Inbraakdetectie met Lazarus

Michaël Van Canneyt

July 8, 2012

Abstract

Een laptop of desktop kan met behulp van een klein Lazarus programma eenvoudig omgetoverd worden tot een inbraakalarm. De Windows API staat toe fotos te maken van een kamer, en met behulp van een eenvoudig algoritme kunnen wijzigingen in de fotos gebruikt worden om een verwittigingsmail te sturen.

1 Introductie

De meeste - zo niet alle - laptops of tablets zijn uitgerust met een webcam. Modernere webcams hebben een ingebouwde bewegingssensor, maar oudere webcams kunnen eenvoudig gebruikt worden als bewegingssensor door middel van huis-tuin en keuken algoritmes. In dit artikel tonen we aan hoe zoiets kan gedmaakt worden met Lazarus.

Lazarus heeft geen kant-en-klaar componenten die kunnen dienen om webcams aan te spreken. De Lazarus-CCR site bevat echter wel een voorbeeld programma (geschreven door Bogdan Razvan Adrian) voor het werken met de Windows API voor video. Voor dit artikel zal dat programma aangepast worden om als bewegingssensor te dienen, en om een mail met in bijlage een beeldje te sturen als er beweging waargenomen wordt.

Windows heeft 2 APIs voor het werken met video. Een oude, legacy API (Video For Windows) en een nieuwe: DirectShow (deel van DirectX). De nieuwe API is veel krachtiger, maar ligt dichter bij de hardware en vereist het gebruik van callbacks en interfaces, deze API is dan ook moeilijker in gebruik dan de oudere Video For Windows. Om het eenvoudig te houden maken we gebruik van de oude API - die overigens nog steeds werkt onder Windows 7.

Het sturen van een mailtje met Lazarus is in een vorig artikel uitvoerig belicht, we zullen daar dan ook niet dieper op ingaan.

2 Video For windows

De Video For windows interface werkt heel eenvoudig. Er wordt een Window Handle aangemaakt, en daarin wordt dan de video stream van de webcam getoond. De webcam (of andere video bron) wordt bestuurd door het sturen van Windows messages naar de handle van de webcam; er zijn een hoop messages die naar de webcam handle gestuurd kunnen worden, we zullen er maar een paar gebruiken.

De API van Video For Windows is vertaald naar Object Pascal. De macros die in de C/C++ interface beschikbaar zijn, zijn ook vertaald. Zij verbergen het gebruik van messages, en leveren een procedurale API op. Dit alles is beschikbaar in de VFW unit, die meegeleverd is met de sources van dit artikel. Alle functies beginnen met de prefix `cap`, van het engelse `capture`.

De belangrijkste functies (of messages) die we nodig hebben, zijn de volgende:

capCreateCaptureWindow Maakt een window handle aan waarmee het beeld van de camera getoond kan worden. Deze handle is nodig voor alle andere operaties.

capDriverConnect Verbindt de window handle met een camera. Er kunnen tot 10 cameras verbonden worden.

capDriverDisconnect Verbreekt de verbinding met de camera.

capDriverGetCaps Haalt de eigenschappen op van de camera?

capOverlay Start of stopt het tonen van het camera beeld in het aangemaakte window. Het tonen gebeurt dmv. video overlay.

capPreview Start of stopt het tonen van het camera beeld in het aangemaakte window. Het tonen gebeurt dmv. software rendering.

capPreviewRate Zet het framerate van de camera.

capPreviewScale Zet herschalen van het beeld aan of af.

capGrabFrameNoStop Zet het huidige frame van de camera in een buffer, maar stopt de camera niet.

capFileSaveDIB Bewaart het frame in de buffer in een bmp bestand.

capFileSetCaptureFile Stelt een bestandsnaam in voor het opnemen van een filmpje (.avi).

capCaptureSequence Begint een filmpje op te nemen. De bestandsnaam moet gezet zijn via `capFileSetCaptureFile`.

capFileSaveAs bewaart het bestand met het opgenomen filmpje.

capCaptureStop Stopt met het opnemen van een filmpje.

Er zijn nog meer functies, maar de bovenstaande zijn voldoende om een kleine applicatie mee te maken. De functie van elk van deze functies is duidelijk, en behoeft geen verdere uitleg. Buiten de `capCreateCaptureWindow` functie, moet aan elke functie de window handle van het capture window worden meegegeven.

