Log

Please note that the investigation into processing 2D range image data is not included in this log, please see the first progress presentation for details that investigation.

Michael - 21/05/2012

Got kinect working with OpenNI drivers and the point cloud library http://www.pointclouds.org.

Have copied some sample code that displays the point cloud in a nice viewer and colours it and will use that as a base to build upon.

Very useful tutorial with lots of visualizer related stuff: http://pointclouds.org/documentation/tutorials/pcl_visualizer.php

Michael 23/05/2012 - With John

Attempting to get planes generated at multiple angles

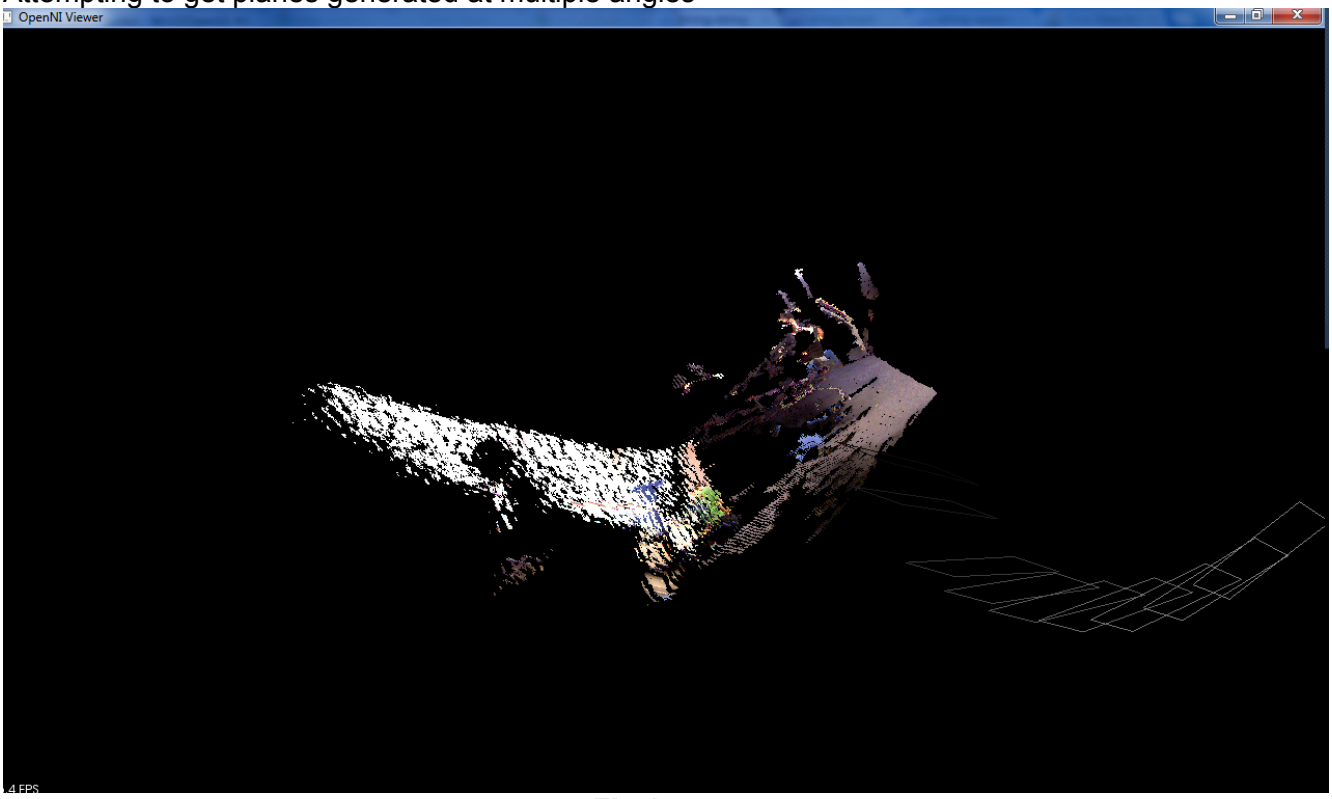

Fig 1

We then used the plane models built into the point cloud library to do brute force detection of the ground plane by looping over various heights and angles of planes and seeing how many points were near these planes.

