Teleop Command/Imitation Learning/Regression (in progress)

Rahul Peddi and Nicola Bezzo

***Abstract*—**

I. Introduction

Unmanned Aerial Vehicles (UAVs) have become more widespread for both civilian and military applications in the recent years. There are several applications for which UAVs are uniquely suited over other robotic systems, such as surveillance, delivery services, and search and rescue. Many of these applications involve the UAV autonomously following trajectories to different goal or task locations. For example, a package delivery UAV may have to autonomously reach multiple locations at certain times to deliver and receive new packages.

Generating this autonomous flight on UAVs, however, can be difficult. Every platform has a different dynamical model, and knowledge of that model is required to generate autonomous flight. Traditionally, autonomous control of UAVs consists of four control inputs: thrust, roll, pitch, and yaw. For these control inputs to actually create motion, knowledge about the physical system (e.g., mass and moments of inertia) is required [CITE]. This information, however, can vary quite a lot for different systems [show image of two very different uavs here], and the specifics and parameters of the dynamic models are not always available.

This need for platform-specific parameters, however, can be avoided if we could learn to send autonomous controls to the vehicle in another way. The parameters are generally only necessary when generating autonomous flight, but when there is a human pilot, the controls are sent to the system in a different manner, usually via teleoperation commands. In this work, we are interested in transferring this knowledge from a human-piloted demonstrations for fast learning and generation of autonomous flight, all while being unaware of platform-specific parameters. It is, however, trivial to replicate the actions of the human pilot to generate the same action autonomously. As a result, we are interested in taking a library of a several demonstrations to enable the vehicle to follow any generated trajectory.

To this end, we propose a framework that aim the solve the following challenges:

- how to enable autonomous flight in aerial vehicles with minimal knowledge of vehicle dynamics
- how to expand the knowledge obtained from several demonstrations to follow any trajectory

Rahul Peddi and Nicola Bezzo are with the Department of Systems and Information Engineering and the Charles L. Brown Department of Electrical and Computer Engineering, University of Virginia, Charlottesville, VA 22904, USA. Email: {rp3cy, nb6be}@virginia.edu

♠[1] In order to address these challenges, our approach leverages the generalized dynamics of UAVs along with regression analysis on demonstrations and elements of control theory to generate autonomous flight and ensure that the vehicle can track any trajectory.

II. Related Work

III. Problem Formulation

In this work, we are interested in using human-piloted demonstrations to develop a framework that enables autonomous UAV navigation over any trajectory with minimal knowledge of platform-specific dynamics.

Formally, the problem we investigate in this work can be stated as:

Problem 1: ***Demonstration-based autonomous control generation***: A UAV has the objective to visit a set of n goals, $\mathbf{g} \in \mathbb{R}^n$, at a set of n target times, $\mathbf{t} \in \mathbb{R}^n$. Given m human-piloted flight demonstrations, the aim is to find a policy that enables autonomous flight to generate and track a trajectory $\mathbf{p_r}(t),\ t \in [0, T]$ that visits the target goals $\mathbf{g}_i$ at the respective target times, $\mathbf{t}_i$, such that the following liveness requirements are met:

1) Position: The UAV should always stay within a certain distance of the generated trajectory:

$$||\mathbf{p}(t) - \mathbf{p_r}(t)|| \leq \delta_d,\ \forall t \in [0, T] \tag{1}$$

where $\mathbf{p_r}(t)$ is the reference position at time t, where T represents an overall time horizon for the trajectory, and δ_d is a threshold for allowable deviation.

2) Time: The UAV should always reach the target locations within a certain threshold of the target time for that goal:

$$\begin{aligned} \tau = t \in [0, T] | \mathbf{p}(t) - \mathbf{g}_i \leq \delta_d \\ |\tau - \mathbf{t}_i| \leq \delta_t,\ i = 1, \ldots, n \end{aligned} \tag{2}$$

where τ is a time at which the liveness condition is met for a specific goal and δ_t is the allowable deviation in time from the target.

