Eurobot 2014

CAN Summary

Simon Grossenbacher

Dieses Dokument gibt eine Übersicht über die Kommunikation zwischen den einzelnen Komponenten des Roboters, die via des CAN-Bus erfolgt.

1 Spielfeld-Definitionen

Die im folgenden Abschnitt 2 aufgeführten Protokolle beziehen sich alle auf Dimensionen und Eigenschaften des Spielfelds. Daher muss eine gleichbleibende Basis geschaffen werden.

Ein genormter Nullpunkt ist eine zwingende Voraussetzung für eine zu verlässliche Navigation auf dem Spielfeld. Er befindet sich in der unteren Ecke des Spielfeldes, wie die Abb. 1 zeigt. Weiter wird eine Winkelauslegung gleich dem Einheitskreis festgelegt.

Vom Eurobot-Komitee wurden die Teamfarben rot und gelb bestimmt[1]. Je nach Farbe bekommen die Roboter unterschiedliche Startplätze zugeteilt. Die rote Mannschaft startet in der linken oberen Ecke, wohingegen das gelbe in der gegenüberliegenden rechten Ecke sein Startfeld hat. Gekennzeichnet werden die Startpunkt mit einer Hand, die die Farbe der Mannschaft trägt (siehe Abb. 1).

2 Protokolle

Die Kommunikation baut auf insgesamt drei Protokollen auf, die folgend näher erläutert werden.

2.1 GoTo-Protokoll

Das GoTo-Protokoll dient hauptsächlich der Kommunikation mit dem Antriebsknoten. Die möglichen Befehle des Protokolls, sowie deren Aufbau sind in der Tabelle 1 ersichtlich.

[1] Nachzulesen im offiziellen Regelement http://www.eurobot.org/attachments/article/2/Eurobot2014_Rules_EN_Final_Version.pdf

Befehl	Parameter	Breite	Bemerkung	Sender	Empfänger
Goto	X-Position	12bit	• 1bit ≅ 1mm • Endposition, Standardabweichung 5mm	Kern	Antrieb
	Y-Position	12bit	• 1bit ≅ 1mm • Endposition, Standardabweichung 5mm		
	Winkel	10bit	• 1bit ≅ 1° • Endposition, Standardabweichung 1°		
	Soll-Geschwindigkeit	8bit	• 1bit ≅ 1% • Durch Regelung kann die Geschwindigkeit variieren		
	Barrieren-Flags	16bit	• optional • 2 bit pro Barriere (MSB + LSB) – 0 0: Kein Hindernis – 0 1: Hindernis (normal) – 1 0: Hindernis (forciert) – 1 1: Hindernis (forciert)		
Goto ACK	Keine Daten			Antrieb	Kern
Stop	Keine Daten			Kern	Alle
Extended Stop	Hindernis	1bit	• 0 ≅ Gegenstand; 1 ≅ Gegner	Kern	Antrieb
State Request	Keine Daten			Kern	Antrieb
State Response	Zeit	24bit	• 1bit ≅ 1ms • 0 → Ziel erreicht • 0xFFFFFF → Zeit unbekannt	Antrieb	Kern

