xE: Fail to detect Index signal
0xF: Fail to find Limit switch

17.Set Dead zone (SDZ: 0x20 + CH)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x20+CH	DZone	-	-	-	-	-

DZone: 0 ~ 255

Action: Set the value of Dead zone to remove FET's PWM null.

18.Set Home search parameter (SHP: 0x30 + CH)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x30+CH	SRL	SDR	OFFSET			

SRL: Search Limit. SRL is maximum number of turns to find limit switch.

SDR: Search direction. (1 or 0)

OFFSET: Offset from Index position.

Action: Set the value for Home search parameter.

19.Set Encoder resolution (SER: 0x38 + CH)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x38+CH	ENC_RE		-	-	-	-

ENC_RE:

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
MDR	AuS	ENC_RES													

ENC_RES: Encoder Resolution

AuS: Auto Scale. AuS=1 sets Auto scale.

MDR: Motor Direction (1: CW, 0:CCW)

20.Set Maximum Acceleration and Velocity(SMAV: 0x40 + CH)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x40+CH	MACC		MVEL		-	-

MACC: Maximum Acceleration.

MVEL: Maximum Velocity.

Action: Set the value for Maximum Velocity and Acceleration.

21.Set Lower position limit (SLP: 0x50 + CH)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x50+CH	MPS	MPOS1				-

MPS

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	-	MPSS	MPSE

MPSS: MPOS1 update (1), MPOS1 ignore (0)

MPSE: Lower Position Limit Enable (1), Lower Position Limit Disable (0),

(NOTE) MPSE is set 1 when power ON.

MPOS1: Lower position limit

Action: Set the value for Lower position limit .

22.Set Upper position limit(SUP: 0x56 + CH)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7

0x01	BNO	0x50+CH	MPS	MPOS2	-

MPS

b7	b6	b5	b4	b3	b2	b1	b0
-	-	-	-	-	-	MPSS	MPSE

MPSS: MPOS2 update (1), MPOS2 ignore (0)

MPSE: Upper Position Limit Enable (1), Upper Position Limit Disable (0),

(NOTE) MPSE is set 1 when power ON.

MPOS1: Lower position limit

Action: Set the value for Upper position limit.

23.Set Home search Acceleration and Velocity(SHV: 0x60 + CH)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x60+CH	HMA	HMV1	HMV2	SRM	LIMD	-

HMA: Home Search Acceleration by 100. Actual acceleration will be calculated by X_acc = HAM/100.

HMV1: Maximum velocity to reach Limit switch.

HMV2: Maximum velocity to reach Offset position

SRM: Search Mode

0: Limit switch & Index

1: Limit switch only

2: No limit switch. LIMD is used to detect jam at the mechanical limit.

LIMD: PWM duty(%) for mechanical limit detection. This value will be used for Limit search mode 2(SRM=2). LIMD can be set by SJP command

Action: Set the values.

24.Gain override (GOV: 0x6F)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x6F	GOVW0	GOVW1	GDUR		-	-

GOVW0: Percent (%) override value of controller gain for channel 0.

GOVW1: Percent (%) override value of controller gain for channel 1.

GDUR: Duration in msec.

(NOTE:: 100 < GOVWx <100)

Action: Overall gain of the PID position controller changes to GOVW0x in GDUR ms from the current value.

25.Set Board number with NEW_BNO number(SNB: 0xF0)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0xF0	NEW_BNO	CANR	-	-	-	-

NEW_BNO: New Board number number

CANR: CAN rate

Action: Set the board number with NEW_BNO.

26.Set JAM and PWM saturation limit (SJP: 0xF2)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0xF2	JAM_LIM		PWM_LIM		LIMD	JAMD

JAM_LIM: JAM detection time in msec

PWM_LIM: PWM saturation detection time in msec.

LIMD: PWM duty(%) for Limit detection. This value will be used for Limit search mode 2. LIMD can be set by SHV command

JAMD: JAM limit in % duty

27.Set Error Bound (SEB: 0xF3)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0xF3	I_ERR		B_ERR		T_ERR	

I_ERR: Maximum Input difference error.

B_ERR: Maximum error.

T_MAX: Max Temperature Warning Temerature

28.Initialize the Board(IBR: 0xFA)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0xFA	-	-	-	-	-	-

Action: Initialize the board with default parameters.

Return:

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x190+BNO + BOFF	CANR	1	BTY	VERSION				-

Returns after completion of saving the value at the memory

29.Request Parameters (RPA: 0x24)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x24	PARM	-	-	-	-	-

Action: Returns value requested by PARM, where CN is Channel number and
OF=0 for CN=0,1,2
OF=5 for CN=3, 4

Return:

A. PARM=CN*6 + 1 + OF:

MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x1C0 +BNO	Kp		Ki		Kd		EncL	EncH

EncH:

b7	b6	b5	b4	b3	b2	b1	b0
MDR	AuS	EncH1					

ENC_RES = EncL + (EncH1<<8);
AuS: Auto Scale. AuS(1) sets Auto scale.
MDR: Motor Direction (1: CW, 0:CCW)

B. PARM=CN*6 + 2 + OF:

MsgID	Byte0	Byte2	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x1C0 +BNO	DZONE		HSD	HSM	HSL0	HSL1	HO0	HO1

DZONE: Dead zone
HSD: Home Search Direction
HSM: Home Search Mode
Home Search Limit =HSL0 | (HSL1<<8)

C. PARM=CN*6 + 3 + OF:

MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x1C0 +BNO	HO2	HO3	MP10	MP11	MP12	MP13	MP20	MP21

Home Offset = HO0 | (HO1<<8) | (HO2<<16) | (HO3<<24)
Lower Position Limit = MP10 | (MP11<<8) | (MP12<<16) | (MP13<<24)

D. PARM=CN*6 + 4 + OF:

MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x1C0 +BNO	MP22	MP23	MAL	MAH	MVL	MVH	SML	SMH

Upper Position Limit = MP20 | (MP21<<8) | (MP22<<16) | (MP23<<24)

Maximum Acceleration = MAL | (MAH<<8);

Maximum Velocity = MVL | (MVH<<8);

Maximum PWM = SML | (SMH<<8);

E. PARM=CN*6 + 5 + OF:

MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x1C0 +BNO	CLL	CLH	RSRV	RSRV	KPtL	KPtH	KDtL	KDtH

Current Limit = CLL | (CLH<<8);

Kpt = KPtL | (KPtH<<8);

Kdt = KDtL | (KDtH<<8);

F. PARM=CN*6 + 6+ OF:

MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x1C0 +BNO	KFtL	KFtH	PATR2	PATR3	PATR4	PATR5	PATR6	PATR7

KFt = KFtL | (KFtH<<8);

(SUM of Bytex from A to F)&0xFF = 0

G. PARM=20:

MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x1C0 +BNO	BNO	RSRV	CANL	CNAH	BTY	RSRV	HMAL	HMAH

BNO: Board Number

CANL | (CANH<<8): CAN rate

BTY : Board Type

HMAL | (HMAH<<8) : Maximum Acceleration for Home limit search

H. PARM=21

MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x1C0 +BNO	HMV1L	HMV1	HMV2L	HMV2H	JAML	JAMH	PWML	PWMH

HMV1L | (HMV1H<<8) : Maximum Velocity for Home limit search

HMV2L | (HMV2H<<8) : Maximum Velocity to Offset position

JAML | (JAMH<<8): JAM error detection time. unit: 0.1sec.

PWML | (PWMH<<8): PWM saturation error detection time. unit: 0.1sec.

I. PARM=22

MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x1C0 +BNO	LIMD	PWMD	IERRL	IERRH	BERRL	BERRH	EERRL	EERRH

LIMD: PWM duty(%) for Limit detection. This value will be used for Limit search mode 2.

PWMD: PWM duty(%) for JAM detection.

IERRL | (IERRH<<8): Maximum Input difference error.

BERRL | (BERRH<<8): Maximum error.

