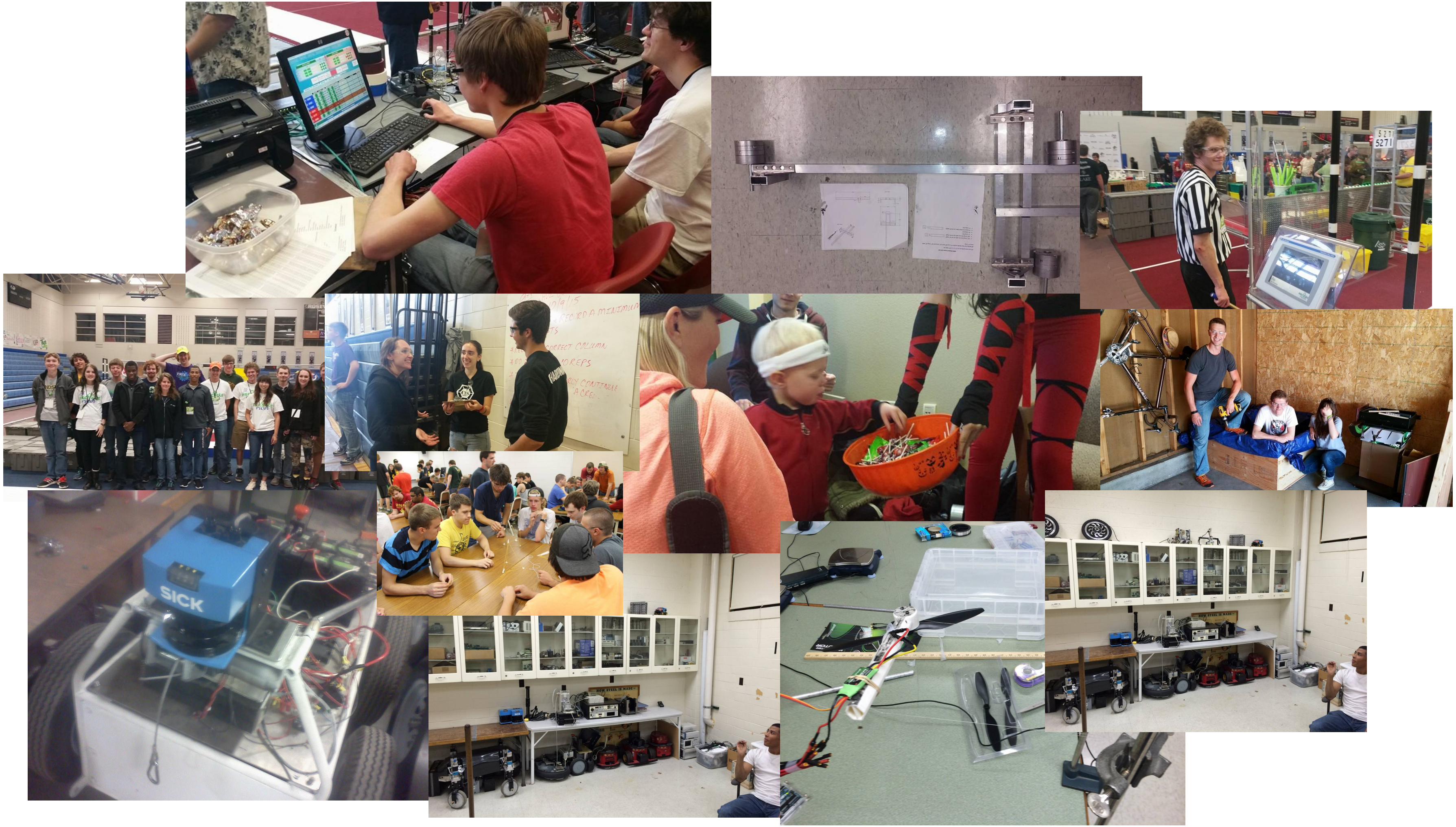
Converting 3D Point Cloud Data into 2D Occupancy Grids suitable for Robot Applications

Jacob Huesman

BISON ROBOTICS


FIRST
FRC
FRC
EXIT


SICK

?

NASA

Design it.

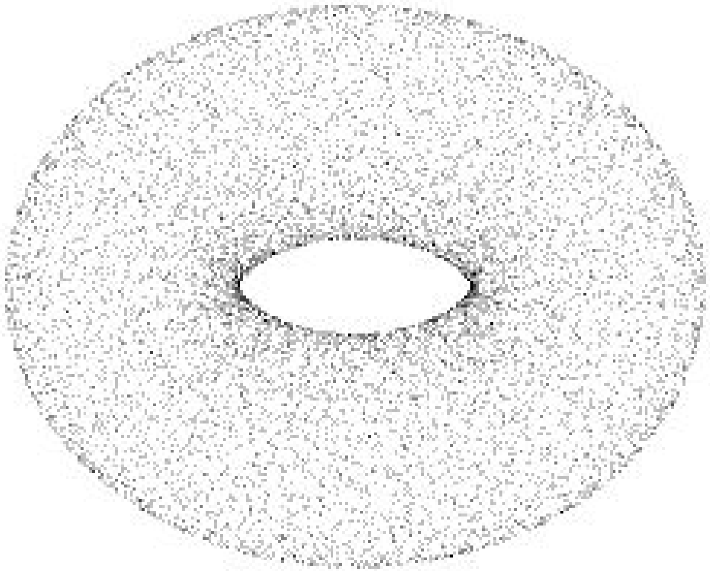
Build it.

