Colored Point-Cloud Processing

Wyatt Newman
November, 2015

Some 3-D sensors, such as the Microsoft “Kinect”, provide 6-D point-cloud data, including x,y,z coordinates and r,g,b color values. That is, each 3-D point in the point cloud has an associated color. A PCL point-cloud object (e.g. called “colored_cloud”) of this type may be instantiated as:

```
pcl::PointCloud<pcl::PointXYZRGB>  colored_cloud
```

Color can be useful for helping to identify objects—particularly if they are similarly shaped but differ significantly by color.

In a colored pointCloud, color components can be accessed via r, g and b fields. For example, the color components of the i'th point in colored_cloud may be printed out as:

```
std::cout << " " << (int) colored_cloud.points[i].r
      << " "    << (int) colored_cloud.points[i].g
      << " "    << (int) colored_cloud.points[i].b << std::endl;
```

The datatype of a color component is an 8-bit, unsigned char, which is converted to an int for printing via the cout function.

Additional functions have been added to the cwru_pcl_utils package and illustrated with the example program “cwru_pcl_utils_color_test.cpp.” The new functions are described below.

Finding points by height and radius: The following function:

```
void filter_cloud_z(double z_nom, double z_eps,
        double radius, Eigen::Vector3f centroid, vector<int> &indices);
```

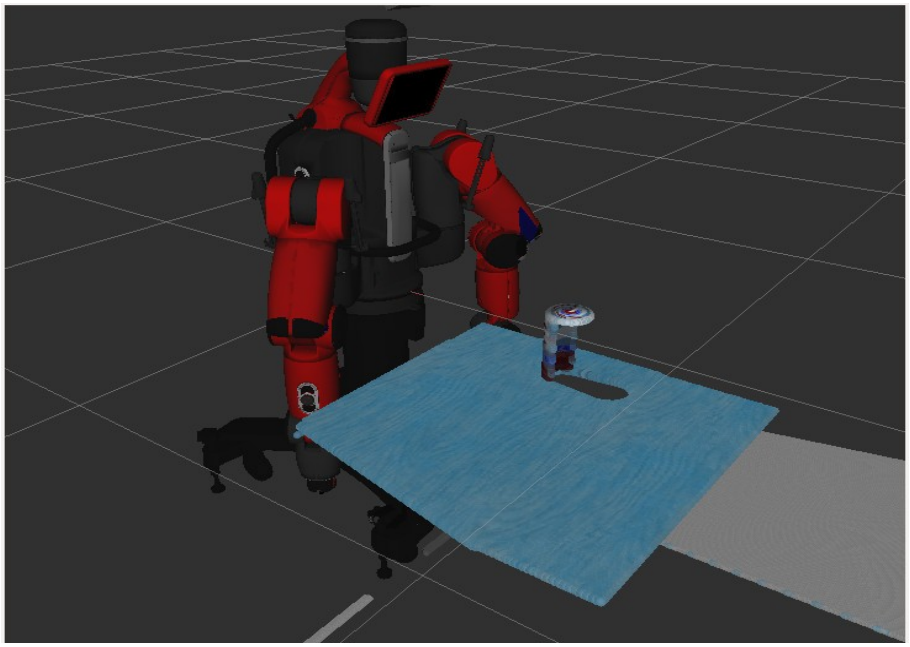
operates on the pointCloud stored at the pointer “pclTransformed_ptr_”. This pointCloud is a transformed version of the input cloud from the Kinect, re-expressed in “torso” coordinates. The objective of this filter_cloud_z() function is to identify a set of points that satisfy multiple conditions. First, qualifying points must lie within tolerance +/- “z_eps” of elevation “z_nom.” That is, the points to be considered lie approximately on a horizontal plane at the specified z-height. In addition, qualifying points must lie within a specified radius, “radius”, of a specified point, “centroid.” All points that fit these requirements are identified by their respective indices. The argument “indices” is a C++ vector of integers, and the filter function populates this vector with the identifying indices of all qualifying points.

