RANSAC

INTRODUCTION

In this exercise, the goal is to create a code implementing a RANSAC (Random Sample Consensus) algorithm to detect the 3 planes in a given Point Cloud.

First of all, a brief explanation of RANSAC, which is one of the simplest, most used and effective algorithm developed for features detection in machine vision.

What RANSAC does is to extract features from a 2D or a 3D image (called Point Cloud) exploiting a very simple idea: the closeness the point to an estimated model, the better the model is.

First of all, it is fundamental to know which type of feature must be found: in our case, we need to find 3 planes in the given point cloud. We know that a plain in the space, without additional constraints, is univocally determined by 3 points. RANSAC selects, RANDOMLY, 3 points within the cloud and, basing on these points, it computes the estimated parameters for the plane passing through these points.

Once the model has been estimated, the distance between every point in the cloud and the estimated plane is computed: in this step, points of the cloud are divided into two dataset: Inliers and Outliers dataset. A point is defined as an inlier for the model into consideration if its distance from the current plane is less than a defined threshold (user-defined), otherwise it is classified as an outlier.

If after scanning all the points the number of found inliers is sufficiently high (it is bigger with respect to a minimum number of inliers defined by the user), this plane is classified as a possible “good model” and the error associated with this model (which usually is defined as the sum of the distances between all the inlier points to the plane) is stored for comparison with next “good models”.

After this very first model, the procedure is repeated for a number of iterations defined by the user. In each iterations three random points are selected, a new model for the plane is evaluated and if the model has enough inliers, it is compared with the previous model: if its total error is less than the total error of the previous model, it means that the current plane fits better the points than the other planes that we previously have estimated.

At the end of the algorithm, the best fit model is returned.

Here you can find a high level description of the RANSAC algorithm:

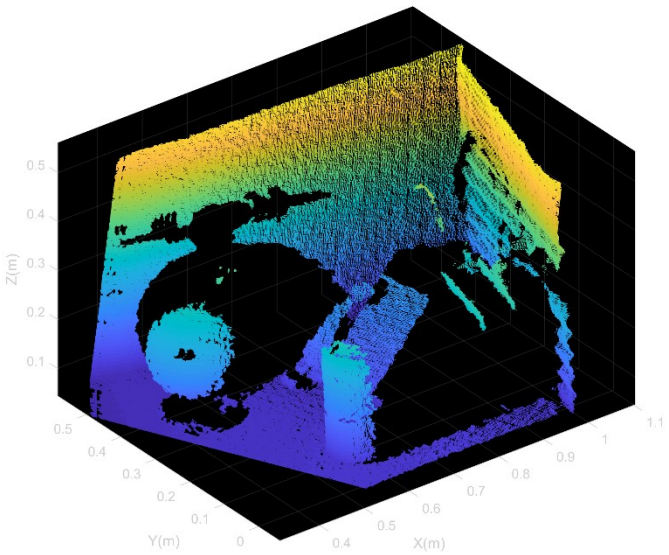
```
Given:
    data – a set of observations
    model – a model to explain observed data points
    n – minimum number of data points required to estimate model
parameters
    k – maximum number of iterations allowed in the algorithm
    t – threshold value to determine data points that are fit well
by model
    d – number of close data points required to assert that a
model fits well to data

Return:
    bestFit – model parameters which best fit the data (or nul if
no good model is found)

iterations = 0
bestFit = nul
bestErr = something really large
while iterations < k {
    maybeInliers = n randomly selected values from data
    maybeModel = model parameters fitted to maybeInliers
    alsoInliers = empty set
    for every point in data not in maybeInliers {
        if point fits maybeModel with an error smaller than t
             add point to alsoInliers
    }
    if the number of elements in alsoInliers is > d {
        % this implies that we may have found a good model
        % now test how good it is
        betterModel = model parameters fitted to all points in
maybeInliers and alsoInliers
        thisErr = a measure of how well betterModel fits these
points
        if thisErr < bestErr {
            bestFit = betterModel
            bestErr = thisErr
        }
    }
    increment iterations
}
return bestFit
```

