Super computing systems

Sugus Sortierer

Systemdesign mit Low Cost Vision Komponenten und EtherCAT

Kunde	
Projekt	EtherCAT
Projektleiter	
Autor(en)	Philipp Stadelmann
Rezensent(en)	

p:\akquisition\sugus sort ethercat\sugus_sortierer\dokumentatic

Phone +41 (0)43 456 16 00 · Fax +41 (0)43 456 16 10 · www.scs.ch

Supercomputing Systems AG · Technoparkstrasse 1 · CH-8005 Zürich

Verteiler

Name	Firma

Versionen

Version	Datum	Autor(en)	Änderungen

Inhalt

Literaturverzeichnis

[1] Beckhoff: Hardware Data Sheet ET1100 EtherCAT Slave Controller, Version 1.3, 28.1.2008

[2] CVS Directory, :pserver:*username*@intra.scs.ch:6707/cvs/akquisition, Repository: DemoSorter

1 Einleitung

1.1 Zweck

1.1.1 EtherCAT

In der Automatisierungstechnik nimmt die Verwendung von Ethernet als physikalische Kommunikationsschicht zu um vermehrt auf günstige Standardkomponenten zurückgreifen zu können. Ethernet ist allerdings nicht direkt geeignet für die Feldbuskommunikation:

- Wenn viele Teilnehmer nur wenige Daten übertragen, entsteht ein grosser overhead.
- Die im Vollduplex-Modus verwendete Sternentopologie führt zu hohen Verkabelungskosten.

Um den Entwicklungsaufwand von EtherCAT besser abschätzen zu können, soll eine Testanwendung implementiert werden.

1.1.2 Low-Cost-Vision Kamera

Gleichzeitig besteht der Bedarf eines Vorführsystems für die Low-Cost-Vision Kamera. Es drängt sich deshalb auf diese beiden Ziele zu vereinen.

1.2 Sugus Sortierer

Der Sugus Sortierer 2004 kann als einfaches Beispiel einer Maschinensteuerung betrachtet werden: Eine Bildverarbeitungsplattform berechnet aus aufgenommenen Bildern die Positionen der einzelnen Sugus. Sie steuert über MODBUS im richtigen Moment Ventile an um die Sugus auszusortieren.

Basierend auf dem Sugus Sortierer 2004 soll ein neues System entworfen werden. Anstelle der damals verwendeten High-End Kamera und der FlexiVision Plattform soll die neue Low-Cost-Vision Kamera der SCS verwendet werden. Als Kommunikationsbus zwischen Kamera und Ventile soll EtherCAT zum Einsatz kommen.

2 System Übersicht

2.1 Sugus Sortierer 2004

Abbildung 1 zeigt die Sugus Sortiermaschine. Die Sugusse werden auf einem Förderband herangeführt. Sie fallen dann im freien Fall zuerst an der Kamera und anschliessend an den Ventilen vorbei.

Grundsystem

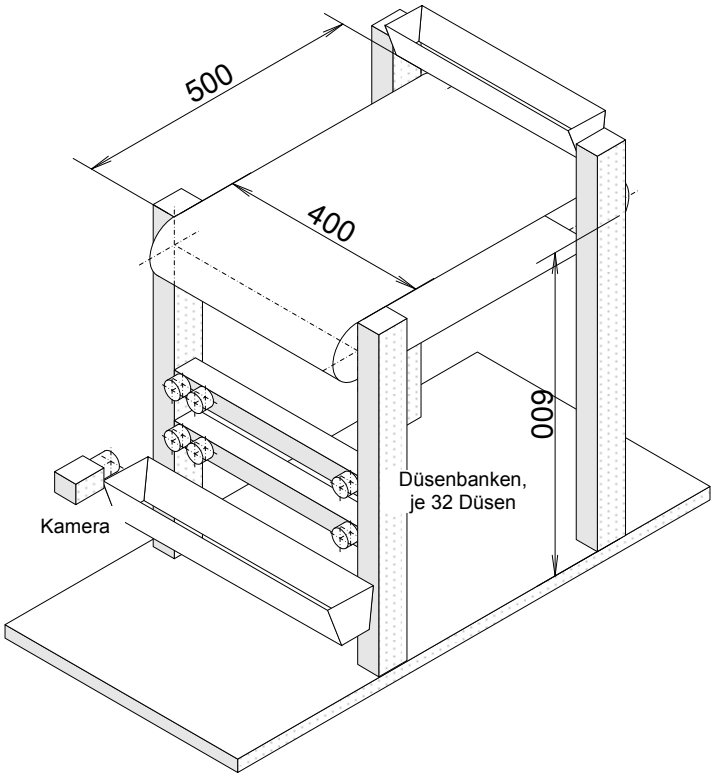


Abbildung 1 – Fördersystem mit Auffangbecken

Abbildung 2 zeigt die Seitenansicht auf die Sortiermaschine. Die Abstandsangaben sind in Millimeter.

