

CYTON ALPHA 7D 1G

Operations Manual



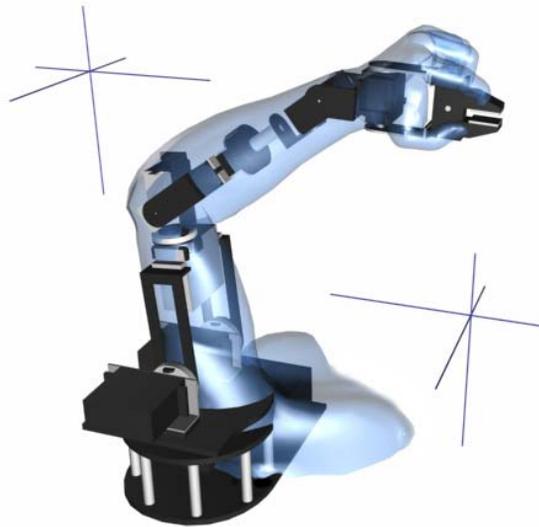
Robai
PO Box 37635 #60466
Philadelphia, PA 19101-0635

Copyright © 2008 Robai. All Rights Reserved.

Copyright © 2008 Robai. All Rights Reserved.	2
Introduction.....	4
Physical Configuration.....	4
Mechanical Structure:	6
Setup Instructions.....	7
Software Installation	7
Hardware Setup.....	7
The Cyton Viewer.....	9
Cyton Viewer Capabilities.....	9
File Options.....	10
Flying the Gripper.....	10
Working with Path Files	12
The Cyton C++ API.....	15
Control Interface	17
Hardware Interface.....	18
Cyton Code Example	20
Tech Support and Contact Info.....	23

Introduction

The Cyton Alpha 7D 1G 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. 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 Alpha 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.

Physical Configuration

The Cyton is designed to mimic the configuration of a human arm. It has a shoulder, an elbow and a wrist. Its hand is a gripper. To accurately simulate the motion of the human

joints the shoulder has two joints the elbow has one joint and the wrist has three. The total of the se axis gives the arm 7 degrees of freedom (DOF). This enables the arm to be kinematically redundant. This allows the arm to reach a given point in space in multiple ways. This allows the arm to reach that same point while reaching around an obstacle or avoiding another manipulator.

Axes Range

Shoulder Base 180 degrees

Shoulder Pitch 170 degrees

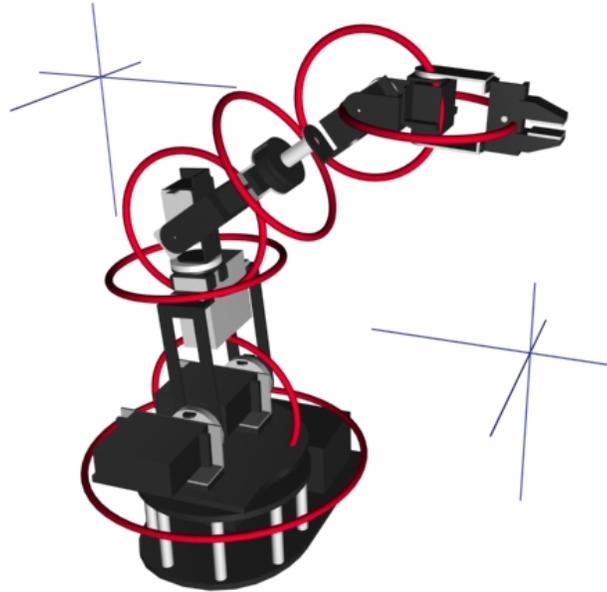
Shoulder Yaw 180 degrees

Elbow Pitch 170 degrees

Wrist Roll 180 degrees

Wrist Yaw 130 degrees

Wrist Pitch 135 degrees



Mechanical Structure:

7 DOF plus gripper; all axes are completely independent. All axes can be controlled simultaneously. Pitching actions have integrated load balancing springs. Key joints have additional planetary ball bearing support.

Electronics: 32 Channel servo controller with 24 free channels for controlling additional peripherals such as other end effectors.

Actuators:

High quality PMDC servo motors with integral gear reduction.

Rated Payload 200 grams

Maximum Payload 300 grams

Rated Speed 0.2 m/s

Joint Speed 60 rpm

Height 60 cm

Reach 48 cm

Repeatability 0.23 cm

Hardware Interface USB



Setup Instructions

Software Installation

Simply insert the CD Rom 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.

Starting the software.

Make sure all connections are firm between the Cyton arm and the PC. Connect the power supply to a power strip. Double click on the Cyton Viewer icon.

