Noname manuscript No.
(will be inserted by the editor)

Real-Time Trajectory Generation for a Swarm of Quad-rotor UAVs using Custom Solver

Min Prasad Adhikari · Anton H. J. de Ruiter

Received: date / Accepted: date

Abstract The problem of real-time trajectory generation with autonomous obstacle and collision avoidance for a swarm of quad-rotors is studied in this paper. In particular, a centralized approach is utilized, and dynamical and control constraints are accommodated. The centralized approach has the benefit of utilizing all available information, in contrast to decentralized approaches. Therefore the problem has the potentiality to serve various important applications that demand reliability and safety in a swarm while addressing a quality task provided a central computer such as a ground station is available. However, computational complexity is one of the challenges that constrain the centralized approach for swarm scalability. Therefore, we study the computational aspect of this swarm problem. We consider a scenario for the swarm problem in which moving targets have been assigned for the swarm to follow along with a reference path for each UAV, where we solve the scenario in two different cases; the first, solve the entire swarm problem at once, and the second, solve iteratively for UAVs while considering trajectories from other UAVs as fixed. The scenarios defined form nonlinear programming problems, which we solve using the method of Finite Horizon Model Predictive Control. We have developed a custom solver for rapidly solving these problems. The comparative study between the two cases for the scenario indicates, the second approach of solving the swarm problem is almost four to seven times computationally faster than the first approach. However, this comes with the price that the found solution may not be completely optimal. In particular, we are able to achieve a 24.4 and 192.11 milliseconds solution time for a swarm of 6 and 32 quad-rotors through a cluttered environment respectively.

Keywords Quad-rotor · Quad-rotor Swarm · Custom Solver · Model Predictive Control

M. P. Adhikari, Ph.D. Student, Department of Aerospace Engineering
Ryerson University, 350 Victoria St, Toronto, ON M5B 2K3
Tel.: +1-437-344-0513
E-mail: minprasad.adhikari@ryerson.ca

A. H. J. de Ruiter, Associate Professor and Canada Research Chair in Spacecraft Dynamics and Control, Department of Aerospace Engineering
Ryerson University, 350 Victoria St, Toronto, ON M5B 2K3
E-mail: aderuiter@ryerson.ca

1 Introduction

The flexibility in speed and the complexity in a trajectory offered by a quad-rotor is a motivating factor for the present study of its swarm application. However, when compared to a quad-rotor, a swarm has a added level of complexity associated with its size; that is computability. Moreover, for a swarm of quad-rotors there may arise an application dependent complexity besides the above. As a model for a quad-rotor is a common between a swarm and a quad-rotor problem, we see that a choice of the UAV model plays a significant role in scalability of a swarm. In [1], the authors have used a simplified model of quadcopter for trajectory generation, which shows a low computation time. However, an intensity of computability increases in a swarm when an individual trajectory has to be generated in contrast to a swarm problem with a leader-follower mode [2]. Thus, real-time generation of trajectory for a large swarm of quad-rotors is still the topic of ongoing research.

In literatures we find various aspects of swarm problems being studied. Honig et al. used an approach that consists of three steps (road map generation, discrete

planning, and continuous refinement to smooth trajectory) for trajectory planning for quad-rotor swarm [3], however, it has no concern with real-time computation as it is a trajectory planning. The paper used a sequential convex programming (SCP) approach to convert a discrete plans from discrete planning steps to smooth trajectories. Although, the reference is yet to investigate the method for operating in an online generation and implementation settings, paper argues that the method is suitable for scaling the number of UAVs in a swarm. Kushleyev et al. studied the trajectory generation and collision avoidance in a swarm with agile micro quad-rotors [4]. The reference presents a formation flying where quad-rotors follow a trajectory generated offline, using mixed-integer quadratic program method to solve the problem. In [5], Luis et al. studied the offline trajectory generation method for a swarm using Distributed Model Predictive Control and argues the advantage against SCP method in terms of computation time. This is straight forward as the increasing number of UAVs in a swarm also increases the overall state and control variables (optimization variables) which in turn increase the size of an SCP problem. A mitigating approach to such scaled SCP has been proposed by [6] where the SCP problem for a swarm is decoupled per UAV in a swarm while considering other UAVs as obstacles. Thus, using the approach a trajectory is generated iteratively for a UAV at a time in a swarm. Although the reference presents a hardware test with four quad-rotors in their testbed, the computability of the approach for a swarm of quad-rotor remains unexplored as solution time are not provided. The approach uses model predictive control (MPC) and limited horizon SCP (associated with the MPC). In [7] Park et al. presented a case study of distributed formation flying where a central computer computes trajectory for all the UAVs, however, the solution time is not discussed. Similarly, in [8] Preiss et al. presented a study on control of a swarm of quad-rotor UAVs in a laboratory environment, where a combination of trajectory planning and online trajectory is been implemented, but does not discuss the computability of the problem. From the literature we understand that none of the citations mentioned above have explored the use of custom solver for a swarm application of quad-rotors, besides [3] which has used the CVXGEN [9] to address a small segment of a swarm problem that solves a support-vector machine problems to find safe corridors for the swarm application.