EERRL | (EERRH<<8): Maximum error. for encoder failure

2. FT Sensor Board

Command Message for Sensor Boards (Message ID=0x01)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Set & Req. Board Info.	BNO	0x01	CANR	-	-	-	-	-
Req. Board Parameters	BNO	0x24	PARM	-	-	-	-	-
Req. to execute NULL	BNO	0x81	EFS	-				
Set FT matrix constant0	BNO	0xA0	SFT00		SFT01		SFT02	
Set FT matrix constant1	BNO	0xA1	SFT10		SFT11		SFT12	
Set FT matrix constant2	BNO	0xA2	SFT20		SFT21		SFT22	
Set Inclino. scale factor	BNO	0xA5	SIF0		SIF1		SIF2	
Set Board Number(BNO) & filter freq.	BNO	0xA8	NEW_BNO	FREQ10		-	-	-
Initialize Board	BNO	0xFA	0xAA	-	-	-	-	-

Read Message for Sensor Boards (Message ID=0x02)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Req. FT[1)] and Tilt in digit[2)]	SBNO	0x00		-	-	-	-	-
Req. FT data[1)] w/scale and Tilt[4)] w/scale	SBNO	0x02	-	-	-	-	-	-
Req. FT data[3)] w/scale and Tilt[2)] in digit	SBNO	0x03	-	-	-	-	-	-
Req. FT data[1)] in digit and Tilt[4)] w/scale	SBNO	0x04	-	-	-	-	-	-
Req. FT data in digit [1)]	SBNO	0x11	-	-	-	-	-	-
Req. FT data w/scale [3)]	SBNO	0x12	-	-	-	-	-	-
Req. Tilt data in digit[2)]	SBNO	0x21	-	-	-	-	-	-
Req. Tilt data/ scale[4)]	SBNO	0x22	-	-	-	-	-	-
Req. Gyro & Temp data[5)]	SBNO	0x13						

SBNO: If SBNO=0xFF all sensor boards accept the command.

Return Message from Sensor Boards (SBNO = BNO – 0x2F)

Description	MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
1) Send FT data	0x40+SBNO + BOFF	MX		MY		FZ		-	-
2) Send Tilt data	0x50+SBNO	SX		SY		SZ		-	-
3) Send FT data	0x40+SBNO	Mx100		My100		Fz10		-	-
4) Send Tilt data	0x50+SBNO	Sx100		Sy100		Gz100		-	-
5) Send Gyro-rate & Temperature	0x50+SBNO	GZL		Temp10		-	-	-	-

Mx = Mx100/100: Mx is a Moment x Nm

My = My1000/100: My is a Moment y Nm

Fz = Fz10/10: Fz is a Force z in N.

Sx = Sx100/100 Sx is a Acceleration in m/sec^2

Sy = Sy100/100 Sy is a Acceleration in m/sec^2

Gz = Gz100/100 Gz is a Gravitational Acceleration in g ($9.8m/sec^2$)

SX, SY, SZ are non-scaled digit data from 3-axis Accelerometer.

MX, MY, FZ are non-scaled digit data from 3-axis F/T sensor.

Temp = Temp10/10 : Temperature in degree C.

$$\begin{Bmatrix} Mx \\ My \\ Fz \end{Bmatrix} = \begin{pmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.001 \end{pmatrix} * \begin{bmatrix} SFT00 & SFT01 & SFT02 \\ SFT10 & SFT11 & SFT12 \\ SFT20 & SFT21 & SFT22 \end{bmatrix} * \begin{Bmatrix} MX \\ MY \\ FZ \end{Bmatrix}$$

Sx100 = SIF0 * SX*1e-3

Sy100 = SIF1 * SY*1e-3

Gz100 = SZ/ SIF2 *1e3

Default Value of SFTxy

	Foot Sensor			Wrist Sensor		
	SFTx0	SFTx1	SFTx0	SFTx1	SFTx2	SFTx2
x=0	-250	0	0	12	0	0
x=1	0	-250	0	0	-12	0
x=2	0	0	-3000	0	0	-250

Default Value of SIFx

SIF0	SIF1	SIF2
600	600	8000

Detail Description of Command Message

1. Set and Request Board Information(RBI: 0x01)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x01	CANR	-	-	-	-	-

CANR: CAN rate in msec. Default: 5(ms)

Return:

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x190+BNO + BOFF	CANR	1	BTY	VERSION				-

BTY, VERSION: See Motor Board section for details.

2. Request to execute NULL the sensors (RNL: 0x81)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x81	EFS	-	-	-	-	-

EFS: 0x00 NULL F/T sensors

: 0x02 NULL Inclinometers

3. Set FT sensor Matrix coefficient0 (SFC0: 0xA0)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0xA0	SFT00		SFT01		SFT02	

4. Set FT sensor Matrix coefficient1 (SFC1: 0xA1)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0xA1	SFT10		SFT11		SFT12	

5. Set FT sensor Matrix coefficient2 (SFC2: 0xA2)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0xA2	SFT20		SFT21		SFT22	

SFT 00 ~ SFT22: Set FT sensor coefficient matrix

Return:

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x190+BNO + BOFF	CANR	1	BTY	VERSION				-

Returns after completion of saving the value at the memory

6. Set Inclinometer(Accelerometer) scale factor (SAS: 0xA5)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0xA5	SIF0		SIF1		SIF2	

(NOTE)

Sx100 = SIF0 * SX*1e-3

Sy100= SIF1 * SY*1e-3

Gz100 = SZ/ SIF2 *1e3

Return: same as above.

7. Set Board Number and Filter frequency (SNBF: 0xA8)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0xA8	NEW_BNO	FREQ10		-	-	-

NEW_BNO: Set the Board number with NEW_BNO.

FREQ = FREQ/10: FREQ is a cut off frequency of the first order Low pass filter.

8. Initialize with default value(IDF: 0xFA)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0xFA	0xAA	-	-	-	-	-

Action: Initialize the Board.

Return: Same as above.

9. Request Parameters (RPA: 0x24)

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0x24	PARM	-	-	-	-	-

Action: Returns value requested by PARM=CN*6+1+OF, where CN is Channel number.

OF=0 for CN=0,1,2

OF=5 for CN=3, 4

Return:

A. PARM=1:

MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x1C0 +BNO	SFT00		SFT01		SFT02		FREQ	

B. PARM=2:

MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x1C0 +BNO	SFT10		SFT11		SFT12		SPARE	

C. PARM=3:

MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x1C0 +BNO	SFT20		SFT21		SFT22		SPARE	

FREQ: Set FT sensor coefficient matrix

$$\begin{Bmatrix} Mx \\ My \\ Fz \end{Bmatrix} = \begin{pmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.001 \end{pmatrix} * \begin{bmatrix} SFT00 & SFT01 & SFT02 \\ SFT10 & SFT11 & SFT12 \\ SFT20 & SFT21 & SFT22 \end{bmatrix} * \begin{Bmatrix} MX \\ MY \\ FZ \end{Bmatrix}$$

(NOTE)

1. MX, MY, FZ are measured in digit value from A/D converter. Mx and My are moment in Nm, and Fz is force in N.
2. Freq = FEEQ/10: Freq is cut off frequency of 1st order low pass filter.

D. PARM=4:

MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x1C0 +BNO	SIF0		SIF1		SIF2		SPARE	

SIF0: Scale factor for Accelerometer SX

SIF0: Scale factor for Accelerometer SY

SIF0: Scale factor for Accelerometer SZ

(NOTE)

Sx = SIF0 * SX*1e-5 (degree)

Sy = SIF1 * SY*1e-5 (degree)

Sz = (SZ –SIF2)/ SIF2 (g/g)

SX, SY, SZ are measured in digit value from Accelerometer. Sx and Sy are inclined angle and Sz is g multiple in z-direction.

3. Power Control Board

Command Message for Power Control Board (Message ID=0x01)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Set & Req. Board Info.[1)]	BNO	0x01	CANR	-	-	-	-	-
Set Switch function	BNO	0x81	SFUNC	-	-	-	-	-
Request Alarm	BNO	0x82	ALRM					
Request Beep	BNO	0x83	BDUR					
Req. Voltage and current	BNO	0xE0	-	-	-	-	-	-
Req. Time and Status	BNO	0xE1	-	-	-	-	-	-

1. Set and request Board Information(RBI: 0x01)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Set & Req. Board Info.[1)]	BNO	0x01	CANR	-	-	-	-	-

CANR: CAN rate in msec. Default: 5(ms)

Return:

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x190+BNO + BOFF	CANR	-	BTY	VERSION				-

BTY, VERSION: See Motor Board section for details.

