

## FreeRTOS Projekt

# Stellaris Robot Model

Autoren Gunnar Heimsch  
Nicola Käser  
Tobias Meerstetter  
Simon Plattner

Dozent Tim Wacher

Datum 12. Juni 2014

Ort Burgdorf

Version 1.0.0

Die Studierenden der Berner Fachhochschule haben im Rahmen des Moduls *Echtzeit-Betriebssysteme* in Gruppen ein kleines Projekt geplant und durchgeführt. Diese Dokumentation beschreibt das Vorgehen sowie das Ergebnis eines solchen Projektes, welches sich mit der Ansteuerung eines Robotermodells auseinandergesetzt hat.

# Vorwort

Einmal mehr im Werdegang eines angehenden Ingenieurs steht die Umsetzung eines Projektes zwischen uns und einer guten Note. Einmal mehr macht diese gute Note nur einen kleinen Teil der gesamten Bewertung des Moduls aus. Und ebenfalls einmal mehr macht die Dokumentation nur einen kleinen Teil der Bewertung des Projektes aus. Aus diesem Grund soll diese Dokumentation unter anderem dazu dienen, dass wir uns im Kurzfassen üben können. Es soll auf abrundendes Prosa verzichtet werden, wo immer dadurch sinnvoll Zeit gespart werden kann.

Diese Dokumentation richtet sich demnach an den bewertenden Dozenten, welcher sowohl die Entwicklungsschritte und die eingesetzten Methoden als auch die Sprache an sich sehr gut beherrscht. Sie soll aufzeigen, was wir uns überlegt haben, jedoch auf alles allgemeine oder unnötige verzichten.

Um den Projekt-Overhead durch die Dokumentation möglichst klein zu halten, verzichten wir auf das Anfügen von Stichwortverzeichnissen, ausführlichen Einleitungen, übermässiger Bebilderung und so ähnlichem.

Trotzdem wünschen wir dem Leser viel Vergnügen!

# Inhaltsverzeichnis

<b>1</b>	<b>Übersicht</b>	<b>1</b>
<b>2</b>	<b>Roboterarme</b>	<b>2</b>
<b>3</b>	<b>Förderbänder</b>	<b>5</b>
<b>4</b>	<b>Flipper</b>	<b>7</b>
<b>5</b>	<b>ECTS-Updater</b>	<b>8</b>
5.1	Funktion CAN_conveyor_status_handler . . . . .	9
5.2	Funktion find_ECTS . . . . .	9
<b>6</b>	<b>Ergebnis</b>	<b>10</b>

# 1 Übersicht

Das Grobe Zusammenspiel der Software ist im Taskdiagramm (Abbildung 1.1) dargestellt. Das genaue Zusammenspiel der Tasks wird in den entsprechenden Kapiteln genauer erläutert. In dieser Dokumentation wird nicht auf jedes Modul und jede Funktion einzeln eingegangen. Die Funktionsbeschreibungen sind dem Kommentar in der Software oder der Doxygen unterlagen zu entnehmen.

