Programmering C eksamen:

Swagway Debugger

af
Mathias Dannesbo

10. maj 2012

Resumé

This journal is a walk-through of the source code of Swagway Debugger, a Windows application for debugging and tuning Kalman-filters, on balancing robots and Segway clones. It is made as a part of the Swagway project.

1 Indledning

Formålet med opgaven er at lave et stykke software der kan bruges sammen med forfatterens eksamenprojekt i Teknikfag A: El. Formålet med teknikfags projektet er at bygge en Segway klon, en motoriseret selvbalancrende tohjulet transportenhed. Navnet på teknikfag A-projektet er Swagway. Under udviklingen af Swagway blev der behov for et stykke software til at teste et filter.[1]

Kort beskrivelse af Swagway funktion

For at Swagway kan holde sig lodret skal den kende vinklen den er fra lodret. Det udregner den ved at læse data fra to sensorer: et accelerometer og et gyroskop.

Et gyroskop kan måle vinkelhastigheder. Det vil sige ændringer i vinklen. Hvis denne vinkelhastighed integreres over tid finder man vinklen gyroskopet har flyttet sig. Problemet med at integrer gyroskopdata er, at pga. måleusikkerheder vil det målte nulpunkt drive væk fra det fysiske nulpunkt.

Et accelerometer kan måle accelerationer. Man kan med hjælp fra tangens og data fra to akser udregne den vinklen accelerometeret står i forhold til jordens tyngdekræft. Problemet med dette er at accelerometer måler alle accelerationer, ikke kun jordens tyngdekræft. Når Swagwayen fx accelerer, bremser eller køre over en sten bliver den udregnet vinkel forkert.

For at komme begge problemer til livs bliver begge sensorer brugt og data fra begge to bliver samlet i et såkaldt Kalman-filter. Kalman-filtret kan tilnærmelsesvis ses som et high-pass og low-pass filter. Det taget de hurtige ændringer (high-pass) fra gyroskop-dataene og bruger accelerometer-dataene til at holde det målte og det fysiske nulpunkt ens.

Swagway Debugger

For at teste om Kalman-filtret er implementeret og tunet korrekt er det nødvendig at sammenligne de tre vinkler fra hhv. gyroskopet, accelerometeret og Kalman-filtret. En måde at gøre dette på er ved at sende disse data fra elektronikken på Swagwayen til en PC og vise dem grafisk. Det er her Swagway Debugger-softwaren kommer ind i billedet.

Swagway Debugger modtager data, over en seriel-port, fra en Arduino microcontroller på Swagwayen og skriver dataene på en graf. Som der står i produktbeskrivelsen:

[1] Websiden for hele projekter findes på `https://github.com/neic/Swagway`. Derfra er der adgang til alt kildekode, rapporter og andre skriftlige fremstillinger.

Opgaven er at lave software der kan modtage data fra en Arduino over en serielport og visualiserer det. Softwaren er til test af filtereringen af data fra et gyroskop og accelerometer. Softwaren viser bl.a. et rullende plot, samt nogle viserinstrumenter.

2 Udførelse

Swagway Debugger er skrevet på Microsoft Windows i C# som en Windows Form Application. Der er ingen eksterne afhængighedder; det bruger kun Microsoft biblioteker:

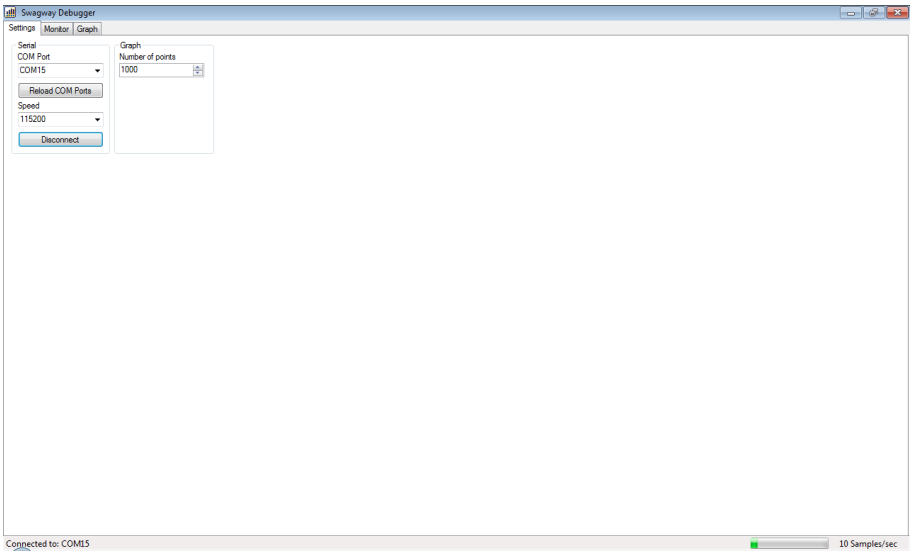
```
9  using System.IO.Ports; // Seriel-porte
10 using System.Text.RegularExpressions; // Regular expressions
11 using System.Globalization; // Forskel på , og .
12 using System.Windows.Forms.DataVisualization; // Grafer
```

