

```
// Projekt      Laser Targeting Project
//
// Fil          PCcontroller.h
//
// Beskrivelse   Implementering af headeren PCcontroller.
//              En header til serial kommunikation.
//
// Forfatter     Erik Gross Jensen
//
// Version       1.0 041200 EGJ - oprindelig version
//              1.1 040202 EGJ - flere kommentarer tilføjet
//              1.2 181108 NVJ - tilrettet 1. sem. projekt
//              1.3 260511 SLT - tilrettet 2. sem. projekt

#pragma once

#include <Windows.h>
#include <iostream>
using namespace std;

class PCcontroller
{
public:
    bool open( int port, int baud );
    bool close();
    bool send( char *sendPtr, int antal );
    int receive( char *rxPtr );
    int inWaitingBuffer();
    char receiveOneChar();
    bool laserOn();
    bool laserOff();
    bool turnLeft();
    bool turnRight();
    bool turnDown();
    bool turnUp();
    bool seek(int x, int y);
    void toInt(int &a, char* c);
private:
    HANDLE HComdev;                //Def af handle til pc
    DWORD dwErrorFlags;
    COMSTAT ComStat;
    DCB dcb;                      //Def af
    variabel til DCB structen
};
```

```

// Projekt      Laser Targeting Project
//
// Fil          PCcontroller.cpp
//
// Beskrivelse  Implementering af klassen PCcontroller.
//              En klasse til serial kommunikation.
//
// Forfatter    Erik Gross Jensen
//
// Version      1.0 041200 EGJ - oprindelig version
//              1.1 040202 EGJ - flere kommentarer tilføjet
//              1.2 181108 NVJ - tilrettet 1. sem. projekt
//              1.3 260511 SLT - tilrettet 2. sem. projekt

#include "PCcontroller.h"
#include <cstring>
#include <stdlib.h>

//*****
bool PCcontroller::open( int port, int baud )
// Åbner en serial port for kommunikation.
// Input:  port : Comport nummer 1,2,3,4 eller 5
//         baud : Hastigheden for kommunikationen: 9600, 19200 eller 38400
// Output: true hvis porten kunne åbnes
//         false hvis porten ikke kunne åbnes. Når der returneres false
skyldes
//         det ofte at et andet program bruger den serielle port
//*****
{
    char portstr[5] = "COM ";

    portstr[3] = 48 + port ;

    // Convert to WCHAR
    DWORD charLength = MultiByteToWideChar (CP_ACP, 0, portstr, -1, NULL,
0);

    wchar_t *wideText;
    wideText = new wchar_t[charLength];
    if(!wideText)
    {
        delete []wideText;
    }

    MultiByteToWideChar (CP_ACP, 0, portstr, -1, wideText, charLength );

    HComdev = CreateFile( wideText, // Navnet på den port der skal
åbnes(COMX)
                                GENERIC_READ | GENERIC_WRITE,
// read/write types
                                0,
                                0,
                                OPEN_EXISTING,
                                0,
                                0);

    delete wideText; // Clean up dynamic memory

    if( HComdev == INVALID_HANDLE_VALUE )
        return false;
    else
    {
        //sæt hastighed m.m.
        GetCommState( HComdev, &dcb );
        dcb.DCBlength = sizeof(DCB);

        //set baud rate
        if( baud == 9600 )

```

```

        dcb.BaudRate = CBR_9600;
    else if( baud == 19200 )
        dcb.BaudRate = CBR_19200;
    else if( baud == 38400 )
        dcb.BaudRate = CBR_38400;
    else
        return false;

    // set databit
    dcb.ByteSize = 8;

    //set paritet
    dcb.Parity = 0;

    //set stopbit
    dcb.StopBits = ONESTOPBIT;

    SetCommState( HComdev, &dcb );

    return true;
}

//*****
bool PCcontroller::close()
// Lukker forbindelsen
// Input:
// Output: true hvis forbindelsen blev lukket ellers false
//*****
{
    if( CloseHandle(HComdev) )
        return true;
    else
        return false;
}

//*****
bool PCcontroller::send( char *sendPtr, int antal )
// Sender et antal karakterer på en seriel port.
// Husk porten skal være åben inden send kan bruges
// Input: *sendPtr : en pointer til den char streng der skal sendes
//         antal    : Er det antal char der skal sendes
// Output: true hvis afsendelse gik godt
//         false hvis der ikke blev sendt noget. Når der returneres
//         false skyldes det ofte at porten ikke er åbnet
//*****
{
    DWORD dwBytesWritten;

    if( WriteFile(HComdev, sendPtr, antal, &dwBytesWritten, 0) )
        return true ;
    else
        return false;
}

//*****
int PCcontroller::inWaitingBuffer()
// Tæller antallet af karakterer der findes i receive buffer
// Husk porten skal være åben inden send kan bruges
// Input:
// Output: returnerer antallet af karakterer der er findes i receive buffer
//*****
{
    ClearCommError( HComdev, &dwErrorFlags, &ComStat );
    return ComStat.cbInQue ;
}

