Hochschule Esslingen
University of Applied Sciences

Studienarbeit

Tracking in OpenCV und C# und die Ansteuerung eines Roboterarms

im Studiengang Technische Informatik

der Fakultät Informationstechnik

Eugen Muschewski

Zeitraum:	01.10.2015 bis 17.02.2016
Erstprüfer:	Prof. Dr.-Ing Harald Melcher
Zweitprüfer:	Prof. Dr.-Ing Andreas Rößler

Betreuer:	Prof. Dr.-Ing Harald Melcher

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Esslingen, den 17. Februar 2016 

Unterschrift

Kurzfassung

Gegenstand der hier vorgelegten Arbeit ist eine Software zur Ansteuerung eines Industrie Roboterarmes. Dies erfolgt über eine USB Webcam, über diese dann Bewegungen im Bild erkannt werden. Der Roboter folgt dem erkannten Objekt in einem 270 Grad Winkel.

Schlagwörter: Studienarbeit, wissenschaftliche Arbeit, C#, C++, OpenCV, Movemaster RM-501,

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Der Einsatz von Robotern ist in der heutigen Industrie bereits sehr verbreitet und seit Langem schon Praxis. Ziel für die Zukunft ist, immer mehr Standard- und Routineeingriffe durch Roboter zu ersetzten.

Im Laufe der Zeit, seit dem Einsatz des ersten programmierbaren Industrieroboters Ultimate im Jahre 1956 (5) stiegen die Zahlen über den Einsatz neu installierter Industrieroboter Jährlich an. 2011 waren es 166.000 (5) Industrieroboter, die in den verschiedensten Bereichen eingesetzt worden sind. Vom Montieren, Palettieren bis zum Verpacken kann ein Industrieroboter, je nach Programmierung, in allen vorstellbaren Bereichen angewandt werden. Da Maschinen mittlerweile in der Lage sind, wie Menschen zu erkennen und je nach Erkennungsmuster einen bestimmten Befehl auszuführen, entstehen für diese immer mehr Einsatzmöglichkeiten.

Die Firma KUKA wirbt für Ihren Roboter KUKA KR AGILUS[1] in einen Tischtennismatch gegen den Tischtennisprofi Timo Boll. Das inszenierte Match sollte zeigen, wie schnell der Roboter auf seinen Gegenspieler reagiert und sofort eine Gegenmaßnahme unternimmt, um den Ball zurückzuspielen.

Es bieten sich viele neue Einsatzmöglichkeiten. Tischtennis ist nur eine davon, aber es gibt auch viele nützliche Vorgänge. Beispiel dafür ist eine visuelle Befehlsvergabe an einen Industrieroboter. Dafür kommt das maschinelle Sehen in Kombination mit Robotern zum Einsatz.

[1] Quelle: http://www.kuka-timoboll.com/en/kuka_kr_agilus_vs_timo_boll/ (abgerufen am 20.10.2015)

1.1 Zielsetzung und Motivation

Ziel dieser Studienarbeit ist es, eine Software zu entwickeln, welche es ermöglicht, einem Industrieroboter vom Modell RM-501 die Bewegungen von Objekten zu erkennen. Dabei wird nicht der Roboter bewegt, sondern der mit einer Kamera ausgestatteter Arm des Roboters, dessen Aufgabe es ist, bewegte Objekte wahrzunehmen.

Um die Realisierung zu demonstrieren soll eine einfache Anwendung entwickelt werden, die eine Kommunikation zwischen Objekt und Maschine ermöglicht. Dazu werden Funktionen wie das Erkennen von bewegten Objekten sowie die Verfolgung der Objekte realisiert. Des Weiteren soll die Software Erweiterbar sein, um neue mögliche Features einzubauen.

Motivation für das Verfassen dieser Arbeit war, dass dieses Thema ein hohes Potential und Freiheitsgrade für die Entwicklung bietet, sowie die Entstehung eines breiten Spektrums durch daraus entstehende Möglichkeiten zur Lösung von Aufgaben. Ebenso kann durch den Einsatz von Programmiersprachen wie C++ / C# und OpenCV ein hoher Erkenntnisfaktor gewonnen werden.

Grundlagen

Um das Projekt zu realisieren wurden die Programmiersprachen C++ und C# mit der Einbindung der OpenCV Bibliothek verwendet. Die Kombination dieser Komponenten wird genauer erläutert. Ebenso wird erklärt, wie eine Bewegung im Bild als Objekt wahrgenommen werden kann. Es wird erläutert, welche Möglichkeiten der Roboterarm Mitsubishi RM-501 dem Anwender bietet.

1.2 C++ und C#

Da für die Realisierung des Projektes nur objektorientierte Programmiersprachen in Frage kamen, fiel die Entscheidung für den Einsatz von C++ und C#. Die Ausführung von C++ Codes ist auf allen Betriebssystemen möglich. C# dagegen benötigt für die Ausführung das .NET Framework. Dieses wird von Linux und MacOS durch Mono Framework ab Version 2.0 vollständig unterstützt und deckt teilweise auch Bereiche der .NET Version 4.5 ab. Für die Realisierung einer grafischen Benutzeroberfläche unter Einsatz von C++ werden externe Bibliotheken wie Qt benötigt. Diese bieten jedoch, im Gegensatz zu Windows-Forms in Visual Studio, weniger Vorteile und Möglichkeiten für Editierung. Auch bei der Threadverwaltung übernimmt C# wesentliche Teile der Arbeit. Dies macht die Entwicklung einfacher und die Implementierung robuster. Für eine einfache und effiziente Realisierung von Multithreading unter Einsatz von C++ muss erneut von externen Tools wie Boost Bibliothek Gebrauch gemacht werden. Die auf der Hand liegenden Vorteile befürworten den Einsatz der Programmiersprache C# für die Realisierung des Projektes. Einzige Schwierigkeit stellt die Einbindung der Programmbibliothek OpenCV dar. Einbindung dieser in C++ erfolgt direkt über DLL. Zusätzlich steht zu dieser eine ausführliche Dokumentation zur Verfügung. Einbindung genannter Bibliothek in C# ist wesentlich umständlicher, dennoch machbar. In C# stehen dafür Wrapper zur Verfügung, die ebenso als Bibliothek eingebunden werden können. Solche wä-

ren z.B. Emgu und OpenCvCharp. Während Emgu als Open Source und kommerziell lizensiert zur Verfügung steht, gibt es OpenCvCharp komplett unter der BSD 3-Clause License frei zur Verfügung.

