



**Hochschule für Angewandte
Wissenschaften Hamburg**
Hamburg University of Applied Sciences

Fakultät Technik und Informatik

Department Informations- und Elektrotechnik - Sommersemester 2016

Verbundprojekt

Prof. Dr.-Ing. Jochen Maaß

Abschlussbericht

Gewerk 5: Positionserfassung und Uhrzeitsynchronisation

Projektbearbeitungszeitraum: 26. November 2015 - 17. März 2016

Berichtbearbeitungszeitraum: 17. März 2016 - 12. Juni 2016

Gruppe

Benjamin Willberger
Mike Zander

MA.-Nr. 2268415
MA.-Nr. 2268453

Inhaltsverzeichnis

1 Einleitung	5
2 Positionserkennung	6
2.1 Vorbetrachtung	6
2.1.1 Die Aruco Bibliothek	6
2.1.2 Zusammensetzung der Aruco Bibliothek	6
2.1.3 Eignungsprüfung der Aruco Bibliothek	9
2.2 Umsetzung	12
2.2.1 Markerbibliothek, Größe und Anordnung	12
2.2.2 Programmablauf und Funktion des Hauptprogramms	17
2.2.3 Winkel- und Positionsbestimmung	19
2.2.4 Bildvorverarbeitung, Übergabe und Aruco Parameter	22
2.2.5 Informationsauswertung	26
2.3 Fehlerbetrachtung und ausgewählte Lösungen	29
2.3.1 Fehlende Marker-Erkennung	29
2.3.2 Fehlerhafte Zuordnung der ID	29
2.3.3 Anfahrt der Stationen	30
2.3.4 Ausfall eines Markers	30
2.3.5 Winkelbestimmungen im Übergangsbereich	31
2.4 Auswertung	32
3 Uhrsynchronisation	38
3.1 Theorie und Verfahren zur Synchronisation der Uhren	38
3.2 Umsetzung der Uhrsynchronisation	40
3.2.1 Uhrauswahl	40
3.2.2 Methoden nach IEEE 1588 zum Synchronisieren der Uhren	42
3.2.3 Gegenüberstellung der Methoden zur Uhrsynchronisation	45
3.3 Programmaufbau	46
3.3.1 Anpassungen in Robot-Detection	48
3.3.2 Simulink Programm	50
3.4 Auswertung	53
3.4.1 Genauigkeitstest der Uhr auf dem Roboter	53
3.4.2 Test des Moving Average Filters	54
3.4.3 Verlauf der Uhr auf dem Roboter bei zeitweisem Ausfall der Synchronisation	56
3.4.4 Test des Alters der letzten Nachricht bei schwankender Bearbeitungszeit des PCs	57
4 Fazit	59
5 Anhang	61

Abbildungsverzeichnis

1	Bearbeitungsschema und Zuordnung zwischen Aufnahme, Marker und ID. [GJMSMCRC14]	6
2	Marker mit der ID null und eins, bei identischer Struktur durch eine Fehlerkennung. [GJMSMCRC14]	7
3	Darstellung eines Markers mit 4x4 Bits unter rotatorischem Einfluss und farblich markierten Gruppen. [GJMSMCRC14]	8
4	Schematische Darstellung eines Ablageplatzes an einer Station mit einem Werkstück (rot).	10
5	Vektorielle Darstellung der Markeranordnung V_{Marker} und der Länge eines Pixels V_{Pixel} und dem daraus resultierendem Winkel α	10
6	Schemata des Roboters, sowie dessen Länge a . Der schwarze Block repräsentiert eine Markeranordnung mit der Länge b	11
7	Schematische Darstellung einer Markeranordnung und deren Zusammensetzung aus vier Markern. Die braun gestrichelte Linie ist nicht Bestandteil der gedruckten Marker und dient der Darstellung des Markerrandes.	13
8	Gegenüberstellung der Winkel zwischen den Marker-ID's bei sich drehendem Roboter. In Abbildung a für zwei und in Abbildung b für drei Marker	14
9	Gegenüberstellung der Winkeldifferenz für mehrmalige Drehungen. In Abbildung a für zwei und in Abbildung b für drei Marker.	16
10	Gegenüberstellung des alten ([BBT14] links und des neuen (rechts) Hauptprogrammablaufplans. Grün hervorgehoben sind Änderungen gegenüber dem alten Hauptprogramm. Rot gekennzeichnet sind Änderungen in dem Unterprogramm.	17
11	UML-Diagramm der veränderten imgtask-Klasse	18
12	Vektoren der vier Ecken eines Markers im Arbeitsfeld (rot), sowie dessen Mittelwert (grün).	19
13	Vektoren der Mittelwerte zweier Marker (grün), dessen Schnittwert (rot), sowie der Differenz zwischen dem Schnittwert und den einzelnen Mittelwerten der Marker (blau).	20
14	Nummerierte Ecken eines Markers (blau), sowie deren Vektoren und den daraus resultierenden Winkeln (grün). Rot markiert ist die Ausrichtung des Markers gegenüber dem Koordinatenursprung.	21
15	Programmablaufplan der Bildvorverarbeitung zur Ermittlung der AOI's.	22
16	Schematische Darstellung der Vorverarbeitungsschritte in der Bildverarbeitung.	25
17	Ablaufplan in der imgtask zur Zuordnung zwischen Winkel, Offset, Positionsvektor und Roboter.	27
18	Zeitlicher Positionsverlauf des zweiten Roboters unter ausgeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ zwischen dem alten Algorithmus und dem neuen System.	32
19	Zeitlicher Positionsverlauf des zweiten Roboters unter ausgeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ zwischen dem verbesserten Algorithmus und dem neuen System.	33
20	Zeitlicher Positionsverlauf aller Roboter unter ausgeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ unter Verwendung des verbesserten Algorithmus.	34

21	Zeitlicher Positionsverlauf aller Roboter unter ausgeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ unter Verwendung des neuen Systems.	34
22	Zeitlicher Positionsverlauf des zweiten Roboters unter eingeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ zwischen dem alten Algorithmus und dem neuen System.	35
23	Zeitlicher Positionsverlauf aller Roboter mit eingeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ und unter Verwendung des verbesserten Algorithmus.	36
24	Zeitlicher Positionsverlauf des zweiten Roboters in einem Kurvenabschnitt unter ausgeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ unter Verwendung des ursprünglichen Algorithmus und dem neuen System.	37
25	Delay und Offset-Berechnung nach IEEE 1588 [Wei04, Seite: 2]	39
26	Uhren in Simulink	41
27	Telegrammablauf Methode 1	42
28	Telegrammablauf Methode 2	43
29	Telegrammablauf Methode 3	44
30	Programmablaufplan	47
31	UML-Diagramm der Methoden der Klasse myudp	49
32	Programmaufbau Simulink-Block	51
33	Verlauf des Uhren-Offsets bei einer einmal gestellten Uhr*	54
34	Verlauf des Alters der letzten Nachricht nach drei verschiedenen Ansätzen	55
35	Verlauf des Alters der letzten Nachricht, wenn Synchronisierung aussetzt	56
36	Verlauf des Alters der letzten Nachricht, wenn sich die Bearbeitungszeit des Kamera-PCs ändert	57



1 Einleitung

In dem Verbundprojekt des Masters Automatisierungstechnik besteht die Aufgabe der Studierenden darin, einen Produktionszyklus zu realisieren. Hierbei stehen den Studierenden vier autonome Roboter zur Verfügung, die dem Transport von Werkstücken zwischen vier Stationen dienen sollen. Damit ein Roboter autonom in seiner Umgebung agieren kann, benötigt er seine aktuelle Position in seinem Arbeitsfeld. Hierzu besitzen die Roboter auf ihrer Oberfläche reflektierende Erkennungsmuster. Diese Muster werden durch LED-Lampen beleuchtet und mithilfe von sechs Deckenkameras erfasst. Eine bestehende Software auf einem Zentralrechner wertet diese aus und übermittelt mit einem UDP-Telegramm die Positionen der Roboter. Das Muster wird aus der Anordnung mehrerer Kreise gebildet und unterscheidet sich anhand der jeweiligen Roboternummer. Im Zuge dieses Verbundprojektes soll das Gewerk fünf, eine alternative Erkennungsmethode untersuchen und dem bisherigem System gegenüberstellen. Ziel dieser Untersuchungen ist es ein stabileres, sowie unter eingeschalteter Raumbeleuchtung funktionierendes System zu entwickeln. Aufgrund der Tatsache, dass die Roboter im vorherigen Semester (Jahrgang 2015) um 2D-Informationsträger der Bibliothek Aruco erweitert wurden, bietet sich dieses System zur Eignungsprüfung an. Hierdurch sollen die Positionserkennungssymbole und der Informationsträger auf ein System mit duality Funktion reduziert werden.

Eine weitere Aufgabe an das Gewerk fünf besteht in der Umsetzung einer Uhrsynchrosynchronisation. Da es bisher nicht möglich ist mithilfe einer Uhr auf dem Roboter festzustellen, wie alt die per UDP eingetroffenen Positionen sind, wird in dieser Projektarbeit eine Möglichkeit zur Lösung dieses Problems erarbeitet. Als Uhr-Master soll dabei der Kamera-PC dienen, da dieser bereits die Roboterpositionen per UDP-Broadcast verteilt. Die Aufgabe des Projekts besteht in der Untersuchung in Hinblick auf die Erweiterung der Verteilung der Positionen und des Zeitstempels, um eine Synchronisation der Roboter zu erreichen. Da die Auswertung der Roboterposition auf dem Kamera-PC und die Verteilung per UDP-Telegramm ein Vorgang von wenigen Millisekunden ist, muss eine Genauigkeit von 5 – 10 ms erreicht werden. Die untere Grenze ist durch den Applikationsaufbau gesetzt, da eine Solvertime von fünf Millisekunden verwendet wird. Es gilt die Roboter auf eine nicht näher spezifizierte Größe zu synchronisieren. Ein Beispiel für eine solche gemeinsame Größe ist die Uhrzeit.

2 Positionserkennung

2.1 Vorbetrachtung

2.1.1 Die Aruco Bibliothek

Die Aruco Bibliothek repräsentiert eine minimale C++ Bibliothek zur Erstellung und Erkennung von 2D-Informationsträgern. Die Entwicklung erfolgte durch S. Garrido-Jurado, R. Muñoz-Salinas, F.J Madrid-Cuevas und M.J. Marín-Jiménez an der Universität Córdoba in Spanien. Sie steht unter den Plattformen Windows, MacOS und Linux zur Verfügung und ist quelloffen. [Rün11, Seite: 6] Die Bibliothek findet in diversen Augmented Reality Anwendungen, die durch steigendes Interesse in der Industrie und dem Consumerbereich immer mehr an Bedeutung gewinnen. Das Konzept der zu entwickelnden Positionserkennung sieht eine Platzierung der Marker auf den Robotern vor, die anschließend durch den Algorithmus detektiert werden.

2.1.2 Zusammensetzung der Aruco Bibliothek

Die Aruco Bibliothek umfasst eine diverse Anzahl an Marker Bibliotheken, die sich durch ihre Anzahl an Markern unterscheiden. Ein Marker setzt sich dabei aus einer bestimmten Anzahl an schwarzen und weißen Feldern zusammen. Diese werden in quadratischer Form angeordnet und bilden gemeinsam eine Matrix. Für die korrekte Erfassung der Marker, sowie der Zuordnung zwischen der Matrix und der dazugehörigen ID, sind die Marker Distanz und die innere Distanz eines Markers ausschlaggebend. Die Marker Distanz beschreibt dabei ein Maß mit dem sich zwei Marker in einer Bibliothek voneinander unterscheiden. Je höher dieses Maß ist, desto stabiler ist die korrekte Zuordnung eines Markers zu seiner ID. Die innere Marker Distanz beschreibt dabei das Maß mit dem sich der eigene Marker unter rotatorischem Einfluss von sich selber unterscheidet. Einem Marker dessen Inhalt z.B aus einem weißen Quadrat besteht, kann keine Orientierung zugeordnet werden, da sein Erscheinungsbild unter Rotation immer identisch bleibt. Beide Faktoren, sowie ein Beispiel der Fehlerkennung sind der Abbildung 1 zu entnehmen.

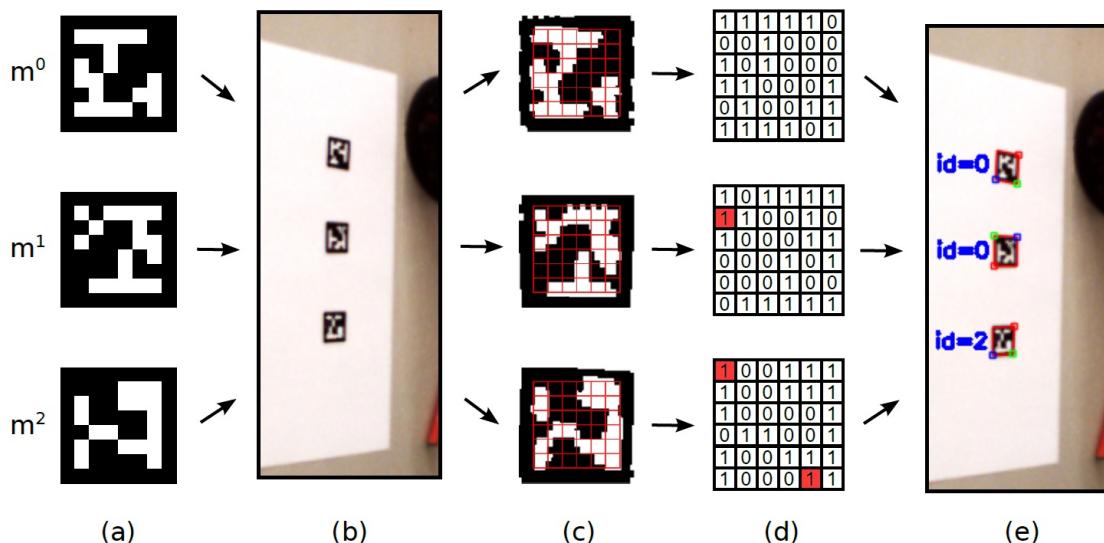


Abbildung 1: Bearbeitungsschema und Zuordnung zwischen Aufnahme, Marker und ID.
 [GJMSMCRC14]

Der Abbildung 1 zu entnehmen ist der Schemenhafte Prozessablauf zwischen Aufnahme und Zuordnung der Marker ID. In Schritt a sind drei unterschiedliche Marker m mit der ID n zu erkennen. Die Marker Distanz zwischen den ersten beiden Markern beträgt $D(m^0, m^1) = 1$, zwischen den Markern null und zwei $D(m^0, m^2) = 5$ und zwischen den letzten beiden $D(m^1, m^2) = 6$. In Abschnitt b erfolgt die Aufnahme der Marker durch eine Kamera. Diese Informationen werden in Schritt c an den Algorithmus übergeben, dessen Interpretation in Teil d zu sehen ist. Hierbei ist anzumerken, dass die rot markierten Bits falsch erkannt werden und in den tatsächlichen Markern (a) nicht gesetzt sind (weiße Bits). Da die Distanz zwischen den Markern m^0 und m^1 nur ein Bit beträgt, kommt es zu einer Fehlinterpretation. Diese Fehlinterpretation wird durch Abbildung 2 deutlicher.

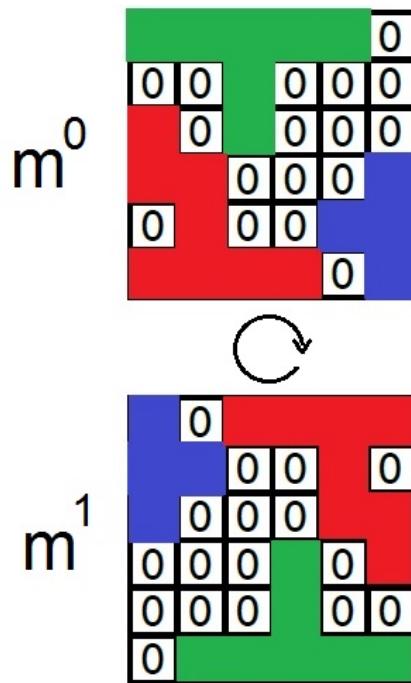


Abbildung 2: Marker mit der ID null und eins, bei identischer Struktur durch eine Fehlerkennung.
 [GJMSMCRC14]

Zu sehen sind zwei identische, jedoch um 180° gedrehte Matrizen. Die Blöcke der gesetzten Bits sind zur Veranschaulichung farblich markiert. Wäre die Marker Distanz zwischen m^0 und m^1 größer als ein Bit, käme es zu keiner Verwechslung. Ein solches Beispiel ist unter Betrachtung der Marker m^1 und m^2 in Abbildung 1 zu erkennen, in der zwar zwei Bits des zweiten Markers falsch detektiert werden, die Zuordnung zwischen Marker und ID jedoch weiterhin korrekt ist. Dieses Verhalten lässt sich mathematisch durch die Hammingdistanz beschreiben. Sie ist ein Maß für die Unterschiedlichkeit einer Zeichenkette. Als Beispiel dienen die zwei Zeichenketten $a : 112344$ und $b : 225544$, dessen Hammingdistanz vier beträgt. Das Ergebnis resultiert aus der Anzahl unterschiedlicher Stellen in den Zeichenketten, die in dem Beispiel durch die ersten vier Ziffern repräsentiert werden. Für die Bildung eines Markers werden je vier Bits zu einem Bereich zusammengefasst. Die Summierung bestimmter Bereiche ergibt dann einen Marker.

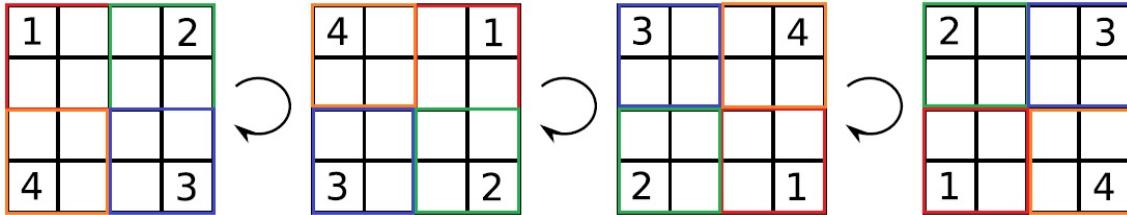


Abbildung 3: Darstellung eines Markers mit 4x4 Bits unter rotatorischem Einfluss und farblich markierten Gruppen. [GJMSMCRC14]

Die Abbildung 3 zeigt einen Marker mit 4x4 Bits, bestehend aus vier farblich markierten Bereichen. Jeder Bereich besitzt dabei vier Bits, die wiederum ein Quartett darstellen. Zu erkennen ist, dass der Marker rotiert, was eine Verschiebung der Bereiche zur Ursache hat. Um einen Marker mit einer hohen inneren Distanz zu erzeugen sind Konstellationen zu wählen, die trotz der Rotation eine hohe Hammingdistanz besitzen. Hierzu werden die möglichen Zusammensetzungen der Bits in einem Quartett betrachtet und die Hammingdistanz zu jeder möglichen Rotation errechnet. Die Ergebnisse sind der Tabelle 1 zu entnehmen.

Gruppe	Quartett	Hammingdistanz		
		90°	180°	270°
Q_1	0000,111	0	0	0
Q_2	1000,0100,0010,0001,1110,0111,1011,1101	2	2	2
Q_3	1100,0110,0011,1001	2	4	2
Q_4	0101,1010	4	0	4

Tabelle 1: Hammingdistanzen der Gruppen Q_1 bis Q_2 unter bestimmten Rotationswinkeln. [GJMSMCRC14]

In der Tabelle 1 werden dabei Quartette mit identischer Hammingdistanz in Gruppen zusammengefasst. Zur Erläuterung ist die erste Gruppe zu betrachten. Sind die vier Felder einer Gruppe mit Einsen gefüllt (Q_1), lässt sich diese Gruppe durch Rotation nicht voneinander unterscheiden, weshalb die Hammingdistanz Null beträgt. Wird hingegen Gruppe Q_4 betrachtet und die Bits einer Gruppe werden abwechselnd gesetzt und nicht gesetzt, beträgt die Hammingdistanz für 90° und 207° vier, da sich in diesem Fall alle vier Bits voneinander unterscheiden. In Abbildung 3 zusätzlich zu erkennen ist, dass sich die Gruppen gegenseitig nicht beeinflussen, da sie örtlich voneinander getrennt sind [GJMSMCRC14]. Somit erzeugt eine Zusammensetzung von Gruppen mit hoher Hammingdistanz eine hohe innere Marker-Distanz, die wiederum eine sichere Zuordnung zwischen Marker und ID erlaubt. Eine hohe innere Marker Distanz wird zum Beispiel erreicht, wenn ein Marker aus den Gruppen Q_3, Q_3, Q_4, Q_3 gebildet wird. Die daraus resultierende Hammingdistanz ist der Tabelle 2 zu entnehmen.

Bereich	Gruppe (vgl. Tabelle 1)	Hemmingdistanz		
		90°	180°	270°
1	Q_3	2	4	2
2	Q_3	2	4	2
3	Q_4	4	0	4
4	Q_3	2	4	2
Totale Distanz		10	12	10
Innere Marker Distanz		$\min(10,12,10) = 10$		

Tabelle 2: Hemmingdistanzen der Gruppen Q_3 , Q_4 sowie dem daraus gebildeten Marker unter bestimmten Rotationswinkeln. [GJMSMCRC14]

Bei einem Rotationswinkel von 180° ist eine minimale Hemmingdistanz von Zehn zu sehen. Anhand dem erläuterten Prinzip lässt sich begründen, warum Bibliotheken mit hoher Anzahl an Bits und geringem Umfang an Markern in der Erkennung sicherer sind. Je mehr Bits zur Verfügung stehen, desto mehr Gruppen mit einer hohen Hemmingdistanz existieren. Da für jeden Marker eine unterschiedliche Konstellation von Gruppen zu wählen ist, sind die Marker mit einer hohen Hemmingdistanz nur begrenzt verfügbar. Sind die sicheren Gruppen verbraucht, werden für weitere Marker die unsicheren Gruppen verwendet, die eine Reduzierung der Hemmingdistanz verursachen. Aus diesem Grund ist für die Erkennung der Roboter eine Bibliothek mit geringer Anzahl an Markern zu verwenden, da hierdurch die Hemmingdistanz höher ist.

2.1.3 Eignungsprüfung der Aruco Bibliothek

Vor Umsetzung und Implementierung der Aruco Bibliothek ist eine erste Einschätzung über die Einsatzfähigkeit zu bilden. Schwerpunkt der Betrachtung ist der Winkel des Roboters sowie dessen Position, die beide innerhalb des Toleranzbereichs liegen müssen. Für eine Analyse der Eignung werden im ersten Schritt die Anforderungen definiert. Diese sind auf der Grundlage der Stationen zu bilden, da hier die höchsten Anforderungen an den Winkel und die Position gestellt werden.

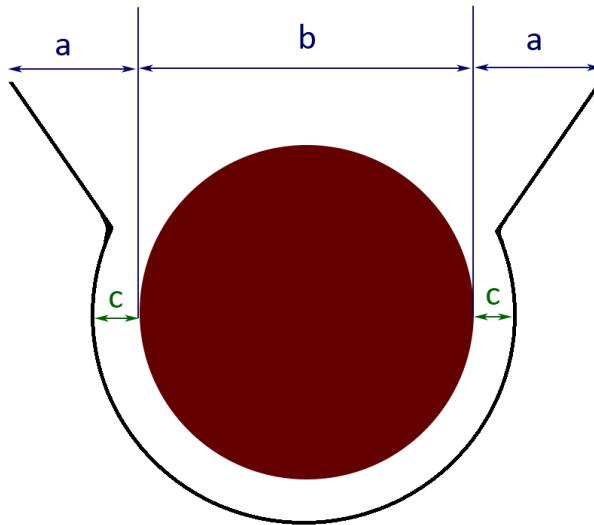


Abbildung 4: Schematische Darstellung eines Ablageplatzes an einer Station mit einem Werkstück (rot).

Der Toleranzbereich des Winkels wird durch den Spielraum des Werkstückes innerhalb des Ablageplatzes gebildet. Dieser beträgt in Anlehnung an Abbildung 4 $c = 2.5 \text{ mm}$ sowie $a = 5 \text{ mm}$. Damit das Werkstück korrekt in den Ablageplatz passt darf der aus dem Toleranzwinkel des Roboters resultierende Fehler nicht mehr als 5 mm betragen. Zur Überprüfung des Toleranzbereiches muss das Übersetzungsverhältnis zwischen Pixel und metrischem System bekannt sein. Hierfür werden die Pixel eines DIN A4 Blattes auf der Höhe des Roboters gemessen und entsprechend der metrischen Maße umgerechnet. Die Aufnahme ergibt, dass ein DIN A4 Blatt eine Länge von $b = 68 \text{ px}$ und eine Höhe von $a = 48 \text{ px}$ besitzt. Die metrischen Daten betragen für die Länge $b = 297 \text{ mm}$ und für die Höhe $a = 210 \text{ mm}$. Durch die Umrechnung mithilfe des Dreisatzes, ergibt sich ein resultierendes Übersetzungsverhältnis von $1 \text{ px} \hat{=} 4,3 \text{ mm}$.

