CIS 540
Project Phase 1 Report

Ankit Mishra Nikhilesh Behera

1. **Assumptions** -
    - Source and destination locations for both the aircrafts cannot be the same to avoid initial and final collisions
    - The sources/destinations for both the aircrafts have to be separated by a minimum of 1 unit distance

2. **Input/Output** -
   Inputs -
   The controller takes the following inputs about the controlled and the other aircraft -
    - Data structure (in) containing the following parameters :
        - Current position (x,y)
        - Final destination (x,y)
        - direction of movement (theta)
        - Incoming message from the other aircraft which indeed contains its given coordinates i.e current, destination and theta

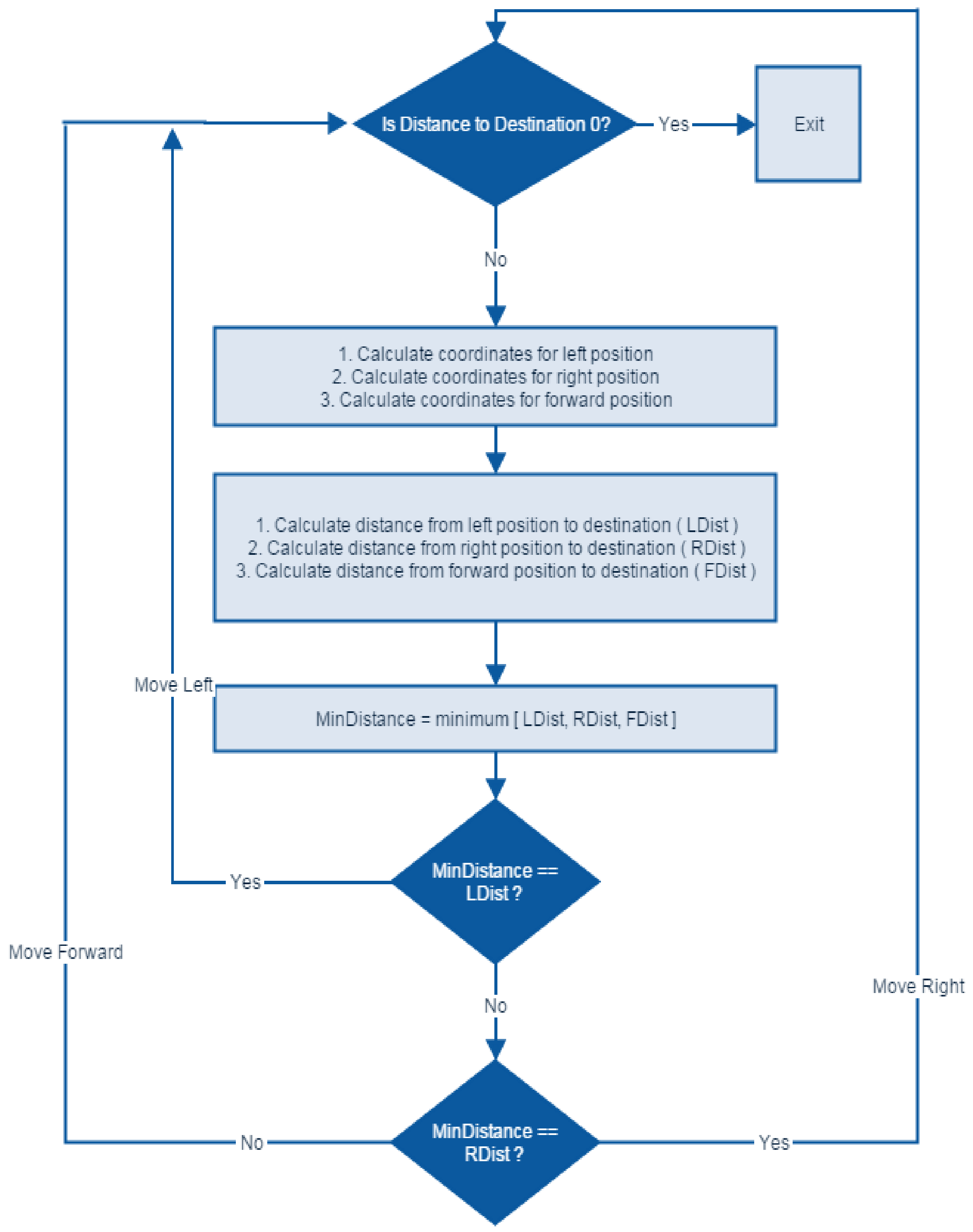
   Outputs -
   The controller generates an output value that indicates whether the aircraft should move forward, move left or move right.