Emgu: http://www.emgu.com

OpenCvCharp: https://github.com/shimat/opencvsharp

Zu Beginn wurde die Realisierung in beiden Programmiersprachen verfolgt. Die finale Realisierung wurde jedoch in C# durchgeführt, da sich diese als wesentlich schlichter und sicherer erwiesen hat.

1.3 OpenCV

OpenCV ist eine freie Programmbibliothek mit Algorithmen für die Bildverarbeitung und maschinelles Sehen. Diese hat C++, C, Python und Java-Schnittstellen und unterstützt die Betriebssysteme Windows, Linux, Mac OS, iOS und Android. OpenCV wurde unter einer BSD-Lizenz veröffentlicht und ist daher sowohl für akademische als auch kommerzielle Nutzung geeignet und kostenlos. Es ist in optimiertem C/C ++ Code geschrieben. Ebenso bietet die Bibliothek die Vorteile von multi-core Verarbeitung. Großen Vorteil bietet OpenCL durch die Integration der Hardware-Beschleunigung. OpenCV hat mehr als 47 tausend Benutzer in der User-Community. Die geschätzte Zahl der Downloads liegt bei mehr als 9 Millionen.

Unter http://opencv.org/ ist eine ausführliche Dokumentation zu finden. Ebenso stellt die Website viele Tutorials für den Einstieg in OpenCV zur Verfügung. So findet man viel Material zu den unterschiedlichsten Projekten der Online Community.

1.4 Erkennung von Objekten

An dieser Stelle werden die eingesetzten Methoden der OpenCV Bibliothek erklärt. Die erklärten Grundlagen werden für das Grundverständnis der Arbeit benötigt.

OpenCV bietet die Möglichkeit, einfache Webcams als Videoquelle zu Verwenden. Somit bietet es eine einfache Möglichkeit, Objekterkennung in den zweidimensionalen Raum zu betreiben.

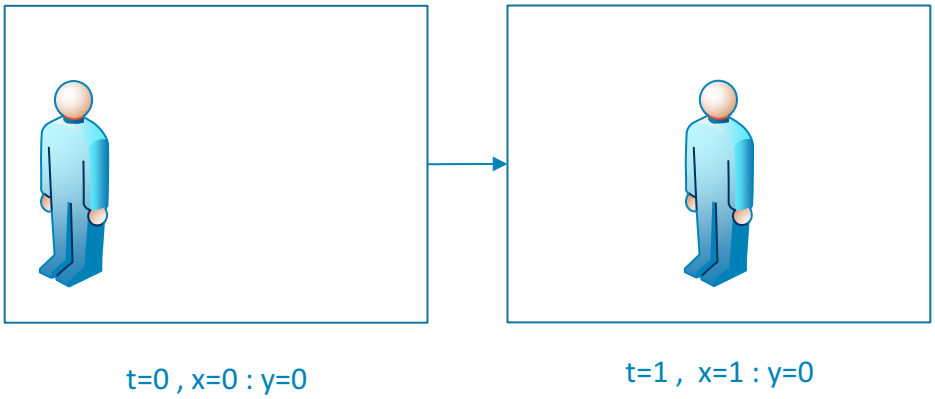


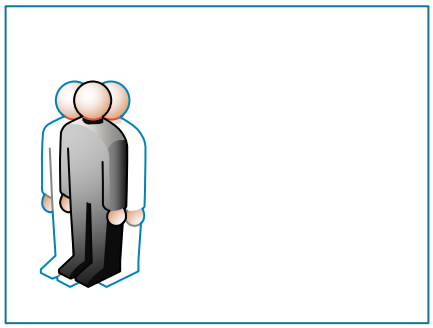
Abbildung 1 – Videoaufnahme einer Person

Die in der Abbildung 1 dargestellte Aufnahme kann auf einer Zeitachse einem Zeitabschnitt zugeordnet werden. Der in der Abbildung links dargestellter Zustand wird dem Zeitpunkt t=0 zugeteilt. Zu diesem Zeitpunkt bewegt sich die abgebildete Person nicht. Der zum Zeitpunkt t=0 aufgenommene Zustand ist fest und wird im Raum bei x=0 als Bild1 gespeichert. Abbildung 1 rechts zeigt, wie sich die Person vom ursprünglichen Standpunkt im Raum zu einem neuen Punkt im Raum bewegt. In diesem Zeitabschnitt von t = 0 zu t=1 bewegt sich die Person zum Punkt x=1, was in dieser Abbildung dargestellt ist. Beide Bilder können verglichen werden. Daraus kann die Summe der absoluten Differenzen gebildet werden, welche als Maß für die Unterschiedlichkeit zweier Bilder dient. Abbildung 2 Stellt die Berechnung dar.

$$absdiff = \sum_{x=0}^{breite-1} \sum_{y=0}^{höhe-1} |Bild2(x-y) - Bild1(x-y)|$$

