

# **Mission Manual: Door Opening**

## ***Version 1.0***

(Will be continuously updated along with the development of each component)



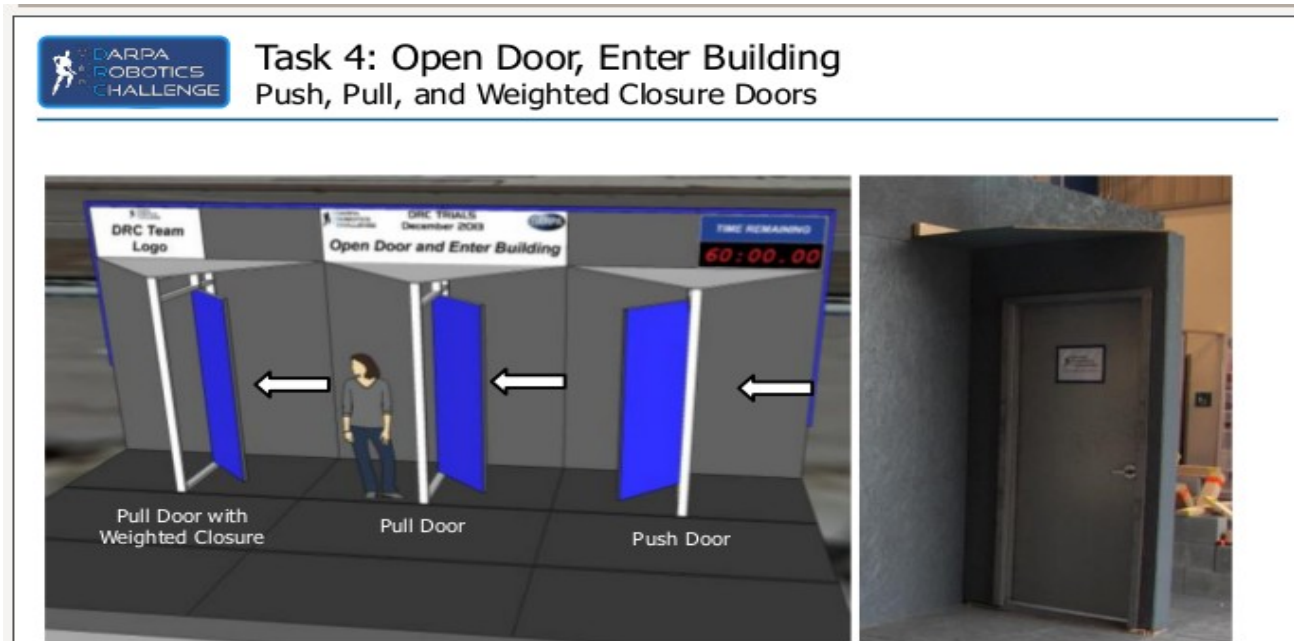
**Dr. Matt Zucker  
Temple Price  
Will Schneider**



**Brittany Killen**

## Event Summary

- **Event Description:** Open door and enter building.
- **DARPA Specification:**



*Setup at DRC Semi-finals 2013.*

- **Key Parameters:**
  - Three 80cm metal doors  
(one push, one pull, one pull with weighted closure)
  - All doors use lever handles

## Event Pipeline

coming soon...