3 Een voorbeeld programma

De applicatie de we zullen maken is heel eenvoudig: een scherm, met daarin een panel en dat panel toont de output van de webcam. Er zijn wat knoppen om de camera te starten en stoppen, en wat standaard windows dialogen te tonen die de eigenschappen van de camera instellen. Er is 1 knop die de bewegingsdetectie start en stopt, en een status balk die gebruikt wordt om wat staus berichten te tonen.

Bij opstart van de main form wordt de windows video handle gemaakt, dmv. de `capCreateCaptureWindow` functie. Deze functie krijgt de handle van een parent window mee (in dit geval de handle van de `pCapture` panel):

```
procedure TMainForm.CapCreate;
begin
  // Destroy if necessary
  CapDestroy;
  with pCapture do
```

```
    FCapHandle := capCreateCaptureWindow('Video Window',
      WS_CHILDWINDOW or WS_VISIBLE or WS_CLIPCHILDREN or WS_CLIPSIBLINGS
      , 5, 5, Width-10, Height-10, Handle, 0);
  if Not CapCreated then
    stCapture.Caption := 'ERROR creating capture window !!!';
end;
```

De capture window handle wordt als child window met een rand van 5 pixels binnen het `pcapture` panel gemaakt. De `CapCreated` functie is een functie van de `TMainForm` class, en kijkt na of `FCapHandle` verschillend is van nul: als de handle 0 is, is het aanmaken van de video capture window mislukt.

Nadat de capture window is gemaakt, kan er geconnecteed worden met de webcam driver. Dit gebeurt met de `CapConnect` functie binnen de form, die de `capDriverConnect` functie oproept met de capture window handle als argument:

```
procedure TMainForm.CapConnect;

Var
  l : integer;
  m : sTRING;

begin
  if Not CapCreated then Exit;
   // Disconnect if necessary
  CapDisconnect;
  // Connect the Capture Driver
  FConnected:=capDriverConnect(FCapHandle, 0);
  if Not FConnected then
    M:='ERROR connecting capture driver.'
  else
    begin
    L:=SizeOf(TCapDriverCaps);
    capDriverGetCaps(FCapHandle,@FDriverCaps,l);
    if FDriverCaps.fHasOverlay then
      M:='Driver connected, accepts overlay'
    else
      M:='Driver connected, software rendering';
    end
  stCapture.Caption:=M;
end;
```

Als de driver zonder problemen aan de capture window is gekoppeld, wordt d.m.v. de `capDriverGetCaps` de driver eigenschappen opgehaald: Het `FDriverCaps` record van type `TCapDriverCaps` wordt opgevuld met de eigenschappen van de webcam. Specifiek wordt nagekeken of de camera in staat is het beeld direct in het video kaart geheugen te schrijven of niet (`fHasOverlay`): indien mogelijk wordt deze eigenschap gebruikt, dit werkt aanzienlijk sneller.

Nadat de connectie met de webcam is gelegd, kan het eigenlijke tonen van het beeld dat de camera opvangt, beginnen. Dit gebeurt met de `capPreview` of `capOverlay` functies:

```
procedure TMainForm.CapEnableViewer;

Var
```

```
  M : String;

begin
  FLiveVideo := False;
  if Not FConnected then
    Exit;
  capPreviewScale(FCapHandle, True);     // Allow stretching
  if FDriverCaps.fHasOverlay then        // Driver accepts overlay
    begin
    capPreviewRate(FCapHandle, 0);       // Overlay framerate is auto
    FLiveVideo:=capOverlay(FCapHandle,True);
    M:='Hardware';
    end
  else                                   // Driver doesn't accept overlay
    begin
    capPreviewRate(FCapHandle, 33);      // Preview framerate in ms/frame
    FLiveVideo:=capPreview(FCapHandle, True);
    M:='Software';
    end;
  if FLiveVideo then
    M:=Format('Video Capture - Preview (%s)',[M])
  else
    M:='ERROR configuring capture driver.';
  stCapture.Caption :=M
end;
```

Merk op dat aan de `capOverlay` of `capPreview` de waarde `True` wordt doorgegeven. Na het oproepen van deze functies (hetgeen gebeurt in de `OnCreate` van het hoofdvenster), is de camera actief, en wordt het beeld getoond in het hoofdscherm. De `bReconnect` knop roept de 2 functies ook op, en kan gebruikt worden om alsnog de camera te activeren indien er iets is misgelopen.