Abbildung 2 – Formel Absolute Differenz

Nach dieser Berechnung und der Kombination der Bilder kann eine Neue Abbildung generiert werden. Diese ist in der Abbildung 3 dargestellt und beschreibt den Stand zwischen den Standpunkten x=0 und x=1.



t=0,5 , x=0,5 : y=0

Abbildung 3 – Differenz Bild

Der Betrachter nimmt dies als Bewegung wahr. Diese Bewegung wird als auch Objekt zugeordnet, da im dritten Bild jeder Punkt der Person bekannt ist. Dies geschieht so schnell, dass jede Bewegung sofort Verarbeitet werden kann. Zu Veranschaulichung kann im Bild für den Betrachter eine Linie durch diese Punkte gezogen werden. Somit würde der Betrachter, wenn er sich bewegt, einen Rahmen um seinen Körper sehen, wenn er diesen Bewegt. Da für die Verarbeitung die Farben nicht relevant sind und dessen Wegfall geringeren Informationsgrad sowie höhere Verarbeitungsgeschwindigkeit ermöglichen würde, wird die Bewegungserkennung meistens mit Bildern in Graustufen durchgeführt. Ebenso wird eine Methode der Segmentierung verwendet. Diese wird auch als Schwellwertbildung (1), um die Differenz zweier Bilder einzublenden und alles andere im Bild einzuschwärzen, verwendet. Hier wird im Bild der Ap-

fel, ähnlich wie beim Negativ Bild, als weißes Objekt dargestellt. Dadurch kann die Positionsbestimmung des Objektes einfacher errechnet werden.

Abbildung 4 – Schwellwertbildung Apfel

1.5 Mitsubishi RM-501

Abbildung 5 – Movemaster RM-501

Der Mitsubishi Movemaster RM-501 ist ein 5-Achsen-Roboterarm mit einem Greifer der aus 2 Fingern besteht. Das Gerät wurde in 80er Jahren gebaut.

Eine Bewegung des Armes kann als Vektor beschrieben werden.

$$q = [\, q1,\ q2,\ q3,\ q4,\ q5 \,]$$

Dabei hat jedes Gelenk noch einen Wertebereich für die Bewegung.

	Bezeichnung	**Wertebereich**
q1	Drehachse	12000 ergibt 360Grad
q2	Schulter	5200
q3	Ellenbogen	3600
q4	Handgelenk	4800
q5	Handgelenk	9600

Ebenso bietet die Drive Unit Befehle, die dirket über eine COM Schnittstelle zum Roboterarm übertragen werden können.

Name	**Eingabe Format**	**Funktionsbeschreibung**
Nest	NT	Mechanischer Uhrschrpunkspunk
Home	HO	Setzt einen neuen Uhrsprung
Origin	OH	Returned zum Uhrschrpunkspunk
Move Immediate	MI(q1,q2,q3,q4,q5,q6)	Bewegt den Arm um die Parameter
Move	MO(q)	Bewegt den Arm zum gesetzten Punkt
Increment	IP	Inkrementiert die Position

Decrement	DP	Dekrementiert die Position
Position set	PS(q1,q2,q3,q4,q5,q6)	Setzt einen Punkt
Here	HE(q)	Speichert die Position nach MI
Position clear	PC(q1,q2)	Löscht die Position
Grip set	GP(q1,q2,q3)	Stiger oder Senkt den Greifdruck
Grip opened	GF(q)	Öffnet den Greifer
Grip closed	SP(q)	Schließt den Greifer
Grip flag	TI(q)	Flag für den Greifer
Speed	IN	Setzt die Geschwindigkeit 0-9
Time	ID	Stoppet die Operation für eine gewisse Zeit 0-99
Input controller	OT((q&b)	input daten vom external controller
Output data	OD((q&b)	output daten zu external controller
Test bit	TB(q1,q2)	Test bit
Error flag	EF(q)	Flag für Fehler
If larger	LG(q1,&b,q2)	Vergleicht Eingangssignal von außen mit dem Parameter von q1 für die Gleichstellung festgelegt, um auf die von q2 angegebenen Zeile zu springen
If equal	EQ(q1,&b,q2)	
If smaller	SM(q1,&b,q2)	
If not equal	NE(q1,&b,q2)	
Go subroutine	GS(q)	Springt zur Subroutine
Return	RT	Kehrt von Subroutine zurück
Repeat	RC(q)	Wiederhollt letzte Ausführung
Next	NX	nächste Ausführung
Go to	GT(q)	Spring zur zeile q

End	ED	Ende einer Subroutine
New	NW	Leert den RAM speicher
Delete	DL(q1,q2)	Löscht Subroutine von Zeile q1-q2
Run	RN(q)	Startet ab Zeile q
Write	WR(q)	Schreibt ab Zeile q
Transfer	TR(q)	Tranferiert von ROM zu RAM
Reset	RS	Leert den Fehlerflag

Tabelle 1 - RM-501 Befehlsübersicht

Wichtige Befehle für Umsetzung
Reset
Home
Move Immediate
Speed

Tabelle 2 - Wichtige Befehle für die Umsetzung

2 Analyse

In diesem Kapitel werden die Anforderungen an das System analysiert und gesammelt, die später für die Implementierung des Systems benötigt werden.

2.1 Anforderungen

2.1.1 Bedienoberfläche

Die Bedienoberfläche für das System muss einfach sein. Sie das aktuelle Kamerabild zeigen. Des Weiteren soll diese das Bild der Schwellwertbildung und das Finale Bild in Form des erkannten Objektes darstellen.

Die Bedienoberfläche soll die Möglichkeit bieten, aus bis zu 6 angeschlossenen Webcams eine für den Betrieb auszuwählen.

