DAQ Simulator

Lars Rikard Rådstoga | 223786

2022-01-18

# Introduction

This document contains answers to the assignment “#1 – C#” given as a compulsory assignment in the course “Software Engineering” at the University of South-Eastern Norway – spring 2022.

# Assignment Setup

This chapter contains task solutions related to the “Assignment setup” chapter in the assignment text.

## SCE1306DaqCoding Screen Shot

Screenshot in Figure 2‑1: Figure 2‑1 shows task parameters which are to be used in this assignment.

Graphical user interface, text, application

Description automatically generated

Figure 2‑1: Generated task parameters.

## Initializing Git

A Git repository was initialized in an empty project folder as seen in Figure 2‑2. First-time Git setup was unnecessary as it had already been done.

Text

Description automatically generated

Figure 2‑2 Running «git init» in a project folder.

## Creating a Windows Forms App

A template was used to create the windows form application in Visual Studio 2019 using the .net5.0 SDK.

# DAQ Simulation Application

A replica user interface as the one given in the assignment was created, it can be seen in Figure 3‑1.

Graphical user interface, application

Description automatically generated

Figure 3‑1 The graphical user interface.

All project files were added to the git repository and added to the next commit with the “git add” command, seen in Figure 3‑2.



Figure 3‑2 All files in project added to the next commit.

The files were then committed with a message, seen in Figure 3‑3. A Git commit is basically confirming a version.

Text

Description automatically generated

Figure 3‑3 Initial commit.

## Sampling Sensor Device Values

A class to use for analog and digital sensor objects was created, see Appendix A: SensorDevice.cs.

Additionally, code for sampling sensor data was created, see Appendix B: Simulator.cs. This class creates objects of the sensors specified in the task generator, lists of datasets generated by the sensors, and a method to iterate all sensors and generate a new set of sensor data. It also contains the MA low pass filter.

Code supporting the sampling button is found in Appendix C: MainForm.cs. This button starts a continuous thread sampling data at the specified intervals by calling the generator method in the Simulator class.

The git repository was finally updated with the changes, as seen in Figure 3‑4.

Text

Description automatically generated

Figure 3‑4 A new round of git add and commit.

When sampling data the rate of sampling should be 10 to 20 times the frequency of the system that is being measured. If the sampling frequency is too slow then a terribly wrong representation of that system will be produced. This is called Aliasing as seen in Figure 3‑5.

Text

Description automatically generated

Figure 3‑5 Badly sampled signal [1].

## Help About Information

A message describing the program was added to a message box that appears when the “help” menu button is pressed, as seen in Figure 3‑6.

Application, timeline

Description automatically generated

Figure 3‑6 A helpful message.

## Logging Sensor Device Values

Code supporting the logging features were implemented and are found in Appendix C: MainForm.cs. The button triggers the program to write newly sampled data to a csv file, it also appends to the file if values are already stored. An example of the CSV can be seen in Appendix D: CSV file and the UI also shows current sensor values as seen in Figure 3‑7.

There are many ways to implement simulators like this one, creating a helper class to generate and store “raw” sensor measurements and apply a filter to them can be a nice way to connect the sampler and the logger. Collections of filtered data can then be easily available to a logger class. It is also then possible to create ways to empty these collections when they are logged. Depending on how much you trust your systems you can create asynchronous methods that waits for the logger to report a successful save before the data is deleted.

Graphical user interface, text, application

Description automatically generated

Figure 3‑7 The UI in action.

The Git repository was updated with the new changes. See Figure 3‑8.

Text

Description automatically generated

Figure 3‑8 Final Git commit.

## Documentation

The Git log can be seen in Figure 3‑9.

Text

Description automatically generated

Figure 3‑9 Git log.

# Summary

Windows forms were used with object oriented programming in C# to create a sensor sampling and logging simulator. Git was used to save to a local repository.

# References

|  |  |
| --- | --- |
| [1] | "List of Bluetooth profiles," [Online]. Available: https://en.wikipedia.org/wiki/List\_of\_Bluetooth\_profiles#Audio/Video\_Remote\_Control\_Profile\_(AVRCP). [Accessed 29 01 2022]. |
| [2] | "Master/slave (technology)," 23 January 2022. [Online]. Available: https://en.wikipedia.org/wiki/Master/slave\_(technology). [Accessed 28 January 2022]. |
| [3] | "Understand TCP/IP addressing and subnetting basics," Microsoft, 23 09 2021. [Online]. Available: https://docs.microsoft.com/en-us/troubleshoot/windows-client/networking/tcpip-addressing-and-subnetting. [Accessed 28 01 2022]. |

# Appendices

Appendix A: SensorDevice.cs

Appendix B: Simulator.cs

Appendix C: MainForm.cs

Appendix D: CSV file

# Appendix A: SensorDevice.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DAQSimulator

{

/// <summary>

/// Simple analog sensor class made out of properties.

