CTA CHEC Door Motors, Explanation of Coding

Robert Peck

Primary email: rp329@student.le.ac.uk

Secondary email: fusionforever235@gmail.com

This document contains a brief explanation of the way in which the NanoJeasy code of the master and slave programs operates. It also contains brief instructions on how to install NanoJEasy and NanoPro onto a computer such that the computer can then be used to load programs onto the nanotec motor controllers.

Please ensure you have access to a copy of the archive file “CTA door motors guides and nanoJeasy code23092016.zip”.

Installation of NanoPro and NanoJEasy

Begin by visiting http://en.nanotec.com/download/software/ and downloading the zip files “NanoJ_Easy_Setup_1.04.zip” and “Nanopro_1.70.8.0.zip” from, respectively, the “NanoJ Easy V1.04” and “Nanopro 1.70.8.0” download links on that webpage. Do NOT download “NanoJ Easy V2.05”, it is incompatible with the type of controller in use here. Save these into your “My documents” folder. Extract both the archive folders to folders within “my documents”.

To install NanoJEasy open the executable “Setup_104.exe” in the folder created when “NanoJ_Easy_Setup_1.04.zip” was extracted. Click next through the language and licence screens of the installer until you reach the screen asking you to specify the folder to install to. At this point you must specify a folder which you have control of, not a system folder but rather a subfolder of “My documents”, “My pictures”, or other such directories. Select, for example, “My documents” as the folder to install to, then type into the text box after it the name for a new subfolder that does not yet exist. For example have something like C:\Users\Your_user_name\Documents\nanojeasy\ typed into it, at this point you can click the “next” button. On the next screen do notslect a program group, continue through the installer until finished.

To install Nanopro, run the “Setup_1_70_8_0.exe” file found in the folder created when “Nanopro_1.70.8.0.zip” was extracted. Click through the language and licence screens, create a folder for it in a non-system folder (not the usual program files or x86 locations) such as C:\Users\Your_user_name\Documents\nanopro\ . When you reach the screen asking about components leave both of them ticked, the continue clicking “next” until finished.

Connecting the motors to a computer also requires installation of a driver. This should be downloaded by downloading the “ZK-RS485-USB_SMCP33_EVA.zip” file from the “ZK-RS485-USB and SMCP33-EVA” link on http://en.nanotec.com/download/drivers/ . This zip will should be extracted but cannot be used until the motor controller is connected to the computer.

With both NanoJEasy and NanoPro installed the motor controller should now be attached to the lead necessary to connect it to the computer, the other end of this lead with the USB should be attached to the computer. You should wait several minutes to ensure the computer has been able to detect the connection of the motor controller. Now go into “control panel” then “device manager” and find “CP2012 USB to UART bridge controller” listed under “other devices”.

Find the folder created when you extracted “ZK-RS485-USB_SMCP33_EVA.zip”, within it find and run “CP210xVCPInstaller_x64.exe” or “CP210xVCPInstaller_x86.exe” if you are using a computer with a 32 bit operating system. Click through the steps in this installer. When this is done go back to device manager, within a few minutes “CP2012 USB to UART bridge controller” should have disappeared from “other devices” and been replaced by “Silicon labs CP210x USB to UART bridge” under the “ports” section of the device manager list. It should have next to it the number of a COM port written in brackets, for example (COM5). Note down the number of this port you will need it again later.

Using NanoPro and Transferring Programs to the Motor Controller

1.Connect Nanotec motor controller to computer by USB cable.

2.Open NanoPro and set the correct COM port for the controller, this is the COM port which you noted down after installing the driver, under the "Communication" tab in NanoPro, then use the "Read configuration from drive" button in the top right hand corner to read configuration. If at this point a message appears about updating the firmware to the 11-12-2014 version then please do so by using the “Firmware change→update to latest firmware” option under “System” at the top of the screen.

