Parameter-free Regression-based Autonomous Control of Off-the-shelf Quadrotor UAVs

Rahul Peddi and Nicola Bezzo

***Abstract*—**

I. Introduction

Unmanned Aerial Vehicles (UAVs) have become widespread for both civilian and military applications in 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. All of these applications require the UAV to autonomously follow 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.

The development of dedicated platforms for such autonomous tasks can be expensive and not necessary since hundreds of off-the-shelf UAVs are available nowadays and for low cost. These platforms are usually well designed, very stable, and ready to fly out of the box but are typically restricted to teleoperation usage.

Generating autonomous flight behavior - subject of this paper - however, can be difficult since every UAV has a different dynamical model, which is typically not available, and reverse engineering is often not possible. Model-based approaches that include system identification, model extraction, and control design are well know procedures to deal with this issue however they are time demanding and often not precise requiring a lot of tuning and testing ♠[1]. Even when these approaches are successful, among similar vehicles there can be model mismatch due to manufacturing error, different usage, physical changes, or aging which can change parameters thus needing more tuning or sophisticated adaptive control architectures to guarantee safe and reliable control.

On the other hand, data-driven machine learning techniques like neural networks, reinforcement learning, and regression techniques have recently emerged and have demonstrated to be effective to learn from training data. The main drawbacks of such procedures is that there is still not a clear reasoning about how these techniques work and typically large and dense training sets are required to obtain precise results.

In this work we propose a novel approach that leverages both model-based and data-driven theories to generate autonomous flight behavior for an aerial vehicle from few user teleoperation demonstrations. We focus on off-the-shelf quadrotor UAVs which have a well studied dynamical and control architecture as will be described later in the paper and are by far the most common UAVs because of the growing hobby and DIY community. Our framework however can scale and apply to any other aerial vehicle. Fig. **??** shows some examples of such quadrotors focus of this paper, all characterized by having the same shape but different motors, propellers, boom size, and electronics and hence different dynamic and control models.

Different from other supervised learning approaches, in this work we leverage the knowledge about the generalized architecture for the UAV dynamics and their controller and use regression-based techniques to extract a model for closed loop trajectory tracking. The main contribution of this work is a generalized framework that combines both model-based and data-driven theories to generate autonomous control for aerial vehicles. ♠[2] ♠[3]
♠[4] ♠[5]

II. Related Work

III. Problem Formulation

In this work, we are interested in developing autonomous navigation for a UAV by leveraging knowledge about its system dynamics and controller and by leveraging a set of human-piloted demonstrations. Formally, the problem we investigate in this work can be stated as:

Problem 1: ***Parameter-free Autonomous Control of UAVs***: Consider a pre-trained ready to fly UAV, with known model architecture $\dot{x} = f(x, u, p)$ and control architecture $u = g(y)$ and unknown internal parameters p. Design a policy to generate the sequence of inputs $\mathbf{J}$ to track a given trajectory $\mathbf{p}_r(t),\ t \in [0, T]$ over a finite user defined horizon T. Specifically the policy should guarantee that the UAV position $\mathbf{p}(t)$ is always within a certain threshold δ_d from the generated trajectory

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

where $||\cdot||$ is the Euclidean norm.

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: cite some work on model id

[2]NB: need to reinforce this statement

[3]NB: need to add a figure here that summarizes the approach

[4]NB: we need to stress how different this work is in comparison with the rest of the literature on learning from demonstration. We need to show our contribution clearly

[5]NB: we need to stress that these quadrotors already come with well designed controllers and at the bare minimum we can teleop

IV. System Models

♠[6] As hinted in the introduction, we use knowledge about the UAV dynamics and control architecture to extract useful information about the system to guide a regression approach for the generation of autonomous control commands for the UAV. As mentioned above, we focus on off-the-shelf tuned and stable quadrotors that typically are ready for teleoperation but not for autonomous flight. The control architecture of the off-the-shelf quadrotor is shown in Fig. 1.