In order to present our approach for a swarm application, we define a swarm problem for quad-rotors as follows. Given an initial point for each quad-rotors in a swarm, the aim of the swarm is to minimize the distance to moving targets in an obstacle-rich environment such that all the UAVs have its own reference path to remain close to while satisfying their state and control bounds, including collision among themselves and avoiding obstacles with robustness. In addition we form two cases for the swarm problem. In the first case the entire swarm problem is solved at once, while in the second case the problem is solved iteratively for UAVs while considering trajectories from other UAVs as fixed. In the present study we aim to investigate the computational complexity associated with the swarm application. Also, to constrain ourselves from the broad topic of trajectory generation for a swarm of quad-rotors, we will only investigate the method of trajectory generation using the central controller approach. The central controller is a central computational system that computes trajectory for all the UAVs in a swarm rather than the trajectories being generated locally onboard each quad-rotor. One main assumption of using the central controller is the knowledge of the position of all the UAVs in a swarm. In this approach, the central controller generates collision-free trajectories for all the UAVs where the controller acts as a high level trajectory planner, and the respective UAV tracks the given trajectory. In the present study, a dynamic and a kinematic model for a quad-rotor are defined, where with certain assumptions the solution from the problem with kinematic model is used to compute inputs to the dynamic model so that we can realize a trajectory by integrating the inputs to the dynamic model. Although a slight different expression to compute the inputs to the dynamic model appeared in [10], it had the purpose of computing commanded roll, pitch and thrust for trajectory tracking, while in this paper we use a reduced form of expressions to retrieve inputs to the dynamic model from the kinematic counterpart. Thus, with an approximate computational complexity associated with the kinematic model we get the inputs for the quad-rotor dynamic model. We solve the above defined problems in a model predictive control (MPC) framework [11], with the same constraints and the cost function as in the original problem, where we use a sequential quadratic programming (SQP) approach as outlined in [12]. The increase in the number of quad-rotors in a swarm increases the problem size which eventually makes the computability of the swarm problem questionable for real-time applications. Hence, to lower the computation time as needed in real-time trajectory generation for a swarm of quad-rotors we use a custom solver(developed by the authors) that avoids unnecessary computations with zero and has the least possible memory requirement.

This paper is arranged in the following order. Section 2 presents problem formulations for swarm of quad-rotors that consist of simplification to a dynamic model for a quad-rotor UAV. Section 3 summarizes the approach used to solve the problems. The numerical examples are presented in Section 4 where we perform a comparable numerical study between the defined two problem formulations, and a Monte-Carlo test for the latter formulation including its scalability study. Finally, Section 5 concludes the paper.

2 Problem Formulation

2.1 Quad-rotor System Equations

2.1.1 A reduced dynamic model

As suggested by Feldman [13], a point mass model is the most detail model needed for a flight performance problem, as the motion of a vehicle is of interest. And, for our study of trajectory generation for a swarm of quad-rotors, we focus on their motion, irrespective of their heading angle. Thus, in order to work with a simple system equation for quad-rotor we reduce the complexity of a dynamic model for a quad-rotor UAV used for simulations in [14] by eliminating the need for heading angle. Also, for the problem of trajectory generation an absence of heading angle does not constraint a nature of flight trajectory a quad-rotor could take in $3D$ space, while pitch and roll angles are available. As such, we see Hehn et al. in [1] have suggested to set the rotational rates along $z-axis$ to be zero in order to compute controls for their model. Thus, conceivably for the purpose of trajectory generation we do not limit the position in space where a quad-rotor could reach by setting the heading angle (ψ) and its time derivatives to be zero. Therefore, we make the following assumptions,

– Assumptions: 1. $\psi = 0$, 2. $\dot{\psi} = 0$, and 3. $\ddot{\psi} = 0$.

Then using the assumptions, the reduced model for a quad-rotor becomes,

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \ddot{\phi} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\phi)\sin(\theta)\frac{T}{M} - \frac{1}{M}k_{dx}\dot{x} \\ -\sin(\phi)\frac{T}{M} - \frac{1}{M}k_{dy}\dot{y} \\ -g + \cos(\phi)\cos(\theta)\frac{T}{M} - \frac{1}{M}k_{dz}\dot{z} \\ \frac{\tau_\phi}{I_x} \\ \frac{\tau_\theta}{I_y} \end{bmatrix} \tag{1}$$

where (x, y, z) represents the position of a quad-rotor in three dimensional space. Euler angles (ϕ, θ) represent the angle of rotation about the x, and y axes respectively in an inertial frame in the anti-clockwise direction. T represents the total thrust due to four motors in a quad-rotor, where $T = F_F + F_R + F_B + F_L = k\sum_{i=1}^{4}\omega_i^2$. F_F, F_B, F_R, F_L represents force due to motors as shown in Figure 1, k is a force constant and ω_i refers to the angular velocity of motor i. k_{d_x}, k_{d_y} and k_{d_z} represent aerodynamic drag constants due to velocity along the x, y and z axes respectively. τ_ϕ, and τ_θ are the torques about each rotational axis due to thrust from the propellers, where $\tau_\phi = L\,(F_L - F_R) = Lk(\omega_1^2 - \omega_3^2)$, and $\tau_\theta = L(F_F - F_B) = Lk(\omega_2^2 - \omega_4^2)$. L is half the distance between two opposite rotors, M is mass of the quadcopter and b is the drag constant due to propeller, shown in Figure 1. The moments of inertia of the quadcopter about each respective axis are represented by I_x, I_y. Let us represent the states and the controls for the quad-rotor dynamic model by $X = [x, y, z, \phi, \theta]^T$ and $U = [\omega_1, \omega_2, \omega_3, \omega_4]^T$ respectively. Since the motors in a quad-rotor have a saturation point in the angular velocity, we define a limitation in thrust as, $T \leq T_{max}$ where $T_{max} > 0$. From this reduced model for a quad-rotor we see a possibility to further simplify a system equation for the UAV, where we could work with a kinematic model for the quad-rotor and still constraint the thrust produced by motors.

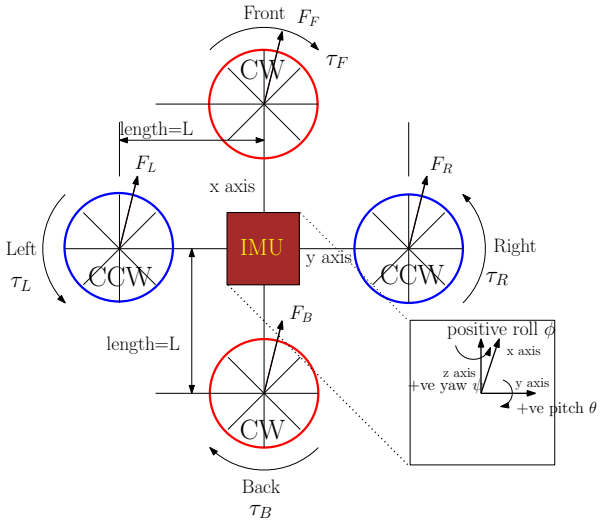

Fig. 1 Quad-rotor Model

2.1.2 A kinematic model

Let us define a kinematic model for a quad-rotor UAV as follows,

$$\begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{y} \\ \dot{v}_y \\ \dot{z} \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} v_x \\ a_x \\ v_y \\ a_y \\ v_z \\ a_z \end{bmatrix} \tag{2}$$

where x, y, z are positions in $3D$ space, v_x, v_y, v_z are velocities and a_x, a_y, a_z are accelerations along each axes x, y, z respectively. For the kinematic model, $X_k = [x, v_x, y, v_y, z, v_z]^T$, and $U_k = [a_x, a_y, a_z]^T$ represent state and control variables respectively.