CODE DESCRIPTION

Let’s procede with the description of the developed code. First of all, the pointCloud.mat model is imported to the workspace and x, y and z coordinates are extracted from the pointCloud as matrix: these 3 matrices have dimension 480x640 single and each element (i,j) represents the value of the coordinate for the pixel (i,j) in the acquired 3D cloud. The original Cloud Ploint is displayed:



SEGMENTATION OF PLANE 1

First, the variables that will store the best values for the final returned model are initialized: bestModel_1 is an array containing the coeff. of the final best Plain 1; bestInliers_1 and bestOutliers_1 are initialized as a NaN point cloud with the same dimensions of the original point cloud (actually, bestOutliers will not be used in the algorithm but it is stored in order to have it in case of necessity); finally, ni_best_1 and no_best_1 are the variables to store the final number of inliers and outliers for the best estimated model, and so are initialize as 0 and inf, respectively.

In addition, the desired threshold, min_inliers and iterations values are selected for the purpose of the code. In this case, this value have been selected after some trials: a threshold of 0.0125 is fine to detect a good plane without including points that actually do not belong to the plane itself. The min_ inliers has a value of 50000 because the plane with the minimum number of inliers (the horizontal plane) has 55000 ca inliers. This value of course has been identified after some trials and analysis of the original point cloud, and in this way, we are sure that the algorithm will take into considerations only model of actual possible planes and not of planes cutting in half the other objects that are in the Point Cloud.

After this initial setup, there is the core of the algorithm. This is made of an external for loop which reiterates the procedure for a number of iterations equal to the number defined above. Initially, at every iteration support variables inliers, outliers, ni, no and totalError are initialize in order to store the information about the current model during the iteration.

Then, the three random points are selected using randi function which returns a value within the specified range, in this case 480x640, so that one point can be defined as a vector with dimension 1x3 (the three values are the x, y, z coordinate of the pixel). Notice that while the selected random point has at least one NaN coordinate, another point is selected (NaN values means that in correspondence of that pixels, there is no object).

x1, y1, z1, x2, y2, z2 and x3, y3, z3 are defined as the coordinate of point 1, point 2 and point 3.

Next step, given the three points, is to compute the estimated coefficients of the plane passing through them. In this case, for reasons explained in the section “CONSIDERATIONS”, we used the definition of the plane that exploits the definition of the vectors passing through the three points.

v1 is the vector passing through points 1 and 2, v2 is the vector passing through point 1 and 3. At this point, the vector of the plain 1 can be computed as the cross product between the two vectors: this vector is orthogonal to the plane and its 3 values are the coefficients a_est, b_est and c_est of the plane.

Finally, since plane equation is ax + by + cz = d, d value is computed imposing the passage through one point.

Now we can procedure with the division of the whole point cloud into inliers and outliers dataset. To do so, two for loops are used to scroll the points along raws and columns of the point cloud matrix. Also here, if the point (i,j) has one coordinate which is NaN, it is simply not taken into consideration and j value is incremented. When the point is an actual point of an object in the scene, the distance of this point from the estimated plane is computed using the formula:

$$d(P_0, p) = \frac{|ax_0 + by_0 + cz_0 + d|}{\sqrt{a^2 + b^2 + c^2}}.$$

This distance is also equal to the error related to the model and this point.

If this distance is less than the threshold → the point in an inlier and so the error is added to the totalError variable, the inliers matrix of NaN is updated and its value in position (i,j) is updated with the values of the coordinates of point in position (i,j) in the original scene. Number of inliers ni is incremented of 1.

If this distance is bigger than the threshold → the point is an outlier and the outliers matrix of NaN is updated and its value in position (i,j) is updated with

the values of the coordinates of point in position (i,j) in the original scene. Number of inliers ni is incremented of 1.

