7MR10070: Software and Robotic Integration
Semester 2

Assignment 1

Written by Alexandros Megalemos

Path Planning Part 1

Task 1

Algorithm (a)

```
Point p = (x, y, z)
List of points P = [p_1, ..., p_n]
List of output points O = [o_1, ..., o_n]
For each p in p ∈ P:
      Retrieve pixel intensity c, where c ∈ [0,1]
      If c = 1 then O ← p
```

Algorithm (b)

```
Point p = (x,y,z)
Point ep is a point p representing an entry point
Point tp is a point p representing a target point
List of entry points EP = [ep_1, ..., ep_n]
List of target points TP = [tp_1, ..., tp_n]
List of ventricle points VP = [p_1, ..., p_n]
List of output points O = [(ep_1, tp_1), ..., (ep_n, tp_n)]
Line l is a line between two points
For each ep in p ∈ EP:
      For each tp in p ∈ TP:
            l = epṫp
            For each p in p ∈ epṫp
                  If does not p ∈ VP then O ← (ep, tp)
```

Algorithm (c)

```
Point p = (x,y,z)
Point ep is a point p representing an entry point
Point tp is a point p representing a target point
List of entry points EP = [ep_1, ..., ep_n]
List of target points TP = [tp_1, ..., tp_n]
List of blood vessel points BVP = [p_1, ..., p_n]
List of output points O = [(ep_1, tp_1), ..., (ep_n, tp_n)]
Line l is a line between two points
For each ep in p ∈ EP:
      For each tp in p ∈ TP:
            l = epṫp
            For each p in p ∈ epṫp
                  If does not p ∈ BVP then O ← (ep, tp)
```

Algorithm (d)

```
Point p = (x,y,z)
Point ep is a point p representing an entry point
Point tp is a point p representing a target point
List of entry points EP = [ep_1, ..., ep_n]
List of target points TP = [tp_1, ..., tp_n]
List of blood vessel points BVP = [p_1, ..., p_n]
List of output points O = [(ep_1, tp_1), ..., (ep_n, tp_n)]
Line l is a line between two points
For each ep in p ∈ EP:
    For each tp in p ∈ TP:
        l = epṫp
        l_p = ⊥ epṫp
        angle = ∠ll_p
        If angle < 55 then O ← (ep, tp)
```

Task 2

The constants included in the following algorithms represent the time taken by ***if-statements***. Perhaps they could have been omitted

Algorithm (a)

```
For each point in target points:
    Get the coordinates (x, y, z) of the point
    Retrieve the pixel value of (x, y, z) in the image
    If pixel value == 1:
        Add the point to the list
```

$Big\ O = O(N_{tp} * R_{pv})$

where N_{tp} *is the total number of target points,* R_{pv} *is the lookup time for the pixel value*

Algorithm (b)

```
For each entry point in entry points:
    For each target point in target points:
        Get the line between the two points
        Get the points of the line
        validLine = true
        For each point on the line:
            If point passes through ventricle:
                validLine = false
                break
        if validLine:
            add (entry, target) point to valid points list
```

$Big\ O = O(2 * N_{ep} * N_{tp} * N_{pl})$

where N_{ep} *is the number of entry points,* NP_{tp} *is the number of target points and* N_{Pl} *is the number of points in the line*

Algorithm (c)

```
For each entry point in entry points:
      For each target point in target points:
            Get the line between the two points
            Get the points of the line
            validLine = true
            For each point on the line:
                  If point passes through blood vessel:
                        validLine = false
                        break
            if validLine:
                  add (entry, target) point to valid points list
```

$Big\ O = O(2 * N_{ep} * N_{tp} * N_{pl})$

where N_{ep} *is the number of entry points,* NP_{tp} *is the number of target points and* N_{Pl} *is the number of points in the line*

Algorithm (d)

```
For each entry point in entry points:
      For each target point in target points:
            Get the line, lineET, between the two points
            For each point on line:
                  If point connects with the cortex:
                        Get perpendicular line where lineET passes
through the cortex
                        Calculate the angle between the two lines
                        If angle < 55:
                              Add (entry, target) point to valid points
list
```

$Big\ O = O(2 * N_{ep} * N_{tp} * N_c * T_a)$

where N_{ep} *is the number of entry points,* NP_{tp} *is the number of target points,* N_c *is the number of cortex points and* T_a *is the time needed to calculate the angle between the* $entry - target$ *line and the perpendicular line where it connects on the cortex*

Task 3

Tests were run using the full data because the test files were crashing for some reason. As in the previous tasks, constants represent the time taken by ***if-statements***.

