Intro til C++ og OpenCV

Maskinsynkurs 4690

Velkommen til kurs i maskinsyn! I dette kurset skal vi se på hvordan en maskin kan tolke og forstå verden rundt seg ved hjelp av bilder. Verktøyene vi bruker er programmeringsspråket C++[1], matrisebiblioteket Eigen[2] og maskinsynbiblioteket OpenCV[3]. Hensikten med dette kompendiet er å gi en liten innføring i disse verktøyene, og å gi nok forkunnskaper til å kunne komme i gang med labøvelsene for kurset.

Det forutsettes at kursets deltagere er fortrolige med programmering, så hovedfokuset vil være på å forklare særegenheter med C++ og bibliotekene vi bruker.

1 C++

C++ er et populært høynivå programmeringsspråk som er en utvidelse av språket C. Populariteten skyldes både at det bygger på C og at det er implementert for de fleste operativsystem. C++ er i mye større grad enn C et høynivå språk. Språket har støtte for objektorientert og generisk programmering og tilbyr flere generelle lagringstyper, iteratorer, algoritmer og I/O-funksjonalitet. Minnehåndtering er i større grad tatt hånd om for brukeren, og det er lettere å skrive trygg og rask kode. Hvis du er helt nybegynner innen C++, les gjerne mer på [1] og søk litt rundt på nettet.

Ragnar Smestad
Forsvarets forskningsinstitutt, e-mail: ragnar.smestad@ffi.no

1.1 Hello, world

Vi begynner med å ta for oss et "hello, world"-eksempel. [4]

```
#include <iostream>

/* This program prints "Hello, world!"
 * to the command line
 */
int main(int argc, char* argv[])
{
    std::cout << "Hello, world!" << std::endl;
    return 0; // Everything is ok!
}
```

Output:

```
Hello, world!
```

I C++ gir *standardbiblioteket*[5] tilgang til mange av byggeklossene vi trenger for å lage et hvilket som helst program. Disse får vi tak i gjennom å bruke `#include <...>`. Man kan tenke på include-kommandoen som en copy/paste-operasjon, der kompilatoren erstatter include-linjen med det faktiske innholdet av filen du inkluderer. Det er to varianter av include som brukes:

include

```
#include "abc.h"
#include <iostream>
```

Forskjellen på de to variantene er hvor det startes å lete etter filen som skal inkluderes. For filer som er omsluttet av `<>` letes det bare under "implementation-defined places". Dette vil typisk si systemkataloger, så `<>` brukes hovedsaklig til filer fra standardbiblioteket. Ved bruk av hermetegn `""`, som er den andre varianten, vil man først sjekke mappen som den gjeldende filen ligger i, og deretter alle andre mapper som ligger i "include path". (Hvordan man setter "include path" blir ikke gjennomgått her). En litt spesiell ting med filer fra standardbiblioteket er at de ikke har noen filendelse (i eksempelet inkluderes filen `iostream`). iostream (forkortelse for "input and output stream") inneholder funksjoner for å skrive tekst til skjerm etc.

iostream

> Streams are serial interfaces to storage, buffers files, or any other storage medium. The difference between storage media is intentionally hidden by the interface; you may not even know what kind of storage you're working with but the interface is exactly the same.[6]

strøm-operatorer

Når vi jobber med "streams" bruker vi helst strøm-operatorer. Vi har insert-operator `<<` for å putte objekter inn i strømmer, og extract-operator `>>` for å lese objekter ut av strømmer. `std::cout` er **standard output stream**, som vi bruker for å skrive tekst til kommandolinjen. I eksempelet sender vi først "Hello, world!" til `std::cout`, og deretter sender vi inn `std::endl`. Det siste medfører at vi skriver et "newline"-tegn (`'\n'`) i tillegg til at vi "flusher" strømmen. Å flushe gjør at teksten skrives ut umiddelbart.

