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 15. Februar 2016 _____

Unterschrift

Kurzfassung

Gegenstand der hier vorgestellten Arbeit ist eine Software zur Ansteuerung eines Industrie Roboterarmes. Dies erfolgt über zwei USB Webcams über diese dann die 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 Roboter ist in der heutigen Industrie bereits sehr verbreitet und seit langen schon Praxis. Ziel für die Zukunft ist immer mehr Standart- und Routineeingriffe durch Roboter zu ersetzten.

Im Laufe der Zeit seit dem Einsatz des ersten Programmierbaren Industrieroboter Ultimate im Jahre 1956 (1). Stiegen die Zahlen über den Einsatz neu installierter Industrieroboter Jährlich an. Stand 2011 waren es 166.000 (1). Eingesetzt werden Industrieroboter in den verschiedensten Bereichen. Von Montieren, Palettieren und Verpacken kann ein Industrieroboter je nach Programmierung in all vorstellbaren Bereichen angewendet werden. Da Maschinen mittlerweile in der Lage sind wie Menschen zu sehen, und je nach Erkennungsmuster einen bestimmten Befehl ausführen können. Entstehen immer mehr Einsatzmöglichkeiten für Maschinen.

Die Firma KUKA wirbt für Ihren Roboter KUKA KR AGILUS[1] mit einen Tischtennismatch zwischen dem Tischtennisprofi Timo Boll und Ihren Roboter. 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 eine davon, aber auch viele nützliche Vorgänge. Wie z.B. das man einer Maschine wie einen Industriearbeiter zeigt was sie zu tun hat, und die Maschine führt es aus. Dafür kommt das maschinelle Sehen in der Kombination von Roboter 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 eine Software zu entwickeln die es ermöglicht einen älteren Industrieroboter RM-501 das Erkennen von bewegten Objekten beizubringen. Dabei wird nicht der Roboter bewegt, sondern nur der Arm des Roboters. Welcher über eine Kamera verfügt. Um 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 Objekt und die Verfolgungen des Objektes realisiert. Des Weiteren soll die Software Erweiterbar sein, um neue mögliche Features einzubauen.

Motivierend an dieser Arbeit ist für den Autoren, dass diese Arbeit während der Ideenfindung und Realisierung es einen Spielerischen Freiheitsgrad besitzt. Und viele neue Einsatzmöglichkeiten und Ideen sich während des Projektes entwickelt haben. Ebenso kann durch den Einsatz mit C# und OpenCV ein hoher Erkenntnisfaktor gewonnen werden.

1.2 Gliederung der Arbeit

Die Arbeit gliedert sich in sechs Bestandteile, wobei der erste Teil die Einleitung darstellt. Hier wird ein kurzer Überblick in das Themengebiet gegeben und die Zielsetzung ebenso so wie die Motivation der Arbeit dargelegt.

Im zweiten Abschnitt werden die Grundlagen erläutert, die für das weitere Verständnis der Arbeit von Bedeutung sind.

Im dritten und vierten Teil der Arbeit wird nach einer Analyse der Anforderungen ein Konzept Vorgestellt was dann so Realisierbar wäre.

Im fünften Abschnitt wird die Implementierung anhand von grundlegenden Funktionalitäten durchgeführt. Dazu werden Funktionen vorgestellt und genutzt die in der OpenCV Bibliothek vorhanden sind. Ebenso dann die Resultierenden Funktionen die zum Ergebnis dieser Arbeit beigetragen haben. Auch wird die Implementierung des Roboterarmes erläutert.

Der sechste Teil und somit auch der letzte Teil dieser Arbeit beinhalten die Zusammenfassung und den Ausblick in das mögliche. So wird drauf eingegangen welche Realisierungsmöglichkeiten und Versuche am Anfang in Raum standen, ebenso wird über eine mögliche Erweiterbarkeit spekuliert.

