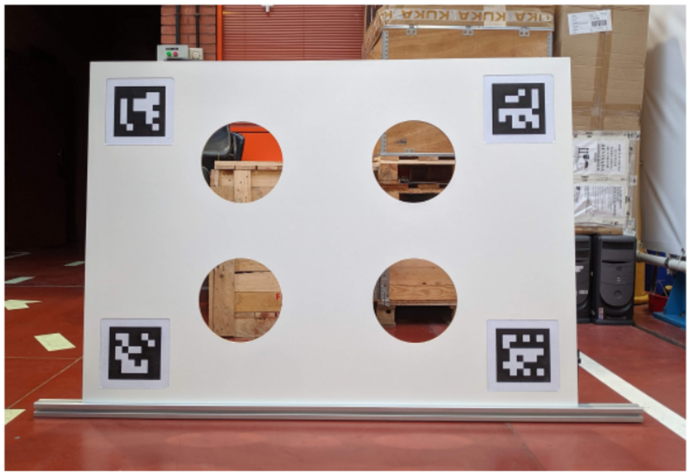
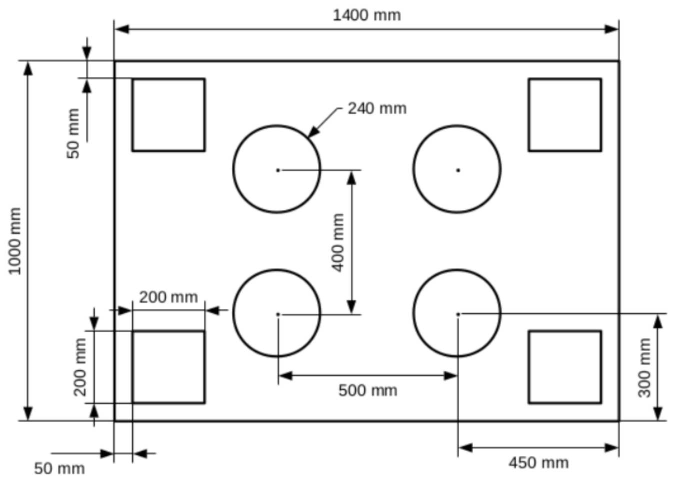
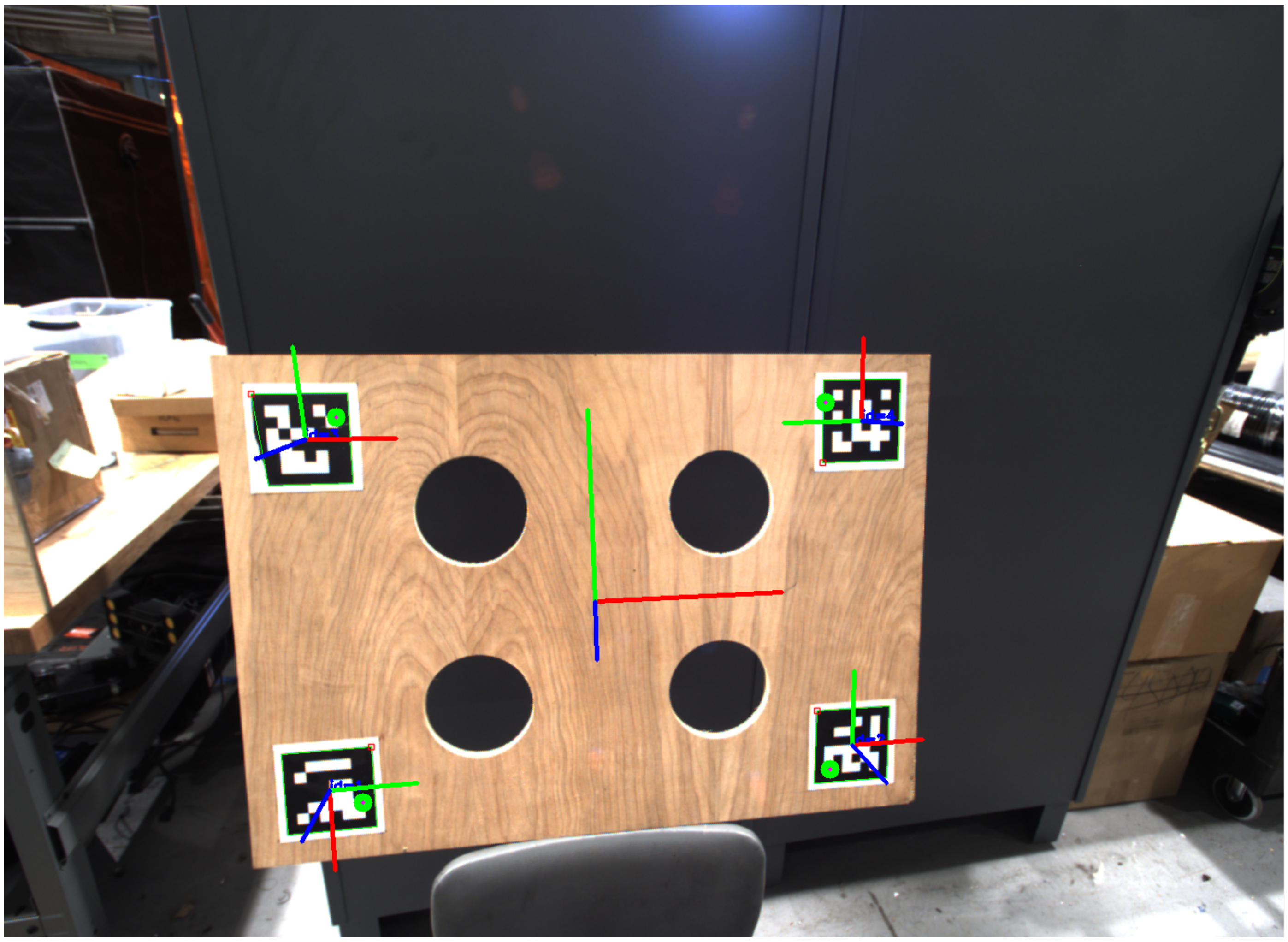
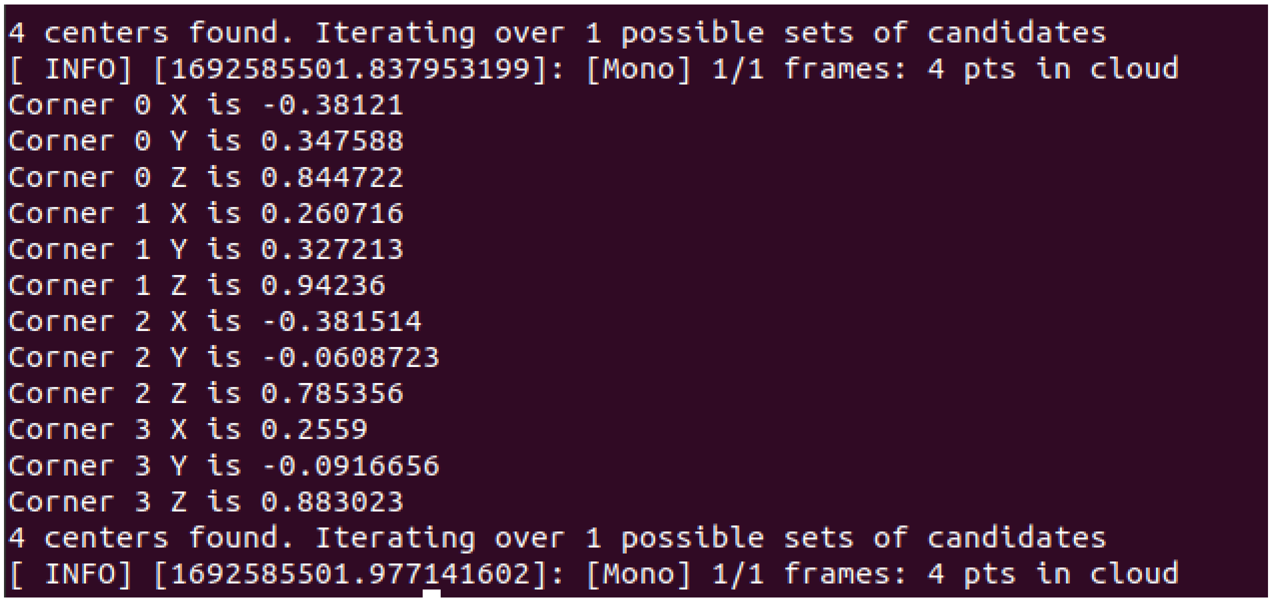
Velo2Cam August 18 Trials