```

```

//*****
int PCcontroller::receive( char *rxPtr )
// Henter de karakterer der findes i receive buffer. Bemærk hvis der
// ingen karakterer er i receive buffer vil funktionen først returnere
// når der kommer en karakter i receive bufferen.
// Input: en pointer til et char array hvor data skal overføres til
// Output: returnerer det antal karakterer der er overført til rxPtr
//*****
{
    DWORD bytesRead, dwBytesRead;

    dwBytesRead = inWaitingBuffer();
    ReadFile( HComdev, rxPtr, dwBytesRead, &byteRead, 0 );
    return byteRead;
}

//*****
char PCcontroller::receiveOneChar()
// Læser 1 karakter fra receive buffer. Bemærk hvis der ingen karakterer
// er i receive buffer vil funktionen først returnere, når der kommer en
// karakter i receive bufferen.
// Input:
// Output: Returnerer den læste karakter.
//*****
{
    char rxBuf;
    DWORD bytesRead;

    ReadFile( HComdev, &rxBuf, 1, &byteRead, 0 );
    return rxBuf;
}

//*****
// Laserfunktionerne samt funktionerne for motorens bevægelser
// Oprindeligt ikke en del af PCcontroller-klassen givet i 1. semester
//*****
bool PCcontroller::laserOn() // Funktion sørger for at tænde laseren
{
    char tmp; // Lav en char
    char sendChar = '5'; // Den char, der skal sendes
    send(&sendChar, 1); // sender charværdien '5', der sendes til µC
    tmp = receiveOneChar(); // char'en modtages og gemmes i tmp
    if(tmp == '0') // Hvis tmp == 0, send igen
        send(&sendChar, 1);

    return true;
}

bool PCcontroller::laserOff() // Funktion sørger for at slukke laseren
{
    char temp; // Lav en char
    char sendChar = '5'; // Den char, der skal sendes
    send(&sendChar, 1); // sender charværdien '5', der sendes til µC
    temp = receiveOneChar(); // char'en modtages og gemmes i temp
    if(temp == '1') // Hvis temp == 1, send igen
        send(&sendChar, 1);

    return true;
}

bool PCcontroller::turnLeft() // Funktion, der får µC til at dreje til venstre
{
    char left = '1'; // Sæt char left til ASCII-værdien '1'
    if(left == '1') // Hvis char'en = '1', så drejes motoren til venstre
        send(&left, 1); // Sender charværdien til µC
    return true;
}

```

```

}

bool PCcontroller::turnRight() // Funktion, der får µC til at dreje til højre
{
    char right = '2'; // Sæt char right til ASCII-værdien '2'
    if(right == '2') // Hvis char'en = '2', så drejes motoren til højre
        send(&right, 1); // Sender charværdien til µC
    return true;
}

bool PCcontroller::turnDown() // Funktion, der får µC til at dreje nedad
{
    char down = '3'; // Sæt char down til ASCII-værdien '3'
    if(down == '3') // Hvis char'en = '3', så drejes motoren nedad
        send(&down, 1); // Sender charværdien til µC
    return true;
}

bool PCcontroller::turnUp() // Funktion, der får µC til at dreje til opad
{
    char up = '4'; // Sæt char up til ASCII-værdien '4'
    if(up == '4') // Hvis char'en = '4', så drejes motoren opad
        send(&up, 1); // Sender charværdien til µC
    return true;
}

bool PCcontroller::seek(int xkor, int ykor) // Funktion, der får µC til at søge
{
    int const size = 5; // Størrelsen af arrayet for at sende koordinater
    med '\0'
    char sendChar = '6'; // sæt char'en 'sendChar' til ASCII-værdien '6'

    char x[size] = {}; // Sætter et x-array for førsteaksen
    char y[size] = {}; // Sætter et y-array for andenaksen
    send(&sendChar, 1); // Send besked om anmodning

    // Send koordinater vha. itoa(int value, char * str, int base) for hhv.
    x og y
    itoa(xkor, x, 10);
    itoa(ykor, y, 10);

    int xlenght = (size-1)-strlen(x);
    int ylenght = (size-1)-strlen(y);

    x[size-1] = '\0';
    for(int i = 0; i < xlenght; i++)
    {
        for(int j = strlen(x); j > 0; j--)
        {
            x[j] = x[j-1];
        }
    }
    x[0] = '0';
}

    y[size-1] = '\0';
    for(int h = 0; h < ylenght; h++)
    {
        for(int g = strlen(y); g > 0; g--)
        {
            y[g] = y[g-1];
        }
    }
    y[0] = '0';
}

    // Sørger for at erstatte et minustegn med tallet 0, ellers 1 for
    positiv
    for(int i = 0; i < size-1; i++)
    {

```

```
        if(x[i] == '-')
        {
            x[i] = '0';
            x[0] = '0';
            break;
        }
        else
            x[0] = '1';
    }

    for(int i = 0; i < size-1; i++)
    {
        if(y[i] == '-')
        {
            y[i] = '0';
            y[0] = '0';
            break;
        }
        else
            y[0] = '1';
    }

    while(inWaitingBuffer() > 0) // Clear thrash
        receiveOneChar();
    send(x, size-1);

    send(y, size-1);

    while(!receiveOneChar() == '1');

    return true;
}
```

```

#ifndef AUTO_H
#define AUTO_H
// File: Auto.h
/*
    Description: The Auto class for scanning and targeting.(Header)
    Author: Group 4
    Date: 26/05 -11
*/
#include <cv.h> // Include Open Source Computer Vision library.
#include "PCcontroller.h" // Include serial communication class.
#include <QThread> // Include QT threading library.

// Log includes
#include <QDate>
#include <QTime>
#include <QTextStream>
#include <QObject>

// Defines structure for checkvalid() parameter, which holds values of Red green
and blue.
struct RGB {
    int Red;
    int Green;
    int Blue;
};

class Auto : public QThread
{
    Q_OBJECT
public:
    Auto(PCcontroller *, IplImage * &);
    // Precondition: Provided value must be of struct RGB and values of R G
    B must be 0-255.
    // Postcondition: All color variables will be initiated so target can be
    located.
    bool checkValid(RGB);

    // Precondition: When initiating scan(), the current location will be
    used as null point of x and y axis.
    // Postcondition: Will run x and y axis in a sequence to cover x 180 and
    y 120 degrees, and move to targets that show up and shoot them.
    void scan();

    // Precondition: If a target is found, it will be the leftmost and
    biggest to the left of the current frame.
    // Postcondition: Will put x and y values into class variables
    targetLockX and targetLockY
    // for found targets and return true/false depending if target was
    found or not.
    bool target();

    // Postcondition: Starts scan function.
    void run();
    bool autoRunning; // To stop it

signals:
    void updateAutoLog(QString message);

private:
    PCcontroller * pcuPtr; // Pointer to object of serial
    class.
    IplImage * &frame; // Pointer to object of
    image from webcam.
    RGB targetColor; // Object of struct RGB,
    that holds each value of R G B.

    int targetLockX; // Holder for target

