optonic
vision · automation

Profinet-Schnittstelle

mikado

Version 2.1

Ein Produkt von
Optonic
www.mikado-robotics.de

Optonic GmbH | Zollhallenstr. 11 | 79106 Freiburg | Deutschland/Germany
Tel.: +49 761-79095-0 | E-Mail: info@optonic.com
© Copyright Optonic GmbH, 2015-2023

Inhaltsverzeichnis

1 Funktion der Schnittstelle **3**

2 Einrichtung **4**
- 2.1 GSDML . . . 4
- 2.2 Byte-Reihenfolge . . . 4
- 2.3 Datentypen . . . 4
- 2.4 Rotation-Konvention . . . 5

3 Wortbelegung **5**
- 3.1 Standard-Wortbelegung . . . 6
- 3.2 Live-Wörter . . . 7
- 3.3 LifeBit . . . 7
- 3.4 Mikado-Status . . . 7

4 Fehler-Codes **8**

5 Genereller Ablauf der Kommunikation **9**

6 Kommandos **10**
- 6.1 `RunMikadoProg` . . . 10
- 6.2 `StartSequence` . . . 11
- 6.3 `PauseMikadoProg` . . . 11
- 6.4 `StopMikadoProg` . . . 11
- 6.5 `ResetMikadoProg` . . . 11
- 6.6 `SyncAll` . . . 12
- 6.7 `ResetInternalIndices` . . . 12
- 6.8 `StartCalib` . . . 12
- 6.9 `CaptureCalib` . . . 12
- 6.10 `FinishCalib` . . . 13
- 6.11 `readPoseAndPublishTool0` . . . 13

7 Beispielablauf für Select-Zellen **13**

1 Funktion der Schnittstelle

Die Schnittstelle bietet die folgenden drei grundlegenden Funktionen:

- Starten und Stoppen des Mikado-Programms bzw. einer einzelnen Sequenz innerhalb des Mikado-Programms.
- Die Synchronisation von Mikado-Variablen mit der SPS.
- Durchführen der Hand-Auge-Kalibrierung.

Für die ersten beiden genannten Funktionen muss ein entsprechendes Programm in Mikado geladen sein.

2 Einrichtung

2.1 GSDML

Die Schnittstellenbeschreibung bei ProfiNet-Anbindung ist über die GSDML-Datei festgelegt:

`GSDML-V2.35-HILSCHER-CIFX RE PNS-20190108.xml`

Melden Sie sich beim Mikado Support, um die GSDML-Datei zu erhalten. Folgendes Profinet-IO Gerät der GSDML ist hierbei am Master einzubinden:

`CIFX RE/PNS V3.5 - V3.5.x`

Der Mikado-Controller stellt folgendes Ein-/Ausgangslayout zur Verfügung. Dabei sind Ein- und Ausgänge aus Sicht des Profinet-Masters definiert:

- 64 Byte Eingänge
- 64 Byte Ausgänge

2.2 Byte-Reihenfolge

Die Byte-Reihenfolge entspricht der Siemens Konvention Big-Endian. Bei der Interpretation der Wörter muss deshalb je nach Konfiguration des Master-Systems ein Byte-Swap erfolgen.

2.3 Datentypen

Daten werden prinzipiell wortweise, also in 2-Byte-Pakete, übertragen. Wie oben erwähnt, ist bei der Interpretation der Daten die Byte-Reihenfolge Big-Endian zu beachten. Folgende Datentypen sind möglich:

- Bits: Das Wort wird in 16 einzelne Bits zerlegt, wobei jedes Bit als Bool interpretiert wird.
- Integer: Das Wort wird als Ganzzahl interpretiert.
- Double: Zwei Wörter (32 Bit) werden gemeinsam als eine Gleitkommazahl interpretiert.
- Pose: 12 Wörter die als sechs Double-Werte interpretiert werden (X,Y,Z,R1,R2,R3).
- PoseList (nur für Output): 14 Wörter die als sechs Double-Werte und zwei Integer-Werte interpretiert werden (X,Y,Z,R1,R2,R3,Len,Idx).
- StringList (nur für Output): 4 Wörter die als ein Double-Wert und zwei Integer-Werte interpretiert werden (Val,Len,Idx).