## Perception

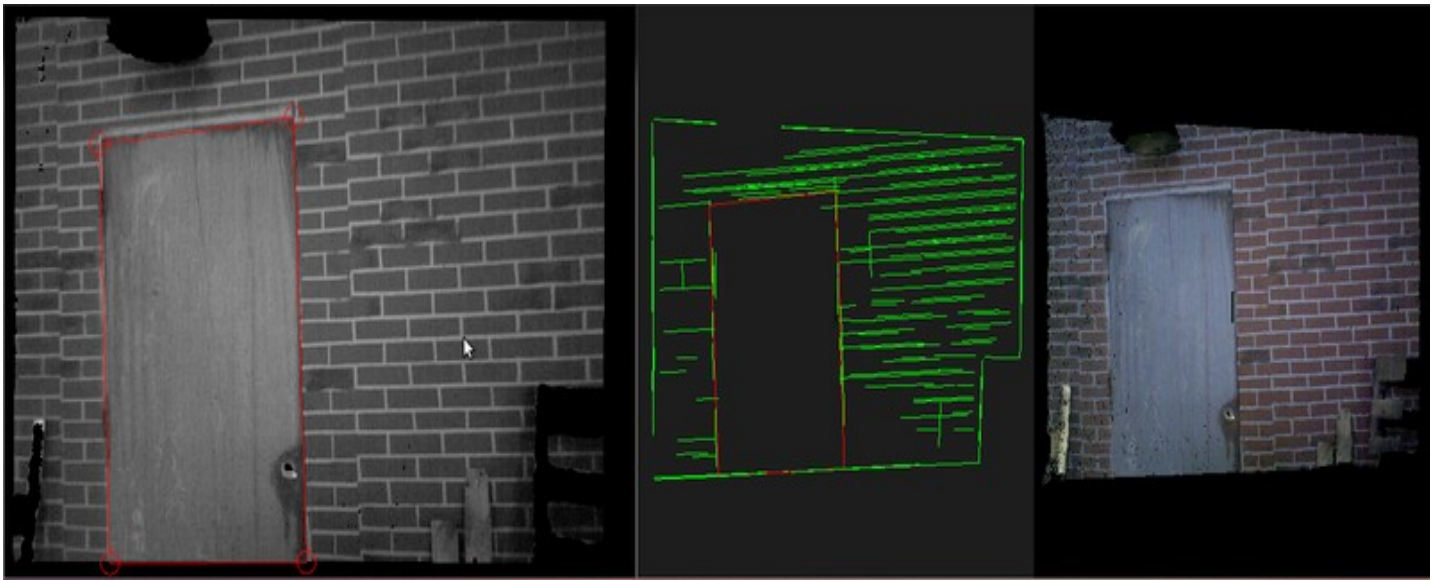
- **Input:** Pointcloud data from Hubo camera head
- **Output:** Rviz Door Detection Panel

Perception can be utilized using the Door Rviz Plugin:

[https://github.com/swatbotics/door\\_rviz\\_plugin](https://github.com/swatbotics/door_rviz_plugin)

Description:

- An Rviz panel developed to detect lines and edges
- User can click on points and find the distances and points needed for the robot



*Line and edge detection being captured using Asus camera.*

## Planning/Execution

- **People To Contact:**  
Dr. Matt Zucker ([mzucker1@swarthmore.edu](mailto:mzucker1@swarthmore.edu))
- **Software Packages Used:**
  - Hubo-Ach
    - <https://github.com/hubo/hubo-ach.git>
  - Planning and Execution can be done using the program Hubomz:
    - <https://github.com/swatbotics/hubomz>

### USE:

#### How to start the program:

The following command would run a doorOpening simulation using the regrasp.txt config file in the dooropen trajectory:

```
./hubodemo ../dooropen/regrasp.txt
```

The following command would run the same simulation, but would not create gui for the user to interact with:

```
./hubodemo -nogui ../dooropen/regrasp.txt
```

The following command would run the same simulation, but would will allow the user to control the simulation and robot using the console:

```
./hubodemo -console ../dooropen/regrasp.txt
```

#### GUI Button commands:

if the user has started up the program with the gui, then the following button presses will do stuff:

"a" - run the whole UNOPTIMIZED trajectory.

If the trajectory is already being run, then stop the simulation.

"s" - run the whole OPTIMIZED trajectory.

If the trajectory is already being run, then stop the simulation

"." - go forward a single robot state

"," - go backward a single robot state

"<" - go backward 10 robot states

">" - go forward 10 robot states

"[" - go to the beginning state

"]" - go to the end state

"r" - toggle the rendering of the robot's geometry.

The robot can be rendered a solid grey, slightly transparent, or invisible

"g" - toggle the collision geometry.

