

GEORGIA INSTITUTE OF TECHNOLOGY

CENTER FOR ROBOTICS & INTELLIGENT MACHINES

HUMANOID ROBOTICS LAB

---

# GT Mission Manual

---

M.X. GREY  
ERIC HUANG  
ANDREW PRICE  
PETER VIEIRA



# Contents

<b>Code Repositories</b>	<b>2</b>
Real-time Code . . . . .	2
ROS Code . . . . .	2
<b>Tasks Overview</b>	<b>3</b>
Wall Drilling Task . . . . .	3
Procedure . . . . .	3
Rubble Clearing Task . . . . .	4
Procedure . . . . .	4
<b>Operation Instructions</b>	<b>5</b>
Starting-up Hubo . . . . .	5
Starting-up Perception . . . . .	6
Camera Calibration . . . . .	7
Walking . . . . .	8
Manipulation . . . . .	10
Upcoming Features . . . . .	11
<b>Troubleshooting</b>	<b>12</b>

# Code Repositories

The following code repositories are required in order to execute our tasks using our code. They consist of real-time processes that run on the robot, and code that uses ROS and runs mostly on a local computer.

## Real-time Code

Code repositories that run in real-time on the robot.

- hubo-ach
  - <https://github.com/hubo/hubo-ach>
    - \* Dependencies: *ach*, *libreadline*.
- hubo-motion-rt
  - <https://github.com/hubo/hubo-motion-rt>
    - \* Dependencies: *hubo-ach*, *RobotKin*, *urdfdom*, *eigen3*.
- hubomz
  - <https://github.com/golems/hubomz>
    - \* Dependencies: *hubo-ach*, *hubo-motion-rt*, *expat*, *freeglut3*.

## ROS Code

Code repositories that use ros-groovy.

- hubo\_init : GUI for initializing Hubo and controlling hubo-ach level settings
  - [https://github.com/hubo/hubo\\_init/](https://github.com/hubo/hubo_init/)
    - \* Dependencies: *hubo-ach*
- hubo\_walk : GUI for ZMP walk generator
  - [https://github.com/hubo/hubo\\_walk](https://github.com/hubo/hubo_walk)
    - \* Dependencies: *hubo-motion-rt*, *hubomz*
- joystick\_integrator : Converts SpaceNav joystick positions to 3D poses
  - [https://github.com/a-price/joystick\\_integrator](https://github.com/a-price/joystick_integrator)
- hubo\_motion\_ros : Provides joint-level and end-effector-level teleoperation, MoveIt-based kinematics server, and temporary ROS trajectory executor.
  - [https://github.com/a-price/hubo\\_motion\\_ros](https://github.com/a-price/hubo_motion_ros)
    - \* Dependencies: *hubo\_ros\_core*, *joystick\_integrator*, *drchubo*, *ros-groovy-control-msgs*, *ros-groovy-moveit-msgs*

# Tasks Overview

## Wall Drilling Task

### Procedure

1. Identify the drill using robot/human judgement from robot perception feedback.
2. Walk to the drill and pick it up using vision and teleopraption from the hubo\_manipulation\_planner panel.
3. Identify wall location using human judgement of robot perception.
4. Walk to the wall and face it using the hubo\_walk panel.
5. Begin static balancing using hubo\_walk panel.
6. Use hubo\_manipulation\_planner to identify points on the wall defining polygon to drill out and it's distance away from the wall. Step closer if needed.
7. Send the points in 6-dimensions to the manipulation daemon using the end-effector interpolation mode to drill the object out.
8. Remove drill from wall and place on table or drop.



Figure 1: Picture of example target to drill out.



# Rubble Clearing Task

## Procedure

1. Identify the location of the rubble/door using robot/human judgement from robot perception feedback.
2. Walk to the rubble and face it using the hubo\_walk panel.
3. Begin static balancing and squat to height which allows picking up rubble items.
4. Use dual-arm teleoperation to pick up the pieces of rubble and toss them to the side.

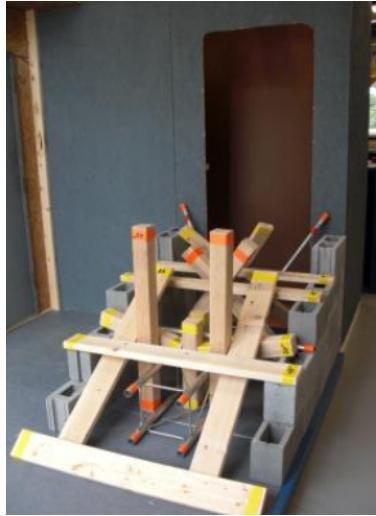


Figure 2: Picture of example rubble scene.