Seitenansicht Auffangbecken

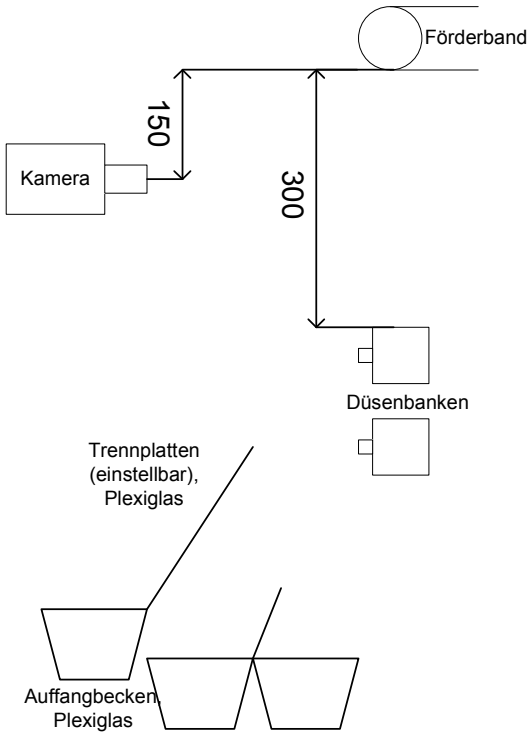


Abbildung 2 – Fördersystem mit Auffangbecken

2.2 Sugus Sortierer 2008

Abbildung 3 zeigt ein Blockschaltbild des Gesamtsystems. Folgende Funktionen werden den einzelnen Blöcken zugeordnet.

2.2.1 EtherCAT Master

Der EtherCAT Master initiiert die Datentransfers. Er schickt periodisch Pakete ins Netzwerk um die Kommunikation zwischen den Slaves zu ermöglichen. Die Slaves können nicht von sich aus einen Datentransfer starten. Sie haben lediglich die Funktionalität von empfangenen Paketen Daten zu lesen, in diese Daten zu schreiben und sie weiterzuleiten.

Zusätzlich läuft auf dem Master das GUI zum Sugus Sortierer, welches die Statistik über die sortierten Sugus anzeigt.