cout

En viktig ting å legge merke til er at strømmen `cout` ligger i *namespacet* `std`. Man får tilgang til en funksjon, klasse eller variabel fra et navngitt namespace ved å legge til namespacenavnet og `::` foran, slik vi nettopp har sett her. Namespace er en nyttig ting å bruke for å unngå kollisjon når to biblioteker definerer en funksjon med samme navn. (Et eksempel kan være `std::min` fra standardbiblioteket og `cv::min` fra OpenCV. Hvis begge bare hadde hett `min` hadde ikke kompilatoren visst hvem av dem du mente å bruke). namespace

Et kjørbart program i C++ må ha en funksjon som heter `main`. Main-funksjonen tar to argumenter, `int argc` og `char* argv[]` som inneholder henholdsvis antallet parametere som ble sendt inn til programmet og en peker til en array med disse argumentene. Hvis programmet vårt heter **hello_world** og vi starter det fra kommandolinjen ved å skrive `main` `argc, argv`

```
./hello_world arg1 arg2
```

så vil `argc` inneholde **3** og `argv` være en peker til et array som inneholder **"/path/til/hello_world"**, **"arg1"**, **"arg2"**. I eksempelet vårt brukes ikke disse verdiene i det hele tatt, og da er det lov å deklarere main med tomme parenteser:

```
int main()
{/* ... */}
```

Returverdien til main skal indikere hvordan programmet ble avsluttet. Hvis alt gikk bra skal det returneres **0**, og hvis alt ikke gikk bra returneres et tall som ikke er 0. Det er ingen standard for hvordan ikke-null verdier tolkes, så om du returnerer 1 eller 100 har ingen ting å si. Man kan eventuelt velge å returnere verdiene `EXIT_SUCCESS` og `EXIT_FAILURE` som er definert i `<cstdlib>`. I C++ er det faktisk lov å utelate `return` i main-funksjonen, noe som fører til at **0** blir automatisk returnert. `return`

I eksempelet er det brukt to typer kommentarer. `/* */` er kommentarer som kan strekke seg over flere linjer, mens `//` slutter ved linjeskift. kommentarer

1.2 Klasser

C++ har som nevnt støtte for objektorientert programmering, og vi kan lett lage egne klasser[7]. Det antas at konseptet med klasser og objektorientert programmering er kjent.

Det er vanlig praksis å skille en klasses deklarasjon (kalles *interface* eller *grensesnitt*) og definisjon (kalles *implementasjon*) i to forskjellige enheter. Interfacet legges i en såkalt headerfil, som bare inneholder navnet på klassen og en opplisting av klassens metoder og variable. Hva som skjer inne i metodene ligger i implementasjonen, som ligger i en egen kildefil eller eventuelt en ferdig kompilert objektfil. Vi tar for oss et enkelt eksempel:

numberkeeper.h

```
#ifndef NUMBERKEEPER_H
#define NUMBERKEEPER_H

/// This useless class does nothing else
/// than to store a number.
class NumberKeeper
{
public:
  /// Constructor which sets the number to be stored
  NumberKeeper(int the_number);

  /// Returns the stored number
  int getTheNumber();

private:
  int the_number_;
};
#endif // NUMBERKEEPER_H
```

Denne headerfilen forteller oss følgende ting:

- Det finnes en klasse som heter `NumberKeeper`.
- Klassen krever et tall i konstruktøren.
- Klassen har en *public* metode `getTheNumber`. Siden den er public kan vi bruke den.
- Klassen har en *private* klassevariabel som heter `the_number_`. Siden den er private er den utilgjengelig for oss som brukere.

Dette er nok informasjon for oss til å bruke klassen. Ut fra kommentarene ser vi hva den kan brukes til, i tillegg til at klasse-, metode- og variabelnavn egentlig er beskrivende nok i seg selv. At private variabler slutter med understrek er en ganske vanlig konvensjon, som vi også bruker i maskinsynkurset.

header guard

I filen er det brukt en *header guard*, som er linjene med `#ifndef` , `#define` og `#endif`. Denne sørger for at filen blir inkludert bare en gang, slik at ikke kompilatoren klager på "multiple definitions of ...". Man kan ikke deklarere to ting med samme navn, og header guards forhindrer dette. (Vit at man kan erstatte header guards med kommandoen `#pragma once`[8], som er enklere å bruke men ikke er standard C++. Vi holder oss til standarden og bruker header guards i maskinsynkurset).