IV. System Dynamics

The type of UAV in question is modeled using a 12th order state vector:

$$\mathbf{q} = \begin{bmatrix} \mathbf{p}_q^\intercal & \phi & \theta & \psi & v_x & v_y & v_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^\intercal$$

where $\boldsymbol{p}_q = [x\ y\ z]^\intercal$ is the world frame position, v_x, v_y and v_z are the world frame velocities, ϕ, θ and ψ are the roll,

[1]NB: we need to stress how different this work is in comparison with the rest of the literature on learning from demosntration

pitch and yaw Euler angles and ω_x, ω_y and ω_z are the body frame angular velocities.

The dynamics of the vehicle are then described as follows:

$$\dot{\boldsymbol{p}}_q^{\mathsf{T}} = \begin{bmatrix} v_x & v_y & v_z \end{bmatrix}$$

$$\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{1}{m} \begin{bmatrix} \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi \\ \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi \\ \cos\theta\cos\phi \end{bmatrix} u_1$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

$$\begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \begin{bmatrix} \frac{I_{yy}-I_{zz}}{I_{xx}}\omega_y\omega_z \\ \frac{I_{zz}-I_{xx}}{I_{yy}}\omega_x\omega_z \\ \frac{I_{xx}-I_{yy}}{I_{zz}}\omega_x\omega_y \end{bmatrix} + \begin{bmatrix} \frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix}$$

[CITE]

♠[2] ♠[3]

In the human piloted demonstrations, the user is controlling u_2, u_3, and u_4, which correspond to the desired roll, pitch, and yaw angles. In the system dynamics, u_1 is the thrust provided to the system. This is not directly sent by the user, but the thrust input is adjusted when the pilot is modifying roll, pitch, and yaw. The model

In this work, we are interested in learning from commands sent from the human pilot to the UAV, so we assume in our training that we are able to collect all information communicated from the pilot to the UAV. During run-time, we are also able to get the world frame position $\mathbf{p}(t)$ of the UAV using a motion capture system.

Through observation of a single demonstrated trial, we are able to identify important commands sent to the system. An example of this is shown below:

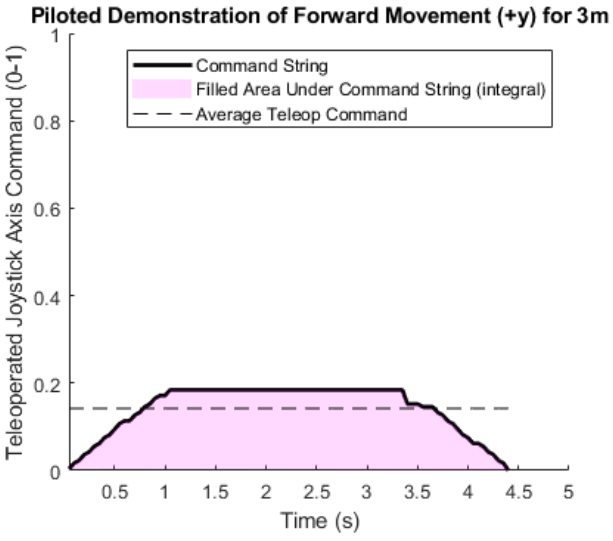


Fig. 1. A sample of a single demonstrated flight

[2]NB: here we need to stress again that these parameters are hard to find online

[3]NB: here we need a model and discussion about the controller

In Fig. 1, we show a sample of a single flight demonstration. The pilot flies the UAV forward over a distance of approximately 3 meters. The command string, in this case, corresponds to pitch, where a command of 0 would imply no pitch adjustment, and 1 corresponds to the UAV's maximum allowable pitch. The minimum allowable pitch command is -1, which would send the UAV in in the negative y-direction. Along with the string of commands sent, the shaded portion of Fig.1, represents the total area under the string of commands, or the integral. Because the pitch is proportional to the linear velocity in the y-direction, we know that the command string is proportional to the velocity, and its integral, therefore, is proportional to the distance traveled.