Dig it.

ROBOTIC MINING COMPETITION

ROS

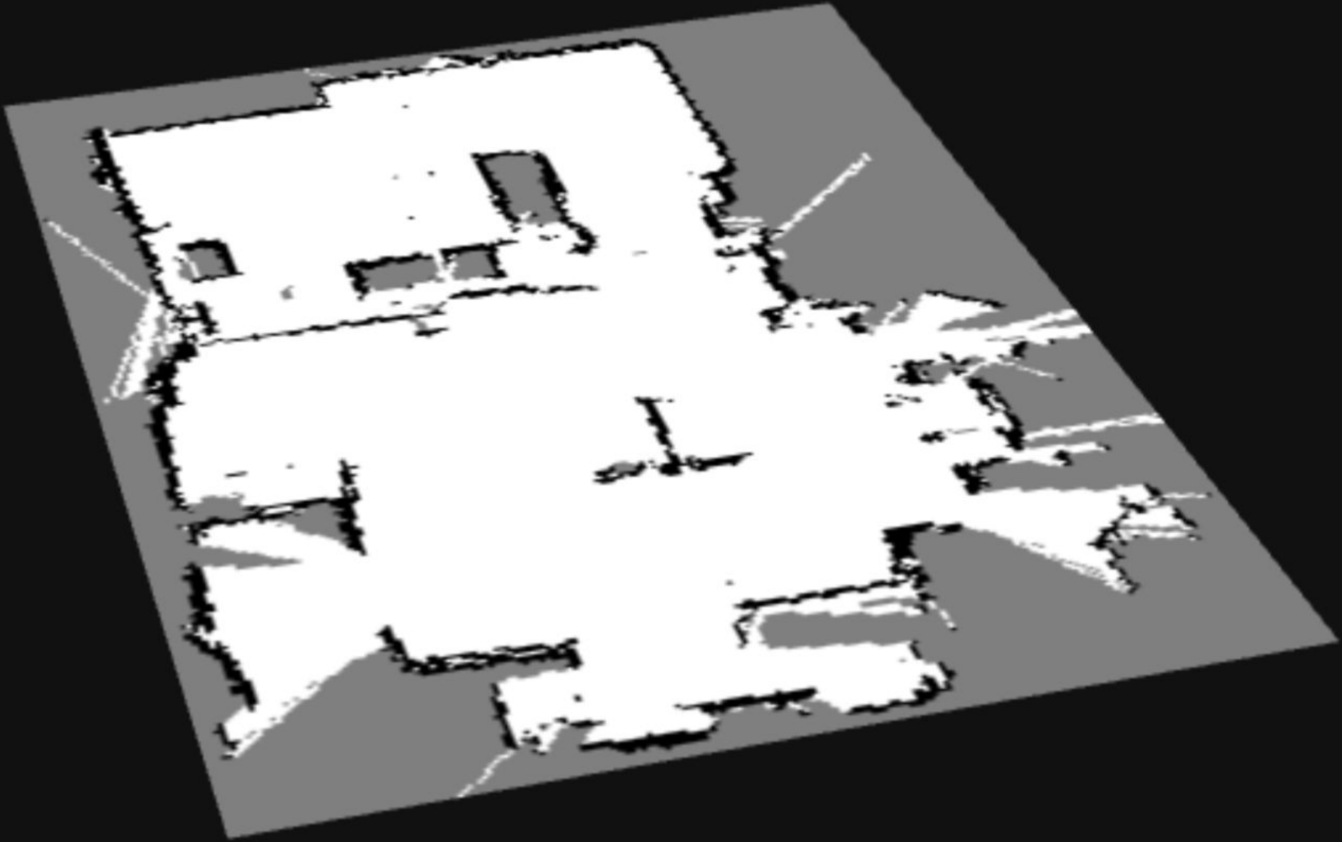

Willow Garage
Personal Robots
PR2
ROBOT POWER
Willow Garage
Delivering Open Source Robotics
OpenCV: Computer Vision Library
ROS: Robot Operating System