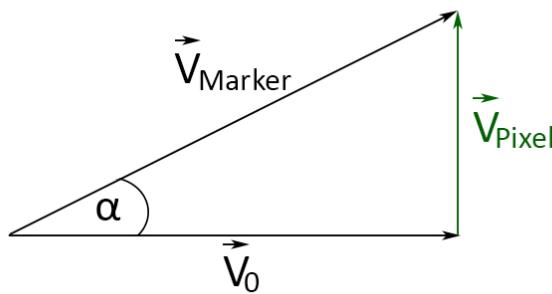


Abbildung 5: Vektorielle Darstellung der Markeranordnung V_{Marker} und der Länge eines Pixels V_{Pixel} und dem daraus resultierendem Winkel α .

Wird für V_{Pixel} die Länge für ein Pixel von $4,3 \text{ mm}$ eingesetzt und die maximale umsetzbare Länge einer Anordnung (Annahme der Länge eines DIN A4 Blattes) für V_{Marker} , resultiert nach den Winkelgesetzen eine Abweichung von $\alpha_{max} = 0,9^\circ$. Da eine

Verkleinerung der Markerlänge (V_{Marker}) in der Umsetzung eine Vergrößerung des Winkels zur Ursache hätte, handelt es sich bei dem Winkel um die minimal mögliche Abweichung. Nachdem der realisierbare Winkel bekannt ist kann dieser zur Berechnung der Abweichung in Y-Richtung verwendet werden.

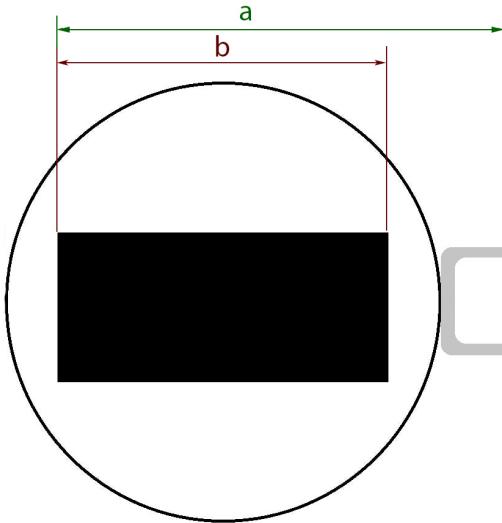


Abbildung 6: Schemata des Roboters, sowie dessen Länge a . Der schwarze Block repräsentiert eine Markeranordnung mit der Länge b .

Der Roboter inklusive dem Greifer besitzt eine Länge von $a = 350$ mm (vgl. Abbildung 6). Besitzt die Markeranordnung eine Länge von $b = 297$ mm, kann mit dem Winkel von 0.9° eine maximale Abweichung von

$$a \cdot \tan(\alpha) = 350 \text{ mm} \cdot \tan(0,9^\circ) = 5,5 \text{ mm} \quad (1)$$

erreicht werden. Die mögliche Abweichung liegt damit 0,5 mm über dem erlaubten Wert. Wird die Marker Anordnung nicht am Ende des Roboters positioniert, sondern näher am Greifer, hätte dies eine Reduzierung der Länge a (vgl. Abbildung 6) zur Ursache, wodurch das Ergebnis in Gl. 1 verringert werden kann. Es ist anzunehmen, dass das System die Anforderungen erfüllt, sofern die Anordnung der Marker einem DIN A4 Blatt entspricht und die Marker in Richtung des Greifers positioniert werden. Weiterhin besitzen die Ablagestationen einen vergrößerten Einfahrtsbereich (vgl. Abbildung 4), sodass der halbe Millimeter durch die Station kompensiert werden kann, wenn diese das Werkstück in die richtige Position führt. Da es sich um eine Annahme handelt ist die Eignung durch die Umsetzung zu erproben.



2.2 Umsetzung

2.2.1 Markerbibliothek, Größe und Anordnung

Die Aruco Bibliothek weist eine diverse Anzahl an Markerbibliotheken auf. Diese unterscheiden sich in der Anzahl an unterschiedlichen Markern, sowie in deren Struktur, Auflösung und Informationsgehalt eines Markers. Die Stabilität eines Markers wird dabei wesentlich durch die Anzahl an ID's der Bibliothek, sowie der Größe eines Markers bestimmt. Hieraus resultiert der Unterschied zwischen zwei Markern, der zur fehlerfreien Erkennung möglichst groß gewählt werden muss (vgl. Kapitel 2.1.2). Daher empfiehlt es sich eine Bibliothek zu verwenden, die eine auf die Anwendung angepasste Anzahl an Markern besitzt [Gar]. Die Aufgabenstellung erfordert die Erkennung von vier Robotern, weshalb eine möglichst kleine Bibliothek zu verwenden ist. Hierfür kommen die nachstehenden Bibliotheken in Frage.

Name	Anzahl der Bits	Anzahl der Marker
DICT_4X4_50	4x4	50
DICT_5X5_50	5x5	50
DICT_6X6_50	6x6	50
DICT_7X7_50	7x7	50
DICT_Aruco_ORIGINAL	5x5	1024

Tabelle 3: Auflistung relevanter Bibliotheksgrößen, sowie deren Größe und die Anzahl an Bits je Marker

Entgegen der Empfehlung der OpenCV Quelle und der Vorbetrachtung fiel die Entscheidung zugunsten der DICT_Aruco_ORIGINAL Bibliothek. Diese wird bereits in der Tabletapplikation verwendet und bietet somit den Komfort, auf allen Systemen genutzt zu werden. Sofern in Zukunft eine alternative Bibliothek Anwendung finden soll, kann diese in der imgtask.cpp Zeile 48 geändert werden:

```
//Use Aruco Dictionary (Original standard Dictionary) and Parameters
```

```
aruco::Dictionary dictionary = aruco::getPredefinedDictionary(aruco::
```

```
DICT_Aruco_ORIGINAL);
```

Nach Auswahl einer geeigneten Bibliothek werden die Größe und die Anordnung der Marker bestimmt. Ausgeschlossen wird hierbei ein einziger großer Marker je Roboter. Die Entscheidung begründet sich mit der Ausfallsicherheit innerhalb der Kameraübergänge und den Stationsbereichen, in denen die Kabelführungen der Stationen die Erkennung unterbinden würden. Wird ein großer Marker verwendet, müssen die Übergänge der Kameras vergrößert werden, sodass der Marker mindestens in einer Kamera erkannt wird. Aufgrunddessen reduziert sich das Arbeitsfeld und eine neue Einmessung bzw. Kalibrierung ist notwendig. Kurzfristige Ausfälle unterhalb der Kabelführungen werden mit zwei Markern nicht unterbunden jedoch reduziert. Somit ist eine Anordnung mit zwei oder mehr Markern zu finden. Tabelle 4 zeigt das beobachtete Verhalten der Markeranordnungen und Größen innerhalb einer selbst programmierten Testsoftware. Hierbei repräsentiert *a* die Anzahl Marker in horizontaler Ebene, *b* die Anzahl Marker in vertikaler Ebene, *c* die Länge eines Markers, *m* den Abstand zwischen den Markern und *r* den Abstand zum Rand. Eine Beispielanordnung zeigt die Abbildung 7.

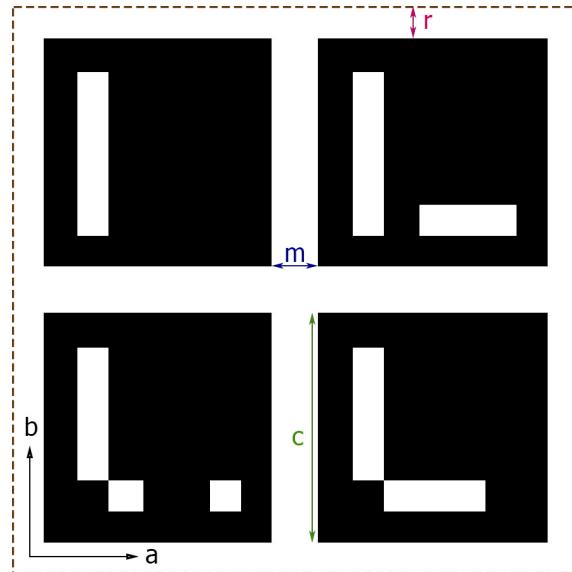
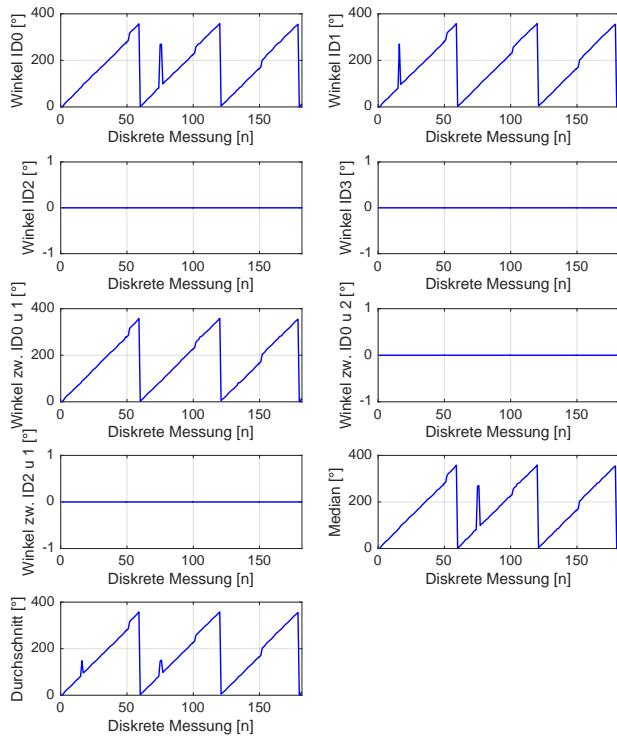


Abbildung 7: Schematische Darstellung einer Markeranordnung und deren Zusammensetzung aus vier Markern. Die braun gestrichelte Linie ist nicht Bestandteil der gedruckten Marker und dient der Darstellung des Markerrandes.

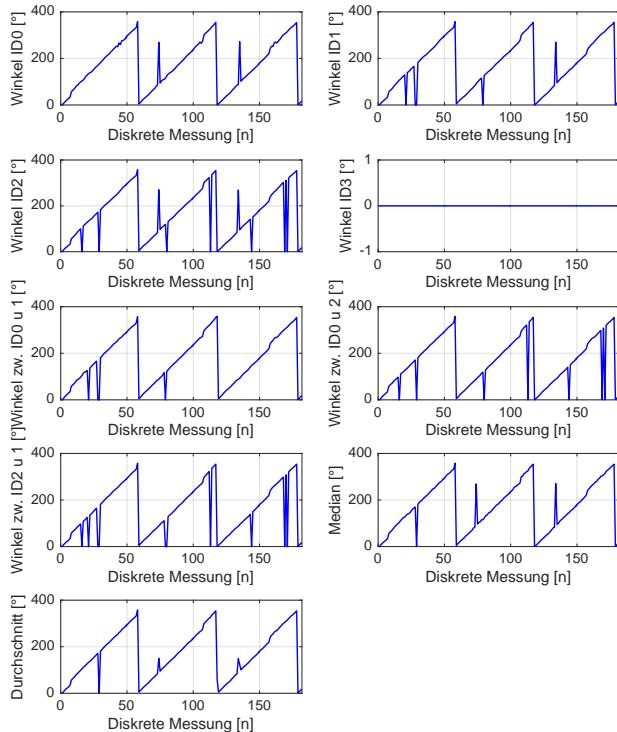
Nr.	a	b	c[mm]	m[mm]	r[mm]	Verhalten
1	2	2	65	1	17	Keine Erkennung möglich, aufgrund des zu geringen Abstandes zwischen den Markern
2	2	2	60	12	17	Erkennung erfolgt korrekt mit einigen Ausfällen. Vorhandene Fläche wird nicht vollständig ausgenutzt.
3	2	2	90	19	4	Erkennung erfolgt korrekt mit wenigen Ausfällen. Vorhandene Fläche ist sehr groß.
4	3	2	60	1	15	Keine Erkennung möglich, aufgrund des zu geringen Abstandes zwischen den Markern
5	3	2	54	11	17	Erkennung nur teilweise erfolgreich, da häufige Ausfälle.
6	1	3	80	12	14	Erkennung erfolgt korrekt.
7	1	2	128	11	16	Erkennung erfolgt korrekt. Vorhandene Fläche ist sehr groß.

Tabelle 4: Erkennungsverhalten unterschiedlicher Marker Anordnungen

Unter Verwendung der Nummern sechs und sieben wurden die stabilsten Resultate beobachtet. Aus diesem Grund werden beide Anordnungen genauer untersucht. Nachdem die Stabilität in statischen Situationen betrachtet wurde, werden die Marker auf einem Roboter befestigt, der sich mehrmals um die eigene Achse dreht. Die Position der Marker wird dabei durch die Testsoftware gespeichert und in der Abbildung 8 gegenübergestellt. Alle Messungen erfolgen bei einer Geschwindigkeit von $500 \frac{\text{mm}}{\text{s}}$, was zu dem Messzeitpunkt der anvisierten Geschwindigkeit des dritten Gewerkes entspricht.



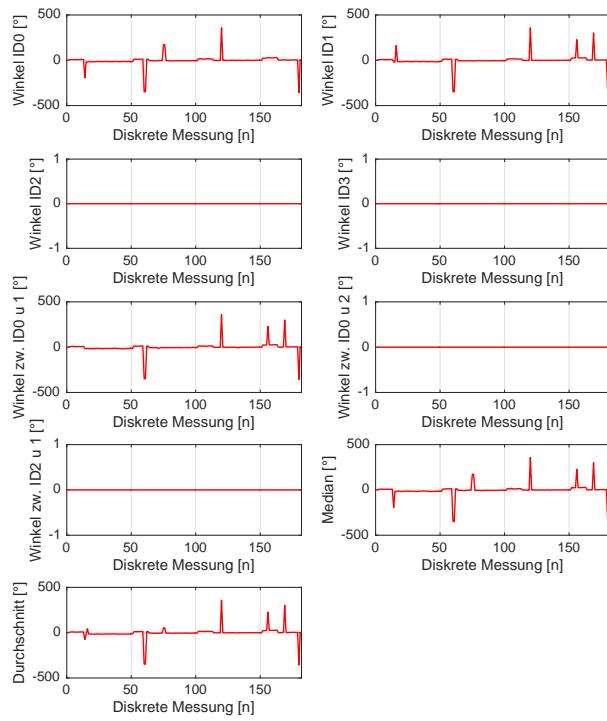
(a) Winkel beider ID's unter Verwendung der Anordnung Nr. 7 aus Tabelle 4



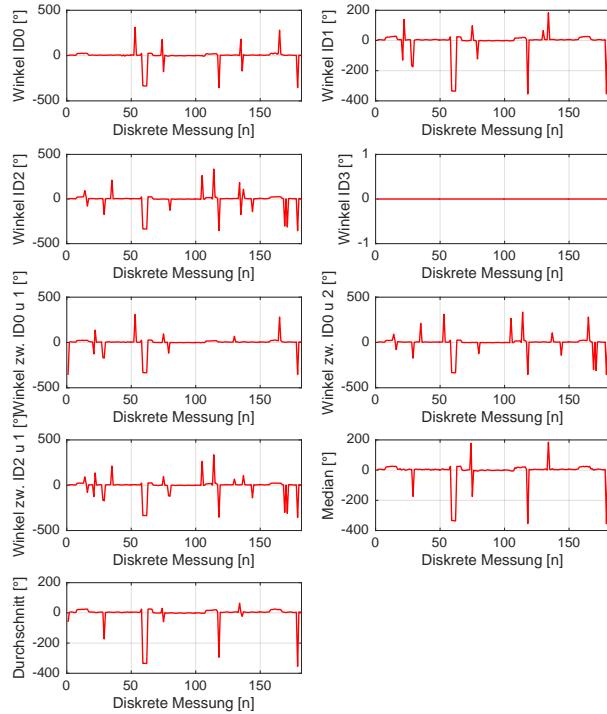
(b) Winkel aller drei ID's unter Verwendung der Anordnung Nr. 6 aus Tabelle 4

Abbildung 8: Gegenüberstellung der Winkel zwischen den Marker-ID's bei sich drehendem Roboter.
 In Abbildung a für zwei und in Abbildung b für drei Marker

Der Gegenüberstellung beider Anordnungen ist zu entnehmen, dass die Anordnung mit zwei Einheiten gegenüber der dreifachen Anordnung stabiler ist. Durch die Anordnung mit drei Markern stehen mehr Werte zur Verfügung jedoch profitiert weder die Schnittwertbildung, noch der Median davon. Die Anordnung mit zwei Markern weist zwar ebenfalls Abweichungen in der Erkennung der Marker auf, kann diese jedoch durch die wenigen Ausfälle in der Schnittwertbildung korrigieren. Die Medianbildung ist in beiden Systemen gegenüber der Schnittwertbildung im Nachteil, da sich die Ausreißer in einer höheren Amplitude äußern, während die Schnittwertbildung diese deutlich reduziert. Werden die Schnittwerte beider Anordnungen miteinander verglichen, weisen diese wenige Unterschiede auf, weshalb beide Anordnungen in Frage kommen. Wird die Differenz der Winkel unter den wiederholten Messungen gebildet, kann eine vage Aussage über deren Reproduzierbarkeit getroffen werden. Es ist anzumerken, dass es sich nicht um eine parametrierte Messung auf einem Messtisch handelt und die Roboter bei höheren Geschwindigkeiten und unebenem Boden einen Grunddrift aufweisen, der in die Messung einfließt. Unter Betrachtung der Differenz zwischen den Winkeln zeigt sich, dass die Anordnung mit zwei Markern weniger Ausreißer besitzt und keine fehlerhafte Erkennung stattfindet. Diese sind in Abbildung 9b in dem Auftreten der dritten ID zu erkennen, trotzdem diese während der Messung nicht existiert. Zur Erfassung der Roboter wird aufgrund der erhöhten Ausfallsicherheit sowie der größeren Länge der Markeranordnung und der daraus verbesserten Winkelberechnung (Vgl. Kapitel 2.1.3) die Anordnung mit zwei Markern verwendet. Für die Winkelberechnung wird die Schnittwertbildung verwendet, da diese gegenüber dem Median Ausfälle besser abfedert.



(a) Differenz der Winkel beider ID's, unter Verwendung der Anordnung Nr. 7 aus Tabelle 4 .



(b) Differenz der Winkel aller drei ID's, unter Verwendung der Anordnung Nr. 6 aus Tabelle 4 .

Abbildung 9: Gegenüberstellung der Winkeldifferenz für mehrmalige Drehungen. In Abbildung a für zwei und in Abbildung b für drei Marker.

2.2.2 Programmablauf und Funktion des Hauptprogramms

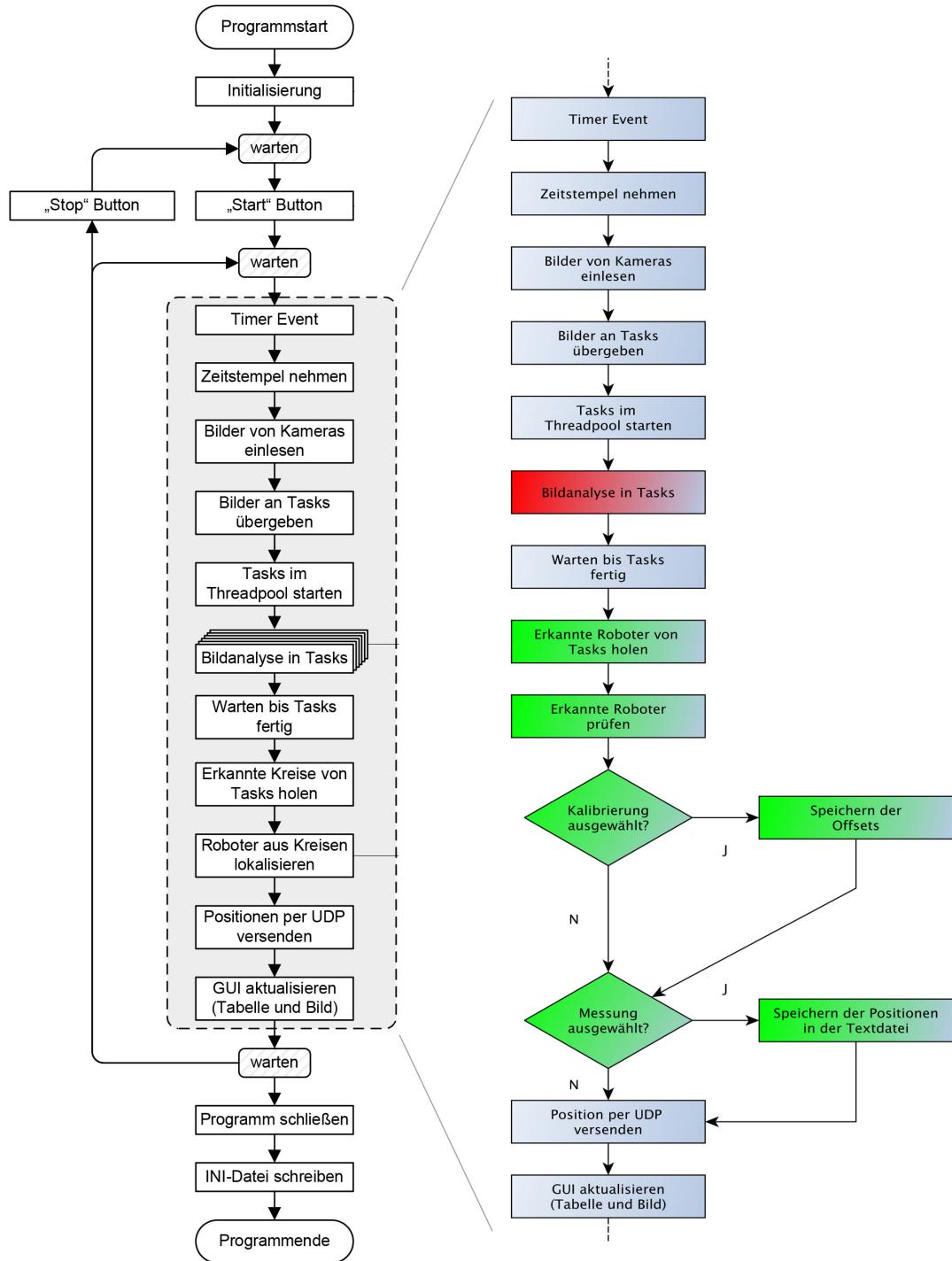


Abbildung 10: Gegenüberstellung des alten ([BBT14] links und des neuen (rechts) Hauptprogrammablaufplans. Grün hervorgehoben sind Änderungen gegenüber dem alten Hauptprogramm. Rot gekennzeichnet sind Änderungen in dem Unterprogramm.

Das Hauptprogramm ist nach dem Programmablauf aus Abbildung 10 strukturiert. Hierin zu erkennen sind der neue sowie der alte Programmablaufplan. Merkmale, die sich unterscheiden, sind grün unterlegt und Änderungen in den Programmabläufen der Unterprogramme rot. Während zuvor alle erkannten Kreise aus den Tasks übergeben wurden, erfolgt nun die Übergabe der detektierten Roboter. Daher werden im Anschluss die Roboterpositionen und nicht die Kreise auf Plausibilität geprüft. Zusätzlich ist das Programm um die Kalibrierung und der Messwerterfassung erweitert, die durch If-Abfragen ausgeführt werden können. Ebenfalls ergänzt wurde die Initialisierung, in der nun Arucoparameter, Bildvorbearbeitungsparameter und die Offsets der Roboter (vgl. Kapitel 2.3.4) aus der setting.ini geladen werden. Auf die Erläuterung von bereits implementierten Abschnitten wird an dieser Stelle verzichtet, diese sind in dem Bericht [BBT14] zu finden. Durch die umfangreichen Änderungen an der Klasse imgtask wird diese in ein UML-Diagramm überführt und in Abbildung 11 dargestellt.

