MAESTOR: MAchine and Environment Software Translation Over Ros

Ryan Young, Eric Rock, John Maloney

Contents

1 Introduction **3**

2 Software Requirements **3**

3 Software Overview **4**

4 Installation **5**
- 4.1 Install All packages . . . 5
- 4.2 Install Packages Independently . . . 6
- 4.3 Testing Installation . . . 6

5 Uninstallation **6**

6 How to use **7**
- 6.1 Run . . . 7
- 6.2 The Console . . . 8
- 6.3 Python Module . . . 8
  - 6.3.1 Module Functionality . . . 8
  - 6.3.2 Creating Your Scripts . . . 9
  - 6.3.3 Running Your Scripts . . . 10
- 6.4 Functionality . . . 10
  - 6.4.1 Commands . . . 12
  - 6.4.2 Properties . . . 12
- 6.5 Initialization . . . 14
- 6.6 Meta Joints . . . 15
- 6.7 Moving . . . 16
  - 6.7.1 Joint Motion types . . . 16
- 6.8 Feedback . . . 17
- 6.9 Running Trajectories . . . 17
- 6.10 Stopping . . . 18

7 Quick Start **19**

8 External Sensors and Demos **20**

9 Extras **20**
- 9.1 Default Joint Name scheme . . . 20
- 9.2 Response Characteristics . . . 22
- 9.3 Support . . . 22

1 Introduction

This repository contains the MAchine and Environment Software Translation Over Ros (MAESTOR) software project. Included are install scripts to generate the execution environment required for MAESTOR, as well as an install script to generate the execution environment for virtualization and visualization of the HUBO robot.

MAESTOR was designed to be an easy to use rapid prototyping tool for developing movements on the humanoid robot HUBO. MAESTOR allows you to visualize code in simulation and execute the same code on real hardware. One of the ways that you can create movements on the robot is by making simple to write python scripts that can be powerful enough to create complex movements even walking. MAESTOR is a great development tool for integrating external sensors with HUBO thanks to it using ROS as a communication bus. This user manual will help walk you through using MAESTOR so you can begin to get started on your own scripts and projects.

All examples of commands to run are centred and in italics for example:

ls -a

2 Software Requirements

For MAESTOR to run properly you must have all of the following:

- Ubuntu 12.04 LTS
- ROS Fuerte
- OpenRAVE (ROS stack)
- hubo-ach
- OpenHubo

Ubuntu 12.04 LTS
Ubuntu 12.04 is the operating system that we built MAESTOR for.

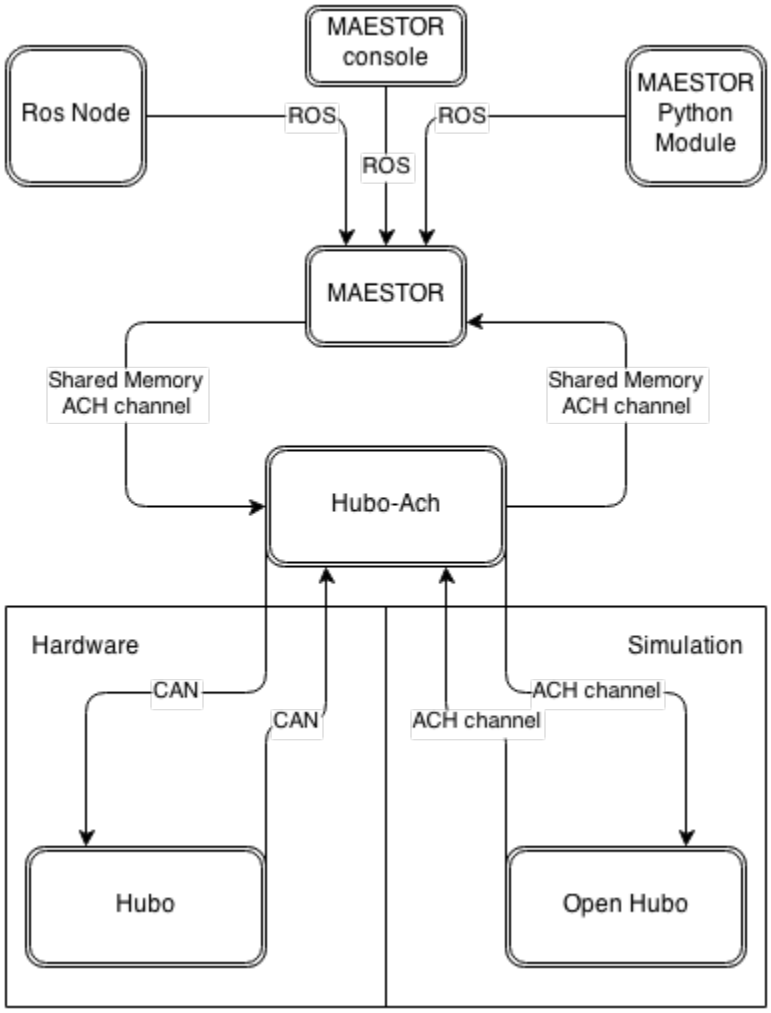
ROS Fuerte
ROS Fuerte is the Robot Operating System that allows us to communicate using TCP packets to different nodes. MAESTOR is a ROS package and also uses ROS for all of its service communication.