Hardware Setup

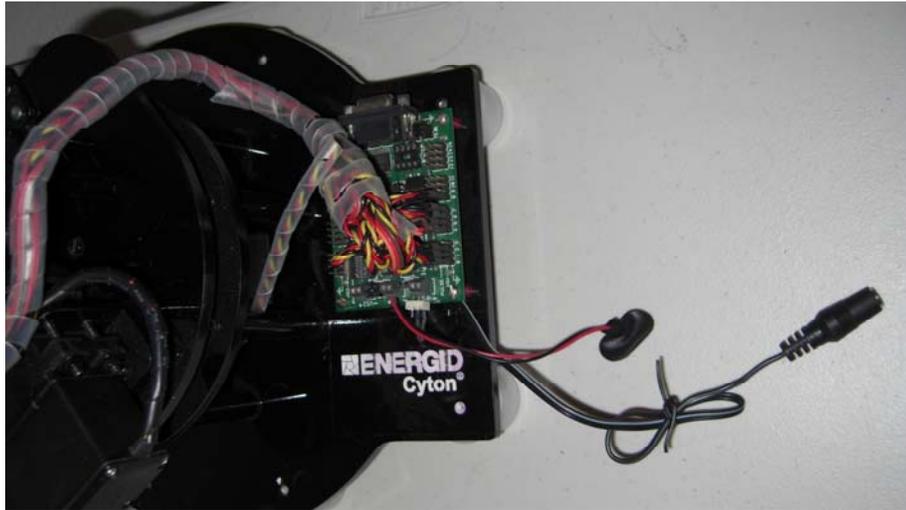
Hardware components



- 6V DC power supply – This powers the servos on the arm
- 9V Battery – Powers Logic Components.
- 3 meter serial cable – 9-pin DB-9 male on one end and 9-pin DB-9 female on other.
- USB to serial adapter – 9-pin DB-9 male connector. (Optional)

Setup Instructions

Place the Cyton arm on firm level surface making sure that the arm is at least 50 cm from any obstructions. Connect the 9V battery (included) to the battery connector. . A mount at the bottom of the Cyton holds the battery in place. Connect the power supply to the power connector on the base of the Cyton arm.