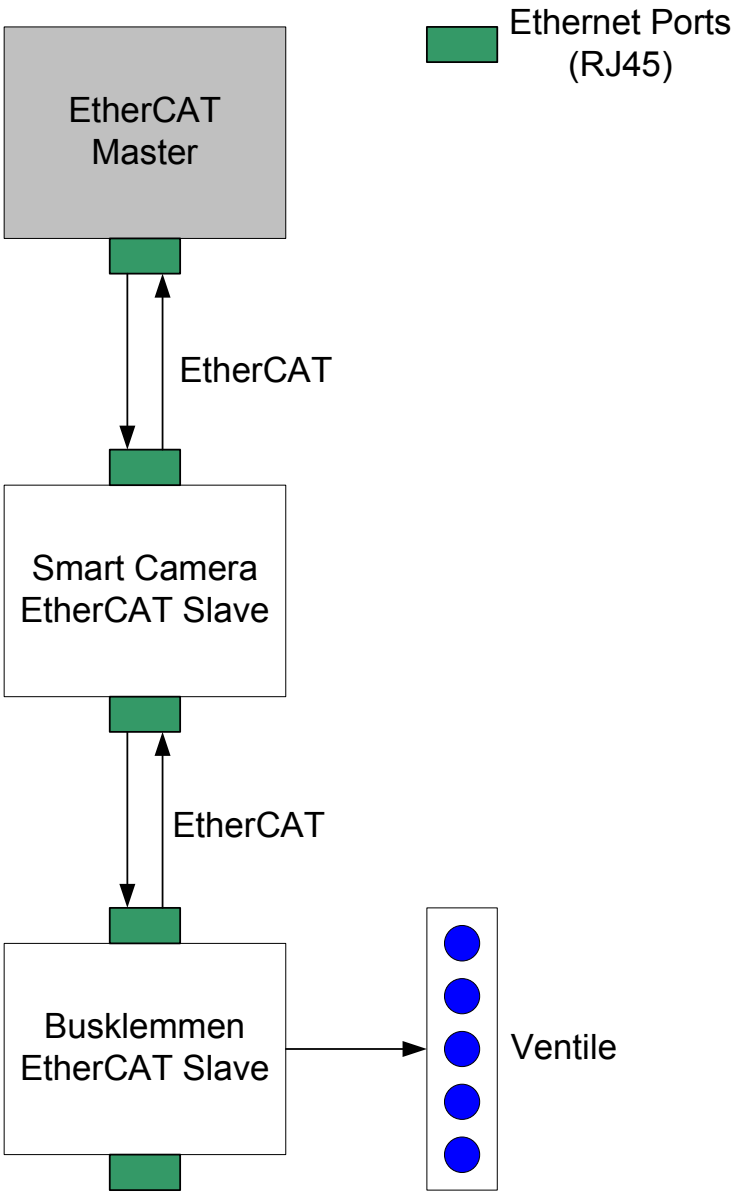


Abbildung 3 – Systemaufbau

2.2.2 Smart Camera

Die Kamera nimmt Bilder der vorbei fliegenden Sugus auf und verarbeitet diese. Sie erkennt an welchen Positionen im Bild welche Sorten Sugus sich befinden. Daraus berechnet sie den Zeitpunkt zu welchem die Ventile geöffnet werden müssen um die Sugus zu sortieren. Im richtigen Moment werden die Signale dann ins EtherCAT Paket geschrieben.

Die Kamera verwaltet ausserdem die Statistik über die sortierten Sugus. Diese Daten schreibt Sie ebenfalls in ein EtherCAT Paket, damit der Master die Anzeigen im GUI aktualisieren kann.

2.2.3 Busklemmen

Die Busklemmen haben keine eigene Kontrolllogik. Die digitalen Ausgänge sind entweder „high“ oder „low“, je nachdem mit welchem Wert sie zuletzt gesetzt wurden. Aus den empfangenen EtherCAT Paketen können Sie den aktuellen Zustand lesen.

2.3 Dimensionen

Die mechanischen Komponenten des Demosystems 2004 können übernommen werden:

- Förderband: 22cm breit
- Ventile: 2 x 8, Abstand: 1.4cm

3 Kamera

3.1 Framerate

Jedes Sugus soll genau in einem Bild als gültiges Objekt erkannt werden. Dadurch vereinfacht sich der Bildverarbeitungsalgorithmus, indem kein zusätzliches tracking der Objekte gemacht werden muss. Damit wird die Framerate massgeblich durch die Zeit bestimmt, die ein Sugus benötigt um durch das Sichtfeld der Kamera durchzufliegen. Folgende Parameter beeinflussen die Framerate:

- Abstand der Kamera vom Förderband
- Grösse des Sichtfeldes der Kamera

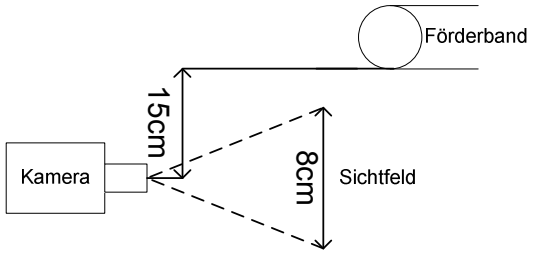


Abbildung 4 – Darstellung der Parameter, welche die Framerate bestimmen

Tabelle 1 zeigt das Parameterset.

Abstand Kamera [cm]	Sichtfeld [cm]	Zeit im Sichtfeld [ms]	Frame Rate [1/s]
15	8	47.1	22

Tabelle 1 – Framerate

3.2 Shutter

Der Shutter der Kamera muss so eingestellt sein, dass die entstehende Unschärfe im Vergleich zur Sugusgrösse verkraftbar ist. Die Dimensionen eines Sugus sind etwa 2.5 x 1.5 x 1 cm^3 im Kern. Dazu kommen noch die beiden Schleifen. In Tabelle 2 wird die Unschärfe anhand der durchschnittlichen Geschwindigkeit berechnet.

Geschw. [m/s]	shutter [ms]	Unschärfe [mm]
1.7	1	1.72

Tabelle 2 – Shutter

3.3 Speicherverwaltung

Die Kameradaten werden mittels DMA Transfer in den Hauptspeicher des DSPs geschrieben. Ein Double-Buffer Prinzip wird angewandt, damit zeitgleich auf einem Bild gerechnet werden kann, während das andere transferiert wird.

3.4 Latenz

Die Kamera hat eine Framerate von 60fps. Diese ist im Wesentlichen gegeben durch den Pixeltakt, mit dem die Kamera die Bilddaten ausgibt. Um ein komplettes Bild (752 x 480) auszugeben braucht sie also 16.7ms. Das breite Förderband bedingt aber ein speziell breites Sichtfeld. Die Sugusse sollten in einem Bild von etwa 22 x 8 cm^2 erfasst werden. Somit werden nur etwa 270 Bildzeilen benötigt womit die Latenz auf unter 10ms sinkt.