3. **Specifications for the controller** -
    - Controller should guide the aircraft from source to destination. Which will include the following sub-requirements -
        - At any given instance the controller should bring the aircraft closer to the destination based on the shortest path algorithm which has been implemented
        - If the aircraft has already reached its destination then the controller should not move the aircraft
    - The controller should avoid collisions of multiple aircrafts. Which includes the following sub-requirements -
        - The controller should check for incoming messages from other aircrafts.
        - The controller should parse the incoming messages to predict next position of other aircrafts.
        - The controller should actively avoid navigating the aircraft to the predicted position of other aircrafts or their vicinity.
        - At all given times there must exist one valid next move for both the aircrafts based on collision avoidance provided the aircrafts are still in transit

4. **Simple controller** -
   The simple controller performs the job of guiding the aircraft from its source to destination as quickly as possible. To facilitate this, we uncorporate the shortest path heuristics. At any point, the controller forces the aircraft to chose the direction that is at minimum distance from the destination.

CIS 540
Project Phase 1 Report

Ankit Mishra Nikhilesh Behera


Is Distance to Destination 0?
Yes
Exit
No
1. Calculate coordinates for left position
2. Calculate coordinates for right position
3. Calculate coordinates for forward position
1. Calculate distance from left position to destination (LDist)
2. Calculate distance from right position to destination (RDist)
3. Calculate distance from forward position to destination (FDist)
MinDistance = minimum [LDist, RDist, FDist]
Move Left
MinDistance == LDist ?
Yes
No
Move Forward
Move Right
MinDistance == RDist ?
No
Yes

CIS 540
Project Phase 1 Report

Ankit Mishra Nikhilesh Behera

At every step of the execution, the following algorithm is executed -

```
in ->contains self details - in.x, in.y, in.theta, in.xd, in.yd,
if( in.x & in.y != in.xd & in.yd)           //if the aircraft has not reached its destination
{

       //based on theta, update the next position coordinates (xLeft, yLeft) (xRight, yRight)
       //(xFront, yFront)
       //here we give the case for theta = 0 or 360. The other cases will be similar.

       if(theta == 0 || theta == 360)
       {
              xLeft=in.x;
              yLeft=in.y+1
              xRight=in.x
              yRight=in.y-1
              xFront=in.x+1
              yFront=in.y

       }
       //code for calculating (xLeft, yLeft) (xRight, yRight) (xFront, yFront) for theta =90,
       180, //270 will come here and will be similar as above
       distanceLeft = (abs(in.xd-xLeft)+abs(in.yd-yLeft))
       distanceRight = (abs(in.xd-xRight)+abs(in.yd-yRight))
       distanceFront = (abs(in.xd-xFront)+abs(in.yd-yFront))

       minDistance=min(distanceLeft, distanceRight, distanceRight )
       if(minDistance==distanceLeft)
       {
              out.val=+1     //move left
       }
       else if(minDistance==distanceRight)
       {
              out.val=-1     //move right
       }
       else if(minDistance==distanceFront)
       {
              out.val=0      //move front
       }
}
else           //aircraft has reached its destination and thus it should not move from the
position
{
       out.val=2
}
```