Tabelle 1: GoTo-Protokoll Befehlssatz

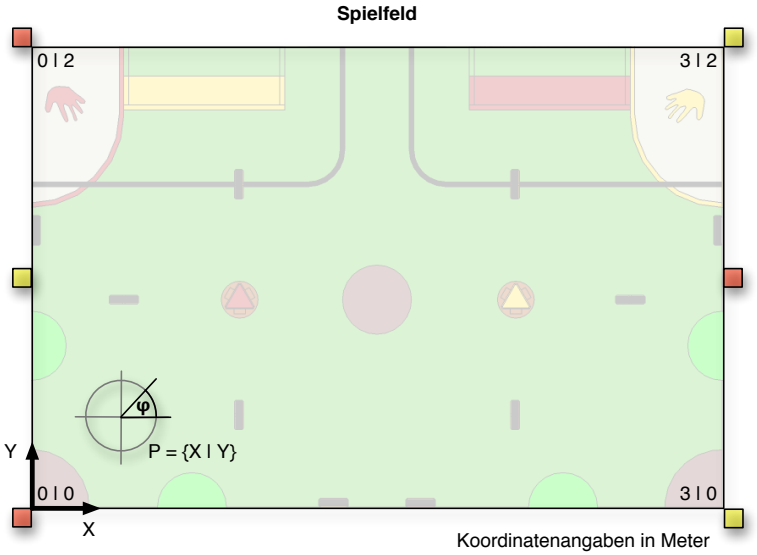

Abbildung 1: Spielfeld-Definition

GoTo-Befehl Der GoTo-Befehl dient der Antriebssteuerung, respektive Wegbestimmung. Wird vor dem erreichen der Endposition ein neuer GoTo-Befehl an den Antrieb gesendet, so wird dieser übernommen und der vorherige verworfen. Die genaue Wegfindung übernimmt die Antriebseinheit, ebenso die Regelung der Geschwindigkeit. Für das Beeinflussen der Wegfindung können zu den Grunddaten (Endposition, Geschwindigkeit und Endwinkel) noch Barrieren gesetzt werden. Die Barrieren, die mit einer ID identifiziert werden (ersichtlich in Abb. 2), werden vom Wegfindealgorithmus der Antriebseinheit gemieden[2].

Goto ACK-Befehl Mit dem Goto ACK-Befehl wird das Empfangen des zuvor erhalten GoTo-Befehls bestätigt.

Stop-Befehl Der Stop-Befehl löst ein definiertes Bremsen aus. Der anfällige zuvor erhaltene GoTo-Befehl wird verworfen.

Extended Stop-Befehl Der Extenden stop-Befehl beinhaltet im Gegensatz zum Stop-Befehl noch Informationen zum Stopp-Grund.

State Request-Befehl Ist ein reiner Polling-Befehl und dient der Kontrolle ob die gewünschte Position bereits erreicht wurde.

State Response-Befehl Folgt direkt auf den State request-Befehl und teilt dem Kernknoten die geschätzte verbleibende Zeit bis zum erreiche der Endposition mit.

[2]Es wird zwischen normal und forciert unterschieden. Beim normalen meiden der Barrieren versucht die Wegfindung einen Weg um die Barriere herum zu finden, kann dies aber nicht garantieren. Wird der forcierte Modus eingesetzt, so wird die Barriere zu 100% von der Wegfindung nicht passiert.

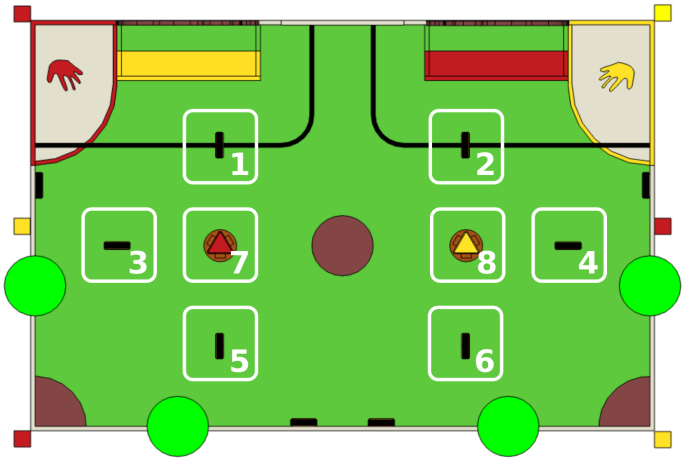

Abbildung 2: ID's der Barrieren

2.2 Estimated Location-Protokoll

Um den Transfer von Navigationdaten zu vereinheitlichen, wird ein entsprechendes Protokoll namens Estimated Location-Protokoll definiert. Der Aufbau ist der Tabelle 2 zu entnehmen.

Estimated Location request-Befehl Dieser Befehl dient lediglich der Anfrage von Positionsdaten und wird pollend eingesetzt.

Estimated Location response-Befehl Folgt direkt auf einen Estimated Location request-Befehl und beinhaltet alle relevanten Positionsdaten.

2.3 General Information Protocol

Zu Beginn einer Spielrunde ist es wichtig das alle Komponenten des Roboters in einem definierten Zustand gebracht werden. Dieser Zustand ist abhängig von der Teamfarbe, wie auch von der Anzahl zu erwartenden Gegner und Verbündeten. Diese Faktoren können sich von Spiel zu Spiel unterscheiden, weshalb zur Austausch dieser Informationen das General Information Protocol definiert wird. Die genaue Struktur des Protokolls ist der Tabelle 3 zu entnehmen.

