EasyCopter

Christoph Breisacher
Oliver Suchan

Seminarkurs-Robotik

Klasse 12 – Kurs 2

1 Inhaltsverzeichnis

1 Inhaltsverzeichnis 1
2 Zielsetzung 2
2.1 Verbindliche Ziele 2
2.2 Optionale Ziele 3
2.3 Neu hinzugekommene Ziele (BETA!) 3
3 AR.Drone 2.0 5
4 Software von Drittanbietern 6
4.1 ROS 6
4.2 Gazebo 7
4.3 Ubuntu 12.04 8
4.4 OpenCV 8
4.5 Qt 9
5 Informationen zum Quellcode 9
5.1 Gesichtserkennung 10
5.2 Ungarische Notation 11
5.3 Globals-Klasse 12
6 Entwicklung des Quellcodes 14
7 Finale Version 16
8 Quellen 20
8.1 EasyLogic – Dokumentation 20
8.2 Qt 5.2 20
8.3 ROS-Fuerte 20
8.4 TUM-Simulator 20
8.5 TUM-ARDrone 20
8.6 qt_ros 20
8.7 Ubuntu 12.04 20
8.8 GitHub 20

2 Zielsetzung

Unsere Ziele sind in zwei Gruppen aufzuteilen.
In der einen Hand gibt es die optionalen Ziele und in der anderen Hand die verbindlichen Ziele.
Ursprünglich war unsere Liste der Ziele eine andere, wie sie in den Meilensteinen zu sehen ist, da wir aber eine gute Idee hatten und diese unbedingt umsetzen wollten, sei es nun ein verbindliches oder optionales Ziel, fügten wir diese Idee(n) zu unserer List von Ziele hinzu.
Die zuvor gesetzten verbindlichen Ziele haben wir alle erreicht und auch die neu hinzugekommenen verbindlichen, beziehungsweise optionalen Ziele.
Leider konnten wir uns dadurch nicht mehr so stark auf unsere zuvor festgelegten optionalen Ziele festlegen.

Unsere Liste sieht nun wie folgt aus:

2.1 Verbindliche Ziele

- ☑ Installation/Einrichtung von ROS auf PC und Drohne
- ☑ Simulation & Steuerung der Drohne am Computer
- ☑ Steuerung der realen AR Drohne mit bestehender Software
- ☑ Ansteuern der Drohne mit eigener Software via ROS
- ☑ Testweise Auswertung der vorliegenden Sensordaten
- ☑ Automatische Verbindung zur Drohne/Simulator
- ☑ Steuerung der Drohne per Maus beziehungsweise Tastatur
- ☑ Bereitstellung von Standard Abläufen wie Abheben/Landen
- ☑ Visualisieren der verschiedenen Sensordaten z.B. Kamerabilder

2.2 Optionale Ziele

- [ ] Vordefinierte Flugmanöver durchfliegen
- [ ] Rekorder-Funktion für Flugmanöver

2.3 Neu hinzugekommene Ziele (BETA!)

Wie der Titel bereits sagt, handelt es sich bei unseren neu hinzugekommenen Zielen hauptsächlich um eine Beta-Phase, da wir nicht genügend Zeit hatten, unser Programm zu 100% auf der physikalisch existenten Drohne auszuprobieren.
Nichtsdestotrotz sind wir der Meinung, dass es auch auf der Drohne einwandfrei funktioniert, da wir uns aber nicht sicher sind, befindet es sich auch noch in der Beta-Phase ansonsten hätten wir es als Alpha-Version eingestuft.

Nun, was ist denn jetzt neu hinzugekommen?

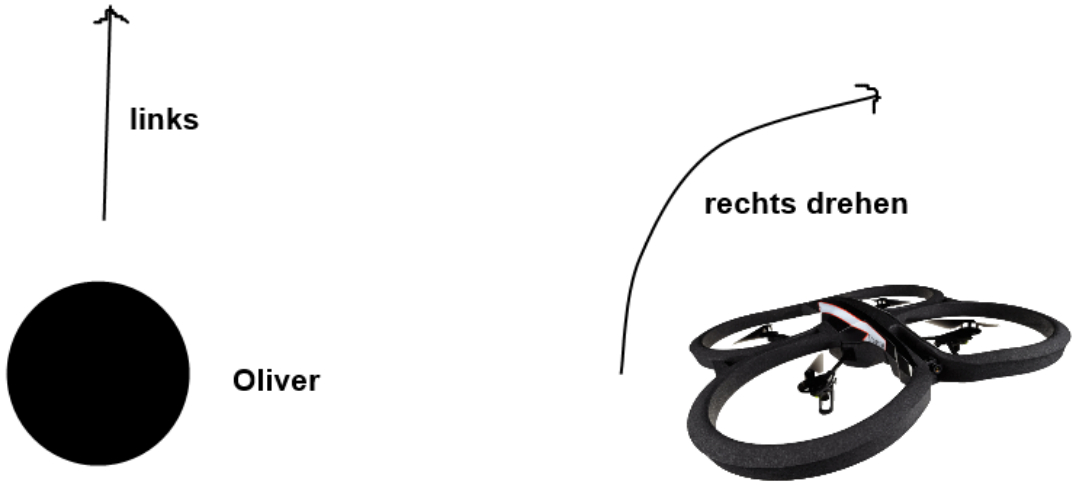
Einerseits hatten wir das Verlangen unser gesamtes Potenzial, im Bezug auf die Kreativität der Programme, die wir für die Drohne entwickelten, auszuschöpfen. Darunter viel auch, die Verarbeitung der Kamerabilder.