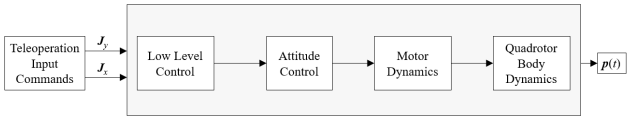


Fig. 1. Off-the-Shelf UAV Teleoperation Control Architecture

The presented structure is a gray box model, where the only information ♠[7] known are the teleoperation inputs, indicated by $\mathbf{J}_x$ and $\mathbf{J}_y$, and the output, which is the position of the vehicle, $\mathbf{p}(t)$, over time. The low level controller usually consists of multiple PID loops [CITE], which are already developed and tuned with certain specific control gains, which can be very difficult to identify. The quadrotor body dynamics can be modeled using a 12^{th} order state vector as follows:

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

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:

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

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

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

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

[CITE]

This dynamical model consists of many platform-specific parameters that are also difficult to identify, much like the controller gains discussed above. Moreover, these parameters often vary a lot between UAVs, and that further complicates the identification problem. The model and architecture provided assume that the quadrotor is moving at a constant z level with a zero yaw constraint, and therefore, we are primarily concerned with two of the inputs, u_2 and u_3. The equations for the control inputs are as follows:

$$\begin{aligned} u_2 &= k_{p,\phi}(\phi_c - \phi) + k_{d,\phi}(\dot{\phi}_c - \dot{\phi})\\ u_3 &= k_{p,\theta}(\theta_c - \theta) + k_{d,\theta}(\dot{\theta}_c - \dot{\theta}) \end{aligned} \tag{2}$$

In this work, we are interested in generating control inputs for autonomous flight when the only information available for training comes from human-piloted demonstrations. From the aforementioned demonstrations, we assume that we can obtain position ad the velocity of the system during training as well as the input commands sent to the vehicles from the user teleoperation. 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:

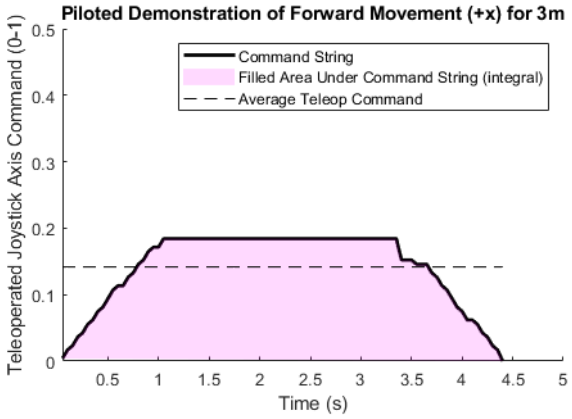


Fig. 2. A sample of a single demonstrated flight

♠[8] In Fig. 2, we show a sample of a single flight demonstration. The pilot flies the UAV forward over a distance of approximately 3 meters, with initial and final velocities of 0. The teleoperation input 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. CONNECT INTEGRAL HERE TO EQ 2 From (2), we can observe that the inputs u_2 and u_3 for roll and pitch rely on the change in the desired Euler roll and pitch angles. In order to capture that that change in the desired angles, which are set by the human teleoperation, we take the area under the curve of the input commands. The shaded portion of Fig.2 represents this integral. Since these changes in roll and pitch affect the global x and y position of the vehicle, the integral that captures these changes is connected

[6]NB: in this section we need to show the dynamical model, and control model and architecture with teleoperation. We need to stress that everything is a gray box with known model but unknown params and that the inputs and outputs are known and used later to obtain a model for autonomous fly. We need to explain the relation between teleop and u1 u2 u3 and u4 and how increasing the joystick increase the inputs and thus the integral measures mobility in the system.

[7]NB: we know input and outout data and the mdoel architecture

[8]NB: we should show this picture in parallel with the speed and position of the vehicle to connect with the integral

to the distance travelled by the vehicle. ♠[9].

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 teleoperated input command, we are able to identify the correct average autonomous input command and therefore, improve the precision in the process of generating autonomous commands, which must be fixed to a certain length (time) and travel a certain distance.

