Abstract—This paper develops an adaptive PID autotuner for multicopters, and presents simulation and experimental results. The autotuner consists of adaptive digital control laws based on retrospective cost adaptive control implemented in the PX4 flight stack. A learning trajectory is used to optimize the autopilot during a single flight. The autotuned autopilot is then compared with the default PX4 autopilot by flying a test trajectory constructed using the second-order Hilbert curve. In order to investigate the sensitivity of the autotuner to the quadcopter dynamics, the mass of the quadcopter is varied, and the performance of the autotuned and default autopilot is compared. It is observed that the autotuned autopilot outperforms the default autopilot.

I. INTRODUCTION

Unmanned aerial vehicles, especially multicopters, have been used in a myriad of applications over the last decade, such as environment mapping, asset monitoring, risk assessment, sports broadcasting, wind-turbine inspection, and their applications continue to grow [1], [2], [3], [4], [5], [6]. In its most common form, a multicopter has four propellers. By controlling the spin rates of the four propellers, a force along a body-fixed axis and moments about three linearly independent body-fixed axes can be independently applied to affect desired translational as well as rotational motions. However, due to the nonlinear and unstable nature of the quadcopter dynamics, precise control of a quadcopter is a well-recognized challenging problem.

Nonlinear techniques such as feedback-linearization [7] and back-stepping [8] have been applied to deal with the nonlinearities and to construct stabilizing controllers. However, these techniques require accurate plant models at all operating conditions [9]. Adaptive techniques have also been investigated to reduce the need for an accurate model [10], [11], however, they also require a sufficiently accurate plant model to construct stabilizing controllers. Iterative learning control is used learn the pitch and roll controller in the closed-loop system in [12]. Reinforcement learning control is used to learn low-level controllers in case of multiple actuator failure in [13]. L1 adaptive control is used to improve the stability margins of a stable control loop in a quadcopter flight control system in [14]. Fuzzy control was used in the position control in [15]. Bidirectional brain emotional learning was used to improve trajectory tracking and handle payload uncertainties in [16]. Model reference adaptive control was used in the attitude controller in [17] to counteract model uncertainties. Adaptive twisting sliding mode control was used in the attitude controller in [18] to resolve the chattering issues in standard sliding mode control. Robust fixed point transformation based adaptive control was used in [19] to improve quadcopter stability in the presence of parameter uncertainties and external disturbances. Adaptive particle swarm optimization was used in a PD controller (APSO-PD) in [20] to improve the rise time, settling time, overshoot, and peak time of a standard PSO controller. Robust adaptive control based on backstepping is used in [21] [22] to provide robust quadcopter altitude and attitude tracking after payload change. Immersion and invariance based adaptive backstepping control is used in [23] to remedy quadcopter attitude instabilities due to disturbance torques or parameter uncertainties. Adaptive fault tolerant control based on the adaptive minimum projection method is used to provide quadcopter stability in the event of actuator failure in [24]. Balanced control is used in [25] to improve quadcopter performance by switching between fuzzy adaptive PID and optimized PID midflight. L1 adaptive control is used in [26] to stabilize a fixed-wing pitch controller, and in [27] to counteract fixed-wing split drag rudder damage. Adaptive control is also used in [28] to resolve instabilities due to trailing vortices in fixed-wing formation flight. Online adaptive model parameter estimation is used in the velocity controller in [29] to mitigate error due to model parameter uncertainty. However, most of these control techniques focus on optimizing a part of the control system, assuming that the rest of the control system is sufficiently good.

Typical quadcopter autopilots are, however, based on cascaded controllers, which consist of an inner loop to stabilize the dynamics, and an outer loop to track position commands. Traditionally, all controllers in the autopilot are constructed using manually tuned PID control laws. In fact, widely used open-source autopilots such as PX4 and ArduPilot contain finely-tuned PID control laws for many commercially available multicopter configurations [30], [31]. These autopilots cannot guarantee stability and thus have a fixed operational envelope, which is usually unknown. Moreover, autopilots tuned for a specific geometry and inertia properties do not perform well in the case where these properties vary with time, such as, in the case of unknown suspended payload, hardware alteration, and dynamic environmental changes. Such variations invariably degrade the performance of the autopilot.
With these motivations in mind, this paper develops an adaptive autotuner for multicopters with unknown dynamics. In particular, all of the fixed-gain control laws of an autopilot are replaced by adaptive control laws that optimize the gain by flying a single learning trajectory. Specifically, the adaptive controllers are updated by the retrospective cost adaptive control (RCAC) algorithm [32]. The learning trajectory is designed such that all possible motions of the quadcopter are excited and thus nonzero control outputs are required from all control laws, which ensures that all of the controller gains are adaptively updated.