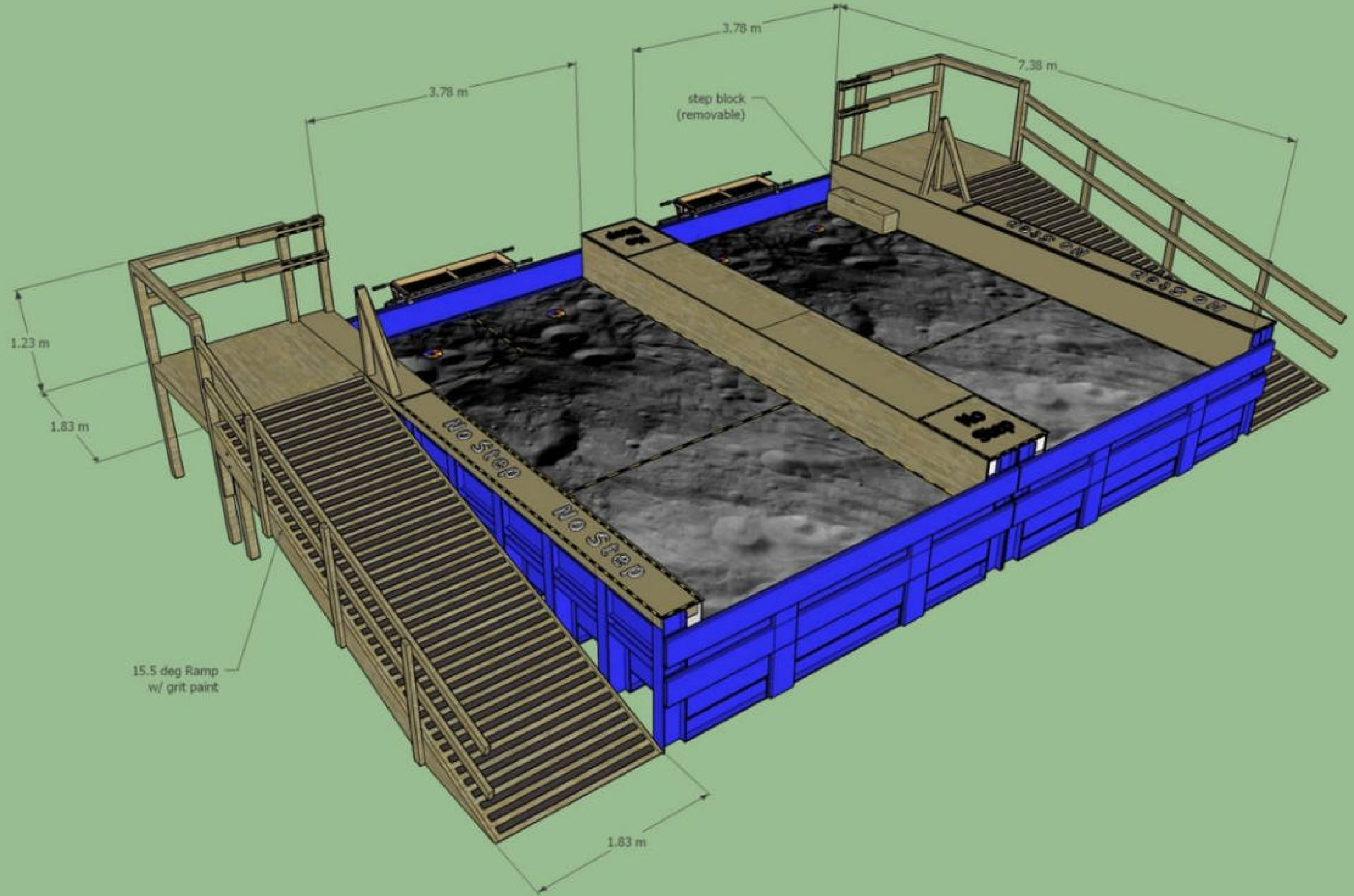

ASUS


SICK

?

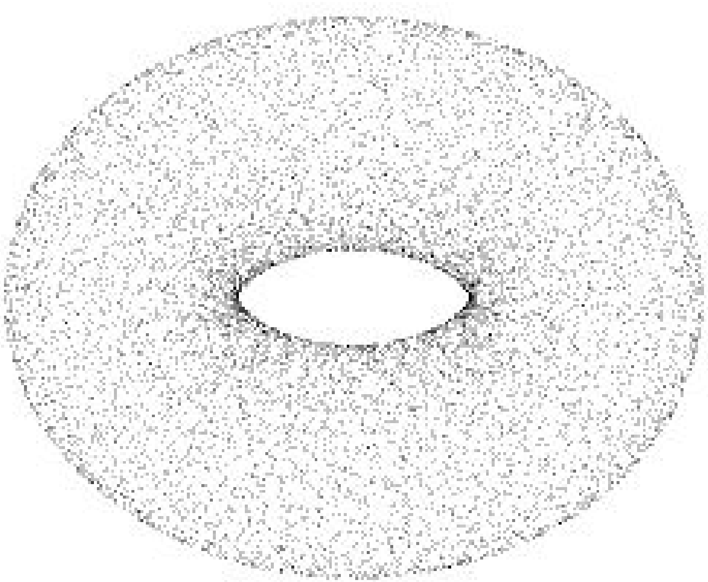


?

