2019 International Conference on Unmanned Aircraft Systems (ICUAS)
Atlanta, GA, USA, June 11-14, 2019

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 requires 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 but due to unknown model parameters, they are typically not ready for autonomous operations. In this paper, we develop a method to enable autonomous flight on vehicles that are designed for teleoperation with minimal knowledge of the dynamical and controller parameters. The proposed method uses a basic knowledge of the control and dynamic architecture along with human teleoperated trajectories as demonstrations 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 autonomous navigation over new trajectories. A statistical approach is also presented together with a satisfiability modulo theories (SMT) solver to assess the learned prediction error and correct to minimize errors in the input generation. A robust control-based strategy is also proposed to adjust autonomous input commands during run-time for closed loop trajectory tracking. 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 (Fig. 1). 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 different dynamical parameters, which are typically not available, and reverse engineering to obtain an estimate of such parameters is often not possible. Model-based approaches that include system identification, model extraction, and control design are well known procedures to deal with this issue,

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

Fig. 1. Some examples of Off-the-Shelf UAVs, courtesy of the web.

however they are time demanding and often not precise, requiring a lot of tuning and testing [1]. Even when these approaches are successful, there can be model mismatch among similar vehicles due to manufacturing error, different usage, physical alterations, 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.

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 contributions of this work are: 1) a framework that combines both model-based and data-driven theories to generate autonomous control for aerial vehicles, 2) a statistical approach to evaluate the learning error from which a satisfiability modulo theories (SMT) solver is also leveraged to minimize the error in the generated commands for autonomous flight, and 3) a robust control technique to correct the generated commands and minimize tracking error at run-time.

Fig. 2. Motivational representation of the proposed work. Photos courtesy of the web.

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 as motivated in Fig. 2. We focus on off-the-shelf quadrotor UAVs which have a well

978-1-7281-0332-7/19/$31.00 ©2019 IEEE

studied dynamical and control architecture 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 teleoperated aerial vehicle. Fig. 1 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.

The remainder of the paper is organized as follows: In Section II, we provide a review of the state of the art in the literature on imitation and supervised learning. In Section III, we formulate our problem statement. In Section IV, we discuss the system dynamics and controller architecture, which is then followed by our methodology in Section V. Finally, we end with a discussion about our experimental validation in Section VI, and we draw conclusions and discuss future work in Section VII.

II. Related Work

In recent years, we have had many advancements in unmanned aerial vehicles (UAVs). A very common and widely researched task for UAVs is motion and path planning. 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, is often limited. In [5], the authors leverage exact knowledge of the model dynamics and controller architecture to identify necessary parameters for trajectory tracking. The same authors perform parameter estimation using particle swarm optimization, which can be a long and iterative process, as for many other model-based parameter and system identification approaches [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 very large datasets to resolve that issue. In [9], the authors explore the idea of observational imitation learning (OIL), where learning is achieved from a relatively small number of demonstrations, but knowledge of the controller architecture is assumed and parameter estimation is performed, 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 while assuming knowledge of several parameters, such as mass and moments of inertia. Despite having to learn certain parameters, the same authors show the ability to learn some maneuvers using only important information from only 20 minutes of flight time, however, they deal with an episodic task that takes a significant amount of testing for convergence. 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 leverages knowledge about the system dynamics architecture combined with regression theory to learn to fly autonomously aerial vehicles. Contrary to most model based approaches, we assume no knowledge of specific dynamical parameters, while only knowing the structure of the dynamical model and controller architecture. Instead of estimating parameters, we use information obtained from human-piloted training demonstrations to directly generate control inputs to track a trajectory, by way of training and analyzing a regression model and implementing online corrections.

III. Problem Formulation

In this work, we are interested in enabling autonomous navigation for a UAV by leveraging knowledge about its system dynamics and controller along with 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 $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ where $\boldsymbol{x}$ is the state and with input $\boldsymbol{u} = g(\boldsymbol{x})$ and unknown physical and control parameters. 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 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. 3. The presented

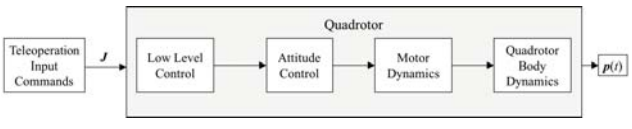


Fig. 3. Off-the-Shelf UAV teleoperation control architecture.

structure is a gray box model, where the only information known include the teleoperation inputs, indicated by $\boldsymbol{J} = [thrust, roll, pitch, yaw]$, the output, which is the position

of the vehicle, $\boldsymbol{p}(t)$, over time, and the basic model architecture (discussed below). 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 [15]:

$$\begin{aligned}
\dot{\boldsymbol{p}}^\intercal &= \begin{bmatrix} v_x & v_y & v_z \end{bmatrix} \\
\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} &= \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{1}{m} \begin{bmatrix} \cos\phi\cos\psi\sin\theta + \sin\phi\sin\psi \\ \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi \\ \cos\theta\cos\phi \end{bmatrix} u_1 \\
\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} &= \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \\
\begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} &= \begin{bmatrix} \frac{I_{yy}-I_{zz}}{I_{xx}}\omega_y\omega_z \\ \frac{I_{zz}-I_{xx}}{I_{yy}}\omega_x\omega_z \\ \frac{I_{xx}-I_{yy}}{I_{zz}}\omega_x\omega_y \end{bmatrix} + \begin{bmatrix} \frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix}
\end{aligned} \tag{2}$$