- The calibration board (0.5x scale) was used along with Aruco tags to attempt calibration
- The ordering of the target seems to be different in the repo but it did not affect the detections
-  

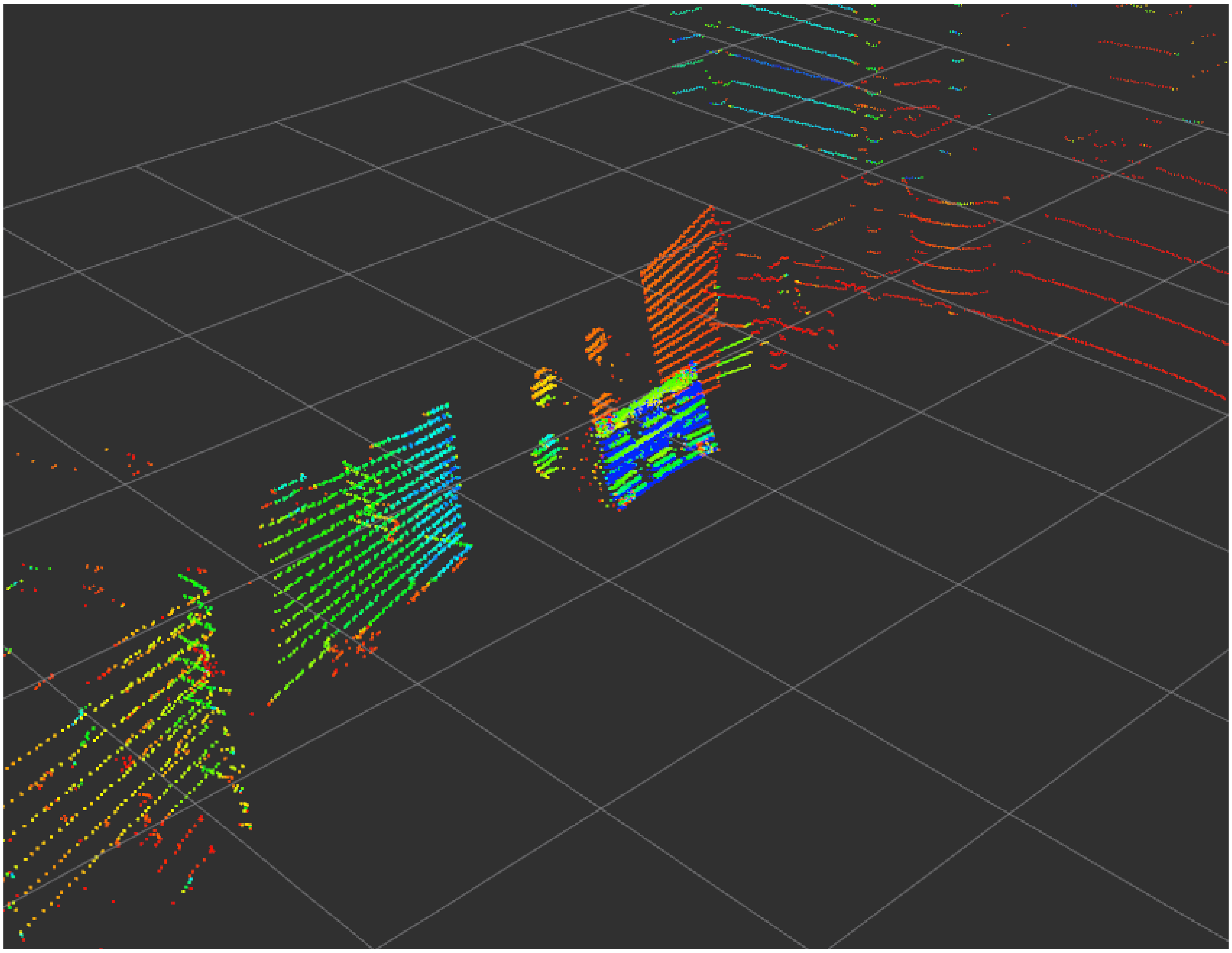
- 

- 

```
4 centers found. Iterating over 1 possible sets of candidates
[ INFO] [1692585501.837953199]: [Mono] 1/1 frames: 4 pts in cloud
Corner 0 X is -0.38121
Corner 0 Y is 0.347588
Corner 0 Z is 0.844722
Corner 1 X is 0.260716
Corner 1 Y is 0.327213
Corner 1 Z is 0.94236
