A Sequential Convex Programming Approach to UAV Linking Formation Optimization

Jason Zhou
2329489

AMATH 515 - Optimization: Fundamentals and Applications

University of Washington

Mar. 8, 2024

1 Problem Motivation and Description

The UAV linking problem is one with many practical applications. Imagine a scenario in which UAVs can be used to transmit information using optical equipment such as a laser or concentrated wireless transmission. Such transmission methods could be extremely fast with minimal loss, but requires the send and receive UAV to be in direct visual line of sight of each other, in other words, the UAVs can be connected by a linear line. Due to the low loss of these transmission methods, the link should be designed around minimizing the total time of transmission (latency), and therefore the distance of transmission should be minimized. Another way to think about it is there could be signal jamming and no-fly zones that the UAVs and signal link must avoid.

This problem clearly lends itself to the form of an optimization problem. We have a straightforward objective cost: to minimize the total distance that the signal link travels, which we can take to be the Euclidean distance between each UAV in the link. The link itself is subject to the constraint of the no-fly and no-signal zones.

For simplicity of visualization and computation, we restrict this problem to be 2D. We see that the feasible region is clearly non-convex, as we are essentially putting holes in the feasible region. This means there can be multiple locally optimal solutions within the feasible area, which numerical solvers and the methods we were introduced to in this course are ill-suited for. [2]

The described problem can be posed as an optimization problem using the framework presented in the course. We have:

$$\underline{\text{Cost function:}} \qquad \min \sum_i ||\boldsymbol{r}_{i+1} - \boldsymbol{r}_i||_2, \quad \boldsymbol{r}_i \in \mathbb{R}^2 \ \forall i \in [1, n-1] \tag{1a}$$

$$\underline{\text{Node constraint:}} \qquad g_k(\boldsymbol{r_i}) \leq 0 \qquad \forall i \in [1, n], \ k \in [1, q] \tag{1b}$$

$$\underline{\text{Link constraint:}} \qquad G_k(\boldsymbol{r}_i, \boldsymbol{r}_{i+1}) \leq 0 \tag{1c}$$

$$\underline{\text{Endpoint constraint:}} \qquad \boldsymbol{r}_1 = \boldsymbol{r}_1^{(0)}, \boldsymbol{r}_n = \boldsymbol{r}_n^{(0)} \tag{1d}$$

for a problem with n UAVs (nodes) and q obstacles (no-fly/no-signal zones). The endpoint constraint forces the first and last nodes in the link to be their initial guess positions and constant through the

optimization. The derivation of the constraints will be explored in the methodology section. We can present a graphical example of the described problem as:

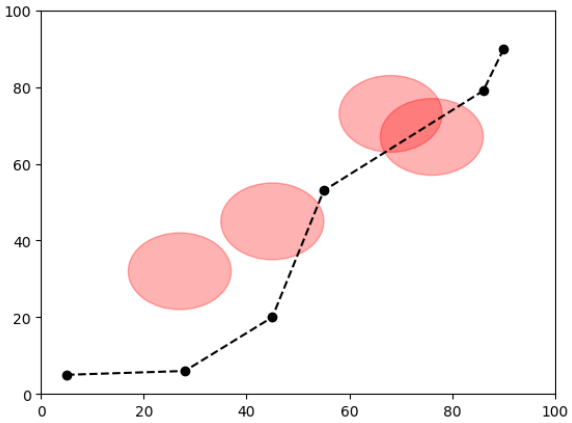


Figure 1: Sample UAV linking problem with 6 UAVs and 4 no-fly/no-signal zones. The node positions are randomly generated, and shows node and link constraints.

2 Methodology

To solve for the feasible and optimal formation, we leverage the method of sequential convex programming (SCP). As we've covered in the course, convex optimization methods are globally optimal and can be relatively fast. However, regarding our non-convex linking problem, and non-convex problems in general, local optimization methods are heuristic and may not find the globally optimal, or even any feasible solutions. On the other hand, globally optimal non-convex optimization methods can be extremely slow as they may need to explore the entire feasible space, which likely requires beyond reasonable time to solve.

