

# MISSION MANUAL: HOSE INSERTION



Allen, Peter, Ph.D

Mayr, Anton                      Caimano, Ben  
ajm2217@columbia.edu    blc2129@columbia.edu



Oh, Paul, Ph.D

Jun, Youngbum  
1075jun@gmail.com

Last Updated: October 1, 2013

# Contents

<b>1</b>	<b>Task Description</b>	<b>3</b>
<b>2</b>	<b>Approach</b>	<b>3</b>
<b>3</b>	<b>Repositories, Installation and Execution Instructions</b>	<b>4</b>
3.1	Repositories . . . . .	4
3.2	Installation . . . . .	4
3.3	Execution . . . . .	5
<b>4</b>	<b>Vision</b>	<b>5</b>
4.1	Brief . . . . .	5
4.2	Activation and Use . . . . .	6
4.3	Done/To Do . . . . .	6
<b>5</b>	<b>Planning</b>	<b>7</b>
5.1	Brief . . . . .	7
5.2	Activation and Use . . . . .	7
5.3	Done/To Do . . . . .	7
<b>6</b>	<b>Control and Balancing</b>	<b>8</b>
6.1	Done/To Do . . . . .	8
6.2	Walking with Load . . . . .	8
6.3	Balancing . . . . .	8
<b>A</b>	<b>Operational Flow</b>	<b>9</b>



## 1 Task Description

The essential task consists of two generalized manipulation tasks and the requisite walking between them. The first manipulation task is the grasping of an end of the hose to allow us to maneuver it towards a desired location (i.e. the proximity of the standpipe or the hydrant). The second is the insertion of the end of the hose into a threaded end. This requires both the physical mating of the hose to the end and the application of a consistent torque while holding the hose in the mated position. This must be done once for the standpipe and once for the hydrant. It is expected that the standpipe receives the male end of the hose and the hydrant receives the female end. The original position of the hose and standpipe to the hydrant is a distance of 15 yards. The robot must traverse the distance while compensating for impulse fluctuations provided by carrying/dragging the hose.

## 2 Approach

Our approach is fairly linear:

1. We locate the hose through a combination of teleoperation and vision.
2. We move to a good position to grasp it.
3. We grasp the hose and then pull it slack.
4. We locate the position of the standpipe.
5. We move to a good position to insert the hose.
6. We insert the hose.

Each manipulation or locomotion step of the process requires user oversight so as to reject solutions that may be correct but unwise. The locate steps utilize user input to massively simplify calculations.

This is a high level overview of the process. For more comprehensive delineation of the operation flow, see Appendix A. For an example of operator commands given along this approach, see Appendix B

## 3 Repositories, Installation and Execution Instructions

### 3.1 Repositories

The primary repository for our mission is the `hubo/hubo_planning_common` repository on github:

`https://github.com/hubo/hubo\_planning\_common`

The `develop` branch contains the latest release. Ideally, by the time of the competition, the `master` branch will be synchronized with this release.

The following repositories are primary dependencies of our code:

`https://github.com/hubo/hubo-ach` - The core hubo firmware layer

`https://github.com/daslrobotics/drchubo/tree/develop` - The models of the robot for simulation

`https://github.com/hubo/openHubo` - The `openRave/CBiRRT/hubo` combo package

`https://github.com/hubo/hubo\_vision\_common` - The vision system common repo

`https://github.com/hubo/hubo\_ros\_core` - The WPI utility repo

`hubo_init` is not strictly necessary but highly useful.

### 3.2 Installation

**Warning: Our project is heavily dependent on ROS infrastructure. Please install and familiarize yourself with ROS before proceeding.**

Each ROS repository should be installed in the `src` folder of your catkin workspace. The `hubo-ach` repository must be compiled using the typical `configure-make-make install` commands. `OpenHubo` should be installed and placed in your `PYTHONPATH`. Finally, `catkin_make` should be called in the root of your catkin workspace to compile the ROS environment.

