AIAA SciTech Forum
23-27 January 2023, National Harbor, MD & Online
AIAA SCITECH 2023 Forum

10.2514/6.2023-1146

Real-Time Motion Planning for Quadcopter Using Adaptive Spherical Expansion and Sequential Convex Optimization

Atakan KARAMAN[1] and H. Burak KURT[2]
Turkish Aerospace, Ankara, Turkey

Murat MILLIDERE[3] and Mushfiqul ALAM[4]
School of Aerospace, Transport and Manufacturing (SATM)
Cranfield University, Cranfield, Bedfordshire, MK43 0AL, United Kingdom

This paper introduces the Adaptive Spherical Expansion and Sequential Convex Programming (ASE-SCP) as a real-time motion planning algorithm. ASE-SCP algorithm is an improved version of the Spherical Expansion and Sequential Convex Programming (SE-SCP) algorithm in terms of computational speed. ASE-SCP is a hybrid real-time motion planning algorithm which combines the advantages of the Adaptive Spherical Expansion, such as approaching the neighborhood of the global optimal path, and the quick convergence ability of the Sequential Convex Programming. The ASE-SCP algorithm first finds a collision-free path using the adaptive spherical expansion approach. After finding a feasible path from the start point to the target point, the feasible path is re-optimized (tuned) using sequential convex optimization to find the suboptimal path. ASE-SCP Algorithm is applied to a quadcopter model to demonstrate its applicability.

Nomenclature

RMS	=	Root Mean Square
EOM	=	Equation of Motion
ZMQ	=	ZeroMQ
PRM	=	Probabilistic Road Maps
FMT	=	Fast Marching Trees
RRT	=	Exploring Random Trees
SCP	=	Sequential Convex Programming
SE-SCP	=	Spherical Expansion Sequential Convex Programming
ASE-SCP	=	Adaptive Spherical Expansion Sequential Convex Programming
CVX	=	Convex Optimization Solver

I.Introduction

Motion planning plays an important role in improving the autonomous flight capability of unmanned aerial vehicles (UAVs) over the last decade [1]. One such UAV is the quadcopter, which is used for many military and civilian applications [2] [3] [4] and may fly in a complex environment with a significant number of obstacles. These

[1] Flight Dynamics, Simulation & Control Engineer.
[2] Flight Dynamics, Simulation & Control Engineer.
[3] Research Fellow, Dynamics, Simulation and Control Group, Centre for Aeronautics
[4] Lecturer, Dynamics, Simulation and Control Group, Centre for Aeronautics

obstacles may be fixed or may move over time. To successfully complete a mission, an autonomous quadcopter requires appropriate algorithms for motion planning. These missions can be formation, aggregation, or cluster [5].

There are many algorithms, such as sampling-based and search algorithms, that have been proposed and applied to quadcopter path planning [1]. However, these algorithms are not efficient for multi-perspectives. The sampling-based algorithms are widely used in practice [6] [7]. Some of these algorithms are Rapid Exploring Random Trees (RRT), Fast Marching Trees (FMT) and Probabilistic Road Maps (PRM) [6]. These algorithms generally focus only on distance and try to minimize it. In other words, the algorithm tries to increase the number of samples to decrease the distance of the path, but computational efficiency will decrease substantially with the increasing sample number.

In SE-SCP, the elapsed time is much shorter compared to other motion planning algorithms. In this hybrid algorithm, first Spherical Expansion (SE) is used to generate a feasible path while avoiding collisions with obstacles. Then Sequential Convex Programming (SCP) is used to optimize the existing path obtained by SE. It means the SE is used to generate a first guess for the SCP initialization. However, when the radius of the sphere is smaller than the predefined value, the SE-SCP algorithm tends to get stuck around the objects. To prevent this phenomenon, the radius of the sphere is adaptively changed, which is the contribution of this study.

In this paper, a new algorithm called Adaptive Spherical Expansion and Sequential Convex Optimization (ASE-SCP) (a modified version of SE-SCP) is proposed. The modified algorithm is faster and more stable than the original SE-SCP.

In the study, a mathematical model of a quadcopter was used [7]. The rest of the paper is organized as follows: Section III models the problem of path planning. In Section IV, the ASE-SCP algorithm is introduced. Section V presents the quadcopter model, simulation environment and real-time communication method. In Section V, the results are analyzed. Finally, Section VII draws the conclusion of this paper and mention the future work.

II. Problem statement

In this paper, the mission of target tracking and obstacle avoidance are considered simultaneously for the dynamic path planning problem, which can be regarded as the real-time optimization problem under constraints. In this section, optimization problem is defined. Let $\mathrm{X} \in \mathbb{R}^n$ represents the workspace, while $\mathrm{X}_{obs} \subset \mathrm{X}$ represent the obstacles in the workspace, and $\mathrm{X}_{unsafe} \subset \mathrm{X}$ represents the unsafe region around an obstacle. This region is decided by the user considering quadcopter dynamics and safety factor in real-time application. Therefore, the region where the quadcopter move safely is $\mathrm{X}_{safe} = \mathrm{X}/\left(\mathrm{X}_{obs} \cup \mathrm{X}_{unsafe}\right)$.

The optimization problem is constrained by the quadcopter dynamics which is given in Appendix A. Quadcopter nonlinear model is given in Eq. (1);

$$\dot{x} = f(x, u) \tag{1}$$

where f is a nonlinear function, x is the state vector of the quadcopter, $u \in U$ is the control inputs and U is the feasible range of control inputs. Feasible range means available control power physically. We cannot provide control power more than available power. It is also one of the optimization problem constraints.