4 Bildverarbeitungs-Algorithmus

4.1 Überblick

Abbildung 5 zeigt den Programmablauf des Bildverarbeitungsalgorithmus. Die einzelnen Funktionen werden in den folgenden Abschnitten erklärt.

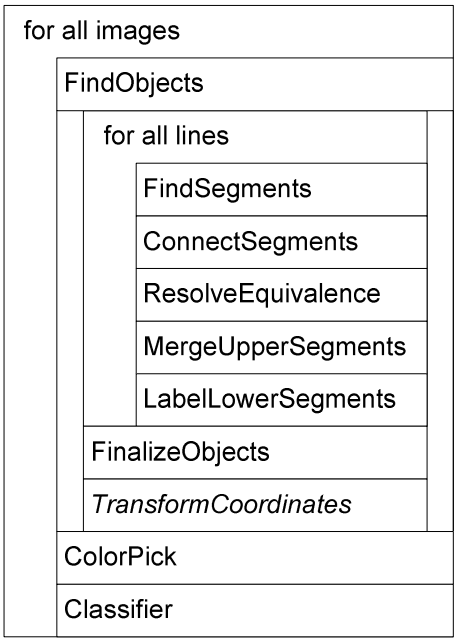


Abbildung 5 – Programmablauf des Bildverarbeitungsalgorithmus

4.2 Begriffserklärung

Objekt Definiert ein Sugus. Ein Objekt besteht im Wesentlichen aus dem Schwerpunkt in x/y Koordinaten, der Fläche und den vier seitlichen Begrenzungen: oben, unten, links und rechts.

Segment Zusammenhängende Strecke von Pixel einer Bildzeile, welche einen Teil eines Sugus repräsentiert.

Aktuelle Zeile Zeile, die in diesem loop-Durchgang bearbeitet wird.

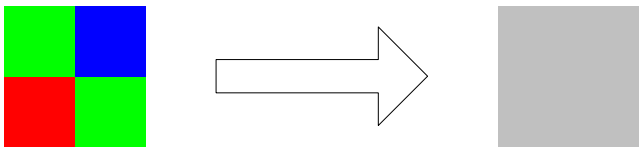
Letzte Zeile Zeile, die im vorherigen loop-Durchgang bearbeitet wurde.

4.3 *FindObjects*

Die in diesem Abschnitt beschriebenen Funktionen sind ziemlich ähnlich für den Sugus Sortierer 2004 implementiert. Sie können im CVS Directory [2] im folgenden file gefunden werden:
`D:\cvsprojects\DemoSorter\DSP\Flexivision\vision\classVisFastLabel.cpp`

Die Tasks *FindSegments*, *ConnectSegments*, *ResolveEquivalence*, *MergeUpperSegments* und *LabelLowerSegments* werden in einer Schleife über alle Bildzeilen ausgeführt.

Das Eingangsbild wird als Graustufenbild interpretiert. Es werden jeweils 4 benachbarte Pixel zu einem Graustufen-Pixel zusammengefasst.



Folglich ist die Anzahl Zeilen und Spalten des Graustufenbildes nur halb so gross, wie diejenige des Originalbildes.

4.3.1 *FindSegments*

Eine Zeile des Bildes wird von links nach rechts nach zusammenhängenden Segmenten (Bruchstücke der Sugusse) durchsucht. Übersteigt der Grauwert eines Pixels einen bestimmten Threshold, wird dieses Pixel zum Startpunkt eines Segments. Alle zusammenhängenden Pixel, die ebenfalls den Threshold überschreiten, gehören zu diesem Segment. Das nächste Pixel, welches den Threshold nicht mehr überschreitet, beendet das Segment. Die Segmente werden gespeichert, indem man sich alle Startpunkte und die Anzahl Pixel der Segmente merkt (Run-Length-Encoding).

4.3.2 *ConnectSegments*

Einzelne nicht verbundene Segmente der gleichen Zeile können über Segmente benachbarter Zeilen miteinander verbunden sein.

Es wird eine Matrix aufgebaut, welche die Verbindungen der Segmente der aktuellen Zeile mit denjenigen der letzten Zeile aufzeigt. Dazu müssen alle Segmente der letzten Zeile mit denen der aktuellen Zeile verglichen werden.