2 Grundlagen

Um die Realisierung wahr werden zu lassen wird in diesem Punkt auf die beiden Programmiersprachen C++ und C# eingegangen, und ebenso die Einbindung der OpenCV Bibliothek. Auch wird erläutert wie eine Bewegung im Bild als Objekt wahrgenommen werden kann. Und es wird erläutert welche Möglichkeiten der Roboterarm von Mitsubishi bietet.

2.1 C++ und C#

Da es sich bei beiden um Objektorientierte Programmiersprachen handelt käme der Einsatz beider sprachen in Frage. Die Ausführung von C++ Codes ist auf allen Betriebssystemen möglich, hingegen C# das .NET Framework benötigt was z.B. unter Linux und MacOS durch Mono ab Framework Version 2.0 komplett unterstützt ist und teilweise auch Bereiche der Version 4.5 abdeckt. Hingegen C++ für eine GUI eine externe Bibliothek wie Qt benötigt und diese nicht so einfach editierbar ist wie Windows Form unter Visual Studio. Auch bei der Threadverwaltung übernimmt C# wesentliche Teile der Arbeit, das macht es natürlich einfacher und auch sicherer gegenüber Abstürze. In C++ wiederrum muss das wieder eine externe Bibliothek sogennant Boost eingebunden werden um eine einfachere Threadverwaltung zu realisieren. Somit mach es unterm Strich es dem jungen Entwickler einfacher und sicherer in C# zu Programmieren. Einziger Streitpunkt ist die Programmbibliothek OpenCV einzubinden. Wohin in C++ diese direkt über DLL erfolgt zu der eine ausführlichen Dokumentation zu Verfügung steht. Ist dies bei C# wesentlich umständlicher aber dennoch machbar. In C# stehen dafür Wrapper zur Verfügung die ebenso als Bibliothek eingebunden werden können. Solche wären z.B. Emgu und OpenCvCharp wohin jedoch Emgu eine Open Source License und eine Commercial License besitzt und OpenCvCharp komplett unter der BSD 3-Clause License frei zur Verfügung gestellt ist.

Emgu: http://www.emgu.com

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

Der Versuch die Realisierung umzusetzen und zu erproben geschah während des Projektes in beiden Sprachen. Für die Finale Realisierung wurde sich auf C# geeinigt. Da es sich um wesentlich schlichter und sicherer erwiesen hatte.

2.2 OpenCV

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

Unter http://opencv.org/ liegt dazu ebenso auch eine ausführliche Dokumentation bei, und ebenso gibt es viele Tutorials um den Einstieg in OpenCV schnell zu erlernen. So findet man viel Material zu den unterschiedlichsten Projekten der Online Community.

2.3 Erkennung von Objekten

Wird hier an der Stelle auf den eingesetzten Methoden mit OpenCV erklärt, da eine ausführliche Erläuterung von Maschinellen Sehen den Rahmen dieser Studienarbeit sprengen würde.

Da in OpenCV mit einer einfachen Webcam gearbeitet werden kann ist dies auch die Videoquelle für das Erkennen von Objekten im 2D Raum.

Angenommen die Videoquelle liefert dieses Ausgangsbild.