Corner 2 X is -0.381514
Corner 2 Y is -0.0608723
Corner 2 Z is 0.785356
Corner 3 X is 0.2559
Corner 3 Y is -0.0916656
Corner 3 Z is 0.883023
4 centers found. Iterating over 1 possible sets of candidates
[ INFO] [1692585501.977141602]: [Mono] 1/1 frames: 4 pts in cloud
```

Lidar Detections

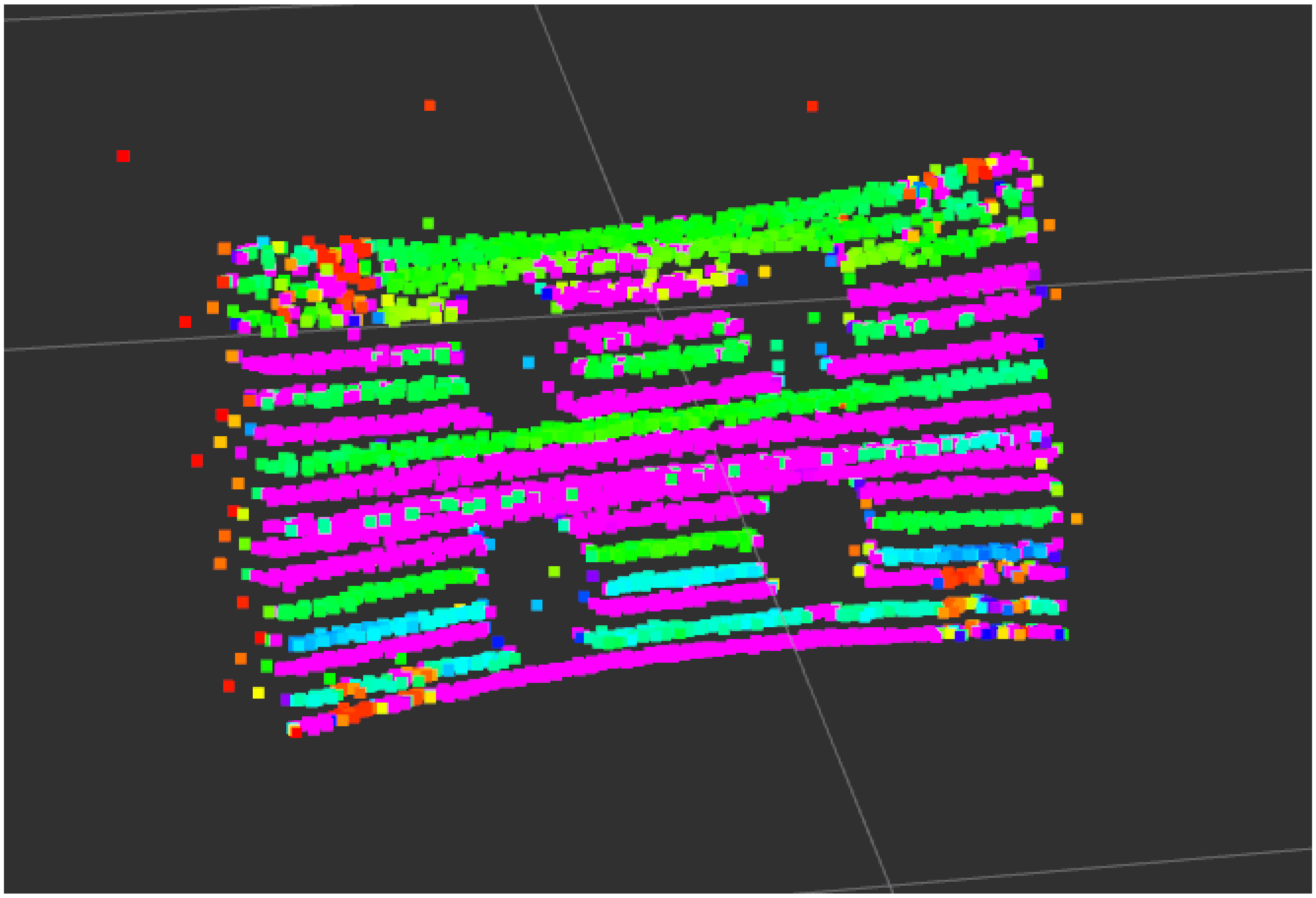
The target geometry for the Lidar are determined by the circular holes and the Lidar points which fall on the surface behind the target. The original cloud is huge and will need to be filtered