Now, in order to get a thrust constraint for (2), we do the following assumptions. Since, $\ddot{x}$ in (1) and a_x in (2) both represent acceleration and $\dot{x}$ and v_x both represent velocity along $x - axis$ we assume $\ddot{x} = a_x$, and $\dot{x} = v_x$. Similarly, along the y and z-axes we assume $\ddot{y} = a_y$, $\ddot{z} = a_z$ and $\dot{y} = v_y$, $\dot{z} = v_z$. Then, using these assumptions we define the following,

$$A_x = \ddot{x} + \frac{1}{M} k_{dx} \dot{x} = a_x + \frac{1}{M} k_{dx} v_x \tag{3a}$$

$$A_y = \ddot{y} + \frac{1}{M} k_{dy} \dot{y} = a_y + \frac{1}{M} k_{dy} v_y \tag{3b}$$

$$A_z = \ddot{z} + \frac{1}{M} k_{dz} \dot{z} = a_z + \frac{1}{M} k_{dz} v_z \tag{3c}$$

Re-arranging the first three equations of (1) and then simplifying them using the expressions in (3) we get the following.

$$A_x = \cos(\phi) \sin(\theta) \frac{T}{M} \tag{4a}$$

$$A_y = -\sin(\phi) \frac{T}{M} \tag{4b}$$

$$A_z + g = \cos(\phi) \cos(\theta) \frac{T}{M} \tag{4c}$$

Now, to calculate the total thrust we square both the sides of equations in (4) and add them together. After simplification we obtain,

$$T = M \sqrt{A_x^2 + A_y^2 + (A_z + g)^2} \tag{5}$$

Then, we write the thrust constraint for (2) as,

$$M \sqrt{A_x^2 + A_y^2 + (A_z + g)^2} \leq T_{max} \tag{6}$$

Hence, once we solve a trajectory generation problem using the kinematic model, we use its solution (state and control inputs) to compute the control inputs for the reduced dynamic model using similar expressions from [10] but with simplifications for our reduced dynamics case. Thus computed inputs are then integrated into the system dynamics used in [14], using Runge-Kutta integration, to realize a trajectory.

2.2 Obstacle Avoidance Constraint

In order to address a trajectory generation problem in an obstacle-rich environment, we include the obstacle avoidance constraint as in [15],

$$h_i(x, y, z) = \ln \left[\left| \left(\frac{x - x_{\{c,i\}}}{a_i} \right)^{p_{x_i}} \right| + \left| \left(\frac{y - y_{\{c,i\}}}{b_i} \right)^{p_{y_i}} \right| + \left| \left(\frac{z - z_{\{c,i\}}}{c_i} \right)^{p_{z_i}} \right| \right] \geq \epsilon_s \tag{7}$$

where $x_{\{c,i\}}, y_{\{c,i\}}, z_{\{c,i\}}$ represent the centre of an obstacle i in three-dimensional spatial coordinates for $i = 1, ..., p$, where p is the number of obstacles, $p_{x_i}, p_{y_i}, p_{z_i} \geq 2$ are exponent terms for the x, y and z components respectively, and a_i, b_i, c_i represent the radius to an obstacle from its centre along x, y, z axes respectively. The constant $\epsilon_s > 0$ is used to ensure that the feasible region for the UAV is closed.

In order to ensure a UAV's safety from an obstacle and to increase the robustness in obstacle avoidance, a robustness function is augmented to the cost of a trajectory generation problem to be defined next, introduced in [16] but with the further modification we write as follows,

$$\overline{r}(X(\cdot)) = \sum_{i=1}^{p} w_i(x, y, z)\, e^{e^{\left(\epsilon_s - \overline{h_i}(x,y,z)\right)}} - 1 \tag{8}$$

where w is a weighting factor also known as robustness factor and it given by,

$$w_i(x, y, z) = q \cdot g \left(\left| \left(\frac{x - x_{\{c,i\}}}{a_i} \right) \right|^{p_{x_i}} + \left| \left(\frac{y - y_{\{c,i\}}}{b_i} \right) \right|^{p_{y_i}} + \left| \left(\frac{z - z_{\{c,i\}}}{c_i} \right) \right|^{p_{z_i}} \right) \tag{9}$$

$$\overline{h_i}(x, y, z) = \ln \left[\left(\frac{x - x_{\{c,i\}}}{r_i} \right)^2 + \left(\frac{y - y_{\{c,i\}}}{r_i} \right)^2 + \left(\frac{z - z_{\{c,i\}}}{r_i} \right)^2 \right] \tag{10}$$

where $r_i = max(a_i, b_i, c_i)$, and $\overline{h_i}$ defines expression related a sphere with radius equal to the maximum radial dimension of an obstacle. In equation (9) $q > 0$, and $g(\cdot)$ is defined as,

$$g(t) = \frac{f(r_b - t)}{f(r_b - t) + f(t - r_a)} \tag{11}$$

for suitably chosen $r_a < r_b$, $t \in (-\infty, \infty)$, and $f(t)$ is defined as,

$$f(t) = \begin{cases} 0 & t \in (-\infty, 0] \\ \exp(-1/t), & t \in (0, \infty) \end{cases} \tag{12}$$

This function is $\mathcal{C}^\infty$, and its derivative is given by

$$df(t)/dt = \begin{cases} 0 & t \in (-\infty, 0] \\ (1/t^2) \exp(-1/t), & t \in (0, \infty) \end{cases} \tag{13}$$

We get $g(t) = 1$ for $t \leq r_a$, and $g(t) = 0$ for $t \geq r_b$. Since for $t < 1$ corresponds to points inside an obstacle, we require $1 < r_a < r_b$, to ensure the weight is non-zero in a region outside the obstacle.