The $\sigma(t) \in \mathrm{X}_{safe}$ represents a feasible trajectory for the quadcopter and $\mathcal{J}\left(\sigma(t)\right)$ represents the cost of this feasible trajectory. The cost function can be defined in several ways depending on the purpose of the quadcopter operation. For example, the cost function can be defined for minimum battery energy consumption, or minimum trajectory distance or a combination of both [6]. In this study cost function is defined to minimize control inputs and trajectory distance. Let $\mathrm{X}_{init} \in \mathrm{X}_{safe}$ and $\mathrm{X}_{target} \in \mathrm{X}_{safe}$ represent initial and desired positions of the quadcopter in the workspace, respectively. These are also optimization constraints; the quadcopter trajectory should start from X_{init} and stop at X_{target}.

After definition of optimization problem and its constraints, the optimization problem can be written in Eq. (2);

$$\begin{aligned} min_{\sigma(t),u(t)} \quad & \mathcal{J}\left(\sigma(t)\right) \\ \text{Subject to} \quad & \sigma(t_0) = \mathrm{X}_{init} \\ & \sigma\left(t_f\right) = \mathrm{X}_{target} \\ & \dot{\sigma}(t) = f\left(\sigma(t), u(t)\right), \forall t \in \left[t_0, t_f\right] \\ & \sigma(t) \in \mathrm{X}_{safe}, \forall t \in \left[t_0, t_f\right] \\ & u(t) \in U, \forall t \in \left[t_0, t_f\right] \end{aligned} \tag{2}$$

The globally optimal solution of equation (2) is the best trajectory for the given cost function given the constraints. However, the given optimization problem is complex, and it is difficult to find a globally optimal solution. This is because the given problem is a non-linear, non-convex optimization problem. The given optimization problem can be solved using global optimization methods such as genetic algorithm, particle swarm optimization, etc., but this may take too much time and is hence not suitable for real-time applications. Therefore, convex optimization methods should be used to find an optimal solution to the given optimization problem. Convex optimization methods provide an optimal solution for real-time problems. To use convex optimization methods, the given problem must be transformed into a convex problem. This procedure is called convexification [8] or convex relaxation [9].

Firstly, the continuous optimization problem is discretized. Continuous time $[t_0, t_f]$ is represented as discrete time steps $[0, 1, 2, \dots, T]$. Optimal trajectory from X_{init} to X_{target} that passes through spheres corresponding to the points $[\mathrm{X}_0, \mathrm{X}_1, \mathrm{X}_2, \dots, \mathrm{X}_{T-1}]$. Therefore, $\sigma(0) = \mathrm{X}_{init}$ and $\sigma(T) = \mathrm{X}_{target}$.

To obtain convex form of optimization problem, norm of the convex functions is used. Therefore, first part of the cost function can be written as a sum of the norms of given inputs for each time step. The other part of the cost function can be written using sum of square of trajectory to make cost function convex. Now cost function of optimization problem is in convex form.

In the case of trajectory should be in safe workspace, these constraints can also be expressed using norm function. The norm of the difference of trajectory points and the sphere center for each sphere should be lower or equal to the radius of each sphere.

Also, available input constraint can be written using norm function. The norm of the inputs should be lower than maximum available inputs for each time step.

Finally, the nonlinear model of quadcopter should be linearized for convex form of optimization problem. One major work in convexification of complex problem lies on how to convexify the nonlinear dynamics. Linearization of nonlinear dynamics is one convexification method. Nonlinear terms are repeatedly linearized at a known solution obtained in the previous iteration [10]. Detailed information about quadcopter linear model can be found in Appendix B.

Mathematical expressions of all constraints and cost function in given Eq. (3). Now our optimization problem is convex. Therefore, we can solve it using convex optimization solver, namely CVX [11].

$$
\begin{aligned}
min_{\sigma(t),u(t)} \quad & \sum_{k=0}^{T-1} \|u(k)\|_2 + abs(\sigma(t) - \sigma(t+1)) & \\
\text{Subject to} \quad & \sigma(0) = \mathrm{X}_{init} & \\
& \sigma(T) = \mathrm{X}_{target} & \\
& \sigma(k+1) = A(k)\sigma(k) + B(k)u(k) & \forall k \in 0, \dots, T-1 \\
& \|\sigma(k) - \mathrm{X}_k\|_2 \leq r_k & \forall k \in 0, \dots, T \\
& \|u(k)\|_2 \leq U_{max} & \forall k \in 0, \dots, T
\end{aligned}
\tag{3}
$$

III. Adaptive Search with Spherical Expansion and Sequential Convex Optimization

In this section, the Adaptive Spherical Expansion algorithm is presented. First, some terms are defined. Let V be the set of vertices. Besides the position of the vertex, V also stores the minimum distance of the vertex to the obstacles. Let E be the set of edges, where each element store the start and end point of the edge, the current trajectory and the cost of the trajectory. Therefore, the graph can be expressed as $G = (V, E)$.

The entire algorithm can be seen as a flow chart in Figure 1. The algorithm can be divided into three steps: Initialization, adaptive search with spherical expansion and sequential convex optimization. A detailed pseudo-algorithm can be found in Appendix C.

A. Initialization Step

In the Initialization section, the initial trajectory, X_{init}, and the target position, X_{target}, are defined in this section. When initializing a new sphere, the sphere is initialized with the x, y position and r, the radius of the sphere. The radius of the sphere expresses the maximum radius of the sphere without intersecting with objects. The search radius, r_{search}, is also initialised with GenerateRandRadius(). The function r_{search} for the adaptive search is calculated as given in Eq. 4,

$$
r_{search} = k_{constant} * GenerateRandomNumberInRange\big([0.0,1.0)\big) \tag{4}
$$

The *GenerateRandomNumberInRange()* function given in Eq. 4 generates a random number in the specified range. Here the specified range is [0.0,1.0). The value r_{search} is used in the *AdaptiveRadiusDecision()* function which is given in algorithm 1.