Bildverarbeitung ist auch schon das Stichwort.
Wir hatten die Idee, die Drohne Gesichter erkennen zu lassen und wenn Gesichter erkannt werden diese zu verfolgen.
Im späteren Verlauf unseres Programmes, entschlossen wir uns dann, dass alle Gesichter, die von der Drohne erkannt werden, im Arbeitsspeicher abgespeichert werden. Werden die gleichen Gesichter erneut erkannt, werden sie nicht noch einmal abgespeichert. Aus der Liste der gespeicherten und detektieren Gesichter, konnte man nun ein, oder mehrere Gesichter auswählen, die dann von der Drohne verfolgt werden sollten. Das heißt so viel wie:

Wenn Oliver nun der Drohne gegenüberstünde, sprich, mit dem Gesicht zur Frontkamera der Drohne gerichtet, so würde unsere Software, aus den Bildern der Drohne, die an unsere Software übermittelt werden, sein Gesicht

erkennen, und in dem Ausgabefenster, sein Gesicht hervorheben – sein Gesicht und wenn möglich seine Augen, werden eingekreist.
Abgesehen davon, wird das hoffentlich erkannte Gesicht, im Arbeitsspeicher abgespeichert und als Auswahloption angezeigt. Das heißt so viel wie, dass das Gesicht zu einer Liste hinzugefügt wird, in der sich alle bisher – ab Programmstart – erkannten Gesichter befinden.

Wählt man nun sein Gesicht aus und aktiviert die Gesichtsverfolgung, so versucht die Drohne stets das Gesicht im Mittelpunkt der Drohnenkamera zu halten. Das heißt so viel wie, bewegt sich Oliver nach rechts, muss sich die Drohne nach links drehen. Bewegt sich Oliver nach unten, bewegt sich die Drohne ebenfalls nach unten usw.



VOGELPERSPEKTIVE

Des Weiteren ist die Drohne befähigt, den Abstand zum Gesicht zu regulieren. Dabei verwenden wir jedoch nur ungefähre Werte:
Wir wissen wie groß ungefähr ein Gesicht ist. Anhand dieses Wissens „kalibrieren“ wir unser Programm – wir benutzen die Größe des Gesichts, aus einer Entfernung von 1m, in Pixel auf dem Kameraframe. Aus diesen Angaben berechnen wir ein Skalar, das uns Aufschluss über die Distanz des Gesichtes zur Kamera gibt. Wie bereits gesagt, ist das nur ein ungefährer Wert!

Nun zurück zur Abstandsregulierung. Ist das Gesicht näher als oder genau 20cm zur Frontkamera der Drohne entfernt, legt die Drohne den Rückwärtsgang ein. Ist das Gesicht mehr als oder genau 50cm von der Frontkamera entfernt, so bewegt sich die Drohne nach vorne, dem Gesicht entgegen. Dabei muss man lediglich hoffen, dass die Drohne keine terroristischen Gedanken besitzt und, exemplarisch Sebastian, nicht ins Gesicht fliegt.

Aus obigen Angaben ergeben sich nun folgende hinzugekommene Ziele:

- ☑ Detektion von Gesichtern (und deren Hervorhebung)
- ☑ Verfolgung von Gesichtern
- ☑ Temporäre Aufbewahrung der detektierten Gesichter
- ☑ Verfolgung speziell ausgewählter Gesichter

3 AR.Drone 2.0

AR.Drone 2.0 lautet der Name des Meisterwerks, das wir programmieren durften. Bei der AR.Drone 2.0 handelt es sich wie der Name schon sagt, um eine Drohne. DOCH HALT! Es ist keine Drohne, mit der wir jeden einzelnen Schüler dieser Schule eliminieren könnten, beziehungsweise keine typische US-Drohne, sondern lediglich ein Quadrocopter, den man extern fernsteuern kann. Was heißt hier lediglich?
Es ist einer der graziösesten Dinge, welches wir jemals programmieren durften. Nun, ein Quadrocopter ist ein Polycopter, beziehungsweise eine Art von Helicopter, nur, dass es nicht 1 Rotor, sondern gleich 4 Stück besitzt.

Ein weiterer Vorteil den diese Drohne uns bot, weswegen wir auch unbedingt solch eine Drohne besitzen wollten, war, beziehungsweise ist, dass man diese Drohne nicht manuell zusammenbauen muss, sondern sie direkt im fertigen Zustand von Parrot SA, den Entwicklern dieser Drohne, geliefert wird. Dadurch waren wir von der Aufgabe, uns um die, teilweise doch komplizierte, Hardware zu kümmern, befreit und konnten uns Voll und Ganz auf die Programmierung der Drohne – im Bezug auf Software – konzentrieren.

Die Drohne besitzt einen intern verbauten WiFi-Chip, wodurch es einerseits möglich ist, als externen Betrachter die Drohne zu steuern. Auch ist es möglich, durch ROS – dazu später mehr – die Drohne vordefinierte Flugmanöver fliegen zu lassen.