2. Set Switch function(SSF: 0x81)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Set & Req. Board Info.[1)]	BNO	0x81	SFUNC	-	-	-	-	-

SFUNC

0x00: SEC_TIME reset to Zero
0x01: Set 12VSEN and LEDR -> Turn On 12V for sensor.
0x02: Clear 12VSEN and LEDR -> Turn Off 12V for sensor.
0x04: Set M_BTN_ON -> Execute power on sequence to turn on 48V.
0x05: Set M_BTN_OFF -> Execute power off sequence to turn off 48V
0x07 Set BEEPF -> Beeper Enable
0x08 Clear BEEPF and Turn off Beep. -> Beeper Disable
0x0A: Set TGL_PC1 -> Turn on PC1.
0x0B Clear TRL_PC1 -> Turn off PC1.
0x0C: Set TGL_PC2 -> Turn on PC2.
0x0D Clear TRL_PC2 -> Turn off PC2.

3. Request Alarm(ALM: 0x82)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Set & Req. Board Info.[1)]	BNO	0x82	ALRM	-	-	-	-	-

ALRM

0x00: Alarm OFF

0x01: Alarm sound 1, Beeper Enable

0x02: Alarm sound 2, Beeper Enable

0x03: Alarm sound 3, Beeper Enable

0x04: Alarm sound 4, Beeper Enable

4. Request Alarm(ALM: 0x82)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Set & Req. Board Info.[1)]	BNO	0x82	BDUR	-	-	-	-	-

BDUR: Beeps for BDUR*0.1 second.

5. Request voltage and current(RVC: 0xE0)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Req. Voltage and Current	BNO	0xE0	-	-	-	-	-	-

Return

Description	MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
Send Voltage and Current	0x60+BNO	VOLT100		AMP100		WATT100		-	-

BNO=0x14

VOLT = VOLT100/100: Main voltage (48V) measured in V.

CURR= AMP100/100: Main power (48V) current in Ampare.

WATT= WATT100/10: Accumulated Power in WH.

6. Request Time and Status information(RTD: 0xE1)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Req. Time & status info.	BNO	0xE1	-	-	-	-	-	-

Return

Description	MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
Send Time	0x60+BNO	SMF	0x00	SEC_TIME		-	-	-	-

SMF

b7	b6	b5	b4	b3	b2	b1	b0
SW_12_SEN	SW_12_DR	SW_48_O	SW_48_R	SW_PC1	SW_PC2	BEEPF	BATT

BATT:	1: Battery operated,	0: External Power operated
BEEPF:	1: Beeper activated,	0: Beeper disabled
SW_PC2:	1: PC2 is ON	0: OFF
SW_PC1:	1: PC1 is ON	0: OFF
SW_48_R	1: 48V Pre-switch is ON.	0: OFF.
SW_48_O	1: 48V Main-switch is ON.	0: OFF.
SW_12_DR	1: 12V for motor driver is ON	0: OFF
SW_12_SEN	1: 12V for sensor is ON	0: OFF

SEC_TIME: Time elapsed since power ON or since reset.

4. IMU Board

Command Message for IMU Board (Message ID=0x01)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Set & Req. Board Info.[1)]	BNO	0x01	CANR	-	-	-	-	-
Req. to execute NULL	BNO	0x81	-	-	-	-	-	-
Req. to execute Calib.	BNO	0x82	-	-	-	-	-	-
Req. parameters	BNO	0x24	-	-	-	-	-	-
Set board with NEW_BNO	BNO	0xA8	NEW_BNO	-	-	-	-	-

Read Message for IMU Boards (Message ID=0x02)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Req. Angle and Rate[1)]	SBNO	0x00	1	-	-	-	-	-

Return Message from IMU Boards (SBNO = BNO – 0x2F)

Description	MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
1) Send FT data	0x50+SBNO	ANGL_X		ANGL_Y		RATE_X		RATE_Y-	

Detail Description of Command Message

1. Set and request Board Information(RBI: 0x01)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Set & Req. Board Info.[1)]	BNO	0x01	CANR	-	-	-	-	-

CANR: CAN rate in msec. Default: 5(ms)

Return:

MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x190+BNO + BOFF	CANR	-	BTY	VERSION				-

BTY, VERSION: See Motor Board section for details.

2. Request to Execute the sensor NULL(RNL: 0x81)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Set & Req. Board Info.[1)]	BNO	0x81		-	-	-	-	-

Action:

- Nulling is a process by which zero-levels of rate gyros are determined. When the Nulling is commanded, IMU gathers data from rate gyros for one second then average it for zero-levels. Therefore, the IMU must be in static state during the Nulling process.

- Nulling must be conducted before sending a message which requests angle and rate. IMU does not respond to the request before the Nulling process is finished.

3. Request to Execute Calibration(REC: 0x82)

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Req. to execute Calib.	BNO	0x82		-	-	-	-	-

Action:

- The accelerometer in IMU measures absolute acceleration in all three directions. Before the first time use, IMU must be calibrated to determine the scale and the bias of the accelerometer. Calibration is conducted for 3 axis in six directions (x, -x, y, -y, z, -z) as follows:

a. Connect the IMU unit.

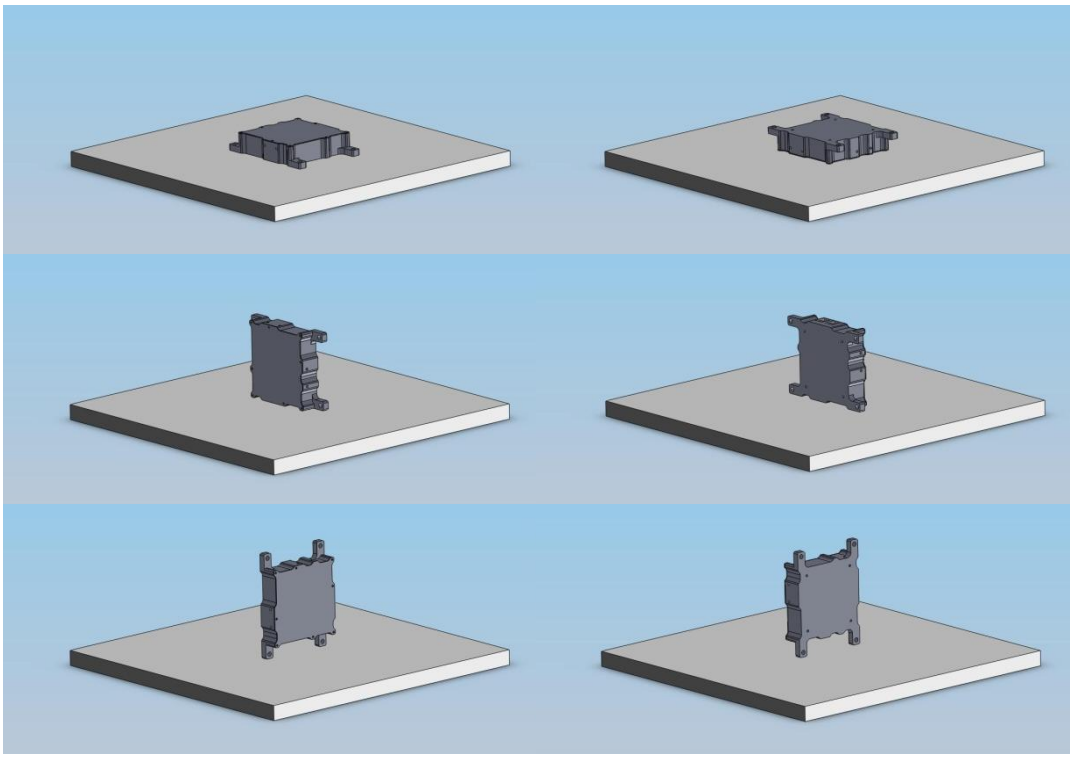
b. Place the IMU on a leveled plate.

c. Send REC (or Push the button "3" in HUBO-i).

d. Wait 10 seconds.

e. Place the IMU on other faces and repeat the process, c and d.

f. After all the faces are done, disconnect the IMU from the power and restart it.