2.4 Rotation-Konvention

Zur Übertragung von Messungen oder Roboterpositionen werden Posen verwendet. Dabei wird die Position als X-, Y-, und Z-Wert und die Orientierung als drei Rotationswerte um die jeweilige Achse übertragen. Je nach Roboterhersteller gibt es hierfür unterschiedliche Konventionen, wobei Mikado die in nachfolgender Tabelle aufgeführten Konventionen unterstützt. Standardmäßig werden die Rotationswerte nach der **Roll-Nick-Gier (RPY)** Konvention interpretiert. Die verwendete Konvention kann im Assistant unter dem Reiter ‚Profinet-Konfiguration' eingestellt werden.

Tab. 1: Unterstützte Konventionen

ID	Konvention	R1	R2	R3
0	*rxyz* (Default) Stäubli	*X* (Roll)	*Y'* (Nick/Pitch)	*Z"* (Gier/Yaw)
1	*sxyz* Fanuc/Motoman	*X* (A)	*Y* (B)	*Z* (C)
2	*rzyx* Kuka/ABB	*Z* (A)	*Y'* (B)	*X"* (C)

3 Wortbelegung

Die Belegung der Wörter kann im Mikado-Assistant unter Profinet-Konfiguration angepasst und erweitert werden. Dort können ebenfalls die mit den Wörtern synchronisierten Mikado-Variablen definiert werden.