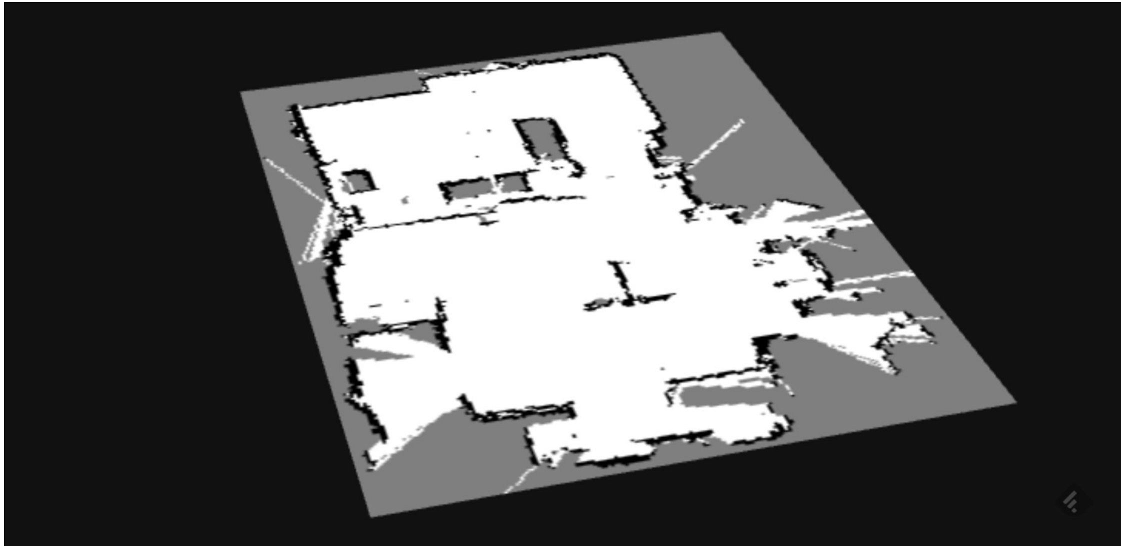

3.78 m
3.78 m
7.38 m
step block
(removable)
1.23 m
1.83 m
No Step
No Step
No Step
No Step
No Step
15.5 deg Ramp
w/ grit paint
1.83 m

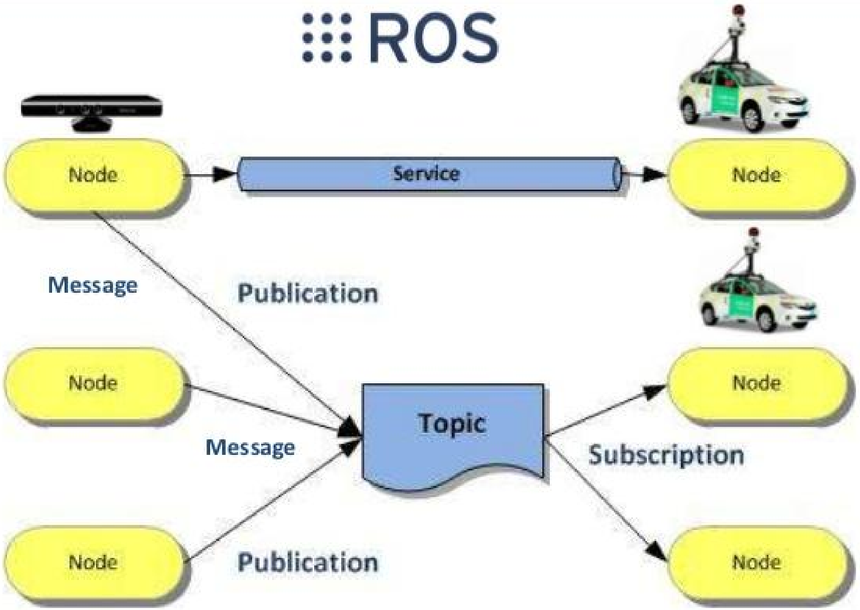
NORTH DAKOTA

SPACE GRANT CONSORTIUM




ASUS




ROS
Node
Service
Node
Message
Publication
Node
Topic
Node
Message
Subscription
Node
Publication
Node

55.74190000 77.90000000 4.15650000 246 246 246

55.74190000 77.90000000 4.15610000 246 246 246

55.74190000 77.90000000 4.15570000 246 246 246

55.74190000 77.90000000 4.15580000 245 245 245

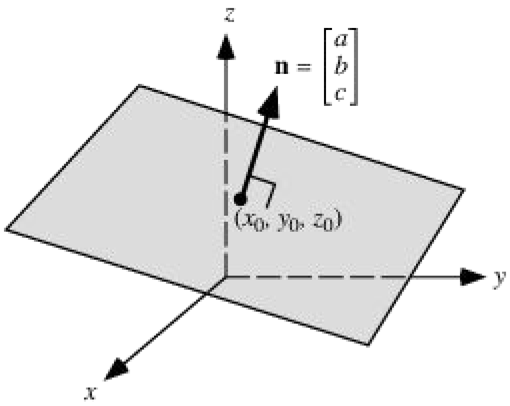
55.74190000 77.90000000 4.15580000 246 246 246

55.74190000 77.90000000 4.15590000 246 246 246

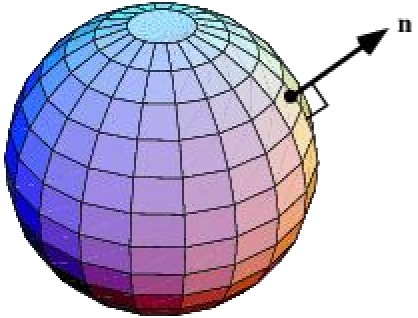
55.74190000 77.90000000 4.15550000 245 245 245