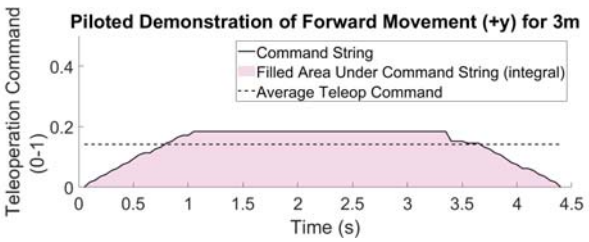
This dynamical model and the control architecture consist of platform-specific parameters that are also difficult to identify, much like the controller parameters/gains discussed above. Moreover, these parameters often vary a lot between UAVs, further complicating the identification problem. The model 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 these two control inputs are as follows:

$$\begin{aligned}
u_2 &= k_{p,\phi}(\phi_c - \phi) + k_{d,\phi}(\dot{\phi}_c - \dot{\phi}) \\
\phi_c &= -\frac{1}{g}(\ddot{x}_{des} + k_{d,x}(\dot{x}_{des} - \dot{x}) + k_{p,x}(x_{des} - x)) \\
u_3 &= k_{p,\theta}(\theta_c - \theta) + k_{d,\theta}(\dot{\theta}_c - \dot{\theta}) \\
\theta_c &= -\frac{1}{g}(\ddot{y}_{des} + k_{d,y}(\dot{y}_{des} - \dot{y}) + k_{p,y}(y_{des} - y))
\end{aligned} \tag{3}$$

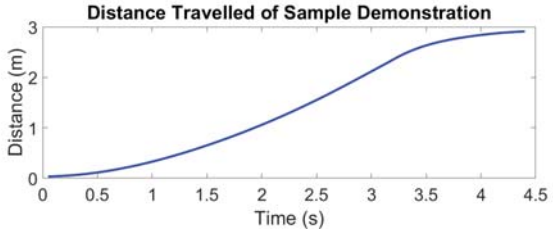
where ϕ_c and θ_c are the desired roll and pitch angles, respectively, and x_{des}, $\dot{x}_{des}$, $\ddot{x}_{des}$ refer to the desired position, velocity, and acceleration in the x direction, while y_{des}, $\dot{y}_{des}$, $\ddot{y}_{des}$ refer to the desired position, velocity, and acceleration in the y direction, respectively.

In this work, the goal is to generate 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 and velocity of the system during training as well as the input commands sent to the vehicles from the user teleoperation.

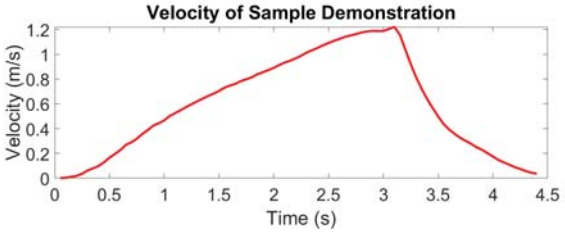
In Fig. 4, we show data from a single flight demonstration on a Parrot BeBop 2 quadrotor – the hobby-grade quadrotor used in this paper to verify our results. The UAV was teleoperated forward manually over a distance of approximately 3 meters, with 0 initial and final velocities. The teleoperation input 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 flies the UAV in the negative y-direction. The teleoperation command, in this 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 depending on the factory setting. The approach presented in this paper is able to correlate any of these commands to the change in position and velocity of the system. From (3), we observe that the inputs u_2 and u_3



(a) Sample of a single demonstration.



(b) Sample distance traveled.



(c) Sample velocity.

Fig. 4. Sample demonstration, distance traveled, and velocity.

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 case. Hence, each stroke of the joystick transforms into motion of the vehicle which, in turn, affects 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 and representative measure for training a model. The shaded area in Fig. 4(a) represents this integral.

Because the integral by itself may not be enough to characterize a unique behavior of the UAV, as the same integral can be achieved with commands strings for different traveling distances and velocities, the average input command is also of interest since (3) impacts changes in traveling velocity in (2). By using the combination of integral and average input commands, we present an approach to generate autonomous commands during a mission defined by traveling through multiple waypoints at specified distances and times.

V. Methodology

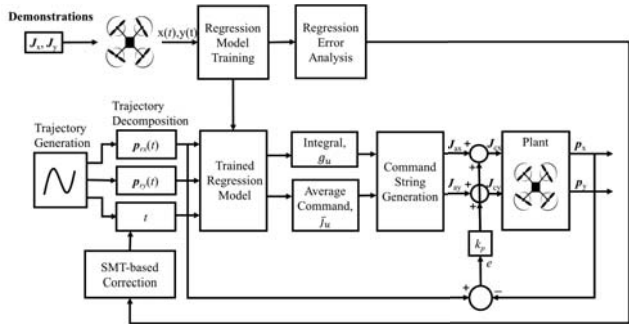


Fig. 5. Block diagram of the proposed framework.