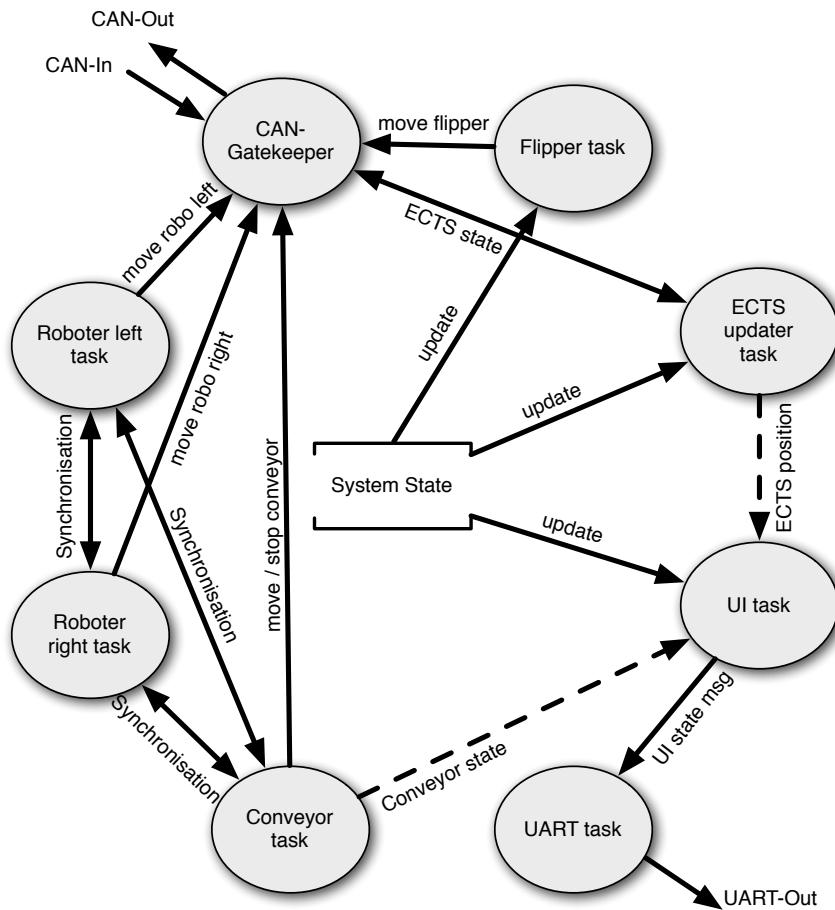


Abbildung 1.1: Taskdiagramm

## 2 Roboterarme

Die Ansteuerung der Roboterarme erfolgt über einen eigenen Task. Pro Arm wird dazu ein entsprechender Task erzeugt. Das bietet den Vorteil, dass beide Arme weitestgehend unabhängig voneinander arbeiten können. Diese Flexibilität könnte sich als Vorteil herausstellen, falls das Projekt zu einem späteren Zeitpunkt erweitert werden würde.

Abbildung 2.1 zeigt den linken Roboterarm. Der rechte ist nahezu identisch. Die Arme unterscheiden sich in der Mechanik leicht voneinander. Dieser kleine Unterschied kann auf Fertigungstoleranzen zurückgeführt werden. Die einzige für das Projekt relevante Konsequenz ist, dass beide Arme nach dem gleichen Befehl leicht verschiedene Positionen anfahren.

