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

Rahul Peddi and Nicola Bezzo

***Abstract*— Autonomous flight in unmanned aerial vehicles (UAVs) generally require platform-specific knowledge of the dynamical parameters and control architecture. Recently, UAVs have become more accessible with off-the-shelf options that are well-tuned and stable for user teleoperation. In this paper, we develop a method to enable autonomous flight on vehicles that are designed for teleoperation with minimal knowledge of the system parameters and the control architecture. The proposed method leverages human teleoperated trajectories to train a thin-plate spline (TPS) regression model, which is then used to manipulate the pre-trained commands to generate new autonomous input commands for new trajectories. A control strategy is also proposed to adjust autonomous input commands in real-time, in order to close the loop for assured autonomy. Finally, we validate the proposed approach with trajectory-following experiments on a quadrotor UAV.**

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. Even if the application focus of this paper is on quadrotors, our framework 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. ♠[1] ♠[2]

II. Related Work

♠[3]

In recent years, we have seen many developments in the study of unmanned aerial vehicles (UAVs). A very common and widely researched task for UAVs is controlling a vehicle such that a certain trajectory can be followed. The authors in [2]–[4] provide well optimized approaches for UAV trajectory generation and tracking by utilizing near perfect knowledge of the controller and dynamical model parameters. Access to these parameters, as they are in our case, are often limited. In [5], the authors leverage exact knowledge of the model dynamics an controller architecture

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: need to reinforce this statement

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

[3]NB: skipping related work for now

to identify necessary parameters for trajectory tracking, and perform parameter estimation using particle swarm optimization, which the authors state can be a long and iterative process, which is the case for many other model-based parameter and system identification works [1], [6].

Some of the more recent advancements in this area feature machine learning and data-driven approaches, such as deep learning (DL), imitation learning (IL), and reinforcement learning (RL). In [7], the authors use DL to learn driving models for ground vehicles by using large-scale video datasets, and in [8], the authors use RL over $1,000$ training samples for accurate quadrotor trajectory tracking. These methods are model and parameter free, but use extremely large datasets to resolve that issue. In [9], the authors explore the idea of observational imitation learning (OIL), where they learn from a relatively small number of demonstrations, but also assume knowledge of the controller architecture and perform parameter estimation, which can be a daunting task without knowledge of the controller architecture. The authors in [10] use apprenticeship learning, which is similar to IL, to train a UAV to perform trajectories with a very simplified model, but there is still a necessity to learn some dynamical parameters, while assuming knowledge of several others. Despite having to learn certain parameters, these authors show the ability to learn trajectory tracking using only important information from only 20 minutes of flight time. The understated desire for smaller training sets is prevalent in many of the works in this area. Some works, such as [11] use a stochastic method known as a Gaussian Process (GP) to accurately tune a simplified UAV controller. Use of stochastic or regression based methods, such as GPs have shown the ability to execute imitation learning with limited training sets and information [12], [13].

Our work is largely based on combining the aforementioned apprenticeship learning theory with a regression method - we show how we can obtain and analyze important information from a relatively small number of demonstrations. Contrary to the model based approaches, we assume no knowledge of specific dynamical parameters, and only a general idea of the dynamical model and controller architecture. This makes parameter estimation a rather unfeasible and inefficient task. Instead, we use information obtained from our human-piloted training demonstrations to directly generate control inputs to track any trajectory, by way of training and analyzing a regression model and implementing online adaptive loop-closure.

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-configured off-the shelf ready to fly UAV, with known model architecture $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p})$ and control architecture $\boldsymbol{u} = g(\boldsymbol{y})$ and unknown internal parameters $\mathcal{P}$. Design a policy to generate the sequence of inputs $\boldsymbol{u}$ to track a user defined reference trajectory $\boldsymbol{x}_r(t),\ t \in [0, T]$ over a finite user defined horizon T. Specifically the policy should ensure that the UAV position $\boldsymbol{p}(t)$ is always within a certain threshold δ_d from the generated trajectory position $\boldsymbol{p}_r(t)$

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

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

IV. System Models

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. 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. The presented structure

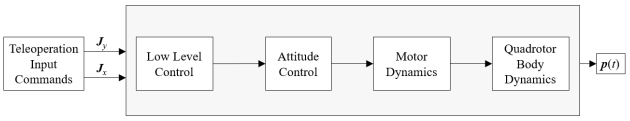


Fig. 1. Off-the-Shelf UAV Teleoperation Control Architecture

is a gray box model, where the only information known are the teleoperation inputs, indicated by $\boldsymbol{J}_x$ and $\boldsymbol{J}_y$, the output, which is the position of the vehicle, $\boldsymbol{p}(t)$, over time, and the basic model architecture. The low level controller usually consists of multiple PID loops [14], which are already developed and tuned with certain specific control gains, which can be very difficult to identify by analyzing the output of the system. The quadrotor body dynamics can be modeled using a 12^{th} order state vector as follows:

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

where $\boldsymbol{p} = [x\ y\ z]^\intercal$ 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:

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

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

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

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

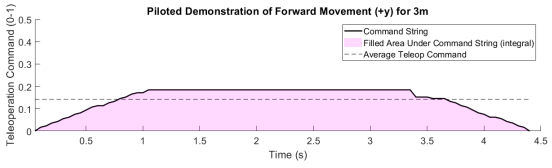
[15]

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 above 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:

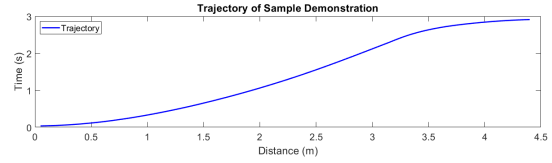
$$
\begin{aligned}
u_2 &= k_{p,\phi}(\phi_c - \phi) + k_{d,\phi}(\dot{\phi}_c - \dot{\phi}) \\
\phi_c &= -\frac{1}{g}(\ddot{x}_{des} + k_{d,x}(\dot{x}_{des} - \dot{x}) + k_{p,x}(x_{des} - x)) \\
u_3 &= k_{p,\theta}(\theta_c - \theta) + k_{d,\theta}(\dot{\theta}_c - \dot{\theta}) \\
\theta_c &= -\frac{1}{g}(\ddot{y}_{des} + k_{d,y}(\dot{y}_{des} - \dot{y}) + k_{p,y}(y_{des} - y)) \qquad (2)
\end{aligned}
$$

where ϕ_c and θ_c are the desired roll and pitch angles, respectively.

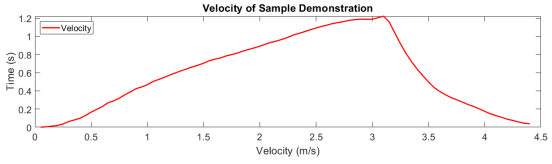
In this work, we are interested in generating control inputs for autonomous flight when the only information available for training comes from human-piloted demonstrations. Given these user demonstrations, we assume that the inputs and outputs for the UAV are visible, i.e., we can obtain position ad 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.



(a) Sample of a Single Demonstration



(b) Sample Trajectory



(c) Sample Velocity

Fig. 2. Sample Demonstration, Trajectory, and Velocity

In Fig. 2(a), we show a sample of a single flight demonstration on a Parrot BeBop 2 quadrotor which is a hobby-grade quadrotor. 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 the negative y-direction. The teleoperation command, in our case, adheres to desired pitch angles, but the user could be adjusting any value that affects motion of the system, such as acceleration, torque, or even voltage provided to the motors. Our approach is able to connect any of these commands to the change in position and velocity of the system. In our case, we connect the integral to the distance travelled by the trajectory in Fig. 2(b) anf the velocity profile shown in Fig. 2(c).

From (2), we observe that the inputs u_2 and u_3 for roll and pitch rely on the change in the desired Euler roll ϕ_c and pitch θ_c angles, and these desired roll and pitch angles are a function of the desired positions and velocities which are provided by the user in our work. Hence, each stroke in the joystick transforms into motion of the vehicle which in turns affect its changes in the global x and y position. Our first goal here is to understand this mapping between user inputs and distance traveled by the quadrotor to build an autonomous behavior. Instead of using the entire string of commands, which could be large, not uniform, and computationally expensive, in this work we use the integral of the commands, in other words the area underneath the time series of commands which is a compact measure for training a model. The shaded area in Fig. 2(a) represents this integral.

Because the integral by itself may not be enough to characterize a unique behavior of the UAV because the same integral can be achieved with commands strings for different traveling distance and velocities, here we collect also the average command. By using the combination of integral and average input commands, we can improve precision during the process of generating autonomous commands during missions, which consists of selecting a certain distance and time during run-time.

V. Methodology

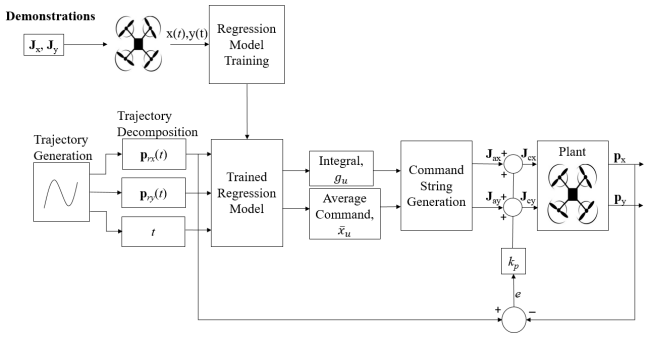


Fig. 3. Block Diagram of Proposed Approach

Our approach, outlined in Fig. 3, consists on a training phase with multiple demonstrations that are used to build a regression model to identify the desired integral of the input commands, g_u, and desired average teleoperation input command, $\bar{j}_u$, from which a sequence of commands are

synthesized to track a user-defined distance, d_u, in a time, t_u. The overall output of the regression is a sequence of commands having the same form of the inputs used for training which in our case are joystick teleoperation commands $\mathbf{J}_x(t)$ and $\mathbf{J}_y(t)$, needed by the system to follow the given trajectory. These actions are however open-loop and prone to tracking errors. Tracking error is reduced by usiing 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 $\mathbf{J}_{cy}(t+1)$. Going back to the architecture in Fig. 3, in the next few sections we will present details for each block beginning with the trajectory generation procedure.