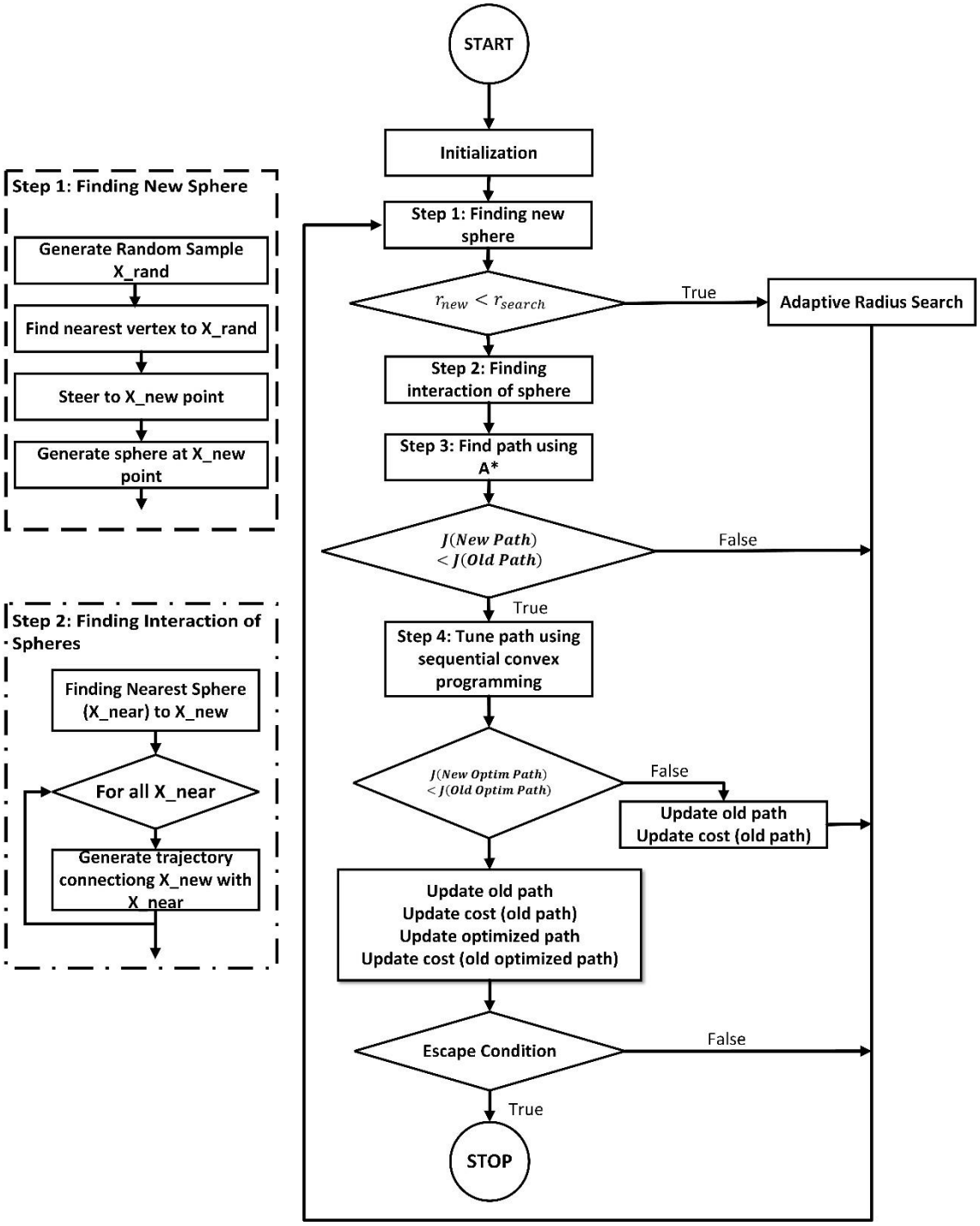

Figure 1: Flowchart of ASE-SCP.

B. Adaptive Search with Spherical Expansion Step

The main objective of this step is to find a feasible and collision-free trajectory starting from the initial position X_{init} and leading to the final position X_{target} in real time. In this step, the algorithm scans the entire map with an adaptive search radius, using some random search techniques.

The random sampling method returns X_{rand}. This random point is drawn from a uniform distribution and can be anywhere in the area except objects. After the random point is determined, the closest point to the random point within the vertex is searched. The point closest to the X_{edge} becomes the $X_{nearest}$ point. In other words, $X_{nearest}$ is the vertex closest to the X_{edge} point. Then $X_{nearest}$ point is calculated. If X_{edge} is within the sphere of $X_{nearest}$, X_{edge} is set to the new X_{new} point. In the other case, where X_{rand} is outside the sphere of $X_{nearest}$, the algorithm finds a new point on the surface of the sphere of $X_{nearest}$ and returns this point as X_{new}. After X_{new} is established, the search radius, r_{search}, is a number created to prevent the algorithm from failing. Without this constraint, the quadcopter is prone to stacking up next to obstacles. Also, this constraint makes for a much faster search algorithm. This is because the algorithm does not take a small step (small sphere) if it is not necessary. When it is necessary (there is little space between two objects), the adaptive search algorithm reduces the search radius to find a feasible path, even if it is small. This search radius is initialized by the *GenerateRandRadius()* function in the initialization step and the search radius changes adaptively in the algorithm by the *AdaptiveRadiusDecision()* function, shown in Algorithm 1.

Algorithm 1: ***AdaptiveRadiusSearch***

1: **if** $r_{new} < r_{search}$, following 3 consecutive iteration **then**
2: $k_{random} = GenerateRandomNumberInRange([0.0, 1.0))$
3: $r_{search} = 0.5 + k_{random} * r_{search}$
4: continue with the next iteration
5: **end if**

Algorithm 1: Pseudo Algorithm of Adaptive Radius Search.

The list of neighbor points X_{near} to the X_{new} is found. All spheres in the set of X_{near} intersect with the sphere X_{new}. These intersections ensure that there is a collision free path between X_{new} and each vertex in X_{near}. Then, the X_{new} and its radius r_{new} are added to the set of vertices V.