Lastly, we are interested in the average command sent to the system, as indicated by the dotted line. The integral itself, while conveying information about the distance traveled, lacks precision in that the same integral can be achieved with commands strings of different lengths and heights. Making use of the average command, we are able to identify the correct height of the teleoperation commands and improve precision in the process of generating autonomous commands, which must be fixed to a certain length (time) and travel a certain distance.

V. Methodology

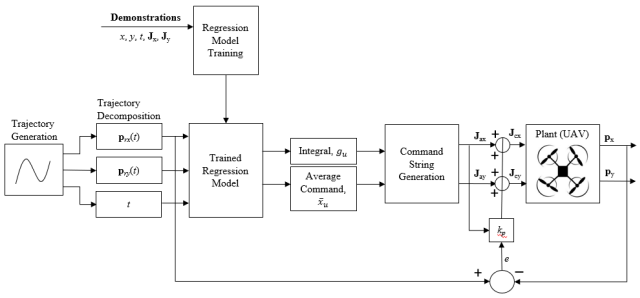


Fig. 2. Block Diagram of Proposed Approach

♠[4]

The proposed approach is outlined in Fig.2. The approach begins with multiple demonstrations that are used to build a regression model that enables identification of an integral, g_u, and average teleoperation command, $\bar{x}_u$, given a user-set target distance, d_u, and time, t_u. With the integral and average velocity,a new command string consisting of commands for both lateral and longitudinal locations is created, . At this stage, we have obtained the actions, $\mathbf{J}_x(t)$ and $\mathbf{J}_y(t)$, that track trajectory $\mathbf{p}_r(t)$. Because these actions are created in open-loop, some error is possible. This error is reduced by closing the loop via a robust control approach to correct and adjust future inputs $\mathbf{J}_x(t+1)$ and $\mathbf{J}_y(t+1)$ to obtain closed loop input commands $\mathbf{J}_{cx}(t+1)$ and $\mathbf{J}_{cy}(t+1)$.

[4]NB: let's use power point fro the diagrams..it's blurry

A. Trajectory Generation and Decomposition

In this section, we show how we generate trajectory based on given waypoints. We want our vehicle to track a smooth trajectory as much as possible with linear motion, so this part of the approach is twofold: generating a smooth trajectory, given waypoints, and decomposing this trajectory into linear parts, as to be congruous with our training set, which consists of training only on linear motion, which will be discussed further in Section V-B.

From the system dynamics, we know that input commands are responsible for specific motions of the system. Roll enables lateral motion while enables longitudinal motion. Yaw enables rotations around the center of mass and thrust is mostly responsible for changes in height along the global z coordinate. For ease of discussion in this work we do not consider yaw, as we are primarily interested in planar motion, which also implies that the height is kept constant. As a result, any planar motion assuming constant z can be represented as a combination of roll and pitch commands, or commands that send the vehicle in along the global x and y coordinates.

For smoothing purposes, we use the minimum jerk trajectory. Jerk is defined as the time derivative of acceleration and is often associated rapidly changing actuator forces. [CITE] The general equation for a minimum jerk trajectory is as follows:

$$\mathbf{p}^*(t) = \arg\min_{p(t)} \int_0^T \mathcal{L}(\dddot{p}, \ddot{p}, \dot{p}, p, t)dt \tag{3}$$

Solving the equation 3, we obtain the form:

$$\mathbf{p}^*(t) = c_5t^5 + c_4t^4 + c_3t^3 + c_2t^2 + c_1t + c_0 \tag{4}$$

where $c_0, \ldots, c_5$ are constants determined using the linear system : *in progress*

Having obtained the smooth trajectory, we are want to find the closest decomposition of said trajectory into lines. This decomposition is done by identifying the critical and inflection points of the trajectory, by taking the first and second derivatives of the trajectory $\mathbf{p}^*(t)$, and setting them equal to zero: ♠[5]

$$\begin{aligned} \frac{d}{dt}\mathbf{p}^*(t) &= 0 \\ \frac{d^2}{d^2t}\mathbf{p}^*(t) &= 0 \end{aligned} \tag{5}$$

To generate the final trajectory, we set these critical points as new waypoints, and use a straight line to create the path between the waypoints. *in progress*