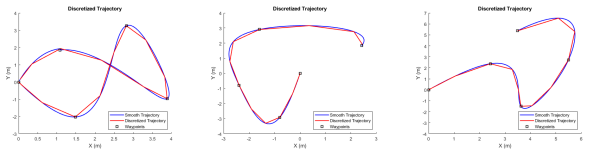
A. Trajectory Generation and Decomposition

In this section, we discuss how trajectories are generated in order to follow any trajectory. From the system dynamics, 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 and yaw can be represented as a combination of roll and pitch commands, or commands that move the vehicle along the global x and y coordinates. Hence when considering a planar x, y trajectory we perform a decomposition in the x and y directions and generate independent roll and pitch commands to follow simultaneously trajectories in the x and y directions, respectively.

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 on quadrotors which have been proven to be 4th order systems [2]. The general equation for a minimum snap trajectory is as follows:

$$\boldsymbol{p}^*(t) = \arg\min_{p(t)} \int_0^T (\dddot{p})^2 dt \tag{3}$$

Having obtained a 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 each interval $[\boldsymbol{p}^*(t_i), \boldsymbol{p}^*(t_i + \Delta t)]$, where t_i is the time at the start of a segment $i = 1, \ldots, n-1$. Here $t_1 = 0$ and $t_n = T$. Some examples of decomposed trajectories are shown in Fig. 4.



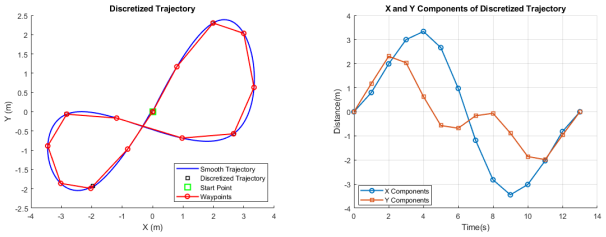
(a) (b) (c)

Fig. 4. Examples of discretized trajectories. All of the trajectories start at the world frame origin $(0, 0)$.

From the discretized trajectory, we can obtain the x and y components of each leg of the trajectory. All components can then be composed in a sequence obtaining two trajectories in the x and y directions as follows:

$$\begin{aligned} \boldsymbol{p}_x^* &= [\boldsymbol{p}_x^*(t_1), \boldsymbol{p}_x^*(t_2), \ldots, \boldsymbol{p}_x^*(t_n - 1)] \\ \boldsymbol{p}_y^* &= [\boldsymbol{p}_y^*(t_1), \boldsymbol{p}_y^*(t_2), \ldots, \boldsymbol{p}_y^*(t_n - 1)] \end{aligned} \tag{4}$$

Fig. 5(a) shows an example of a generated trajectory with its discretization while Fig. 5(b) displays the x and y components stitched together.



(a) Discretized Trajectory (b) X and Y Components

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

These components, when executed simultaneously, correspond to the discretized trajectory pictured in Fig. 5(a). The smaller is Δt the better is the approximation of the smooth trajectory. Having obtained the x and y trajectory 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. ♠[4]

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 teleoperated trials $i = 1, \ldots, m$. As our goal is to track a trajectory i, we are interested in training on the distance traveled in the demonstrated trajectory d_i, the time t_i needed by the pilot to traverse the i^{th} trajectory , and the sequence of user teleoperation input commands $\boldsymbol{J_i}$ associated with the i^{th} trajectory The outputs of our training phase are

- The integral of the string of teleoperation input commands, $g_i = \int_0^{t_i} \boldsymbol{J}_i$
- The average teleoperation input command, $\bar{j}_i \in \boldsymbol{J}_i$
- The position of the system occupied during its motion, $\boldsymbol{p}(t)$

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 $\boldsymbol{J_i}$♠[5]. 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 covering different distances with varying speeds. 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

[4]NB: it may be better to show the trajectory generation later

[5]NB: be careful that it's $\boldsymbol{J}_i$ and not $\boldsymbol{J_i}$, i.e., the i is not bold

an extended periods where there is no forward motion at the ends of the trajectories in Fig. 4.

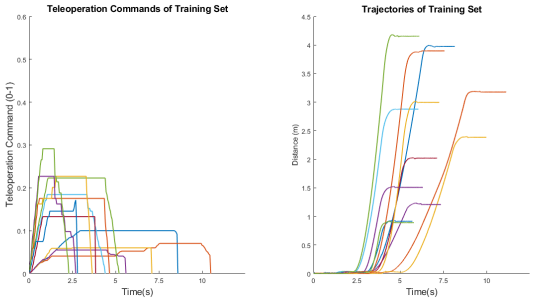


Fig. 6. Training Set, Teleoperation Commands pictured left, Trajectories pictured right

♠[6] 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. Although in this work for ease of discussion we focus on 0 initial and final velocities, if a continuous motion between the discretized components of the trajectory is needed than training can be executed on the previously obtained trajectories by segmenting the existing training data to include trajectories for demonstrations where the velocity does not start and end at 0 speed. Three separate types of segments will need to be evaluated in this case:

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

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 outlined segments. The segments are formed by cutting the training data and trajectories at the start, the middle, and the end. Shown in Fig. 7 are four sample from our training set of the velocities and trajectories that correspond to the intermediary segments starting and finishing with the same valocity.