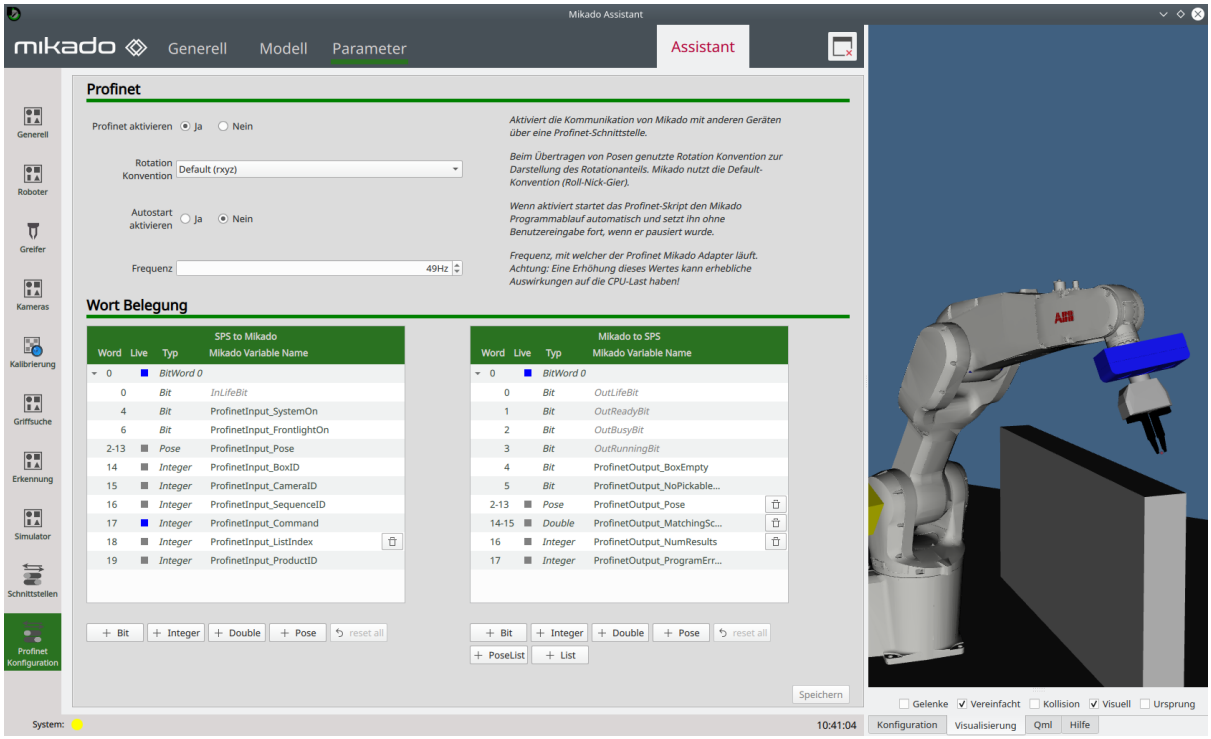


Abb. 1: Assistant zum Anpassen der Wortbelegung und synchronisierter Mikado-Variablen

3.1 Standard-Wortbelegung

Größtenteils fungieren die verwendeten Wörter zur Übertragung der Werte von Mikado-Variablen. Davon ausgenommen sind die in Abbildung *Übersicht der Standard-Belegung von Input und Output Byte-Arrays* grau gekennzeichneten Wörter, deren Werte unabhängig vom Mikado-Programm gesetzt werden.

SPS zu Mikado

Word	Byte	Bit		
0	0	0	`InLifeBit` (live)	
		1		
		2		
		3		
		4	`InSystemOn` (live)	
		5		
		6	`InFrontLightOn` (live)	
		7		
	1	8		
		9		
		10		
		11		
		12		
		13		
		14		
		15		
1	2			*reserviert für Bits*
	3			
2	4		Double	`InPoseX`
	5			
3	6			
	7			
4	8		Double	`InPoseY`
	9			
5	10			
	11			
6	12		Double	`InPoseZ`
	13			
7	14			
	15			
8	16		Double	`InPoseR1`
	17			
9	18			
	19			
10	20		Double	`InPoseR2`
	21			
11	22			
	23			
12	24		Double	`InPoseR3`
	25			
13	26			
	27			
14	28		Integer	`InBoxID`
	29			
15	30		Integer	`InCameraID`
	31			
16	32		Integer	`InSequenceID`
	33			
17	34		Integer	`InCmd` (live)
	35			
18	36		Integer	`InListIndex`
	37			
19	38		Integer	`InProductID`
	39			
20	40			
	41			
21	42			
	43			
22	44			
	45			

Mikado zu SPS

Word	Byte	Bit		
0	0	0	`OutLifeBit` (live)	
		1	`OutReadyBit` (live)	
		2	`OutBusyBit` (live)	
		3	`OutRunningBit` (live)	
		4	`OutBoxEmptyBit` (live)	
		5	`OutNoPickableObjectsBit` (live)	
		6		
		7		
	1	8		
		9		
		10		
		11		
		12		
		13		
		14		
		15		
1	2			*reserviert für Bits*
	3			
2	4		Double	`OutPoseX`
	5			
3	6			
	7			
4	8		Double	`OutPoseY`
	9			
5	10			
	11			
6	12		Double	`OutPoseZ`
	13			
7	14			
	15			
8	16		Double	`OutPoseR1`
	17			
9	18			
	19			
10	20		Double	`OutPoseR2`
	21			
11	22			
	23			
12	24		Double	`OutPoseR3`
	25			
13	26			
	27			
14	28		Double	`OutMatchingScore`
	29			
15	30			
	31			
16	32		Integer	`OutNumResults`
	33			
17	34		Integer	`OutErrorCode` (live)
	35			
18	36			
	37			
19	38			
	39			
20	40			
	41			
21	42			
	43			
22	44			
	45			

Abb. 2: Übersicht der Standard-Belegung von Input und Output Byte-Arrays

3.2 Live-Wörter

Die in Abbildung *Übersicht der Standard-Belegung von Input und Output Byte-Arrays* mit *live* gekennzeichneten Aus- und Eingangswörter werden in jedem Zyklus synchronisiert. Dies entspricht standardmäßig einer Rate von 50 Hz. Alle anderen Wörter werden nur bei Bedarf synchronisiert oder durch den Aufruf des Kommandos `SyncAll`.