OpenRAVE (ROS stack)
OpenRAVE is used for visualizing the robot in simulation and also providing the physics in simulation.

hubo-ach
: hubo-ach is how we communicate to the can boards to pass can messages to the motor boards.

OpenHubo
: OpenHubo is an OpenRAVE extension that simulates HUBO specifically.

3 Software Overview



MAESTOR is a ROS node that allows advertises ROS services that allows it to interact with other ROS nodes. MAESTOR manipulates the robot of simulation by using the program Hubo-Ach. Hubo-Ach and MAESTOR communicate through shared memory files called ACH channels. Hubo-Ach can operate in either Hardware mode, on a real robot, or in simulation mode using the OpenRAVE extension OpenHubo to simulate Hubo. When operating on hardware Hubo-Ach communicates using the CAN bus on the robot and uses CAN to talk to the motor boards. In simulation Hubo-Ach uses Ach channels to communicate with the simulation. MAESTOR provides access to all of this through ROS and allows more

functional joint control with included interpolation functions, inverse kinematics, balancing and more.

4 Installation

Installation scripts exist to install dependencies such as ROS, OpenRAVE, Hubo-Ach and OpenHUBO.

4.1 Install All packages

Pay strict attention to permissions for all scripts. Some require root access and some mandate non-root access.

To install all dependencies and complementary packages, change directory to the install directory and run the install all script:

./install-all.sh

4.2 Install Packages Independently

The install all script just calls each of these scripts individually. If there are errors when using the install all script it is recommended to install each package using these install scripts in this order or you can install only the packages you need.

Installation of ROS:

sudo ./install-ros-fuerte.sh

Installation of OpenRAVE:

sudo ./install-openrave.sh

Installation of Hubo-Ach:

sudo ./install-hubo-ach.sh

Installation of OpenHUBO:

./install-openHubo.sh

Installation of the MAESTOR ROS pacakge:

./install-maestor.sh

4.3 Testing Installation

After installation, it is recommended for the user to interact with these packages in a new terminal in order to ensure that environment variables are properly loaded. If any errors occur, please reboot the operating system and try again,to confirm that the errors are not related to unloaded environment variables.

To test to see if everything is working run the command:

maestor sim-real

This will open up the full MAESTOR environment including ROS, OpenHUBO, and hubo-ach. If everything opens up without any errors then your software installed correctly.

5 Uninstallation

If you want to remove all traces of MAESTOR and all of it's dependencies from your computer you can do that with the uninstallation script.

Navigate to Maestor-Install-Directory/install/uninstall
Run:

source uninstall.sh

You will be prompted to uninstall each of the separate components installed by the install scripts. NOTE: Using source to call the script is necessary because otherwise it will unable to read the OPENHUBO_DIR environment variable.

