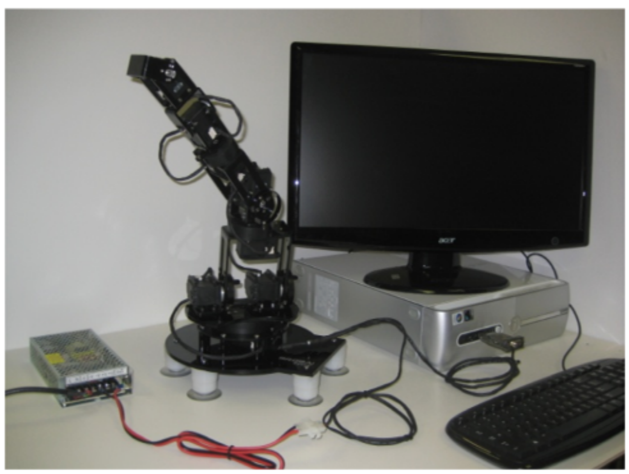
Cyton

User's Manual



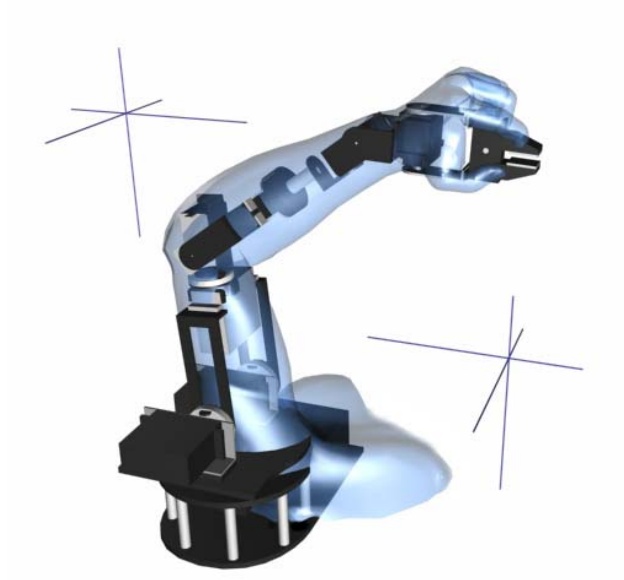
ROBAI

www.robai.com

Robai
PO Box 37635 #60466
Philadelphia, PA 19101-0635

Introduction

The Cyton is a seven degree of freedom manipulator arm with a gripper end effector. Humanoid manipulators offer profound advantages. With many degrees of freedom they are able to reach around obstacles, reconfigure for strength, improve accuracy, and manipulate objects with fluid motion. Cyton comes with configurable control software that makes it easy to exploit its kinematic redundancy with built-in interfaces to input devices, over the Internet, or using your own programs such as a USB joystick plugin. This software controls the Cyton arm in real time based on desired behaviors that are configured off line.



Combined with Actin SE visualization, reasoning, and control software, the Cyton performs advanced control by exploiting kinematic redundancy. With built-in networking software, it can be controlled remotely through a local area network or over the Internet.

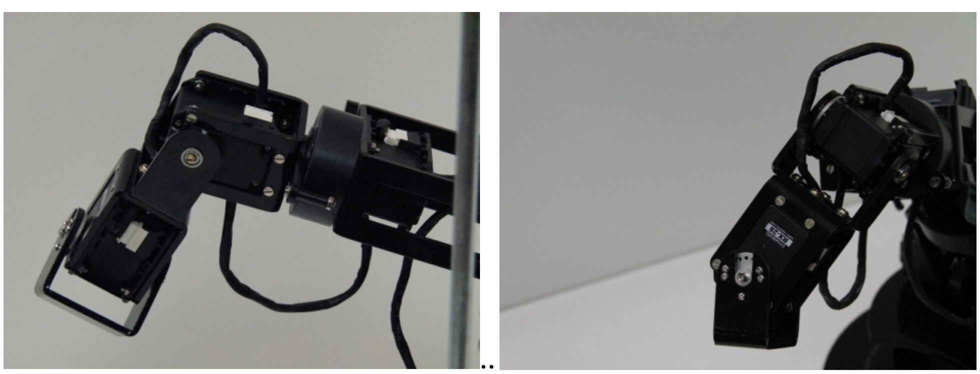
Mechanical Structure:

The Cyton provides robust, powerful, and intelligent manipulation for various applications. The V2 extends the capabilities of the Alpha is several ways.

Intelligent Actuators
The high performance, intelligent actuators give feedback and data values on position, speed, voltage, and temperature via the Cyton Control GUI.

USB Interface
The arm can be controlled through a USB 2.0 interface to a PC. Data transfer rates from the USB port to the arm are rated up to 1Mbps.

Training Mode
The arm can be put into a zero torque record mode allowing developers to script desired paths. End users can manually grab and move the gripper and record the paths for playback.



Cyton Wrist Yaw: Various Positions (Cyton V2 without gripper shown)

Specifications:
Cyton Axes Range

- Shoulder base 300 degrees
- Shoulder Pitch 180 degrees
- Shoulder Yaw 300 degrees
- Elbow Pitch 240 degrees
- Wrist Roll 300 degrees
- Wrist Yaw 200 degrees
- Wrist Pitch 200 degrees

Other Specifications for:

Cyton Veta
- Rated Payload: 300 g
- Maximum Payload: 350 g
- Rated speed: 0.4 m/s
- Joint speed: 75 rpm
- Reach 48 cm
- Repeatability: 0.3cm

Cyton V2
- Rated Payload: 1 Kg
- Maximum Payload: 1.0 Kg
- Rated speed: 0.4 m/s
- Joint speed: 75 rpm
- Reach 43.4 cm
- Repeatability: 0.1cm

The Cyton comes with the Actin-SE software allowing it to perform advanced control by exploiting its many degrees of freedom. With available networking software, it can be controlled remotely through a local area network or over the internet.

The high performance servos and advanced kinematics combined with Energid's Actin-SE software give smooth, humanlike motion.

Setup Instructions

Hardware Setup

Components required (supplied in Kit)

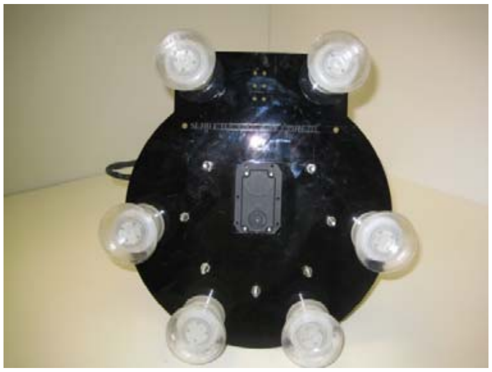
- 16V DC power supply[1].
- USB2Dynamixel connector.

Setup Instructions

- Place the Cyton arm on firm level surface and make sure that the arm has at least 60 cm clearance all around it.
- Ensure that the vacuum cups at the bottom of the

[1] Includes a 16/12 V power supply for Cyton Veta

base of the arm are well engaged with the surface on which it is mounted.



[underside of Cyton]

- Use extension cable provided along with the arm to connect USB2 Dynamixel with the arm. The connector on the cable with power line missing goes into the USB2 Dynamixel.

[USB2 Dynamixel]

- The other connector of the cable is connected to the Cyton arm. While doing the connection at the arm end, make sure that the markings on the connector of the cable and the one fixed on the base of the arm are aligned.
- This cable also has one more branch onto which the DC supply for the arm is given. Once these connections are made, you can plug the USB2Dynamixel into your computer's USB port.

[16 V DC Power supply]

- Now you can power the arm by turning plugging in the DC supply. The LEDs located at the top of the arm's servos should blink once when power is turned ON.

[complete setup]

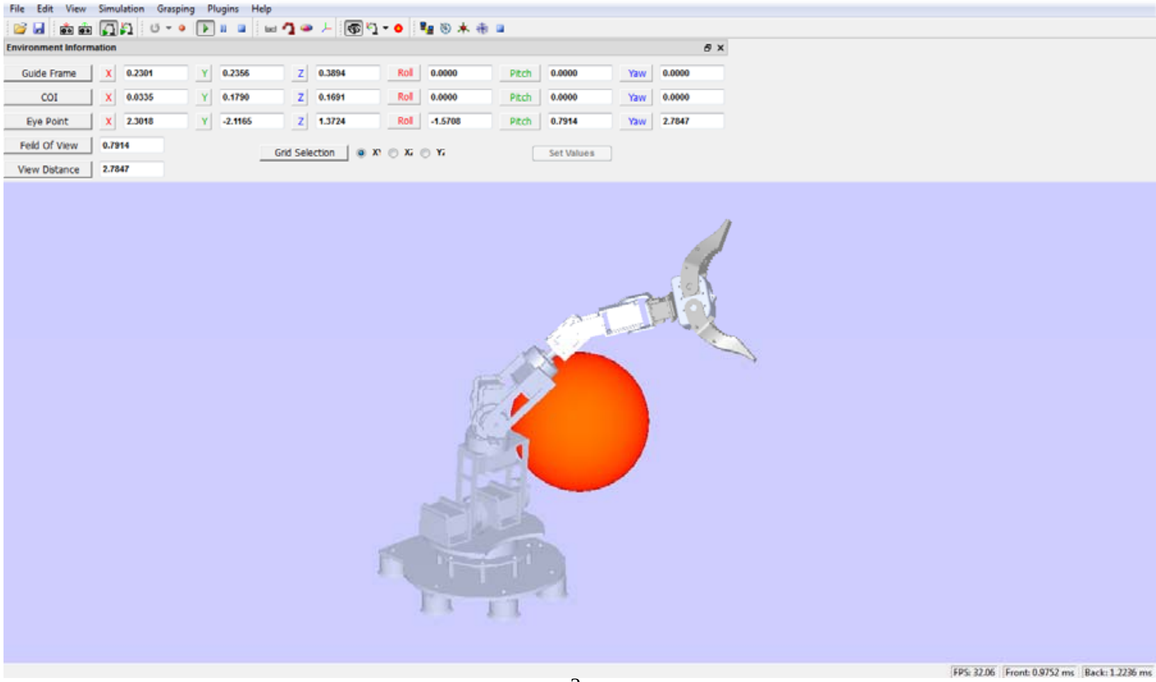
Software Installation

Insert the CD Disk into the CD drive. If the CD does not start automatically you should be able to browse to the CD folder and select cytonSetup.exe. This will start the installation program. Follow the instructions to complete installation.

The Cyton Viewer

Cyton Viewer Capabilities

The Cyton Viewer can be used to both simulate motion of the robot and to directly control the robot. It has several powerful features that allow for end-effector or joint level control the Cyton. The figure below shows the viewer with a Cyton model loaded.



[2]

Changing perspective

- Eyepoint. The eyepoint icon changes the viewer into eyepoint mode. In this mode the eyepoint can be changed by dragging the mouse.

- Center of interest: The center of interest (i.e. the direction where the eyepoint is looking) can be changed by entering COI mode with the COI icon and dragging the mouse.

File Options

Shown are the options available within the file menu. These are described below.

Open
Opens in a Cyton model file. The Cyton viewer currently comes with one file called Cyton.ecz. In addition, new files can be created using the Cyton C++ API.

Save Image As
Save a snapshot of the current robot. Currently .tif is the only supported image format.

---

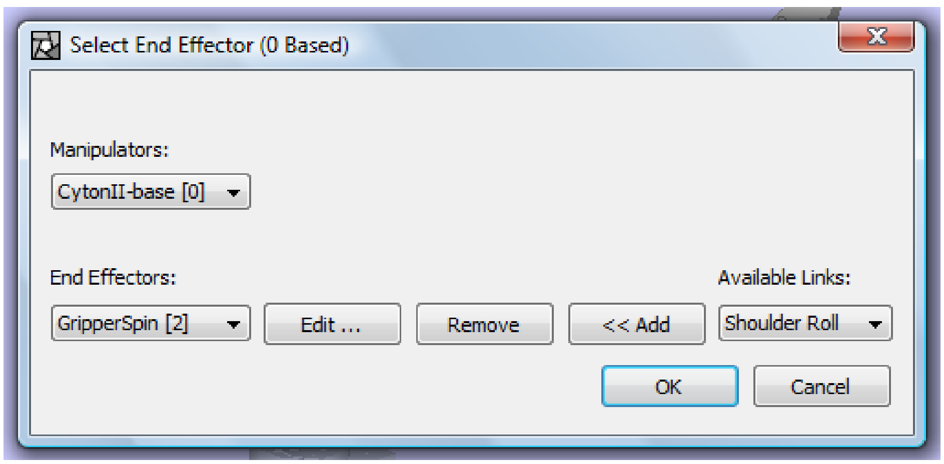
[2] Cyton V2 with custom gripper shown

Flying the Gripper

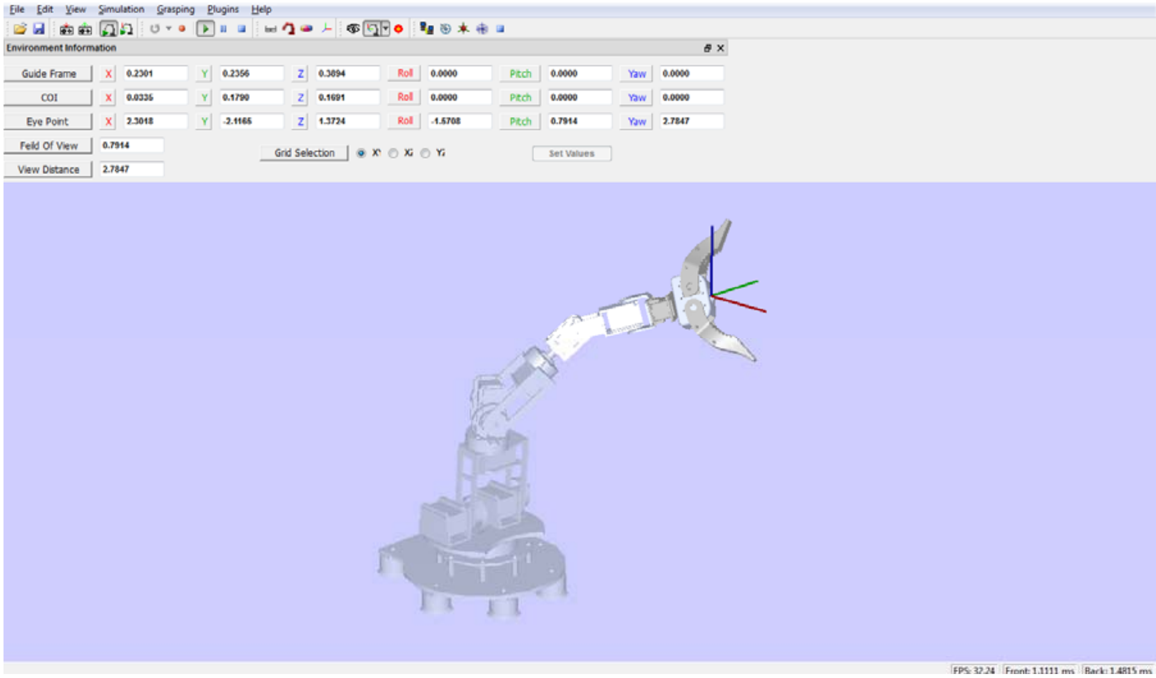
Using the Cyton Viewer, it is possible to command the gripper (or end-effector) to go to an arbitrary position and orientation in space as long as it's within the arms workspace. To do this it's important to calibrate the viewer view with the position of the actual robot.

The Guide Frame

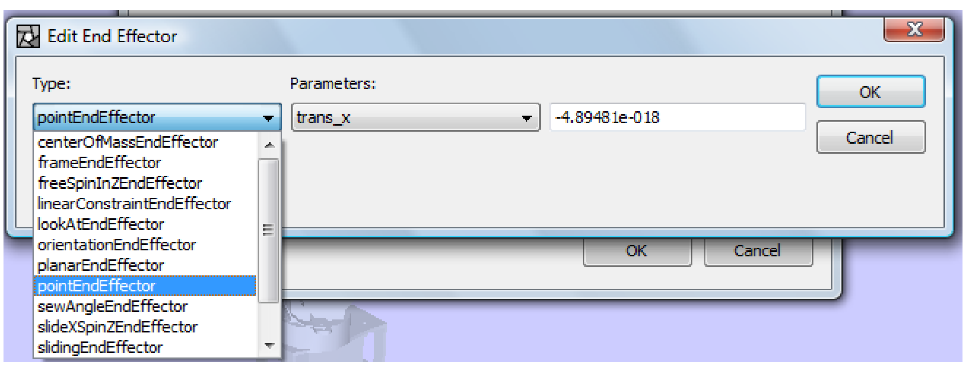
The Guide Frame is what the Cyton Viewer uses to specify the desired gripper position. The figure below shows a Guide Frame (with red, green, and blue axes) just in front of the gripper. You can move the Guide Frame by first selecting the set guide frame button . This will bring up the following dialog box.