Om het tonen van het beeld van de camera te stoppen, worden opnieuw de `capOverlay` en `capPreview` functies gebruikt, maar i.p.v. `True` wordt als argument `False` doorgegeven. de `CapDisableViewer` functie roept de correcte functie op:

```
procedure TMainForm.CapDisableViewer;
begin
  if FLiveVideo then
    begin
    if FDriverCaps.fHasOverlay then
      capOverlay(FCapHandle,False)
    else
      capPreview(FCapHandle,False);
    FLiveVideo := False;
    end;
end;
```

Om een filmpje op te nemen, volstaat het de `capFileSetCaptureFile`,`capCaptureSequence` en `capFileSaveAs` functies op te roepen. Tijdens het opnemen is het best het tonen van het camera beeld op scherm te stoppen. Dit gebeurt met de hierboven getoonde `capDisableViewer` functie. Als bestandsnaam wordt een naam met het opname start tijdstip gebruikt:

```
procedure TMainForm.CapRecord;
```

```
Const
  FN = '"Clip-"yyyy-mm-ss-hh-nn-ss".avi"';

begin
  // Stop if not yet stopped.
  CapStop;
  CapDisableViewer;
  // Construct filename
  FFileName:=ExtractFilePath(Application.ExeName);
  FFileName:=FFileName+FormatDateTime(FN,Now);
  stCapture.Caption:='Recording '+FFileName;
  bRecord.Caption := 'S&top';
  // Set filename
  capFileSetCaptureFile(FCapHandle,PChar(FFileName));
  // Start recording
  capCaptureSequence(FCapHandle);
  // Save file.
  capFileSaveAs(FCapHandle, PChar(FFileName));
  FRecording := True;
end;
```

Stoppen van video opname gebeurt met de `capCaptureStop` functie. Zodra de opname gestopt is, wordt aan de bestandsnaam van het filmpje het tijdstip van stoppen toegevoegd, en wordt het beeld van de camera terug op het scherm getoond:

```
procedure TMainForm.CapStop;

Const
  FN = '"---"yyyy-mm-ss-hh-nn-ss".avi"';

Var
  RFN : String:

begin
  if Not FRecording then
    Exit;
  FRecording := False;
  // Stop recording
  capCaptureStop(FCapHandle);
  // Rename file with timestamp
  RFN:=ChangeFileExt(FFileName, FormatDateTime(FN,Now));
  RenameFile(FFileName, RFN);
  // Show preview again on screen
  CapEnableViewer;
  stCapture.Caption := 'Recording stopped';
  bRecord.Caption := '&Record';
end;
```

4 Bewegingsdetectie

Met dit alles kan de camera al gebruikt worden om filmpjes te maken en op schijf op te slaan. Maar wat als de camera als bewegingssensor gebruikt moet worden ?

De camera API van Video For Windows kan ook het huidige video frame als een beeldje opslaan. Door dit op gezette tijdstippen te doen, en te kijken of er tussen de opeenvolgende beeldjes een betekenisvol verschil is, kan beweging voor de camera worden waargenomen, en kan er bevoorbeeld een mailtje met het beeld van de camera verstuurd worden. Om te vermijden dat er te veel mails verstuurd worden, wordt bij beweging slechts om de minuut een beeldje vestuurd.

Om dit te doen, is dus een timer nodig (`TMotion`). De timer is gedisabled, en met een druk op een knop kan de timer aangezet worden. In het timer event wordt het volgende gedaan:

```
procedure TMainForm.TMotionTimer(Sender: TObject);

begin
  Inc(FTicks);
  SaveTempFrame;
  if CheckDifferent then
    begin
    If MinutesBetween(Now,FLastSend)>1 then
      begin
      FLastSend:=Now;
      SendPicture;
      end;
    end;
end;
```

`FTicks` is een tellertje. de `SaveTempFrame` functie schrijft het huidige camera frame naar schijf. De `CheckDifferent` functie kijkt of er een vorig beeldje was, en geeft `True` terug als er een significant verschil is tussen het vorige en het huidige beeldje. Indien dat zo is, wordt gekeken of er genoeg tijd verstreken is en zo ja, wordt het beeldje verstuurd.