6 How to use

6.1 Run

To run MAESTOR, type the command maestor and an argument:
Usage:

maestor <Command>

<Command>:

sim
-Starts MAESTOR with hubo-ach in simulation mode with physics

sim-real
-Start MAESTOR with hubo-ach in simulation mode without physics (Real time)

real
-Starts MAESTOR with hubo-ach in hardware mode

console
-Starts the MAESTOR Python console

kill
-Kills MAESTOR, hubo-ach, and roscore

script <scriptName>
-Runs the python script that is located in the maestor/scripts/ directory with then name <scriptName>. The script must follow regular python conventions and end with .py as well as be in the correct directory.

To actually run MAESTOR you need to choose either the sim, sim-real, or real argument.

The differences between each mode of operation are as follows:

sim
-Hubo-Ach runs in simulation mode, and opens the openHubo simulation. Physics is enabled and the positions of motors can be retrieved. Simulation mode with physics enables will run slightly slower than real time. Two terminals will open, one that is running OpenRave and another that is running roscore. The terminal running OpenRave will require elevated privileges. The terminal that you ran MAESTOR from will also require elevated privileges after you give them to OpenRave.

sim-real
-Hubo-Ach runs in simulation mode without physics. This option is meant for previewing intended robot motion, and should be much faster than simulation with physics. You can not retrieve the positions of the motors with physics off. Two terminals will open, one that is running OpenRave and another that is running roscore. The terminal running OpenRave will require elevated privileges. The terminal that you ran MAESTOR from will also require elevated privileges after you give them to OpenRave.

real

-Hubo-Ach runs in hardware mode. This option is meant for running MAESTOR on the actual robot. No simulation is opened and a hardware CAN interface is expected. Your terminal will immediately ask for your password then a bunch of text will appear over the password prompt. You can still type your password even though the text is appearing on screen and you need to.

When MAESTOR is finished starting up the original terminal that you ran the MAESTOR command from will be back in your control. There are two suggested options for controlling MAESTOR once it is started. The first is opening up the MAESTOR console and the second is using the MAESTOR Python module. Both ways are wrappers that allow you to make service calls over ROS.

6.2 The Console

To open the MAESTOR console type the command in a terminal:

maestor console

Once the console is open you will see a prompt that looks like a regular Python interpreter prompt:

>>>

In this prompt you can use any of the MAESTOR functions by typing their names and any arguments they have. To close out of the console you can press ctrl-d when at a blank prompt. You can reopen the console at any time as long as MAESTOR is running. You can find all of the functions that are available for use and their parameters in the table under the functionality section titled MAESTOR Functions. The underlying workings for this console is a Python script that makes direct ROS service calls and returns whatever they return. For more precise work and performing complex movements and tasks it is highly recommended to use the Python module.

6.3 Python Module

To interact with MAESTOR using scripts or your own programs in Python I recommend using the MAESTOR Python module. The MAESTOR Python module is contained in a file called Maestor.py that resides in the maestor/scripts/ directory. I strongly suggest saving scripts in this directory for convenience. If you save your files in a different directory you will either need to add Maestor.py to the PythonPATH or include Maestor.py in the directory that you save your script.

6.3.1 Module Functionality

The MAESTOR Python module is a class that you can import into your Python script or project and create an object of. If you are familiar with ROS the MAESTOR module creates a ROS node that is used for calling services on the MAESTOR node. If you aren't familiar

with ROS you can use the MAESTOR module to avoid having to use it. The MAESTOR module allows you to use all of the functions that are listed in the MAESTOR functions table and a one more that was added for convenience:

waitForJoint(<Joint>)
-Blocking call that waits for the joint <Joint> to no longer require motion.

The way that the module works is you create a *maestor* object in your script and call the functions on that object. The object handles all of the ROS stuff and makes the necessary calls.