Hvordan klassen løser sin oppgave er for oss helt ukjent, med mindre vi har tilgang på implementasjonen. I dette tilfellet har vi det:

numberkeeper.cpp

```
#include "numberkepper.h"

NumberKeeper::NumberKeeper(int the_number)
  : the_number_{the_number}
{}

NumberKeeper::getTheNumber()
```

```
{
  return the_number_;
}
```

Denne implementasjonen viser følgende:

- Scope-operatoren som vi kjenner igjen fra namespace brukes også når man implementerer en metode fra en klasse. Vi ser fra eksempelet at vi må ha med `NumberKeeper::` foran både konstruktøren og `getTheNumber`.
- Før krøllparentesene i konstruktøren har vi brukt en *member initializer list*[9]. Syntaksen for dette er kolon-tegnet, etterfulgt av en kommaseparert liste av initialiseringer av klassemedlemmer. (Vi har bare ett medlem og derfor ingen liste). Alternativet er å skrive `the_number_ = the_number;` mellom krøllparentesene.
- Metoden `getTheNumber` returnerer bare medlemsvariabelen `the_number_` (noe vi antagelig kunne gjettet).
- Cpp-filen inkluderer header-filen.

2 OpenCV

OpenCV er et pouplært softwarebibliotek for maskinsyn og maskinlæring, og i maskinsynkurset brukes dette flittig i labøvelsene for å prøve ut teorien i praksis. Litt mer generelt om OpenCV er hentet fra nettsiden:

> OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.
>
> The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. The library is used extensively in companies, research groups and by governmental bodies. (...)
>
> It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers. [3]

Med OpenCV ligger altså alt til rette for å ha det morsomt med maskinsyn! Dette kapittelet tar for seg noen utvalgte deler av OpenCV som vil være nyttige for å komme i gang med programmeringen av labøvelser. Det vil stadig bli henvist til dokumentasjonen, så ta gjerne en titt med en gang: https://docs.opencv.org/. I dette kompendiet henvises det i skrivende stund til versjon 3.3.1. I den nevnte dokumentasjonen er det også et introduksjonsavsnitt som gir en innføring i de vanligste API-konseptene i OpenCV. I stedet for å gjengi alt med egne ord her, oppfordres det i stedet til å lese gjennom denne introduksjonen[1].

dokumentasjon

introduksjon

2.1 cv::Mat

Den kanskje aller vanligste datatypen i OpenCV er `cv::Mat`, en klasse som representerer en n-dimensjonal en- eller flerkanals "dense" array. Med andre ord, dette er den typiske klassen for å representere bilder. Man finner den i dokumentasjonen under *Main modules:* > *Core functionality* > *Basic structures* > *cv::Mat*[2]. Det finnes

`cv::Mat`

[1] https://docs.opencv.org/3.3.1/d1/dfb/intro.html

[2] https://docs.opencv.org/3.3.1/d3/d63/classcv_1_1Mat.html

også en egen intro til `cv::Mat` i dokumentasjonen under *OpenCV Tutorials* > *The Core Functionality (core module)* > *Mat - The Basic Image Container*[3].

En viktig egenskap til `cv::Mat` er at klassen hovedsaklig er en peker til et minne hvor bildedataene ligger, og at klassen sørger for sikker minnehåndtering av dette minnet. Resten av klassen (omtalt som *matrix header*) inneholder informasjon om bildets størrelse og annen metainformasjon. Grunnen til at det er gjort slik, er for å unngå unødvendig kopiering av potensielt store mengder data når man kaller på funksjoner som skal prosessere bilder. Hvis man kaller en funksjon som tar en `cv::Mat` som parameter, så kopieres bare headeren, og kopien vil peke på minnet til den originale `cv::Mat`-en.

```
// lag en ny matrise, minne allokeres under panseret.
cv::Mat a(/* konstruktørparametere */);

cv::Mat b(a);           // bruk "copy constructor"
cv::Mat c = a;          // bruk "assignment operator"
```

Alle matrisene, både `a`, `b` og `c` peker faktisk på den samme ene datablokken i minnet. Headerene er forskjellige, men hvis man endrer på pikselverdier i en av matrisene vil det påvirke de to andre også.