SCP solves a non-convex optimization problem by iteratively solving convexified versions of the non-convex problem. Starting from some initial guess solution (which may or may not be feasible), each subsequent solution gets closer and closer to a local minima of the original problem. These solutions are then used as initial guess of the next iteration, and this procedure is repeated until the solution converges upon some local minima with guaranteed feasibility.

2.1 Constraint Linearization

2.1.1 Node Constraints

To create a convex sub-problem, we must convexify the non-convex node and link constraints. We can first write the non-convex formulation of the node constraints:

$$g_k(\boldsymbol{r}_i) \leq 0 \ \forall k \in [1, q], \ i \in [1, n] \tag{2}$$

where $k = 1, 2, \ldots, q$ are the individual no-fly zones where nodes cannot be placed, and $i = 1, 2, \ldots, n$ are the individual nodes. For simplicity, we assume that the no-fly zones are circular in shape, but the approach can be expanded to any geometric shape using its function. In 2D space, the equation of a circle centered at (x_c, y_c) with radius r is simply:

$$(x - x_c)^2 + (y - y_c)^2 = r^2 \tag{3}$$

so if a node satisfies a node constraint, the distance from the node $\boldsymbol{r}_i = [x_i, y_i]^\top$ to the center of the circular constraint $\boldsymbol{c}_k = [x_k, y_k]^\top$ should be greater than or equal to the radius r_k, which we write as:

$$r_c \leq \sqrt{(x_i - x_k)^2 + (y_i - y_k)^2} \implies 1 - \left(\frac{(x_i - x_k)^2}{r_c^2} + \frac{(y_i - y_k)^2}{r_c^2} \right) \leq 0 \tag{4}$$

$$\implies 1 - [x_i - x_k \;\; y_i - y_k] \begin{bmatrix} 1/r_k^2 & 0 \\ 0 & 1/r_k^2 \end{bmatrix} \begin{bmatrix} x_i - x_k \\ y_i - y_k \end{bmatrix} \leq 0 \tag{5}$$

$$\implies 1 - (\boldsymbol{r}_i - \boldsymbol{c}_k)^\top A (\boldsymbol{r}_i - \boldsymbol{c}_k) \leq 0 \tag{6}$$

$$\implies g_k(\boldsymbol{r}_i) \leq 0 \tag{7}$$

since $\boldsymbol{c}_k$ are constants, and we then take for every permutation of node i and constraint k. As this constraint is non-convex, we can then linearize each node to each constraint by perturbing $\boldsymbol{r}_i$ by some $\Delta \boldsymbol{r}_i$ to get:

$$g_k(\boldsymbol{r}_i) = g_k(\bar{\boldsymbol{r}}_i + \Delta \boldsymbol{r}_i) = g_k(\bar{\boldsymbol{r}}_i) + \frac{\partial g_k}{\partial \boldsymbol{r}_i} \Delta \boldsymbol{r}_i \tag{8}$$

$$= (1 - (\bar{\boldsymbol{r}}_i - \boldsymbol{c}_k)^\top A (\bar{\boldsymbol{r}}_i - \boldsymbol{c}_k)) - (\bar{\boldsymbol{r}}_i - \boldsymbol{c}_k)^\top (A + A^\top)(\boldsymbol{r}_i - \bar{\boldsymbol{r}}_i) \leq 0 \tag{9}$$

where $\bar{\boldsymbol{r}}_i$ is the reference value upon which we optimize for variable $\boldsymbol{r}_i$.

2.1.2 Link Constraints

Now we move onto the link constraint. The approach above will not work directly as we want to keep our optimization variables restricted to the positions of the nodes themselves, and not the discretized points on the link between the nodes. If we discretize the link between linked nodes $i, i+1$ to T equally spaced points, then we can write any discretized point as:

$$\boldsymbol{p}_t^{(i,i+1)} = \boldsymbol{r}_i + t \cdot \frac{\boldsymbol{r}_{i+1} - \boldsymbol{r}_i}{T} \;\; \forall t \in [1, T] \tag{10}$$

We then avoid expanding the set of optimization variables by parameterizing the discretized points using $\boldsymbol{r}_i, \boldsymbol{r}_{i+1}$. We can then make the claim that [3]:

$$g_k\left(\boldsymbol{p}_t^{(i,i+1)}\right) = 1 - \left(\boldsymbol{p}_t^{(i,i+1)} - \boldsymbol{c}_k\right)^\top A \left(\boldsymbol{p}_t^{(i,i+1)} - \boldsymbol{c}_k\right) \leq 0 \iff \int_1^T \max\left\{0, g_k\left(\boldsymbol{p}_t^{(i,i+1)}\right)\right\}^2 dt = 0 \tag{11}$$

