# IROS 2016 Grasping and Manipulation Competition Simulation Framework

### Release v1.0

### Kris Hauser

### 8/10/2016

This package describes the simulation framework for the IROS 2016 Grasping and Manipulation Challenge.

In addition to its use in the IROS competition, this package is also meant to be an open framework for benchmarking of robot hands and manipulation controllers. Contributions are enthusiastically welcomed for new robots, object datasets, task scenarios, and evaluation protocols!

The package is based on the Klamp't library, using the Python 2.7 language. Klamp't is developed by Duke University for robotics research, visualization, and education. Crucially for this competition, it can perform realistic robot physics simulation with scanned 3D objects. It can also simulate several robot sensors, such as force/torque sensors, contact sensors, and depth cameras.

## Robots

By default, the robot is a free-floating gripper. The gripper base may be moved arbitrarily in (x,y,z,yaw,pitch,roll) space, while the fingers may be controlled individually. The gripper currently implemented is the Righthand Robotics Reflex Gripper. It provides 4 DOF of actuation and several contact sensors.

You can design your own robots or grippers in URDF format with Klamp't's additional XML specifications. See the Klamp't URDF import tutorial at http://motion.pratt.duke.edu/klampt/tutorial_import_robot.html for more details.

For more advanced sensors, actuators, and underactuated transmission mechanisms, you can develop simulation plugins that allow you to apply and inspect underlying forces in the physics simulation. Documentation for this feature is still under development, but its main functionality should be fairly straightforward. See the Klampt/Python/klampt/sim/simulation.py file for the API, and the plugins/reflex_col.py file for an example of an implementation for the Reflex hand.

## Object Datasets

The framework currently supports the following two object datasets:

- ycb: the YCB dataset.
- apc2015: the Amazon Picking Challenge 2015 dataset.

## Dependencies

Foremost, this package requires the Klamp't 0.6.x or 0.7 Python APIs. There are some API changes between the two versions, most significantly in the Python API import structure and the physics engine's sensor simulation (specifically, the ContactSensor, LaserSensor, and DepthCameraSensor types, and functionality for "baking in" sensors into robot files).

Version 0.7 is strongly recommended to get the latest features and bug fixes, but be warned that it is still under active development and some of the older online documentation has not yet been updated. It is possible to use 0.6.x for this competition, but you will not have the capability to simulate certain sensors (e.g., contact sensors).

Both Klamp't 0.6.x and 0.7 have been tested on Linux Ubuntu 14.04, and should work with most versions of Linux. They should also work with Mac, possibly with a little work. Windows builds for v0.7 are not yet available. Windows power users can build the package with some some (substantial) work and Visual Studio.

Foremost, you will need to install the Klamp't Python API from source. Follow the Installation Tutorial found on http://motion.pratt.duke.edu/klampt/tutorial_install.html. You will need to install Assimp and PyQt4 as instructed in the tutorial.

The master branch is still at v0.6.2, so to get v0.7, you will need to perform the following tweaks:

- Before you build, switch to the v0.7 branch by running the following command line in the Klampt/ directory:

```
> git checkout v0.7
```

- and run the following in Klampt/Library/KrisLibrary before you build the dependencies:

```
> git checkout plan_devel
```

Now build as normal.

Once Klamp't is built and installed, download the IROS2016ManipulationChallenge folder as follows

```
> git clone https://github.com/krishauser/IROS2016ManipulationChallenge.git
```

Then download the YCB and APC2015 datasets as follows (note these are rather large datasets):

```
> cd IROS2016ManipulationChallenge/data/objects
> python download_ycb.py
> python download_apc2015.py
```