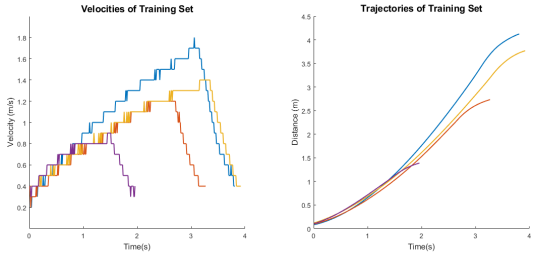


Fig. 7. Training Data with Trajectory Considerations

[6]NB: use subfigures and place the fig files for each figure in the repository so we can edit the figures. As we have discussed in the previous paper, titles and text in the figrue needs to be big and clear and readable and figrues need to span the entire column

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 input command. The general form of a thin-plate spline equation is

$$f(x,y) = a_1 + a_x x + a_y y + \sum_{i=1}^{m} w_i U(||(x_i,y_i) - (x,y)||) \quad (5)$$

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:

$$\sum_{i=1}^{m} w_i = \sum_{i=1}^{m} w_i x_i = \sum_{i=1}^{m} w_i y_i = 0 \quad (6)$$

and the function U is of the form

$$U(r) = r^2 \log r \quad (7)$$

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,

$$\begin{bmatrix} K & P \\ P^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{a} \end{bmatrix} = \begin{bmatrix} \boldsymbol{z} \\ \boldsymbol{o} \end{bmatrix} \quad (8)$$

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, $\boldsymbol{o} \in \mathbb{R}^{3\times1}$ is a column vector of zeros, $\boldsymbol{w} \in \mathbb{R}^{m\times1}$ and $\boldsymbol{z} \in \mathbb{R}^{m\times1}$ are formed from w_i and z_i, respectively, and $\boldsymbol{a}$ 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,

$$g_u = f(d_u, t_u) \quad (9)$$

$$\bar{j}_u = h(d_u, t_u) \quad (10)$$

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 splines just introduced.

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 [16]. 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 [17] with known values, which are our training points:

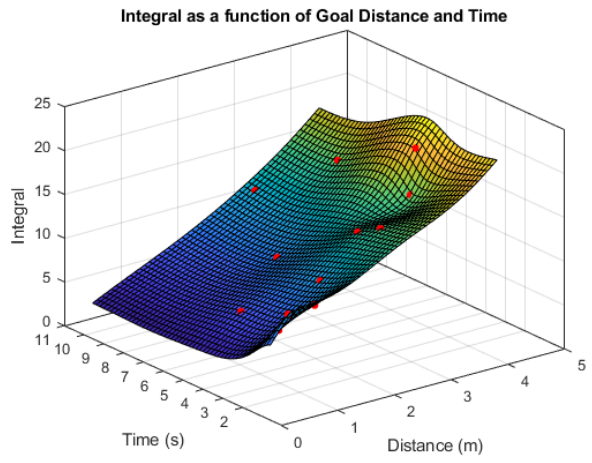


Fig. 8. Integral vs Distance and Time

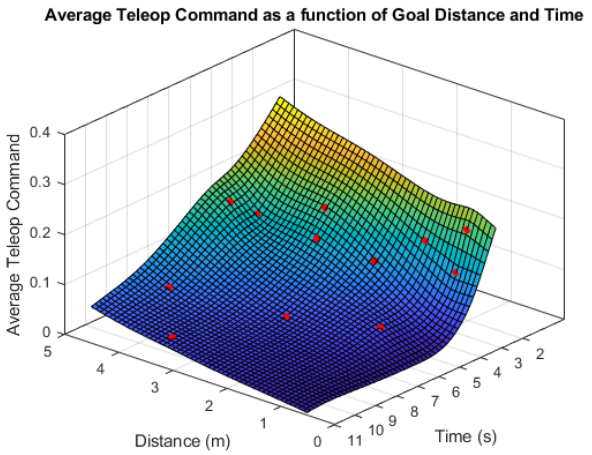


Fig. 9. Average Teleoperation Input Command vs Distance and Time

$$\sigma_{est} \approx \frac{s}{\sqrt{m}} \tag{11}$$

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♠[7]. The t-statistic is defined as the ratio of the distance between a test point from a known training point to its standard error and is computed as follows:

$$t_{\hat{\beta}} = \frac{|\hat{\beta} - \bar{\beta}|}{\sigma_{est}} \tag{12}$$

where $\hat{\beta}$ is the test point and $\bar{\beta}$ is the nearest training point. In equation (12), $t_{\hat{\beta}} \geq 1$ indicates that the test point $\hat{\beta}$ propagates an error higher than the standard error. Therefore, it is desirable to have $t_{\hat{\beta}} < 1$. After selecting a desired set-point for the t-statistic, t^*,the maximum allowable departure from known points is obtained as:

$$\sigma_{est} t^* = |\hat{\beta} - \bar{\beta}| \tag{13}$$

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

$$\begin{aligned} \hat{\beta}_{\min} &= \bar{\beta} - \sigma_{est} t^* \\ \hat{\beta}_{\max} &= \bar{\beta} + \sigma_{est} t^* \end{aligned} \tag{14}$$

where $\beta_{\min}$ and $\beta_{\max}$ are the lower and upper bounds for known parameter $\bar{\beta}$. ♠[8]

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 ♠[9]: $\sigma_{d,est} t^*$ and $\sigma_{t,est} t^*$. In (9) and (10), 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 [18], 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 the TPS regression. An alternative method is to use standard error ellipses, which can be accurately attributed to smooth surfaces as per [18]. A general equation for a standard error ellipse for our maximum departure is shown below:♠[10]

$$\left(\frac{x}{\sigma_{d,est} t^*}\right)^2 + \left(\frac{y}{\sigma_{t,est} t^*}\right)^2 = 1 \tag{15}$$

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

$$\begin{aligned} x_i &= d_i + \sigma_{d,est} t^* \cos\theta \\ y_i &= t_i + \sigma_{t,est} t^* \sin\theta \end{aligned} \tag{16}$$

where θ is an angle in $[0, 2\pi]$, and $i = 1, \ldots, m$ reflects each of the m training samples. Fig. 10(a), shows a pictorial representation of the standard error ellipses around each training point. Fig. 10(b) is a modified version of Fig. 8 with a conforming boundary [19] created by the ellipses on Fig. 10(a).

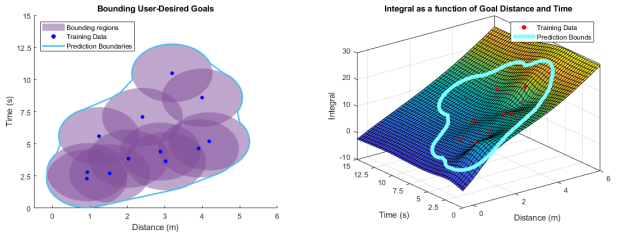
Given these prediction bounds, we are able to assess whether or not our trajectory components will have accurate results in the command generation portion of this paper, which follows.

[7]NB: CITATION

[8]NB: so each new testing point gets a different bound? Is it based only on the closest training point? What if we have 100 points close by? How do you take into account this situation that is different from having only 1 point

[9]NB: I don't understand this..why are you separating this values?

[10]NB: this section is too long..just go to the point. State that we are using this method to compute error region or whatever they are called



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

Fig. 10. Prediction Intervals and Bounds

C. Autonomous Input Generation

In order for the system to autonomously reach a waypoint generated by the discretized trajectory $\boldsymbol{p_r}(t)$, g_u and $\bar{j}_u$ are obtained using equations (9) and (10), and the offline training samples are leveraged to generate a new string of commands. We first check back with our generated trajectory components to ensure that the distances of each segment are within the bounds of our error ellipses, as calculated in (16), and displayed in Fig.10(a). If the error regions are violated, there is a possibility that the UAV will not reach the points in the desired trajectory, which is an indication of the strength and viability of the training set ♠[11]. After checking for possible violation of the error ellipses, commands are generated for each component of the generated trajectory.

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, $\boldsymbol{e} \in \mathbb{R}^m$, 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, $\boldsymbol{J}^*$, ♠[12]

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

This optimal pre-trained sample is then adjusted to reflect the user-set time, t_u. This optimal pre-trained sample is then adjusted to reflect the user-set time, t_u by performing, such that important features in the optimal command vector are not lost. Interpolation methods are often used for re-sizing complex images, and have shown effectiveness in minimizing feature loss [20]. 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 [21]. The general form of a bicubic interpolation equation is:

♠[13]

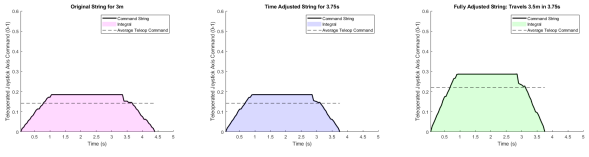
$$b(x) = \sum_{i=0}^{3} a_i j^i \tag{19}$$

where j is an entry in vector $\boldsymbol{J}^*$ and a represents the coefficients of the function at each point ♠[14]. 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 $\boldsymbol{J}^*$. After resizing, we obtain the time adjusted input vector $\boldsymbol{J}'$.