Algorithm (a)

$Big\ O = O(N_{tp} * R_{pv})$

$where\ N_{tp}\ is\ the\ total\ number\ of\ target\ points, R_{pv}\ is\ the\ lookup\ time\ for\ the\ pixel\ value$

Time taken: 0.001s (average of 10 runs)

Rejected trajectories: 78336

Algorithm (b)

$Big\ O = O\left(2 * N_{ep} * N_{tp} * log(N_{vp}) + N_{vp} log(N_{vp})\right)$

$where\ N_{ep}\ is\ the\ number\ of\ entry\ points, NP_{tp}\ is\ the\ number\ of\ target\ points\ and\ N_{vp}\ is\ the\ number\ of\ ventricle\ points$

Changes made to the algorithm: Instead of traversing through all the points in the line between the entry and target point, we instead search using a tree whose lookup time should be $log(N_{vp})$ instead of N_{vp}. However, we'll need to create the tree (once), and that should take $N_{vp} log(N_{vp})$.

Time taken: 3s (average of 10 runs)

Rejected trajectories: 10205

Algorithm (c)

$Big\ O = O\left(2 * N_{ep} * N_{tp} * log(N_{bvp}) + N_{bvp} log(N_{bvp})\right)$

$where\ N_{ep}\ is\ the\ number\ of\ entry\ points, NP_{tp}\ is\ the\ number\ of\ target\ points\ and\ N_{bvp}\ is\ the\ number\ of\ blood\ vessel\ points$

Changes made to the algorithm: Instead of traversing all the points in the line between the entry and target point, we instead search using a tree whose lookup time should be $log(N_{bvp})$ instead of N_{bvp}. However, we'll need to create the tree (once), and that should take $N_{vp} log(N_{bvp})$.

Time taken: 60s (average of 10 runs)

Rejected trajectories: 47855

Algorithm (d)

$$Big\ O = O\left(2 * N_{ep} * N_{tp} * log(N_{cp}) * T_a + N_{cp} log(N_{cp})\right)$$

where N_{ep} *is the number of entry points,* NP_{tp} *is the number of target points and* N_{cp} *is the number of points cortex points and* T_a *is the time needed to calculate the angle between two vectors*

Changes made to the algorithm: Instead of traversing all the points in the line between the entry and target point, we instead search using a tree whose lookup time should be $log(N_{cp})$ instead of N_{cp}. However, we'll need to create the tree (once), and that should take $N_{cp} log(N_{cp})$.

Time taken: 105s (average of 10 runs)

Rejected trajectories: 17426

Task 4

Here we combine all three constrains under the same nested for loop (between entry and target points). We start with the least complex (in terms of time and space) algorithm and work our way down to the most complex. Therefore:

1. First, create the OBB trees required for each task
2. Then, filter for targets that are within the hippocampus
3. After that, filter for entry/target trajectories that do not pass through the ventricles
4. After that, we filter for entry/target trajectories that do not pass through blood vessels
5. Finally, we filter so that only trajectories of a certain angle (degrees) are accepted

By doing this, we rule out most of the trajectories before we reach the more expensive checks in our overall algorithm. This is clearly demonstrated by the total time taken of ~25 seconds, whereas before filtering just for valid angles would take more than 100 seconds.

Time taken: 25s (average of 10 runs)