Our framework, outlined in Fig. 5, consists of 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 for roll $\boldsymbol{J}_x(t)$ and pitch $\boldsymbol{J}_y(t)$, needed by the system to follow the given planar trajectory. These actions are, however, open-loop and prone to tracking errors. A statistical analysis is applied to the training set and used to correct the input generation error through an SMT-based approach. Tracking error is farther reduced by using a robust control approach to correct and adjust future inputs $\boldsymbol{J}_x(t+1)$ and $\boldsymbol{J}_y(t+1)$ to obtain closed loop input commands $\boldsymbol{J}_{cx}(t+1)$ and $\boldsymbol{J}_{cy}(t+1)$. Going back to the diagram in Fig. 5, in the next few sections we will present details for each block beginning with the training and regression analysis procedures.

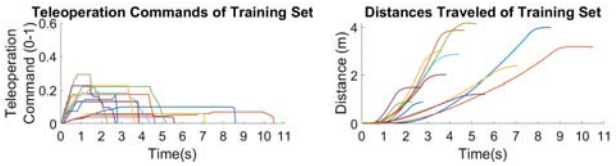
A. Regression Training from Demonstrations

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 any trajectory, 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 associated sequence of user teleoperation input commands $\boldsymbol{J}_i$. 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 $\boldsymbol{J}_i$. The integral, meanwhile, describes the area under the string of commands, as discussed in the previous section.

1) Offline Training Data: The 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. Specifically, in Fig. 6(a), we have the teleoperation commands sent by the pilot, and in Fig. 6(b), the corresponding distances traveled are displayed.



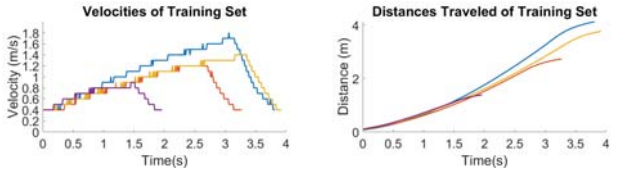
(a) Training teleoperation commands. (b) Training distances travelled.

Fig. 6. Training set.

With this data, we can train a model for trajectories that start and end with 0 speed. This training is only effective if the desired trajectory involves stopping intermittently throughout an operation. Although in this work for ease of discussion, we focus on 0 initial and final velocities, if a continuous motion throughout a trajectory is needed, then training can be executed on smooth 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

To expand the same 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 samples from our training set of the velocities and trajectories that correspond to the intermediary segments starting and finishing with the same velocity $v = 0.4$m/s.



(a) Training intermediate velocities. (b) Training intermediate trajectories.

Fig. 7. Intermediate training set.

2) Thin-Plate Spline Regression: The offline training data is applied to a thin-plate spline (TPS) [16] surface regression to describe the relationship between training inputs and integral and average input command. Given two inputs $\mathbf{x}, \mathbf{y}$, and the output $\mathbf{z}$ ($\mathbf{x}$, $\mathbf{y}$, and $\mathbf{z}$ here are axis notations), the general form of a thin-plate spline equation is

$$f(\mathbf{x},\mathbf{y}) = a_1 + a_{\mathbf{x}}\mathbf{x} + a_{\mathbf{y}}\mathbf{y} + \sum_{i=1}^{m} w_i U(||(\mathbf{x}_i,\mathbf{y}_i) - (\mathbf{x},\mathbf{y})||) \tag{4}$$

where $a_1, a_{\mathbf{x}},$ and $a_{\mathbf{y}}$ are scalar coefficients, w_i is a coefficient that corresponds to each data point, subject to the following smoothing condition:

$$\sum_{i=1}^{m} w_i = \sum_{i=1}^{m} w_i \mathbf{x}_i = \sum_{i=1}^{m} w_i \mathbf{y}_i = 0 \tag{5}$$

and the function U is of the form

$$U(r) = r^2 \log r \tag{6}$$

where $r = ||(\mathbf{x}_i,\mathbf{y}_i) - (\mathbf{x},\mathbf{y})||$. Given the corresponding $\mathbf{z}_i$ for each $(\mathbf{x}_i,\mathbf{y}_i)$ pair, we solve the following linear system to obtain the coefficients $w_i,\ldots,w_m$ and $a_1, a_{\mathbf{x}}, a_{\mathbf{y}}$,

$$\begin{bmatrix} K & P \\ P^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{a} \end{bmatrix} = \begin{bmatrix} \boldsymbol{z} \\ \boldsymbol{o} \end{bmatrix} \tag{7}$$

where $K_{ij} = U(||(\mathbf{x}_i,\mathbf{y}_i) - (\mathbf{x}_j,\mathbf{y}_j)||)$, each i^{th} row of P is $(1,\mathbf{x}_i,\mathbf{y}_i)$, $\mathbf{0} \in \mathbb{R}^{3\times3}$ 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 $\mathbf{z}_i$, respectively, and $\boldsymbol{a}$ is the column vector with elements $a_1, a_{\mathbf{x}}, a_{\mathbf{y}}$.