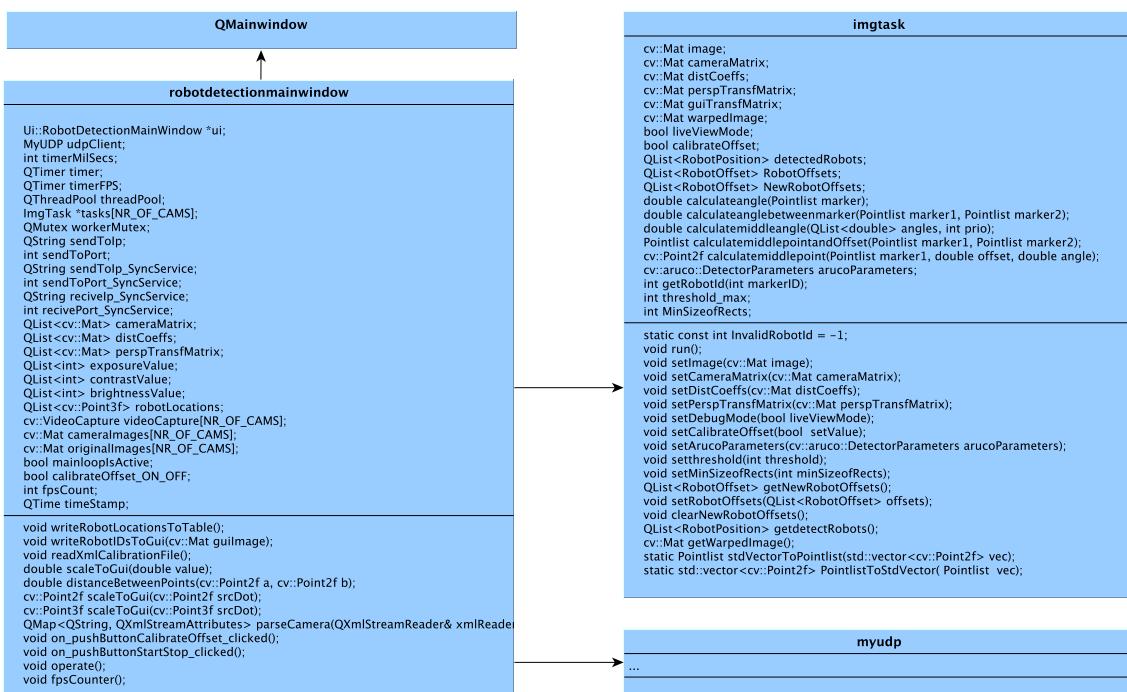


Abbildung 11: UML-Diagramm der veränderten imgtask-Klasse

Die Klassen weisen dabei die folgenden neuen Eigenschaften auf:

robotdetectionmainwindow:

robotLocations Enthält die Positionen sowie die Winkel der Roboter in korrekter Reihenfolge der ID und wird an die Roboter versendet.

calibrateOffset_ON_OFF Gibt an, ob der Nutzer den Button Kalibrierung ausgewählt hat, wodurch im späteren Programmablauf die Offsets der Roboter in der setting.ini gespeichert werden.

imgtask:

calibrateOffset Übernimmt den Wert aus der Eigenschaft „calibrateOffset_ON_OFF“ aus der robotdetectionmainwindow.

detectedRobots Enthält die detektierten Roboter inklusive deren ID, Winkel und Position in Form der Struktur:

```
struct RobotPosition
{
    int id;
    cv::Point3f coordinates;
};
```

RobotOffsets Enthält die aktuelle Roboter ID, sowie die Offsets des geradzahligen und des ungeraden Markers in folgender Struktur:

```
struct RobotOffset
{
    int id;
    double offsetMarkerEven;
    double offsetMarkerNotEven;
};
```

NewRobotOffsets Enthält die Roboter ID, sowie die Offsets des geraden und des ungeraden Markers, nachdem eine Parametrierung ausgeführt wurde.

arucoParameters Enthält die Parameter für den Aruco Algorithmus, die den Reglern der GUI entstammen.

MinSizeofRects Beinhaltet die maximale Größe von aussortierbaren AOI's.

2.2.3 Winkel- und Positionsbestimmung

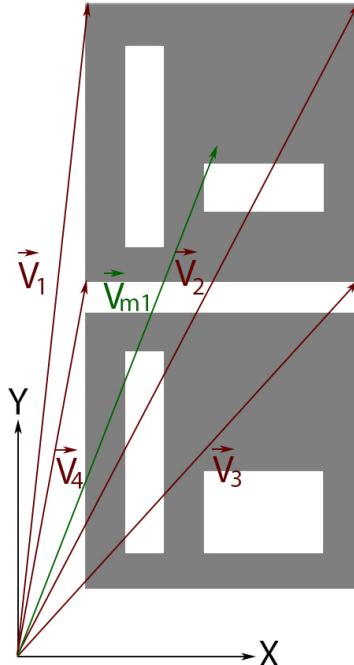


Abbildung 12: Vektoren der vier Ecken eines Markers im Arbeitsfeld (rot), sowie dessen Mittelwert (grün).

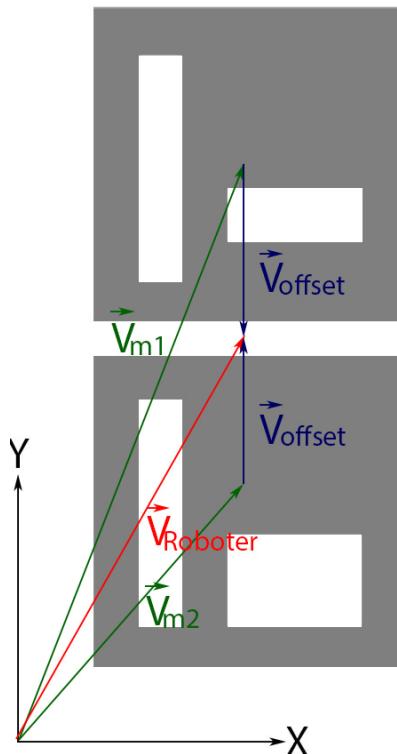


Abbildung 13: Vektoren der Mittelwerte zweier Marker (grün), dessen Schnittwert (rot), sowie der Differenz zwischen dem Schnittwert und den einzelnen Mittelwerten der Marker (blau).

Die Abbildungen 12 und 13 zeigen das Berechnungsprinzip für die Positionsbestimmung eines Roboters. In Abbildung 12 sind die vier Vektoren eines Markers V_1 bis V_4 zu erkennen, die das Ergebnis des Aruco Algorithmus darstellen. Durch die Bildung des Mittelwertes nach:

$$V_{m1} = \frac{1}{4} \sum_{i=1}^4 V_i$$

resultiert der Mittelpunkt eines Markers. Dieselbe Rechnung wird für den zweiten Marker ausgeführt, sodass die Mittelpunkte beider Marker vorhanden sind. Wie in Abbildung 13 zu erkennen ist werden diese ebenfalls gemittelt, woraus der Mittelpunkt der Anordnung zweier Marker entsteht. Dieser Punkt entspricht dem Mittelpunkt des Roboters und wird im Anschluss an diesen versendet. Nach Unterkapitel 2.3.4 kann bei Ausfall eines Markers auf den Offset zurückgegriffen werden. Hierbei wird auf einen der Vektoren V_{m1} oder V_{m2} der Offset Vektor V_{offset} addiert bzw. subtrahiert.

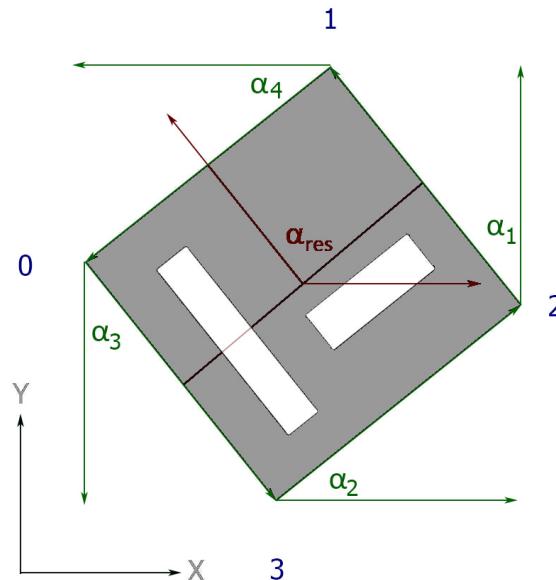


Abbildung 14: Nummerierte Ecken eines Markers (blau), sowie deren Vektoren und den daraus resultierenden Winkeln (grün). Rot markiert ist die Ausrichtung des Markers gegenüber dem Koordinatenursprung.

Für die Berechnung des Roboterwinkels werden für einen Marker vier Winkel berechnet. Zu erkennen ist, dass jeder Winkel einen eigenen Offset zum Ursprung besitzt und zusätzlich in seinem Quadranten variiert. Daher werden die Winkel zwischen den Punkten getrennt betrachtet. Die Codeumsetzung ist dabei wie folgt strukturiert:

```
//Calculate Middlevalue for Marker 1
if(marker1.size ()>0)
{
    for( int i = 0;i<4;i++)
    {
        temp1.x=temp1.x+marker1[ i ].x;
        temp1.y=temp1.y+marker1[ i ].y;
    }
    n++;
}

    middlepointMarker1.x = temp1.x/4;
    middlepointMarker1.y = temp1.y/4;
}

//Calculate Middlevalue for Marker 2
if(marker2.size ()>0)
{
    for( int i = 0;i<4;i++)
    {
        temp2.x=temp2.x+marker2[ i ].x;
        temp2.y=temp2.y+marker2[ i ].y;
    }
    n++;
}

    middlepointMarker2.x = temp2.x/4;
    middlepointMarker2.y = temp2.y/4;
}

//Calculate Middlepoint of Robot
middlepoint.x = ( middlepointMarker1.x+middlepointMarker2.x ) / 2;
```

```

middlepoint.y = (middlepointMarker1.y+middlepointMarker2.y)/2;

//Safe Middlepoint and first Position in Returnvalue
Pointinformations.append(middlepoint);

//Calculate Difference between Markers and Middlepoint (Offset),
//and save it at Position 1 and 2 in Returnvalue
Pointinformations.append(Point2f(middlepoint.x-middlepointMarker1.x,middlepoint.y-
    middlepointMarker1.y)); Pointinformations.append(Point2f(middlepoint.x-
    middlepointMarker2.x,middlepoint.y-middlepointMarker2.y));

return Pointinformations;

```

Für eine höhere Genauigkeit des Winkels, fließen zwei weitere Winkel zwischen den Markern in die Berechnung mit ein. Hierbei werden die Winkel der Vektoren zwischen den äußereren Kanten der beiden Marker errechnen. Gleichzeitig dienen sie als Entscheidungsgrundlage in Winkelbereichen zwischen 359°-1° (vgl. Kapitel 2.3.5), da sie durch die größeren Vektoren zwischen den beiden Markern, eine höhere Genauigkeit aufweisen.

2.2.4 Bildvorverarbeitung, Übergabe und Aruco Parameter

Die Umsetzung des Aruco Algorithmus zeigt ein sehr rechenintensives Verhalten, weshalb die Implementation der Bibliothek hohe Einbußen in der Wiederholungsrate erzeugt. Aus diesem Grund wird das Bild mithilfe bekannter Bildoperatoren gefiltert, um ausschließlich relevante Abschnitte (sog. AOI Area of interest) dem Aruco Algorithmus zu übergeben. Hierzu erfolgen die nachstehenden Bearbeitungsschritte.

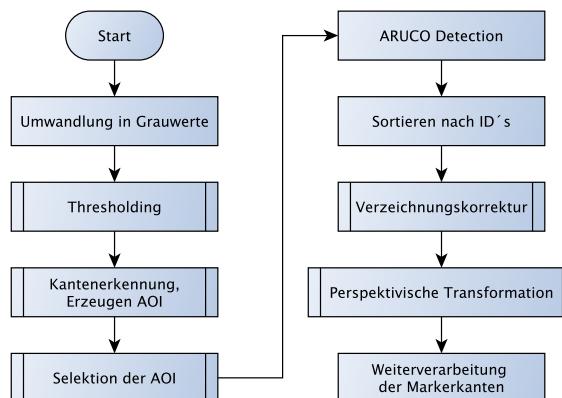


Abbildung 15: Programmablaufplan der Bildvorverarbeitung zur Ermittlung der AOI's.

Im ersten Schritt erfolgt die Schwellwertberechnung nach folgendem Schema [tea]:

$$dst(x, y) = \begin{cases} \text{src}(x, y) & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$$

In der Gleichung ist der Helligkeitswert der Quelle „src“ sowie der zu erreichende Schwellwert „thresh“ und das resultierende Ergebnis „dst“ zu finden. Implementiert ist der Aufruf in der Imgtask.cpp:

```
threshold(workImage, workImage, threshold_max, 200, THRESH_TOZERO);
```

Hierbei repräsentiert threshold_max den Schwellwert der erreicht werden muss, damit ein Pixel nicht auf Null gesetzt wird. Dieser Parameter wird über die Eigenschaft des Objektes imgtask an die GUI übergeben und kann von dem Anwender variiert werden. Im Anschluss erfolgt die Erfassung der Konturen im Bild, zu der auch die Ränder der Marker gehören :



```
findContours(workImage, contours, hierarchy, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
```

Anschließend können auf Grundlage der Konturen Rechtecke gebildet werden, die den relevanten AOI's entsprechen. Da bekannt ist, welche Fläche die Marker in dem Bild aufweisen (anhand einer Messung), können die gefundenen AOI's auf die relevanten reduziert werden.

```
if(rect.size().area() > 1000 && rect.size().area() < MinSizeofRects)
{...
```

Hierbei wird der Parameter MinSizeofRects auf die GUI geführt und kann von dem Anwender beeinflusst werden. AOI's, deren Fläche die gewünschten Kriterien erfüllen, werden anschließend an den Aruco-Algorithmus übergeben:

```
cv::aruco::detectMarkers(roi, dictionary, tempmarkerCorners, tempmarkerIds,
arucoParameters, rejectedCandidates);
```

Die Parameter besitzen dabei folgende Funktion

Übergabeveriable	Funktion
roi	Bereich der nach Markern durchsucht werden soll
dictionary	Bibliothek der Marker. (Vgl. Kapitel 2.2.1)
tempmarkerCorners	Vektor mit den Koordinaten der vier Ecken der detektierten Marker
tempmarkerIds	Vektor mit den ID's der erkannten Marker, in Reihenfolge der Eckenanordnung
arucoParameters	Parameter für den Aruco-Algorithmus
rejectedCandidates	Gefilterte Objekte

Tabelle 5: Übergabeparameter und Variablen des Aruco Algorithmus

Die Variable „arucoParameters“ besitzt weitere Parametrierungsmöglichkeiten, die direkt an den Algorithmus übergeben werden. Hierfür ist eine Variable vom Typ cv::aruco::DetectorParameters notwendig. Diese ist ebenfalls eine Eigenschaft des Objektes imgtask, wodurch relevante Einstellungen an die GUI überführt werden. Die Parameter besitzen dabei folgende Funktionen

adaptiveThreshWinSizeMin Anfangsgröße des Thresholdfensters in Pixel.

adaptiveThreshWinSizeMax Endgröße des Thresholdfensters in Pixel.

adaptiveThreshWinSizeStep Stufen zwischen der maximalen und minimalen Fenstergrößen des Thresholding.

minMarkerPerimeterRate Minimaler prozentualer Anteil der Marker Größe gegenüber dem zu analysierendem Bild.

maxMarkerPerimeterRate Maximaler prozentualer Anteil der Marker Größe gegenüber dem zu analysierendem Bild.



polygonalApproxAccuracyRate Höchster erlaubter Fehler bei der Erstellung approximierter polygonaler Rechtecke auf Basis der übergebenen Markerinformationen. Je verzerrter das Bild ist, desto höher ist der mögliche Fehler und eine hohe Rate ist zu wählen.

minCornerDistanceRate Minimaler Abstand zwischen den Ecken innerhalb eines Markers.

minMarkerDistanceRate Minimaler Abstand zwischen den Ecken von zwei unterschiedlichen Markern. Sind zwei Marker zu nah beieinander, wird der kleinere ignoriert.

minDistanceToBorder Minimaler Abstand zwischen dem Marker und dem Rand des zu analysierenden Bildes. Aufgrund der Übergabe von AOI's wird dieser Parameter nicht weiter beachtet.

markerBorderBits Definiert das Vielfache der internen Bits eines Markers im Vergleich zu dem Markerrand. Da keine Änderung an dem Verhältnis zwischen Rand und internen Bits bei der Marker Erstellung vorgenommen wurden, wird diese Einstellung vernachlässigt.

perspectiveRemovePixelPerCell Gibt die Größe der Pixel im Suchraster des Algorithmus an. Wertänderungen haben einen starken Einfluss auf die Performance, weshalb dieser Parameter auf die GUI geführt wird.

errorCorrectionRate Multiplikator zur Angabe maximal korrigierbarer Bits. Der Wert ist ebenfalls von der ausgewählten Bibliothek abhängig und wirkt sich zudem merkbar auf die Performance des Systems aus.

doCornerRefinement Aktiviert oder deaktiviert die Subpixelverbesserung. Die Aktivierung erfolgt durch das Setzen eines Wertes für cornerRefinementMaxIterations, der größer Null ist. Eine Null deaktiviert die Subpixelverbesserung.

cornerRefinementWinSize Legt die minimale Größe der Subpixel fest und ist nicht an die GUI übergeben.

cornerRefinementMaxIterations Gibt die Anzahl der Wiederholungen zur Subpixelverbesserung an. Dieser Wert hat starken Einfluss auf die Performance und ist ebenfalls an die GUI übergeben.

cornerRefinementMinAccuracy Gibt den minimal auftretenden Fehler an, bevor die Subpixelverbesserung beendet wird.

Die optimalen Einstellungen sind der Tabelle 6 zu entnehmen und basieren auf empirischen Messungen. Aufgrund der Schwellwertbildung in der Selektion der AOI's kann das Thresholding in dem Aruco Algorithmus vernachlässigt werden. Dieser ist notwendig, um nur die relevanten Konturen aus dem gesamten Bild zu extrahieren und muss somit nicht mehr angewendet werden. Die Verwendung des Thresholdings durch Aruco hätte eine Performanceeinbuße zur Ursache, da weitere AOI's aufgrund der fehlenden Schwellwertbildung erkannt werden. Die gesamte Vorverarbeitung bis zur Berechnung der Position und des Winkels kann nachstehender Abbildung entnommen werden.

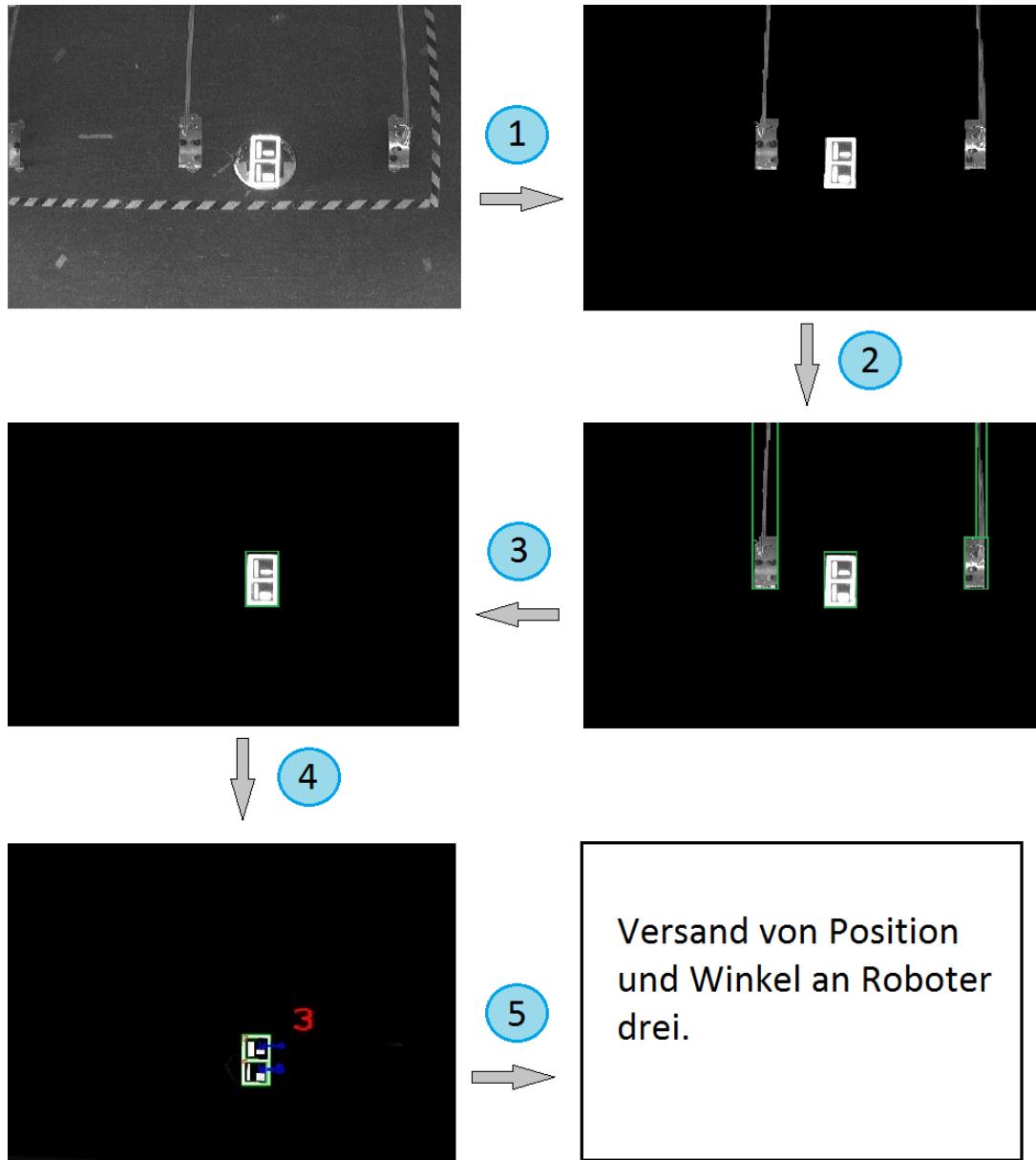


Abbildung 16: Schematische Darstellung der Vorverarbeitungsschritte in der Bildverarbeitung.

In der Abbildung 16 sind die vorbearbeitenden Schritte schematisch dargestellt. Zu sehen ist, dass die originale Aufnahme (erstes oberes linkes Bild) in Schritt eins durch das Thresholding um ihre unrelevanten Informationen reduziert wird. In diesem Beispiel sind die Stationen ebenfalls hell genug, um als relevant eingestuft zu werden. Dieses Verhalten entspricht nur unter eingeschalteter Raumbeleuchtung der Praxis, soll hier aber der Veranschaulichung von Bearbeitungsschritt drei dienen. Im zweiten Schritt werden die Konturen um noch vorhandene Objekte gelegt, aus denen im Anschluss die grünen AOI's gebildet werden. In Schritt drei werden zu große oder zu kleine AOI's aussortiert, sodass in Schritt vier nur die Markeranordnung an den Aruco Algorithmus übergeben wird. Im fünften Schritt erfolgt dann die Berechnung von Winkel und Position, damit diese anschließend versendet werden.



2.2.5 Informationsauswertung

Der Bildvorverarbeitung folgt im Anschluss die Auswertung in Form der in Kapitel 2.2.3 vorgestellten Methode. Für die Zuordnung zwischen Winkel, Offset, Positionsvektor und Roboter steht der Schleifenablauf in der imgtask.cpp Datei zur Verfügung. Dieser wird anhand des Programmablaufplans in Abbildung 17 erläutert. Dieser zeigt auf, warum das Sortieren der erkannten Marker notwendig ist. Jeder Roboter verfügt über zwei Marker, dessen ID's eine gerade sowie die darauffolgende ungerade Zahl aufweisen. So besitzt der erste Roboter die Marker mit der ID Null und eins, also einer geraden und einer ungeraden Zahl. Für einen iterativen Ablauf ist die Sortierreihenfolge von klein nach groß notwendig, damit die korrekte Reihenfolge der zugehörigen Marker immer identisch ist und der Ablauf iterativ in einer Schleife erfolgen kann. Somit kann einfach geprüft werden, ob die zweite zugehörige ID ebenfalls erkannt wurde. Auf dessen Grundlage erfolgen dann die sich unterscheidenden Verfahren zur Bestimmung der Position und des Winkels. Handelt es sich bei dem aktuellen Element in der QList um eine ungerade Zahl, ist die dazugehörige gerade Zahl den Prozess bereits durchlaufen. Daher existiert bereits der Winkel des dazugehörigen Markers mit der geraden ID und die Schnittwertbildung kann diese berücksichtigen. Handelt es sich um eine gerade ID und die dazugehörige ungerade ID existiert, wird die Schleife verlassen und der Iterator inkrementiert. Fehlt die ungerade ID, erfolgt die Winkelberechnung auf Grundlage der Marker- Winkel sowie der Position und der Addition des Offsets. Durch das vorherige Sortieren kann bei der Existenzprüfung des zweiten Markers mit einfacherem In- oder Dekrementieren des Listenoperators auf den dazugehörigen Marker zugegriffen werden. Ein zur Übersicht stark gekürzter Ausschnitt der Berechnung soll im Vergleich mit der Abbildung 17 die Passage erläutern und zukünftige Änderungen erleichtern.

