Teleop Command/Imitation Learning/Regression (in progress)

Rahul Peddi and Nicola Bezzo

***Abstract*—**

I. Introduction

♠[1]Unmanned Aerial Vehicles have become more widespread for both civilian and military applications in the recent years. T

II. Related Work

III. Problem Formulation

In this work, we are interested in finding a policy that enables autonomous UAV navigation over a user-defined trajectory.

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.

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]NB: Motivation here is that it is hard to generate autonomous flight because every platform is different and you need to show what's the typical iteration to generate autonomous flght with quadrotor based on model.

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, 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:

$$\begin{aligned} \dot{\boldsymbol{p}}_q^\intercal &= \begin{bmatrix} v_x & v_y & v_z \end{bmatrix} \\ \begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{1}{m} \begin{bmatrix} \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi \\ \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi \\ \cos\theta\cos\phi \end{bmatrix} u_1 \\ \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} &= \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \\ \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} &= \begin{bmatrix} \frac{I_{yy} - I_{zz}}{I_{xx}}\omega_y\omega_z \\ \frac{I_{zz} - I_{xx}}{I_{yy}}\omega_x\omega_z \\ \frac{I_{xx} - I_{yy}}{I_{zz}}\omega_x\omega_y \end{bmatrix} + \begin{bmatrix} \frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} \end{aligned}$$

[CITE]

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 this work, we are interested in learning from commands send by the human pilot to the UAV. We assume that we are able to collect all information sent from the pilot to the UAV during demonstrations. 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:

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 theorize that the command string is proportional to the velocity, and its integral, therefore, is proportional to the

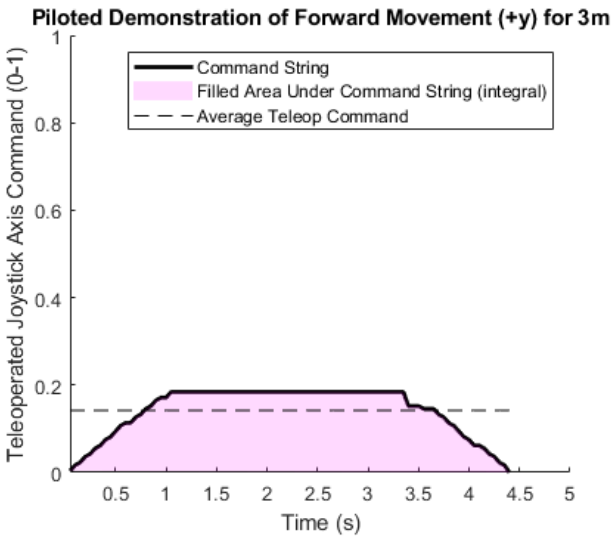


Fig. 1. A sample of a single demonstrated flight

distance traveled ♠[2]. 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 improve precision in the process of generating autonomous commands, that must be fixed to a certain length (time) and travel a certain distance.♠[3]

V. Methodology

In our approach, we leverage the data from multiple demonstrations to build a regression model that enables identification of an appropriate integral ♠[4], g_u, and average teleoperation command, $\bar{x}_u$ given a user-set distance, d_u, and time, t_u. With the appropriate integral and average velocity, the demonstrated command string with the closest integral, g_i, to the fitted integral, g_u, is resized to match the user-set time, t_u and fitted average velocity, $\bar{x}_u$, to obtain the autonomous command string $\mathbf{x}_a$. This command is sent to the physical UAV for implementation, where the on-board computer collects data regarding the error between actual position, $\mathbf{p}(t)$, and the expected position, $\mathbf{p}_r(t)$ is determined based on the trajectory. This error is reduced by ♠[5] using a proportional controller on the input $\mathbf{x}_a$ to obtain the adjusted input commands, $\mathbf{x}_c$. ♠[6]

Block Diagram

[2]NB: use the linearized model from AMR and show that a variation in u2 and u3 generate a moment which in turns create a motion.

[3]NB: we need to think how to pose the command better. A reviewer here can still ask why and what so special about the command.

[4]NB: what's this appropriate integral? Connect with the previous section better

[5]NB: leveraging a robust control approach on the input/output to correct and adjust future inputs.