We can prove this claim $\forall t \in [1, T]$ by considering the cases in the element-wise maximum operator. If $g_k\left(\boldsymbol{p}_t^{(i,i+1)}\right) < 0$, the operator will evaluate to 0. Otherwise, the integration will then be the sum of all non-negative quantities of $\int_1^T \left(g_k\left(\boldsymbol{p}_t^{(i,i+1)}\right)\right)^2 dt$, which will equal to 0 if and only if each individual quantity over the integral is equal to 0. As with the node constraint, we require $g_k\left(\boldsymbol{p}_t^{(i,i+1)}\right) \leq 0$ for all i, j in their respective ranges. Therefore, we must have $g_k\left(\boldsymbol{p}_t^{(i,i+1)}\right) \leq 0 \iff \int_1^T \max\left\{0, g_k\left(\boldsymbol{p}_t^{(i,i+1)}\right)\right\}^2 dt = 0$.

We can then write this integral in summation form and evaluate the maximum operator based on the individual discretized point. This also would be easier to numerically implement later on. Noting that $\Delta t = 1$ as we step through our discretized points, we have:

$$\int_1^T \max\left\{0, g_k\left(\boldsymbol{p}_t^{(i,i+1)}\right)\right\}^2 dt = 0 \iff \sum_{t=1}^{T} \max\left\{0, g_k\left(\boldsymbol{p}_t^{(i,i+1)}\right)\right\}^2 = 0 \tag{12}$$

while this can be implemented, we note that the $g_k\left(\boldsymbol{p}_t^{(i,i+1)}\right)$ above needs to be linearized in the same way as equation (9), where $\bar{\boldsymbol{p}}_t^{(i,i+1)}$ would be parmeterized by $\bar{\boldsymbol{r}}_i, \bar{\boldsymbol{r}}_{i+1}$. We can then write the constraint in terms of our optimization variables $\boldsymbol{r}$:

$$G_k(\boldsymbol{r}_i, \boldsymbol{r}_{i+1}) = \int_1^T \max\left\{0, g_k\left(\boldsymbol{r}_i + t \cdot \frac{\boldsymbol{r}_{i+1} - \boldsymbol{r}_i}{T}\right)\right\}^2 dt = 0 \tag{13}$$

for all discretization points $t \in [1, T]$ and obstacles $k \in [1, q]$.

2.2 Problem Revisions

The methodology above describes the optimization problem posed by equation (1), upon which the initial implementation was based. The constraints are observable, and makes sense in the physical interpretation of the problem. Through the results generated in section 4.1, we see that the constraints are relatively insufficient in defining a problem that's consistently solveable via SCP. The following revisions and extensions to the original problem were formulated and added to the implementation after observing the initial results.

2.2.1 Trust Region

A common component of SCP is the utilization of a convex trust region. In 2D, the trust region can be interpreted as a box constraint around the reference values of the optimization variables. This restricts the optimized variable to some distance from the previous iteration's solution, and therefore bounds the feasible space further. The intuition behind introducing a trust region, especially when non-convex constraints are linearized, can be illustrated in the figure below:

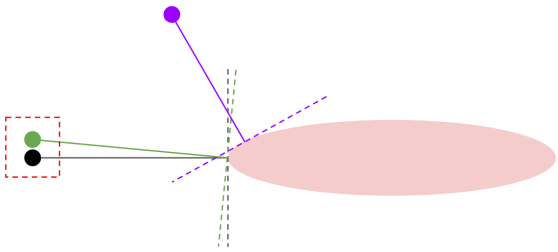


Figure 2: Example of how linearized constraints can differ within and outside of trust region

In figure 2, the red dotted line is the trust region, and the red ellipse is the constraint. Let the black point be the reference. A point within the trust region, such as the green point, would yield a linearized constraint (and therefore solution) not too different than that of the black point. However, the purple point, which may be more optimal to the cost function, can yield a vastly different linearization.