De interessante functies zijn `SaveTempFrame` en `CheckDifferent`. De eerste is heel eenvoudig:

```
Procedure TMainForm.SaveTempFrame;

begin
  capGrabFrameNoStop(FCapHandle);
  capFileSaveDIB(FCapHandle,PChar(FFrameFile));
end;
```

`FFrameFile` is de bestandsnaam, die wordt berekend als het programma start.

De `CheckDifferent` functie is de moeilijkste functie in het programma: Het moet het beeldje dat in `SaveTempFrame` bewaard werd, laden en vergelijken met het vorige beeldje.

Dit gebeurt door de pixels van het beeldje om te zetten in grijswaarden, en pixel voor pixel te vergelijken met het vorige beeldje (als er een was, de eerste keer is er natuurlijk nog geen). De grijswaarde wordt berekend door het gemiddelde te nemen van de R,G,B waardes van de kleur.

Het verschil tussen 2 opeenvolgende beeldjes kan op 2 manieren uitgedrukt worden: het aantal pixels dat verschilt kan geteld worden, of het verschil in grijswaarde tussen de pixels kan geteld worden.

Gewoon het aantal verschillende pixels tellen, levert slechte resultaten: De kleuren in de beeldjes die de camera maakt, fluctueren: Als de grijswaarden van 2 opeenvolgende stil-

staande beeldjes pixel voor pixel vergeleken worden, levert dat altijd een bijna 100% verschillend beeldje: geen 2 pixels op dezelfde locatie blijven hetzelfde. Een kleine statistiek maken toont aan dat de grijswaarden van de pixels met ongeveer 5 % fluctueren voor een stilstaand beeld. Dit brengen we in rekening door 2 opeenvolgende pixels pas als verschillend te nemen als de grijswaarde meer dan 5% verschilt.

Eenmaal het aantal verschillende pixels tussen 2 opeenvolgende beeldjes geteld is, moeten besloten worden of het een betekenisvol verschil is. Een beetje experimenteren leert dat als er bewegen voor de camera, ongeveer 10% verschillende pixels oplevert.

De 2 parameters van het algoritme zijn dus

- De fluctuatie die mag optreden tussen 2 kleurwaarden voor 1 pixel.
- Het relatieve aantal pixels dat mag verschillen tussen 2 beeldjes.

In het programma zijn er 2 spinedits, die toestaan deze waarden in te stellen (in %). Deze percentuele waarden worden in het begin van de `CheckDifferent` functie omgezet naar absolute waarden.

Het algoritme begint met het beeldje te laden in een tijdelijke bitmap, en alloceert dan een array voor de grijswaarden. Kleuren in FPC beeldjes zijn een record van word-sized R,G,B waardes), dus de array bevat words voor de grijswaarden.

```
function TMainForm.CheckDifferent : boolean;

Const
  MaxColor = Cardinal($FFFF);

Var
  A : Array of Word;
  R,C,I,PD,DC,TH,TC : Integer;
  D,MD: Int64;
  G : Word;
  P : TFPColor;

begin
  Result:=Length(FLastImage)<>0;
  FTempBMP.LoadFromFile(FFrameFile);
  TC:=FTempBMP.Height*FTempBMP.Width;
  TH:=Round(MaxColor/100*SETreshold.Value);
  MD:=TC*MaxColor;
  SetLength(A,TC);
```

De `MD` is het maximaal mogelijke kleurverschil tussen 2 beeldjes (Dus $FFFF maal het aantal pixels), De waarde `TH` is het minimale verschil in kleur tussen 2 pixels voordat ze als verschillend beschouwd worden. `FLastImage` is de array van grijswaarden die voor het vorige beeldje gebruikt werd.