B. Regression Based Training and Evaluation

In order to build the appropriate policy for autonomous command generation, we perform offline training on data collected over m human-piloted trials. Because the goal indicated in our problem statement is to be able to track a certain trajectory, the inputs of the training phase are $\{d_i, t_i, \mathbf{J_i}\}$, where $i = 1, \ldots, m$ reflects the number of demonstrations, d_i is the distance travelled in each trial, t_i is the length of the trial, and $\mathbf{J}_i$ is the sequence of user teleoperation commands. The outputs of our training phase are

- The integral of the string of teleoperation commands, $g_i = \int_0^{t_i} \mathbf{J}_i$
- The steady-state, or average, teleoperation command, $\bar{j}_i \in \mathbf{J}_i$
- The position of the system at all times, $\mathbf{p}(t)$

The steady-state command, in this case, provides information about the pilot's average in-flight velocity, and is obtained by taking the mean of all entries in $\mathbf{x_i}$. The integral, meanwhile, describes the area under the string of commands, as discussed in the previous section.

Our training data in this work consists of multiple demonstrations of a human-pilot flying forward at varying speeds for varying distances. In Fig.3, we show the raw data received from the demonstrations. In this case, we have training samples where the pilot flies forward and then stops, meaning the initial and final velocities are both zero, which is why there are an extended periods where there is no forward motion at the ends of the trajectories.

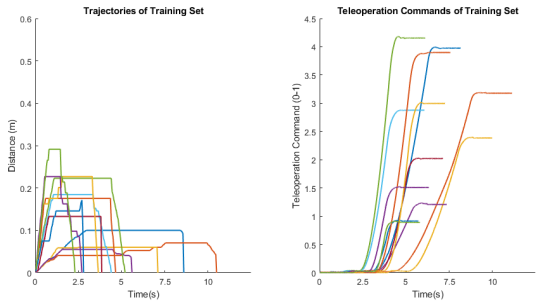


Fig. 3. Training Set, Teleoperation Commands pictured left, Trajectories pictured right

This data, however, would only be effective if trajectories involved stopping at each waypoint. In order to further expand the information obtained, considerations were made on the training data to include three different types of segments of commands:

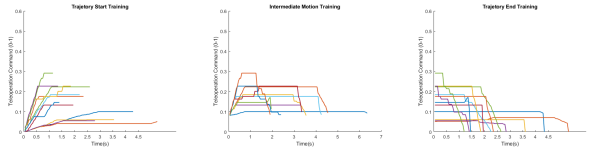
- Trajectory Start
- Intermediate Motion
- Trajectory End

The teleoperation commands for each of these segments are pictured in Fig4.

Having trained on not only full strings of commands, but also on segments enables more specific command generation based on where the system is in relation to the provided trajectory. For example, if the command for the end of the trajectory needs to be generated, then the training from the trajectories in 4(c) is used.

The offline training data is applied to a thin-plate spline surface fit to describe the relationship between training inputs

[5]NB: why? We can discretize the trajectory and get the x and y components



(a) Trajectory Start (b) Int. Motion (c) Trajectory End

Fig. 4. Training Data with Trajectory Considerations

and integral and average velocity. The general form of a thin-plate spline equation is

$$f(x,y) = a_1 + a_x x + a_y y + \sum_{i=1}^{m} w_i U(||(x_i,y_i) - (x,y)||) \quad (6)$$

where $a_1, a_x,$ and a_y are scalar coefficients, w_i is a coefficient that corresponds to each specific trial, subject to the following condition:

$$\sum_{i=1}^{m} w_i = \sum_{i=1}^{m} w_i x_i = \sum_{i=1}^{m} w_i y_i = 0 \quad (7)$$

and the function U is of the form

$$U(r) = r^2 \log r \quad (8)$$

Given the corresponding z_i for each (x_i, y_i) pair, we are able to solve the following linear system to obtain the coefficients $w_i, \ldots, w_m$ and a_1, a_x, a_y,

$$\begin{bmatrix} K & P \\ P^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \mathbf{a} \end{bmatrix} = \begin{bmatrix} \mathbf{z} \\ \mathbf{o} \end{bmatrix} \quad (9)$$