3.3 LifeBit

Um den Status der Kommunikation zwischen Mikado und SPS zu überprüfen wird ein LifeBit verwendet. Dabei liest die SPS den Wert von `OutLifeBit` und schreibt diesen unverändert auf `InLifeBit`. Der Mikado-Profinet-Adapter liest den Wert von `InLifeBit` und schreibt den negierten Wert auf `OutLifeBit`. Verändert sich der Wert von `OutLifeBit` nicht regelmäßig, gilt die Kommunikation als unterbrochen und ein entsprechender Fehler wird im Mikado-Status angezeigt.

3.4 Mikado-Status

Der Status von Mikado wird mittels drei verschiedener Bits an die SPS übertragen. Folgende Tabelle fasst die Bedeutung der einzelnen Bits zusammen.

Tab. 2: Bedeutung der einzelnen Status-Bits

Name	Erläuterung	Wert
`OutReadyBit`	System-Status von Mikado.	True = Mikado betriebsbereit False = Mikado nicht bereit
`OutBusyBit`	Status des Profinet-Adapters.	True = Kommando wird entgegengenommen und verarbeitet. False = Adapter ist bereit neues Kommando entgegenzunehmen
`OutRunningBit`	Programm-Status von Mikado	True = Mikado-Programm läuft (entspricht in Mikado: Grün oder Gelb) False = Programm läuft nicht (entspricht in Mikado: Orange, Rot oder Grau)

Bemerkung: Neue Kommandos werden nur entgegengenommen, wenn `OutBusyBit` = False

4 Fehler-Codes

Das Ausgangs-Wort `OutErrorCode` kodiert Fehlermeldungen. Folgende Tabelle fasst alle möglichen Fehler-Codes zusammen und erläutert deren Bedeutung.

Tab. 3: Mögliche Fehlermeldungen

Code	Fehlername	Beschreibung
0	`NoError`	Befehl erfolgreich
>0	Fehler im Mikado-Programm	Fehler im Mikado Ablauf. Details siehe Mikado-Programm
-100	`UnexpectedError`	Unerwarteter Fehler aufgetreten
-1	`ServiceCallFailed`	Aufruf fehlgeschlagen
-2	`WrongType`	Falscher Datentyp
-3	`WrongIndex`	Angeforderter Index nicht verfügbar
-4	`NoReplyValue`	Keinen Rückgabe-Wert erhalten
-5	`TooManyReplyValues`	Anzahl der Rückgabewerte höher als erwartet
-6	`ErrorMikadoProgram`	Aufgerufenes Mikado-Programm hat einen Fehler verursacht oder den Ablauf in einem invaliden Zustand verlassen.
-7	`IndexOutOfRange`	Angegebener (Result-)Index außerhalb des Wertebereichs
-8	`UnknownCommand`	unbekanntes Kommando erhalten
-9	`LifebitClientMissing`	Lifebit des Clients wurde nicht erkannt. Outputs werden in diesem Zustand nicht aktualisiert.
-11	`TriggeringFailed`	Ausführung einer Mikado-Sequenz fehlgeschlagen
-12	`ReadingVariableFailed`	Mikado-Variable konnte nicht gelesen werden (existiert die Variable im aktuell geladenen Mikado-Programm?)
-13	`WritingVariableFailed`	Mikado-Variable konnte nicht geschrieben werden (existiert die Variable im aktuell geladenen Mikado-Programm?)
-15	`CancelCalibrationFailed`	Abbrechen einer bereits gestarteten Kalibrierung fehlgeschlagen
-16	`CaptureForCalibrationFailed`	Aufnahme für laufende Kalibrierung nicht erfolgreich
-17	`FinishCalibrationFailed`	Kalibrierung berechnen und speichern war nicht erfolgreich
-18	`StartCalibrationFailed`	Kalibrierung konnte nicht gestartet werden