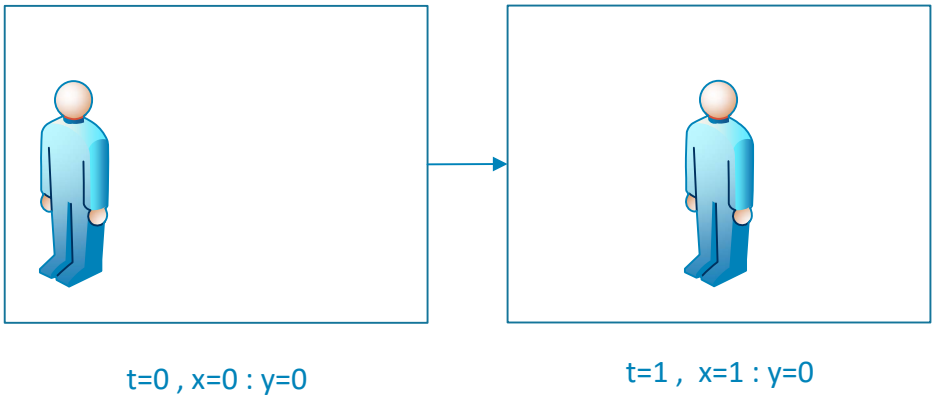


Abbildung 1 – Videoaufnahme einer Person

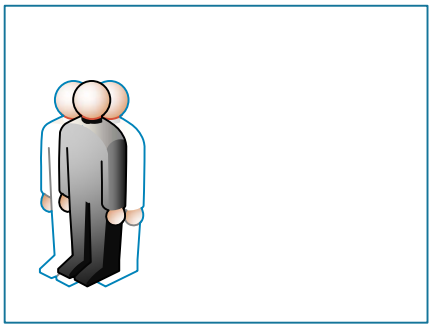
Diesen kann man auch auf der Zeitachse als den Zustand t=0 bezeichnen. Das Objekt in diesen Fall diese Person bewegt sich erstmal nicht. Das Bild ist zum Zeitpunk t=0 ist also Fix und im Raum bei x=0 und wird als Bild1 gespeichert.

Nun bewegt sich die Person von derzeitigen Standpunk im Raum zu einen Beliebigen Punkt im Raum innerhalb einer kurzen Zeitspanne also bei t=1 zu x=1 somit ergibt sich ein Bild2. Nun können beide Bilder verglichen werden und daraus kann nun die Summe der Absoluten Differenz gebildet werden. Sie dient als Maß für die Unterschiedlichkeit zweier Bilder.

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

Abbildung 2 – Formel Absolute Differenz

Dadurch kann ein drittes Bild erzeugt werden welches nun die Differenz beider Bilder enthält. Die Person befindet sich im dritten Bild nun zwischen x=0 und x=1.



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

Abbildung 3 – Differenz Bild

Der Betrachter nimmt dies nun als Bewegung wahr. Und diese Bewegung wird nun als auch Objekt identifizierbar da im dritten Bild jeder Punkt der Person bekannt ist. Dies geschieht so schnell das jede Bewegung sofort Verarbeitet werden kann. Es kann also für den Betrachter zur Veranschaulichung eine Linie durch diese Punkte im dritten Bild gezogen werden. Somit würde der Betrachter wenn er sich bewegt einen Rahmen um seinen Körper sehen wenn er diesen Bewegt. Zur einfacheren Verarbeitung wird Hauptsächlich in Grau-Bildern gearbeitet. Ebenso wird eine Methode der Segmentierung verwendet sogenannte Schwellwertbildung um die Differenz zweier Bilder einzublenden und alles andere im Bild einzuschwärzen. Hier wird im Bild der Apfel ähnlich wie beim Negativ Bild als weißes Objekt dargestellt. Dadurch kann einfacher die Positionsbestimmung des Objektes errechnet werden.

(1)

Abbildung 4 – Schwellwertbildung Apfel

2.4 Mitsubishi RM-501

Abbildung 5 – Movemaster RM-501

Der Mitsubishi Movemaster RM-501 ist ein 5-Achsen-Roboterarm mit einen Greifer der aus 2 Finger besteht. Das Gerät wurde ca. 1984 gebaut.

Eine Bewegung des Armes kann als Vektor beschrieben werden kann.

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

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 an die dirket über eine COM Schnittstelle zum Roboterarm übertragen werden.

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

3 Analyse

An dieser Stelle werden nun die Anforderungen an das System analysiert und gesammelt. Die später dann in der Implementierung des System umgesetzt werden. Aber ergebenen sich auch Probleme

3.1 Anforderungen

3.1.1 Bedienoberfläche

Die Bedienoberfläche für das System muss einfach sein, ebenso soll sie das Aktuelle Kamerabild zeigen sowie das Bild der Schwellwertbildung und das Finale Bild in dem das erkannte Objekt angezeigt wird.