Hvis man eksplisitt vil lage en kopi, så kan man bruke

```
cv::Mat a(/* ... */);
//Enten
cv::Mat b = a.clone();
//Eller
cv::Mat b;
a.copyTo(b);
```

For å lage en ny `cv::Mat` ser vi fra dokumentasjonen at vi har flere konstruktører å velge mellom:

```
Mat (int rows, int cols, int type)
Mat (Size size, int type)
Mat (int rows, int cols, int type, const Scalar &s)
Mat (Size size, int type, const Scalar &s)
```

Både `Size` og `Scalar` er egne datatyper i OpenCV, som det er lett å lete frem i dokumentasjonen. Verdien `int type` er litt sær og fortjener et eget avsnitt. Dette er en makro på formatet

```
CV_<bit-depth>{U|S|F}C(<number_of_channels>)
```

for eksempel `CV_8UC1` for et enkanals bilde med unsigned char, `CV_32FC3` for et trekanals bilde med 32-bits floats, osv.

[3] https://docs.opencv.org/3.3.1/d6/d6d/tutorial_mat_the_basic_image_container.html

2.2 Lese inn bilde fra fil eller kamera

imread

For å lese inn bilde fra fil benytter vi oss av funksjonen `cv::imread`[4]. Den tar to argumenter: filnavn og "flags". Med ulike flagg kan du velge mellom flere måter å lese inn bildet på. Standardargumentet er `cv::IMREAD_COLOR`, som betyr at hvis du ikke spesifiserer noe annet vil OpenCV konvertere et enkanals gråtonebilde til et trekanals bilde ved innlesing! Det er derfor typisk mer anvendelig å kalle funksjonen slik:

```
cv::Mat image = cv::imread("filnavn", cv::IMREAD_UNCHANGED);
```

VideoCapture

Innlesing av bilder fra et kamera gjøres via klassen `VideoCapture`[5]. I tillegg til å lese fra kamera kan klassen lese fra videofil, url eller til og med en bildesekvens (for eksempel vil filnavnet `img_\%02d.jpg` tolkes som sekvensen img_00.jpg, img_01.jpg, img_02.jpg, ...). For å lese inn fra kamera, så angis bare "kameraindeksen". Dette er 0 for det første kameraet som er koblet til maskinen (ofte det integrerte webcameraet), og så øker det til 1, 2, ... for hvert nye kamera du kobler til. Å koble til et kamera kan foregå slik:

```
cv::VideoCapture cap(0);

if (!cap.isOpened())
{
  std::cerr << "ERROR! Unable to open camera\n";
  return EXIT_FAILURE;
}
```

2.3 Vise frem et bilde

imshow

Veldig ofte har vi lyst til å vise fram et bilde på skjermen, og da bruker vi funksjonen `cv::imshow`[6]. Dokumentasjonen har en viktig kommentar til bruk av `cv::imshow`:

waitKey

> This function should be followed by `cv::waitKey` function which displays the image for specified milliseconds. Otherwise, it won't display the image. For example, `waitKey(0)` will display the window infinitely until any keypress (it is suitable for image display). `waitKey(25)` will display a frame for 25 ms, after which display will be automatically closed. (If you put it in a loop to read videos, it will display the video frame-by-frame)

Hvis du trykker en tast mens `cv::waitKey` venter vil funksjonen returnere en tallverdi som avhenger av tasten. Hvis ingenting trykkes før tiden har gått ut returneres -1.