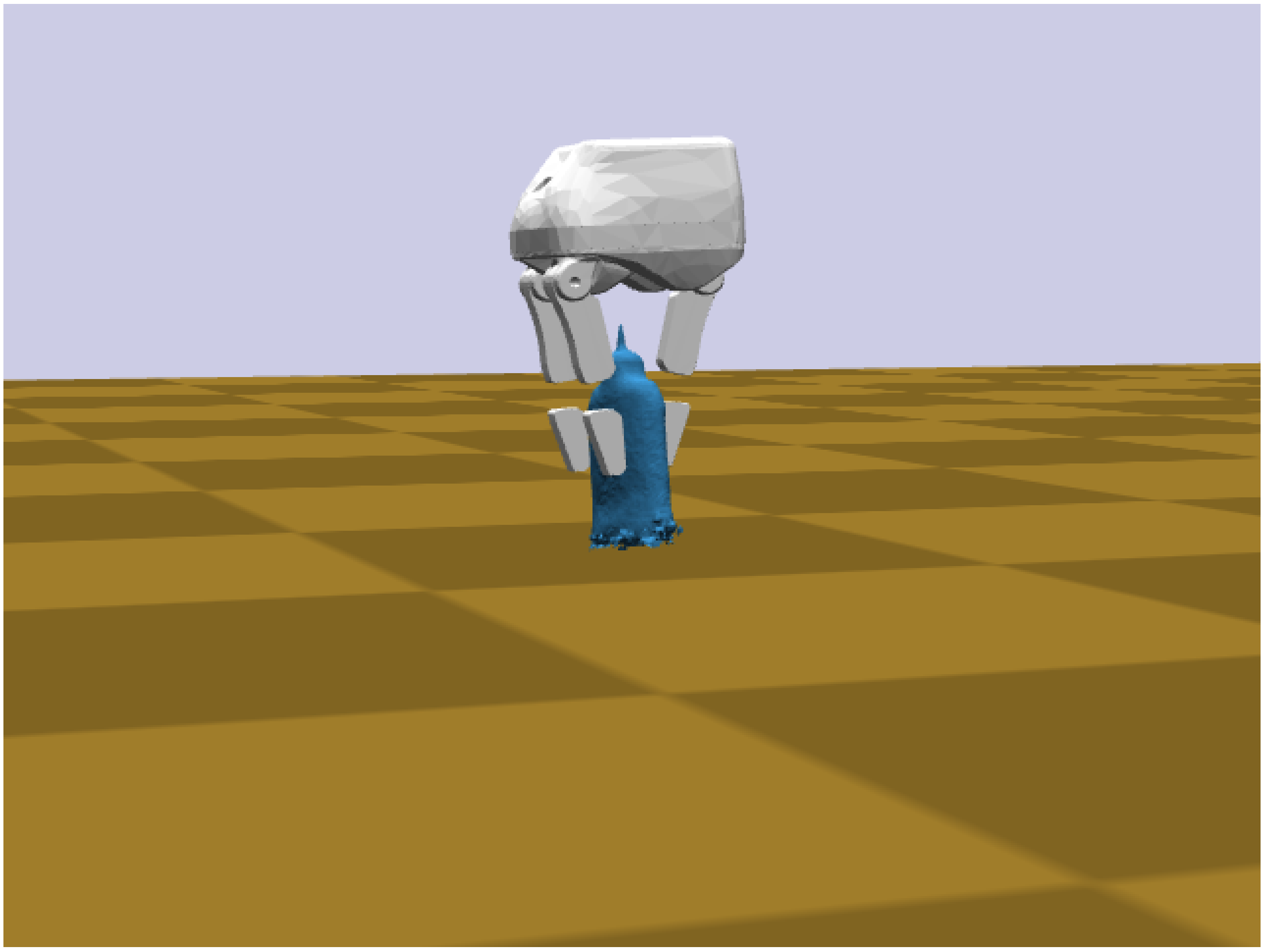
## Running the framework

[ Note: Version 0.6.x users will need to change some imports as instructed in the comments at the top of test.py and plugins/reflex_col.py ]

The main test file is main.py. A basic debugging mode is run as follows:

```
> python main.py [dataset] [object index or name]
```

If you do not specify a dataset or an object, one will be chosen for you at random. You will see a dialog box to edit the transform of the gripper. Once you press "OK", you will then see a simulation pop up where the gripper hopefully picks up the object.



If you re-run the program with the same object, you will see the last gripper transform you selected. This is because the transforms are being saved to IROS2016ManipulationChallenge/resources/[dataset]/*.xform.

To modify the controller for the basic test you may modify the simple_controller.py file. In this file, a single function called make() produces a control function that gets called repeatedly during the simulation. You are free to put whatever code you wish into the control function.

## Running the competition tasks

In the competition, your job will be to develop a controller that drives the robot such that it performs the following tasks. Teams are allowed to use any robot model they wish, provided that it sufficiently reproduces the behavior of some physical robot hand (either a commercial product or an experimental device are acceptable). Teams are also permitted to access "omniscient" sensor data, such as the positions of moving objects and contact forces, rather than using simulated sensors. However, teams that restrict themselves to using simulated sensors or use other innovative control approaches are eligible for honorable mentions.

### Programming

Teams will demonstrate their work on their own machines and will rewrite balls_controller.py and shelf_controller.py as necessary to accomplish the tasks. Any external modules and data may be used as needed. However, the scenario setup, simulation, and visualization code for each scenario in main.py must remain unchanged (for example, object coefficients of friction may not be increased to make objects easier to grasp).

Teams will rewrite the make() functions inside balls_controller.py and shelf_controller.py to produce a new control loop function. Please inspect the example code in each file for detailed instructions about how to send commands and access sensors in your control loop.

For competition, judges will copy a new file main_competition.py onto teams' computers and run it. This file will be very similar to main.py and will use teams' supplied controllers to drive the robot in instances of the task scenarios. Teams will not be told which instances they will be judged on until the day of competition.