However, due to the high frequency of infeasible convex sub-problems observed in the initial implementation, further restricting the feasible space does not appear to be a good approach. Instead, we utilize the penalized trust region (PTR) method. In place of hard constraints on the optimization variables themselves, we penalize large deviations from initial guess by adding an additional term to the cost function. We then have a revised version of equation (1a):

$$\underline{\text{Cost with PTR:}} \qquad \min \sum_i ||\boldsymbol{r}_{i+1} - \boldsymbol{r}_i||_2 + \sum_j \rho ||\boldsymbol{r}_j - \bar{\boldsymbol{r}}_j||_2 \quad \forall i \in [1, n-1], j \in [2, n-1] \tag{14}$$

where $\bar{\boldsymbol{r}}_j$ is the reference value upon which we optimize for variable $\boldsymbol{r}_j$, and $\rho \in \mathbb{R}$ is a weight term for the trust region penalty. An interesting thing to note is that ρ can be positive or negative, and so we can actually *incentivize* solutions that deviate greatly from the initial guess, although this would likely cause the solution to never converge. The ρ parameter can be a tuned parameter or optimization variable, depending on the problem setup.

2.2.2 Grouped Nodes

In figure 6a, we notice that the converged solution grouped several nodes together on a singular line. While mathematically optimal, this formation has no benefits and could be detrimental in application. There can be two possible solutions to this issue: (1) the number of nodes in the link can be optimized such that nodes that are found to lie on or close to a linear link between other nodes in the converged solution can be discarded, and (2) a lower bound can be placed on the length of the link. I unfortunately do not have time to explore either approaches in the implementation, but we can analyze the introductory theory to both in this section.

In the first approach, we would then propose an entirely new optimization problem, or use a hard-coded method. One approach would be to post-process the converged solution and remove any node on or close to the linear link of its neighboring nodes, and the new link in the absence of the removed node can be checked for feasibility using equation (13). Another approach would be to add n as an optimization variable, possibly with an extremely low initial guess and conservative upper bound, and optimize by varying n as well as the positions of each node.

The second approach is less sensible but more mathematically interesting. We can write it as:

$$\underline{\text{Link minimum constraint:}} \qquad \min ||\boldsymbol{r}_{i+1} - \boldsymbol{r}_i||_2 < \ell_{min} \quad \forall i \in [1, n-1] \tag{15}$$

The lower bound of the 2-norm is known to be non-convex, as in physical space it would represent a ball with an empty core, which is clearly a non-convex set. We can either linearize the constraint as we have done with the other constraints in our problem, penalize the smallest link length in the formation, or try to apply lossless convexification (LCVX) on the constraint. The latter approach is the most interesting, as it is a method that's been shown to work in very specific problems related to spacecraft landing and rendezvous [1] [4] [5]. Such a constraint is not physically sensible for the problem, as removing a redundant node would be preferable in most forseeable physical scenarios, and enforcing a set number of nodes with minimum link length would almost certainly result in a less than optimal overall link. However, exploring the application of LCVX for this specific problem and constraint would hold a lot of mathematical value and rigor, and is an interesting extension to this project.

3 Implementation

The problem and solution is implemented in code using Python, and the Jupyter Notebook used to generate the results can be found at the Github link stated in the appendix.

We first define functions to linearize the node and link constraints as described in equations (9) and (13) respectively. In the implementation, we note that the individual links are discretized into 10 uniformly distributed points. This is a tuned parameter that notably affects likelihood of a feasible solution being found, and 10 was found to be the finest discretization that yielded consistently feasible solutions.

The optimization problem was then set up using the `CVXPY` library of Python, which lends itself really well to convex optimization problems as long as the problem itself is convex or convexified. Essentially, this method of implementation turns the complexity of this project to *how* to set up a non-convex problem to use convex solvers and methods such as the ones presented in the course, which I find to be an interesting extension upon the content presented in the course, especially since the vast majority of applied problems are non-convex.

We utilize the `ECOS` solver provided and defaulted to by `CVXPY`. `ECOS` is appropriate for our problem as it is specifically for second-order cone programming (SOCP), which our convex sub-problems belongs to as we define distance as the 2-norm. `ECOS` is also lightweight and well-documented, which makes debugging an easier process. The implementation for all the above are contained in the second cell of the notebook.