After new point X_{new} and its neighbors X_{near} are added to vertices V, new trajectory is created with new vertex X_{new} and vertices in set of X_{near}. In this way, a set of graphs with the existing vertices is generated. These graphs and their costs are contained in set of edges. The trajectories are created with straight lines between the vertices. Therefore, the cost of the trajectories is calculated using Euclidian distances. In addition, this method reduces the computation time. These processes are repeated multiple time and spherical expansion run for new vertex and multiple edges. The vertex is listed in set of vertices and the edges are added to the graph.

C. A* Path Finding Algorithm

In this step, the shortest paths in the graph $G = (V, E)$ from initial position X_{init} to the target position X_{target} are found using A* path finding algorithm. A* Algorithm try to find the shortest path. In the adaptive search with spherical expansion step, many paths are found using spheres. The shortest path is found with A* algorithm among many alternative paths. How the A* algorithm works is not the subject of this article. More detailed information can be found in [12].

D. Sequential Convex Optimization Step

Next, using sequential convex programming (SCP), the shortest path which is found by A* is optimized at the cost of minimum distance and minimum inputs.

If the current shortest path step is lower than the previous one, then the optimal trajectory is found using Sequential Convex Programming (SCP). The SCP optimization tries to find the optimal trajectory from X_{init} to X_{target}, through spheres corresponding to the points $[X_0, X_1, X_2, \dots, X_{T-1}] \in Path$. The SCP optimization function uses the CVX package to find the optimal trajectory. CVX package solves the optimization problem in Eq. (3). As mentioned earlier, the optimization problem has a convex form so the CVX package finds the optimal trajectory quickly.

The optimal trajectory is within given spheres and optimizes the shortest path for given costs. Then the new optimal trajectory is optimized recursively using CVX package. This sequential optimization provides a better optimal trajectory.

After each step of the ASE-SCP algorithm, there are escape and break conditions; these are to allow the algorithm to run in real time. The aim of these condition is to make the algorithm run in real time. If the quadcopter cannot find the optimal path or it takes too much time for searching, escape condition terminates the algorithm and skips to the next iteration.

IV. Quadcopter Model, Simulation Environment & Real-Time Communication

To analyze the performance of the ASE-SCP algorithm, it is examined in a simulated environment. For this purpose, a quadcopter simulation model is developed. First, quadcopter`s equation of motion (EOM), actuators, stability and control augmented systems are modelled in MATLAB/Simulink environment. Then, C code of the Simulink model is generated because the communication between C code and the proposed algorithm is simpler.

The proposed algorithm, ASE-SCP, is developed in Python environment. The main reason for this is that CVXPY (a library written in Python) is used to solve the convex optimization problems.

To run the simulation in real-time, both the quadcopter flight dynamic model and the ASE-SCP algorithm run separately in different threads. Both can communicate with each other. Therefore, neither the ASE-SCP algorithm nor quadcopter flight dynamic model need to wait for each other`s output to continue. This means that as soon as a new waypoint is available, the quadcopter moves to that point. In addition, the waypoint can be changed if the quadcopter does not complete previous one. This enables real-time communication.

Both threads use the ZeroMQ (ZMQ) library to communicate with each other. ZMQ is an asynchronous messaging library. ZMQ supports all languages such as C, C++, Python, etc. ZMQ provides many messaging patterns like Publishing/Subscriber, Request/Reply, Client/Server and others. In our case, Publisher/Subscriber message pattern is chosen. Publish/Subscribe is classic pattern where the senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. The messages are published without knowing whether there are subscribers and if so, which ones. In our communication problem, the ASE-SCP thread publish desired waypoints and subscribe quadcopter current point to feed the algorithm as shown in Figure 2. Simulation subscribes waypoint and publish the quadcopter current position.

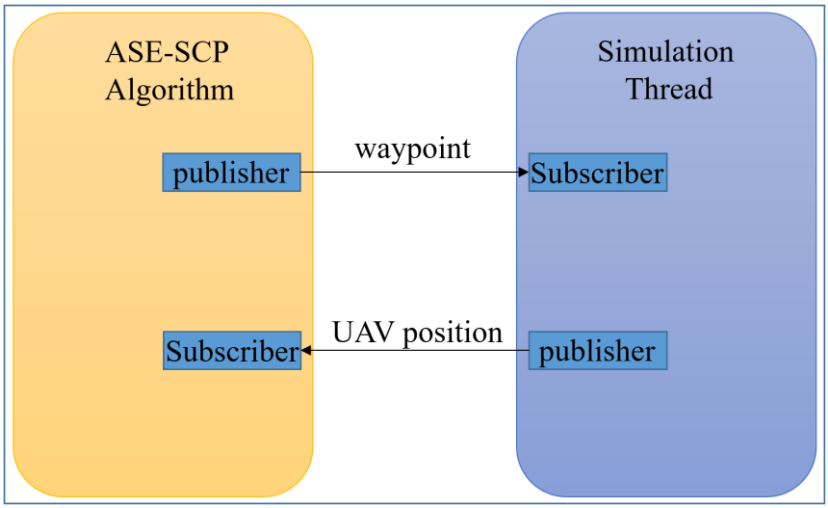

Figure 2: Simulation & ASE-SCP communication interfaces

V. The Results and Discussion

An example case is examined in this section. The start point is taken [10, 10][metres] and the target point is at [95, 95][metres]. In Figure 3-Figure 5, we can see the performance of the proposed method. In Figure 3 and Figure 4, `green square` indicates the current position of quadcopter, `red line` indicates the feasible path is founded by A* algorithm, `blue line` indicates suboptimal path by SCP algorithm, and `black circle` indicate obstacles.

In Figure 3, ASE-SCP algorithm performance can be seen for different object sets for random time shots. As shown in the figure, the algorithm can generate collision-free trajectories. As seen in Figure 3 (a) and (b), both simulations are very close to the target position, While ASE-SCP finds the shortest path in Figure 3(a), the founded path is not the shortest in Figure 3(b). Because ASE-SCP does not guarantee to find the shortest path. A* algorithm tries to find the shortest path using the given graph $G = (V, E)$, however we don't give all graph to A* algorithm. If we wait ASE-SCP algorithm search all map to obtain all graph, it takes so much time. Therefore, real time path planning is not possible anymore. So, this is the trade-off. ASE-SCP algorithm presents an object-free path in real-time, so that algorithm gives us any object-free path, when it is available.