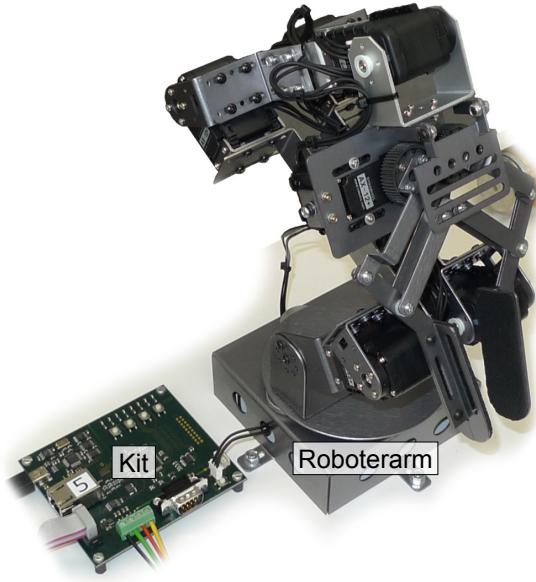


Abbildung 2.1: Linker Roboterarm

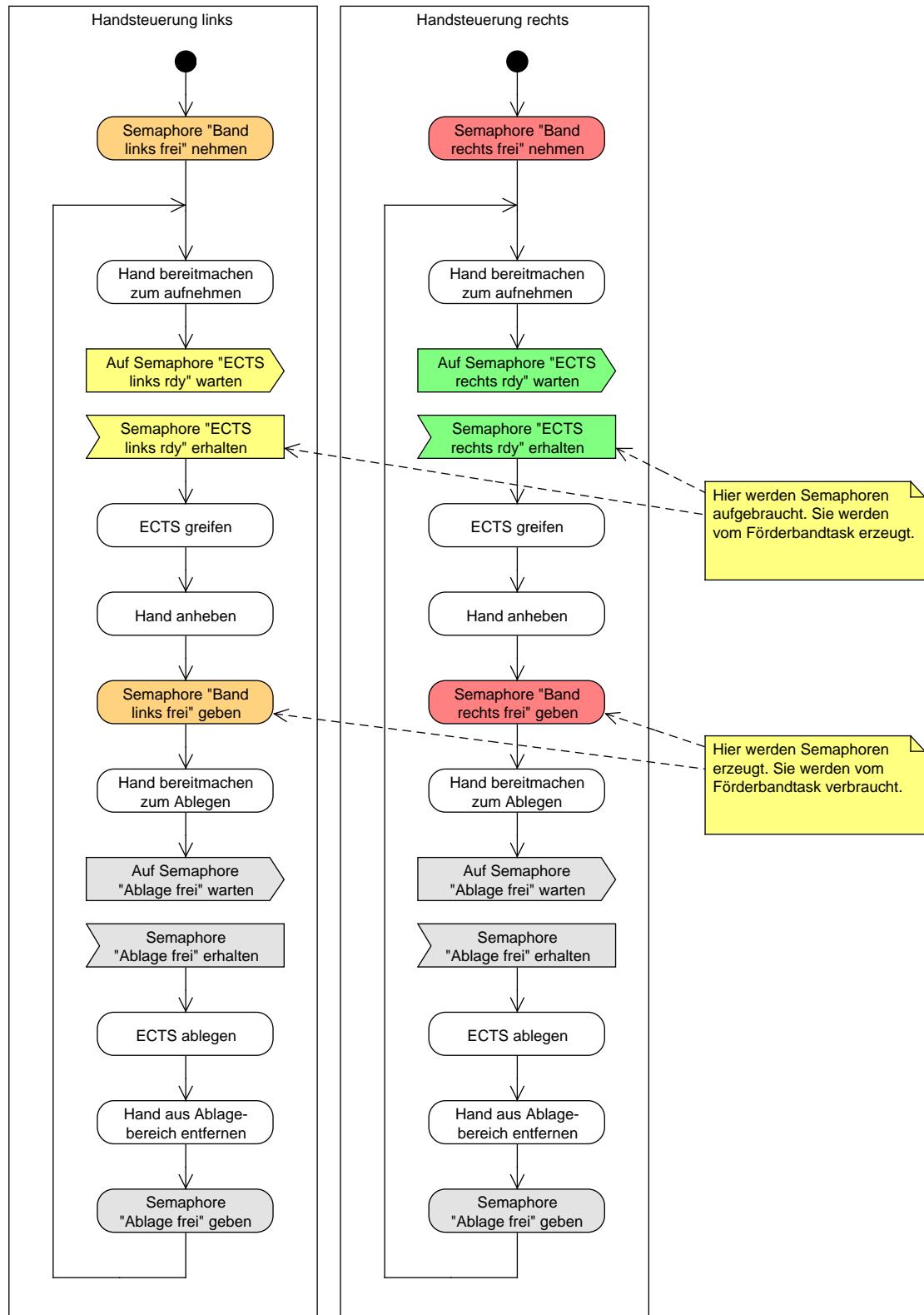
Abbildung 2.2: Aktivitätsdiagramm *Roboterarme*

Abbildung 2.2 zeigt die Funktionsweise der Arme auf. Zu beachten ist dabei, dass die Roboterarme sowohl gegenseitig, als auch zwischen sich und den Förderbändern, eine Synchronisation mittels Semaphoren vorsehen. Dabei enthalten die Semaphoren jeweils eine Information über den Zustand der Anlage. Sie schützen dadurch faktisch einen Teil des Raumes vor mehrfacher Nutzung (zum Beispiel zwei Roboterarme am gleichen Ort).

Um die *Gesten* der Roboter möglichst einfach einstellen zu können, wurde ein einfaches kinematisches Modell der Roboter erstellt. Auf diese Weise kann eine vorgegebene *Hand*-Stellung in die zugehörigen Gelenk-Winkel überführt werden. Aufgrund der Anzahl sowie Anordnung der Gelenke kommt dieses inverse Kinematik-Modell mit analytischen Berechnungen aus - es sind keine (numerischen) Näherungslösungen nötig.

### 3 Förderbänder

Das Modell verfügt über drei Förderbänder, die jeweils über zwei Lichtschranken verfügen. In Abbildung 3.1 ist das linke Förderband abgebildet. Das Förderband rechts ist gleich aufgebaut und einfach um 180° gedreht, dies gilt auch für die Anordnung der Lichtschranken. Lediglich der Motor und die Elektronik befinden sich aus praktischen Gründen auf der anderen Seite was aber die Ansteuerung nicht beeinflusst. Beim Förderband in der Mitte befinden sich die Lichtschranken nicht am Anfang sondern kurz vor dem Verteiler („Flipper“).