6.3.2 Creating Your Scripts

To create a script you can run directly, the script must be executable which can be done with the bash command:

chmod +x <scriptName>

This command is not a part of MAESTOR but it gives all users on a system the permission to execute that file. For both running scripts directly and running them with the maestor command you need to have the proper shebang. An example of the Python shebang is:

#!usr/bin/env python

To import the maestor object from the MAESTOR module add this line to your script:

from Maestor import maestor

This will allow you to create the maestor object. You can create the maestor object by writing something like:

robot = maestor()

Where robot is any variable name you want. From there you can make all of the calls by calling the functions on robot as an example:

robot.setProperty("RSP", "position", -1)

NOTE
Make sure that your script knows where the Maestor.py module file is. This can be done by either adding it to the Python path or having a copy in the directory where your script is. If your script does not know where the Maestor.py module is it will return a lot of error messages.

Most scripts will require that you run the start up sequence that is described in Section 7

6.3.3 Running Your Scripts

To run scripts that are in the maestor/scripts/ directory you can run the command:

maestor script <scriptName>

Where <scriptName> is the name of a script in the directory excluding the .py extension. For example the command:

maestor script testure

Will run the script testure.py which is located in the maestor/scripts/ directory. An alternative to running your script through the maestor command is to run it directly.

./<scriptName>

Another alternative way of running this is using the Python interpreter in non-interactive mode with the command:

Python <scriptName>

If you use the last method of running the scripts then you do not need to include the shebang nor does it have to be executable. Though it is usually good practice to do these things anyway for scripts.

6.4 Functionality

MAESTOR provides different functions through ROS services that can be called through ROS nodes, the MAESTOR console, or using the MAESTOR Python Module. Below is a table of all of the functions that MAESTOR offers and a description of them.

MAESTOR Functions	
Function	**Description**
setProperty(<Joint>, <Property>, <Value>)	Set the <Property> of the <Joint> to the <Value>.
loadTrajectory(<Name>, <Path>, <Read>)	Load a trajectory into MAESTOR with the name <Name> from the <Path>. The <Read> parameter is a boolean where if it is true the file that is open is read from and if it's false the file is written to in a trajectory format.
command(<Name>, <Target>)	Runs the command <Name> with the option of <Target> where Target is a joint or nothing. The list of all commands is below.
setProperties(<Joints>, <Properties>, <Values>)	Sets multiple properties on multiple joints to different values. <Joints>, <Properties>, and <Values> each must have the same number of sections that are delimited by white space.
unignoreFrom(<Traj>, <Joint>)	Unignore the <Joint> column from the trajectory named <Traj>
requiresMotion(<Joint>)	Returns True if <Joint> is not at its goal position
getProperties(<Joint>, <Properties>)	Get the value of specified properties on the joint <Joint>. <Properties> can be a string of properties delimited by spaces.
ignoreAllFrom(<Traj>)	Ignore all of the joint columns in the trajectory named <Traj>
initRobot(<Path>)	Initialize the robot with the init file specified by <Path>. The empty string "" denotes the default path. Do not run this function more than once in a session.
startTrajectory(<Traj>)	Starts the trajectory with the name <Traj>
ignoreFrom(<Traj>, <Joint>)	Ignore the <Joint> column from the trajectory named <Traj>
setTrigger(<Traj>, <Frame>, <TargetTraj>)	Sets a trigger at the <Frame> in <Traj> to stop executing <Traj> and begin executing <TargetTraj>
unignoreAllFrom(<Traj>)	Unignore all of the joint columns in the trajectory named <Traj>
extendTrajectory(<Traj>, <Path>)	Appends the trajectory located at <Path> to the end of the loaded trajectory <Traj>
stopTrajectory(<Traj>)	Stops executing the trajectory <Traj>

6.4.1 Commands

A lot of how MAESTOR operates is through the use of commands and setting and getting properties. This section will talk about all of the different commands that can be run using the command function and all of the properties that you can get and set on joints.