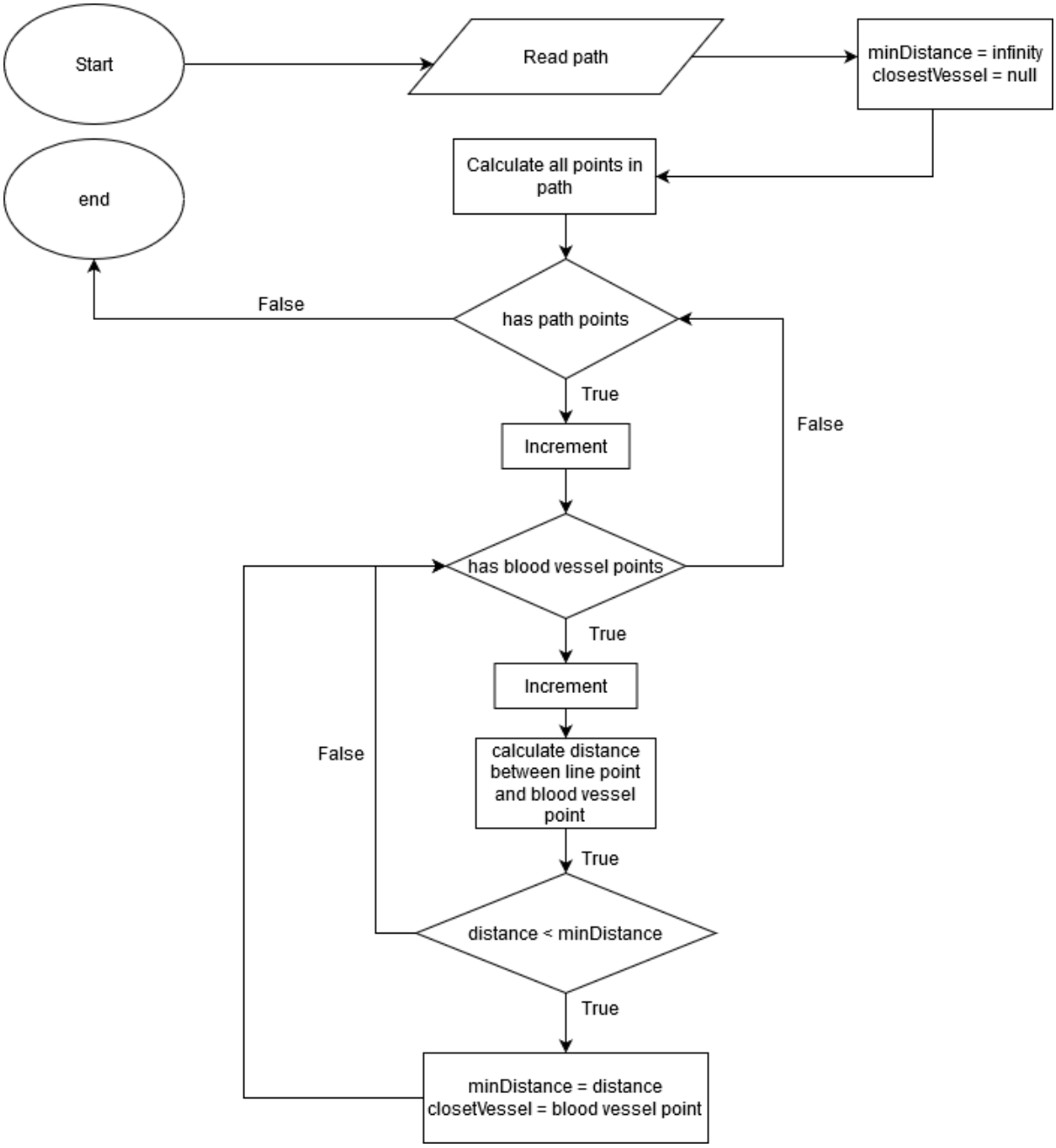
Rejected trajectories: 88603

Path Planning Part 2

Task 1

(a) A naïve/brute-force solution to the problem. It iterates over all possible points and finds the minimum distance between a vessel and a point in the path

```
minDistance = +infinity
closestVessel = null
allPoints = getAllPointsInPath(entry, target)
for each point in allPoints:
     for each bloodVesselPoint:
          Calculate distance (e.g. Euclidean) between the line
point and blood vessel point
          If distance < minDistance:
               minDistance = distance
               closestVessel = bloodVesselPoint
```



1:simplified flowchart of the algorithm

(b) In order to test that my code works, I used a subset of the data outputted by combining all the hard constraints. I visually chose a trajectory with obviously large distance from the blood vessels and another one with an obviously small distance from the blood vessels. I then inspected the output and seeing that it was as expected, I started using more trajectories and larger sets of data.

Code / Git Repository

https://github.com/Meldanen/kcl/tree/master/robotics/slicerTutorials/Tutorial%202%20and%203

The tests should be able to run all code required for the assignment. If you wish you run smaller parts of the code, you can either comment out some of them, or go to *Assignment1Logic.run()* and again, comment/uncomment the parts you want to run.
The submitted version of this only runs the combination of all three constraints when using the GUI to save time.