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

$$g_u = f(d_u,t_u) \tag{8}$$

$$\bar{j}_u = h(d_u,t_u) \tag{9}$$

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.

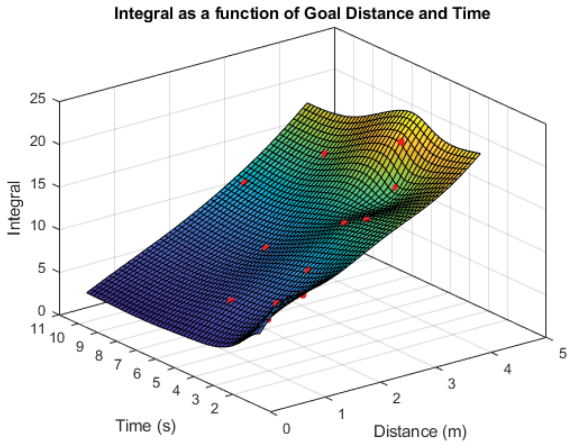


Fig. 8. Integral vs distance and time.

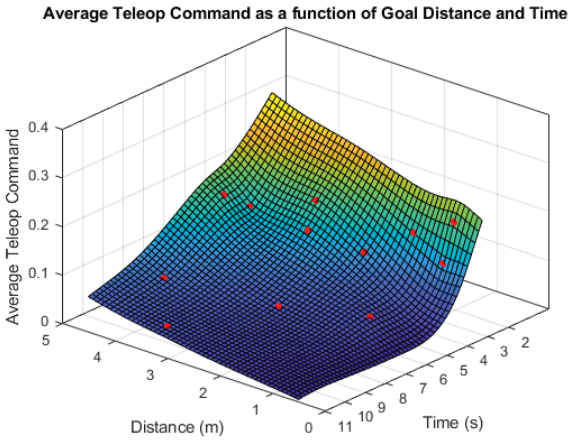


Fig. 9. Average teleoperation input command vs distance and time

3) Regression Error Analysis: 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 [17]. To deal with this problem, we consider the *standard error* of the estimate, which is a statistic that can be used to measure the accuracy of predictions given a certain type of regression [18] with known values, which are our training points:

$$\sigma_{est} \approx \frac{s}{\sqrt{m}} \tag{10}$$

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

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

where $\hat{\beta}$ is the test point and $\bar{\beta}$ is the nearest training point. In equation (11), $t_{\hat{\beta}} \geq 1$ 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 $\sigma_{est}t^*$. This maximum departure, $\sigma_{est}t^*$, is used to set the bounds for each training data point, for example:

$$\begin{aligned} \beta_{\min} &= \bar{\beta} - \sigma_{est}t^* \\ \beta_{\max} &= \bar{\beta} + \sigma_{est}t^* \end{aligned} \tag{12}$$

where $\beta_{\min}$ and $\beta_{\max}$ are the lower and upper bounds for any training point $\bar{\beta}$.

In our case, there are two inputs we are primarily concerned about for prediction; distance and time, as in (8) and (9). The distance and time values are used together as input pairs, so we take the convex hull around standard error ellipses to connect the two together [20]. The standard error ellipse, in general, is computed as follows:

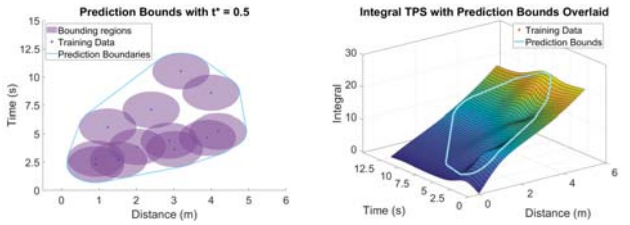
$$\left(\frac{d}{\sigma_{d,est}t^*}\right)^2 + \left(\frac{t}{\sigma_{t,est}t^*}\right)^2 = 1 \tag{13}$$

where d and t represent values corresponding to the axes for distance and time, respectively, and $\sigma_{d,est}$ and $\sigma_{t,est}$, are standard error values for distance and time, representing the x and y axes of the TPS regression, respectively. The ellipses at each point in our training data are calculated using the following equations:

$$\begin{aligned} \boldsymbol{d}_i &= d_i + \sigma_{d,est}t^* \cos\theta \\ \boldsymbol{t}_i &= t_i + \sigma_{t,est}t^* \sin\theta \end{aligned} \tag{14}$$

where θ is an angle from $[0, \ldots, 2\pi]$, and $\boldsymbol{d}_i$ and $\boldsymbol{t}_i$ are all of the points on the perimeter of the ellipse for training point (d_i, t_i), and $i = 1, \ldots, m$ where m reflects the total number of training points. A convex hull is then computed around the ellipses to provide a concise measure for the combination of distances and times that result in low error outputs when input into the TPS regression. This convex hull is generated using the Quickhull algorithm [21], in which we take the outermost points of the ellipses and form a boundary denoted $Conv(M)$, where M represents the set of all points on the perimeters of all ellipses: $\{\boldsymbol{d}_i, \ldots, \boldsymbol{d}_m\}, \{\boldsymbol{t}_i, \ldots, \boldsymbol{t}_m\}$.