This function operates on a pointCloud of type pcl::PointCloud<pcl::PointXYZ>::Ptr, and these points have been transformed into torso coordinates. However, the indices of chosen points are valid as well with respect to the colored pointCloud

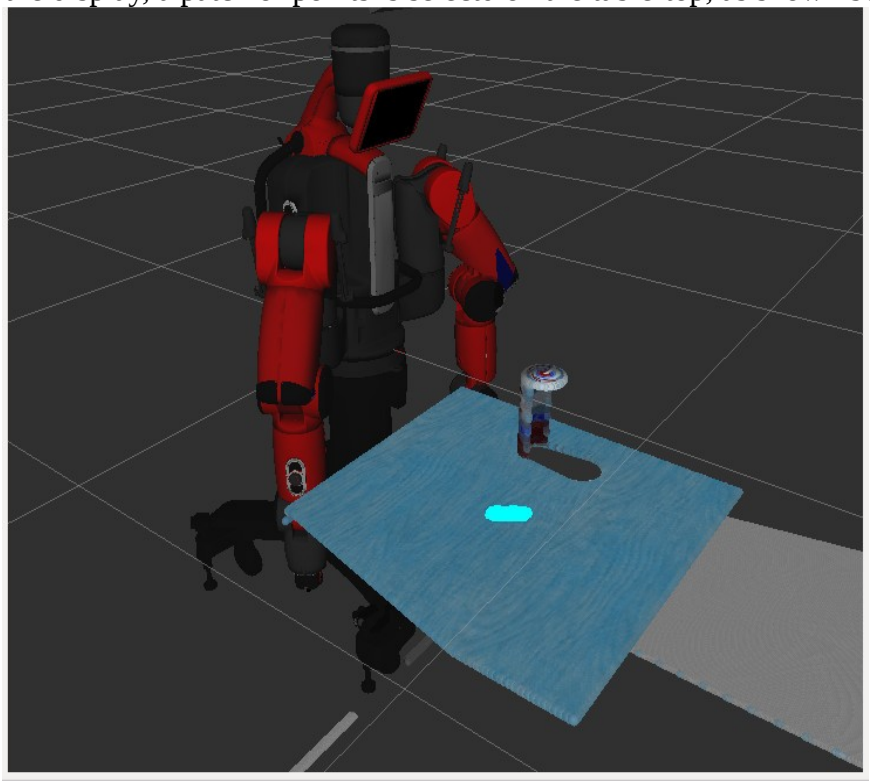
```
pcl::PointCloud<pcl::PointXYZRGB>::Ptr pclKinect_clr_ptr_
```

Although the transformation to torso coordinates discarded the color components and transformed all Cartesian coordinates, the correspondence between the transformed cloud and the original colored cloud is preserved via the point indices.

In the example program “cwru_pcl_utils_color_test.cpp,” the z-height of interest and the reference point “centroid” are computed from a patch of selected points. An illustrative example starts from the scene below, which shows objects displayed in Rviz via a colored pointCloud display.



Interacting with the display, a patch of points is selectd on the table-top, as shown below:



The selected patch is analyzed to identify a planar fit, which returns a surface normal, a z-height (relative to the torso frame) and the centroid of the selected points.

The program output at this point is:

[INFO] [1447636699.422727657, 2016.974000000]: RECEIVED NEW PATCH w/ 87 * 1 points
[INFO] [1447636699.422808449, 2016.974000000]: done w/ selected-points callback
[INFO] [1447636699.422861375, 2016.974000000]: transforming selected points
transforming npts = 87
[INFO] [1447636699.423709938, 2016.974000000]: fitting a plane to the transformed selected

points:
centroid: 0.760523 -0.320986 -0.155301
min eval is 3.72773e-13, corresponding to component 2
corresponding evec (est plane normal): -8.17265e-07 1.97898e-07 1

From the output, the surface-normal of the plane is very well aligned with the z-axis of the torso frame: (0;0;1). The centroid of the selected points is (0.760, -0.321, -0.155), which is consistent with the plane's distance from the origin (not displayed). This result provides the z-height and the centroid needed to find co-planar points within a specified radius. Using these arguments, the example program then calls:

cwru_pcl_utils.filter_cloud_z(plane_dist, z_eps, radius,centroid,selected_indices);

This function operates on the transformed data (not just the selected points). It finds all points that are within the specified z_eps tolerance of the z-coordinate "plane_dist", and from among these points, it finds which of them are within "radius" of the point "centroid." Points that pass both of these tests are recorded by installing their respective point indices in the vector "selected_indices."