Die Bedienoberfläche soll allen Geräten, die am COM Port angeschlossen sind, die Möglichkeit bieten, die Drive Unit des Roboterarmes auszuwählen.

Die Bedienoberfläche kann den Roboterarm starten, stoppen, auf den Mechanischen Ursprung sowie auf die Home Position setzten. Auch soll die Möglichkeit bestehen, Fehler der Drive Unit zurückzusetzten.

Durch die Bedienoberfläche sollen das Bildrauschen entfernt und die Objektgröße verändert werden können.

2.1.2 Objekterkennung und Tracking

Das System soll die Bewegung eines Objektes im 2D Raum sowie auch das Erreichen dessen Stillstandes erkennen können. Ebenso sollen die Positionsdaten vom Objekt angezeigt werden. Das Objekt wird dann verfolgt.

2.1.3 Ansteuerung des RM-501

Das System soll den Roboterarm bewegen können. Dieser soll sich immer in die Richtung des erkannten Objektes ausrichten.

2.2 Problemszenarien

2.2.1 Drehachse des Roboters

Wenn der Roboter seine Grenze von 12000 Inkrementen, was 360° entspricht, erreicht und die Bewegung somit mehr als eine Umdrehung beträgt, setzt die Drive Unit eine Fehlermeldung ab. Das bedeutet, dass er an seine Mechanische Grenze gestoßen ist.

Dies kann passieren, wenn das erfasste Objekt immer weiter nach Links oder Rechts im Blickfeld bewegt wird.

2.2.2 Mehrere Objekte bewegen sich

Wenn sich mehrere bewegende Objekte im Blickfeld befinden, werden beide als solche erkannt. Das Problem dabei ist, dass sich das System nur ein Objekt aussuchen muss, welches verfolgt werden soll.

2.2.3 Lichtverhältnisse und Kamerarauschen

Wenn die Lichtverhältnisse für eine zur Verfolgung benötigte Qualität der Aufnahme nicht tauglich sind oder durch schlechte Kamerahardware ein Kamerarauschen vorhanden ist, besteht das Problem, dass die Videoquelle für die Weiterverarbeitung nicht geeignet ist.