In Figure 3(c) and (d), simulation is in beginning phase. Red paths are A* paths. As seen, in Figure 3(c) algorithm finds the shortest path at the beginning of the simulation, while Figure 3(d) does not find the shortest path in the beginning of the simulation.

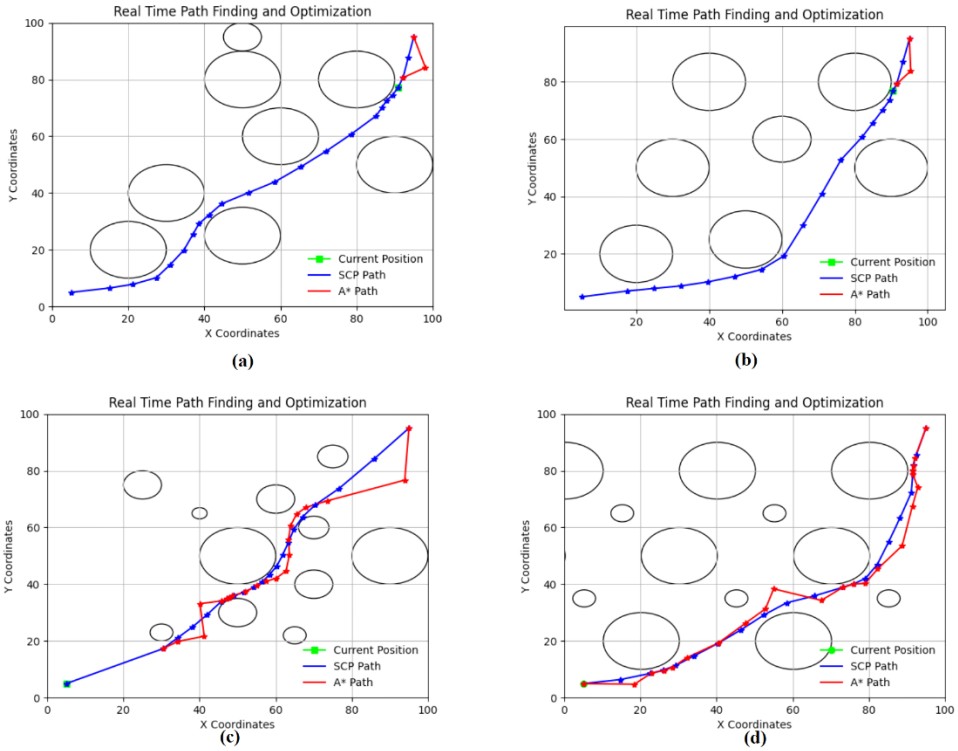

Figure 3: ASE-SCP Algorithm performance for different object sets.

In Figure 4, time sequence of one simulation from start to end can be seen. Once the optimization increment is finished, the current time t_c is updated and the process restarts. In this case example, the algorithm found a shorter trajectory at each time step except t_4. There is a little difference in the suboptimal paths after second optimization increment, t_2. Suboptimal paths converged to shortest path with increasing time.

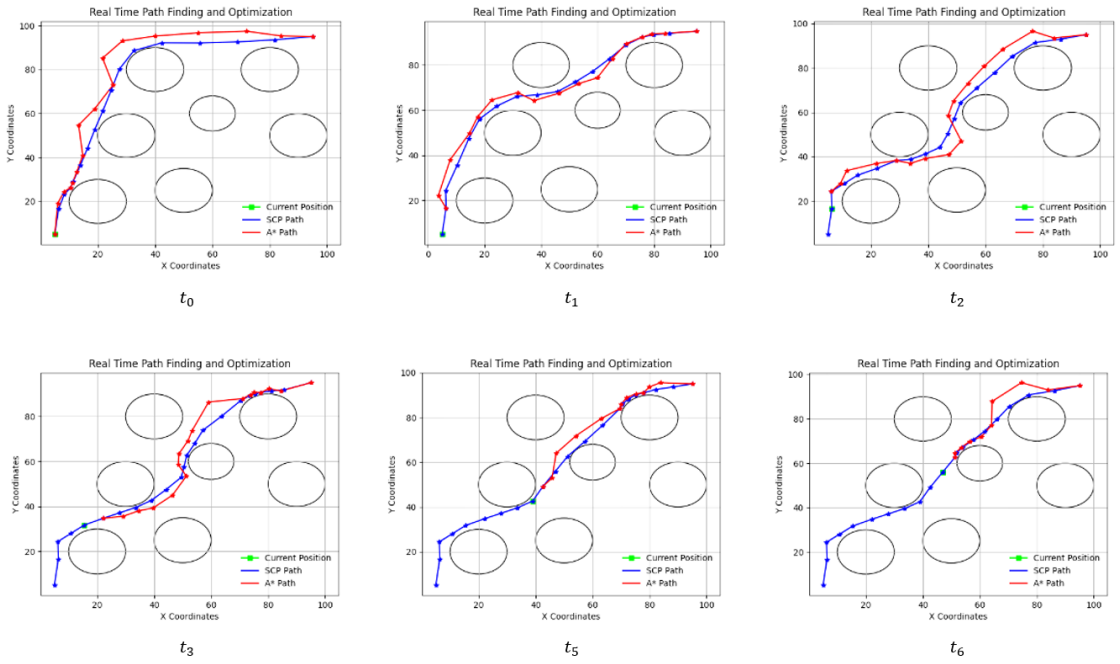

Figure 4: ASE-SCP Algorithm time sequences for selected object set.