The filtering of the above cloud is done in two steps:

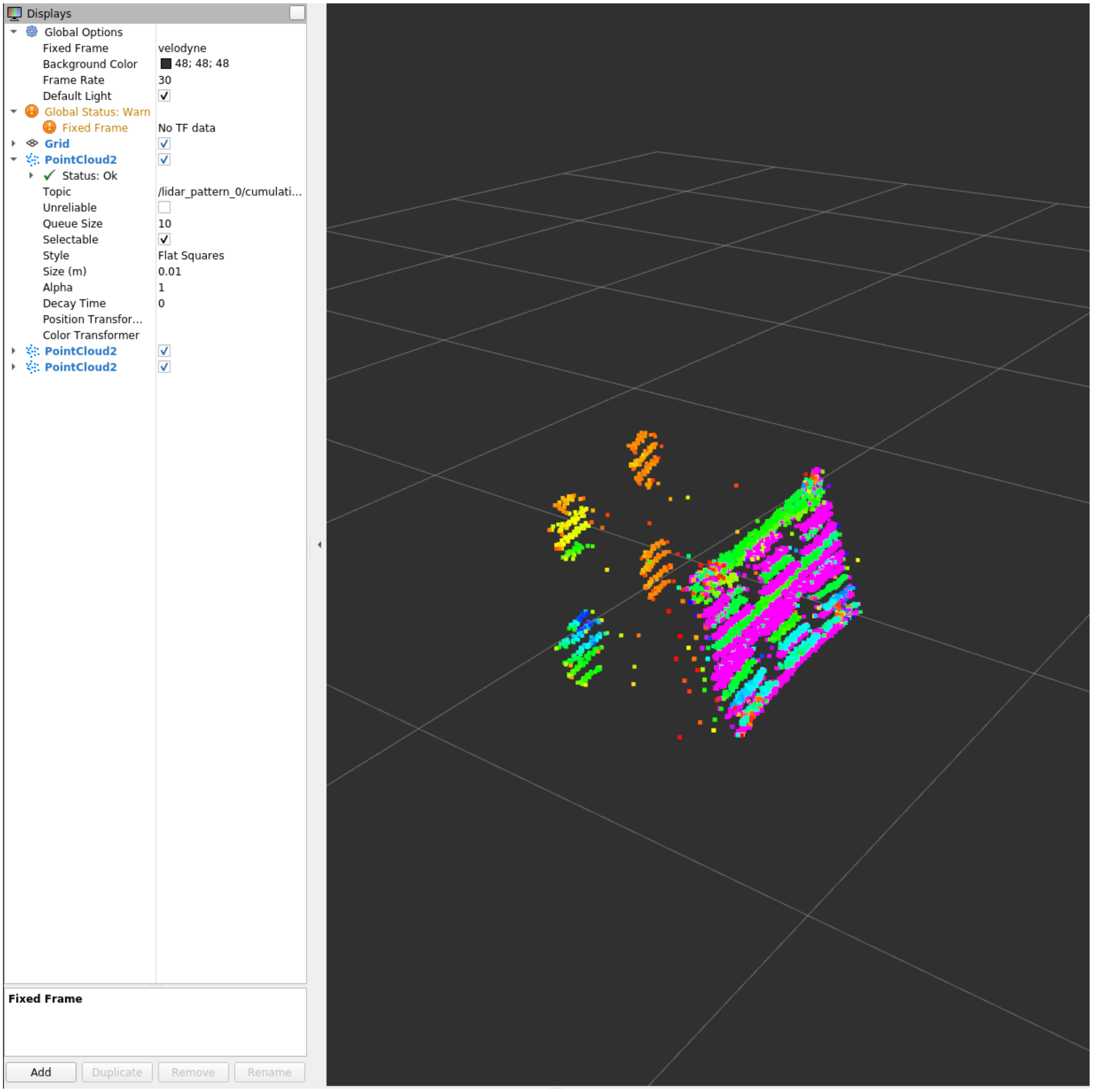
Range Filtered Cloud

This pointcloud will need to be filtered using the **max and min passthrough radius** to remove all points except the target board as shown below