Note: if you have developed a new hand module, ensure that it can be run by changing the constant 'reflex' at the bottom of main.py to the name of the new hand. Your team will be judged on whatever simulated hand it wishes to use for competition as long as it qualitatively matches the behavior of some actual robot hand.

Bindings from Python to other software packages can be done through a number of means, either by embedding external modules into Python code or by running parallel processes and performing inter-process communication (IPC). Instances of the embedding approach would include building Python wrappers or using the ctypes library to call shared library functions. Instances of the IPC approach would include the use of Rospy or implementing custom serial communications.

### Runs and Scoring

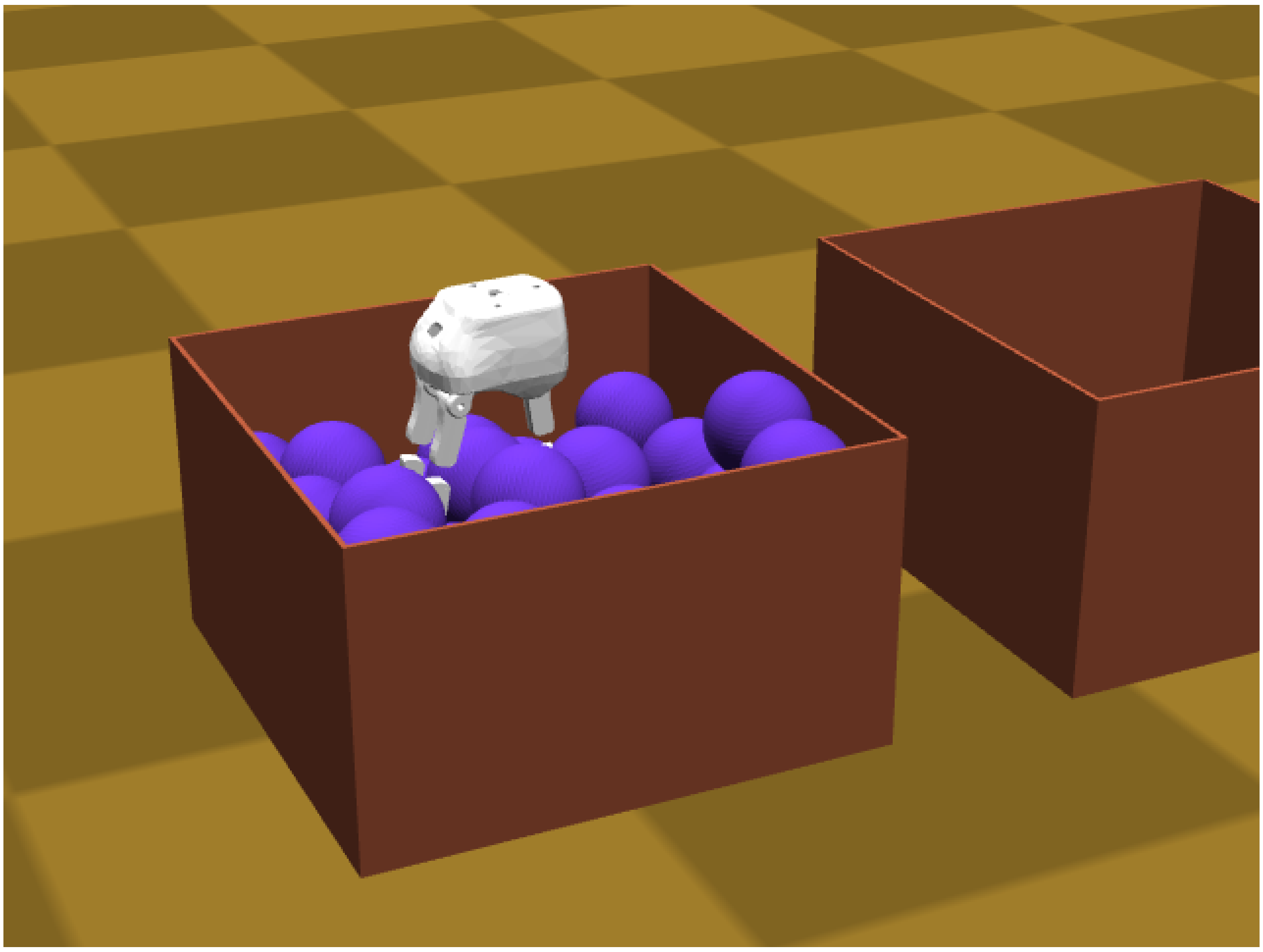
Teams will have up to 30 minutes to demonstrate the execution of these tasks, and consists of some number of ``runs'' on a given task. Each run continues until the team decides to stop it. The best scores on each task, maximized over all runs conducted on that task, will be recorded. Individual task scores will then be totaled to obtain an overall score.

