Lab 8: Programming with B&R Automation Studio

Introduction

Throughout this course, we've been working with transfer functions in the s domain representing differential equations in the continuous-time domain. For instance, in Lab 3 we learned about the PID controller defined as $C_{PID}(s) = k_p + \frac{k_i}{s} + k_d s$. Later in Lab 7 we approximated the true derivative term with a proper transfer function $C_{vf}(s) = \frac{500s}{s+500}$ for velocity feedback. In this lab, we want to consider what is actually happening when we create and execute these s domain transfer functions in Simulink.

Each time a controller is constructed in Simulink, it is built and downloaded to the B&R X20 Controller. The B&R X20 Controller, like any other processing unit, is a digital computer (there are analog computers out there but none to the scale which can compute complex control algorithms). This means the controller is only executing your control law at fixed intervals in time, i.e. the sample period. This sample period has been one millisecond for the whole semester. So how does the controller execute the differential equation $u_{PID}(t) = k_p e(t) + k_i \int e(t)dt + k_d \dot{e}(t)$?

The short answer is, it doesn't. Behind the scenes when you build your control algorithm in Simulink, all continuous transfer functions are converted to discrete transfer functions according to the sample period. A discrete-time transfer function is a transfer function in the z domain, which you may remember from your ME 360 course. This course is focused on continuous-time systems, so we will not go into many details of discrete time systems.

Instead of relying on Simulink to build our discrete transfer functions, this lab presents a procedure for approximating the discrete transfer functions and directly programming the discrete-time controller dynamics in B&R Automation Studio.

Objectives

- Use the Tustin approximation (trapezoidal rule) to approximate the discrete time transfer function(s) of your control algorithm.
- Complete an introduction to B&R Automation Studio by running default code to test open-loop control of the XY Stage and reference trajectory generation.
- Use your approximated discrete transfer functions to program the PID controller in Structured Text.

Experiment

Discrete Time Transfer Functions:

A discrete-time controller will not have the same transfer function as a continuous-time controller. But, a continuous-time controller can be translated into a discrete-time controller, i.e. its Discrete Equivalent. For this lab, use the Tustin approximation:

$$s \leftrightarrow \frac{2}{T}\frac{z-1}{z+1}$$

T is the sample period, e.g. one millisecond for the B&R X20 Controller. Thus, you can substitute s in the continuous-time transfer function with this Discrete Equivalent. For example, the discrete equivalent of the integrator using this approximation is as follows.

$$\frac{Y(s)}{X(s)} = \frac{1}{s} \leftrightarrow \frac{Y(z)}{X(z)} = \frac{T}{2}\frac{z+1}{z-1}$$

Instead of representing a continuous-time differential equation, the discrete transfer function now represents a discrete-time Difference Equation:

$$y[k+1] - y[k] = \frac{T}{2}(x[k+1] + x[k])$$

This is the Tustin approximated discrete equivalent of $y(t) = \int x(t)dt$. It can also be represented as:

$$y[k] = y[k-1] + \frac{T}{2}(x[k] + x[k-1])$$

Taking a closer look, you should notice this is simply the trapezoidal rule for integration, hence why the Tustin approximation is often called the trapezoidal rule. For small sample periods T (one millisecond), this approximation is quite accurate.

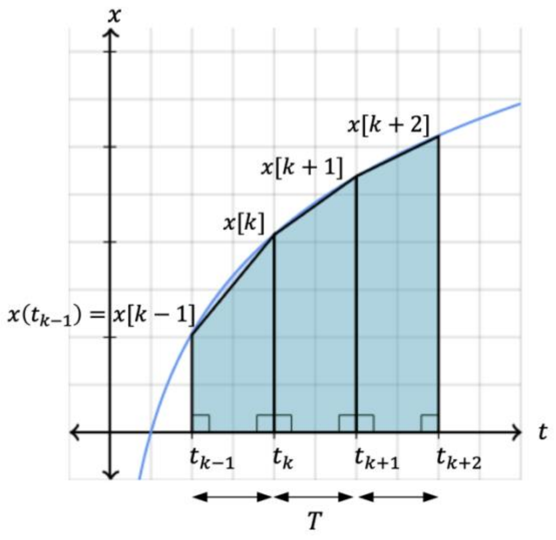


Figure 1: Trapezoidal Approximation for Integration

B&R Automation Studio:

Automation Studio is the integrated development environment (IDE) for all B&R controllers. Though we have not directly used Automation Studio yet, the Simulink models that are built are part of a bigger Automation Studio project that is all together downloaded onto the controller (this is done in the background when you see the code generation progress pop-up after click *Build*).

Automation Studio uses several text-based languages to program with including C and C++, but the most commonly used language is Structured Text. Structured Text is very similar to languages like C but has one major difference in that variables assignments are performed with a " := " instead of just "=". You will see examples of this in the default code as well as other syntax rules. Your job will be to write the portion of the code corresponding to the closed-loop control algorithm we completed last lab, PI control with velocity feedback, in Structured Text.

Procedure

Getting Started:

1. Turn on the B&R X20 Controller by flipping the breaker switch.
2. Run Automation Studio by searching 'B&R Automation Studio 4.3 English' or finding the icon on the desktop.
3. Acknowledge any licensing pop-ups by clicking *OK*.
4. Choose 'AS_HydroLabs' from the recent projects list. If you don't see any recent projects, you can located the project .apj file in 'C:\BRprojects\bnr_repo\AS_HydroLabs\AS_HydroLabs.apj'.
5. The left pane shows the *Logical View*. Expand the *Student* folder and then further expand *ProgLab8*.

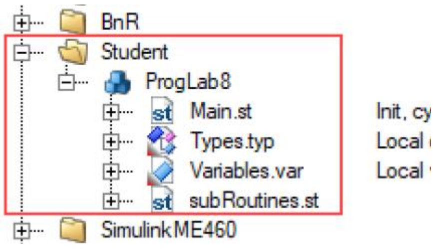
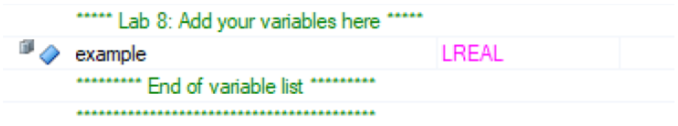


Figure 2: Logical View, Student Folder

6. Open *Main.st* by double-clicking the file. This is the Structured Text program you will be working with. Notice all the code located within *_Cyclic*. This code is executed every one millisecond. You control algorithm will be placed here.
7. Look for the *IF* and *ELSE* conditions in *Main.st*. This is how we will switch between open-loop and closed-loop control based on the *flagOpenLoop* variable. The open-loop control algorithm is given, where should your closed-loop control algorithm go?
8. Open *Variables.var* by double-clicking the file. All the variables used by *Main.st* are declared and initialized in this variable files (aside from a few global variables). Many variables have already been defined for you, and you should not change these. You will use several of these variables, such as *kp_x* and *ki_x*. You will also create your own variables to use. Find the variable *example*. This is where you should create your new variables. To practice, right-click the variable *example* and choose 'Add Variable', create the variable *error_x* and make the type an *LREAL* (double/64-bit floating type). The initial value is zero when the *value* entry is left blank.



```
***** Lab 8: Add your variables here *****
example                                LREAL
********* End of variable list *********
******************************************
```

Figure 3: Variables.var example Variable

9. Additionally, go ahead and set the PID gain values for both axes based on your final tuned gains from last lab. Assign this in the values column to the PID gain variables already created for you.
10. *getReferences* is a sub-routine to generate reference signals *ref_x* and *ref_y*. Use these variables as your reference values.
11. *measurePositions* is a sub-routine to measure the current position of each axis and assigns the values to *pos_x* and *pos_y*. Use these values for your current position.

12. *copyToGlobals* is a sub-routine that simply records values to be uploading to Simulink in order to view with the scope blocks.

