ME 145
Robotic Planning and Kinematics

Lab Session No. 1

LINES AND SEGMENTS

Instructions

Submit your code through iLearn (or as otherwise instructed). Your code and reports are due on Wednesday, October 18th , 6:00 pm. No late submissions will be accepted.

E1.6 **Programming: Lines and segments (*30 points*).**

In this exercise you are asked to begin implementing a number of basic algorithms that perform geometric computations. Consider the following planar geometry functions:

computeLineThroughTwoPoints (*10 points*)

Input: two distinct points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ on the plane.

Output: parameters (a, b, c) defining the line $\{(x, y) \mid ax + by + c = 0\}$ that passes through both p_1 and p_2. Normalize the parameters so that $a^2 + b^2 = 1$.

computeDistancePointToLine (*10 points*)

Input: a point q and two distinct points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ defining a line.

Output: the distance from q to the line defined by p_1 and p_2.

computeDistancePointToSegment (*10 points*)

Input: a point q and a segment defined by two distinct points (p_1, p_2).

Output: the distance from q to the segment with extreme points (p_1, p_2).

For each function, do the following:

- explain how to implement the function, possibly deriving analytic formulas, and characterize special cases,
- program the function, including correctness checks on the input data and appropriate error messages, and
- verify your function is correct on a broad range of test inputs (at least 5 examples per function).

Hints: *To check two points are distinct, use a tolerance equal to* 0.1. *Regarding computeDistancePointToLine, compute the orthogonal projection of* p *onto the line. Regarding computeDistancePointToSegment, the distance depends on whether or not the orthogonal projection of* p *onto the line defined by* p_1 *and* p_2 *belongs or not to the segment between* p_1 *and* p_2.

E1.7 **Programming: Polygons (*30 points*).**

Requires Exercise (E1.6).

A polygon with n vertices is represented as an array with n rows and 2 columns. Consider the following functions:

computeDistancePointToPolygon (*15 points*)

Input: a polygon P and a point q.

Output: the distance from q to the closest point in P , called the distance from q to the polygon.

computeTangentVectorToPolygon (*15 points*)

Input: a polygon P and a point q.

Output: the unit-length vector u tangent at point q to the polygon P in the following sense: (i) if q is closest to a segment of the polygon, then u should be parallel to the segment, (ii) if q is closest to a vertex, then u should be tangent to a circle centered at the vertex that passes through q, and (iii) the tangent should lie in the counter-clockwise direction.

For each function, do the following:

- explain how to implement the function, possibly deriving analytic formulas, and characterize special cases,
- program the function, including correctness checks on the input data and appropriate error messages, and
- verify your function is correct on a broad range of test inputs (at least 5 examples per function).

Hint: *Determine which segment or vertex of* P *is closest to* q *to determine whether to use the segment or vertex tangent case.*

Programming Note: It is convenient to learn to visualize points, vectors and polygons in your chosen programming environment. To visualize a red square in Matlab, one can execute the commands:

```
» mysquare = [0 0; 0 1; 1 1; 1 0];
» fill (mysquare(:,1), mysquare(:,2), ’r’);
» axis ([-0.5 1.5, -0.5 1.5]);
```