Start Configuration Set-Befehl Mit diesem Kommando werden die 5 Informationen Teamfarbe, Anzahl Gegner, Anzahl Verbündete und Durchmesser der Gegner gesendet.

Start Configuration Confirm-Befehl Dieser Befehl dient lediglich der Bestätigung das alle Daten erfolgreich empfangen wurden.

Node Check Request Mit dem Kommando kann ein spezifischer Busteilnehmer (Knoten) auf seine Verfügbarkeit geprüft werden. Die ID der Nachricht referenziert dabei auf einen bestimmten Knoten und deshalb einmalig sein.

Befehl	Parameter	Breite	Bemerkung	Sender	Empfänger
Estimated Location Request	Keine Daten			Kern	Antrieb o. Navigation
Estimated Location Response	X-Position	12bit	• 1bit ≅ 1mm • Standardabweichung 5mm • 0xFFF wenn Position unbekannt	Antrieb, Extern o. Navigation	Kern o. Navigation
	Y-Position	12bit	• 1bit ≅ 1mm • Standardabweichung 5mm • 0xFFF wenn Position unbekannt		
	Winkel	10bit	• 1bit ≅ 1° • Standardabweichung 1° • 0x3FF wenn Position unbekannt		
	ID-Quelle	4bit	• Identifikation der Koordinatenquelle		

Tabelle 2: Estimated Location-Protokoll Befehlssatz

Befehl	Parameter	Breite	Bemerkung	Sender	Empfänger
Start Configuration Set	Teamfarbe	1bit	• 0 entspricht gelb • 1 entspricht rot	Kern	Antrieb/Navi
	Anzahl Gegner	2bit	• 0 entspricht 0 Gegnern • 1 entspricht 1 Gegner • 2 entspricht 2 Gegner		
	Anzahl Verbündete	1bit	• 0 entspricht keinem Verbündeten • 1 entspricht einem Verbündeten		
	Durchmesser Gegner 1	6bit	• 0-50cm in cm-Schritten • 0 entspricht keinem Gegner 1		
	Durchmesser Gegner 2	6bit	• 0-50cm in cm-Schritten • 0 entspricht keinem Gegner 2		
Start Configuration Confirm	Keine Daten			Antrieb/Navi	Kern
Node Check Request	Keine Daten			Kern	Antrieb/Navi
Node Check Response	Keine Daten			Antrieb/Navi	Kern

Tabelle 3: General Information Protocol Befehlssatz

Node Check Response Um die Verfügbarkeit zu bestätigen muss der Knoten eine entsprechende Nachricht quittieren.

3 CAN-Protokoll

Während des Betriebs können die in Tabelle 5 aufgeführten Kommunikationen stattfinden.

Die Priorität einer Nachricht ist durch den spezifischen Kommando-Indentifier gegeben. In der Tabelle 4 sind die Kommandos absteigend bezüglich ihrer Priorität sortiert.

Bezeichnung	Priorität	Identifier Binär	Identifier Hex
Emergency Shutdown	Notfall	000 0000 0001	0x001
Emergency Stop	Notfall	000 0000 0010	0x002
Stop Drive	Normalbetrieb	000 0000 0100	0x004
Goto XY	Normalbetrieb	000 0000 1000	0x008
Goto Confirm	Normalbetrieb	000 0000 1100	0x00C
Goto State Request	Normalbetrieb	000 0001 0000	0x010
Goto State Response	Normalbetrieb	000 0001 0100	0x014
Navi Position Request	Information	000 0100 0000	0x040
Navi Position Response	Information	000 1000 0000	0x080
Kalmann Position Request	Information	000 1100 0000	0x0C0
Kalmann Position Response	Information	001 0000 0000	0x100
Enemy 1 Position Request	Information	001 0100 0000	0x140
Enemy 1 Position Response	Information	001 1000 0000	0x180
Enemy 2 Position Request	Information	001 1100 0000	0x1C0
Enemy 2 Position Response	Information	010 0000 0000	0x200
Confederate Position Request	Information	010 0100 0000	0x240
Confederate Position Response	Information	010 1000 0000	0x280
Start Configuration Set	Information	010 1100 0000	0x2C0
Start Configuration Confirm	Information	011 0000 0000	0x300
Check Navi Request	Information	011 0100 0000	0x340
Check Navi Response	Information	011 1000 0000	0x380
Check Drive Request	Information	011 1100 0000	0x3C0
Check Drive Response	Information	100 0000 0000	0x400