Die Matrix ist quadratisch und kann maximal die Dimension 2*MaxSegments haben. Soviel Speicher muss reserviert sein. Die Elemente der Matrix stellen Verbindungen zwischen den verschiedenen Segmenten sowohl der aktuellen, als auch der letzten Zeile dar. Die Verbindungen der letzten Zeile werden an den Stellen 0..(#Segmente letzte Zeile -1), die der aktuellen Zeile an den Stellen (#Segmente letzte Zeile)..(#Segmente aktuelle Zeile - 1) gespeichert. Es wird vorerst nur ein Quadrant der Matrix ausgefüllt.

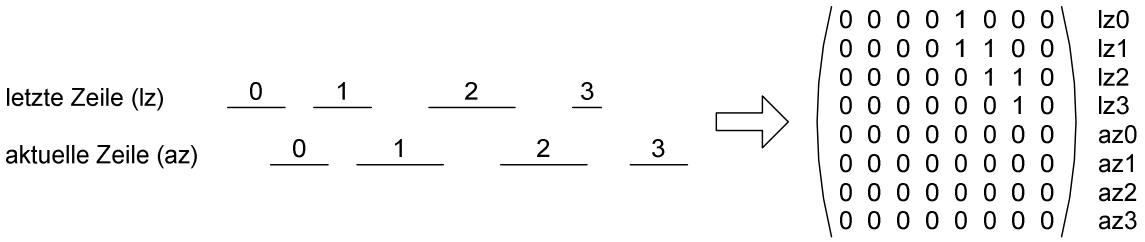


Abbildung 6 – Entstehung der Verbindungsmatrix

Für zwei Segmente benachbarter Zeilen, Segment 1 von a bis b und Segment 2 von c bis d, gelten folgende Bedingungen, dass sie nicht verbunden sind:

- a > d
- c > b

4.3.3 *ResolveEquivalence*

Die Matrix wird vervollständigt. Zuerst wird die Matrix symmetrisch gemacht und die Diagonalen werden hinzugefügt. Dann werden Verbindungen zwischen Segmenten, die über andere Segmente zustande kommen, hinzugefügt. Dies kann folgendermassen ausgeführt werden: Die Matrix wird Zeile für Zeile durchgearbeitet. Wird an der Stelle i der Zeile n eine ´1´ gefunden, bedeutet dies, dass Segment i mit Segment n verbunden ist. Folglich ist Segment i auch mit allen anderen Segmenten verbunden, die eine Bindung mit n aufweisen (alle anderen ´1´-en der Zeile n). Um diese Bindungen auch Zeile i aufzuprägen, kann deshalb die binäre „ODER“ Funktion der Zeile n mit der Zeile i in die Zeile i geschrieben werden. Die Matrix aus Abbildung 6 wird zur Matrix in Abbildung 7.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Abbildung 7 – Vervollständigen der Verbindungsmatrix

4.3.4 *MergeUpperSegments*

Die obere Dreiecksmatrix des linken oberen Quadranten der Verbindungsmatrix (siehe Abbildung 8) zeigt nun die Verbindungen auf, die aufgrund der aktuellen Zeile neu in der letzten Zeile zustande gekommen sind.

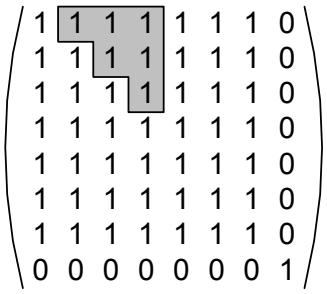


Abbildung 8 – Verbindungsmatrix: Verbindungen zwischen Segmenten der letzten Zeile

Diese Verbindungen müssen nun überprüft werden. Falls zwei Segmente, die eine Verbindung aufweisen, ungleiche Labels haben, müssen die Objekte mit den entsprechenden Labeln zusammengeführt werden. Die Flächen der Objekte werden zusammengezählt, die Koordinaten der Schwerpunkte und die Begrenzungen werden angepasst. Eines der beiden Objekte wird als ungültig markiert, damit keine Duplikate mehr vorhanden sind.

Die Label der Segmente der aktuellen Zeile werden angepasst, so dass alle Segmente der verbundenen Objekte das gleiche Label haben.

4.3.5 *LabelLowerSegments*

Der linke untere Quadrant der Verbindungsmatrix (siehe Abbildung 9) zeigt die Verbindungen der Segmente der aktuellen Zeile mit den Segmenten der letzten Zeile auf.