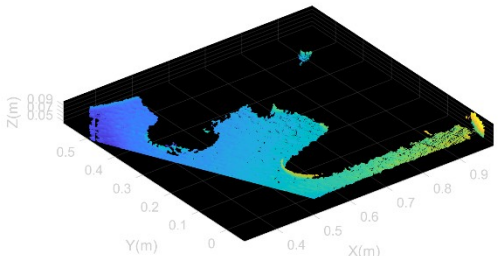
After the division of the whole point cloud, if the number of inliers of this plane is greater than the min_inliers value, the model is compared with the previous found potential “good models”. If the total error of the current model is less than the bestError (at first iteration, bestError is inf), all the variables to describe the best model are updated with the current values. On the screen is displayed that we may have found a potential plane!

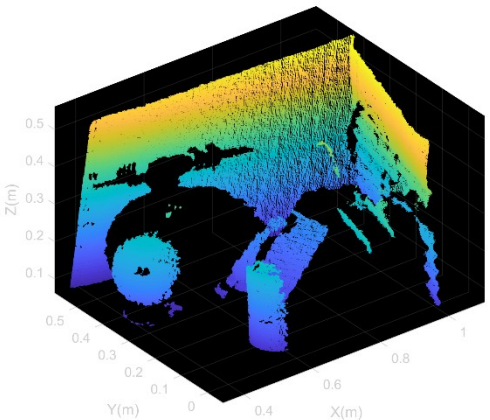
A particular condition is added to the for loop: if the current model has a number or inliers greater than 80000, the loop stops immediately returning the best model. This is an useful addition because, after many trials, I noticed that the two vertical planes have both a number of inliers bigger than 80000 if a good plane is identified. In this way, if the encounter of this plane is reached before the maximum number of iterations, we can skip directly to the other plane detection because we are sure that if the number of inliers is so high, the algorithm must have detected one of the two vertical planes.

When the loop has been executed, we can print the resulting identify plane:

Since we have updated bestInliers cloud with the values of the corresponding inliers points, plane_1 is simply this cloud and can be shown with function pcshow.

After that, remain_ptCloud is equal to bestOutliers matrix and can be displayed.



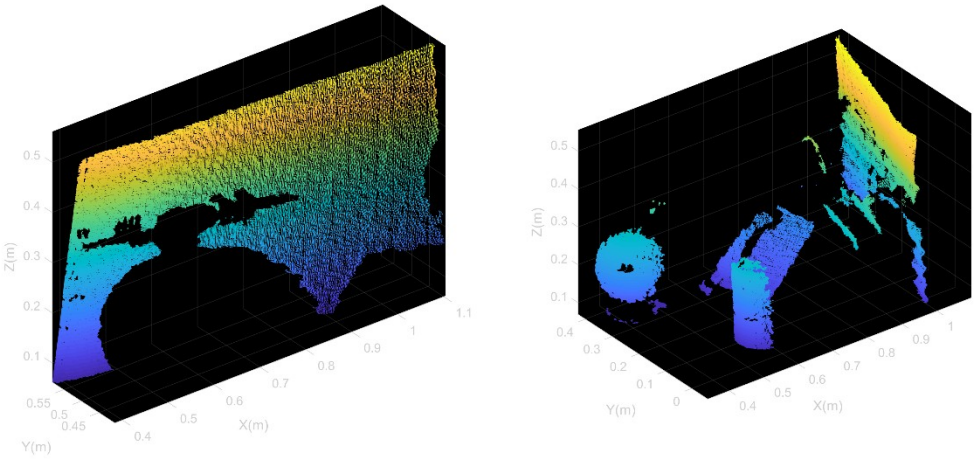


SEGMENTATION OF PLANE 2

The same exact algorithm is repeated to identify the plane 2. However, a proper reinitialization of the point cloud is necessary: if the point cloud would not be updated with the same value of the remain_ptCloud, the algorithm could detect again the same plane as plane 1!