3 Konzept

3.1 Architektur

3.1.1 Klassendiagramm

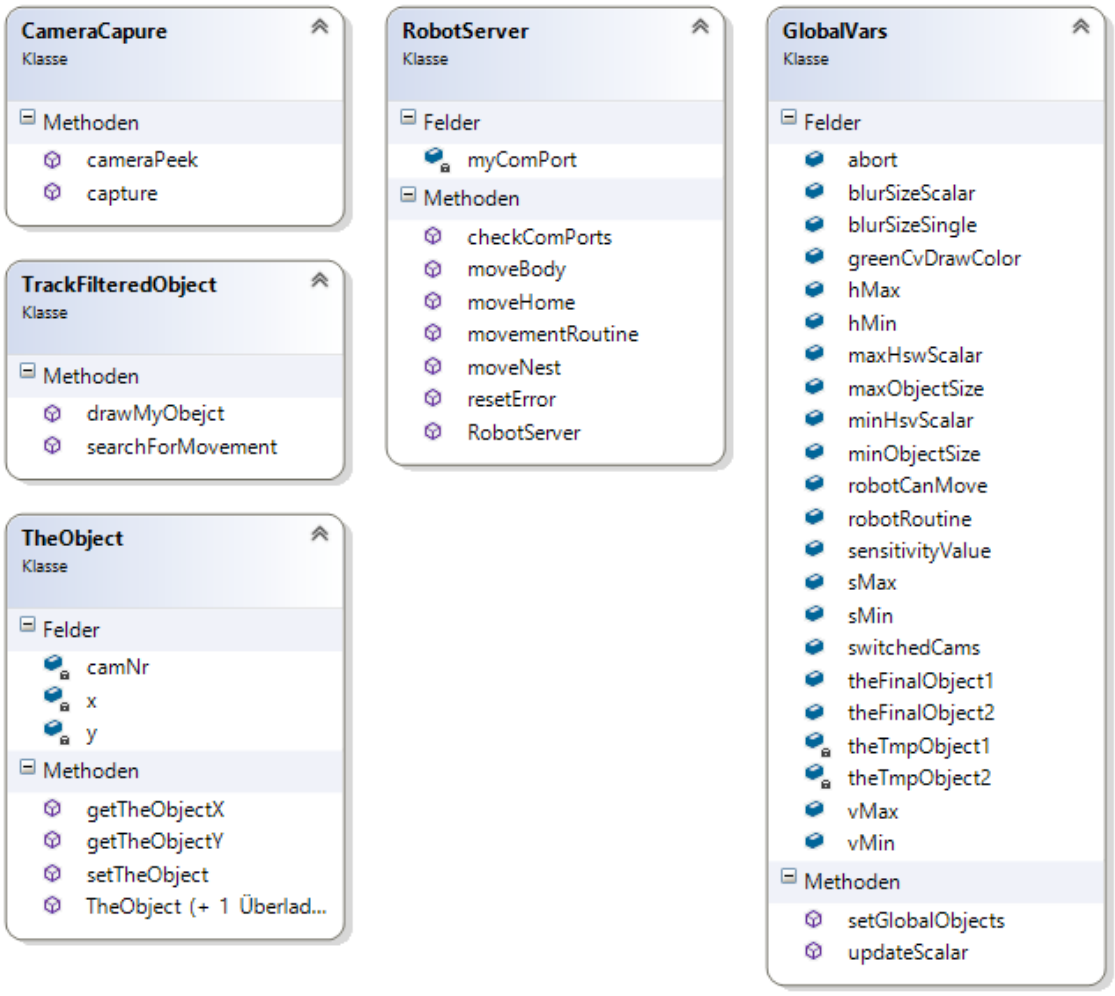


Abbildung 6 – Klassendiagramm

3.1.2 UML Aktivitätsdiagramm

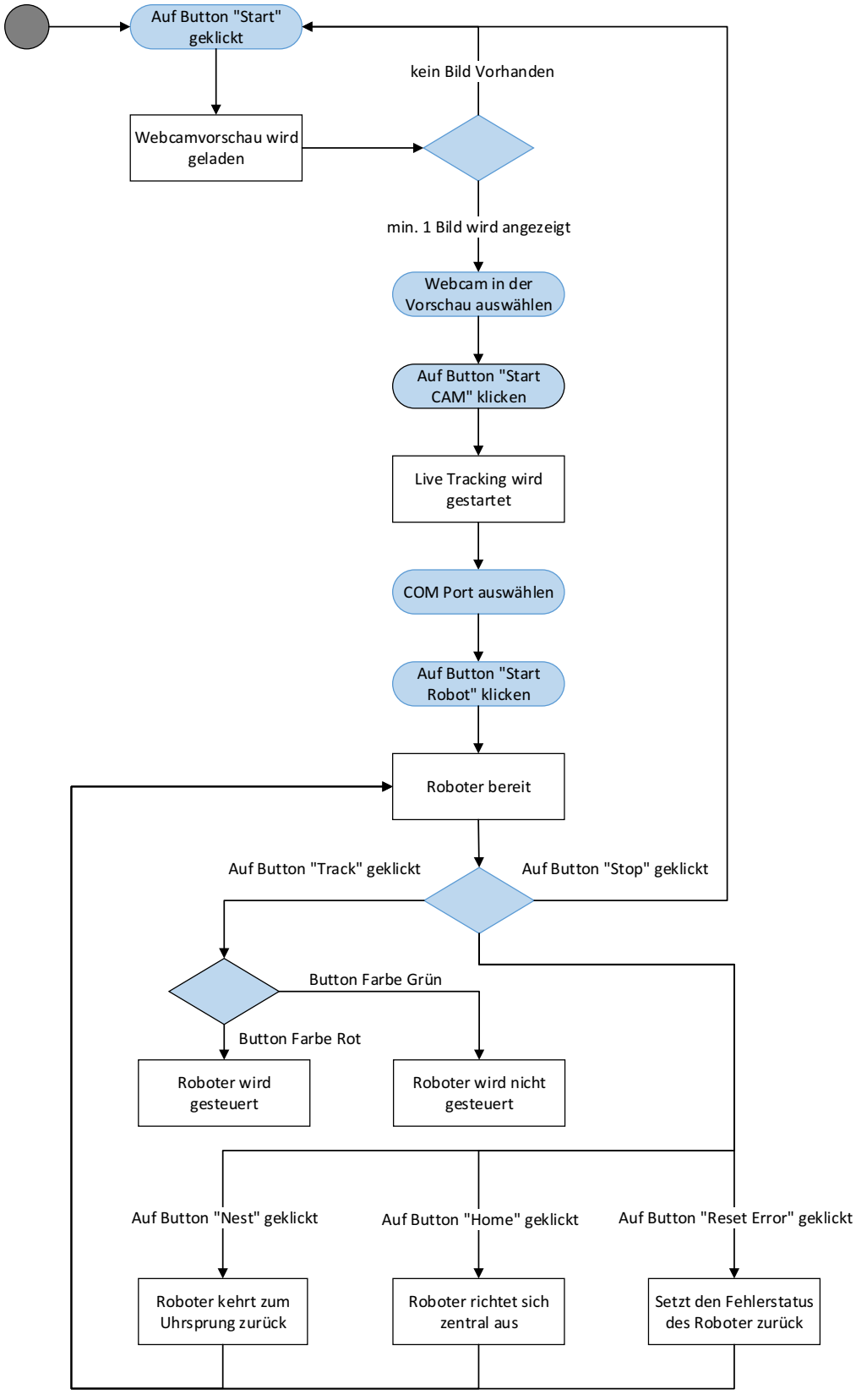


Abbildung 7 – UML Aktivitätsdiagramm

4 Implementierung

4.1 GUI

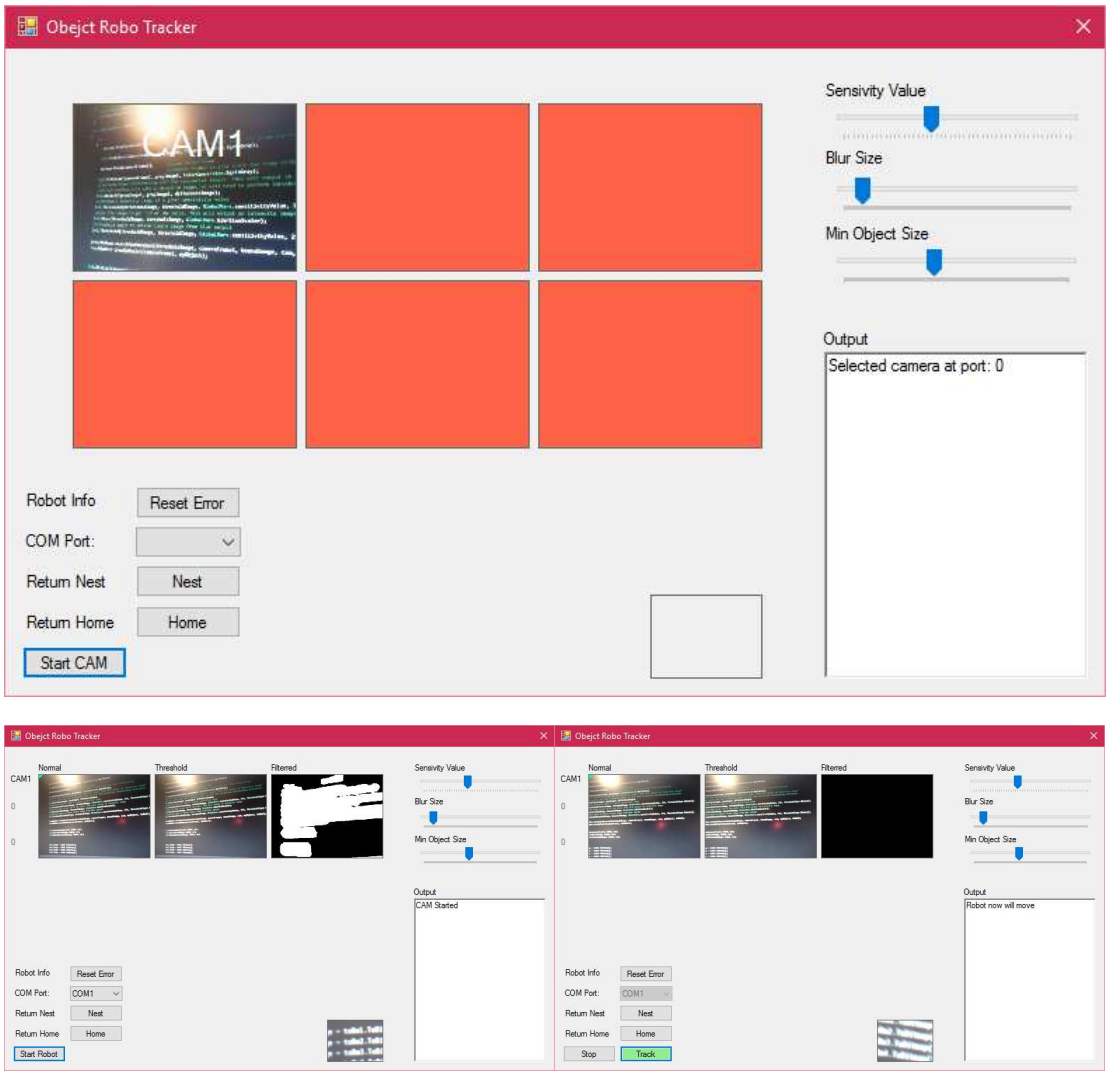


Abbildung 8 – GUI

Die Bedienoberfläche wurde einfach gestaltet. Über das Vorschaubild kann ausgesucht werden, welche Kamera für den Betrieb ausgewählt wird. Die Restlichen Tasten sind nach der Erwartungskonformität gestaltet. Der Button Start

CAM Startet die ausgewählte Kamera. Eine Output TextBox gibt Information des Betriebes wieder.

4.2 Realisierung des Tracking

In der Klasse CameraCapture wird ein Video erzeugt. Es werden 2 aufeinanderfolgende Bilder genommen und gefiltert. Abschließend wird das Schwellwertbild für die Objekterkennung weitergereicht.

```
stream.Read(cameraFrame1);     //get first frame form video
                               //convert frame1 to gray scale for frame differencing
Cv2.CvtColor(cameraFrame1, grayImage1, ColorConversion.BgrToGray);


stream.Read(cameraFrame2);      //read second frame
                                //convert frame2 to gray scale for frame differencing
Cv2.CvtColor(cameraFrame2, grayImage2, ColorConversion.BgrToGray);
//perform frame differencing with the sequential images. This will output an "intensity image"
//do not confuse this with a threshold image, we will need to perform thresholding afterwards.
Cv2.Absdiff(grayImage1, grayImage2, differenceImage);
//threshold intensity image at a given sensitivity value
Cv2.Threshold(differenceImage, thresholdImage, GlobalVars.SENSITIVITY_VALUE, 255, ThresholdType.Binary);
//blur the image to get rid of the noise. This will output an intensity image
Cv2.Blur(thresholdImage, thresholdImage, GlobalVars.blurSz);
//threshold again to obtain binary image from blur output
Cv2.Threshold(thresholdImage, thresholdImage, GlobalVars.SENSITIVITY_VALUE, 255, ThresholdType.Binary);


TrackMyMove.searchForMovement(thresholdImage, cameraFrame1, boundImage, cam, myObject, toBm4);
TrackMyMove.drawMyObejct(cameraFrame1, myObject);
```