[4] https://docs.opencv.org/3.3.1/d4/da8/group__imgcodecs.html#ga288b8b3da0892bd651fce07b3bbd3a56

[5] https://docs.opencv.org/3.3.1/d8/dfe/classcv_1_1VideoCapture.html

[6] https://docs.opencv.org/3.3.1/d7/dfc/group__highgui.html#ga453d42fe4cb60e5723281a89973ee563

I tillegg til `cv::Mat`-en vi vil vise frem, tar `imshow`-funksjonen et vindusnavn som argument. Dette gjør at vi kan vise bilder i det samme vinduet om igjen og om igjen. For å opprette et vindu kaller vi på funksjonen `cv::namedWindow`[7] . Vi oppsummerer med et fullstendig eksempel på hvordan man kan hente bilder fra et kamera og vise fram.

namedWindow

```cpp
#include "opencv2/highgui.hpp"
#include <iostream>

int main()
{
  cv::VideoCapture input_stream(0);

  if (!input_stream.isOpened())
  {
    std::cout << "could not open camera" << std::endl;
    return EXIT_FAILURE;
  }

  const std::string window_title = "window 0";
  cv::namedWindow(window_title);

  cv::Mat frame;
  while (true)
  {
    input_stream >> frame;
    if (frame.empty())
    { break; }

    cv::imshow(window_title, frame);
    if (cv::waitKey(15) >= 0)
    { break; }
  }

  return EXIT_SUCCESS;
}
```

2.4 Prosessere bilder

Dette avsnittet gir en innføring i prosessering av bilder. Vi starter med å iterere over et bilde og samtidig jobbe på enkeltpixler. Vi oppretter et tomt bilde uten å lese fra fil eller kamera denne gangen. Til dette benytter vi metoden `cv::Mat::zeros` som gir oss et bilde med alle pixlene satt til 0. Deretter løper vi gjennom hele bildet og setter en og en pixelverdi. Verdien regnes ut fra en formel som avhenger av pixelets posisjon.

zeros

[7] https://docs.opencv.org/3.3.1/d7/dfc/group__highgui.html#ga5afdf8410934fd099df85c75b2e0888b

```cpp
#include "opencv2/imgproc.hpp"
#include "opencv2/highgui.hpp"
#include <iostream>

int main()
{
  cv::Mat frame = cv::Mat::zeros(16, 16, CV_8UC1);

  for (int row = 0; row < frame.rows; ++row)
  {
    for (int col = 0; col < frame.cols; ++col)
    {
      const uchar pixel_value = (row+1) * (col+1) - 1;
      frame.at<uchar>(row,col) = pixel_value;
    }
  }

  cv::resize(frame, frame, cv::Size(256, 256), 0, 0,
             cv::INTER_LINEAR);

  cv::imshow("gradient", frame);
  cv::waitKey(0);

  return EXIT_SUCCESS;
}
```

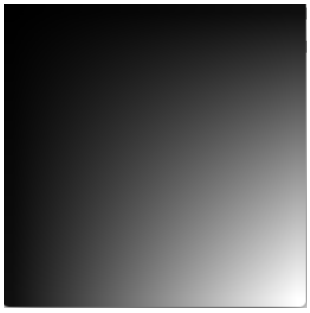

Fig. 1: Gradientbilde

resize

interpolasjon

Resultatet ser vi i fig. 1. Legg merke til at vi også brukte funksjonen `cv::resize`[8] for å endre bildet vårt (som bare var 16 x 16 pixler) til en størrelse som var bedre egnet for visning. For å få pen gradient ble det valgt interpolasjonsmetode `cv::INTER_LINEAR`, altså lineær interpolasjon.