Tabelle 4: Kommunikation Prioritäten

Bezeichnung	Protokoll	Befehl	Sender	Empfänger
Emergency Shutdown	GTP	Stop	Kern	Antrieb & Navigation
Emergency Stop	GTP	Extended Stop	Kern	Antrieb
Stop Drive	GTP	Stop	Kern	Antrieb
Goto XY	GTP	Goto	Kern	Antrieb
Goto Confirm	GTP	Goto ACK	Antrieb	Kern
Goto State Request	GTP	State Request	Kern	Antrieb
Goto State Response	GTP	State Response	Antrieb	Kern
Navi Position Request	ELP	Estimated Location Request	Kern	Navigation
Navi Position Response	ELP	Estimated Location Response	Navigation	Antrieb
Kalmann Position Request	ELP	Estimated Location Request	Kern	Antrieb
Kalmann Position Response	ELP	Estimated Location Response	Antrieb	Kern
Enemy 1 Position Request	ELP	Estimated Location Request	Kern	Navigation
Enemy 1 Position Response	ELP	Estimated Location Response	Navigation	Kern
Enemy 2 Position Request	ELP	Estimated Location Request	Kern	Navigation
Enemy 2 Position Response	ELP	Estimated Location Response	Navigation	Kern
Confederate Position Request	ELP	Estimated Location Request	Kern	Navigation
Confederate Position Response	ELP	Estimated Location Response	Navigation	Kern
Start Configuration Set	GIP	Start Configuration Set	Kern	Antrieb & Navigation
Start Configuration Confirm	GIP	Start Configuration Confirm	Antrieb & Navigation	Kern
Check Navi Request	GIP	Check Node Request	Kern	Navigation
Check Navi Response	GIP	Check Node Response	Navigation	Kern
Check Drive Request	GIP	Check Node Request	Kern	Antrieb
Check Drive Response	GIP	Check Node Response	Antrieb	Kern

Tabelle 5: CAN-Kommunikation

4 Anwendung CAN-Gatekeeper

Der CAN-Gatekeeper bietet eine vielzahl von Möglichkeiten, die in diesem Abschnitt erläutert werden[3].

4.1 Nachrichten Listener

Alle Module der Applikation die CAN-Nachrichten empfangen wollen, müssen sich im Gatekeeper eintragen. Diese Listener (Zuhörer) werden anschliessend beim Eintreffen einer entsprechenden Nachricht vom Gatekeeper benachrichtigt.

Den Module stehen zwei verschieden Arten von Listener zur Verfügung:

setQueueCANListener Die erste Möglichkeit besteht darin, auf eine spezifische Nachrichten-ID[4] eine Queue einzutragen. Die Definition der Queue muss dabei im entsprechenden Modul erfolgen[5]. Beim Eintreffen einer Nachricht wird die Queue nach dem FIFO-Prinzip mit den Daten befüllt.

setFunctionCANListener Neben der Queue kann als zweite Möglichkeit auch eine Callback-Funktion im Gatekeeper eingetragen werden. Diese Funktion[6] wird beim Eintreffen der richtigen Nachricht direkt vom Gatekeeper ohne Verzögerung aufgerufen. Zu Beachten ist, dass sie weder blockierend noch zu lange ausfallen darf (keine zeitintensiven Algorithmen, die eine Ausführungszeit über).

Die Anzahl der nötigen Listener-Plätze muss mit dem define `CAN_LISTENER_BUFFER_SIZE`in der Datei `CANGatekeeper.h` eingestellt werden.

