

Dynamically Modified Trajectory Control using a 2 DOF Manipulator

Or

How to Teach a Robot to Dance

Eric J. Wilhelm
2.165 Robotics and Mechatronics Final Project
December 10, 1999

Abstract

A system to control the 2.165 Laboratory 2 DOF manipulator with a source of music was designed. The system uses the derivative of the Fast Fourier Transform to generate meaningful position commands, which are fed into the closed loop control of the robot. The system had a maximum lag time between a change in the music signal and a torque being generated at the robot of 30 ms. Trajectory tracking was investigated using this system to generate a continuously changing path. Arimoto's Learning algorithm had an RMS error of 11.1mm while PID + friction compensation had an error of 20.2mm despite the path not being closed. The RMS error of PID + friction + dynamic compensation was 49.7mm due to errors in the model of the system.

Introduction

The goal of this project was to design a system to control the 2.165 laboratory robot using sound as an input to make the robot appear as though it were dancing to music, and to use this system to investigate tracking control of a trajectory which is dynamically modified. It is best thought of in two parts: the position commands generated in response to music, and the way the lab robot was able to follow those commands.

Experimental Setup

This section describes the hardware and software used to generate commands based on music. It begins with a description of the algorithm used external of the lab computer to process the music and then discusses how that data is transferred to the lab computer and changed into useful commands. The lab robot's hardware and software is only briefly mentioned when directly applicable to the discussion of the music position generator. A full discussion including the device driver, interrupt service routine, and hardware can be found in the lab handouts and hardware manuals.

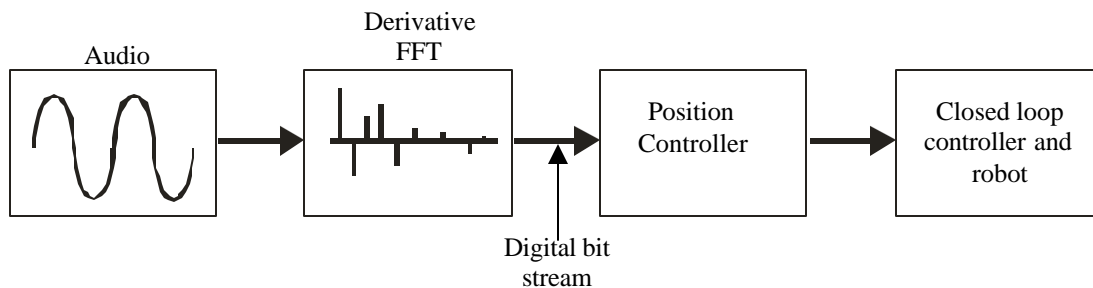


Figure 1 System Block Diagram

Figure 1 shows the overall block diagram of the system. An analog music signal is digitized and fed into a computer. The Fast Fourier Transform (FFT) is used to decompose the signal into its frequency components every n sampling periods (where n is discussed in the FFT section). The derivative FFT is found from two successive outputs of the FFT and useful information is extracted from the derivative FFT by the Position controller. These position commands are fed into the closed loop controller and robot. The flow line label digital bit stream represents the change from a laptop to the lab computer. A Pentium-II 366 MHz computer was used to process the audio and find the derivative FFT. The input for the position controller was sent in digital form from the laptop's parallel port the digital input of the lab computer's hardware. The Position controller ran along side the closed loop controller in the interrupt service routine on the lab computer. Two computers were used because it was unclear if the lab computer would be able to process the audio and run the closed loop control at the same time. Processing the audio on a separate computer also allowed that side of the system's development to be done outside of the [D'Arbeloff Laboratory](#).

Audio

The system could be operated in real-time, using the input from a live music source, but was not for this set of experiments. Varying degrees of the audio processing were done offline. Using prerecorded music from a source such as a .wav file (which is pulse code modulation encoding or PCM, the same format used on compact discs) removed the analog to digital (A/D) conversion step; using a prerecorded and MPEG (Moving Picture Experts Group) compressed audio source removed both the A/D conversion step and the FFT calculation because the MPEG format actually stores the audio data in a combination of frequency space and time space, so the FFT is pre-calculated. For the experiments described here both MPEG and .wav audio sources were used, and the differences between the two formats had no effect on the results.