```

```
frame x value.
    int targetLockY;                // Holder for target
frame y value.

    int areaPixelReticle;           // Quadrant for target to be in
to turn on laser.
    int perIntervalPixelX;          // Scan interval for x axis
motor.
    int perIntervalPixelY;          // Scan interval for y axis
motor.
    int maxMinX;                    // Defines edges for x
scan sequence.
    int maxMinY;                    // Defines edges for y
scan sequence.
    int xyIncrease;                 // Pixel to move, to
fine-tune pointing location for target.

    int rColorPlus, rColorMinus, gColorPlus, gColorMinus, bColorPlus,
bColorMinus; // Holds RGB deviation values.
    int colorDeviation;              // How big deviation of
colors can be +-.
    int validAmountPixelTarget;      // Amount of pixels along x axis
that must be to qualify as target.
};
#endif
```



```

// File: Auto.cpp
/*
    Description: The Auto class for scanning and targeting.(Implementation)
    Author: Group 4
    Date: 26/05 -11
*/
#include "Auto.h"

Auto::Auto(PCcontroller * pcu, IplImage * &framept) : frame(framept)
{
    pcuPtr = pcu;
    // Pointer for serial communication.

    //Initiate Variables.
    areaPixelReticle = 10; // Makes the quadrant
    (reticle) where target must be within for laser to start.
    colorDeviation = 30; // How much the RGB can
    differentiate +-.
    validAmountPixelTarget = 10; // How many pixels must
    be in one line scan of picture before its designated a target.
}

bool Auto::checkValid(RGB color)
{
    //Initiate Variables.
    perIntervalPixelX = frame->width/2; // Half a frame to be
    moved on X.
    perIntervalPixelY = frame->height/2; // Half a frame to be
    moved on Y.
    maxMinX = 1.5*frame->width; // To reach 180
    on x axis and frame spans 60 degree.. we need 90 degree from origo in both
    directions.
    maxMinY = frame->height; // To reach 120
    degree, we need 60 degree from origo in both directions, since frame size is 60
    degree(Defined).
    xyIncrease = 30; // Number of pixels to
    move axis when approaching to perfect in position for target.

    // Check if the color is within R parameters(0-255) if not return false for
    check.
    if(color.Red >= 0 && color.Red <= 255)
        targetColor.Red = color.Red;
    else
        return false;

    // Check if the color is within G parameters(0-255) if not return false for
    check.
    if(color.Green >= 0 && color.Green <= 255)
        targetColor.Green = color.Green;
    else
        return false;

    // Check if the color is within B parameters(0-255) if not return false for
    check.
    if(color.Blue >= 0 && color.Blue <= 255)
        targetColor.Blue = color.Blue;
    else
        return false;

    // Assign the deviation variables for RGB, if exceeds 255 then it will be
    255, if below 0 it will become 0.
    if(targetColor.Red+colorDeviation > 255) rColorPlus = 255; else rColorPlus =
    targetColor.Red+colorDeviation;
    if(targetColor.Red-colorDeviation < 0) rColorMinus = 0; else rColorMinus =
    targetColor.Red-colorDeviation;
    if(targetColor.Green+colorDeviation > 255) gColorPlus = 255; else gColorPlus =
    targetColor.Green+colorDeviation;
    if(targetColor.Green-colorDeviation < 0) gColorMinus = 0; else gColorMinus =
    targetColor.Green-colorDeviation;
}

```

```

    if(targetColor.Blue+colorDeviation > 255) bColorPlus = 255; else bColorPlus
= targetColor.Blue+colorDeviation;
    if(targetColor.Blue-colorDeviation < 0) bColorMinus = 0; else bColorMinus =
targetColor.Blue-colorDeviation;

    return true; // All RGB seems fine, and deviation values assigned, return
true for check.
}

void Auto::scan()
{
    // Initiate scope variables.
    int seekX = 0, seekY = 0, retraceX = 0, retraceY = 0;
    bool reverseX = false, reverseY = false, yMove = false, xTurn = false,
laserSet = false;

    // Continue continously till aborted by thread.
    while(1)
    {
        //-----No longer should be running-----//
        if(autoRunning == false) // Stopping
            break;
        //-----//
        // While theres no target on frame.
        while(!target())
        {
            //-----No longer should be running-----//
            if(autoRunning == false) // Stopping
                break;
            //-----//
            // If the internal tracking of the y axis motor has hit
highest or lowest coordinate(internal coordinate), set/unset bool to make it go
reverse or forward.
            if(seekY == maxMinY)
                reverseY = true; // Go down.
            else if(seekY == -maxMinY)
                reverseY = false; // Go up.

            // If y axis is going up and y hasnt just already moved and
x axis is at rightmost or leftmost position.
            if(xTurn == false && reverseY == false && (seekX == -maxMinX
|| seekX == maxMinX)) {
                pcuPtr->seek(0,perIntervalPixelY); // Move y to
next position, up.
                seekY = seekY+perIntervalPixelY; // Add interval
to internal tracking variable..
                yMove = true;
// Set yMove true, so it cannot be moved again before x has moved with atleast
on target() between movements.
            }
            else if(xTurn == false && reverseY == true && (seekX == -
maxMinX || seekX == maxMinX)) { // If y is going down
                pcuPtr->seek(0,-perIntervalPixelY); // Move y to
next position, down.
                seekY = seekY-perIntervalPixelY; // Subtract
moved interval to internal tracking variable
                yMove = true;
// Set yMove true, so it cannot be moved again before x has moved with atleast
on target() between movements.
            }

            // If the internal tracking of the x motors has hit
rightmost or leftmost coordinate, set/unset bool to make it go right or left.
            if(seekX == maxMinX)
                reverseX = true; // Go left.
            else if(seekX == -maxMinX)
                reverseX = false; // Go right.