#### Console commands:

if the user has started the simulation with the -console option. The gui commands are disabled in this instance, even though there is a gui there for visualization purposes. This method also allows the user to control the -nogui simulation.

help , h - print usage

quit , q - quit the program

run , r - run the whole trajectory

run <int>, r <int> - run for <int> number of steps  
                     <int> can be negative  
 view, v - run the trajectory in simulation  
 view <int>, v <int> - view for <int> number of steps  
                     <int> can be negative  
 zero, z - return to the zero state  
 pause, p - pause the running trajectory  
 gotophase <int>, gp <int> - from the current state  
                     go to the phase numbered <int>  
 gotostate <int>, gs <int> - from the current state  
                     go to the state with an index of intVal  
                     the destination state can be lower than the current state  
 info, ls, i - print out information on state and phase  
 set <variable> <value> - allows the user to set the  
                     value of some variables  
     Setable variables: slowdown <double>  
 replan - replan the trajectory starting from an arbitrary state.  
     This is liable to fail if the starting state is not the zero state.  
     It is recommended that you run the <zero> command before running this  
     command.  
 replan <int> - This command replans from the current state to and the specified  
                     phase. <int> corresponds to the phase that the user wants the  
                     robot to start at.

The main way that this console program can be used is to use the view command to view trajectories in simulation, and then use run to run them on the actual robot. if something goes wrong in the course of running the robot, then the user can use send the command "p" or "pause" to pause the robot. If the robot is in a bad place, the user can reset the robot to the zero state, view the trajectory to see if the trajectory is valid. Run the valid trajectory. Then use the replan command to replan the action from the current zero state.

VIEW EVERY TRAJECTORY BEFORE YOU RUN IT.

### IMPORTANT NOTES:

The replan <int> command is a dangerous little beast. It rarely gives a useable trajectory, and it is difficult to be able to intuitively tell which phase you should replan from.

The `info` command has very good synergy with `gotophase` and `gotostate` commands because it lists the current state number, the total number of states, the current phase, etc.

The `< set slowdown <double> >` command is extremely important. This should be set to a value of about 2 before testing motion on the robot. This just makes things safer.

The hands in simulation do not act like the hands in real life. In simulation, the hands simply interpolate from open to closed over the course of a phase. This is not how the real hands work, so be take what the hands do with a grain of salt.

## CONFIG FILE INFO:

The config files are very dense little monsters, the best way to get an idea of what each variable does is to look at the internal documentation in the config files. That being said, below I have outlined a couple of important parameters below.

doCollisions, doLargeIK, doFailedIK - these are different kinds of failures that you can tell the simulation to look for. Setting doCollisions to true will test for self collisions and robot-door collisions.

doLargeIK will test for discontinuities in the robot's movement caused by faulty IK.

doFailedIK will test for IK failures (when the robot fails to get to the specified place ).

initPosX, initPosY, initPosZ, initRot - set the initial starting point of the robot.

initialHipSwivelAngle, finalHipSwivelAngle, regraspHipSwivelAngle - the robot swivels its torso in order to best open the door. These variables control the degree to which the robot does this.

initDoorAngle, finalDoorAngle, regraspDoorAngle - this is how far in degrees that the door is open at different phases.

doRegrasp - This is set to true for a trajectory that does regrasping.

For a trajectory that does not do regrasping, several phases are skipped.

dt\_msec - this is the length in time that a single state takes in milliseconds.

Changing this will change the hertz of the trajectory.

optimize - If this is set to true, then the doorplanner will use chomp to optimize the robot's motion.

## EXECUTION ON ROBOT:

Uses the same commands as above.

1. ssh [Hubo@192.168.0.----](mailto:Hubo@192.168.0.----)
2. cd hubomz
3. cd build
4. Load Motion
  - ex. ./hubodemo -nogui ../dooropen/regrasp.txt
5. Set Slowdown Speed
  - ex. set slowdown 4 (slows down motion four times)

6. Run Motion
  - ex. Run
  - 1. Also, you can run parts of the motion using the gotophase command
    - ex. gotophase 2 (goes to second phase position instead of running whole motion)
7. quit
  - when finished running trajectory make sure to type quit to exit Hubomz

## **FUTURE GOALS:**

The console commands are not meant to be the final iteration of the user interface. These commands provide an easy framework for gui buttons and commands. It should be very simple to drop the function calls and methods that enable the console commands to work into a gui environment. This system also provides a great place to test possible commands and their use to the operator.