RBE550: Motion Planning

Project 1

Student 1: Sumukh, Porwal
Student 2: Krutika, Muley

Theoretical Questions

1. Consider $\mathcal{A}$, a unit disc centered at the origin in the workspace $\mathcal{W} = \mathbb{R}^2$. Suppose $\mathcal{A}$ is described by the algebraic primitive $H = (x, y)|x^2 + y^2 \leq 1$. Demonstrate that rotating this primitive about the origin does not alter its representation. To prove this, show that any point within the rotated primitive *H´* is also within *H*, and vice versa.

   Solution:

   To prove this, we will show that any point within the rotated primitive *H'* is also within *H*, and vice versa.

   Let the rotation be by an angle θ about the origin. The coordinates of a point (x', y') in the rotated frame are related to the original coordinates (x, y) by:

   $$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

   This gives:

   $$x' = x\cos\theta - y\sin\theta$$
   $$y' = x\sin\theta + y\cos\theta$$

   Now, consider any point (x', y') within *H'*. And:

   $$(x')^2 + (y')^2 \leq 1$$

   Substituting the expressions for x' and y' in terms of x and y:

   $$(x\cos\theta - y\sin\theta)^2 + (x\sin\theta + y\cos\theta)^2 \leq 1$$

   Expanding the terms, we get:

   $$x^2\cos^2\theta - 2xy\cos\theta\sin\theta + y^2\sin^2\theta + x^2\sin^2\theta + 2xy\sin\theta\cos\theta + y^2\cos^2\theta \leq 1$$

   Simplifying using the identity $\cos^2\theta + \sin^2\theta = 1$:

   $$x^2(\cos^2\theta + \sin^2\theta) + y^2(\cos^2\theta + \sin^2\theta) \leq 1$$
   $$x^2 + y^2 \leq 1$$

   This is the condition for the original primitive *H*. Thus, any point (x', y') in *H'* satisfies the condition for being in *H*. By the same argument, any point in *H* satisfies the condition for being in *H'*.

   Therefore, rotating the primitive *H* about the origin does not alter its representation, as *H* = *H'*.

2. Let S be a square object parameterized by (x, y, θ, s), where (x, y) denotes its position in the X-Y plane, θ represents its rotation relative to its center, and s is the length of its sides. Given a random point $p_r = (x_r, y_r)$, provide a pseudocode algorithm to determine if p_r is inside the square. The function should return `False` if the point is inside or on the edge of the square, and `True` otherwise. Ensure to account for all edge cases in your solution.

   Solution:

Algorithm 1 Check if Point is Inside a Rotated Square

function IsPointInsideSquare($x, y, \theta, s, x_r, y_r$)
  Rotate the point p_r by $-\theta$:
  rotateX = $(x_r - x)\cos(\theta) + (y_r - y)\sin(\theta)$
  rotateY = $-(x_r - x)\sin(\theta) + (y_r - y)\cos(\theta)$
  Define the bounds of the axis-aligned square:
  half_side = $s/2$
  left = $-$half_side
  right = half_side
  bottom = $-$half_side
  top = half_side
  Check if the rotated point is inside the axis-aligned square:
  if rotateX $\geq$ left **and** rotateX $\leq$ right **and** rotateY $\geq$ bottom **and** rotateY $\leq$ top **then**
    return `False` ▷ Point is inside or on the edge of the square
  else
    return `True` ▷ Point is outside the square

Programming Component

1. Implement a sampler that generates uniformly distributed points on a disk in $\mathbb{R}^2$ space using OMPL. Complete the following tasks:

   - Implement the `sampleNaive(ob::State *state)` function in `DiskSampler.cpp`. Use the following process to perform naive sampling on a disk with a radius of 10:
     (a) Sample random polar coordinates $r \sim [0, 10]$ and $\theta \sim [0, 2\pi)$.
     (b) Convert the polar coordinates to Cartesian coordinates (x, y) in $\mathbb{R}^2$.

     Evaluate whether these points are uniformly distributed on the disk. Explain why they may not be uniformly distributed and include the `naive_samples.png` figure in your report.
   - Implement the `sampleCorrect(ob::State *state)` function in `DiskSampler.cpp`. Use a correct sampling process to generate uniformly random points on the disk. (Uniform here means samples drawn from a uniform distribution over the area of the disk, not a grid pattern). Many methods can achieve this, but the most elegant solution involves a single code change. Describe what you changed and why, and include the `correct_samples.png` figure in your report.