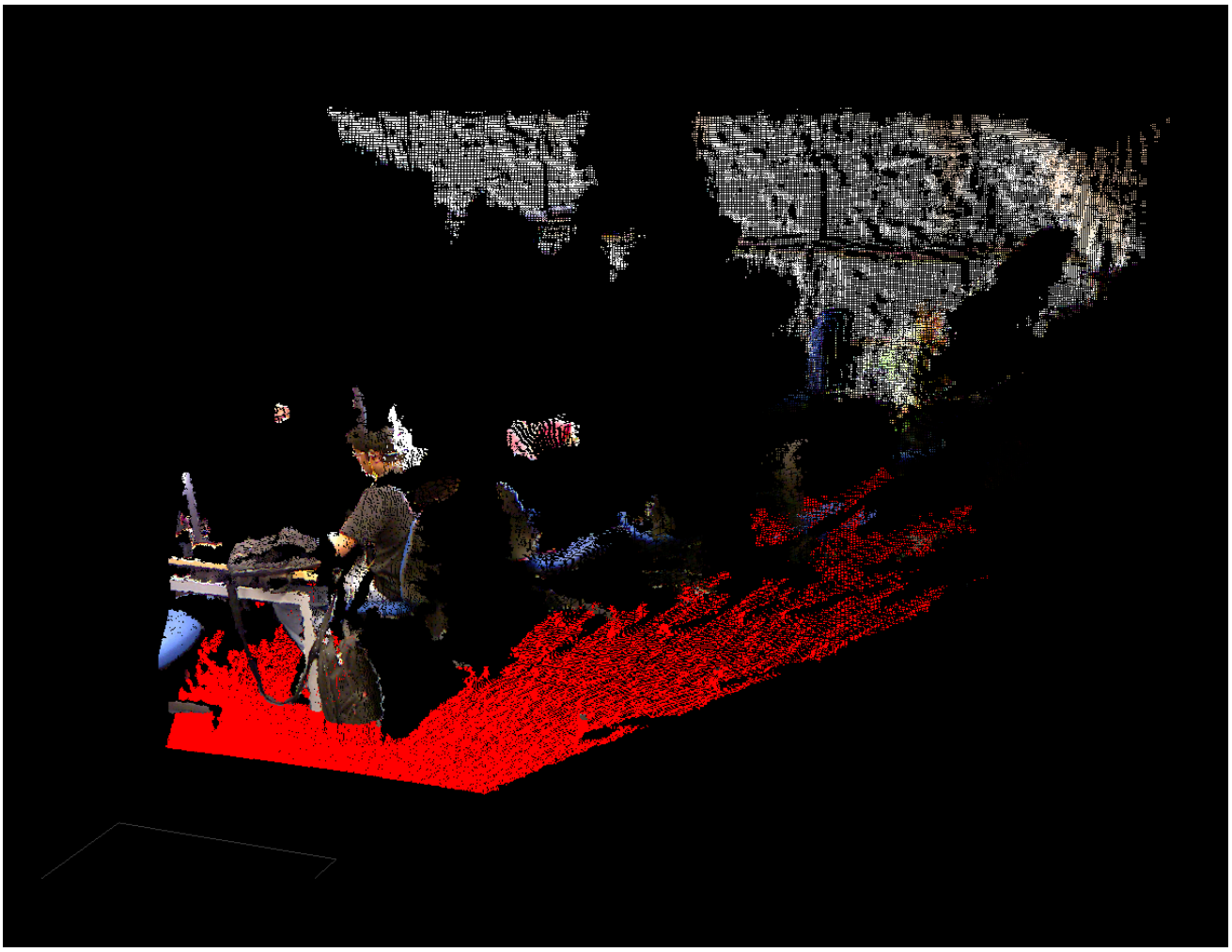

Fig 2

Michael (with John) - 24/05/2012

Useful object detection video with point cloud library: http://www.youtube.com/watch?v=jHKzBMKk4hY
They do voxel type downsampling type thing, then they segment the image using a pass through filter, then they do ransac plane detection before doing clustering.

This explains how to do some stuff like extracting planes http://pointclouds.org/documentation/tutorials/extract_indices.php#extract-indices

We have SAC model detecting the floor plane. It should extract planes perpendicular to the y axis but it seems to like choosing people as floor too and other random planes that don't seem to be perpendicular to the y axis. SOLVED: We were setting the model as SACMODEL_PLANE rather than SACMODEL_PERPENDICULAR_PLANE, changing this helped.

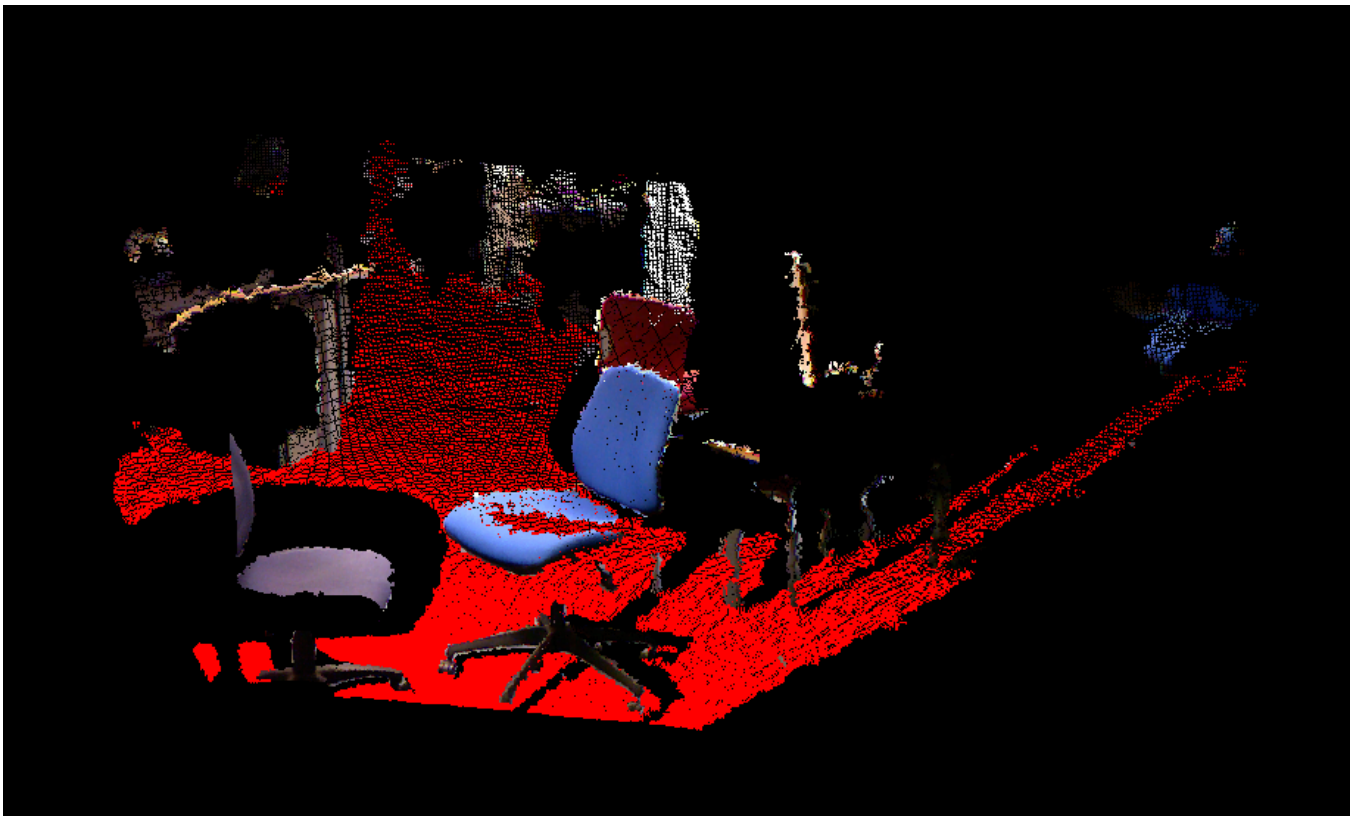

Fig 3

Image above: ground plane extracted and shown in red here. The cloud image has been pre-filtered to remove the ceiling and objects closer to the camera than the start of the floor.

We also experimented with some voxel filter downsampling to increase speed, but it does not seem to increase speed noticeably at this stage so for now we will not include it.