The next step is to adjust the input vector such that the system reaches the user-set goal d_u. This is achieved ♠[15] by leveraging the average input command information $\bar{j}_u$. Because we have identified in Section IV that distance is a function of this average input command and time, the scale time-adjusted vector $\boldsymbol{J}'$ is scaled such that its mean is equivalent to $\bar{j}_u$. We then obtain

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

♠[16] In Fig. 11, we show visually the steps of command generation ♠[17]. In Fig. 11(a), we start with the original pre-trained command string that was shown above in Fig. 2(a). This flight travelled ♠[18]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 aspect of bicubic interpolation from (19). If the vector were 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 (20). At this point we have obtained the autonomous command string $\boldsymbol{J}_a$, which is then sent to the UAV to follow the trajectory $\boldsymbol{p_r}(t)$.



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

Fig. 11. Command Generation Process

♠[19]

[11]NB: so what you do? We need to state and formulate how the trajectory is adapted to avoid this problem. Perhaps we can present the trajectory generation here and discuss in detail how to sample the intermediate waypoint to avoid problems

[12]NB: show $\boldsymbol{e}$

[13]NB: there are too many "This is done" in this section. They brake the flow of the paper...go to the point ...This optimal pre-trained sample is then adjusted to reflect the user-set time, t_u by performing...

[14]NB: no clue what this means...what are these coefficients?

[15]NB:

[16]NB: does the integral remain the same after all of this? Shouldn't we check and make sure that the integral is fine and in case change something while maintaining d and t?

[17]NB: rewrite this sentence

[18]NB: lasted?

[19]NB: where is the part of the text in which you show how to generate the prediction of the position occupied by the vehicle by following the determined sequence of inputs?

D. Online Adaptation of Generated Commands

The commands generated in the previous section are generated and sent to the UAV in open-loop. While we have experimentally verified open-loop results for reasonable accuracy♠[20], we desire more assured trajectory tracking by closing the loop by controlling for any possible error. While executing generated command string, $\boldsymbol{J_a}(t)$, we monitor the error between the position of the vehicle, $\boldsymbol{p}(t)$ and the reference position as per the generated trajectory, $\boldsymbol{p_r}(t)$ ♠[21]:

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

This error, $\xi(t)$, is attributed to the most recent command sent to the system. In order to adjust the next input to correct for the error, we use a proportional-integral (PI) controller to adjust the subsequent command, $\boldsymbol{J}_a(t+1)$, to obtain the closed-loop command $\boldsymbol{J}_c(t+1)$ as follows:

$$\boldsymbol{J}_c(t+1) = \boldsymbol{J}_a(t+1) + k_p\xi(t) + k_i \int_0^t \xi(t')dt' \tag{22}$$

In our work, the controller gains, k_p and k_i, are static values that are set based on the input being provided to the system. The gains are set as follows:

$$\begin{aligned} k_p &= r_p\bar{j}_a \\ k_i &= r_i\boldsymbol{J}_a(t) \end{aligned} \tag{23}$$

♠[22]

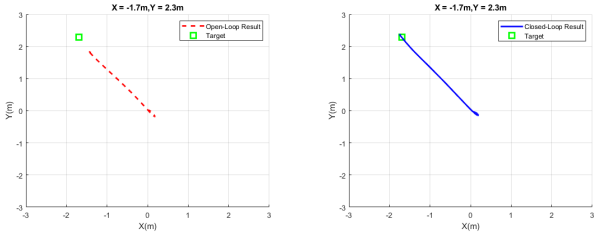
where $\bar{j}_a$ is the average input from the open loop commands, and $\boldsymbol{J}_a(t)$ is the previously sent input command ♠[23]. In (23), r_p and r_i represent the ratios applied to each of the gains. These correction ratios depend on the amount of adjustment desired by the user, which depends on certain constraints, such as flight time and flight conditions ♠[24] (i.e., different correction ratios may be desirable for indoor and outdoor flight due to disturbances and other effects). Using this controller, we are able to ensure that the error stays within the assigned threshold ♠[25]

VI. Experiments

The proposed approach was validated experimentally using a Parrot Bebop 2 quadrotor UAV, which is tuned and stable for teleoperation off-the-shelf. Since experiments were conducted indoors, a Vicon motion capture system was used to obtain the position and velocity of the vehicle. Alternatively, one could use GPS and IMU data to obtain similar features.

The training set was generated by having a human teleoperate the system 12 times with motion that only consisted of forward pitch commands. The user was instructed to fly different distances with different speeds to diverse the training set We were also able to use this same training set to generate roll commands for our UAV. The regression and error analysis was done in Matlab to obtain the surfaces pictured in Section V-B, and are shown in Figs. 8, 9, and 10. Next, the command generation was done using the Image Processing Toolbox in Matlab ♠[26]. Lastly, the experiments were conducted using ROS in conjunction with the Robotics System Toolbox to interface our regression models within the Matlab environment with ROS. ♠[27]

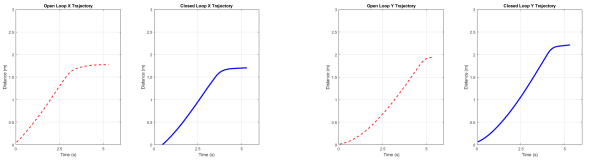
The first experiments consisted on tracking a diagonal distance, to validate that x and y components as discussed in Section V-A can be attained accurately. The trajectory we used was one where the UAV travels -1.7m in the x-direction and 2.3m in the y direction. The open-loop and closed-loop versions of the diagonal trajectories are presented in Fig. 12, where it is clear that the closed-loop solution tracks to the target ♠[28], while the open loop trajectory falls short of the setpoint. ♠[29]