Als externer Betrachter kann man die Drohne durch bestimmte APPs, die sowohl für Android, iOS und etwaige Betriebssysteme verfügbar sind, steuern. Dabei muss man sich dann mit dem Smartphone zu der Drohne über WiFi verbinden, um ihr so die Befehle mitzuteilen. Das heißt natürlich, ist die Drohne außer Reichweite des WiFi-Netzwerkes, so ist sie voll und ganz auf sich gestellt…
Zum Glück hatten wir noch nie mit solchen Ernstsituationen zu kämpfen. (Hoffentlich bleibt das auch so…)

4 Software von Drittanbietern

4.1 ROS

ROS steht für Robot Operating System und ist ein Framework, das zur Kommunikation zwischen Robotern und PC dient. Im normalen Auslieferungszustand, ist die Drohne noch nicht per ROS steuerbar, doch dafür gibt es bestimmte Entwicklungen, von der technischen Universität München, die genau dies ermöglichen sollen. Dabei handelt es sich sozusagen um ein Paket für ROS, welches ROS um spezielle Funktionen erweitert.
ROS wurde uns durch einen Lehrer – Herr Malenica – erstmals vorgestellt und wir erkannten sofort das Potential dieses Frameworks. Aufgrund dessen entschlossen wir uns letztendlich auch, unsere Programme auf Basis von ROS zu programmieren.

Bei ROS startet man zuerst sozusagen einen lokalen Server – den *roscore*. Mit diesem *roscore* ist es dann möglich, Sensordaten abzufragen oder bereitzustellen. Dafür gibt es die Möglichkeit, sogenannte Nodes zu erzeugen, sich für diese zu bewerben oder diese zu abonnieren, welche für, auf ROS basierenden, Programmen verfügbar sind, wenn sich diese mit dem *roscore* verbinden. Wenn man solch ein Node abonniert, muss man einen Callback

festlegen, der bei Änderungen des Nodes aufgerufen wird. Sprich wenn man zum Beispiel Sensordaten eines Roboters abonniert, wird dieser Callback aufgerufen, wenn die Sensordaten geändert werden. Nun gibt es auch die Möglichkeit sich für bestimmte Nodes zu bewerben, das heißt, man kann selbst Befehle an diese Nodes senden, die dann zum Beispiel die Steuerung der Drohne beziehungsweise des Roboters übernehmen.
Wie bereits gesagt, kann man auch eigene Nodes erzeugen, die dann die obigen Funktionen bereitstellen.

Die Möglichkeiten die man mit ROS machen kann, sind dadurch grenzenlos. Beispielsweise gibt es noch eine Erweiterung, die wir verwendeten, die es ermöglicht, neben ROS, Qt zu verwenden.
Dazu jedoch später mehr.

Dadurch, dass es bei neueren Versionen von ROS oftmals zu Komplikationen mit dem Simulator kommt – ein Simulator in dem man seine Programme auf virtuellen Robotern testen kann –, entschlossen wir uns, eine ältere Version von ROS zu verwenden, die auch empfohlen wird.
Dabei handelt es sich um die sogenannte ROS-Fuerte-Version.
In dieser Version ist obiger Simulator standardmäßig implementiert und muss nicht erst mit viel Not und Mühe, sowie es bei neueren ROS-Versionen der Fall ist, den Quellcode des Simulators kompilieren usw.

Ein weiterer Grund, weswegen wir uns für ROS-Fuerte entschlossen haben, abgesehen davon, dass unser Lehrer – Herr Malenica – uns auch dazu riet.

4.2 Gazebo

Gazebo lautet der oben angesprochene Simulator. Dieser ist in unserer ROS-Version standardmäßig implementiert und erzeugt so auch keine Konflikte. Wie bereits gesagt, kann man mit diesem seine Software in einer virtuellen Welt testen, in der die physikalischen Gesetze aber dennoch gelten sollen. Das war ein riesen Vorteil, da wir ja in 2 Gruppen mit dieser Drohne arbeiteten. Wenn also die eine Gruppe gerade die Drohne brauchte, so konnte die andere Gruppe einfach den Simulator verwenden und man stand sich nicht im Weg.

Des Weiteren ist es mit dieser Software, beziehungsweise RviZ, eine Software die in diesem Softwarepaket enthalten ist, möglich, eigene Umgebungen nach Belieben zu schaffen.

4.3 Ubuntu 12.04

Ubuntu 12.04 ist eine Linux Distribution die wir für ROS zur Verwendung gezogen haben. Dies taten wir aufgrund der schwachen Kompatibilität ROS‘ zu anderen Betriebssystemen. Sprich, wenn man ROS-Fuerte verwendet, wird empfohlen, Ubuntu zu verwenden. Dabei gibt es auch die Auswahl zwischen 12.04 und noch älteren Versionen. Da wir uns aber für die neuere Version entschieden hatten, landete Ubuntu 12.04 auf unseren Rechnern. Und als Fazit kann ich, Oliver, nur sagen, dass ich es nicht bereue, Ubuntu auf meinen Laptop gemacht zu haben und im Gegenzug dafür, Windows „runtergeschmissen“ habe. Zwar bietet Ubuntu nicht die Möglichkeit, Spiele zu spielen, aber dafür hat so ziemlich jeder einen Desktop-PC mit Windows. Abgesehen davon, ist es genau dadurch möglich, auf Linux-Betriebssystemen effizienter zu arbeiten, da man nicht alle paar Minuten das Verlangen hat, ein Spiel zu starten und dies den ganzen Seminarkurs über zu spielen.