/// Initialize with a sensorId.

/// use the dVal getter to get random double value between -5 and 5.

/// </summary>

class AnalogSensorDevice

{

public Random rSenVal = new Random();

public int sId { get; set; }

public double dVal { get { return rSenVal.NextDouble()\*10-5; } }

}

/// <summary>

/// Simple digital sensor class made out of properties.

/// Initialize with a sensorId.

/// Use the iVal getter to get random integer values between 0 and 1.

/// </summary>

class DigitalSensorDevice

{

public Random rSenVal = new Random();

public int sId { get; set; }

public int iVal { get { return rSenVal.Next(0,2); } }

}

}

# Appendix B: Simulator.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

namespace DAQSimulator

{

class Simulator

{

//Amount of sensors given by assignment

const int analogSensorDevices = 7;

const int digitalSensorDevices = 3;

//Amount of data pairs to use for the MA LP filter

const int movingAverageIterations = 5;

//Arrays of sensors

AnalogSensorDevice[] analogSensors = new AnalogSensorDevice[analogSensorDevices];

DigitalSensorDevice[] digitalSensors = new DigitalSensorDevice[digitalSensorDevices];

//List of one list of datasets per iteration:

//every inner list contains a dataset with all sensor values for that index of outer list.

//!Could be improved to only hold a finite amount of data here.

List<List<double>> analogDatasets = new List<List<double>>();

List<List<int>> digitalDatasets = new List<List<int>>();

//Properties

public List<double> analogData { get { return analogDatasets.Last(); } }

public List<int> digitalData { get { return digitalDatasets.Last(); } }

public Simulator()

{

for (int i = 0; i < analogSensorDevices; i++)

{

analogSensors[i] = new AnalogSensorDevice { sId = i };

}

for (int i = 0; i < digitalSensorDevices; i++)

{

digitalSensors[i] = new DigitalSensorDevice { sId = i };

}

}

/// <summary>

/// Simulator iterates all sensors each time step.

/// Generates a value for them and filters them if there is data to allow it.

/// </summary>

public void Sim()

{

List<double> analogValues = new List<double>();

List<int> digitalValues = new List<int>();

//for each analog sensor

for (int i = 0; i < analogSensorDevices; i++)

{

//New sensor value

double newSensorValue = analogSensors[i].dVal;

int n = 1;

//Filter when there is previous data

for (int j = 0; j < analogDatasets.Count && j < movingAverageIterations; j++)

{

//analogDatasets[dataSet][sensor]

newSensorValue += analogDatasets[j][i];

n++;

}

newSensorValue /= n;

analogValues.Add(newSensorValue);

}

//for each digital sensor

for (int i = 0; i < digitalSensorDevices; i++)

{

digitalValues.Add(digitalSensors[i].iVal);

}

//Add new values to the datasets

analogDatasets.Add(analogValues);

digitalDatasets.Add(digitalValues);

}

}

}

# Appendix C: MainForm.cs

using System;

using System.Collections.Generic;

using System.Windows.Forms;

using System.Threading;

using System.IO;

using System.Text;

namespace DAQSimulator

{

public partial class MainForm : Form

{

TimeSpan samplingTime = new TimeSpan(0, 0, 0, 2, 100);

TimeSpan loggingTime = new TimeSpan(0, 0, 0, 47, 0);

DateTime nextSample = DateTime.Now;

DateTime nextLog = DateTime.Now;

List<List<double>> analogSamples = new List<List<double>>();

List<List<int>> digitalSamples = new List<List<int>>();

List<DateTime> sampleTimes = new List<DateTime>();

private static Thread sampler;

public MainForm()

{

InitializeComponent();

txtSampling.Text = nextSample.TimeOfDay.ToString();

txtLogging.Text = nextLog.TimeOfDay.ToString();

sampler = new Thread(Simulator);

sampler.Name = "SimulatorThread";

}

/// <summary>

/// Updates the text box showing currently simulated sensor values.

/// </summary>

/// <param name="value"></param>

public void UpdateSensorTextBox(string value)

{

if (InvokeRequired)

{

this.Invoke(new Action<string>(UpdateSensorTextBox), new object[] { value });

return;

}

txtSensorValues.Text = value;

}

/// <summary>

/// Updates the text box indicating the next sampling time.