4.1.1 Datentyp `CAN_data_t`

Alle empfangenen Daten werden in der `union CAN_data_t` abgelegt und entweder via Zeiger an die Listener-Funktion (siehe Abschnitt 4.1.2) weitergeleitet und/oder in die Listener-Queue kopiert. Die `union` ist dabei wie im Listing 1 ersichtlich aufgebaut.

```
1  /**
2   *  \brief rx data-handler
3   */
4  typedef union
5  {
6      /* GotoXY data-set */
7      struct
8      {
9          uint16_t goto_x; /*!< x-position */
10         uint16_t goto_y; /*!< y-position */
11         uint16_t goto_angle; /*!< angle in degree */
12         uint8_t goto_speed; /*!< speed in % */
13         uint16_t goto_barrier; /*!< barrier-flags */
14     };
```

[3] Es gilt zu beachten, dass der folgend vorgestellte Gatekeeper auf der Basis vom FreeRTOS aufgebaut wurde und deshalb nicht generisch einsetzbar ist.

[4] ein Übersicht über die möglichen Nachrichten bietet die Tabelle 5 auf Seite 9

[5] die Queue muss den Datentyp `CAN_data_t` aufnehmen können

[6] Die Funktion muss die Form `void foo(uint16_t id, CAN_data_t* data)` aufweisen

```
15
16      /* ELP data-set */
17      struct
18      {
19          uint16_t elp_x; /*!< x-position */
20          uint16_t elp_y; /*!< y-position */
21          uint16_t elp_angle; /*!< angle in degree */
22          uint8_t elp_id; /*!< id of the information source */
23      };
24
25      /* GIP data-set */
26      struct
27      {
28          uint8_t gip_color; /*!< teamcolor (0=yellow; 1=red) */
29          uint8_t gip_enemy; /*!< enemey quantity (0=0 enemey; 1=1 enemy; 2=2
                enemies) */
30          uint8_t gip_confederate; /*!< confederate quantity (0=0 confederate;
                1=1 condeferate) */
31          uint8_t gip_enemy_1_size; /*!< 0-50cm with 1cm step-size */
32          uint8_t gip_enemy_2_size; /*!< 0-50cm with 1cm step-size */
33      };
34
35      /* State Response data-set */
36      uint32_t state_time; /*!< time until the roboter stops */
37
38      /* Extended stop data-set */
39      uint8_t stop_obstacle_id; /*!< id of the obstacle */
40
41  } CAN_data_t;
```

Listing 1: Datantyp CAN_data_t

4.1.2 Funktionspointer `*CAN_function_listener_t`

Um erfolgreich eine Funktion im Gatekeeper eintragen zu können, muss eine wie im Listing 2 ersichtliche Funktion definiert werden.

```
1  /**
2   * \brief function pointer for CAN-listeners
3   */
4  typedef void (*CAN_function_listener_t) (uint16_t, CAN_data_t*);
5
6  /**
7   * \brief Example
8   */
9  void foo(uint16_t id, CAN_data_t* data)
10 {
11         /* write your code here */
12 }
```

Listing 2: Funktionspointer `*CAN_function_listener_t`

4.2 Anwendung

4.2.1 Initialisierung

Für das erfolgreiche Einsetzten des Gatekeepers sind nur zwei Schritte nötig:

1. Bevor das Scheduling des Taskmanagers beginnt, müssen in der Initialisationsphase alle Listener im Gatekeeper eingetragen werden (siehe dazu den vorherigen Abschnitt). Wichtig: Der Gatekeeper selber darf zu dieser Zeit noch **nicht** initialisiert sein!

2. Als letztes Modul der Applikation muss der Gatekeeper mit der Funktion `initCANGatekeeper` initialisiert werden.

Die Abbildung 3 zeigt die Anwendung zur Verdeutlichung anhand von zwei Beispielmodulen.