```

```

        // If x is going right and y doesnt need to move by itself.
        if(reverseX == false && yMove == false) {
            pcuPtr->seek(perIntervalPixelX,0); // Move x to
next position, right.
            seekX = seekX+perIntervalPixelX; // Add to
internal tracking variable.
            xTurn = false;
// Set xTurn to false in case y had to make a move before.
        }
        else if(reverseX == true && yMove == false) { // If x is
going left and y doesnt need to move by itself.
            pcuPtr->seek(-perIntervalPixelX,0); // Move x to
next position, left.
            seekX = seekX-perIntervalPixelX; // Subtract
interval from internal tracking coordinate.
            xTurn = false;
// Set xTurn to false in case y had to make a move before.
        }

        // If y axis has moved, set xTurn, so y axis cant move on
next scan, but x can(target will be executed between scans).
        if(yMove == 1){
            xTurn = true; // Disallow y to turn.
            yMove = false; // Allow x to turn.
        }

        // Calculate the x-y coordinates from origo. (coords given from
target are from upper left screen).
        // These coords are to be saved in variables, so last scan spot
can be found again from target location.
        retraceX = targetLockX-(frame->width / 2);
        retraceY = (frame->height / 2)-targetLockY;
        pcuPtr->seek(retraceX,retraceY); // Move to
target locations.

        // While there is a target on screen.
        while(target())
        {
            //-----No longer should be running-----//
            if(autoRunning == false) // Stopping
                break;
            //-----//
            // If the laser is off and current target location is
within our reticle quadrant, start laser.
            if(laserSet == false && (targetLockX < (frame->width /
2) + areaPixelReticle) && (targetLockX > (frame->width / 2) -
areaPixelReticle) && (targetLockY < (frame->height / 2) + areaPixelReticle) && (targetLockY >
(frame->height / 2) - areaPixelReticle))
            {
                pcuPtr->laserOn(); // Laser on.
                laserSet = true; // Variable to make sure
serial aint overloaded with commands.
            }
            else if(laserSet == true && !((targetLockX < (frame-
>width / 2) + areaPixelReticle) && (targetLockX > (frame->width / 2) -
areaPixelReticle) && (targetLockY < (frame->height / 2) + areaPixelReticle) &&
(targetLockY > (frame->height / 2) - areaPixelReticle)))
            {
                pcuPtr->laserOff(); // Laser Off.
                laserSet = false;
            }

            // Increment/decrement x axis to get into better
precision position of target.
            if(targetLockX > (frame->width / 2) + areaPixelReticle){
                pcuPtr->seek(xyIncrease,0);
                retraceX = retraceX + xyIncrease;
            }

```

```

        else if(targetLockX < (frame->width / 2) -
areaPixelReticle){
            pcuPtr->seek(-xyIncrease,0);
            retraceX = retraceX - xyIncrease;
        }

        // Increment/decrement y axis to get into better
precision position of target.
        if(targetLockY > (frame->height / 2) + areaPixelReticle)
{
            pcuPtr->seek(0,-xyIncrease);
            retraceY = retraceY - xyIncrease; // Continue to
track movement from last known scan spot.
        }
        else if(targetLockY < (frame->height / 2) -
areaPixelReticle){
            pcuPtr->seek(0,xyIncrease);
            retraceY = retraceY + xyIncrease; // Continue to
track movement from last known scan spot.
        }
    }
    // Update log
    QDate date = QDate::currentDate();
    QTime time = QTime::currentTime();
    int day, month, year;
    QString strHour, strMinutes, strSeconds;
    day = date.day();
    month = date.month();
    year = date.year();
    if(time.hour() < 10)
        QTextStream(&strHour) << '0' << time.hour();
    else
        QTextStream(&strHour) << time.hour();
    if(time.minute() < 10)
        QTextStream(&strMinutes) << '0' << time.minute();
    else
        QTextStream(&strMinutes) << time.minute();
    if(time.second() < 10)
        QTextStream(&strSeconds) << '0' << time.second();
    else
        QTextStream(&strSeconds) << time.second();
    QString finalMessage;
    QTextStream(&finalMessage) << day << '/' << month << '/' << year
<< " - " << strHour << ':' << strMinutes << ':' << strSeconds << ": " <<
"Target with color:" << targetColor.Red << ' ' << targetColor.Green << ' ' <<
targetColor.Blue << " shot";
    emit updateAutoLog(finalMessage);

    // Making sure laser will be put off, if its on after target
    hunting.
    if(laserSet == true)
    {
        pcuPtr->laserOff();
        laserSet = false;
    }

    // Retrace back to last scan spot.
    pcuPtr->seek(-retraceX, -retraceY);
    retraceX = 0; retraceY = 0; // Reset trace variables for next
"target hunting".
    }

}

bool Auto::target()