The contribution of the work presented in this paper is the development of an adaptive autotuner that does not require any prior knowledge of the system dynamics, and instead uses a simple learning trajectory to adapt the autopilot gains; and its numerical and experimental demonstration. The performance of the autotuner is demonstrated by flying a test trajectory and comparing its performance with a finely-tuned autopilot. The improvements due to the autotuning process are demonstrated through simulations and flight tests.

The paper is organized as follows. Section II briefly presents the control system architecture implemented in the PX4 autopilot. Section III explains the setup of the adaptive autotuner. Section IV describes the RCAC algorithm used to tune the control parameters. Section V presents simulation and flight test results showing the performance of the PX4 autopilot using the stock and autotuned control parameters. Finally, section VI concludes the paper with a summary and future research directions.

II. QUADCOPTER AUTOPILOT

This section reviews the quadcopter autopilot considered in this work. This autopilot is based on the control architecture implemented in the PX4 autopilot. The notation used in this paper is described in more detail in [33].

The control system consists of a mission planner and two nested loops as shown in Figure 1. The mission planner generates the position, velocity, and azimuth setpoints from the user-defined waypoints using the guidance law described below. The outer loop consists of the position controller, whose inputs are the position and velocity errors, defined as the difference between the setpoints and the measurements. The output of the position controller is the thrust vector setpoint. Note that the thrust vector is expressed in terms of the Earth-fixed frame. The inner loop consists of the attitude controller, whose inputs are the thrust vector setpoint, the azimuth and azimuth-rate setpoints, as well as the attitude and the angular rate measured in the body-fixed frame. The output of the attitude controller is the moment vector setpoint.

The control system implemented in the PX4 autopilot thus consists of 27 gains. In particular, the outer loop includes three gains in $G_r$ and nine gains in $G_v$; and the inner loop includes three gains in $G_q$ and 12 gains in $G_\omega$. In standard practice, these 27 gains are manually tuned and require considerable expertise.

The mission planner uses a guidance law described in Appendix A to generate the position and the azimuth setpoints. Using a user-specified maximum velocity $v_{\text{max}}$ and maximum acceleration $a_{\text{max}}$, the guidance law generates a trajectory that consists of a constant acceleration phase, a cruise phase, and a constant deceleration phase. A similar guidance law is used to generate azimuth setpoints, given a user-specified maximum angular velocity $\dot{\psi}_{\text{max}}$ and maximum angular acceleration $\ddot{\psi}_{\text{max}}$.

The position controller consists of two cascaded linear controllers as shown in Figure 2. The first controller $G_r$ consists of three proportional controllers. The second controller $G_v$ consists of three decoupled PID controllers and a velocity setpoint feedforward controller, and yields the thrust vector setpoint.

The force vector setpoint along with the azimuth setpoint are used to calculate the attitude setpoint, which is represented as a quaternion. The attitude controller consists of two cascaded controllers $G_q$ and $G_\omega$ as shown in Figure 3. The first controller $G_q$ is an almost globally stabilizing controller [34] that consists of three proportional gains. The second controller $G_\omega$ consists of three PID controllers and an angular rate setpoint feedforward controller, and yields the moment vector setpoint.

The outer loop consists of three decoupled PID controllers and a velocity setpoint feedforward controller, and yields the thrust vector setpoint. The inner loop consists of three proportional controllers and an angular rate setpoint feedforward controller, and yields the moment vector setpoint.
III. Adaptive Autotuner

The adaptive autotuner is constructed by replacing the fixed-gain controllers in the autopilot described in the previous section with adaptive controllers that are updated by retrospective cost optimization. In particular, each fixed-gain controller described in the previous section is replaced by an adaptive controller parameterized with the same structure. Specifically, denoting the \( i \)th component of the input and the output of \( G_r \) at step \( k \) by \( z_{r,i,k} \) and \( u_{r,i,k} \), the velocity setpoint

\[
u_{r,i,k} = G_r(k, \mathbf{q})g(z_{r,i,k}), \tag{1}\]

where \( G_r(k, \mathbf{q}) = \text{diag}(\theta_{r,1,k}, \theta_{r,2,k}, \theta_{r,3,k}) \), \( g(z) \) is a error-normalization function, and the gains \( \theta_{r,i,k} \) are updated by the RCAC algorithm described in the next section. The functions \( g(z) \) used in this work are given in Table II. Similarly,

\[
u_{v,i,k} = G_v(k, \mathbf{q})g(z_{v,i,k}), \tag{2}\]

where, for \( i \in \{1, 2, 3\} \), the entry \((i, i)\)

\[
G_{v,i,i}(k, \mathbf{q}) = \frac{q_{i} + \theta_{v,i+1}}{q_{i} - 1} + \frac{\theta_{v,i+3}(q_{i} - 1)}{q_{i}^2}. \tag{3}
\]

