Inhaltsverzeichnis

II Abbildungsverzeichnis

III Abkürzungsverzeichnis

CT	-	Zeitschrift c't (**c**omputing **t**oday)
GUI	-	**G**raphical **U**ser **I**nterface
IP	-	**I**nternet **P**rotocol
KI	-	**K**ünstliche **I**ntelligenz
MVP	-	**M**odel **V**iew **P**resenter
POV	-	**P**oint **O**f **V**iew (Steuerkreuz)
RCC	-	**R**obot **C**ontrol **C**lient
RCS	-	**R**obot **C**ontrol **S**erver
RCU	-	**R**obot **C**ontrol **U**tility
SSL	-	**S**mall **S**ize **L**eague

1 Einleitung

Diese Studienarbeit ist im Zuge des RoboCup-Projekts der Tigers Mannheim an der DHBW Mannheim entstanden. Ziel dieses Projekts ist die Teilnahme am RoboCup 2011 in der Small Size League (SSL).

Der RoboCup bezeichnet unter anderem die Weltmeisterschaft im Roboterfußball. Des weiteren umfasst der RoboCup auch die *Rescue*- und die *At-Home-League*, wo die Roboter entsprechend der Anforderungen spezielle Tests durchlaufen müssen. Seit 1997 wird dieser internationale Wettbewerb jährlich ausgetragen. Die Motivation des RoboCup ist, die Forschung auf dem Gebiet der künstlichen Intelligenz und die Entwicklung autonomer mobiler Systeme zu fördern. Außerdem wurde zum Ziel erklärt, 2050 den amtierenden menschlichen Fußballweltmeister mit einem Roboterteam zu schlagen [1].

Die Tigers Mannheim, RoboCup-Team der DHBW Mannheim, beabsichtigen an der Small Size League des RoboCups teilzunehmen. In dieser Liga spielen 2 Teams à 5 Roboter gegeneinander. Diese haben eine zylindrische Form und sind maximal 15 cm groß. Gesteuert werden sie über eine zentrale künstliche Intelligenz, deren Input das Echtzeitkamerabild des Spielfelds ist.

Unter die Aufgaben der Zentralsoftware der Tigers Mannheim fällt unter anderem, die Kamerabilder zu interpretieren, basierend auf dem aktuellen Spielgeschehen Schlüsse zu ziehen, die Roboter mit entsprechenden Befehlen zu steuern und durch diesen Verarbeitungsaufwand entstehende Latenzen zu kompensieren. Entsprechend dieser verschiedenen Aufgabenfelder wurde die Zentralsoftware modular aufgebaut [2].

Im Zuge der Entwicklung der Roboter und der zugehörigen Software stellte sich heraus, dass für verschiedene Zwecke ein Modul zur manuellen Steuerung der Roboter, eine sogenannte Robot Control Utility (RCU), hilfreich wäre. Einerseits würde sie bei Demonstrationen die Anschaulichkeit verbessern. Andererseits ist auch das Testen beispielsweise der eigenen KI mit manuell gesteuerten Robotern als Gegner denkbar.

2 Prototyp der RCU

Das Projekt der Tigers Mannheim ist inzwischen an dem Punkt angelangt, an dem der erste funktions- und einsatzfähige Roboter, der sogenannte Tigerbot, konstruiert wurde. Damit zuvor schon Ergebnisse im Bereich der Programmierung vorgezeigt werden konnten, wurden einfachere Roboter aus fertigen Bausätzen verwendet (CT-Bots). Für diese Roboter war auch schon ein Prototyp der RCU vorhanden. Dieser Prototyp ist wie auch alle anderen Softwaremodule der Zentralsoftware in Java geschrieben. Der Tigerbot ist jedoch aufgrund seiner ausgefeilter Konstruktion und zusätzlichen Funktionalitäten komplexer zu steuern als die CT-Bots.
Schon beim Prototypen der RCU wurde die Idee umgesetzt, die RCU mittels einer Server-Client-Struktur an die Zentralsoftware anzubinden, damit die Steuerung der Roboter mit einem beliebigen Rechner möglich ist, vorausgesetzt er ist über ein Netzwerk mit dem Zentralrechner verbunden. Außerdem konnte man schon die Roboter mit einem Gamepad steuern, was auch in der aktuellen RCU umgesetzt wurde.

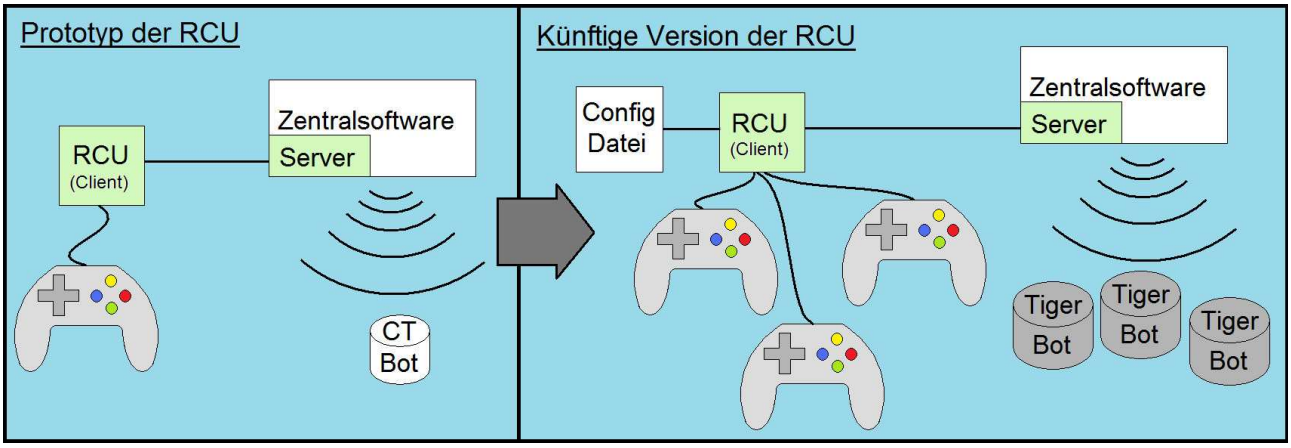

Abbildung 1: Vergleich der Anforderungen an die RCU-Versionen

Wie in Abbildung 1 dargestellt, wurde im Prototypen noch nicht umgesetzt, dass man mit nur einem Client der RCU mehrere Roboter mit mehreren Gamepads gleichzeitig steuern kann. Die verwendeten Befehle für die Steuerung des CT-Bots müssen für den Tigerbot aufgrund der unterschiedlichen Funktionalitäten abgeändert und erweitert werden. Des Weiteren ist im Gegensatz zu Prototypen die Belegung der Tasten und Achsen auf dem verwendeten Gamepad mit den verschiedenen Befehlen variabel gestaltet worden. Der Benutzer der RCU kann eine solche Belegung in einer Konfigurationsdatei speichern und laden.

Wegen dieser erweiterten Anforderungen an die RCU, angesichts der Softwarestruktur des Prototypen und im Hinblick auf die zu erhaltende Codequalität der RCU wurde es als sinnvoll erachtet, nicht auf den Prototyp aufzusetzen sondern die RCU von Grund auf neu zu entwerfen und zu implementieren.