[6]NB: in this section or before we need to stress and clarify that this approach allows us to figure out the actions for the system to follow autonomously any trajectory but we need still to close the loop. Technically is a 2 phase method:1) open loop actions and 2) corrections for closed loop

A. Trajectory Generation and Decomposition

♠[7]

In this section, we examine the target goals and times, $\mathbf{g}$ and $\mathbf{t}$ *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 $\{d_i, t_i, \mathbf{x_i}\}$, where $i = 1, \ldots, m$ ♠[8], d_i is the distance travelled in each trial, t_i is the length of the trial, and $\mathbf{x_i}$ is the string of user teleoperation commands. ♠[9]In addition to the inputs given by the pilot, we are also interested in two entities

- The integral of the string of teleoperation commands, $g_i = \int_0^{t_i} \mathbf{x_i}$
- The steady-state teleoperation command, $\bar{x}_i \in \mathbf{x_i}$

The steady-state command and integral are necessary portions of the analysis ♠[10]because of the directly proportional relationship between teleop commands and system velocity. 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. Because the teleop commands are proportional to velocity, this area value gives important information about the distance traveled, as position is the integral of velocity over time. ♠[11]

The offline training data is applied to a thin-plate spline surface fit to describe the relationship between training inputs and integral and average velocity. The general form of a thin-plate spline equation is

$$f(x,y) = a_1 + a_x x + a_y y + \sum_{i=1}^{m} w_i U(||(x_i, y_i) - (x, y)||) \quad (3)$$

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:

$$\sum_{i=1}^{m} w_i = \sum_{i=1}^{m} w_i x_i = \sum_{i=1}^{m} w_i y_i = 0 \quad (4)$$

[7]NB: from the model we extract that inputs commands are responsible for specific motions of the system. Roll allows lateral motion while pitch allows longitudinal motion. Yaw enable rotations around the center of mass and thrust is mostly responsible for changes in hight along the global z coordinate. For ease of discussion in this work we don't consider yaw as we are primarily interested in planar motion ...however our approach is generalizable.... any planar motion assuming constant z can be represented as a combination of roll and pitch command...and here we need to explain how to decompose trajectories in segments

[8]NB: what is i?

[9]NB: here it's confusing. Be careful, we are mixing input with outputs.Inout is the distance, time and sequence of commands while the output is the intgeral of the commands, the steady stae teleop, and the state, position, covered by the system

[10]NB: aren't we already discussing this before in section IV?

[11]NB: here we need to show some examples of the training set. Show a few training data sampled trajectories and what we extract from these trajectories.

and the function U is of the form

$$U(r) = r^2 \log r \tag{5}$$

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,

$$\begin{bmatrix} K & P \\ P^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \mathbf{a} \end{bmatrix} = \begin{bmatrix} \mathbf{z} \\ \mathbf{o} \end{bmatrix} \tag{6}$$

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♠[12]: $g_i = f(d_i, t_i)$ and $\bar{x}_i = h(d_i, t_i)$ 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,

$$g_u = f(d_u, t_u) \tag{7}$$

$$\bar{x}_u = h(d_u, t_u) \tag{8}$$

♠[13] where d_u and t_u are the user-set desired distance and time, respectively. In Figs.2 and 3, 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 fall on the surface created with the thin-plate psline.

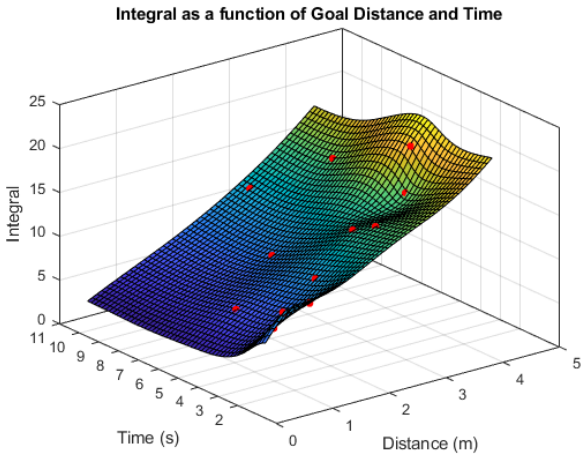


Fig. 2. Integral vs Distance and Time

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

[12]NB: connect more with the thing plate spline i.e., in our work the first regression is applied to d and t as inputs to obtain the integral of the commands $g_i = f(d_i, t_i)$ while the second one is ...