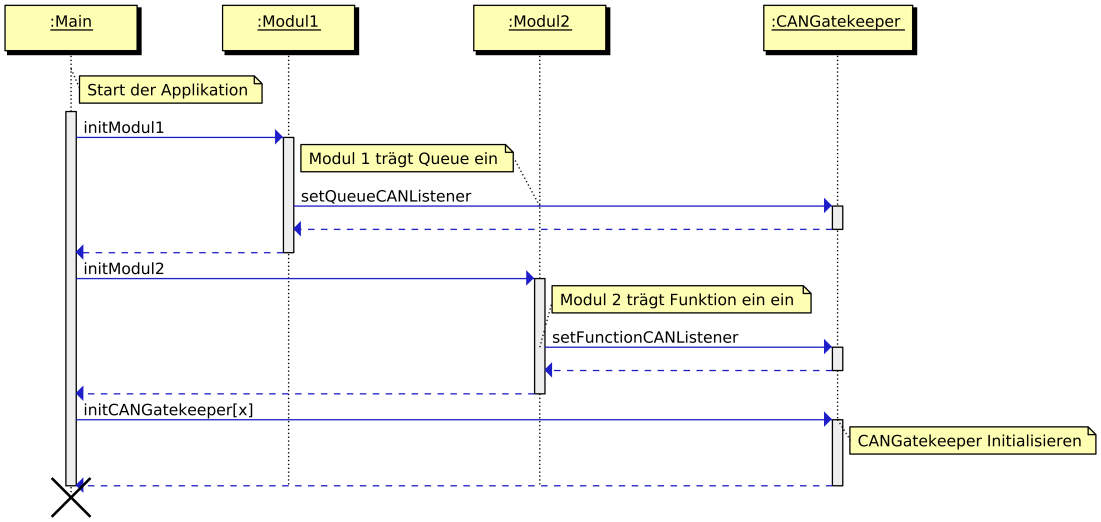

Abbildung 3: Anwendung des `CANGatekeepers` Moduls

4.2.2 Betrieb

Es kann zwischen fünf verschieden Kommunikationsabläufe auf dem CAN-Bus unterschieden werden.

Position anfahren Mit Hilfe des GTP kann der Roboter von A nach B navigiert werden. Dabei stehen dem Kernknoten zum einen der Goto XY-Befehl zur Steuerung um zum

anderen der State Request-Befehl zur Verfügung[7]. Der Ablauf der Kommunikation ist der Abbildung 4 zu entnehmen.

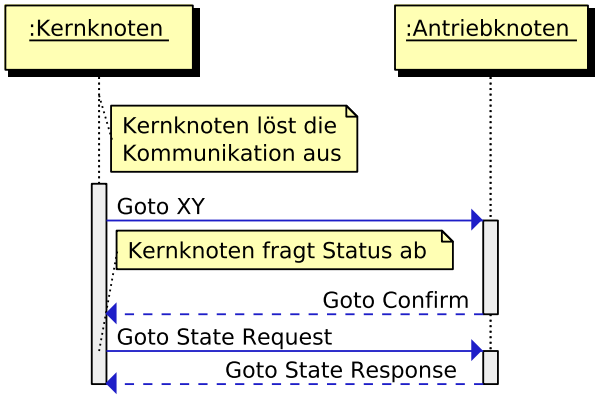

Abbildung 4: Kommunikationsablauf Position anfahren

Notaus-Pilz Wird der Notaus-Pilz betätigt, so nimmt das System einen zuvor definierten Zustand an. Dazu wird das Emergency Shutdown-Kommando broadcast an alle Knoten des Bus versendet. Der Vorgang ist in der Abbildung 5 ersichtlich.

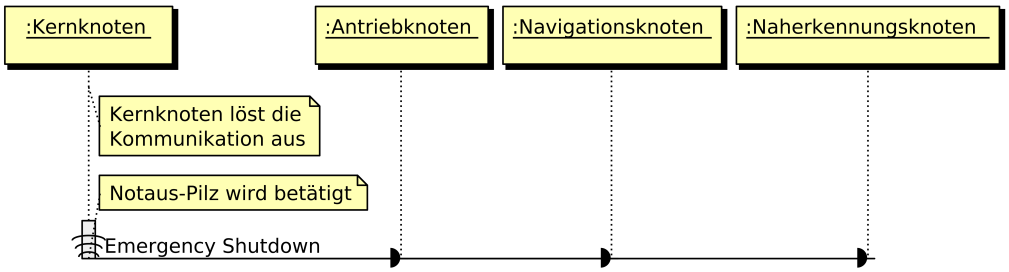

Abbildung 5: Kommunikationsablauf Notaus-Pilz

[7] siehe Abschnitt 2.1 *GoTo-Protokoll* auf Seite 1

Hindernis Ein Hindernis kann entweder ein gegnerischer Roboter oder ein Element des Spielfelds sein. Wird ein solches detektiert, so hat dies einen sofortigen Stopp zu Folge. Der dazu nötige Befehl Emergency Stop wird vom Kernknoten an die Antriebseinheit gesendet.