3 Planung und Entwurf

3.1 Klassendesign & MVP

Während der Planungs- und Entwurfsphase wurden Überlegungen in Hinsicht auf den Aufbau des Clients und seine Funktionsweise angestellt. Die Klassen sollten treffend benannt werden und so fein wie nötig aufgeteilt werden. Angelehnt an die Struktur aller Softwaremodule in der Zentralsoftware der Tigersmannheim, findet das Model-View-Presenter Prinzip auch in der RCU Anwendung. Dieses Programmierparadigma sieht die strikte Trennung des Datenmodels von der Benutzereingabe (View) vor. Die Kommunikation dieser beiden Programmteile läuft ausschließlich über den Presenter ab, er behandelt alle Benutzereingabe in der View und Veränderungen im Datenmodell. Der View ist die grafische Oberfläche (graphical user interface – GUI), mit Hilfe der Benutzer die RCU steuert und die Verwaltung der Konfigurationen (siehe Kapitel 4.1.4) verwendeten kann. Das Model beinhaltet das Auslesen des Controllers, das Übermitteln der Daten an die Zentralsoftware und das Konfigurationsverwaltung.

Schon in der Entwurfsphase wurde deutlich, dass der Presenter unterteilt werden muss. Als übergeordnete Komponente wurde der RCUPresenter definiert, seine Aufgabe besteht in der Steuerung des Clients, der Beherrschung der einzelnen Controller[1] und dem Initialisieren der grafischen Oberfläche. Die Controller werden durch den ControllerPresenter repräsentiert. Der Begriff ControllerPresenter ist hier eine andere Bedeutung zugewiesen, als man es in Bezug auf das Programmierparadigma MVC und im speziellen MVP gemeinhin kennt. Der ControllerPresenter ist hier ein Presenter, der sich um die Datenverarbeitung eines angeschlossenen Controllers kümmert. In diesem untergeordneten Presenter wird das Datenmodel und ein Teil der GUI für jeden Controller erzeugt. Zum Datenmodel jedes Controllers gehört die Tasten- und Befehlserkennung (GamePadReader und GamePadInterpreter), sowie das Senden der Daten an die Zentralsoftware (ActionTranslator und Actionsender). Von den ControllerPresenter losgelöst soll der ConfigManager arbeiten, d.h. alle Controller greifen auf die gleiche Instanz der Verwaltung der Konfigurationen zu. Auf den Aufbau und die Funktionsweise der grafischen Oberfläche wird in Abschnitt 3.3 weiter eingegangen.

Die Segmente Model und View kommunizieren nie direkt miteinander, jeder Informations-austausch läuft entweder über den RCUPresenter oder die ControllerPresenter. Wenn der Benutzer in der GUI beispielsweise den Start/Stop-Button drückt, geht dieses Ereignis an den RCUPresenter

[1] Controller sind die Tastatur und alle anderen Steuergeräte (Gamepad, Joystick usw.)

und dieser leitet es an das Model weiter (Überwachung der Controllerkomponenten und Beginn der Datenübertragung). Das Ziel dieser strengen Trennung ist die komplette Abkopplung der View vom Model und andersherum. Damit kann erreicht werden, dass eine der beiden Komponenten ausgetauscht wird und bei korrekter Implementierung weiter mit dem jeweils bestehen gebliebenen Teil arbeiten kann. Bei der RCU bezieht sich das MVP-Prinzip allerdings auf die übersichtliche Aufteilung der Softwaremodule.

3.2 Programmablauf

Zur Planung der Benutzersteuerung und dem Ablauf der einzelnen Klassen wurde in der Entwurfsphase ein Programmablaufplan erstellt (siehe Anhang). Dieser Plan sollte uns zum einen als Unterstützung im Klassendesign dienen und zum anderen den späteren, geplanten Programmablauf festlegen. Der Programmablaufplan enthält insgesamt fünf Aktionen, die der Benutzer auslösen kann, dazu gehört der Programmstart, die Roboterwahl, das Einstellen der Tastenbelegung und Applikationsstart bzw. -stop. Unter diesen Punkten wurde möglichst einfach und ohne Programmcode der Ablauf der Aktion beschrieben. Im Unterschied zur üblichen Verwendung und Erstellung des Programmablaufplans wurde für diese Arbeit eine textuelle Beschreibung angefertigt, um nicht erst die Suche und Einarbeitung für ein geeignetes Zeichentool nötig zu machen.

Am Beispiel der Aktion „Tastaturbelegung eingeben“ (Abbildung 2) soll der Plan kurz erklärt werden. Diese Aktion definiert den Ablauf, den das Programm beim Klicken eines Textfeldes durch den Benutzer durchläuft. Wie bereits erwähnt, wurde kein Programmcode verwendet und die Umschreibungen so allgemein wie möglich gehalten. Die Formulierung sind so gewählt, dass auch ein externes Teammitglied die Anweisungen verstehen und unsere Planung nachvollziehen kann. Unter den Punkten „Meldung weist auf...“ oder „Abbruch der Überwachung bei“ stehen jeweils zwei Unterpunkte, die verschiedene Möglichkeiten zur Kommunikation mit dem Benutzer darstellen. Die Überwachung soll also entweder per Maus oder nach einer bestimmten Zeit abbrechen.

Tastenbelegung eingeben

- User clickt auf Textfield → keine Eingabe möglich (Felder nicht disabled)
- Meldung weist auf Drücken des gewünschten Button hin
    1. Popup
    2. Statuszeile
- Überwachung aller Controller Components
- Bei Wert != 0 → Zuweisung Befehl/Taste
- Abbruch der Überwachung bei
    - Popup: per Mouse beendet
    - Zeit vergangen
- Textfeld füllen

Abbildung 2: "Tastenbelegung eingeben" aus dem Programmablaufplan

3.3 Funktionsweise der GUI

Die grafische Oberfläche der RCU besteht aus vier wesentlichen Teilen: eine Menüleiste am oberen Rand, ein Buttonleiste zur Programmsteuerung (Start/Stop, Exit und ClearLogging), die ControllerPanel im Fenster zentriert und ein Loggingbereich am unteren Rand. Diese Aufteilung bildet eine übersichtliche Anordnung der grafischen Komponenten und die Trennung von unterschiedlichen Eingabe- oder Ausgabemöglichkeiten. Als Layout wurde für alle Komponenten das so genannte MIG-Layout verwendet. Mit diesem Layoutpattern lassen die Elemente für den Entwickler leicht anlegen und organisieren und es besitzt ein sehr gutes Verhalten gegenüber Größenänderungen.

