Teleop Command/Imitation Learning/Regression (in progress)

Rahul Peddi and Nicola Bezzo

***Abstract*—**

I. Introduction

II. Related Work

III. Problem Formulation

In this work, we are interested in finding a framework for a UAV to learn autonomous flight from demonstration and adapt online this flight in order to accurately follow any given trajectory within certain bounds. In addition, we are interested in finding a method to correct for any error we may discover during run-time.

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

Problem 1: ***Learning to Fly from Demonstration of Training Samples***: A human pilot flies the UAV for m different trials, recording the data for each flight. Using the trials, we aim to:

1) build a model based on the data given in the training sets, where the inputs are distance travelled d_i and time taken to travel that distance, t_i, where $i = 1, \ldots, m$ and m is the number of trials.
2) evaluate the accuracy of this model, and identify boundary conditions for model effectiveness, $[d_{\min} d_{\max}]$ and $[t_{\min} t_{\max}]$.
3) generate open-loop autonomous control input, $\mathbf{x}_a$, to send to the system given a new user-set distance and time, d_u and t_u

In addition to developing the open-loop control input, $\mathbf{x}_a$, we are also interested in developing a method to correct the input during run-time. Formally, this can be states as:

Problem 2: ***Correcting for error in-flight***: In the event that error occurs when testing the generated autonomous input $\mathbf{x}_a$, we find a method to

1) adjust the remaining portion of the control input, $\mathbf{x}'_a \in \mathbb{R}^{1\times(T-t)}$, using a PI controller to obtain a closed-loop input $\mathbf{x}_c$

IV. System Dynamics

V. Approach

Block Diagram and Explanation*

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

A. Regression Model Training

In order to build the appropriate model for autonomous command generation, we perform offline training on data collected over m human-piloted trials. The inputs of the training phase are $\{d_i, t_i, \mathbf{x_i}\}$, where $i = 1, \ldots, m$, d_i is the distance travelled in each trial, t_i is the length of the trial, and $\mathbf{x_i}$ is the string of teleoperation commands given the by the pilot. In addition to the inputs given by the pilot, we are also interested in two entities

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

The steady-state command and integral are necessary portions of the analysis because of the directly proportional relationship between teleop commands and system velocity. The steady-state command, in this case, provides information about the pilot's average in-flight velocity. The integral, meanwhile, describes the area under the string of commands. Because the teleop commands are proportional to velocity, this area value gives important information about the distance traveled, as position is the integral of velocity over time.

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

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

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

$$\sum_{i=1}^{m} w_i = \sum_{i=1}^{m} w_i x_i = \sum_{i=1}^{m} w_i y_i = 0 \quad (2)$$

and the function U is of the form

$$U(r) = r^2 \log r \quad (3)$$

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

$$\begin{bmatrix} K & P \\ P^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \mathbf{a} \end{bmatrix} = \begin{bmatrix} \mathbf{z} \\ \mathbf{o} \end{bmatrix} \quad (4)$$

where $K_{ij} = U(||(x_i, y_i) - (x_j, y_j)||)$, $P_i* = (1, x_i, y_i)$, $\mathbf{0} \in \mathbb{R}^{3\times3}$ is a matrix of zeros, $\mathbf{o} \in \mathbb{R}^{3\times1}$ is a column vector of zeros, $\mathbf{w} \in \mathbb{R}^{m\times1}$ and $\mathbf{z} \in \mathbb{R}^{m\times1}$ are formed from w_i and z_i, respectively, and $\mathbf{a}$ is the column vector with elements $a1, ax, a_y$.

Given the general framework for performing the thin-plate spline, we develop two separate relationships for our specific application:

$$g_i = f(d_i, t_i) \tag{5}$$

$$\bar{x}_i = h(d_i, t_i) \tag{6}$$

With equations (5) and (6), we are able to obtain an estimated integral and steady-state command, g_u and $\bar{x}_u$, for any given desired distance and time,

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

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

where d_u and t_u are the user-set desired distance and time, respectively.

B. Autonomous Behaviour Generation

In order for the system to autonomously reach a user-set goal, g_u and $\bar{x}_u$ are obtained using equations (5) and (6), and the offline training samples are leveraged to generate a new string of commands.

From the training set of m samples, we select the trial that has the closest integral, g_i, to the estimated integral g_u. This is done by forming an error vector, $\mathbf{e} \in \mathbb{R}^{1\times m}$, where each element is defined by

$$e_i = |a_i - a_u|, \; i \in \{1, \ldots, m\} \tag{9}$$

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

$$\mathbf{x}^* = \mathbf{x}_i \in \mathbf{x} | e_i = \min_e(\mathbf{e}) \tag{10}$$

This optimal pre-trained sample is then adjusted to reflect the user-set time, t_u. This is done by performing bicubic interpolation to resize the vector $\mathbf{x}^*$. Bicubic interpolation is selected here, as it performs better than nearest-neighbor and bilinear interpolation methods, while only marginally increasing computational complexity. The general form of a bicubic interpolation equation is

$$p(x) = \sum_{i=0}^{3} a_i x^i \tag{11}$$

Bicubic interpolation takes the weighted sum of the four nearest neighbors of each entry in the command vector in order to identify the intermediate points between each value in $\mathbf{x}^*$.

C. Online Adaptation of Generated Commands

D. Trajectory Decomposition

VI. Experiments

VII. Conclusions

VIII. Acknowledgement

References

[1] G. O. Young, Synthetic structure of industrial plastics (Book style with paper title and editor), in Plastics, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 1564.

[2] W.-K. Chen, Linear Networks and Systems (Book style). Belmont, CA: Wadsworth, 1993, pp. 123135.

[3] H. Poor, An Introduction to Signal Detection and Estimation. New York: Springer-Verlag, 1985, ch. 4.