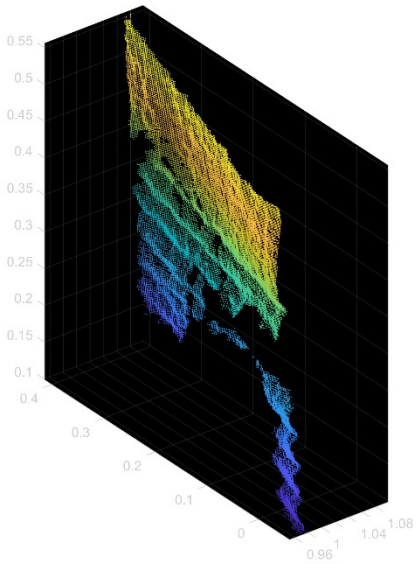
So in the first part of the section, ptCloud is updated, together with x, y and z coordinate and the variables to store the best values for the best model are initialize exactly at the same way done for plane 1.

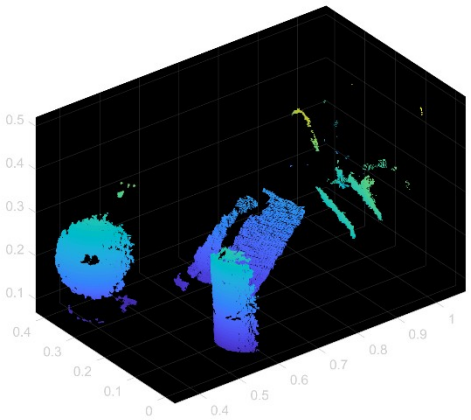
The result after Plane 2 removal are:



SEGMENTATION OF PLANE 3

Procedure is reiterated once again, point cloud is updated with the remaining point cloud at the same way it is done for plane 2 and variables for plane 3 are initialized at the beginning. Final result, after Plane 1, Plane 2 and Plane 3 removal is:





FINAL CONSIDERATION AND ANALYSIS OF RESULTS

The developed code is a very basic implementation of RANSAC. Since this algorithm is so much used in real applications, there exists many version of it. For example, some versions recompute feature parameters in order to better fit the inliers, or other versions do not take into considerations only 3 random points, but a set of random points and averaging them to have faster performances. The one implemented has only the goal to show how the basic principle work and demonstrate that even a simply version like this one can efficiently detect the correct features: as a matter of fact, the code has been run many times, and in almost every occasion it is able to detect the planes. In addition, some nice info about the current number of the iteration, the value of the final plane equation are displayed to show to the user what is happening real-time.

The created code is not so fast as better and more professional implementations, but it can still be considered acceptable: usually, it is able to detect one of the three wanted plane within the 40 iterations. However, just to be sure, the default value is of 100 iterations.

Other important considerations that have to be done are:

1) Being the whole algorithm stochastic, if we are very lucky the algorithm detects the planes every time at the very first iteration. On the other hand, if we are very unlucky, the algorithm is not able to detect the correct plane within the max number of iterations. So, even if the code have been tested many times and has proven to be sufficiently robust, there is no 100% guarantee that every time it works.
2) The order of plane detection is random: sometimes the code has detected the horizontal plane as the plane 1 and the vertical ones later, other times this plane has been detected as the very last. This is of course related to the casualty of the algorithm.
3) There are different ways in literature to evaluate which of the model is the best. In this one, the comparison between model errors, which is given by the sum of the distance of all the inliers with respect to the model plane, is done and has proven to be robust because in this way algorithm searches for the plane which best fit the actual plane, and not for a plane with a high number of inliers. In fact, we cannot say that comparing the number of inliers to define the best model will always detect the best fit plane, while with the error comparison this is always done.
4) Originally, I tried to evaluate a_est, b_est, c_est and d solving the known system of symbolic equations for the plane passing through 3 points. However, it seemed that the syms variables slow down a lot the code and also I had many NaN values as distance value. Evaluating the coefficients as afore explained, it is a way better and faster technique.

Finally, the very last part of the code is just an extension to detect the cylinder in point cloud using pcfitcylinder function. This has been done here to show

that if the planes have been removed, there is no need to specify a range where the cylinder is located.

NOTES

Together with this report, I attached a pdf containing an example of the command window for an iteration of the code