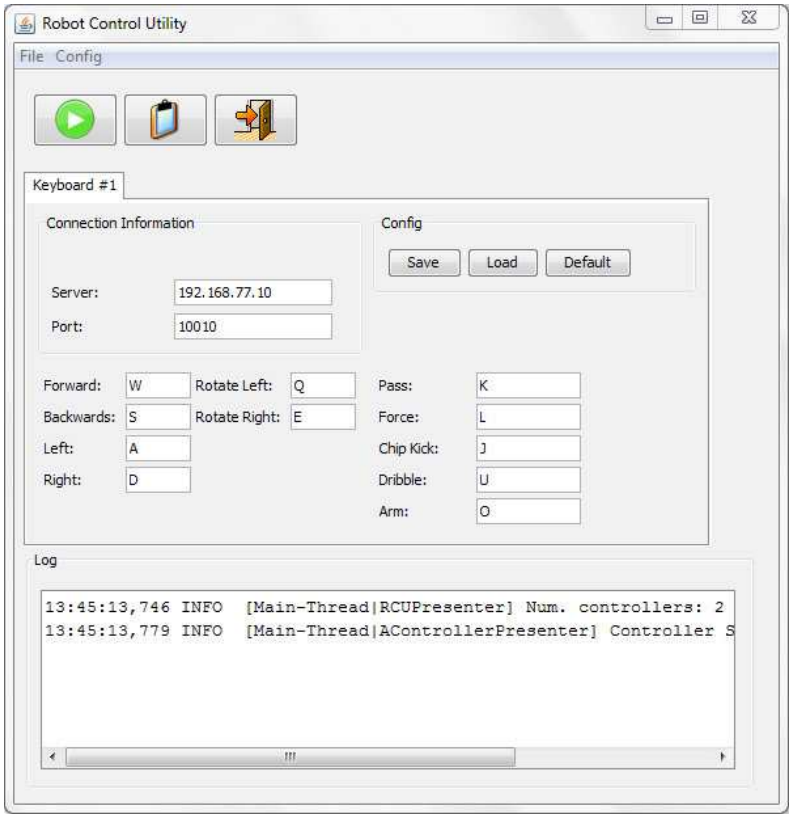

Abbildung 3: RCU Mainframe

Die Menüleiste am oberen Rand des Fenster dient im aktuellen Entwicklungsstand nur zum Schließen des Clients, allerdings wurde vorsorglich die unterschiedlichen Reiter „File“ und „Config“ angelegt. Unter dem Menüpunkt „File“ lassen sich weitere Funktionalitäten zur Steuerung einbinden, der Punkt „Config“ soll später das Skillsystem beinhalten.

Das Loggingfeld im unteren Bereich gibt dem Benutzer und Entwickler wichtige Rückmeldung, ob Aktionen erfolgreichen abliefen oder Rückschlüsse über Fehler. Dabei ist für den Benutzer nur die

Textmeldung interessant, für den Entwickler wird zusätzlich ein Zeitstempel und der Thread- und Klassenname angezeigt. Um das Loggingfeld zwischendurch zu leeren, steht der ClearLogging-Button in der oberen Leiste zur Verfügung.

Abbildung 4: RCU ControllerPanel

Der Hauptaugenmerk liegt auf den ControllerPanels. Um die unbestimmte Anzahl der Controller darzustellen, werden ein JTabbedPane und mehrere JPanel verwendet. Für jeden erkannten Controller wird ein neuer Reiter im JTabbedPane angelegt und in diesem ein ControllerPanel angezeigt. Zur Unterscheidung werden die Controllernamen in den Reiter angezeigt, somit hat der Benutzer einen ungefähren Überblick über die erkannten Eingabegeräte. Bisher unterstützt die RCU zwei Controller und die Tastatur, d.h. diese drei bekommen einen besonderen Namen und werden durchnummeriert. Ist ein Controller nicht erkannt wurden, erscheint als Name nur der Controllertyp, meist „Gamepad“. Die Controllerpanel sind in sich aufgeteilt in weitere vier Bereiche: ConnectionInformation, Movement, Action und Config. Diese vier Komponentgruppierungen stellen controllerspezifische Einstellungen dar.

In der ConnectionInformation befinden sich zwei Textfelder zur Eingabe von Serveradresse und Portnummer. Die Serveradresse dient zum Verbindungsaufbau zur Zentralsoftware und mittels der Portnummer wählt man den gewünschten Bot, für die CT-Bots die 1000x und für die TigerBots die 1001x (x jeweils zwischen 0..9).

Das Movement- und Actionpanel wird verwendet, um die Tastenbelegung für die Befehle festzulegen, die Aufteilung dient dabei nur zur besseren Übersicht. Durch Labels werden die Textfelder gekennzeichnet, um die Befehle zuzuordnen. Die Textfelder sind ausgeschaltet, d.h. man kann den Feldinhalt nur markieren, aber nicht etwas hinein schreiben. Nach dem Anklicken eines

dieser Textfelder fordert das Loggingfeld zum Betätigen der gewünschten Komponente auf und in der Titelzeile erscheint ein Counter, der die Zeit bis zum automatischen Abbrechen signalisiert. Mehr zu dieser Routine im Kapitel 4.2.3. Hat der Benutzer eine Komponente gedrückt, erscheint der Bezeichner im Textfeld, damit ist die Komponente diesem Befehl zugeordnet.
Im ConfigPanel kann der Benutzer die Konfigurationen verwalten (siehe Kapitel 4.1.4) der RCU. Nach Anklicken des Save- bzw. Loadbuttons wird ein so genannter FileChooser geöffnet. In diesem Fenster kann die gewünschte Datei zum Laden oder Speichern ausgewählt werden. Beim Speichern der Konfiguration kann entweder eine bestehende Daten überschrieben werden oder eine neue Datei wird angelegt. Zur besseren Bedienbarkeit wird auf Vorhandensein und Korrektheit der Dateiendung überprüft. Ist diese falsch oder hat sie der Benutzer vergessen, ergänzt das Programm den Namen automatisch um die Endung. Zur Konfigurationsverwaltung gehört auch der Button Default. Je nach Controllertyp (Keyboard oder Gamepad) wird damit eine intern abgespeicherte Konfiguration, die erfahrungsmäßig am besten geeignet ist, geladen und in den Textfeldern angezeigt.

4 Implementierung

In diesem Kapitel wird erläutert, wie sich die Anwendung RCU auf die verschiedenen Module und Hardwaregeräte aufteilt. Des weiteren werden besondere Aspekte der Implementierung der RCU näher beleuchtet. Schwerpunkte sind dabei die Client-Server-Verbindung, die Serialisierung von Objekten und der Zugriff auf die Ausgaben des Controllers.

4.1 Architektur

4.1.1 Verteilung der RCU

Die Robot Control Utility umfasst ein Softwaresystem, welches einer Server-Client-Struktur unterliegt. Der Großteil des Arbeitsaufwands dieser Studienarbeit liegt auf dem Client der RCU, während das Servermodul in der Zentralsoftware abgesehen von einigen Modifikationen auf den Server des Prototyps aufbaut.

Das Verteilungsdiagramm in Abbildung 5 zeigt, wie sich die Software der RCU auf verschiedene Geräte verteilt.

Um die Tigerbots steuern zu können wird genau ein Rechner mit der Zentralsoftware, samt Robot Control Server benötigt. Dieser ist über ein Netzwerk mit einem oder mehreren Rechnern verbunden, auf denen der RCC läuft. An jedem dieser Rechner wiederum sind ein oder mehrere Controller angeschlossen.