# Operation Instructions

This section details how to operate Hubo from the initialization phase to execution of the tasks. The different phases include *Starting-up Hubo*, *Starting-up Perception*, *Walking*, and *Teleop Manipulation*.

## Starting-up Hubo

This section describes how to start-up Hubo.

1. Turn on motor controllers with remote
2. Connect to Hubo's computer using ssh, and start the pipeline.

```
$ ssh hubo@192.168.0.201
$ ssh sudo service hubo-motion start
```
3. On your local computer (outside of the ssh session), launch RViz

```
$ roscore
$ rosrun rviz rviz
```
4. Click on the *HuboInitPanel* panel tab.
5. Ensure "IP Address" is correct.
6. Click "Connect" to send and receive ach channels between Hubo's computer and the remote computer. If it won't connect see the Troubleshooting section.
7. Click "Home All" to home all of Hubo's joints, if not on the ground and not homed. If already homed and standing, continue skip to step.
8. If any active joints do not home, indicated by the color in the *Joint State* section not being gray, click the "Rehome". Button color meanings are
  - GRAY: homed
  - PURPLE: not homed
  - RED: error
  - WHITE: inactive
9. If any joints won't home, you can do one of two options:
  - (a) Click the *Reset* radio button and then click the joint(s) that won't home. Then click the *Home* radio button and click the joint(s) that didn't home. Then press the "Rehome" button again.
  - (b) Run "sudo service hubo-motion stop", turn off the motor power, wait a moment, turn it back on, and run "sudo service hubo-motion start". Then try homing again from the beginning.
10. Once the joints are all homed, press the "Start Sensors" button to initialize the IMU and Force/Torque sensors.
11. Click on the *Sensor State* tab and ensure the sensor values are updating and are reasonable. If they aren't, then restart the whole process.

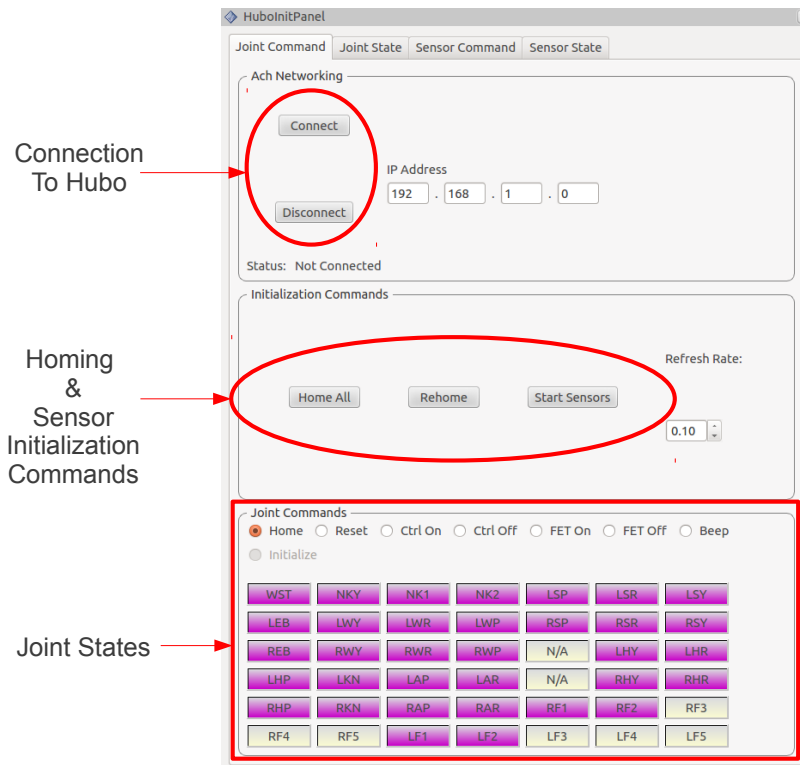


Figure 3: Picture of hubo\_init panel.

## Starting-up Perception

This section is heavily under construction. Generally, what you will need to do is something like this:

```
$ roslaunch hubo_drc_vision hubo_openni.launch
```

## Camera Calibration

In many circumstances, you may find that an approximate camera calibration is insufficient for real manipulation tasks. In order to alleviate this problem, the intrinsic and extrinsic properties of the camera need to be computed. For intrinsic calibration, we used the ROS package `camera_calibration` ([http://wiki.ros.org/camera\\_calibration](http://wiki.ros.org/camera_calibration)). If using a PrimeSense camera, both the infrared and rgb cameras will need to have their intrinsics calibrated as documented in the package.