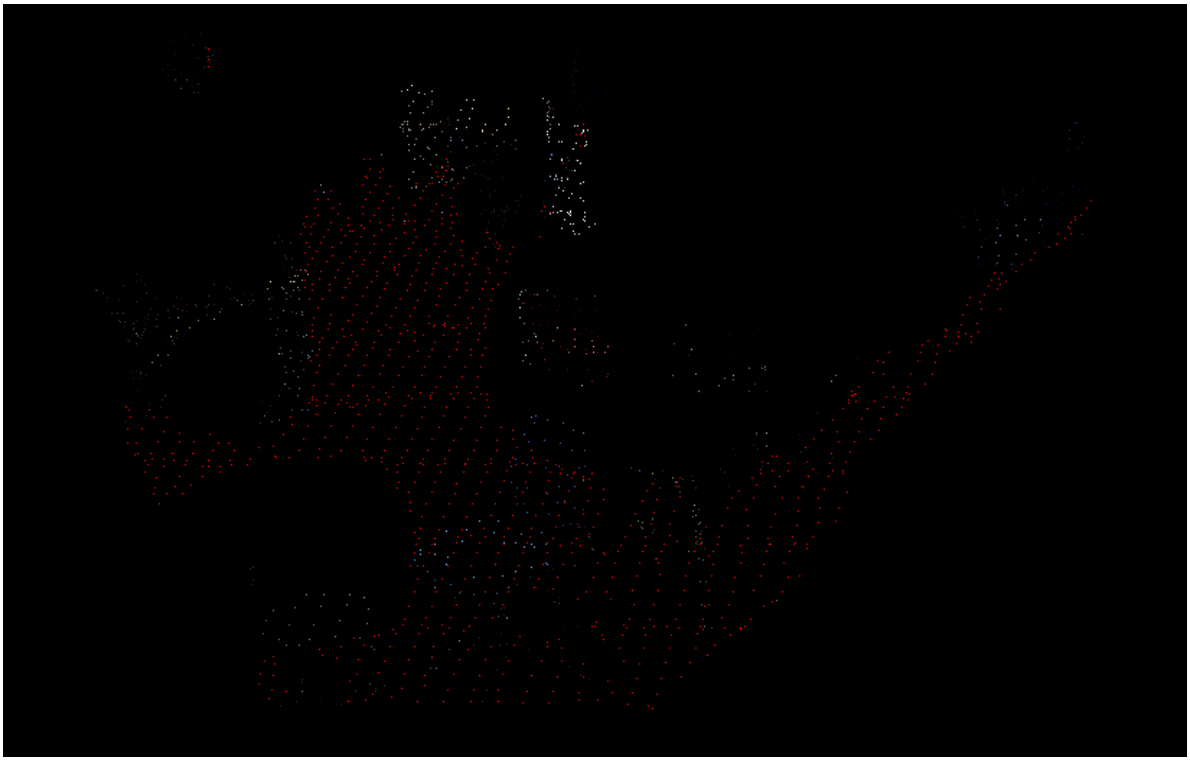

Fig 4: Voxel grid downsampling test on removed point cloud with ground coloured and ceiling removed. It can be seen that the point cloud points are uniformly spaced.

Michael - 25/05/2012

Spent ages trying to get image segmentation with rotation working but I think we finally have something working. The next step is to use these segments minus floor to determine which has the least points in them. Additionally we could try voxel downsampling to speed it all up.

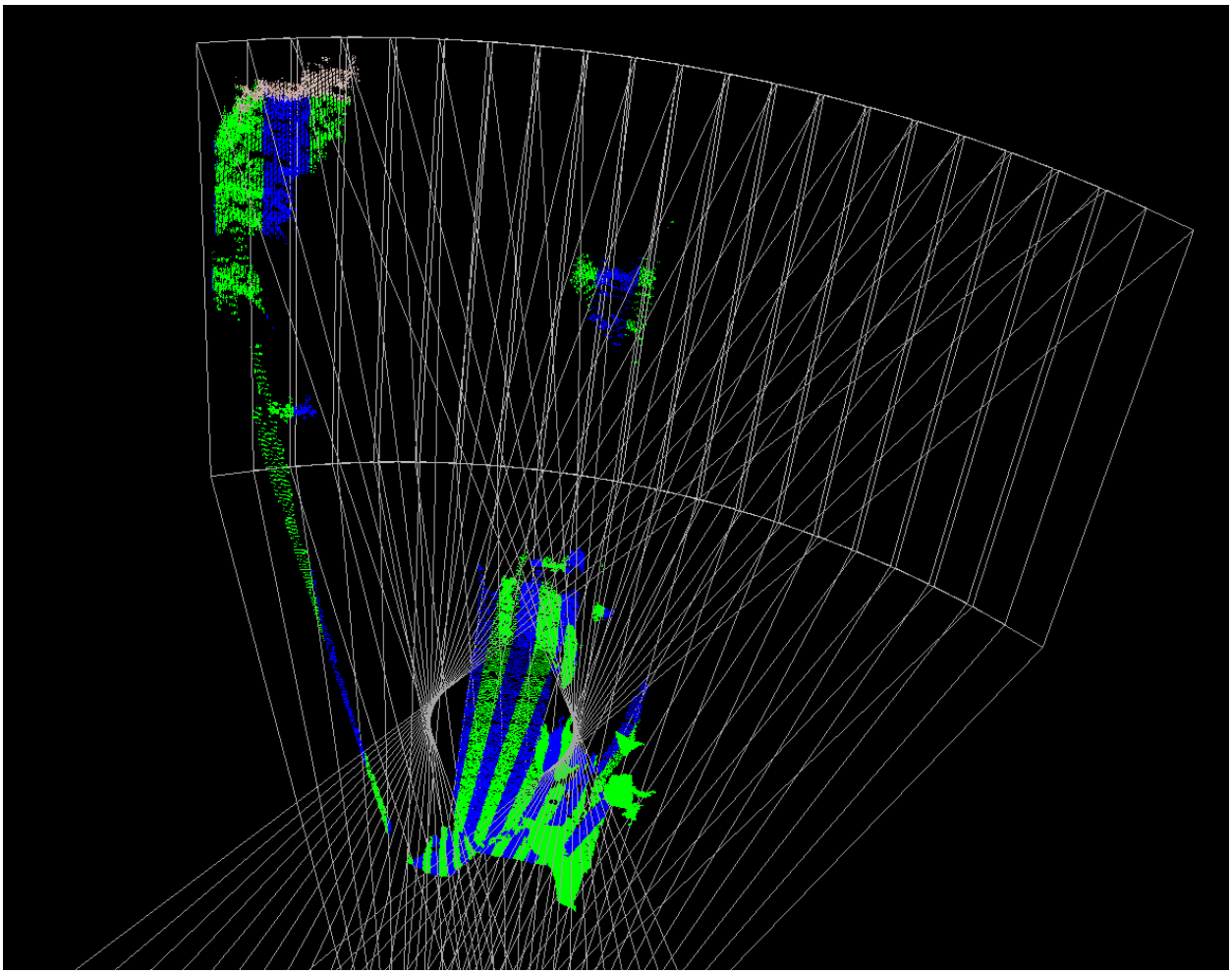

Fig 5: Checking our box rotation algorithm. Boxes are being drawn at each of the angles we wish to sample at. The point cloud is also being coloured alternately at each angle. The actual boxes are much bigger than they appear here (see fig 5a below) but look smaller because they are being overwritten by other boxes.