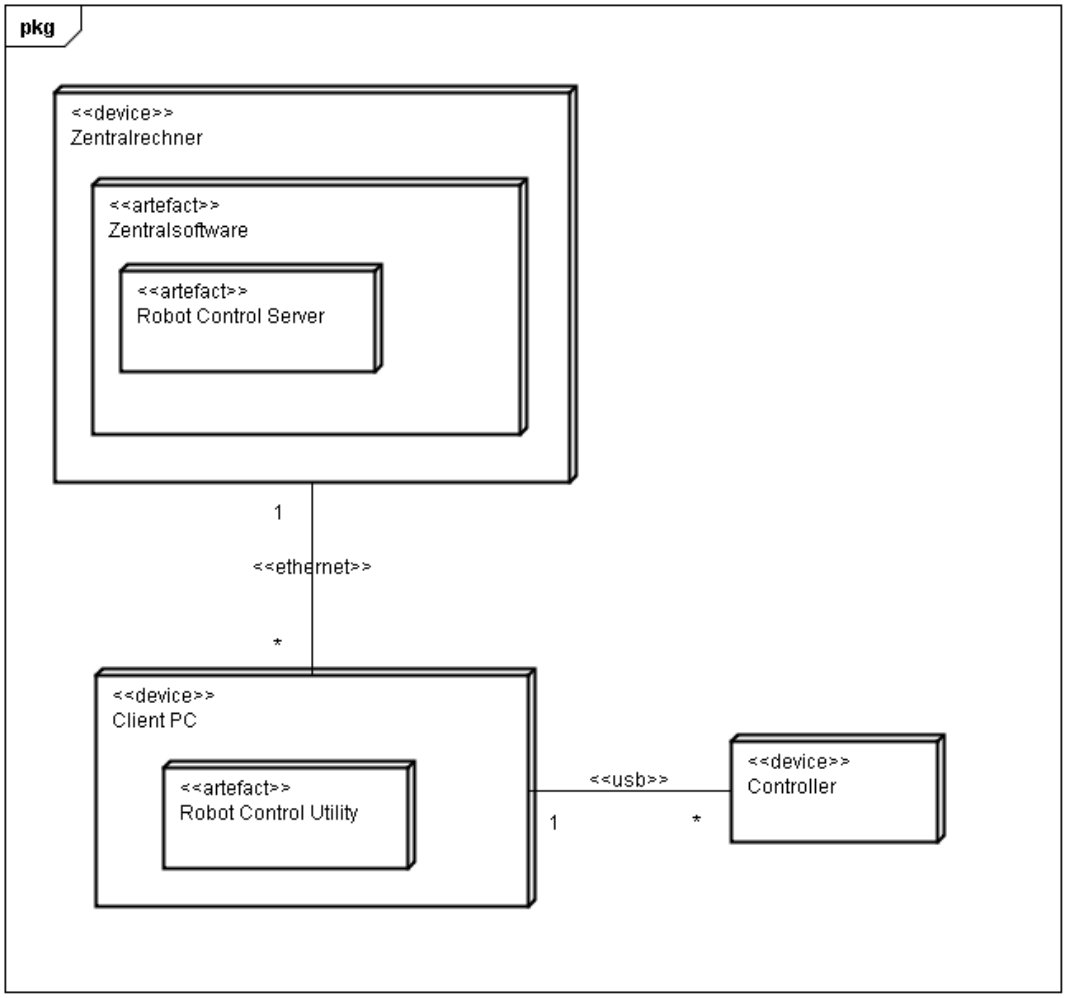

Abbildung 5: Verteilungsdiagramm der RCU

4.1.2 Client-Server-Vebindung

Die Verbindung zwischen dem RCC und dem RCS ist eine Client-Server-Verbindung. Dies wurde auf beiden Seiten mit Java Sockets realisiert.Der RCS läuft als Modul der Zentralsoftware auf mehreren Threads. Auf jedem Thread hört der Server einen festgelegten Port ab. Jeder der Ports entspricht einem verfügbaren Tigerbot.

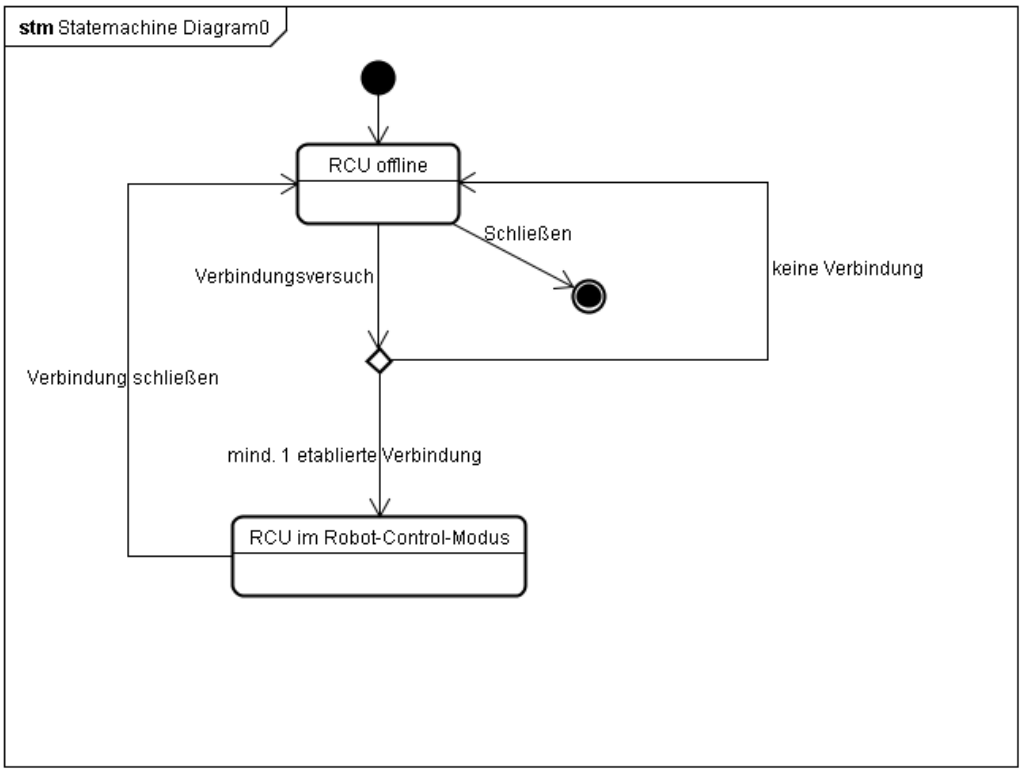

Abbildung 6: Zustandsdiagramm der Client-Server-Verbindung der RCU

Der RCC versucht bei einem Verbindungsaufbau alle Clientsockets mit dem entsprechenden Serversocket zu verbinden. Dies gelingt nur, wenn einerseits mindestens genauso viele Server- wie Clientsockets initialisiert wurden und andererseits jeder Clientsocket mit korrekter IP-Adresse und korrektem Port versehen wurde. Wenn nach einem Verbindungsaufbau nicht alle Clientsockets verbunden werden konnten, werden dennoch die etablierten Verbindungen aufrechterhalten. Der RCC geht dann in den Control-Robot-Modus, das bedeutet, der Benutzer kann nun über die etablierten Verbindungen die Roboter mittels Controller steuern. Alle Clientsockets, deren Verbindungsversuch fehlgeschlagen ist, werden in diesem Modus ignoriert. Weitere Verbindungen aufbauen, IP- oder Port-Einstellungen vornehmen und ähnliches ist jedoch erst wieder möglich, wenn der RCC den Control-Robot-Modus wieder verlassen hat.