The display from this operation is:
[INFO] [1447636699.428311399, 2016.985000000]: getting indices of coplanar points within radius 0.050000 of patch centroid
 number of points in range = 1424

Accessing pointClouds via Index Vectors: Since pointClouds can be very large, one should avoid making copies of them. In part, this is accomplished by using pointers to pointClouds instead of passing arguments by value. Additionally, one can refer to a subset of points from a large pointCloud by listing the point indices of interest.

Once desired processing has been performed, a pointCloud of interest can be displayed by extracting the points of interest and publishing them. The function:

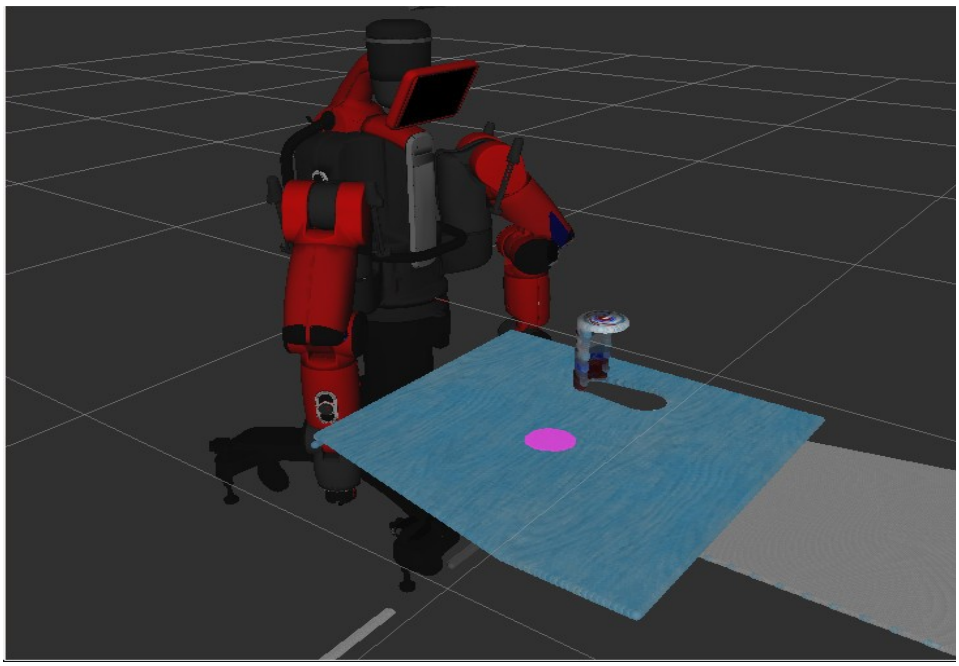
```
void copy_indexed_pts_to_output_cloud(vector<int> &indices,
    PointCloud<pcl::PointXYZRGB> &outputCloud);
```

Accepts a vector of point indices, operates (by default) on the colored pointCloud pclKinect_clr_ptr_ (a member variable of the cwru_pcl_utils class). It copies the header of this pointCloud to the provided "outputCloud", then it copies the x,y,z and r,g,b data from of the points of interest (listed in "indices") to the outputCloud. Although the indices originated from inspection of a transformed, uncolored cloud, the identified indices are still valid with respect to the original, non-transformed, colored cloud.

The points that satisfy the height and radius specifications are published as follows:

```
//convert datatype to compatible ROS message type for publication
pcl::toROSMsg(display_color_cloud, pcl2_display_cloud);
//update the time stamp, so rviz does not complain
pcl2_display_cloud.header.stamp = ros::Time::now();
//publish a point cloud that can be viewed in rviz (under topic pcl_cloud_display)
pubCloud.publish(pcl2_display_cloud);
```

The resulting rviz display appears as below:



The points that satisfy the height and radius specifications are displayed in magenta.

Interpreting Color from pointClouds: Next, the identified points of interest (as listed in "selected_indices") may be analyzed with respect to color. This is illustrated with the following function:

```
Eigen::Vector3d find_avg_color_selected_pts(vector<int> &indices);
```

which is invoked in the example as:

```
avg_color = cwru_pcl_utils.find_avg_color_selected_pts(selected_indices);
```

Inside this function, the red, green and blue values of the points of interest are averaged using:

```
for (int i=0;i<npts;i++) {
   index = indices[i];
   pt_color(0) = (double) pclKinect_clr_ptr_->points[index].r;
   pt_color(1) = (double) pclKinect_clr_ptr_->points[index].g;
   pt_color(2) = (double) pclKinect_clr_ptr_->points[index].b;
   avg_color+= pt_color;
}
avg_color/=npts;
```

If the indentified points have a (roughly) uniform color, the average color is representative of the object of interest, and additional points belonging to this object may be found by color matching. The color returned is a 3-D floating-point vector with the three values representing average red, green and blue components, respectively.