Mit dem Schwellwertbild kann nun in der Klasse TrackFilteredObject nach Konturen gesucht werden. Diese werden In einem Array gespeichert. Aus der Anzahl dieser kann bestimmt werden, ob zu viel Bildrauschen vorhanden ist oder das Objekt zu groß ist. Ist ein Objekt gefunden worden, so wird der Ausschnitt davon in einem Template gespeichert und die Koordinaten dessen ermittelt und gesichert, um im Fall einer Bewegungslosigkeit nach dem zuvor gefundenen Objekt zu suchen. Wenn es nun eine 80% Übereinstimmung gibt werden die Koordinaten diesem zugewiesen. (2) (3)

```
//find contours in our image
MatOfPoint[] contours = Cv2.FindContoursAsMat(tempImage, ContourRetrieval.External, ContourChain.ApproxNone);
//if contours vector is not empty, we have found some objects
//if robot doesnt move track my next object
if (GlobalVars.trackingRobot == false)
{
```

```
    if (contours.Length > 0 && contours.Length < 3)
    {

        // we have he a line of point p1, p2, pN... and then find that center
        foreach (MatOfPoint contour in contours)
        {
            // reset centers
            centerX = 0;
            centerY = 0;

            if (contour.Count > GlobalVars.minObjectSize && contour.Count < GlobalVars.maxObjectSize)
            {
                // for each point in the contour ... contour count give you the hight x witdh
                for (int ii = 0; ii < contour.Count; ii++)
                {
                    // get point
                    Point pts = contour.At<Point>(ii);
                    Cv2.Circle(boundImage, pts, 1, GlobalVars.greenCvDrawColor);

                    // set X Y
                    centerX += pts.X;
                    centerY += pts.Y;
                }

                // set center
                centerX /= contour.Count;
                centerY /= contour.Count;

                // add center
                myObject.setTheObject(centerX, centerY);

                // for compare when to much noise
                IplImage detectedObjecPicture = cameraFeed.ToIplImage();
                CvRect roiRect = new CvRect(myObject.getTheObjectX() - 40, myObject.getTheObjectY() - 30, 80, 60);
                // Refion of Interest when the Object is big we want keep te last track
                Cv.SetImageROI(detectedObjecPicture, roiRect);
                Mat regionOfInterestiImage = new Mat(detectedObjecPicture);

                regionOfInterestiImage.CopyTo(miniPicture);

            }
        }
    }
    else
    {
        // keep track of the object if we dont have a 80% match the object the doesn´t exist anymore
        try
        {
            resulted = cameraFeed.MatchTemplate(miniPicture, MatchTemplateMethod.CCoeffNormed);
            resulted.MinMaxLoc(out minVal, out maxVal, out minLoc, out maxLoc);

            if (maxLoc.X != 0 && maxVal > 0.80)
            {
                myObject.setTheObject(maxLoc.X + 40, maxLoc.Y + 30);
            }
            else
            {
                myObject.setTheObject(0, 0);
            }
        }
        // the object doesn´t exist anymore
        catch { }

    }
```

4.3 Ansteuerung des RM-501

Zur Ansteuerung des Arms wird eine Verbindung über die COM-Schnittstelle benötigt. Über diese Schnittstelle können Befehle direkt an die Drive Unit gesendet werden. Dafür muss zuerst ein neuer Port erstellt und geöffnet werden. Um nun einen Move Befehl auszuführen, muss zuvor überprüft werden, ob sich der Arm noch in seinen Achsen-Wertebereich befindet, um keinen Fehler zu erzeugen.

```
myComPort = new SerialPort(comPort, 9600, Parity.Even, 8, StopBits.One);

myComPort.RtsEnable = true;

…
```

```
if (myComPort.IsOpen)
{
     if (!myComPort.CtsHolding && myComPort.DsrHolding)
     {
          if (bodyLeftRight >= -5900 && bodyLeftRight <= 5900)
          {
               if (bodyUpDown >= 1700 || bodyUpDown <= -1700)
               {
                    b = 0;
               }
               myComPort.WriteLine("MI" + a + "," + b + "," + b + ",-0,-0,-0");
               bodyLeftRight += a;
               bodyUpDown += b;
          }

     }
     myComPort.Close();

}
```

4.4 Probleme während Implementierung

Wie schon im Kapitel Problemszenarien beschrieben, musste überlegt werden, wie diese gelöst werden konnten.

4.4.1 Tracking

Die ersten Probleme im stationären Fall waren das Bildrauschen und die Größe des Objektes. Dies konnte gelöst werden, indem der User über einen Regler das Kamerarauschen bei schlechten Lichtverhältnissen anpassen konnte. Ebenso wird über Regler eingestellt, wie grob oder fein die Kontur des Objektes verarbeitet werden soll, um z.B. eine zusammengesetzte Bewegung im Bild entgegen zu wirken. Zuletzt kann ebenso über einen dritten Regler die minimale Objektgröße eingestellt werden, um Entfernung, aber ebenso bei Rauschen kleine Details umgehen zu können.

Das zweite Problem stellte die Veränderung des gesamten Bildes während der Bewegung des Armes dar. Es konnte nun nicht mehr identifiziert werden, welcher Teil vom Bild sich verändert hat. Die Menge an erkannten Objekten stieg somit rasch an und verhinderte die Verfolgung.

Um dies zu beheben musste genau gesagt werden, wie viele Konturen im Schwellwertbild erlaubt sind. Ein fester Wert der Größe 3 hat sich nach Testversuchen als anwendbar erwiesen.

Wurde nun der Arm bewegt, musste noch herausgefunden werden, wo sich das zuvor erkannte Objekt aufhält, um dieses nicht zu verlieren.

Dazu wurde das von CV bereitgestellte Matching Implementiert, das im neu entstandenen Bild des zuvor gespeicherten Templates des Objektes sucht.

4.4.2 Roboter

Da die Drive Unit keine Abfrage über den aktuellen Wert eines Gelenkes ermöglicht, musste ein eigenes Koordinatensystem mit einen Wertebereich eingeführt werden, das dem Wertebereich der Achsen entspricht.