V. Methodology

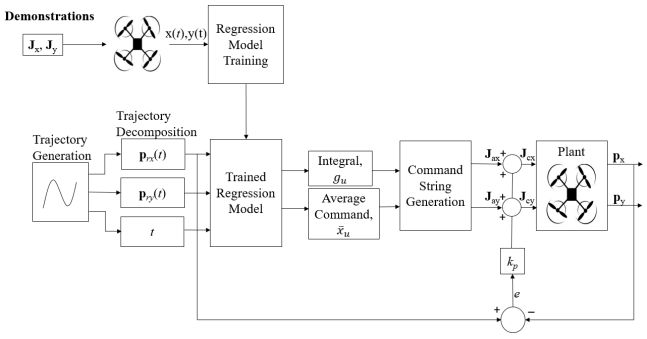


Fig. 3. Block Diagram of Proposed Approach

The proposed approach is outlined in Fig.3. The approach begins with multiple demonstrations that are used to build a regression model that enables identification of the desired integral of the input commands, g_u, and desired average teleoperation input command, $\bar{j}_u$, 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 directions is created ♠[10]. At this stage, we generate the inputs, $\mathbf{J}_x(t)$ and $\mathbf{J}_y(t)$, needed by the system to follow the given trajectory. Because these actions are created in open-loop, accurate trajectory tracking cannot be guaranteed. Tracking 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 $\mathbf{J}_{cy}(t+1)$. First, we will discuss the trajectory decomposition, as our problem ultimately is to accurately track any trajectory.

A. Trajectory Generation and Decomposition

In this section, we discuss how a smooth trajectory is generated between waypoints, and how this trajectory is decomposed into x and y components. From the system dynamics, we know that input commands are responsible for specific motions of the system. Roll enables lateral motion while pitch enables longitudinal motion. Therefore, any planar motion assuming constant z ♠[11] can be represented as a combination of roll and pitch commands, or commands that move the vehicle along the global x and y coordinates. Because this is possible♠[12], we desire to decompose the trajectory into x and y components, as any smooth trajectory can be represented as a combination of these components.

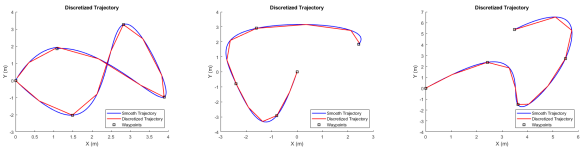
In order to generate a trajectory, we first select waypoints through which the UAV should travel. In order to determine the optimal path in our case, we generate a minimum snap trajectory. Minimum snap trajectories are associated with low control effort as they are based on the fourth derivative of position and ensure smoothness [CITE]. The general equation for a minimum snap 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 (3), we obtain the form:

$$\mathbf{p}^*(t) = c_7t^7 + c_6t^6 + c_5t^5 + c_4t^4 + c_3t^3 + c_2t^2 + c_1t + c_0 \tag{4}$$

where $c_0, \ldots, c_7$ are constants associated with specific waypoints in the trajectory. ♠[13] Having obtained the smooth trajectory, we then discretize the trajectory into n parts, each of which will have a time-step of $\delta t = \frac{T}{n}$, where T is the total amount of time allotted for the full trajectory. The discretization is done using linear interpolation within the trajectory over the interval $[\mathbf{p}^*(t_s), \mathbf{p}^*(t_{s+1})]$, where t_s is the time at the start of a segment and t_{s+1} is the end of the segment, defined as: $t_{s+1} = t_s + \delta t$. Some examples of decomposed trajectories are shown in Fig.4. All of the trajectories shown in Fig.4 start at the world frame origin $(0, 0)$.



(a) (b) (c)

Fig. 4. Discretized Trajectories

From the discretized trajectory, we can obtain the x and y components of each leg of the trajectory. Each component is calculated as follows:

$$\begin{aligned} c_x(t_s, t_{s+1}) &= \mathbf{p_x}^*(t_{s+1}) - \mathbf{p_x}^*(t_s) \\ c_y(t_s, t_{s+1}) &= \mathbf{p_y}^*(t_{s+1}) - \mathbf{p_y}^*(t_s) \end{aligned} \tag{5}$$

where $c_x(t_s, t_{s+1})$ and $c_y(t_s, t_{s+1})$ are the x and y components ♠[14] between each segment. These values are taken for each of the n trajectory segments with the same time-step, δt, and are appended to one another to create the desired output for which we will generate input commands.