zyx Filtered Cloud

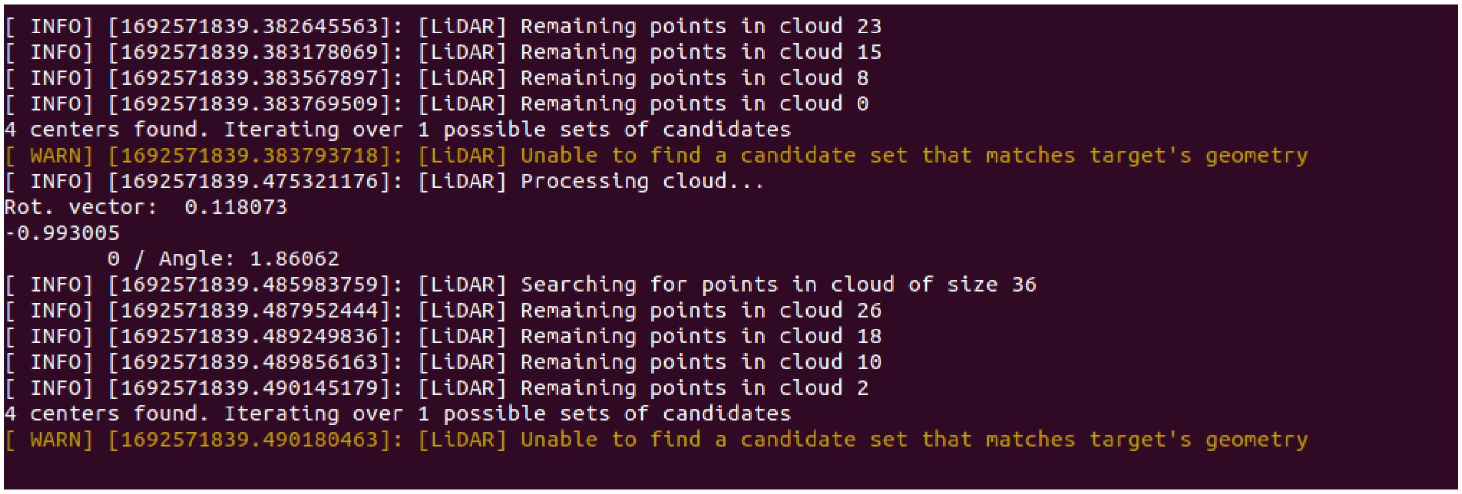
The passthrough_x_velo, passthrough_y_velo, and passthrough_z_velo needs to be set to filter the cloud to make only the target and the scan lines passing through the target circles


Displays
Global Options
Fixed Frame velodyne
Background Color ■ 48; 48; 48
Frame Rate 30
Default Light ✓
Global Status: Warn
Fixed Frame No TF data
Grid ✓
PointCloud2 ✓
Status: Ok
Topic /lidar_pattern_0/cumulati...
Unreliable
Queue Size 10
Selectable ✓
Style Flat Squares
Size (m) 0.01
Alpha 1
Decay Time 0
Position Transfor...
Color Transformer
PointCloud2 ✓
PointCloud2 ✓

Fixed Frame

Issues with Lidar Detections

Although the filtering seems to work and 4 centers are found, it cannot still find the target geometry. The error is shown below:



```
[ INFO] [1692571839.382645563]: [LiDAR] Remaining points in cloud 23
[ INFO] [1692571839.383178069]: [LiDAR] Remaining points in cloud 15
[ INFO] [1692571839.383567897]: [LiDAR] Remaining points in cloud 8
[ INFO] [1692571839.383769509]: [LiDAR] Remaining points in cloud 0
4 centers found. Iterating over 1 possible sets of candidates
[ WARN] [1692571839.383793718]: [LiDAR] Unable to find a candidate set that matches target's geometry
[ INFO] [1692571839.475321176]: [LiDAR] Processing cloud...
Rot. vector:  0.118073
-0.993005
        0 / Angle: 1.86062
[ INFO] [1692571839.485983759]: [LiDAR] Searching for points in cloud of size 36
[ INFO] [1692571839.487952444]: [LiDAR] Remaining points in cloud 26
[ INFO] [1692571839.489249836]: [LiDAR] Remaining points in cloud 18
[ INFO] [1692571839.489856163]: [LiDAR] Remaining points in cloud 10
[ INFO] [1692571839.490145179]: [LiDAR] Remaining points in cloud 2
4 centers found. Iterating over 1 possible sets of candidates
[ WARN] [1692571839.490180463]: [LiDAR] Unable to find a candidate set that matches target's geometry
```

I looked up the calculations which determine if a good enough match has been found for the detected circles

```
/**
    Geometric consistency check
    At this point, circles' center candidates have been computed
  (found_centers). Now we need to select the set of 4 candidates that best fit
  the calibration target geometry. To that end, the following steps are
  followed: 1) Create a cloud with 4 points representing the exact geometry of
  the calibration target 2) For each possible set of 4 points: compute
  similarity score 3) Rotate back the candidates with the highest score to their
  original position in the cloud, and add them to cumulative cloud
  **/
  std::vector<std::vector<int>> groups;
  comb(centroid_candidates->size(), TARGET_NUM_CIRCLES, groups);
  double groups_scores[groups.size()];  // -1: invalid; 0-1 normalized score
  for (int i = 0; i < groups.size(); ++i) {
    std::vector<pcl::PointXYZ> candidates;
    // Build candidates set
    for (int j = 0; j < groups[i].size(); ++j) {
      pcl::PointXYZ center;
      center.x = centroid_candidates->at(groups[i][j]).x;
      center.y = centroid_candidates->at(groups[i][j]).y;
      center.z = centroid_candidates->at(groups[i][j]).z;
      std::cout << "For Group " << j << std::endl;
      std::cout << "Centroid X is " << center.x << std::endl;
      std::cout << "Centroid Y is " << center.y << std::endl;
      std::cout << "Centroid Z is " << center.z << std::endl;
      candidates.push_back(center);
    }

    // Compute candidates score
    Square square_candidate(candidates, delta_width_circles_,
                            delta_height_circles_);
    groups_scores[i] = square_candidate.is_valid()
                           ? 1.0
                           : -1;  // -1 when it's not valid, 1 otherwise
  }

  int best_candidate_idx = -1;
  double best_candidate_score = -1;
  for (int i = 0; i < groups.size(); ++i) {
    if (best_candidate_score == 1 && groups_scores[i] == 1) {
      // Exit 4: Several candidates fit target's geometry
      ROS_ERROR(
          "[LiDAR] More than one set of candidates fit target's geometry. "
          "Please, make sure your parameters are well set. Exiting callback");
      return;
    }
    if (groups_scores[i] > best_candidate_score) {
      best_candidate_score = groups_scores[i];
      best_candidate_idx = i;
    }
```

```
}

if (best_candidate_idx == -1) {
  // Exit 5: No candidates fit target's geometry
  ROS_WARN(
      "[LiDAR] Unable to find a candidate set that matches target's "
      "geometry");
  return;
}
```

