REMOTE DATA ACQUISITION USING LABVIEW

A Design Project Report

Presented to the Engineering Division of the Graduate School
of Cornell University

in Partial fulfillment of the Requirements for the Degree of

Master of Engineering (Electrical)

by

Andre Harrison

Project Advisor: Andy Ruina

Degree Date: August 2006
Abstract

Master of Electrical Engineering Program
Cornell University
Design Project Report

Project Title: Remote Data Acquisition Program using LabView

Author: Andre Harrison

Abstract:

This report presents the design and testing of a LabView program that receives, plots, and saves data over a wireless RS-232 connection. The purpose of the program is to view the various changing parameter values of a walking robot in real-time in order to test and evaluate its walking cycle. The program has been tested for both functionality and speed, which will determine how many parameters can be viewed at one time.
Executive Summary

The purpose of this report is to present the design and implementation of a real-time LabView Data Acquisition program. The program is designed to be used by the Marathon Walking Robot project in order to view real-time plots of various position, velocity, acceleration, control signal, and other miscellaneous parameters in order to debug problems, study the dynamics of the system, and to potentially compare against simulation results. This report is expected to be viewed by future users of the program and those who wish to improve upon and expand its capabilities.

The LabView program was able to successfully receive data from the micro-controller unit. The program was able to plot and save multiple parameters that were received in real-time. The program was able to change what data was being saved, what parameters were plotted and how many graphs were displayed not only at initialization, but while the program was running without losing data at a transmission rate of 1kb/s. The current limitation of the program is that it must have this same functionality at 25kb/s, which it currently does not. Future improvements such as the ones mentioned in this report should may reduce the program loop time so that it can accept higher data rates even, while displaying four graphs at once.
Table of Contents

Introduction

Background

Program Design

Overview
List of Figures
Introduction

The goal of the Marathon Walking Robot project is to develop a bipedal robot able to set new records in walking distance and efficiency. The robots developed by the lab utilize the idea of passive dynamic walking for motion; doing this requires knowing something about the dynamics of the robot in order to understand the robot’s motion. In order to collect this information, while the robot is walking the lab needed a way to view and collect the value of several parameters as they changed over time. In order to do this a Data Acquisition program was made.

Background

Previous Work

In August of 2005 a program was designed by Ko Ihara to plot and save data output over an R2-232 port. It is very useful for debugging code and plotting data. It is a very simple program that works, but due to its simplicity it presents some limitations.

Limitations

- The code on the robot micro-controller was developed to be used with HyperTerminal as an early debugging technique and not specifically for the LabView program. Thus the Data parsing and communication is done using strings. In order to send one data point the numerical value is converted to a string and a string header is attached and then sent to the LabView program, which searches for the header and must convert the string number back into an actual number. This number is then added to an array and plotted. This requires a lot of bytes and reduces how much data can be collected in a given period of time.
• Only two graphs or sets of data can be saved or viewed at any one time.

• In order to change the data being plotted the program must be stopped and the new header string must be typed in. This means the user must know or remember the different header strings for each parameter to be plotted.

**Project Description**

My project serves as a major update to this in order to address the above limitations and to add some more functionality. To increase the amount of data that can be sent per second my program will use a more efficient encoding scheme for communication. Abstract the encoding details so that the user interface is more “user friendly”. At completion my program should be able to receive and save 1-4 (minimum) 8 (maximum) different parameters at 100 data points per second, originally 1 - 16 parameters. Display 1 - 4 (minimum) 6 (maximum) different plots in real-time. Save data simply of the last few seconds of data.

It will also receive data over the RS-232 port with the following port settings. 8 data bits, no parity, 1 stop bit, no flow control and a baud rate of 115K. These are the setting used to program the Innovation First micro-controller the M.C.U. Used by the project group at the start of the project and for most of the year. To remove the need of a long wire to be connected between the robot and computer a bluetooth RS-232 module was purchased to replace the wire. This module supported 115K baud, but late in the year we discovered it could only actually communicate in bursts and so could only transmit a maximum of 25Kbps. This severely cut down how much down how much data could be received and significantly scaled back the project goals.

The purpose of the program is not to only help debug the robot, but to also allow the
robot to be studied and possibly simulated based on the measured parameter values. The saved parameters could also be used to check against a simulated robots expected parameter values as a function of time, if a constructed robot is successfully simulated.

**Program Design**

**User Interface**

The user interface is made up of the following parts:

- Plot area where 1 – 4 plots are displayed based upon the plot view selection menu.
- Graph menu – controls what parameter is plotted on the graph
- Channel select menus - control what parameter(s) are saved or can be plotted.
- Plot view selection – selects how many plots are shown in the plot area.
- Save button and file path – controls the name and location of where the data is saved and when to save the data.
- Reset button – clears all arrays storing the parameter data points.

**High level view**

(high level flow chart of software operation [to be added/created])

**Overview**

Data received over the RS-232 port is encoded into packets and each packet is made up of a header and a data point. The header not only indicates the start of the data point it also tells what parameter the data point is for. Based on the parameters chosen in the front panel channel select menus the headers that correspond to those parameters are searched for in the received
data. When one of the headers is found the corresponding information is decoded into a data point and added to the end of the array of data points for that parameter. The arrays for the parameters are then sent to the graphs(s) to be plotted if that parameter was selected on the graphs menu.