5. **Complete controller** -

The complete controller performs the job of guiding the aircraft from its source to destination as quickly as possible **without any collisions**. To facilitate this, initially we point the aircraft in the best suited direction based on the theta value from the source to destination also we incorporate the shortest path heuristics along with collision avoidance system. The algorithms roughly, performs the following steps in each round -

- If we receive an incoming message then we predict the 3 possible next coordinates of the other aircraft based on the incoming message and block those coordinates for the aircraft. **Addition -** Blocking a direction means that the aircraft cannot take that step even if that step is the most optimal step. So when the aircraft receives a message from the other aircraft, it predicts the 3 possible next steps that the other aircraft can take. The aircraft then actively avoids all three predicted points, unless both aircrafts have different first preferences. Further, while choosing the next position of the aircraft the controller also checks if the two aircrafts are exchanging positions, if yes then such next positions are also blocked. This avoids collision as an aircraft will never go to a coordinate point, if that point is a potential next position of the other aircraft or results in a position exchange.
- Then the controller calculates the step distance from all four next possible positions and sorts them out in ascending order of the step distance. Here, if the the step distances for any 2 directions are equal then the tie is broken by calculating the euclidean distance from that point to the destination and choosing that direction that minimizes the euclidean distance. Further, if the euclidean distance is also same for 2 directions then we chose the next step direction based on ideal heading direction.
- The ideal heading direction is defined by assessing the current heading and the placement of the destination coordinates from the aircraft's current coordinates, as follows -

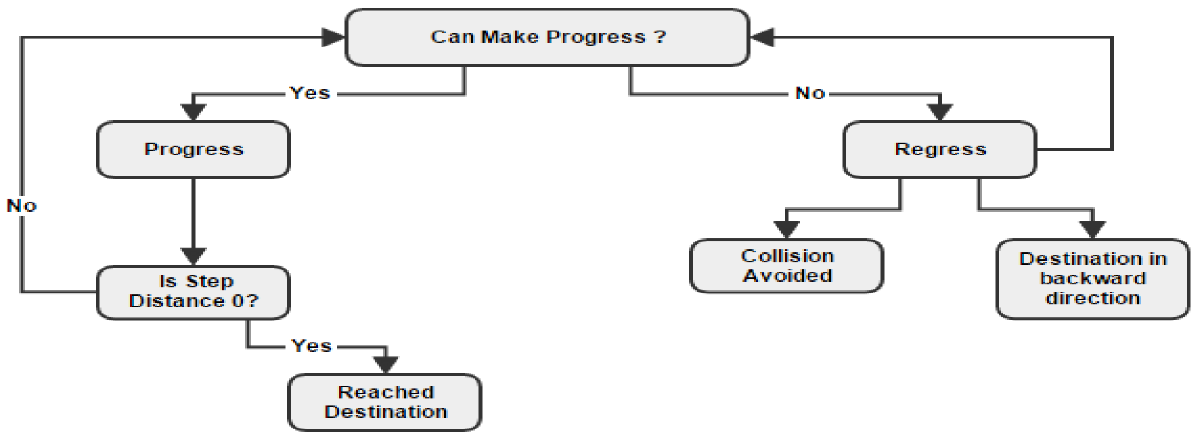
Quadrant of placement of (xd,yd) as compared to (x,y)	**Theta (Degrees)**	**Preference order**
Quadrant 1	0	1. Left 2. Forward 3. Right
Quadrant 1	90	1. Right 2. Forward 3. Left
Quadrant 2	90	1. Left 2. Forward 3. Right
Quadrant 2	180	1. Right 2. Forward 3. Left

Quadrant 3	180	1. Left 2. Forward 3. Right
Quadrant 3	270	1. Right 2. Forward 3. Left
Quadrant 4	270	1. Left 2. Forward 3. Right
Quadrant 4	0	1. Right 2. Forward 3. Left