55.74190000 77.89990000 4.15580000 245 245 245

1	1	1	1	1	1	1
0	0	0	0	0	0	1
0	0	0	0	0	0	1
0	0	0	0	0	0	1
0	0	0	0	0	0	1
0	0	0	0	0	0	1
0	0	0	0	0	0	1



z
n = [a b c]
(x0, y0, z0)
y
x



n

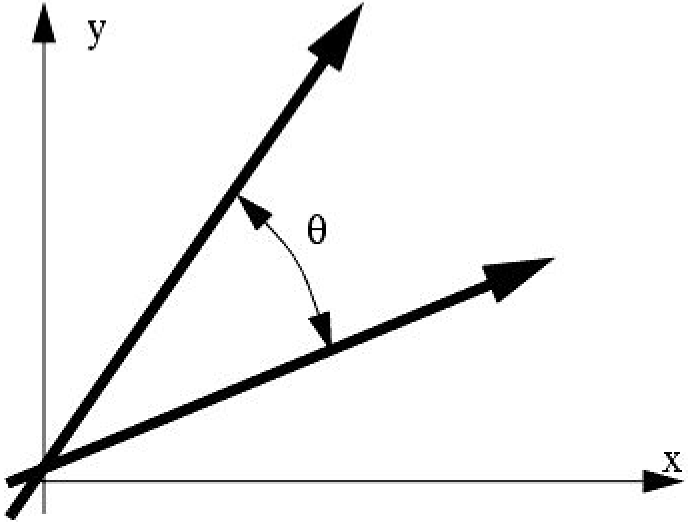
$$\vec{g} \cdot \vec{n} = |\vec{g}||\vec{n}| cos\theta$$

$$\theta = \arccos\left(\frac{\vec{g} \cdot \vec{n}}{|\vec{g}||\vec{n}|}\right)$$

$$\theta = \arccos(\vec{g} \cdot \vec{n})$$

$$\theta = \arccos\left(g_x * n_x + g_y * n_y + g_z * n_z\right)$$

$$\theta = \arccos(n_z)$$


y
θ
x

cloud_to_map

```
// ------------------------------------------------------
// -----Update current PointCloud if msg is received-----
// ------------------------------------------------------
void callback(const PointCloud::ConstPtr& msg) {
  boost::unique_lock<boost::mutex>(mutex);
  currentPC = msg;
  newPointCloud = true;
}
```

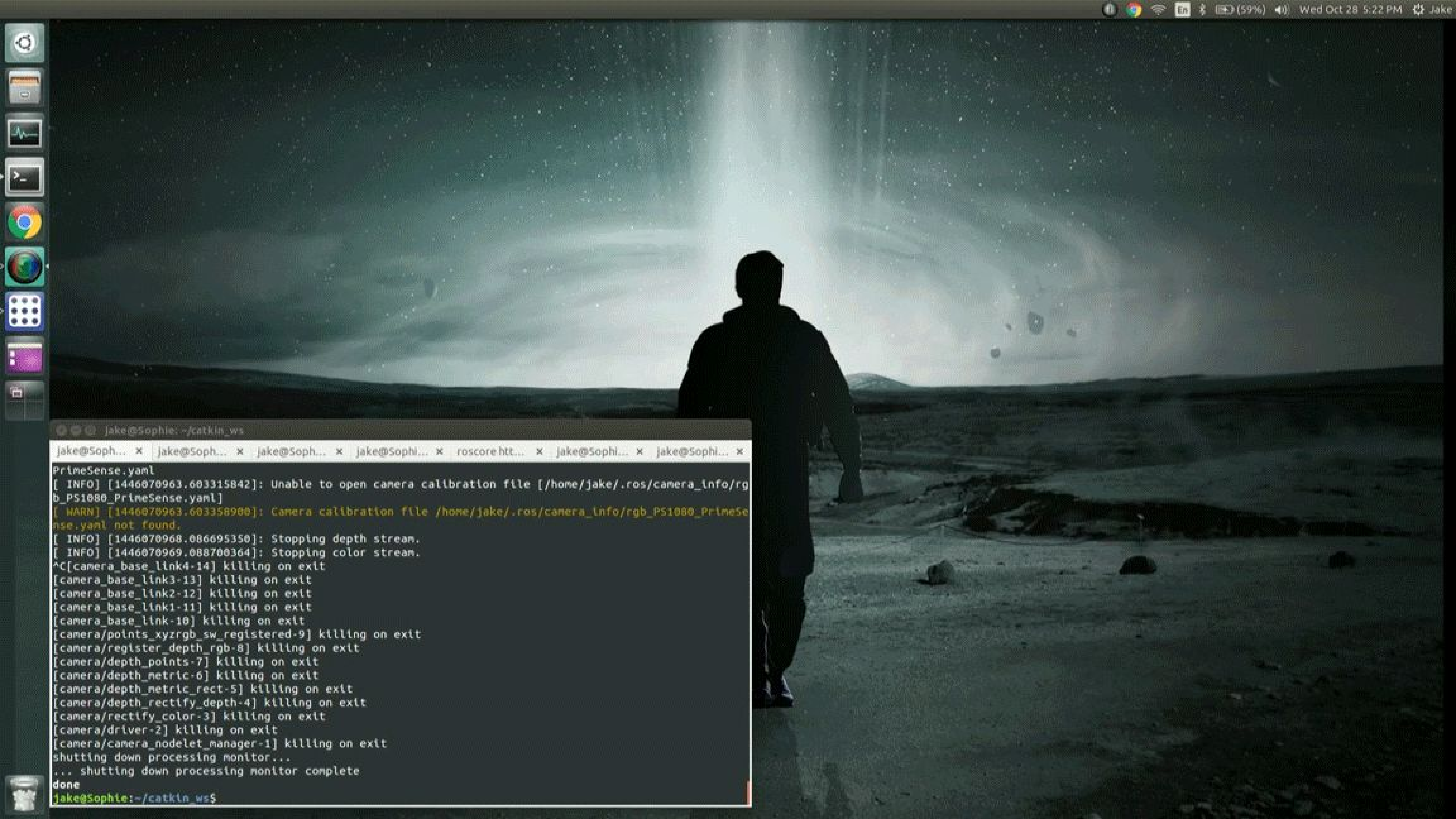
```
// ----------------------------------------------------------------
// -----Calculate surface normals with a search radius of 0.05-----
// ----------------------------------------------------------------
void calcSurfaceNormals(PointCloud::ConstPtr& cloud, pcl::PointCloud<pcl::Normal>::Ptr normals) {
  pcl::NormalEstimation<pcl::PointXYZRGB, pcl::Normal> ne;
  ne.setInputCloud(cloud);
  pcl::search::KdTree<pcl::PointXYZRGB>::Ptr tree(new pcl::search::KdTree<pcl::PointXYZRGB>());
  ne.setSearchMethod(tree);
  ne.setRadiusSearch(param.searchRadius);
  ne.compute(*normals);
}
```