In Fig. 10(a), the convex hull around standard error ellipses created with $t^* = 0.5$ around each training point is depicted. Fig. 10(b) is the TPS regression model for the integral (8) overlaid with the convex hull [22] created from the ellipses on Fig. 10(a).



(a) Prediction bounds. (b) Prediction bounds on surface.

Fig. 10. Prediction intervals and bounds.

An alternate method would have been to use standard error rectangles [20] instead, which create rectangular intervals around each data point. Rectangular regions are however not necessarily accurate for smoothed surfaces, as they have non-differentiable corners, which are incongruous with smoothing conditions met by the TPS regression.

With this analysis, we are able to provide boundary conditions for our training set, when determining what kind of trajectories we can follow. In the next section, we discuss how we build trajectories and how we use these boundaries with a satisfiability modulo theories (SMT) solver [23] to ensure that our UAV can indeed track generated trajectories.

B. Trajectory Generation and Decomposition

In this section, we discuss how we generate, discretize, and decompose smooth trajectories, subject to the regression error analysis presented in the previous section. Relating back to the system dynamics, we have established 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 and ψ 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. The optimal path is then determined by using 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 4^{th} order systems [2]. The general equation for a minimum snap trajectory is the solution of the following functional [2]:

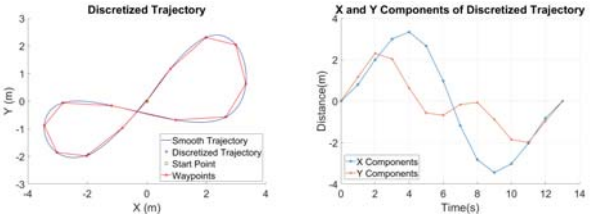
$$\boldsymbol{p}^*(t) = \arg\min_{p(t)} \int_0^T (\dddot{p})^2 dt \tag{15}$$

The obtained smooth trajectory is then discretized into n segments.

Each segment 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 $t_n = T$. 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:

$$\begin{aligned} \boldsymbol{p}_x^* &= [\boldsymbol{p}_x^*(t_1), \boldsymbol{p}_x^*(t_2), \ldots, \boldsymbol{p}_x^*(t_n)] \\ \boldsymbol{p}_y^* &= [\boldsymbol{p}_y^*(t_1), \boldsymbol{p}_y^*(t_2), \ldots, \boldsymbol{p}_y^*(t_n)] \end{aligned} \tag{16}$$

Fig. 11(a) shows an example of a generated trajectory with its discretization while Fig. 11(b) displays the x and y sequences together.



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

Fig. 11. Example of a generated trajectory with its x, y decomposition.

These components, when executed simultaneously, correspond to the discretized trajectory pictured in Fig. 11(a). The smaller Δt is, the better is the approximation of the smooth trajectory. Each of the components are then evaluated with the prediction bounds from Fig. 10. If the error regions are violated, there is a possibility that the UAV will not reach the points in the discretized trajectory. To resolve this issue, we use an SMT solver, which is a verification tool that solves a first order satisfiability problem. In our work, the condition that we want to satisfy is that each of the distances and their corresponding time is within the convex hull, $Conv(M)$ presented in Section V-A.3. Using an SMT solver, we check if all the distances in both components $d_{x,i} = ||\boldsymbol{p}_x^*(t_i{+}1){-}\boldsymbol{p}_x^*(t_i)||$ and $d_{y,i} = ||\boldsymbol{p}_y^*(t_i{+}1){-}\boldsymbol{p}_y^*(t_i)||$ are attainable in time-step Δt subject to $Conv(M)$:

$$(d_{x,i}, \Delta t) \wedge (d_{y,i}, \Delta t) \in Conv(M), \text{ with } i = 1, \ldots, n-1 \tag{17}$$

If this condition is satisfied, then the trajectory is feasible. If not, and if there exists a solution, the solver returns a time-step Δt^* for each segment of the trajectory that satisfies the condition in (17) together with the following constraint

$$\sum_{i=1}^{n}(t_{i+1} - t_i) = T \tag{18}$$

to ensure that the sum of the new time-steps is the total time allotted for the trajectory T, while also ensuring that any segment has a distance-time pair within the convex hull $Conv(M)$. If the solver still cannot obtain a result, it is indicative of an unattainable distance or distances within the trajectory, meaning that the trajectory as a whole is unattainable, and the waypoints need to be changed for example by increasing n and reducing Δt. Provided that all of the conditions for the SMT solver are met, we can obtain the correct x and y trajectory components and their corresponding times. We now discuss the command generation procedures that we use to enable autonomous tracking of trajectories.