Die Kommunikation zwischen Client und Server beläuft sich lediglich auf Nachrichten vom Client zum Server. Datenaustausch vom Server zum Client ist nicht vorgesehen und wurde auch nicht implementiert.

Schließt einer der beiden die Verbindung, wird beim jeweils anderen eine entsprechende Exception geworfen. Diese wird abgefangen und so wird auch auf der anderen Seite die Verbindung sauber geschlossen.

4.1.3 Serialisierbare Objekte

In der RCU bietet es sich an bestimmten Stellen an, serialisierbare Objekte zu verwenden. Beim Serialisieren wird das betreffende Objekt in einen binären Datenstrom umgewandelt. Dieser Datenstrom kann dann, wie in Abbildung 8 dargestellt, beispielsweise in eine Datei geschrieben oder in einem Netzwerk zu einem Empfänger gesendet werden. [3]

Um ein serialisiertes Objekt nach Auslesen einer Datei oder nach dem Empfang über das Netzwerk wieder als Objekt nutzbar zu machen, muss man es deserialisieren. Der Datenstrom wird wieder in das Ursprungsobjekt umgewandelt.

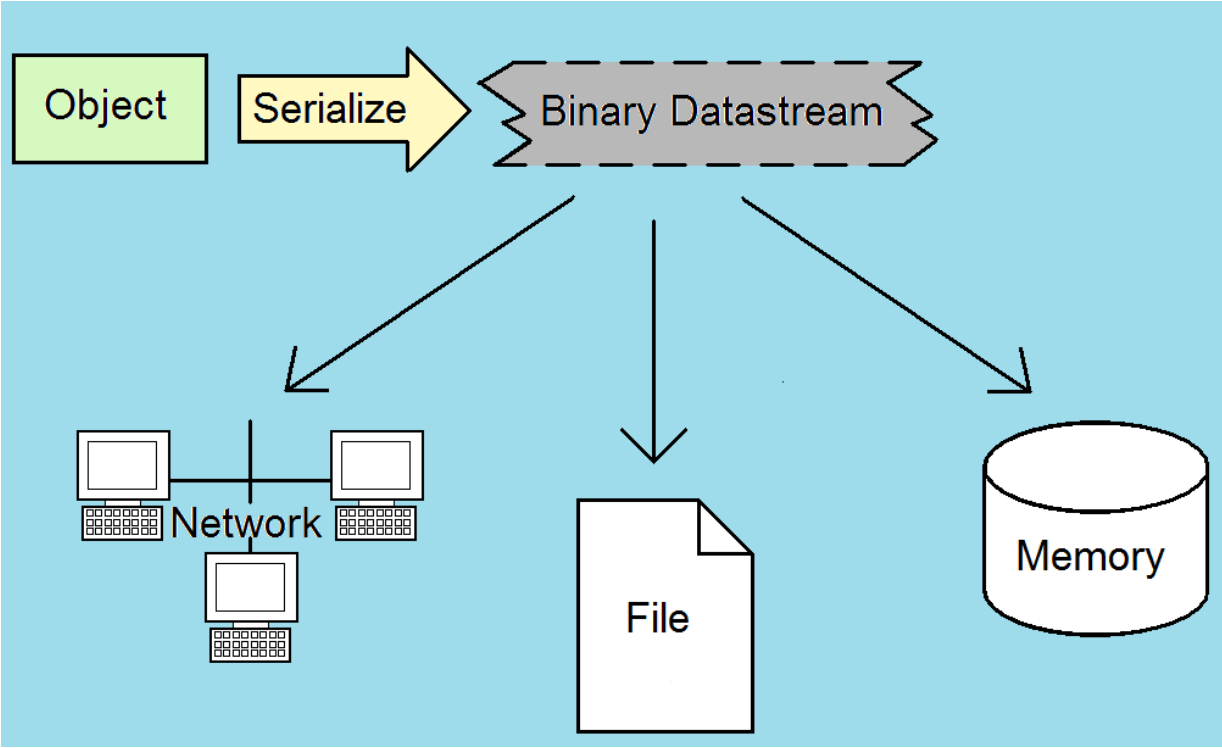

Abbildung 7: Schema einer Serialisierung

In der RCU werden serialisierbare Objekte zum einen bei der Datenübertragung vom Client zum Server und zum anderen beim Speichern und Laden von Konfigurationsdateien genutzt.

Im Falle der Datenübertragung wurde eine Klasse *ActionCommand* angelegt, die alle zur Robotersteuerung notwendigen Parameter enthält. Diese Klasse erbt von der Java-Klasse *Serializable*. So lässt sich jede Instanz von *ActionCommand* direkt über den Outputstream des Clientsockets an den Server versenden. Der Server hat dann diese Instanz zur Verfügung und verarbeitet sie weiter.

4.1.4 Verwaltung der Konfigurationen

Eine Konfiguration der RCU bezeichnet eine Zuweisung von möglichen Controllereingaben[2] auf folgende Roboterbefehle:

- Fahre vorwärts/rückwärts
- Fahre nach rechts/links
- Drehung nach rechts/links
- Balldribbling
- Verschiedene Schussbefehle

Da die Steuerung der Roboter mit unterschiedlichen Eingabegeräten (Controller), wie Tastatur, Gamepad oder Joystick vorgesehen ist, ist eine es notwendig dem Benutzer eine komfortable Möglichkeit zu bieten, eine solche Konfiguration selbst zu erstellen.
In der Software ist auch für Tastatur, Gamepad und Joystick eine Standartkonfiguration implementiert worden. Die Identifizierung der verschiedenen Komponenten eines Gamepads oder Joysticks unterscheidet sich jedoch von Modell zu Modell.
Über die Benutzeroberfläche der RCU kann man eine eigene Konfiguration festlegen. Klickt der Benutzer einen der aufgelisteten Roboterbefehle an, wird dieser zusammen mit der darauf folgenden Eingabe des Controllers in einer Hashmap gespeichert.
Eine Hashmap ist eine Tabelle, in der Objekte in *Keys* (dt.: Schlüssel) und *Values* (dt.: Werte) unterteilt werden. Jedes Wertobjekt wird einem eindeutigen Schlüsselobjekt zugeordnet.
Auf Basis dieser Konfigurations-Hashmap werden alle Eingaben des Controllers in der RCU gefiltert, sodass ausschließlich die in der Konfiguration zugewiesenen Controllereingaben zu Roboterbefehlen weiterverarbeitet und an den Server gesendet werden. Damit der Benutzer auch nach einem Neustart der RCU noch auf eine eigens erstellte Konfiguration zugreifen kann, wurde eine Möglichkeit zur persistenten Speicherung der Konfigurationen umgesetzt.
Der Benutzer kann über die grafische Oberfläche der RCU die erstellte Konfiguration in einer Datei speichern. Dabei wird die Serialisierbarkeit von Hashmaps ausgenutzt [4]. Die Konfigurations-Hashmap wird als serialisierter Datenstrom (siehe Kapitel 4.1.3) in der vom Benutzer angegebenen Datei gespeichert. Sollte die angegebene Datei schon vorhanden sein, so wird sie überschrieben.

[2] Eine Controllereingabe ist zum Beispiel ein bestimmter Knopf eines Gamepads / Taste der Tastatur.

