Learning initial P.I.D values for a controller by observing operator's manual control - Report

Manash Pratim Das MPDMANASH@GMAIL.COM
Indian Institute of Technology
Kharagpur, WB, India

1. Introduction

Finding initial P.I.D values for a control loop using Root Locus method requires a good knowledge of the characteristic equations of the system and complex transfer functions. And knowing these initial P.I.D values are very important for the control of unstable systems like rotor-crafts, since starting with wrong initial values would lead to catastrophic crashes. Once a stable initial values are known, further improvement in the behavior can be achieved by tuning those parameters close to those initial values using the standard trial and error methods.

However, if the characteristic equations of the system are not known, or if the system is overly complex, or if one simply wants to get a stable set of initial P.I.D values for rapid prototyping, then by the method described below, one can control the system manually and let the system learn the initial values of the controller based on the errors and control inputs as observations. This comes from the observation that we humans adapt to systems very easily and can come up with non-precise but stable control strategies without knowing the system model and its parameters. If a pilot knows how to fly quadcopter A, the pilot can quickly adapt to a heavier or larger quadcopter B and fly them stably.

Hence, the following is an attempt to build a similar tool that I used for our team's participation in International Aerial Robotics Competition (IARC) 2016. Control inputs are a function of setpoint errors. The idea is to use a dataset of set point errors and the corresponding control inputs for them to estimate a set of P.I.D values that follow a behavior similar to human input. So, a linear regression hypothesis model was used and the parameters were obtained using Gradient Descent. It should be noted that cyber-physical systems needs higher precision of control and classical control approaches would lead to far better P.I.D values, however if you are in a hurry and you don't want to crash your quadcopter, this method would serve you well. I illustrate the approach on quadcopters with stick_position (thrust and roll, pitch, yaw rates) as control input similar to the common joystick controller. Here y denotes the a single channel stick position and x denotes the errors on which y depends.

2. Hypothesis

I used the following P.I.D control equation

$$stick\_position = P * error + I * sum\_of\_error + D * error\_difference \tag{1}$$

So, I considered the hypothesis $h_\theta(x)$ to be:

$$h_\theta(x) = \theta^T x \tag{2}$$

where

$$\theta = \begin{bmatrix} P \\ I \\ D \end{bmatrix} \tag{3}$$

$$x = \begin{bmatrix} error \\ sum\_of\_error \\ error\_difference \end{bmatrix} \tag{4}$$

3. Gradient Descent

The cost function $J(\theta)$ is defined as

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x^i), y^i) \tag{5}$$

$$Cost(h_\theta(x^i), y^i) = \frac{1}{2}[h_\theta(x^i) - y^i]^2 \tag{6}$$

And the Gradient Descent update function is given by

$$\theta_j = \theta_j - \alpha \sum_{i=1}^{m} [h_\theta(x^i) - y^i] x^i \tag{7}$$

4. Codes and Setup

4.1 Codes

The code scripts are available here [1].

- prepare_dataset.py: This file is used to filter the raw data and prepare *.csv format dataset files for the learning algorithm.
- learnP.I.D.m: MATLAB script which performs the gradient descent.
- Dataset: Directory where the sample training datasets are stored.
- Dataset/X_x.csv and Dataset/Y_x.csv contains x and y for x-axis respectively.

4.2 Setup

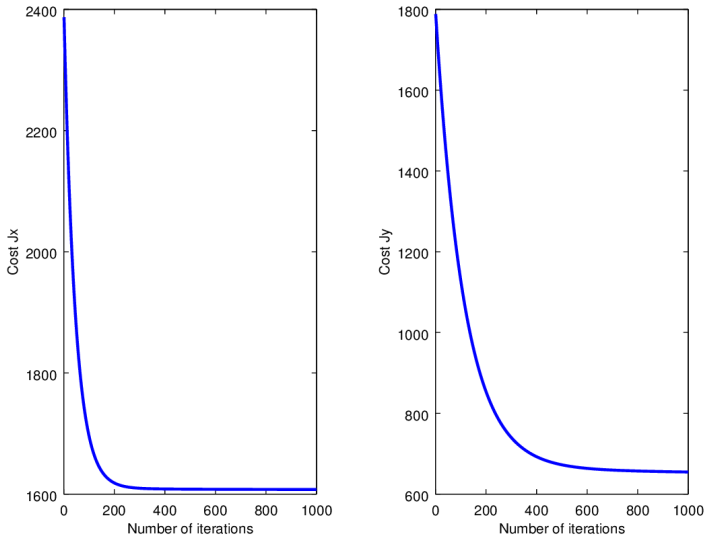
- ark_controls [2] : ROS package for autonomous control of multi-copter using mavros. Contains the P.I.D controller and the joystick interface for the user to manually control the multi-copter and record raw dataset.
- ark_simultaor [3] : Gazebo simulator used for testing.

5. Results

This tool was successfully used, most significantly during our participation in International Aerial Robotics Competition 2016, mission 7. Though the had to manually fine tune the P.I.D controller, we got a good initial value to start with. The gradient descent algorithm might give negative P, I, D values, which has to be checked in further development. I have generally observed negative value for D as in this case:
Learnt P, D values for X axis: 65.552191, -9.405076
Learnt P, D values for Y axis: 122.533537, 16.243696

The most recent dataset I collected had erroneous values for sum_of_error and hence ignored I to find only P and D values. The P, D values gave a better partially tuned controller to begin with though I used very low (~ 0.01) value in place of -9 for the D_x.

Figure 1: Convergence on the sample dataset with $\alpha = 0.03$



6. Links

1 https://github.com/mpdmanash/learnPID

2 https://github.com/quadrotor-IITKgp/ark_controls

3 https://github.com/quadrotor-IITKgp/ark_simulator

Updated on: 11 November 2016