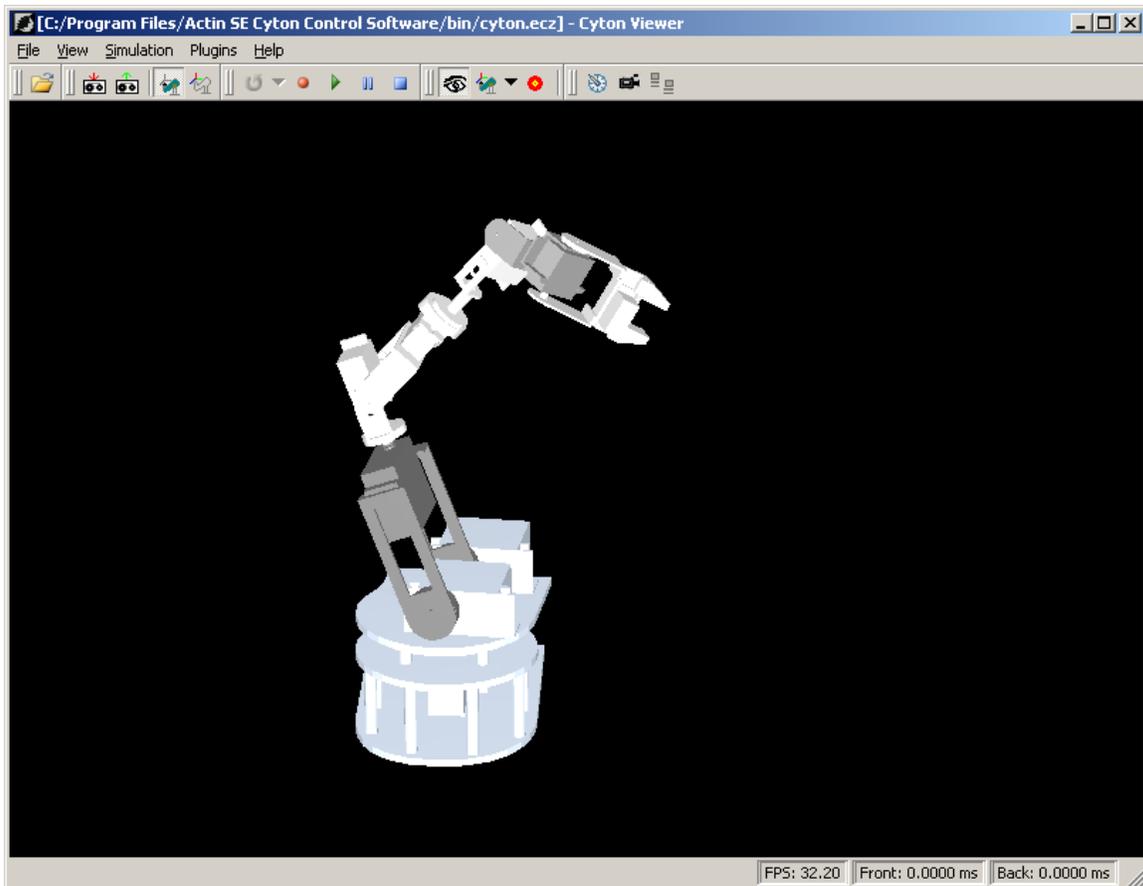


Next connect the serial cable to the port on the Cyton controller card. Then either connect the serial cable to the serial port on your pc or connect the cable to the serial to USB adapter. Then connect the USB adapter to an available USB port on your PC. Before powering up the arm and starting the Actin control software make sure the arm is not blocked by other objects. The software will start by initializing a calibration pose for the arm. This will command the arm to extend fully straight up. If anything obstructs the path of the arm it could be damaged.

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.



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. More files will be made available by Robai over time. 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.

Flying the Gripper

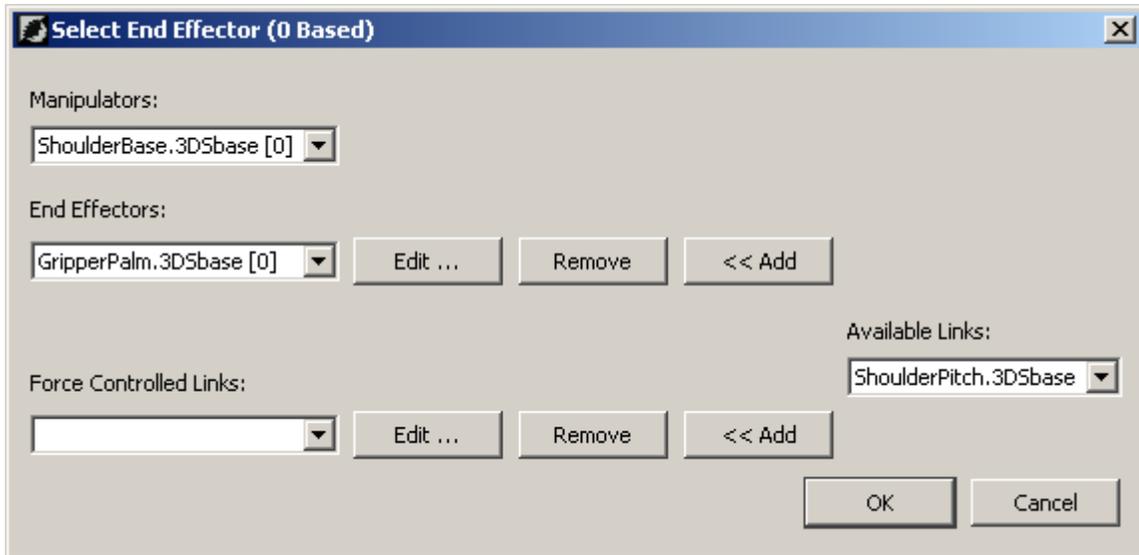
Using the Cyton Viewer, it is possible to command the gripper (or end-effector) to go to an arbitrary position and orientation in space as long as it's within the arms workspace. To do this it's important to calibrate the viewer view with the position of the actual robot.