where $K_{ij} = U(||(x_i, y_i) - (x_j, y_j)||)$, $P_i* = (1, x_i, y_i)$, $\mathbf{0} \in \mathbb{R}^{3\times3}$ is a matrix of zeros, $\mathbf{o} \in \mathbb{R}^{3\times1}$ is a column vector of zeros, $\mathbf{w} \in \mathbb{R}^{m\times1}$ and $\mathbf{z} \in \mathbb{R}^{m\times1}$ are formed from w_i and z_i, respectively, and $\mathbf{a}$ is the column vector with elements a_1, a_x, a_y.

Given the general framework for performing the thin-plate spline, we develop two separate relationships for our specific application. In our work, the first regression is applied to d (distance) and t (time) as inputs to obtain the integral of the commands $g_i = f(d_i, t_i)$ while the second one has the same inputs to obtain the average teleoperation command $\bar{x}_i = h(d_i, t_i)$. With the functions we have obtained, we are able to find an estimated integral and steady-state command, g_u and $\bar{x}_u$, for any given desired distance and time, d_u and t_u, respectively,

$$g_u = f(d_u, t_u) \quad (10)$$

$$\bar{x}_u = h(d_u, t_u) \quad (11)$$

where d_u and t_u are the user-set desired distance and time, respectively. In Figs.5 and 6, we show the surfaces that represent the two functions $g_i = f(d_i, t_i)$ and $\bar{x}_i = h(d_i, t_i)$, respectively. The points in these figures represent actual training data points, all of which lay on the surface created with the thin-plate spline.

While a thin-plate spline is continuous, and a result can be obtained with any combination of distance and time as an

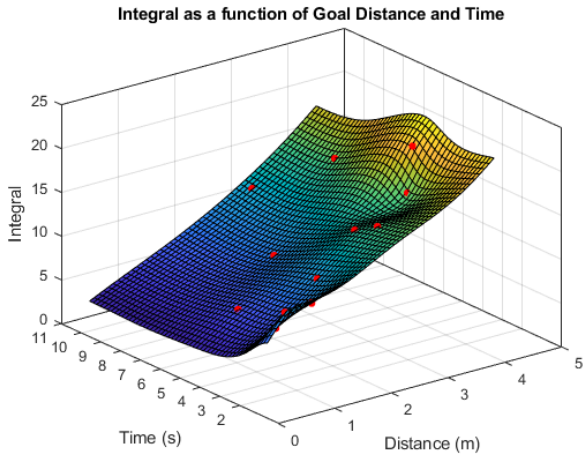


Fig. 5. Integral vs Distance and Time

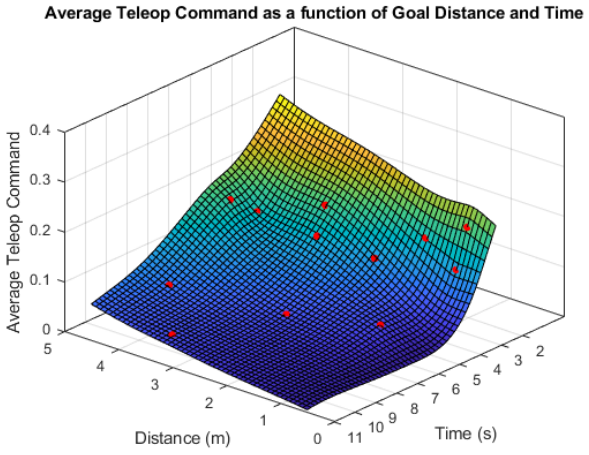


Fig. 6. Average Teleop Command vs Distance and Time

input pair, the accuracy of the results of any pair (d_u, t_u) can suffer as the distance between evaluation points and training points increases [CITE]. In order to quantify this, we leverage the standard error of the estimate, which is a statistic used to measure the accuracy of predictions given a certain type of regression with known values:

$$\sigma_{est} \approx \frac{s}{\sqrt{m}} \quad (12)$$

where s is the sample standard deviation of all of the points in the training set and m is the number of training samples. The standard error, σ_{est}, is then used to determine the t-statistic for different test values. The t-statistic is defined as the ratio of departure of a test point from a known point to its standard error and is defined as

$$t_{\hat{\beta}} = \frac{|\hat{\beta} - \bar{\beta}|}{\sigma_{est}} \quad (13)$$

where $\hat{\beta}$ is the test point and $\bar{\beta}$ is the nearest known point. In equation (13), if $t_{\hat{\beta}} \geq 1$, that means that the test point