4. Request Parameters

Description	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Req. Paremeters.[1)]	BNO	0x24	PRF	-	-	-	-	-

Return:

PRF=1

Description	MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
Send Time	0x1C0+BNO	ACC_X_GAIN		ACC_Y_GAIN		ACC_Z_GAIN		-	-

PRF=2

Description	MsgID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
Send Time	0x1C0+BNO	ACC_X_BAIS		ACC_Y_BAIS		ACC_Z_BAIS		-	-

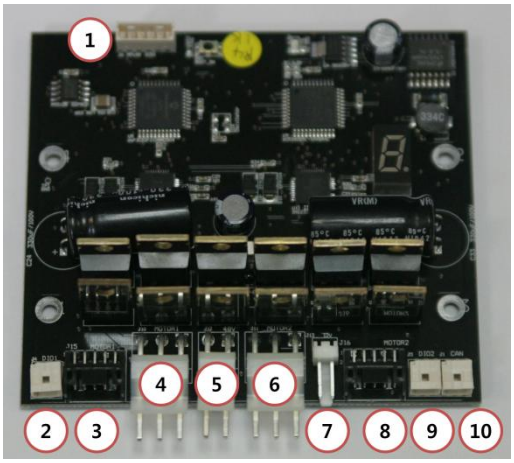
5. Set Board Number (0xA8)

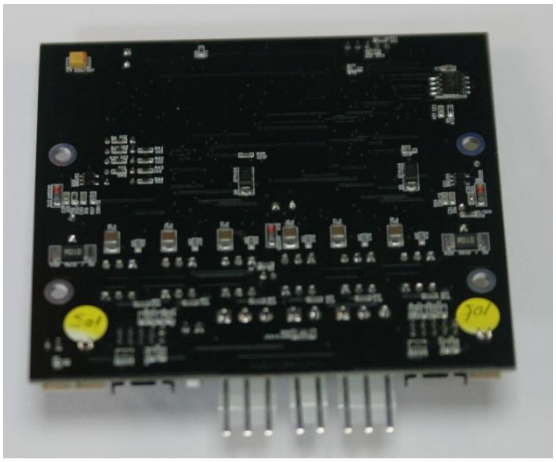
MsgID	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
0x01	BNO	0xA8	NEW_BNO	-	-	-	-	-

Action: Set the board number with NEW_BNO.

Connection Lines

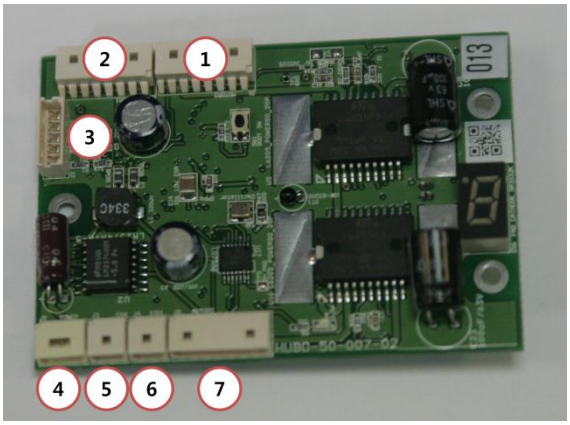
1. 2ch BLDC motor controller(Lower body, shoulder pitch and roll)

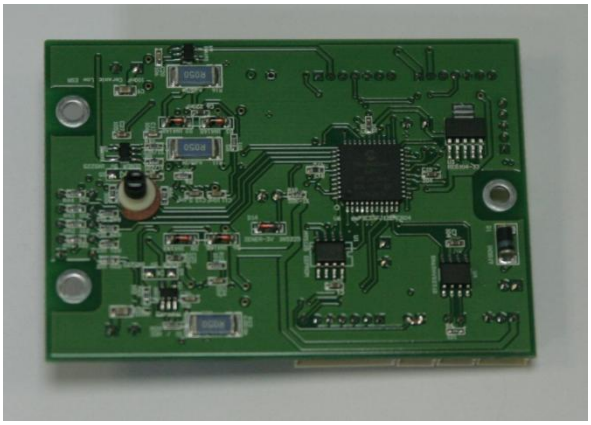




1. MPLAB debugger line
2. Limit switch 1
3. Encoder line 1
4. Motor 1
5. 48V motor power line
6. Motor 2
7. 12V board power line
8. Encoder line2
9. Limit switch 2
10. CAN line

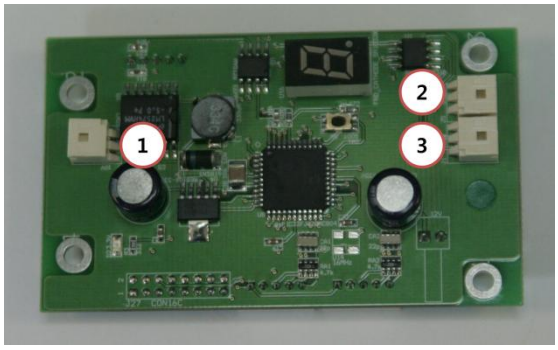
2. 3ch DC motor controller (Neck)

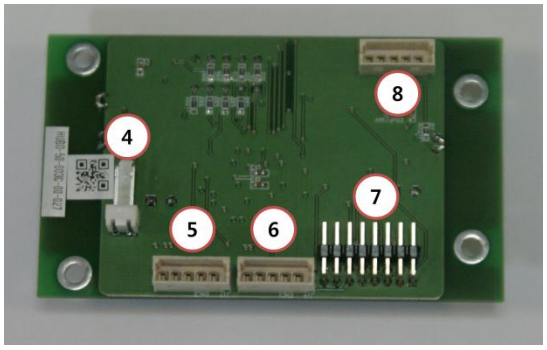




1. Motor 2
2. Motor 1
3. MPLAB debugger line
4. 48V-12V-GND line
5. CAN line
6. Limit switch 1
7. Motor 3

3. 2ch BLDC motor controller (Shoulder yaw and elbow)

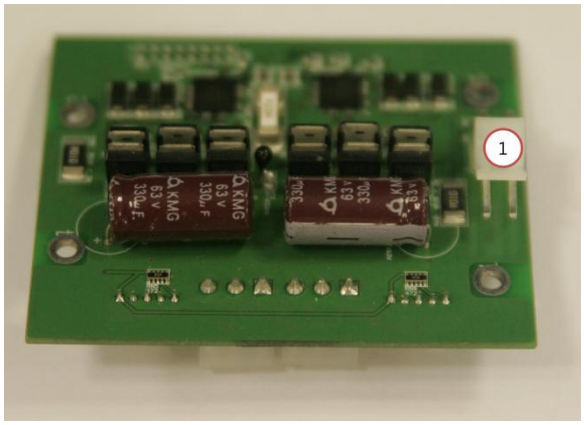


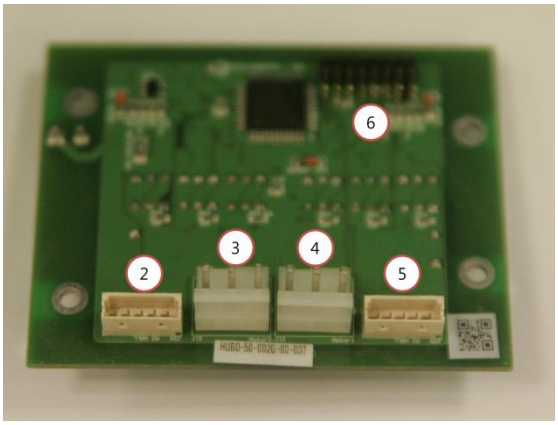


1. Limit switch 1
2. Limit switch 2
3. CAN line

4. 12V board power line
5. Encoder line 2
6. Encoder line 1
7. Controller-Amp connect line
8. MPLAB debugger line