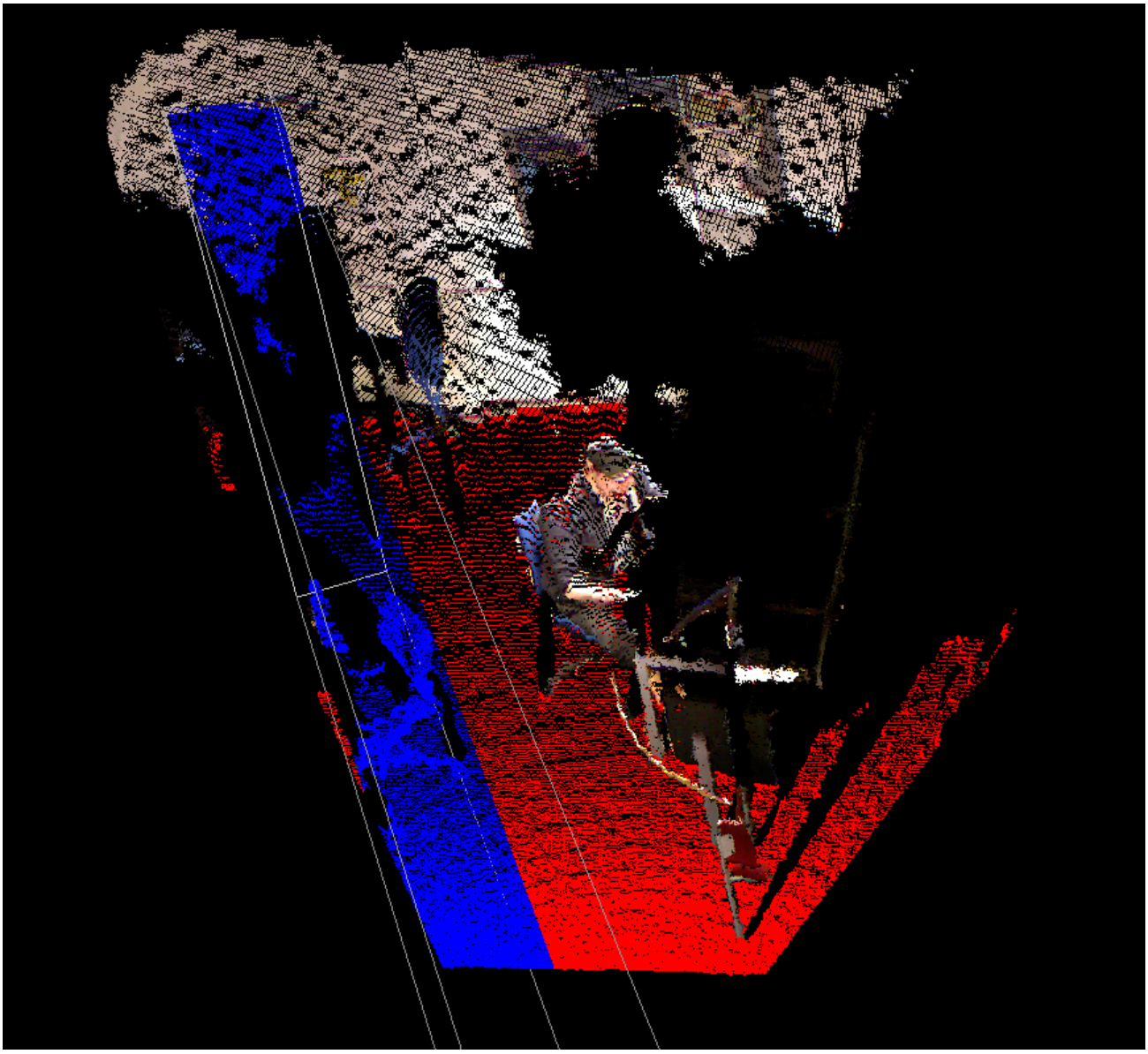

Fig 5a: Box image segmentation with only one box.

Michael - 26/05/2012

Managed to get a pretty good ‘best direction’ estimation using the ground plane! Basically the idea is that the number of points in the ground plane is a pretty good estimator of how suitable a surface is for driving on, since objects present will shadow the ground plane and therefore reduce the number of points. We therefore use the extracted ground plane, slice it into 0.5m (need to check this how units in the point cloud library correspond to the real world) strips projecting out from the camera (this took a long time to work out how to do) and choose the direction that has the most points. Some images can be seen below in Figs 5 and 6. There are some problems with this method however:

1. Objects like tables that do not nesc. shadow much of the ground plane are not nesc. taken into account
    - Proposed solution: Use the rest of the data without the ground plane to help

choose best direction - best direction will have balance of lots of ground points but not many above ground points.

2. There are more points closer to the kinect than there are further away, therefore the measure if biased towards close points. This may actually be a good thing but we should come up with a solution.
    - Proposed solution: Use the VoxelGrid Point Cloud Library filter to solve this.

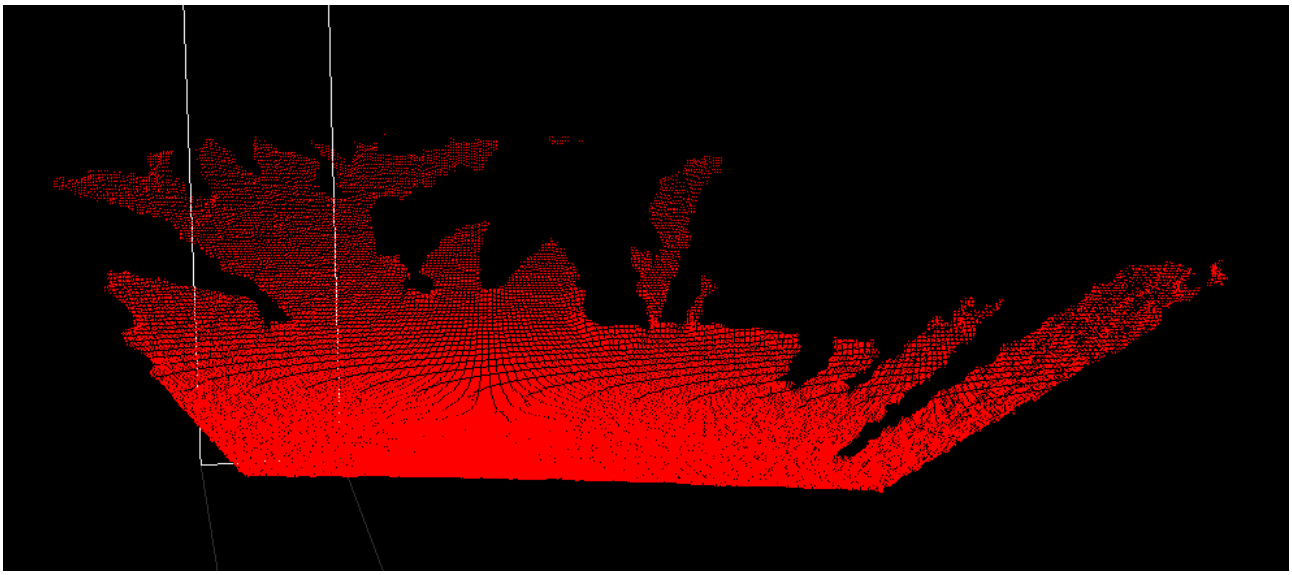

Fig 6: The extracted ground plane. The white cuboid shows the direction with most points.