3.Compile the relevant code in NanoJEasy by opening the .java file with NanoJEasy and clicking the "white gear wheel" icon on the taskbar at the top. It is important that for code to compile it must have the same name for its overall class(Line 4 in Master.java), as it does for the overall file. At this point it is also important that the characters are UTF-8 not ANSI or ASCII.

4.Close NanoJEasy.

5.In NanoPro make sure that the data in the "motor settings" tab is correct, if it is not then edit any fields that are incorrect. They should read:

Step mode=Half step

Drive Step angle=1.8^0 /step

Phase current= 50%, Current: 0.9A, Peak Current:0.9A

Phase current during idleness=25%, Current:0.45A, Peak Current:0.45A

Reverse clearance=0 steps

Send state byte automatically upon end of record=unticked

Rotary encoder type= Incremental with index

Rotary encoder resolution=greyed out

Reverse encoder direction=unticked

Controller type=SMCI12 RS485

Motor type=ST4118

Motor name=ST4118L1804

Wiring=greyed out

Gear Factor numerator=1

Gear Factor denominator=1

Ramp=4230.95

Speedmode control=velocity loop

6.In NanoPro go into the "Statusdisplay" tab.

7.Tick the "Autostart" box.

8.Use the "Transfer program" button to find the prg file created by compiling the relevant code.

9.Select that file and click the "Open" button in the file browser windows.

10.Once the transferring of the program is complete click the large "Save configuration to drive" button at the top.

11.Disconnect the USB cable from the computer, turn off the power to the motor controller, then turn the power to the motor controller back on a few seconds later.

12.Now the program should begin executing and looping.

The procedure described above can be repeated for the other motor by disconnecting the motor that has just been programmed and connecting the other. The slave motor needs the slave program written to it and the master motor requires the master program.

A Short introduction to NanoJEasy Coding

NanoJEasy is a java based language, with some special additions explained it should therefore be quite understandable to anyone fluent in java coding. Throughout it uses the same coding structure as java and all the usual java functions such as if and while loops can be used. The syntax

is the same as in java and like in java variables must be defined at the start and all lines require semicolons to end them. However rather than the usual print functions and functions for reading from files that may be used in java code NanoJEasy instead has certain functions added in that allow it to read the input circuits and control the motors and output channels. These are defined in detail in section 2.5 of the pdf at http://us.nanotec.com/fileadmin/files/Handbuecher/Programmierung/Programming_Manual_V2.7.pdf .Within the programs Master.java and Slave.java only a few of these available functions are used, specifically:

io.SetInputXSelection(0); sets input X to be user defined rather than be used to start a movement profile or other such default things.

io.SetOutputXSelection(0); sets output X to be user defined rather than light up when power is on or other such default things.

util.Sleep(integer); pauses the program for an integer number of milliseconds.

drive.StopDrive(0); brings the motor to an immediate stop.

drive.SetMode(5); sets the motor to speed mode where it will accelerate up to a defined speed upon starting then maintain this speed until told to stop.

drive.SetMode(1); sets the motor to relative position mode where it will move by a set number of steps in a specified direction and stop after completing these steps.

drive.SetMaxSpeed(integer); sets the speed the motor will accelerate up to then move at in either speed or relative position mode.

drive.SetAcceleration(integer); sets the rate of acceleration.

drive.SetRampType(0); sets the mode of acceleration used in this program, other integers give other modes.

drive.SetTargetPos(CloseDist); in relative position mode this sets the number of steps to make before stopping.

drive.SetDirection(integer);this controls which direction the motor will turn in, the integer can be 0 or 1.

drive.StartDrive( ); this starts the motor moving with whatever motor settings are defined when this line is reached.

io.SetDigitalOutput(integer); this sets the outputs but works in a rather unusual way. In practice it is desired to have some outputs active and others inactive. Due to the way the circuitry works for another controller to measure a high voltage as an input the output must be set to 0 within the program of the controller outputting the signal. Therefore it is suggested that you find a binary to integer table and find the integer value which causes the value 0 to appear on all the channels you would like to be set to 1. The use this integer. For example with three outputs if you wanted to set

the first and third to 1 and the second to 0 then you would use the integer 5 2 because in binary 2 is 010, a zero in the first and third columns and a 1 in the second column. The first column is taken to mean the 1s column, the second column the 2s column, the third column the 4s column.