C. Autonomous Input Generation

In order for the system to autonomously reach a waypoint generated by the discretized trajectory, g_u and $\bar{j}_u$ are obtained using equations (8) and (9) with each component of our trajectory, 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, $\boldsymbol{e} \in \mathbb{R}^m$, where each element is defined by

$$e_i = |g_i - g_u|, \ i = 1, \ldots, m \tag{19}$$

The lowest error in $\boldsymbol{e} = \{e_1, \ldots, e_m\}$ is then found and paired with the appropriate pre-trained sample, $\boldsymbol{J}^*$,

$$\boldsymbol{J}^* = \boldsymbol{J}_i \in \boldsymbol{J} | e_i = \min_e(\boldsymbol{e}) \tag{20}$$

This closest pre-trained sample is then adjusted to reflect the user-set time, t_u by performing bicubic interpolation, so 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 [24]. Bicubic interpolation is the chosen method for re-sizing, as it performs better than nearest-neighbor and bilinear interpolation methods, while only marginally increasing computational complexity [25]. The general form of a bicubic interpolation equation is:

$$b(j) = \sum_{i=0}^{3} c_i j^i \tag{21}$$

where j is an entry in vector $\boldsymbol{J}^*$ and c_i represents the coefficients of the best-fit cubic 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 around each entry in $\boldsymbol{J}^*$. This method is a result from [24], where it is described in more detail. After resizing, the time adjusted input vector $\boldsymbol{J}'$ is obtained.

The next step is to adjust the input vector such that the system reaches the waypoint at distance d_u 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 time-adjusted vector $\boldsymbol{J}'$ is scaled such that its mean is equivalent to $\bar{j}_u$:

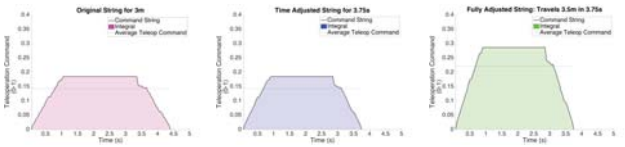
$$\boldsymbol{J}'' = \boldsymbol{J}'\left(\frac{\bar{j}_u}{\bar{j}'}\right) \tag{22}$$

Lastly, we correct for integral error. While using the fitted average input, $\bar{j}_u$, scales the commands, there may still be some error between the integral of our new command string and the desired integral, g_u. The ratio between the two integrals is leveraged to re-scale the command string:

$$\boldsymbol{J}_a = \boldsymbol{J}'\left(\frac{g_u}{\int_0^{t_u} \boldsymbol{J}'' dt}\right) \tag{23}$$

In Fig. 12, each step of the command generation process is visualized, starting with the original pre-trained command string in Fig. 12(a). This flight lasted approximately 4.4 seconds for 3 meters. For testing purposes, the user 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. 12(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 (21). If the command string were adjusted without using an interpolation

method, there may be slight variation in the mean, which could damage the integrity of the final input command string, shown in Fig. 12(c), which is obtained using (22) and (23). At this point we have obtained the autonomous command string $\boldsymbol{J}_a$, which is then sent to the UAV to reach a waypoint within the generated trajectory.



(a) Original string. (b) Time adjustment. (c) Final commands.

Fig. 12. Command generation process.

From the autonomous command, the expected path of the vehicle is generated. The integral of the final command string is taken at each time-step to determine the expected position at all times:

$$g_J(t) = \int_0^t \boldsymbol{J}_a dt \tag{24}$$

Because we know the relationship between the integral and the distance travelled at all times, we are able to leverage the equation from the TPS regression for the integral, (8), such that the inputs are time elapsed and integral of the autonomous command string and the output is the distance traveled:

$$d_J(t) = f^{-1}(g_J(t), t) \tag{25}$$

where $g_J(t)$ is the integral of the string of generated commands at time t, and $d_J(t)$ is the resultant distance travelled at t. The operation outlined in this section is performed twice to obtain autonomous command strings for each input that we are controlling, u_2 and u_3, which represent roll and pitch, respectively.

D. Online Adaptation of Generated Commands

The commands generated in the previous section are generated and sent to the UAV in open-loop. While the open-loop autonomous controls generated in this manner may be reasonably accurate when there is no noise and disturbance, we desire more assured trajectory tracking by controlling for any possible error, thereby closing the loop. While executing the 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, $\boldsymbol{p}_r(t)$:

$$\xi(t) = ||\boldsymbol{p}_r(t) - \boldsymbol{p}(t)|| \tag{26}$$

This error at time t, $\xi(t)$, is attributed to the most recent command sent to the system. A proportional controller is implemented to adjust the subsequent command, $\boldsymbol{J}_a(t+1)$, to obtain the closed-loop command $\boldsymbol{J}_c(t+1)$ as follows:

$$\boldsymbol{J}_c(t+1) = \boldsymbol{J}_a(t+1) + k_p \xi(t) \tag{27}$$

In our work, the controller gain, k_p, is a static value that is set based on the input being provided to the system as follows:

$$k_p = r_p \bar{j}_a \tag{28}$$

where $\bar{j}_a$ is the average input from the open loop commands and r_p is a constant value applied to the average input. The gain is chosen in this manner as to avoid drastic corrections that would affect the stability of the system. If the error at time t, $\xi(t)$, is very high, then having $r_p \geq 1$ could result in a drastic and unstable change in the control input. The set value of the correction ratio depends on the amount of adjustment desired by the user, which depends on constraints, such as flight time and flight conditions, i.e., different correction ratios may be desirable for indoor and outdoor flight. Using this controller, the position error is lowered to ensure that the UAV does indeed reach the generated waypoint as expected.