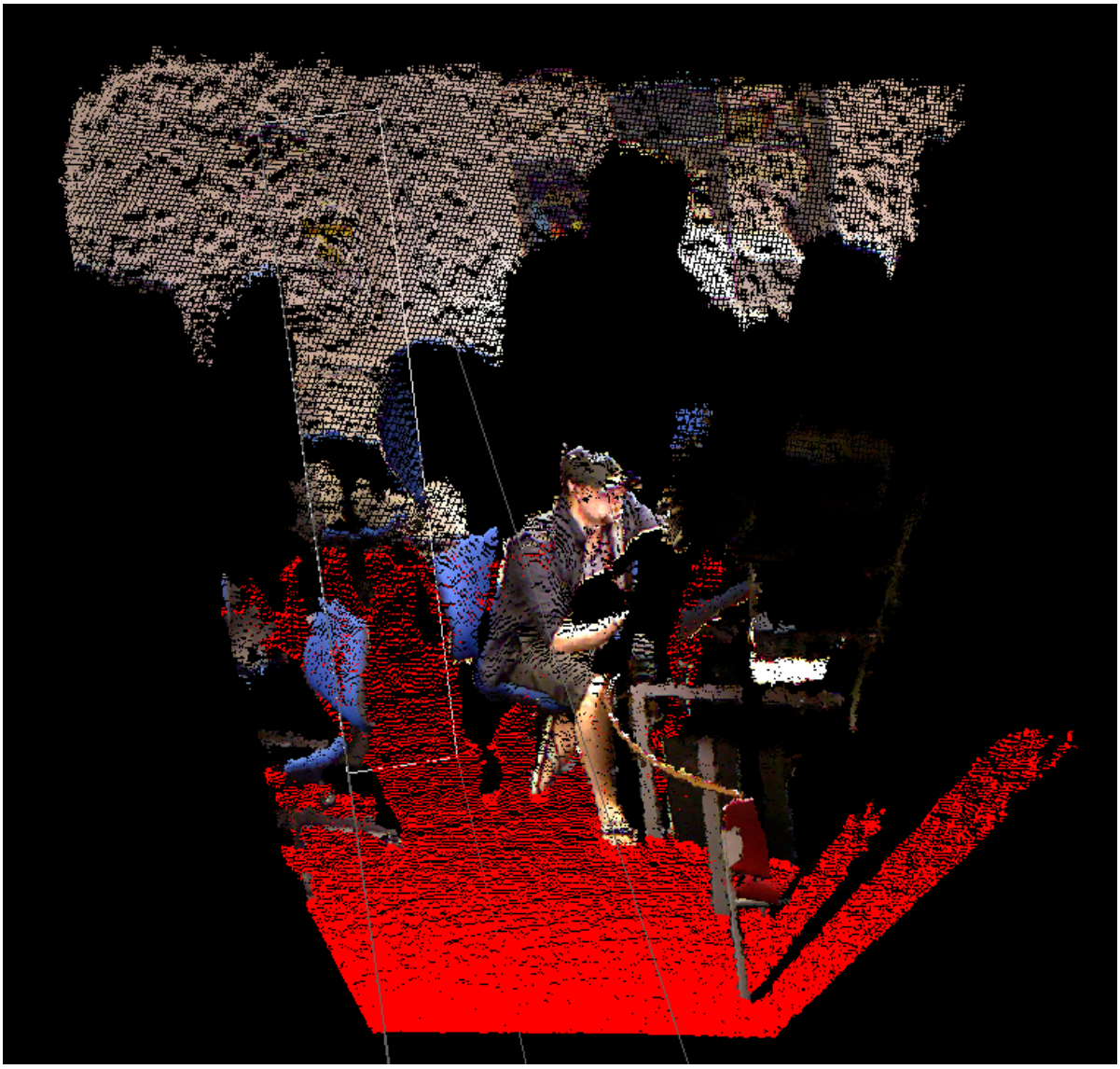

Fig 7: The whole point cloud with the ground plane coloured red. The white cuboid shows the direction where the most points lay.

UPDATE: We now pre-process the point cloud with the VoxelGrid filter to downsample the data and remove the problem where there are more points closer to the camera. The VoxelGrid filter creates a uniform point density across the image (assuming points are present in an area) which removes this problem. This produces much better detection results (see Fig 8) and seems to work very well! It has also had the side effect of greatly speeding up processing so it now runs much more quickly at about 7.8FPS. I am pretty sure there are some sound opportunities to speed up the program more for example where we copy point clouds/get point clouds our of filters rather than just the point indices - this is useful for debugging though and so we will keep it as it is now for the moment.