$\hat{\beta}$ propagates an error higher than the standard error. As a result, it is desirable to have $t_{\hat{\beta}} < 1$. Using a certain set-point for the t-statistic, the maximum allowable departure from known points is obtained:

$$\sigma_{est} t_{\hat{\beta}} = |\hat{\beta} - \bar{\beta}| \tag{14}$$

The appropriate departure is used to set the bounds for each training data point, for example:

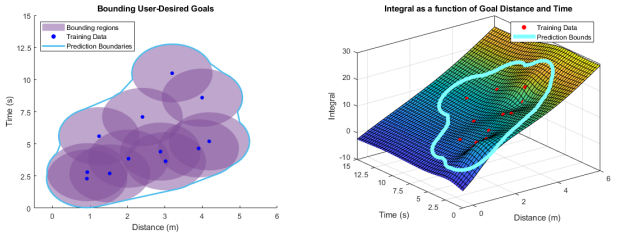
$$\begin{aligned} \hat{\beta}_{\min} &= \bar{\beta} - \sigma_{est} t_{\hat{\beta}} \\ \hat{\beta}_{\max} &= \bar{\beta} + \sigma_{est} t_{\hat{\beta}} \end{aligned} \tag{15}$$

where $\beta_{\min}$ and $\beta_{\max}$ are the lower and upper bounds for known parameter $\bar{\beta}$.

In our case, there are two parameters we are primarily concerned about for prediction; distance and time. As a result, we perform this calculation twice over, treating each of the two parameters as statistically independent, to obtain a generic interval for each point in the training set, denoted $[d_{i\,\min}, d_{i\,\max}]$ and $[t_{i\,\min}, t_{i\,\max}]$, where $i = 1, \ldots, m$. Because we are ultimately interested in using the distance and time values together as input pairs, we then obtain a maximum Euclidean distance from each of the training data points using:

$$\Delta_i = \sqrt{(\sigma_{d_{est}})^2 + (\sigma_{t_{est}})^2} \tag{16}$$

Using the maximum distance for each point Δ_i, we are able the generate intervals around each point, where the data is within the standard error of the estimate. Fig.7(a), contains a pictorial representation of the standard error intervals around each point. In Fig.7(b), the surface shown in Fig.5 is modified with a conforming boundary [CITE] created by the intervals.



(a) Prediction Bounds (b) Prediction Bounds on Surface

Fig. 7. Prediction Intervals and Bounds

C. Autonomous Behaviour Generation

In order for the system to autonomously reach a user-set goal, g_u and $\bar{j}_u$ are obtained using equations (10) and (11), and the offline training samples are leveraged to generate a new string of commands. A string of commands is generated for each leg of a trajectory, as described in Section V-A.

From the training set of m samples, we select the trial that has the closest integral, g_i, to the estimated integral g_u. This is done by forming an error vector, $\mathbf{e} \in \mathbb{R}^m$, where each element is defined by

$$e_i = |g_i - g_u|, \; i = \{1, \ldots, m\} \tag{17}$$

The lowest error is then found and is paired with the appropriate pre-trained sample, $\mathbf{J}^*$,

$$\mathbf{J}^* = \mathbf{J}_i \in \mathbf{J} | e_i = \min_e(\mathbf{e}) \tag{18}$$