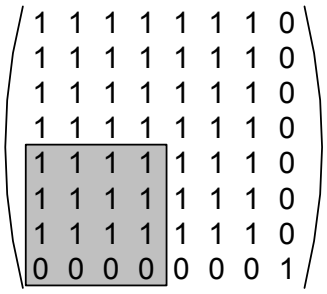


Abbildung 9 – Verbindungsmatrix: Verbindungen zwischen Segmenten der letzten Zeile und Segmenten der aktuellen Zeile

Dieses Feld muss nun Zeile für Zeile abgearbeitet werden. Wird auf einer Zeile eine ´1´ gefunden, kann das Segment der aktuellen Zeile dem Objekt mit dem entsprechenden Label hinzugefügt werden. Wird in einer Zeile keine ´1´ gefunden, besteht keine Verbindung zwischen dem Segment der aktuellen Zeile und einem bestehenden Objekt. Es muss ein neues Objekt mit eigenem Label aus dem Segment erzeugt werden.

4.3.6 *FinalizeObjects*

Nachdem alle Zeilen des Bildes abgearbeitet wurden, sind nun alle Objekte im Bild bekannt. Die Liste der Objekte wird bereinigt. Duplikate sowie zu kleine Objekte und solche, die zu Nahe an den Bildrändern sind, werden entfernt. Das Ergebnis ist eine Liste von erkannten Objekten die eindeutig ist.

4.4 *ColorPick*

Die in diesem Abschnitt beschriebenen Funktionen sind ziemlich ähnlich für den Sugus Sortierer 2004 implementiert. Sie können im CVS Directory [2] im folgenden file gefunden werden:
`D:\cvsprojects\DemoSorter\DSP\Flexivision\vision\classVisColorPick.cpp`

Jedem Objekt wird ein Farbwert (Farbton, Sättigung und Helligkeit) zugewiesen. Der Farbwert ergibt sich aus der Mittelung der RGB-Pixel eines 3x3 Fensters um den Schwerpunkt des Objekts. Zur Erzeugung eines RGB-Pixels werden wiederum 4 benachbarte Pixel genommen (1 rotes, 1 blaues und 2 grüne). Somit berechnet sich der Farbwert also aus einem 6x6 Pixel Fenster.

Bemerkung: Die Grösse des Fensters (3x3) kann eventuell dazu führen, dass zu viel weisse Schrift erkennt wird und nicht genügend Farbe vom Sugus. Dann muss das Fenster grösser gemacht werden.

4.5 *Classifier*

Die in diesem Abschnitt beschriebenen Funktionen sind ziemlich ähnlich für den Sugus Sortierer 2004 implementiert. Sie können im CVS Directory [2] im folgenden file gefunden werden: `D:\cvsprojects\DemoSorter\DSP\Flexivision\vision\classVisDemoSorterClassifier.cpp`

Aus den Farbwerten wird mittels Threshold Evaluation die Sugus-Sorte bestimmt. Die Anzahl detektierter Sugusse wird mitgezählt und für die Statistik abgespeichert.

Ausserdem wird aus dem Schwerpunkt der Objekte der Zeitpunkt für die Aktivierung der Ventile berechnet. Die seitlichen Begrenzungen der Objekte bestimmen welche Ventile aktiviert werden müssen. Dieser Datensatz wird in ein FIFO geschrieben. Ebenfalls müssen die Befehle für die Deaktivierung der Ventile in dieses FIFO geschrieben werden.

5 Ventile

5.1 Schaltzeit

Die Ventile müssen so schnell geschaltet werden können, dass aufeinander folgende Sugusse korrekt verarbeitet werden können. Bei einer Fördergeschwindigkeit von ca. 21cm/s sind aufeinander folgende Sugusse zeitlich mindestens 48.5ms verzögert. Somit sollte eine Zeitauflösung von 10ms ausreichen. Bei einer Geschwindigkeit von etwa 2.5m/s bewegt sich das Sugus in 10ms etwa 2.5cm weit. Die Zeitauflösung für die Ventile ist die kritische Anforderung für das EtherCAT Protokoll. Die EtherCAT Frames müssen mit dieser Taktrate verschickt werden können.

Die verwendeten Ventile vom Typ FESTO MHA2-MS1H-3/2G-2K haben eine Schaltzeit von 2ms und eine maximale Schaltfrequenz von 330Hz.

5.2 Ventilposition

Mit den getroffenen Annahmen in Kapitel 3 über das Sichtfeld und die Kameraposition ergibt sich mit der Ventilposition die maximale Latenz, die das System aufweisen darf. Bei einer Ventilposition von 30cm unterhalb des Förderbandes, beträgt sie 50.5ms (Abstand Austritt Sichtfeld – Ventile). Wenn die Ventile weiter unten positioniert werden, vergrössert sich die Latenz. Auch mit der Kameraposition und dem Sichtfeld kann der Parameter noch beeinflusst werden.