MAESTOR commands can be run using the function:

command(<Command>, <Target>)

Where <Command> can be a number of different commands where some have a target and others do not. The Table below goes through all of the commands and if it has a target parameter what it is. Both <Command> and <Target> must be strings so if the target is empty the empty string must be passed, e.g.

command("HomeAll", "")

Commands		
Command	**Target**	**Description**
Enable	<Joint>	Enables the joint named <Joint>
EnableAll		Enables all of the joints on the robot
Disable	<Joint>	Disables the joint named <Joint>
DisableAll		Disables all of the joints on the robot
ResetJoint	<Joint>	Reset the board of the joint named <Joint>
ResetAll		Reset all of the boards on the robot
Home	<Joint>	Home the joint named <Joint>
HomeAll		Home all of the joints on the robot
InitializeSensors		Initialize the sensors on the robot to begin receiving feedback from them
Zero	<Joint>	Return the <Joint>'s position to zero
ZeroAll		Return all of the joint's positions to zero
BalanceOn		Turn active balancing on
BalanceOff		Turn active balancing off

6.4.2 Properties

For the functions getProperties, setProperty, and setProperties MAESTOR either sets or get properties on robot components. For each of the commands the syntax is slightly different.

setProperty(<Joint>, <Property>, <Value>)

-This function takes a string <Joint> which can be any robot component and sets the property specified by a string <Property> to the value <Value> where <Value> is a float. An example is:

setProperty("RSP", "position", -1)

setProperties(<Joints>, <Properties>, <Values>)

-This function takes strings of robots components, properties, and values each set must be delimited by spaces. The string of robot components is passed in as <Joints>, string of properties as <Properties> and the string of values as <Values>. An example is:

setProperty("RSP", "position", -1)

getProperties(<Joint>, <Properties>)

-This function takes in a robot component as a string for <Joint> and a space delimited list of properties for <Properties>. An example is:

getProperties("RSP", "position velocity goal")

Not every component has every type of property but below is a table of all of the properties in MAESTOR and the components that have those properties.

Properties		
Property	**Components**	**Description**
position	Joints, Meta Joints	The current encoder read position of the joint
goal	Joints, Meta Joints	The current goal position of the joint
speed	Joints, Meta Joints	The current set speed for interpolation
inter_step	Joints, Meta Joints	The current interpolation step.*** ***WARNING***
velocity	Joints, Meta Joints	The current moving speed of the joint as read from the encoder
goal_time	Joints, Meta Joints	The time it is estimated to finish interpolating
motion_type	Joints	The type of motion that the joint should use.
temp	Joints	The temperature of the joint
homed	Joints	A boolean that signals if the joint is homed
zeroed	Joints	A boolean that signals if the joint is zeroed
errored	Joints	A boolean that tells if a joint has an error
jamError	Joints	A boolean that tells if a joint has a jam error
PWMSaturatedError	Joints	A boolean that tells if a joint has a saturation error
bigError	Joints	A boolean that tells if a joint has a big error

encoderError	Joints	A boolean that tells if a joint has a encoder error
driveFaultError	Joints	A boolean that tells if a joint has a drive error
posMinError	Joints	A boolean that tells if a joint has a position minimum error
posMaxError	Joints	A boolean that tells if a joint has a position maximum error
velocityError	Joints	A boolean that tells if a joint has a velocity error
accelerationError	Joints	A boolean that tells if a joint has a acceleration error
tempError	Joints	A boolean that tells if a joint has a temperature error
x_acc	IMU	The acceleration in the X direction
y_acc	IMU	The acceleration in the Y direction
z_acc	IMU	The acceleration in the Z direction
x_rot	IMU	The rotation about the X axis
y_rot	IMU	The rotation about the Y axis
m_x	FT	The moment in the X direction
m_y	FT	The moment in the Y direction
f_z	FT	The force in the Z direction
meta_value	Meta Joint	The forward kinematics position of the meta joint. Getting the position of the meta joint returns a more accurate version of this.

*** **WARNING:** Manipulating the interpolation step in any way may cause the robot to skip an interpolation step. The interpolation step is generated using a fourth order polynomial whose coefficients are calculated when the position is set. Manipulating an interpolation step could cause serious harm to the robot because that is the number that is sent to the motor boards. If it is required to manipulate the interpolation use extreme caution in operation.