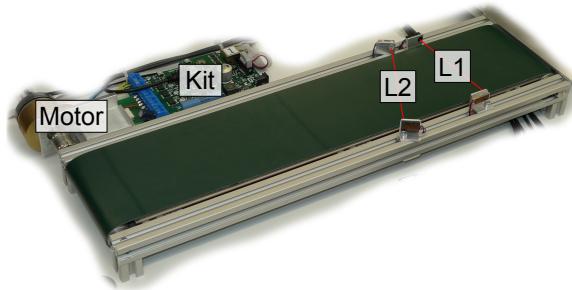


Abbildung 3.1: Förderband Links

Für die Ansteuerung des linken und rechten Förderbands wird je ein eigener Task verwendet, damit der parallele Ablauf einfach behandelt werden kann. Der Taskablauf ist jedoch bei beiden Tasks der selbe; dieser ist in Abbildung 3.2 ersichtlich. Das mittleren Förderbands wird im Flipper-Task angesteuert und ist somit in Kapitel 4 beschrieben.

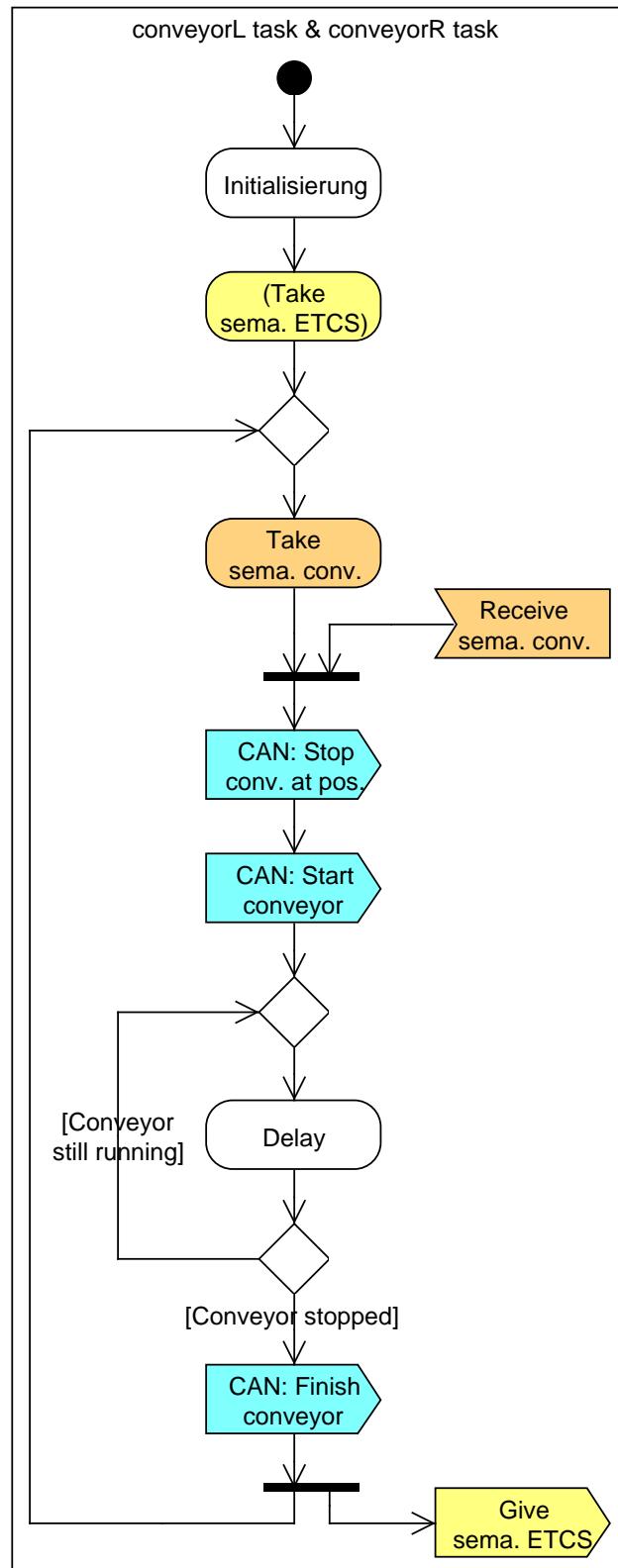


Abbildung 3.2: Aktivitätsdiagramm der Förderband-Tasks

## 4 Flipper

Am Ende des mittleren Förderbandes befindet sich eine Vorrichtung mit der die ECTS-Steine auf das linke oder rechte Förderband geschoben werden können.

