# Dextrus Hand Controller

This report is meant to explain the steps used by me to design the position controller for the Dextrus hand.

The Dextrus hand is a tendon driven hand that uses a DC motor, with an optical encoder, that rotates a spool curling a wire used as tendon. The idea behind the controller is to associate each encoder count to a finger position.

# Dextrus Hand Controller

This report is meant to explain the steps used by me to design the position controller for the Dextrus hand.

## *First Step – obtaining a model for the motor*

To do that I used the System Identification Toolbox from MATLAB. There you can load signals in time and specify which is the input and the output in order to get a model from MATLAB.

In order to get the signals I used the interface between MATLAB and Arduino that you can find in our repository named ArduinoIO. Simply saying just load the Arduino microcontroller with the code Adioes.ino contained in the file and paste a copy of the file Arduino.m in your MATLAB search path directory.

By doing that you will be able to control your Arduino through MATLAB. In order to do that you will have to follow these steps:

1. Create an Arduino object by typing a=arduino('COM3') replacing "COM3" by the port in which your Arduino is attached. You can find this information by opening the Arduino IDE and going to Tools->Port;
2. Now you can use commands as digitalWrite(a,3,0) to send the pin 2 from your Arduino to digital LOW. You can also use analogWrite(a,3,100) in the same way to send a 100 PWM signal through the pin 3;
3. To use the Arduino to count the encoder signals connect the encoder channels to the pins 18 and 19 from the Arduino and use the command on MATLAB: encoderAttach(a,0,18,19). By doing this you will enable your Arduino to count the encoder signals. To get the count just type encoderRead(a,0). In case you wish to reset your count just type encoderReset(a,0);
4. Connect the pin 2 to the Phase pin from the motor driver and the pin 3 to the PWM pin from the motor driver.
5. Type pinMode(a,2,'output'), that is set as default input, and you will be able to change the rotation direction by digitalWrite(a,2,0) or digitalWrite(a,2,1), and the output voltage by analogWrite(a,3,*number from 0 to 250*).

At this point I used the program "resposta_encoder.m" that sets the input to 100 (PWM signal) and stores the output – encoder count – and the instant of time at what which of the count was measured saving the variables in your workspace – T for the instant of times and x for the counts.

As the System Identification Toolbox works just with a fixed sampling time signals you will have to obtain a fitting model for x that was not measured at a constant sampling rate (to see that just type T and see the different intervals of time between each sampling time). In order to do this open the Curve Fitting Toolbox and load the vectors x and T, choose the fitting type (I used spline), at the end

export the model to the workspace naming it as "fittedmodel". To get a new signal sampled with a constant sampling rate just create a new time vector tau=0:T(length(T))/99:T(length(T)); and write xnew=fittedmodel(tau);.

Now open the System Identification Toolbox and click Import Data->Time domain data, load the vector xnew to output write the input as 100*ones(100,1) and change the sampling rate to the value of tau(2). Choose the type of model (I chosen State-Space Model – order 2 – method PEM – domain continuous – form Canonical). The state-space model is the ideal in this case because we use a Linear Quadratic Regulator controller.

## Second Step – designing the controller

By completing the first step you will have a state-space model for the Dextrus finger, extract its matrices A, B, C and D. Since the order of the model is two, this will also be the number of states of the model, but note that the second state will not necessarily be the derivative from the first state. So after we will have to figure out a way to estimate that.

The LQR controller is a controller from the type $u(t) = K * \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$, with the K that minimizes the cost function defined as $J = \int \left( x(t)^T * Q * x(t) + u(t)^T * R * u(t) \right) dt$. A suggestion to choose Q and R is:

$$Q = \begin{bmatrix} \frac{\alpha_1^2}{x_{1max}^2} & 0 \\ 0 & \frac{\alpha_2^2}{x_{2max}^2} \end{bmatrix}, with\ \alpha_1^2 + \alpha_2^2 = 1$$

$$R = \frac{\rho}{u_{max}^2}$$

Where $\rho$ is a coefficient to be tuned according to the effect you desire. The larger R is, the smaller u will be and consequently the slower the controller will be. Remember that $x_2$ is not necessarily the derivative of the first state, so you can set $x_{2max} \to \infty$ with no upper bond.

Having A, B, Q and R, just type [K,S]=lqr(A,B,Q,R) on MATLAB to get the K to the controller.

Now to estimate the second state $x_2$ we can assume that the it is a linear combination of the first state $x_1$ and its derivative $\dot{x_1}$. Simulate it and do a linear regression in order to obtain the constants

$$x_2 = a * x_1 + b * \dot{x_1}$$

Write:

t=0:0.1:5;

u=ones(1,100); %input

[y,t,x]=lsim(ss(A-B*K,B,C,D),u,t) ; %simulation y->output and x->state vector

x1=x(1, :) ; %first state

x2=x(2, :) ; %second state

x1d=[0,diff(x1)/0.1] ; %derivative of the first state

[x1',x1d']\x2' %the result will be the constants mentioned above

The values I got from this method were a=0.2585 and b=-0.05853.

Now you can write your state vector as $x = \begin{bmatrix} x_1 \\ a * x_1 + b * \dot{x_1} \end{bmatrix}$. To finish you will have your controller as:

$$u(t) = K * \left( \begin{bmatrix} x_1 \\ a * x_1 + b * \dot{x_1} \end{bmatrix} - x_f \right)$$

Where $x_f$ is the final state vector desired.

## Structure of the program using ROS

The structure of the program is very simple. It subscribes to two topics: /posicao – that sends the final position desired – and /phidgets/encoder – that sends the motor's position. Notice that we are no longer using the Arduino to count the pulses from the encoder we are using the component from Phidgets to do that. That is because when the motor rotates too fast the Arduino loses the count and gets stuck.

It calculates the controller value and publishes this value on the /controle topic. We have to consider the fact that the controller may eventually return a value above 250 that is the maximum value for PWM, so make sure to not send a value greater than 250 or lesser than -250.

Load the Arduino with the code with a subscriber to the /controle topic.

Final observation: in the function vai_lqr.m I use a technique called Time-varying LQR or trajectory feedback that is used to make sure the finger will follow your model. But I think it is overkill in this case. More about here http://underactuated.csail.mit.edu/underactuated.html?chapter=10

Russ Tedrake. *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832).* Downloaded in Fall, 2014 from http://people.csail.mit.edu/russt/underactuated/