For extrinsic calibration, we have written a small subroutine to assist with the calculation. There are several steps:

1. Attach checkerboards to the flat places on the hands. Example checkerboards are provided in the `hubo_gt_ros/hubo_drc_vision/resources` folder.
2. Measure the transform from the hand origin to the checkerboard origin. These values are currently hard-coded for a single checkerboard in the main function of `camera_calib.cpp`; this needs to be expanded to accept ROS parameters.
3. Launch the calibration procedure as follows:

```
$ roslaunch hubo_drc_vision calibrate_camera.launch
```

This will load a list of joint angles from a csv file and play that "trajectory" through the motion pipeline. A second node will monitor the vision system and will compute the least-squares transform that best solves the system of observed and expected points. This transform will be published into TF, and the results can be viewed in RVIZ (use `calibration.rviz` to automatically configure RVIZ to display the relevant topics).

## Walking

This section describes how to make Hubo walk. And assumes the robot is already up and running and standing on the ground on two feet.

1. Click on the *HuboWalkPanel* tab.

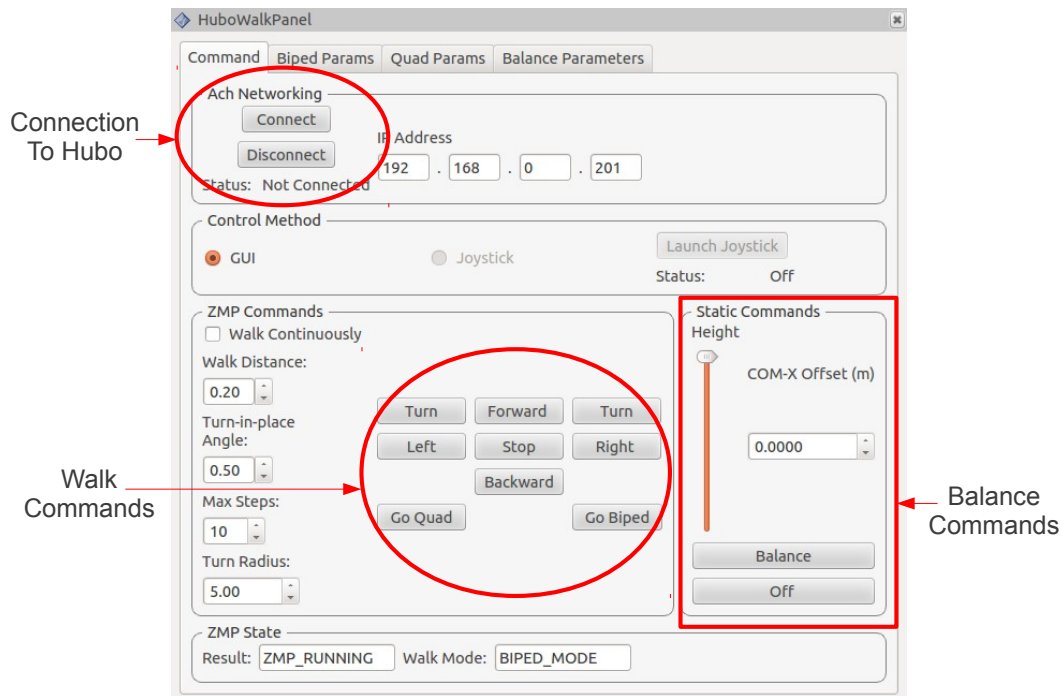


Figure 4: Picture of hubo\_walk panel.

2. Ensure the IP Address is correct.
3. Click “Connect” to send and receive ach channels between Hubo’s computer and the remote computer. If it won’t connect see the Troubleshooting section.
4. Based on vision feedback enter desired walk distance, max number of steps, and turn-in-place angle (for turning in place).
5. Click the “Go Quad” button to transition into quadruped mode.
6. Click the walk command desired to get to your desired goal. (Eventually the interface will use a desired location and orientation, and the ZMP walk generator will create a footstep plan to get there.) The *Walk Continuously* checkbox determines whether or not to walk the specified distance or walk until receiving a stop command. The *ZMP State* section at the bottom of the panel display the state of the ZMP walker generator.
7. Click “Go Biped” to get back up on two feet in order to manipulate objects.
8. In the *Balance Commands* section of the panel, click ”Balance” to go into static balancing mode.

9. The x-location of the center of mass can be adjusted in the *COM-X Offset (m)* field. Then click the "Balance" button to send the command.
10. The squat height of the robot can be adjusted by sliding the *Height* slider up and down, which automatically sends the command to the robot.