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

Die Bedienoberfläche soll aus 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 Uhrsprung und auf die Home Position setzten. Auch soll die Möglichkeit bestehen Fehler der Drive Unit zurückzusetzten.

Durch die Bedienoberfläche kann Bildrauschen und Objektgröße gesteuert werden können.

3.1.2 Objekterkennung und Tracking

Das System soll ein bewegendes Objekt im 2D Raum erkennen und weiterhin erkennen können wenn dieses zum Stillstand kommt. Ebenso sollen die Positionsdaten vom Objekt vorhanden sein. Das Objekt wird dann getracked.

3.1.3 Ansteuerung des RM-501

Das System soll den Roboterarm bewegen können. Dieser soll sich immer zum erkannten Objekt ausrichten.

3.2 Problemszenarien

3.2.1 Drehachse des Roboter

Wenn der Roboter seine Grenze von 12000 = 360 Grad erreicht und darüber hinaus will. Setzt die Drive Unit die Error Flag. Das heißt das er an seine Mechanische Grenze gestoßen ist.

Dies kann passieren wenn das erfasste Objekt immer weiter nach Links oder Rechts vom Bild läuft.

3.2.2 Mehrere Objekte bewegen sich

Wenn sich mehrere bewegende Objekte sich im Bild befinden, werden auch beide als solche Erkannt. Das Problem dabei ist, dass sich das System nun ein Objekt aussuchen muss das getracked werden soll.

3.2.3 Lichtverhältnisse und Kamerarauschen

Wenn die Lichtverhältnisse ungünstig sind und / oder ein Kamerarauschen vorhanden ist z.B. durch eine schlechte Kamerahardware. Besteht das Problem das die Videoquelle für die Weiterverarbeitung schlecht ist.