In Figure 5, the time performance of the presented algorithm can be seen. Each bar indicates elapsed time for A* and SCP, respectively. Red-colored parts show the time elapsed for A* and blue-colored parts show the time elapsed for SCP. As seen, the SCP method takes roughly 0.30 seconds to complete each time step, with little variation. The computation time in the A* algorithm is more than the computation time in the SCP algorithm; particularly in the initial time step (more than 1 second). Because there is no prior knowledge of the workspace in the first step so it takes time to collect enough vertices, and give a graph to the A* algorithm.

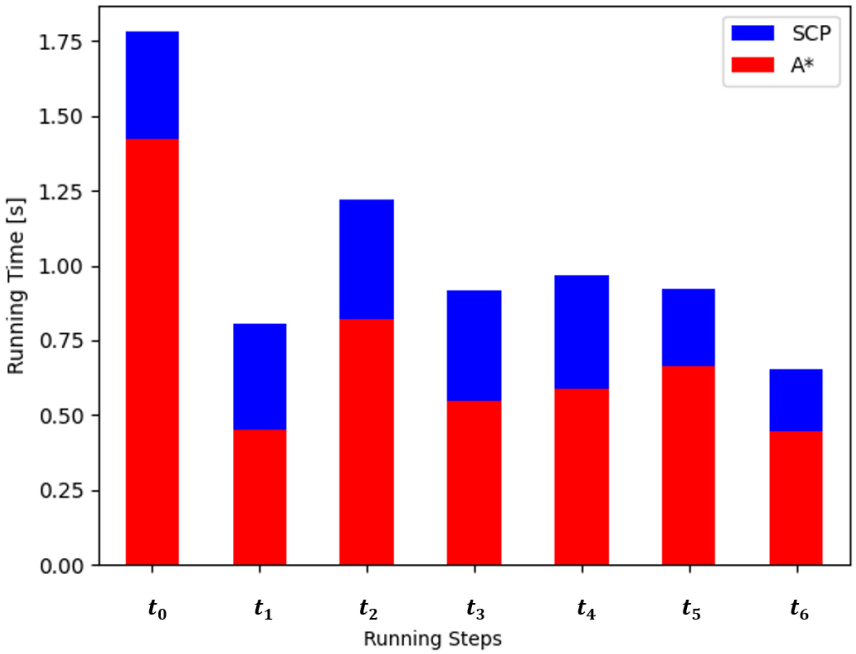

Figure 5: ASE-SCP algorithm time performance.

VI. Conclusion and Further Works

In this work a real-time motion planning algorithm was developed using the ASE-SCP. An enhancement step was also introduced, and this step avoids the algorithm to get stuck around the obstacles. The developed algorithm was further improved with an emergency protocol that stops quadcopter in critical scenarios.

Multiple basic simulations were performed to evaluate the real-time capabilities of the algorithm The relevant simulations have demonstrated that the optimized route can guide quadcopter track target and avoid obstacle at the same time. The algorithm proved that it could perform nearly real-time motion planning for static targets in environment with fixed obstacles. According to the problem of obstacle avoidance, A* algorithm is employed. Then, SCP was utilized to plan a suboptimal path in real time.

For future work, the proposed algorithm will be implemented to track moving targets and avoid static obstacles and moving threats at the same time. It will also be extended a three dimensional (3D) real time motion planning challenges for terrain following and landing scenarios.

References

[1] L. Qaun, L. Han, B. Zhou, S. Shen and F. Gao, "Survey of UAV motion planning," *IET Cyber-Systems and Robotics,* vol. 2, no. 1, pp. 14-21, 2020.

[2] S. M. LaValle, Planning Algorithms, Cambridge: Cambridge University Press, 2006.

[3] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki and S. Thrun, Principles of Robot Motion: Theory, Algorithms, and Implementations, Boston: MIT Press, 2005.

[4] S. Bandyopadhyay, F. Baldini, R. Foust, A. Rahmani, J.-P. d. l. Croix, S.-J. Chung and F. Y. Hadaegh, "Computationally Efficient Motion Planning Algorithms for Agile Autonomous Vehicles in Cluttered Environments," *IEEE Conference on Control Technology and Applications,* 2017.

[5] H. Hamann, Swarm Robotics:A Formal Approach, Springer Cham, 2018.

[6] F. Baldini, S. Bandyopadhyay, R. Foust, S.-J. Chung, A. Rahmani, J.-P. d. l. Croix, A. Bacula, C. M. Chilan and F. Y. Hadaegh, "Fast Motion Planning for Agile Space Systems with Multiple Obstacles," *AIAA/AAS Astrodynamics Specialist Conference,* 2016.

[7] L. E. Kavraki, P. Svestka, J.-C. Latombe and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation,* vol. 12, no. 4, pp. 566-580, 1996.

[8] B. Açıkmeşe and L. Blackmore, "Lossless Covexification of a Class of Optimal Control Problems with Non-convex Control Constraints," *Automatica,* vol. 47, no. 2, pp. 341-347, 2011.

[9] X. Liu, Z. Shen and P. Lu, "Exact Convex Relaxation for Optimal Flight of Aerodynamically Controlled Missiles," *IEEE Transactions on Aerospace and Electronic Systems,* 2016.

[10] X. Liu, P. Lu and B. Pan, "Survey of convex optimization for aerospace applications," *Astrodynamics,* vol. 1, no. 1, pp. 23-40, 2017.

[11] S. P. Boyd, *Convex Optimization Toolbox (CVX).*

[12] S. Erke, D. Bin, N. Yiming, Z. Qi, X. Liang and Z. Dawei, "An improved A-star based path planning for autonomous land vehicles," *International Journal of Advanced Robotic Systems,* vol. 17, no. 5, 2020.

[13] F. Sabatino, Quadrotor control: modellling, nonlinear control design, and simulation, Stockholm: KTH, 2015.

Appendix

A. Quadcopter Mathematical Model

The quadcopter dynamics will be presented by using Newton and Euler equations for the 3D motion of a rigid body. This mathematical model of quadcopter will provide required equations for simulation, control, and optimization activities, this model taken from reference [13].