[9] NB: here show that because u2 depends on ϕ_c and ϕ_c depends on x then this integral represents motion but what you really need to stress is that it is a compact measure without having to use the entire string of commands for learning

[10] NB: how?

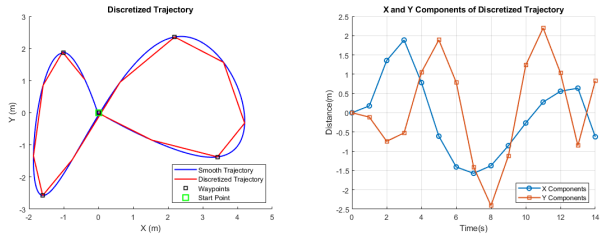
[11] NB: and yaw

[12] NB: ?

[13] NB: we need to think if these equations are really necessary here. for now leave them but keep the comment

[14] NB: what's a component? length/distance to travel?

In Fig. 5, we show an example of a generated trajectory with its discretization and the x and y components that are stitched together.



(a) (b)

Fig. 5. Example of a Generated Trajectory with its x, y Decomposition

In Fig. 5(b), we show the components at each interval. These components, when executed simultaneously, correspond to the trajectory pictured in Fig. 5(a). Having obtained the trajectory x and y components, we will now discuss the training and command generation procedures that we use to build a policy that enables autonomous tracking of the trajectory.

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. As our goal is to track a trajectory, we are interested in training on the distance traveled in the demonstrated trajectory, the time in which the pilot was able to traverse the trajectory, and the sequence of user teleoperation input commands that correspond to the trajectory: $\{d_i, t_i, \mathbf{J_i}\}$, respectively, where $i = 1, \ldots, m$ reflects the number of demonstrations. The outputs of our training phase are

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

The average 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 $\mathbf{J_i}$. The integral, meanwhile, describes the area under the string of commands, as discussed in the previous section.

1) Offline Training Data: Our training data in this work consists of multiple demonstrations of a human-pilot flying forward at varying speeds for varying distances. In Fig. 6, we show the raw data received from the demonstrations. In this case, we have training samples where the pilot flies forward and then stops, which is why there are an extended periods where there is no forward motion at the ends of the trajectories in Fig. 4.

With this data, we can train a model for trajectories that start and end with 0 speed. This training would only be effective if trajectories involved stopping intermittently throughout the trajectory. However, we desire a continuous movement between the discretized components of the trajectory. To satsify this requirement, we segment the training data

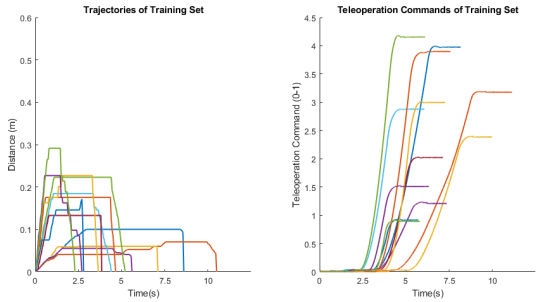


Fig. 6. Training Set, Teleoperation Commands pictured left, Trajectories pictured right

to include trajectories for demonstrations where the velocity does not start and end at 0. We are interested in three separate types of segments:

a. Trajectory Start, where the segment begins with a velocity of 0 and ends with a certain nonzero velocity
b. Trajectory Intermediate, where the segment begins and end with a certain nonzero velocity
c. Trajectory End, where the segment begins with a certain nonzero velocity and ends with a velocity of 0

In order to expand our data for this purpose, we train on portions of the training set pictured in Fig. 6, which correspond to each of the three segments we outlined. The segments are formed by cutting the training data and trajectories at the start, the middle, and the end. Shown in Fig. 7 is an example of the velocities and trajectories of the intermediary segments.

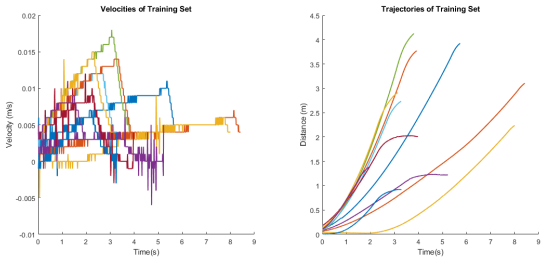


Fig. 7. Training Data with Trajectory Considerations

Training on not only trajectories that start and end with a velocity of 0, 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 middle of the trajectory (i.e., between two points that are neither the start nor end of the trajectory) needs to be generated, then the training from the trajectories in Fig. 7 is used.