FFT

Here only the parameters relevant to the system are discussed, as a full discussion of the FFT algorithm is best left to a text on digital signal processing. The FFT was carried out with a sample size of 288 for both the right and left channel of audio yielding a total sample size of 576 samples. Since the FFT algorithm typically works best with sample sizes that are powers of 2, the sample size for each channel was increased to 512 by padding it with zeros. A sample size of 288 represents the trade off between a highly accurate FFT with a large number of individual bands and a very responsive system. If the FFT is being calculated in real-time, it can only be found at a maximum rate of once every 6.5 ms for a 44.1 kHz audio sampling frequency since that is the amount of time it takes to collect the 288 samples, regardless of calculation time.

The derivative FFT was found by

$$\frac{FFT[k+1, n] - FFT[k, n]}{\Delta t},$$

where $FFT[k, n]$ is the value of the n -th band at sampling time k and Δt is the sampling time. The sampling time used on the laptop was 10 ms. Since the audio processing was not done under a strictly controlled interrupt service routine, the sample time had the ability to increase as the processing demands of the laptop increased. However, this was not a problem because the audio processing was always the only program running on the laptop.

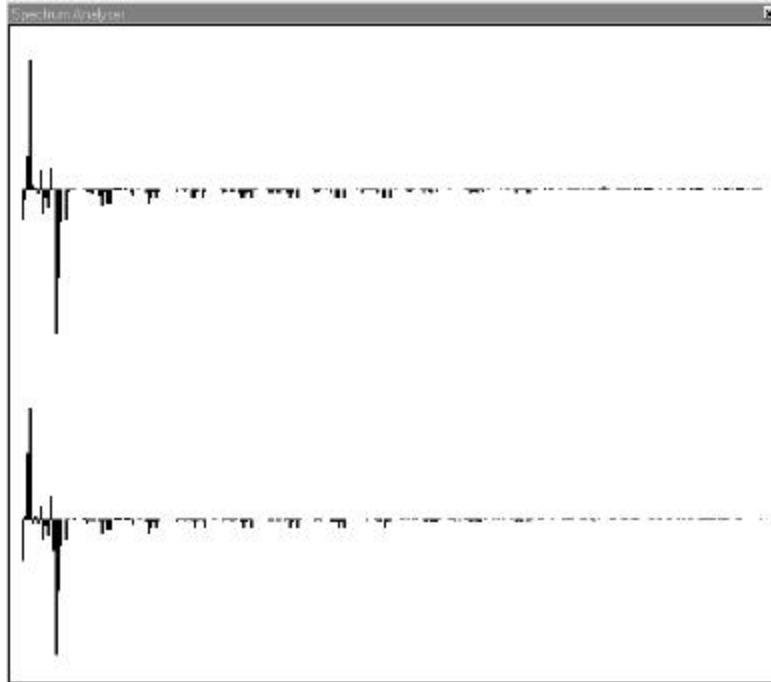


Figure 2 Derivative FFT Plot to screen

The derivative FFT was plotted to the screen of the laptop using C++ graphics tools. A screen capture is shown in Figure 2. As is typical in music, harmonics spaced at regular intervals, decreasing in magnitude as frequency increases, are clearly visible. Low, mid, and high frequency ranges were set to extract useful data. When the average magnitude of the values within a range exceeded that range's threshold value a useful musical change was assumed to have occurred. A bit was changed on the laptop's parallel port when a change was detected. This corresponds to a 100 Hz, 3 bit data stream from the laptop to the lab computer. The parallel port limited this scheme to 8 bits, or 256 possible states, to send information to the lab computer. However an external digital interface could be used to greatly increase the number of possible states.

Position Controllers

Two controllers were developed to take data from the derivative FFT and use it to control the lab robot. The first, Circular position control, has the robot trace a circle whose center point is constantly moving. The second, Arc position control, sweeps the endpoint of the robot through an arc which is "disturbed" by changes in the music. Both controllers were written in C++ and assimilated into the UserFunc.ccp function, which executed every 10 ms.

Circular Position Controller

Figure 3 shows a position plot of the commanded position and the instantaneous center generated by the circular position controller in response to a music signal. The time to complete one circle was set by the low frequency while the mid and high frequencies caused movements of the center in the x and y directions. The iteration time in the figure

is approximate 900 ms. Setting the low frequency range such that it captures changes in the music caused by a bass drum or another instrument which is relatively repetitive allows the controller to generate commands which “sync” the robot and the tempo of the music. Changes in the mid and high frequency give the dance variety by shifting the movement around the robot’s workspace.