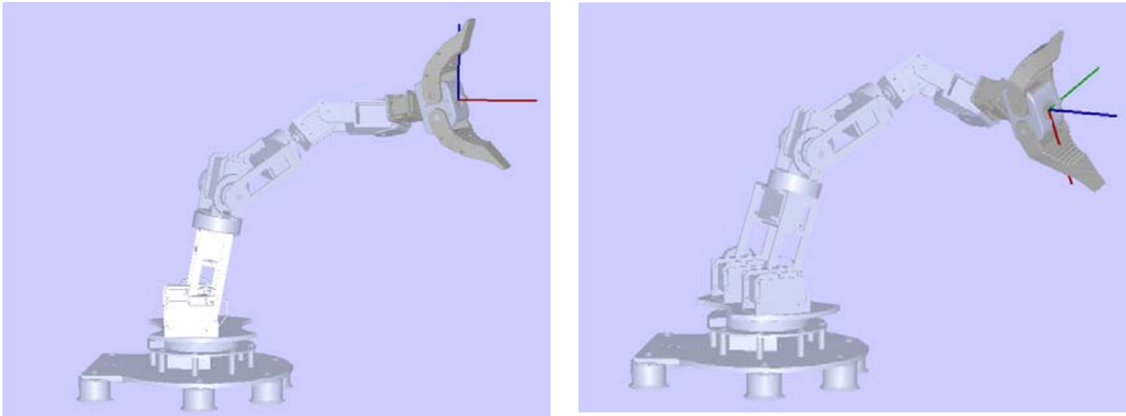
An end-effector can be placed on any link of the robot. By default the end-effector is placed at the gripper. Once the "OK" button is clicked you can move the Guide Frame within the viewer dragging with either the left or right mouse buttons depressed. Holding down the right mouse button will allow you to change the position of the Guide Frame and the left mouse button allows you to rotate the Guide Frame.



Depending on the type of end-effector selected the gripper will either move to a specific position in space (with a point end-effector) or position and orientation (with a frame end-effector). The type of end-effector can be selected with the drop-down list below under the Edit End-Effector.



The difference between a frame end-effector and a point end-effector can be seen in the images below. Note that with a point end-effector the orientation of the gripper is arbitrary—only the position of the Guide Frame is important). With a frame end-effector (shown on the right) both position and orientation are considered—note that the gripper is aligned along the red axis of the Guide Frame.



Left: The arm moved to the guide frame using a point end-effector (i.e. position only). Right: The arm moved to the guide frame using a frame end-effector (i.e. frame end-effector).

Working with Path Files

The Cyton Viewer allows you to capture paths of the robot for future playback. This is useful for certain applications.

Recording a Path – Pressing the record button puts the viewer into record mode. When in this mode robot positions will be stored in memory until the stop button is pressed. Positions can be stored in one of two formats: Manipulator (Joint) mode, or Guide Frame mode. In Manipulator mode the viewer records all of the joint angels for the robot at each timestep, whereas in Guide Frame mode only the commanded gripper positions at each timestep are recorded.

This means that a Guide Frame mode path file may result in different joint positions when rerun depending on the control method being used for the Cyton. For instance, a control system configured to minimize kinetic energy will result in different joint angle trajectories than a control system configured to minimize potential energy.

A path file recorded in Manipulator mode, by contrast, is guaranteed to always give the same joint trajectories. By default the Cyton Viewer records in Manipulator mode. You can enter Guide Frame mode by pressing and can revert back to Manipulator Mode by pressing .

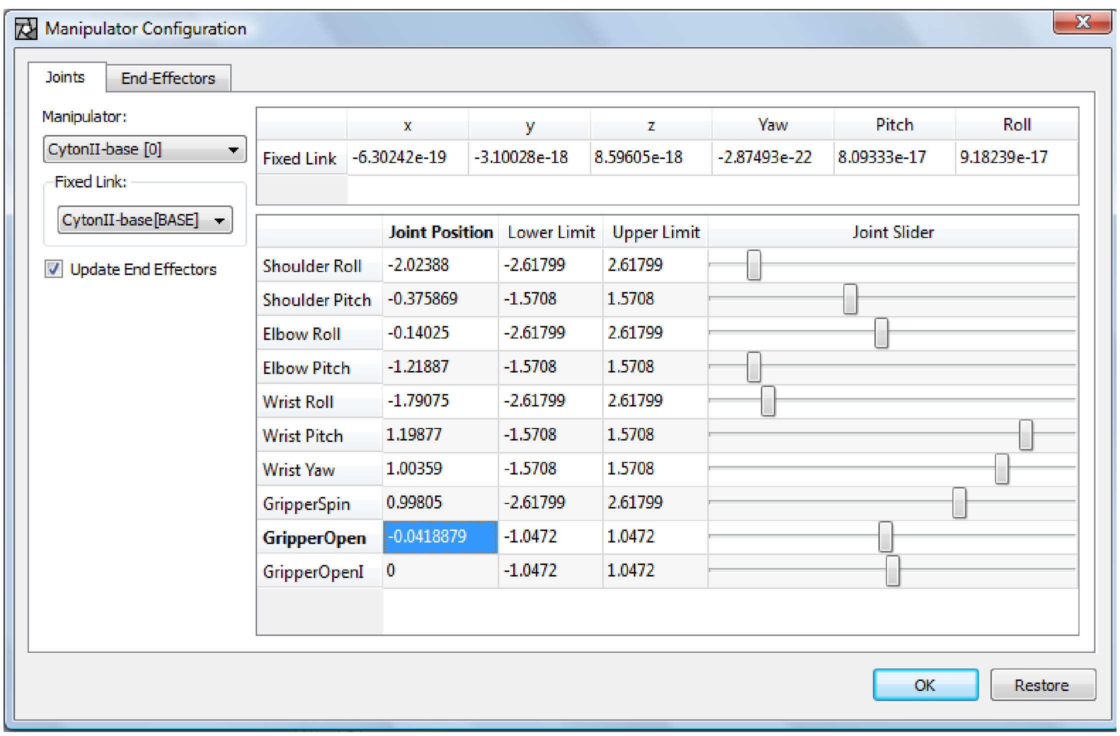
Save Path File – This allows you to save a path just recorded. A Save File dialog box will appear asking for the name and location of the file to be saved.

Load Path File – This allows you to load a previously saved path file. Once loaded, the record mode buttons should automatically change to indicate whether the path is in Guide Frame mode or Manipulator mode.

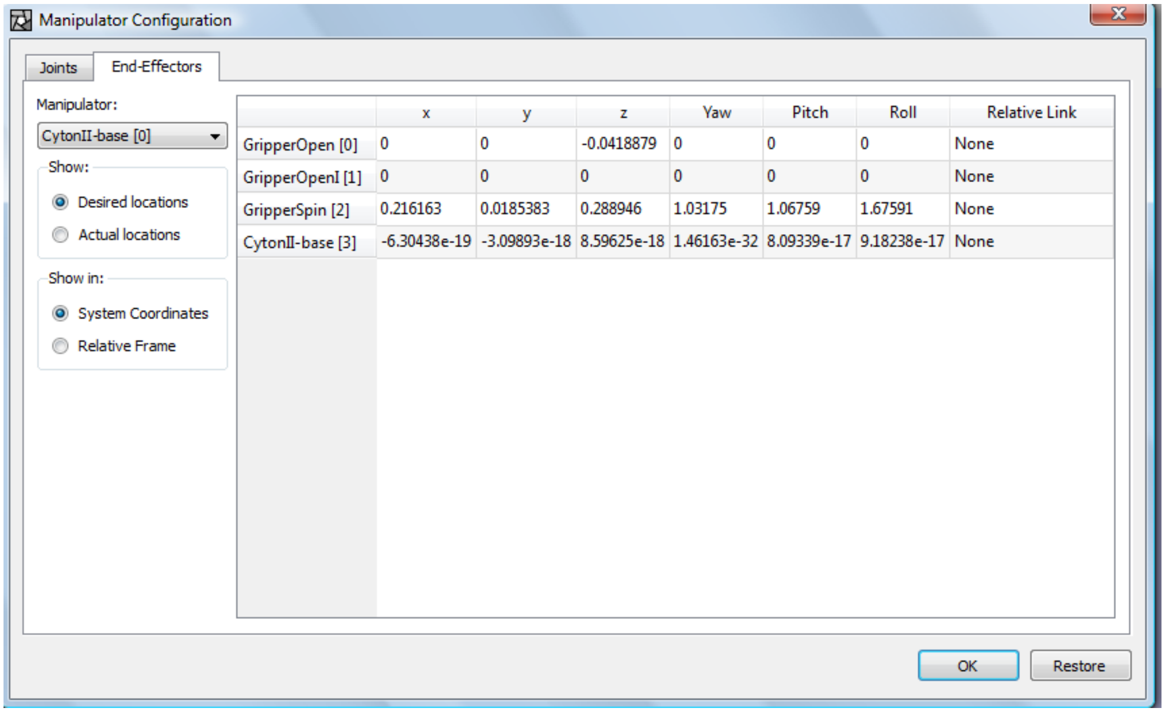
Playback Mode – Once a path is loaded it is still necessary to specify that you would like to playback the path. If the playback mode button is pressed hitting the play button. Opening the dropdown list for the Playback mode button will allow you to select whether or not the playback should be repeated. If in Guide Frame mode the Cyton manipulator should be checked under the Guide Frame Manipulators dialog box—this should be the default.

The Manipulator Configuration Tool

Manipulator Configuration – More precise tasking of the robot can be achieved by using the manipulator configuration tool. This tool allows you to independently move each joint. It also lets you change the joint limits for each joint. Each joint on the robot can be moved using the slider bars on the right. The upper and lower joint limits can be set directly in the edit boxes at the left. The figure below shows the manipulator configuration tool with the Joints tab.



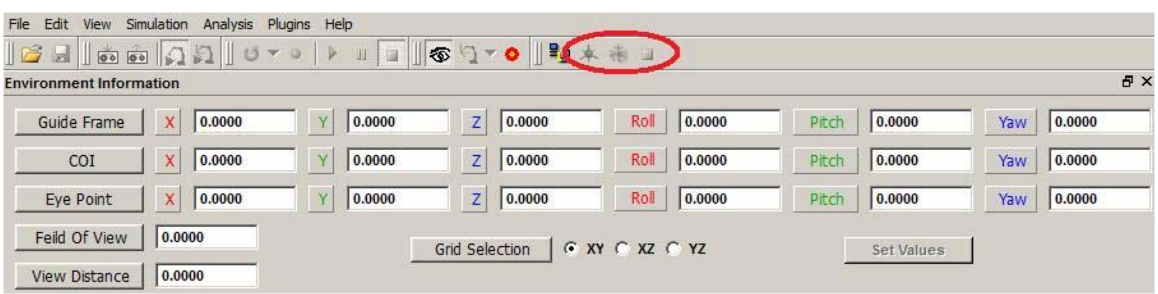
The End-Effectors tab (shown below) allows you to directly control the position and orientation of the end-effector. This is useful when a precision gripper position is required.



I. How to use the Environment Info Plugin

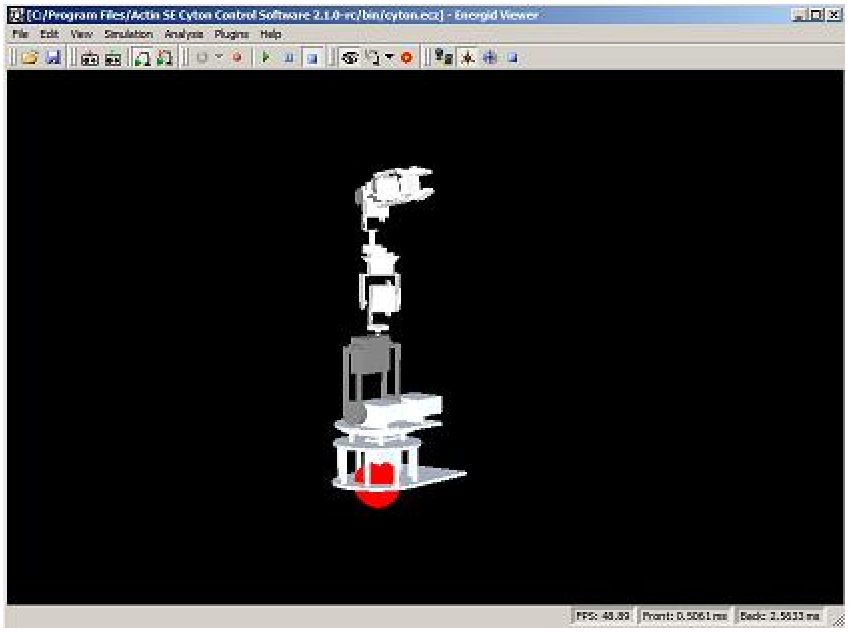
EnvironmentInfoPlugin gives a user an idea of the environment inside the Cyton Viewer. It displays to the user the origin and XYZ-planes. It also gives the user information of the visualization parameters like Eye point, Center of interest, field of viewer and view distance. It also provides the user with information about the location of guide frame in the environment. The plugin also helps a user to set Eye point, Center of Interest and guide frame location.

Once the plugin is loaded into the viewer, you will have 3 new buttons in the toolbar and new tab called **Environment Information**.



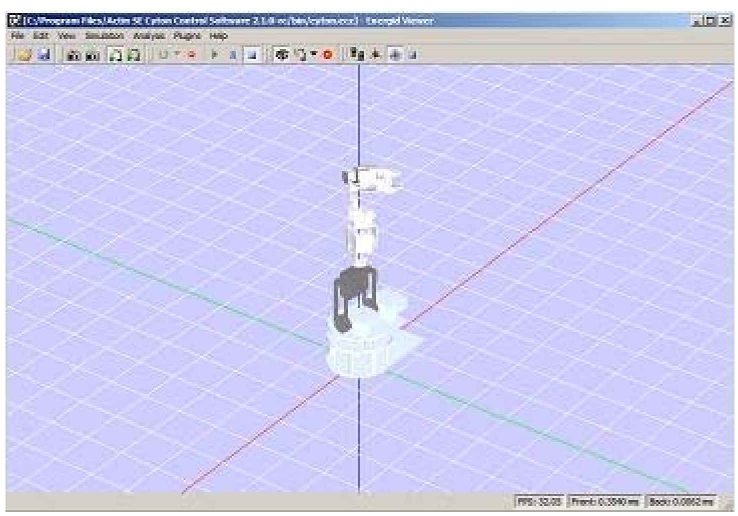
In the **Environment Information** tab, the location of the Guide Frame, Center Of Interest(COI), Eye Point, Field of View and View Distance are displayed. The 3 new buttons on the viewer as part of environment info plugin are **Show Origin** , **Show Axis** and **Set Values** . These buttons will be enabled only after model is loaded into the viewer.

When **Show Origin** button is clicked, it will display the location of origin(0,0,0) in the viewer environment. The origin is displayed as a red ball.

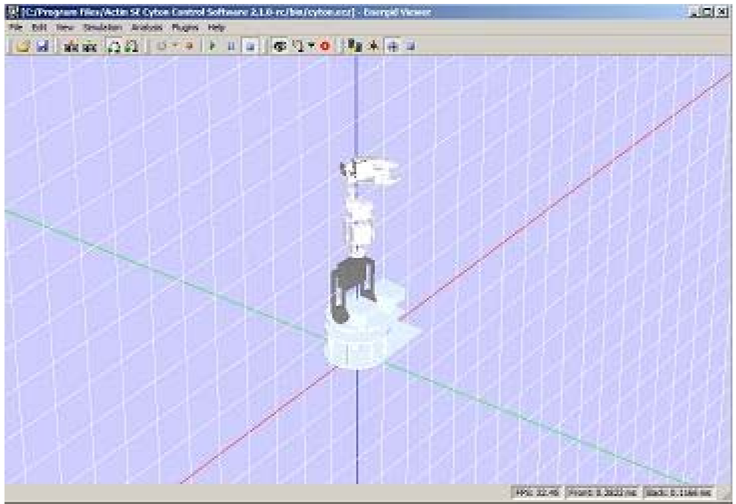


When **Show Axis** button is clicked, it will display the XY grid by default. To display the required grid you will have to change the selection in **Grid Selection** 

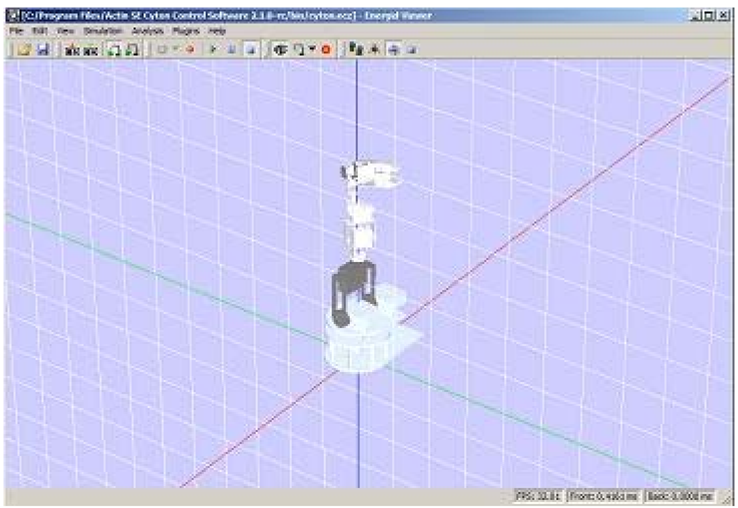
in the **Environment Information** tab.