4 Konzept

4.1 Architektur

4.1.1 Klassendiagramm

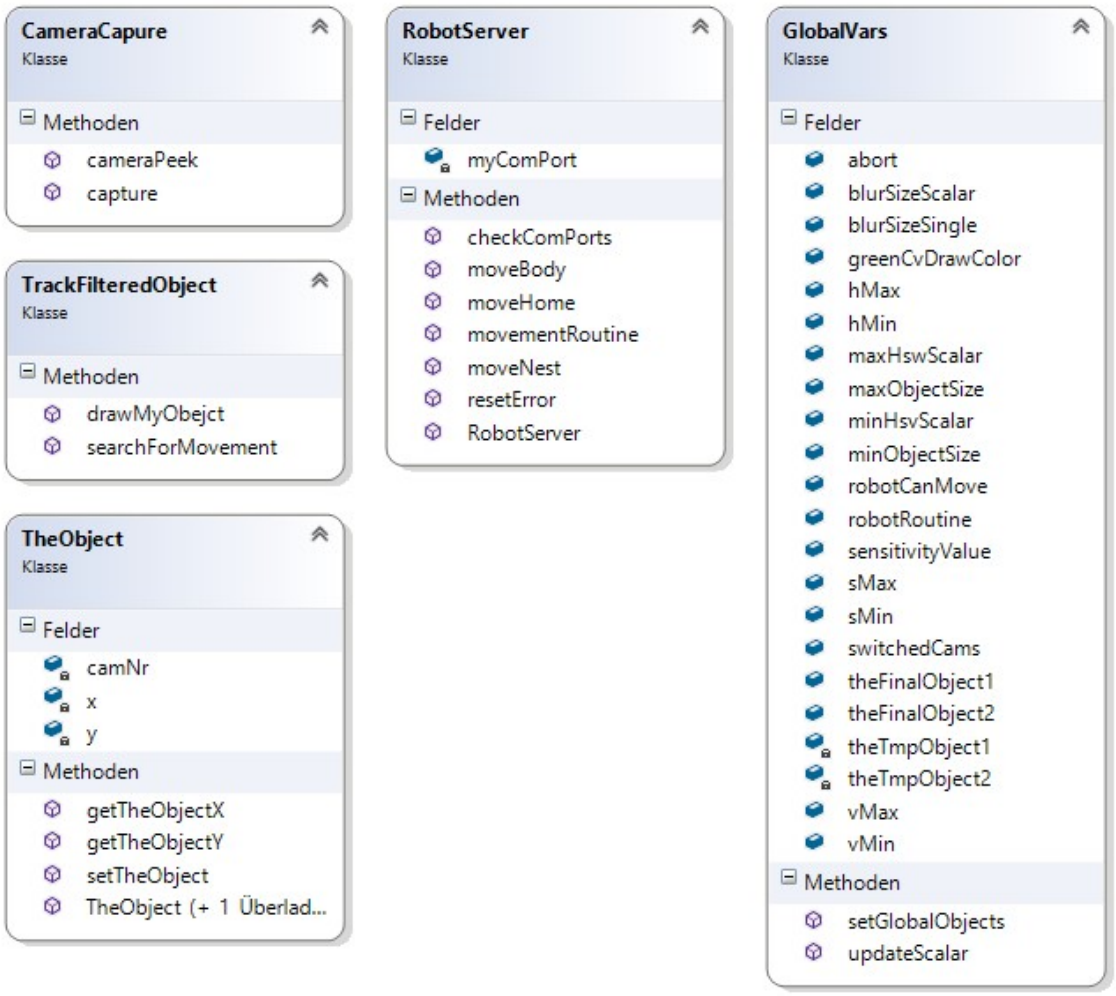


Abbildung 6 – Klassendiagramm

4.1.2 UML Aktivitätsdiagramm

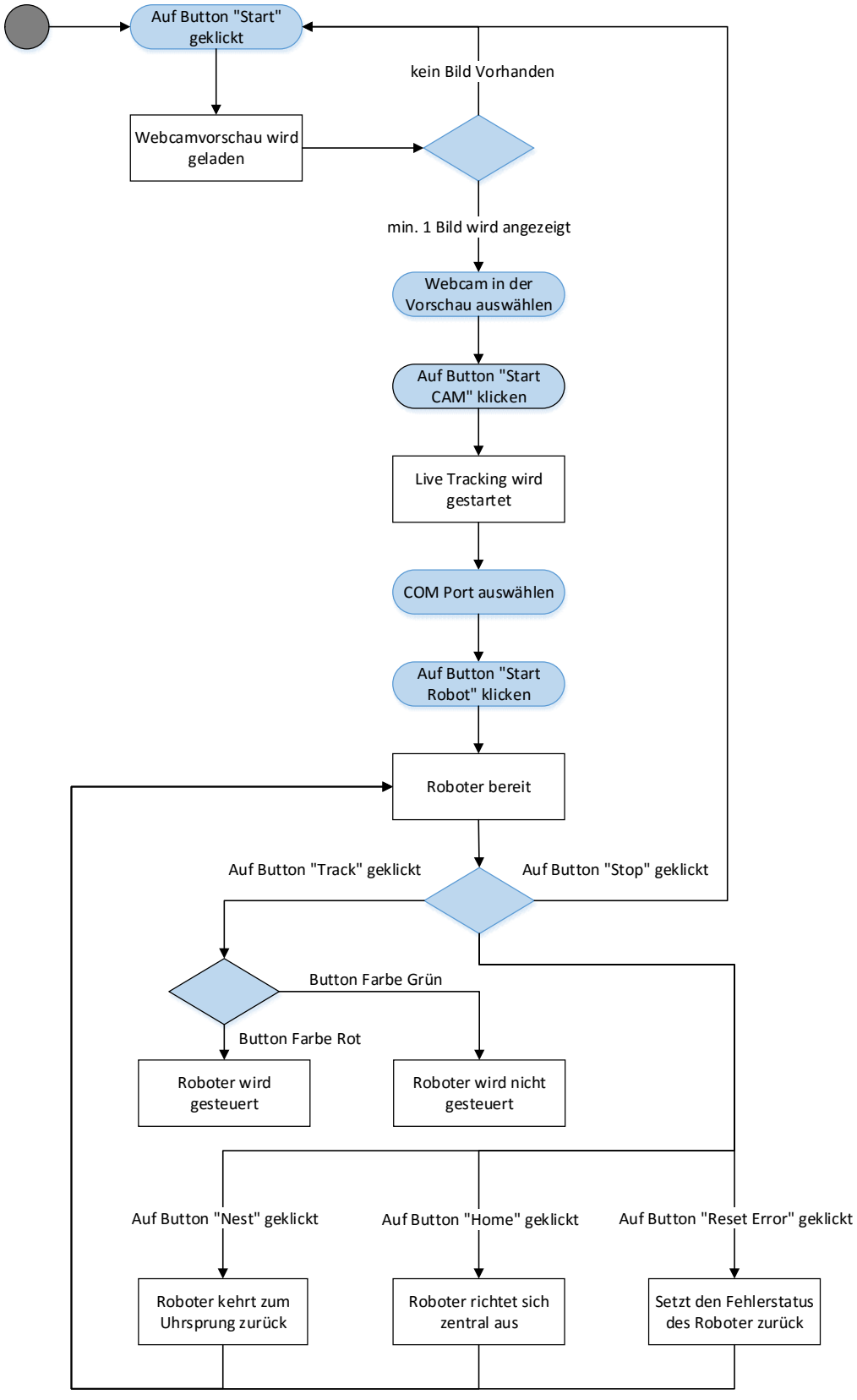


Abbildung 7 – UML Aktivitätsdiagramm

5 Implementierung

5.1 GUI

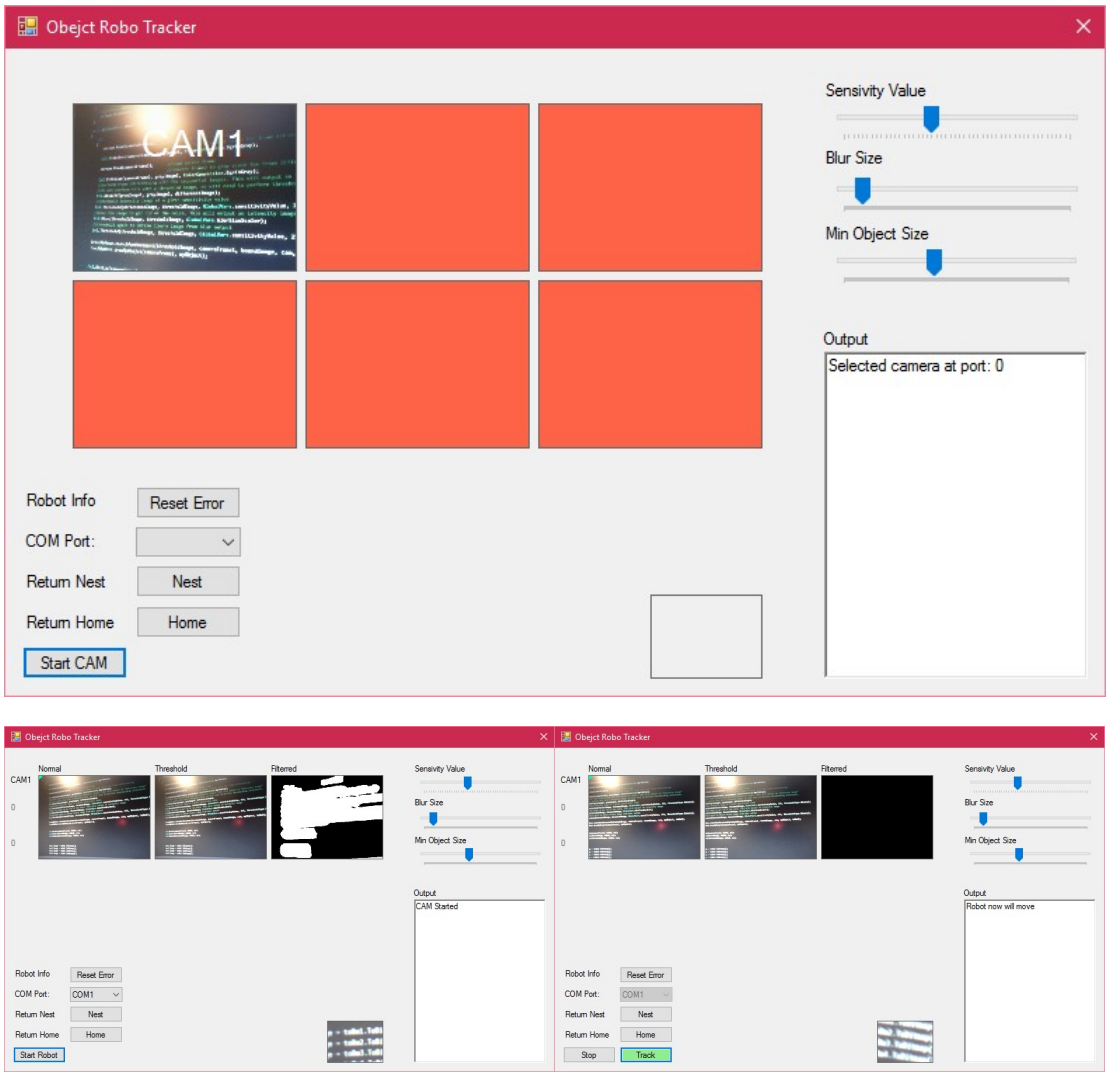


Abbildung 8 – GUI

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

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

5.2 Realisierung des Tracking

In der Klasse CameraCapture wird ein Video erzeugt. Es werden 2 aufeinanderfolgende Bilder genommen und gefiltert. Anschließend wird das Schwellwertbild weitergereicht um Objekte zu erkennen.

```