4.1.5 Multithreading

Die RCU führt während der Laufzeit mehrere Threads gleichzeitig aus. Das ist notwendig, da mehrere Aufgaben von der RCU parallel zu bewältigen sind.

Das Sequenzdiagramm in Abbildung 9 soll dies verdeutlichen. Die in dem Sequenzdiagramm verwendeten Klassennamen und Methoden entsprechen nicht unbedingt den Implementationen der RCU. Sie dienen hier nur der Vereinfachung und Veranschaulichung des umgesetzten Multithreadings.

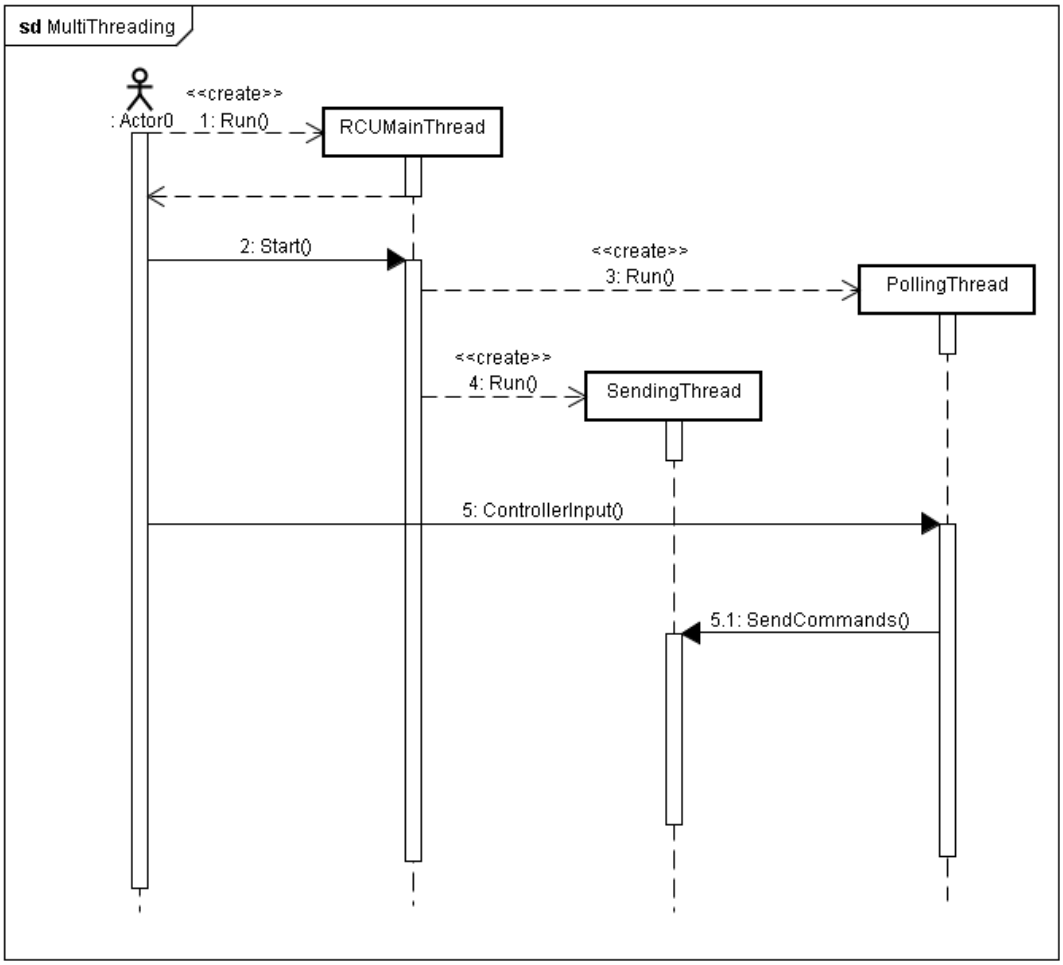

Abbildung 8: Sequenzdiagramm zum Multithreading der RCU

Zunächst muss das RCU-Programm vom Benutzer (Actor) gestartet werden (Run). Nachdem der Mainthread der RCU alle notwendigen Instanzen erzeugt und initialisiert hat, wartet er auf Eingaben des Benutzers. Der Benutzer kann nun die Steuerung der Roboter mit Controller aktivieren (Start). An diesem Punkt ist es unumgänglich, den Programmablauf auf mehreren Threads aufzuteilen, denn einerseits muss die RCU auf weitere Benutzereingaben reagieren können

und gleichzeitig den verwendeten Controller auslesen (PollingThread) und auf Basis dieses Inputs entsprechende Befehle an den Server aussenden (SendingThread).
Deshalb werden, wie im Sequenzdiagramm mit der Methode „Run()“ dargestellt, entsprechende Threads erzeugt und ausgeführt.
Die Abbildung stellt jedoch nur die benötigten Threads für nur einen Controller, der einen Roboter steuert, dar. Für jeden angeschlossenen Controller werden ein PollingThread und ein SendingThread angelegt. Clientseitig, also in der RCU, existiert dabei generell keine Begrenzung der angeschlossenen Controller. Beim Verbindungsaufbau mit dem Server werden jedoch nur so viele Controller mit einem Roboter verbunden, wie Roboter ansteuerbar sind. Davon ist auch die Anzahl der Polling- und SendingThreads abhängig. Eingaben aller überschüssigen Controller werden dann ignoriert.

4.2 Aspekte der Implementierung

4.2.1 Auslesen der Controller mittels JInput

Die Hauptaufgabe der RCU ist das Verarbeiten von Controllereingaben zu Steuerbefehlen für den Tigerbot. Ein Controller ist in diesem Sinne ein Gamepad, ein Joystick oder eine Tastatur.
JInput ist ein Opensource-Framework, welches Java-Klassen und Methoden mit der Funktionalität, Controller auszulesen anbietet.
Die Programmiersprache Java zeichnet sich unter anderem durch die so genannte *Sandbox* aus, welche die Laufzeitumgebung aller Javaprogramme bezeichnet. Dies dient zum Schutz und zur Beschränkung von Speicherressourcen. So sind Programme, die in Java geschrieben wurden, abgekoppelt vom jeweiligen Betriebssystem und ihr Zugriff auf beispielsweise Hardwareschnittstellen ist nicht ohne weiteres möglich. Deshalb schließt das JInput-Framework eine Dynamische Bibliothek (Dynamic Link Library) ein, die den Zugriff auf Gerätetreiber der angeschlossenen Controller ermöglicht.
Eine der Anforderungen an die RCU ist, dass der Benutzer selbst festlegen können soll, mit welchem Knopf, welcher Taste oder welchem Stick des jeweiligen Controllers der Roboter gesteuert werden soll. Es müssen also alle Roboterbefehle mit jeder Controllerkomponente verknüpfbar sein. Dies erfordert eine gewisse Gleichbehandlung der verschiedenen Komponenten und ihrer Ausgabedaten. Wie dies ermöglicht wurde, soll der folgende Abschnitt erläutern.
In der JInput-Klassenstruktur enthält ein Controller mehrere Komponenten. Eine Komponente kann ein Knopf, eine Achse oder ein Steuerkreuz sein. Jede Komponente hat einen eindeutigen Identfier, über den sie ansteuerbar ist. Außerdem enthält jede Komponente eine Variable (Float), die den

Status der Komponente repräsentiert. Beispielsweise ist der Wert eines Knopfes genau „1.0“, wenn man ihn drückt und „0.0“ drückt man ihn nicht. Achsen hingegen ist der Wert der Achse entsprechend der Position des Sticks oder Schalters auf der Achse innerhalb des Intervalls [-1.0, 1.0].