The Guide Frame

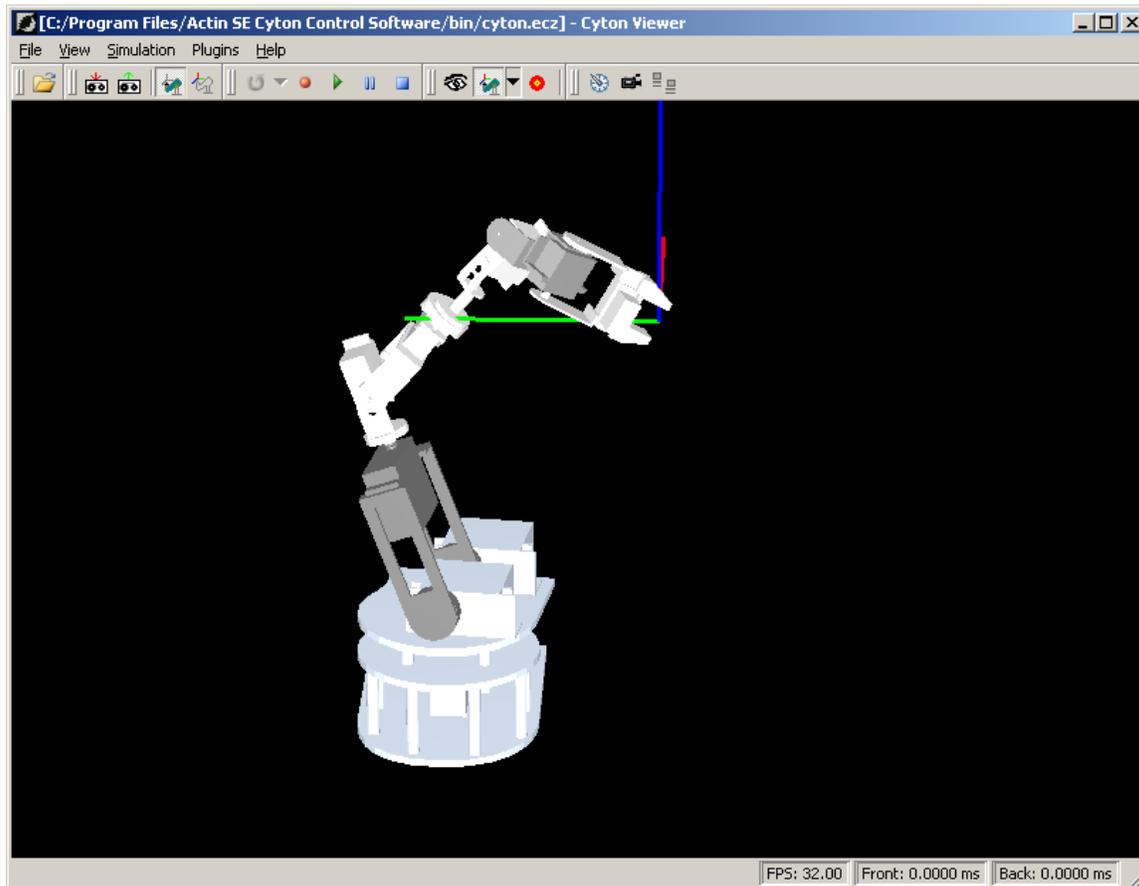
The Guide Frame is what the Cyton Viewer uses to specify the desired gripper position. The figure below shows a Guide Frame (with red, green, and blue axes) just in front of the gripper. You can move the Guide Frame by first selecting the set guide frame button



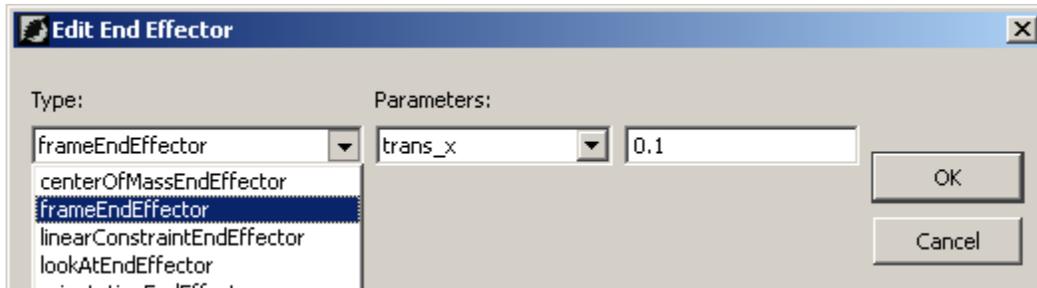
. This will bring up the following dialog box.



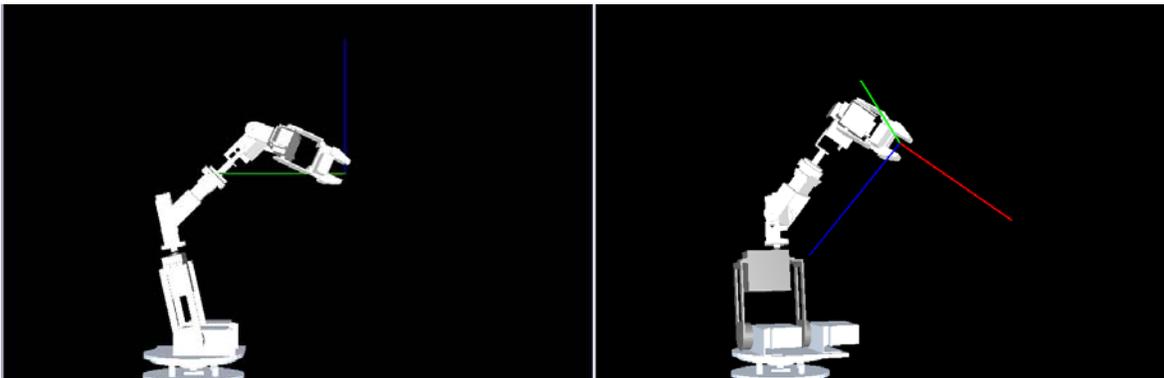
An end-effector can be placed on any link of the robot. By default the end-effector is placed at the gripper. Once okay is selected you should be able to move the Guide Frame within the viewer dragging with either the left or right mouse buttons depressed. Holding down the right mouse button will allow 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 very useful for certain applications.