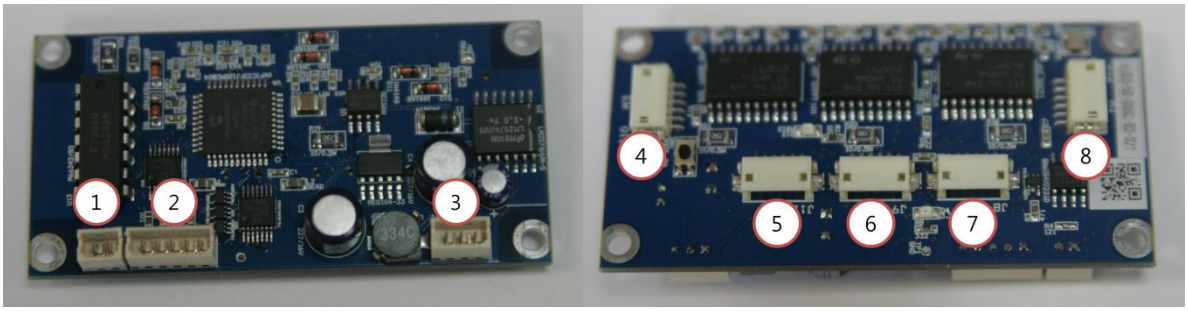
4. 2ch BLDC motor amp. (Shoulder yaw and elbow)





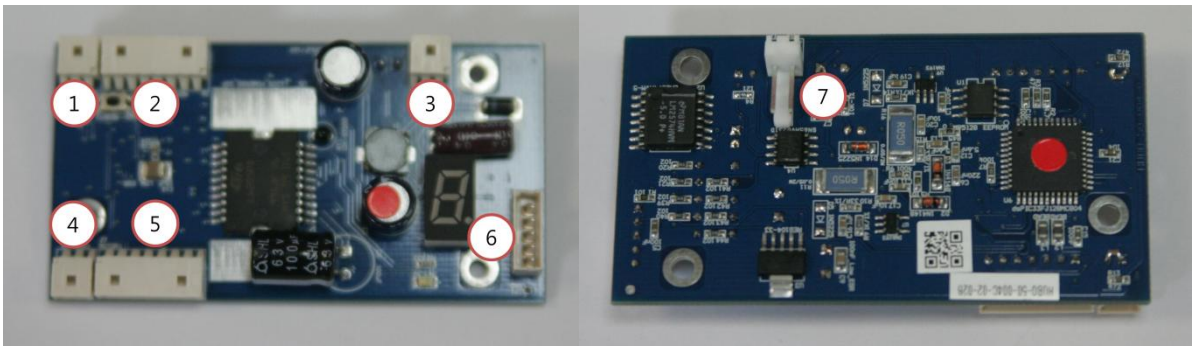
1. 48V motor power line
2. Hall sensor 2
3. Motor 2
4. Motor 1
5. Hall sensor 1
6. Controller-Amp connect line

5. 5ch DC motor controller (Hand)



1. CAN line
2. MPLAB debugger line
3. 12V-12V-GND line
4. Motor 3
5. Motor 4
6. Motor 0
7. Motor 2
8. Motor 1

6. 2ch DC motor controller (Wrist yaw and pitch)



1. Limit switch 2
2. Motor 2
3. CAN line
4. Limit switch 1

5. Motor 1
6. MPLAB debugger line
7. 12V board power line

Programming with Windows Visual C++

April 25, 2012

1 Add New Motion

- Create a new motion activated through a GUI button press

1.1 Add New Motion Button

- Open the Rainbow.dsw file in the C://Rainbow folder using Visual C++
- Dialog boxes are under Resource tab below the left panel in the Dialog folder
- Open IDD_USER_DIALOG (or a dialog window you created)
- Add a new button via the tool panel
  - Click the button icon (When the mouse is placed over it, it will say "Button").
  - Then click and drag on dialog window
- Right click on the new button, then select **Properties**
  - Set the Button ID to its corresponding variable name in the code (e.g. IDC_RAISE_ARMS)
  - Caption: What is displayed on the GUI (e.g. Raise Both Arms)
- Set the function to run on button click:
  - Double Click on button, change the function name or accept the suggested name, click OK (e.g. OnRaiseArms)
  - On OK, the program will bring up file UserDlg.cpp with your new function

1.2 Modify Button Click Function in UserDlg.cpp

- This code goes in the new function created to run on button click (OnRaiseArms)
- Make sure no other commands are running

```
if (pSharedMemory -> CommandFlag ==NO_ACT)
   {...}
```

Set the command flag to be the desired action (This flag tells the RTX system what code in core.cpp to run)

```
PSharedMemory-> CommandFlag = NewCommandName;
```

Example Code:

```
void CUserDlg::OnRaiseArms()
{
    /* OnRaiseArms()
    **
    ** If no command is currently running, set the appropriate
    ** command flag so that the robot will raise both of its
    ** arms together.
    **
    */


    if(pSharedMemory->CommandFlag == NO_ACT)     // Make sure no other command is running
    {
       pSharedMemory->CommandFlag = RAISE_ARMS; // Run the case statement (in Core.cpp)
                                                //corresponding to RAISE_ARMS}
    }

}
```

1.3 Add Command Name to CommonDefinition.h

- Open CommonDefinition.h (under header files for Rainbow)
- Add CommandName to Commands from Win32 to RTX

```
typdef enum
{
OLDcommandNames,RAISE_ARMS
}_COMMAND_FLAG;
```

1.4 Add command to Core.cpp

- Open the Core.cpp file
  - Click File View tab, expand Rainbow files folder and Source Files subfolder
- Add case for new command

  ```
  Switch (PSharedMemory -> CommandFlag)
  ```

  - This code will be run when the CommandFlag is set to the command name
- Set the joint angles and the duration
- Set MoveJointAngle (joint,float angle, float ms, mode)
  - joint = joint # to move
  - angle = degrees to set for joint
  - ms = move time in milliseconds
  - mode = absolute (0x01) or relative (0x00) motion
  - Absolute moves to the specified angle, relative moves the specified # of degrees from the current position
- Set the control mode

```
PSharedMemory -> MotorControlMode = CTRLMODE_POSITION_CONTROL_WIN;
```

- When finished, reset command Flag to NO_ACT

  ```
  PSharedMemory -> CommandFlag = NO_ACT;
  ```

Example Code:

```
case RAISE_ARMS:
  /* case RAISE_ARMS:
  **
  ** Have the robot raise both arms from the shoulder using absolute joint references.
  ** This motion will take 2 seconds to complete.
  **
  */

  // Make sure the joints are not already moving
  if((Joint[RSP].MoveFlag == false) && (Joint[LSP].MoveFlag == false))
  {


     SetMoveJointAngle(RSP,-40.0f,2000.0f,0x01); // Raise the right arm so the
                                                 //shoulder is at -40 degrees
     SetMoveJointAngle(LSP,-40.0f,2000.0f,0x01); // Raise the left arm so the
                                                 //shoulder is at -40 degrees


     pSharedMemory->MotorControlMode = CTRLMODE_POSITION_CONTROL_WIN; // Tell the program
                                                                      // to move the arm
     pSharedMemory->CommandFlag = NO_ACT;  // Set the command flag so the system knows
                                           // the motion has ended
  }
  break;
```

1.5 Other Useful Code

1.5.1 Reading & Setting Text

- (button,textbox,etc.)
- To read text off a GUI element:

  ```
  CString strTemp;                                          // C string used to store
                                                            // read strings
  GetDlgItem(IDC_RAISE_ARMS)->GetWindowText(strTemp);// Read the display text
                                                            // written on the element
                                                            // (in this case a button)
                                                            // IDC_RAISE_ARMS
  ```

- To set text on a GUI element:

  ```
  GetDlgItem(IDC_RAISE_ARMS)->SetWindowText("New Display Text");
  ```