Der er fem globale variabler:

```
18     string readFromUART; // Holder data læst direkte fra UART
19     string rxStringBuffer; // Holder rest af sidste pakke
20     List<string> rxListBuffer = new List<string>(); // Holder
           pakker som liste
21
22     int packageCount = 0;
23     int oldPackageCount = 0;
```

De tre første er buffere der bliver brugt til midlertidig at gemme modtaget data i, de to sidste er tællere over antallet af pakker.

2.1 Indstillinger



Ved opstart af applikationen indlæses alle tilgængelige serial-porte og der sættes en standardindstilling for valget af seriel hastighed:

```
34      /* Kaldes ved opstart: finder seriel-porte */
35      private void Form1_Load(object sender, EventArgs e)
36      {
37        LoadSerialPorts(); // Opdater listen med serialporte
38        cbSpeed.SelectedIndex = 10; // Sætter 115200 baud som
              standardindstilling
39      }
```

Ved afslutning af applikationen lukkes en eventuel åben seriel-port:

```
41      /* Kaldes ved afslutning: lukker seriel-porte */
42      private void Form1_FormClosing(object sender,
            FormClosingEventArgs e)
43      {
44        if (serialPort.IsOpen) // Luk seriel-porten hvis den er å
              ben
45          {
46            serialPort.Close();
47          }
48      }
```

`LoadSerialPorts()` kaldes ved opstart og ved tryk knappen `btReload`. Den føjer alle systemets serial-porte til listen, sætter den første seriel-port som standardindstilling og advare i statuslinjen hvis der ikke er fundet nogle seriel-porte:

```
50      /* Finder systemets seriel-porte */
51      private void LoadSerialPorts()
52      {
53        cbComPort.Items.Clear(); // Ryd listen
54
55        foreach (string s in SerialPort.GetPortNames()) // For
              alle serielporte i systemet
56          {
57            cbComPort.Items.Add(s); // Føj til listen
58          }
59
60        if (cbComPort.Items.Count > 0) // Hvis der er fundet
              seriel-porte
61          {
62            cbComPort.SelectedIndex = 0; // Sætter den første
                  seriel-port som standardinstilling
63          }
64        else
65          {
66            lbConnectionStatus.Text = "No COM-ports found"; //
                  Ellers advarer i statuslinjen
67          }
68      }
```

Eventhandleren for `btConnect` både forbinder og afbryder til en seriel-port. Hvis der er afbrud finder den hvilken seriel-port og hastighed der er valgt, forbinder og opdaterer statuslinjen og dens egen tekst til "Disconnect". Hvis der er oprettet forbindelse afbryder den og ligeledes opdaterer statuslinjen og dens egen tekst til "Connect":

```
76      /* Forbind og afbryd til serielporten */
77      private void btConnect_Click(object sender, EventArgs e)
```