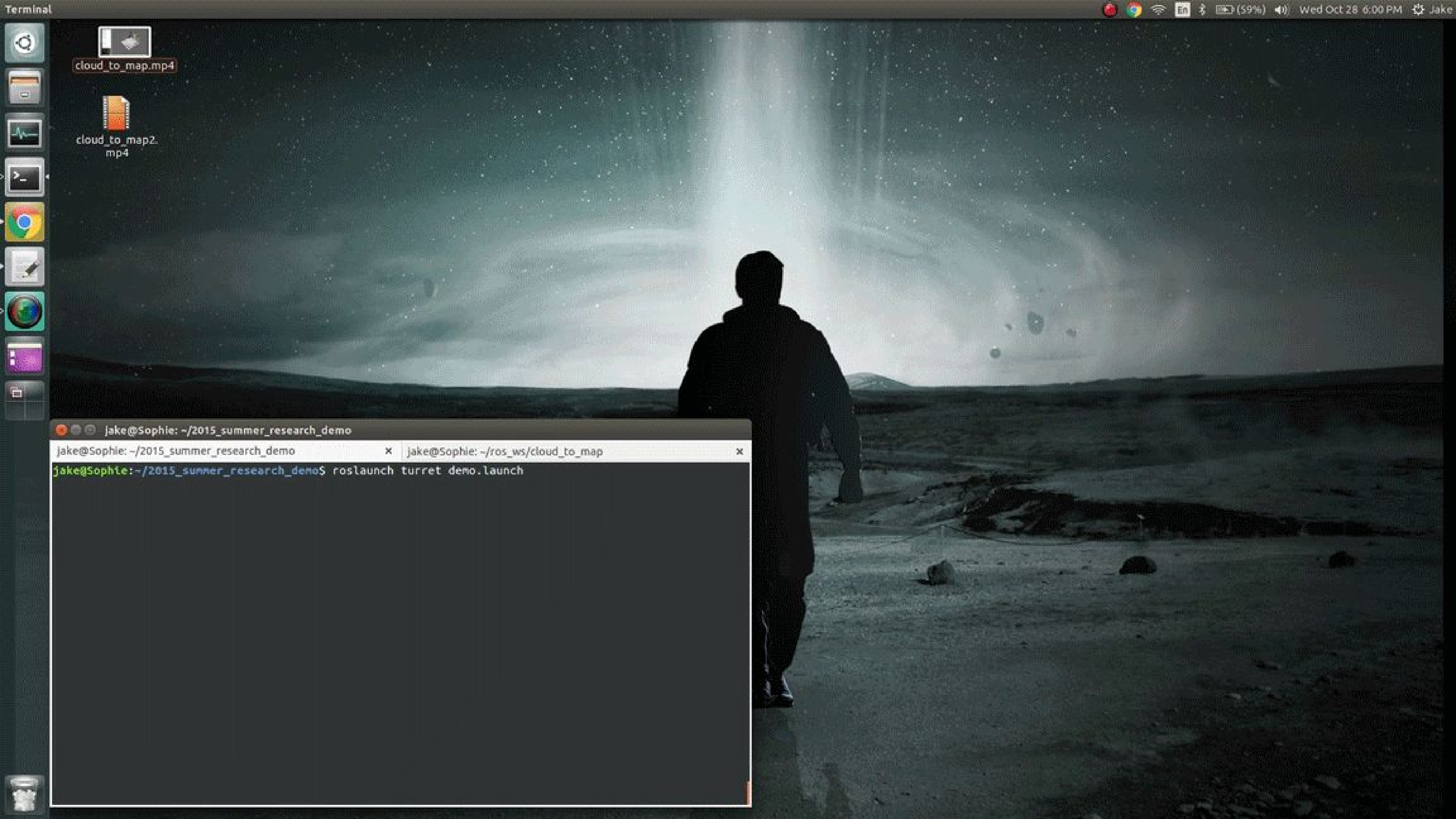
```
// ------------------------------------------
// -----Calculate size of Occupancy Grid-----
// ------------------------------------------
void calcSize(double *xMax, double *yMax, double *xMin, double *yMin) {
  for (size_t i = 0; i < currentPC->size(); i++) {
    double x = currentPC->points[i].x;
    double y = currentPC->points[i].y;
    if (*xMax < x) {
      *xMax = x;
    }
    if (*xMin > x) {
      *xMin = x;
    }
    if (*yMax < y) {
      *yMax = y;
    }
    if (*yMin > y) {
      *yMin = y;
    }
  }
}
```