The complied and uncompiled C++ code for the Position controllers is available on the 2.165 Lab computers in the users/ewillhelm/ directory in the file UserFunc.cpp, so only a very brief description of the code used position controllers is given here. If the state of the low frequency bit changes from false to true, a counter is zeroed and starts counting once every 10 ms. When the state of the low frequency changes again from false to true the value of the counter is used to determine the time between beats in the music. This number is multiplied by two and set as the interval, so that every two beats the robot completes a circle. A digital filter is used to ignore changes that are too fast; they are presumed to be the same beat which has already initiated the counter, just one sampling period later. If the state of the mid or high frequency range bits change from false to true the center point has a constant value added to it for the next 15 sampling periods (150 ms). Moving the center point a small amount over 15 sampling periods prevents very abrupt movements which cause the robot to move jerkily. Again, a digital filter is used to ignore changes, which come in too fast.

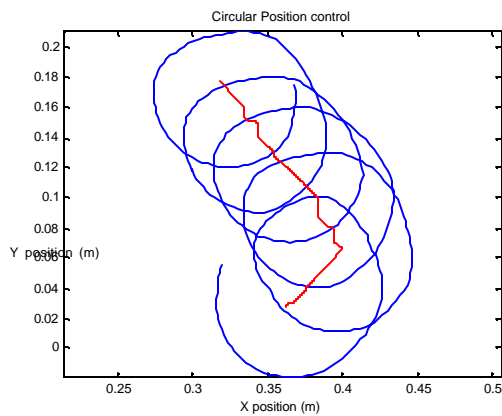


Figure 3 Circular Position Controller commanded position

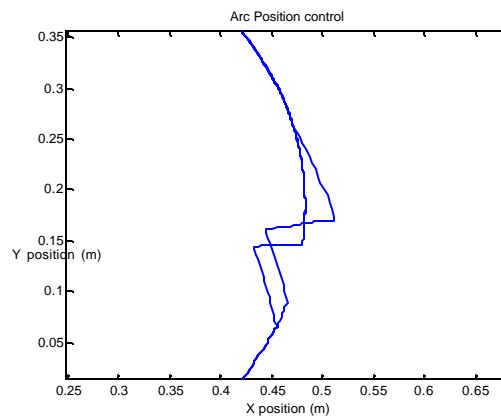


Figure 4 Arc Position Controller commanded position

Arc Position Controller

Figure 4 shows a plot of the commanded position generated by the Arc position controller in response to a source of music. Similarly the Circular position controller the interval is set by the low frequency, and the controller sweeps the first joint of the robot back and forth at a constant velocity every interval. Thus the robot appears to be waving its arm back and forth every two beats. If the high or mid frequency bits change a constant value is added to the position of the second joint for 15 sampling periods. These disturbances make the robot appear to act on frequencies from instruments such as cymbals or horns. Once the 15 sampling periods are over the second joint returns to its equilibrium position over another 15 sampling periods. As in the Circular position controller, digital filters are used to reject changes in state which occur too quickly.

Results

This section gives the experimental results of the system. As per the goals of the system the results are divided in the ability to control the 2.165 lab robot with music, a responsive system, and tracking control using music to generate a continuously varying command signal.

Control with Music

As shown in the video presentation, the robot was actually controlled in a meaningful way by a source of music. A quantitative measure of this result is very difficult to obtain, and this report is not the proper forum for a discussion on what is good or bad robotic dancing.

Responsive System

The responsiveness of a system can be thought of as the maximum time for an input to create a useful output. For this system that maximum time is approximately 30 ms. This delay is calculated as follows, 10 ms to calculate the FFF[k], another 10 ms to calculate FFT[k+1] and the derivative FFT, and then at most 10 ms for the lab computer to read the data and generate a command. Outputs as seen in physical movement of the robot are limited, of course, by things such as the power of the actuators and the inertia of the robot and are not taken into account here. 30 ms, or 33 Hz is near the limit as to what the human eye can perceive. Two representative examples are TV and computer monitors. TV signals are typically still images cycled at 24-30 Hz giving the impression of motion. Computer monitors have a rate at which they redraw the image on the screen, which is usually around 60-80 Hz. Based on these metrics, 33 Hz can be considered a responsive system.

Tracking Control

A typical run of the robot is shown in Figure 5. Tracking of the commanded position was determined by calculating the root mean square (RMS) value of the error. Both polar and rectangular coordinates were used. In polar coordinates the radius of the desired circle was determined by looking at the desired position with respect to the instantaneous circle center. The actual radius was found again with respect to the instantaneous center and this difference summed over time in the RMS sense. In rectangular coordinates the error was the simply the distance between desired and the actual position. Polar coordinates were used in an attempt to remove the lag of the system from the RMS calculation. If the robot is following a circular trajectory the error can be thought of as the difference between the radii while the lag is the difference in angle.