## Task 1

Task 1 is to lift as many balls as possible from a box and deposit them into a second box. Code for this controller should be placed into balls_controller.py. To run the program, enter:

```
> python main.py balls [# of balls]
```

Balls that are successfully transferred receive 1 point, and balls that are dropped outside either box incur a penalty of 0.5 points.

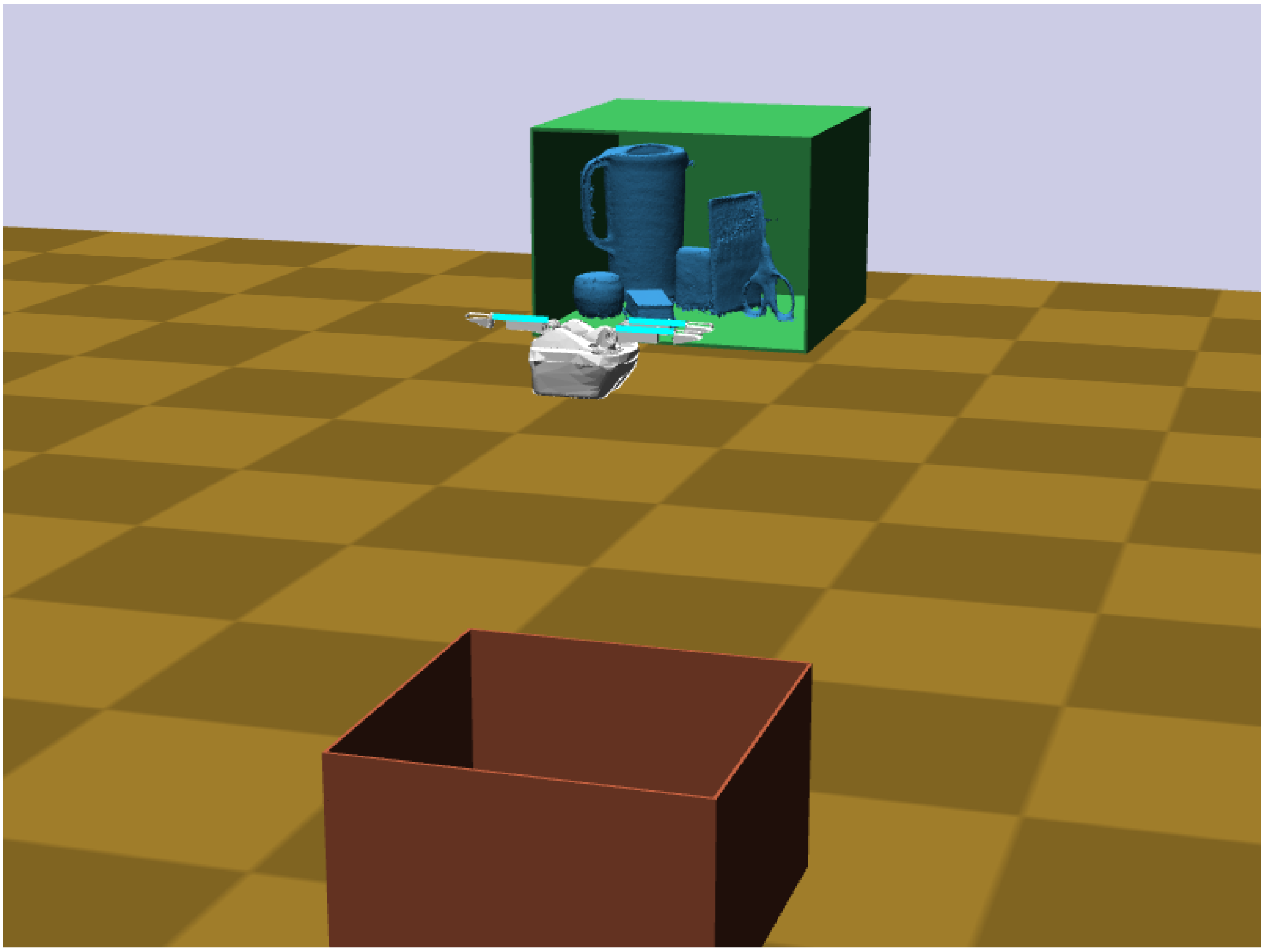


## Task 2

Task 2 is to extract as many objects as possible from a clutered shelf. Code for this controller should be placed into shelf_controller.py. To run the program, enter:

```
> python main.py shelf [# of objects]
```

Objects that are successfully extracted and placed into the box are given 5 points each.



## Bug Reporting and Contact

Bugs in the framework can be reported to the IROS Grasping and Manipulation Challenge mailing list (TBD). Bug fixes will be also reported to the mailing list.

Other questions can be directed to Kris Hauser at kris.hauser@duke.edu and Alessio Rocchi at rocchi.alessio@gmail.com.