Dan kan de loop voor het vergelijken van de pixels gestart worden. Voor elke pixel wordt een grijswaarde berekend, en terug opgeslagen in het beeldje. Tegelijkertijd wordt het verschil met de vorige grijswaarde voor de pixel berekend, en opgeteld bij het totaal. Het totaal aantal verschillende pixels wordt ook opgehoogd, indien nodig.

```
  I:=0;
  D:=0;
  dc:=0;
```

```
  For R:=0 to FTempBMP.Height-1 do
    For C:=0 to FTempBMP.Width-1 do
      begin
      P:=FTempBMP.Colors[C,R];
      G:=(P.blue+P.red+P.Green) div 3;
      P.Blue:=G;
      P.Red:=G;
      P.Green:=G;
      FTempBMP.Colors[C,R]:=P;
      A[i]:=G;
      if (I<Length(FLastImage)) then
        begin
        PD:=Abs(G-FLastImage[i]);
        If (PD>TH) then
          begin
          inc(DC);
          D:=D+Abs(PD);
          end;
        end;
      Inc(i);
      end;
```

Als de loop afgelopen is, slaan we de array met grijswaarden op in `FLastImage`, en wordt het resultaat van de functie berekend. De statusbalk wordt gebruikt om wat cijfers te tonen: het aantal verschillende kleuren, en het totaal aantal verschillende pixels. Als het functieresultaat aangeeft dat het beeldje verschilt, wordt het beeldje (dat nu is omgezet in een beeldje met alleen grijswaarden) ook opgeslagen op schijf:

```
  FLastImage:=A;
  STCapture.Caption:=Format('Try %d - Color:  %d (%f %%) Pixels: %d/%d (%f %%)',
                            [FTicks, D, D/MD*100, DC, TC, DC/TC*100]);
  if Result then
    begin
    Result:=(D/MD*100)>SETrigger.Value;
    if Result then
      FTempBMP.SaveToFile(FBWFrameFile);
    end;
end;
```

Al wat nu nog moet gebeuren, is het opgeslagen beeldje mailen naar een mail adres. Dit gebeurt met behulp van synapse. De werking van synapse is al in een vorige bijdrage uitvoerig uitgelegd, zodat de werking van de `SendPicture` functie duidelijk zou moeten zijn:

```
procedure TMainForm.SendPicture;

Var
  Mime : TMimeMess;
  P : TMimePart;
  B : Boolean;
  AText,AServer,ATO : String;
  L : TStringList;

begin
```

```
  STCapture.Caption:='Sending picture';
  ATO:='editor@blaisepascal.eu';
  AServer:='mail.blaisepascal.eu';
  AText:=FormatDateTime('dd/mm/yyyy hh:nn:ss',Now);
  AText:=Format('Camera heeft beweging ontdekt om %s',[AText]);
  Mime:=TMimeMess.Create;
  try
    Mime.Header.ToList.Text:=ATo;
    Mime.Header.Subject:='Beweging ontdekt';
    Mime.Header.From:=ATo;
    P:=Mime.AddPartMultipart('mixed',Nil);
    L:=TstringList.Create;
    try
      L.Text:=AText;
      Mime.AddPartText(L,P);
      Mime.AddPartBinaryFromFile(FFrameFile,P);
      Mime.EncodeMessage;
      B:=SendToRaw(ATo,ATo,AServer,Mime.Lines,'','');
    finally
      L.Free;
    end;
    if not B then
      STCapture.Caption:='Failed to send picture'
    else
      STCapture.Caption:='Sent picture to '+ATo;
  finally
    Mime.Free;
  end;
end;
```

5 conclusie

Het is eenvoudig om een filmpje op te nemen met behulp van een webcam en lazarus. De webcam als bewegingssensor gebruiken is ook niet moeilijk, zoals de code hierboven aantoont: het hier gepresenteerde algoritme is misschien niet het best mogelijke, maar het is eenvoudig en begrijpbaar. Het is eenvoudig aan te passen, er is ruimte voor variatie: de grijswaarde zou anders berekend kunnen worden, het verschil tussen 2 pixels kan ook anders berekend worden. slechts een deel van het beeldje zou kunnen gebruikt worden. Er zijn waarschijnlijk ook betere algoritmes denkbaar dan rechttoe-rechtaan vergelijken van 2 beeldjes: de termen `Motion`, `Detection Algorithm` ingeven in Google levert een hoop wetenschappelijke publicaties op over het onderwerp.