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 in different conditions.

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 diversify the training set. We also used this same training set to generate roll commands for our UAV since the vehicle is symmetric.

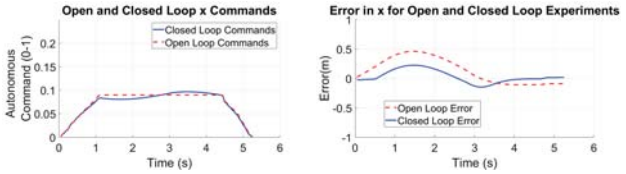
The regression and error analysis was done in Matlab to obtain the surfaces pictured in Section V-A, and shown in Figs. 8, 9, and 10. The experiments were conducted using ROS in conjunction with the Robotics System Toolbox in Matlab to interface our regression models and generated commands for the UAV.

The first experiment consisted of tracking a diagonal path, to validate that x and y components can be attained accurately, as discussed in Section V-B. Starting from the world frame origin (0,0), we set the target goal to $(-1.7, 2.3)$. The time set for this trajectory was 5.25s. Using the regression training, we obtained the open-loop commands for the x and y directions, which are pictured in red in Figs. 13(a) and 13(c), respectively. The open-loop commands yield the trajectory in Fig. 14(a).

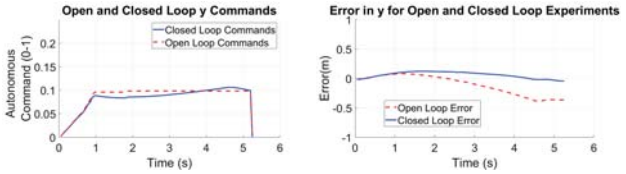
To show the effect of the proposed online adaptation procedure, we ran the same experiment with our controller, and compared results obtained in open and closed loop systems. In Fig. 13, the adapted inputs and the error associated with each of the trials in the x and y directions are shown. Using the adapted commands, we obtain the trajectory pictured in Fig. 14(b), where it is clear that the UAV reaches the target goal. Fig. 14(c), show the overlayed sequence of snapshots for the closed loop diagonal experiments in Fig. 14(b).

To further validate the approach, a square-shaped trajectory was tested, and the experimental data and actual quadrotor experiments are shown in Fig. 15 and Fig. 16, respectively.

Lastly, the letters U, V, and A were created and tracked with the UAV. using the same approach and the results are

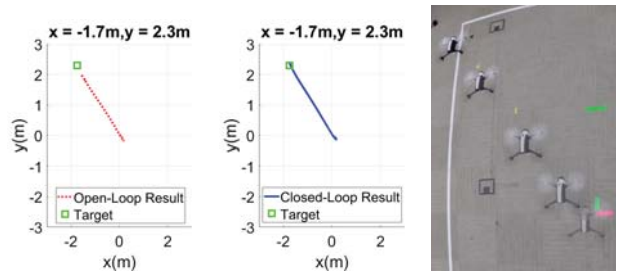


(a) x commands. (b) x error.



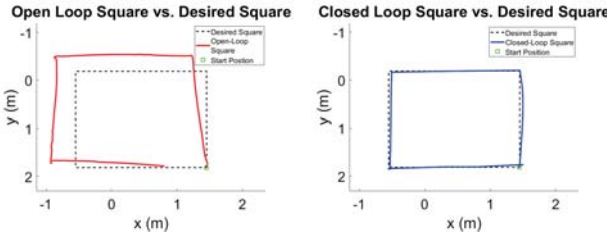
(c) y commands. (d) y error.

Fig. 13. The generated commands and the error in the x and y directions



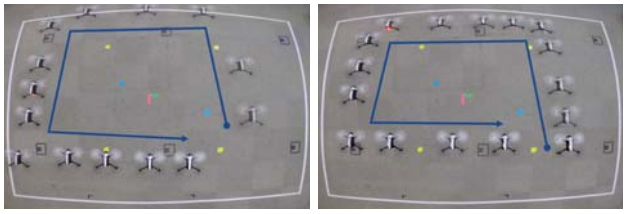
(a) Open loop results. (b) Closed loop results. (c) Experiment.

Fig. 14. Diagonal experiment results.



(a) Open loop square. (b) Closed loop square.

Fig. 15. Square-shaped trajectory in open and closed loop.



(a) Open loop square. (b) Closed loop square.

Fig. 16. Square-shaped trajectory experiments.

demonstrated in Fig. 17. For all of our tests we recorded a maximum error of 23% over the entire trajectory when running our approach in open loop and 4% when closing the loop.

VII. Conclusions and Future Work

In this work, we have presented a complete framework to enable 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 during run-time to reduce tracking error. Besides a framework for generating commands for autonomous operations, we have also proposed a statistical approach to estimate the error in the prediction and an SMT-based technique to minimize this error and generate safe inputs. Our experimental results show that the generated commands in open loop can get very close to the desired behavior but as expected they are not enough to guarantee robust tracking due to noise and disturbances at run-time. By adding a corrective procedure, however, we demonstrated that we can achieve good trajectory tracking.