/// </summary>

/// <param name="value"></param>

public void UpdateNextSamplingText(string value)

{

if (InvokeRequired)

{

this.Invoke(new Action<string>(UpdateNextSamplingText), new object[] { value });

return;

}

txtSampling.Text = value;

}

void Simulator()

{

Simulator sim = new Simulator();

nextSample = DateTime.Now + samplingTime;

while (true)

{

sim.Sim();

string sensorValueText = "";

List<double> newAnalogSamples = new List<double>();

List<int> newDigitalSamples = new List<int>();

for (int i = 0; i < sim.analogData.Count; i++)

{

sensorValueText += $"Analog Sensor {i + 1}: {Math.Round(sim.analogData[i], 3).ToString()}{Environment.NewLine}";

newAnalogSamples.Add(sim.analogData[i]);

}

for (int i = 0; i < sim.digitalData.Count; i++)

{

sensorValueText += $"Digital Sensor {i + 1}: {sim.digitalData[i]}{Environment.NewLine}";

newDigitalSamples.Add(sim.digitalData[i]);

}

analogSamples.Add(newAnalogSamples);

digitalSamples.Add(newDigitalSamples);

sampleTimes.Add(nextSample);

UpdateSensorTextBox(sensorValueText);

nextSample += samplingTime;

UpdateNextSamplingText(nextSample.TimeOfDay.ToString());

Thread.Sleep(samplingTime);

}

}

private void SensorValues\_Enter(object sender, EventArgs e)

{

}

private void SampleData(object sender, EventArgs e)

{

if (!sampler.IsAlive)

{

sampler.Start();

}

else

{

//Not sure how to pause and resume a thread...

//sampler.Join();

}

}

private void btnLogging\_Click(object sender, EventArgs e)

{

if(nextLog < DateTime.Now + loggingTime)

{

nextLog = DateTime.Now + loggingTime;

txtLogging.Text = nextLog.TimeOfDay.ToString();

var csv = new StringBuilder();

csv.AppendLine("Time,Analog\_1,Analog\_2,Analog\_3,Analog\_4,Analog\_5,Analog\_6,Analog\_7,Digital\_1,Digital\_2,Digital\_3");

for (int i = 0; i < sampleTimes.Count; i++)

{

string newLine = sampleTimes[i] + ",";

for (int j = 0; j < analogSamples[i].Count; j++)

{

newLine += analogSamples[i][j]+",";

}

for (int j = 0; j < digitalSamples[i].Count; j++)

{

if(j == digitalSamples[i].Count - 1)

{

newLine += digitalSamples[i][j];

}

else

{

newLine += digitalSamples[i][j] + ",";

}

}

csv.AppendLine(newLine);

}

File.AppendAllText("SensorData.csv", csv.ToString());

}

}

private void helpToolStripMenuItem\_Click(object sender, EventArgs e)

{

// Make the messagebox

MessageBox.Show($"The program starts generating sensor values when the SAMPLE button is clicked. {Environment.NewLine}" +

$"When the LOG TO FILE button is pressed the newly sampled data is added to a file if the log time cooldown period is completed.",

"Program Description", MessageBoxButtons.OK);

}

}

}

# Appendix D: CSV file

Time,Analog\_1,Analog\_2,Analog\_3,Analog\_4,Analog\_5,Analog\_6,Analog\_7,Digital\_1,Digital\_2,Digital\_3

06.02.2022 23:24:26,-4,7503737498775,-4,969343347460657,-1,024821021605665,0,3626657348883642,-1,3056869135730373,2,5024819781549654,4,901963192923908,1,0,0

06.02.2022 23:24:28,-3,8158641773349906,-1,4546220919371686,-2,7999919922091028,2,49863231670979,-2,812033224763364,-0,3809341161423059,3,610334188495918,0,0,1

06.02.2022 23:24:31,-2,0510549231980564,-1,2606399830402684,-2,8436999106051863,1,741406120549921,-2,262583935600356,1,2909996725111266,2,057304422087333,1,0,1

06.02.2022 23:24:33,-2,431931158806476,-2,2989101326708883,-2,4435760138759277,0,6252133682332373,-0,6438186717268474,0,6874322615039683,2,1197322280858946,1,1,1

06.02.2022 23:24:35,-2,6783741707191995,-1,7422199629133348,-1,0829879572070145,0,3271627176989936,-1,4306113927612443,1,3698958453582115,3,0691319289163053,0,1,0

06.02.2022 23:24:37,-2,3465940625623776,-1,7236290188616277,-0,9689510088890251,0,7773352904605377,-1,667909553663142,0,3758186876661229,2,095633794054839,0,1,0

06.02.2022 23:24:39,-2,260432841455145,-1,578042289744151,-2,3912605885686014,1,098028043718711,-1,187352916390644,1,2681694486806336,2,961001148972505,1,1,1

06.02.2022 23:24:41,-2,418190353201481,-1,2216799827548737,-2,177069240021707,0,6311202383589867,-0,5834119526747358,1,1958363771108769,1,8870742966981637,0,1,0