4.4 OpenCV

OpenCV ist ein Bildverarbeitungs-Framework, das demnach natürlich viele Möglichkeiten bietet, Bilder zu verarbeiten. Darunter auch die Erkennung von bekannten Objekten, wie zum Beispiel Gesichter, weswegen wir auch das OpenCV-Framework auf unsere Seite der Macht zogen.
Des Weiteren gab es eine Vorstellung von Herrn Malenica über dieses Framework, was natürlich unseren Ideentopf überlaufen ließ und zu unserer finalen Idee führte. Apropos, OpenCV ist standardmäßig in das ROS-Framework integriert.

4.5 Qt

Auch Qt ist ein Framework und ein gefundenes Fressen für jeden Hobby-Programmierer, aber auch für die, die aus schulischen Gründen programmieren müssen.
Nun was für ein Framework genau ist denn Qt?
Qt ist ein Framework, was gemacht wurde, um eine sogenannte cross-platform-abilitiy für jeden Programmierer zu bieten. Sprich, Christoph und Oliver programmieren heiter ein Programm und wollen dieses nun auch für andere Betriebssysteme verfügbar machen, nachdem sie es für Linux verfügbar gemacht haben.
Dann verwenden sie jetzt Qt. Nur für kurze Zeit!

Spaß bei Seite.
Qt kann man sich runterladen wann immer man möchte.

Abgesehen von dieser cross-platform-ability bietet Qt die Möglichkeit, sehr leicht, gute Graphical User Interfaces – GUIs - zu erzeugen, was auch der Grund war, dass wir es benutzten, denn wir wollten ja nicht nur ein gut funktionierendes Programm, sondern ein schön aussehendes, gut funktionierendes Programm.
Eine simple Konsolenanwendung ist da vielleicht nicht das Richtige gewesen, besonders für unsere Zwecke (Gesichtserkennung, Gesichter-Auswahl, usw.).

Die doch so gute cross-platform-ability von Qt, fand in unserer Software keine Anwendung, auch wenn es theoretisch funktionieren sollte – es sei denn ROS UND(!) Qt sind für jenes Betriebssystem nicht verfügbar.

5 Informationen zum Quellcode

Unser Programm wurde komplett in C++11 programmiert, in Verwendung mit Qt 5.2 beziehungsweise Qt 4. Auch wäre es möglich, dass wir unsere Software in Python schreiben, welche auch eine sehr schöne Programmiersprache ist, jedoch ist deren Funktionalität auf ROS bezogen, im Gegensatz zu C++(11), stark eingeschränkt.

C++11 verwendeten wir aufgrund des Threadings in unserer *FlightController*-Klasse.

5.1 Gesichtserkennung

Anhand unserer Gesichtserkennung sind ein paar Eigenheiten von Qt erkennbar. Und zwar die sogenannte Introspektion, beziehungsweise Reflexion und deren *Signals* und *Slots:*

```
globals.hpp:

    void addDetectedFace(cv::Mat p_DetectedFace);

signals:

    void addDetectedFaceSig(cv::Mat p_DetectedFace);


globals.cpp:
void Globals::addDetectedFace(cv::Mat p_DetectedFace)
{
    emit addDetectedFaceSig(p_DetectedFace);
}


wainwindow.hpp:
private slots:
    void onAddDetectedFace(cv::Mat p_DetectedFace);


mainwindow.cpp:
QObject::connect(Globals::getInstance(),
SIGNAL(addDetectedFaceSig(cv::Mat)), this,
SLOT(onAddDetectedFace(cv::Mat)));


imageconverter.cpp:
Globals::getInstance()->addDetectedFace(p_Frame(faces[i]));
```

Was hier genau vor sich geht, ist eventuell nicht direkt erkennbar, weswegen wir uns für eine extra Erwähnung dieser Stelle entschlossen. Und zwar passiert in diesen wenigen Zeilen Code folgendes:

In der *Globals*-Klasse, werden erstmal die Funktionen „*addDetectedFace*“ und „*addDectedFaceSig*“ deklariert, wobei „*addDetectedFaceSig*“ nochmal eine spezielle Funktion ist, da es sich hierbei um ein sogenanntes Signal handelt. Dieses Signal kann nun mit Funktionen anderer Klasse verbunden werden. Wird nun dieses Signal emittiert, so werden alle mit ihr verbunden Funktionen ausgeführt.
Dies ist in der *globals.cpp* zu sehen. Wird die normale Methode „*addDetectedFace*“ von außerhalb aufgerufen, so wird das Signal „*addDetectedFaceSig*“ emittiert. Da in der *mainwindow.cpp* die Methode „*onAddDetectedFace*“ mit diesem Signal verbunden wurden, wird jedes Mal, wenn das Signal emittiert wird, auch diese Funktion aufgerufen.

Das ist zum Beispiel in der *imageconverter.cpp* exemplarisch erkennbar. Hierbei wird die normale Methode, die sich in der *Globals*-Klasse befindet, für jedes detektierte Gesicht aufgerufen. Da wir die detektierten Gesichter aber anzeigen lassen möchten, wird darum die Funktion in der *MainWindow*-Klasse aufgerufen, wodurch wir genau das bewerkstelligen können.