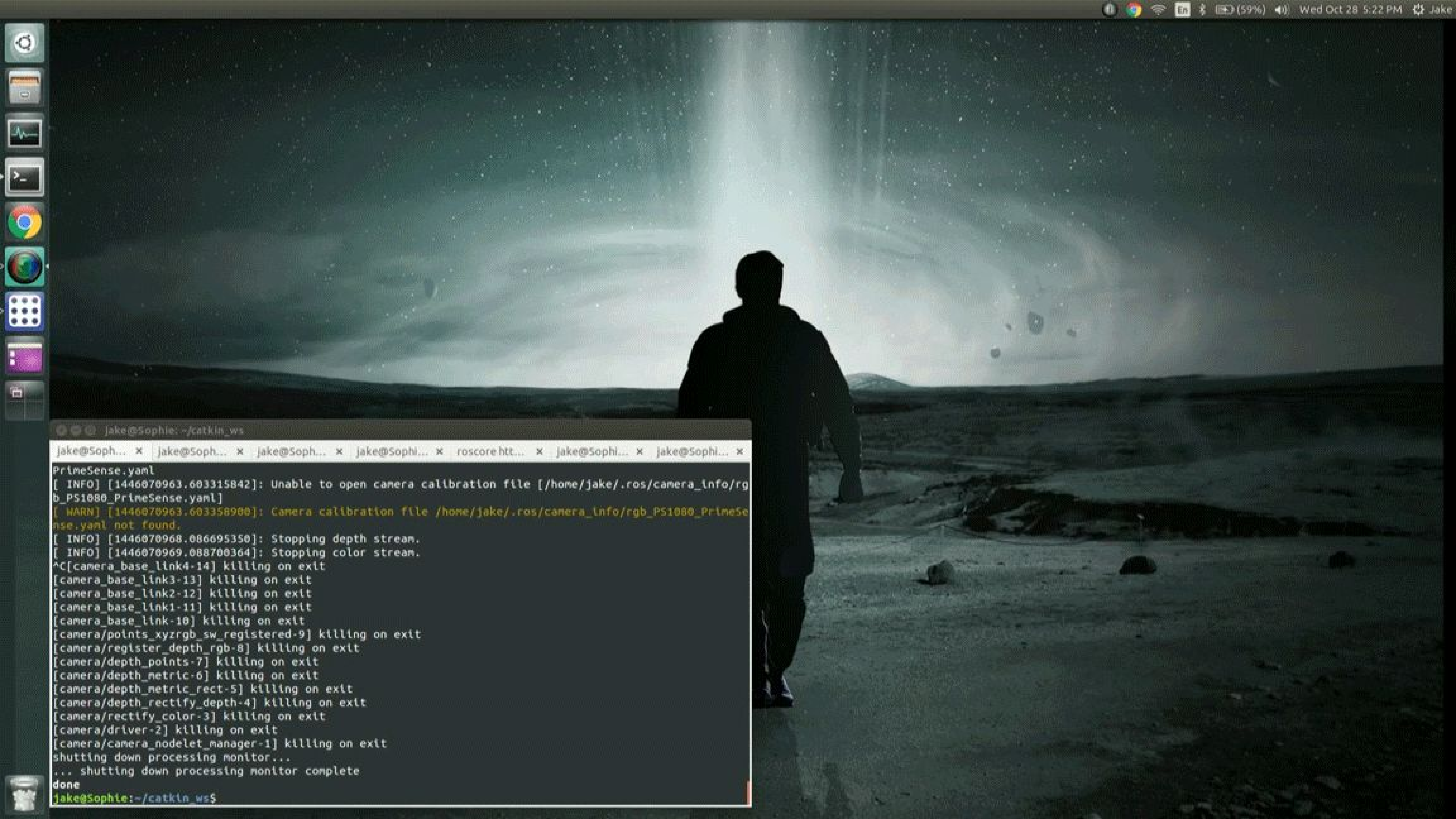
```cpp
// ---------------------------------------
// -----Populate grid with cost values-----
// ---------------------------------------
void populateMap(NormalCloud::Ptr cloud_normals, std::vector<int> &map, double xMin, double yMin,
    double cellResolution, int xCells, int yCells) {
  double deviation = param.deviation;

  for (size_t i = 0; i < currentPC->size(); i++) {
    double x = currentPC->points[i].x;
    double y = currentPC->points[i].y;
    double z = cloud_normals->points[i].normal_z;

    double phi = acos(fabs(z));
    int xCell, yCell;

    if (z == z) {
      xCell = (int) ((x - xMin) / cellResolution);
      yCell = (int) ((y - yMin) / cellResolution);
      if ((yCell * xCells + xCell) > (xCells * yCells)) {
        std::cout << "x: " << x << ", y: " << y << ", xCell: " << xCell << ", yCell: " << yCell
            << "\n";
      }
      if (phi > deviation) {
        map[yCell * xCells + xCell]++;
      } else {
        map[yCell * xCells + xCell]--;
      }
    }
  }
}
```

```cpp
// ----------------------------------
// -----Generate Occupancy Grid-----
// ----------------------------------
void genOccupancyGrid(std::vector<signed char> &ocGrid, std::vector<int> &countGrid, int size) {
  int buf = param.buffer;
  for (int i = 0; i < size; i++) {
    if (countGrid[i] < buf) {
      ocGrid[i] = 0;
    } else if (countGrid[i] > buf) {
      ocGrid[i] = 100;
    } else if (countGrid[i] == 0) {
      ocGrid[i] = 0;
    }
  }
}
```

```cpp
// ------------------------------------------
// -----Initialize Occupancy Grid Msg-----
// ------------------------------------------
void initGrid(nav_msgs::OccupancyGridPtr grid) {
  grid->header.seq = 1;
  grid->header.frame_id = param.frame;
  grid->info.origin.position.z = 0;
  grid->info.origin.orientation.w = 1;
  grid->info.origin.orientation.x = 0;
  grid->info.origin.orientation.y = 0;
  grid->info.origin.orientation.z = 0;
}

// ------------------------------------
// -----Update Occupancy Grid Msg-----
// ------------------------------------
void updateGrid(nav_msgs::OccupancyGridPtr grid, double cellRes, int xCells, int yCells,
    double originX, double originY, std::vector<signed char> *ocGrid) {
  grid->header.seq++;
  grid->header.stamp.sec = ros::Time::now().sec;
  grid->header.stamp.nsec = ros::Time::now().nsec;
  grid->info.map_load_time = ros::Time::now();
  grid->info.resolution = cellRes;
  grid->info.width = xCells;
  grid->info.height = yCells;
  grid->info.origin.position.x = originX;
  grid->info.origin.position.y = originY;
  grid->data = *ocGrid;
}
```