io.GetDigitalInput( ); this reads the values on the inputs. It produces an integer number which when written in binary represents the inputs on the various channels. To work out what an integer value from this function corresponds to in physical terms consider that integer written out in binary, the first input channel controls whether the 1s column is a 1 or 0, the second channel controls whether the 2s column is a 1 or 0, the third input controls whether the 4s column is a 1 or 0 and so on. Any combination of inputs therefore has a unique numerical value. For the program to determine which inputs currently have value 1 (high voltage applied) or 0 (low voltage applied) various mathematical operations must be used, these include dividing by powers of 2 and considering the remainders or in some cases comparing with lists of integers known to have certain values in certain columns when written in binary.

Editing the Master or Slave Programs

It is likely that simple edits to some of the numerical values used in the programs may be needed when placing the programs on motors in a real shutter controlling frame rather than the simple test rig used during programming. The directions have already been set such that they should still be correct on the finished structure but the speed of moving the doors and the number of steps made in the final stages of closing may require alteration. Fortunately these numbers have been defined in the program as variables at the start. They can therefore be changed throughout the whole program simply by changing the value at the start.

motorSpeed controls the speed at which the motors rotate except during the first few moments of travel when they are accelerating up to speed. It is set to 500 in the program but may require a larger value to make the doors close faster or a smaller one to slow the door motion and reduce stress on the motors. It is found on line 105 in the master program and line 99 in the slave program. The value should be the same in both programs otherwise the closing will not be synchronised.

CloseDist defined on line 91 in the slave program and line 108 in the master this value, currently set to 80, controls the number of steps taken at the end of the closing sequence. On the finished structure it will be almost certain to need its value changing. The value should be set by a mixture of calculation using the desired angle and the known step size and also by trial and error, starting with a lower value and gradually testing larger and larger values. If this value is too high it could cause the doors to drive through the closed sensors and potentially damage the detectors of the camera below them.

Other numerical values within the code should not need editing,:

Changing the values used in the if loops would cause the controller to incorrectly interpret input signals from the sensors, the other controller and the camera safety board.

Changing the delays used in the many util.Sleep() parts of the programs could result in the motor being unable to synchronise and unable to check whether an input is a true signal or a short random blip of noise.

Changing motor settings such as the acceleration or ramping may make the motion jerky and violent.

Explanation of the lines within Master.java and Slave.java

For detailed description of the code's functionality and means of operation the comments within the code should be viewed, this can be done by opening the Master.java and Slave.java files in NanoJEasy. You can also open them in a text editor such as notepad but you should NOT attempt to edit them except within NanoJEasy, the file names must not be modified unless the name of the class is changed to reflect them. Please be warned that the files Master12092016.java and Slave12092016.java included in older versions of this archive are of no use, as the changed file name means that upon compiling they become too large.

The correct Master and Slave files have modification dates of the 23rd of September 2016.

When compiled the code as it stands has the sha256 hashes of:

for the Master.prg file:

Sha256: 0F658364C191A0378853C8DDE37433C5C8467DAD42ED41E533B0455E2C1FC1E8

for the Slave.prg file:

Sha256: 0132452298DFE6B108AA349081641C1FF8FDEE4F4C983E4136CCFB454E7824D6

Before compiling he code's hashes are

for the Master.java file:

Sha256: 8779840A4AB287809FBEDB71DB62201C41273652B634BC924A4A3EBE7ADF54B2

Size: 42817 bytes

for the Slave.java file:

Sha256: BE238999CD5CA07BB597EFB10AE6F04E4C2138C105607FA0340A7357BD9C4145

Size: 34902 bytes

These should allow you to verify that the code is the correct version.