Let us call $[x\ y\ z\ \phi\ \theta\ \psi]^T$ the vector including the linear and angular position of the quadcopter in the earth frame and $[u\ v\ w\ p\ q\ r]^T$ the vector including the linear and angular velocities in the body frame.

$$\mathbf{v} = \mathbf{R}\,.\,\mathbf{v_B} \tag{4}$$
$$\boldsymbol{\omega} = \mathbf{T}\,.\,\boldsymbol{\omega_B} \tag{5}$$

And the **T** is angular transformation matrix:

$$\boldsymbol{T} = \begin{bmatrix} 1 & s(\phi)t(\theta) & c(\phi)t(\theta) \\ 0 & c(\phi) & -s(\phi) \\ 0 & \frac{s(\phi)}{c(\theta)} & \frac{s(\phi)}{c(\theta)} \end{bmatrix} \tag{6}$$

So, the kinematic model of the quadcopter is:

$$\begin{aligned} \dot{x} &= w[s(\phi)s(\psi) + c(\theta)c(\psi)s(\theta)] - v[c(\phi)s(\psi) - c(\psi)s(\phi)s(\theta)] + u[c(\psi)c(\theta)] \\ \dot{y} &= v[c(\phi)c(\psi) + s(\phi)s(\psi)s(\theta)] - w[c(\psi)s(\phi) - c(\phi)s(\psi)s(\theta)] + u[c(\theta)s(\psi)] \\ \dot{z} &= w[c(\phi)c(\theta)] - u[s(\theta)] + v[c(\theta)s(\phi)] \\ \dot{\phi} &= p + r[c(\phi)t(\theta)] + q[s(\phi)t(\theta)] \\ \dot{\theta} &= q[c(\phi)] - r[s(\phi)] \\ \dot{\psi} &= r\frac{c(\phi)}{c(\theta)} + q\frac{s(\phi)}{c(\theta)} \end{aligned} \tag{7}$$

and the total force acting on the quadcopter:

$$m(\boldsymbol{\omega_B} \wedge \mathbf{v_B} + \dot{\mathbf{v}}_\mathbf{B}) = \mathbf{f_B} \tag{8}$$

where m is the mass of the quadcopter, $\wedge$ is the cross product and $\boldsymbol{f_B} = \left[f_x\ f_y\ f_z\right]^T \in \mathbb{R}^3$ is the total force. Euler's equation gives the total torque applied to the quadrotor:

$$\mathrm{I} \cdot \dot{\boldsymbol{\omega}}_\mathbf{B} + \boldsymbol{\omega_B} \wedge (\mathbf{I} \cdot \boldsymbol{\omega}_\mathbf{B}) = \mathbf{m_B} \tag{9}$$

where $\boldsymbol{m_B} = \left[m_x\ m_y\ m_z\right]^T \in \mathbb{R}^{3x3}$ is the total torque and I is the diagonal inertia matrix:

$$\mathbf{I} = \begin{bmatrix} \mathrm{I_x} & 0 & 0 \\ 0 & \mathrm{I}_y & 0 \\ 0 & 0 & \mathrm{I}_z \end{bmatrix} \in \mathbb{R}^{\boldsymbol{3x3}} \tag{10}$$

So, the dynamic model of the quadcopter in the body frame is:

$$\begin{aligned} f_x &= m(\dot{u} + qw - rv) \\ f_y &= m(\dot{v} - pw + ru) \\ f_z &= m(\dot{w} + pv - qu) \\ m_x &= \dot{p}I_x - qrI_y + qrI_z \\ m_y &= \dot{q}I_y + prI_x - prI_z \\ m_z &= \dot{r}I_z - pqI_x + pqI_y \end{aligned} \tag{11}$$

Set **u** to be the control vector: $\mathbf{u} = \left[f_t\ \tau_x\ \tau_y\ \tau_z\right]^T \in\ \mathbb{R}^4$. Let us consider the values of the input forces and torques proportional to the squared speeds of rotors, their values are the following:

$$\begin{aligned} f_t &= b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ \tau_x &= bl(\Omega_3^2 - \Omega_1^2) \\ \tau_y &= bl(\Omega_4^2 - \Omega_2^2) \\ \tau_z &= d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \end{aligned} \tag{12}$$

where l is the distance between any rotor and the center of gravity of the drone, b is the thrust factor and d is the drag factor. And after the substitution, the dynamic model of the quadcopter in the body frame is:

$$\begin{aligned} &-mg[s(\theta)] = m(\dot{u} + qw - rv) \\ &mg[c(\theta)s(\phi)] = m(\dot{v} - pw + ru) \\ &mg[c(\theta)c(\phi)] - b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) = m(\dot{w} + pv - qu) \\ &bl(\Omega_3^2 - \Omega_1^2) = \dot{p}I_x - qrI_y + qrI_z \\ &bl(\Omega_4^2 - \Omega_2^2) = \dot{q}\iota_y + prI_x - prI_z \\ &d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 + \Omega_3^2) = \dot{r}I_z - pqI_x + pqI_y \end{aligned} \tag{13}$$

B. Linear Model

The linearization's procedure is developed around an equilibrium point $\bar{\boldsymbol{x}}$, which for fixed input $\bar{\boldsymbol{u}}$ is the solution of the algebraic system.

$$\hat{\mathbf{f}}(\bar{\mathbf{x}}, \bar{\mathbf{u}}) = \mathbf{0} \tag{14}$$

The simplification is made by approximating the sine function to its argument and cosine function to unity. Therefore, the linearization will be performed on a simplified model called to *small oscillations*. Only the approximation is valid if the argument is small.