(a) Open-Loop Trajectory (b) Closed-Loop Trajectory

Fig. 12. Diagonal Trajectories of UAV Experiments

In addition to the diagonal trajectories, the x and y component trajectories in open and closed loop are presented in Fig. 13. The x trajectories were flipped to appear positive for the purposes of visualization. ♠[30] ♠[31] ♠[32]

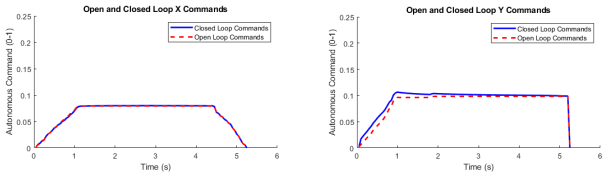


(a) X Component Trajectories (b) Y Component Trajectories

Fig. 13. Component Trajectories of UAV Experiments

Within the same experiment, the commands sent to the system in open and closed loop were also recorded. These commands in Fig. 14, show how the commands change when we close the loop using the approach in Section V-D.

[20]NB: where? I haven't seen any results yet.
[21]NB: how is this calculated?
[22]NB: I don't understand the k_i
[23]NB: is this a scalar, matrix, vector? bold upper case letters usually indicate a matrix usually or a vector...we need to mention that it is the component at time t
[24]NB: I don't understand how r is chosen
[25]NB: what threshold?
[26]NB: ??? What is this? Why you need this?
[27]NB: this section needs to be rewritten
[28]NB: What does it mean "solution tracks to the target"?
[29]NB: where is the error plot?
[30]NB: why? Please put them in the right side...this doesn't help.
[31]NB: where are the generate inputs?
[32]NB: you need to explain better how from the desired trajectory we decompose into two components, generate the inputs and then the results

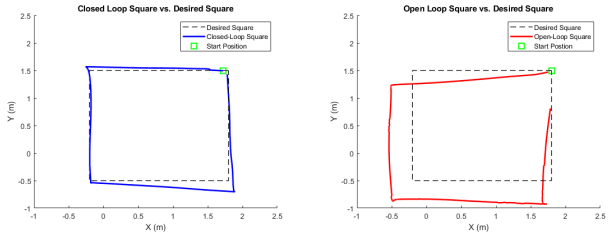


(a) X Component Commands (b) Y Component Commands

Fig. 14. Commands Generated in UAV Experiments

In Fig. 14(a), it is evident that there is little correction being done. This is because the commands given in the x direction proved to only slightly deviate from the desired target. The commands in Fig 14(b) show a clear adjustment, as the open loop trajectory clearly does not reach the target. For this experiment, the correction ratios were set to $r_p = r_i = 1/(ft_u)$, where t_u is the target time set for this trajectory, and f is the rate at which autonomous commands are sent to the system.

To further validate the approach, a square-shaped trajectory was tested.



(a) Closed Loop Square (b) Open Loop Square

Fig. 15. Square-Shaped Trajectory in Open and Closed Loop

Lastly, the letters U, V, and A were created with the UAV. The planned trajectories were created using minimum snap trajectories, as discussed in Section V-A and actual trajectories travelled by the UAV are shown in Fig. VI.

VII. Conclusions

In this work, we have presented an approach that enables autonomous flight on off-the-shelf quadrotors that are primarily tuned for teleoperation. Our approach shows that we can use a small training set along with a regression analysis to generate commands for any trajectory, along with adapting and closing the loop in real-time to reduce tracking error. For future work, we plan to increase the capabilities of our work by adding yaw commands and changing the altitude of the UAV. ♠[33]

VIII. Acknowledgement

References

[1] R. Louali and A. Benghezal, "An incremental model-based method for the implementation of an uav linear control system," in *2016 8th International Conference on Modelling, Identification and Control (ICMIC)*, Nov 2016, pp. 792–799.

[33]NB: need to add that we want to test with other sensors like cameras and imu which add errors

[2] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012. [Online]. Available: https://doi.org/10.1177/0278364911434236

[3] G. Farid, H. Hamid, S. Karim, and S. Tahir, "Waypoint-based generation of guided and optimal trajectories for autonomous tracking using a quadrotor uav," *Studies in Informatics and Control*, vol. 27, 06 2018.

[4] D. Invernizzi, M. Lovera, and L. Zaccarian, "Geometric trajectory tracking with attitude planner for vectored-thrust vtol uavs," in *2018 Annual American Control Conference (ACC)*, June 2018, pp. 3609–3614.

[5] L. Yang and J. Liu, "Parameter identification for a quadrotor helicopter using pso," in *52nd IEEE Conference on Decision and Control*, Dec 2013, pp. 5828–5833.