2.3 Collision avoidance constraint

In order to avoid collision among quad-rotors, we use the collision avoidance constraint for a swarm as in [17],

$$\left\| \begin{bmatrix} x^k - x^j \\ y^k - y^j \\ z^k - z^j \end{bmatrix} \right\| \geq R_{CA}, \ \forall k \neq j \tag{14}$$

where k and j refer to two different UAVs in a swarm, and R_{CA} is the radius of collision avoidance which also represents the centre distance between two UAVs.

2.4 Swarm problems with moving targets

We now introduce swarm problem of our interest. Given initial points for a swarm of quad-rotor we focus on the trajectory generation problem where each quad-rotor has a reference path to remain close to while all the quad-rotors in the swarm aim to reach their respective moving targets by avoiding collision among themselves and maneuvering around obstacles with robustness, and satisfying state and control constraints. However, the moving target in our problem is not like a definitive goal point that forms an equality constraint to represent the end-point constraint. Instead, it forms the cost to minimize the distance to a target plane (for example, $x = 300$, can represent a target plane at 300 meters along $x - axis$), which up on intersection with a reference path (for example, $y = 20$ and $z = 20$ can represent a reference path) forms a target point. The purpose of the cost function is to drive/fly quad-rotors along the desired direction, rather than a hard constraint as the end-point constraint.

One significant advantage of replacing a definitive goal point with the cost is, even if there is an obstacle at a target point the problem still becomes end-point feasible. For example, let us suppose a swarm of quad-rotor has to reach an imaginary moving target point in an obstacle-rich environment, and the moving target passes through multiple obstacles. To deal with such a scenario, to be able to fly the swarm safely, avoiding obstacles and remain close to the target point, we use the cost but not the end-point equality constraint. Alternatively, for such a scenario we may think of an end-point inequality constraint, such that the problem becomes end-point feasible if the inequality in the problem is satisfied when a quad-rotor in a swarm reaches near the target radius (suppose the target point lies inside an obstacle). However, we cannot define a general inequality constraint that can accommodate all the unknown obstacles in various scenarios. The problem as well serves applications that has a target point far enough or may not be reachable. Such as, a swarm chasing a suspect in city like environment, or a swarm flying to a destination for aerial cover; where the destination is a stationary target.

Figure 2 shows a representation for this type of scenario where two quad-rotors aim to reach their respective moving target point along their reference path towards the desired direction of flight. As shown in the figure, both the reference path pass through obstacles along which the target point moves. Such scenario is best suited when quad-rotors in a swarm are required to progress equally in a desired direction if moving targets have the same velocity lower than the UAVs velocity (when obstacles are absent or after avoidance of obstacles). As stated in the introduction, we study the

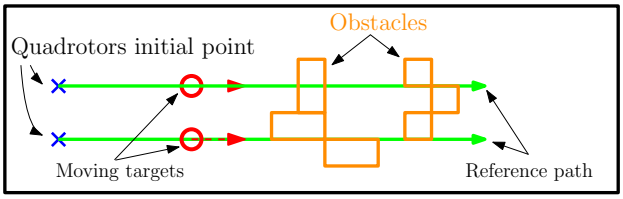

Fig. 2 Swarm problem with moving targets

swarm problem in two cases. In the first case, we define a problem that solves for the entire swarm at once. Whereas, in the second case we iteratively solve for an agent at a time while considering the trajectories for other UAVs as fixed.

2.4.1 Case A: solve the entire swarm problem at once.

Let us suppose there are N_s number of quad-rotors in a swarm. Let $\left(X^j, U^j\right)$ represents the state–control pair for a j^{th} quad-rotor in the swarm. Let $y^j\left(t\right)$ and $z^j\left(t\right)$ represent the position of a j^{th} quad-rotor along y and z axes at time t, and let $y^j\left(t\right) = y_d^j$ and $z^j\left(t\right) = z_d^j$ are two planes representing a desired path for the quad-rotor, where y_d^j and z_d^j are constants. Also, let us define

$$X_m^j\left(t\right) = \left[x\left(t\right)_m^j, \dot{x}\left(t\right)_m^j, y\left(t\right)_m^j, \dot{y}\left(t\right)_m^j, z\left(t\right)_m^j, \dot{z}\left(t\right)_m^j\right]^T$$

as the state of a moving target at time t associated with a quad-rotor j. We also define a term that represents the rate of change of the control for j^{th} quad-rotor in a swarm as, $\dot{U}^j\left(\cdot\right)$. Then, we define the cost function to the trajectory generation problem for a swarm of quad-rotors with their respective moving target points to avoid collision and obstacles, and remain close to their respective reference path while flying from (x_0^j, y_0^j, z_0^j) at time τ_0, up to final time τ_f as,

$$J(X\left(\cdot\right), U\left(\cdot\right)) = \sum_{j=1}^{N_s}\left[\int_{\tau_0}^{\tau_f} F\left(X^j\left(t\right), U^j\left(t\right), t\right) dt\right] \tag{15}$$

where $X = [X^{1T}, ..., X^{N_sT}]^T$, $U = [U^{1T}, ..., U^{N_sT}]^T$, and $j = 1, 2, ..., N_s$.

$$F\left(X^j(t), U^j(t), t\right) = \begin{pmatrix} \alpha\left(y^j(t) - y_d^j\right)^2 + \\ \beta\left(z^j(t) - z_d^j\right)^2 + \eta\left(X^j(t) - X_m^j(t)\right)^2 + \\ \overline{r}\left(X^j(\cdot)\right) + \zeta\left(\dot{U}^j(t)\right)^2 \end{pmatrix} \tag{16}$$

where the term $\left(y^j(t) - y_d^j\right)^2$ in the cost function minimizes the square of deviation from the plane $y^j(t) = y_d^j$, and $\alpha > 0$ is the penalty term associated with the term. Similarly, $\beta > 0$ is the penalty term associated with $\left(z^j(t) - z_d^j\right)^2$ that minimizes the square of deviation from the plane $z^j(t) = z_d^j$. Likewise, $\eta > 0$ is associated with the cost that minimizes the square of difference between the moving target and UAVs, and the term $\overline{r}(X)$ is the robustness function as in (8). Furthermore, we have added the square of the rate of change of controls in the cost as, $\left(\dot{U}^j(t)\right)^2$, by minimizing which we minimize the chances of jerky controls, and $\zeta > 0$ is the penalty term associated with that.