- In the above step we also calculate the distance for backward motion. Since the aircraft cannot take a backward step, this is then converted into either a left step or a Right step based on which direction reduces the step distance to the destination while not being blocked by the other aircraft.
- Then the controller forces the aircraft to chose the direction that is at minimum distance from the destination, from the directions which are not blocked in the previous steps.
- The algorithm gives higher priority to the aircraft that is 1 unit away from the destination and also collision avoidance is not considered in this case as the aircraft closest to the destination is given higher preference whereas the other aircraft avoids those given coordinates
- Once the aircraft has reached its destination, then the aircraft coordinates are not changed.

Ankit Mishra Nikhilesh Behera

6. Proof of Correctness -



- ***OverView -***
  To prove the correctness of our controller we consider 2 goals i.e. reaching the destination and avoiding collision in transit. To prove the correctness of first goal, we argue the case of non colliding path and colliding path. For non colliding path the proof constitutes proving correctness of our shortest path algorithm in which the aircraft reaches its destination eventually. For the colliding path case we argue that the aircraft can either make progress at each step or make regress. If the aircraft makes progress at all steps then it converges to the non colliding case of shortest path algorithm. Further, if the aircraft makes regress at any given step then we argue that the aircraft will not continue making regress to avoid collision, which implies that the aircraft will make progress and reach its destination. Lastly, we argue that an aircraft will not cycle between regress and progress. A cycle is formed when the aircraft repeats the same decision at every given coordinate point in the cycle. Further, at the point of beginning of a cycle, the value of theta for the aircraft will not be the same when it comes back to the same point. Since, our controller takes decisions based on the **theta** value, it will result in the controller making a different decision as compared to the first time it arrived on the same position. This happens as our controller changes direction preferences based on the current theta value and the optimal direction of heading, causing the aircraft to not form a physical cycle in its path and since our algorithm is a greedy algorithm, it also prevents a regress-progress cycle.
  We prove the correctness of the second goal i.e. the controller avoids collision, as the controller does not allow the aircraft to move to a point where the other aircraft can potentially move, thus avoiding collision.

CIS 540
Project Phase 1 Report

Ankit Mishra Nikhilesh Behera

- ***Definitions*** -
    - **Step Distance** - Total number of steps required by the aircraft to reach the destination
    - **Euclidean Distance** - Euclidean distance between the source and destination
    - **Progress** - An aircraft makes progress when the step distance from the current point to the destination is lesser than the step distance from the previous point to the destination
    - **Regress** - An aircraft makes regress when the step distance from the current point to the destination is more than the step distance from the previous point to the destination

- ***Claim-1*** - If an aircraft makes progress at all given times then the aircraft is guaranteed to reach the destination
    - If the aircraft makes progress at every given step then, as per the definition of progress, the aircraft step distance reduces at every step by at least a count of 1 (the step count is an integer variable).
    - Thus eventually the distance becomes 0 and the aircraft reaches its destination.

- ***Claim-2*** - The algorithm ensures that the aircraft doesn't make regress at every step
    - In any given round, the aircraft will make regress to either avoid a collision or if the destination is the the opposite direction as the aircrafts direction.
    - Case 1 - The destination is the the opposite direction as the aircrafts direction.
        - If the destination is in the opposite direction as the aircrafts direction(theta) then the aircraft will make regress if it takes a step in the forward direction.
        - But the algorithm is designed to detect such a case and give preference to either Left or Right steps, based on their step difference from the destination.
        - Thus, If there is no collision avoidance then the aircraft will never take a forward step when the destination direction is opposite to the aircrafts direction.
    - **Case 2** - To avoid a collision
        - If the aircraft makes regress to avoid a collision then there will be 2 sub cases based on the state of the other aircraft.
            - **Sub-Case 1** - Other aircraft is repeatedly making progress
                - When an aircraft makes regress to avoid collision such that the other aircraft is making progress the we know that the other aircraft will eventually reach its destination based on the **Claim-1**.
            - **Sub-Case 2** - Other aircraft also makes regress