```

```

{
    // Initiate scope variables.
    uchar * ptr;
    int colorSet = 0;
    int current = 0, total = 0, prevcurrent = 0, prevtotal = 0;

    // For loop for running through all y axis frame pixels.
    for (int yLoop = 0; yLoop < frame->height; yLoop++)
    {
        current = 0; // Finds how many pixels after eachother are within
value.
        total = 0; // Is used to add up all x values where pixel is within
RGB deviation.

        // For loop for running through all x axis frame pixels per y frame
pixel.
        for (int xLoop = 0; xLoop < frame->width; xLoop++)
        {
            ptr = cvPtr2D(frame, yLoop, xLoop, NULL); // Return pointer
to array element giving RGB numbers for the xy pixel.

            // If the xy pixel in frame currently being investigated is
within color parameters, count up current, and add x value to total.
            if(((int)ptr[2] <= rColorPlus && (int)ptr[2] >= rColorMinus)
&& ((int)ptr[1] <= gColorPlus && (int)ptr[1] >= gColorMinus) && ((int)ptr[0] <=
bColorPlus && (int)ptr[0] >= bColorMinus))
            {
                colorSet = 1;
                current = current+1;
                total = total+xLoop;
            }
            // If there has been a pixel within deviation, and next is
not within value, break out of x loop.
            else if((colorSet == 1) && !(((int)ptr[2] <= rColorPlus &&
(int)ptr[2] >= rColorMinus) && ((int)ptr[1] <= gColorPlus && (int)ptr[1] >=
gColorMinus) && ((int)ptr[0] <= bColorPlus && (int)ptr[0] >= bColorMinus)))
            {
                break;
            }
        }

        colorSet = 0; // Reset colorSet to 0, since no more pixels within
value.

        // If the new summation of lenght of color within value is bigger
than the old y axis frame pixel.
        if(current > prevcurrent)
        {
            prevcurrent = current; // Put new/bigger value into
placeholder.
            prevtotal = total; // Put new summation of x axis
values for color within values into placeholder.
            targetLockY = yLoop; // Put the value of Y into
targetLockY, to save value of y coordinate where target is biggest and leftmost.
        }
    }

    if(prevcurrent > 0)
        targetLockX = prevtotal/prevcurrent; // Calculate average of x,
hereby finding the mid point of the target.

    // If our summation number within value of color defined, are longer than a
pre defined lenght need to define it as a target, return true else false.
    if(prevcurrent >= validAmountPixelTarget)
        return true;
    else
        return false;
}
void Auto::run()

```

```
{  
    scan(); // Start the scanning cycle.  
}
```

```
#ifndef FEJLPOPUP_H
#define FEJLPOPUP_H
// File: FejlPopup.h
/*
    Description: Class for displaying error message for wrong color.(Header)
    Author: Group 4
    Date: 26/05 -11
*/

#include <QMessageBox>

class FejlPopup : public QMessageBox
{
    Q_OBJECT
public:
    explicit FejlPopup(QWidget *parent = 0);

signals:

public slots:

};

#endif // FEJLPOPUP_H
```

```
// File: FejlPopup.cpp
/*
    Description: Class for displaying error message for wrong color.
(Implementation)
    Author: Group 4
    Date: 26/05 -11
*/
#include "FejlPopup.h"

FejlPopup::FejlPopup(QWidget *parent) : QMessageBox(parent)
{
    // Settings
    this->setWindowTitle("Error");
    this->setText("Invalid color value");
}
```



```

/*****
 *
 ** Form generated from reading UI file 'mainwindow.ui'
 **
 ** Created: Thu 26. May 17:37:33 2011
 **    by: Qt User Interface Compiler version 4.7.0
 **
 ** WARNING! All changes made in this file will be lost when recompiling UI file!
 *****/

#ifndef UI_MAINWINDOW_H
#define UI_MAINWINDOW_H

#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QGridLayout>
#include <QtGui/QGroupBox>
#include <QtGui/QHBoxLayout>
#include <QtGui/QHeaderView>
#include <QtGui/QLineEdit>
#include <QtGui/QMainWindow>
#include <QtGui/QPushButton>
#include <QtGui/QTextEdit>
#include <QtGui/QVBoxLayout>
#include <QtGui/QWidget>

QT_BEGIN_NAMESPACE

class Ui_MainWindow
{
public:
    QWidget *centralWidget;
    QVBoxLayout *verticalLayout_2;
    QVBoxLayout *verticalLayout;
    QHBoxLayout *horizontalLayout_2;
    QGroupBox *groupBox_2;
    QGroupBox *groupBox;
    QGridLayout *gridLayout_2;
    QGridLayout *gridLayout;
    QTextEdit *txtLog;
    QHBoxLayout *horizontalLayout;
    QGroupBox *groupBox_4;
    QVBoxLayout *verticalLayout_4;
    QVBoxLayout *verticalLayout_3;
    QPushButton *btnAuto;
    QPushButton *btnManuel;
    QGroupBox *groupBox_3;
    QHBoxLayout *horizontalLayout_4;
    QHBoxLayout *horizontalLayout_3;
    QLineEdit *txtSend;
    QPushButton *btnSend;

    void setupUi(QMainWindow *MainWindow)
    {
        if (MainWindow->objectName().isEmpty())
            MainWindow->setObjectName(QString::fromUtf8("MainWindow"));
        MainWindow->setWindowModality(Qt::NonModal);
        MainWindow->resize(798, 577);
        MainWindow->setMaximumSize(QSize(1000, 1000));
        QIcon icon;
        icon.addFile(QString::fromUtf8("iconApp.ico"), QSize(), QIcon::Normal,
QIcon::Off);
        MainWindow->setWindowIcon(icon);
        MainWindow->setWindowOpacity(1);
        MainWindow->setIconSize(QSize(32, 32));
        centralWidget = new QWidget(MainWindow);