To define the problem, we define source and destination nodes, which are stationary, and either set or generate a random distribution of nodes within the square defined by the the source and destination nodes on diagonal corners. We further order the nodes based on their distance from the destination, in order of greatest to least. This pre-processing of initial distribution is important for two reasons: (1) as noted, the local nature of SCP can be highly sensitive to the initial guess, and so constraining the initial guess yields a higher chance of feasible solutions being found, and (2) to ease code complexity later on, ordering the list of nodes as a pseudo-linked list allows for the optimal cost to be easily calculated, and for sensible solutions to be generated. The ordering of points is especially important for reasonable solutions, which can be shown in figure 4.

Lastly, convergence criteria is also something defined and tuned in the implementation. The SCP implementation will sequentially use the solution of the previous convex sub-problem as the initial guess of the next convex sub-problem, until the optimal objective cost value does not change by more than 0.01 for 5 iterations. Changing these values would affect the convergence time/possibility of the algorithm.

3.1 Deviation from Theory

In the implementation of the link constraint defined by equation (13), the inequality is implemented as $G_k(\boldsymbol{r}_i, \boldsymbol{r}_{i+1}) \leq 0.000001$ due to the original equality constraint violating DCP (disciplined convex programming) rules, which is a benefit of using the `CVXPY` library. It isn't immediately clear why this is a violation, and I hope to investigate this with more time. However, this means the solution links could very slightly violate the constraints. This approach also prevents nodes from being placed on the edges of constraints, as the inequality in equation (9) would have permitted for this case. The node constraints were also changed to have the inequality ≤ -0.2 to further create a buffer between the nodes and the constraints, which is an approach we could not use for the link constraint due to the maximum operator.

4 Results

4.1 Initial Problem Formulation

As mentioned previously, as SCP is a local method, it can be highly dependent and sensitive to the initial guess and other parameters used in the problem. All of this was clearly observed throughout the implementation of this algorithm.

Let's start with a nice case, where converged solution makes sense on inspection:

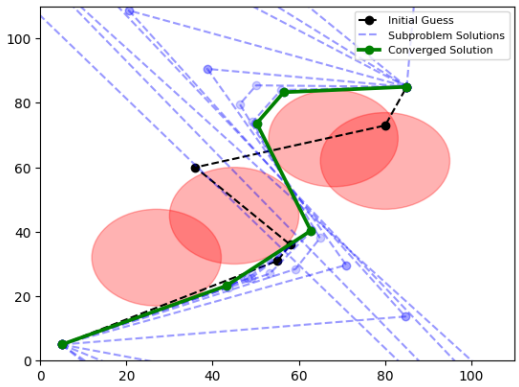


Figure 3: Sub-problem and converged solution, converged cost of 140.51, converged in 18 iterations

We see that the solution appears to be the most optimal for these conditions, and follows all constraints. We note that the initial guess appears to violate numerous node and path constraints, so it appears SCP has performed well, and solved this problem in 7.14 seconds.

Even in this simple case, we can make tiny changes to make this problem completely infeasible. As mentioned in the implementation section, we pre-condition the problem setup by ordering the nodes in the initial guess by their distance to the destination node. However, without sorting, we get:

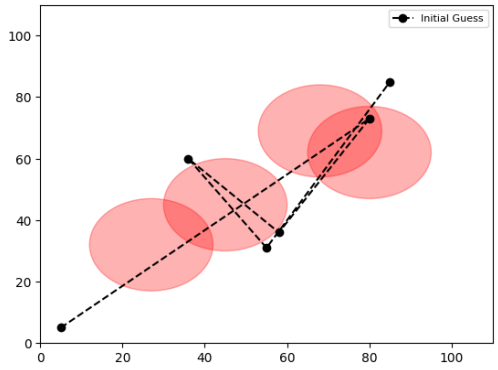


Figure 4: Identical initial conditions as figure 3, SCP failed to find feasible solution

so we clearly see the sensitivity of SCP with respect to the initial guess.