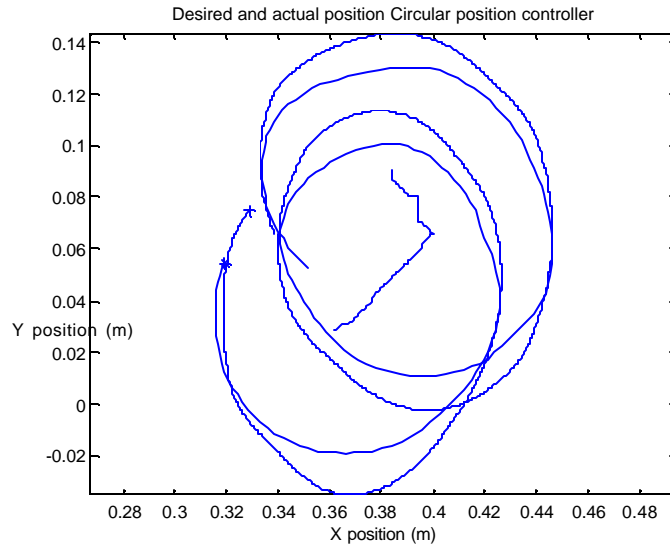


Figure 5 Desired and actual position using Circular position controller, desired starts at the ‘*’ and actual starts at the ‘+’, 2 seconds in length.

Figure 6 shows the RMS error, in both rectangular and polar coordinates, for four different closed loop controllers with positions generated by Circular position control from a source of music. The Circular position controller is the more interesting of the two position controllers because it generates commands which do not generally all lie on the same arc. The music source and time of trial was the same for each trial.

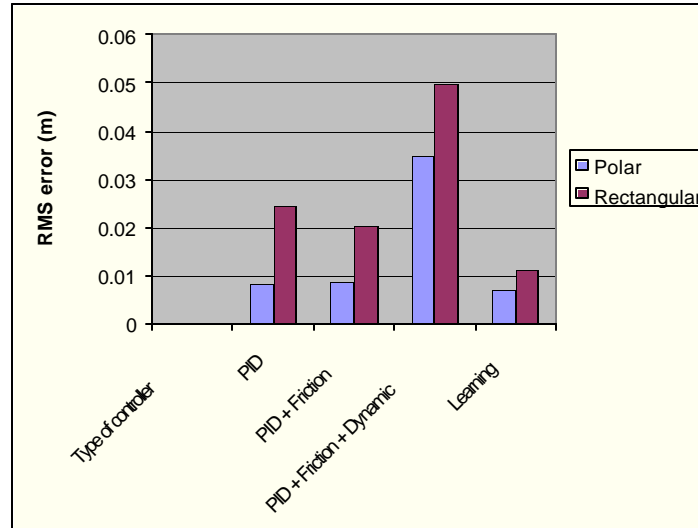


Figure 6 RMS error in polar and rectangular coordinates for four different closed loop controllers

Surprisingly the RMS error for learning control was the lowest. Figure 7 shows the RMS error of PID+Friction compensation and Arimoto’s learning algorithm over time. The gains associated with the learning controller were $\eta = 100$ and $\lambda = 10$.

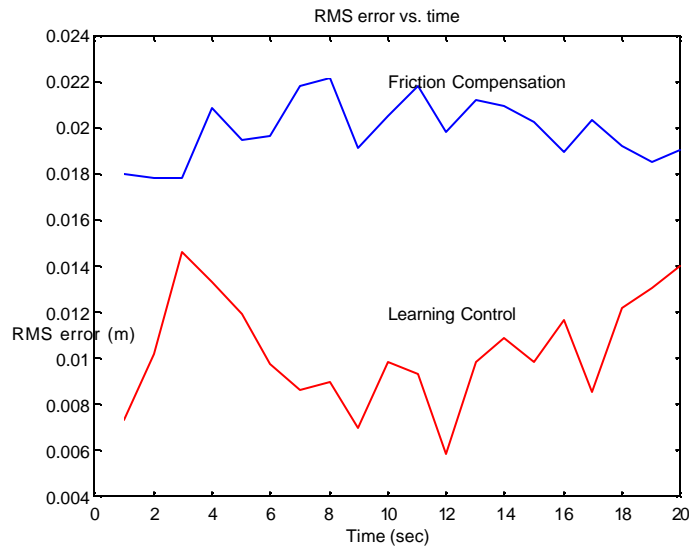


Figure 7 RMS error over time.