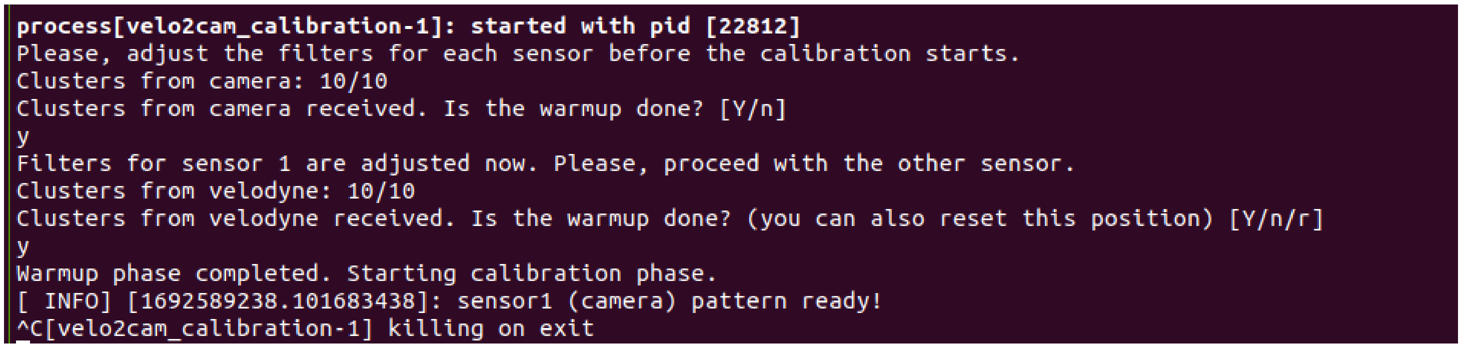
```
bool is_valid() {
    pcl::PointCloud<pcl::PointXYZ>::Ptr candidates_cloud(
        new pcl::PointCloud<pcl::PointXYZ>());
    // Check if candidates are at 5% of target's diagonal/2 to their centroid
    for (int i = 0; i < _candidates.size(); ++i) {
      candidates_cloud->push_back(_candidates[i]);
      float d = distance(_center, _candidates[i]);
      // NOTE: target_diagonal is calculated as: _target_diagonal = sqrt(pow(width, 2) +
      if (fabs(d - _target_diagonal / 2.) / (_target_diagonal / 2.) >
          GEOMETRY_TOLERANCE) {
        return false;
      }
    }
    // Check perimeter?
    std::vector<pcl::PointXYZ> sorted_centers;
    sortPatternCenters(candidates_cloud, sorted_centers);
    float perimeter = 0;
    for (int i = 0; i < sorted_centers.size(); ++i) {
      float current_distance = distance(
          sorted_centers[i], sorted_centers[(i + 1) % sorted_centers.size()]);
      if (i % 2) {
        if (fabs(current_distance - _target_height) / _target_height >
            GEOMETRY_TOLERANCE) {
          return false;
        }
      } else {
        if (fabs(current_distance - _target_width) / _target_width >
            GEOMETRY_TOLERANCE) {
          return false;
        }
      }
      perimeter += current_distance;
    }
    float ideal_perimeter = (2 * _target_width + 2 * _target_height);
    if (fabs((perimeter - ideal_perimeter) / ideal_perimeter >
             GEOMETRY_TOLERANCE)) {
      return false;
    }

    // Check width + height?
    return true;
  }
```

Looking at the above code, I decided to try two things:

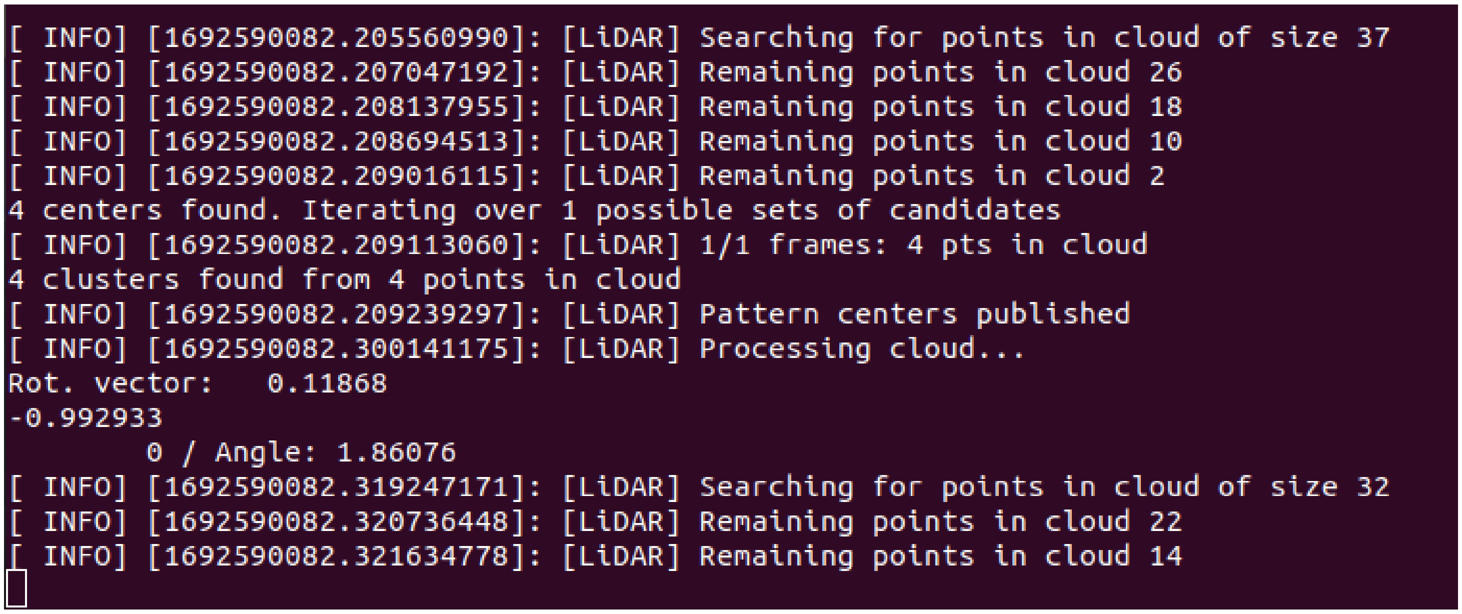
1. Increase the GEOMETRY_TOLERANCE value
2. Remove the perimeter check just to try

Increasing the Geometry tolerance from 0.06 to 0.1 did help and it accumulates valid clusters over many frames now (not very consistent though). Futher, this incosistency
causes the calibration to get stuck post the warm-up phase



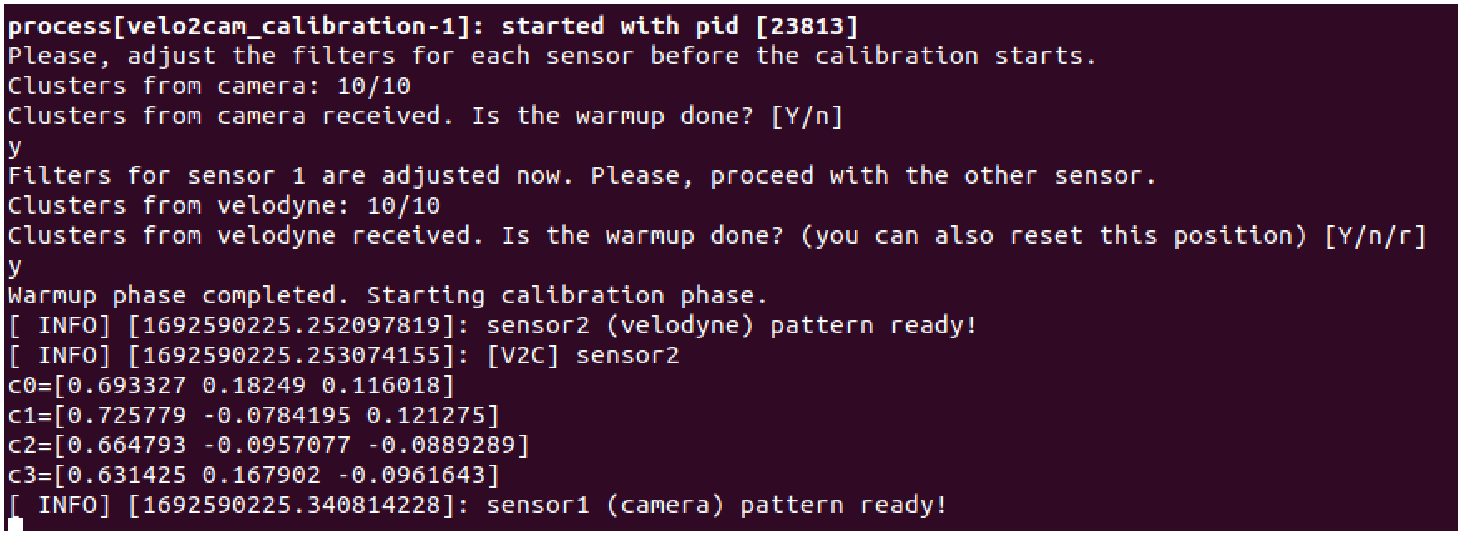
```
process[velo2cam_calibration-1]: started with pid [22812]
Please, adjust the filters for each sensor before the calibration starts.
Clusters from camera: 10/10
Clusters from camera received. Is the warmup done? [Y/n]
y
Filters for sensor 1 are adjusted now. Please, proceed with the other sensor.
Clusters from velodyne: 10/10
Clusters from velodyne received. Is the warmup done? (you can also reset this position) [Y/n/r]
y
Warmup phase completed. Starting calibration phase.
[ INFO] [1692589238.101683438]: sensor1 (camera) pattern ready!
^C[velo2cam_calibration-1] killing on exit
```

