Zachary Brock
F1/10 Autonomous Racing
Spring 2020

Gap Finding Algorithm Explanation

The following pseudocode illustrates the function of my gap finding algorithm.

```
Function findGapInLidarScan(scan)
      preprocessedScan = preprocessScan(scan)
      lidarGaps = findLidarGaps(preprocessedScan)
      Publish lidarGaps
      Find widest gap in lidarGaps
      Publish widest gap

Function preprocessScan(scan)
      Initialize empty array preprocessedScan
      Truncate preprocessedScan to only truncatedAngle ahead of vehicle
      For every range in scan:
            If range is NaN or less than minimumRange:
                  Append 0 to array
            Else if range is greater than maximumRange or is Inf:
                  Append maximumRange to array
            Else:
                  Append range to array
      Return preprocessedScan

Function findLidarGaps(preprocScan)
      start, index, size = 0
      Initialize empty gaps array
      While index is within length of preprocScan:
            start = index
            range = preprocScan(index)
            While range > 0 and currentIdx+1 within preprocScan:
                  next_range = preprocScan(index+1)
                  delta = absolute value of range - next_range
                  If delta > range*maxPercentDifference, break
                  Increment index, size
            If size > 1:
                  gap = calculateGap(start,start+size,preprocScan)
                  Append gap to gaps
            Increment index
      Return gaps

Function calculateGap(start,end,preprocScan)
      Calculate gapAngle between start and end indices
      a = preprocScan(start), b = preprocScan(end)
      gapWidth = sideFromLawOfCosines(a,b,gapAngle)
      beta = angleFromLawOfCosines(a,b,gapWidth)
      midline = sideFromLawOfCosines(gapWidth/2,a,beta)
      phi = angleFromLawOfCosines(a,midline,gapWidth/2)
      Use phi to identify length, heading of midpoint vector projected to
       preprocScan
      Calculate gapPosition by transforming length, heading to x-y
       coordinates
      Return gapPosition, gapWidth
```

At a high level, the algorithm functions as detailed in lines 1-6: First, the incoming scan message is preprocessed by removing NaN and Inf values and restricting the ranges within the scan to be within minimumRange and maximumRange, as shown in lines 8-18. Values less than minimumRange are turned to zero and values greater than maximumRange are turned to maximumRange. The scan is also Next, gaps in the scan message are identified by determining if successive range measurements are close enough in value to each other to be considered part of the same continuous gap, as shown in lines 20-35. The method for determining whether these ranges are close enough is given on line 29, where the parameter maxPercentDifference is multiplied by the given range to determine the maximum value permissible before a point is deemed to be part of a different gap. Using a percentage modifier helps prevent issues where points that are further away along long, straight obstacles are identified as other gaps which would be present if a constant comparator were used. Once the gap has been identified, the gap's width and center are calculated using the calculateGap function in lines 37-48. This method uses the Law of Cosines to obtain the length of the line between the extreme ranges as identified in the findLidarGaps function and then determine the angle to the midpoint of this line, as shown in Figure 1 below. The angle is then used to project the midline out to the gap measured by the scan message and identify its location in space by transforming into the scan's reference frame.

The important user-set parameters are minimumRange, maximumRange, maxPercentDifference, and truncatedAngle. These parameters and their effect are illustrated in Figure 2 below. minimumRange and maximumRange define the minimum and maximum cut-off for the ranges in the scan message. If minimumRange is reduced, more space close to the car is observed, but there is a chance that the robot will not be able to identify "deeper" gaps if large obstacles are close to the car. If minimumRange is increased, the likelihood that the car will be distracted by close obstacles is reduced, but potential paths close to the car may be missed. If maximumRange is increased, the car can look further ahead for different gaps, but may miss better gaps closer to the vehicle. If maximumRange is reduced, the vehicle may not be able to respond quickly enough to changes down the road. The parameter maxPercentDifference helps determine the maximum range delta between consecutive range values by multiplication with a given range value. If maxPercentDifference is increased, the likelihood that noisy data and ranges measured nearly parallel to a wall will be identified as separate gaps is reduced, but the resolution ability to identify smaller gaps is also reduced. If maxPercentDifference is reduced, these issues are mitigated at the expense of increased susceptibility to noisy gap center identification. The parameter truncatedAngle determines what angular range in front of the vehicle is considered for gap identification. If truncatedAngle is reduced, the vehicle will consider a smaller range which may help increase the vehicle's ability to actually navigate toward identified gaps; however, this is at the expense of potentially missing gaps to the side of the vehicle, which could be a problem if the vehicle is facing a large obstacle. If truncatedAngle is increased, these effects are reversed, at the expense of potentially increasing the noise of the widest gap signal to the controller if large gaps are identified on the sides.