6.5 Initialization

Before you move the robot joints it is very important that you initialize the robot. This includes initializing the software, homing all of the joints, enabling the joints, and initializing the sensors. The following section will walk you through how to do this.

The first thing to do is to initialize the software this is done with the command:

initRobot(<Path>)

Where path is a string and the empty string, "", refers to the default directory. The default directory is:

/opt/ros/fuerte/stacks/maestor/models/hubo_default.xml

This command loads Robot configuration information from the xml file specified. Before this command, MAESTOR cannot perform any other commands or functions.

The next thing you have to do is home the joints. Homing is when the joints move to their default zero position, their home. This step is not required when running any type of simulation and if it is done in simulation you will not see any movement. When you run HomeAll you can not move any joints for about 10 seconds so be careful about this when homing in simulation. You can home all of the joints by running the command:

command("HomeAll", "")

If you want to home a single joint instead of all of the joints you can use the command:

command("Home", <JointName>)

NOTE It is best to home the joints multiple times until every joint is in the correct home position. Occasionally some joints may get stuck in strange positions especially the wrists. In these cases you can reset the joint using the command:

command("ResetJoint", <JointName>)

Once you have all of the joints homed correctly it's time to enable them for control. Before you do this step make sure that every joint is in the correct home position. If it is off from it the joint will jump to the 0 position as soon as you enable. This can be very hazardous and potentially draw to much current on the real robot. In simulation you can enable right after you initialize MAESTOR. To enable the joints run the command:

command("EnableAll", "")

From here you are fully initialized and are ready to control the robot. I recommend running these initialization methods by hand in the MAESTOR console then running scripts for motions. The reason I say this is because when you run everything on the real robot it may take a large number of homing attempts before you get it right.

6.6 Meta Joints

MAESTOR handles its implementation of inverse kinematics by using an abstraction that we call Meta Joints. Meta Joints are special robot components then control a group of regular joints. For example the Right Foot Meta Joints manipulate all of the joints in the right leg. MAESTOR has implementation for Forward and Inverse Kinematics for the legs and a simplified Forward and Inverse Kinematics for the arms. Both implementations of the Inverse kinematics are done using geometry not inverse solvers so they are reasonably quick.

Foot Meta Joints -Each foot has three meta joints. They are the RFX, RFY, and RFZ and the LFX, LFY, and LFZ. The X an Y coordinates of both feet start at zero and the Z coordinates start at -.56 units. To crouch down you can run the command:

setProperties("RFZ LFZ", "position position", "-.52 -.52")

Arm Meta Joints -Each arm has three meta joints. They are the RAX, RAY, and RAZ and the LAX, LAY, and LAZ. There aren't any default values for the Arm meta joints but it may take some playing around in simulation to get used to where the best values for the meta joints are.

Neck Meta Joints -The neck originally does not have a neck pitch and neck roll on the hubo2+ so we created meta joints that work like a neck pitch and neck roll. They are NKP and NKR.

Each three letter code above is a Meta Joint and have properties that can be set and got.

6.7 Moving

One of the biggest things that you will want to do with MAESTOR is move joints. This can be done with the setProperty function.

To move A Single Joint:

setProperty(<Joint>, "position", <value>)

A grounded example of this command could be:

setProperty("RSP", "position", -1)

When you set the position of a joint to a value MAESTOR moves the joints at the current speed that it is set at to that position. The value should be in radians. A good check for safe positions for HUBO is to use Hubo-in-the-browser written by William Hilton.

To change the speed that the joint moves you can use the command:

setProperty(<Joint>, "speed", <value>)

The speed is in radians per second and should only be a positive number. You can use decimals for speeds such as .3 and 1.5 as well as whole number values. I do not recommend using speeds faster than 2 but there is no coded upper limit.

6.7.1 Joint Motion types

These are motion types that come from hubo-ach for each joint. The default is 1 and I highly recommend not changing it to anything else. But the option is there.

0 - Filtered motion. Hubo-ach will internally filter and interpolate the goal position.