Bei einem Steuerkreuz gibt es neun Zustände, die repräsentiert werden müssen:

Eine oder keine von acht möglichen Richtungen kann gedrückt werden. Diese möglichen Werte dieser Komponente teilen sich im Abstand von 0.125 auf das Intervall [0.0, 1.0] auf.

Einer der Ansprüche an die RCU ist, dass es für den Benutzer in der Konfiguration des Controllers möglich sein soll, jeden beliebigen Roboterbefehl mit jeder beliebigen Komponente seines Controllers zu verknüpfen.

Das bedeutet es wird eine Gleichbehandlung der verschiedenen Komponenten des Controllers verlangt. Um dies zu realisieren wurden Klassen implementiert, die von der Komponenten-Klasse von JInput erben und den Wert aller Komponenten auf ein Intervall [0.0, 1.0] beschränken bzw. aufteilen.

Beim Knopf ist keine Modifikation notwendig, da seine Werte bereits innerhalb dieses Intervalls liegen.

Eine Achse hingegen wird den Instanzen zweier Klassen übergeben. Die eine repräsentiert den negativen, die andere den positiven Teil der Achse. So kann beispielsweise das Schwenken des Sticks in die eine Richtung mit dem Vorwärts-Befehl und das Schwenken des Sticks in die entgegengesetzte Richtung sinnvollerweise mit dem Rückwärts-Befehl verknüpft werden.

Um die acht Richtungen eines Steuerkreuzes voneinander unabhängig mit beliebigen Befehlen belegen zu können, war es nötig eine Klasse anzulegen, deren Instanz genau eine Richtung der übergebenen Steuerkreuzkomponente repräsentiert.

Auf diese Weise ist eine intuitive Konfiguration derart möglich, dass unterschiedliche Richtungen von Achsen und Steuerkreuzen mit unterschiedlichen, beliebigen Befehlen belegt werden können.

Das Auslesens des Controllers folgt erst, nachdem eine Verbindung zum Server hergestellt werden konnte. Von da an, werden auf die Werte der in der Konfiguration berücksichtigten Komponenten alle 5ms ausgelesen und verarbeitet. Wird die Verbindung zum Server getrennt, stoppt auch das Auslesen des Controllers.

4.2.2 Abstrakte Klasse AControllerPresenter

Bei der Einbindung des Gamepads konnte die geplante Programmstruktur gut umgesetzt werden. Als die Tastatursteuerung implementiert werden sollte, ergaben sich einige Probleme. Die Instanzen des ControllerPresenters werden an unterschiedliche Klassen in Model und View übergeben, um dort Zugriff darauf zu haben. Für die Tastatur sollte nun ein neuer Presenter, der KeyboardPresenter, angelegt werden, der die Kontrolle der Tastatur übernimmt. Um diesen Presenter an View und Model zu übergeben, gibt es zwei Möglichkeiten: entweder man überlagert die Methoden, d.h. es gibt die Methoden mehrfach und sie besitzen unterschiedliche Übergabeparameter, oder es wird eine abstrakte Klasse geschrieben, von der beide Presenter erben. Da überlagerte Methoden die Übersichtlichkeit des Codes negativ beeinflussen würden, fiel die Entscheidung auf die Implementierung der abstrakten Klasse AControllerPresenter.

Im folgenden soll kurz erklärt werden, wie abstrakte Klassen in Java funktionieren. Wenn mehrere Klassen eine ähnliche Aufgabe und Implementierung besitzen, kann man eine Basisklasse schreiben, die alle gemeinsamen Elemente beinhaltet und von der andere Klassen erben können. Die abgeleiteten Klassen übernehmen alle Variablen und Methoden von dieser Basisklasse. In den abgeleiteten Klassen müssen abstrakte Methoden anschließend überschrieben (implementiert) werden. Durch Vererbung ist es also möglich eine gewisse Grundfunktionalität für mehrere Objekte bereitzustellen, die von den einzelnen Klassen unabhängig erweitert werden können.

Die meisten Aufgaben des GamepadPresenters gleichen denen der Tastatur, da auch für die Tastatur die JInput Bibliothek benutzt wird. Diese Methoden und Variablen wurden nun in den abstrakten Presenter kopiert und die Vererbung implementiert. Zum aktuellen Stand besteht der Unterschied zwischen Gamepad und Tastatur nur in der Methode `getType()`, die als Rückgabewert den Controllertypen besitzt. Mit der Einführung der Klasse AControllerPresenter fand auch eine Änderung bzw. Spezifikation der Namensgebung statt. Um den genauen Unterschied zwischen Controller, Gamepad und Tastatur herauszustellen, wurde folgendes Schema erarbeitet: Ein Controller ist jede mögliche Art von Eingabesystem für die RCU, d.h. Gamepad und Keyboard. Die Gamepads, mit denen die RCU entwickelt und maßgeblich getestet wurde, sind in Abbildung 6 zu sehen.

Abbildung 9: Bei der Entwicklung verwendetes Gamepad

4.2.3 Unabhängigkeit von der Controllerwahl am Beispiel der ButtonSelectAction

In Kapitel 4.2.1 wurde beschrieben, wie die RCU unabhängig von dem Controller ist, d.h. der Typ des Controllers und die vorhandenen Komponenten müssen der RCU nicht bekannt sein. Somit wird auf der einen Seite einen hohen Bedienkomfort beim Benutzer erzeugt, der theoretisch jeden von JInput unterstützten Controller verwenden kann. Auf der anderen Seite war es nicht notwendig mehrere Controller einzeln implementieren, wodurch die RCU sonst eine sehr begrenzte Zahl von zwei bis drei Gamepads unterstützen würde. Am Beispiel der Klasse ButtonSelectAction.java soll die Unabhängigkeit von der Controllerwahl erklärt werden.

Die ButtonSelectAction ist Teil der View und dient zur Erkennung, welche Komponente gerade gedrückt wurde. Die Klasse implementiert die Interfaces MouseListener und Runnable, d.h. sie kann als MouseListener auf Teile der View gelegt werden und unterstützt Multithreading. Im Controllerpanel besitzen alle Textfelder zur Robotersteuerung (Movement und Action) diesen Mouselistener. Als einzige Mausaktion ist das Klicken in ein Textfeld implementiert, andere Methoden wie mouseEntered oder mousePressed werden zur Erkennung nicht benötigt. Die run-Methode, die durch das Interface Runnable vom Entwickler implementiert werden muss, startet beim Klicken in ein Textfeld, dass die ButtonSelectAction als MouseListener verwendet. In dieser run-Methode findet die Erkennung der gedrückten Controllerkomponente statt. Zunächst werden von dem aktuellen Controller alle Komponenten ausgegeben und in einem Array gespeichert. Eine while-Schleife mit der Abbruchbedingung `!Thread.interrupted()` 'pollt' den Controller immer wieder, d.h. der Controller überprüft intern, ob sich Werte geändert haben. Mit der Methode