Discussion

The discussion section will focus on the trajectory tracking results. Finding that learning control has the lowest RMS error was surprising for two reasons: the trajectory of the robot does not return to its starting point forming a closed loop, and the physical performance of the robot under learning control was not as smooth as PID or PID + friction compensation.

When run under learning control the robot would follow the circle for several seconds and then start shaking, but it continued to follow the proper trajectory. The dynamics of the transmission between joint #2 and the motor caused the shaking. One could actually see the tension in the belt being increased and then relaxed. Changes in the learning control gains had the effect of changing how long the robot would run until the transmission dynamics were excited, but had little effect on the overall RMS error.

Before the dynamics of the transmission were fully excited the trajectory tracking of learning control was much better than the other controllers. The time rate change of the center point of the commanded circle was small enough that the errors from the last circle, even though translated, were still helpful in driving the endpoint closer to the desired endpoint. Arimoto's learning algorithm is based on a command, which includes the robot's error the last time around, and is of the form

$$u_i[k+1] = u_i[k] + \alpha e_i[k] + \beta \bar{e}[k],$$

where $u_i[k]$ is the command at the k -th iteration of tracing, α and β are constants, and $e_i[k]$ is the error at the k -th iteration. The commands and the errors are summed over time until the error is zero and the command at iteration $k+1$ equals the command at

iteration k . In this system the time rate change of the center point was slow enough that errors from the last from the last iteration were still applicable to the $k+1$ iteration.

From a slow-motion animation of the commanded position and the actual position it was clear that the actual position would often anticipate the changes in the command signal before they occurred under learning control. A similar animation of the PID+friction compensation controller showed that the actual position merely followed the command signal with a lag and error.

A very interesting continuation of this work would be to develop a controller based on learning control, which based the command signal only on a limited number of iterations in the past rather than the complete history. Arimoto's learning controller computes the command signal as a sum of the all the past command signals and errors, which certainly plays a part in exciting the higher order dynamics of the transmission system when the system does not trace a close loop. Although the system would not converge on the desired trajectory, it might be possible to determine that the error must lie within a bounded region. Optimizing the control would then be reducing the size of this region for a given set of system dynamics and time rate change of the "center".

PID + friction + dynamic compensation had the greatest RMS error due to errors in the model. The dynamic compensation controller attempts to calculate the expected torques at the joints from the movement of the system and is entirely dependent on a model of the system. The dynamic compensator for the 2.165 robot performed well within a narrow region of the total workspace. However, if the robot left this region the errors in the model began to dominate and inappropriate commands were calculated causing the robot to shake, sometimes violently. Slightly changing the mass parameters in the model only caused the useable workspace to decrease.

Conclusions and Applications

Although the original problem was vaguely stated, the system performance was quite good. The system was able to simulate dancing to a wide variety of music and offered a different and insightful look at the use of learning control.

Derivative FFT control is useful for systems that model a natural response. As micro controllers become increasingly more powerful and cheaper they are beginning to be found in an ever widening array of products. Products such as Sony's Robot Dog *ABIO* and Hasbro's *Furbies* attempt to model the natural responses of humans or animals. For these types of systems derivative FFT control can offer efficient but accurate sound recognition and control. In a rudimentary sense, the music controlled 2.165 system was able to recognize and act upon sounds from a single instrument within a complicated musical signal in real-time.

Learning control tends to have very good trajectory tracking for a wide range of motions and different systems configurations, but has the draw back of requiring a closed path and multiple iterations. The results from this system show that a learning controller might be able to be split in a local and global control algorithm. One can imagine a task which involves a very well defined motion in a small volume to be completed at a position that is not known ahead of time. Examples include medical robotic sewing of stitches after surgery or welding a series of seams in a known configuration on a moving

assembly line. The learning controller keeps the error within a desired range in the local region and the “center” is moved along a global path, possibly by an outer control loop. This type of system could be useful in applications where other model based controllers fail due to unknown process parameters either in the robot manipulator itself or from the environment.

References

S. Arimoto, S. Kawamura, and F. Miyazaki, “Better Operation of Robots by Learning” *Journal of Robotic Systems*, 1(2), 123-140, Wiley, 1984.

W. Press, S. Teukolsky, W. Vetterling, B. Flannery, *Numerical Recipes in C : The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1993.