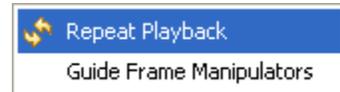
 **Recording a Path** – Pressing the record button puts the viewer into record mode. When in this mode robot positions will be stored in memory until the stop button is

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 angles 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.

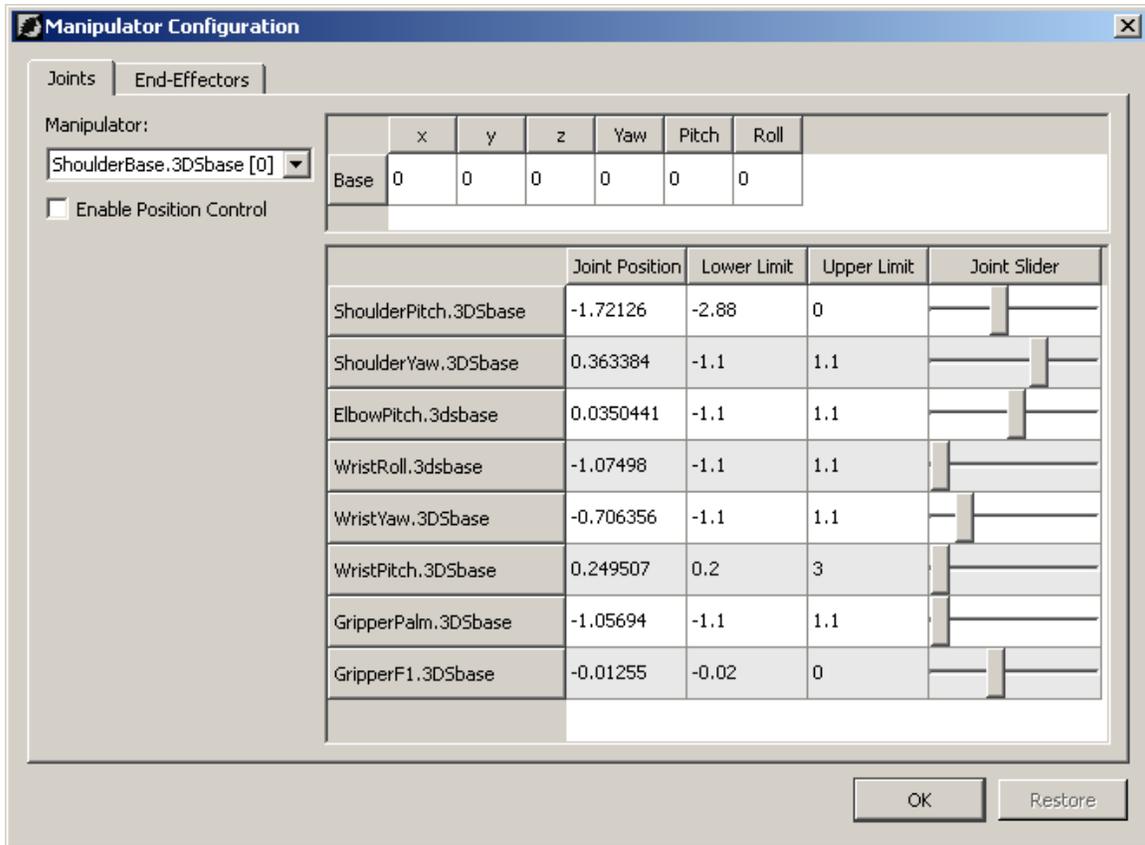
 **Load Path File** – Allows you to load a previously saved path file. Once loaded, the record mode buttons should automatically change to indicate whether the path is in Guide Frame mode or Manipulator mode.