When performing color matching, variations in lighting can significantly change the component values. However, the relative strengths of these values should be more immune to lighting intensity (though coloring of ambient light can still interfere). Color "normalization" can be performed by dividing by total intensity, as follows:

```
normalized_avg_color = avg_color/avg_color.norm();
```

Better results may be obtained by converting the color to an alternative representation, such as HSV, but the illustrated approach is simple and may be adequate.

Output from example color averaging is:

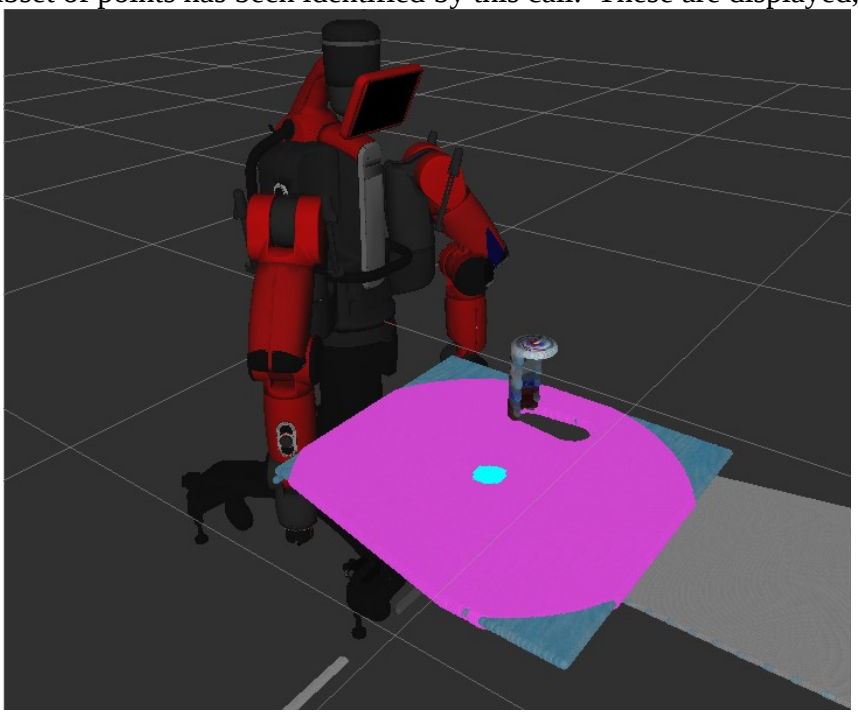
[INFO] [1447636699.672282389, 2017.225000000]: computing average color of representative points...
[INFO] [1447636699.673036616, 2017.225000000]: avg color = 75.608146, 140.173455, 171.016152

Finding Similarly Colored Points: After establishing a color of interest, the search for additional points of interest can be expanded. In the example code, a larger region of coplanar points is identified by increasing the radius specification, which is performed in the example with:

```
cwru_pcl_utils.filter_cloud_z(plane_dist, z_eps, radius_large,centroid,selected_indices);
```

The output from this function call was:
[INFO] [1447636699.673102209, 2017.225000000]: expanding search for coplanar points using radius= 0.500000
number of points in range = 126957

A much larger subset of points has been identified by this call. These are displayed, as below.



Next, this set of points is reduced by finding points that are colored similar to a color of interest, with:

```
cwru_pcl_utils.find_indices_color_match(selected_indices,
      normalized_avg_color, color_match_thresh, selected_indices_colormatch);
```

This function has inputs of “selected_indices”, which specifies the subset of identified coplanar points (within specified radius). A normalized color vector is also specified, which was previously determined via interactive point selection. A “color_match_thresh” parameter defines a tolerance

for accepting or rejecting a color match between a candidate point and the reference color. This threshold value may need to be iterated to find a suitable choice. The color-match function will use these parameters to identify qualifying points, which it returns by populating the corresponding point indices in " selected_indices_colormatch."