6 Kommunikation

6.1 EtherCAT Performance

Gemäss Beckhoff beträgt die Update-Zeit für die Daten von 1'000 verteilten Ein-/Ausgängen nur 30us. Der Transfer eines maximalen Ethernet Frames von 1486 Bytes Prozessdaten geschieht in 300us. Selbst wenn diese Angaben für Marketing Zwecke beschönigt wurden, sollten wir mit der Zeitauflösung von 10ms für die Ventilsteuerung keine Probleme kriegen.

6.2 Master

Der Master muss alle EtherCAT Transfers initiieren. Er muss also mit der gleichen Periode Pakete verschicken, wie die Ventile angesteuert werden sollen. Wenn das System zuverlässig arbeiten soll, muss auf dem Master ein Real-Time OS laufen, damit das timing eingehalten werden kann. EtherCAT bietet zudem die Möglichkeit zur Synchronisation der einzelnen Komponenten.

6.2.1 Synchronisation

Der Master synchronisiert sich mit dem Kamera Slave mittels Distributed Clocks. Diese Operation wird vom Master-Stack unterstützt. Damit wird erreicht, dass die einzelnen Operationen aufeinander abgestimmt sind: versenden von EtherCAT frames, Triggern der Kamera, usw.

6.2.2 Implementation/Inbetriebnahme

Der Master-Stack muss für das Master System eventuell angepasst werden. Eine Applikation muss geschrieben werden, die die EtherCAT Kommunikation steuert.

6.3 Frame

Das EtherCAT Datengramm beschreibt, welche Speicher-Bereiche ausgetauscht werden sollen. In einem Ethernet Frame können mehrere EtherCAT Datengramme verschickt werden (siehe Abbildung 10 – EtherCAT Datengramme). Die folgenden Abschnitte beschreiben die einzelnen Datengramme.

Abbildung 10 – EtherCAT Datengramme

6.3.1 Ventilsteuerung

Dieses Datengramm enthält die I/O Map für die Ventile. Mit dem richtigen Kommando kann der Kamera-Slave die Befehle ins Datengramm schreiben und der Ventilsteuer-Slave liest sie wieder raus, ohne dass der Master noch interagieren muss (siehe [1]).

CMD 13: ARMW: Auto Increment Read Multiple Write: Slave increments address. Slave puts read data into the EtherCAT datagram if received address is zero, otherwise slave writes the data into memory location.

6.3.2 Statistik

Dieses Datengramm enthält den Platzhalter für die Statistikdaten. Der Kamera-Slave schreibt die Daten ins Frame und der Master liest sie aus und stellt sie im GUI dar. Die Statistikdaten beinhalten die Anzahl detektierter Sugusse aller Sorten.

6.3.3 Initialisierung/Sugus Sorte Wahl

Dieses Datengramm enthält Initialisierungs-Informationen, wie zum Beispiel Reset oder welche Sugus-Sorten aussortiert werden sollen. Es könnte auch azyklisch in einem eigenen Ethernet Frame verschickt werden, weil das Setup nicht ständig ändert. Damit könnte man die benötigte Bandbreite verringern.

6.4 Slave

Nachdem der Slave mit dem Master synchronisiert ist, generiert er periodisch mit der Framerate einen Interrupt für den Kameratrigger. Somit weiss er in welchem Verhältnis zur System Zeit das Bild aufgenommen wurde und kann den Vorhalt für die Ventile richtig berechnen.

Auf dem Slave läuft ein Task, der periodisch die Daten zwischen dem EtherCAT Slave device und dem DSP Hauptspeicher austauscht. Er schreibt die Daten vom Ventil-Kommando FIFO sowie die Statistikdaten in den Slave-Speicher. Er holt die Initialisierungs-Daten vom Slave ab.

6.4.1 Implementation

Um die ganzen Mailbox-Funktionalitäten wie „Ethernet over EtherCAT“ oder „File Access over EtherCAT“ zu ermöglichen, kann noch ein Aplikations-Layer auf dem Slave in Betrieb genommen werden. Dies wird jedoch für die Basis-Anwendung des Sugus-Sortierers nicht benötigt.

6.5 Konfiguration

Für das EtherCAT Netzwerk muss eine Konfigurationsdatei erstellt werden. Sie beinhaltet die Devices, deren Topologie und die Speicherbereiche, die ausgetauscht werden können.

Diese Datei wird zu Beginn vom Master-Stack verarbeitet. Er initialisiert anhand dieser Informationen die Register der Slaves.