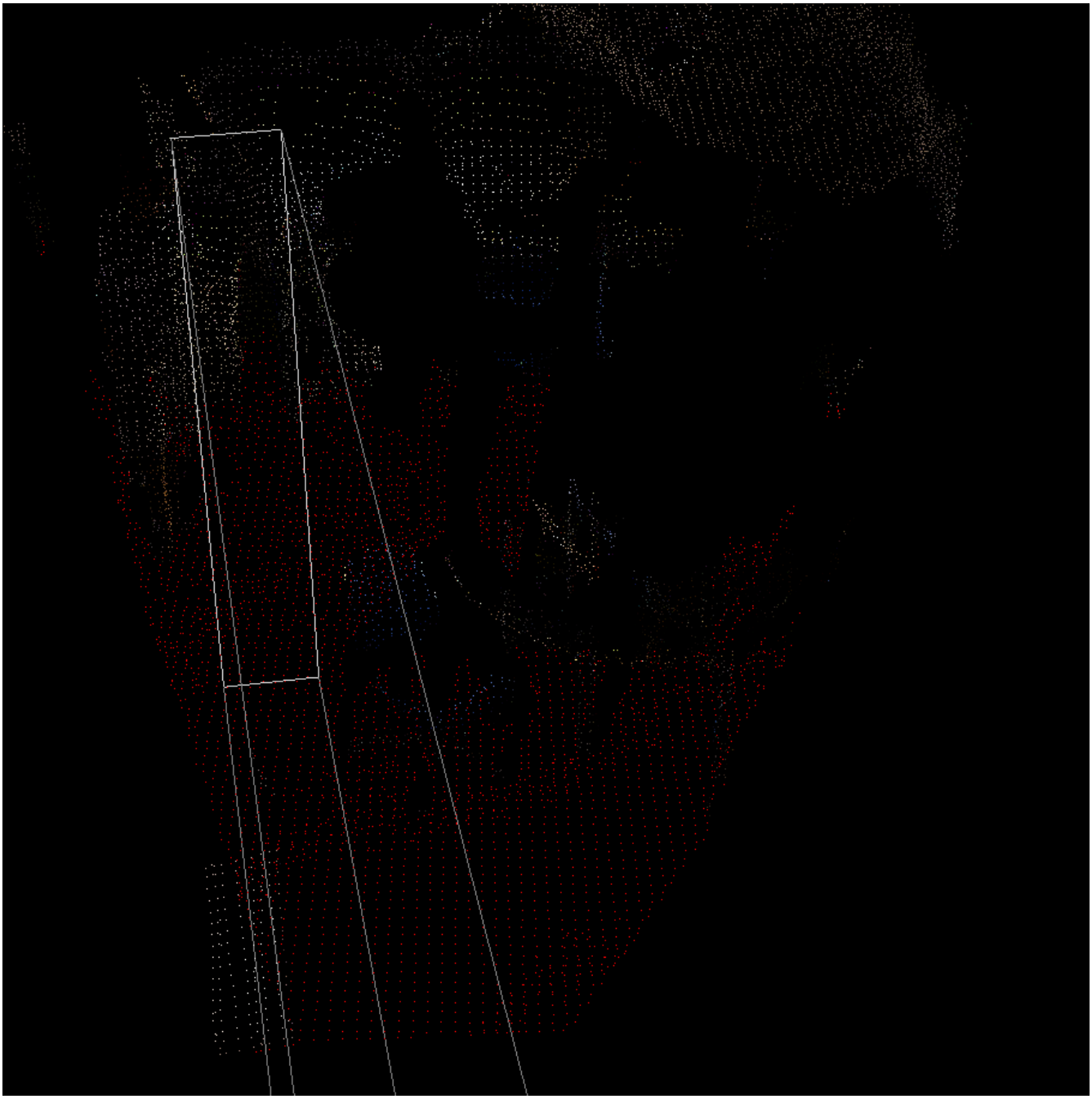

Fig 8: VoxelGrid downsampling preprocessing gives better detection results. Leaf size for the VoxelGrid filter is 0.05, 0.05, 0.05.

IDEA: We could try using an A* pathfinding algorithm like the one here: http://code.activestate.com/recipes/577457-a-star-shortest-path-algorithm/
Perhaps we could generate the map by using a VoxelGrid with a big y leaf size.

Michael (with John) - 28/05/2012

Got occupancy map working, although there are some odd weirdnesses with some directions possibly being dilated more than others.

Also have integrated A* algorithm code found at http://code.activestate.com/recipes/577457-a-star-shortest-path-algorithm/

We take into account the size of the wheelchair by growing the object boundaries by half the width of the wheelchair and shrinking the ground boundaries by the same amount.

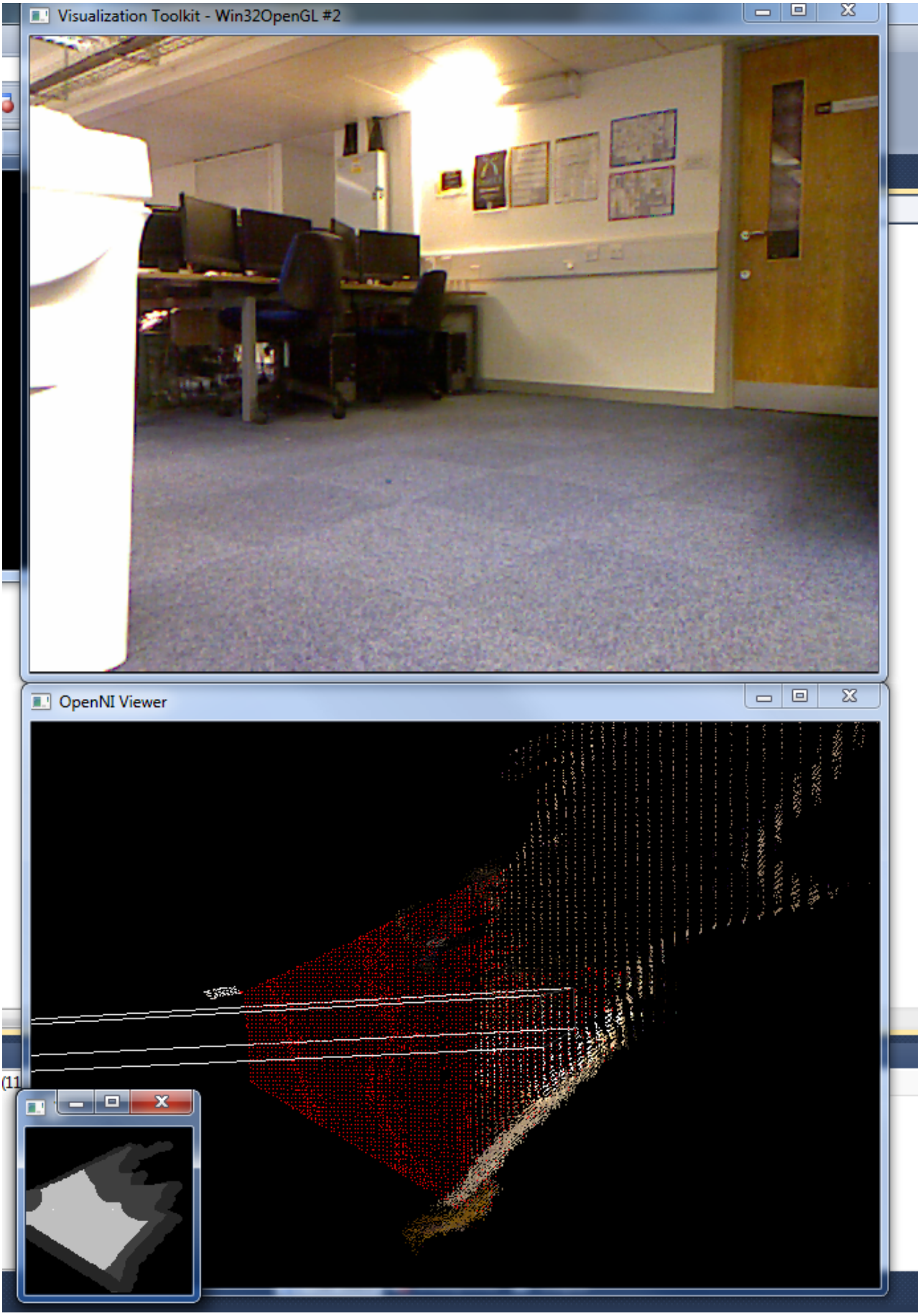
Visualization Toolkit - Win32OpenGL #2
OpenNI Viewer

Fig 9: 2D Occupancy map (bottom left of image above) generated using data from the point cloud. Points that have ground underneath them but no obstructions above the ground plane are classed as free for movement. All others are classed as obstructed.

Michael (with John) - 29/05/2012

We implemented threshold based detection of directions so multiple directions are now displayed rather than just the best one. The conditions for a direction being valid are reasonably simple: if a direction has over a set threshold of ground points as well as under a set threshold of object points (those above the ground plane) then the direction is considered passable.

We also implemented variable length detection where the detection box length and ground points threshold can be controlled by keyboard presses. We scale the detection thresholds and length of the detection box by the same variable, but found we needed to add an offset to the ground points threshold to take into account the fact that ground points do not begin until a reasonable distance away from the kinect (while the detection box does).