This operation is performed with the lines:

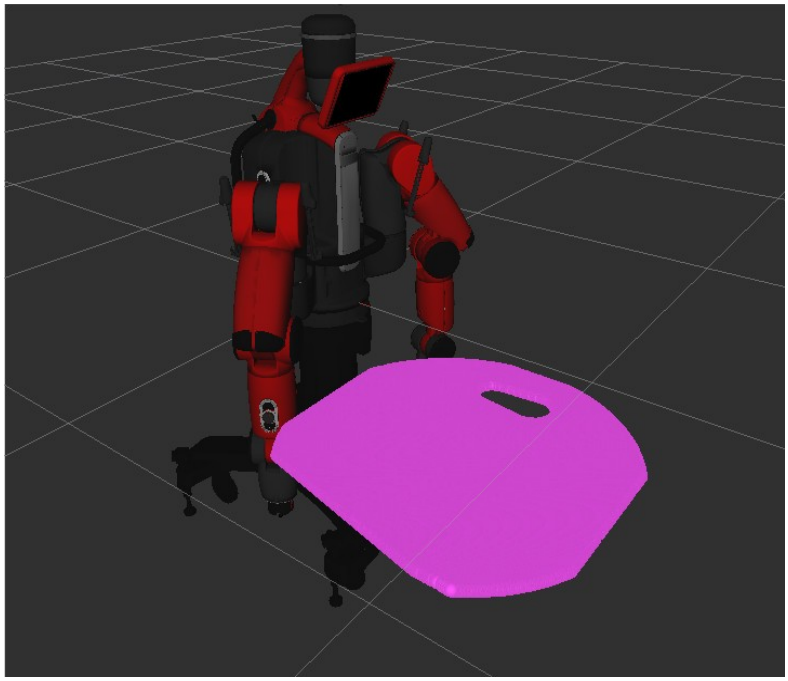
```
  for (int i=0;i<npts;i++) {
      index = input_indices[i];
      pt_color(0) = (double) pclKinect_clr_ptr_->points[index].r;
      pt_color(1) = (double) pclKinect_clr_ptr_->points[index].g;
      pt_color(2) = (double) pclKinect_clr_ptr_->points[index].b;
      pt_color = pt_color/pt_color.norm(); //compute normalized color
      if ((normalized_avg_color-pt_color).norm()<color_match_thresh) {
         output_indices.push_back(index);  //color match, so save this point index
         npts_matching++;
      }
   }
```

Each point among the candidates is described by a normalized color. If this color is within the color tolerance, the point index is saved in the provided vector of indices.

Output from this function call was:

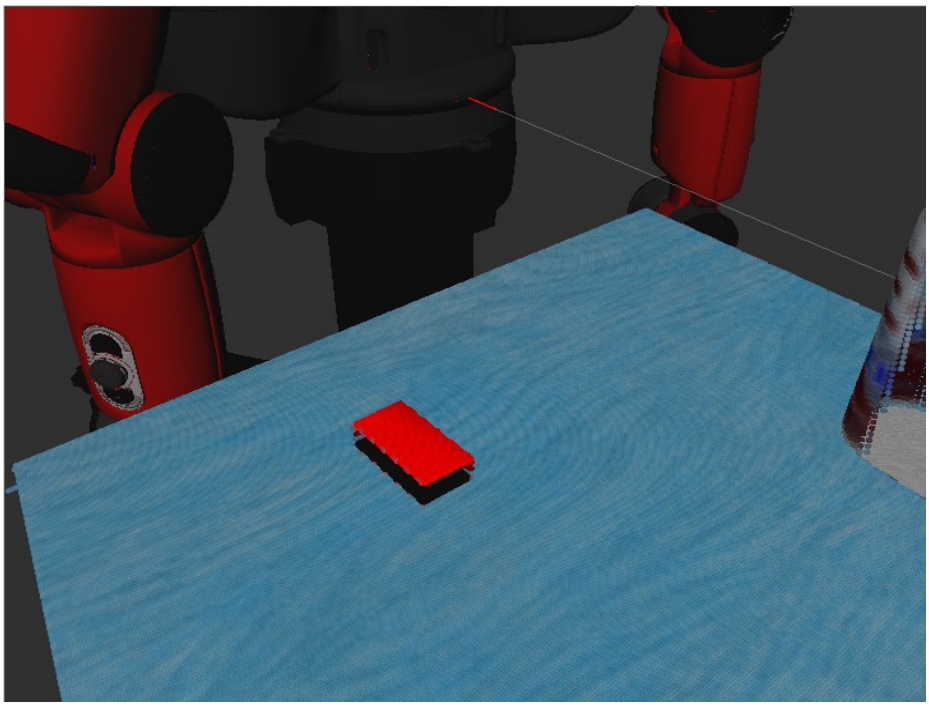
```
[ INFO] [1447636699.899131818, 2017.451000000]: extracting points colored similar to patch average:
enter color threshold (0 to 1): 0.1
[ INFO] [1447636715.098364303, 2032.476000000]: found 125320 color-match points from indexed set
```