 **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.

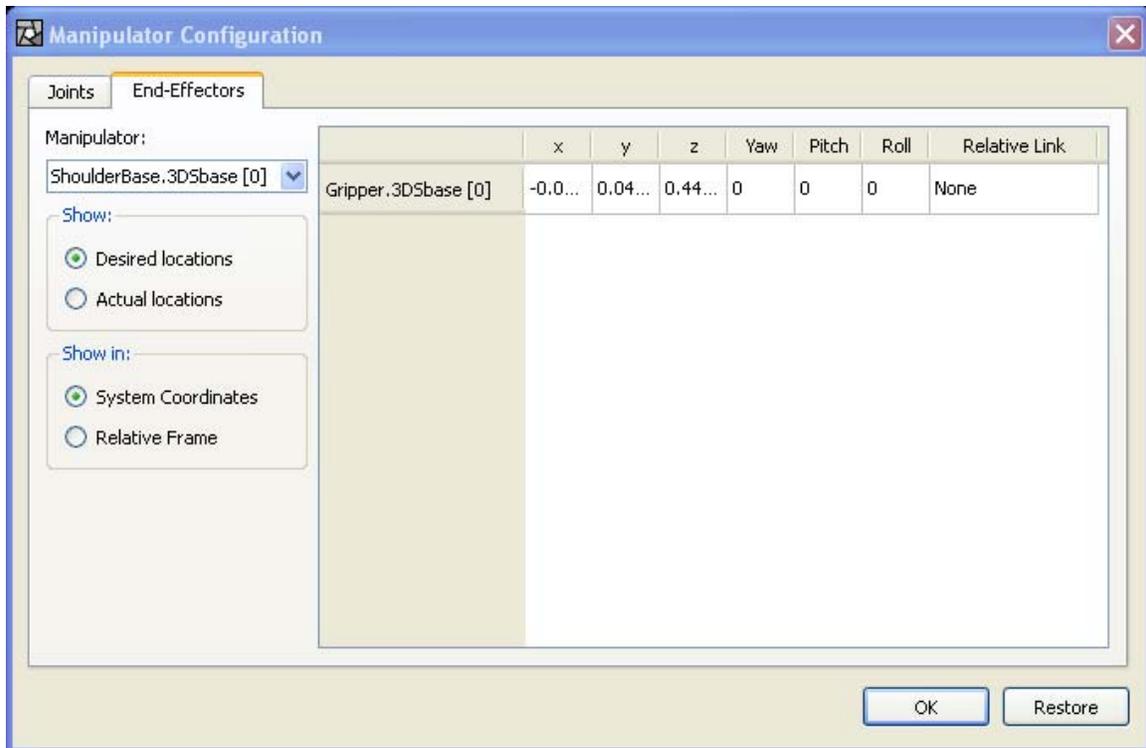


The Manipulator Configuration Tool

 **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. The figure below shows the manipulator configuration tool with the Joints tab. 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 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.



The Cyton C++ API

CYTON Alpha 7D-1G Control System API Rev. 1.0.0.2

*All units in SI unless otherwise specified.

In this section we present both the Cyton control interface and the Cyton hardware interface. The control interface uses Actin-SE to compute joint angles for the robot. The hardware interface allows for direct control of the joints.

The Cyton Config file (*cyton_config.txt*)

In the folder holding the test applications and the Cyton Viewer you will see a file called *cyton_config.txt*. This file is used for specifying 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:

```
COMX -- The serial. port that Cyton is connected to
1 – reset On Shutdown (if true (=1) reset the robot before shutting down
8 – Degrees of freedom (including the gripper) This should stay constant
----next 8 lines
pulseOffset MinPulse MaxPulse MaxPulseRate ScaleFactor
```

These are the values used for each of the servos starting with the base servo.

Control Interface

Method	Description
<pre>void setConfigurationFilename (const EcString& filename);</pre>	set the cyton configuration filename
<pre>const EcString &configurationFilename () const;</pre>	Return the currently configured filename
<pre>bool setDesiredEndEffectorPose (const EcCoordinateSystemTransformation& desiredPose, const EcGeneralMotion& desiredFrameVelocity, const unsigned int endEffectorIndex=0)</pre>	Set desired end-effector pose and rates.
<pre>bool getPropagationResults (EcPropagationResultData& propagationResults) const</pre>	Get the propagation results from the last advance.
<pre>bool computeEndEffectorPose (const EcRealVector& angles, EcCoordinateSystemTransformation& pose) const</pre>	Compute forward kinematics and get the end-effector pose, given a set of joint angles.
<pre>virtual bool initializeAll (const EcRealVector& angles, const Ec::coordinateSystemTransformation& basePose);</pre>	initialize joint angles and base pose /// The quaternion uses the I={0,0,0,1} formalism.
<pre>static void sleep (const double& timeInSeconds)</pre>	A utility function to sleep for a specified number of seconds
<pre>EcBoolean calculateNewJointValuesAndRates (const EcReal timeInSeconds, EcRealVector& jointValues, EcRealVector& jointRates);</pre>	calculate new joint values