[6] M. Liu, G. K. Egan, and F. Santoso, "Modeling, autopilot design, and field tuning of a uav with minimum control surfaces," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 6, pp. 2353–2360, Nov 2015.

[7] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3530–3538, 2017.

[8] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, pp. 2096–2103, 2017.

[9] G. Li, M. Muller, V. Casser, N. Smith, D. L. Michels, and B. Ghanem, "Oil: Observational imitation learning," 2018.

[10] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010. [Online]. Available: https://doi.org/10.1177/0278364910371999

[11] F. Berkenkamp, A. P. Schoellig, and A. Krause, "Safe controller optimization for quadrotors with gaussian processes," *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 491–496, 2016.

[12] K. Graeve, J. Stueckler, and S. Behnke, "Learning motion skills from expert demonstrations and own experience using gaussian process regression," in *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, June 2010, pp. 1–8.

[13] M. Vaandrager, R. Babuka, L. Buoniu, and G. A. D. Lopes, "Imitation learning with non-parametric regression," in *Proceedings of 2012 IEEE International Conference on Automation, Quality and Testing, Robotics*, May 2012, pp. 91–96.

[14] E. Yel, T. X. Lin, and N. Bezzo, "Self-triggered adaptive planning and scheduling of uav operations," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018, pp. 7518–7524.

[15] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, "The grasp multiple micro-uav testbed," *IEEE Robotics Automation Magazine*, vol. 17, no. 3, pp. 56–65, Sep. 2010.

[16] K. C. Assi, H. Labelle, and F. Cheriet, "Modified large margin nearest neighbor metric learning for regression," *IEEE Signal Processing Letters*, vol. 21, no. 3, pp. 292–296, March 2014.

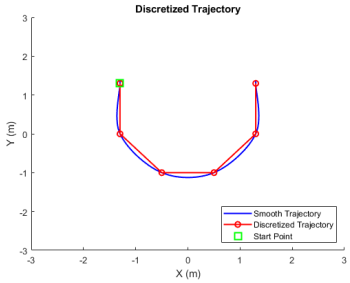
[17] S. Gonalves and H. White, "Bootstrap standard error estimates for linear regression," *Journal of the American Statistical Association*, vol. 100, no. 471, pp. 970–979, 2005. [Online]. Available: https://doi.org/10.1198/016214504000002087

[18] J. Gong, "Clarifying the standard deviational ellipse," *Geographical Analysis*, vol. 34, no. 2, pp. 155–167, 2002. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1538-4632.2002.tb01082.x

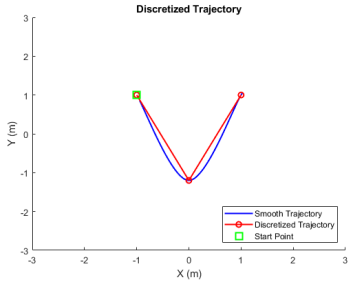
[19] P. Hu, L. Chen, J. Wang, and K. Tang, "Boundary-conformed tool path generation based on global reparametrization," in *2015 14th International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics)*, Aug 2015, pp. 165–172.

[20] Z. Dengwen, "An edge-directed bicubic interpolation algorithm," in *2010 3rd International Congress on Image and Signal Processing*, vol. 3, Oct 2010, pp. 1186–1189.

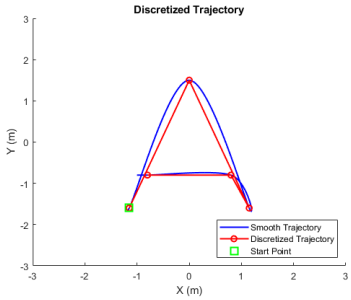
[21] P. H.S., S. H.L., and B. M. K.N., "Image scaling comparison using universal image quality index," in *2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies*, Dec 2009, pp. 859–863.



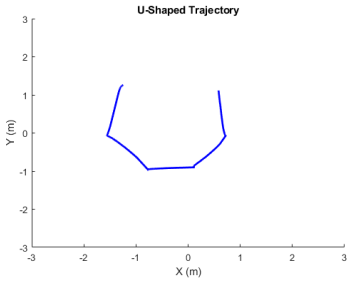
(a) Planned U Trajectory



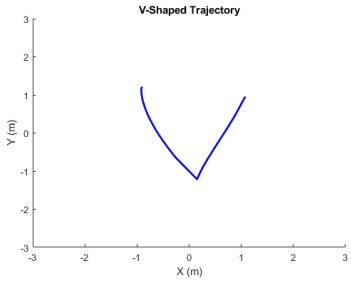
(b) Planned V Trajectory



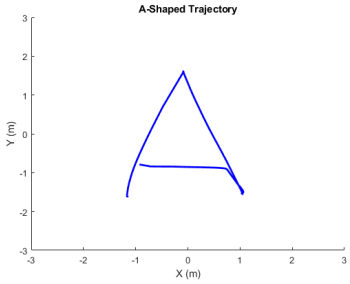
(c) Planned A Trajectory



(d) UAV U Trajectory



(e) UAV V Trajectory



(f) UAV A Trajectory