1 - Non-filtered motion. Hubo-ach will set the motor reference directly.

6.8 Feedback

Another important feature that Maestor has is feedback. All of the feed back can be done with the getProperties function. I do not recommend trying to implement time critical feedback loops using the getProperties function because of ROS latency issues. If you really really really want to implement time critical feed back email Ryan Young: rdy29@drexel.edu and he can help you to do it.

To request multiple values from a single joint use the command:

getProperties(<Joint>, <Properties>)

Where properties is a space delimited string list of properties or a single property. The list of properties can be found in Table 3

6.9 Running Trajectories

*****WARNING*****

Trajectories can be very dangerous. There is no interpolation of the joint angles that you put in your trajectory file. If the gap is too big or the trajectory file has a bug in it the robot may cause harm to its self. Use trajectories with caution.

A trajectory is defined as a chain of non-zero length of Whitespace Separated Value files. Each file may contain columns of position values to be sent sequentially to joints which may be indicated by an optional header. A default header assumes that all 40 joints will be included in the following order:

RHY RHR RHP RKP RAP RAR LHY LHR LHP LKP LAP LAR RSP RSR RSY REP RWY RWR RWP LSP LSR LSY LEP LWY LWR LWP NKY NK1 NK2 WST RF1 RF2 RF3 RF4 RF5 LF1 LF2 LF3 LF4 LF5

All trajectory commands take strings as arguments unless specified otherwise. To load a trajectory, the following command is used:

loadTrajectory(<Trajectory Name>, <path to trajectory>, <playback>)

Where Trajectory Name is a short name to be used to refer to that trajectory later, and playback is a boolean value that should be passed as true if the trajectory is to be read from and false if the trajectory is to be written to.

To run a trajectory that is loaded:

startTrajectory(<Trajectory Name>)

To stop a running trajectory:

stopTrajectory(<Trajectory Name>)

In addition, Trajectories have the following useful features:

To force a trajectory to ignore an input from one of its named columns, use this command:

ignoreFrom(<Trajectory Name>, <Column Name>)

To unignore a previously ignored column:

unignoreFrom(<Trajectory Name>, <Column Name>)

To ignore all columns:

ignoreAllFrom(<Trajectory Name>)

To unignore all columns:

unignoreAllFrom(<Trajectory Name>)

To append a trajectory file to the end of a loaded trajectory*:

extendTrajectory(<Trajectory Name>, <path to trajectory>)

* A trajectory may only be extended if the new trajectory has the same number of columns, the same columns present (not necessarily in the same order), and must start relatively close to the last position vector of the current trajectory. Ignoring extra columns from loaded trajectories may be used to extend trajectories with shorter columns.

To set a trigger for another loaded trajectory** to be played at a certain integer frame***:

setFrame(<Trajectory Name>, <frame number>, <Trajectory Name to be Loaded>)

** This method can be used to have a trajectory start itself when the trajectory has finished

*** The "frame" of the trajectory is the number of rows of data which have been read. Comments/headers/empty lines do not count. A user can specify a frame of -1 to load another trajectory at the end of the given trajectory.

6.10 Stopping

Once MAESTOR is started it will run continuously in the back ground as will all of it's dependant packages. To stop MAESTOR run this command in a bash terminal:

maestor kill

7 Quick Start

If you want to get started immediately and read about the details of what you can do later, follow these steps:

- Run the MAESTOR run script:

maestor (run type)

For example, if running in simulation:

maestor sim

or, on hardware:

maestor real

If you chose "real" as your run type:

- A terminal should have opened on tty7 (This may be different if there are already processes running on these terminals.)
- It is not required but it is highly recommended to open tty7 (Ctrl - Alt - F7) and make sure the hubo-ach console is running (It may require elevated privileges.)

If you chose "sim" as your run type:

- X Terminals should have opened with hubo ach, roscore and additional simulation windows. Hubo-Ach's terminal will require elevated privileges. Type your password to give them.