5 Genereller Ablauf der Kommunikation

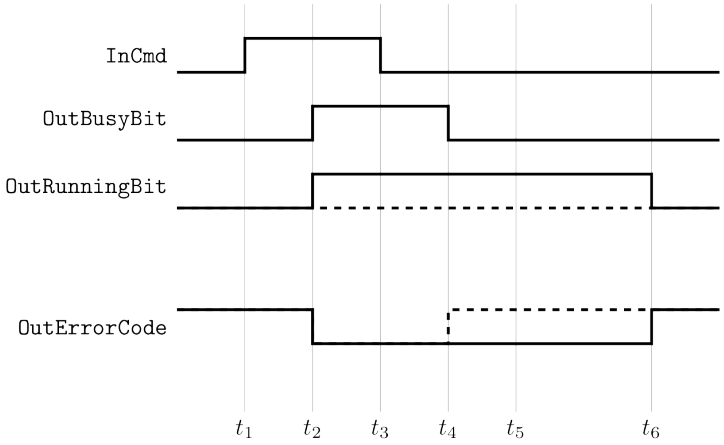
Die folgende Ablaufbeschreibung gilt für alle Kommandos. Bedingung für die Kommunikation ist `OutReadyBit = 1` von Mikado, welches gesetzt wird, wenn Mikado und Kamera bereit sind. Das SPS-Kommando `InCmd` darf nur gesetzt werden, wenn `OutBusyBit = 0`.

Tab. 4: Genereller Ablauf der Kommunikation

Zeit	SPS	Mikado
$< t_1$	Warte auf `OutReadyBit = 1` Warte auf `OutBusyBit = 0` Parameter setzen	Warten auf Kommando
t_1	`InCmd` setzen Warte auf `OutBusyBit = 1`	

t_2		`OutErrorCode = 0`
		`OutBusyBit = 1`
		Ausführung des Befehls
		`OutRunningBit = 1` (MIK-Programm starten)

t_3	`InCmd` zurücksetzen auf `0` Warte auf `OutBusyBit = 0`	
t_4		`OutErrorCode` schreiben `OutBusyBit = 0`
t_5	Lesen von `OutErrorCode` Warte auf `OutRunningBit = 0`	
t_6	Lesen von `OutErrorCode` Ggf. Lesen von Variablen	`OutRunningBit = 0` (MIK-Programm endet)



- Wird ein Kommando gesetzt (`InCmd` $\neq$ `0`), antwortet Mikado mit `OutBusyBit = 1` und setzt zunächst den `OutErrorCode` zurück. Auf `OutBusyBit = 1` antwortet die SPS mit `InCmd = 0` und wartet dann auf `OutBusyBit = 0`.
- Die SPS wertet den `OutErrorCode` aus. Wenn diese den Wert 0 hat, ist das Kommando erfolgreich ausgeführt worden.
- Je nach Kommando wird das `OutRunningBit = 1` und signalisiert, dass Mikado gestartet wurde und das aktuelle Programm abarbeitet. Wird das `OutRunningBit = 0` können der `OutErrorCode` und die entsprechende Output-Variablen ausgelesen werden.

6 Kommandos

Bemerkung: Wichtig: Vor dem Setzen des Kommandos zuerst zugehörige Parameter setzen.

Die notwendigen Informationen für ein Kommando werden zu Beginn des Kommandos eingelesen. Variablen und Ergebnisse hängen von der Implementierung im Mikado-Programm ab. Die meisten Output-Wörter repräsentieren Mikado-Variablen und können im Mikado-Programm individuell bearbeitet werden. Beispielsweise können so eigene Fehlercodes über ProgramErrorCode nach außen gegeben werden.