The controllers \( G_q(k, \mathbf{q}) \) and \( G_w(k, \mathbf{q}) \) are similarly parameterized by the gains \( \theta_q \in \mathbb{R}^3 \) and \( \theta_w \in \mathbb{R}^{12} \).

The adaptive gains \( \theta_r, \theta_q, \theta_v, \) and \( \theta_w \) are updated in a learning trajectory, which consists of flying through the waypoints described in Table I, parameterized by \( \hat{z}_{\text{hov}}, x_{\text{inc}}, y_{\text{inc}}, \) and \( z_{\text{inc}} \). Note that the waypoints are denoted by coordinates \((x, y, z, \psi)\), which correspond to the three components of the position vector and azimuth of the multicopter in the East(x)-North(y)-Up(z) (ENU) coordinate frame.

**TABLE I**

| Waypoint | Coordinate | Remark |
|----------|------------|--------|
| 1        | (0, 0, \( z_{\text{hov}} \), 0) | Take-off |
| 2        | (0, 0, \( z_{\text{hov}} + z_{\text{inc}} \), 0) | Move along +z direction |
| 3        | (0, 0, \( z_{\text{hov}} \), 0) | Move along –z direction |
| 4        | (0, \( y_{\text{inc}} - z_{\text{hov}} \), 0) | Move along +y direction |
| 5        | (0, \( y_{\text{inc}} + z_{\text{hov}} \), 0) | Move along –y direction |
| 6        | (0, 0, \( z_{\text{inc}} \), 0) | Move along +z direction |
| 7        | (\( x_{\text{inc}} \), 0, \( z_{\text{hov}} \)) | Move along +x direction |
| 8        | (\( -x_{\text{inc}} \), 0, \( z_{\text{hov}} \)) | Move along –x direction |
| 9        | (0, 0, \( z_{\text{hov}} \), 0) | Move along +z direction |
| 10       | (0, 0, \( z_{\text{hov}}, \pi/2 \)) | Turn counterclockwise |
| 11       | (0, 0, \( z_{\text{hov}}, \pi \)) | Turn counterclockwise |
| 12       | (0, 0, \( z_{\text{hov}} \), 0) | Turn clockwise |
| 13       | (0, 0, 0, 0) | Land |

The gains \( \theta_r, \theta_q, \theta_v, \) and \( \theta_w \) obtained at the end of the learning trajectory are the autotuned gains and the autopilot implemented with the autotuned gains is the autotuned autopilot.

IV. RCAC Algorithm

This section briefly reviews the retrospective cost adaptive control (RCAC) algorithm. RCAC is described in detail in [35] and its extension to digital PID control is given in [32]. Consider a SISO PID controller with a feedforward term

\[
u_k = K_p,k g(z_{k-1}) + K_i,k \gamma_k - 1 + K_d,k (g(z_{k-1}) - g(z_{k-2})) + K_f,k r_k, \tag{4}\]

where \( K_p,k, K_i,k, K_d,k, \) and \( K_f,k \) are time-varying gains to be optimized, \( z_k \) is an error variable, \( r_k \) is the feedforward signal, and, for all \( k \geq 0 \),

\[\gamma_k \triangleq \sum_{i=0}^{k} g(z_i). \tag{5}\]

Note that the integrator state is computed recursively using \( \gamma_k = \gamma_{k-1} + g(z_{k-1}) \). For all \( k \geq 0 \), the control law can be written as

\[
u_k = \phi_k \theta_k, \tag{6}\]

where the regressor \( \phi_k \) and the controller gains \( \theta_k \) are

\[
\phi_k \triangleq \begin{bmatrix} g(z_{k-1}) \\ \gamma_k - 1 \\ g(z_{k-1}) - g(z_{k-2}) \end{bmatrix}^T, \quad \theta_k \triangleq \begin{bmatrix} K_p,k \\ K_i,k \\ K_d,k \\ K_f,k \end{bmatrix}. \tag{7}\]

Note that the P, PI, or PID controllers can be parameterized by appropriately defining \( \phi_k \) and \( \theta_k \). Various MIMO controller parameterizations are shown in [36].