Uploading to Simulink Scopes:

1. Copy the file ‘Lab8_ScopeUpload.slx’ from ‘N:/HydraulicsLab/ME460’ to your folder on the C:\ drive: ‘C:\ME460_SPXX\ABX\Lab8’.
2. Run MATLAB and change your workspace to your folder on the C:\ drive. Start Simulink by opening the .slx file.
3. There is nothing you have to do with this Simulink model other than *Build* once. *Build* the Simulink model and once completed, *Connect to Target*.
4. You should see on the encoder scopes that the square wave reference signals appear in blue.
5. As you develop your code in Automation Studio, you will only have to *Connect* and *Disconnect* this Simulink model, you only had to *Build* once.

Watch Window:

1. You may leave the Simulink model connected. Go back to Automation Studio and right-click on *ProgLab8*, choose ‘Open’ then ‘Watch’. You should see a small handful of more important variables from the default Structured Text code. This Watch window allows you to read and write to these variables while the controller is running.

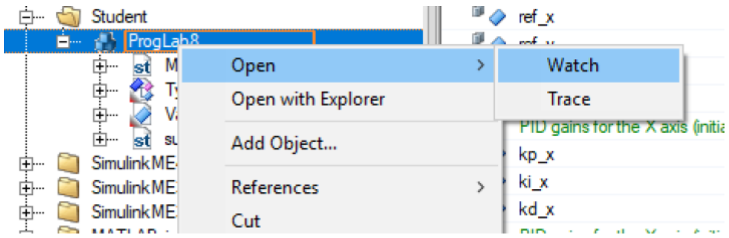


Figure 4: Opening the Watch Window

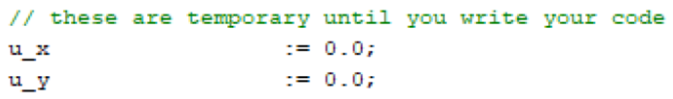
2. Position the windows on your monitor so that you can see the Simulink model, both encoder scopes, and the Automation Studio Watch window.
3. Find the variables *flagSquareReference*. Change the value to *FALSE* by entering *0* into the values column of the watch window. What happens in your Simulink scopes?
4. Change *flagOpenLoop* to *TRUE*. Change the value of *uOL_x* or *uOL_y* to *0.7*. In your Simulink model make sure to click the *Run* pushbutton. Hold down on the dead-mans-switch with your thumb. What happens?
5. Zero the axes using the open-loop control and then click the *Halt* pushbutton in Simulink to reset the encoder values. Switch back to closed-loop control. Hold down the dead-mans-switch and click the *Run* pushbutton in Simulink. What happens?

Program the Closed-Loop Algorithm:

1. Create the *LREAL* variables *error_x* and *error_y* (or name them something similar).
2. Write a statement to assign your error variables in the closed-loop section of *main.st* (you must use the given reference and position variables).
3. Create the *LREAL* variables *proportional_x*, *integral_x*, and *derivative_x*, or something similar. Do the same for the y axis. You will need to assign these variables according to the control law. Then, you should have the statement:

```
u_x := proportional_x + integral_x + derivative_x;
```

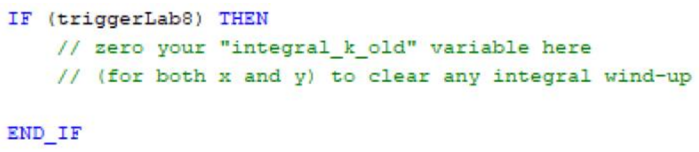
This should replace the existing lines of code:



```
// these are temporary until you write your code
u_x                     := 0.0;
u_y                     := 0.0;
```

Figure 5: Assign Closed-Loop Control Values

4. You must use *u_x* and *u_y* to assign your final control value.
5. Use variables like *error_x_old* to represent values like *e[k-1]*. What do you have to do at the very end of your closed-loop algorithm to update "old" values for the next sample?
6. The rest is up to you, check with your TA to see if your written program is on the right track.
7. One important note: as you get further along in your development you will notice the *if* statement on the Boolean variable *triggerLab8*. This *if* statement should be used to zero any integral windup. Ask your TA how you can perform anti-integral windup correctly.



```
IF (triggerLab8) THEN
    // zero your "integral_k_old" variable here
    // (for both x and y) to clear any integral wind-up

END_IF
```

Figure 6: Zero Integral Windup

Run Your Program:

1. To compile and download your newly written program in Automation Studio, simply click the *Transfer* icon in the top toolbar.