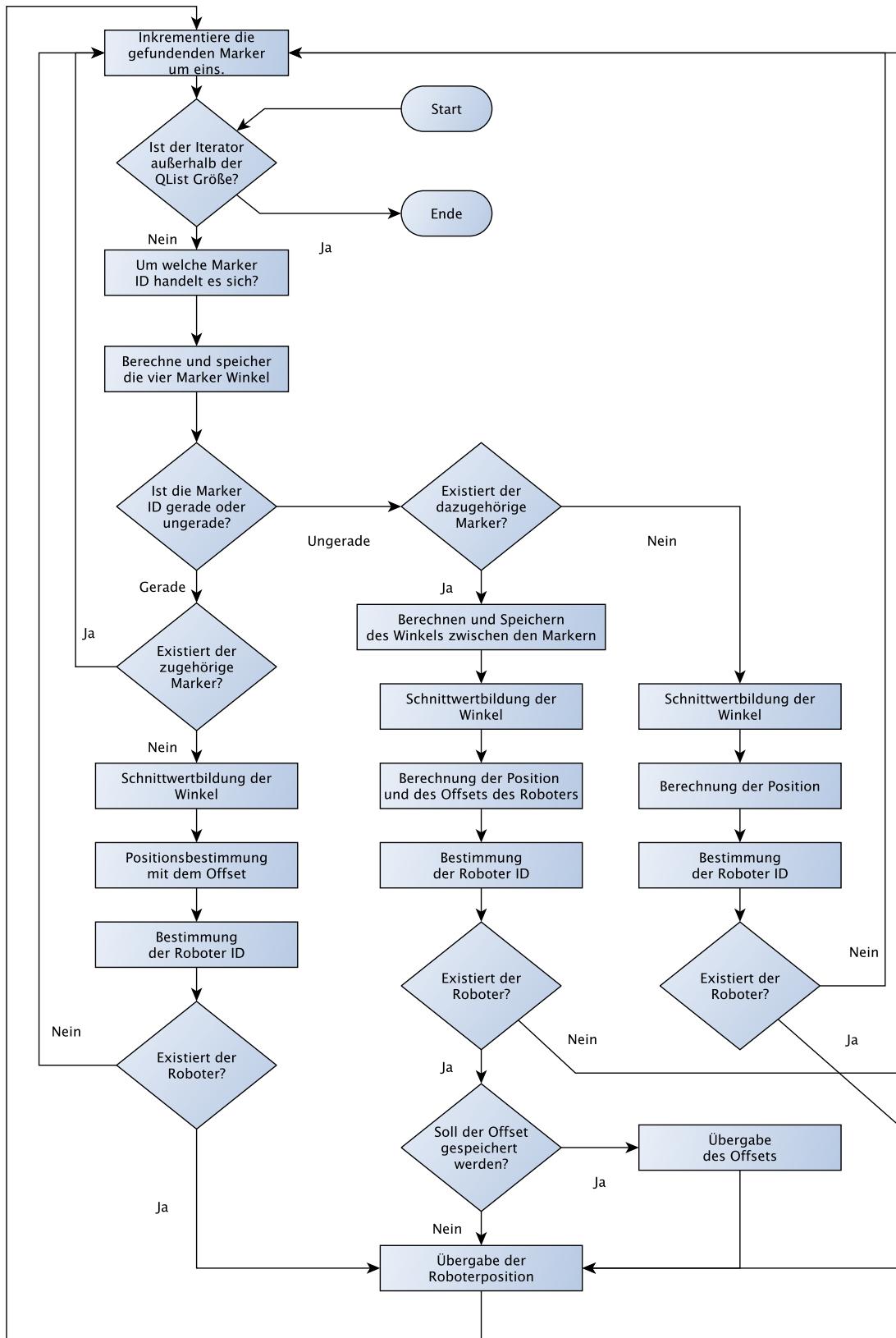


Abbildung 17: Ablaufplan in der imgtask zur Zuordnung zwischen Winkel, Offset, Positionsvektor und Roboter.



```
//For each detected Marker...
for( int i = 0; i<markerList.size(); i++)
{
    RobotMarker currentMarker = markerList[i];

    //find the detected ID (Each Roboter have two Marker, that are all possible IDs)
    for( unsigned int a = 0; a<MAX_NR_OF_ROBOTS*2; a++)
    {

        //Current MarkerID is in a
        if(currentMarker.id == a)
        {

            //Calculate the Angle from Marker i and save it
            tempangle = calculateangle(currentMarker.warpedCornerPoints);

            if(tempangle != 0)
                angles.append(tempangle);

            //Is the current ID not even, check the second (paired) marker
            //Note the Order for the IDs on the Roboter (See Report).
            if(a % 2 != 0)
            {

                //If the second marker (paired from even Marker) also exists
                t = i;
                if(t-1 >= 0 && markerList[t-1].id == a-1)
                {

                    ...
                }
                //Otherwise (The second paired Marker does not exist),
                //Add the Offset to get the correct Position of one Marker
                else
                {

                    ...
                }
                //Clear the temporary Variables for next two Markers
                angles.clear();
                angleMiddleValue = 0;
            }
        }

        //If the current ID even, calculate Position with Offset
        //and clear the Variables for next two Marker.
        else if((markerList.size() > i+1 && markerList[i+1].id != a+1) || (markerList.at
            (markerList.size() - 1).id == a))
        {

            ...
        }

        //Clear the temporary Variables for next two Markers
        angles.clear();
        angleMiddleValue = 0;
    }
}
}
```

2.3 Fehlerbetrachtung und ausgewählte Lösungen

Um sicherzustellen, dass das System zuverlässig arbeitet, muss es auf sein Fehlverhalten untersucht und Mängel behoben werden. Zusätzlich sollen mögliche Anwenderfehler erkannt und wenn möglich auf Seiten der Software gelöst werden. Dafür erfolgt die Ausführung diverser Situationen von denen man annehmen kann, dass sie im Laufe der Betriebszeit auftreten. Weiterhin sind Situationen zu unterbinden, die Probleme in andere Gewerke verursachen können.

2.3.1 Fehlende Marker-Erkennung

Aufgrund der unterschiedlichen Einflussfaktoren auf die Marker wie Position, Kamera, Beleuchtung, Winkel und zu erkennender ID, müssen ideale Einstellungen gefunden werden, sodass alle Zielkoordinaten korrekt erfasst werden. Hierzu werden alle Raumpunkte des Arbeitsfeldes mit jedem Roboter durchfahren und die Einstellungen so angepasst, dass der Roboter erkannt wird. Die Raumpunkte basieren auf der Einteilung des zweiten Gewerkes. Die auf diese Weise ermittelten Einstellungen repräsentieren gleichzeitig die Standardwerte für das Programm und sind in der setting.ini des Systems gespeichert. Die ermittelten Einstellungen lauten wie folgt:

Parameter	Wert	Skalierung
Threshold	161	1
Min Size of Rects	8000	1
Corner Refinement Min. Accuracy	10	100
Error Correction Rate	527	100
Corner Refinement Max. Iterations	1	1
Adaptive Thresh Win Size Min.	10	1
Adaptive Thresh Win Size Step	99	1
Adaptive Thresh Constant	20	1
Min Marker Perimeter Rate	91	100
Max Marker Perimeter Rate	307	100
Polygonal Approx Accuracy Rate	11	100
Perspective Remove Pixel Per Cell	1	1

Tabelle 6: Optimale Konfigurationsparameter für das Programm inklusive Skalierungsfaktoren.

Der erste Parameter gibt dabei den minimalen Helligkeitswert eines Pixels an, um die Schwellwertbildung zu passieren, der zweite Wert gibt die maximale Größe der erkannten AOI's als Fläche an. Die weiteren Parameter sind in Abschnitt 2.2.4 erläutert. Bei den Parametern ist der verwendete Skalierungsfaktor zu beachten. Dieser dient der Skalierung für die GUI, damit die Schiebereglerwerte in den relevanten Bereichen agieren. Somit wird z.B anstatt eines Wertes von zehn, der Wert 0,1 an den Aruco Parameter Corner Refinement Min. Accuracy übergeben.

2.3.2 Fehlerhafte Zuordnung der ID

Diverse Testfahrten haben gezeigt, dass der Aruco Algorithmus sehr zuverlässig in der Erkennung der Marker ID's ist. In Überlagerungsbereichen der Kameras, kann es zu einer Reduzierung der ID's kommen. Voralledem im Überlagerungsbereich der vier Kameras kommt es zur Reduzierung der Marker auf vier Ebenen. Da die Roboter über zwei Marker



verfügen kommt es an einer Position zu einer Erkennung von zwei Robotern. Während einer der Marker korrekt erfasst wird, wird dessen zweiter Marker reduziert und einem anderen Roboter zugeordnet. Um dieses Verhalten zu unterbinden, wird ein Prüfalgorithmus in das Programm integriert. Dieser prüft alle Positionen der möglichen Roboter auf einen minimalen Abstand zueinander. Befinden sich zwei Roboter innerhalb eines Radius von 500 mm werden beide ignoriert. Aus diesem Grund können keine Sprünge entstehen, die einen hohen Anstieg der Robotergeschwindigkeit zur Folge hätten, da der Beobachter die Geschwindigkeit anhand der Entfernung zwischen der Ist- und der Sollposition bestimmt.

2.3.3 Anfahrt der Stationen

Das erforderliche Maß an Genauigkeit von Position und Winkel der Roboter wird durch die Ablagebereiche der Stationen bestimmt. Um zu prüfen, ob das System mit der Marker Erkennung in den Anfahrtssituationen funktioniert, werden die zwei Roboter mit der ID eins und zwei dazu verwendet, die letzte Station über einen Zeitraum von einer Stunde redundant an zu fahren. Während der erste Roboter einen Anfahrtswinkel von 0° verwendet, beträgt der Winkel für den zweiten Roboter 180°, sodass die Station von beiden Seiten angefahren wird. Die Auswahl der letzten Station begründet sich durch die dort bestehenden Überlagerungsbereiche zwischen zwei Kameras, die erhöhte Anforderungen an das System stellen. Für die wiederholenden Anfahrten fahren die Roboter an eine definierte Position vor der Station. Sobald die Position erreicht ist, wird einer der Ablageplätze angefahren. Im Anschluss fährt der Roboter auf seine Ursprungssituation, um in Anschluss einen anderen Ablageplatz der gleichen Station anzufahren. Während der Durchführung besitzen beide Roboter ein Werkstück. Eine Anfahrt wird dann als erfolgreich interpretiert, wenn das Werkstück auf dem Ablageplatz sicher liegen würde, ohne dabei zu kippen. Während die Anfahrten unter ausgeschalteter Raumbeleuchtung in dem Messzeitraum stets erfolgreich sind, kann es mit eingeschalteter Raumbeleuchtung zu Ausfällen nahe der Stationen kommen. Die Ursache ist in den reflektierenden Materialeigenschaften der Stationen zu finden. Je nach Thresholding Wert, passieren diese den Filter und werden mit dem Marker als ein AOI übergeben. Der Aruco Algorithmus interpretiert die Stationen dann als Bestandteil des Markers und interpretiert diesen nicht korrekt. Eine mögliche Lösung, die weiterhin das System unter eingeschaltetem Licht stabilisieren würde, wären getrennte Parameter für ein- und ausgeschalteter Raumbeleuchtung. Durch Anpassen des Threshold Wertes kann bereits eine Verbesserung festgestellt werden. Eine ideale Lösung repräsentiert eine automatische Umschaltung zwischen den Hell- und Dunkelparametern. Diese kann durch Messung des mittleren Grauwertes in den Aufnahmen erfolgen, wenn dieser einen bestimmten Wert unter- oder überschreitet. Die Umsetzung dieser Lösung erfolgte aufgrund fehlender Zeit nicht, soll an dieser Stelle jedoch erwähnt werden.

2.3.4 Ausfall eines Markers

Während der Umsetzung ist zu erkennen, dass einer der Marker je nach Situation nicht korrekt erkannt wird. In diesem Fall erfolgt eine Offset Berechnung auf Basis der Position des zweiten Markers. Um in Zukunft auf mögliche Änderungen in der Anordnung der Marker oder Kameras reagieren zu können, wird eine Kalibrierung in das System integriert. Diese verlangt die vollständige Erkennung der Marker auf dem Spielfeld und errechnet anschließend den Betrag der Mittelpunkte zwischen den Markern unter den Robotern. Sind diese erfasst, werden Sie in der setting.ini hinterlegt. Fällt ein Marker aus, kann auf diesen Offset zurückgegriffen werden und mithilfe des Winkels die aktuelle Position des Roboters errechnet werden. Hierfür ist die Funktion



```
point2f ImgTask::calculatemiddlepoint( Pointlist marker1 , double offset , double angle )
```

zu verwenden. Der Funktion sind der Winkel des erkannten Markers, die Position und der dazugehörige Offset aus der config.ini zu übergeben. Der Rückgabeparameter entspricht anschließend der Position des Roboters.

2.3.5 Winkelbestimmungen im Übergangsbereich

Da zur Bestimmung des Winkels auf eine Schnittwertbildung zurückgegriffen wird, kann es in dem Bereich von $359^\circ - 1^\circ$ zu fehlerhaften Werten kommen. Diese treten z.B bei der Schnittwertbildung von $0,1^\circ$ und $359,5^\circ$ auf, deren Lösung 179.8° ist. Für die Lösung dieses Problems wird auf eine priorisierende Winkelbestimmung zurückgegriffen, die sich im Grenzfall nach dem Winkel zwischen den beiden Markern orientiert. Hierzu ist die Funktion

```
double ImgTask::calculatemiddleangle( QList<double> angles , int prio )
```

in der imgtast.cpp Datei implementiert. Diese benötigt eine Liste mit allen Winkeln, sowie die um eins inkrementierte Position des Winkels in der Liste nach dem priorisiert werden soll. Für den Wert Null erfolgt hingegen keine Priorisierung. Der Rückgabewert entspricht dabei dem Schnittwert aller übergebenen Winkel. An dieser Stelle sei anzumerken, dass die Umrechnung in Polarkoordinaten und deren Schnittwertbildung die hier verwendete Fallunterscheidung überflüssig erscheinen lässt. Da sich beide Systeme in ihrem Rechenaufwand nur marginal unterscheiden und die größten Unterschiede in der Form zu finden sind, wird der Vorteil der Priorisierung gewählt. Diese sind in Anlehnung an Kapitel 2.2.4 zu finden, wonach die Länge zwischen den beiden Markern größer und der daraus resultierende Winkel genauer gegenüber den Winkeln der Marker ist. Somit wird sich im Grenzfall, der häufig in Anfahrtssituationen der Stationen (0° oder 180°) entsteht, für die genauere Variante entschieden.

2.4 Auswertung

In diesem Kapitel werden verschiedene Messungen durchgeführt, um die Vor- und Nachteile des neuen und des alten Systems zu erfassen. Hierzu fährt der zweite Roboter eine Kreisstrecke im Raum ab, auf der er mehrere Kameras und die Kabeltrasse der Stationen passiert. Die Messung erfolgt mit dem Regelungsmodell des dritten Gewerkes. Aufgrund der Tatsache, dass das alte Programm über zwei Algorithmen verfügt, wurden beide gegenüber dem Marker- basierendem System getestet. Die erste Messung erfolgt bei ausgeschalteter Raumbeleuchtung und unter Verwendung der LED-Lampen. Während des gesamten Zeitraumes ist der Roboter das einzige Objekt auf dem Feld und fährt mit einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$. Weiterhin werden zeitgleich keine Tests- oder Anwendungen anderer Gewerke ausgeführt.

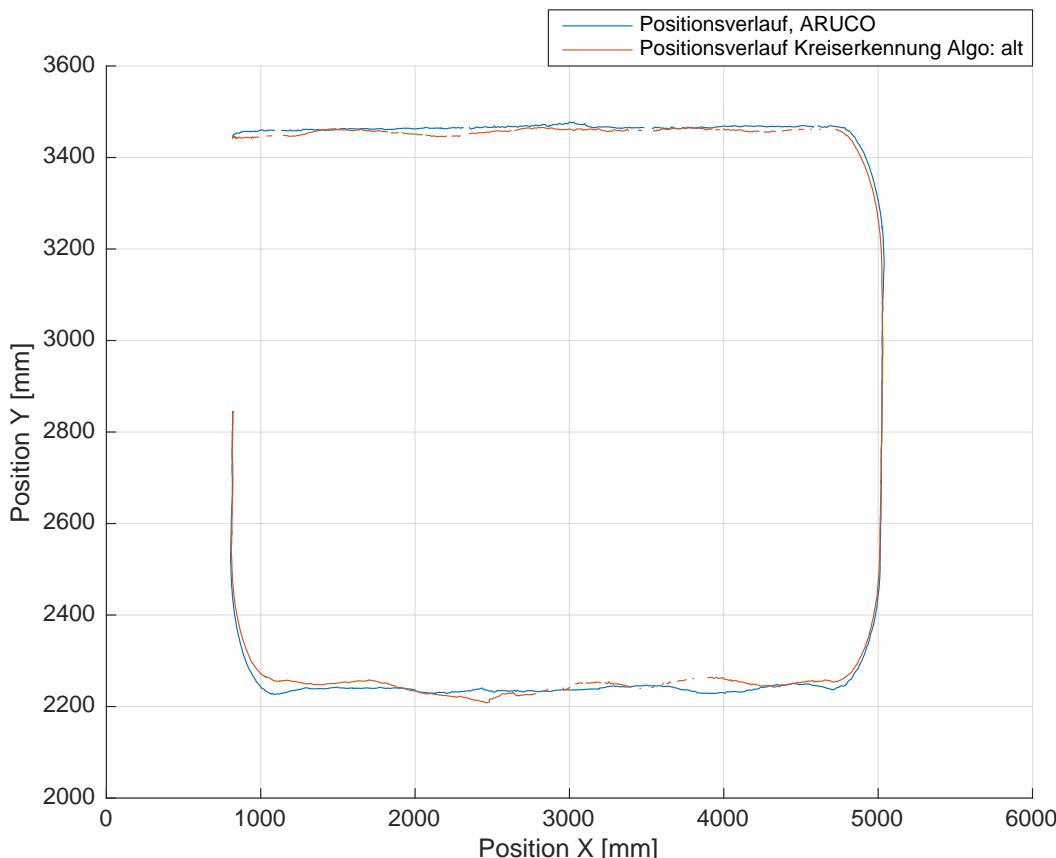


Abbildung 18: Zeitlicher Positionsverlauf des zweiten Roboters unter ausgeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ zwischen dem alten Algorithmus und dem neuen System.

Die Gegenüberstellung der beiden Messungen zeigt, dass das neue System gegenüber dem alten Algorithmus sowohl stabiler als auch ruhiger ist. Zwar sind weiterhin geringe Ausfälle unter den Kabelführungen der Stationen zu erkennen, diese sind jedoch nur von kurzer Dauer. Während beide Systeme in der oberen Hälfte sowie der rechten Seite nahe zu gleiche Messungen liefern, unterscheiden sie sich in der unteren Hälfte. Hier fällt das alte System deutlich häufiger aus. Unter Verwendung eines geeigneten Beobachters sind beide Systeme geeignete Erkennungsverfahren, wobei das neue System robuster und

ruhiger arbeitet. Da der erste Algorithmus des alten Systems angepasst wurde, erfolgt die Gegenüberstellung des neuen Systems gegenüber dem Alten mit verbessertem Algorithmus. Die Messung erfolgte unter identischen Umgebungsbedingungen.

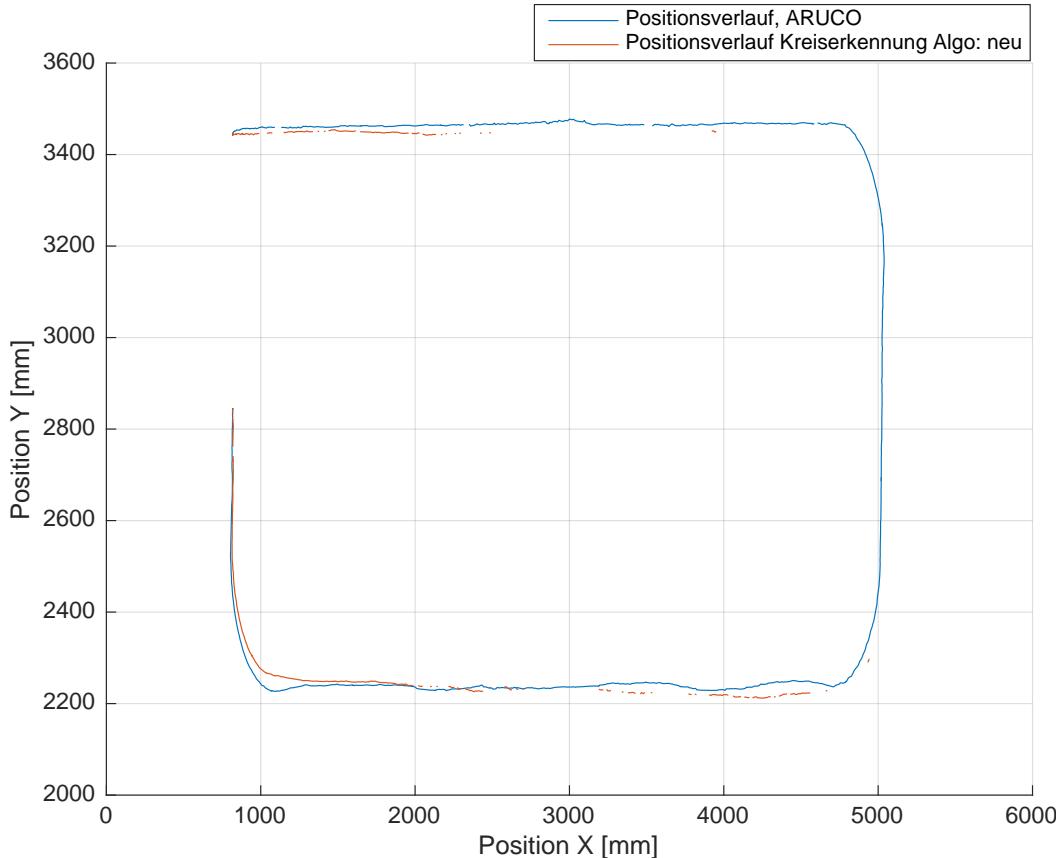


Abbildung 19: Zeitlicher Positionsverlauf des zweiten Roboters unter ausgeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ zwischen dem verbesserten Algorithmus und dem neuen System.

Die Messung zeigt entgegen der Erwartungen ein verschlechtertes Erkennungsverhalten. Zu sehen ist ein totaler Ausfall in vertikaler Fahrtrichtung innerhalb des rechten Abschnittes, sowie in vielen Bereichen des oberen Teils. Dieses Verhalten wird ebenfalls von anderen Gewerken bestätigt, die vermehrt den alten Algorithmus verwenden. Die Ursache der Ausfälle ist in der falschen Erkennung der RoboterID zu finden. Werden alle Roboter während der Messung betrachtet (vgl. Abbildung 20), sind die Verwechslungen zu erkennen. Schlussfolgern lässt sich, dass die Erfassung der Roboter zwar korrekt funktioniert, die Zuordnung jedoch fehlerhaft ist. Dieser Fehler ist in dem neuen System nicht zu finden (vgl. Abbildung 21), wonach eine höhere Erkennungsgüte resultiert.

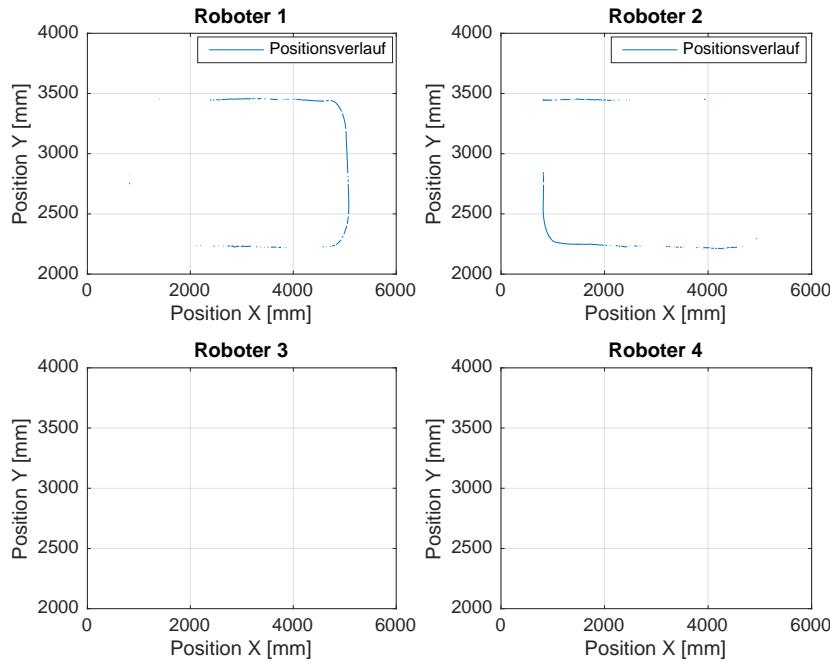


Abbildung 20: Zeitlicher Positionsverlauf aller Roboter unter ausgeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ unter Verwendung des verbesserten Algorithmus.

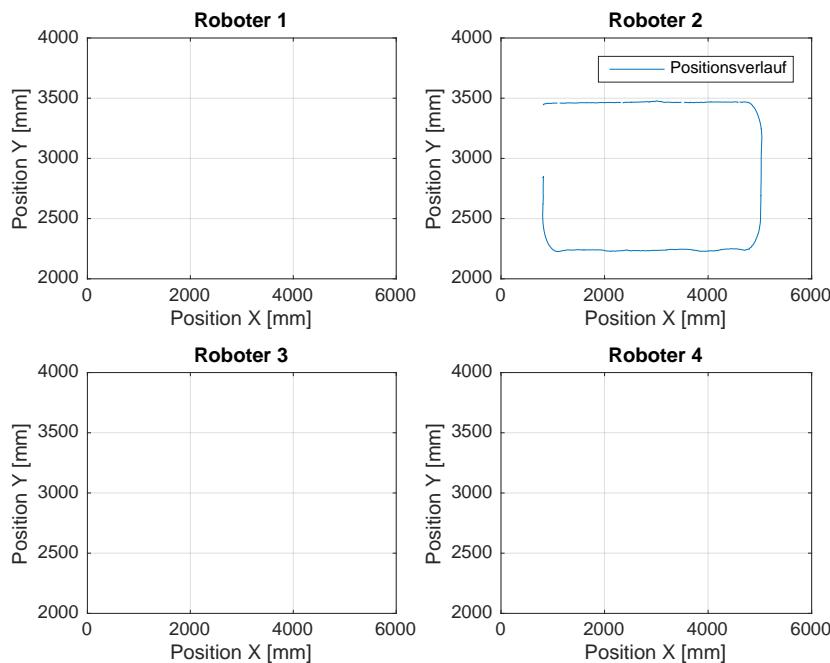


Abbildung 21: Zeitlicher Positionsverlauf aller Roboter unter ausgeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ unter Verwendung des neuen Systems.