$$\begin{aligned} \dot{\phi} &\approx p + r\theta + q\phi\theta \\ \dot{\theta} &\approx q - r\phi \\ \dot{\psi} &\approx r + q\phi \\ \dot{p} &\approx \tfrac{I_y - I_z}{I_x} rq + \tfrac{\tau_x}{I_x} \\ \dot{q} &\approx \tfrac{I_z - I_x}{I_y} pr + \tfrac{\tau_y}{I_y} \\ \dot{r} &\approx \tfrac{I_x - I_y}{I_z} pq + \tfrac{\tau_z}{I_z} \\ \dot{u} &\approx rv - qw - g\theta \\ \dot{v} &\approx pw - ru + g\phi \\ \dot{w} &\approx qu - pv + g - \tfrac{f_t}{m} \\ \dot{x} &\approx w(\phi\psi + \theta) - v(\psi - \phi\theta) + u \\ \dot{y} &\approx v(1 + \phi\psi\theta) - w(\phi - \psi\theta) + u\psi \\ \dot{z} &\approx w - u\theta + v\phi \end{aligned} \tag{15}$$

And the compact form is:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{16}$$

So, after the perform the linearization, the linear model is:

$$\dot{\mathbf{x}} = \mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{u} + \mathbf{D} \cdot \mathbf{d} \tag{17}$$

$$\begin{gathered} \dot{\phi} = p \\ \dot{\theta} = q \\ \dot{\psi} = r \\ \dot{p} = \frac{\tau_x}{I_x} \\ \dot{q} = \frac{\tau_y}{I_y} \\ \dot{r} = \frac{\tau_z}{I_z} \\ \dot{u} = -g\theta \\ \dot{v} = g\phi \\ \dot{w} = -\frac{f_t}{m} \\ \dot{x} = u \\ \dot{y} = v \\ \dot{z} = w \end{gathered} \tag{18}$$

Here are the matrices that associated to the linear system:

$$\boldsymbol{A} = \frac{\delta \mathrm{f}(\mathbf{x}, \mathbf{u})}{\delta \mathbf{x}}\Big|_{\substack{\mathrm{x}=\bar{\mathrm{x}} \\ \mathrm{u}=\bar{\mathrm{u}}}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{19}$$

$$\boldsymbol{B} = \frac{\delta \mathrm{f}(\mathbf{x}, \mathbf{u})}{\delta \mathbf{u}}\Big|_{\substack{\mathrm{x}=\bar{\mathrm{x}} \\ \mathrm{u}=\bar{\mathrm{u}}}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{20}$$

$$\boldsymbol{D} = \frac{\delta \mathrm{f}(\mathbf{x}, \mathbf{u}, \mathbf{d})}{\delta \mathbf{d}}\Big|_{\substack{\mathrm{x}=\bar{\mathrm{x}} \\ \mathrm{u}=\bar{\mathrm{u}}}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{I_z} \\ \frac{1}{m} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{m} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{21}$$

C. Pseudo Algorithm

Algorithm 1 ASE-SCP

$X_{init}, r_{init} \leftarrow GenerateSphere()$
$X_{goal}, r_{goal} \leftarrow GenerateSphere()$
$r_{search} \leftarrow GenerateRandRadius()$
$V \leftarrow \{X_{init}[r_{init}], X_{goal}[r_{goal}]\}$
E $\leftarrow \emptyset$
$G = (V, E)$
$c(P_{init,goal,old}) \leftarrow \infty$, $c(\sigma_{init,goal,old}) \leftarrow \infty$
while True **do**
    betterOptimizedPathFlag = 0
    $X_{rand} \leftarrow GenerateSamples()$
    $X_{nearest} \leftarrow Nearest(G, X_{rand})$
    $X_{new} \leftarrow Steer(X_{nearest}, X_{rand}, r_{nearest})$
    $X_{new}, r_{new} \leftarrow GenerateSphere(X_{new})$
    if $r_{new} < r_{search}$, following 3 consecutive iteration **then**
        $k_{random} = GenerateRandomNumberInRange([0.0, 1.0))$
        $r_{search} = 0.5 + k_{random} * r_{search}$
        continue with next iteration
    end if
    $X_{near} \leftarrow NearIntersectedSphere(G, X_{new}, r_{new})$
    for all $X_n \in X_{near}$ **do**
        $\sigma_{n,new}, c(\sigma_{n,new}), \sigma_{new,n}, c(\sigma_{new,n}) \leftarrow SimpleTrajectory(X_n, X_{new}, r_n, r_{new})$
        $E \leftarrow E \cup \{(X_n, X_{new})[\sigma_{n,new}, c(\sigma_{n,new})], (X_{new}, X_n)[\sigma_{new,n}, c(\sigma_{new,n})]\}$
    end for
    $P_{init,goal,new}, c(P_{init,goal,new}) \leftarrow Astar(G, X_{init}, X_{goal})$
    if $c(P_{init,goal,new}) < c(P_{init,goal,old})$ **then**
        $\sigma_{init,goal,new}, c(\sigma_{init,goal,new}) \leftarrow SCP(X_{init}, X_{goal}, P_{init,goal,new})$
        if $c(\sigma_{init,goal,new}) < c(\sigma_{init,goal,old})$ **then**
            $\sigma_{init,goal,old} \leftarrow \sigma_{init,goal,new}$
            $c(\sigma_{init,goal,old}) \leftarrow c(\sigma_{init,goal,new})$
            $P_{init,goal,old} \leftarrow P_{init,goal,new}$
            $c(P_{init,goal,old}) \leftarrow c(P_{init,goal,new})$
            betterOptimizedPathFlag = 1
        else
            $P_{init,goal,old} \leftarrow P_{init,goal,new}$
            $c(P_{init,goal,old}) \leftarrow c(P_{init,goal,new})$
        end if
    end if
    if loop iteration < 50 **then**
        Continue
    else if betterOptimizedPathFlag == 1 & loop iteration > 50 **then**
        Break While Loop
    else if betterOptimizedPathFlag == 0 & loop iteration > 50 & loop iteration $\leq$ 50 **then**
        Continue
    else if betterOptimizedPathFlag == 0 & loop iteration > 100 **then**
        Break While Loop
    else
        Break While Loop
    end if
end while