To determine the controller gains \( \theta_k \), let \( \theta \in \mathbb{R}^{l_\theta} \), and consider the retrospective performance variable defined by

\[
\hat{z}_k(\theta) \triangleq g(z_k) + \sigma(\phi_{k-1} \theta - u_{k-1}), \tag{8}\]

where \( \sigma \in \mathbb{R} \). The sign of \( \sigma \) is the sign of the leading numerator coefficient of the transfer function from \( u_k \) to \( \hat{z}_k \). Furthermore, define the retrospective cost function

\[
J_k(\theta) \triangleq \sum_{i=0}^{k} \hat{z}_i^T R \hat{z}_i(\theta) + (\phi_k \theta)^T R_k(\phi_k \theta) + (\theta - \theta_0)^T P_0^{-1}(\theta - \theta_0), \tag{9}\]

where \( \theta_0 \in \mathbb{R}^{l_\theta} \) is the initial vector of PID gains and \( P_0 \in \mathbb{R}^{l_\theta \times l_\theta} \) is positive definite.

**Proposition IV.1.** Consider (6)–(9), where \( \theta_0 \in \mathbb{R}^{l_\theta} \) and \( P_0 \in \mathbb{R}^{l_\theta \times l_\theta} \) is positive definite. Furthermore, for all \( k \geq 0 \), denote the minimizer of \( J_k \) given by (9) by

\[
\theta_{k+1} \triangleq \arg \min_{\theta} J_k(\theta). \tag{10}\]

Then, for all \( k \geq 0 \), \( \theta_{k+1} \) is given by

\[
\theta_{k+1} = \theta_k - \sigma P_{k+1} \phi_{k-1}^T R_k \hat{z}_k + \sigma(\phi_{k-1} \theta_k - u_{k-1}) \\
- P_{k+1} \phi_k^T R_k \phi_k \theta_k, \tag{11}\]

where

\[
P_{k+1} \triangleq P_k - \sigma \phi_{k-1}^T R_k \phi_{k-1} \tag{12}\]
where
\[ P_{k+1} = P_k - P_k \Phi_k^T \left( \tilde{R}^{-1} + \Phi_k P_k \Phi_k^T \right)^{-1} \Phi_k, \] (13)
and
\[ \Phi_k \triangleq \begin{bmatrix} \sigma \phi_{k-1} \\ \phi_k \end{bmatrix}, \quad \tilde{R} \triangleq \begin{bmatrix} R_z & 0 \\ 0 & R_u \end{bmatrix}. \] (14)

**Proof.** See [37]

Finally, the control is given by
\[ u_{k+1} = \phi_{k+1} \theta_{k+1}. \] (15)

## V. Experimental Results

This section describes the experimental results obtained in the simulation environment and in the physical flight tests. In both the simulation and the physical flight tests, the autotuner is used to tune the 27 controller gains described in Section II using the learning trajectory described in Section III. Note that in the autotuning mode, all of the gains in both loops are initialized at zero. The hyperparameters \( R_u, P_0 \), and the error-normalization function used in the RCAC algorithm are shown in Table II. Furthermore, \( \sigma = R_z = 1 \) in all four controllers.

### TABLE II

| Controller | \( P_0 \) | \( R_u \) | \( g(z) \) |
|------------|-----------|-----------|-----------|
| \( \mu \)  | 0.01      | 0.01      | \( z \)   |
| \( \mu_o \)| 0.1       | 0.01      | \( \text{erf} \left( \frac{\sqrt{2} z}{\sigma} \right) \) |
| \( \mu_d \)| 1         | 0.001     | \( z \)   |
| \( \mu_o \)| 0.0001   | 0.1       | \( \text{erf} \left( \frac{\sqrt{2} z}{\sigma} \right) \) |

### A. Simulation Flight Tests

First, the autotuner is tested in a simulation environment, where the quadcopter is simulated using jMAVSim. In the learning trajectory, \( z_{\text{inc}} = 5 \) m, and \( z_{\text{inc}} = y_{\text{inc}} = z_{\text{inc}} = 5 \) m. The guidance law generates the setpoints using \( v_{\text{max}} = 6 \) m/s, \( a_{\text{max}} = 1 \) m/s², \( \dot{\psi}_{\text{max}} = 2 \) rad/s, \( \psi_{\text{max}} = 0.5 \) rad/s².

