======== Humanoid configuration ==========

- **Center of Mass**: kept at 0.93 meters
- **Head:** 33.0 cm above Center of Mass
- **Lidar:** 15.0 cm above Head
- **Kinect**: 7.0 cm above Head

======== How to load mat data ==========
Use “load_data.py” to load the data in python. The file includes: “get_joint()”, “get_lidar()”, “get_rgb()”, “get_depth()”, “getIRCalib()”, “getRGBCalib()”, “getExtrinsics_IR_RGB()”, “replay_lidar()”, “replay_rgb()”, “replay_depth()”. The outputs of “get_lidar()”, “get_rgb()”, “get_depth()” are arrays and each element is a dictionary with components described below. The length of the array is the number of measurements. The functions “replay_lidar()”, “replay_rgb()”, “replay_depth()” can be used to visualize and understand the data. You can change the start_frame, end_frame, and interval of the reply functions (e.g., in line 58 of load_data.py):

```
for i in xrange(start_frame, end_frame, interval):
```

========Joint angles (train_joint*.mat)==========

- x['ts']: timestamps (Absolute time)
- x['head_angles']: contains head and neck angles: array([[Neck angle], [Head angle]]) **(useful)**

========Lidar Information (train_lidar*.mat)==========
Hokuyo Lidar sensor: http://www.hokuyo-aut.jp/02sensor/07scanner/download/pdf/UTM-30LX_spec_en.pdf

- x['ts']: timestamps (Absolute time)
- x['delta_pose']: relative odometry between last reading [x, y, theta] (+x: forward, +y: left, +theta: counterclockwise rotation around z)
- x['scan']: 1x1081 lidar scan data, range -135° to 135°.

======== Camera Information (DEPTH_*.mat, RGB_*.mat)==========
Kinect v2 sensor: http://smeenk.com/kinect-field-of-view-comparison/

Camera data is provided only for training sets 0 and 3 and the test set.

- DEPTH_* contains depth images in **millimeters**

Intrinsic and Extrinsic camera parameters are provided by the functions “getIRCalib()”, “getRGBCalib()”, “getExtrinsics_IR_RGB()”. See the comments inside for details.

======== p3_utils.py==========
Contains implementations of the lidar scan to map correlation function and Bresenham's line rasterization algorithm, which is useful for determining the cells in an occupancy grid observed by a laser beam. The file also contains examples of how to use these two functions. You can consider obtaining a faster alternative to bresenham2D() by using cv2.drawContours() to obtain the cells corresponding to a whole lidar scan rather than individual rays.