   Solution:

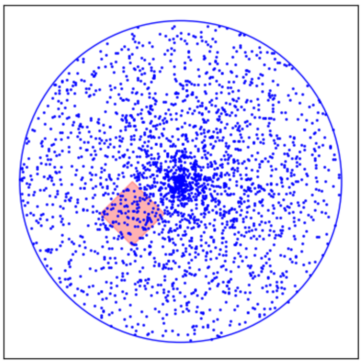

Figure 1: Sampling implementation using `sampleNaive(ob::State *state)` function

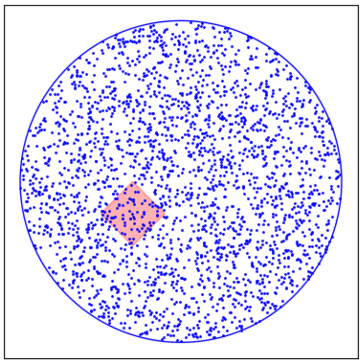

Figure 2: Sampling implementation using `sampleCorrect(ob::State *state)` function

In the above Figure 1 and Figure 2 the number of states plotted are 2500 and 2500 respectively.

In the `sampleNaive` method (Figure 1), the radius r is sampled uniformly in the range $[0, 10]$, and the angle θ is uniformly sampled in the range $[0, 2\pi)$. This approach results in a non-uniform distribution of points within the disk because sampling r uniformly means that each possible radius value is equally likely. However, the area of a disk grows with the square of the radius, i.e., πr^2. Thus, points further from the center are sampled more frequently in proportion to their area. For example, the area of the annulus between radii r and $r + \delta r$ increases with r, leading to a higher density of points in the outer regions of the disk compared to the center.

To correct the non-uniform distribution, the `sampleCorrect` method (Figure 2) adjusts the way the radius r is sampled. The change involves sampling r^2 uniformly instead of r. Instead of sampling r uniformly from $[0, 10]$, we sample r^2 uniformly from $[0, 100]$. By sampling r^2 uniformly, we ensure that the probability density function for r is proportional to r. This adjustment ensures that points are uniformly distributed over the area of the disk.

2. Implement a collision checker for a point and a translated and rotated square using the algorithm proposed in Theoretical Question 2. Complete the following task:

   - Implement the `isValid(ob::State *state)` function in `DiskSampler.cpp`. This function should check for collisions with a square obstacle of edge size $2 * \sqrt{2}$, located at $(-3, -2)$ and rotated by $\pi/4$. Use the visualize.py script to verify the correctness of your implementation. Include the final figure produced by the visualization script in your report.

   Solution:

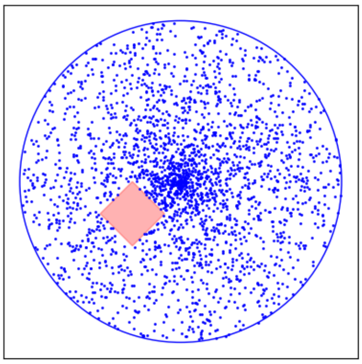

Figure 3: Sampling implementation using `sampleNaive(ob::State *state)` function

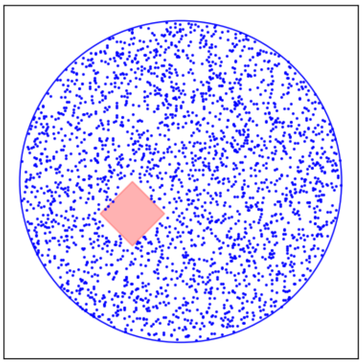

Figure 4: Sampling implementation using `sampleCorrect(ob::State *state)` function

Figure 5: Figures with `isValid(ob::State *state)` implemented

In the above Figure 3 and Figure 4 the number of states plotted are 2498 and 2499 respectively.

Here, in the above image (Figure 5), the obstacle square is places at $(-3, -2)$ with edge size $2 * \sqrt{2}$ and rotated by $\pi/4$ and all the points lying inside or at the edge of the square are not considered valid and `isValid(ob::State *state)` function return `true` for those points and they aren't plotted on the disk, unlike in the Figure 1 and 2 where all the generated points were plotted on the disk.