Figure 4 shows the response of the quadcopter in the learning trajectory. Note that the response improves as RCAC re-optimizes the controllers continuously in real time. Figure 5 shows the gains adapted by the RCAC algorithm for all four controllers. The gains at the end of the learning trajectory are saved, are the autotuned gains and constitute the autotuned autopilot.

Next, the quadcopter is commanded to follow a test trajectory, whose waypoints are generated using a second-order Hilbert curve, first with the default autopilot, and next with the autotuned autopilot. The default autopilot gains and the actuator constraints in PX4 are specified in the \texttt{mc_pos_control_params.c}, \texttt{mc_att_control_params.c}, and \texttt{mc_rate_control_params.c} files. Note that the results in this paper are based on PX4 version V1.11.3. Figure 6 shows the response of the quadcopter obtained with the autotuned autopilot and the default autopilot. Note that the autotuned autopilot outperforms the default autopilot in terms of position tracking error.

In order to quantify and compare the performance of the autotuned autopilot with the default autopilot, a position-tracking cost variable \( J \) defined by
\[ J \triangleq \frac{1}{T} \sum_{i=0}^{N} z_i^T z_i, \] (16)
where \( z_i \in \mathbb{R}^3 \) is the position error and \( T \) is the total flight time in seconds, is computed for each test. Table III shows the cost default autopilot cost \( J_D \) and the autotuned autopilot cost \( J_A \). Note that the autotuned autopilot is relatively 38% better than the default autopilot in simulation.

Next, the sensitivity of the autotuning process to changes

[1] https://github.com/JAParedes/PX4-Autopilot/tree/RCAC_MC_AutoTuner
in the physical parameters of the quadcopter is investigated. To do so, the mass of the quadcopter is scaled in the dynamic model simulated by jMAVSim. Specifically, the mass of the quadcopter is multiplied by $\alpha \in \{0.5, 0.75, 1, 1.25, 1.5\}$. For each value of $\alpha$, the autopilot gains are tuned using the autotuner. Note that the RCAC hyperparameters are not changed. The performance of the autotuned and the default autopilots is compared for each value of $\alpha$ by flying the test trajectory defined by the second-order Hilbert curve. Figure 7 shows the cost $J$ computed for the position controller in each flight flown with the autotuned and the default autopilot. Note that the autotuned autopilot is re-tuned for each value of $\alpha$, whereas default autopilot is fixed for all values of $\alpha$.

### B. Physical Flight Tests

Next, the autotuner is tested in physical flight tests conducted in the M-Air facility at the University of Michigan, Ann Arbor with the Holybro X500 quadcopter frame. The M-Air is equipped with a motion capture system that allows high-precision position and attitude measurements.

In the learning trajectory, $z_{\text{hov}} = 2$ m, $x_{\text{inc}} = y_{\text{inc}} = 4$ m, and $z_{\text{inc}} = 1$ m. The guidance law generates the setpoints using $v_{\max} = 3$ m/s, $a_{\max} = 1.2$ m/s$^2$, $\dot{\psi}_{\max} = 2$ rad/s, $\ddot{\psi}_{\max} = 0.5$ rad/s$^2$. Figure 8 shows the response of the quadcopter in the learning trajectory. Figure 9 shows the gains adapted by the RCAC algorithm for all four controllers.

Next, the quadcopter is commanded to follow a test trajectory with the autotuned autopilot and the default autopilot in the M-Air facility. The waypoints for the test trajectory are generated using a second-order Hilbert curve. Figure 10 shows the response of the quadcopter obtained with the autotuned autopilot and the default autopilot during the physical flight tests. Table III shows the position-tracking cost $J$ computed for the test trajectory flown during the physical flight tests. Note that the autotuned autopilot is relatively 32% better than the default autopilot.

### VI. Conclusions and Future Work

This paper presented an adaptive autotuner that can automatically tune a multicopter autopilot without using any modeling information about the multicopter. The adaptive
autotuner, implemented in the PX4 flight stack, consists of adaptive PID controllers that are updated by the retrospective cost adaptive control algorithm by flying a single learning trajectory.

