**Initialization**: *Input1*: Obstacles: Boundary points of α-complex shape (i.e. boundary co-ordinates of obstacles)

*Input2*: Co-ordinates of working sensor nodes (i.e. nodes outside the obstacles covered area)

/\* Computes Euclidean Minimum Steiner tree with Obstacle(s) \*/

COMPUTE-GSMT(*Input1*, *Input2*)

CANDIDATE\_SET ← COMPUTE-CANDIDATE-SET(*Input1*, *Input2*)

Initialize MAX\_POINT to a positive number

while MAX\_POINT ≥ 0

COST = 0

Assign MAX\_POINT a negative value

for each point p ∈ CANDIDATE\_SET

DeltaCost ← DELTA-MST(*Input2*+ SteinerPoints, p)

if DeltaCost > COST

COST ← DeltaCost

if MAX\_POINT ≥ 0

SteinerPoints = SteinerPoints ∪ {MAX\_POINT}

for each point p ∈ SteinerPoints

if Degree ≤ 2

SteinerPoints = SteinerPoints - {p}

FINAL\_GSMT ← KRUSKAL(*Input2* + SteinerPoints)

DISTANCE\_GSMT ← **Σ** Edges.distance of FINAL\_GSMT

return FINAL\_GSMT

/\* Computes Candidate set of points which can be used as Steiner Points required for COMPUTE-GSMT \*/

COMPUTE-CANDIDATE-SET(*Input1*, *Input2*)

CANDIDATE\_SET ← {}

ConvexHull = Convex\_Hull(*Input2*)

for each point p ∈ Points inside ConvexHull

if point p ∉ Points within Obstacles

CANDIDATE\_SET = CANDIDATE\_SET ∪ {p }

return CANDIDATE\_SET

/\* Computes difference between costs of two MST’s one before and one after adding SteinerPoint \*/

DELTA-MST(SetOfPoints, TestPoint)

MST\_ONE ← KRUSKAL-MODIFIED(SetofPoints, Obstacles)

MST\_TWO ← KRUSKAL-MODIFIED(SetOfPoints + TestPoints, Obstacles)

COST1 ← **Σ** Edges.distance of MST\_ONE

COST2 ← **Σ** Edges.distance of MST\_TWO

return COST1 – COST2

/\*Computes cost of MST; if edge intersects Obstacle(s) then distance is set to infinity \*/

KRUSKAL-MODIFIED(SetOfPoints, Obstacles)

Edges ← {}

for each point p ∈ SetOfPoints

for each point p’ ∈ SetOfPoints

if edge(p, p’) intersects Obstacles

Distance ← ∞

else

Distance ← CALCULATE-DISTANCE (p, p’)

Edges.add(p, p’, Distance)

Sort(Edges, key = Distance)

MST ← {}

while MST does not contains (SetOfPoints – 1) edges and MST does not connect all SetOfPoints

for each edge ∈ Edges

if edge(p, p’) does not create cycle

MST = MST ∪ {edge(p, p’)}

return MST

/\* Computes Rectilinear Minimum Steiner Tree with Obstacle(s) \*/

COMPUTE-RSMT(*Input1*, *Input2*)

CANDIDATE\_SET ← HANAN-GRID-SET(*Input1*, *Input2*)

Initialize MAX\_POINT to a positive number

while MAX\_POINT ≥ 0

COST = 0

Assign MAX\_POINT a negative value

for each point p ∈ CANDIDATE\_SET

DeltaCost ← DELTA-MST(*Input1*+ SteinerPoints, p)

if DeltaCost > COST

COST ← DeltaCost

If MAX\_POINT ≥ 0

SteinerPoints = SteinerPoints ∪ {MAX\_POINT}

for each point p ∈ SteinerPoints

if Degree ≤ 2

SteinerPoints = SteinerPoints – {p}

FINAL\_RSMT ← KRUSKAL(*Input2* + SteinerPoints)

DISTANCE\_RSMT ← **Σ** Edges.distance of FINAL\_RSMT

return FINAL\_ESMT

/\*Computes set of Hanan Points excluding points in Obstacle(s); which can be used as Steiner Points in COMPUTE-RSMT \*/

HANAN-GRID-SET(*Input1*, *Input2*)

CANDIDATE\_SET ← {}

HananPoints ← HANAN-GRID(*Input2)*

for each point p ∈ HananPoints

if point p ∉ Points within Obstacles

CANDIDATE\_SET = CANDIDATE\_SET ∪ {p}

return CANDIDATE\_SET