XY-Grid



XZ-Grid

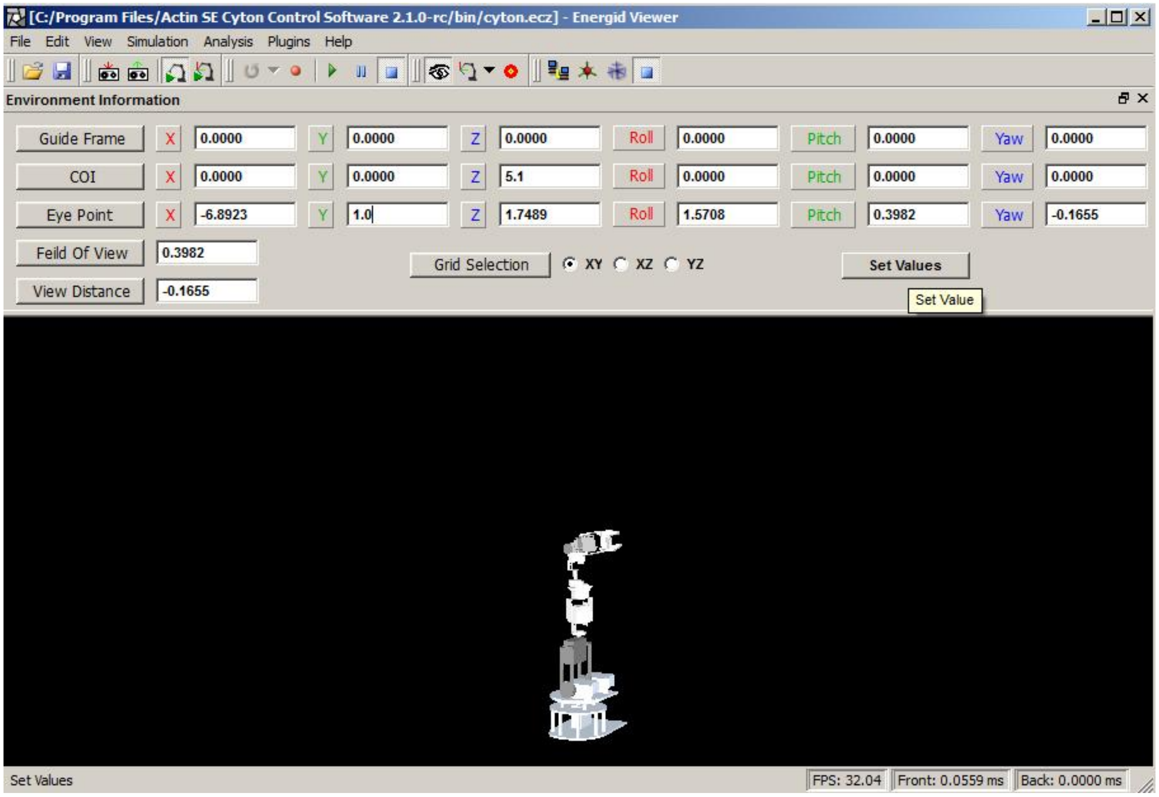


YZ-Grid

The red, green and blue lines are X,Y and Z axis respectively passing through the origin.

When **Set Values** button is clicked, we will be able to edit the Guide Frame location, COI and Eye Point values that are being displayed on the **Environment Information** tab.

A user can edit these values to desired ones and click **Set Values** Set Values in the **Environment Information** tab for it to come into action.



II. How to use the 3 Axis View Plugin

Introduction

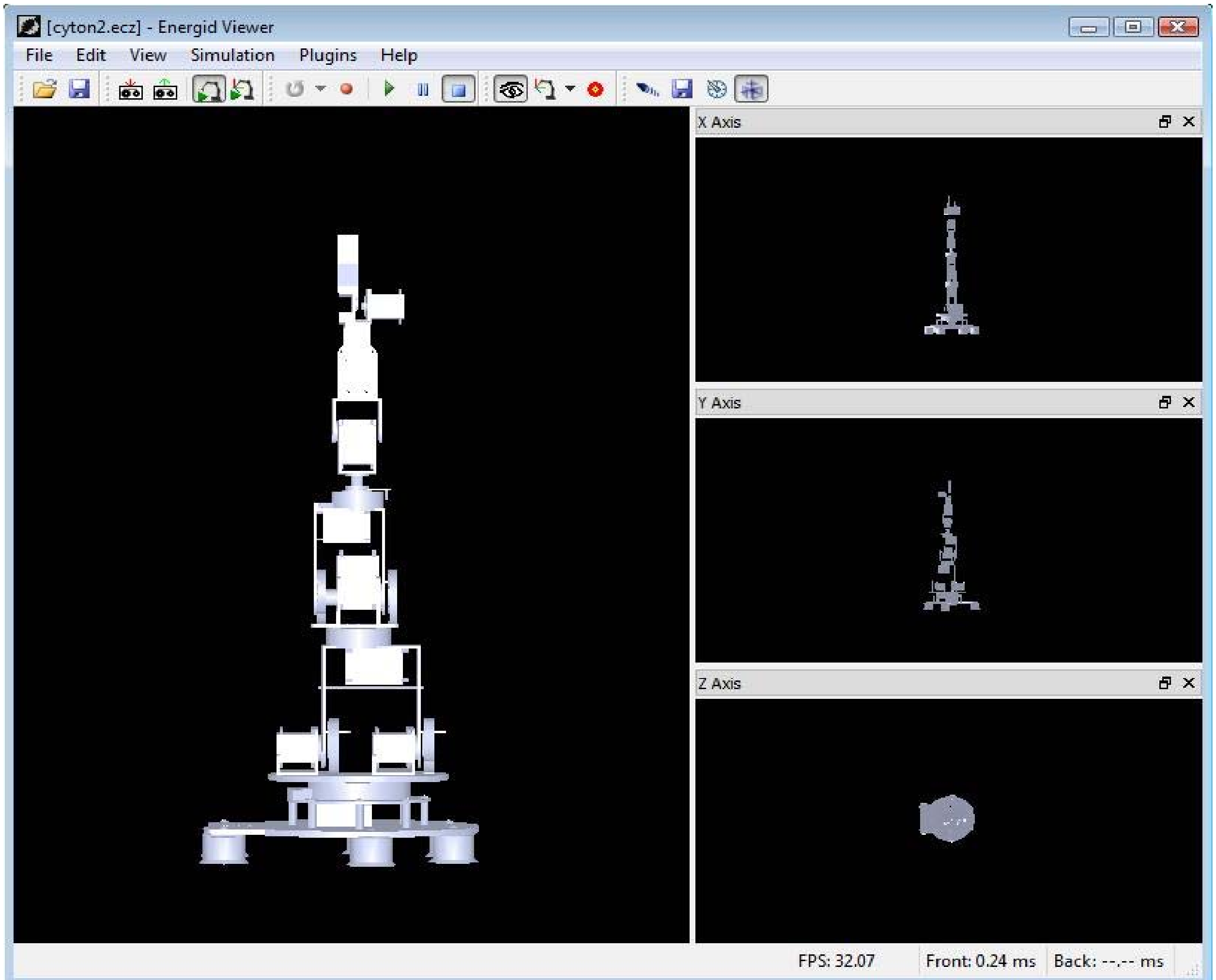
This plugin displays 3 separate views along X, Y and Z axis in the viewer. Using this plugin you will be able to have a better views on the robot and in turn better control of it.

Description

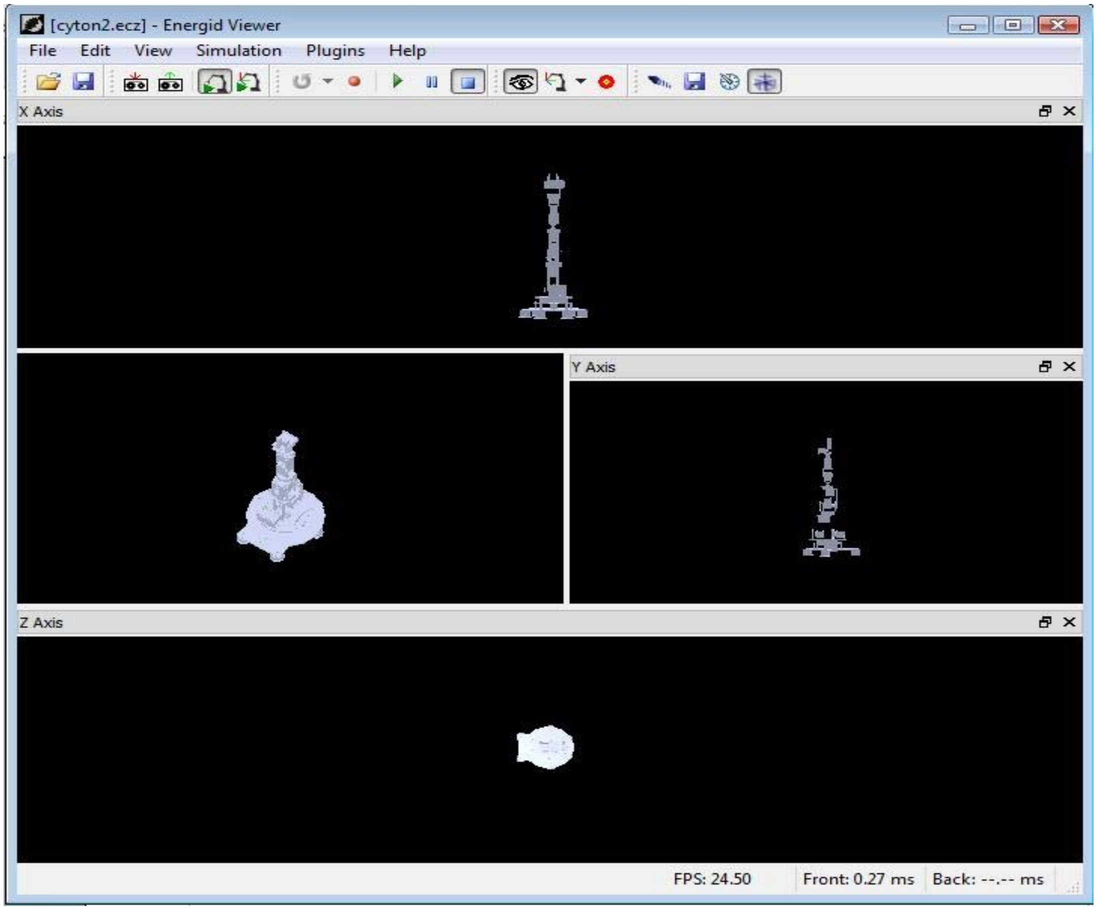
When the plugin is loaded into the viewer; a button called

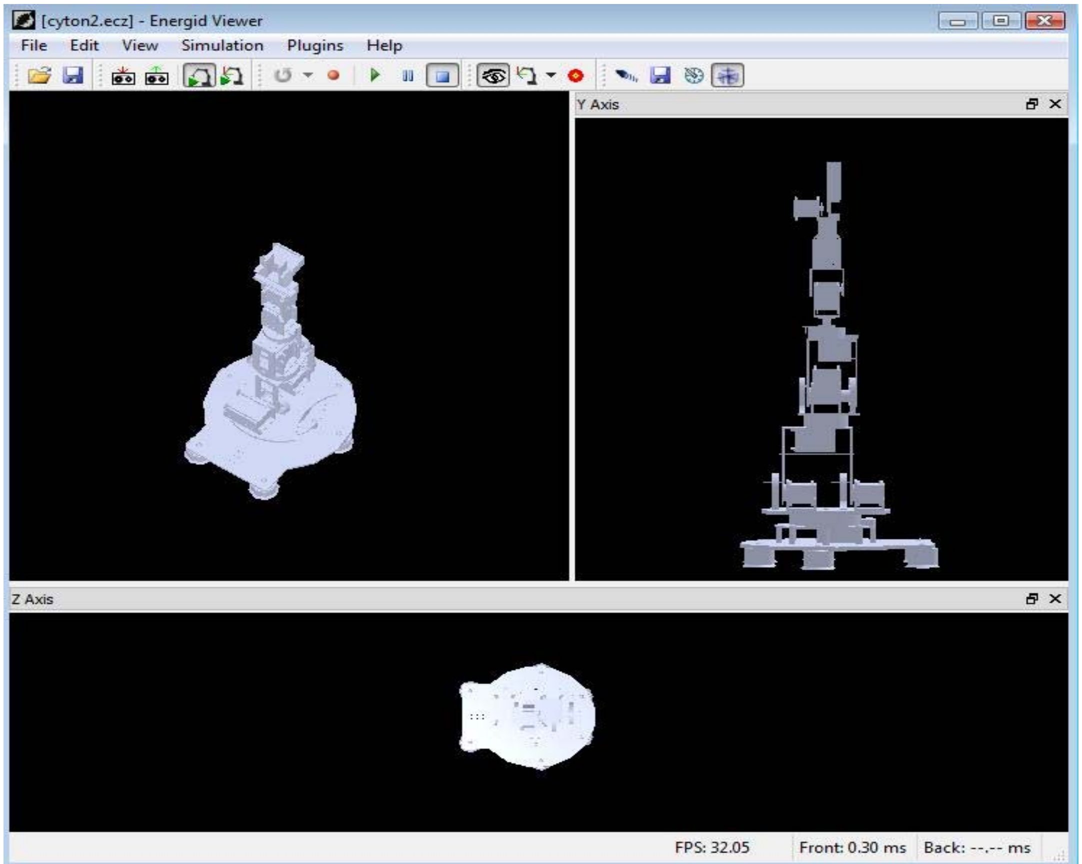
 **Three-Axis View Mode** gets displayed on the toolbar. By clicking this button you will enable Three-Axis View mode in the viewer.

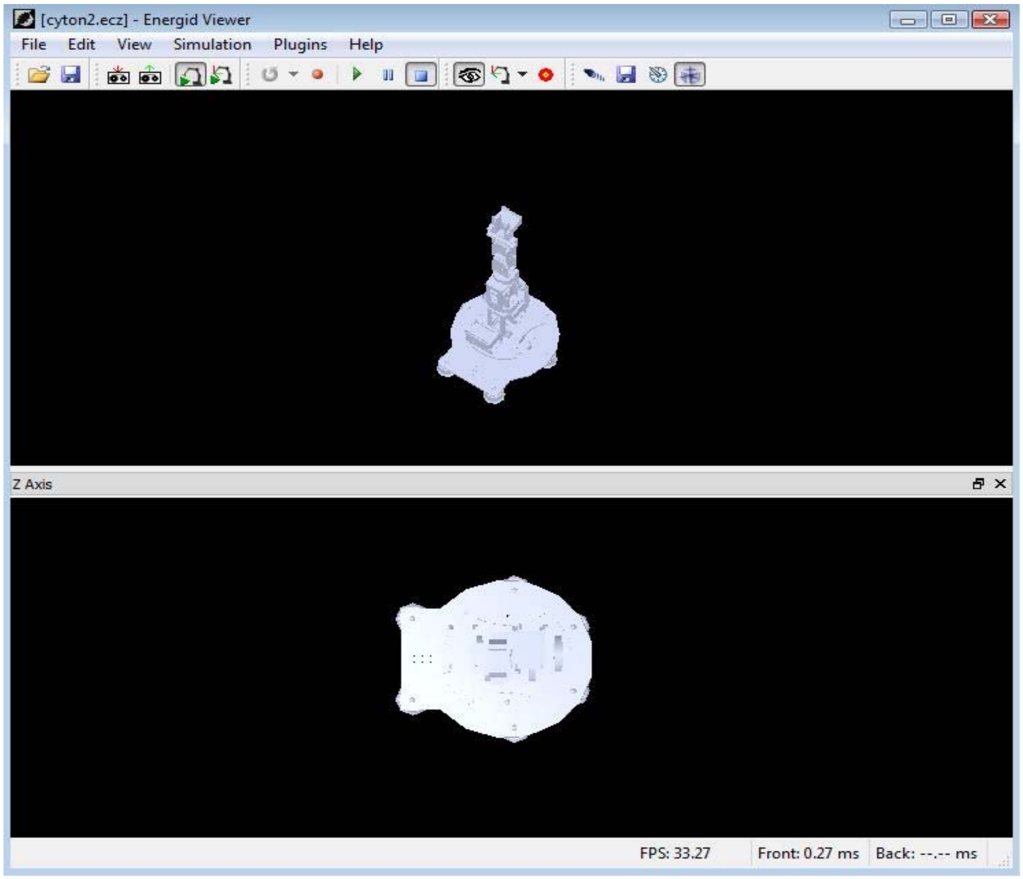
When Three-Axis View mode is enabled, the viewer gets additional windows\widgets which displays views along X, Y and Z axis.