Tab. 5: Verfügbare Kommandos

InCmd	Kommandoname	Beschreibung
0	`Idle`	Wartezustand
1	*RunMikadoProg*	Mikado-Programm starten/fortsetzen
2	*StartSequence*	Eine Sequenz starten. Hierfür muss SequenceID zuvor gesetzt sein
3	*PauseMikadoProg*	Ablauf nach Ausführung des aktuellen Tasks anhalten
4	*StopMikadoProg*	Mikado-Programm stoppen
5	*ResetMikadoProg*	Mikado-Programm stoppen und alle Variablen und Zustände im Mikado zurücksetzen
6	*SyncAll*	Handshake ausführen um alle Input-Variablen einzulesen und alle Output-Variablen zu schreiben
7	*ResetInternalIndices*	Setzt den Index aller ausgegebenen PoseListen und StringListen auf -1.
8	*StartCalib*	Hand-Auge-Kalibrierung starten
9	*CaptureCalib*	Löst eine Bildaufnahme und Messung des Kalibriermusters aus
10	*FinishCalib*	Hand-Auge-Kalibrierung berechnen und speichern
11	*readPoseAndPublishTool0*	Setzt die Pose des Roboterflanschs in Mikado

6.1 `RunMikadoProg`

Mit diesem Kommando wird das aktuelle Mikado-Programm gestartet oder fortgesetzt. Die notwendigen Informationen für das Kommando werden zu Beginn des Kommandos eingelesen:

- `InProductID`
- `InBoxID`
- `InListIndex`

Nach erfolgreicher Ausführung des Kommandos wechselt das `RunningBit = True` und signalisiert das laufende Mikado-Programm. Sobald dieses von Mikado zurückgesetzt wird, kann der

ErrorCode ausgewertet werden und die Ergebnisse mit dem Kommando

- SyncAll

abgerufen werden.

6.2 StartSequence

Mit diesem Kommando wird die Mikado-Sequenz mit der in Wort SequenceID hinterlegten ID ausgeführt. Die ID einer Sequenz lässt sich ab Programm-Version 3 in dem Textfeld rechts neben dem Sequenz-Namen einstellen. In früheren Programm-Versionen muss der Sequenz-Name gleich der ID gesendeten ID sein, beispielsweise die Ganzzahl ‚42'. Die notwendigen Informationen für das Kommando werden zu Beginn des Kommandos eingelesen:

- InSequenceID
- InProductID
- InBoxID
- InListIndex

Nach erfolgreicher Ausführung des Kommandos wechselt das RunningBit = 1 und signalisiert das laufende Mikado-Programm. Sobald dieses von Mikado zurückgesetzt wird, kann der ErrorCode ausgewertet werden und die Ergebnisse mit dem Kommando

- SyncAll

abgerufen werden.

6.3 PauseMikadoProg

Mit diesem Kommando wird die Ausführung des aktuellen Programms pausiert.

6.4 StopMikadoProg

Mit diesem Kommando wird die Ausführung des aktuellen Programms gestoppt.

6.5 ResetMikadoProg

Mit diesem Kommando wird die Ausführung des aktuellen Programms gestoppt und der Zustand des Programms und aller Variablen werden zurückgesetzt.

6.6 SyncAll

Alle Input-Variablen von der SPS lesen, sowie alle Output-Variablen von Mikado auf die Profinet-Schnittstelle schreiben.

Bemerkung: Anmerkung zum Verhalten bei synchronisierten Listen:

Für Mikado-Variablen vom Typ `PoseList` oder `StringList` (siehe auch Abschnitt *Datentypen*) wird bei diesem Kommando das nächste Element in der Mikado-Variable geholt. Ist der aktuelle Index bereits beim letzten Messergebnis, so beginnt der Index wieder beim ersten Ergebnis (0). Listenwert an der Stelle Index, Länge der Liste und aktueller Index liegen danach an den Ausgabedaten des Feldbusses an. Durch wiederholtes Aufrufen dieses Kommandos können somit iterativ alle Werte der Liste übertragen werden.

6.7 ResetInternalIndices

Setzt den Index aller ausgegebenen PoseListen und StringListen auf -1. Beim nächsten Aufruf von `SyncAll` liegt somit das erste Element (idx=0) an den Ausgabedaten des Feldbusses an.

6.8 StartCalib

Bemerkung: Dieses Kommando findet in der Regel nur bei Select-Zellen Anwendung.

Mit diesem Kommando wird die Hand-Augen-Kalibrierung gestartet. Die notwendigen Informationen für das Kommando werden zu Beginn eingelesen:

- `InCameraID`
- `InFrontLightOn`

6.9 CaptureCalib

Bemerkung: Dieses Kommando findet in der Regel nur bei Select-Zellen Anwendung.