2) Thin-Plate Spline Regression Analysis: The offline training data is applied to a thin-plate spline (TPS) surface regression to describe the relationship between training inputs and integral and average velocity. The general form of

a thin-plate spline equation is

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

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 smoothing 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 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 $\bar{j}_i = h(d_i, t_i)$. With the functions we have obtained, we are able to find an estimated integral and steady-state command, g_u and $\bar{j}_u$, for any given desired distance and time, d_u and t_u, respectively,

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

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

In Figs. 8 and 9, we show the surfaces that represent the two functions $g_i = f(d_i, t_i)$ and $\bar{j}_i = h(d_i, t_i)$, 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 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 [CITE] 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

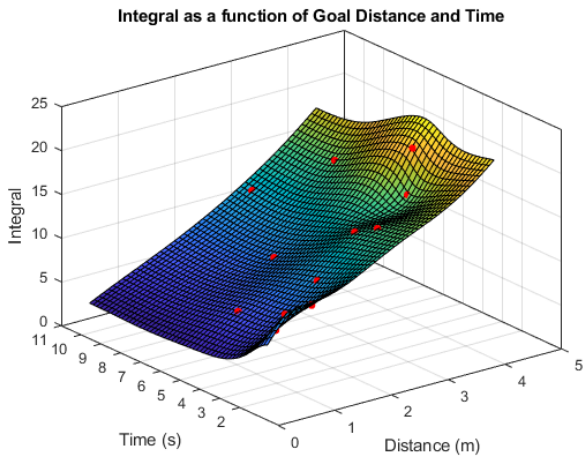


Fig. 8. Integral vs Distance and Time

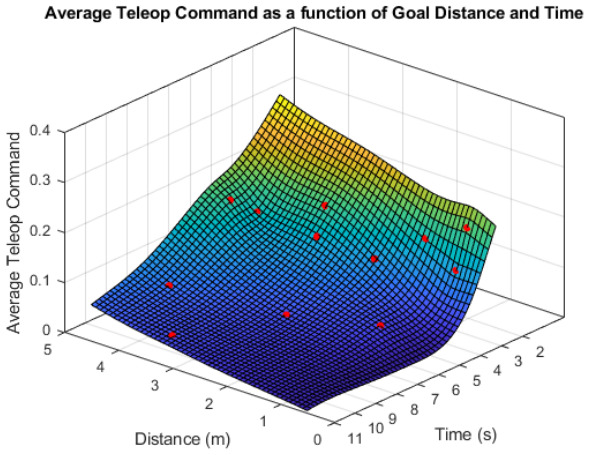


Fig. 9. Average Teleop Command vs Distance and Time

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. Therefore, 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:

$$\sigma_{est} t_{\hat{\beta}} = |\hat{\beta} - \bar{\beta}| \quad (14)$$

This maximum departure, $\sigma_{est} t_{\hat{\beta}}$, is used to set the bounds for each training data point, for example:

$$\hat{\beta}_{\min} = \bar{\beta} - \sigma_{est} t_{\hat{\beta}}$$
$$\hat{\beta}_{\max} = \bar{\beta} + \sigma_{est} t_{\hat{\beta}} \quad (15)$$