        - When an aircraft makes regress to avoid collision and the other aircraft also makes regress then we show that eventually both aircrafts will stop blocking each other.
        - **Assumption 1 -** Assume that both the aircrafts keep blocking each other for ever.
        - At any given step, if the aircrafts block each other then, they are both likely to make regress.
        - But if an aircraft keeps on making regress then, eventually the destination direction keeps on becoming closer to the opposite direction to the aircrafts direction.
        - This makes that the ideal heading will be in the backward direction and as per the Case 1, we know that in such a scenario the aircraft will give preference to moving left or right over keep on moving in the forward direction.
        - In such a case, the algorithm moves in the direction (left or right), which is away from the other aircraft.
        - This violates the **Assumption 1** and proves our claim for **Sub-Case 2**.

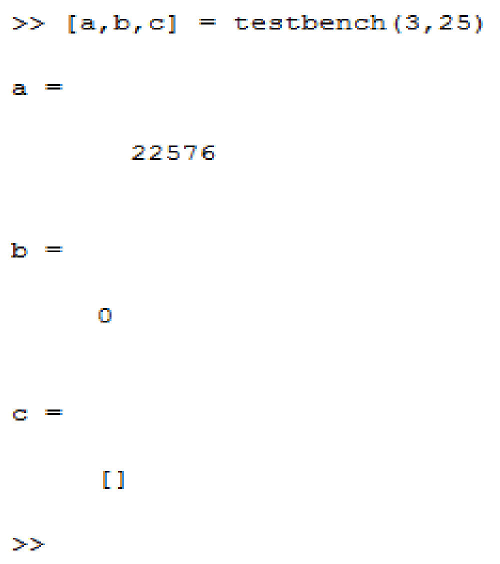
- ***Claim-3*** - The algorithm ensures that the aircraft can make progress at every step
    - Algorithm is based on the shortest path heuristic and at every round it attempts to reduce the step distance for the aircraft.
    - Based on the proof for claim-1 we can guarantee that the algorithm ensures that the aircraft can make progress at every step.

- ***Claim-4*** - Ensuring that two aircrafts do not collide
    - Based on the algorithm, a given a aircraft blocks all the possible positions for the other aircraft in the subsequent round. In this way it is guaranteed that the two aircrafts cannot collide at any given round. Explanation added in the algorithm mentioned above.