Nachdem sowohl das neue als auch das alte System gegenübergestellt wurden, erfolgt der gleiche Versuch unter eingeschalteter Raumbeleuchtung. In dieser Messung wurde

für beide Systeme der Threshold-Wert angepasst, sodass der Roboter in der Mitte einer Kamera korrekt erkannt wird.

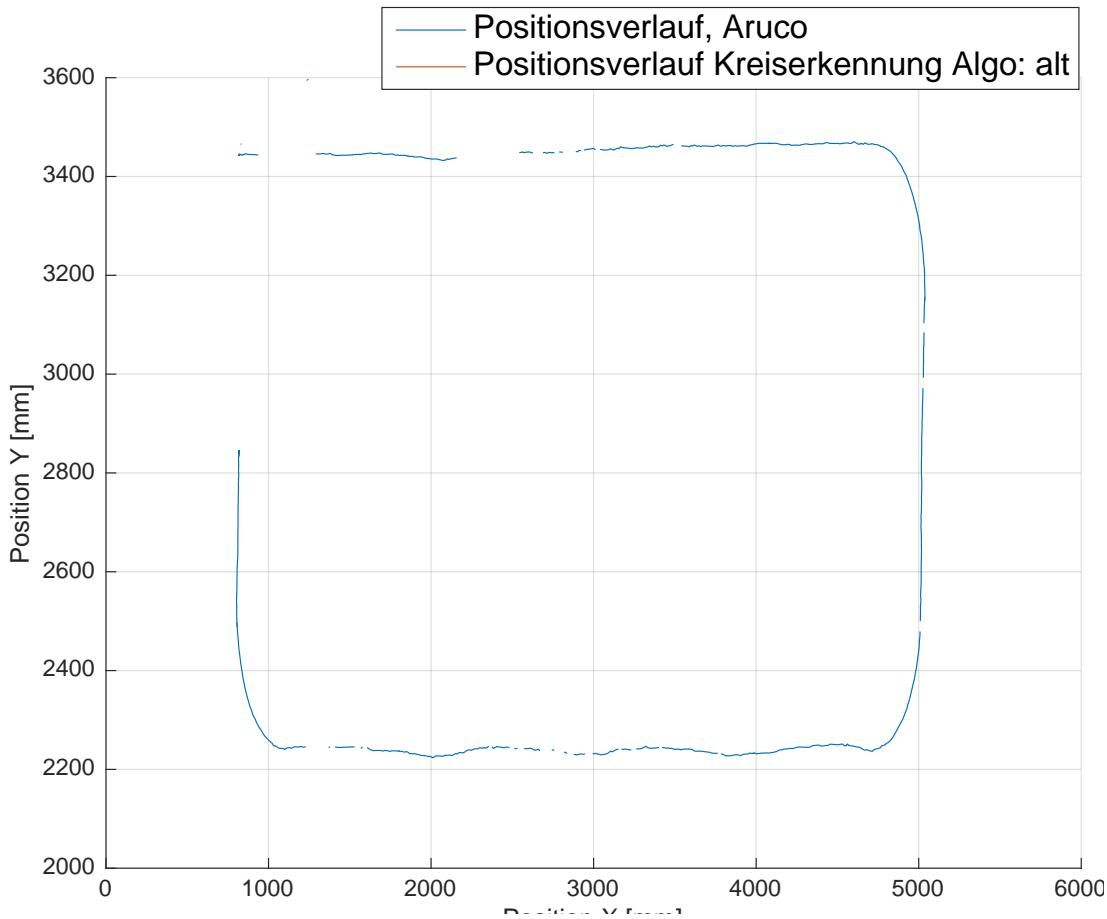


Abbildung 22: Zeitlicher Positionsverlauf des zweiten Roboters unter eingeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ zwischen dem alten Algorithmus und dem neuen System.

Aufgrund der problematischen Erkennung des Roboters in der Anfangsphase der Messung manövriert der Roboter gegen die Wand, was einen Ausfall zur Folge hat. Auch eine Positionsänderung erzielt kein Erfolg, da der Roboter nach wenigen Millimetern nicht mehr erkannt wird. Das neue System hingegen ist in der Anfangsphase kurzzeitig stabil, weist einen längerfristigen Ausfall auf und fängt sich bei ca. 3500 mm/3450 mm (X/Y) wieder. Anschließend sind zwar weitere, jedoch kurzfristige Ausfälle vorhanden. Unter Nutzung eines stabilen Beobachters ist das System somit geeignet auch unter einem ausgeleuchtetem Raum zu fungieren. Der verbesserte Algorithmus weist ein ähnlich instabiles Verhalten auf, dessen Ursache wiederholt in der fehlerhaften Zuordnung der Roboter zu finden ist.

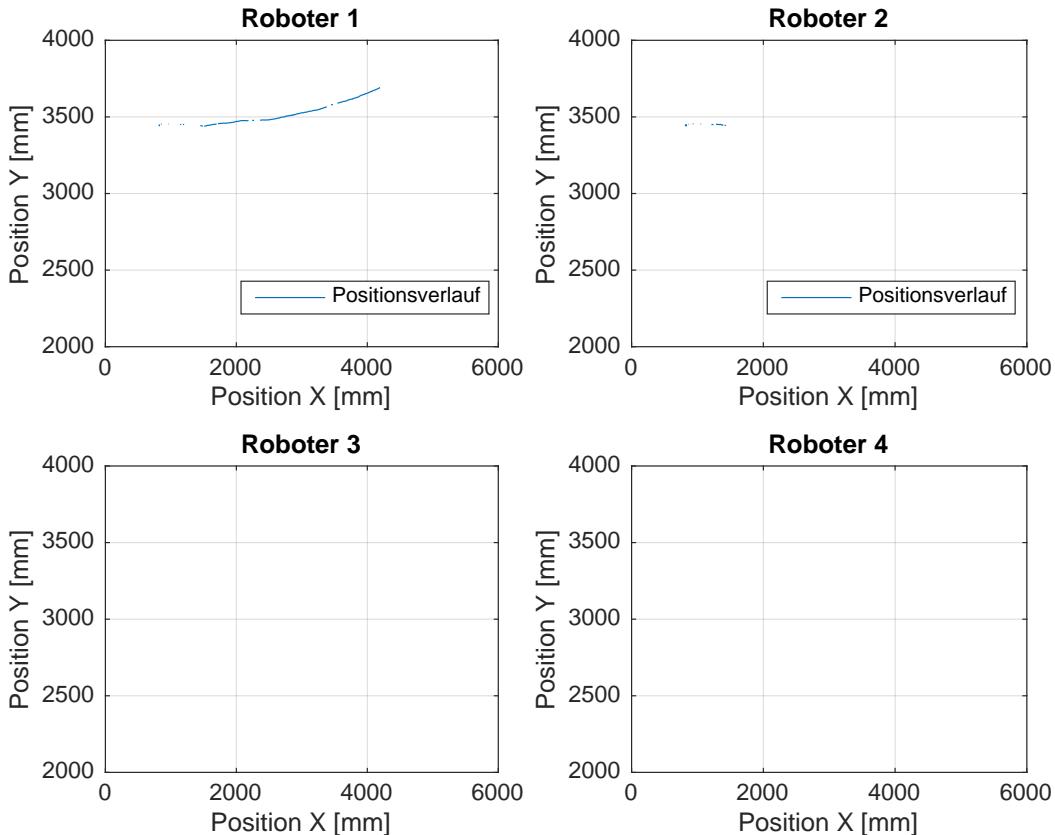


Abbildung 23: Zeitlicher Positionsverlauf aller Roboter mit eingeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ und unter Verwendung des verbesserten Algorithmus.

Zwar funktioniert die Positionserkennung erwartungsgemäß, jedoch stimmt die Zuordnung nicht. Hierdurch erhält der zweite Roboter keine Positionsdaten und fährt autonom auf dem Beobachter. Der Drift in Richtung Wand ist durch den Kurvenverlauf des ersten Roboters zu erkennen. Durch Auslösen des Kollisionsschutzes wird der Roboter gestoppt und die Messung abgebrochen.

Nach Gegenüberstellung der Stabilität werden Messungen in einer Kurve durchgeführt. Diese sollen die vorherigen Messungen stützen und eine Aussage über die Redundanz erlauben. Dafür werden die Bereiche in denen das alte System mit dem alten Algorithmus geringe bis keine Ausfälle hat vergrößert und die Messpunkte differenziert dargestellt. Als geeignete Stelle erscheinen die Kurven, da hier ebenfalls eine Abweichung in der Position vorhanden ist, was der Darstellung zu gute kommt.

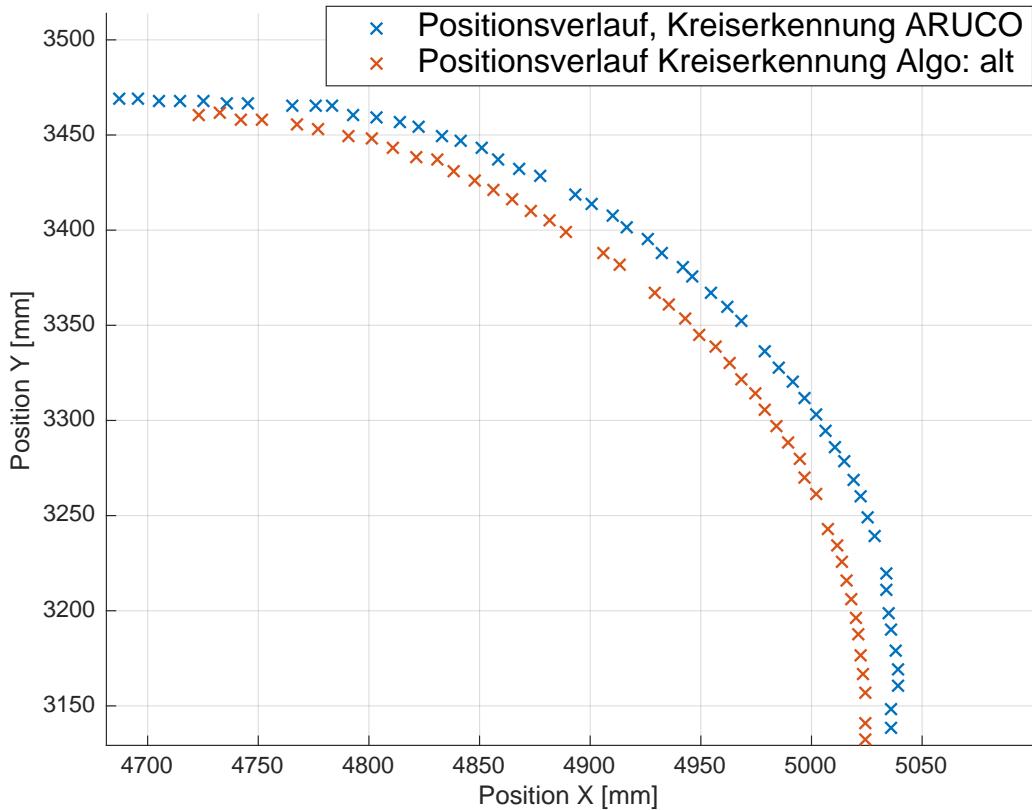


Abbildung 24: Zeitlicher Positionsverlauf des zweiten Roboters in einem Kurvenabschnitt unter ausgeschalteter Raumbeleuchtung bei einer Geschwindigkeit von $350 \frac{\text{mm}}{\text{s}}$ unter Verwendung des ursprünglichen Algorithmus und dem neuen System.

Wie der Grafik 24 zu entnehmen ist, weisen die Messpunkte des alten System eine höhere Dichte auf. Dies spiegelt sich ebenfalls in der Wiederholungsrate des Programmes wieder, die während der Messungen beobachtet werden. Es ist zu erkennen, dass das neue System mehr Rechnerleistung benötigt und somit vier bis zehn Frames langsamer arbeitet. Die Wiederholungsrate ist außerdem von der Anzahl der Roboter abhängig, weshalb die Dichte der Messungen für den zweiten Roboter unter idealen Bedingungen zu bewerten ist. Ein Vorteil in der Wiederholrate ist unter eingeschaltetem Licht zu finden, da die Framerate durch Einschalten der Raumbeleuchtung nur geringfügig absinkt, was bei dem alten System nicht gegeben ist. Dieses Verhalten ist mit den großen und reflektierenden Flächen der Aruco - Marker zu begründen, da nach dem Thresholding nur sehr wenige bis keine ähnlich großen und stark reflektierenden Objekte übrig sind.



3 Uhrsynchronisation

In der weiteren Dokumentation wird auf die Uhrsynchronisation eingegangen. Dabei werden teilweise englische Begriffe verwendet, da an diesen Stellen ein direkter Bezug zu dem in englischer Sprache geschriebenen und kommentierten Programm hergestellt wird. Außerdem ist das ursprüngliche Programm für acht Roboter ausgelegt. Dies wurde beibehalten, wodurch die Uhrsynchronisation auch erfolgreich stattfindet, wenn Roboter mit den Nummern fünf bis acht per WLAN mit dem Kamera-PC in Kontakt stehen.

3.1 Theorie und Verfahren zur Synchronisation der Uhren

Grundlegend dienen Uhren zum gleichzeitigen, synchronen Ausführen von Aufgaben oder ähnlichen zeitkritischen Vorgängen zwischen Personen und/oder Geräten [AAH97, Seite: 10]. Bei technischen Systemen wird zwischen einer impliziten und einer expliziten Zeit unterschieden. Bei impliziten Systemzeiten gibt es keine Uhr, die im Hintergrund arbeitet, sondern Abläufe in Hard- und Software, beispielsweise regelmäßige Triggersignale, welche für einen synchronen Ablauf sorgen. Bei expliziten Systemzeiten ist hingegen eine Uhr implementiert. Der große Vorteil dabei ist, dass Kommunikation und Ausführung voneinander entkoppelt sind. [DM , Seite: 3] Eine Aktion kann also zum gewünschten Zeitpunkt und nicht erst bei dem Eintreffen eines Triggersignals ausgeführt werden. Da es bei der gegebenen Aufgabenstellung generell darum geht, eine Zeitdifferenz von wenigen Millisekunden anzugeben, ist eine explizite Systemzeit ungünstig, da das dazu notwendige Triggersignal eine extreme Belastung für das WLAN-Netz darstellen würde. Aus diesem Grund und weil es in Hinsicht auf Paketverluste sinnvoll ist, die Zeitbestimmung von der Kommunikation zu entkoppeln, ist auf den Robotern eine explizite Systemzeit umgesetzt. Daraus folgt, dass eine Möglichkeit geschaffen werden muss, um vom geplanten Uhr-Master (Kamera-PC) die Uhrzeit an alle Roboter zu übertragen. Im einfachsten Fall kann dazu ein per UDP-Broadcast verteilter Zeitstempel genutzt werden. Da dessen Übermittlung aber mit einer unbekannten, variablen Latenz behaftet ist, ist diese Lösung nicht zufriedenstellend. Zur einmaligen Bestimmung der Latenz kann ein UDP-Telegramm zum Roboter gesendet werden, worauf dieser sofort antworten soll. Die Hälfte der Zeitdifferenz im PC zwischen dem Absenden des ersten Telegramms und dem Eintreffen der Antwort des Roboters entspricht dabei der Latenz. Theoretisch können somit die aktuelle Uhrzeit des Kamera-PCs und die gerade berechnete Latenzzeit an die Roboter versendet werden. Zum Zeitpunkt des Versendens kann die Latenzzeit allerdings schon wieder von der zuvor berechneten Zeit abweichen. Aus diesem Grund ist eine Möglichkeit der Uhrsynchronisation anzuwenden, bei der die Bestimmung der Latenzzeit und der aktuellen Uhrzeit gekoppelt sind. Diese Möglichkeit bietet das in IEEE 1588 spezifizierte Precision Time Protocol (PTP). Dabei handelt es sich um ein Protokoll, welches auf IP Multicast Kommunikation basiert [DM , Seite: 11] und das Synchronisieren von Uhren im Millisekundenbereich ermöglicht [Wei04, Seite: 2]. Der Ablauf dieses Verfahrens ist in der folgenden Abbildung dargestellt:

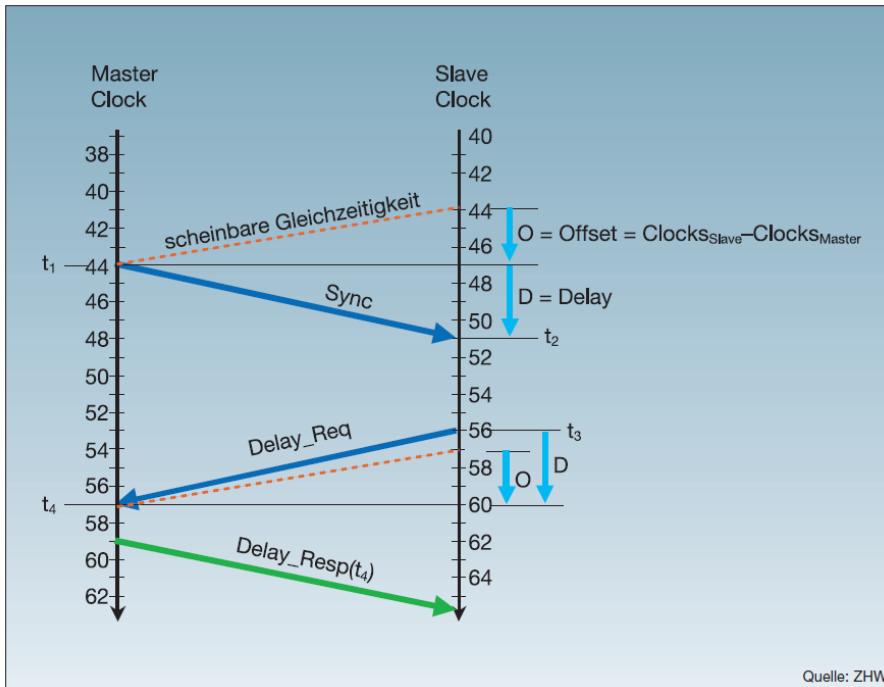


Abbildung 25: Delay und Offset-Berechnung nach IEEE 1588 [Wei04, Seite: 2]

Die orangen Linien zeigen dabei den Versatz der Uhren an und die blauen bzw. der grüne Pfeil stellen Meldungen, also Telegramme, zwischen Uhr-Master und Uhr-Slave dar. Der eigentliche Vorgang der Synchronisation lässt sich in zwei Punkte unterteilen. Zunächst wird vom Uhr-Master eine Sync-Meldung an den Slave gesendet, welche als Zeitstempel die Zeit t_1 beinhaltet und zum Zeitpunkt t_2 beim Slave eintrifft. Nach diesem Austausch könnte, wie zuvor beschrieben, der Slave bereits seine Uhr auf den Zeitstempel t_1 stellen, wodurch diese mit der zur Übertragung benötigten Zeit versetzt laufen würde. Um diese Abweichung zu beseitigen, wird daraufhin, zum Zeitpunkt t_3 , eine Delay_Req Meldung vom Slave an den Master gesendet. Der Master antwortet abschließend auf diese Anfrage und übermittelt dabei den Zeitpunkt t_4 , zu dem der Delay_Req bei ihm eingetroffen ist. Nach diesem Ablauf können mit den vier im Slave gespeicherten Zeitstempeln der Uhrenversatz, auch Offset genannt (Formel 2) und die Übertragungszeit bzw. Delay (Formel 3) wie folgt berechnet werden:

$$d = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \quad (2)$$

$$o = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \quad (3)$$

Bei dem hier vorgestellten Telegrammablauf handelt es sich bereits um eine vereinfachte Version, die eine Genauigkeit im Millisekundenbereich erreicht. Für präzisere Anwendungen wird Hardware direkt am Netzzugang integriert, welche eine exaktere Bestimmung der Zeitstempel ermöglicht. Bei einer reinen Softwarelösung variiert beispielsweise die Zeit, mit der die interruptgesteuerten Routinen zum Auslesen des Empfangspuffers aufgerufen werden [DM , Seite: 7]. Mit Maßnahmen dieser Art und unter anderem höher auflösenden Zeitstempeln arbeitet seit 2005 die Arbeitsgruppe P1588 an einer Synchronisierung im Nanosekundenbereich. [DM , Seite: 9]. Zu beachten ist, dass einmal gestellte Uhren fortlaufend und in den Anforderungen entsprechenden Zeitabständen neu synchronisiert werden,



da sogar identische Fabrikate unterschiedlich schnell arbeitende, taktgebende Bauteile enthalten [Wei04, Seite: 3].

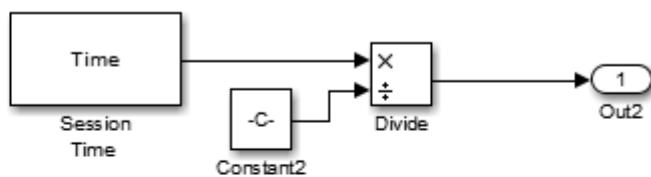
3.2 Umsetzung der Uhrsynchronisation

In Matlab Simulink gibt es die Möglichkeit, unter starken Hardwarebeschränkungen fertige PTP-Bausteine zur Synchronisation zu verwenden. Dazu sind spezielle Netzwerkkarten [Corb] eines bestimmten Herstellers und spezielle Switches notwendig [Cora]. Um unter den gegebenen Hard- und Softwareumgebungen arbeiten zu können, ist sich folglich für eine reine Softwarelösung entschieden wurden. Außerdem wird nicht der standardisierte Protokollstack des PTP verwendet, sondern es wird ein eigener Telegrammablauf per UDP implementiert. Für die Umsetzung des IEEE 1588 Protokolls gilt es, eine angepasste, auf die Roboter zugeschnittene Eigenentwicklung auf Basis des PTP zu entwickeln.

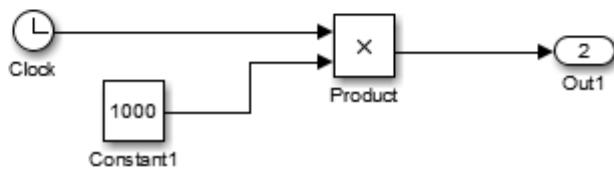
3.2.1 Uhrauswahl

Das Synchronisationsverfahren nach IEEE 1588 setzt voraus, dass auf dem Slave, also den Robotern, eine zeitgebende Funktion implementiert ist. Diese ergibt, verrechnet mit dem Offset, welcher mit Formel 3 bestimmt wird, die synchronisierte Uhr. Da ein Format der Art hh:mm:ss:ms mit aufwendiger Strukturierung verbunden ist und nicht benötigt wird, ist eine vereinfachte Variante umgesetzt wurden. Dabei wird auf jedem Roboter eine Variable vom Typ double verwendet, welche die Anzahl der Millisekunden seit Mitternacht enthält. Für diese zeitgebende Funktion wurden die drei in Frage kommenden Möglichkeiten gegenübergestellt und getestet:

1)



2)



3)

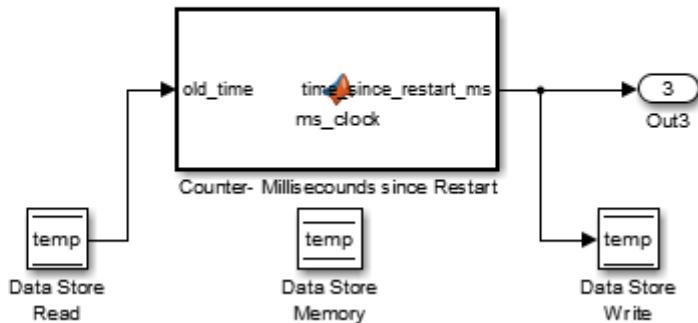


Abbildung 26: Uhren in Simulink

Bei Uhr 1 handelt es sich um den Zähler des CPU-Taktes, welcher dividiert mit $1.193e+3$ die Systemzeit in Millisekunden, seit Einschalten des Roboters, angibt. Der Zahlenwert der Uhr 2 entspricht der Simulationszeit in Mikrosekunden bzw. in Millisekunden nach der Multiplikation mit 1000. Bei Uhr 3 handelt es sich um einen Eigenbau. Dabei wird bei jedem Aufruf der Funktion der vorherige Wert um 5 erhöht. Mit einer eingestellten Solvertime von 5 ms ergibt sich auch somit eine zeitgebende Funktion. Es gilt zu überprüfen, ob der zeitliche Verlauf von Uhr 1, welcher direkten Bezug zur Hardware hat, von den anderen beiden Uhren abweicht. Über eine Langzeitmessung von 15 Minuten wurden die drei Uhren-Verläufe aufgezeichnet und auf eine Zeitabweichung hin überprüft. Dabei zeigen alle drei Verläufe dieselbe Steigung.