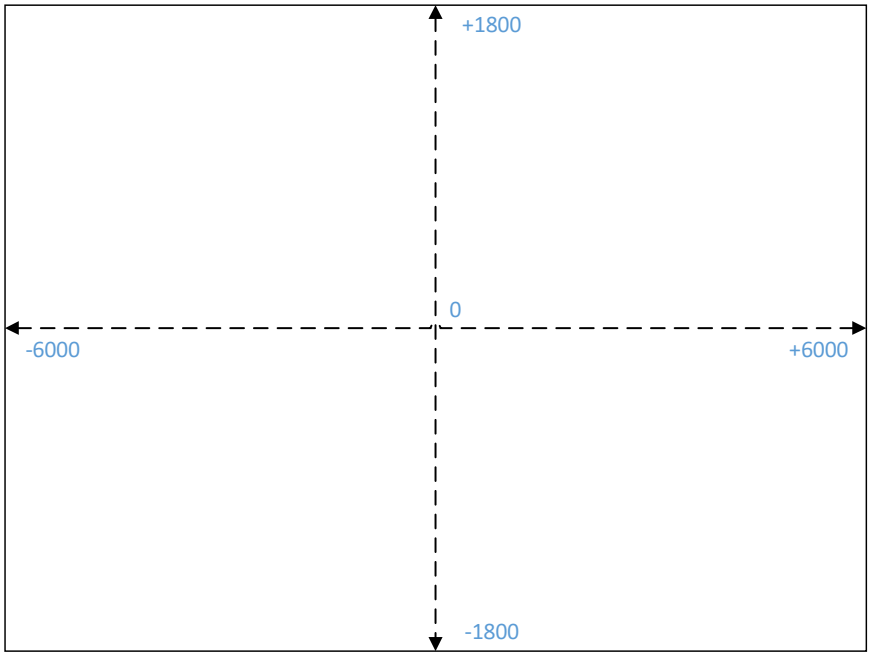


Abbildung 9 – Koordinatensystem

Damit konnte verhindert werden, dass die Grenzen überschritten werden und ein Fehlerzustand ausgelöst wird.

Das zweite Problem bestand darin, dass eine Befehlseingabe bei bereits laufender Ausführung eines Befehls zu einem Fehlerzustand führte, wodurch kein weiterer Befehl mehr angenommen werden konnte. Erst nach Zurücksetzen des Fehlers durch den Reset Befehl konnten neue Befehle angenommen werden.

Die Drive Unit ist in diesen Fall nicht in der Lage mitzuteilen, ob sich der Arm im Moment in Bewegung befindet oder nicht. Um dies zu verhindert musste Timing eingeführt werden, welches 1,5 Sekunden lang weitere Befehle blockiert.

5 Zusammenfassung und Ausblick

In der Arbeit konnte die Umsetzung der visuellen Vergabe eines Befehls an den industriellen Manipulator umgesetzt werden. Es wurde erreicht, dass der Roboter den Befehlen Folge leisten konnte. Dabei entstanden Probleme wie Lichtverhältnisse oder unpassende Größe des Objektes und die Veränderung des Gesamtbildes während einer Bewegung des Armes. Weiterhin erlaubt die Drive Unit keine Abfragen über den Aktuellen Wert eines Gelenkes. Während der Bewegung des Arms versetzten weitere Befehle das System in einen Fehlerzustand. Diese Probleme konnten durch Einsatz von Reglern, Testverfahren und internen Koordinaten gelöst werden. Durch die Regler ist der User in der Lage, Kamerarauschen bei schlechten Lichtverhältnissen anzupassen. Ebenso wird über Regler eingestellt, wie grob oder fein die Kontur des Objektes verarbeitet werden soll, um z.B. einer zusammengesetzten Bewegung im Bild entgegen zu wirken. Letztens kann ebenso über einen dritten Regler die minimale Objektgröße eingestellt werden, um Entfernung, aber auch bei Rauschen kleine Details umgehen lassen zu können. Durch Testverfahren konnte herausgestellt werden, dass viele Konturen im Schwellwertbild entstehen. Ebenso konnte herausgefunden werden, wann der nächste Befehl an den Arm gesendet werden kann.

Diese Arbeit hat sich hervorragend dazu geeignet, ein Projekt und verschiedene Ansatzmöglichkeiten zur Umsetzung zu erproben und zu erlernen. Anfangs bestand die Komplikation im Umgang mit der Sprache C++, Grafische Bedienelemente zu erstellen und Threads ordnungsgemäß zu betreiben. Dadurch konnte durch viel einarbeiten in C++ viel neues Wissen gewonnen werden. Ebenso war die mangelnde Erfahrung mit C# eine Motivation, C# höher zu erlernen, da vieles von Microsoft Visual Studio zur Umsetzung der Idee bereitgestellt wurde. Somit konnten komplexe Programmabschnitte leichter umgesetzt werden. Auch die Auseinandersetzung mit der Programmbibliothek konnte ein

gezieltes Wissen an Bildverarbeitung vermitteln. Da hier aber nur ein Teil der Möglichkeiten von OpenCv verwendet wurde, fokussiert sich dies auf das Maschinelle sehen. Schlussendlich ergab sich noch die Kommunikation zwischen Software und Roboter, was ebenso einen Interessanten Teil dieser Arbeit ausmachte.

Da diese Software frei unter Github mit Quellcode der Öffentlichkeit zur Verfügung gestellt wird war es ebenso meine Aufgabe, die Software erweiterbar zu gestalten. So zum Beispiel die Möglichkeit zur Verwendung mehrere Kameras, um bspw. Stereo Vision, das Sehen im 3D Raum, zu ermöglichen oder auch die Abdeckung eines 360 Grad Winkels. Weiterhin könnten mehrere Roboter gleichzeitig bedient werden.

Abschließend war diese Arbeit ein sehr guter Einstieg in komplexere Softwareentwicklung und eine gute Gelegenheit, in verschiedenen Bereichen Erfahrung zu sammeln.

Literaturverzeichnis

1. OpenCV Basic Thresholding Operations. [Online] [Zitat vom: 08. 02 2016.] http://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html.

2. OpenCV Finding contours in your image. [Online] [Zitat vom: 17. 02 2016.] http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html.

3. OpenCV Operations on Arrays. [Online] [Zitat vom: 17. 02 2016.] http://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html.

4. Robotics Exchange. *Mitsubishi Robot Manual RM-501.* [Online] [Zitat vom: 08. 02 2016.] http://www.mitsubishirobot.com/ta112.html.

5. Wikipedia. Industrieroboter. [Online] 30. 09 2015. https://de.wikipedia.org/wiki/Industrieroboter.