The set of points that qualified as co-planar and matching color was large (over 125,320 points). These are copied to a local colored pointCloud object and published for display, yielding the display below:

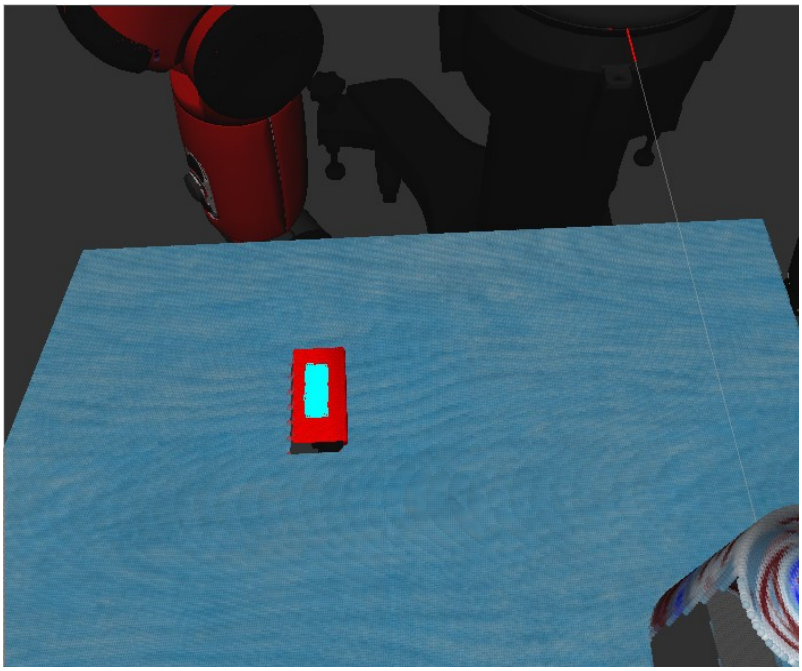


As shown, color matching has correctly identified more points on the table surface.

This process is illustrated for another case below: a red block on the table. The initial scene is:



Points on the red block are selected:

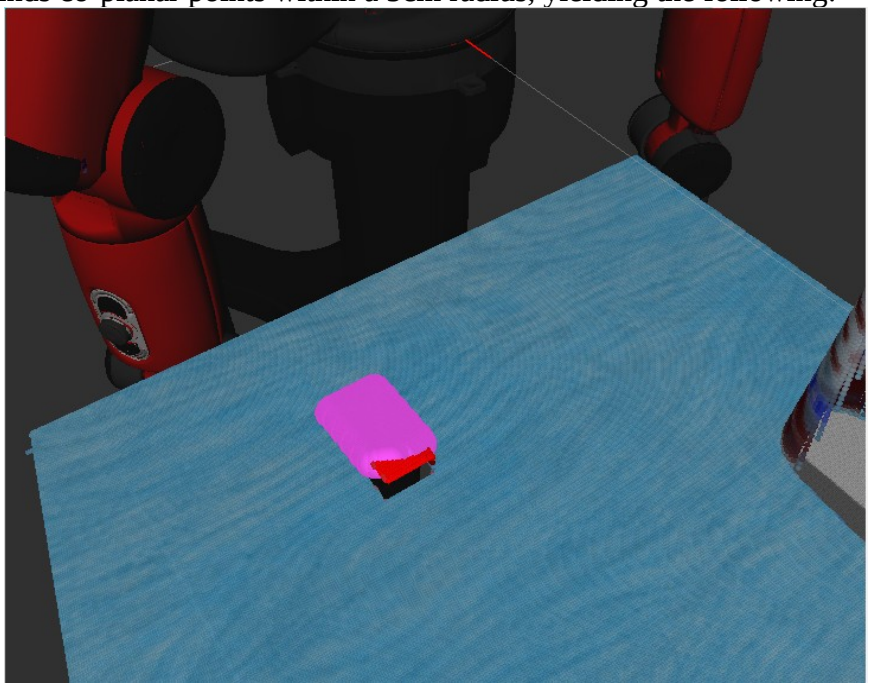


The first selection results in the output:

[INFO] [1447640360.907010997, 3402.767000000]: RECEIVED NEW PATCH w/ 15 * 1 points
[INFO] [1447640360.907051543, 3402.767000000]: done w/ selected-points callback
[INFO] [1447640360.907075420, 3402.767000000]: transforming selected points
transforming npts = 15
[INFO] [1447640360.907270847, 3402.767000000]: fitting a plane to the transformed selected points:
centroid: 0.485981 -0.342726 -0.135499

min eval is 4.84774e-14, corresponding to component 2
corresponding evec (est plane normal): 4.98607e-06 -6.76792e-06   1
max eval is 0.00752872, corresponding to component 0
corresponding evec (est major axis):   -0.99556 -0.0941325 4.32684e-06
[INFO] [1447640360.908072317, 3402.768000000]: normal: 4.98607e-06 -6.76792e-06   1;
dist = -0.135494

Next, the code finds co-planar points within a 5cm radius, yielding the following:



and which produces the output:

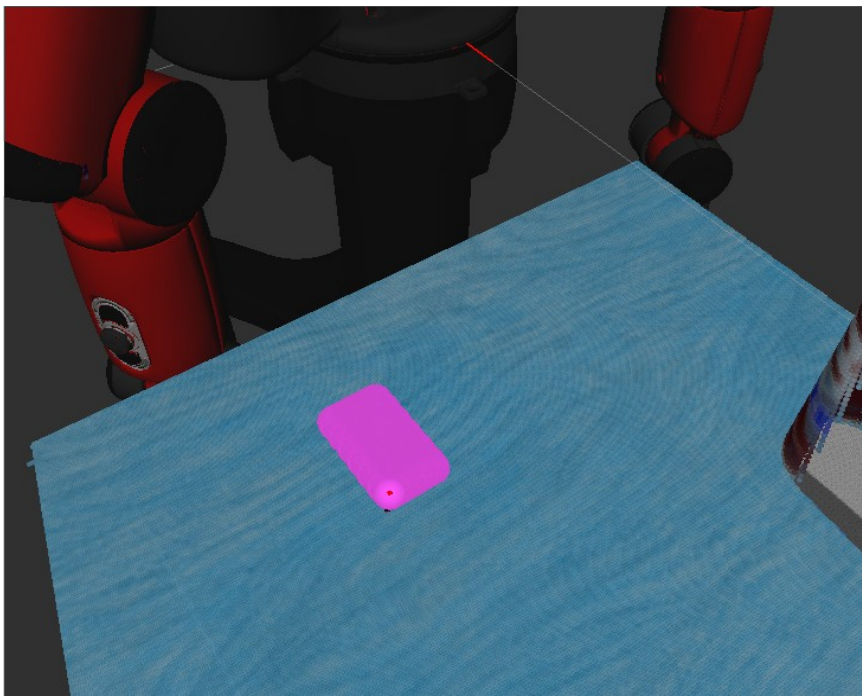
[INFO] [1447640360.908100591, 3402.768000000]: getting indices of coplanar points within radius 0.050000 of patch centroid
 number of points in range = 1035
copying cloud w/ npts =1035

The average color is then computed and printed out:

[INFO] [1447640361.047081386, 3402.904000000]: computing average color of representative points...
[INFO] [1447640361.047563692, 3402.904000000]: avg color = 242.485990, 0.607729, 0.746860

This color is dominantly red (as expected), with the “r” term in (r,g,b) nearly saturated (where saturation is 255).

Having identified the color of interest, the search for coplanar points is expanded, resulting in:



with printed output:

[INFO] [1447640361.184385079, 3403.038000000]: extracting points colored similar to patch average:
enter color threshold (0 to 1): 0.1
[INFO] [1447640426.355278927, 3466.907000000]: found 1227 color-match points from indexed set

copying cloud w/ npts =1227

As shown above, pointCloud data can be interpreted both in terms of spatial and color properties to help identify objects of interest.