### 3.3 Execution

The primary ROS package for our planner is the `openhubo_planning_server` package. The `OpenHuboPlanningServer` node must be running for the simulator to work. It can be launched simply with the command:

```
roslaunch openhubo_planning_server planning_server.launch
```

This will automatically be called when our rviz panel is loaded. Once the node is launched, it can be sent `AddObject`, `Grasp`, and `Move` service calls to simulate robot movements. `AddObject` will add specific obstacles to the environment. `Grasp` will alter the state of the fingers (and return the trajectory thereof). `Move` will actively find a desired trajectory and return it. In addition, it listens on several message topics for commands to the robot state. Any client that can build a `TSR` message and the requisite service requests can direct the server.

The client for our use will be packaged into the rviz panel. See the Vision, Planning and Control sections for details on how to use the panel.

## 4 Vision

### 4.1 Brief

Our vision system is based upon the fine work of WPI and Georgia Tech. The user will be shown a low fps ( 1fps) point cloud feed from the robot sensors via the rviz interface. Additionally, the rviz interface will show the configuration state through a 3d model of the robot.

The user will direct the interface to map an idealized model to the the point cloud. (Typical models are low-detail representations of the ends of the hose, a standpipe, and a hydrant.) The user also has control over the position of the idealized model in the robot frame. This can be used for brute placement or guess positions for the mapping. The end result is assisted positioning and orienting of the desired object frame.

Moreover, the point clouds extracted from the sensors are intended to be transformed into a form of occupancy grid. (The occupancy grid will most likely be an octomap.) This occupancy grid will be used to check for expected collisions in later path planning.

## 4.2 Activation and Use

**Warning: the following is a prediction of how to use the final developed project. These features are not currently implemented.**

The following should be done before the vision system is used:

- Add the hubo\_planning\_common panel to the rviz interface.
- Make sure that the sensor-handling ROS node from hubo\_vision\_common is spinning and without undesired errors. If it is not, start said node.
- Start the octomap\_server node.

Once the previous steps are complete, click the "Initialize Sensors Representation" button in the panel to add the point cloud objects to the rviz display.

To add a model to be positioned and oriented, right click on the screen, select "Insert Model" in the drop-down menu, and select the appropriate model in the submenu. This model will appear as an interactive marker in the clicked location. In addition, its position and orientation will be provided via ROS service to any ROS node that requests it. It can be positioned and oriented via the 6-DOF control markers that appear with it. Once the position is close to desired, select "Align Object" in the right-click menu to attempt to fit the marker to the point cloud algorithmically.

## 4.3 Done/To Do

**To Do:**

- Adapt/create semi-autonomous object recognition algorithm
- Capture environment and include it in path-planner for obstacle avoidance/collision checking
- Test algorithm in real environment

**Done:**

- Set up Primesense RGBD Camera
- Display robot and sensed environment via rviz

## 5 Planning

### 5.1 Brief

Our planning system involves a strong ROS infrastructure built on Rosen Diankov's openRave, Dmitry Berenson's CBiRRT2 and Rob Ellenberg's openHUBO. A basic persistent state server node is able to receive simulation commands and update itself via a hubo-ach interface.

The planning server consists of three main components for its algorithm:

- Environment  
The environment is loaded from a file specified in the launch file for OpenhuboPlanningServer. It contains merely the robot model, a camera frame, and a floor. It is populated with data from the sensors. It is primarily used for obstacle avoidance.
- Robot  
The robot model is also loaded from a file specified in the launch file for OpenhuboPlanningServer. Its joint state is modified by the algorithm and may be populated by hubo-ach.
- Target Object/Constraints  
The server may be given several manners of TSR constraints to plan on relative to objects or frames. In either case, the input is abstracted to a stereotypical CBiRRT2 input.

### 5.2 Activation and Use

As mentioned in subsection 3.3, to start the planning server run:

```
roslaunch openhubo_planning_server planning_server.launch
```

From that point, right click an object you added as in subsection 4.2 and select an appropriate option (reach, walk to, insert, turn). This will generate a trajectory which will be listed in the panel. Simply select the new trajectory and click run.