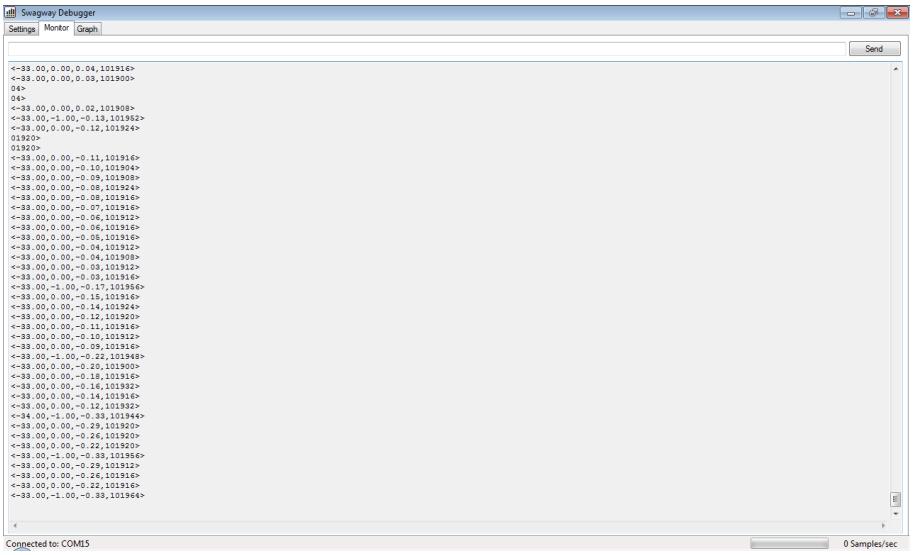
```
78      {
79        if (btConnect.Text == "Connect") // Hvis der skal
              forbindes
80          {
81            if (!serialPort.IsOpen) // Hvis der ikke er forbundet
82              {
83                serialPort.PortName = cbComPort.SelectedItem.
                      ToString(); // Find navnet på porten
84                serialPort.BaudRate = int.Parse(cbSpeed.
                      SelectedItem.ToString()); // Find hastigheden
85                serialPort.Open(); // Forbind
86              }
87
88            if (serialPort.IsOpen) // Hvis forbindelsen lykkedes
89              {
90                lbConnectionStatus.Text = "Connected to: " +
                      serialPort.PortName; // Skriv i statuslinjen
91                btConnect.Text = "Disconnect"; // Lav knappen om
                      til afbryd
92              }
93          }
94        else // Hvis der skal afbrydes
95          {
96            if (serialPort.IsOpen) // Hvis der er forbindelse
97              {
98                serialPort.Close(); // Afbryd
99              }
100
101           lbConnectionStatus.Text = "Disconnected"; // Skriv i
                  status linjen
102           btConnect.Text = "Connect"; // Lav knappen om til
                  forbind
103         }
104     }
```

NumericUpDown-tælleren, udPackages, bestemmer hvor meget data der skal vises på grafen. Når værdien i tælleren ændres ændre den dynamisk skala. Endeligt opdaterer den grafens X-akses maksimalværdi:

```
106     /* Længden af X-aksen på grafen */
107     private void udPackages_ValueChanged(object sender,
            EventArgs e) // Bliver kaldt hver gang tallet ændres
108     {
109       if (udPackages.Value <= 1000) // Hvis værdien er under
              eller ligemed 1000 skal den ændres med 100
110         {
111           udPackages.Increment = 100;
112         }
113
114       if (udPackages.Value > 1000) // Hvis værdien er over 1000
               skal den ændres med 1000
115         {
116           udPackages.Increment = 1000;
117         }
118
```

```
119         if (udPackages.Value == 1100) // Er værdien 1100 skal den
                være 2000
120           {
121             udPackages.Value = 2000;
122           }
123
124         chart1.ChartAreas.First().AxisX.Maximum = (double)
                udPackages.Value; // Set X-aksens maksimum værdi
125       }
```

2.2 Seriel modtagelse og monitor



Når der, efter der er forbundet til en seriel-port, ankommer data lægges det i bufferen `readFromUART` og `ReadToMonitor` kaldes:

```
131       /* Læs inkommende data til buffer og kald ReadToMonitor()
              */
132       private void serialPort_DataReceived(object sender,
              SerialDataReceivedEventArgs e)
133       {
134         readFromUART = serialPort.ReadExisting();
135         this.BeginInvoke(new EventHandler(ReadToMonitor));
136       }
```

`ReadToMonitor` Skriver den modtaget data, `readFromUART`, direkte til monitoren uden at behandle den. Den undesøger om der er meget data i monitoren og sletter en smugle af det ældste hvis det er tilfældet. Til slut kalder den `CleanData()`.

```
138       /* Skriver rå data til monitor, ryder op i monitor og
              kalder CleanData()*/
139       private void ReadToMonitor(object sender, EventArgs e)
140       {
141         tbMonitor.AppendText(readFromUART); // Tilføjer til
              monitor
```

```
142
143         if (tbMonitor.TextLength > 50000) // Hvis der mere end ca
                . 2000 pakker i monitor
144           {
145             tbMonitor.Lines = tbMonitor.Lines.Skip(20).ToArray();
                    // Slet 20 linjer
146           }
147
148         CleanData(readFromUART); // Kalder CleanData()
149       }
```