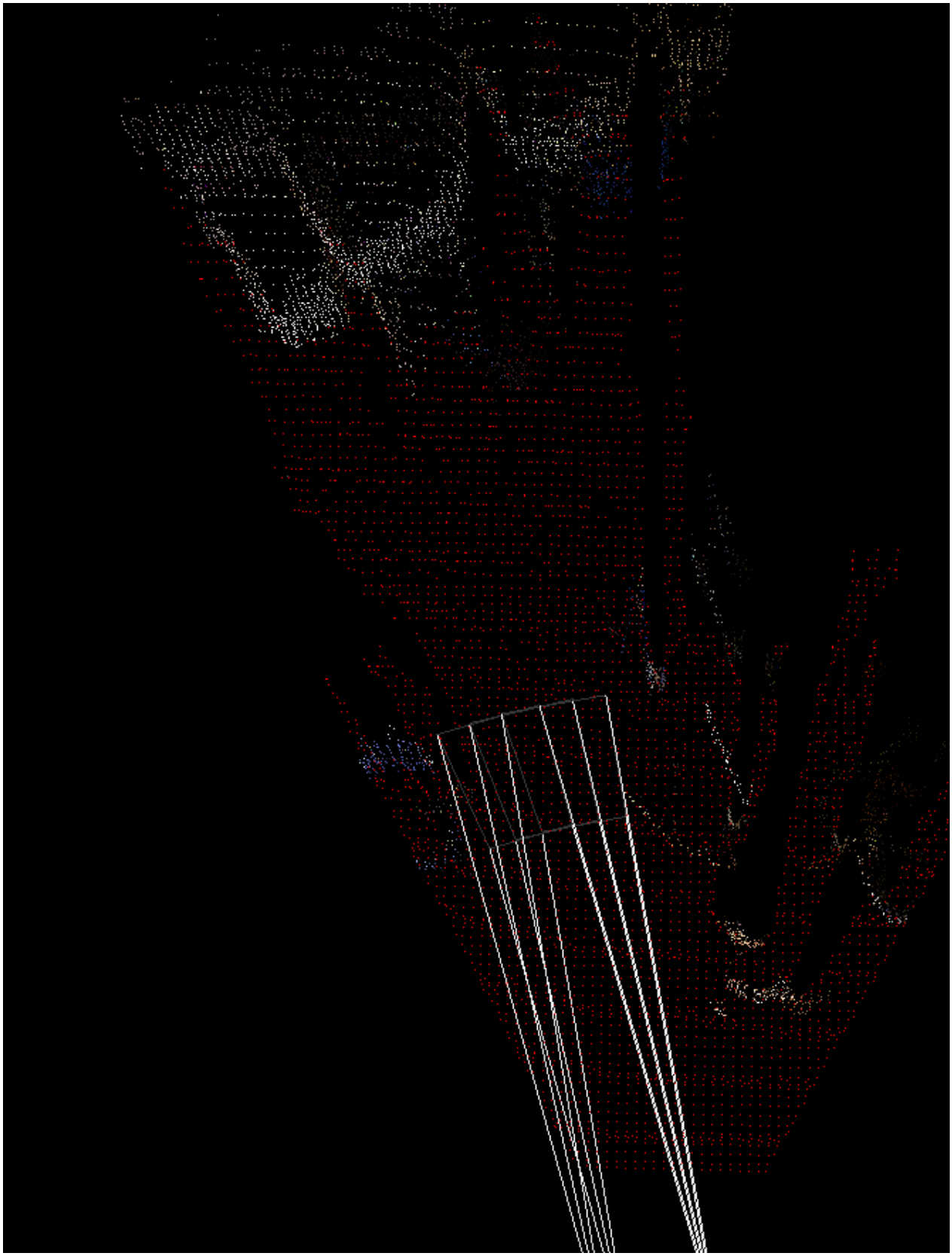

Fig 10: Thresholded detection which multiple directions shown as well as variable length detection. Pressing the up and down keys will increase or decrease the length of the

detection boxes respectively. The ground points threshold required to make a direction valid is scaled by the same amount the length of the detection box is scaled (plus an offset).

We have also implemented multiple direction pathfinding by setting several endpoints for the pathfinding algorithm in an arc a certain distance from the kinect.This distance can also be changed by keypress. However we did find the pathfinding approach was quite slow so it can be toggled on and off by a press of the Ctrl key. It is initially toggled off.