Navigation Für die Kalman-Filterung der Koordinationsdaten der Navigationseinheit müssen die Rohdaten zum Antriebskonten transferiert werden. Dafür sendet der Kernknoten ein entsprechendes Request, das von der Navigation entgegen genommen und beantwortet wird. Der Antriebskonten empfängt den Response und erhält auf diesen Weg die benötigten Daten. Der Ablauf ist in der Abbildung 6 aufgezeigt.

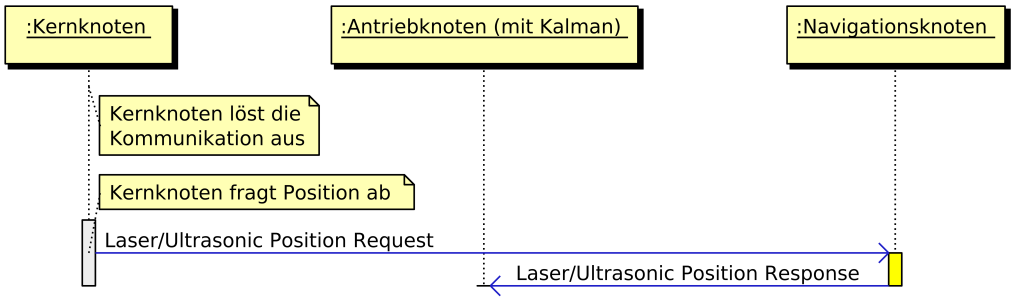

Abbildung 6: Kommunikationsablauf Navigation

Position Die genaue Position des Roboters wird vom Kalman-Filter errechnet. Die Daten müssen anschliessend zum Kernknoten transferiert werden. Der Kommunikationsablauf wird in der Abbildung 7 grafisch dargestellt.

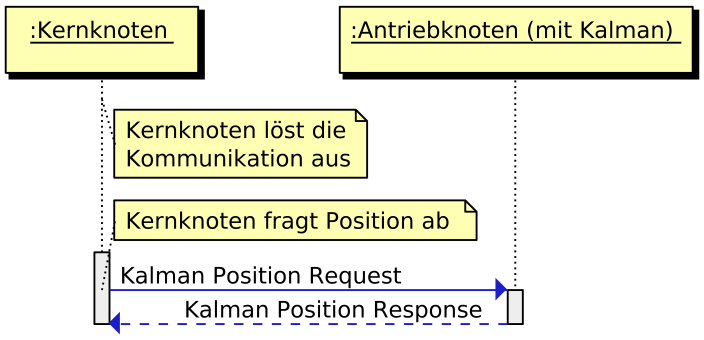

Abbildung 7: Kommunikationsablauf Position

5 System

Alle in der Tabelle 5 aufgeführten Kommandos können jederzeit auf dem Bus vorkommen. Es ist von immenser Wichtigkeit, dass alle Busteilnehmer entsprechend darauf wie definiert reagieren.

5.1 Startup

Während das System hochfährt, wird der in Abb. 8 definierte Ablauf abgearbeitet. Wird eine Komponente nicht nach den Spezifikationen eingehalten, so hat dies ein Blockieren des Systems zur Folge.

5.2 Match

Der Kommunikationsablauf während eines Match unterscheidet sich nur unwesentlich zu dem des Startup-Vorgangs. Die ELP-Polling Aktivitäten sind nachwievor vorhanden. Dazu kommen noch stochastisch einige GTP Befehle, die aber Buskapazität nur bedingt belasten und daher keine Probleme darstellen.