## Manipulation

This section describes how to teleoperate Hubo in order to dual-arm manipulate objects in the environment.  
NB: First install all software outlined in Code Repositories

1. If you haven't already, start up ROS

```
$ roscore  
$ rosrn rviz rviz
```

Configuring RViz interface

- If the Displays panel is not visible already, click on the Panels menu and click on Displays.
- Under Global Options, set Fixed Frame to /Body\_RAP.
- Click Add and scroll to RobotModel.
- Click Add and scroll to Pose. Set Topic to /rh\_pose. Set Axes Length to 0.1 and Axes Radius to 0.01.
- Click Add and scroll to TF. Set Marker Scale to 0.1.
- Click Add and scroll to InteractiveMarkers. Set Topic to /joint\_controls/update.

2. Start the teleoperation interface

```
$ roslaunch hubo_motion_ros fullbody_teleop.launch
```

3. (Optional) Start up Vision system as discussed in Starting-up Perception.

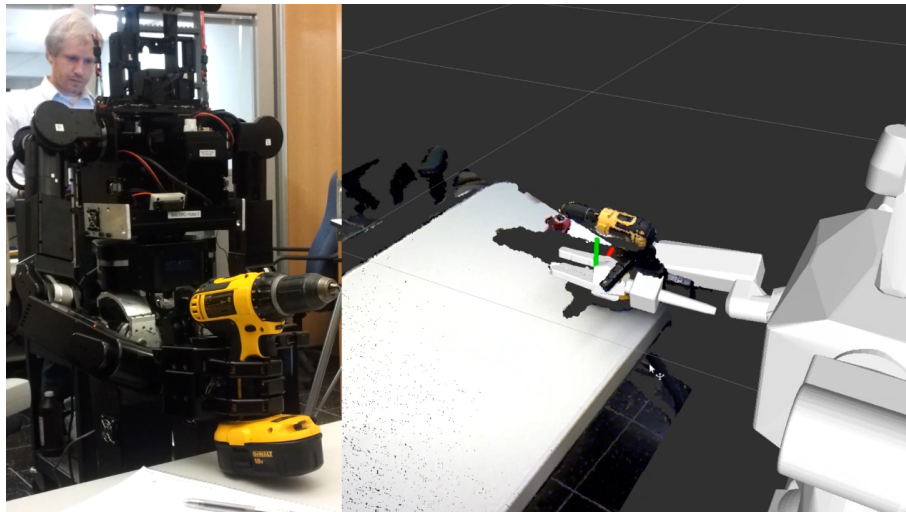


Figure 5: Picture of teleoperation of picking up a drill.

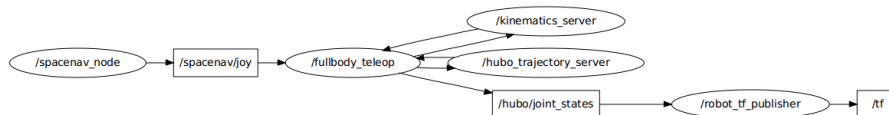


Figure 6: Simplified ROS Graph created by fullbody\_teleop.launch.

## Upcoming Features

This section lists the features which we intend to implement and stabilize prior to the DRC. Many of these features we consider to be crucial for successful teleoperation of the task.

1. End effector Force-Torque workspace control
2. Cooperative dual-arm manipulation
3. Simultaneous dual-arm manipulation
4. Human-assisted planning phase for collision avoidance



# Troubleshooting

Table 1: Troubleshooting Table

Symptom	Causes	Solution
Can't ssh to Hubo's PC	<ol style="list-style-type: none"> <li>1. Not on correct network.</li> <li>2. Hubo's PC is stuck in the Grub menu</li> </ol>	<ol style="list-style-type: none"> <li>1. Connect to the correct network.</li> <li>2. Connect a monitor and keyboard to Hubo's PC and debug, or just a keyboard and press enter.</li> </ol>
Can't connect to the robot in RViz panel	<ol style="list-style-type: none"> <li>1. Not on the correct network.</li> <li>2. The daemons are not running on Hubo</li> </ol>	<ol style="list-style-type: none"> <li>1. Connect to the correct network.</li> <li>2. Ensure daemons are running by typing <code>ps aux   grep daemon</code> or restarting Hubo and running <code>sudo service hubo-motion start</code>.</li> </ol>
Receiving ACH_OVERFLOW from rviz command. HuboInitPanel joint names don't showing up correctly.	Don't have matching or most up-to-date installs of hubo-ach, hubo-motion-rt, hubomz or possible another repo.	Make sure you are on the right branch in each repo and reinstall on both machines.
Robot homes only the upper or lower body, or neither.	CAN line got corrupted.	Shutdown and turn off the Hubo PC and turn it back on and try again.