5.2 Ungarische Notation

Die Ungarische Notation ist eine bestimmte Schreibweise für Bezeichner, die diese eindeutig identifizieren und mehr Aufschluss über diese geben soll. Anfänglich war geplant, dass wir die Ungarische Notation vollständig in unserem Quellcode verwenden. Wir zogen es bis zum Schluss durch und gaben jeder Variable einen der Ungarischen Notation entsprechenden Bezeichner. Da wir am Schluss jedoch sahen, dass es den Code vom lesen her irgendwie blockierte, entschlossen wir uns schlichtweg, nur einen Teil der Ungarischen Notation beizubehalten und den Rest zu löschen. Der Teil der übrig geblieben ist, ist lediglich die Präfix-Schreibweise der Ungarischen Notation. Dadurch sieht „unsere“ Ungarische Notation nun wie folgt aus:

{Präfix}{Bezeichner}

Darunter diese Präfixe:

Präfix	Sichtbarkeit	Beispiel
m_	Member-Variable	m_LastName
p_	Methodenparameter	p_NewValue
i_	Interfaceparameter (Argument von Funktionen)	i_NewValue
s_	statische Variable	s_InstanceCount
g_	globale Variable	g_Timestamp

Die Änderungen die unsere Bezeichner durchmachten, sind auf unserem GitHub-Repository einzusehen.

5.3 Globals-Klasse

Der folgende Text ist ein Zitat aus dem EasyLogic-Projekt in Ct, das Herrn Lempp bekannt sein sollte:

Die Globals-Datei ist eine nach der Singleton-Designstruktur erstellte Klasse. Ihre Aufgabe ist, für alle Klassen „globale“ Informationen bereitzustellen. Diese Klasse bekommt eine extra Widmung in dieser Dokumentation, da es ein sehr kontrovers diskutiertes Thema ist, ob man in C++ sogenannte Globals-Klassen/-Dateien verwenden sollte. Sie gilt als sehr riskant, da, wenn man sich mal ein Projekt mit mehreren Tausend oder sogar Millionen Zeilen Code vorstellt, die alle auf diesen globalen Variablen basieren und diese sogar bearbeiten, kann es schwer werden, herauszufinden, wer wann auf diese Variablen zugreift. Des Weiteren erhöht die Verwendung von globalen Variablen die Komplexität des Quellcodes. Da das erste Problem bei uns jedoch nicht auftreten kann, entschlossen wir uns, unser Globals-

Konzept beizubehalten. Üblicherweise werden solche globalen Variablen ganz anders angelegt:

globals.h:

```
#ifndef GLOBALS_H
#define GLOBALS_H

extern int g_iGlobalInteger;

#endif // GLOBALS_H
```

globals.cpp:

```
#include "globals.h"

int g_iGlobalInteger = 0;
```

main.cpp:

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

int main(int argc, char **argv)
{
    ::g_iGlobalInteger += 1;
    std::cout << ::g_iGlobalInteger << std::endl;
    return 0;
}
```

Das resultierende Ergebnis wird ohne Zweifel „1“ lauten.

Durch den ::-Operator wird gesagt, dass die globale Variable verwendet werden soll, falls eine gleichnamige lokale Variable existieren sollte.
Hier ein Beispiel dafür:

main.cpp:

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

int main(int argc, char **argv)
{
    int g_iGlobalInteger = 4;
    g_iGlobalInteger += 2;
    ::g_iGlobalInteger += 1;
    std::cout << ::g iGlobalInteger << std::endl;
    std::cout << g_iGlobalInteger << std::endl;
    return 0;
}
```

Die Ausgaben lauten:

- 1 (globale *g_iGlobalInteger* Variable)
- 6 (lokale *g_iGlobalInteger* Variable)

Natürlich widerspricht das obige Beispiel der Ungarischen Notation, jedoch geht es nur um die Verdeutlichung der Unterscheidung zwischen lokaler und globaler Variable.

Natürlich hätten wir diese Struktur auch in unserem Projekt einbauen können, aber da wir die Unübersichtlichkeit dadurch förmlich riechen konnten, entschlossen wir uns, das Singleton-Designpattern zu verwenden.

Im Gegensatz dazu, werden globale Variablen bei uns folgendermaßen benutzt:

(rein exemplarisch)

```
Globals::getInstance()->m_CurrentFaces.clear();
```

Es ist eindeutig viel mehr Schreibaufwand. Jedoch ist der Sinn der Variablen so viel einfacher nachzuvollziehen.

6 Entwicklung des Quellcodes

Legende:

- Normale Klasse
- Singleton-Klasse
- Graphical User Interface

Zu Anfang bestand unser Projekt lediglich aus der *MainWindow*-Klasse und aus der *FlightController*-Klasse. Nach und nach kamen dort dann immer mehr und mehr Funktionen hinzu, die dem finalen Programm immer mehr und mehr

ähneln sollten. Die *FlightController*-Klasse war die erste Klasse, die wir vollständig zu Ende bringen konnten. Mit ihr entstand die *Globals*-Klasse.

Funktionen der Klassen:

- MainWindow:
  Das Kontrollzentrum der Drohne. Mit ihr war es möglich die Drohne zu steuern. Dabei wurde die *FlightController*-Instanz zur Hilfe gezogen.

- FlightController:
  Ermöglicht das Steuern der Drohne. Hierbei werden einfach Befehle an den *roscore* gesendet, nachdem man sich mit ihm verband, die dann ausgewertet werden und eine Aktion der Drohne bestimmen.

- Globals:
  Sollte globale Informationen, die für alle Klassen verfügbar sein müssen, speichern. Sie wurde nur jeweils durch neue Attribute ergänzt. Die Aufgabe blieb stets die gleiche.

