Gap Finding Algorithm

CS 519 / ECE 599

Insuk Joh

Part 1. Data Pre-processing

```
float[ ] shrinkData(bigdata[ ], sizeNewData)
{
    i_incr = sizeBigData / sizeNewData
    i_start = modulo (sizeBigDta, sizeNewData)
    for i = 0: sizeNewData
    newData[i] = bigData[i_start + i * i_incr]
    return newData[ ]
}

float[ ]  LidarToAngle(Lidar[ ])
{
    for i = 0: size(Lidar[ ])
    angle[i] = min_angle + i * incr_angle
    return angle [ ]
}

float[ ] polarToXCoord(distance[ ], angle[ ])
{
    for i = 0: size(distance[ ])
    xCoord[i] = distance[i] * cos(angle[i])
    return xCoord[ ]
}

float[ ] polarToYCoord(distance[ ], angle[ ])
{
    for i = 0: size(distance[ ])
    yCoord[i] = distance[i] * sin(angle[i])
    return yCoord[ ]
}
```

Part 2. K-Means Clustering

```
void k-means clustering(data[ ], numOfClusters)
{
    Step 1. Randomly assign cluster to all data points.
    Step 2. Iterate until the assigned cluster number changes
        (a) Compute the centroid(=average of x,y values) of each cluster
        (b) Assign cluster to each data whose centroid is closest.
```

Part 3. Gap detection

```
bool isGap(data[ ], cluster)
{
    max_depth = maximum distance (polar coord) to the centroid of cluster
        if depthOfCluster[k] > 0.5 * maxDepth
            cluster[k] is cluster
}

Int getNumberOfCluster(data[ ])
{
```

```
    Count number of valid gaps using isGap()
}

float getWidth(data[ ], cluster)
{
    If (centroid is close to y axis)
        Width = maxXInCluster – minXInCluster
    Elseif (centroid is close to x axis)
        Width = maxYInCluster – minYInCluster
}

float getDepth(data[ ], cluster)
{
    If (centroid is close to y axis)
        Width = maxYInCluster – minYInCluster
    Elseif (centroid is close to x axis)
        Width = maxXInCluster – minXInCluster
}
```

The gap finding algorithm consists of three parts: data pre-process, K-mean Clustering, and gap detection. First, the data pre-processing part converts the raw data from Lidar into angle, distance, x-coordinate, and y-coordinate. It also gets certain number of sampled data from the raw data, because computation over the entire data would cause computational delay, causing the algorithm inaccurate. Second, the k-means clustering part makes groups of data points to represent the walls or obstacles. In the gap finding algorithm, the weighted k-means clustering is applied over the polar coordinate, with weight on the distance variable. The weight causes the cluster to be formed over the distance than the angle. Third, the gap detection part finds how to recognize a cluster as a gap and calculates the number of gaps, the width, and depth.