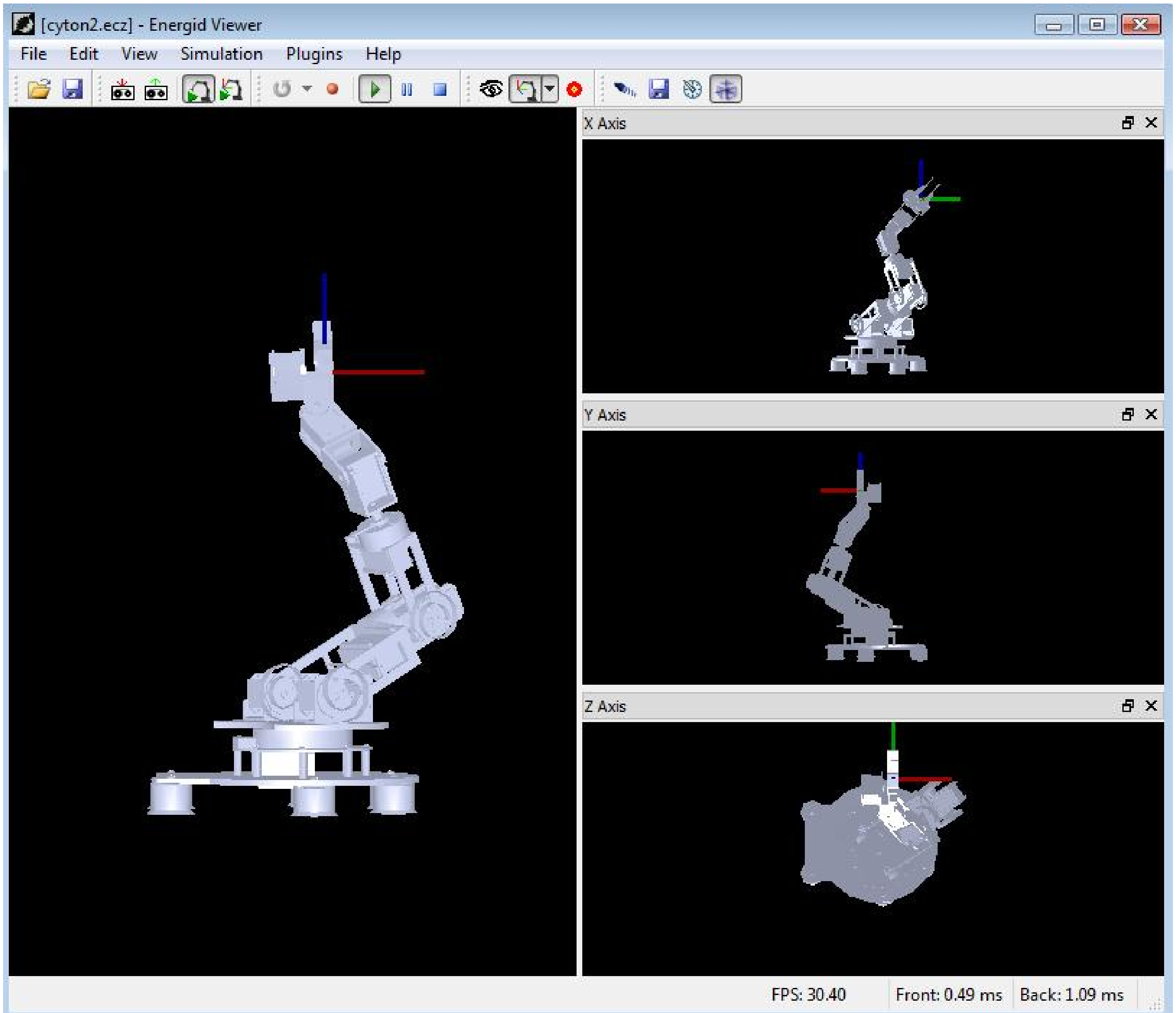
The Eye Point and the Center Of Interest of the robot can only be changed in the main window. However the robot's view can be zoomed in the X Axis, Y Axis and Z Axis windows by using scroll of the mouse. The three axis(X Axis, Y Axis and Z Axis) windows can be removed or separated from the main window; as well as be docked at different locations along the main window as per user convenience.


[cyton2.ecz] - Energid Viewer
File Edit View Simulation Plugins Help
X Axis
Y Axis
Z Axis
FPS: 24.50
Front: 0.27 ms
Back: --.-- ms


[cyton2.ecz] - Energid Viewer
File
Edit
View
Simulation
Plugins
Help
Y Axis
Z Axis
FPS: 32.05
Front: 0.30 ms
Back: --.-- ms



The Guide Frame set on the end effector of the robot can be controlled effectively from three axis(X Axis, Y Axis and Z Axis) windows in addition to the main window. This will help a user to control\move the robot's end effector more precisely to a location.

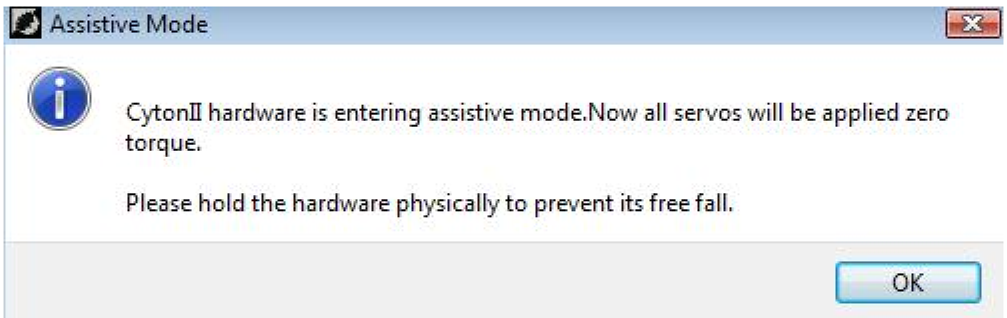

[cyton2.ecz] - Energid Viewer
File
Edit
View
Simulation
Plugins
Help
X Axis
Y Axis
Z Axis
FPS: 30.40
Front: 0.49 ms
Back: 1.09 ms

III. How to use the Assistive\Teach Mode

Introduction
This is a special feature associated with the Cyton arm. In this mode a user can train the arm to move in any desired trajectory to do a task and this trajectory will be recorded by the Cyton viewer as a path file. This recorded path file can be saved and be reused later in time to perform that task again.
Description

 **Assistive\Teach Mode**

When you press this button that appears on the Cyton Viewer toolbar, the cyton arm will enter into assistive\teach mode. Once the button is pressed; following message box will be displayed asking the user to be ready for assistive\teach mode operation.



During assistive/teach mode, all the servos of the arm are applied with zero torque. So if the arm is in an upright position, it could collapse and damage itself. So this message box prepares the user to be ready to hold the arm for assistive\teach mode operation. Once the user presses OK button of this message box, zero torque will be applies to all joints of the arm and the arm enters the assistive\teach mode during which the arm loose its stiffness and it becomes free to move.

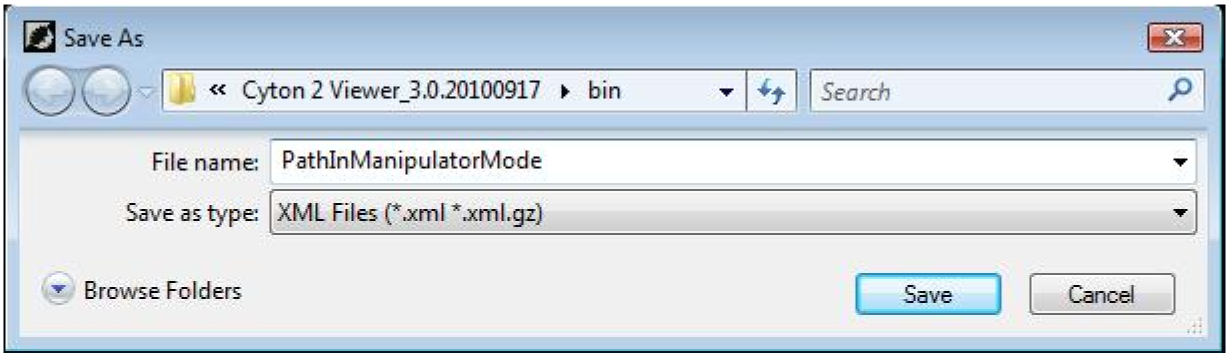
Once the arm is in assistive\teach mode, the user will be able to physically move the arm in desired trajectory or make it perform a task. During assistive\teach mode operation, the movement of the actual arm will be reflected in the Cyton Viewer. The Cyton model in the viewer will reproduce the movement of the actual Cyton hardware in real time.

Once the desired trajectory is traced or task is performed by the arm, the assistive mode can be exited by pressing(or unchecking) the **Assistive\Teach Mode** button on the toolbar of Cyton Viewer.

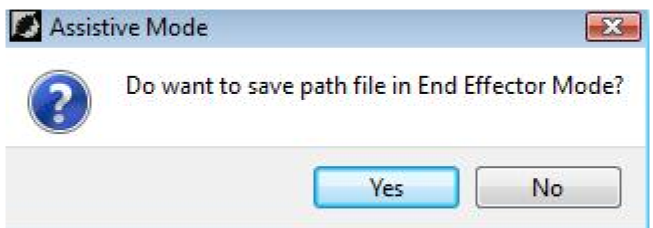
The path file recorded during assistive\teach mode can be saved by clicking

 **Save Assistive Mode Path** button on the toolbar of Cyton Viewer. This button is located next to the **Assistive\Teach Mode** button in the viewer.

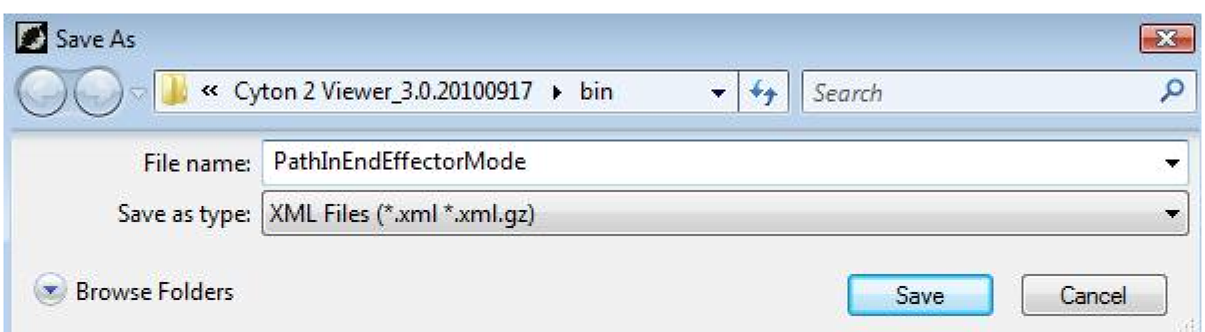
When **Save Assistive Mode Path** button is clicked, the user will be asked the location and file name with which the file has to be saved. This path file saved is recorded in Manipulator\Joint mode.



Once this file is saved, a message box asking whether to save the path file in End Effector mode pops up.



If Yes is clicked, the user will be asked the location and file name with which the file has to be saved. This path file saved is recorded in End Effector Mode.



These path file generated can be treated like normal path files. They can be loaded into viewer and replayed as usual.

Manipulator\Joint Mode Vs End Effector Mode Recording
In Manipulator\Joint mode recording, the joint position(angle) and joint velocity of each joints are recorded into the path file for each time step. When the path is replayed, inverse kinematics computation is not required as joint positions and velocities can be directly feed to the hardware.

In End Effector mode recording, the end effector coordinate and orientation of the arm is recorded for each time step. When the path is replayed, inverse kinematics computation is done to convert end effector coordinate and orientation value to joint positions and velocities for the hardware.

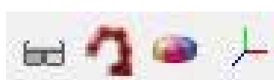
IV. How to use the Overlay Plugin

Introduction

In cyton viewer you can display some information visually by overlaying it on the robot and these information are displayed with the help of Overlay Plugin. The information that are displayed by Overlay Plugin are the following:

a)Bounding Volumes Of Links
b)Mass Distribution Of Links
c)Frames Of Links

When the Overlay Plugin is loaded into the viewer, an overlay toolbar that accomodates 4 buttons are formed on the viewer.



These 4 new buttons formed as part of Overlay Plugin are

 **Stereo**

 **Show Bounding Volume**

 **Show Mass Property Ellipsoids**

 **Show Frames**

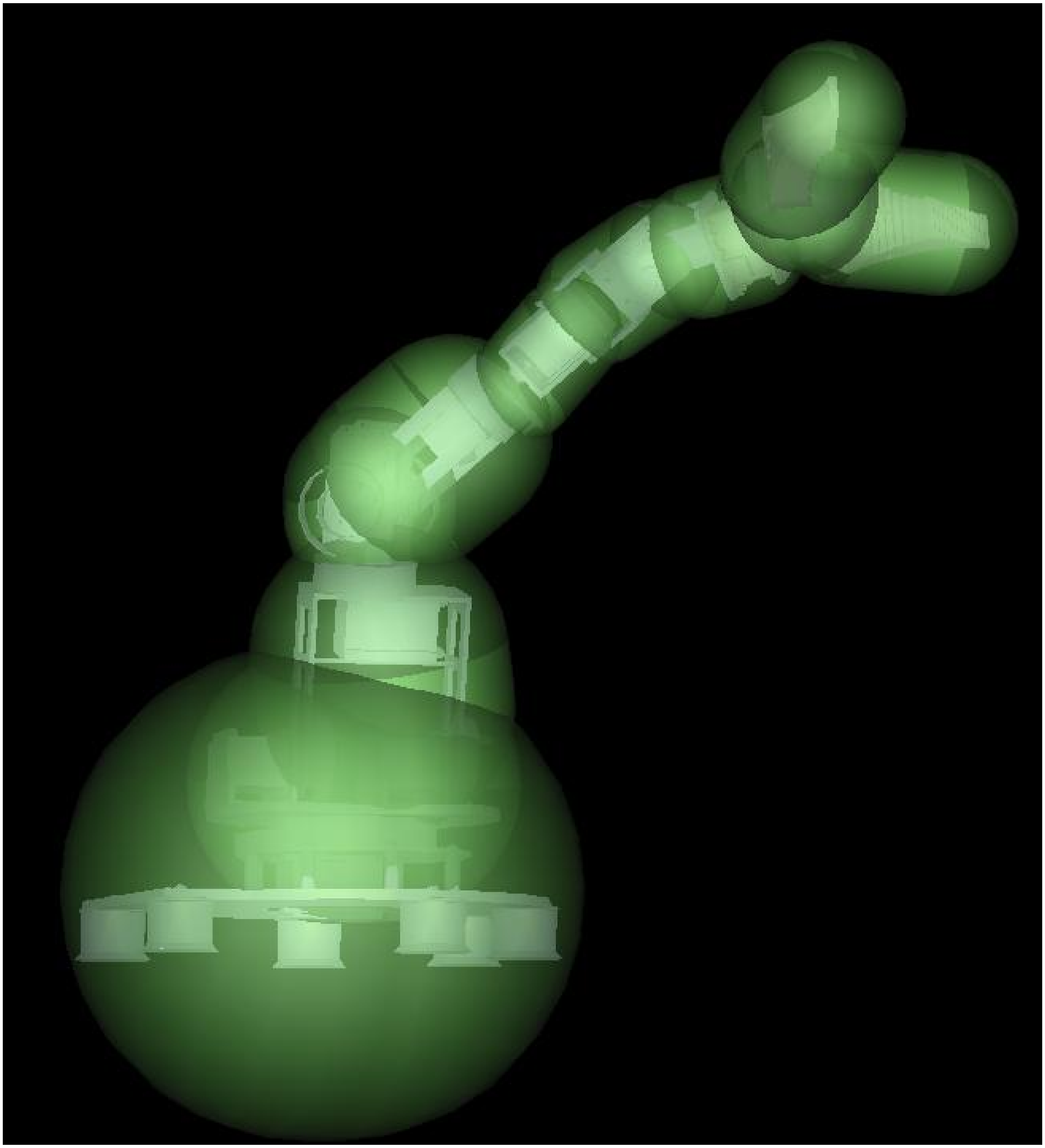
Description

 **Stereo**

Normally Stereo button is disabled, but when a usb shutter glass is connected to your computer, this button will be enabled in the viewer. By clicking the button then, you will be able to view the simulation in the viewer in 3D.

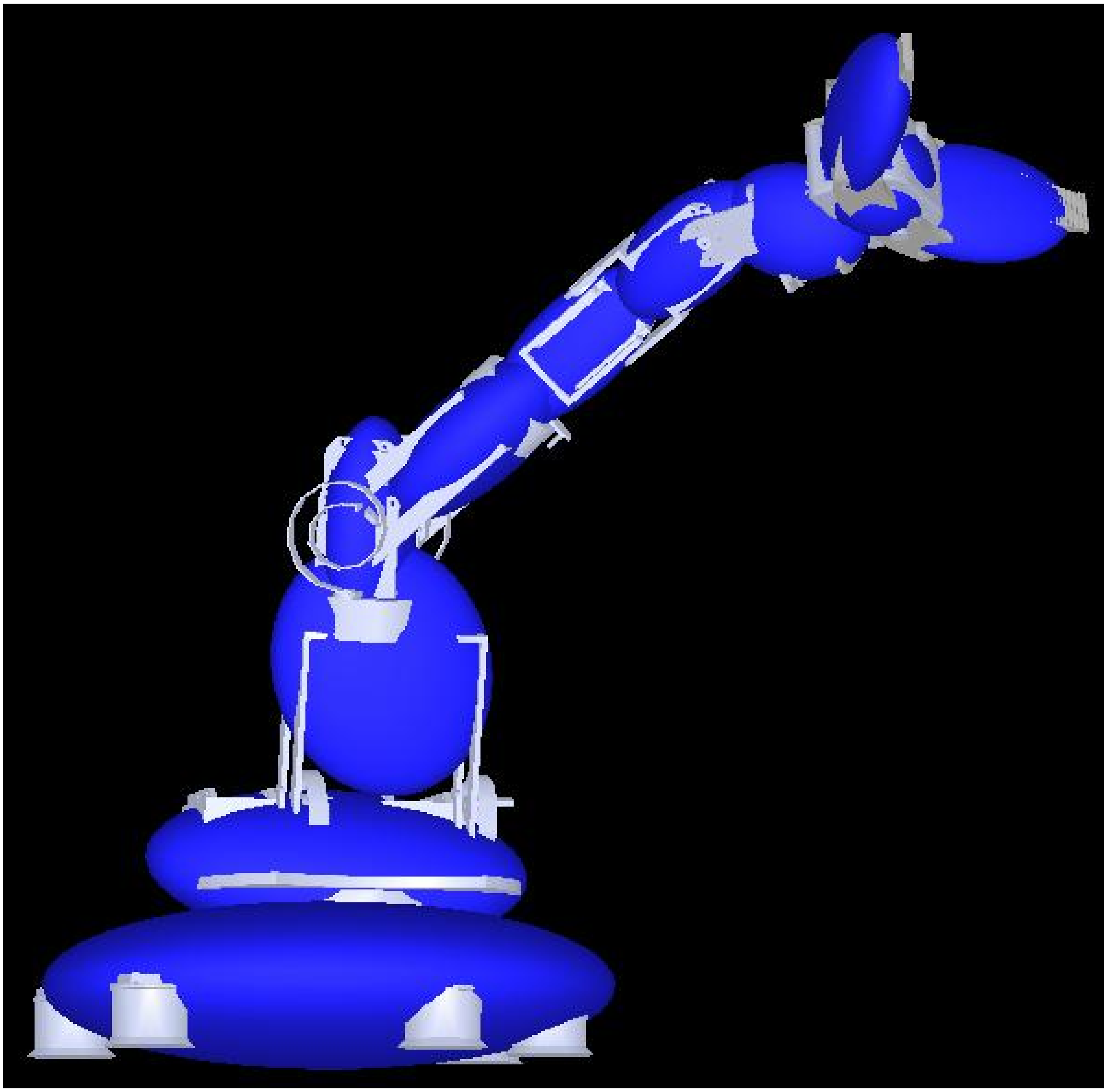
 **Show Bounding Volume**

Pressing the Show Bounding Volume button on the toolbar shows the bounding volumes of all links of all manipulators in the system. Bounding volumes are displayed in a transparent green color. Bounding volumes can be of any shape but the most common is the capsule. Bounding volumes are used in collision reasoning to speed up distance calculations.