Nun, wir waren zwar mit unserem Werk zufrieden – denn wir konnten die Drohne über die GUI steuern – doch wir gingen einen Schritt weiter und implementierten die Gesichtserkennung. Dabei erzeugten wir eine neue Klasse, die genau das ermöglichen sollte. Die *ImageConverter*-Klasse.

Erweiterung zu *Funktionen der Klassen*:

- ImageConverter:
  Frägt Kamerabilder ab und verarbeitet sie. Markiert Gesichter und Augen, falls sie erkannt werden.

Als auch diese Hürde geschafft war, wollten wir die tatsächliche Gesichtserkennung einbauen. Sprich, 2 Gesichter vergleichen und schauen, ob diese identisch sind. Diese Klasse trägt den Namen *FaceRecognition*. Des Weiteren implementierten wir eine Schnittstelle für die GUI. Dabei werden lediglich die detektierten Gesichter mit einer Liste, aller detektierten Gesichter – ab Programmstart – verglichen und die neuen Gesichter in einer visuellen Liste angezeigt. Wählt man nun eines oder mehrere dieser Gesichter aus,

werden diese Gesichter von der Drohne verfolgt. Sofern die Gesichtsverfolgung aktiv ist. Dazu erweiterten wir die *ImageConverter*-Klasse um die Track-Funktion.

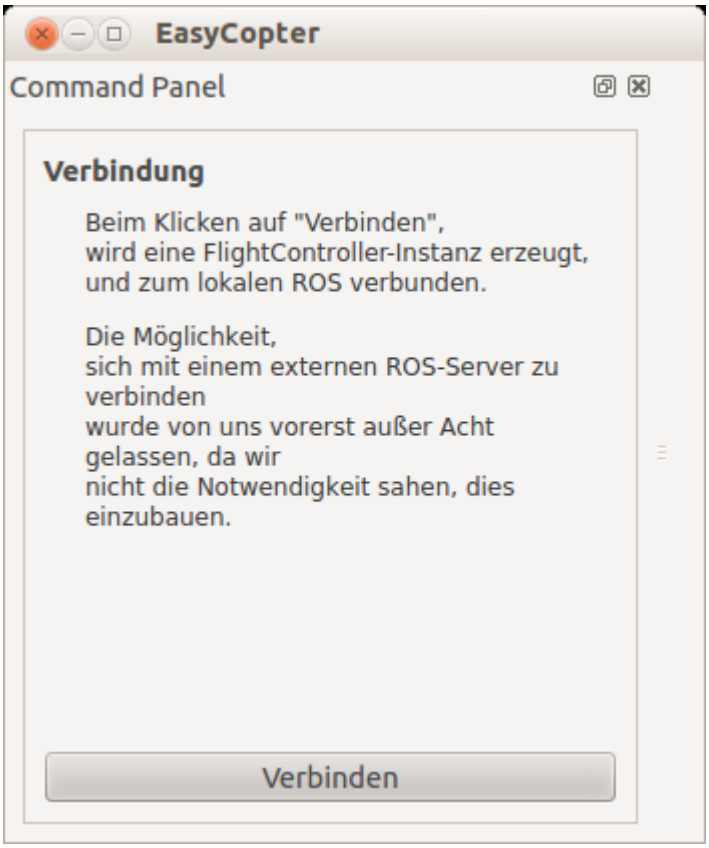
Erweiterung zu *Funktionen der Klassen*:

- ImageConverter:
  Gesichtsverfolgung erkannter Gesichter durch die Drohne.

- FaceRecognition:
  Vergleicht Gesichter und schaut, ob diese identisch sind.

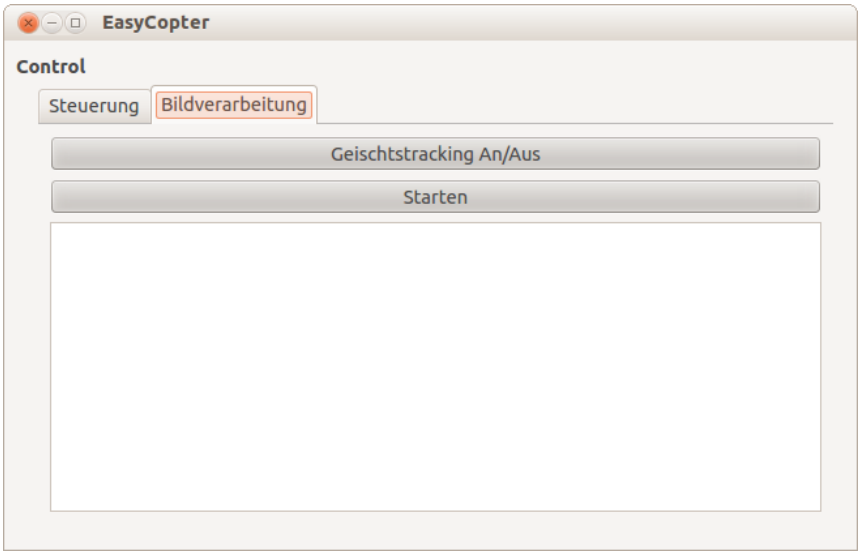
- MainWindow:
  Auswahl detektierter Gesichter und starten der Gesichtsverfolgung.