I koden ser vi eksempel på at en `cv::Mat` innholder metadata om matrisen sin. I forløkkenes testuttrykk hentes bildets høyde og bredde ut med `frame.rows`

[8] https://docs.opencv.org/3.3.1/da/d54/group__imgproc__transform.html#ga47a974309e9102f5f08231edc7e7529d

og `frame.cols`. Vi kunne faktisk også brukt `frame.size().height()` og `frame.size().width()` hvis vi ville gå veien om `cv::Size`. Size

En litt spesiell ting er at vi må fortelle `cv::Mat::at` hva slags datatype vi har i bildet vårt. I eksempelet står det `frame.at<uchar>` for å indikere at bildet inneholder **`unsigned char`** (forkortet til uchar), akkurat som spesifisert med `CV_8UC1` da matrisen ble opprettet. Les mer om hvordan dette blir for andre datatyper og flere kanaler i tutorial *Operations with images*[9]

Andre nyttige operasjoner på bilder er det å konvertere mellom datatyper eller å konvertere fra fargebilde til gråtonebilde og motsatt. For å endre type bruker vi metoden `cv::Mat::convertTo`[10]. Den kan samtidig skalere pixelverdiene, slik at vi i følgende eksempel går fra heltallsverdier $\{0, 1, \ldots, 255\}$ til flyttall i intervallet $[0, 1]$. convertTo

```
cv::Mat uchar_image(16, 16, CV_8UC1);
cv::Mat float_image;
uchar_image.convertTo(float_image, CV_32FC1, 1/255.0f);
```

For å endre fra fargebilde til gråtonebilde brukes funksjonen for å endre fargerom, `cv::cvtColor`[11] med parameteren `cv::COLOR_BGR2GRAY`. Ved å bruke andre konverteringskoder kan vi konvertere til og fra HSV, L*a*b* og flere andre fargerom. cvtColor

```
cv::Mat color_image = imread("color_image.jpg");
cv::Mat gray_image;
cv::cvtColor(color_image, gray_image, cv::COLOR_BGR2GRAY);
```

2.5 Lykke til!

Fig. 2: Lykke til!

[9] https://docs.opencv.org/3.3.1/d5/d98/tutorial_mat_operations.html
[10] https://docs.opencv.org/3.3.1/d3/d63/classcv_1_1Mat.html#adf88c60c5b4980e05bb556080916978b
[11] https://docs.opencv.org/3.3.1/d7/d1b/group__imgproc__misc.html#ga397ae87e1288a81d2363b61574eb8cab

```cpp
#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc.hpp"

int main()
{
  cv::Mat frame = cv::Mat::zeros(256, 512, CV_8UC1);
  frame += 242;

  const auto font = cv::FONT_HERSHEY_SIMPLEX;
  cv::putText(frame, "Lykke til!", {50,150}, font, 3,
              {0,0,0}, 2, cv::LINE_AA);
  cv::imwrite("tnx.png", frame);

  return EXIT_SUCCESS;
}
```

References

[1] Wikipedia. *C++*. URL: https://no.wikipedia.org/w/index.php?title=C%2B%2B&oldid=17629715 (visited on 01/18/2018).

[2] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http://eigen.tuxfamily.org. 2010.

[3] OpenCV team. *Open Source Computer Vision Library*. URL: https://opencv.org/ (visited on 01/18/2018).

[4] Wikipedia. *Hello, world*. URL: https://no.wikipedia.org/w/index.php?title=Hello,_world&oldid=18113685 (visited on 01/18/2018).

[5] Wikipedia. *C++ Standard Library*. URL: https://en.wikipedia.org/w/index.php?title=C%2B%2B_Standard_Library&oldid=817321592 (visited on 01/18/2018).

[6] Manasij Mukherjee — cprogramming.com. *A Gentle Introduction to C++ IO Streams*. URL: https://www.cprogramming.com/tutorial/c++-iostreams.html (visited on 01/18/2018).

[7] Wikipedia. *C++ classes*. URL: https://en.wikipedia.org/w/index.php?title=C%2B%2B_classes&oldid=816783232 (visited on 01/18/2018).

[8] Wikipedia. *Pragma once*. URL: https://en.wikipedia.org/w/index.php?title=Pragma_once&oldid=818631485 (visited on 01/18/2018).

[9] cppreference.com. *Constructors and member initializer lists*. URL: http://en.cppreference.com/w/cpp/language/initializer_list (visited on 01/18/2018).