Through experimentation, we also see how SCP can be stuck in a local optimum and fail to find globally optimal solutions, or even improve upon a (visually) obvious sub-optimal solution. The following figure is a clear example of this phenomenon:

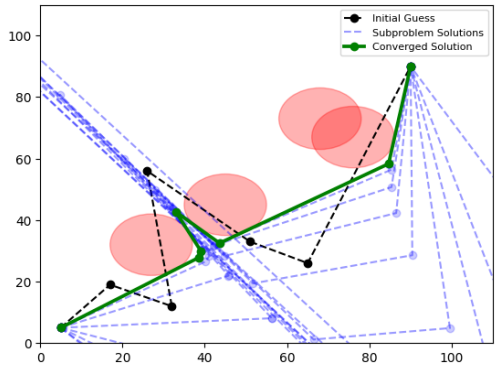
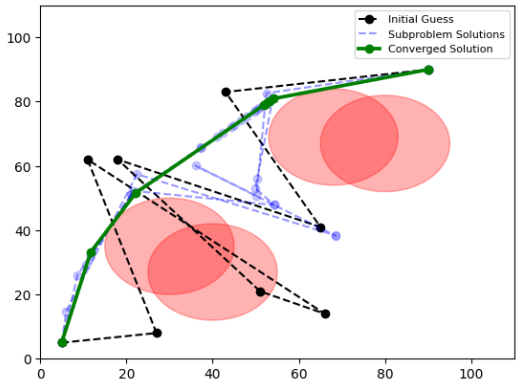


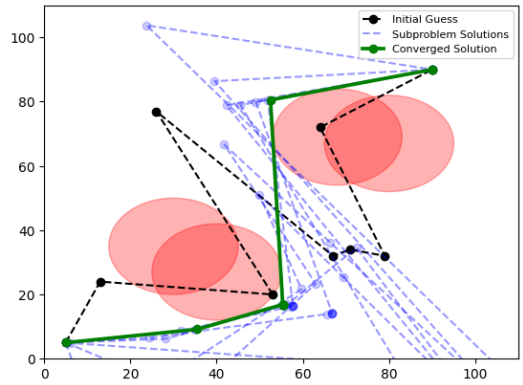
Figure 5: Clear globally sub-optimal solution. Converged cost = 152.2

we see that the fourth node in the initial guess became stuck in a local optima, and SCP was not able to move it to the other side of the obstacles which would have yielded a more optimal solution.

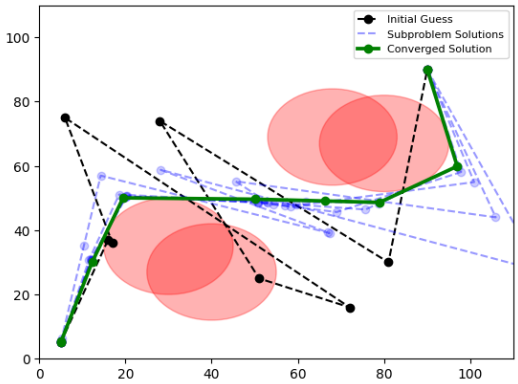
We can also observe that for identical constraints and number of nodes, we can converge to entirely different solutions. By varying the initial guess on a set of constraints, we get:



(a) Converged cost = 130.54



(b) Converged cost = 154.47



(c) Converged cost = 158.95

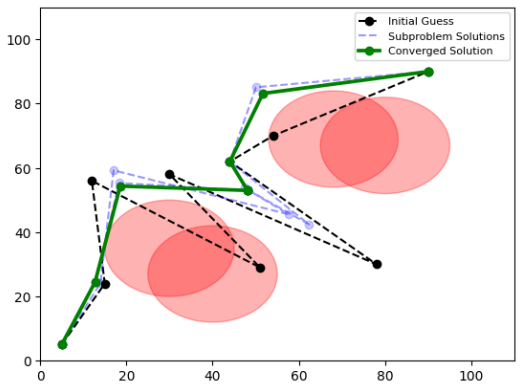
Figure 6: SCP generating different locally optimal solutions for identical constraint conditions

even with the aforementioned pre-conditioning of the initial guess, we still observed notable sensitivity of problem feasibility based on initial conditions. Executing the cell titled `Generate random results` in the Jupyter Notebook, we find that over 20 attempts, only 7 converged to a feasible solution, and they resembled one of the three examples shown above.