Es ist nicht möglich, eine Aussage zu treffen, wie exakt eine Millisekunde des PC104-Boards verglichen mit einer Millisekunde einer „echten“ Uhr ist. Handelsübliche Mainboards und Industrie PCs wie das verwendete PC104-Board sind in der Regel nicht mit einer Echtzeituhr ausgestattet. Diese Eigenschaft steht nicht in direktem Zusammenhang mit der Echtzeitfähigkeit des Systems. So kann das PC104-Board die Ansprüche der Echtzeitfähigkeit zwar erfüllen, die Taktgenauigkeit der CPU reicht aber unter Umständen nicht aus, um damit einmal gestellte Uhren über einen langen Zeitraum taktgenau ohne Abweichungen laufen zu lassen.

Im Simulink Programm wird die Uhr 3 verwendet. Die drei Uhren zeigen zwar gleiches Verhalten, aber orientiert am Grundgedanken nach IEEE 1588 stellt diese Uhr die hardwarenahste Variante dar.

3.2.2 Methoden nach IEEE 1588 zum Synchronisieren der Uhren

Im Folgenden werden drei Implementierungsmöglichkeiten einer Uhrsynchronisation auf Basis des PTP vorgestellt. Bei allen Varianten ist der Kamera-PC der Uhr-Master und die Roboter bilden die Uhr-Slaves. Die Kommunikation erfolgt per UDP-Telegramme, welche für ein übersichtliches Handling folgende Struktur aufweisen:

Variable (double)	Funktion
1.	ID (z.B. ID 1113 -> ID1 von Roboter 3)
2.	Roboter Nummer
3.	Nutzdaten (z.B. Zeitstempel)
weitere	Nutzdaten

Tabelle 7: Standard Telegrammstruktur für die Uhrsynchronisation

Dank der Verwendung einer eindeutigen ID für jedes Telegramm ist es möglich, durch das Hinzufügen weiterer IDs den selben UDP-Socket zum Senden oder Empfangen zu nutzen, ohne eine Fehlfunktion der Software zu bewirken, da dieses Merkmal als Identifizierung im Programm verwendet wird. Die im Folgenden gezeigten beispielhaften Abläufe der Uhrsynchronisation dienen ausschließlich als Illustration und Visualisierung des Ablaufs. Latenzzeiten und die Zykluszeit, in der die Kameradaten versendet werden, wurden frei gewählt. Aus diesem Grund dienen die Zahlenwerte, welche einer Zeit in Millisekunden entsprechen könnten, nur als Richtwert und lassen keinen Rückschluss auf die Geschwindigkeit der Synchronisation zu.

Methode 1:

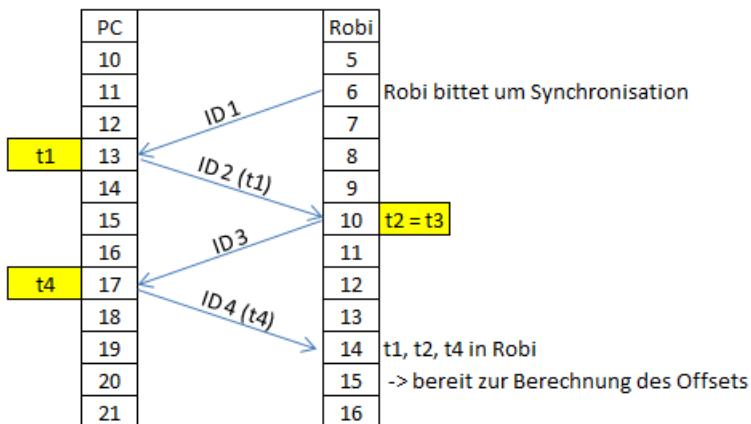


Abbildung 27: Telegrammablauf Methode 1

Bei dieser Variante ist eine eigene UDP-Verbindung zum Austausch der notwendigen Telegramme umgesetzt. Dabei fragt der Roboter zyklisch nach einer Synchronisation und der PC antwortet daraufhin mit dem Zeitstempel t_1 . Ist dieser von dem Roboter empfangen und ausgewertet wurden, wird noch im selben Zyklus die zweite Hälfte des Ablaufs zur Bestimmung der Delay-Zeit gestartet. Dazu übermittelt der Roboter die ID3 und erhält als Antwort den Zeitstempel t_4 . Dabei ist die Besonderheit, dass sich die Berechnung von Offset und Delay dahingehend vereinfacht, dass die Zeitstempel t_2 und t_3 identisch sind. Somit können mit den Zeitstempeln t_1 , t_2 und t_4 die gesuchten Größen bestimmt werden.

Methode 2:

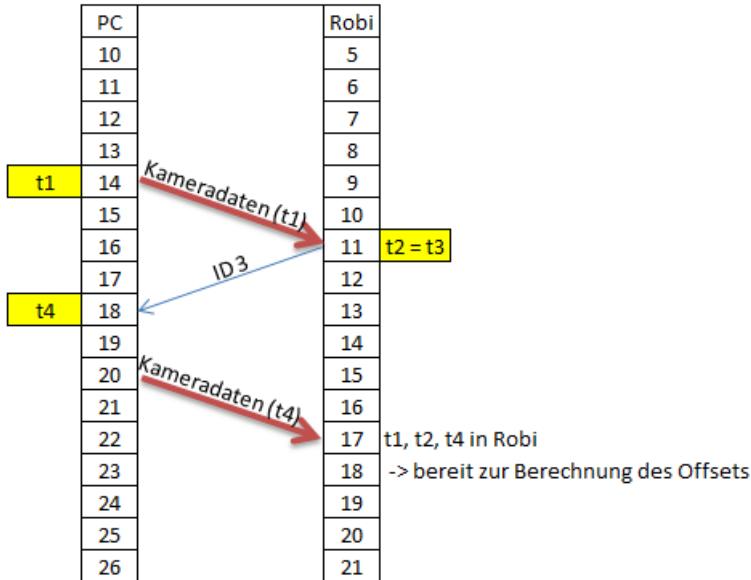


Abbildung 28: Telegrammablauf Methode 2

In diesem Fall sind die Kameradaten, also das Broadcast-UDP-Telegramm des Kamera-PCs, welches die Postion der Roboter verbreitet, für den Telegrammablauf zur Synchronisation der Uhren mitgenutzt worden (rote Pfeile vom PC zum Roboter). Zum besseren Verständnis der Umsetzung dient dieser Ausschnitt des UDP-Telegramms des Kamera-PCs, für speziell dieses Verfahren:

Variable (double)	Funktion
...	...
31.	ID für Roboter 1
32.	ID für Roboter 2
33.	ID für Roboter 3
...	...
41.	Daten (Zeitstempel) für Roboter 1
42.	Daten (Zeitstempel) für Roboter 2
43.	Daten (Zeitstempel) für Roboter 3
...	...

Tabelle 8: Standard Telegrammstruktur für die Uhrsynchrosnisation

Die Struktur dieses Telegrammausschnitts bietet sich an, da somit die eingetroffenen Daten auf dem Roboter mit übersichtlichen Schleifen ausgewertet werden können. Bei dieser Methode der Uhrsynchrosnisation sorgt eine im Robot-Detection Programm implementierte Logik dafür, dass die zu diesem Zeitpunkt von den Robotern benötigten Daten im Nutzdatenteil (Variable 41-48) des Telegramms stehen. Ein Beispielablauf ist im Folgenden für Roboter eins erklärt: Dieser empfängt die per UDP-Broadcast verteilten Kameradaten mit der ID1 (Variable 31) zum Zeitpunkt t_2 . Aufgrund der ID ist dem Roboter somit bekannt, dass im Nutzdatenteil (Variable 41) der Zeitstempel t_1 enthalten ist. Unmittelbar nach dem Empfang antwortet der Roboter mit dem Telegramm der ID3. Trifft dieses

auf dem Kamera-PC ein, wird dort der Zeitstempel t_4 aufgenommen. Dieser wird in den Nutzdatenteil (Variable 41) geschrieben und die ID für diesen Roboter wird auf 4 gesetzt (Variable 31). Diese Information wird somit automatisch mit den nächsten Kameradaten an den Roboter übertragen. Treffen diese ein, erkennt die auf dem Roboter arbeitende Software an der ID4, dass sich im Nutzdatenteil der Zeitstempel t_4 befindet. Zu diesem Zeitpunkt sind t_1 , t_2 und t_4 auf dem Roboter eingetroffen bzw. erfasst worden und es kann die Berechnung der gesuchten Größen erfolgen. Die Besonderheit bei dieser Umsetzung liegt darin, dass kein zusätzlicher UDP-Dienst auf dem Kamera-PC implementiert wird, sondern ausschließlich die Kameradaten zur Uhrsynchronisation genutzt werden. Ähnlich wie bei Methode 1 ist auch hierbei der Zeitstempel t_2 gleich t_3 .

Methode 3:

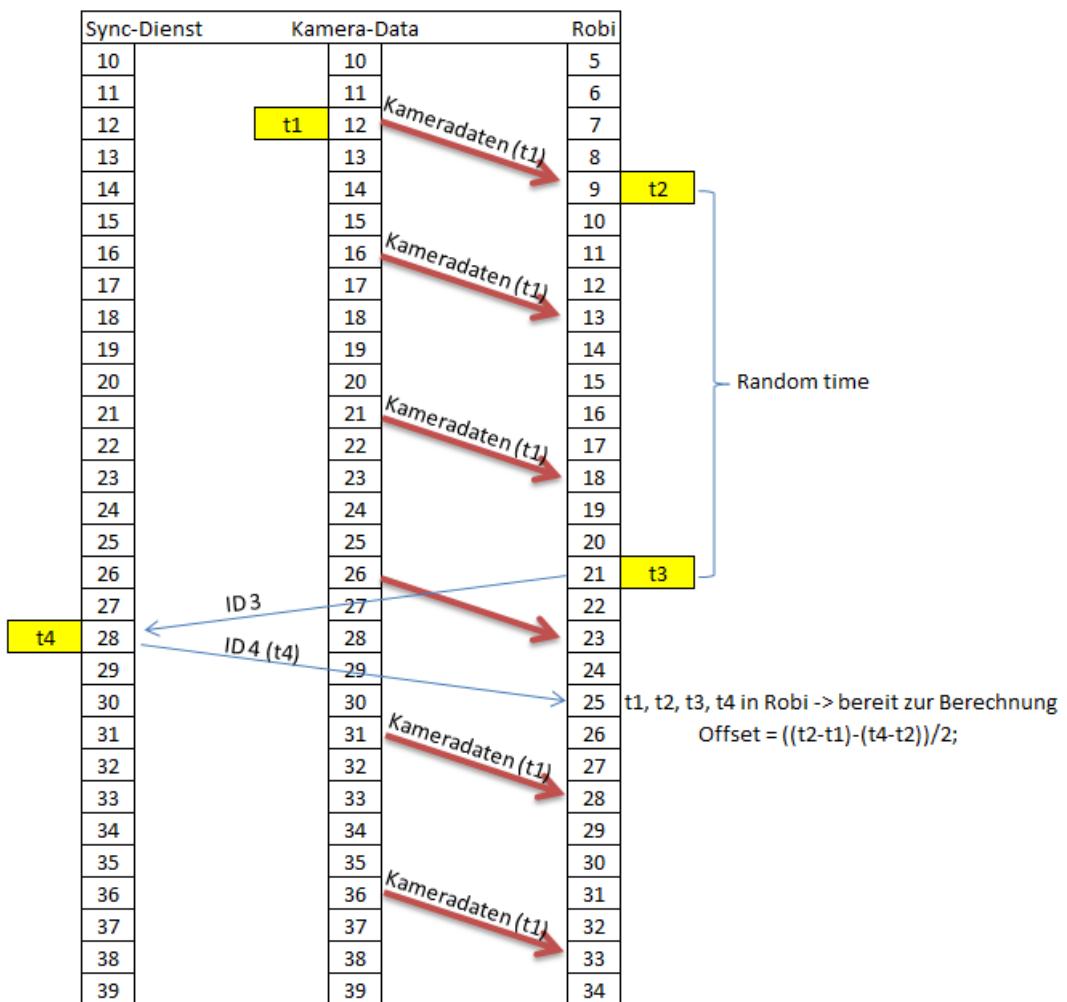


Abbildung 29: Telegrabblauf Methode 3

Diese Lösung stellt eine Kombination der beiden zuvor geschilderten Abläufe dar. Dabei wird ein zusätzlicher UDP-Dienst auf dem Kamera-PC implementiert und die Kameradaten werden zur Uhrsynchronisation genutzt. In letzteren ist ein Zeitstempel angehängt, welcher den Sendezeitpunkt t_1 des jeweiligen Telegramms enthält. Wird ein solcher Datensatz von einem Roboter empfangen, wird der Empfangszeitpunkt t_2 gespeichert und eine Zufallszeit zwischen 300 ms und 400 ms läuft ab. Während dieser Zeit haben eintreffende Kameradaten

keine Auswirkung auf den aktiven Synchronisationsvorgang. Ist die Zeit abgelaufen, wird der Zeitpunkt t_3 gespeichert und ein Telegramm der ID3 an den Kamera-PC, noch genauer an den zusätzlich implementierten Sync-Dienst, gesendet. Trifft dieses dort ein, wird der Zeitstempel t_4 erfasst, welcher mit einem Telegramm der ID4 an den Roboter übermittelt wird. Auf diesem sind anschließend die Zeitstempel t_1 , t_2 , t_3 und t_4 gespeichert und die gesuchten Größen zur Synchronisation der Uhren können berechnet werden.

3.2.3 Gegenüberstellung der Methoden zur Uhrsynchronisation

Alle drei vorgestellten Möglichkeiten funktionieren und führen zudem selben Ergebnis. Methode 1 ist dabei komplett von den Kameradaten entkoppelt. Ist dies gewünscht, führt dieses Vorgehen zu einem kompakten Quellcode auf den Master- und Slave-Systemen und es ergibt sich eine gut überschaubare Logik durch den sequentiellen Ablauf. Allerdings bietet es sich für die hier vorliegende Anwendung an, die Synchronisation in die Kameradaten einzubetten, da diese Telegramme ohnehin alle 20 ms - 50 ms (abhängig von der Bildverarbeitungszeit des Kamera-PCs) versendet werden. Dazu würde sich Methode 2 eignen, allerdings ist für diese Umsetzung sehr viel Logik notwendig. In der Robot-Detection Software müssen zu den entsprechenden Zeitpunkten die passenden Zeitstempel im Telegramm eingebettet sein, welche von der Gegenseite anhand der ID richtig interpretiert werden müssen. Außerdem führt der Zeitpunkt des Eintreffens der Kameradaten dazu, dass alle Roboter, die zu diesem Zeitpunkt einen Zeitstempel t_1 erwarten, unverzüglich mit der ID3 antworten. Dies bringt zwangsläufig Kollisionen mit sich und verringert die Anzahl der erfolgreichen Synchronisierungsvorgänge. Abhilfe dagegen würde eine variable Wartezeit schaffen, welche in Methode 3 umgesetzt ist. Diese Wartezeit wird mithilfe einer Zufallszahl festgelegt, damit einer mehrfach auftretenden Kollision mit anderen, zyklisch sendenden Netzwerkgeräten aus dem Weg gegangen werden kann. Der zusätzliche Sync-Dienst erleichtert die Verständlichkeit des Ablaufs und macht die komplexe Umschaltlogik von Methode 2 überflüssig. Zum Abwägen der Möglichkeiten wurden die Vor- und Nachteile in einer Tabelle gelistet:

	Methode 1	Methode 2	Methode 3
Beschreibung	Eigener UDP-Dienst zur Synchronisation	Synchronisation erfolgt alleine mit Kameradaten	Kameradaten und zusätzlicher UDP-Dienst
Vorteil	<ul style="list-style-type: none"> - einfach nachzuvollziehender Telegrammablauf - gut auf andere Anwendungen zu übertragen 	<ul style="list-style-type: none"> - kein zusätzlicher UDP-Dienst notwendig 	<ul style="list-style-type: none"> - Kameradaten werden auf einfachste Weise mitgenutzt - simpler, zusätzlicher UDP-Dienst sorgt für einfachen Telegrammablauf
Nachteil	<ul style="list-style-type: none"> - Telegramm der Kameradaten bleibt ungenutzt 	<ul style="list-style-type: none"> - komplexer Telegrammablauf - verzögertes / chaotisches Verhalten bei Kollisionen 	<ul style="list-style-type: none"> - zusätzlicher UDP-Dienst

Tabelle 9: Vor- und Nachteile der drei Methoden zur Uhrsynchronisation

Nach diesen Überlegungen wird als passende Umsetzung die Methode 3 gewählt. Deren Nachteil ist bei der geplanten Anwendung zu ignorieren, da es diesbezüglich keine Beschränkungen oder Vorgaben gibt. Somit bleiben die Vorteile, wodurch die Uhrsynchronisation zu einem guten Ergebnis führen wird. Im weiteren Dokument ist ausschließlich von dieser gewählten Methode die Rede.

3.3 Programmaufbau

Das Programm der Slaves (Roboter) ist in Matlab Simulink umgesetzt. Für den Uhr-Master (Kamera-PC) wird das bestehende Programm „Robot-Detection“ an den notwendigen Stellen erweitert und angepasst. Der grundlegende Kommunikationsablauf, um die Uhrsynchronisation zu implementieren, ist in dem folgenden Flowchart dargestellt:

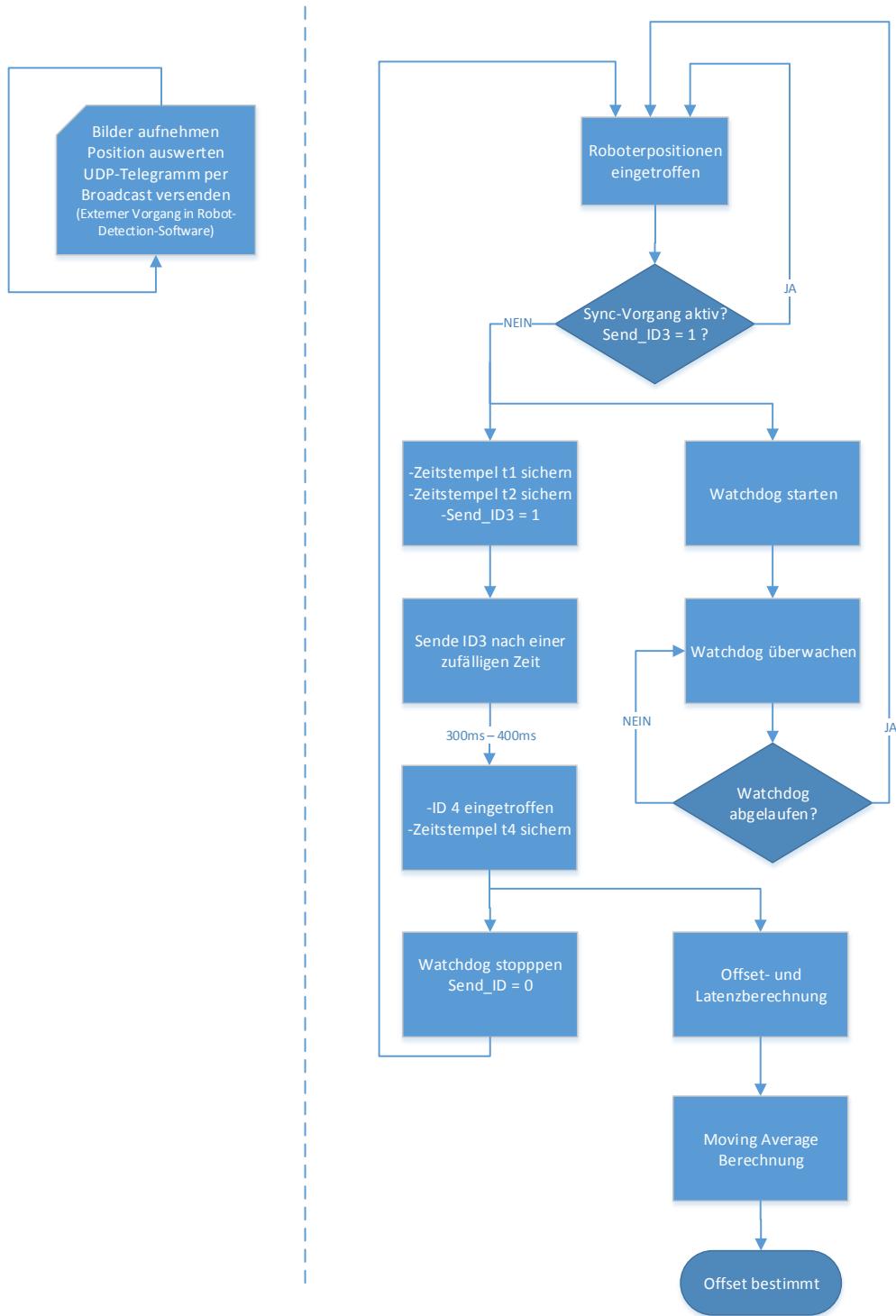


Abbildung 30: Programmablaufplan

Links von der gestrichelten Linie ist der Programmablauf dargestellt, welcher auf dem Kamera-PC implementiert ist und rechts der der Roboter. Der Kamera-PC nimmt nach wie vor in einer Schleife die Bilder auf, wertet daraus die Positionen aus und verteilt diese anschließend per UDP-Broadcast. Die Software auf den Robotern arbeitet wie folgt: Treffen auf einem Roboter neue Kameradaten ein, prüft dieser, ob bereits ein Synchronisationsvorgang aktiv ist. Falls nein, wird ein neuer gestartet, falls bereits ein Vorgang aktiv ist,



wird nichts weiter unternommen. Zur besseren Veranschaulichung ist der Zustand der Variablen send_ID3 in den entsprechenden Schritten mit angegeben. Dieser spiegelt im Simulink Programm wieder, ob ein Synchronisationsvorgang aktiv ist oder nicht. Wird also ein neuer Vorgang gestartet, wird der Zeitstempel t_1 aus den empfangenen Daten gespeichert. Außerdem wird der Zeitstempel t_2 gesichert, welcher dem Empfangszeitpunkt entspricht. Zusätzlich wird ein Watchdog-Timer gestartet, dessen Funktion im späteren Verlauf detaillierter erläutert wird. Daraufhin wird das Senden der ID3 vorbereitet und eine Zufallszeit ermittelt, mit dessen Verzögerung das Telegramm zu einem Zeitpunkt t_3 versendet wird. Trifft dieses auf dem Kamera-PC ein, speichert er dazu einen aktuellen Zeitstempel t_4 und versendet diesen als Telegramm mit der ID4 an den Roboter. Dort eingetroffen, wird der Zeitstempel aus den Daten gelesen und gespeichert. Somit befinden sich zu diesem Zeitpunkt wieder alle notwendigen Größen zur Berechnung von Offset und Delay auf dem Roboter. Der Watchdog kann gestoppt werden und Send_ID3 wird wieder auf false gesetzt, wodurch ab diesem Zeitpunkt wieder eine neue Synchronisation beim Eintreffen neuer Kameradaten gestartet werden kann. Gleichzeitig wird der errechnete Offset auf ein Moving Average Filter gegeben, wodurch ein geglätteter Verlauf des Offsets entsteht.

Der Watchdog hat dabei die Aufgabe, bei Telegrammverlusten, was bei der Übermittlung per UDP durchaus vorkommen kann, das Programm wieder in einen definierten Zustand zu bringen, um einen Deadlock zu verhindern. Dazu wird dieser beim Beginn des Synchronisationsvorgangs gestartet und sobald das Telegramm mit der ID4 vom PC eingetroffen ist, gestoppt. Bleibt dieses Telegramm allerdings aus, weil ID3 oder ID4 nicht erfolgreich übermittelt wurden, läuft der Timer ab. Ist dies der Fall, wird Send_ID3 auf false gesetzt und es kann mit den nächsten eintreffenden Kameradaten eine neue Synchronisation gestartet werden. Bis dahin gespeicherte Zeitstempel werden dabei überschrieben und führen nicht zu einer fehlerhaften Berechnung.

3.3.1 Anpassungen in Robot-Detection

Die notwendigen Anpassungen mit der IDE Qt beschränken sich auf die Klassen myudp und robotdetectionmainwindow. Die Änderungen werden im Folgenden näher erläutert und beschrieben. Der Telegrammaufbau der Kameradaten, auf den im Weiteren vermehrt eingegangen wird, findet sich im Anhang in Tabelle 11.