 **Show Mass Property Ellipsoids**

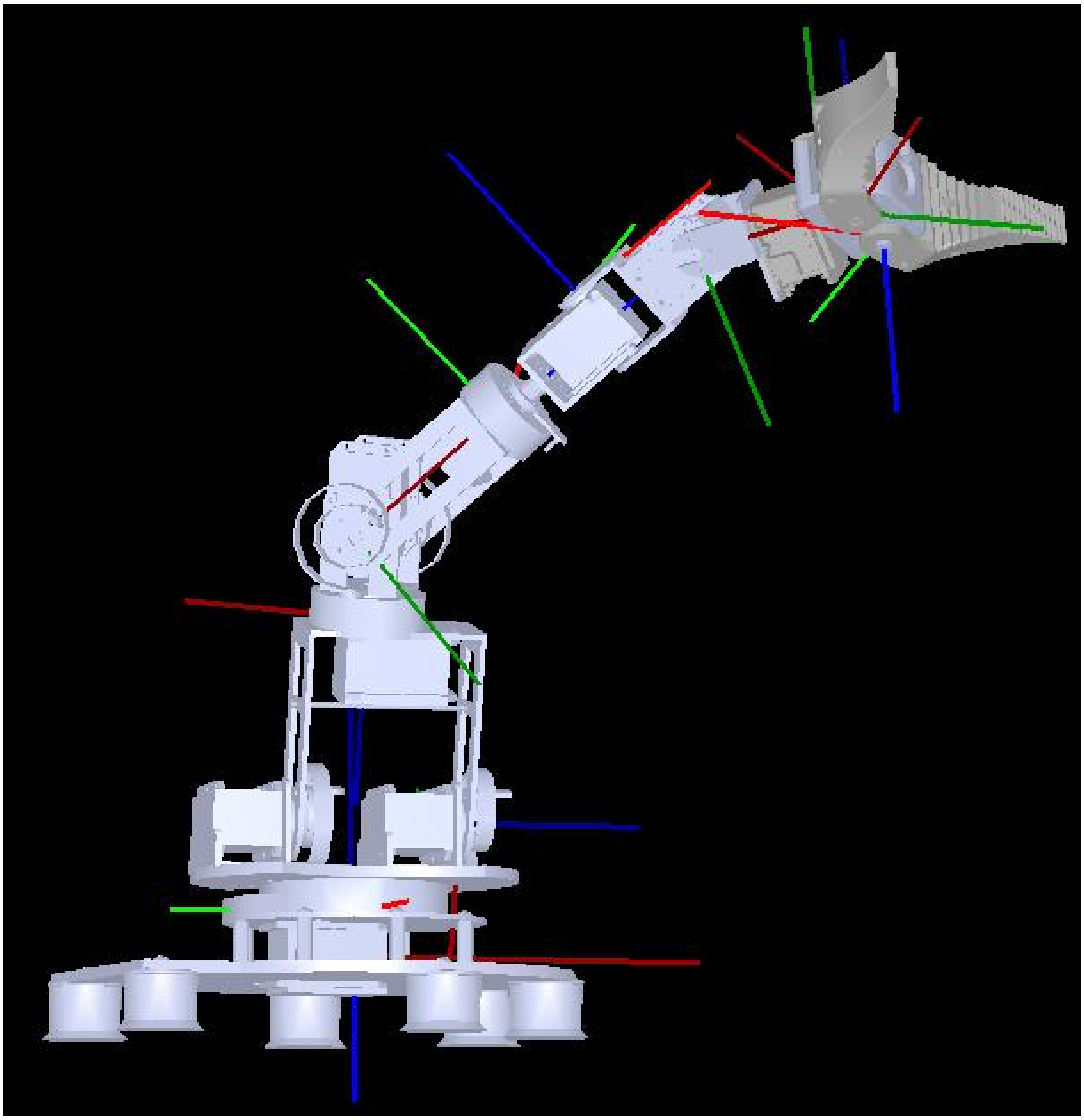
Pressing the Show Mass Property Ellipsoids button will overlay the mass property ellipsoids onto the links of manipulators. This is a great visualization tool to help verify whether the mass properties (inertia matrices) are reasonable.



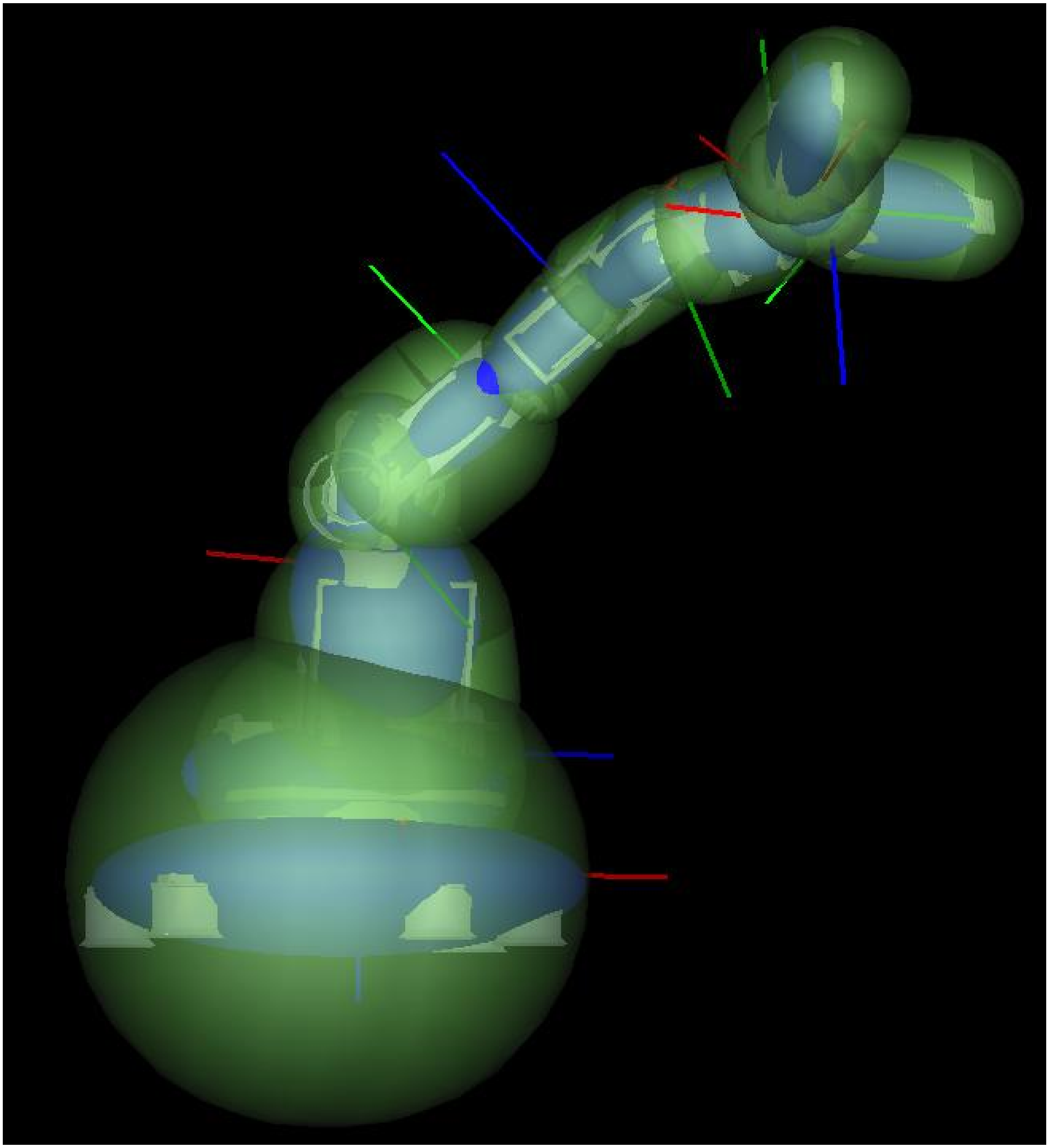
 **Show Frames**

Sometimes it is very insightful to be able to see all the frames (primary and D-H) of the robot. Clicking on the Show Frames button will show all those frames as well as the world coordinate frame. The world coordinate frame is typically designated by the frame with the longest axes.

For all frames, the red, green, and blue axes represent the X, Y, and Z axes, respectively. The primary frame and the D-H frame of a link can occasionally coincide so they will appear as one frame.



Naturally, these information overlays can be turned on/off independent of one anther and therefore can be combined as shown below.

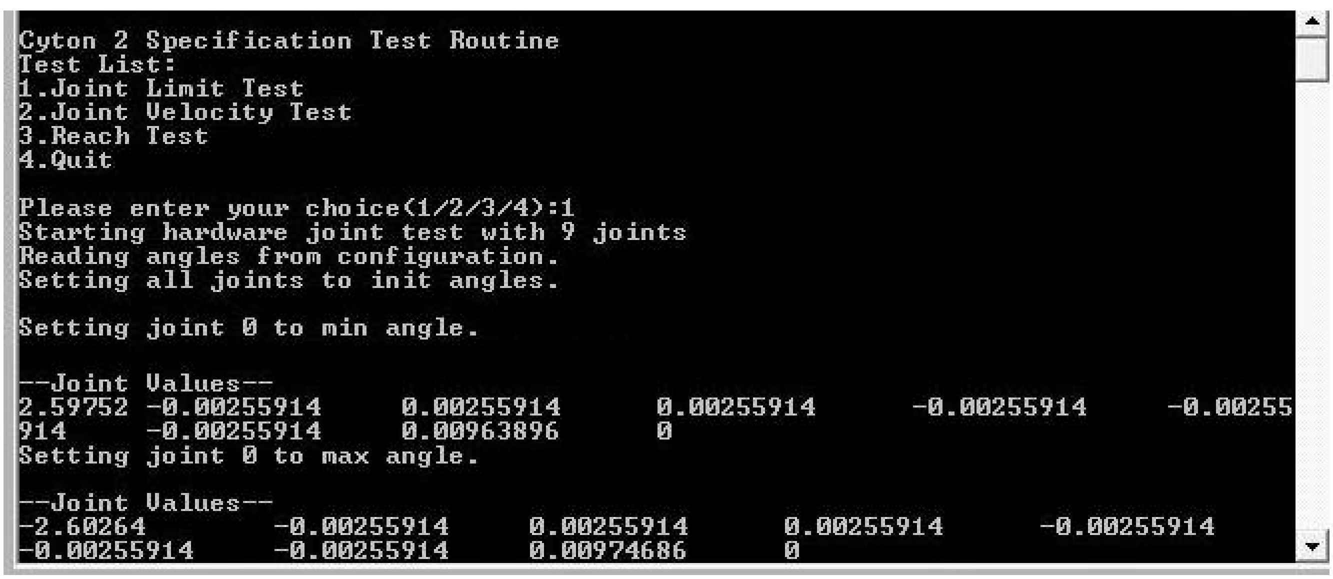


Cyton Spec Tests

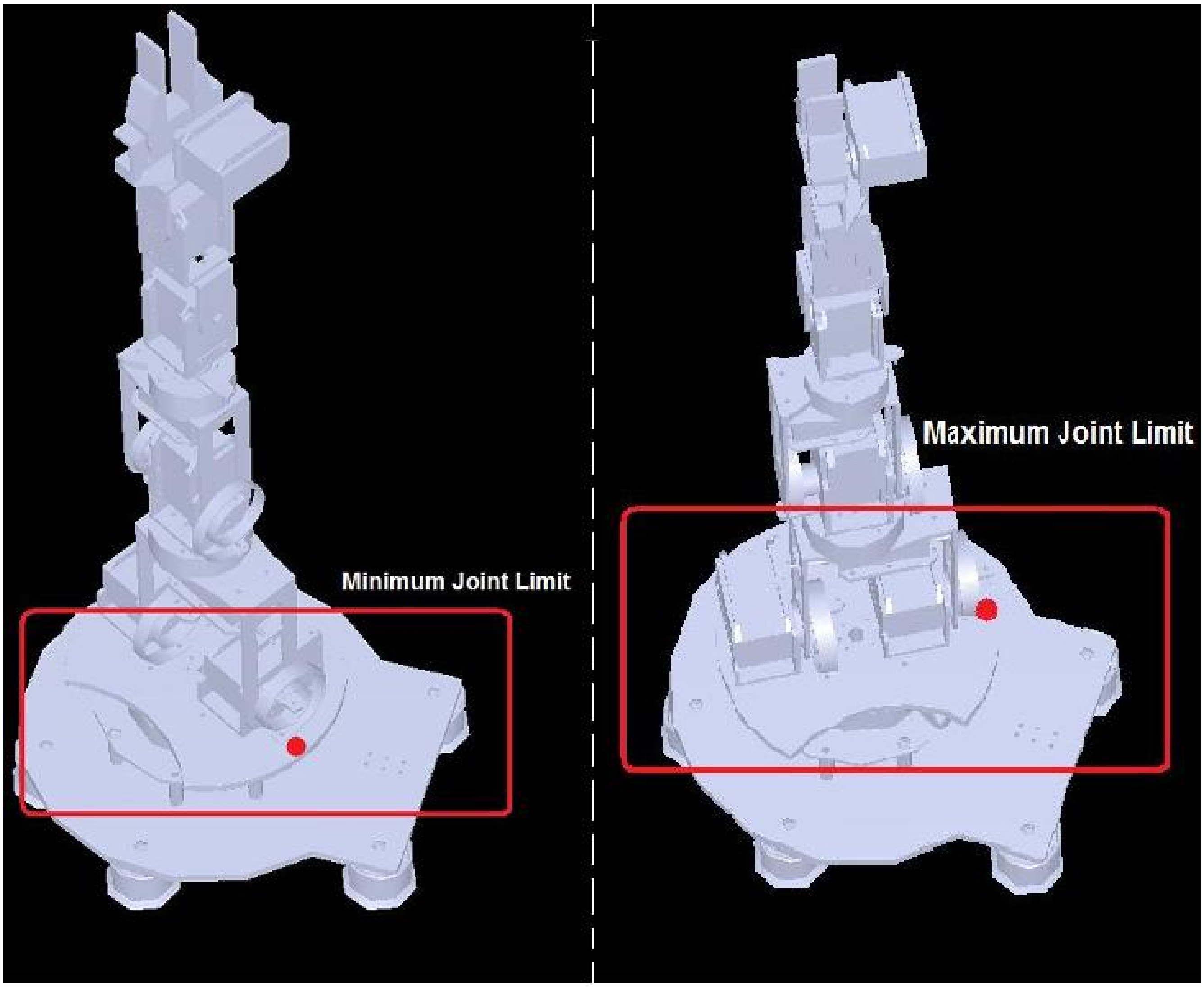
Introduction
A set of tests called spec-tests can be conducted on the Cyton arms in order to verify their specification. The following are the tests presently available for verifying some of the specification of the arm:

- Joint Limit Test
- Joint Velocity Test
- Reach Test

Test Description
1)Joint Limit Test
Joint Limit test is conducted in order to verify the joint limits of all the joints in the Cyton arm. During this test each of the joints of the Cyton arm is made to move to its minimum and maximum joint limits one after the other. When a joint reaches its minimum/maximum joint limit, the joint values of all the joint from the arm are read and displayed on the console screen.


```
Cyton 2 Specification Test Routine
Test List:
1.Joint Limit Test
2.Joint Velocity Test
3.Reach Test
4.Quit

Please enter your choice(1/2/3/4):1
Starting hardware joint test with 9 joints
Reading angles from configuration.
Setting all joints to init angles.

Setting joint 0 to min angle.

--Joint Values--
2.59752 -0.00255914     0.00255914      0.00255914      -0.00255914     -0.00255
914     -0.00255914     0.00963896      0
Setting joint 0 to max angle.

--Joint Values--
-2.60264        -0.00255914     0.00255914      0.00255914      -0.00255914
-0.00255914     -0.00255914     0.00974686      0
```

Screen shot of console screen while Joint Limit test is being run for a Cyton V2.