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 gerade ein Objekt gefunden worden wird der Ausschnitt davon in einem Template gespeichert und die Koordinaten davon gespeichert. Um nun wenn gerade keine Bewegung stattfindet nach dem zuvor gefundenen Objekt zu suchen. Wenn nun eine 80% Übereinstimmung gibt werden die Koordinaten diesem zugewiesen.

```
//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 { }

    }
```

5.3 Ansteuerung des RM-501

Zur Ansteuerung des Arms wird eine Verbindung über die COM-Schnittstelle benötigt. Über diese 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 davor noch überprüft werden ob sich der Arm noch in seinen Aschen-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();

}
```

5.4 Implementierungshürden

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

5.4.1 Tracking

Die ersten Probleme im stationären Fall waren, Bildrauschen und die Größe des Objektes. Dies konnte gelöst werden in dem der User über einen Regler das Kamerarauschen bei schlechten Lichtverhältnissen anpassen kann. 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. Und letztens kann ebenso über einen dritten Regler die minimale Objektgröße eingestellt werden, um Entfernung aber ebenso bei Rauschen kleine Details umgehen lassen.

Das zweite Problem war das sich bei einer Bewegung des Armes das gesamte Bild ändert. Es konnte nun nicht mehr identifiziert werden welcher Teil vom Bild nun anders war. Somit wurde zunächst zu viel auf einmal erkannt.