Removing the Perimeter check and increasing Geometry Tolerance to 0.5 did seem to help as seen below



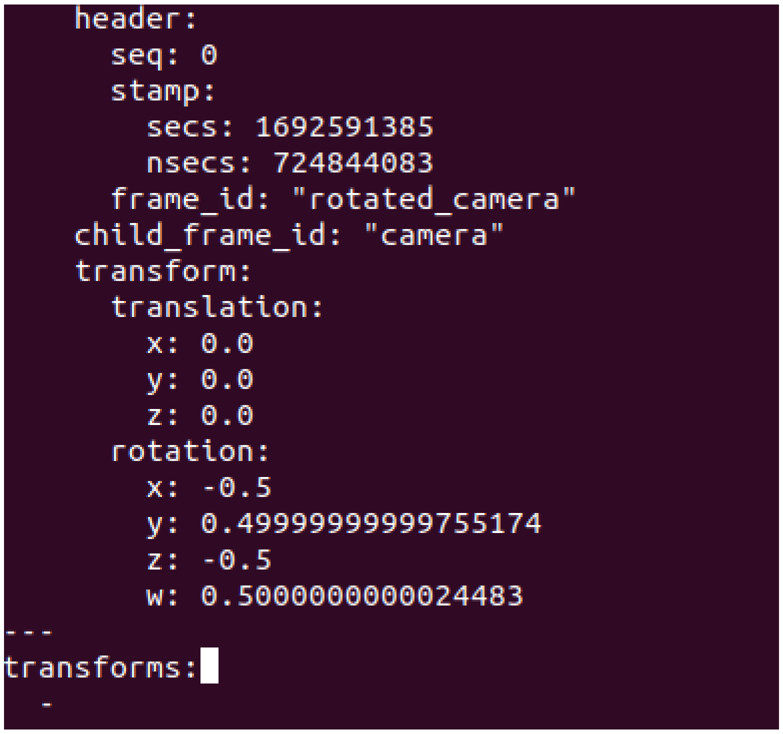
```
[ INFO] [1692590082.205560990]: [LiDAR] Searching for points in cloud of size 37
[ INFO] [1692590082.207047192]: [LiDAR] Remaining points in cloud 26
[ INFO] [1692590082.208137955]: [LiDAR] Remaining points in cloud 18
[ INFO] [1692590082.208694513]: [LiDAR] Remaining points in cloud 10
[ INFO] [1692590082.209016115]: [LiDAR] Remaining points in cloud 2
4 centers found. Iterating over 1 possible sets of candidates
[ INFO] [1692590082.209113060]: [LiDAR] 1/1 frames: 4 pts in cloud
4 clusters found from 4 points in cloud
[ INFO] [1692590082.209239297]: [LiDAR] Pattern centers published
[ INFO] [1692590082.300141175]: [LiDAR] Processing cloud...
Rot. vector:   0.11868
-0.992933
        0 / Angle: 1.86076
[ INFO] [1692590082.319247171]: [LiDAR] Searching for points in cloud of size 32
[ INFO] [1692590082.320736448]: [LiDAR] Remaining points in cloud 22
[ INFO] [1692590082.321634778]: [LiDAR] Remaining points in cloud 14
```

The calibration node does not exit on it's own and user needs to wait till sensor1 pattern and sensor2 pattern are done as shown below



```
process[velo2cam_calibration-1]: started with pid [23813]
Please, adjust the filters for each sensor before the calibration starts.
Clusters from camera: 10/10
Clusters from camera received. Is the warmup done? [Y/n]
y
Filters for sensor 1 are adjusted now. Please, proceed with the other sensor.
Clusters from velodyne: 10/10
Clusters from velodyne received. Is the warmup done? (you can also reset this position) [Y/n/r]
y
Warmup phase completed. Starting calibration phase.
[ INFO] [1692590225.252097819]: sensor2 (velodyne) pattern ready!
[ INFO] [1692590225.253074155]: [V2C] sensor2
c0=[0.693327 0.18249 0.116018]
c1=[0.725779 -0.0784195 0.121275]
c2=[0.664793 -0.0957077 -0.0889289]
c3=[0.631425 0.167902 -0.0961643]
[ INFO] [1692590225.340814228]: sensor1 (camera) pattern ready!
```

The **final output is stored as a launch file which publishes a static transform** between Lidar and Camera. The output from the above testing is seen below. From manual approximation of the extrinsics, the results look clearly wrong.



```
    header:
      seq: 0
      stamp:
        secs: 1692591385
        nsecs: 724844083
      frame_id: "rotated_camera"
    child_frame_id: "camera"
    transform:
      translation:
        x: 0.0
        y: 0.0
        z: 0.0
      rotation:
        x: -0.5
        y: 0.49999999999755174
        z: -0.5
        w: 0.5000000000024483
---
transforms:
  -
```

Trial with Enabled Perimeter Check and Looser Geometric Tolerance

I set the Geometry Tolerance to 0.3 (30cm) which is probably too huge. This resulted in the node working, but the transformation to be same as above.

The angles translated to Euler angles (degrees):
`[-89.99999999971946, -0.0, -89.99999999971946]`

Inference From Above Tests

1. I'm yet to verify why the translational component is always zero
2. The Aruco tags are sensitive and the camera needs to be very close to detect them accurately
3. I'm not sure if the node exited correctly (since I had to Ctrl+C) even though it generated a transformation matrix
4. It could be worth to try using a full scale target and see if that improves anything