```

```

centralWidget->setObjectName(QString::fromUtf8("centralWidget"));
verticalLayout_2 = new QVBoxLayout(centralWidget);
verticalLayout_2->setSpacing(6);
verticalLayout_2->setContentsMargins(11, 11, 11, 11);
verticalLayout_2->setObjectName(QString::fromUtf8("verticalLayout_2"));
verticalLayout = new QVBoxLayout();
verticalLayout->setSpacing(6);
verticalLayout->setObjectName(QString::fromUtf8("verticalLayout"));
horizontalLayout_2 = new QHBoxLayout();
horizontalLayout_2->setSpacing(6);
horizontalLayout_2-
>setObjectName(QString::fromUtf8("horizontalLayout_2"));
groupBox_2 = new QGroupBox(centralWidget);
groupBox_2->setObjectName(QString::fromUtf8("groupBox_2"));
groupBox_2->setMinimumSize(QSize(610, 440));
groupBox_2->setFlat(false);

horizontalLayout_2->addWidget(groupBox_2);

groupBox = new QGroupBox(centralWidget);
groupBox->setObjectName(QString::fromUtf8("groupBox"));
gridLayout_2 = new QGridLayout(groupBox);
gridLayout_2->setSpacing(6);
gridLayout_2->setContentsMargins(11, 11, 11, 11);
gridLayout_2->setObjectName(QString::fromUtf8("gridLayout_2"));
gridLayout = new QGridLayout();
gridLayout->setSpacing(6);
gridLayout->setObjectName(QString::fromUtf8("gridLayout"));
txtLog = new QTextEdit(groupBox);
txtLog->setObjectName(QString::fromUtf8("txtLog"));
txtLog->setReadOnly(true);

gridLayout->addWidget(txtLog, 0, 0, 1, 1);

gridLayout_2->addLayout(gridLayout, 0, 0, 1, 1);

horizontalLayout_2->addWidget(groupBox);

verticalLayout->addLayout(horizontalLayout_2);

horizontalLayout = new QHBoxLayout();
horizontalLayout->setSpacing(6);
horizontalLayout->setObjectName(QString::fromUtf8("horizontalLayout"));
groupBox_4 = new QGroupBox(centralWidget);
groupBox_4->setObjectName(QString::fromUtf8("groupBox_4"));
verticalLayout_4 = new QVBoxLayout(groupBox_4);
verticalLayout_4->setSpacing(6);
verticalLayout_4->setContentsMargins(11, 11, 11, 11);
verticalLayout_4->setObjectName(QString::fromUtf8("verticalLayout_4"));
verticalLayout_3 = new QVBoxLayout();
verticalLayout_3->setSpacing(6);
verticalLayout_3->setObjectName(QString::fromUtf8("verticalLayout_3"));
btnAuto = new QPushButton(groupBox_4);
btnAuto->setObjectName(QString::fromUtf8("btnAuto"));
btnAuto->setEnabled(true);

verticalLayout_3->addWidget(btnAuto);

btnManuel = new QPushButton(groupBox_4);
btnManuel->setObjectName(QString::fromUtf8("btnManuel"));
btnManuel->setEnabled(false);

verticalLayout_3->addWidget(btnManuel);

verticalLayout_4->addLayout(verticalLayout_3);

```

```

horizontalLayout->addWidget (groupBox_4);

groupBox_3 = new QGroupBox (centralWidget);
groupBox_3->setObjectName (QString::fromUtf8 ("groupBox_3"));
horizontalLayout_4 = new QHBoxLayout (groupBox_3);
horizontalLayout_4->setSpacing (6);
horizontalLayout_4->setContentsMargins (11, 11, 11, 11);
horizontalLayout_4-
>setObjectName (QString::fromUtf8 ("horizontalLayout_4"));
horizontalLayout_3 = new QHBoxLayout ();
horizontalLayout_3->setSpacing (6);
horizontalLayout_3-
>setObjectName (QString::fromUtf8 ("horizontalLayout_3"));
txtSend = new QLineEdit (groupBox_3);
txtSend->setObjectName (QString::fromUtf8 ("txtSend"));

horizontalLayout_3->addWidget (txtSend);

btnSend = new QPushButton (groupBox_3);
btnSend->setObjectName (QString::fromUtf8 ("btnSend"));

horizontalLayout_3->addWidget (btnSend);

horizontalLayout_4->addLayout (horizontalLayout_3);

horizontalLayout->addWidget (groupBox_3);

verticalLayout->addLayout (horizontalLayout);

verticalLayout_2->addLayout (verticalLayout);

MainWindow->setCentralWidget (centralWidget);

retranslateUi (MainWindow);

QMetaObject::connectSlotsByName (MainWindow);
} // setupUi

void retranslateUi (QMainWindow *MainWindow)
{
    MainWindow->setWindowTitle (QApplication::translate ("MainWindow",
"MainWindow", 0, QApplication::UnicodeUTF8));
    groupBox_2->setTitle (QApplication::translate ("MainWindow", "Live feed:",
0, QApplication::UnicodeUTF8));
    groupBox->setTitle (QApplication::translate ("MainWindow", "Log:", 0,
QApplication::UnicodeUTF8));
    groupBox_4->setTitle (QApplication::translate ("MainWindow", "Choice:", 0,
QApplication::UnicodeUTF8));
    btnAuto->setText (QApplication::translate ("MainWindow", "Auto", 0,
QApplication::UnicodeUTF8));
    btnManuel->setText (QApplication::translate ("MainWindow", "Manuel", 0,
QApplication::UnicodeUTF8));
    groupBox_3->setTitle (QApplication::translate ("MainWindow", "Auto:", 0,
QApplication::UnicodeUTF8));
    txtSend->setInputMask (QApplication::translate ("MainWindow", "HH-HH-HH;
", 0, QApplication::UnicodeUTF8));
    btnSend->setText (QApplication::translate ("MainWindow", "send", 0,
QApplication::UnicodeUTF8));
} // retranslateUi

};

namespace Ui {

```

```
    class MainWindow: public Ui_MainWindow {};  
} // namespace Ui  
  
QT_END_NAMESPACE  
  
#endif // UI_MAINWINDOW_H
```

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H
// File: mainwindow.h
/*
    Description: The GUI class. (Some of the functions are under the class
    'MainWindow' as 'slots' and 'signals')
                Also contains 'Manuel'´s function 'checkPressedKey' as a
    keyEvents.
                (Header)
    Author: Group 4
    Date: 26/05 -11
*/