Now we define the trajectory generation problem for a swarm of quad-rotor as, given an initial condition (x_0^j, y_0^j, z_0^j) for a j^{th} quad-rotor at time τ_0, find the control-state pair $(X(t), U(t))$ to minimize the cost (15) subject to

$$\begin{gathered} \text{Quad-rotor model (2)+} \\ \text{Obstacle constraint (7)+} \\ \text{Collision constraint with } k^{th} \text{ quad-rotor (14)+} \\ \text{thrust limitation (??)} \\ X = \left[X^{1T}, ..., X^{N_sT}\right]^T \\ U = \left[U^{1T}, ..., U^{N_sT}\right]^T \\ j, k = 1, ..., N_s \end{gathered} \tag{Problem 1}$$

2.4.2 Case B: solve iteratively for UAVs in a swarm

The most of the required components of the problem definition are already detailed above, thus, we briefly define the additional component to this second case problem. In this case, we consider an approach of iteratively solving for UAVs in a swarm while considering trajectories from other UAVs as fixed, thus the cost function to the second case differs from the first case. In the second case we define the cost function for the j^{th} quad-rotor as,

$$J(X^j(\cdot), U^j(\cdot)) = \int_{\tau_0}^{\tau_f} F\left(X^j(t), U^j(t), t\right) dt \tag{17}$$

where all the parameters are as defined in (15). Then, the problem for this case is defined as, given an initial condition (x_0^j, y_0^j, z_0^j) for a j^{th} quad-rotor at time τ_0, and trajectories of all other UAVs in the swarm as fixed, find the control-state pair $\left(X^j(t), U^j(t)\right)$ for j^{th} UAV in the swarm, to minimize the cost (17) for a j^{th} quad-rotor subject to

$$\begin{gathered} \text{Quad-rotor model (2)+} \\ \text{Obstacle constraint (7)+} \\ \text{Collision constraint with } k^{th} \text{ quad-rotor (14)+} \\ T^j \leq T_{max} \\ k = 1, ..., N_s \end{gathered} \tag{18}$$

Once we have defined (18), we solve the swarm problem as follows,

$$\begin{gathered} \text{for } j = 1 : N_s \\ \quad \text{Solve (18)} \\ \quad \text{Update } X, U \\ \text{end} \end{gathered} \tag{Problem 2}$$

Using a MPC framework we solve the swarm problem (18), in which X^j and U^j are design variables while the trajectories for the other UAVs are considered fixed. At each MPC window, we solve (Problem 2) iteratively for each UAV where we initialize the problem at the solution of previous MPC window. We use SQP approach to solve (18) in (Problem 2) where we have set the maximum number of SQP iteration to be 5.

3 The Approach

Now we outline the method to solve the above problems. Although, we may solve the problems for entire trajectory, the appearance of new obstacles during flight anyway requires re-solving of the problem. Therefore, it is not necessary to solve for the entire trajectory, rather a portion of the trajectory over an immediate short window is enough. Thus, we solve the problem using the finite horizon model predictive control (MPC) approach [11], with the same constraints and the cost function as the original problems. In order to solve the problems, they are first discretized at the sampling time up to the finite horizon as in [18,19]. Then, the discretized problems are approximated with a quadratic programming (QP) problem in which we do the second order Taylor approximation to the cost function and the first order Taylor approximation to constraints. To solve the QP problem we use the sequential quadratic programming (SQP) approach as outlined in [12]. The SQP algorithm solves an NLP problem by solving sequences of QP sub-problem within it, in which the QP sub-problem is solved using a custom solver.

The custom solver is generated using a custom solver generation program which is developed by the authors to solve QP problems with the positive semi-definite Hessian, in a computationally efficient way [20]. Although, custom solver is generated only for QP problems with the positive semi-definite Hessian, there may arise a QP problem within the SQP approach which may have the indefinte Hessian to the overall cost function due to the robustness function (8) used. In order to deal with such an indefinite Hessian case, we take an approach which is similar to infeasible start Newton method ([21], Chapter 10.3), where by choosing an appropriate slack variables in the robustness function the overall cost Hessian could be made very close to positive semi-definite, resulting in the use of custom solver. A custom solver generated using the custom solver generation program has a fixed known memory, hardcoded in C programming language. It is also free from programming overhead or library function (such as functions for matrix operations). The aim of using a custom solver, rather than a general QP solver is to eliminate the unnecessary requirement for storing and computation with zeros of a large sparse matrix (the Hessian and Jacobian) of a QP problem.

In terms of stability to the MPC problem, the standard method of establishing the closed-loop stability does not apply due to the absence of end-point cost or constraints [22]. However, under certain controllability conditions the MPC trajectory converges even without an end-point cost, provided the long enough prediction horizon in comparison to the control window, as demonstrated by Reble in [23]. As a recent case-specific example for a fixed-wing UAV, Alexander Joos has shown that a sufficient finite horizon could yield asymptotic stability of MPC problem [24]. In this paper we do not investigate the theoretical aspects of stability for the MPC problems as our focus is on computational aspects. However, we will numerically demonstrate it.

4 Numerical Example

We consider an example with six quad-rotors in a swarm that fly along their respective reference path towards the positive $x-axis$, with the aim of minimizing the distance to their respective moving targets. The parameters for quad-rotors are taken from Table 1, while the starting point for each quad-rotor in the swarm, along with their respective reference paths are given in Table 2. The parameters for this example are taken from Table 3. For the example scenario we take seven stationary obstacles distributed around the space such that reference paths for quad-rotors are obstructed, and two moving obstacles such that they intersect quad-rotors on their way. The starting point for moving targets is $(x_a = 20)\ m$. The targets move along the positive $x-axis$ with the velocity vector $(3, 0, 0)\ m/s$, and we stop the simulation once the targets cross $x = 300\ m$.