1.5.2 Optional: Sequence Variable

If you need to track location in a sequence, add an unisgned char at the top of core.cpp. (e.g. unsigned char arm_motion_sequence;) It will be considered 0x00 if not set, but best practice is to run an initialization command that sets it to 0x00, then run your commands as usual. Use the value stored in the variable to advance through code sections , as shown below:

```
case RAISE_ARMS:
  /* case RAISE_ARMS:
  **
  ** Have the robot raise both arms from the shoulder using absolute joint references.
  ** Then lower the arms back to the sides. Each half of this motion will take 2
  ** seconds to complete.
  **
  */

  // Make sure the joints are not already moving before running the initial sequence
  if((arm_motion_sequence == 0x00) && (Joint[RSP].MoveFlag == false) &&
  (Joint[LSP].MoveFlag == false))
  {
    SetMoveJointAngle(RSP,-40.0f,2000.0f,0x01); // Raise the right arm so the shoulder
                                                // is at -40 degrees
    SetMoveJointAngle(LSP,-40.0f,2000.0f,0x01); // Raise the left arm so the shoulder
                                                // is at -40 degrees

    arm_motion_sequence = 0x01; // Advance to the next part of the sequence

    pSharedMemory->MotorControlMode = CTRLMODE_POSITION_CONTROL_WIN;     // Tell the
                                                                         // program to
                                                                         // move the arm
  }

  // Wait for the joints to finish moving, then run the second step in the sequence
  else if((arm_motion_sequence == 0x01) && (Joint[RSP].MoveFlag == false) &&
  (Joint[LSP].MoveFlag == false))
  {
      SetMoveJointAngle(RSP,0.0f,2000.0f,0x01); // Lower the right arm so the shoulder
                                                //  is at 0 degrees
      SetMoveJointAngle(LSP,0.0f,2000.0f,0x01); // Lower the left arm so the shoulder
                                                // is at 0 degrees

      arm_motion_sequence = 0x00; // Reset to the beginning of the sequence

      pSharedMemory->MotorControlMode = CTRLMODE_POSITION_CONTROL_WIN;  // Tell the
                                                                        // program to
                                                                        // move the arm
      pSharedMemory->CommandFlag = NO_ACT;              // Set the command flag
                                                        // so the system knows
                                                        // the motion has ended
  }

  break;
```

Hubo-i:

Hubo Protocol commander

Key definitions

[PREV]:	Display page backward.
[NEXT]:	Display page forward
[ESC]:	Escape from the current menu
[ENT]:	Enter the menu
[UP]	Up Arrow key
[DOWN]	Down Arrow key
[LEFT]	Left Arrow key
{RIGHT]	Right Arrow key
1[FET ON]:	Turn ON motor driver.
2[FET OFF]	Turn OFF motor driver.
3[ENC ZERO]	Set position value to zero and Reset error flags.
4[SRV ON];	Turn ON feedback controller.
5[SRV OFF]:	Turn OFF feedback controller.
6[HOME]:	Execute homing process.
7[FUNC 1]:	Toggle the control mode between 'position' and 'current'.
8[FUNC 2]:	Reserved
9[SCAN]:	Scan and find the channel.
0[MENU]:	Enter the MENU function.
#[PLUS]	+ sign key
*[MINUS]	- sign key

Motor control commander

1. Scan menu: Scan and Select the board

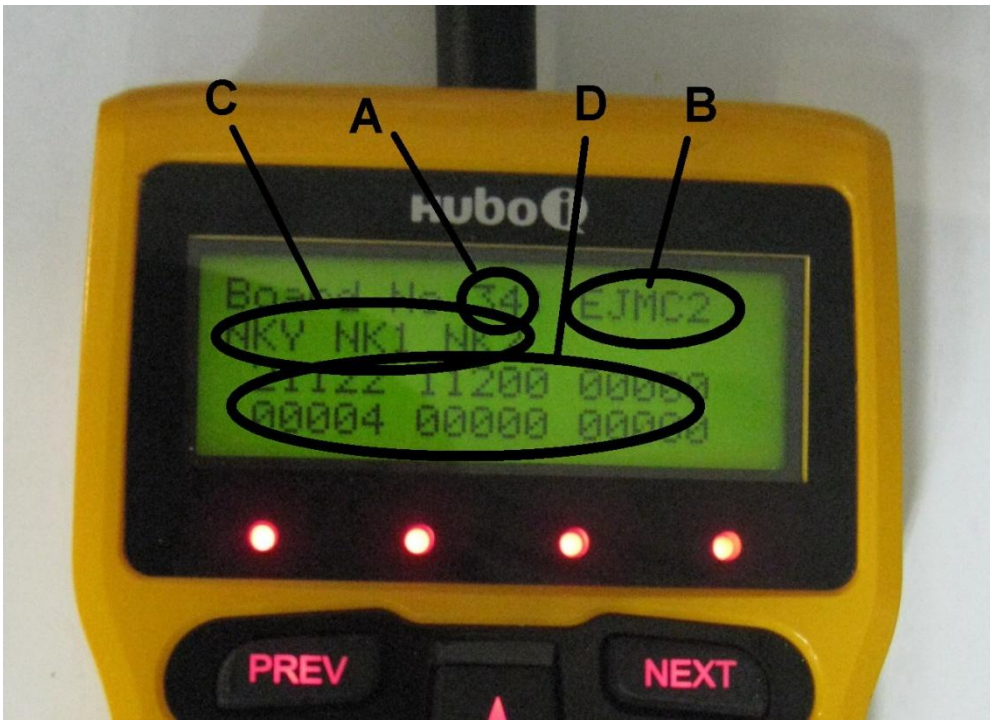
Select the board by moving cursor with arrow keys. Press ENT to enter the Control mode of the Board. The display shows you detail information about the board as the cursor moves:

A. Board No: Board number or BNO.

B. Board attribution number

JMCx :	Joint Motor Controllers
EJMCx:	Extended JMC
FTx	F/T sensors
IMUx:	IMU sensors

C. Name of Axis. See Appendix.



D. Type of boards:

1: 1CH – 2 Motor BLDC Board

2: 2CH – 2 Motor BLDC Board

3: 3CH DC motor Board for Neck joint or 2CH DC Board for Wrist

4: 1CH – 1 Motor BLDC Board for Waist

5: 5CH -5 Motor Board for Hand(Finger)

6: F/T Sensor for Foot

7: Firmware for IMU Board

8: F/T Sensor for Wrist

9: Smart Power Control Board

0: Not available or CAN disconnected

2. Motor control main menu: Control motors.

This menu generates motor commands and displays all the status of the boards.

Displays:

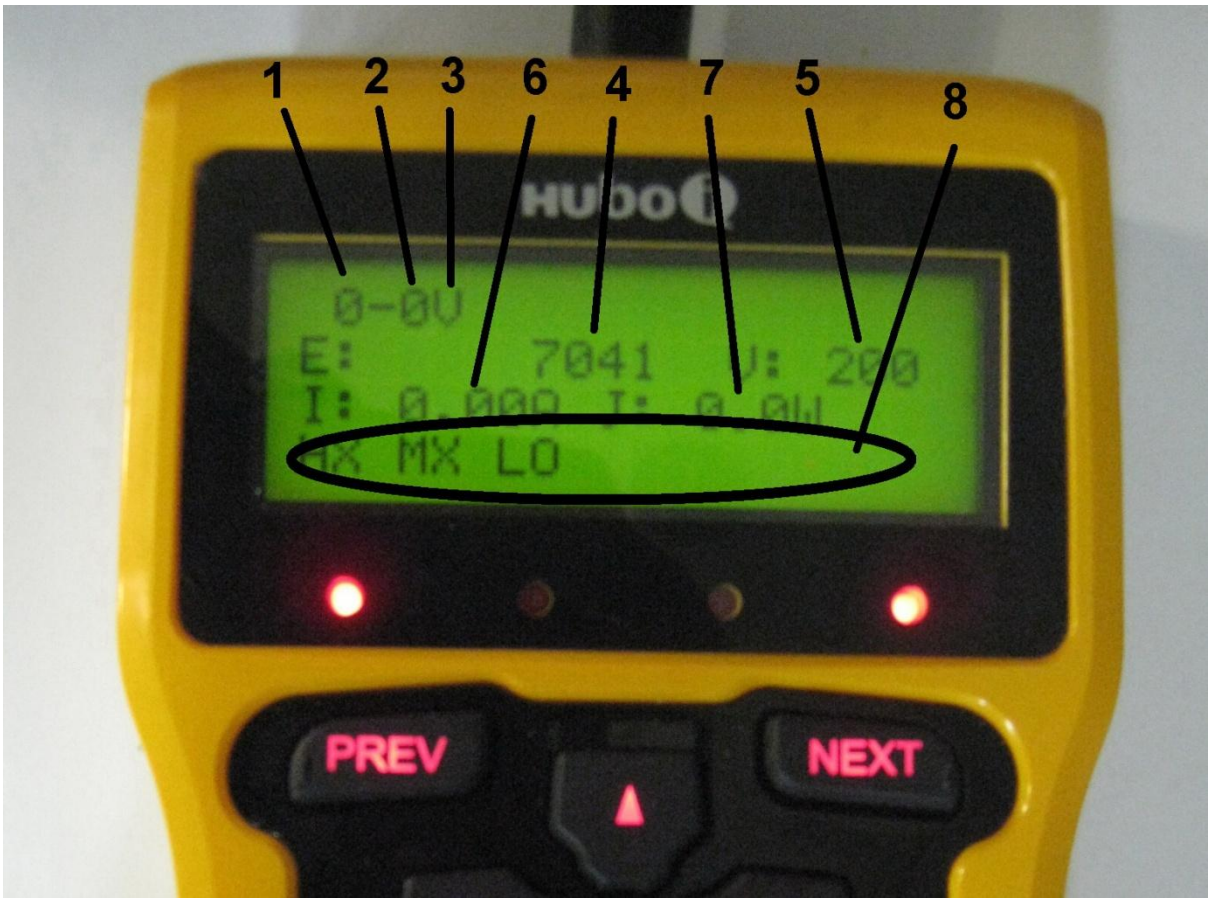
1. **BNO number**
2. **Motor channel number**
3. **Control mode**

    D: Open loop mode

    V: Velocity control mode

    P: Position control mode

    C: Current control mode



4. **Encoder readout**
5. **Command value corresponding to control mode.**
6. **Current in Ampare**

7. **Estimated thermal load for the motor in Watt**.
8. **Board Status and flags:**

HX:	Servo OFF
HO:	Servo ON
MX:	Feedback OFF
MO:	Feedback ON
LX:	Limit switch OFF
LO:	Limit switch ON
F0:	Motor 0 FAULT detected
F1:	Motor 1 FAULT detected
E:	Big Error or Encoder failure detected
B:	Big Input error detected
J:	Motor Jam detected
S:	PWM output saturation detected
L:	Lower limit detected
U:	Upper limit detected
M0:	Motor 0 failed
M1:	Motor 1 failed
V:	Over Max. velocity command detected
A:	Over Max. acceleration command detected

Basic Functions

A. Driver Enable

Long pressing of 1[FET ON] button enables motor driver. 'HO' is displayed

B. Driver Disable

Long pressing of 2[FET OFF] button disables motor driver. 'HX' is displayed.

C. Reset Driver

Long pressing of 3[ENC ZERO] button sets encoder value to zero and reset error flag to zero.

D. Feedback ON

Long pressing of 4[SRV ON] button turns on the feedback control. If the control mode is 'position' then the motor will be position locked. 'MO' or 'CO' are displayed according to the control mode..

E. Feedback OFF

Long pressing of 5[SRV OFF] button turns off the control. If the control mode is 'position' then the motor will be position unlocked. 'MX' or 'CX' are displayed according to the control mode..

F. Execute Homing process

Long pressing of 6[HOME] initiates homing process. The status of searching process will be displayed as follows:

??:	Search for a limit switch.
L :	Limit switch found
LB:	Both limit switch and Index are found
LZ:	Start to move to Offset position.
OF	Move to Offset position
OK:	Homing process is successfully done.
XL:	Backward search of limit switch is failed
LX:	Search of Index is failed
XX:	Search of limit switch is failed

If the homing process is successfully completed the position feedback loop is turned on with displaying 'HO' and "MO". The command mode becomes Open loop mode (D). If the homing process is failed the servo driver and feedback are disabled with displaying 'HX; and 'MX'.

G. Change control mode

Long pressing of 7[FUNC 1] button toggles the mode between 'Position' and 'Current'. Servo driver and feedback control are disabled with displaying 'HX; and 'MX'

Commands:

[NOTE: Symbol Definition]

	Symbol	Unit	BNO 0 ~ 11, 35	BNO 32~34	BNO 36, 37
Encoder Resolution	Er	BAU	4,000	128	64
CAN rate	Cr	Hz	200	100	100
Sampling rate	Fs	Hz	1,000	1,000	1,000
Sampling Period	Ts	sec	0.001	0.001	0.001
Encoder counts	P	BAU	-	-	-

BAU=Basic Angle Unit.

There are two modes for commanding motor: Open loop and closed loop. There are also three kinds of closed loop feedback control modes: current, velocity and position control.

User can select one out of them by changing the control mode by [PREV] and [NEXT] keys or by 7[FUNC 1].

A. Open loop mode (D)

1. Enable servo driver by pressing 1[FET ON] button for more than 1 sec until beep sounds. Confirm 'LO' is displayed.
2. Change duty value by [UP] and [DOWN] keys. The value varies from 0(%) to 100(%).
3. Run the motor in the direction as pressing the arrow keys, [LEFT] or [RIGHT].
4. Long pressing of 2[FET OFF] button disables servo driver. 'LX' confirms the servo off.

B. Velocity mode (V)

1. Enable feedback controller by pressing 4[SRV ON] button for more than 1 sec until beep sounds. Confirm 'MO' and 'LO' are displayed.
2. Change velocity value by [UP] and [DOWN] keys. The value of velocity (v) is given by

   v = P / Ts

   Thus the speed of motor in rpm can be calculated by

   (rpm) = v * Fs / Er * 60

   For example, Velocity value= 200 gives you 3000 rpm.
3. Run the motor in the direction as pressing the arrow keys, [LEFT] or [RIGHT].
4. Long pressing of 5[SRV OFF] button disables servo driver. 'MX' and 'LX' confirm the servo and feedback off.

C. Position mode (P)

1. Enable feedback controller by pressing 4[SRV ON] button for more than 1 sec until beep sounds. Confirm 'MO' and 'LO' are displayed.
2. Change position value by [UP] and [DOWN] keys. The value of position is given by encoder counts.
3. Long pressing of 5[SRV OFF] button disables servo driver. 'MX' and 'LX' confirm the servo and feedback off.

D. Current mode(C)

1. Long pressing the button 7[FUNC 1] toggles the modes between 'velocity' and 'current'.
2. Enable feedback controller by pressing 4[SRV ON] button for more than 1 sec until beep sounds. Confirm 'MO' and 'LO' are displayed.
3. Change current value by [UP] and [DOWN] keys. The value of current is given by Ampare.

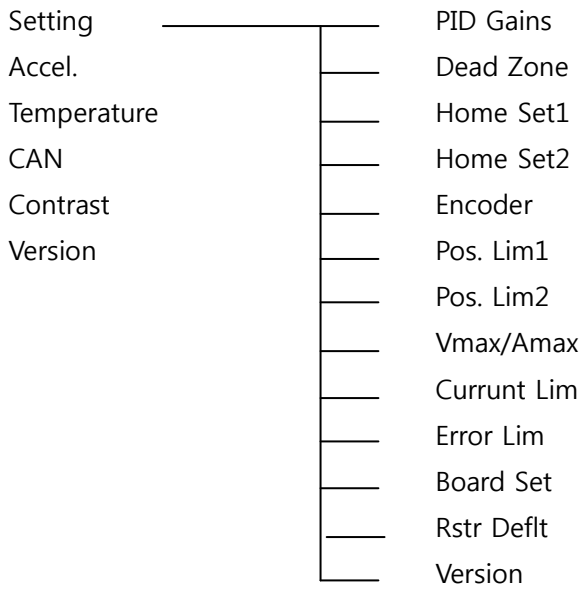
4. Long pressing of 5[SRV OFF] button disables servo driver. 'MX' and 'LX' confirm the servo and feedback off.