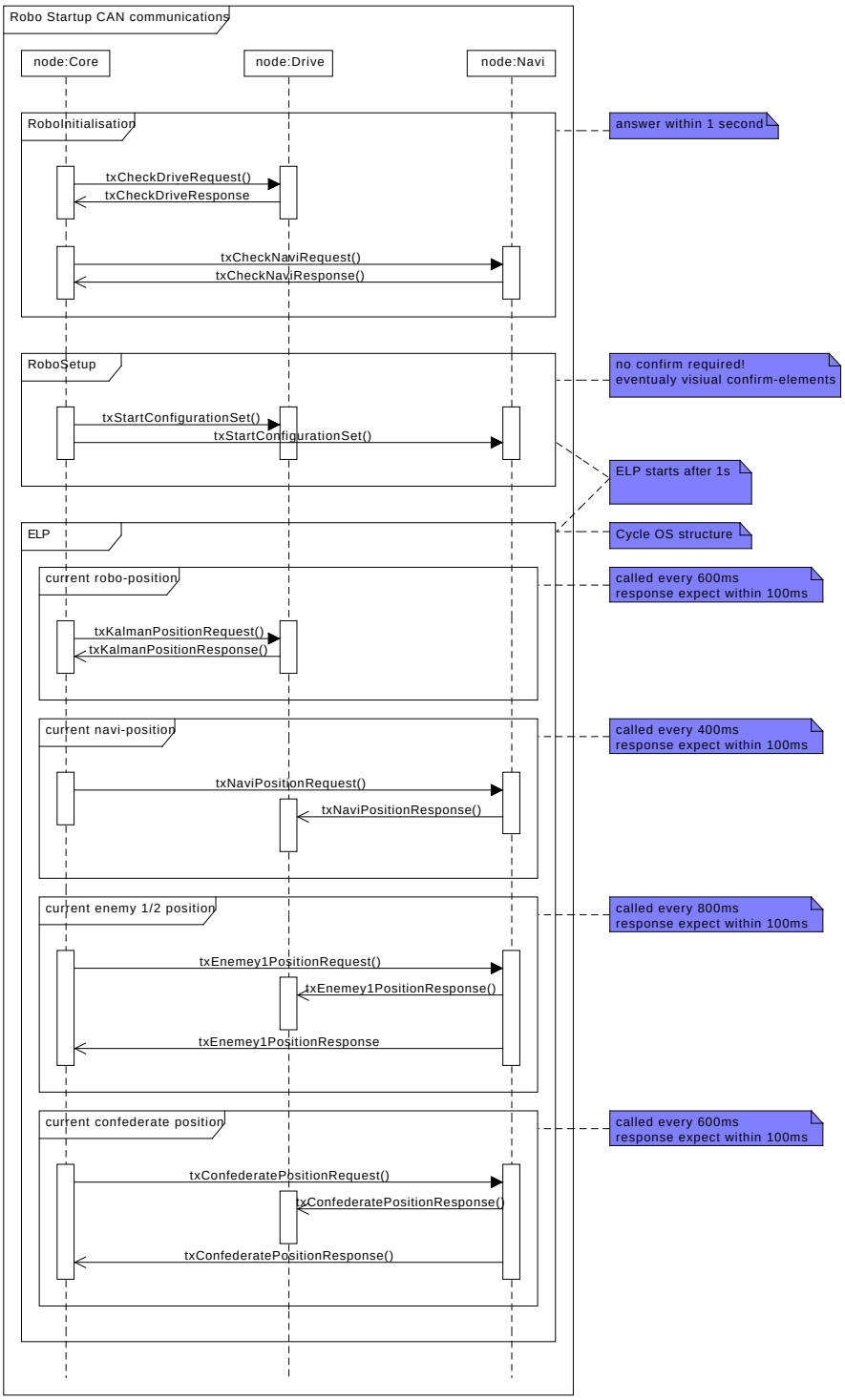

Abbildung 8: Kommunikationsablauf während Startup

6 Verbesserungsvorschläge Gatekeeper

Trotz sorgfältiger Entwicklung ist der Gatekeeper noch nicht am Ende seiner Möglichkeiten. Durch Anpassungen, die folgend erläutert werden, könnte seine Effizienz noch gesteigert werden.

- Für das Eintragen von neuen CAN-Listener in den Hardware-Filtern des CAN-Controllers wird aktuell ein Array verwendet, das persistent Arbeitsspeicher benötigt obwohl es nur zu Beginn benötigt wird. Ein direktes Beschreiben der Filter-Register könnte diese Speicherverschwendung überflüssig machen.
- Beim senden wird, sollten alle der drei zur Verfügungen stehenden Mailboxen (Register zum Zwischenspeichern von zu senden CAN-Nachrichten) besetzt sein, in einer `while`-Schlaufe maximal 50ms gewartet. Durch das einsetzen des Transmit-Complete-Interrupts könnte dies weggelassen und die Ausführungsgeschwindigkeit des Codes gesteigert werden.