#include <QMainWindow>
#include <qthread.h>
#include <QObject>
#include "cv.h"
#include "highgui.h"
#include <QtGui/QFrame>
#include <QtGui/QGroupBox>
#include "PCcontroller.h"
#include "Auto.h"
#include "FejlPopup.h"
#include <QKeyEvent>

// Functions
QImage IplImage2QImage(const IplImage *iplImage); // Convert OpenCV image to Qt
image

namespace Ui {
    class MainWindow;
}

// Classes
// -Main window
class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0); // Device is index for cam
    ~MainWindow();

public slots:
    void manuelSelected();
    void autoSelected();
    void turnOffManuel();
    void autoProcess(); // Called when 'send' is pressed

signals:
    void updateLog(QString message);

private:
    Ui::MainWindow *ui;

protected:
    // Events for the function: "checkKeyPressed()"
    void keyPressEvent(QKeyEvent* event);
};

// -Webcam feed frame
class VideoFrame : public QFrame
{
public:
    VideoFrame(QGroupBox*); // Groupbox is the parent
    ~VideoFrame();

protected:

```

```
void paintEvent(QPaintEvent* event);

private:
    CvCapture* capture; // For webcam feed
};

// -Threads
class ThreadCam : public QThread // Updates the "Cam feed"
{
    Q_OBJECT // Needed for signals and slots
public:
    void run();

signals:
    void updateCam();
};

#endif // MAINWINDOW_H
```

```

// File: mainwindow.cpp
/*
    Description: The GUI class.(Some of the functions are under the class
'MainWindow' as 'slots' and 'signals')
                Also contains 'Manuel'’s function 'checkPressedKey' as a
keyEvents.
                (Implementation)
    Author: Group 4
    Date: 26/05 -11
*/

#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "cv.h"
#include "highgui.h"
#include "qpainter.h"
#include <QTime>
#include <QDate>
#include <QTextStream>
#include <cstdlib> // Used in hex conversion

// Global variables
IplImage* theImage; // Image used for video feed and for scanning
PCcontroller thePCcontroller;
Auto* theAuto;
bool manuelRunning = true;

// Functions
// -To convert OpenCV images into Qt images
QImage IplImage2QImage(const IplImage *iplImage)
{
    int height = iplImage->height;
    int width = iplImage->width;

    if (iplImage->depth == IPL_DEPTH_8U && iplImage->nChannels == 3)
    {
        const uchar *qImageBuffer = (const uchar*)iplImage->imageData;
        QImage img(qImageBuffer, width, height, QImage::Format_RGB888);
        return img.rgbSwapped();
    } else if (iplImage->depth == IPL_DEPTH_8U && iplImage->nChannels == 1){
        const uchar *qImageBuffer = (const uchar*)iplImage->imageData;
        QImage img(qImageBuffer, width, height, QImage::Format_Indexed8);

        QVector<QRgb> colorTable;
        for (int i = 0; i < 256; i++){
            colorTable.push_back(qRgb(i, i, i));
        }
        img.setColorTable(colorTable);
        return img;
    }
    return QImage();
}

// -MainWindow functions
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    // Variables
    // -Video
    ThreadCam* mThread = new ThreadCam(); // WARNING: Dynamic
    VideoFrame* mA = new VideoFrame(ui->groupBox_2);

    // Signals connect
    connect(ui->btnAuto, SIGNAL(clicked()), this, SLOT(autoSelected()));
    connect(ui->btnManuel, SIGNAL(clicked()), this, SLOT(manuelSelected()));
}

```

```

connect(mThread, SIGNAL(updateCam()), mA, SLOT(update()));
connect(this, SIGNAL(updateLog(QString)), ui->txtLog,
SLOT(append(QString)));
connect(ui->btnSend, SIGNAL(clicked()), this, SLOT(autoProcess()));

// Start up
// -Main window
this->move(0, 0);
this->setFocusPolicy(Qt::StrongFocus);
this->setFocus(Qt::ActiveWindowFocusReason);

// -Disable button for manuel and disable auto related buttons
ui->btnManuel->setEnabled(false);
ui->btnSend->setEnabled(false);
ui->txtSend->setEnabled(false);

// -Video feed
mA->setGeometry(20, 20, 580, 420);
mThread->start();
mThread->setPriority(QThread::LowestPriority);

// -PC->pcontroller
#warning Set to static values
//thePCcontroller.open(3, 9600); // Port 3 with baud rate of 9600
bool connected = false;
for(int i = 0; i < 9; i++) // Checks 9 ports
{
    if(connected = thePCcontroller.open(i, 9600)) // Connected
        break;
}
if(connected == false)
    this->setWindowTitle("The system is not connected");

// -Auto
theAuto = new Auto(&thePCcontroller, theImage);
theAuto->autoRunning = false;
connect(theAuto, SIGNAL(updateAutoLog(QString)), ui->txtLog,
SLOT(append(QString))); // For log update
}

MainWindow::~MainWindow()
{
    delete ui;
    delete theAuto;

    // Turn off laser
    thePCcontroller.laserOff();

    // Turn off connection
    thePCcontroller.close();
}

// -MainWindow slots(GUI functions)
void MainWindow::manuelSelected()
{
    // Status
    manuelRunning = true;
    theAuto->autoRunning = false; // To stop the thread

    // Disable button for manuel and disable auto related buttons
    ui->btnManuel->setEnabled(false);
    ui->btnSend->setEnabled(false);
    ui->txtSend->setEnabled(false);

    // Enable auto button
    ui->btnAuto->setEnabled(true);