**Encoding scheme**

The micro-controller on the robot transmits a stream of packets through the RS-232 port. Each packet consists of a header byte and five data bytes. In order to distinguish a header byte from an data byte the header is only allowed to take on values from 128 to 255, msb is always a 1, while the data bytes can only have values from 0 to 127, msb is always a 0,. This limits the maximum number of possible parameters that can be used to 128, but this is much higher than the number of independent parameters the robot has. It also means that each data byte only holds 7 of the 16 bits for each data point. Fig. 3 below shows the encoding scheme that is being used. This encoding scheme is preferred due to its simplicity of implementation and the low overhead the scheme requires. *(add stuff related to wireless module)*

Fig. 3 Data encoding diagram. The top 3 bits of the seven available bits in the most significant data byte are not used.

As the figure above shows there is no error detection used in this scheme, there are several reasons for this. The main reason is that error detection would require the micro-
controller to remember old values long enough for an error to be detected in the LabView
program and send a command telling the M.C.U. to resend the data. The amount of memory and
processing time this would require on the M.C.U. were simply impractical. Also, because 100
data points per second are sent for each parameter and the robot does not move extremely
quickly the parameters should not change extremely quickly so a single erroneous data point
should not matter. Finally, the environment the program will be used is expected to have low
data corruption.

**Design choices**

There are many possible programs that I could have chosen to create the Data Acquisition
program task such C, Perl, Matlab, or Java. This version was done in LabView for several
reasons. LabView is a well known program used to interface with external devices. It has many
pre-built modules able to do complex tasks very simply such as opening and closing various I/O
ports, numeric, string searches, and especially real-time graph displays. In most other languages
such C, Perl, Java, and even Matlab generating a GUI would be a project in itself. In the end
LabView was the best possible program to create a relatively nice GUI fairly quickly. The
difficulty that comes with using LabView is that it is a completely graphical language and while
everything is transferable as programs get bigger adding and modifying parts becomes more
difficult. Modifying a existing LabView program is very much like rewiring a circuit schematic
implemented using discrete components.

Time Stamps – *expound later*

**Functions** *could describe in detail*
These describe the functional blocks of the program. Most of these were separate VI's, but were combined with their calling VI's in order to increase the program speed.

- **DAQ_4c_wsavexvi** – Contains the main while for the entire program. Displays and plots 1-4 graphs of data based on the users choice. Also saves data in a file specified by the user.

- **Master ring Select.vi** – Contains all the parameter names that can be saved. By altering this all the four channels select rings are changed at run-time.

- **Case structure** – disables graphs not being used in order to speed up execution time

- **Big parse with sync.vi** – Reads in data, searches for header byte for each selected parameter, decodes the data bytes into a data point and adds it to the array for that parameter.

- **Sync.vi** – Called by big parse w sync.vi. Aligns the parse with the incoming data, formats the incoming data so the 1st byte, of the block of bytes being searched, is the header. Allows the program to wait for 6*n bytes, where is n an integer, to be received before searching for the header types. The optimum number for n is the number of packets sent between time stamps. This means that each search should find one data point per loop. Also searches for the time stamp header in order to iterate to the next set of data points.

- **Parsing block** – Called by Big Parse with sync.vi. Associates a chosen parameter to a header value, resets the array on reset and shifts data points when the time stamp has been received.

- **comm_decoder_faster.vi** – Called by parsing block section. Looks for parameter header and decodes data point into a value and adds that to array for that parameter.
• Channel select and format_vi – Chooses one of the four parameters being saved, based on the front panel selection, and formats the data to be plotted.

Test Results

A simple program was designed by Jason Cortell our Lab manager to test the communication protocol and the program using one of the newly setup micro-controllers in the lab. The program I designed successfully was able to plot and save data. It could plot on multiple graphs, change parameters of a channel change what was being plotted and the number of graphs shown without stopping the program. Unfortunately the program worked on a functional level, but it was unable to run fast enough to handle all the incoming data at the rate it was being sent. At the time this report was written the program is too slow to plot 100 data points per second for four different parameters.

Future Improvements

In order to increase its speed to handle all of the VI's would have to be combined into one. Thus the program would not have to open, run, and close multiple sub VI's every loop, which is what it is doing now and is a very time consuming process. The sub VI's were created in order to make the development process easier to and to allow the overall program to be edited easier. Now that the program is functionally working there is no need for the sub VI's.

The next step in the program was it was working at the desired speed is to add a control to the program to allow commands to be sent to the M.C.U. This is a relatively simple thing to do and would allow the robot to take commands from the user, it could allow variable values, control constants to be changed speeding up the debugging process. The communication
protocol described above could still be used for transmitting data to the micro controller. Once a simple control is added a nice feature would be to get the program to pick and change what data the micro-controller is actually sending out. By telling the M.C.U. what parameters have been selected on the channel select menus the M.C.U. will only send data points for those parameters and by changing what parameters are selected on the channel select menu the M.C.U. will send the new parameter and stop sending the old.

Right now in order to change what header corresponds to a parameter value or to add more header values since all 128 possible values were not programmed in it is necessary to edit every parsing block section and change the case statements. This is a problem that arose when changing from a bunch of sub VI's to one VI. It would be better if there was one master case block that all the other case blocks inherited their cases from.

Conclusion {to be written later}