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

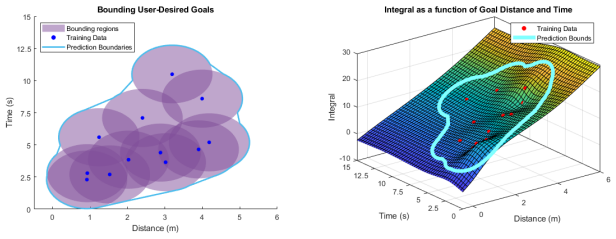
In our case, there are two inputs we are primarily concerned about for prediction; distance and time. As a result, we perform this calculation twice over, treating each of the two quantities as statistically independent, to obtain maximum departure values for each of the parameters: $\sigma_{d,est}t_{\hat{\beta}}$ and $\sigma_{t,est}t_{\hat{\beta}}$. In (10) and (11), the distance and time values are used together as input pairs, so a method to tie the two independent departure values together is desirable. Standard error rectangles [CITE], which create rectangular intervals around each data point, could be used for this purpose, but they are not necessarily accurate for smoothed surfaces, as rectangular regions would have non-differentiable corners, which are incongruous with smoothing conditions met by our TPS regression. An alternative method is to use standard error ellipses, which can be accurately attributed to smooth surfaces as per [CITE]. A general equation for a standard error ellipse for our maximum departure is shown below:

$$\left(\frac{x}{\sigma_{d,est}t_{\hat{\beta}}}\right)^2 + \left(\frac{y}{\sigma_{t,est}t_{\hat{\beta}}}\right)^2 = 1 \tag{16}$$

The ellipses at each point in our training data is calculated using the following parametrized equations:

$$\begin{aligned} x &= d_i + \sigma_{d,est}t_{\hat{\beta}}\cos r \\ y &= t_i + \sigma_{d,est}t_{\hat{\beta}}\sin r \end{aligned} \tag{17}$$

where r is the parameter from $[0, 2\pi]$, and $i = 1, \ldots, m$ and reflects each of the m training samples. Fig. 10(a), contains a pictorial representation of the standard error ellipses around each point. In Fig. 10(b), the surface shown in Fig. 8 is modified with a conforming boundary [CITE] created by the ellipses. {i need to explain that we don't actually care about this boundary but this is the best way to show it on the surface. also should i discuss that rectangles will have corners that poke out of the surface?}



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

Fig. 10. Prediction Intervals and Bounds

Given these prediction bounds, we are able to assess whether or not our trajectory components will have accurate results in the command generation portion of this paper, which follows.

C. Autonomous Input 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 component 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

$$e_i = |g_i - g_u|, \; i = \{1, \ldots, m\} \tag{18}$$

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

$$\mathbf{J}^* = \mathbf{J}_i \in \mathbf{J} | e_i = \min_e(\mathbf{e}) \tag{19}$$

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

$$b(x) = \sum_{i=0}^{3} a_i j^i \tag{20}$$

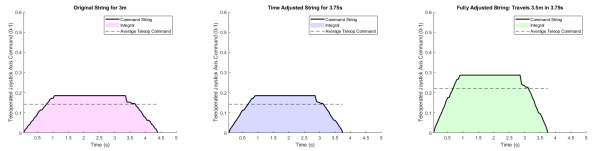
where j is an entry in vector $\mathbf{J}^*$ and a represents the coefficients of the function at each point. Bicubic interpolation takes the weighted sum of the four nearest neighbors of each entry in the command vector in order to identify a function for the intermediate points between each value in $\mathbf{J}^*$. 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

$$\mathbf{J}_a = \mathbf{J}'\left(\frac{\bar{j}_u}{\bar{j}'}\right) \tag{21}$$

In Fig. 11, we show visually the steps of command generation. In Fig. 11(a), we start with the original pre-trained command string that was shown above in Fig. 2. 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. 11(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 (20). 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. 11(c), which

is obtained using (21). At this point we have obtained the autonomous command string $\mathbf{J}_a$, which is then sent to the UAV to follow the trajectory $\mathbf{p_r}(t)$.



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

Fig. 11. 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, $\mathbf{p_r}(t)$:

$$\xi(t) = \mathbf{p}(t) - \mathbf{p_r}(t) \tag{22}$$

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

$$\mathbf{J}_c(t+1) = \mathbf{J}_a(t+1) + k_p(t)\xi(t) \tag{23}$$

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 use the ratio of the error to desired position:

$$k_p(t+1) = k_p(t) + \frac{\xi(t)}{\mathbf{p_r}(t)} \tag{24}$$

The quantity $\frac{\xi(t)}{\mathbf{p_r}(t)}$ related the current error to the desired position within the trajectory.

VI. Experiments

VII. Conclusions

VIII. Acknowledgement

References

[1] G. O. Young,