This optimal pre-trained sample is then adjusted to reflect the user-set time, t_u. This is done by performing vector interpolation to re-size $\mathbf{J}^*$, such that important features in the optimal command vector are not lost. These methods are often used for re-sizing complex images, and have shown effectiveness in minimizing feature loss [CITE]. Bicubic interpolation is our chosen method for re-sizing, as it performs better than nearest-neighbor and bilinear interpolation methods, while only marginally increasing computational complexity [CITE]. The general form of a bicubic interpolation equation is

$$p(x) = \sum_{i=0}^{3} a_i x^i \tag{19}$$

where x is an entry in vector $\mathbf{J}^*$ and a represents the coefficients of the function at each point. Bicubic interpolation takes the weighted sum of the four nearest neighbors of each entry in the command vector in order to identify a function for the intermediate points between each value in $\mathbf{J}^*$. After resizing, we obtain the time adjusted input vector $\mathbf{J}'$.

The next step is to adjust the input vector such that the system reaches the user-set goal d_u. This is done by leveraging the average velocity information, that is, $\bar{j}_u$. Because distance is a function of average velocity and time, the scale time-adjusted vector $\mathbf{J}'$ is scaled such that its mean is equivalent to $\bar{j}_u$. We then obtain

$$\mathbf{J}_a = \mathbf{J}' \left(\frac{\bar{j}_u}{\bar{j}'} \right) \tag{20}$$

In Fig.8, we show visually the steps of command generation. In Fig.8(a), we start with the original pre-trained command string that was shown above in Fig.1. This flight travelled approximately 4.4 seconds for 3 meters. For testing purposes, our use input values are $d_u = 3.5$m and $t_u = 3.75$s. As indicated, a time adjustment is made first; this is shown in Fig.8(b). It is important to note that the average command, indicated by the dashed line, is still the same, which is a verification of feature retention using bicubic interpolation from (19). If the vector were just stretched without using an interpolation method, we would expect slight variation in the mean, which could damage the integrity of the final step, shown in Fig.8(c), which is obtained using (20). At this point we have obtained the autonomous command string $\mathbf{J}_a$, which is then sent to the UAV to reach the goals set by the user.

(a) Original String (b) Time Adjustment (c) Final Commands

Fig. 8. Command Generation Process

D. Online Adaptation of Generated Commands

The commands generated in the previous section are generated and sent to the UAV in open-loop. In order to close the loop, we propose a method to control for any possible error. While executing generated command string, $\mathbf{x_a}(t)$, we constantly monitor for error between the position of the vehicle, $\mathbf{p}(t)$ and the reference position as per the generated trajectory, $\mathbf{p_r}(t)$:

$$\xi(t) = \mathbf{p}(t) - \mathbf{p_r}(t) \tag{21}$$

The error $\xi(t)$ is attributed to the most recent command $\mathbf{J}_a(t)$, and a proportional controller is used to adjust the following command $\mathbf{J}_a(t+1)$ to obtain the closed-loop command $\mathbf{J}_c(t+1)$ as follows

$$\mathbf{J}_c(t+1) = \mathbf{J}_a(t+1) + k_p(t)\xi(t) \tag{22}$$

The proportional gain, $k_p(t)$, is automatically tuned based loosely off the Ziegler-Nichols method [CITE] where the proportional gain is increased until there is stable and consistent oscillation in the output, which in our case, is the position error, $\xi(t)$. In order to set the interval of gain increase each iteration, we leverage the ratio of reference position to actual position:

$$k_p(t+1) = k_p(t) + \left(\frac{\mathbf{p_r}(t)}{\mathbf{p}(t)}\right)\left(\frac{1}{t_u}\right) \tag{23}$$

where t_u refers to the user-set time, and adjusts the update of k_p such that the rate of increase is controlled for iteration time. If the rate of increase is not controlled in this manner, it would suggest that the controller is controlling for error over the entire remaining string of commands, rather than the first subsequent command. Each subsequent command is adjusted because our positional liveness constraint is defined as maintaining a certain distance between actual and reference positions at all times, from (1). If the constraint was for the UAV to reach the goal at a certain time immaterial of the path/trajectory, adjusting the entire remaining portion of the command string would be ideal.

VI. Experiments

VII. Conclusions

VIII. Acknowledgement

References

[1] G. O. Young,