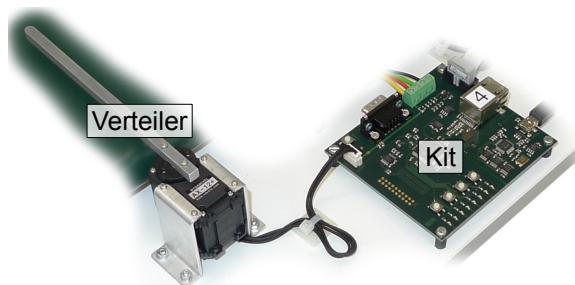


Abbildung 4.1: Flipper und zugehörige Elektronik

Auch für dieses System verwenden wir einen eigenen Task. Zusätzlich wird in diesem aber auch gleich das mittlere Förderband angesteuert. Das mittlere Förderband wird im Gegensatz zu den beiden anderen nicht immer gestoppt und läuft also während der ganzen Programmlaufzeit ständig.

In unserem Projekt haben wir dafür entschieden die ECTS immer abwechselnd auf das linke und das rechte Förderband zu schieben damit beide Seiten etwa gleich ausgelastet sind. Dementsprechend ist auch der Taskablauf relativ einfach.

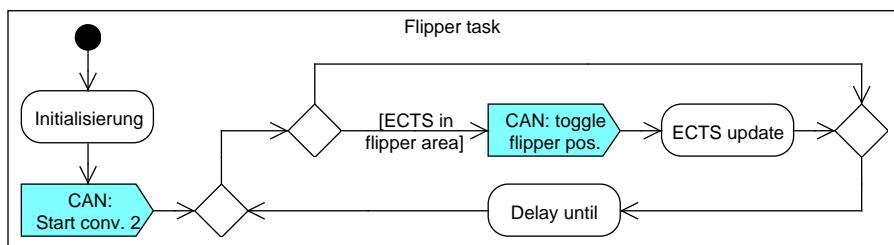


Abbildung 4.2: Aktivitätsdiagramm des Flipper-Tasks

## 5 ECTS-Updater

Damit der Zustand aller ECTS-Steine zu jeder Zeit abgefragt werden kann (z. B. für die graphische Darstellung), wurde ein Task entworfen der diese Informationen regelmäßig aktualisiert. Abgelegt sind diese Informationen jeweils in einer Struktur die folgende Daten enthält:

- id (*uint8\_t*):** Die ID des ECTS zur Identifikation
- x (*uint16\_t*):** Die x-Position des ECTS, sie entspricht der zurückgelegten Distanz auf dem jeweiligen Förderband in Millimeter
- y (*uint16\_t*):** Die y-Position des ECTS, sie entspricht der links-rechts-Verschiebung auf dem Fliessband (aufgeteilt in 8 „Bahnen“)
- z (*z\_pos*):** Das Teilsystem bei dem sich das ECTS befindet

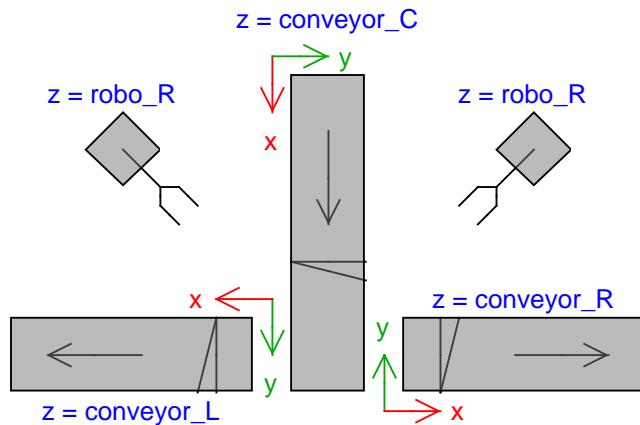


Abbildung 5.1: Übersicht über die Positionsdaten

In Abbildung 5.2 ist der Taskablauf ersichtlich. Die x- und y-Positionen aller ECTS-Strukturen werden mit Informationen von Lichtschranken aktualisiert. Zusätzlich wird die x-Position regelmäßig mit einem von der Förderband-Geschwindigkeit abhängigen Wert inkrementiert. Beim Wechseln in ein anderes Teilsystem wird die x-Position wieder auf 0 gesetzt und die z-Position aktualisiert.

