Calibration, Filtering, and Segmentation

Thursday, April 11, 2019 7:47 PM

Calibration is a necessary step in the context of many sensors. Here we will discuss the camera, which typically consists of at least one lens. While it would be ideal to have it operate with the perfection of a pinhole camera, real cameras use lenses. These lenses cause distortion which can look good artistically but is bad for our robot that runs on tight mathematical principles and needs as accurate picture of the world as possible. That said, we always need to calibrate our camera so that we can convert the image into the pinhole camera equivalent with a mathematical transform.

A camera experiences two kinds of distortion, radial and tangential. Radial distortion is from the lens and it gets worse as you go radially outward in the image. This is from the spherical curvature of the lens. Tangential distortion is from the orientation of objects and how they appear to the viewer vs. what they look like head-on.

A good quote from the course instructors "*To remove systematic inconsistencies you must first understand their origin*." It's a very universal saying. And of course, knowing these systemic errors brings out measurements closer to ground truth, which is true of science in general.

A procedure using opencv or cv2 to calibrate and remove camera distortion

1. Take multiple photos of a perfect chess board on a flat surface
2. Use cv2.findchessboardcorner
3. Calibrate with cv2.calibratecamera
4. Undistort with cv2.undistort

This gives you a flat pin-hole camera-like image.

Intrinsic vs Extrinsic parameters

Roboticists break up the characterization of a system into two categories. Intrinsic has to do with all of the physical properties of the system itself such as the focal length or optical center. Extrinsic has to do with things such as the origin of the camera frame in the world reference frame.

The cv2.calibratecamera function takes into consideration the intrinsic parameters and spits out an intrinsic camera matrix and distortion coefficients. Note, camera in this case is indeed an optical camera or anything operating on the same principles. The cv2.undistort function will take these outputs and produce an image accordingly.

As for the Extrinsic parameters, we have a relation between the intrinsic parameters and the extrinsic ones which is a simple matrix equation $z_c[u, v, 1] = K[RT][x_w, y_w, z_w, 1]$ where the arrays represent columns or vectors. This equation depicts the relationship between the camera's frame and the world frame. (Need to expand on this)

In any case, the extrinsic parameters show us where the camera is and lets us string together multiple images to create what we're really looking for, a **point cloud dataset**. This process is called **image registration**.

At this point the material skips over the transformation of image data into point cloud data and just gives us the point cloud data directly. Need to return to discuss said transformation.

Filtering and Segmenting

First off, we want to eliminate and noise or objects that we don't need, next we want to clear out potentially adversarial data which is data that could be confused for what we're looking for but is not.

So, as the first piece of the perception pipeline we apply some filtering and do a RANSAC segmentation.

Here's the step-by-step

1. Downsample your point cloud by applying a Voxel Grid Filter
2. Apply a Pass through Filter to isolate the table and objects
3. Perform RANSAC plane fitting to identify the table
4. Use the ExtractIndices Filter to create new point clouds containing the table and objects separated.

```python
# Import PCL module
import pcl

# Load Point Cloud file
cloud = pcl.load_XYZRGB('tabletop.pcd')


# Voxel Grid filter-------------------------
# creating voxelgrid filter object
vox = cloud.make_voxel_grid_filter()
#voxel size
LEAF_SIZE = 0.01
vox.set_leaf_size(LEAF_SIZE,LEAF_SIZE,LEAF_SIZE)
#call filter function to get downsampled point cloud
cloud_filtered = vox.filter()
filename = 'voxel_downsampled.pcd'
pcl.save(cloud_filtered, filename)

# PassThrough filter-----------------------
#note how we use a different class this time. It's not just a cloud but is a filtered cloud!
passthrough = cloud_filtered.make_passthrough_filter()

#ok now we're defining a z-axis. Apparently this package has origin at the point cloud? No
#we probably have that from before.
filter_axis = 'z'
passthrough.set_filter_field_name(filter_axis) #ah so it's just a name now?
axis_min = 0.6
axis_max = 1.1
passthrough.set_filter_limits(axis_min, axis_max)

#and again with this filter function
cloud_filtered = passthrough.filter()
filename = 'pass_through_filtered.pcd' #pcd = point cloud data
pcl.save(cloud_filtered, filename)
```

Here we're doing voxel down sampling. So essentially averaging a volume unit or voxel and using that as the point, this makes our data nice and sparse. 1cm^3 is the size used here

Here we do pass through filtering which is essentially just a cutoff filter. In this case it cuts every above and below some limits on the z axis

```python

# RANSAC plane segmentation-----------------
seg = cloud_filtered.make_segmenter()

#set the model type
seg.set_model_type(pcl.SACMODEL_PLANE)
seg.set_method_type(pcl.SAC_RANSAC)

max_distance = 0.01
seg.set_distance_threshold(max_distance)

```

RANSAC or Random Sample Consensus is a segmentation technique which uses a model fit, in this case our model is a plane and what we do is identify points in the data set that fit the equation of a plane, then once we have the plane we say how far we can deviate from it. Anything that fits the model is an inlier. (I really want to know more about this technique!) By

```
max_distance = 0.01
seg.set_distance_threshold(max_distance)

inliers, coefficients = seg.segment()

# Extract inliers----------------------------
#the extract indices filter is not really a filter but it just lets you get the point cloud
#associated with the indices of interest that you've identified.
extracted_inliers = cloud_filtered.extract(inliers, negative=True)
filename = 'extracted_inliers.pcd'
pcl.save(extracted_inliers, filename)

# Save pcd for table
# pcl.save(cloud, filename)


# Extract outliers----------------------------
outlier_filter = cloud_filtered.make_statistical_outlier_filter()
#this is where we choose the number of data points we want
outlier_filter.set_mean_k(50)
#set threshold scale factor
x = 1.0
#any point with a mean distance larger than global mean will be considered an outlier
outlier_filter.set_std_dev_mul_thresh(x)
cloud_filtered = outlier_filter.filter()

# Save pcd for tabletop objects
```

plane we say how far we can deviate from it. Anything that fits the model is an inlier. (I really want to know more about this technique!) By the way, RANSAC sometimes takes a while to converge so there's a trade-off between segmentation optimality and time efficiency.

Lastly, we use this statistical outlier filter which can filter out noise. I'm not exactly sure how though. I suppose it's an object attached to our segmented model that checks around the model? Perhaps not though.