getPolledData() kann man anschließend auf die neuen Werte zugreifen. Nachdem man alle Werte aktualisiert hat, beginnt eine for-Schleife über alle Komponenten, die vorher in dem Array 'comps' gespeichert wurden. In dieser for-Schleifen werden mehrere Abfragen gemacht um zu erkennen, welche Art von Komponente (Steuerkreuz, Analogstick oder Button) gedrückt wurden. Die RCU unterscheidet intern zwischen POV (Steuerkreuz), positiven und negativen Achsen und Buttons (siehe Abschnitt 4.2.1). Wenn sich der Wert einer diesen Komponenten geändert hat, wird der Wert und der Name an den Presenter übergeben, das Textfeld beschrieben und eine Log-Information gegeben. Um den Thread zu beenden gibt am Ende der while-Schleife eine Abfrage auf zwei Möglichkeiten. Wenn eine Komponente gewählt wurde, wird ein boolean-Wert auf true gesetzt und damit der Thread unterbrochen. Die zweite mögliche Abbruchbedingung ist der Ablauf einer bestimmten Zeit, ein so genannter Timeout. Beim Start des Threads wird die aktuelle Systemzeit in Millisekunden gespeichert und mit der neuen Systemzeit in der Abfrage verglichen. Wenn nach sechs Sekunden keine Auswahl getroffen wurde, wird der Thread ebenfalls abgebrochen. Damit wird verhindert, dass es zu einer Endlosschleife kommt. Dem Benutzer wird das Ablaufen der Zeit in dem Fenstertitel kenntlich gemacht.

5 Fazit

Die Entwicklung vom Prototypen zur aktuellen Version der RCU wurde entsprechend der anfangs gesetzten Anforderungen durchgeführt. Es können nun mehrere Controller an einer RCU verwendet werden. Die Funktionsweise der RCU ist unabhängig vom Typ des Controllers. Es wurde eine funktionsfähige Verwaltung der Konfigurationen implementiert und getestet. Die RCU ist derzeit bereits im operativem Einsatz.

Für die Zukunft ist eine Weiterentwicklung der RCU vorgesehen. So soll den bisher vorhandenen Roboterbefehlen eine Anzahl erweiterter Befehle hinzugefügt werden. Diese Befehle sollen die bisher implementierten in ihrer Komplexität übersteigen. Ein Beispiel für einen erweiterten Befehl wäre „Fahre einen Kreis“.

IV. Literatur

[1] RoboCup German Open – Offizielle Website;
URL: http://www.robocup-german-open.de/de/robocup, letzte Einsicht: 15.12.10

[2] Tigers Mannheim – Offizielle Website;
URL: http://tigers-mannheim.de/, letzte Einsicht: 15.12.10

[3] MSDN Microsoft: Serialisierung;
URL: http://msdn.microsoft.com/de-de/library/ms233843.aspx,
letzte Einsicht: 15.12.10

[4] Christian Ullenboom: Java ist auch eine Insel;
URL: http://openbook.galileocomputing.de/javainsel8/,
letzte Einsicht: 15.12.10

[5] JInput-Projekt – Offizielle Website;
URL: https://jinput.dev.java.net/, letzte Einsicht: 15.12.10

[6] Andrew Davison: Java Prog. Techniques for Games; 2006

Anhang

Programmablaufplan

Programmstart

- Applikation starten
- Frame erscheint
- ControllerTab für jeden Controller erzeugen
- DefaultConfig für jeden Controller laden, d.h. in Textfelder schreiben

Botwahl

- User gibt ServerIP ein → Abfrage, ob Server vorhanden
- User gibt Ports des jeweiligen Roboters ein → Abfrage

Tastenbelegung eingeben

- User clickt auf Textfield → keine Eingabe möglich (Felder nicht disabled)
- Meldung weist auf Drücken des gewünschten Button hin
    1. Popup
    2. Statuszeile
- Überwachung aller Controller Components
- Bei Wert != 0 → Zuweisung Befehl/Taste
- Abbruch der Überwachung bei
    - Popup: per Mouse beendet
    - Zeit vergangen
- Textfeld füllen

Applikationstart

View:

- StartButton → StopButton
- ControllerConfigPanel deaktivieren

Model:

- ~~Config aus View lesen ODER View übergibt Buttons an Model~~
  Presenter holt Buttons aus View und übergibt an Model
- Controller überwachen

  Bei Änderung der Werte:

- Werte der Components mit Befehl übergeben
- Befehl umwandeln
- Zeichen über Sockets an ausgewählten Server schicken
- Server wandelt empfangene Daten in Befehle und Werte um
- Übergibt Bot und Befehl an BotManager

Applikation stoppen

View:

- StopButton → StartButton
- ControllerConfigPanel aktivieren

Model:

- Serververbindung trennen
- Überwachung der Controller beenden

Kurzfassung

Das Studentenprojekt der Tigers Mannheim hat zum Ziel, ein Team autonomer Roboter, gesteuert durch eine zentrale künstliche Intelligenz, zum Fußballspiel zu befähigen. Zu Testzwecken wird eine komfortable Möglichkeit zur manuellen Steuerung der Roboter benötigt. Diese wurde im Rahmen dieser Studienarbeit konzipiert und implementiert. Dabei konnte auf einen bereits vorhandenen Prototypen, der Basisfunktionalitäten implementiert hat, zurückgegriffen werden.

Aufgrund der komplexen Anforderungen an die Steuerung der Roboter und in Hinblick auf die Codequalität, wurde nicht auf dem Prototypen aufgesetzt, sondern eine entsprechende Software von Grund auf neu entwickelt.

Die Roboter sollen, inspiriert durch die Steuerung von Computerspielen, mit Tastatur, Gamepad oder Joystick gesteuert werden.

Die bereits vorhandene Zentralsoftware der Tigers Mannheim ermöglicht den Zugriff auf die Roboter. Es wurde eine Client-Server-Anwendung entwickelt, wobei der Schwerpunkt der Arbeit die Entwicklung des Clients war. Das Clientprogramm, welches auf dem MVP Prinzip basiert, liest die angeschlossenen Controller mittels des Open-Source-Frameworks JInput aus und wandelt diese Daten in für die Roboter ausführbare Befehle um und sendet diese an den Server innerhalb der Zentralsoftware.

Die Software wurde erfolgreich getestet und befindet sich bereits im operativen Einsatz.

Künftig ist eine Erweiterung der Steuerung um komplexere Befehle vorgesehen.

Abstract

The student project of the Team Tigers Mannheim aims at enabling a group of robots to play soccer, so that they can participate in RoboCup 2011, the world championship in robot soccer. To run tests a convenient possibility to control the robots manually is needed. Within the scope of this work such a control utility has been concieved and implemented. There has already been a prototype before, that provided some basic functionalities, but because of the complex requirements the decision was made to not let the Robot Control Utility base upon this prototype. Furthermore that simplifies to maintain the aspired code quality.
Inspired by the control of a computer game, the robots are meant to be controlled by controllers such as keyboards, gamepads and joysticks.
The access to the robots is provided by the central software of the Tigers Mannheim. The Robot Control Utility polls the controller inputs by means of the open source framework Jinput and converts them into robot commands. These commands are sent to the central software via a client server connection.
According to these requirements a runnable software was created. It has been tested successfully and it is already in use.