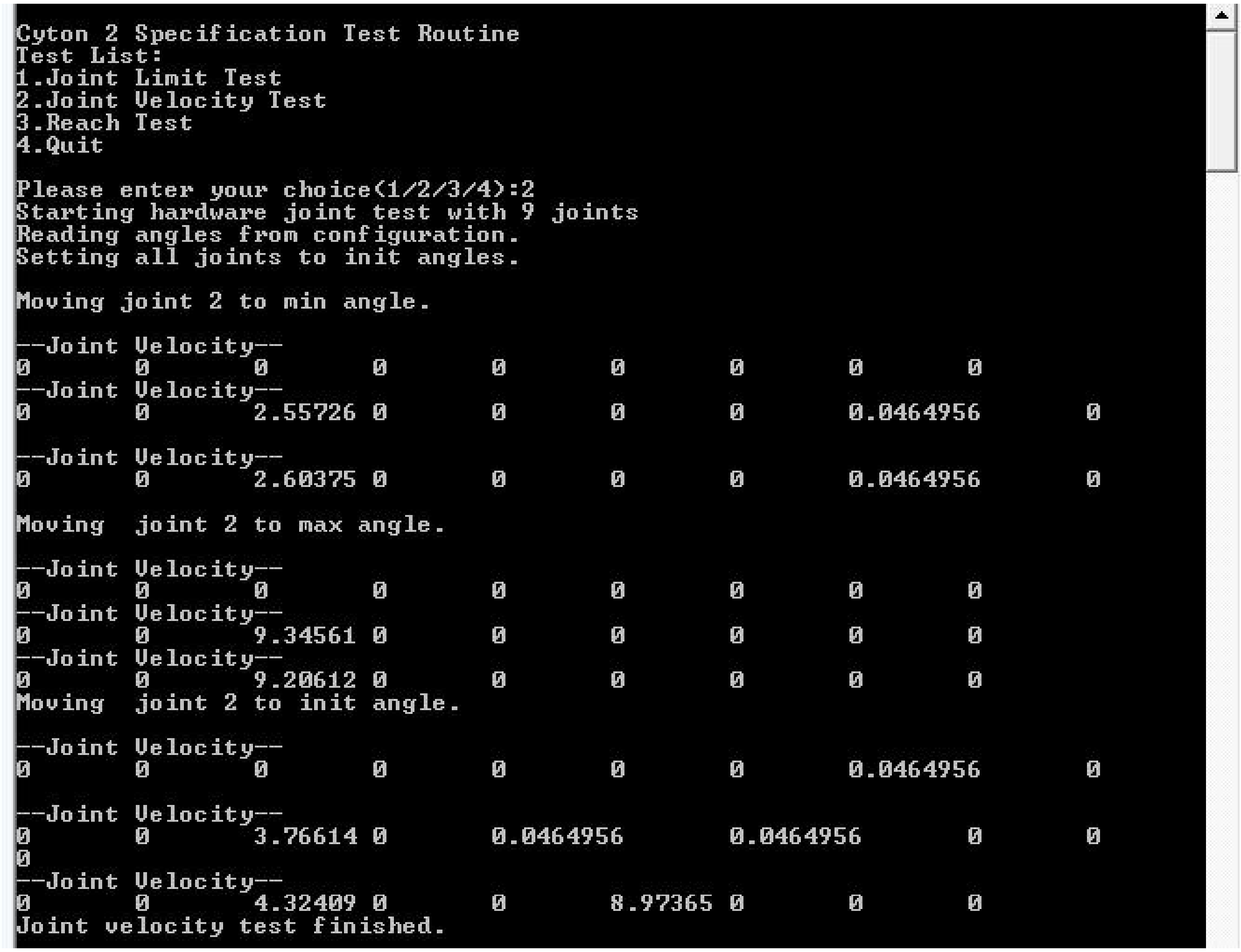


State of hardware based on the values in the above console screen picture.

2) Joint Velocity Test

Joint Velocity test is conducted in order to verify if the hardware is capable achieving the specified joint velocity. During this test, one of the joints of the Cyton V2 arm is made to move at different joint velocities and during the movement, joint velocities of all the joints are read back from the hardware and displayed on the console screen.

In this test we will only be moving the Elbow Roll joint at different speeds between initial to minimum joint limit, minimum to maximum joint limit and maximum to initial joint limit.



```
Cyton 2 Specification Test Routine
Test List:
1.Joint Limit Test
2.Joint Velocity Test
3.Reach Test
4.Quit

Please enter your choice(1/2/3/4):2
Starting hardware joint test with 9 joints
Reading angles from configuration.
Setting all joints to init angles.

Moving joint 2 to min angle.

--Joint Velocity--
0       0       0       0       0       0       0       0       0
--Joint Velocity--
0       0       2.55726 0       0       0       0       0.0464956       0

--Joint Velocity--
0       0       2.60375 0       0       0       0       0.0464956       0

Moving  joint 2 to max angle.

--Joint Velocity--
0       0       0       0       0       0       0       0       0
--Joint Velocity--
0       0       9.34561 0       0       0       0       0       0
--Joint Velocity--
0       0       9.20612 0       0       0       0       0       0
Moving  joint 2 to init angle.

--Joint Velocity--
0       0       0       0       0       0       0       0.0464956       0

--Joint Velocity--
0       0       3.76614 0       0.0464956       0.0464956       0       0
0
--Joint Velocity--
0       0       4.32409 0       0       8.97365 0       0       0
Joint velocity test finished.
```

Screen shot of console screen while Joint Velocity test for a Cyton V2 is run.

3) Reach Test

Reach test is conducted in order to verify the physical reach of the Cyton arm. During this test the Cyton arm is made to stretch itself in four directions(one after the other). During this stretch, the reach of the Cyton arm can be measured using a measuring tape from the base of the Cyton arm.

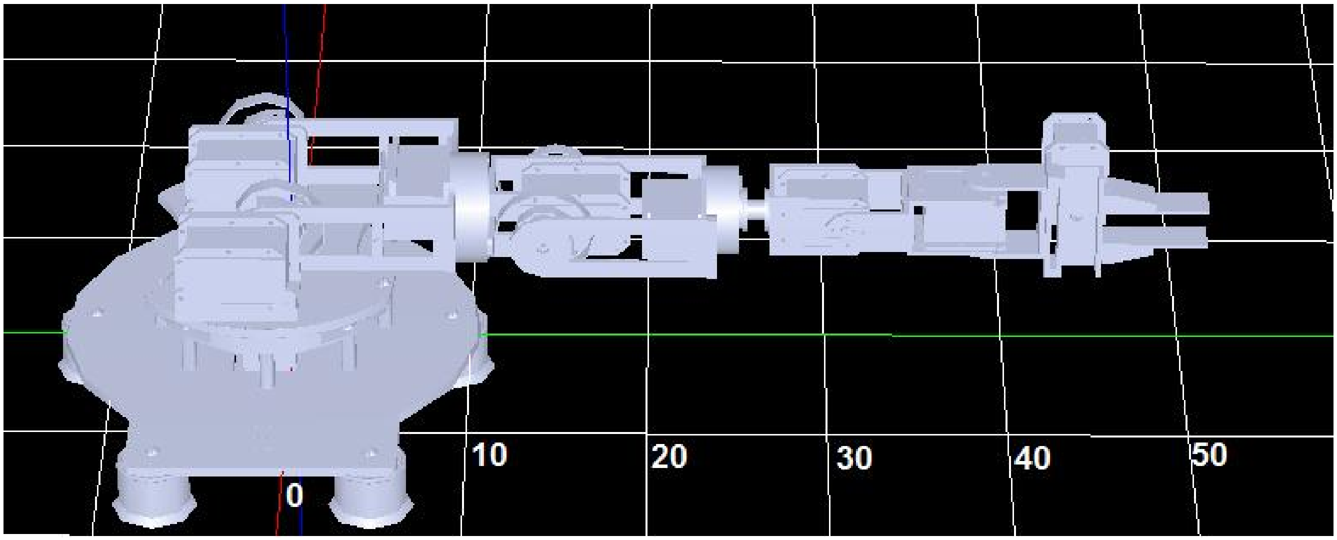
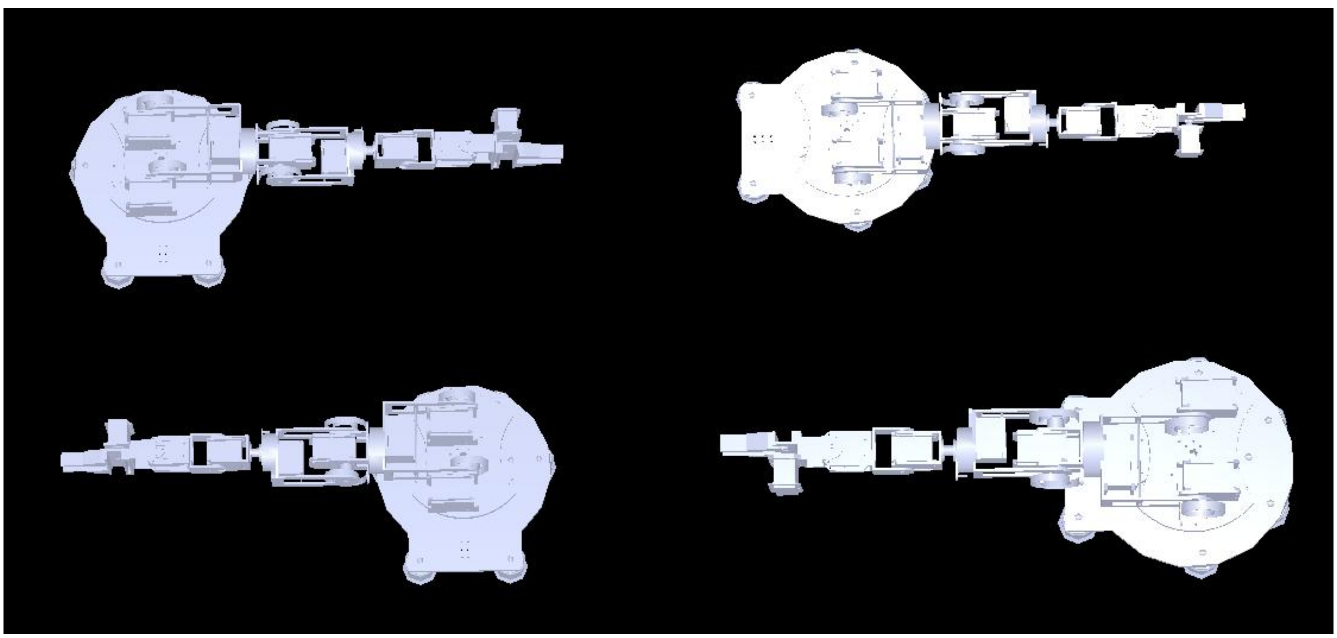


Illustration of how reach can be measured during Reach test.



Four directions in which Cyton V2 arm stretch's during the Reach test.

Using the Cyton C++ API

The available API for the Cyton hardware is made up of two components, 1) a control-based calculation API used to perform inverse-kinematics calculations and other associated commands, and 2) a hardware API to directly control the Cyton arm via code. The control API uses ActinSE, a lightweight and easy to use version of Energid's complete Actin toolkit. Ultimately, the output of ActinSE is a set of joint positions that can then be fed to the hardware API to command a new position. The summary of the available methods for both ActinSE and the hardwareInterface is given in **Appendix A**. Figure 17 below shows the fundamental classes used within ActinSE.

actinSE::Array3	A three-element vector
actinSE::ControlSystem	Basic interface to the Cyton control system
actinSE::CoordinateSystemTransformation	A rotation and a translation describing a new frame
actinSE::EndEffector	End-effector interface
actinSE::Orientation	Description of a 3D rotation

Figure 1:ActinSE fundamental classes

Cyton Code Example

Below is a section of code showing the ActinSE and hardware interface API in use. The complete source code is provided with the installation in examples/cytonControlExample.cpp.

```
EcBoolean
testControlAndHardware
   (
   actinSE::ControlSystem& control,
   cyton::hardwareInterface& hardware,
   const EcU32 endEffectorId
   )
{
   EcRealVector jointAngles;
   EcRealVector jointRates;

   // Pull the starting position.
   CoordinateSystemTransformation initialPose, currentPose;
   EndEffectorVector eeVec;
   control.getParam<ControlSystem::EndEffectors>(eeVec);
   eeVec[endEffectorId].getParam<EndEffector::ActualPose>(initialPose);
   Array3 pos = initialPose.translation();
   std::cout << "Getting current end-effector position " << pos << std::endl;

   // Desired pose is located 10cm away along X and Y.
   CoordinateSystemTransformation desiredPose = initialPose;
   pos = desiredPose.translation() + Array3(0.1,0.1,0.0);
   desiredPose.setTranslation(pos);
   std::cout << "Attempting to move EE(" << endEffectorId <<") to position " << pos <<
std::endl;

   // Set the desired final position.
   EcBoolean passed =
eeVec[endEffectorId].setParam<EndEffector::DesiredPose>(desiredPose);

   // move to the desired pose.  If running the rendered version, it will display the
progress.
```

```
   const EcU32 steps = 200;
   const EcReal simRunTime = 2.0;
   const EcReal simTimeStep = simRunTime/steps;

   for(EcU32 ii=0; ii<steps; ++ii)
   {
      // get the current time
      EcReal currentTime = simTimeStep*ii;

      /// calculate new joint values
      passed = control.calculateToNewTime(currentTime);
      passed &= control.getParam<ControlSystem::JointAngles>(jointAngles);
      if(!passed)
      {
         std::cerr << "Unable to calculate new joint values.\n";
         break;
      }
      std::cout << "Step: " << ii << " Joint Angles: " << jointAngles << "\n";

      // Pass joint values to the hardware.
      passed = hardware.setJointCommands(currentTime, jointAngles,
cyton::JointAngleInRadiansBiasScale);
      if(!passed)
      {
         std::cerr << "Unable to set joint angles to hardware.\n";
         break;
      }

      // Check to see if we have reached our destination
      eeVec[endEffectorId].getParam<EndEffector::ActualPose>(currentPose);
      if(currentPose.approxEq(desiredPose, 1.0e-3))
      {
         std::cout << "Reached goal position in " << currentTime << " seconds (" << ii <<
" steps)\n";
         break;
      }
   }

   return passed;
}
```

Listing-1. Sample code utilizing ActinSE and Cyton hardwareInterface. It attempts to move the designated end-effector 10cm in X and Y. It will use ActinSE to calculate a new position at each timestep, pass it to the hardwareInterface, and then check to see if the position has reached the desired final position.

The Cyton Config file (*CytonConfig.xml*)

In the folder holding the test applications and the Cyton Viewer you will see a file called ***CytonConfig.xml*** This file is used for specifiy the serial port to use and various parameters for calibration. Note that generally all default settings should be sufficient. The format of the file is as follows:

The Cyton2Config.xml file contains values for various parameters for each servo in Cyton arm. Each servo in the arm is configured based on these values. The parameters in the config file are explained below:

```
m_ftdiSerialNum    : Serial no. of the USB2Dynamixel.
m_bps              : Baud rate in bits per second for communication.
m_readTimeoutms    : Read time out in milliseconds.
m_writeTimeoutms   : Write time out in milliseconds.
m_positionThreadHz : Speed of refreshing status data from dynamixel in hertz.
m_openLoop         : Flag for open and closed loop operation. When set 1,
                     operates in open loop. When set 0,operates in closed loop.
m_realTime         : If this flag is set as 1, joint rate will be computed.If
                     the flag is set as 0, joint rate will be taken from the
                     config XML.
count              : Number of servos connected.
```

m_id : Servo id.
m_goalPosition : Goal position to move to while configuring the servos in radians. Value range from -2.617 to 2.617 radians(ie from -150 to 150 degrees).
m_LEDState : Sets the required LED state. When set 0,LED turns OFF. When set 1,LED turns ON.
m_returnDelay : Sets the return delay time in milliseconds. Value ranges from 0 to 500 ms.
m_cwlimit : Sets clockwise angle limit in radians. Value ranges from -2.617 to 2.617 radians(ie from -150 to 150 degrees).
m_ccwlimit : Sets counter clockwise angle limit in radians. Value ranges from -2.617 to 2.617 radians(ie from -150 to 150 degrees).
m_angleOffset : Sets an angular offset to the servo. The value is set in radians and can be positive or negative. However using angular offset can reduce the angular working range of the servo by the offset applied.
m_maxTorque : Sets percentage of torque limit to be used. It only gets set on re-powering the servo.
m_movingSpeed : Velocity with which servos move while the arm is controlled in radians per second. Value ranges from 0 to 11.9 radians per second (ie 0 to 114 RPM).
m_setupMovingSpeed : Velocity with which servos move during startup and stop in radians per second.Its value ranges from 0 to 11.9 radians per second (ie 0 to 114 RPM).
m_torqueLimit : Sets percentage of torque limit to be used. The Value gets set immediately as it is set unlike m_maxTorque.
m_baudRate : Baud rate in bits per second for the servo. It must be same as that of the m_bps.
m_tempLimit : Sets the temperature limit in degree Celsius. Value ranges from 10 to 99 degree Celsius.
m_torqueEnable : Flag to enable and disable torque. When set 1,torque is enabled. When set 0,torque is disabled or is in 0 torque mode.
m_lvlimit : Sets lower voltage limit. Values range from 5 to 25V.
m_alarmLED : Sets alarm LED. Values range from 0 to 126.
m_alarmShutdown : Sets alarm shutdown. Values range from 0 to 126.
m_hvlimit : Sets higher voltage limit. Values range from 5 to 25V.
m_srlevel : Sets status return level. When set 0,no status return. When set 1,status return only for read packet. When set 2,status return for all packets.
m_punch : Sets punch for dynamixel servos. Values range from 0 to 1023.
m_cwcm : Sets clockwise compliance margin. Value ranges from 0 to 254.
m_ccwcm : Sets counter clockwise compliance margin. Value ranges from 0 to 254.
m_cwcs : Sets clockwise compliance slope. Values range from 0 to 254.
m_ccwcs : Sets counter clockwise compliance slope. Value ranges from 0 to 254.
m_lock : Sets lock on dynamixel servos. When set 1, restricts writing data into dynamixel from address location 0x18 to 0x23.
m_invert : Sets invert flag. When set 1,servos move in the actual direction. When set -1,servos move in the reverse

direction.

m_map : Used to map joint angle and joint velocity of one servo to another. If map of a servo with id 4 is set as 3 then servo with id 4 will function same as the servo with id 3.

m_servoType : It’s a flag indicating the type of the servo used. If its value is 0,it indicated RX/DX series servos are used. If its value is 1,it indicates EX series servos are used.

m_isGripper : It’s a flag indicating if the servo is gripper servo. If its value is 0,it indicates it is not a gripper servo. If its value is 1,it indicates it is a gripper servo.