Änderung in myudp.cpp

In dieser Klasse befindet sich der neu hinzugefügte UDP-Dienst und das angepasste Telegramm der Kameradaten. Die zusätzliche Kommunikationsschnittstelle ist notwendig, da bei der gewählten Methode der Synchronisierung der Telegrammbau auf zwei Dienste aufbaut. Da zwischen diesen beiden ein Daten- und Variablenaustausch stattfindet, bietet es sich an, die neu zu implementierenden Funktionen und Schnittstellen in der bestehenden Klasse vorzunehmen. Anhand des UML-Diagramms werden die Funktionen im Folgenden näher erläutert, mit „->“ gekennzeichnete Funktionen stellen neu erstellte Programmteile dar:

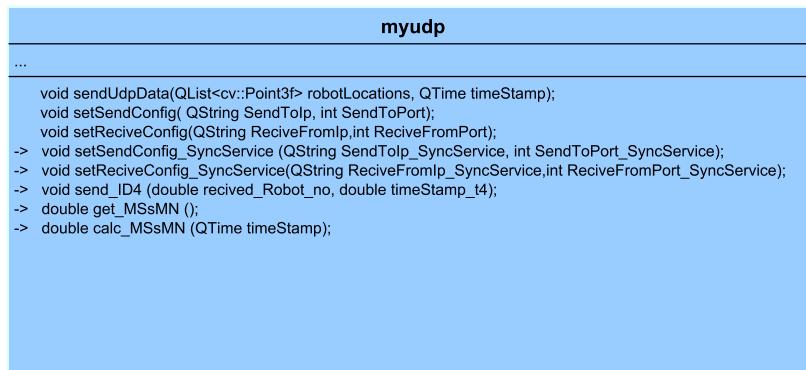


Abbildung 31: UML-Diagramm der Methoden der Klasse myudp

Für die neue Schnittstelle sind, angelehnt an den Ausgangszustand, neue send- und receive-Config-Funktionen implementiert. Da dieser neue UDP-Socket Daten empfangen muss, ist es notwendig, dem Socket mit der Funktion „bind“ die Empfangsparameter, IP-Adresse und Port zu übergeben. Als IP-Adresse ist dabei die des Kamera-Rechners anzugeben. Treffen bei dieser Konfiguration Daten auf dem PC ein, warten diese im Empfangspuffer auf ihre Weiterverarbeitung. Dies wird mit dem Signal readyRead zurückgemeldet, welches in dem Fall true wird. Die gängige Art, solche Signale in Qt zu verarbeiten, sind Signal/Slot-Verbindungen. Dabei führt das Signal dazu, dass eine gewünschte Funktion ausgeführt wird. Hierbei handelt es sich um die programmierte Funktion readyRead_SyncService. Diese erstellt zu Beginn den Zeitstempel t_4 und prüft anschließend, ob es sich bei den eingetroffenen Daten um ein Telegramm der ID3 handelt. Ist dies der Fall, wird die Funktion send_ID4 aufgerufen. Darin wird die Telegrammstruktur nach Tabelle 7 aufgebaut, der übergebene Zeistempel t_4 darin eingebettet und abschließend das Telegramm per Broadcast versendet.

Die Funktion sendUpdData wird vom Hauptprogramm aufgerufen, sobald die Berechnungen der zuletzt aufgenommen Bilder abgeschlossen sind und die Positionen zum Versenden bereit stehen. Darin wird das Telegramm nach der Struktur, welche im Anhang (Tabelle 11) zu finden ist, zusammen gestellt. Dabei wurden zwei Zeitstempel und acht Sende IDs an freie Reservepositionen im Telegramm eingefügt. Der erste Zeitstempel an Stelle 28 des Telegramms enthält den Zeitstempel, im bekannten „Millisekunden seit Mitternacht Format“, wann das Bild aufgenommen wurde. Der zweite befindet sich an 56. und somit an letzter Stelle des Telegramms. Dies liegt daran, dass dieser den Absendezeitpunkt darstellt, also dem Zeitstempel t_1 entspricht. Je später dieser im Algorithmus bestimmt wird, desto näher liegt der Zeitpunkt am wirklichen Absendezeitpunkt, desto genauer ist die Synchronisierung. Weiterhin wurde an den Stellen 31 bis 38 die ID für das zu übertragende Telegramm hinzugefügt. Grund dafür ist, dass zum einen über dieselbe Schnittstelle weitere Telegramme mit anderen IDs versendet werden können. Zum anderen ist somit ausgeschlossen, dass es zu Fehlinterpretationen kommt, wenn andere Gewerke auf demselben Port per Broadcast Daten versenden.

Die Funktion calc_MSsMN bestimmt aus einem Standard Qt-Zeitstempel eine double-Variable im „Millisekunden seit Mitternacht Format“. Diese Funktion ist notwendig, wenn ein Zeitstempel wie jener an Stelle 28 im Telegramm an die Klasse myudp übergeben wird. Für den Fall, dass der Zeitstempel hingegen in der eigenen, myudp Klasse erstellt wird, ist die Funktion get_MSsMN implementiert. Diese liefert als Return-Wert eine double-Variable im „Millisekunden seit Mitternacht“ Format.



Änderung in robotdetectionmainwindow.cpp

Im Konstruktor der Klasse robotdetectionmainwindow werden für das vorhandene Objekt udpClient der Klasse myudp die Funktionen zum Konfigurieren des Sendens und Empfangens aufgerufen.

IP-Adressen und Ports

Zum Senden und Empfangen wurden jeweils getrennte Ports verwendet. Grund dafür ist, dass davon eine große Anzahl zur Verfügung steht und sich die Übersichtlichkeit und eventuelle Fehlersuche dadurch deutlich besser gestalten.

	Sendet an Port	Empfängt auf Port	IP-Adresse (PC)
Kameradaten	25000	-	192.168.0.20
Sync.-Dienst	25111	25110	192.168.0.20

Tabelle 10: Verwendete IP-Adressen und Ports

3.3.2 Simulink Programm

Zur Umsetzung der Uhrsynchronisation auf den Robotern ist ein Simulink Programm erstellt worden. Dieses setzt den in Abbildung 30 (rechte Hälfte) gezeigten Telegrammablauf und weitere Zusatzfunktionen um. Stark abstrahiert lässt sich das Simulink Programm mit der folgenden schematischen Darstellung erläutern. Dabei wurden nur die obersten Funktionsblöcke dargestellt und lediglich die Ein- und Ausgänge als Signale eingezeichnet. Pfeile zwischen den einzelnen Funktionsblöcken verdeutlichen dabei die zusätzlich notwendigen Signalverbindungen zwischen den Bereichen.

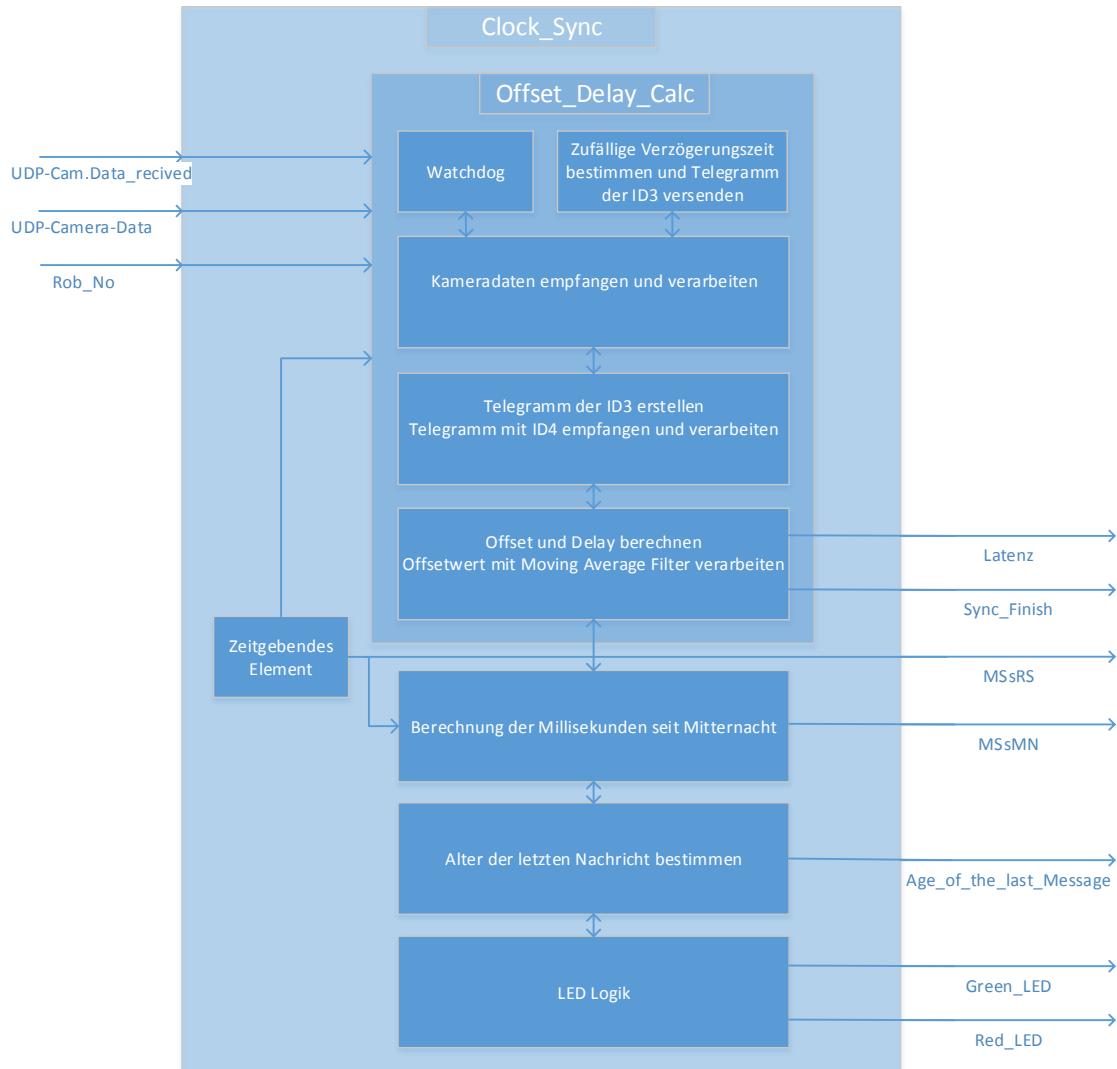


Abbildung 32: Programmaufbau Simulink-Block

In der obersten Ebene befindet sich der Block namens `Clock_Sync` mit drei Eingängen und sieben Ausgängen. Die drei Eingänge können mit Signalen der Starter-Kits bedient werden. `UDP-Camera_Data_recived` wird dabei mit dem Signal des N-Ausgangs des `UDP-Receive Binary Blocks` verschaltet, welcher die Kameradaten empfängt. `UDP-Camera_Data` muss mit dem `UDP-data` Ausgang des `UDP Interface Blocks` (aus dem Starterkit) verbunden werden. Für die Roboter-Nummer, welche an Eingang `Rob_No` angelegt werden muss, gibt es einen eigenen Funktionsblock in den Starter-Kits.

Kleinere Funktionsblöcke in der schematischen Übersicht zeigen kleine Hilfsfunktionen an, die größeren enthalten wiederum weitere Logik im größeren Umfang. Der Block des zeitgebenden Elements enthält die in Kapitel 3.2.1 gewählte Uhr.

Offset- und Delay-Berechnung

Der eigentliche Block der Offset- und Delay-Berechnung (`Offset_Delay_Calc`) enthält fünf Unterfunktionen: Der `Watchdog` führt dabei die in Kapitel 3.3 beschriebene Funktion aus. Der Funktionsblock zum Erstellen einer zufälligen Verzögerungszeit gibt mithilfe der folgenden Programmzeile eines Matlab Skripts einen zufälligen Wert zwischen 300 und 400



aus.

$$\text{random time} = 300 + \text{randi}(100);$$

Dank einer selbstgeschriebenen Funktion wird dieser Wert als Verzögerung in Millisekunden interpretiert. Dadurch startet, um diese Zeit versetzt, der Sendevorgang des Telegramms der ID3, wobei gleichzeitig t_3 bestimmt wird. Weiterhin befindet sich in dieser Ebene ein Funktionsblock zum Entgegennehmen und Auswerten der Kameradaten. Dieser bearbeitet die erste Hälfte des Synchronisationsablaufs, also das Speichern von t_1 und t_2 . Zusätzlich ist dort die Verwaltungslogik implementiert, welche entscheidet, ob ein neuer Synchronisationsvorgang gestartet werden soll oder nicht. Ein weiteres Unterprogramm stellt die zweite Hälfte des Synchronisationsablaufs bereit. Darin wird das Telegramm mit der ID3 erstellt und das mit der ID4 empfangen, woraus der Zeitstempel t_4 gelesen wird. Mit den bisher beschriebenen Funktionsblöcken können bereits alle benötigten Zeitstempel zur Bestimmung von Offset und Delay auf dem Roboter gespeichert werden. Die Berechnung der beiden gesuchten Größen erfolgt in einem weiteren Unterprogramm nach Formel 2 und Formel 3. Der damit bestimmte Offset wird an dieser Stelle auf eine selbstgeschriebene Moving-Average-Funktion gegeben. In dieser Funktion wird jeder neu eingetroffene Offset-Wert an die erste Stelle eines Vektors der Länge 15 geschrieben. Der letzte Wert wird dabei jeweils aus dem Vektor gestoßen und geht somit nicht mehr in die anschließende Mittelwertbildung dieser 15 Zahlen ein. Das Ergebnis der Rechnung bildet den geglätteten Offset. Außerdem wird an dieser Stelle der Sync_Finish Ausgang gebildet, welcher seinen Zustand von false auf true wechselt, sobald der synchronisierten Uhr das erste Mal vertraut werden kann. Dieser Zeitpunkt ist erreicht, wenn 20 mal erfolgreich ein Offset bestimmt wurde.

Der neue, zusätzliche UDP-Dienst kommt bei dem hier beschriebenen Unterprogramm also zum Tragen, indem darüber das Telegramm mit der ID3 versendet wird und das mit der ID4 empfangen wird.

Zusatzfunktionen

Die drei Zusatzfunktionen sind unter dem Block der Uhrsynchronisation dargestellt und stellen nützliche Implementierungen dar, die erst durch die Bestimmung des Offsets möglich sind. Somit wird aus der zeitgebenden Funktion des Roboters und dem berechneten Offset die Größe der Millisekunden seit Mitternacht (MSsMN) nach folgender Formel berechnet:

$$MSsMN = \text{Roboter time} - \text{Offset} \quad (4)$$

Dadurch ergibt sich erstmals eine mit dem Kamera-PC synchrone Größe, wodurch eine Aussage über die Aktualität der eingetroffenen Roboterpositionen in den Kameradaten zu treffen ist. Dazu wird der im UDP-Telegramm der Kameradaten eingebrachte Zeitstempel (Variable 56) mit der aktuellen MSsMN-Zeit des Roboters verrechnet.

$$\text{Age of the last Message} = MSsMN - \text{PC timestamp} \quad (5)$$

Nach jedem Eintreffen neuer Kameradaten liegt dieser Wert um einen Solverschritt verzögert am Ausgang dieses Blocks an.

Weitere nützliche Zusatzfunktionen bieten zwei LEDs, welche am Roboter montiert sind. Nach dem Einschalten des Roboters blinkt eine grüne LED im Sekundentakt des zeitgebenden Elements. Ist die Uhrsynchronisation abgeschlossen, blinkt diese für 1,5 sec mit hoher Frequenz und anschließend im Sekundentakt mit der synchronisierten Größe der Millisekunden seit Mitternacht. Um aus dieser double-Variablen das gewünschte Blinken zu



erzeugen, wird diese zunächst durch 1000 geteilt, um einen Wert in Sekunden zu erhalten. Mit der darauf angewendeten Modulo-Funktion wird der Ausgang auf true oder false geschaltet. Dadurch ergibt sich ein Blinken im Sekudentakt.

Neben der grünen ist noch eine rote LED angebracht. Diese dient als Unterstützung während der Fehlersuche und Inbetriebnahme. Sie leuchtet auf, sobald der Roboter die Zahl Null in den drei für ihn geltenden Positionsangaben des UDP-Pakets der Kameradaten empfängt. Somit lässt sich zum Beispiel auf weite Distanz feststellen, dass der Roboter in diesem Moment nicht erkannt wird und aus diesem Grund eine nicht erwartete Bewegung ausführt. Als weitere Funktion blinkt die LED im SOS-Takt, wenn die Position des Roboters 100 mal in Folge nicht ermittelt werden konnte. Damit lässt sich leicht erkennen, ob der Roboter an einer Postion steht, der von den Kameras nicht erfasst wird.

Angesteuert werden die beiden LEDs über die digitalen Ausgänge des Roboters. Dabei handelt es sich um eine IO-Expander Platine, welche per RS232-Schnittstelle mit dem PC104-Board verbunden ist. Um diese IOs anzusprechen, muss das RS232-Protokoll zwischen den beiden Komponenten angepasst werden. Tabelle 2.4 aus der Diplomarbeit über die Roboter [Köh10, Seite: 22] zeigt die Bits an, mit welchen sich die digitalen Ausgänge ansteuern lassen: Byte 4 Bit 1 und 2. Wichtig ist dabei, die Befehle zum Ansteuern mit einer bitweisen Oder-Verknüpfung durchzuführen, damit die anderen Bits in diesem Byte unverändert ihren Zustand behalten.

3.4 Auswertung

Die Auswertung, wie erfolgreich die Synchronisation ist, gestaltete sich als schwierig. Lediglich geringe Datenmengen können auf dem PC104-Board gespeichert werden um diese nachträglich auszuwerten. Aus diesem Grund wurden für die Aufnahme der Messwerte Programme in Simulink und Qt geschrieben, welche es ermöglichen, im zyklischen Abstand von einer Sekunde, Daten per UDP von den Robotern an einen gewünschten Rechner zu senden. Auf diesem wurden die Messwerte automatisch in ein für Matlab gängiges Format in eine Textdatei geschrieben. Die folgenden Ergebnisse stammen aus diesen beiden Möglichkeiten der Messwertaufnahme.

3.4.1 Genauigkeitstest der Uhr auf dem Roboter

Nach und nach stellte sich bei Experimenten heraus, dass der CPU-Takt des PC104-Boards keinen guten Uhrquarz darstellt. Es erhärtete sich der Verdacht, dass eine einmal synchronisierte Uhr nach kurzer Zeit schon stark von der Master-Uhr abweicht. Zum Nachweis dieses Problems wurde der folgende Gedanke weiter geführt: Eine Armbanduhr, die am Montag 5 Minuten vor geht, geht auch am Dienstag noch 5 Minuten vor. Übertragen auf den Roboter bedeutet das, nachdem die Uhr einmal synchronisiert wurde, müsste der berechnete Offset-Wert annähernd konstant bleiben.

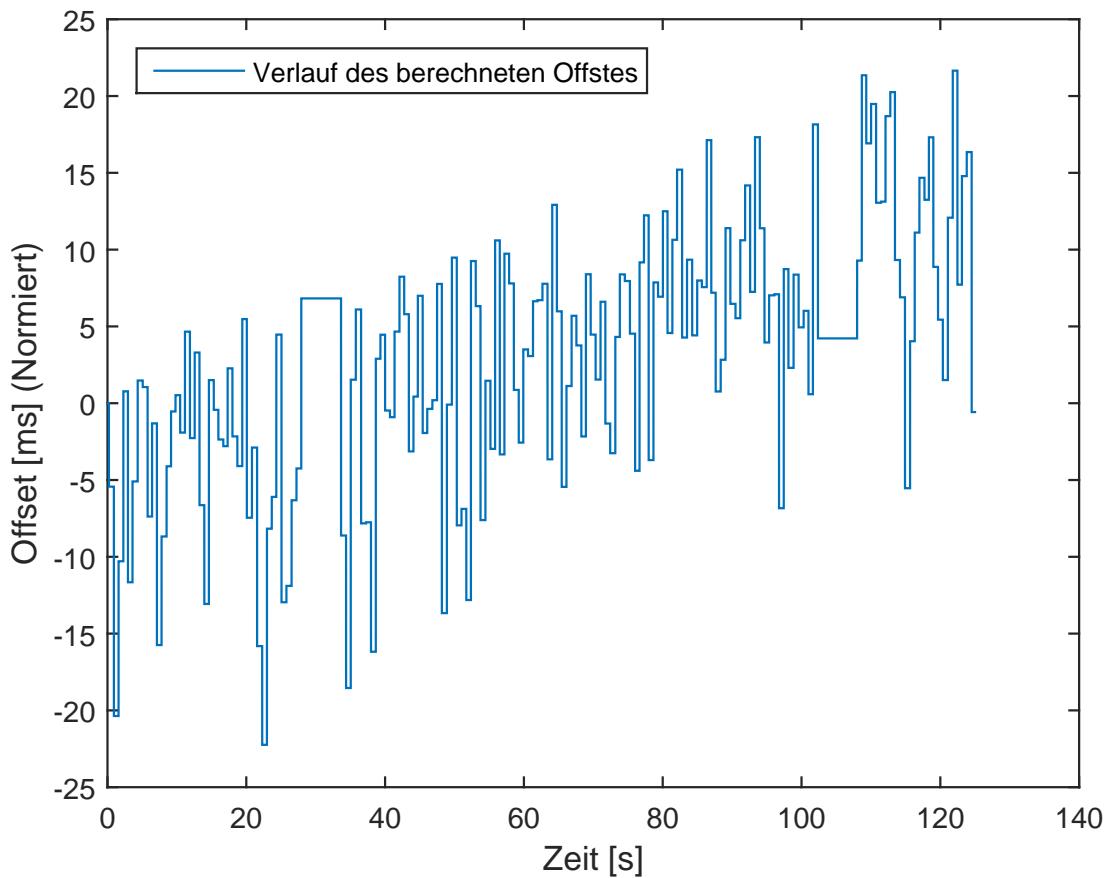


Abbildung 33: Verlauf des Uhren-Offsets bei einer einmal gestellten Uhr*

* Der Offsetwert entspricht der Differenz zwischen der Roboter-Uhr und der PC-Uhr, wobei als Größe der Wert der Millisekunden seit Mitternacht verwendet wird. Außer wenige Millisekunden nach Mitternacht ergeben sich daher Werte im neunstelligen Bereich. Aus diesem Grund wurde der Offset-Wert, zur besseren Übersicht, auf Null normiert. Nicht der exakte Wert, sondern dessen Verlauf ist an dieser Stelle von Interesse.

Für diese Messreihe wird alle fünf Millisekunden der im Roboter berechnete Offsetwert gespeichert. Dabei ist deutlich zu sehen, dass bereits nach 120 Sekunden, eine Abweichung von 10-15 Millisekunden zu erkennen ist. In Abbildung 33 ist außerdem gut die Funktionsweise der implementierten Watchdog-Funktion zu erkennen. Zwischen 20 und 40 sowie 100 und 120 Sekunden kommt es jeweils einmal zu einem Telegrammverlust, was dazu führt, dass der Synchronisierungsvorgang an den Stellen nicht beendet werden kann. Anstattdessen greift nach fünf Sekunden der Watchdog ein und setzt den Vorgang zurück. Unmittelbar danach erfolgt wieder die nächste Berechnung.