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

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

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

5.4.2 Roboter

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

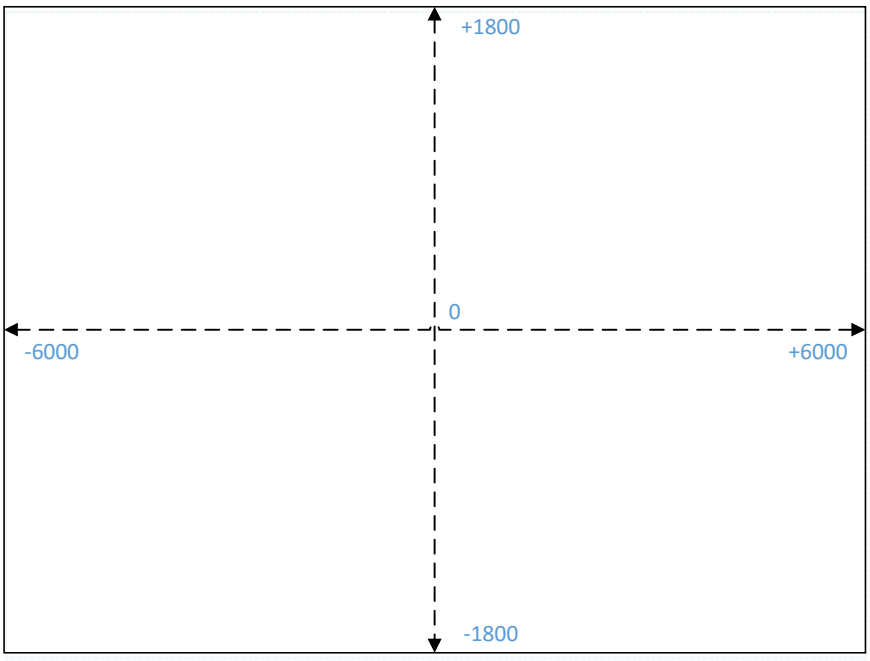


Abbildung 9 – Koordinatensystem

Nun kann verhindert werden dass die Grenzen überschritten werden und ein Fehlerflag ausgelöst wird.

Das zweite Problem bestand darin das wenn ein Befehl momentan in der Ausführung ist (der Arm bewegt sich nach Links) führt das Senden eines nächstes Befehles zu einen Fehlerflag wodurch kein weiterer Befehl mehr angenommen werden kann. Erst nach zurücksetzten des Flags, durch den Reset Befehl können neue Befehle angenommen werden. Die Drive Unit ist im diesen Fall nicht in der Lage zu sagen ob der Arm gerade in Bewegung ist. Um dies zu verhin-

dert musste ein Timing eingeführt werden. Das 1,5 Sekunden weitere Befehle blockiert.

6 Zusammenfassung und Ausblick

Diese Arbeit hat sich hervorragend dazu geeignet ein Projekt in verschiedene Ansatzmöglichkeiten zur Umsetzung zu erproben und zu erlernen. Denn am Anfang bestand die Komplikation im Umgang mit C++ Grafische Bedienelemente zu erstellen und Threads Ordnungsgemäß zu betreiben. Dadurch konnte durch viel einarbeiten in C++ viel neues Wissen und Lösungsmöglichen erworben werden. Ebenso mit wenig Erfahrung in C# war die Motivation noch höher C# zu erlernen, da vieles über Microsoft Visual Studio zur Umsetzung der Idee schon mitgeliefert wurde. Somit konnten komplexe Ideen leichter umgesetzt werden. Auch die Auseinandersetzung mit der Programmbibliothek konnte ein gezieltes Wissen an Bildverarbeitung vermittelt. 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 die Aufgabe die Software Erweiterbar so zu gestallten. Dass in Zukunft mehrere Kameras verwendet werden können um z.B. stereo vision also das sehen im 3D Raum zu ermöglichen, oder auch mit mehreren Kameras einen 360 Grad Winkel abzudecken. Oder auch mehrere Roboter gleichzeitig zu betreiben.

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

Literaturverzeichnis

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

2. opencv documentation. [Online] [Zitat vom: 08. 02 2016.] http://docs.opencv.org/2.4/_images/Threshold_Tutorial_Theory_Example.jpg.

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