Hardware Interface

API Rev. 1.0.0.2

<pre>hardwareInterface (const EcString &configFile = "cyton_config.txt");</pre>	<p>Constructor. Optional argument allows you to specify a different configuration file.</p>
<pre>void setSerialPort (const EcString &port);</pre>	<p>Specify a serial port to use for the connection to the hardware. port String name of serial port to use. Platform dependent.</p>
<pre>EcStringVector availableSerialPorts () const;</pre>	<p>Examine current hardware configuration to list available serial ports. return EcStringVector A vector of strings representing the port names of the serial devices available. Platform dependent. Empty list returned if not available, or plugin not loaded.</p>
<pre>void setResetOnShutdown (const EcBoolean resetOnShutdown);</pre>	<p>Flag indicating whether or not to reset Cyton joints to their initialization position before powering down. resetOnShutdown Whether or not to reset on power down.</p>
<pre>EcBoolean init ();</pre>	<p>Initialize hardware, which includes reading in configuration file, opening serial port and resetting hardware to a known good state. Return EcBoolean Success or failure of initialization.</p>
<pre>EcBoolean reset ();</pre>	<p>Send a reset command to the hardware to move joints back to resting position. Return EcBoolean Success or failure.</p>
<pre>EcBoolean shutdown ();</pre>	<p>Unloads plugin device driver. Return EcBoolean Success or failure of shutdown command.</p>
<pre>EcBoolean setJointCommands (const EcReal timeNow, const EcRealVector &jointCommands, const StateType stateType = JointAngleInRadians);</pre>	<p>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. timeNow Current time. jointCommands Vector of joint angles to move servos to. stateType Optional unit conversion for input jointCommands. Return EcBoolean Success or failure of set command.</p>
<pre>EcBoolean getJointStates (EcRealVector &jointStates, const StateType stateType =</pre>	<p>Retrieve servo information. Depending on the stateType parameter it will return the last commanded position (default) or any of the configuration parameters for the</p>

<pre>JointAngleInRadians) const;</pre>	<p>servos (joint bias, min angle, max angle, reset angle, max joint rate, joint scale).</p> <p>jointState Vector of returned values.</p> <p>stateType Type and unit of requested values.</p> <p>Return EcBoolean Success or failure of query command.</p>
<pre>EcBoolean waitUntilCommandFinished (const EcU32 timeoutInMS) const;</pre>	<p>Wait for the last command to finish, up to a specified maximum time in milliseconds.</p> <p>timeoutInMS Maximum time to wait in milliseconds before failing.</p> <p>Return EcBoolean Success or failure of wait command.</p>
<pre>EcU32 numJoints () const;</pre>	<p>Retrieve the number of joints currently configured.</p> <p>Return EcU32 Number of joints in Cyton arm. 8 for this version of the hardware.</p>

Cyton Code Example

Below is sample code from the hardware interface. Note that this code is provided with your Cyton install and should be buildable and runnable.

```
////////////////////////////////////
//
// Function:      testNormalized
// Description:  Perform a series of tests on the hardware using
normalized
//              joint angles (values in the range [-1,1]). It is
//              currently setup to perform the following tasks:
//              1. Set all joints to -1 (minimum joint angle).
//              2. Set all joints to 1 (maximum joint angle).
//              3. Set all joints to 0 (initialization joint angle).
//
// I/O:          Returns EcTrue on success or EcFalse on failure.
//
// Notes:       The test is currently setup to take 4 seconds to
complete
//              each task, with a total wait time of 10 seconds to
wait
//              before proceeding to the next task.
// Revisions:
////////////////////////////////////
//
EcBoolean
hardwareExample::testNormalized
(
)
{

    EcRealVector minAngleNormalized(m_NumJoints);
    EcRealVector maxAngleNormalized(m_NumJoints);
    EcRealVector centerAngleNormalized(m_NumJoints);

    for(EcU32 ii=0; ii<m_NumJoints; ++ii)
    {
        minAngleNormalized[ii]    = -1.0; // Go to minimum joint angle.
        maxAngleNormalized[ii]    =  1.0; // Go to max joint angle.
        centerAngleNormalized[ii] =  0.0; // Reset to zero angle.
    }

    // Set all joints to their absolute minimum value.
    std::cout << "Setting all joint to min normalized angles.\n";
    if(!m_hw.setJointCommands(4.0, minAngleNormalized,
cyton::AngleNormalized))
    {
        std::cerr << "Problem setting min normalized angles.\n";
        return EcFalse;
    }
    m_hw.waitUntilCommandFinished(10000); // Let the hardware achieve
its position. 10 sec
}
```

```

    // Set all joints to their absolute maximum value.
    std::cout << "Setting all joint to max normalized angles.\n";
    if(!m_hw.setJointCommands(8.0, maxAngleNormalized,
cyton::AngleNormalized))
    {
        std::cerr << "Problem setting max normalized angles.\n";
        return EcFalse;
    }
    m_hw.waitUntilCommandFinished(10000); // Let the hardware achieve
its position. 10 sec

    // Set all joints to their 'center' position.
    std::cout << "Setting all joint to 'center' normalized angles.\n";
    if(!m_hw.setJointCommands(12.0, centerAngleNormalized,
cyton::AngleNormalized))
    {
        std::cerr << "Problem setting center normalized angles.\n";
        return EcFalse;
    }
    m_hw.waitUntilCommandFinished(10000); // Let the hardware achieve
its position. 10 sec

    return EcTrue;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
// Function:      testRadians
// Description:   Perform a series of tests on the hardware. It is
//               currently setup to perform the following tasks:
//               1. Set all joints to their minimum joint angle.
//               2. Set all joints to their maximum joint angle.
//               3. Set all joints to their initialization joint
//               angle.
//               //
// I/O:          Returns EcTrue on success or EcFalse on failure.
//               //
// Notes:       In each case the parameters are queried from the
//               configuration file. The test is currently setup to
//               take 4 seconds to complete each task, with a total
//               wait
//               time of 10 seconds to wait before proceeding to the
//               next task.
// Revisions:
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
EcBoolean
hardwareExample::testRadians
(
)
{
    EcRealVector jointAngle(m_NumJoints);

    // Pull information from configuration
    std::cout << "Reading min angles from configuration.\n";