Table 1 Quad-rotors Parameters

Parameters	values	Parameters	values
$g\ (m/s^2)$	9.81	K_{dz}	0.25
$M\ (kg)$	0.468	$k\ (Nm\ s^2/rad^2)$	$1.14 * 1e-7$
$L\ (m)$	0.255	$b\ (N\ s^2/rad^2)$	$2.98 * 1e-6$
K_{dx}	0.25	$I_z\ (kg\ m^2)$	$8.801 * 1e-3$
K_{dy}	0.25	$I_x = I_y\ (kg\ m^2)$	$4.856 * 1e-3$

Table 2 Starting points for quad-rotors (units in meter)

Quad-rotor(j)	Initial Point (x_0^j, y_0^j, z_0^j)	y_d^j	z_d^j
1	$(0, 100, 20)$	100	20
2	$(0, 100, 40)$	100	40
3	$(0, 150, 20)$	150	20
4	$(0, 150, 40)$	150	40
5	$(0, 200, 20)$	200	20
6	$(0, 200, 40)$	200	40

Table 3 Problem 1: Parameters

Parameter	Value	Parameter	Value
$\alpha,\ \beta$	0.5, 0.5	T_{max}	$2 \times M \times g$
η	0.1	N	20
ζ	0.2	Δt(sampling time)	0.3 s
$[\tau_0, \tau_f]$	$[0, 6]$ s		

All the problems in this paper are solved using a laptop computer with 1.8 GHz (Intel Core i5) processor and 4 GB of (1600 MHz DDR3) RAM. Figure 3 shows the resulting trajectories with both the cases overlapped where the trajectories in the red colour represent the results from (Problem 1), while the trajectories in the black colour represent the results from (Problem 2). In the figure, the starting points and the target points along the reference path are indicated for each UAV, including trajectories belonging to each UAVs are shown as well with the even numbered UAVs at the top while the odd numbered UAVs at the bottom. In among the moving obstacles the speed have been indicated above them, where the moving obstacle starting from $y = 300$ m at the height of 20 m and goes towards $x-axis$ with 5 m/s, while the other moving obstacle starting from same height at $y = 0$ moves along the positive $y-axis$ with 3 m/s. As indicated in the figure, each UAV has been given a reference path represented by the green dashed line. In the figure, trajectories for UAV 4 between the problems almost overlaps, while for UAV 2, and UAV 6 the trajectories between the problems are very close. Whereas, for UAV 1, UAV 3, and UAV 5 the trajectories between the problem do not overlap and

this align with the propositions from [25] where in the case of obstacles the trajectories between the problems may not be the same. Moreover, we note that the trajectories between the UAVs may not progress equally while avoiding obstacles (in the presence of obstacles), however, after obstacles have been avoided the UAVs lagging behind speed up to catch-up with other UAVs. As the UAVs could speed up to 5 m/s while moving targets are moving at 3 m/s, all UAVs in the absence of obstacles can catch–up with their respective moving target and thus progress equally in the direction of flight. Figure 4 compares the solution time between

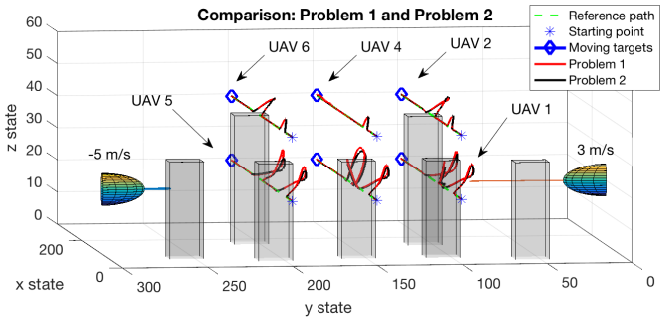

Fig. 3 Comparison of resulting trajectories

(Problem 1) and (Problem 2) at each MPC time instant. In the figure, the solution time for a UAV within a MPC instant in (Problem 2) is approximately 4 ms in average with maximum of 9.4 ms, resulting in the total solution time for the swarm to 24.4 ms at the maximum at a MPC instant, and 17.8 ms at the minimum. This is mainly due to the reduction in problem size where a QP sub-problem within the (Problem 2) solves less complex problem compared to the case of (Problem 1), where the solution time for (Problem 1) is more than 104.7 ms for all the instances, with the maximum of 176.5 ms of a MPC instant. In terms of the computational complexity of the problems, (Problem 1) has 35994 nonzero entries in the KKT matrix within its QP sub-problem, whereas the problem within (Problem 2) has 6189 nonzero entries. While both (Problem 1) and (Problem 2) solve the swarm scenario, the formulation in (Problem 2) has an advantage in terms of solution time as seen in Figure 4. Thus, to examine the solution time for range of scenarios, in the following a Monte-Carlo test for (Problem 2) is presented.

4.1 Monte-Carlo test

The computational result of (Problem 2) shown above may not be a sufficient demonstration of real-time computability of the swarm problem. Therefore, in this section we perform a Monte-Carlo test for the problem

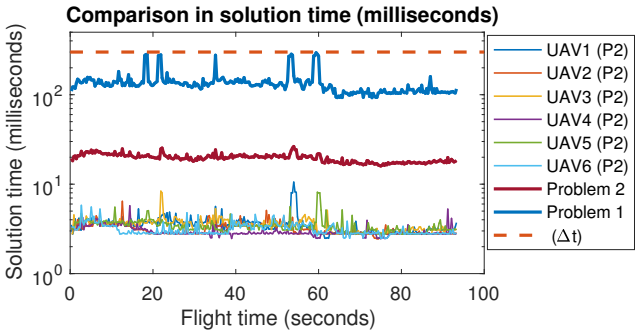

Fig. 4 Comparison of solution time

with 100 scenarios. In any scenario among 100, the number of obstacle ranges from 2 to 20, obstacles parameters (in (7)) a_i, b_i, and c_i range from 20 to 50 m, and p_x, p_y, and p_z range from 2 to 10. The obstacles parameters that determines its location and shape are randomly initialized with uniform distribution. Obstacles are distributed in the space between 100 m to 800 m along the positive $x-axis$ (not at 0 so as to avoid obstacles near the initial point), -300 m to 300 m along the y-axis and 0 m to 400 m along the z-axis. By choosing such distribution, we expect that the obstacles may fall in or along the desired reference path and as a consequence, an avoidance maneuver is necessary for safe flight. In any scenario initialized there are at least one stationary and one moving obstacle, and at the maximum, we can have ten stationary and ten moving obstacles. Moving obstacles in the scenario are initialized with velocity V_O, and angles ψ_O and γ_O which are angles about the positive $x-axis$ and above the $x-y$ plane respectively. These moving obstacle parameters are drawn from ranges $V_O \in [0.5, 5.55]$ m/s, $\psi_O \in [-\pi, \pi]$ rad, and $\gamma_O \in [-\pi/2, \pi/2]$ rad. For the test, the following kinematics is used for a moving obstacle.