7 Scheduling

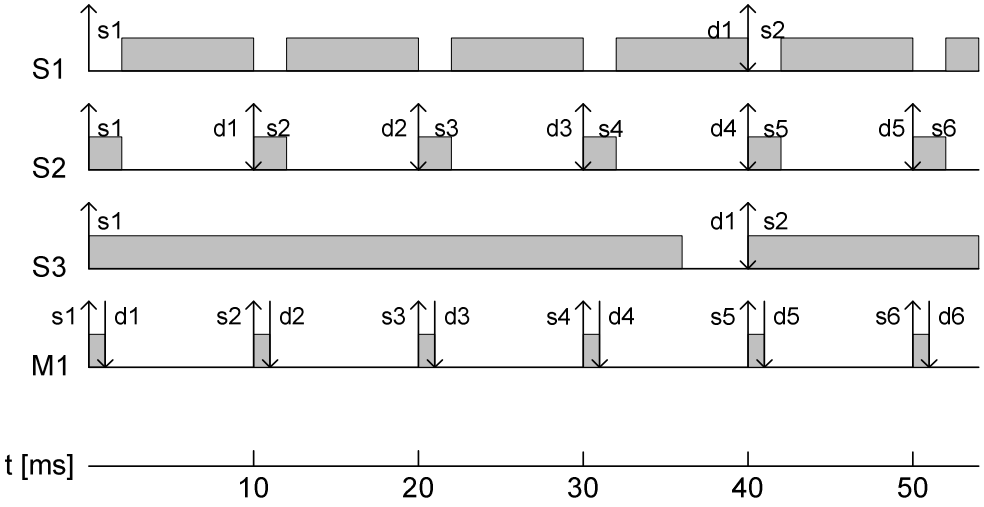
Auf dem EtherCAT Master muss folgender Task laufen:

- Initiieren der Transfers. Er muss die EtherCAT frames periodisch losschicken.

Auf dem EtherCAT Slave „Kamera“ müssen folgende Tasks laufen:

- Bildverarbeitung
- Bildtransfer von der Kamera in den DSP Hauptspeicher
- Kommunikation mit dem EtherCAT Slave ASIC (Daten vom DSP zum Slave und zurück)

Abbildung 11 zeigt das Scheduling der einzelnen Tasks.



S1 Bildverarbeitung
S2 EtherCAT Slave Kommunikation
S3 Bildtransfer (Kamera - Hauptspeicher)
M1 EtherCAT Pakete starten

Startzeitpunkt s

Endzeitpunkt d

Abbildung 11 – Scheduling der Tasks

8 GUI

Die Statistik-Daten, die der Master empfängt, sollen analog zum Sugus Sortierer 2004 in einem GUI dargestellt werden. Es bietet sich an, den Code wieder zu verwenden und nur das nötigste anzupassen.

9 Benötigte Komponenten

9.1 Master

- Master Protokoll Stack (Software)
  - Beckhoff ET9200
    - Source Code (C)
    - Betriebssysteme: Windows XP/CE, kann auf andere Plattformen portiert werden.
    - Kosten: 1000Euro, allenfalls Workshop 1Tag, 430Euro
- System mit Real Time OS (Hardware + Software)
  - Windows CE, da Beckhoff Master für dieses System geschrieben ist und Ferag auch Windows CE einsetzt.

9.2 Kamera-Slave

- Evaluations Kit (Hardware + Software)
  - Beckhoff EL9820
    - HW: Eval Board
    - SW: Slave Sample Code, State Machine
    - Kosten: 430Euro, allenfalls Workshop 1 Tag, 430 Euro
  - Adapter Karte Beckhoff EL980
    - Kosten: 35Euro
- Konfigurationstool für die Slave Devices (Software)
  - TwinCAT 30 Tage Lizenz im Evaluation Kit dabei
  - EtherCAT Konfigurator ET9000 von Beckhoff
    - Kosten: 210Euro

9.3 Ventile-Slave

Muss nochmals abgeklärt werden, welche Komponenten des bestehenden Systems übernommen werden können. Eventuell können die Klemmen KL2408 von Beckhoff übernommen werden.

- EtherCAT-Koppler (Hardware)
  - Beckhoff EK1100
    - Anzahl: 1
    - Preis: 98Euro
- EtherCAT-Klemmen (Hardware)
  - Beckhoff EL2008
    - 8-Kanal-Digital-Ausgangsklemmen 24V DC
    - Anzahl: 2
    - Preis: 38.50Euro
- E-Bus Endklemme (Hardware)
  - Beckhoff EL9010
    - Anzahl: 1
    - Preis: 9.50Euro

10 Tasks

- Bildverarbeitungs-Algorithmus
- EtherCAT Slave Kommunikation (SPI Schnittstelle)
- Master-Stack „portieren“, kompilieren
- Real-Time Betriebssystem in Betrieb nehmen
- User Applikation für Master-Stack
- EtherCAT Konfigurationsdatei erstellen (mit Software-Tool)