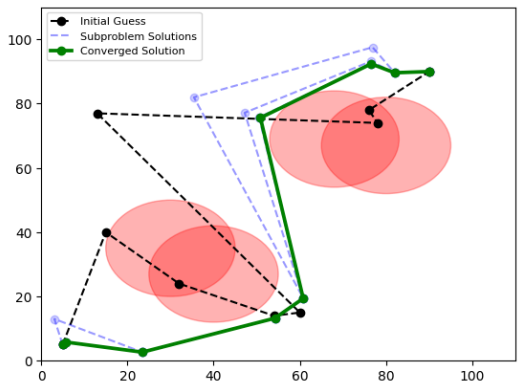
4.2 Revised Problem Formulation

4.2.1 Penalized Trust Region

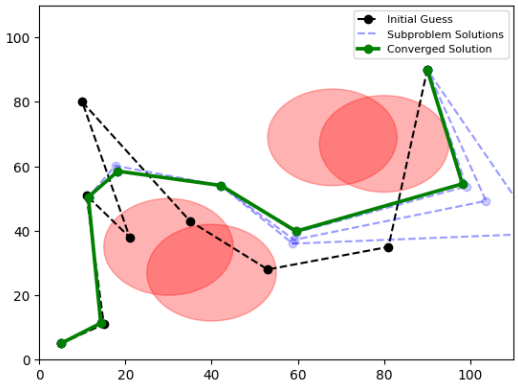
With the cost function adapted to equation (14), we repeat the random initialization procedure using the same conditions as those shown in figure 6 and observe:



(a) Converged cost = 152.59



(b) Converged cost = 162.53



(c) Converged cost = 185.48

Figure 7: Converged solutions for same conditions as figure 6 with PTR implemented

We see that with the revised cost function with PTR, the three distinct types of solutions from figure (6) can be somewhat replicated with much fewer convergence iterations (the convergence values for all three solutions above were first reached after 4-5 SCP iterations). Further, in another 20 random initializations of the same problem, a feasible solution was converged to in 15 cases, compared to 7 converged cases without PTR implemented.

While the PTR addition makes the SCP implementation noticeably more robust to random initial conditions, it also furthers the heuristic nature of SCP. We observe that the converged solutions display

higher convergence costs compared to their comparable solutions in figure 6, and we can clearly observe sub-optimalities, particularly in figure 7a, and overall node and link placements further away from the constraints.

The implementation also appears to be sensitive to the weight ρ of the PTR term. The above results were generated using $\rho = 1.1$, and a small change to $\rho = 2$ can yield:

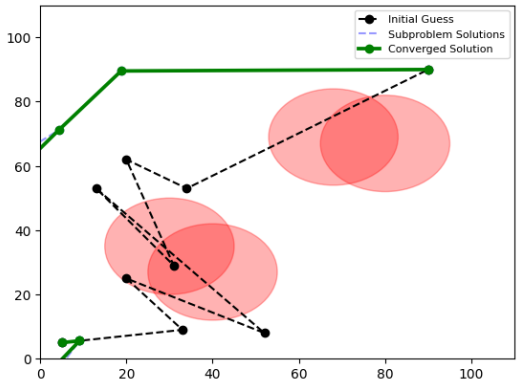


Figure 8: Converged solution with $\rho = 2$, cost > 400

clearly, the ρ term can be very important for the PTR method, and should be tuned carefully, if not optimized for as part of the problem as well.

Finally, as a lucky and fun touch, we observe that using PTR with a good initial guess can cause SCP to convere extremely rapidly (in one iteration):

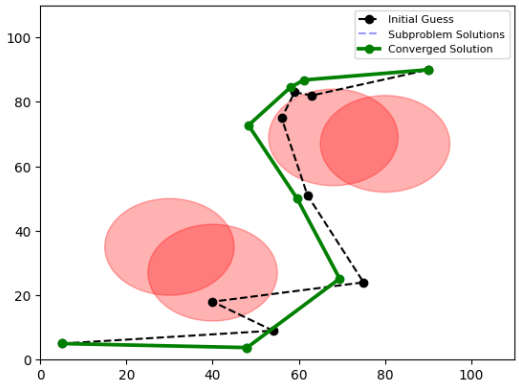


Figure 9: Single-shot converged solution with cost = 173.59

5 Conclusion

In application, SCP has been shown to be effective at a wide range of highly complex and non-convex problems. Some of these examples in the aeronautics/astronautics field include solving powered descent guidance [6] and real-time trajectory optimization and obstacle avoidance for quadcopters [7].