$$\begin{bmatrix} \dot{x}_O \\ \dot{y}_O \\ \dot{z}_O \end{bmatrix} = \begin{matrix} V_O \cos(\psi_O)\cos(\gamma_O) \\ V_O \sin(\psi_O)\cos(\gamma_O) \\ V_O \sin(\gamma_O) \end{matrix} \tag{19}$$

where x_O, y_O, and z_O represent the position of an obstacle. In order to have a meaningful way to terminate the simulation and analyse the results, we stop the simulation for each scenario once the moving target crosses $x = 800$ m. In the following we summarize the results.

Figure 5 shows the statistical result of solution time for (Problem 2), in which almost all of instances takes two ranges, 2–4 and 4–6 milliseconds. Although the solution time for the swarm is much important than for a UAV in the swarm, but for the approach we take the total solution time for the swarm are most likely to be at the multiple of the solution time for a UAV, due to the same sized problem to be solved iteratively. The figure

shows that for a little over 20% of the instances in the Monte-Carlo test, custom solver takes 2–4 milliseconds of solution time for a quad-rotor, while for approximately 76% of the instances the sovler takes from 4–6 ms of solution time. The number of nonzero entries in the KKT matrix of a QP problem posed in this test is 8279 (the custom solver is generated to account up to 20 obstacles), as such the computational complexity of this problem seems roughly as similar to the example problem above with the variation of 1–2 ms solution time.

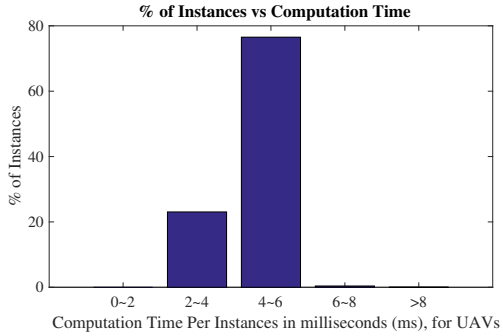

Fig. 5 Computation time per instances for quad-rotors

Finally, Figure 6 shows that, all the instances in the Monte-Carlo test are solvable within 30 milliseconds using the custom solver in the laptop computer.

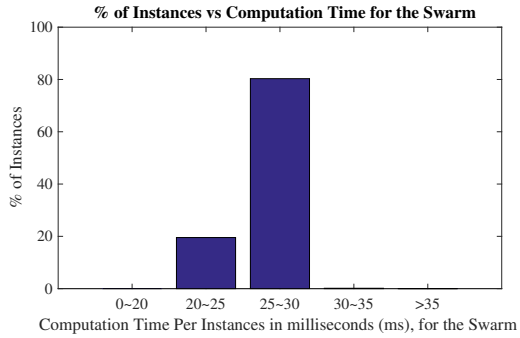

Fig. 6 Computation time per instances for the swarm

4.2 Scalability study

Encouraged by the solution time of the approach in (Problem 2), we study its scalability with different number of UAVs the swarm. For this study we use the same obstacle configuration as in the numerical example at the beginning of this Section, whereas the starting point for UAVs are distributed along $y-axis$ and $z-axis$ with equal distances between them at $x=0$. Moreover, a reference path for each UAV are defined such that it extends parallel from the starting point of the UAV along $x-axis$. In addition to the swarm study with 6 quad-rotors, we solve the swarm problem with $16, 32$ and 64 quad-rotors and compare the solution time. For the case of 16 UAVs, the starting point are assigned 30 m apart along y and $z-axis$ with the first one at $(0, 30, 30)$ m, while for the swarm problem with 32 and 64 quad-rotors, the starting point for UAVs are equally spaced at 20 m apart along y and $z-axis$ with the first one at $(0, 20, 20)$ m. As similar to the example at the beginning of this Section, we terminate the simulation once the moving target crosses $x=300$ m.

Figure 7 shows the solution time for the swarm problems. While the solution time for the swarm increases with the increasing number of UAVs, it is notable from Table 4 that the average solution time for (18) does not scale linearly with the increasing number of UAVs in the swarm. From the study we note the following, the maximum solution time of a MPC time instance to solve the swarm problems with 6, 16, 32 and 64 UAVs are 24.44, 74.32, 192.11, and 554.99 milliseconds respectively. It is also notable that the laptop computer is at least able to generate real-time trajectory for up to 32 quad-rotor UAVs within the allocated sampling time which is set to 300 milliseconds. Table 4 presents the

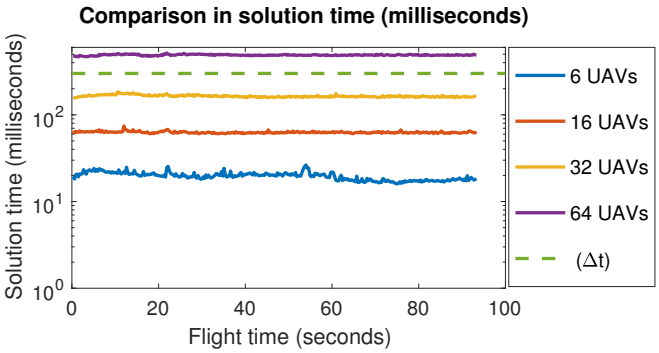

Fig. 7 Computation time for various swarm size

average solution times along with problem complexity for comparison. In the table, Avg_{tq} represents the average solution time for (18), Avg_{tsw} refers to the average solution time for Problem (Problem 2), and $Nnz_{problem}$ is the number of non-zero in the KKT system of a QP sub-problem within the (Problem 2), this also represents the problems computational complexity as the solution time substantially depends on the amount of non-zero in the KKT system. In the table, we see that the Avg_{tq} does not grow linearly as it is directly related to $Nnz_{problem}$, however Avg_{tsw} grows little over the linear rate. It is also worth noting that the problem size does not double up at doubling the number of quad-rotors in the swarm.