Changing the channel

Press [ENT] button for more than 1 sec until beep sounds. Change the channel with number keys when the cursor blinks at the channel number. Press [ENT] shortly to set the channel.

Setting the parameters

Press 0[MENU] button for more than 1 sec until beep sounds. Menu will be listed as follows:



A. Accel.: Set the acceleration value for velocity and position control. The value of acceleration (a) is given by

$(rad/sec^2) = 2\pi * a * Fs^2 / Er$ or

$(P/sec^2) = a * Fs^2$

B. Temperature: Display the temperature of the target board in degree C.

C. CAN: Set the CAN rate in Hz.

D. Contrast: Adjust the LCD contrast by pressing [LEFT] or [RIGHT] buttons.

E. Version: Display HUBO-i version number.

F. Setting: Enter the sub-menu for each channel.

Sub-menus for Setting

A. PID Gains: Set the PID gain of velocity and position control. PID gain is given by

$$(\mathrm{PWM}) = (\mathrm{Kp} + \mathrm{Kd}\frac{\mathrm{d}}{\mathrm{dt}} + \mathrm{Ki}\int \mathrm{dt})(\mathrm{postion\ error})/1000.$$

For current control mode, the PD controller is used with low pass filter for current measurement with cut off frequency, Kf. Kp and Kd are scaled by 1000 as above and Kf is given by Hz.

B. Dead Zone: Dead zone value is determined by FET characteristics. The value is given by encoder counts.

C. Home Set1: Set motor homing parameters 1. There are three values to be set:
Offset: Encoder value to be offset from Index position of the encoder
Home Lim: Search range for limit switch. The value is given by rotation counts of the motor.

Lim dir.: Direction to search a limit switch.

D. Home Set2: Set motor homing parameters 2. There are four or five parameters to be set.
V_max1: Search velocity to find a limit switch.
V_max2: Velocity to get offset from the Index position.
A_max: Acceleration to reach V_amx1 and V_max2.
Lim Mode: Set search mode
0: Search home with limit switch and Index signal
1: Search home with limit switch only
2: Search home with motor stops with the prescribed pwm duty value in %. Default value is 25%. This value can be changed by re-entering Home Set2 menu if it is not shown.

E. Encoder: Set encoder resolution and motor direction
Enc Res: Encoder resolution
Auto Scale: Set auto scale. Default is 1. This function is ignored except finger control boards.
Motor Dir: Set Motor running direction. 1:CW, 0:CCW

F. Pos Lim1:Set lower limit of the motor position
Pos Lim1: Lower limit of the motor position in encoder counts.
Enable(1/0): Enable(1) or disable(0) lower limit function. This function works for velocity and position modes only. If this function is enabled, the motor will stop

below in this position value.

G. Pos Lim2: Set upper limit of the motor position

Pos Lim2: Upper limit of the motor position in encoder counts.

Enable(1/0): Enable(1) or disable(0) lower limit function. This function works for velocity and position modes only. If this function is enabled, the motor will stop over this position value.

H. Vmax/Amax: Set the value for Vmax and Amax. Vmax limits the excessive velocity command to up to Vmax.

$(rpm)_{max}$ = Vmax * Fs / Er *60

Default value of Vmax=1500 yields 22,500 rpm or 2,356 rad/sec

Amax also limits the excessive acceleration command to up to Amax.

$(rad/sec^2)_{max}$ = 2π * Amax * Fs^2 / Er

Default value of Amax=20(P/Ts) yields 31,415 rad/sec^2.

I. Current limit: Set JAM and Power saturation limit.

JAM: Disable motor driver and JAM error flag will be issued if [JAMD]% of duty sustained for [JAM_LIM/10] seconds without movement.

Power Sat: Disable motor driver and power saturation flag will be issued if full duty sustained for PWM_LIM second.

Default values

	Unit	BNO 0 ~ 11,35	32, 33	34	36, 37
JAM_LIM	0.1 sec	3	15	15	15
JAMD	%	50	50	40	80
PWM_LIM	0.1 sec	5	10	10	20
LIMD	%	25	25	20	25
I_ERR	BAU	30,000	10,000	120	120
B_ERR	BAU	20,000	10,000	400	1000
T_MAX	degC	45	45	45	45

J. Error limit: Set Error limits. There are three parameters to be set.

Input err (I_ERR): Disable motor driver Set maximum input reference command change in Cr^{-1} sec. I_ERR can be converted to (rpm) by

$(rpm)_{I\_ERR}$ = I_ERR * Cr / Er * 60.

(ex) $(rpm)_{I\_ERR}$ =30000 * 200 /4000 * 60 = 90,000 (rpm)

Big err (B_ERR): Disable motor driver. Set maximum error between reference and position.

Max. temperature warning(T_MAX): Set the warnig temperature.

K. Board set

Board No: Change board number (BNO). Turn off and re-boot the board to initialize the parameters.

CAN rate: Read only. This value shows actual CAN rate measured by the board's internal clock.

No of CH: Read only.

L. Rstr Deflt: Restore board's parameter with default values. Offset value is remained unchange. Press [ENT] button for 1 second until beeps to execute this function

.

M. Version: Read only. Displays version number.

Seven Segment Display of Motor Driver Board.

1. Initialization

A. **Hard initialization**:

Hard initialization can be performed by one of two ways:

1) Firmware programming at the first time
2) Power ON with pressing reset button.

Hard initialization sets all the parameters with default values and set BNO to zero. Turning Power ON with hard initialization conditions the seven-segment displays the letter 'H' followed by 'A' and two zeros which stands for Board number, BNO=0.

B. **Soft initialization**:

After changing board number (BNO) one should execute power off and on to re-initialize the board for the new setup. Turning Power ON the seven-segment displays the letter 'H' followed by 'A' and two digits which stands for Board number, BNO

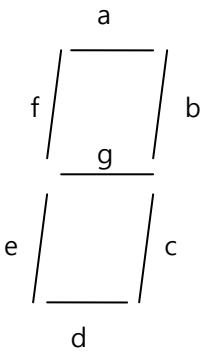
C. Restoring Defaults

Board initialize command (JBR) initiates the defaults restoring process. All the parameters are restored by default values except Offset. During the process the seven-segment displays the letter 'H' followed by 'A' and two digits which stands for Board number, BNO. It is strongly suggested that turn off and on the power after this process.

D. Normal power ON

Turning power on the board the letter 'A' will be displayed followed by two digits which stands for Board number (BNO).

2. Normal operation



A. Blinking g-segment:

g-segment blinks with rate of Timer interrupt frequency divide by 500. The frequency of timer interrupt is 1000Hz.. Thus g-segment should blink 2 times a second if the board is properly working.

B. Alternate blinking of f and b-segment:

f and b-segments blink alternately if 'read' command is received, Blinking these segments usually mean that 'request to send encoder readout' is receiving, i.e., the encoder is read by controller.

C. Alternate blinking of e and c-segment

e and c-segments blink alternately if 'command' instruction is received, Blinking these segments usually mean that position command is receiving, i.e., servo feedback is working

D. Toggling d-segment

d-segment toggles if any status or flags of the board has changed. It is useful to check the status change of limit switch and so on.

E. Blinking a-segment:

a- segment toggles if any state change of CN(change notification) pin.

3. Home Search operation

The seven segment display shows the status progress during home search operation as a number.

1: Start to find the Limit switch.
2: Limit switch-ON is found.
3: Limit switch-OFF is found and go backward. to get ON
4: Index signal detected.
5: Start to move to Offset position.
6: Arrived at Offset position. -> Success!

d: Fail to find backward limit switch signal.
e: Fail to detect Index signal
F:: Fail to find Limit switch

4. Error display

Motor driver will be disabled if critical error condition is encounter. The display shows the status of fatal error starting with 'F' and two successive digits. First digit shows channel number and the second shows the error type. The error types are shown in below:

1: JAM Error
2: Big Input Change Error
3: Power Saturation Error
4: Big Error or Encoder Failure
5: Driver Fault signal detected
6: Motor Failuare(for Board Type 1)

Error status is cleared by pressing 3[ENC_ZERO] or by REZ command.