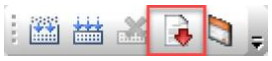
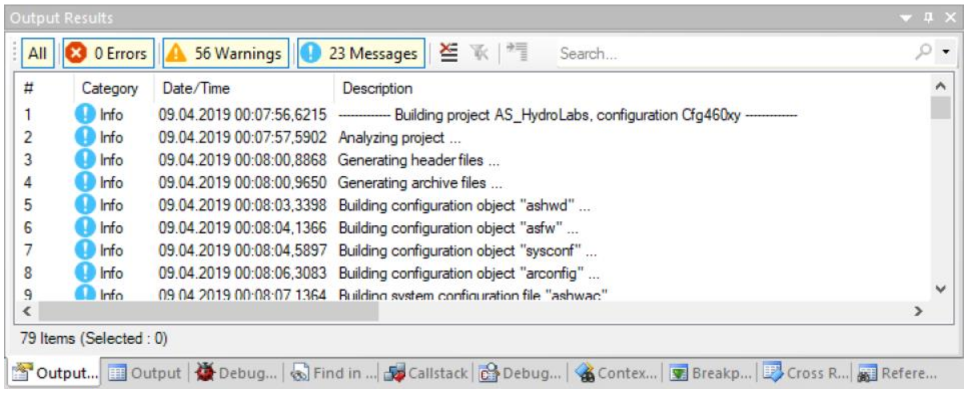


Figure 7: Transfer Icon

2. If there are any errors, look through the *Outputs* dialog at the bottom to debug them. Ignore any warnings. If there are no errors, you should get a pop-up that will now allow you to *Transfer*, click 'Transfer'.



```
#   Category  Date/Time                 Description
1   Info      09.04.2019 00:07:56,6215  ----------- Building project AS_HydroLabs, configuration Cfg460xy -----------
2   Info      09.04.2019 00:07:57,5902  Analyzing project ...
3   Info      09.04.2019 00:08:00,8868  Generating header files ...
4   Info      09.04.2019 00:08:00,9650  Generating archive files ...
5   Info      09.04.2019 00:08:03,3398  Building configuration object "ashwd" ...
6   Info      09.04.2019 00:08:04,1366  Building configuration object "asfw" ...
7   Info      09.04.2019 00:08:04,5897  Building configuration object "sysconf" ...
8   Info      09.04.2019 00:08:06,3083  Building configuration object "arconfig" ...
9   Info      09.04.2019 00:08:07,1364  Building system configuration file "ashwac"
79 Items (Selected : 0)
```

Figure 8: Outputs Results with Debug Messages

3. Re-connect your Simulink model if it needed.
4. Use open-loop control to align each axis in the center.
5. Use the *Halt* pushbutton to zero the encoders when centered.

6. While halted, use your thumb to hold down the dead-mans-switch and then you may click the *Run* pushbutton.
7. Test out your closed-loop algorithm on the circle trajectory. You may tune your gains in the watch window if necessary. Make use of the *disable_x* and *disable_y* flags to only perform closed-loop control on one of the axes.

Report Questions

1. How did your previously tuned PID gains perform with the Tustin approximation when programmed in Automation Studio? What is the difference in performance between the Automation Studio program and the previous lab's Simulink program?
2. Are there any advantages to using text-based languages like Structured Text to program control algorithms compared to graphic programming as in Simulink?
3. What are some other Discrete Equivalent approximations that could be used to translate continuous-time transfer functions?