    // Log
    QDate date = QDate::currentDate();

```



```

    QTime time = QTime::currentTime();
    int day, month, year;
    QString strHour, strMinutes, strSeconds;
    day = date.day();
    month = date.month();
    year = date.year();
    if(time.hour() < 10)
        QTextStream(&strHour) << '0' << time.hour();
    else
        QTextStream(&strHour) << time.hour();
    if(time.minute() < 10)
        QTextStream(&strMinutes) << '0' << time.minute();
    else
        QTextStream(&strMinutes) << time.minute();
    if(time.second() < 10)
        QTextStream(&strSeconds) << '0' << time.second();
    else
        QTextStream(&strSeconds) << time.second();
    QString finalMessage;
    QTextStream(&finalMessage) << day << '/' << month << '/' << year << " - " <<
strHour << ':' << strMinutes << ':' << strSeconds << ": " << "Manuel selected";

    emit updateLog(finalMessage);
}
void MainWindow::autoSelected()
{
    // Disable button for auto and enable auto related buttons
    ui->btnAuto->setEnabled(false);
    ui->btnSend->setEnabled(true);
    ui->txtSend->setEnabled(true);

    // Enable manuel button
    ui->btnManuel->setEnabled(true);

    // Disable manuel steering
    turnOffManuel();

    // Log
    QDate date = QDate::currentDate();
    QTime time = QTime::currentTime();
    int day, month, year;
    QString strHour, strMinutes, strSeconds;
    day = date.day();
    month = date.month();
    year = date.year();
    if(time.hour() < 10)
        QTextStream(&strHour) << '0' << time.hour();
    else
        QTextStream(&strHour) << time.hour();
    if(time.minute() < 10)
        QTextStream(&strMinutes) << '0' << time.minute();
    else
        QTextStream(&strMinutes) << time.minute();
    if(time.second() < 10)
        QTextStream(&strSeconds) << '0' << time.second();
    else
        QTextStream(&strSeconds) << time.second();
    QString finalMessage;
    QTextStream(&finalMessage) << day << '/' << month << '/' << year << " - " <<
strHour << ':' << strMinutes << ':' << strSeconds << ": " << "Auto selected";

    emit updateLog(finalMessage);
}
void MainWindow::autoProcess()
{
    // Converts text from the textbox into integer values(base 10)
    struct RGB tmpRGB; // To contain final color
    char strColors[9]; // To contain the text from the textbox. ('buffer')
    std::string tmpString = ui->txtSend->text().toStdString(); // Used for

```

```

temporary conversion from QString to std::string
memcpy(strColors, tmpString.c_str(), 9); // Copies the text string over into
the color 'buffer'

if(strlen(tmpString.c_str()) == 8) // Checking for blank characters
{
    // -Get the RGB values one by one
    char* colorP = 0; // To point to color tokens
    colorP = strtok(strColors, "-"); // Split into first token
    char * p;
    long colorBase10 = strtoul( colorP, & p, 16 ); // Convert from hex to
base 10
    tmpRGB.Red = colorBase10;
    colorP = strtok(NULL, "-"); // Next token
    colorBase10 = strtoul( colorP, & p, 16 );
    tmpRGB.Green = colorBase10;
    colorP = strtok(NULL, "-"); // Next token
    colorBase10 = strtoul( colorP, & p, 16 );
    tmpRGB.Blue = colorBase10;
    if(theAuto->checkValid(tmpRGB))
    {
        // Start auto thread
        if(theAuto->autoRunning != true) // So no multi threads opening
        {
            theAuto->start();
            theAuto->setPriority(QThread::LowestPriority);
        }
        theAuto->autoRunning = true;
    }
    else
    {
        // Error popup
        FejlPopup fejlPopup(this);
        fejlPopup.exec();
    }
}
else
{
    // Error popup
    FejlPopup fejlPopup(this);
    fejlPopup.exec();
}
}

void MainWindow::turnOffManuel()
{
    manuelRunning = false;
}

void MainWindow::keyPressEvent(QKeyEvent *event)
{
    static bool laserOn = false; // Used for toggle
    if(manuelRunning == true)
    {
        switch (event->key())
        {
            {
                case Qt::Key_Left:
                    thePCcontroller.turnLeft();
                    break;
                case Qt::Key_Right:
                    thePCcontroller.turnRight();
                    break;
                case Qt::Key_Up:
                    thePCcontroller.turnUp();
                    break;
                case Qt::Key_Down:
                    thePCcontroller.turnDown();
                    break;
                case Qt::Key_Space:

```

```

        if(laserOn == false)
        {
            thePCcontroller.laserOn();
        }
        else
            thePCcontroller.laserOff();
        laserOn = !laserOn;
        break;
    }
}

// VideoFrame functions
VideoFrame::VideoFrame(QGroupBox* g) : QFrame(g)
{
    // capture from video device #0
    capture = cvCaptureFromCAM(0);

    // Additional settings
    cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, 800); // Only some
    predefined sizes seem to work
    cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, 600); // Only some
    predefined sizes seem to work
}
VideoFrame::~~VideoFrame()
{
    // Release capture source
    cvReleaseCapture(&capture);
}

void VideoFrame::paintEvent(QPaintEvent* event)
{
    // Variables
    QFrame::paintEvent(event);
    QImage imgA;
    QPainter* painter = new QPainter();

    // Image capturing and drawing
    painter->begin(this);

    if(capture) // If 'capture' initialised
    {
        cvGrabFrame(capture);
        theImage = cvRetrieveFrame(capture);
    }

    if(theImage != NULL) // Convert if image exists
        imgA = IplImage2QImage(theImage);

    painter->drawImage(0, 0, imgA);
    painter->setPen(QColor(255, 0, 0));
    painter->drawEllipse(this->width()/2, this->height()/2, 10, 10);
    painter->drawEllipse(this->width()/2 - 5, this->height()/2 - 5, 20, 20);

    painter->end(); // close the painter
}

// Threads
void ThreadCam::run()
{
    while(1)
    {
        emit updateCam();
        this->msleep(10);
    }
}

```

```
// File: main.cpp
/*
    Description: Driver class for the system.
    Author: Group 4
    Date: 26/05 -11
*/
#include <QtGui/QApplication>
#include "mainwindow.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```