3.4.2 Test des Moving Average Filters

Auf Basis der in Kapitel 3.4.1 vorgestellten Messergebnisse wurde festgelegt, dass die Synchronisation stetig in Abständen unter einer Sekunde erfolgen muss. Um dabei Sprünge in den berechneten Offset-Werten zu glätten, wurde ein Moving Average Filter implementiert. Dessen Funktion wird mit der folgenden Messreihe nachgewiesen. Dazu wird im Roboter

nach Formel 5 das Alter der letzten eingetroffenen Nachricht auf drei verschiedene Arten berechnet und graphisch dargestellt:

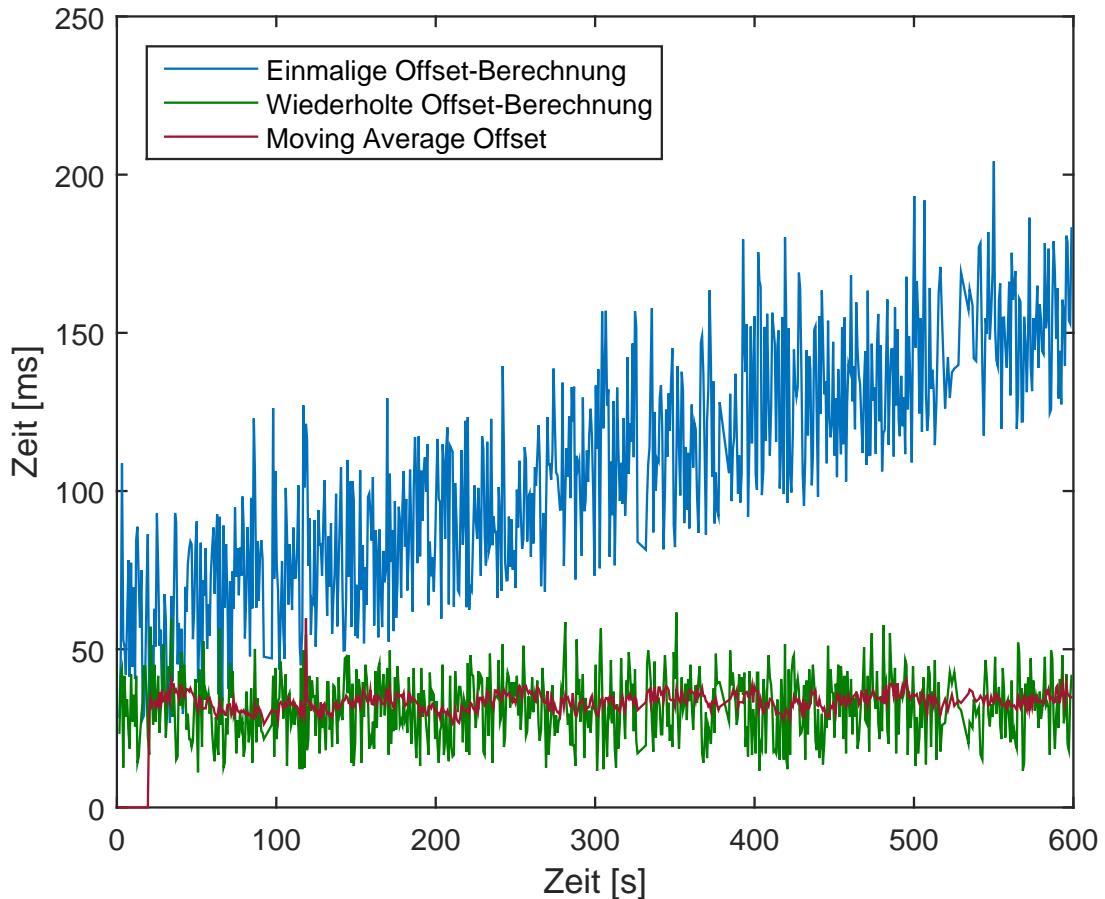


Abbildung 34: Verlauf des Alters der letzten Nachricht nach drei verschiedenen Ansätzen

Für diesen Test wurde eine modifizierte Variante des Robot-Detection Programms verwendet. Dieses sendet alle 10ms Daten an die Roboter. Im ersten Fall, dem blauen Zeitverlauf, wurde die Uhr nur einmal gestellt. Dabei zeigt sich, wie schon in Abbildung 33, dass sich mit zunehmender Zeit ein Drift abzeichnet. Eigentlich ist zu erwarten, dass das Alter der letzten Nachricht annähernd konstant bleibt, da die Bearbeitungszeit des Kamera-PCs und die Latenzzeit erwartungsgemäß nicht kontinuierlich zunehmen. Als erste Optimierung, dargestellt als grüne Linie, wurde der Offset mit jedem erfolgreich abgeschlossenen Synchronisationsvorgang neu berechnet. Dabei zeigt sich bereits schon das erwartete Ergebnis, das berechnete Alter der letzten Nachricht bewegt sich nur noch in einem kleinen Wertebereich und steigt nicht an wie im Fall der einmal gestellten Uhr. Da allerdings durch die jedes mal leicht variierende Latenz der berechnete Offsetwert schwankt, wird das in Kapitel 3.3.2 beschriebene Moving Average Filter umgesetzt. Dessen Ausgangssignal ist in rot dargestellt. Nachdem genügend Synchronisationsvorgänge abgeschlossen sind, um zum ersten Mal einen Wert auszugeben, lässt sich gut erkennen, dass es sich um den Mittelwert des ständig berechneten Signals handelt. Im zeitlichen Verlauf wird bei Sekunde 90 und 330 deutlich, dass im Signal des Moving Average Filters immer noch die Information des Alters der letzten Nachricht unverfälscht enthalten ist. Diese markanten

Verläufe existieren ebenfalls im Signalverlauf der einmal gestellten Uhr und müssen somit durch eine Anomalie im Sendevorgang der Daten entstanden sein.

3.4.3 Verlauf der Uhr auf dem Roboter bei zeitweisem Ausfall der Synchronisation

Ebenfalls gilt es zu testen, wie der Verlauf der Uhr auf dem Roboter aussieht, nachdem diese über einen längeren Zeitraum nicht synchronisiert werden kann. Dabei ist zu erwarten, dass der bekannte Drift stattfindet. Allerdings muss sich die Uhr wieder stellen, nachdem eine erneute Synchronisierung erfolgt. Zum Testen dieses Verhaltens wird das Telegramm der ID4 im Roboter nicht verarbeitet.

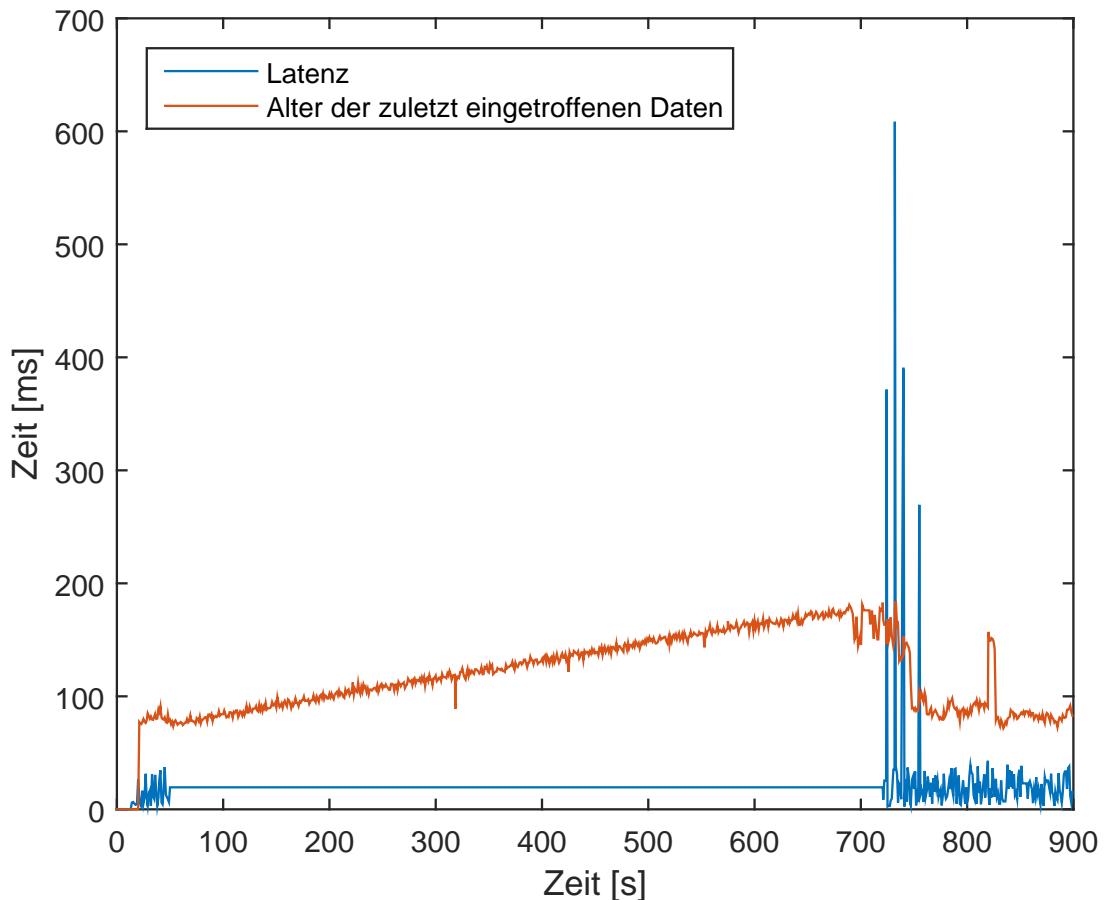


Abbildung 35: Verlauf des Alters der letzten Nachricht, wenn Synchronisierung aussetzt

Dargestellt ist in blau die ermittelte Netzlatenz und in rot das Alter der letzten Nachricht, beide Größen werden sekündlich an den Kamera-PC gesendet und dort abgespeichert. Der Kamera-PC sendet ca. alle 60 ms Kameradaten an die Roboter. Nachdem der im Test betrachtete Roboter eingeschaltet ist, wird zehn Sekunden später die Robot-Detection Software gestartet. Anschließend ist nach 12 Sekunden die Synchronisierung abgeschlossen. Bis Sekunde 50 erfolgt regelmäßig der Synchronisationsvorgang und das Alter der letzten Nachricht wird plausibel ausgegeben. Plausibel deswegen, weil das Alter der letzten Nachricht ca. 80 ms beträgt und dies der Summe aus der Bearbeitungszeit des Kamera-PCs (ca. 60 ms) und der Latenz (ca. 20 ms) entspricht. Nach Sekunde 50 wird die Synchronisation

für 11 Minuten unterdrückt. Für diesen Zeitraum zeigt sich der Anstieg des Alters der letzten Nachricht, da der erwartete Drift der Uhrzeit des Roboters einsetzt. Dass keine Synchronisierung erfolgt, ist auch an der Latenz zu erkennen, welche über diesen Zeitraum den zuletzt berechneten Wert beibehält. Bei Sekunde 720 erfolgt zum ersten Mal wieder ein erfolgreicher Synchronisationsvorgang. Im Anschluss ist zu sehen, wie sich das Alter der letzten Nachricht über einen Zeitraum von 20 Sekunden wieder an den plausiblen Wertebereich annähert und weiterhin zuverlässig bestimmt wird. Die benötigte Zeit zum Erreichen dieses Bereichs ist damit zu begründen, dass beim erneuten Kommunikationsstart Latenzspitzen entstehen, welche zunächst für ein erhöhtes Alter der letzten Nachricht sorgen.

3.4.4 Test des Alters der letzten Nachricht bei schwankender Bearbeitungszeit des PCs

Als abschließender Test wird die gestellte Uhr auf ihre Fähigkeit hin getestet, das Alter der letzten Nachricht auch bei schwankenden Bearbeitungszeiten des PCs zuverlässig und plausibel zu bestimmen. Für diese Messung wird die Zykluszeit des Robot-Detection Programms künstlich mit Hilfe von Dummy-Debug-Ausgaben zwischen 40 ms und 20 ms variiert:

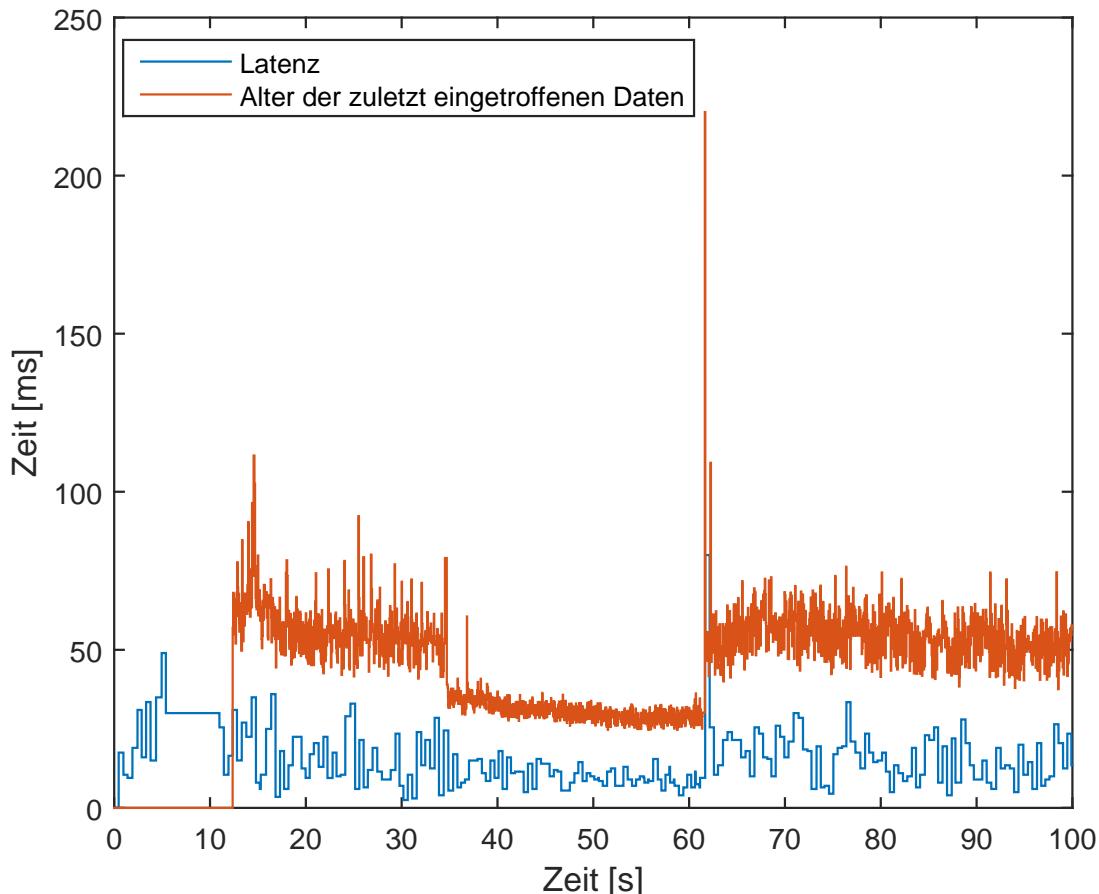


Abbildung 36: Verlauf des Alters der letzten Nachricht, wenn sich die Bearbeitungszeit des Kamera-PCs ändert

Die Messwerte werden alle 5 ms auf dem Roboter gespeichert und mit Matlab geplottet. In rot ist das Alter der letzten Nachricht dargestellt und in blau die Latenzzeit. Auf den ersten Blick fällt auf, dass die Anzahl der blauen Messwerte deutlich geringer ist als die der roten. Das ist mit der in Kapitel 3.2.2 erklärten Random-Wartezeit zu begründen. Das Alter der letzten Nachricht wird hingegen bei jedem Eintreffen neuer Daten vom Kamera-PC berechnet. Nachdem bei Sekunde 12 die Synchronisierung abgeschlossen ist, wird das Alter der letzten Nachricht mit einem Wert von ca. 55 ms angegeben. Bei einer Bearbeitungszeit von ca. 40 ms und einer Latenzzeit von ca. 15 ms ist dieser Wert, welcher sich aus der Summe der beiden Zeiten bildet, plausibel. Ab Sekunde 35 benötigt das Robot-Detection Programm nur noch 20 ms Bearbeitungszeit. Unmittelbar nach dem Sprung stellt sich wieder ein plausibler Wert für das Alter der letzten Nachricht ein. Beim erneuten Umschalten der Bearbeitungszeit, bei Sekunde 63, zeigt sich wie schon in Abbildung 35 eine Latenzspitze. Diese sorgt dafür, dass zu diesem Zeitpunkt UDP-Pakete mit einer großen Verzögerung von 100 ms bis 220 ms bei dem Roboter eintreffen. Nach diesen hohen Werten wird das Alter der letzten Nachricht im weiteren Verlauf wieder zuverlässig bestimmt.



4 Fazit

Für das Verbundprojekt konnte erfolgreich ein Erkennungsverfahren für Marker in einer bestehenden Software ersetzt werden. In der Vorbetrachtung war der Erfolg des Systems unter den gestellten Anforderungen nicht eindeutig feststellbar, weshalb eine praktische Umsetzung notwendig war. Die Vorbetrachtung ergab ein Übersetzungsverhältnis zwischen Pixel und metrischem System von $1 \text{ px} \hat{=} 4.3 \text{ mm}$ auf Höhe der Roboter. Die Länge der Markeranordnung ist möglichst groß zu wählen, um eine ausreichend genaue Winkelbestimmung zu erreichen. Bezüglich der Anordnung der Marker, wurde sich zugunsten einer zweifachen Anordnung je Roboter in Richtung der Roboterachse entschieden, da diese gegenüber einer Anordnung mit drei Markern stabiler und sicherer ist. Die Aruco Bibliothek findet entgegen der empfohlenen Markerbibliothek Verwendung, da diese bereits in der Tabletapplikation enthalten ist und somit eine einfache Schnittstelle zwischen Datenbank und Roboter bildet. Zur Bestimmung der Roboterpositionen ist die Mittelwertbildung der beiden Marker hinreichend genau und ist gemeinsam mit der Winkelberechnung implementiert. Diese verwendet unter idealen Bedingungen vier Winkel je Marker sowie zwei Winkel zwischen den Markern. Zur Steigerung der Stabilität wird die Position des Roboters während des Ausfalls eines Markers durch einen Offset errechnet, der durch eine implementierte Kalibrierung erfasst werden kann. Zur Vermeidung von Mehrfach- bzw. Fehlerkennungen in den Überlagerungsbereichen der Kameras werden Roboter, die sich innerhalb eines Abstandes von 500 mm befinden, aussortiert. Sofern die Bilder aller sechs Kameras ungefiltert an den ARUCO Algorithmus übergeben werden, reduziert sich die Wiederholrate der Anwendung auf zehn bis 15 Frames/Sekunde. Zur Steigerung der Rate wurde daher zusätzlich eine Reduzierung der Bilder auf Bereiche mit Robotern umgesetzt, mit der die Anwendung auf 20 bis 26 Frames/Sekunde verbessert wird. Aufgrund des notwendigen Thresholdings in der Vorverarbeitung kann diese Funktion in dem ARUCO Algorithmus vernachlässigt werden. Die Gegenüberstellung mit dem Kreis basierten System zeigt nach der Vorverarbeitung Einbußen von vier bis zehn Frames/Sekunde in dem Marker basierenden System auf, kann diese jedoch durch eine deutlich bessere Ausfallsicherheit in der Positionserkennung kompensieren. Weiterhin ist das Marker basierende System unter eingeschalteter Raumbeleuchtung zwar fehleranfälliger in der Erkennung der Marker, gegenüber dem Kreis basiertem System jedoch deutlich stabiler in der Positionserkennung. Das einschalten der Raumbeleuchtung hat in beiden Systemen eine Reduzierung der Framerate zur Ursache, die sich in dem Marker basierenden System weniger stark auf die Framerate auswirkt. An dieser Stelle weist der Kreiserkennungsalgorithmus Schwierigkeiten bei der korrekten Zuordnung zwischen Kreisanzahl und der Roboter ID auf, da eine Vielzahl an Kreisen detektiert wird. Für Robotergeschwindigkeiten von $350 \frac{\text{mm}}{\text{s}}$ ist die Framerate des Marker basierenden Systems, trotz Einbußen hinreichend hoch. Dies bestätigen Kreisfahrten durch den Raum und den Überlagerungsbereichen der Kamera, sowie wiederholte Anfahrten der Stationen mit einem Werkstück. In beiden Messungen werden die Zielpositionen ohne Ausfall eines Werkstückes oder Auslösen der Kollisionsleiste erreicht. Somit konnte erfolgreich ein stabiles Erkennungsverfahren innerhalb des geforderten Projektzeitraumes in eine bestehende Software implementiert werden, obwohl keine bzw. geringe Erfahrungen in der Bildverarbeitung oder der C++ Programmierung vorhanden waren. Während der Durchführung konnten somit wertvolle Kenntnisse in der Bildverarbeitung und der Programmierung erworben werden, die in Zukunft immer höheren Stellenwert in der Industrie und dem Consumer Bereich haben.

Um die Uhrsynch�ronisation erfolgreich umzusetzen, wurde zunächst eine passende zeitgebende Funktion auf den Robotern implementiert und im Anschluss ein passendes Verfahren



auf Basis des IEEE 1588 Standards gesucht, wobei drei Lösungsansätze miteinander verglichen wurden. Dabei stellte sich die gewählte Methode als besonders geeignet heraus. Ohne Abstriche oder Einbußen eingehen zu müssen, konnte dieser Ansatz erfolgreich implementiert werden. Zusätzlich wurde mit Hilfe des Moving Average Filters eine Möglichkeit gefunden, den Uhren-Drift zu korrigieren. Die Funktionstüchtigkeit wurde mit verschiedenen Tests nachgewiesen. Am anschaulichsten war dabei die praktische Vorstellung, bei der die grünen LEDs von mehreren Robotern nach der Synchronisation im Takt blinkten. Für die späteren Gruppen sind dabei sicherlich die Tests der Berechnung des Alters der letzten eingetroffenen Kameradaten wichtiger. Dazu wurde erfolgreich nachgewiesen, dass auch nach einer längeren Zeit ohne Synchronisation, diese wieder automatisch startet und nach kürzester Zeit zu plausiblen Ergebnissen führt. Ein anderer Test zum Nachweis der Funktionstüchtigkeit war das Umschalten zwischen den Bearbeitungszeiten der Robot Detection Software. Dabei folgt das berechnete Alter der zuletzt eingetroffenen Kameradaten sofort. Einer der wichtigsten Punkte, die im Verlauf der Arbeit auffielen, sind die Latenzspitzen bei schwankender Netzlast. Daraus resultieren später eintreffende Kameradaten bei den Robotern. Dies spiegelt genau den Aufgabenbreich wider, bei dem ein Kalman-Filter zu einer optimalen Benutzung von Kamera- oder Beobachterdaten führt. Abschließend wird ein ausführlich dokumentiertes und kommentiertes Softwarepaket abgegeben, welches alle daran gestellten Anforderungen erfüllt. Besonders interessant war dabei die nahe Arbeit an der Netzwerktechnik, wobei in der Testphase mit Hilfe von Analyse-Tools (Wireshark, Advanced IP Scanner, SmartSniff) teilweise jedes Paket einzeln nachvollzogen werden musste. Aus Sicht des Lernerfolgs, war es wertvoll, auf zwei unterschiedlichen System zu arbeiten, somit Qt neu kennen lernen zu können und gleichzeitig den Umgang mit Simulink deutlich zu verbessern.

5 Anhang

Variable (double)	Funktion
1	Roboter 1 - X-Position
2	Roboter 1 - Y-Position
3	Roboter 1 - Winkel
4	Roboter 2 - X-Position
5	Roboter 2 - Y-Position
6	Roboter 2 - Winkel
7	Roboter 3 - X-Position
8	Roboter 3 - Y-Position
9	Roboter 3 - Winkel
10	Roboter 4 - X-Position
11	Roboter 4 - Y-Position
12	Roboter 4 - Winkel
13	Roboter 5 - X-Position
14	Roboter 5 - Y-Position
15	Roboter 5 - Winkel
16	Roboter 6 - X-Position
17	Roboter 6 - Y-Position
28	Roboter 6 - Winkel
19	Roboter 7 - X-Position
20	Roboter 7 - Y-Position
21	Roboter 7 - Winkel
22	Roboter 8 - X-Position
23	Roboter 8 - Y-Position
24	Roboter 8 - Winkel
25	Stunde (des Kamera PCs)
26	Minute (des Kamera PCs)
27	Sekunde (des Kamera PCs)
28	Zeitstempel der Bildaufnahme
29	**Reserve**
30	**Reserve**
31	ID für Roboter 1
32	ID für Roboter 2
33	ID für Roboter 3
34	ID für Roboter 4
35	ID für Roboter 5
36	ID für Roboter 6
37	ID für Roboter 7
38	ID für Roboter 8
39	**Reserve**
40	**Reserve**

Variable (double)	Funktion
40	**Reserve**
41	**Reserve**
42	**Reserve**
43	**Reserve**
44	**Reserve**
45	**Reserve**
46	**Reserve**
47	**Reserve**
48	**Reserve**
49	**Reserve**
50	**Reserve**
51	**Reserve**
52	**Reserve**
53	**Reserve**
54	**Reserve**
55	**Reserve**
56	Zeitstempel - Absendezeitpunkt (t_1)

Tabelle 11: Telegrammaufbau der Kameradaten



References

- [AAH97] ALLAN, David W. ; ASHBY, Neil ; HODGE, Clifford C.: The Science of Timekeeping / Hewlett Packard. 1997. – Forschungsbericht
- [BBT14] BADEN, Markus ; BÄÖESSIS, Fabian ; TODZY, Steffen: Gewerk 5 à Bildverarbeitung / Hochschule fÄEr Angewandte Wissenschaften Hamburg. 2014. – Forschungsbericht
- [Cora] CORPORATION, The M.: *IEEE 1588 Precision Time Protocol - Execution Synchronization.* <http://de.mathworks.com/help/xpc/examples/ieee-1588-precision-time-protocol-execution-synchronization.html>
- [Corb] CORPORATION, The M.: *Precision Time Protocol.* <http://de.mathworks.com/help/xpc/precision-time-protocol.html>
- [DM] DREHER, Andreas ; MOHL, Dirk: Präzise Uhrzeitsynchronisation - IEEE 1588 / Hirschmann Automation and Control GmbH. -. – Forschungsbericht
- [Gar] GARRIDO, Sergio: *Detection of ArUco Markers.* http://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html#gsc.tab=0
- [GJMSMCRC14] GARRIDO-JURADO, S. ; MUÑOZ-SALINAS, R. ; MADRID-CUEVAS, F.J ; R.MEDINA-CARNICER: Generation of fiducial marker dictionaries using Mixed Integer Linear Programming / Department of Computing and Numerical Analysis. University of Cordoba. University of Cordoba. 14071 Cordoba (Spain), 2014. – Forschungsbericht
- [Köh10] KÖHLER, Elard: Entwicklung einer kamera-basierten Bahnführungs-Regelung fÄEr einen omnidirektionalen Roboter mit Matlab xPC Target. / Hochschule für Angewandte Wissenschaften Hamburg. 2010. – Forschungsbericht
- [Rün11] RÜNZ, Martin: Augmented Reality Software : Proseminar Augmented Reality in der Anwendung / Institut fÄEr Computervisualistik, Universität Koblenz. 2011. – Forschungsbericht
- [tea] TEAM, Opencv dev: *Basic Thresholding Operations.* <http://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html>
- [Wei04] WEIBEL, Hans: Zeitsynchronisation im Netzwerk / Zuercher Hochschule Winterthur (ZHW). 2004. – Forschungsbericht