For this project, we focused on SCP as an algorithm, and learning and analyzing the behavior of SCP and the quality of generated results under a variety of conditions. As such, simplifications were made to the problem itself, compared to the cases of SCP presented above. For the problem, we focused on optimizing the final formation for a group of UAVs, with the assumption that the formation is reachable from a controls and dynamics and point of view. An extension to this project would be integrating the controls and dynamics of UAVs as constraints of the optimization problem itself. The dynamics are time dependent and need to be discretized in time and propagated throughout the problem to ensure that the trajectory required to reach the optimal solution is reachable by the UAVs. While this adds many layers of mathematical and computational complexity, the general underlying theory and methods presented in this project still stands. The solutions yielded by the algorithm itself can still be sensitive to tuned parameters and initial guesses, and the constraints can still be successively linearized/convexified using the methods used in this project, along with many others.

The results of this project mostly explored the sensitivity of SCP to initial guesses of the solution, as most other parameters (link discretization, PTR weight, convergence tolerance) were tuned to a certain extent, and then held constant. Results were also compared between constraining the feasible solution space to the explicit constraints themselves, and implicitly constraining certain algorithm choices by penalization in cost function. The observed feasibility and solution optimality behaviors broadly agree with the theory and mathematical reasoning presented in the methodology section.

In the works cited above of applied SCP and in many other cases, the problem itself is formulated in such a way that makes it appropriate for SCP. Parameters can be carefully optimized/chosen/tuned to specific problems, which itself can be an area of research and extension for this project. Initial guesses/conditions and parameters can be Monte Carlo'ed to measure the robustness of the SCP implementation, which would also be an interesting extension of this project. Other methods of convexifying the problem, other solvers, and alternative ways to pose the optimization problem using higher dimensional variables and constraints can all impact the solutions to this project, and possible areas of exploration beyond this submission.

In all, this project was a fun experience and great opportunity to explore SCP and apply it to a problem that is relevant to my personal research. During this process, many research questions and extension possibilities arose, and I am excited to pursue them after completing this project. Hopefully, this report is as interesting and enjoyable to read as it was for me to write it.

6 Appendix

The Jupyter Notebook used to generate all figures can be found at this link. The cells are appropriately titled based on the relevant sections and figures generated. The sections that utilize randomized initial node positions may yield infeasible problems as noted. The final cell is a fun addition that randomizes initial nodes and obstacle position and sizing.

References

[1] Behçet Açikmese, John M. Carson, and Lars Blackmore. Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem. IEEE Transactions on Control Systems Technology, 21:2104–2113, 2013.

[2] Giuseppe C. Calafiore, Laurent El Ghaoui, Laurent El, Ghaoui, Constantin Caramanis, Vu Duong, Michael Jordan, Jitendra Malik, Arkadi Nemirovksi, Yuri Nesterov, Jorge Nocedal, Kannan Ramchan-dran, Anant Sahai, Suvrit Sra, Marc Teboulle, and Lieven Vandenberghe. Optimization models. 2014.

[3] P. Elango, D. Luo, A.G. Kamath, S. Uzun, T. Kim, B. Açikmese, and M. Mesbahi. Successive convexification for trajectory optimization with continuous-time constrant satisfaction. 2024.

[4] Danylo Malyuta and Behcet Acikmese. Lossless Convexification of Optimal Control Problems with Semi-continuous Inputs. arXiv e-prints, page arXiv:1911.09013, Nov 2019.

[5] Danylo Malyuta, Michael Szmuk, and Behcet Acikmese. Lossless convexification of non-convex optimal control problems with disjoint semi-continuous inputs. arXiv e-prints, page arXiv:1902.02726, Feb 2019.

[6] Taylor P. Reynolds, Danylo Malyuta, Mehran Mesbahi, Behçet Açikmese, and John M. Carson. A real-time algorithm for non-convex powered descent guidance. AIAA Scitech 2020 Forum, 2020.

[7] Michael Szmuk, Carlo Alberto Pascucci, and Behçet Açikmese. Real-time quad-rotor path planning for mobile obstacle avoidance using convex optimization. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 1–9, 2018.