7 Finale Version



Hierbei handelt es sich um unser Startfenster. So sieht es aus, wenn man unsere Software startet. Ein Klick auf „Verbinden“ stellt nun die Verbindung zu ROS vollständig her.

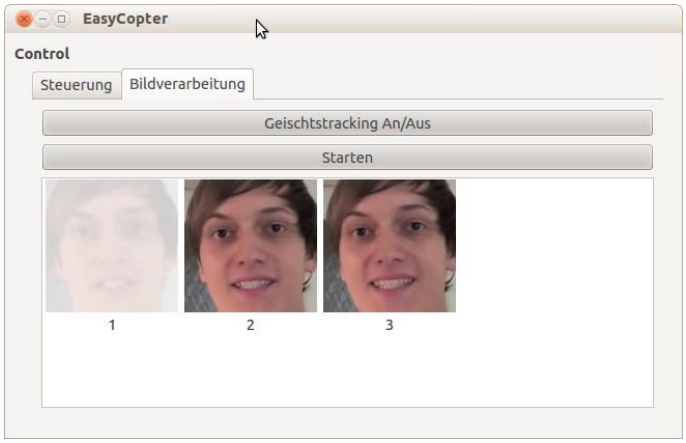
Nachdem man den Button „Verbinden“ betätigt hat, ordnet sich das Fenster neu an und sieht dann wie folgt aus:



Der Button „Gesichtstracking An/Aus“, aktiviert wie der Name schon sagt, die Verfolgung von Gesichtern. Dazu muss zuerst jedoch der Button „Starten“ betätigt werden. Nachdem man dies gemacht hat, öffnet sich folgendes Fenster (in diesem Fenster wird nun zum Test ein Video abgespielt, sonst werden die Kamerabilder der Frontkamera der Drohne angezeigt):

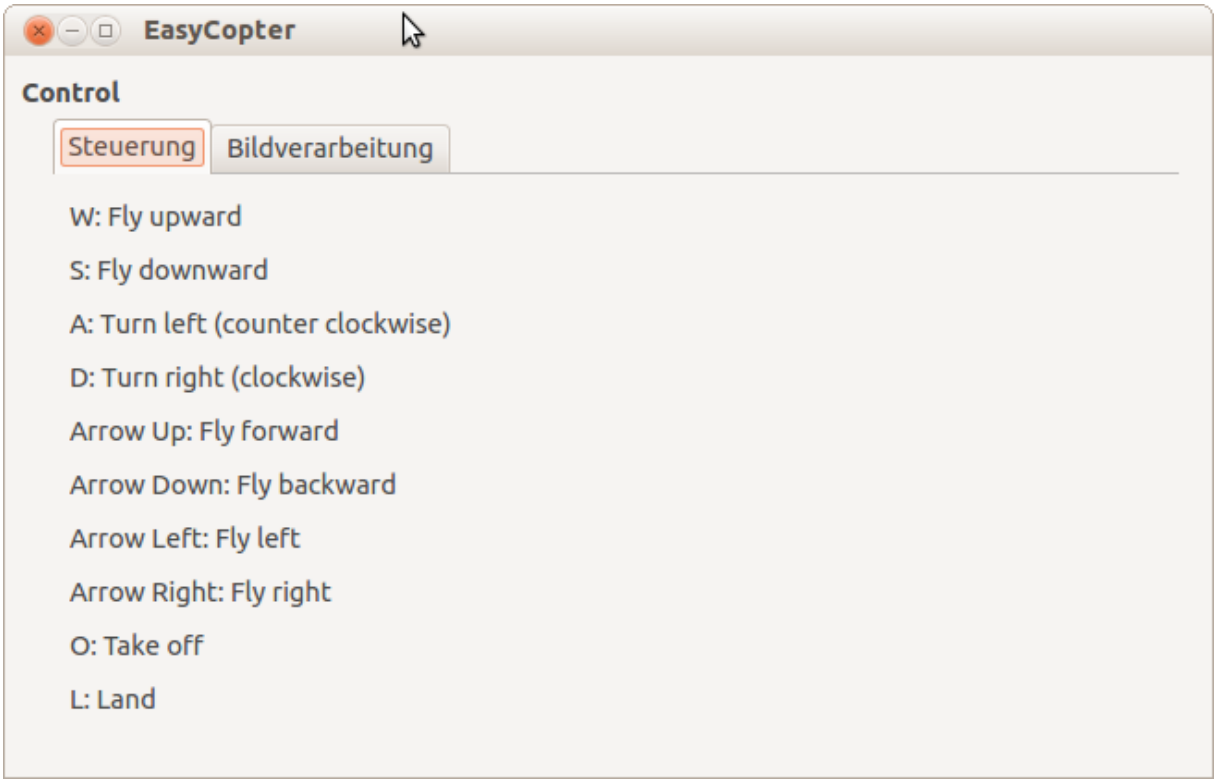
Werden nun Gesichter erkannt, so werden sie auf jedem Frame hervorgehoben, durch Umkreisung.