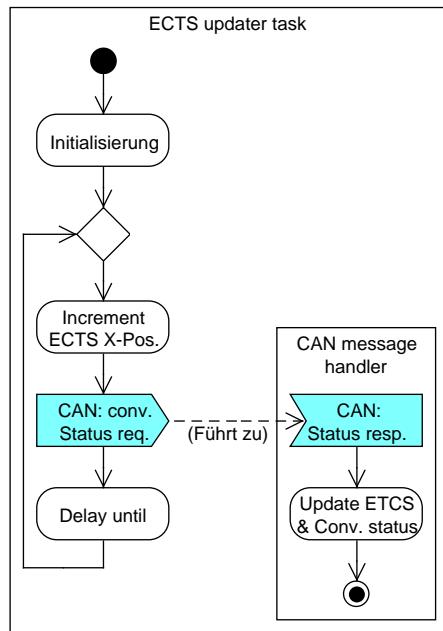


Abbildung 5.2: Aktivitätsdiagramm des ECTS-Updater-Tasks

Zur Vereinfachung des Taskaufbaus und damit auch andere Tasks einfach die ECTS-Informationen aktualisieren können (beispielsweise beim Wechsel vom Förderband zum Roboterarm), wurden Hilfsfunktionen entwickelt. Diese werden nachfolgend kurz erklärt.

## 5.1 Funktion CAN\_conveyor\_status\_handler

Die Funktion `CAN_conveyor_status_handler` wird aufgerufen, nachdem per CAN eine Antwort auf die Statusanfrage der Lichtschranken empfangen wurde. In ihr werden die Daten der Antwort verarbeitet und damit die ECTS-Strukturen aktualisiert.

## 5.2 Funktion find\_ECTS

Wenn eine ECTS-Struktur mit Informationen aus dem System aktualisiert werden soll muss vorerst die richtige ECTS-Struktur gefunden werden, denn die ECTS-Steine unterscheiden sich nur durch die Position. Für diese Aufgabe wurde die Funktion `find_ECTS` implementiert. Als Eingangsparameter wird die z-Position angegeben in der sich das ECTS-Befindet. Die Funktion liefert dann einen Zeiger auf die gefundene ECTS-Struktur an dieser z-Position zurück. Wenn sich mehrere ECTS an dieser Stelle befinden (bei den Förderbändern), so wird ein Zeiger auf die ECTS-Struktur mit der grössten x-Position zurückgegeben.

## 6 Ergebnis

Unser Ziel war es, einen möglichst schnellen Kreislauf der ECTS zu ermöglichen. Das sollte mittels verschiedener Optimierungen<sup>1</sup> erreicht werden. Vorgesehen war zu diesem Zweck das beschleunigen der Roboterarme. Sie sollten möglichst flüssig handeln und Wartezeiten vermeiden.

Das gesteckte Ziel konnte nicht erreicht werden. Grund dafür ist die Hardware - sie ist wesentlich langsamer als angenommen. Beispielsweise führt der Roboter nur maximal etwa 2-3 Befehle pro Sekunde aus. Für eine flüssige Bewegung ist das deutlich zu wenig. Weiterhin schwingen die Arme nach jeder Bewegung nach. Da eine Rückkoppelung nicht gegeben ist, kann dieses Verhalten nicht auf vernünftige Weise kompensiert werden.

Während unserer Tests haben sich einige Hardwarekomponenten anders verhalten als gemäss Handbuch zu erwarten wäre<sup>2</sup>. Dieser Sachverhalt wurde mit dem Dozenten besprochen.

Im Bezug auf das verwendete Echtzeitbetriebssystem konnte eine funktionierende Lösung entwickelt und implementiert werden. Das System läuft dank der umgesetzten Task-Synchronisation stabil<sup>3</sup>.

---

<sup>1</sup>Optimierungen gegenüber der Beispillösung; der Quellcode dieser anderen Lösung stand nicht zur Verfügung.

<sup>2</sup>Insbesondere der Totalausfall einer der beiden Arme kam unerwartet.

<sup>3</sup>Das System wurde aus Zeitgründen nicht ausgiebig getestet. Bei den durchgeführten Tests konnte jedoch nie ein Synchronisationsproblem oder ein Deadlock beobachtet werden.