-On occasion one of the xterminals will be directly over top of the other. If this happens and the one running hubo-ach that needs elevated permissions is on the bottom, simply move the top one and give hubo-ach permission.

Once everything has finished starting up for both real and simulation, the terminal you started MAESTOR in will be back in your control.
To open the MAESTOR console type the command:

maestor console

This will give you a python console that has the MAESTOR api and commands loaded. The function ls() will give you a list of all the functions you have available. Each console line starts with: >>> like the Python Interpreter.

-To begin controlling the robot or the simulation you must first initialize the robot:

initRobot("")

The parameter is the path to a config file. Empty string is default

- Next the joints must be homed. This is especially important in hardware mode. Usually when operating HUBO you will have to reset joints and re-home a few times before you can enable the robot. You can home using the command:

command("HomeAll", "")

- In order to move the joints they must be enabled. This can be done with the command EnableAll

command("EnableAll", "")

- From here the robot in hardware or simulation can move all of its joints. A simple test is to move the right arm up

setProperty("RSP", "position", -1)

The robot should move its arm up. From here you can begin to manipulate joints from the console using the setProperty function. A detailed guide listing the functionality is below in the How to use section.

8 External Sensors and Demos

Because MAESTOR is built off of ROS services it can be used to integrate external sensors with controlling HUBO. A lot of the demos that we have made use this feature and I suggest taking a look at them if you want to get an idea of how it works. It usually consists of creating your own Python scripts with a maestor object and another object that handles the external sensors.

To run hubo demos with MAESTOR, download them from the github repositories:

https://github.com/RyanYoung25/FaceTrackingWaistDemo
https://github.com/RyanYoung25/TwoArmSoundDemo
https://github.com/RyanYoung25/BiotacArmDemo

Each demo has a README with installation instructions which includes specifying them in the ros package path.

Some demos require other dependencies, see their README for more information.

9 Extras

9.1 Default Joint Name scheme

Joints:

WST Waist
NKY Neck Yaw
NK1 Neck Tilt 1
NK2 Neck Tilt 2
LSP Left Shoulder Pitch
LSR Left Shoulder Roll
LSY Left Shoulder Yaw

LEP Left Elbow Pitch
LWY Left Wrist Yaw
LWP Left Wrist Pitch
RSP Right Shoulder Pitch
RSR Right Shoulder Roll
RSY Right Shoulder Yaw
REP Right Elbow Pitch
RWY Right Wrist Yaw
RWP Right Wrist Pitch
LHY Left Hip Yaw
LHR Left Hip Roll
LHP Left Hip Pitch
LKP Left Knee Pitch
LAP Left Ankle Pitch
LAR Left Ankle Roll
RHY Right Hip Yaw
RHR Right Hip Roll
RHP Right Hip Pitch
RKP Right Knee Pitch
RAP Right Ankle Pitch
RAR Right Ankle Roll
RF1 Right Finger 1
RF2 Right Finger 2
RF3 Right Finger 3
RF4 Right Finger 4
RF5 Right Finger 5
LF1 Left Finger 1
LF2 Left Finger 2
LF3 Left Finger 3
LF4 Left Finger 4
LF5 Left Finger 5

Sensors:

IMU Body IMU
LAI Left Ankle IMU
RAI Right Ankle IMU

LAT Left Ankle Force Torque
RAT Right Ankle Force Torque
LWT Left Wrist Force Torque
RWT Right Wrist Force Torque

9.2 Response Characteristics

Maestor operates by default at an update cycle rate of 200 Hz. At these speeds, Maestor is able to push joint positions down and receive the updated position back with an average latency of 10 ms. Currently, it is questionable whether or not this response time is adequate for feedback loops. If the application merely requires joint references, rather than a feedback loop, the latency for commands sent to the robot is less than 1 ms on average. Test statistics are located in /maestor/test/timing, though they may be slightly cryptic to decipher.

9.3 Support

Contact us with any questions you have:

Ryan Young: rdy29@drexel.edu Jacky Speck: jas522@drexel.edu Eric Rock: igm@drexel.edu