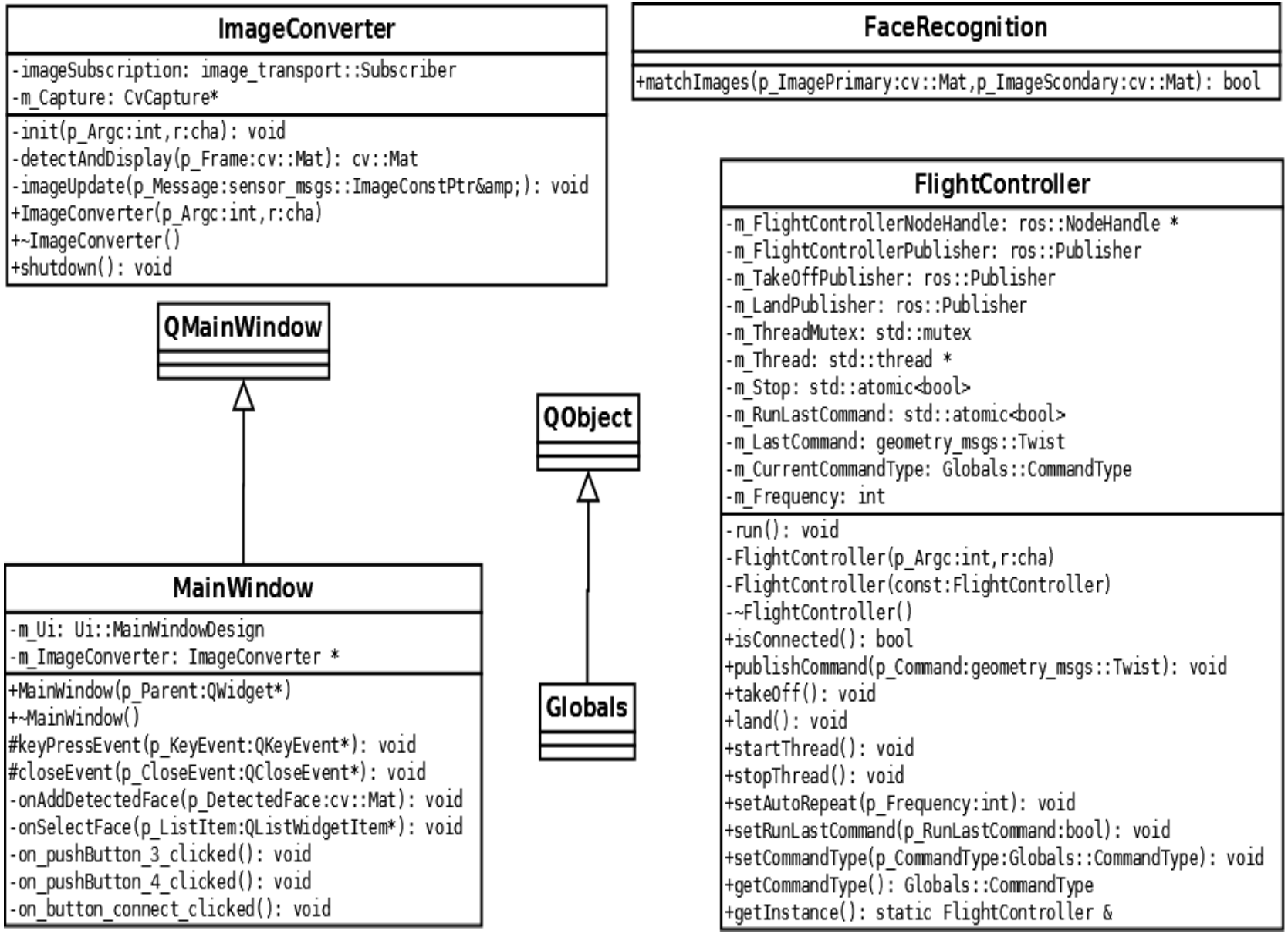
Wird ein Gesicht erkannt und ist dieses scheinbar noch nicht in der Liste der detektierten Gesichter drinnen, so füllte sich obige Liste, mit diesen Bildern.



Das hier nun 3 mal das gleiche Gesicht erkannt wird liegt daran, dass jeweils die interessanten Punkte unterschiedlich sind. Aufgrund dessen, ist es auch möglich, mehre Gesichter auszuwählen, um diesen „Fehler“ zu korrigieren. Wählt man nun eines der Gesichter aus und betätigt – wie zu vor gesagt – den Button „Gesichtstracking An/Aus“, so versucht die Drohne die ausgewählten Gesichter zu verfolgen. Dabei können auch unterschiedliche Gesichter verwendet werden.

Der Tab „Steuerung“ stellt lediglich dar, welche Tasten, für welche Aktionen gedrückt werden müssen. Sprich, anhand von welcher Tasten die Drohne beispielsweise nach vorne fliegt. Dabei muss immer DIESES Fenster den Fokus haben.





Christoph Breisacher
Oliver Suchan

8 Quellen

8.1 EasyLogic – Dokumentation

Das diesjährige Ct-Projekt von Philipp Faller und Oliver Suchan, dessen Dokumentation bei Herr Lempp auffindbar sein sollte.
Siehe Abschnitt: „Informationen zum Quellcode -> Globals-Klasse“

8.2 Qt 5.2

http://qt-project.org/downloads

8.3 ROS-Fuerte

http://wiki.ros.org/fuerte/Installation/Ubuntu

8.4 TUM-Simulator

http://wiki.ros.org/tum_simulator

8.5 TUM-ARDrone

http://wiki.ros.org/tum_ardrone

8.6 qt_ros

http://ros.informatik.uni-freiburg.de/roswiki/qt_ros.html?distro=fuerte

8.7 Ubuntu 12.04

http://www.ubuntu.com/download/desktop

8.8 GitHub

https://github.com/cranktec/EasyCopter