Datapakkerne fra Swagwayen er af formen < ⟨gyro⟩, ⟨acc⟩, ⟨kalm⟩, ⟨tid⟩ >

Hvor
⟨gyro⟩ er vinklen målt med gyroskopet med to decimaler og eventuelt negativt fortegn,
⟨acc⟩ er vinklen målt med accelerometeret med to decimaler og eventuelt negativt fortegn,
⟨gyro⟩ er vinklen udregnet af Kalman-filtret med to decimaler og eventuelt negativt fortegn og
⟨tid⟩ er tiden i µSS siden sidste pakke blev sendt.

`CleanData()` tager den rå modtaget data, `readFromUART`, og tilføjer det til en ny buffer, `rxStringBuffer`. Den indeholder allerede en rest data fra sidste gang. Denne buffer bliver splittet og hver pakke bliver indsat i listen `rxListBuffer`. Den sidste, ikke fuldstændige, pakke bliver lagt tilbage i `rxStringBuffer` og er klar til at bliver parret sammen ved næste kald af `CleanData()`:

```
151       /* Tager rå data, rengør det og sender det til graf */
152       private void CleanData(string input)
153       {
154         rxStringBuffer += input; // Tilføjer nyt rå data fra UART
155         rxListBuffer = rxStringBuffer.Split('<').ToList(); //
              Splitter ved hver begyndelse af ny pakke '<'
156         rxStringBuffer = rxListBuffer[rxListBuffer.Count() - 1];
              // Ligger den sidste, ikke fuldstændige, pakke
              tilbage i buffer
157         rxListBuffer.Remove(rxListBuffer.Last()); // Sletter den
              ikke fuldstendige pakke fra listen
```

Pakkerne i `rxListBuffer` bliver renset for alt der står efter ">" og bliver undersøgt om de passer ind i formen for en pakke af en regular expression. Hvis den gør dette bliver pakken splittet til et array af `strings` for derefter at bliver parset til et array af `floats`. Dette array bliver givet vidre til til funktionen `plotPackage()`:

```
159         /* For alle nye pakker */
160         for (int i = 0; i < rxListBuffer.Count(); i++)
161           {
162             /* Slet alt efter slutningen af pakken '>' */
163             int j = rxListBuffer[i].IndexOf('>'); // Finder
                  indexet af '>'
164             if (j > 0)
165               {
166                 rxListBuffer[i] = rxListBuffer[i].Substring(0, j)
                      ; // Klipper fra begyndelsen til indexet af
                      '>'
167               }
168             /* End : Selt efter slutning*/
```

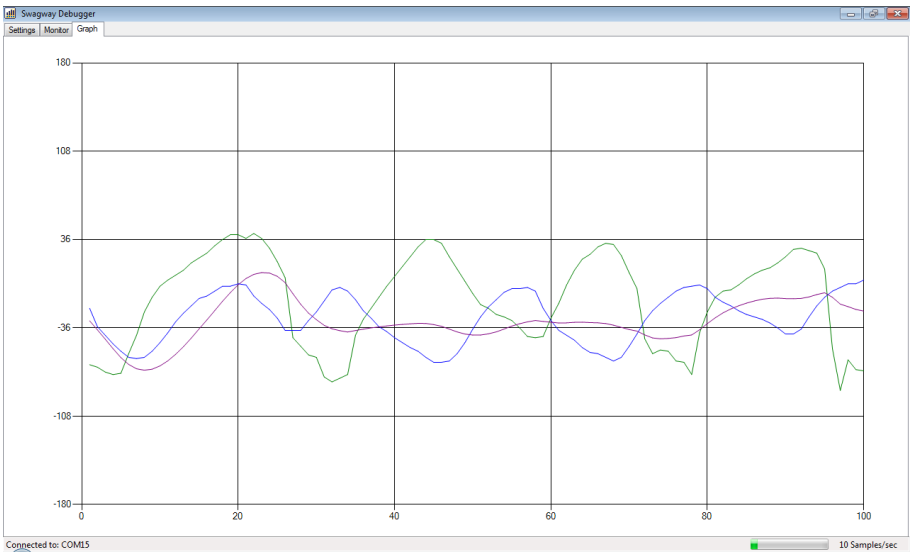
```
169
170             /* God pakke? Så plot den */
171             if (Regex.Match(rxListBuffer[i], @"^((-?\d{1,3}[.]\d\
                    d,){3,3}\d{4,})$").Success) //Pakken undersøges
                    om den passer ind.
172               {
173                 string[] stringPackage = rxListBuffer[i].Split
                        (',').ToArray(); // Pakken splittes til et
                        string-array
174                 float[] floatPackage = new float[4];
175
176                 for (int k = 0; k < stringPackage.Length; k++) //
                         String-arrayet parses til et float-array
177                   {
178                     float.TryParse(stringPackage[k], NumberStyles
                            .Float, CultureInfo.InvariantCulture, out
                             floatPackage[k]);
179                     /* TryParse prøver at parse, hvis det ikke er
                             muligt bliver pakken droppet.
180                      * CultureInfo er nødvendigt for at kunne
                             bruge "." som decimaltegn */
181                   }
182
183                 packageCount++;
184                 plotPackage(floatPackage); // Send float-arrayet
                        til grafen
185               }
186             /* End : God pakke */
187           }
188       /* End : For alle nye */
189     }
```