Wed Oct 28 5:22 PM
Jake
jake@Sophie: ~/catkin_ws
jake@Soph...
jake@Soph...
jake@Soph...
jake@Sophi...
roscore htt...
jake@Sophi...
jake@Sophi...
PrimeSense.yaml
[INFO] [1446070963.603315842]: Unable to open camera calibration file [/home/jake/.ros/camera_info/rgb_PS1080_PrimeSense.yaml]
[WARN] [1446070963.603358900]: Camera calibration file /home/jake/.ros/camera_info/rgb_PS1080_PrimeSense.yaml not found.
[INFO] [1446070968.086695350]: Stopping depth stream.
[INFO] [1446070969.088700364]: Stopping color stream.
^C[camera_base_link4-14] killing on exit
[camera_base_link3-13] killing on exit
[camera_base_link2-12] killing on exit
[camera_base_link1-11] killing on exit
[camera_base_link-10] killing on exit
[camera/points_xyzrgb_sw_registered-9] killing on exit
[camera/register_depth_rgb-8] killing on exit
[camera/depth_points-7] killing on exit
[camera/depth_metric-6] killing on exit
[camera/depth_metric_rect-5] killing on exit
[camera/depth_rectify_depth-4] killing on exit
[camera/rectify_color-3] killing on exit
[camera/driver-2] killing on exit
[camera/camera_nodelet_manager-1] killing on exit
shutting down processing monitor...
... shutting down processing monitor complete
done
jake@Sophie:~/catkin_ws$


Terminal
Wed Oct 28 6:00 PM
Jake
cloud_to_map.mp4
cloud_to_map2.
mp4
jake@Sophie: ~/2015_summer_research_demo
jake@Sophie: ~/2015_summer_research_demo
jake@Sophie: ~/ros_ws/cloud_to_map
jake@Sophie:~/2015_summer_research_demo$ roslaunch turret demo.launch


En
(59%)
Wed Oct 28 5:22 PM
Jake
jake@Sophie: ~/catkin_ws
jake@Soph... jake@Soph... jake@Soph... jake@Sophi... roscore htt... jake@Sophi... jake@Sophi...
PrimeSense.yaml
[INFO] [1446070963.603315842]: Unable to open camera calibration file [/home/jake/.ros/camera_info/rgb_PS1080_PrimeSense.yaml]
[WARN] [1446070963.603358900]: Camera calibration file /home/jake/.ros/camera_info/rgb_PS1080_PrimeSense.yaml not found.
[INFO] [1446070968.086695350]: Stopping depth stream.
[INFO] [1446070969.088700364]: Stopping color stream.
^C[camera_base_link4-14] killing on exit
[camera_base_link3-13] killing on exit
[camera_base_link2-12] killing on exit
[camera_base_link1-11] killing on exit
[camera_base_link-10] killing on exit
[camera/points_xyzrgb_sw_registered-9] killing on exit
[camera/register_depth_rgb-8] killing on exit
[camera/depth_points-7] killing on exit
[camera/depth_metric-6] killing on exit
[camera/depth_metric_rect-5] killing on exit
[camera/depth_rectify_depth-4] killing on exit
[camera/rectify_color-3] killing on exit
[camera/driver-2] killing on exit
[camera/camera_nodelet_manager-1] killing on exit
shutting down processing monitor...
... shutting down processing monitor complete
done
jake@Sophie:~/catkin_ws$

?