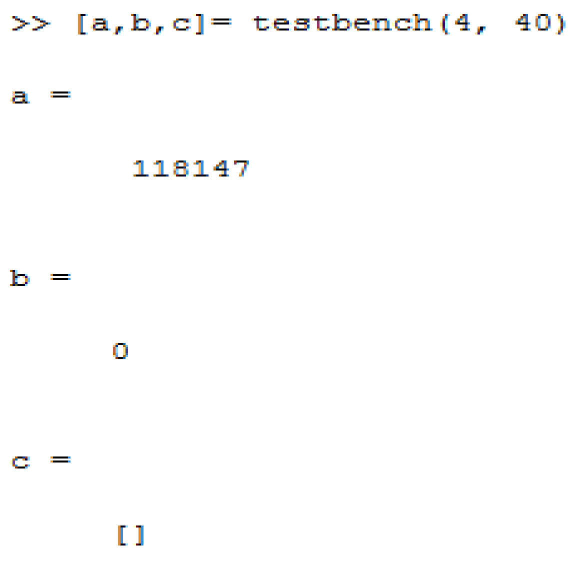
CIS 540
Project Phase 1 Report

Ankit Mishra Nikhilesh Behera

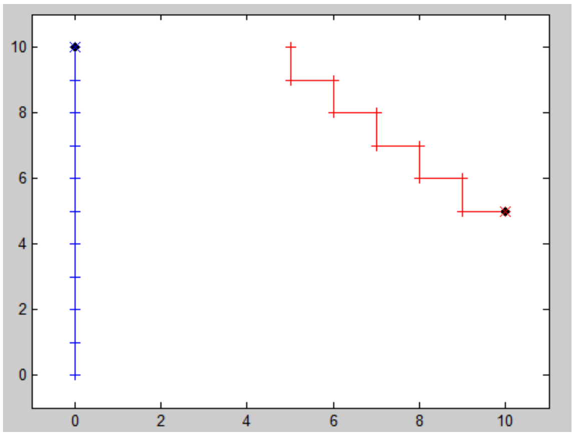
7. Results -



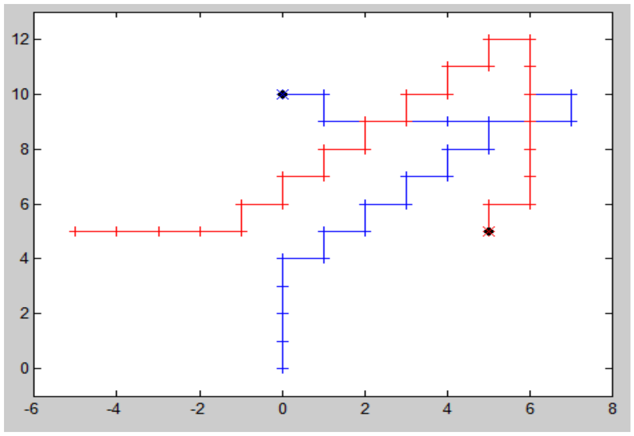
```
>> [a,b,c] = testbench(3,25)

a =

       22576


b =

     0


c =

     []

>>
```



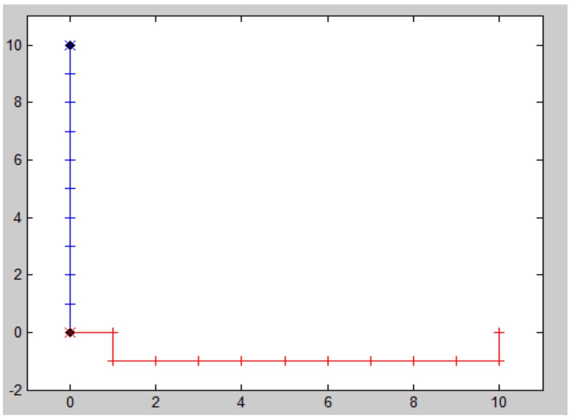
```
>> [a,b,c]= testbench(4, 40)

a =

      118147


b =

     0


c =

     []
```



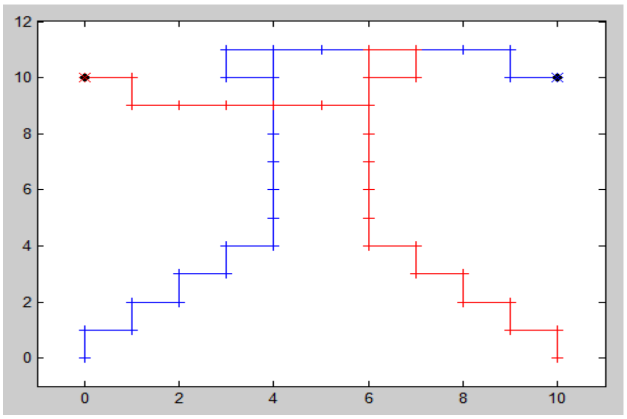
runSimulation([0, 0], [0, 10], [5, 10], [10, 5], 40)



runSimulation([0, 0], [0, 10], [-5, 5], [5, 5], 40)



runSimulation([0, 0], [0, 10], [10, 0], [0, 0], 40)



runSimulation([0, 0], [10, 10], [10, 0], [0, 10], 40)