Applikationen kan også sende data. Det er undladt af journalen. Se hvordan virker på linje 195–218 af Form1.cs.

2.3 Graf

Når en ny pakke er blevet renset, bliver hver del af den tilføjet grafen i forskellige serier. Tiden bliver ignoreret. Der bliver så undersøgt om der er flere pakker end grænseværdien. Hvis der er det, bliver de fjernet fra grafen:

```
224     /* Tilføjer pakke til graf og rydder op */
225     private void plotPackage(float[] package)
226     {
227       for (int i = 0; i < package.Length-1; i++) // Plot alle
              punkterne i pakken til hver sin serie
228         {
229           chart1.Series[i].Points.AddY(package[i]);
230         }
231
232       if (packageCount > udPackages.Value) // Slet alle pakker
              over grænseværdi
233         {
234           for (int i = 0; i < package.Length-1; i++)
```




```
235                 {
236                   chart1.Series[i].Points.RemoveAt(0);
237                 }
238             }
239         }
```

2.4 Statuslinje

En timer kalder hvert sekund eventhandleren `timerSample_Tick` undersøger hvor mange pakker der er tilføjet det sidste sekund. Den skriver antallet til statuslinjen og opdaterer en statusbjælke:

```
245     /* Udregn antal pakker per sekund */
246     private void timerSample_Tick(object sender, EventArgs e)
            // Kaldes hvert sekund
247     {
248       int diff = packageCount - oldPackageCount; // Forskellen
              i antallet pakker siden sidste sekund
249       lbSamplesPerSec.Text = diff.ToString() + " Samples/sec";
              // Skriv det på status linjen
250
251       if (diff>pbSamplesPerSec.Maximum) // Hvis forskellen er
              større en maksimum af statusbjælken
252         {
253           pbSamplesPerSec.Maximum = diff; // Forstør makimum
254         }
255
256       pbSamplesPerSec.Value = diff; // Set statusbjælken til
              antallet af pakker per sekund
257       oldPackageCount = packageCount;
258     }
```

3 Debug_test.ino

Swagway projektet med gyroskopet og accelerometeret er afleveret, så for at teste Swagway Debugger er der lavet et lille test program til en Arduino. Til Arduinoen er der tilkoblet to potentiometerere til at simulere data fra gyroskop og accelerometer. Den del af softwaren som har interesse er den som sender pakken med data:

```
88 void sendToGraph()
89 {
90   Serial.print("<");
91   Serial.print(gyroAngle); //0
92   Serial.print(",");
93   Serial.print(accAngle); //1
94   Serial.print(",");
95   Serial.print(estAngle); //2
96   Serial.print(",");
97   Serial.print(micros()-sinceLastSend); //3
98   Serial.println(">");
99 }
```

4 Konklusion

Softwaren blev udviklet og opfyldte både de mål der blev sat i programmering C, samt blev brugt i teknikfag A-projektet til at teste Kalman-filteret.