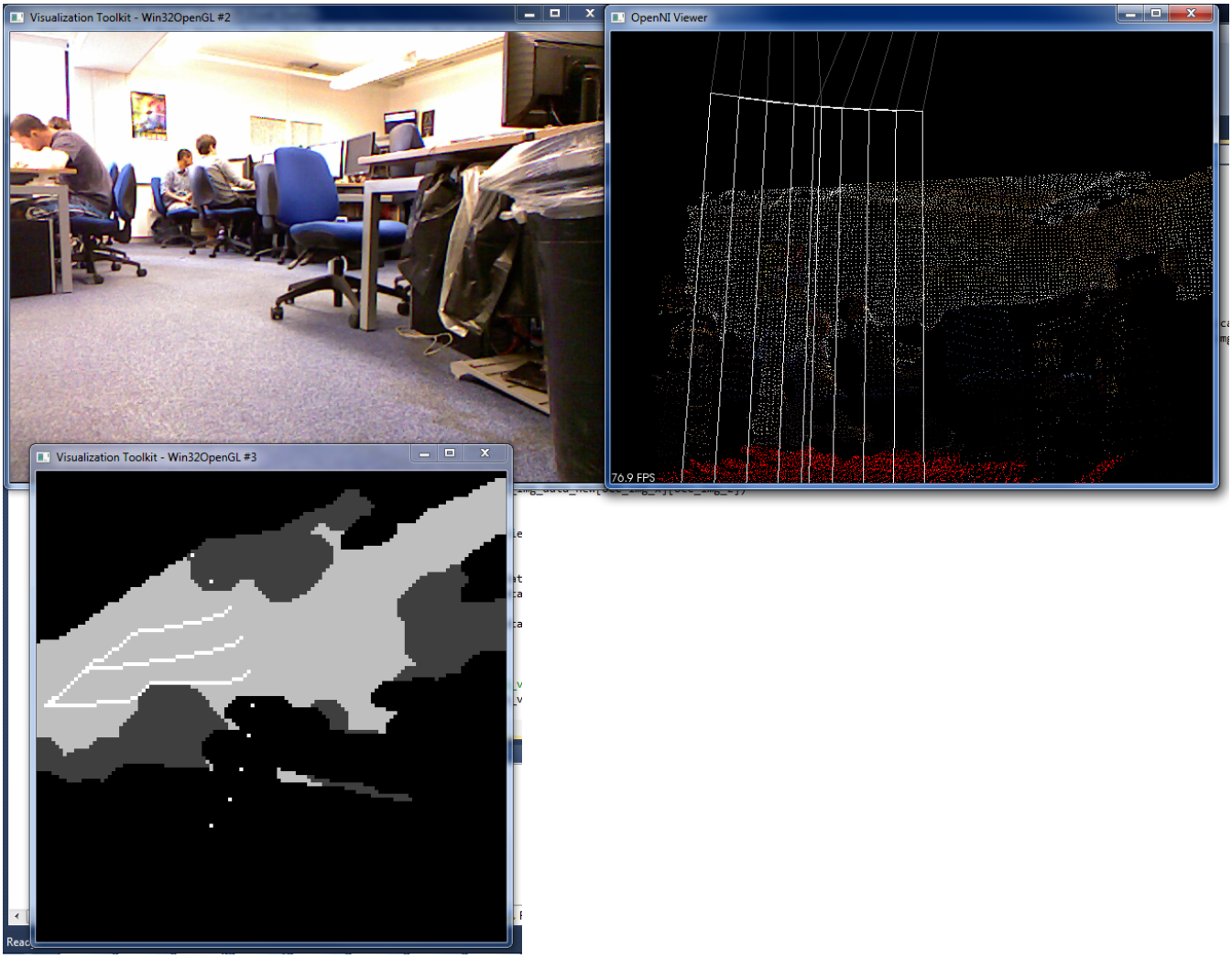

Fig 11: Multiple pathfinding using the occupancy map seen in the bottom left. Light grey represents space where the floor has been detected so therefore it is safe to move over and dark grey represents objects detected in the point cloud.

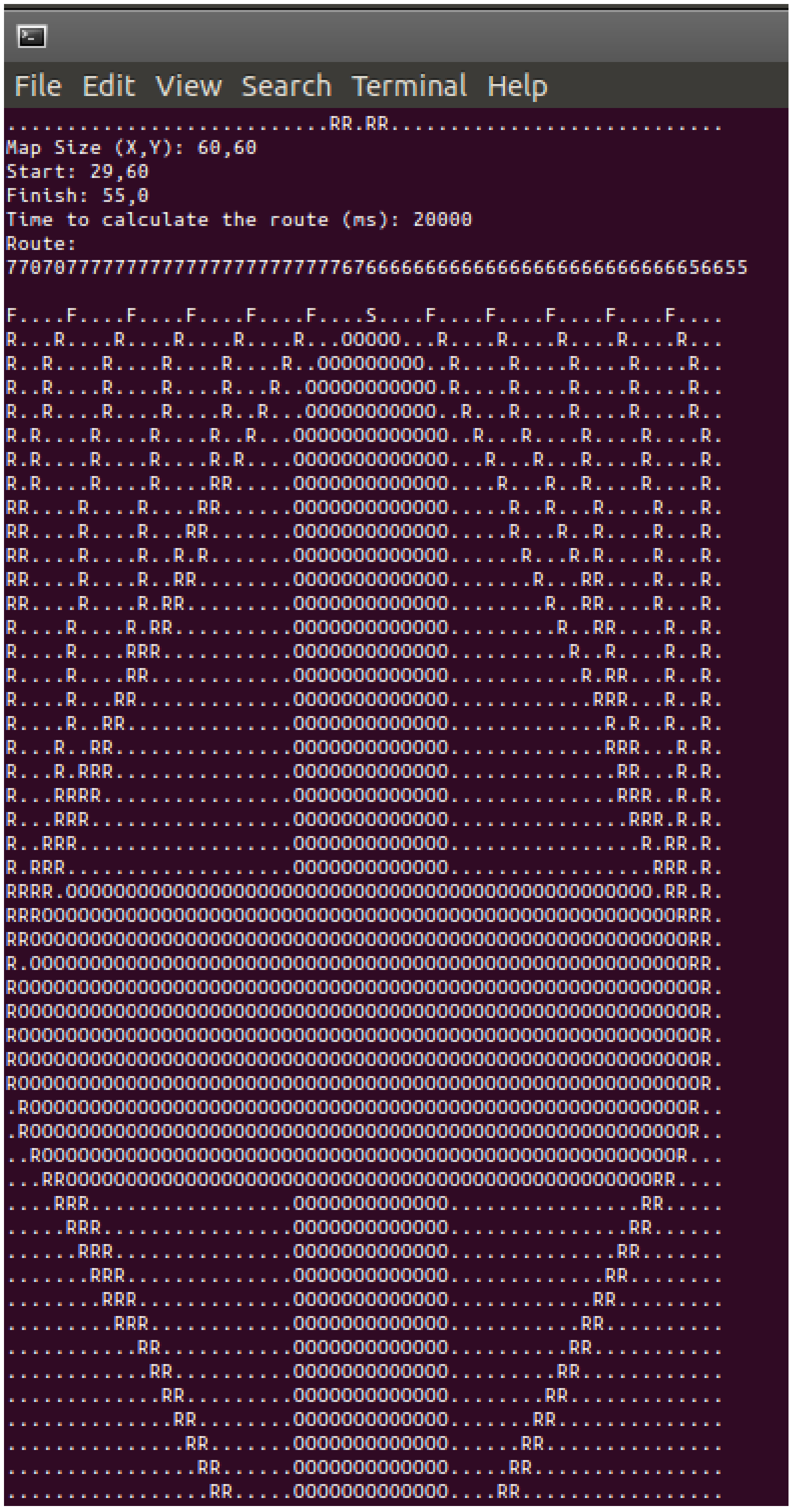

Fig 12: Testing of the pathfinder algorithm with multiple finish points. The console window shows a occupancy map and resulting routes. The '.' characters represent free space, the 'O' characters represent obstacles, the 'S' character is offscreen down the bottom of the map and represents represent the start of routes and the 'F' characters represent finish points while the 'R' characters represent the routes found.

Michael (with John) - 30/05/2012

Mostly doing presentation today, but (hopefully) managed to find and remove annoying runtime error where if the point cloud happened to be empty, the program would attempt to index points in it and cause an error. We solved by checking to see if the point clouds used are empty and if so then skipping the algorithm for that frame.

References

- http://www.araa.asn.au/acra/acra2010/papers/pap151s1-file1.pdf Talks about navigating using an occupancy grid and point cloud
- http://www.ipb.uni-bonn.de/uploads/tx_ikgpublication/Plane_Detection_in_Point_Cloud_Data.pdf :
    - A couple of plane detection algorithms for point clouds
- We use the Point Cloud Library (PCL) http://pointclouds.org/ for a lot of things in our algorithms including voxel grid downsampling, plane detection using RANSAC and visualisation of point cloud data - this is an excellent library with a huge number of features, authough it is stillquite rough around the edges, inconsistant and difficult to use at times.
- Our implementation has been based on the file found here: http://svn.pointclouds.org/pcl/trunk/visualization/tools/openni_viewer.cpp which extracts point cloud data and image data from a kinect and displays it on screen
- Our A* pathfinding algorithm has been copied from here: http://code.activestate.com/recipes/577457-a-star-shortest-path-algorithm/