For more information on these parameters please see the manual provided by the manufacturer. These can be obtained for the RX-10, Rx-28, and RX-64 servos at http://www.robotis.com/zbxe/dynamixel_en .

Tech Support and Contact Info

For tech support contact :

By email:

support@robai.com

By phone:

412-307-3050
(between 9 a.m. and 5 p.m. Eastern Standard Time)

By standard mail:

Robai
PO Box 37635 #60466
Philadelphia, PA 19101-0635

www.robai.com

Appendix A: Software API and Class Reference

This appendix describes the C++ API delivered with the Cyton robots. An HTML version of the class reference is kept online at:

http://robai.com/php/developers.php?linkid=4

---

cyton::hardwareInterface

cyton::hardwareInterface Class Reference

Public Types

enum	OverrideEnum { OverrideNone = 0x0, OverridePort = 0x1, OverrideReset = 0x2 }
typedef EcU32	OverrideType Variable type to hold override options.

Public Member Functions

	hardwareInterface (const EcString &pluginName, const EcString &configFile="")
virtual	~hardwareInterface ()
virtual void	setPort (const EcString &port)
virtual EcStringVector	availablePorts () const
virtual void	setResetOnShutdown (const EcBoolean resetOnShutdown)
virtual EcBoolean	resetOnShutdown () const
virtual EcBoolean	init ()
virtual EcBoolean	reset ()
virtual EcBoolean	shutdown ()
virtual EcBoolean	setJointCommands (const EcReal timeNow, const EcRealVector &jointCommands, const StateType stateType=JointAngleInRadians, const EcRealVector &jointVelocities=EcRealVector())
virtual EcBoolean	getJointStates (EcRealVector &jointStates, const StateType stateType=JointAngleInRadians) const
virtual EcBoolean	waitUntilCommandFinished (const EcU32 timeoutInMS) const
virtual EcU32	numJoints () const
virtual EcBoolean	setLowRate (const EcBoolean lowRate)
Ec::Plugin *	plugin ()

Detailed Description

Member Typedef Documentation

typedef EcU32 cyton::hardwareInterface::OverrideType

Variable type to hold override options.

Member Enumeration Documentation

enum cyton::hardwareInterface::OverrideEnum

Bitfield options that hold whether or not to override parameters within the configuration file when initializing hardware. Currently the only override parameters are the port device and resetOnShutdown flag.

Enumerator:

OverrideNone	Pull everything from config file.
OverridePort	User specified port.
OverrideReset	User specified reset.

Constructor & Destructor Documentation

cyton::hardwareInterface::hardwareInterface (const EcString & pluginName, const EcString & configFile = "")

Constructor. Does not initialize hardware.

Parameters:

[in]	*pluginName*	Name of hardware plugin to utilize
[in]	*configFile*	Optional hardware configuration file

virtual cyton::hardwareInterface::~hardwareInterface ( )

Destructor. Shuts down device driver if loaded.

Member Function Documentation

virtual EcStringVector cyton::hardwareInterface::availablePorts ( ) const

Examine current hardware configuration to list available ports.

Returns:

EcStringVector A vector of string representing the port names of the devices available. Platform dependent. Empty list returned if not available, or plugin not loaded.

virtual EcBoolean cyton::hardwareInterface::getJointStates (EcRealVector & jointStates, const StateType stateType = JointAngleInRadians)

Retrieve servo information. Depending on the stateType parameter it will return the last commanded position (default) or any of the configuration parameters for the servos (joint bias, min angle, max angle, reset angle, max joint rate, joint scale).

Parameters:

[out]	*jointStates*	Vector of returned values
[in]	*stateType*	Type and unit of requested values

Returns:

EcBoolean Success or failure of query command

virtual EcBoolean cyton::hardwareInterface::init ( )

Initialize hardware, which includes reading in configuration file, opening the port and resetting hardware to a known good state.

Returns:

EcBoolean Success or failure of initialization

virtual EcU32 cyton::hardwareInterface::numJoints ( ) const

Retrieve the number of joints currently configured.

Returns:

EcU32 Number of joints in the loaded system

Ec::Plugin* cyton::hardwareInterface::plugin ( )

Retrieve a handle to the loaded plugin.

Returns:

Ec::Plugin* The loaded plugin

virtual EcBoolean cyton::hardwareInterface::reset ( )

Send a reset command to the hardware to move joints back to resting position.

Returns:

EcBoolean Success or failure of reset command

virtual EcBoolean cyton::hardwareInterface::resetOnShutdown ( ) const

Accessor to retrieve state of whether reset will occur before power down.

Returns:

EcBoolean EcTrue if a reset will occur or EcFalse if not

virtual EcBoolean cyton::hardwareInterface::setJointCommands (const EcReal timeNow,
const EcRealVector & jointCommands,
const StateType stateType = **JointAngleInRadians**,
const EcRealVector & jointVelocities = **EcRealVector()**
)

Sends commands to Cyton hardware to move joints to a specified location. A time difference is calculated from the previous command to determine the rate at which to move the joints.

Parameters:

[in]	*timeNow*	Current time
[in]	*jointCommands*	Vector of joint angles to move servos to
[in]	*stateType*	Optional unit conversion for input jointCommands
[in]	*jointVelocities*	Vector of joint velocities

Returns:

EcBoolean Success or failure of set command

virtual EcBoolean cyton::hardwareInterface::setLowRate (const EcBoolean lowRate **)**

Give the ability to rate limit the joints. If enabled, it will limit the arm at 25% of max rate.

Parameters:

[in] *lowRate* Turn rate limiting on or off

Returns:

EcBoolean Success or failure of command

virtual void cyton::hardwareInterface::setPort (const EcString & port **)**

Specify a port to use for the connection to the hardware.

Parameters:

[in]	*port*	String name of port to use. Platform dependent

virtual void cyton::hardwareInterface::setResetOnShutdown (const EcBoolean resetOnShutdown)

Flag indicating whether or not to reset Cyton joints to their initialization position before powering down.

Parameters:

[in]	*resetOnShutdown*	Whether or not to reset on power down

virtual EcBoolean cyton::hardwareInterface::shutdown ( )

Unloads plugin device driver.

Returns:

EcBoolean Success or failure of shutdown command

virtual EcBoolean cyton::hardwareInterface::waitUntilCommandFinished (const EcU32 timeoutInMS)

Wait for the last command to finish, up to a specified maximum time in milliseconds.

Parameters:

[in] *timeoutInMS* Maximum time to wait in milliseconds before failing

Returns:

EcBoolean Success or failure of wait command

---

actinSE::ControlSystem Class Reference

Public Types

enum **ParamTypeEnum** { **Rendering**, **SimulationTime**, **JointAngles**, **JointPose**, **JointVelocities**, **BasePose**, **EndEffectors**, **CraigDHParameters** }

Public Member Functions

	ControlSystem () Constructor.
	~ControlSystem () Destructor.
	ControlSystem (const **ControlSystem** &orig)
ControlSystem &	**operator=** (const **ControlSystem** &orig)
EcBoolean	**operator==** (const **ControlSystem** &rhs) const
EcBoolean	**loadFromFile** (const **EcString** &fileName)
EcBoolean	**saveToFile** (const **EcString** &fileName)
template<ParamTypeEnum prm, typename ParamType > **EcBoolean**	**setParam** (const ParamType &value)
template<ParamTypeEnum prm, typename ParamType > **EcBoolean**	**getParam** (ParamType &value) const
template<ParamTypeEnum prm, typename ParamType > **EcBoolean**	**getParam** (ParamType &value, const **EcU32** subIndex) const
template<ParamTypeEnum prm, typename ParamType > const ParamType &	**param** () const
template<ParamTypeEnum prm, typename ParamType > ParamType	**param** ()
EcBoolean	**reset** ()
EcBoolean	**calculateToNewTime** (const **EcReal** timeInSeconds)

actinSE::EndEffector Class Reference

Public Types

enum	**EETypeEnum** {
	UnknownEndEffector = 0,
	PointEndEffector,
	OrientationEndEffector,
	FrameEndEffector,
	LinearConstraintEndEffector
	}

enum	**ParamTypeEnum** {
	DegreesOfConstraint,
	ActualPose,
	RelativeLink,
	MotionThreshold,
	DesiredPose,
	DesiredVelocity,
	Gain,
	HardConstraint
	}

enum	**EEStateFlagsEnum** { **EmptyStateFlags** = 0x0,

	RelativeLinkFlag = 0x1, **HardConstrained** = 0x2, **Attached** = 0x4 }
typedef **EcU32**	**EEStateFlags**

Public Member Functions

	EndEffector (const **EETypeEnum** eeType)
	~EndEffector () Destructor.
	EndEffector (const **EndEffector** &orig)
EndEffector &	**operator=** (const **EndEffector** &orig)
EcBoolean	**operator==** (const **EndEffector** &rhs) const
template<ParamTypeEnum prm, typename ParamType > **EcBoolean**	**setParam** (const ParamType &value)
template<ParamTypeEnum prm, typename ParamType > **EcBoolean**	**getParam** (ParamType &value) const
template<ParamTypeEnum prm, typename ParamType > ParamType	**param** ()
EEStateFlags	**stateFlags** () const
EETypeEnum	**endEffectorType** () const
EcString	**name** () const

actinSE::Array3 Class Reference

Public Member Functions

	Array3 () constructor
	Array3 (const **EcReal** x, const **EcReal** y, const **EcReal** z) constructor from three reals
	~Array3 () destructor
	Array3 (const **Array3** &orig) copy constructor
Array3 &	**operator=** (const **Array3** &orig) assignment operator
EcBoolean	**operator==** (const **Array3** &orig) const equality operator
EcBoolean	**operator!=** (const **Array3** &orig) const inequality operator
Array3 &	**operator+=** (const **Array3** &v2) add another vector to this vector and set this vector to the result
Array3 &	**operator-=** (const **Array3** &v2) subtract another vector from this vector and set this

	vector to the result
Array3 &	**operator*=** (**EcReal** s) multiply this vector times a scalar and set this vector to the result
Array3	**operator+** (const **Array3** &v2) const returns a vector equal to this vector plus another
Array3	**operator-** (const **Array3** &v2) const returns a vector equal to this vector minus another
Array3	**operator*** (const **EcReal** a) const
Array3	**operator/** (const **EcReal** a) const
Array3	**cross** (const **Array3** &v2) const returns a vector equal to this vector cross another (vector cross product)
EcReal	**dot** (const **Array3** &v2) const returns a vector equal to this vector dot another (vector dot product)
EcReal	**mag** () const returns the magnitude of this vector
EcReal	**prod** () const returns the product of the three elements
EcReal	**magSquared** () const returns the magnitude squared of this vector (a fast operation)
Array3	**unitVector** () const returns a unit vector in the same direction as this vector
Array3 &	**normalize** () normalizes this vector
EcBoolean	**approxEq** (const **Array3** &v2, const **EcReal** tol) const tests that each element of this vector is within a tolerance of another
EcReal	**distanceTo** (const **Array3** &vec) const find the Euclidean distance to another point
EcReal	**distanceSquaredTo** (const **Array3** &vec) const find the Euclidean distance squared to another point
void	**computeDirectionalVector** (const **Array3** &destination, const **EcReal** mag, **Array3** &result) const compute a vector which points from this vector (point) to the other vector (point) with a given magnitude.
void	**set** (const **EcReal** x, const **EcReal** y, const **EcReal** z) sets the z value of the vector
const EcReal &	**operator[]** (const **EcU32** index) const returns a value by index (0, 1, or 2) - const version.
EcReal &	**operator[]** (const **EcU32** index) returns a value by index (0, 1, or 2) - nonconst version.

actinSE::CoordinateSystemTransformation Class Reference

Public Types

enum	**ModeEnum** { **NO_CHANGE**, **ARBITRARY**, **NO_TRANSLATION**, **NO_ROTATION** }

Public Member Functions

	CoordinateSystemTransformation () Default constructor.
	CoordinateSystemTransformation (const **Array3** &trans, const **Orientation** &orient)
	CoordinateSystemTransformation (const **Array3** &trans)
	CoordinateSystemTransformation (const **Orientation** &orient)
	~CoordinateSystemTransformation () Destructor.
	CoordinateSystemTransformation (const **CoordinateSystemTransformation** &orig)
CoordinateSystemTransformation &	**operator=** (const **CoordinateSystemTransformation** &orig)
EcBoolean	**operator==** (const **CoordinateSystemTransformation** &orig) const
ModeEnum	**mode** () const
const **Array3** &	**translation** () const
void	**setTranslation** (const **Array3** &value)
const **Orientation** &	**orientation** () const
void	**setOrientation** (const **Orientation** &value)
void	**outboardTransformBy** (const **Array3** &translation, const **Orientation** &orientation)
void	**outboardTransformBy** (const **Array3** &translation)
CoordinateSystemTransformation &	**operator*=** (const **CoordinateSystemTransformation** &xform2)

CoordinateSystemTransformation	**operator*** (const CoordinateSystemTransformation &xform2) const
Array3	**operator*** (const Array3 &vec) const
void	**transform** (Array3 &vec) const
void	**transform** (const Array3 &from, Array3 &to) const
void	**transform** (const Array3 &firstFrom, Array3 &firstTo, const Array3 &secondFrom, Array3 &secondTo) const
EcBoolean	**approxEq** (const CoordinateSystemTransformation &xform2, EcReal tol) const
CoordinateSystemTransformation	**inverse** () const
CoordinateSystemTransformation &	**invert** ()
EcBoolean	**interpolation** (const CoordinateSystemTransformation &coordSysxForm1, const CoordinateSystemTransformation &coordSysxForm2, const EcReal &factor)

actinSE::Orientation Class Reference

Public Member Functions

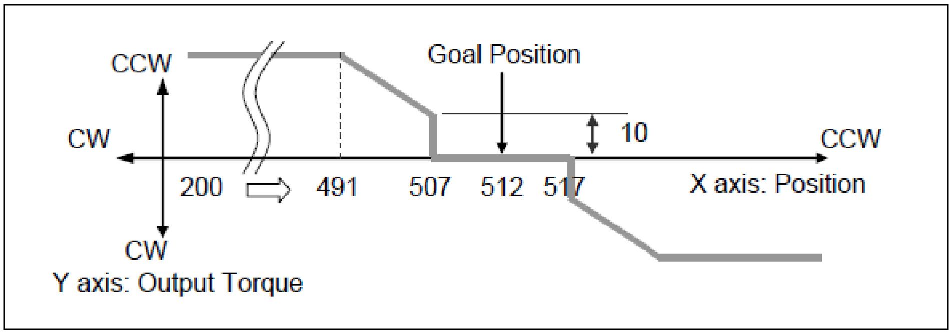
	Orientation () constructor
	Orientation (const EcReal w, const EcReal x, const EcReal y, const EcReal z) constructor from four reals: w, x, y, and z
	~Orientation () destructor
	Orientation (const Orientation &orig) copy constructor
Orientation &	**operator=** (const Orientation &orig) assignment operator
EcBoolean	**operator==** (const Orientation &orig) const equality operator
Orientation &	**operator*=** (const Orientation &orient2)
Orientation	**operator*** (const Orientation &orient2) const
Array3	**operator*** (const Array3 &vec) const
void	**transform** (Array3 &vec) const
void	**transform** (const Array3 &from, Array3 &to) const
void	**transform** (const Array3 &firstFrom, Array3 &firstTo, const Array3 &secondFrom, Array3 &secondTo) const
EcBoolean	**approxEq** (const Orientation &orient2, const EcReal tol=EcTOLERANCE) const