Table 4 Swarm problems with their average solution time

Quad-rotors	Avg_{tq} in ms	Avg_{tsw} in ms	$Nnz_{problem}$
6	3.49	20.94	6189
16	3.94	63.01	7937
32	5.17	165.48	10977
64	7.65	489.75	17057

5 Conclusion

A swarm problem with moving targets is studied. Utilizing a centralized approach, swarm problems are solved that accommodated the dynamical, and the control constraints. Among the two cases considered for the present study, the second case that solves for UAVs iteratively within a MPC time instant is computationally faster than the first case that solves for the entire swarm at a MPC time instant. In particular, we were able to achieve 24.4 milliseconds of solution time for a swarm with 6 quad-rotors using the second approach. From the Monte-Carlo test for the swarm with 6 UAVs, we note the consistency in the solution time where the swarm problem with the second approach is solvable within 30 milliseconds. Results from the scalability study show that the swarm problem with 32 quad-rotors are solvable within the sampling time in the laptop computer.

References

1. Markus Hehn and Raffaello D'Andrea. Quadrocopter trajectory generation and control. In *18th IFAC World Congress*, Milano (Italy), August 28 - September 2 2011.
2. Michael James Campobasso. Leader-follower trajectory generation and tracking for quadrotor swarms. Master's thesis, Embry-Riddle Aeronautical University, Daytona Beach, Florida, April 2017.
3. Wolfgang Honig, James A. Preiss, T.K. Satish Kumar, Gaurav S. Sukhatme, and Nora Ayanian. Trajectory planning for quadrotor swarms. *IEEE TRANSACTIONS ON ROBOTICS*.
4. Alex Kushleyev, Daniel Mellinger, and Vijay Kumar. Towards a swarm of agile micro quadrotors. In *Robotics: Science and Systems*, pages 9–13, Sydney, NSW, Australia, July 2012.
5. Carlos E. Luis and Angela P. Schoellig. Trajectory generation for multiagent point-to-point transistions via distributed model predictive control. *IEEE Robotics and Automation Letters*, 2018.
6. Daniel Morgan, Giri P Subramanian, Soon-Jo Chung, and Fred Y Hadaegh. Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and sequential convex programming. *The International Journal of Robotics Research*, 35(10):1261–1285, 2016.
7. Myoung-Chul Park, Sung-Mo Kang, Jae-Gyeong Lee, Gwi-Han Ko, Koog-Hwan Oh, Hyo-Sung Ahn, Young-Cheol Choi, and Ji-Hwan Son. Realization of distributed formation flying using a group of autonomous quadcopters and application to visual performance show. In *IEEE Transportation Electrification Conference and Expo, Asia-Pacific (ITEC)*, Busan, Korea, June 1-4 2016.
8. James A. Preiss, Wolfgang Honig, Gaurav S. Sukhatme, and Nora Ayanian. Crazyswarm: A large nano-quadcopter swarm. In *IEEE International Conference on Robotics and Automation (ICRA)*, Singapore, May 29 - June 3 2017.
9. Jacob Mattingley. Cvxgen: a code generator for embedded convex optimization [webpage], [Online: Accessed on 31 October 2017].
10. Z. Zuo. Trajectory tracking control design with command-filtered compensation for a quadrotor. *IET Control Theory and Applications*, 4(11):2343–2355, 2010.
11. P. Falcone, M. Tufo, F. Borrelli, J. Asgari, and H.E. Tseng. A linear time varying model predictive control approach to the integrated vehicle dynamics control problem in autonomous systems. In *Proceedings of the 46th IEEE Conference on Decision and Control*, New Orleans, LA, USA, Dec. 2007.
12. Richards H. Bryd, Jean Charles Gilbert, and Jorge Nocedal. A trust region method based on interior point techniques for nonlinear programming. *HAL Archives*, (00073794):44, May 2006.
13. Michael A. Feldman. Efficient low-speed flight in a wind field. Master thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, July 1996.
14. Teppo Luukkonen. Modelling and control of quadcopter. *Independent research project in applied mathematics*, August 22 2011.
15. L. P. R. Lewis. *Rapid motion planning and autonomous obstacles avoidance for unmanned vehicles*. PhD thesis, Naval Postgraduate School, Dec 2006.
16. M. A. Hurni, P. Sekhavat, and I. M. Ross. Autonomous trajectory planning using real-time information updates. *AIAA Guidance, Navigation and Control Conference and Exhibit, Honolulu, Hawaii*, August 18-21 2008.
17. Daniel Morgan, Soon-Jo Chung, and Fred Y. Hadaegh. Model predictive control of swarms of spacecraft using sequential convex programming. volume 37. JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS, November-December 2014.
18. Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY*, 18(2):267–278, March 2010.
19. D. Lam, C. Manzie, and M. Good. Model predictive contouring control. *IEEE Conference on Decision and Control*, pages 6137–6142, Dec. 2010.
20. Vandenberghe L. The cvxopt linear and quadratic cone program solvers [online]. March 2010. [Accessed: 15 March 2017].
21. Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2009.
22. Mayne D. Q., Rawlings J. B., Rao C. V., and Scokaert P. O. M. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, June 2000.
23. Marcus Reble. *Model Predictive Control for Nonlinear Continuous-Time Systems with and without Time-Delays*. PhD thesis, Institut für Systemtheorie und Regelungstechnik, Universität Stuttgart, Stuttgart, February 2013.
24. Alexander Joos. *Real-Time Predictive Motion Planning for Fixed-Wing Aerial Vehicles*. Phd thesis, Universität Stuttgart, February 2014.
25. Daniel Morgan, Soon-Jo Chung, and Fred Y. Hadaegh. Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and model predictive control. Kissimmee, Florida, 5-9 January 2015. AIAA Guidance, Navigation and Control Conference.