This work brings together data driven and model based approaches and leverages both worlds to speed-up the learning and the deployment of autonomous vehicles. In our current and future work we plan to use onboard cameras and leverage machine vision techniques to enable autonomous flight and predict and bound future estimates about the reachable states of the system. The online adaptation part will be also expanded to consider safety critical situations like obstacle avoidance. More experiments with smoother trajectories are also in our agenda.

VIII. Acknowledgement

This work is based on research sponsored by NSF under grant #1816591 and DARPA under Contract No. FA8750-18-C-0090.

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.

[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.

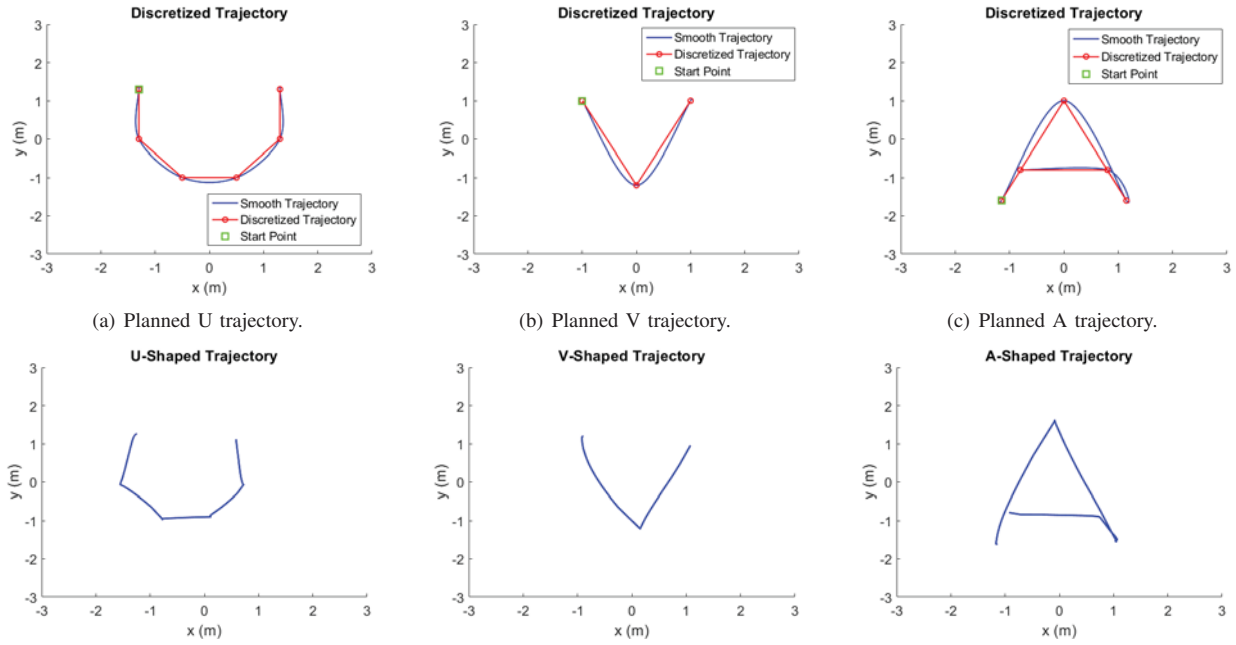
[4] D. Invernizzi, M. Lovera, and L. Zaccarian, "Geometric trajectory tracking with attitude planner for vectored-thrust vtol uavs," in *2018 Annual American Control Conference (ACC)*, June 2018, pp. 3609–3614.

[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.



(a) Planned U trajectory. (b) Planned V trajectory. (c) Planned A trajectory.

(d) Real U trajectory. (e) Real V trajectory. (f) Real A trajectory.

Fig. 17. U-V-A trajectory planning and execution.

[9] G. Li, M. Mueller, V. Casser, N. Smith, D. L. Michels, and B. Ghanem, "Teaching uavs to race with observational imitation learning," *CoRR*, vol. abs/1803.01129, 2018. [Online]. Available: http://arxiv.org/abs/1803.01129

[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] G. Donato and S. Belongie, "Approximate thin plate spline mappings," 07 2001.

[17] 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.

[18] 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

[19] T. Kyun Kim, "T test as a parametric statistic," *Korean Journal of Anesthesiology*, vol. 68, p. 540, 11 2015.

[20] 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

[21] S. Srungarapu, D. P. Reddy, K. Kothapalli, and P. J. Narayanan, "Fast two dimensional convex hull on the gpu," in *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*, March 2011, pp. 7–12.

[22] A. P. Renold and S. Chandrakala, "Convex-hull-based boundary detection in unattended wireless sensor networks," *IEEE Sensors Letters*, vol. 1, no. 4, pp. 1–4, Aug 2017.

[23] Y. Fu and M. H. Shuvo, "An approach to analyzing adaptive intelligent vehicle system using smt solver," in *2016 International Conference on Control, Decision and Information Technologies (CoDIT)*, April 2016, pp. 313–319.

[24] 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.

[25] H. Prashanth, H. Shashidhara., and K. B. Murthy, "Image scaling comparison using universal image quality index," *2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies*, pp. 859–863, 2009.