```

```

    if(!m_hw.getJointStates(jointAngle, cyton::MinAngleInRadians))
    {
        std::cerr << "Unable to get minimum angles from
getJointStates.\n";
        return EcFalse;
    }
    // Set all joints to their configured minimum value.
    std::cout << "Setting all joint to min angles.\n";
    if(!m_hw.setJointCommands(0.0, jointAngle))
    {
        std::cerr << "Problem setting min angles.\n";
        return EcFalse;
    }
    m_hw.waitUntilCommandFinished(10000); // Let the hardware achieve
its position. 10 sec

    // Pull information from configuration
    std::cout << "Reading max angles from configuration.\n";
    if(!m_hw.getJointStates(jointAngle, cyton::MaxAngleInRadians))
    {
        std::cerr << "Unable to get maximum angles from
getJointStates.\n";
        return EcFalse;
    }
    // Set all joints to their configured maximum value.
    std::cout << "Setting all joint to max angles.\n";
    if(!m_hw.setJointCommands(4.0, jointAngle))
    {
        std::cerr << "Problem setting max angles.\n";
        return EcFalse;
    }
    m_hw.waitUntilCommandFinished(10000); // Let the hardware achieve
its position. 10 sec

    // Pull information from configuration
    std::cout << "Reading init angles from configuration.\n";
    if(!m_hw.getJointStates(jointAngle, cyton::InitAngleInRadians))
    {
        std::cerr << "Unable to get init angles from getJointStates.\n";
        return EcFalse;
    }
    // Set all joints to their init value.
    std::cout << "Setting all joint to init angles.\n";
    if(!m_hw.setJointCommands(8.0, jointAngle))
    {
        std::cerr << "Problem setting initialize angles.\n";
        return EcFalse;
    }
    m_hw.waitUntilCommandFinished(10000); // Let the hardware achieve
its position. 10 sec

    return EcTrue;
}

```

The following code snippet shows how to use Actin-SE in conjunction with the hardware interface to control the arm.

```

////////////////////////////////////
//
// Function:      testControlAndHardware
// Description:  Consolidated example that positions the end-effector
and
//              then pushes the calculated joint values to the
hardware.
//              It sets an initial location and tells it to move to
the
//              new location. There are 100 timesteps generated for a
//              2 second interval. The simulation will run based on
//              internal constraints and rates to achieve the desired
//              pose before the 2 seconds.
// I/O:          Returns EcTrue on success and EcFalse on failure.
// Notes:       none
// Revisions:
////////////////////////////////////
//
EcBoolean
controlExample::testControlAndHardware
(
)
{
    EcRealVector jointAngles;
    EcRealVector jointRates;

    // -----
    // test end-effector pose/motion calculation
    // -----

    EcReal time=0.0;

    Ec::coordinateSystemTransformation initialPose;

    initialPose.setTranslation(Ec::Vector(-0.0122878,-
0.136491,0.467294));

    // Desired pose is located 10cm away along X and Y.
    Ec::coordinateSystemTransformation desiredPose = initialPose;
    desiredPose.setTranslationX(desiredPose.translation().x()+0.1);
    desiredPose.setTranslationY(desiredPose.translation().y()+0.1);

    //cyton::hardwareInterface
hardwareInterface("cyton_low_rate_test_config.txt");

    // execution parameters
    EcU32 steps = 200;
    EcReal simRunTime = 2.0;
    EcReal simTimeStep = simRunTime/steps;

    // Set the desired final position.
    EcBoolean passed =
m_ControlInterface.setDesiredEndEffectorPose(desiredPose,2);

```

```

    // move to the desired pose.  If running the rendered version, it
will display the
    // progress.
    for(EcU32 ii=0; ii<steps && passed; ++ii)
    {

        // get the current time
        EcReal currentTime = simTimeStep*ii;

        /// calculate new joint values
        passed &=
m_ControlInterface.calculateNewJointValuesAndRates(currentTime,
jointAngles, jointRates);
        std::cout << "Step: " << ii << " Joint Angles: " << jointAngles
<< "\n";

        // Pass joint values to the hardware.
        // passed &= hardwareInterface.setJointCommands(currentTime,
jointAngles, cyton::JointAngleInRadiansBiasScale);
    }
    if(passed)
    {
        std::cout << "Control with Hardware test passed.\n";
    }
    return passed;
}

```

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