EcBoolean	**angleAxisBetween** (const Orientation &q2, EcReal &theta, Array3 &axis) const
void	**interpolation** (const Orientation &orient1, const Orientation &orient2, const EcReal factor)
Orientation	**inverse** () const
Orientation &	**invert** ()
void	**set** (const EcReal w, const EcReal x, const EcReal y, const EcReal z)
void	**setFrom321Euler** (const EcReal psi, const EcReal theta, const EcReal phi)
void	**get321Euler** (EcReal &psi, EcReal &theta, EcReal &phi) const
void	**setFrom123Euler** (const EcReal phi, const EcReal theta, const EcReal psi)
void	**get123Euler** (EcReal &phi, EcReal &theta, EcReal &psi) const
void	**setFromAngleAxis** (const EcReal angle, const Array3 &axis)
void	**getAngleAxis** (EcReal &angle, Array3 &axis) const
void	**setFromRodriguesVector** (const Array3 &vector)
void	**getRodriguesVector** (Array3 &vector)
const EcReal &	**operator[]** (const EcU32 index) const
void	**getDcmRows** (Array3 &row0, Array3 &row1, Array3 &row2) const
void	**setFromDcmRows** (const Array3 &row0, const Array3 &row1, const Array3 &row2)
void	**getDcmColumns** (Array3 &col0, Array3 &col1, Array3 &col2) const
void	**setFromDcmColumns** (const Array3 &col0, const Array3 &col1, const Array3 &col2)
Array3	**xAxis** () const
Array3	**yAxis** () const
Array3	**zAxis** () const

Cyton Terminology

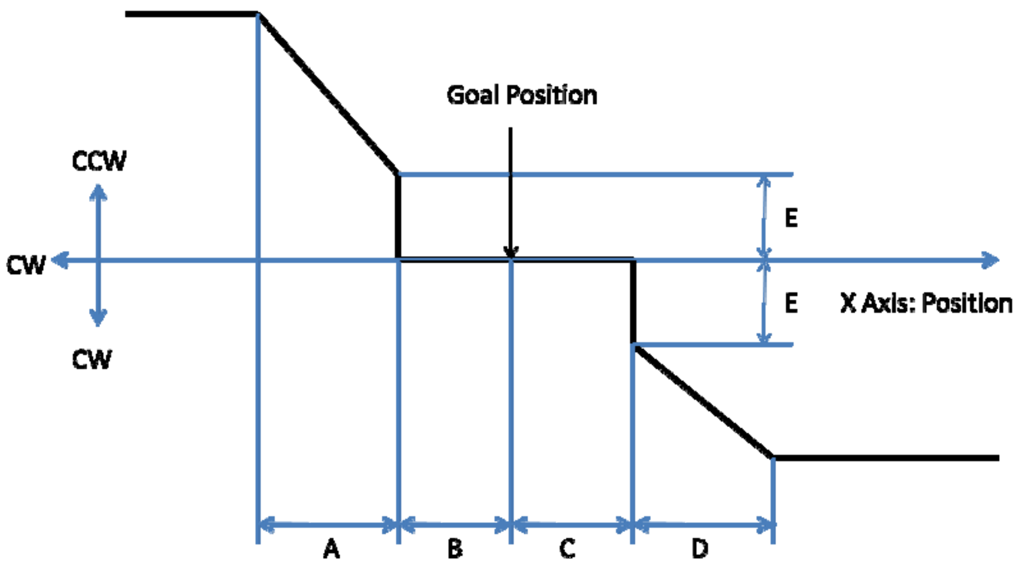
Punch: The limit value of torque being reduced when the output torque is decreased in the Compliance Slope area. It is the minimum torque. The default value is 32 (0x20) and can be extended up to 1023 (0x3FF).

Compliance: Compliance is to set the pattern of output torque. Making Margin & Slope well use of it will result in shock absorption, smooth motion, etc. The length of A, B, C, and D in the below graph (Position vs. Torque curve) is the value of Compliance. Compliance Margin is available from 0 to 254 (0xFE) while Compliance Slope is valid from 1 to 254 (0xFE).



From the current position 200 to 491 (512-16-5=491), movement is made with appropriate torque to reach the set speed; from 491 to 507 (512-5=507), torque is continuously reduced to the Punch value; from 507 through 517 (512+5=517), no
[3] torque is generated. 1

[3] Robotis Dynamixel RX-64 User's Manual.



A: CW Compliance Slope
B: CW Compliance Margin
C: CCW Compliance Margin
D: CCW Compliance Slope
E: Punch

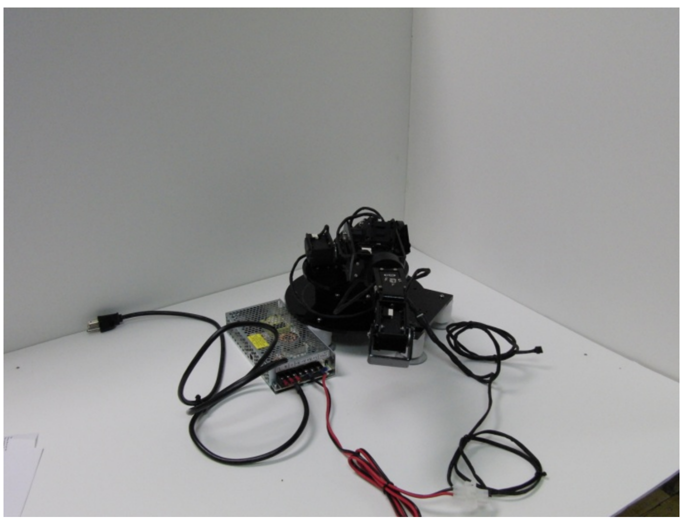
B and C (Compliance Margin) are the areas where output torque is O.

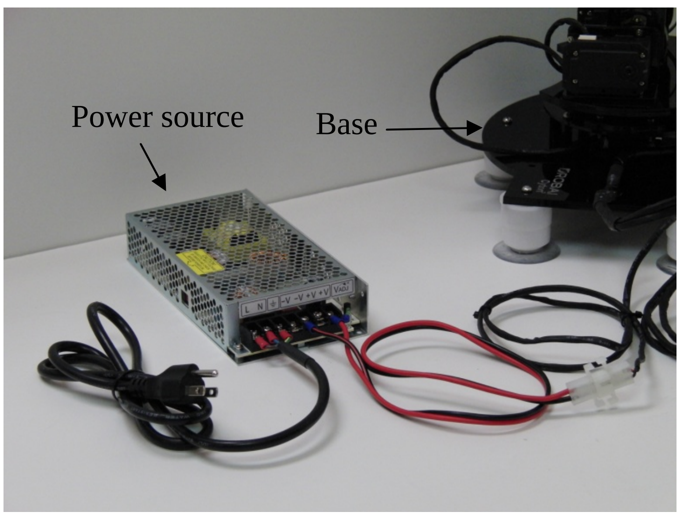
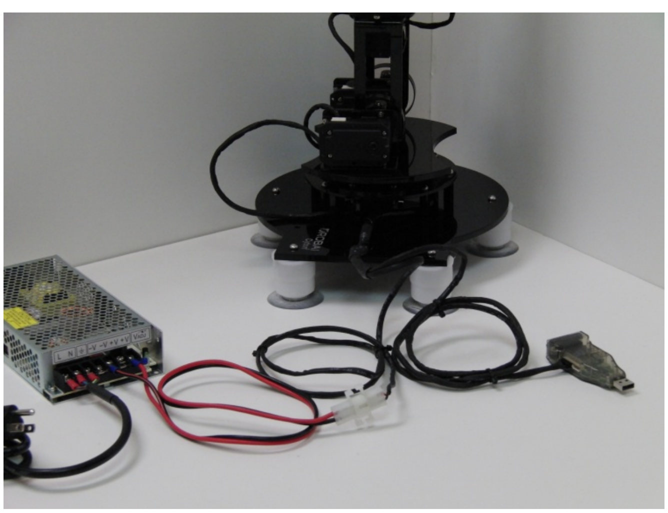
A and D (Compliance Slope) are the areas where output is reduced when they are getting close to Goal Position. The wider these areas are, the smoother the motion is.

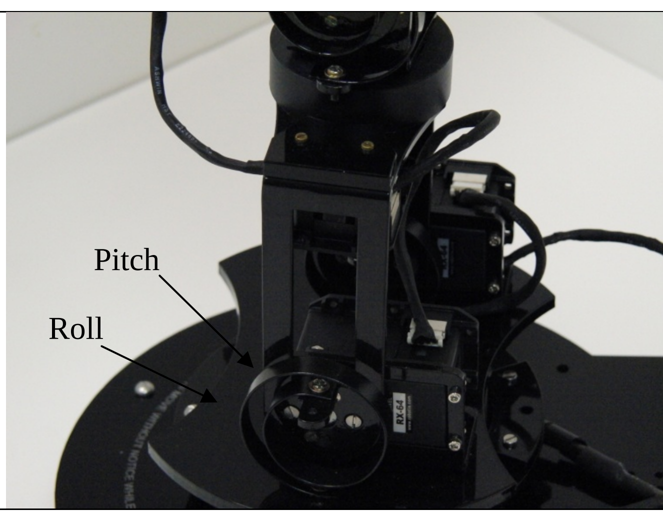
Appendix B - Photo Album

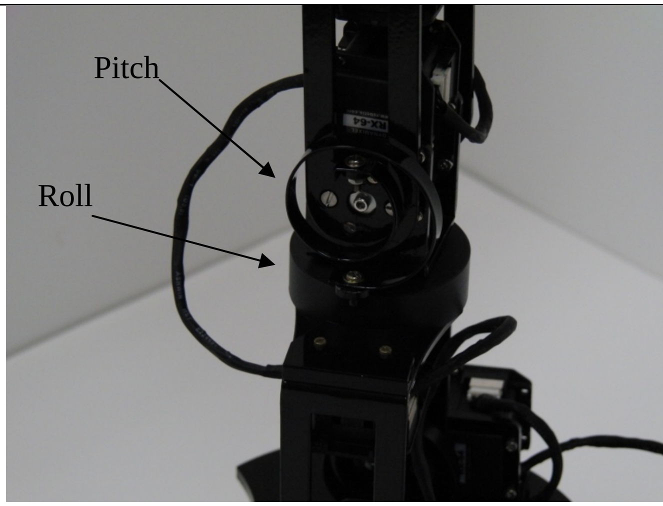
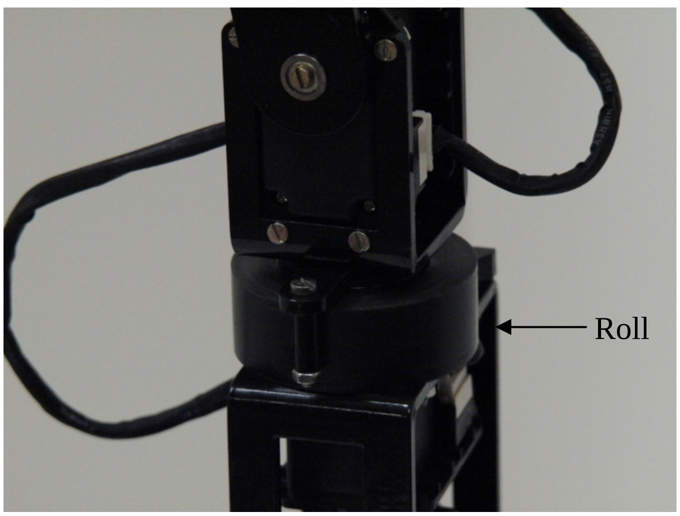
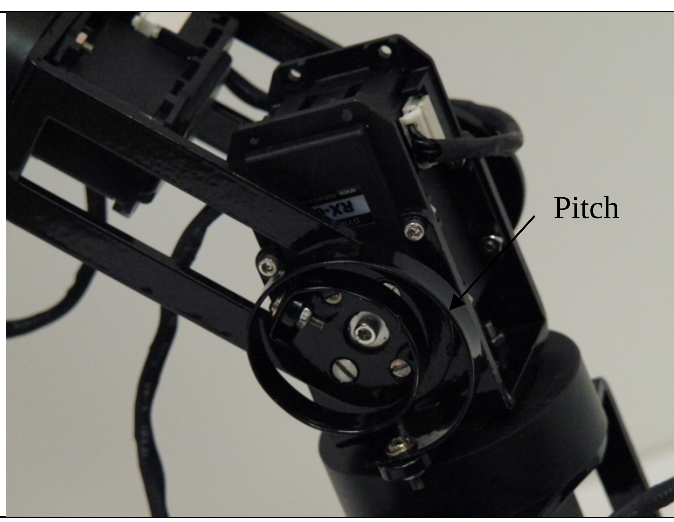
Cyton Photo Gallery

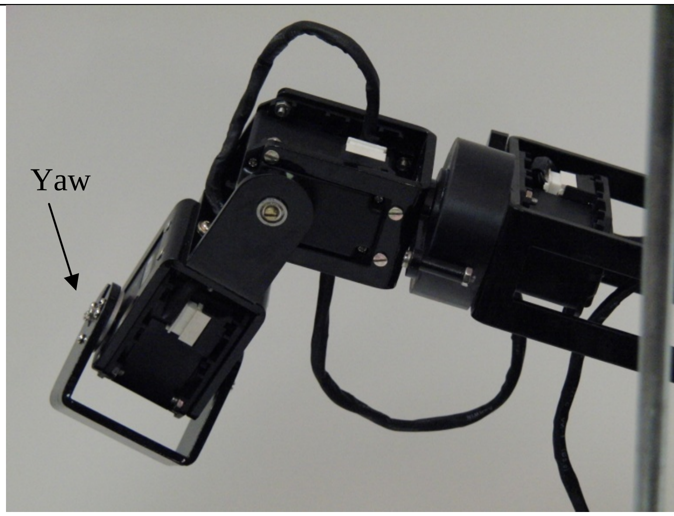
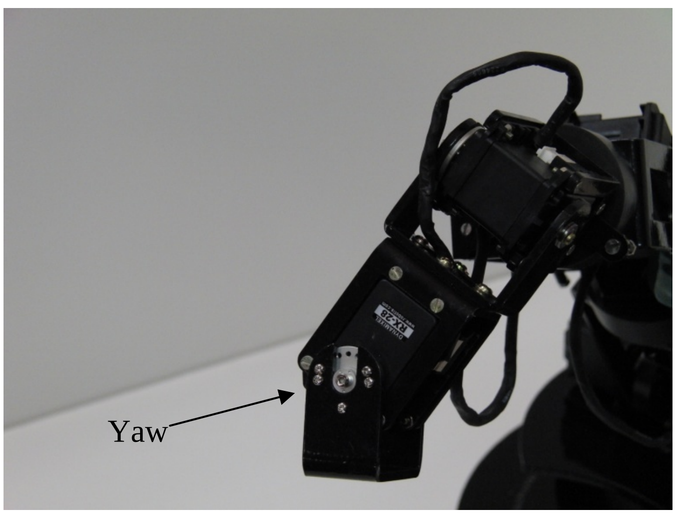
	***1) Box packaging of Cyton (offset view & unpacked)***
	***2) Box packaging and Contents (front view & unpacked)***

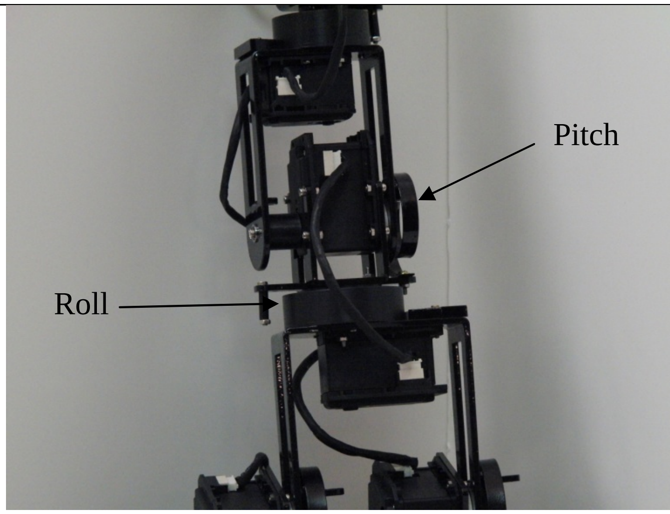
	***3) Box packaging and Contents (top view & packed)***
	***4) Cyton Arm, Wiring, Power Source (Front View & Power Off Position)***
	***5) Cyton Arm, Wiring Power Source & Dongle (Off set view & Zero Position)***

	***6) Cyton Arm, Wiring, Power Source & Dongle (Side view & Zero Position)***
	***7) Cyton Base, Wiring, Power Source (Off set & closeup view)***
	***8) Cyton Base, Wiring, Power Source & USB2 Dynamixel (offset & long shot view)***

	*9) Cyton USB2 Dynamixel (offset & closeup view)*
	*10) Cyton Arm Assembly (Zero Position & long shot view)*
	*11) Cyton Shoulder Roll & Pitch* *(Zero position & closeup view)*
	12) Cyton Elbow

	***(Zero position & closeup view)***
	***13) Cyton Wrist Roll (Zero position & closeup view)***
	***14) Cyton Wrist Pitch (125^0 position & closeup view)***
	15) Cyton Wrist Yaw

	*(35^0 position & closeup view)*
	*16) Cyton Wrist Yaw (125^0 position & offset view)*
	17) Elbow Roll & Pitch Assembly (Side & closeup view)


Pitch
Roll