### 5.3 Done/To Do

**To Do:**



- Load graphically determined environment
- Perfect state mechanism

**Done:**

- Take TSR input
- Plan cleanly
- Load state from physical robot
- Return trajectory of movement in hubo-ach and ROS trajectory format

## **6 Control and Balancing**

**#Will come from Youngbum#**

### **6.1 Done/To Do**

### **6.2 Walking with Load**

### **6.3 Balancing**

## A Operational Flow

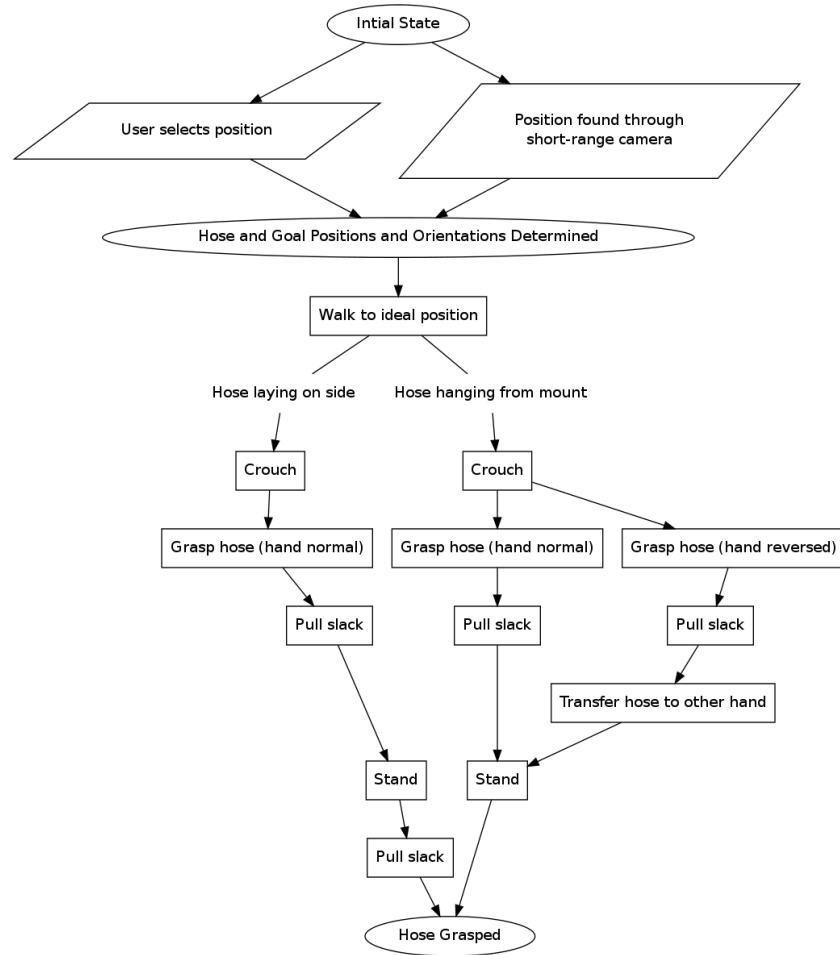


Figure 1: Grasping Operational Flow



Figure 2: Insertion Operational Flow

## B Example Task Walkthrough

The follow are the basic steps of the task:

1. Start the planning server and launch rviz with the panel loaded.
2. Orient the robot head through general control until the head of the hose is visible.
3. Place a target object in rviz near the head of the hose.
4. Tell rviz to align the object.
5. Select 'walk to' in rviz for the object and run the trajectory.
6. Select 'grab' in rviz for the object and run the trajectory.
7. Orient the robot head through general control until the stand pipe is visible.
8. Place a target object in rviz near the input of the stand pipe.
9. Tell rviz to align the object.
10. Select 'walk to' in rviz for the object and run the trajectory
11. Select 'insert' in rviz for the object and run the trajectory
12. Select 'turn' in rviz for the object and run the trajectory repeatedly until the hose is secure.