[13]NB: PROBLEM: training is performed starting from 0 reaching 0...on a new trajectory online how can you consider different starting velocities? Can we do some considerations on the training?

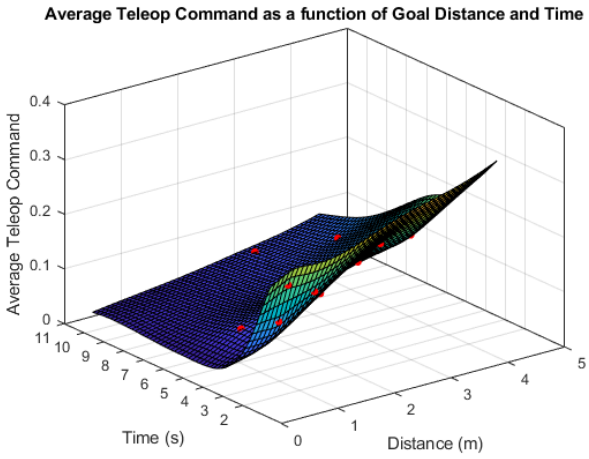


Fig. 3. Average Teleop Command vs Distance and Time

$$\sigma_{est} \approx \frac{s}{\sqrt{m}} \tag{9}$$

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

$$t_{\hat{\beta}} = \frac{|\hat{\beta} - \bar{\beta}|}{\sigma_{est}} \tag{10}$$

where $\hat{\beta}$ is the test point and $\bar{\beta}$ is the nearest known point. In equation (10), 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:

$$\sigma_{est} t_{\hat{\beta}} = |\hat{\beta} - \bar{\beta}| \tag{11}$$

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

$$\begin{aligned} \hat{\beta}_{\min} &= \bar{\beta} - \sigma_{est} t_{\hat{\beta}} \\ \hat{\beta}_{\max} &= \bar{\beta} + \sigma_{est} t_{\hat{\beta}} \end{aligned} \tag{12}$$

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:

$$\Delta_i = \sqrt{(\sigma_{d_{est}})^2 + (\sigma_{t_{est}})^2} \tag{13}$$

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.

C. Autonomous Behaviour Generation

In order for the system to autonomously reach a user-set goal, g_u and $\bar{x}_u$ are obtained using equations (7) and (8), and the offline training samples are leveraged to generate a new string of commands.

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, $\mathbf{e} \in \mathbb{R}^{1\times m}$, where each element is defined by

$$e_i = |g_i - g_u|,\ i = \{1, \ldots, m\} \tag{14}$$

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

$$\mathbf{x}^* = \mathbf{x}_i \in \mathbf{x} | e_i = \min_e(\mathbf{e}) \tag{15}$$

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 $\mathbf{x}^*$, 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

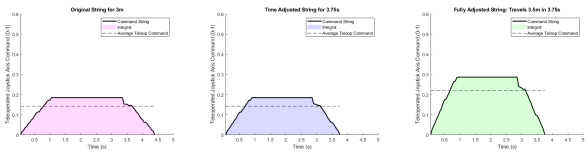
$$p(x) = \sum_{i=0}^{3} a_i x^i \tag{16}$$

where x is an entry in vector $\mathbf{x}^*$ 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 $\mathbf{x}^*$. After resizing, we obtain the time adjusted input vector $\mathbf{x}'$.

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, $\bar{x}_u$. Because distance is a function of average velocity and time, the scale time-adjusted vector $\mathbf{x}'$ is scaled such that its mean is equivalent to $\bar{x}_u$. We then obtain

$$\mathbf{x}_a = \mathbf{x}'\left(\frac{\bar{x}_u}{\bar{x}'}\right) \tag{17}$$

In Fig.4, we show visually the steps of command generation. In Fig.4(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.4(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 (16). 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.4(c), which is obtained using (17). At this point we have obtained the autonomous command string $\mathbf{x}_a$, 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. 4. 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, $\mathbf{p_r}(t)$:

$$\xi(t) = \mathbf{p}(t) - \mathbf{p_r}(t) \tag{18}$$

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

$$\mathbf{x}_c(t+1) = \mathbf{x}_a(t+1) + k_p(t)\xi(t) \tag{19}$$

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:

$$k_p(t+1) = k_p(t) + \left(\frac{\mathbf{p_r}(t)}{\mathbf{p}(t)}\right)\left(\frac{1}{t_u}\right) \tag{20}$$

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,