The adaptive autotuner was validated in a simulation environment, and its performance was compared against the default autopilot by scaling the mass of the quadcopter. The adaptive autotuner outperformed the default autotuner in all cases without any change in the hyperparameters of the adaptive algorithm. Next, using the same hyperparameters used for the simulation, the adaptive autotuner was used to tune the autopilot for the X500 Holybro quadcopter in the M-Air facility, where the adaptive autotuner outperformed the default autotuner in the test trajectory.

Future work will focus on using the adaptive autopilot to improve the performance of the default autopilot flying a multicopter with an unknown suspended payload and under various actuator failure conditions.

APPENDIX A

Let $r_c \in \mathbb{R}^3$ denote the current position, $r_d \in \mathbb{R}^3$ denote the desired position, and $d$ denote the distance between the current and the desired position, that is, $d \triangleq ||r_d - r_c||_2$. Let $a_{\text{max}} > 0$ and $v_{\text{max}} > 0$. The desired trajectory consists of an acceleration phase, a cruise phase, and a deceleration phase along the straight line between $r_c$ and $r_d$.

Let the setpoints $r(t) \in \mathbb{R}^3$ be given by the guidance law

$$r(t) = r_c + (r_d - r_c) \frac{q(t)}{d},$$

where $q(t)$ is calculated as shown below. Using the fact that $r(0) = r_c$ and $r(T_i) = r_d$, where $T_i$ is the total flight time, it follows that $q(0) = 0$ and $q(T_i) = d$. Note that $q(t)$ is the distance between $r(t)$ and $r_c$.

Let $T_1$ denote the time to reach the speed $v_{\text{max}}$ at maximum acceleration $a_{\text{max}}$. Thus, $T_1 = \frac{v_{\text{max}}}{a_{\text{max}}}$ and $q(T_1) = \frac{1}{2}a_{\text{max}}T_1^2 = \frac{1}{2}a_{\text{max}}^2$.

First, consider the case where $d \geq 2q(T_1)$. In this case, the quadcopter cruises at $v(t) = v_{\text{max}}$ for $t \in [T_1, T_2]$, where $T = T_2$ denotes the time at which deceleration phase starts, and is calculated as shown below. Note that

$$q(T_2) = \frac{1}{2}a_{\text{max}}T_2^2 + v_{\text{max}}(T_2 - T_1).$$

Since the quadcopter takes $T_2$ seconds to decelerate from $v_{\text{max}}$ to 0 speed at constant deceleration $a_{\text{max}}$, the total flight time is $T_i = T_1 + T_2$ and

$$q(T_3) = \frac{v_{\text{max}}^2}{a_{\text{max}}} + v_{\text{max}}(T_2 - T_1) = d. \tag{19}$$

It follows from (19) that $T_2 = \frac{d}{v_{\text{max}}}$ and thus, the total flight time $T_i = \frac{v_{\text{max}}}{a_{\text{max}}} + \frac{d}{v_{\text{max}}}$. The distance $q(t)$ is thus given by

$$q(t) = \begin{cases} \frac{1}{2}a_{\text{max}}t^2, & t \in [0, T_1), \\ \frac{1}{2}a_{\text{max}}T_1^2 + v_{\text{max}}(t - T_1), & t \in [T_1, T_2), \\ \frac{1}{2}a_{\text{max}}T_1^2 + v_{\text{max}}(T_2 - T_1) + v_{\text{max}}(t - T_2) - \frac{1}{2}a_{\text{max}}(t - T_2)^2, & t \in [T_2, T_i]. \end{cases} \tag{20}$$

Next, consider the case where $d < 2q(T_1)$. In this case, the quadcopter velocity does not reach $v_{\text{max}}$, that is, there is no cruise phase. Let $T_1$ denote the time instant at which maximum velocity $v_{\text{max}}$ is reached. Thus, $T_1 = \frac{v_{\text{max}}}{a_{\text{max}}}$ and

$$q(T_1) = \frac{1}{2}a_{\text{max}}^2. \tag{21}$$

Assuming that the quadcopter decelerates at the same rate to rest, it follows that $q(T_1) = d/2$, and thus $v_{\text{max}} = \sqrt{a_{\text{max}}}$. $T_1 = \sqrt{\frac{d}{a_{\text{max}}}}$ and $T_i = 2\sqrt{\frac{d}{a_{\text{max}}}}$. The distance $q(t)$ is thus given by

$$q(t) = \begin{cases} \frac{1}{2}a_{\text{max}}t^2, & t \in [0, T_1), \\ \frac{1}{2}a_{\text{max}}T_1^2 + v_{\text{max}}(t - T_1), & t \in [T_1, 2T_1). \end{cases}$$