Mit diesem Kommando werden Posen für die Hand-Augen-Kalibrierung gesammelt. Der Kalibrierungsprozess muss vorher durch das Kommando `StartCalib` gestartet worden sein. Es sollten zwischen 10 und 20 Posen gesammelt werden. Dieses Kommando muss dementsprechend häufig ausgeführt werden. Die notwendigen Informationen für ein Kommando werden zu Beginn des Kommandos eingelesen:

- `InPose`

Beispielablauf:

- Roboter fährt Pose an
- Das Kommando `readPoseAndPublishTool0` wird ausgelöst
- Das Kommando `CaptureCalib` wird ausgelöst
- Mikado führt eine Bildaufnahme durch

6.10 `FinishCalib`

Bemerkung: Dieses Kommando findet in der Regel nur bei Select-Zellen Anwendung.

Mit diesem Kommando wird die Hand-Augen-Kalibrierung berechnet. Dies ist der letzte Schritt im Prozess der Hand-Augen-Kalibrierung. Dies erfolgt nach einer ausreichenden Anzahl von gesammelten Patternposen mit Hilfe von `CaptureCalib`.

6.11 `readPoseAndPublishTool0`

Bemerkung: Dieses Kommando findet in der Regel nur bei Select Zellen Anwendung.

Mit diesem Kommando wird eine von der SPS übermittelte Pose eingelesen und in Mikado als aktuelle Pose des Roboterflanschs (tool0) gesetzt. Für die Hand-Auge-Kalibrierung muss die Position des Roboters bei einer Bildaufnahme bekannt sein, wobei Mikado Select selbst keine Informationen über den Roboterzustand hat. Die notwendigen Informationen für das Kommando werden zu Beginn des Kommandos eingelesen:

- `InPose`

7 Beispielablauf für Select-Zellen

Anhand des Beispiel-Programms `select_findObjects.program` wird im Folgenden gezeigt, wie sich Mikado über Profinet steuern lässt. Das Beispiel-Programm findet sich ab Mikado-Version 2.1 in der Programm-Bibliothek.

Voraussetzungen:

- Es muss eine Produktliste geladen sein, die ein Produkt mit dem Namen `P_0` enthält.
- Es muss eine Szene geladen sein, die eine ROI mit dem Namen `ROI_0` enthält.

Tab. 6: Beispiel-Ablauf für eine Select-Zelle

#	SPS	Mikado
Finde Objekte		
1	Setze `InSequenceID = 1` Setze `InBoxID= 0` Setze `InProductID = 0` Setze `InCmd= 2` (`RunSequence`)	
2		Starte Sequenz mit ID = 1 (*Finde Objekte*) Setze `OutNumResults` Ende der Sequenz: `OutRunningBit = 0`
Kommuniziere Ergebnisse		
3	Setze `InCmd = 6` (`syncAll`) Lese `OutNumResults` Setze `InListIndex = 0` Setze `InSequenceID = 2` Setze `InCmd = 2` (`RunSequence`)	
4		Starte Sequenz mit ID = 2 (*get Result*) Setze `OutPose` w.r.t. `InListIndex` Ende der Sequenz: `OutRunningBit = 0`
5	Setze `InCmd = 6` (`syncAll`) Lese `OutPoseX`, `OutPoseY`, ... Setze `InListIndex = 1` Setze `InSequenceID = 2` Setze `InCmd = 2` (`RunSequence`)	
6		*wiederhole Schritt 4*
7	*wiederhole Schritt 5 mit* `InListIndex++`	
8	Setze `InCmd = 6` (`syncAll`) Lese `OutPoseX`, `OutPoseY`, ... Setze `InListIndex` = `OutNumResults` Setze `InSequenceID = 2` Setze `InCmd = 2` (`RunSequence`)	
9		*wiederhole Schritt 4*
10	Setze `InCmd = 6` (`syncAll`) Lese `OutPoseX`, `OutPoseY`, ...	